Smart Analog IC101

R21AN0013EJ0100
Rev.1.00
Feb 01, 2015

How to Use Smart Analog IC101's API and Sample Code for Other MCUs

Introduction

This application note describes how to use API functions and sample code to control Smart Analog IC 101 (referred to as “SAIC101” herein) when using an RL78 family MCUs that does not belong to the RL78/L13 group.

Target Device

Smart Analog IC 101 (part name: RAA730101)

Contents

1. Related Application Notes

The following application notes also provide information related to SAIC101 and should be referred to as needed in combination with this document.

- Smart Analog IC101 API Specification (R21AN0015EJ)
- Smart Analog IC101 Tutorial for Sample Code Introduction and API Builder SAIC101 (RL78/L13) (R21AN0012EJ)
- RL78/G13 Serial Array Unit (UART Communication) (R01AN0459EJ)
- RL78/G13 Serial Array Unit for 3-Wire Serial I/O (SPI Master Transmission/Reception) for CubeSuite+, IAR, and e2 studio Development Environments (R01AN1367EJ)

2. How to use Smart Analog IC101's API and sample code for other MCU

The Smart Analog IC101 sample code calls the RL78/L13 serial communication module function (which is generated by CubeSuite+ Code_Generator for RL78_78K and the code generator plug-in included in e^2 studio) from the API function. Functions generated by the RL78 family code generation tool are compatible for all RL78 family MCUs. When using an MCU that is supported by the RL78 family code generation tool, the user can automatically include the SAIC101 sample code into user project files created with CubeSuite+ or e^2 studio by adding an MCU definition file to the API Builder SAIC101 coding assistance tool. However, when using an MCU that is not supported by the RL78 family code generation tool, the user will need to create functions equivalent to and compatible with the functions generated by the code generation tool, and manually include the sample code or API file into the project files.

The following is the basic flow for how to use Smart Analog IC101's API and sample code for other MCUs.

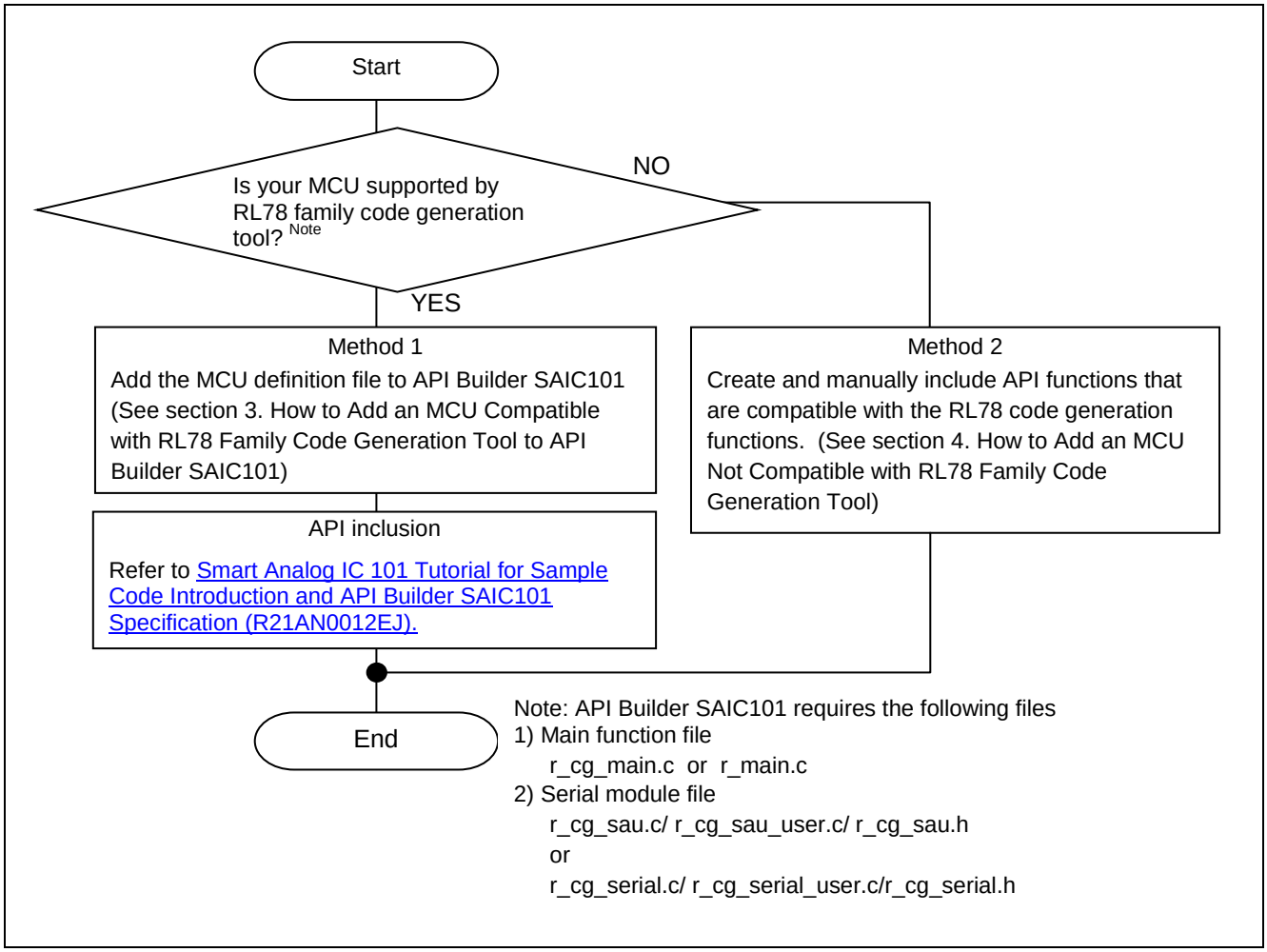


Figure 2-1 Routine Flow for Replacing MCU

Code Generator Plug-in

http://www.renesas.com/products/tools/coding_tools/coding_assistance/cg_p/index.jsp

3. How to Add an MCU Compatible with RL78 Family Code Generation Tool to API Builder SAIC101

When using an MCU that is supported by the RL78 family code generation tool, the user can automatically include the SAIC101 sample code in user project files of CubeSuite+ or e^2 studio by adding an MCU definition file to the API Builder SAIC101 coding assistance tool. (The CubeSuite+ Code_Generator for RL78_78K or the code generator plug-in included in e^2 studio must output source code for serial communication equivalent to code output for RL78/L13.) This section serves as a tutorial for RL78/G14 (R5F104PJ), providing an example of adding an MCU that is supported by the RL78 family code generation tool.

API Builder SAIC101 download URL: http://www.renesas.com/smart_analog_api_builder

File name: API_Builder_SAIC101_Ver1.1.zip

3.1 Preparation

First, confirm the part name of the RL78 family MCU to be added. The part name must be one that is recognized by CubeSuite+ or e^2 studio.

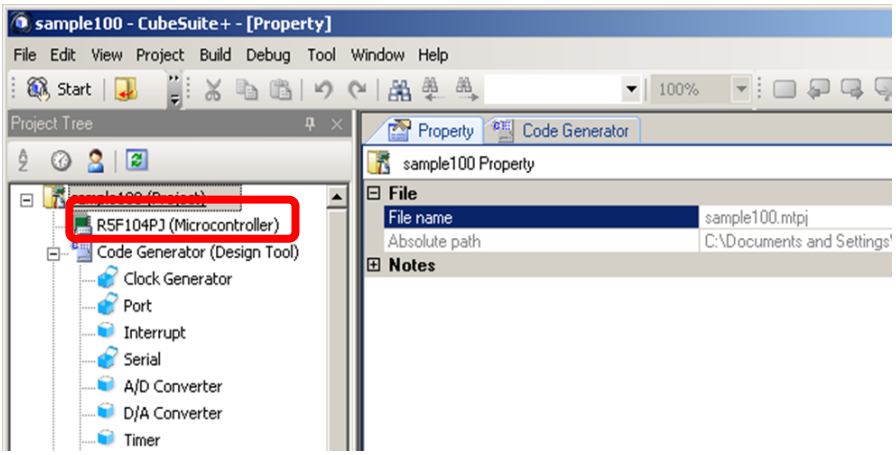


Figure 3-1 Confirm RL78 family MCU Part Name

Next, create the MCU definition file. The definition file will be created in "Chips" which is stored at the same level in the project tree as the folder that stores API_Builder_SAIC101.exe. Copy the Template.csv file stored in "Chips," and change the file name (in this case, the file name is changed to R5F104PJ).

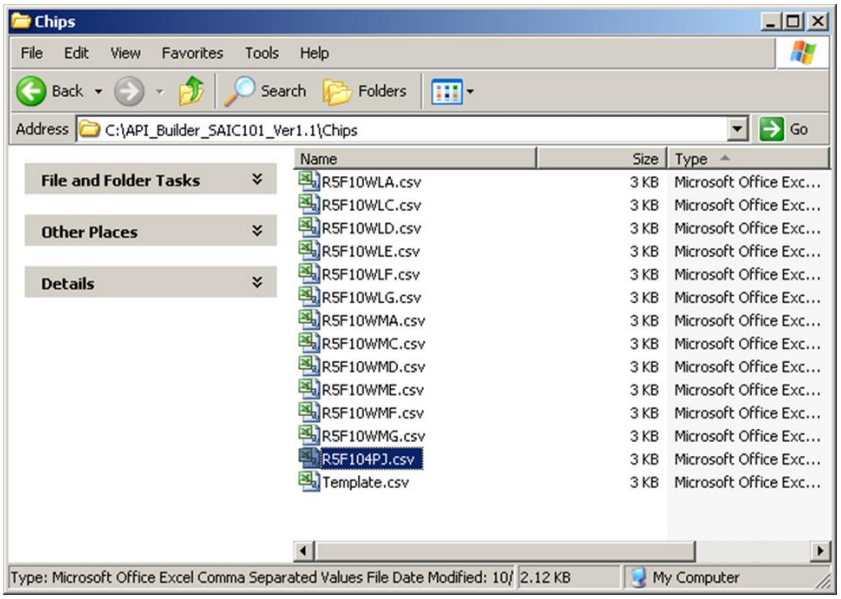


Figure 3-2 Folder for Storing MCU Definition Files

3.2 How to Edit the MCU Definition File

This section describes how to edit the MCU definition file created in Section 3.1. Use either Excel or a text editor to edit the file. The following example uses Excel.

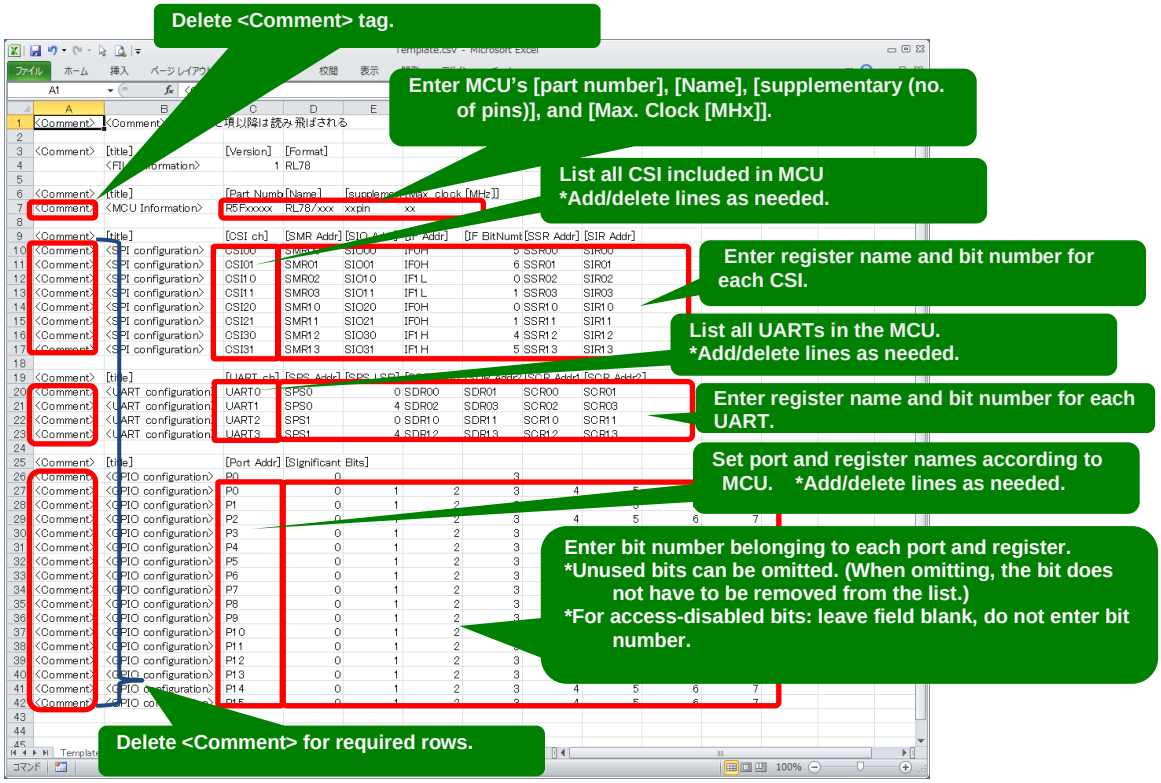


Figure 3-3 Fields Requiring Editing

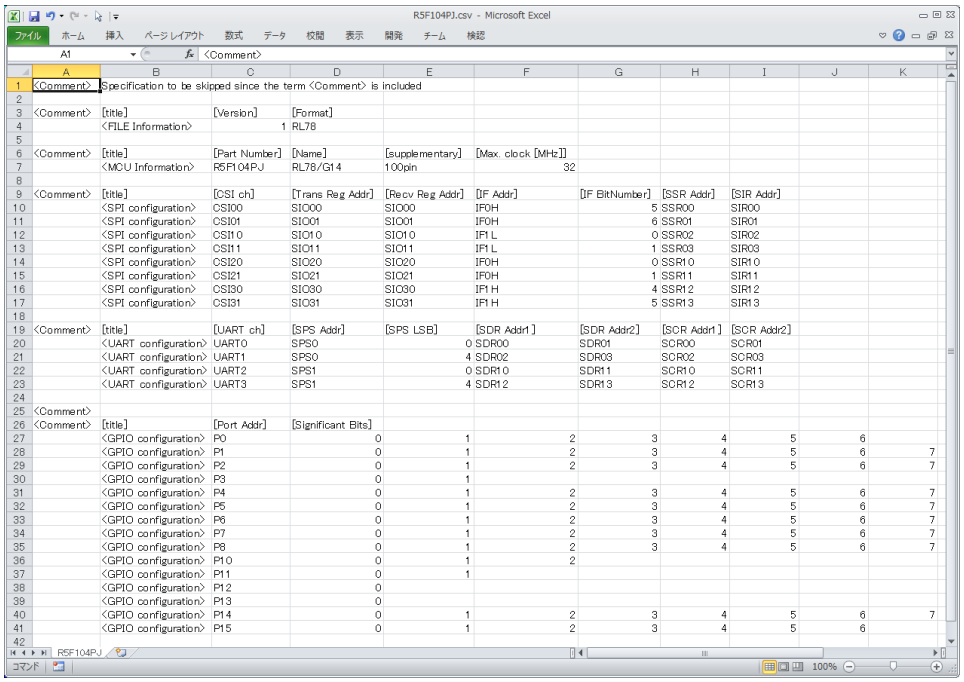


Figure 3-4 Example of Changes to MCU Definition File

3.3 Confirming Addition of MCU Definition File

Save the file edited in section 3.2 and start the API Builder SAIC101.

(The MCU definition file is read during API Builder SAIC101 startup. If API Builder SAIC101 is already running, please close the software and then restart.)

If the MCU definition file is not successfully edited, the message shown in Figure 3-5 will appear at startup. This message indicates that the file has not been read successfully and the MCU has not been added to the coding assistance tool. Please correct the MCU definition file as needed to ensure it is successfully edited and read.

Figure 3-5 MCU Definition File Editing Error Message

When the API Builder SAIC101 is started up and the project created in CubeSuite+ or e^2 studio is read successfully, the message shown in Figure 3-6 is displayed, indicating the RL78 family MCU part name. This completes the confirmation process.

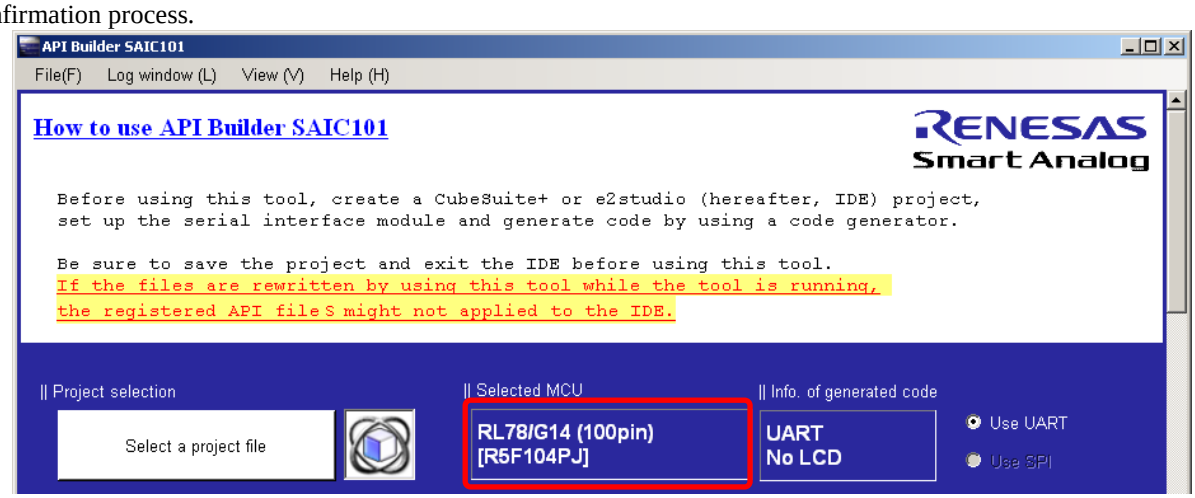


Figure 3-6 Confirmation of New MCU Definition File Added to SAIC101

For more details on the above settings, please refer to Smart Analog IC 101 Tutorial for Sample Code Introduction and API Builder SAIC101 (RL78/L13) (R21AN0012EJ).

4. How to Add an MCU Not Compatible with RL78 Family Code Generation Tool

The SAIC101's API was designed assuming use of RL78/L13 source code generated by CubeSuite+ Code_Generator for RL78_78K or the code generator plug-in included in e^2 studio for project files. Therefore, when replacing an MCU that is not supported by the RL78 family code generation tool, the user needs to create functions for processing and data input/output equivalent to functions output by the code generation tool. The user will also need to add global variables and modify definitions based on the user system environment.

4.1 Changes to API

This section describes the changes required for the API file (Table Table 4-1).

Table 4-1 API File

Communication Method	API file
UART	r_sa_uart_control_register.c / r_sa_uart_control_register.h / r_sa_uart_control_register_user.c
SPI	r_sa_spi_control_register.c / r_sa_spi_control_register.h / r_sa_spi_control_register_user.c

4.1.1 Replacing Global Constant to Store Serial Module Information

Replace the global constant to store serial module information based on the method of communication as shown in Figure 4-1 for UART communications and Figure 4-2 for SPI communications. The element number in this array is specified by the enumeration used to specify the global variable to store serial module information. When replacing the MCU, make sure you replace the serial module function of the element number that corresponds to the channel number to be used. Refer to section 4.2.1 for information regarding Creating Serial Communication Functions corresponding to the user system.

Target File: r_sa_uart_control_register_user.c

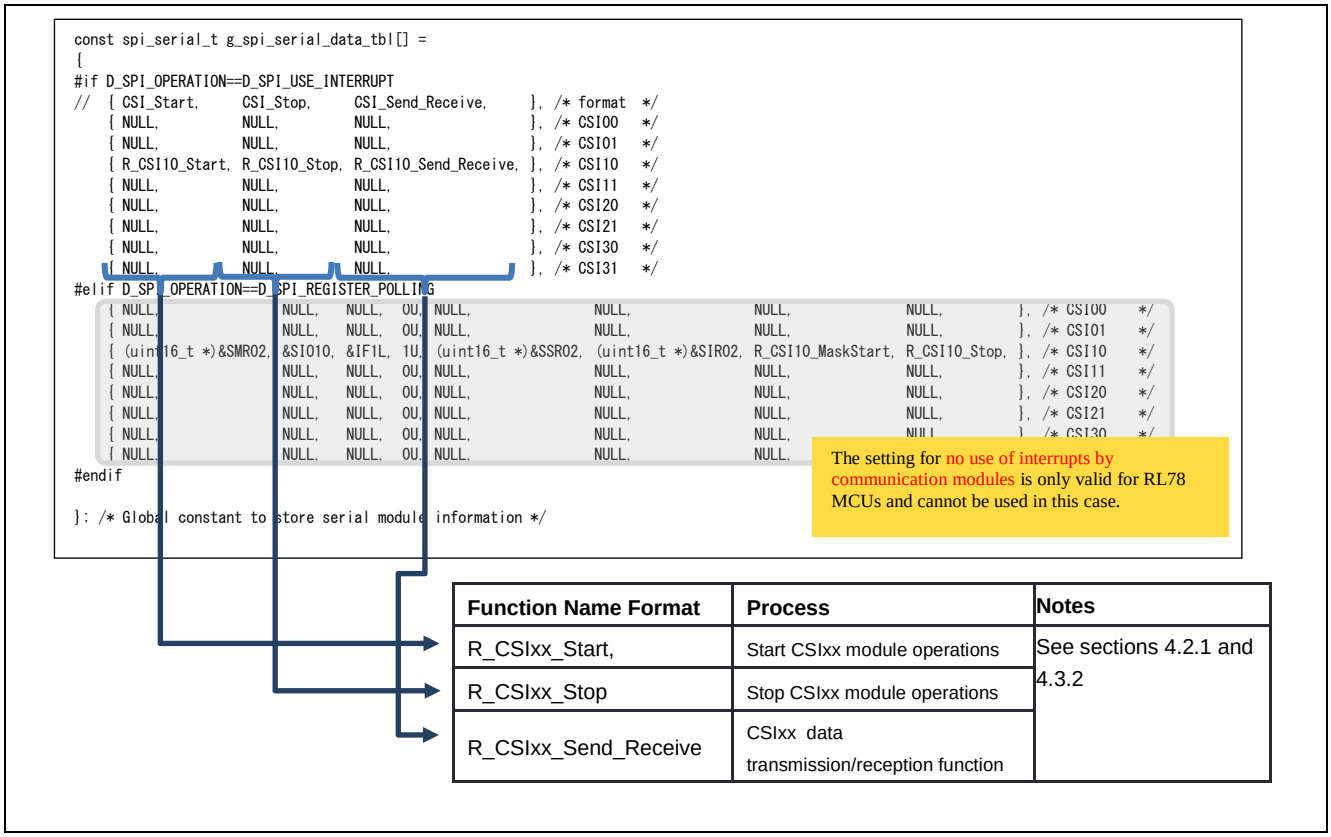
```
const uart_serial_t g_uart_serial_data_tbl[] =
{
#if (D_UART_OPERATION==D_UART_USE_INTERRUPT)
//  { R_UARTx_Start,  R_UARTx_Stop,  R_UARTx_Receive,  R_UARTx_Send,  R_UARTx_GetHeader,  R_UARTx_Getdata,  R_UARTx_SettingChange,  }, /* format */
    { NULL,           NULL,          NULL,             NULL,         NULL,               NULL,             NULL,                  }, /* UART0  */
    { R_UART1_Start,  R_UART1_Stop,  R_UART1_Receive,  R_UART1_Send,  R_UART1_GetHeader,  R_UART1_Getdata,  R_UART1_SettingChange,  }, /* UART1  */
    { NULL,           NULL,          NULL,             NULL,         NULL,               NULL,             NULL,                  }, /* UART2  */
    { NULL,           NULL,          NULL,             NULL,         NULL,               NULL,             NULL,                  }, /* UART3  */
    { NULL,           NULL,          NULL,             NULL,         NULL,               NULL,             NULL,                  }, /* UART4  */
    { NULL,           NULL,          NULL,             NULL,         NULL,               NULL,             NULL,                  }, /* UART5  */
#elif D_UART_OPERATION==D_UART_REGISTER_POLLING
    /* Not supported */
#endif
/*
/*
}; /* global constant to store serial module information */
```

(1) (2) (3) (4) (5) (6) (7)

No.	Function Name Format	Process	Notes
(1)	R_UARTx_Start	Start UARTx module operations	See sections エラー! 参照元が見つかりません。 and 4.3.1
(2)	R_UARTx_Stop	Stop UARTx module operations	
(3)	R_UARTx_Receive	UARTx reception function	
(4)	R_UARTx_Send	UARTx transmission function	
(5)	R_UARTx_GetHeader	UARTx header acquisition function	
(6)	R_UARTx_Getdata	UARTx data acquisition function	
(7)	R_UARTx_SettingChange	UARTx setting change function	

Figure 4-1 Changes to Global Constant to Store Serial Module Information (UART)

Target File: r_sa_spi_control_register_user.c



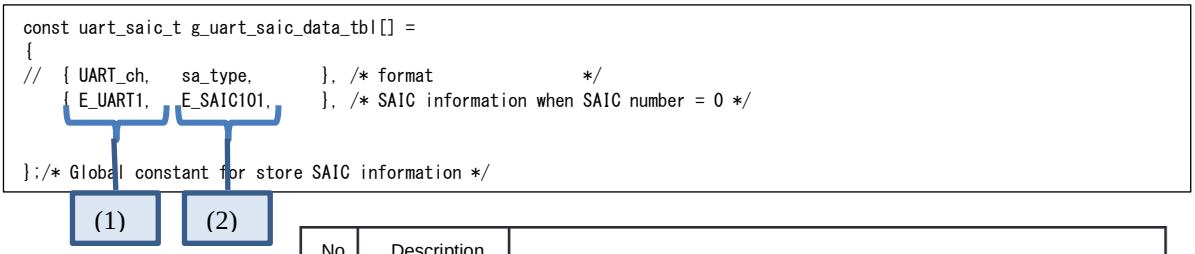
Function Name Format	Process	Notes
R_CSIxx_Start,	Start CSIxx module operations	See sections 4.2.1 and 4.3.2
R_CSIxx_Stop	Stop CSIxx module operations	
R_CSIxx_Send_Receive	CSIxx data transmission/reception function	

Figure 4-2 Changes to Global Constant to Store Serial Module Information (SPI)

4.1.2 Replacing Global Constant to Store SAIC Information

Replace the global constant to store SAIC information based on the method of communication, as shown in Figure 4-3 for UART and Figure 4-4 for SPI.

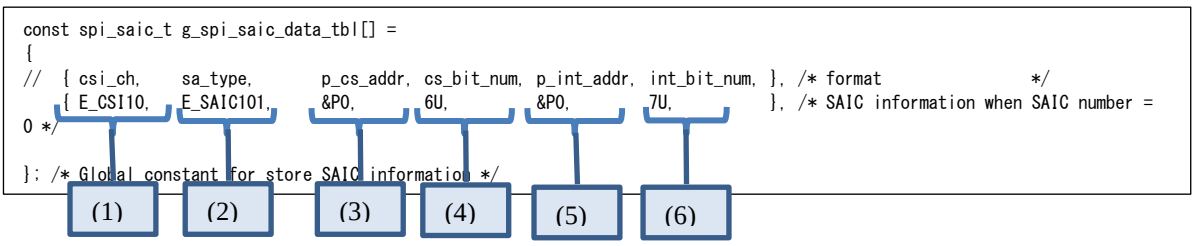
Target File: r_sa_uart_control_register_user.c



No.	Description example	Description
(1)	E_UART1	Specify the ENUM e_uart_ch_t for specifying the global constant to store serial module information for the element number of the array stored in the function replaced in section 4.1.1.
(2)	E_SAIC101	Specify the ENUM e_saic_type_t for specifying the SAIC type (part name) for the type of SAIC connected to the serial module specified in (1).

Figure 4-3 Changes to Global Constant to Store SAIC Information (UART)

Target File: r_sa_spi_control_register_user.c



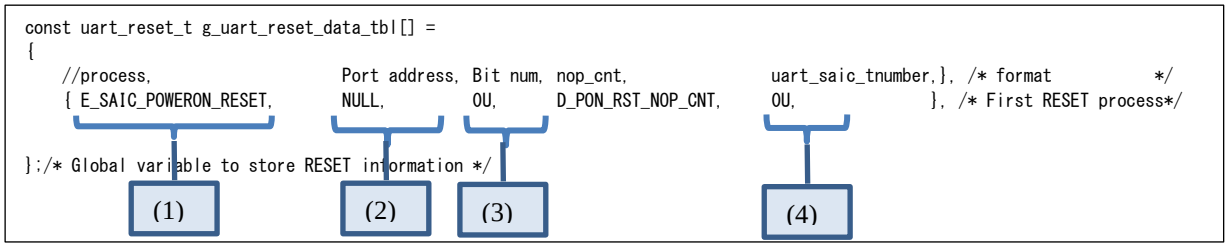
No.	Description example	Description
(1)	E_CSI10	Specify the ENUM e_uart_ch_t for specifying the global constant to store serial module information for the element number of the array stored in the function replaced in section 4.1.1.
(2)	E_SAIC101	Specify the ENUM e_saic_type_t for specifying the SAIC type (part name) for the type of SAIC connected to the serial module specified in (1)
(3)	&P0	Define address of port register connected to CS pin.
(4)	6U	Define bit number of port register connected to CS pin.
(5)	&P0	Define address of port register connected to INT pin.
(6)	7U	Define bit number of port register connected to INT pin.

Figure 4-4 Changes to Global Constant to Store SAIC Information (SPI)

4.1.3 Replacing Global Constant to Store RESET Information

Although unnecessary when using UART, refer to Figure 4-5 when using SPI to replace the global constant to store RESET information based on the method of communication.

Target File: r_sa_spi_control_register_user.c



No.	Description example	Description
(1)	E_SAIC_POWERON_RESET	Specify RESET process in enumeration for specifying RESET process e_reset_process_t.
(2)	NULL	Define address of port register connected to RESET pin when using external RESET. For all other settings, define NULL.
(3)	0U	Define bit number of port register connected to RESET pin when using external RESET. For all other settings, define 0.
(4)	0U	Define SAIC number for internal RESET. Although this API does not use this setting for any type of RESET process other than internal, still specify the SAIC number. (This is used in API Builder SAIC101.)

Figure 4-5 Changes to Global Constant to Store RESET Information (SPI)

4.1.4 Corrections to #include Header Files

As the API uses the types and functions output by the RL78 family code generation tool, the API includes the header file output by the code generation tool. This header file must be replaced with a header file defined by the user-created serial communication function or a header file that reflects the contents detailed in sections 4.2.4 and 4.2.5.

Table 4-2 Files Requiring Corrections

Communication Method	Files
UART	r_sa_uart_control_register_user.c r_sa_uart_control_register.c
SPI	r_sa_spi_control_register_user.c r_sa_spi_control_register.c

Table 4-3 #include Header File Changes

Item	Existing definition	Changes
Serial communication definition	#include "r_cg_sau.h"	Replace with the header file defined by the user-created serial communication function
Type definition	#include "r_cg_macrodriver.h"	Replace with the header that reflects the contents detailed in sections 4.2.4 and 4.2.5

4.1.5 Corrections to Macro Declarations for User Environment-dependent Settings

To adjust settings according to the target MCU, refer to section **4.2 Macro Declarations for User Environment-dependent Settings in the** Smart Analog IC101 API Specification (R21AN0015EJ).

In addition, the API calculates the number of loops for the internal software wait from the minimum number of steps required for loop processing and uses seven clocks as required in the RL78 MCU. Change the definitions in Tables. Table 4-4 and Table 4-5 accordingly to correct for the difference in number of steps required by the loop processing in the target MCU.

Target File: r_sa_uart_control_register_user.c

Table 4-4 Software Wait Loop Count Definition (UART)

Existing Definition	Change
#define D_PON_RST_NOP_CNT ((uint32_t)((D_WAIT_PON_RST_TIME_MS / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	Change section 7.0F accordingly to reflect the minimum number of steps required for loop processing in the target MCU. Input range: float type
#define D_UART_4800BPS_HALF_BIT ((uint16_t)((D_UART_4800BPS_HALF_BIT_MS / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_UART_250kbps_HALF_BIT ((uint16_t)((D_UART_250kbps_HALF_BIT_MS / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_UART_5MS_NOP_CNT ((uint16_t)((D_UART_5MS / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_UART_1800US_NOP_CNT ((uint16_t)((D_UART_1800US / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_UART_270US_NOP_CNT ((uint16_t)((D_UART_270US / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_UART_250US_NOP_CNT ((uint16_t)((D_UART_250US / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	

Target File: r_sa_spi_control_register_user.c

Table 4-5 Software Wait Loop Count Definition (SPI)

Existing Definition	Change
#define D_PON_RST_NOP_CNT ((uint32_t)((D_WAIT_PON_RST_TIME_MS / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	Change section 7.0F accordingly to reflect the minimum number of steps required for loop processing in the target MCU. Input range: float type
#define D_HARD_RESET_NOP_CNT ((uint32_t)((D_WAIT_HARD_RESET_TIME_MS / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_SPI_5MS_NOP_CNT ((uint16_t)((D_SPI_5MS / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_SPI_1800US_NOP_CNT ((uint16_t)((D_SPI_1800US / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_SPI_820US_NOP_CNT ((uint16_t)((D_SPI_820US / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_SPI_250US_NOP_CNT ((uint16_t)((D_SPI_250US / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	
#define D_SPI_3US_NOP_CNT ((uint16_t)((D_SPI_3US / (1.0F / D_CPU_CLK_MHZ)) * 1000.0F / **7.0F**))	

4.2 Tasks Required for User-created Source File

This section describes the changes required for the user-created source file.

4.2.1 Creating Serial Communication Functions

The API uses the functions output by the RL78 family code generation tool for hardware access to serial communication functions. Additional functions are also created for UART-related hardware access. The user will need to provide equivalent functions when using a replacement MCU. The following lists the functions necessary for using the API. For more details regarding functions, see section 4.3.

Table 4-6 Serial Communication Functions

Communi-cation Method	Function Name in Sample	Argument	Return Value	Description
UART	R_UART1_Start	None	None	Goes to UART communications wait state.
	R_UART1_Stop	None	None	Stops UART communication.
	R_UART1_Receive	uint8_t * const rx_buf, uint16_t rx_num	MD_STATUS	Starts UART reception of data.
	R_UART1_Send	uint8_t * const tx_buf, uint16_t tx_num	MD_STATUS	Starts UART transmission of data.
	R_UART1_GetHeader	uint8_t *packet_data uint8_t rx_buffer[] uint16_t read_pos	uint8_t	If data is received, acquires 1 byte of header data for analysis.
	R_UART1_Getdata	uint16_t rx_cnt	uint8_t	Checks for data exceeding specified receive data count.
	R_UART1_SettingChange	uint8_t setting	None	Changes UART module communication setting.
SPI	R_CSI10_Start	None	None	Goes to 3-wire serial I/O communication wait state.
	R_CSI10_Stop	None	None	Stops 3-wire serial I/O communication.
	R_CSI10_Send_Receive	uint8_t * const tx_buf, uint16_t tx_num, uint8_t * const rx_buf	MD_STATUS	Starts CSI transmission/reception of data.

4.2.2 Call API Functions, Assign Global Variables

— When using API functions, first execute the Smart Analog initialization functions (R_SAIC_SPI_Init, R_SAIC_UART_Init).

— When unsure of the default SAIC101 baud rate, always call the communication setting negotiation function (R_SAIC_UART_Negotiation) after executing Smart Analog initialization functions but before starting UART communications. This will allow you to adjust the SAIC101 UART baud rate, parity, and other communication settings to satisfy MCU and SAIC101 conditions.

— Set the bit corresponding to enum e_uart_ch_t of the UART reception completion flag (g_uart_rx_end_flag) after receiving the number of bytes specified in the UART reception function argument.

— Set the bit corresponding to enum e_uart_ch_t of the UART transmission completion flag (g_uart_tx_end_flag) after transmitting the number of bytes specified in the UART transmission function argument.

— After transmission/reception of the number of bytes specified in the 3-wire serial (clock synchronous serial) communication transmission/reception function argument is complete, specify the argument corresponding to enum e_csi_ch_t, and call the API's R_SAIC_SPI_CSDisable function to set the CS pin to high. Also add a completion process as needed.

— If an overrun error occurs during 3-wire serial communication, specify the argument corresponding to enum e_csi_ch_t, call the R_SAIC_SPI_CSDisable function to set the CS pin to high, and set the bit corresponding to enum e_csi_ch_t of the overrun error flag (g_csi_overrun_flag). Also add a communication completion process as needed.

4.2.3 Global Variable Definitions Used by API for Serial Communications

The API determines whether operations are in the serial communication state or error state based on the global variables listed in Table 4-7. These global variables should be declared on the user side, ensuring that they can be referenced by API functions. In addition, the user should add the corresponding process items to the user's source code as described in section 4.2.2

Table 4-7 Global Variables Used by API for Serial Communications

Communication Method	Part Name	Variable Name	Description	API Functions Used
UART	uint8_t	g_uart_tx_end_flag	UART transmission completion flag. Sets bit corresponding to e_uart_ch_t when transmission is completed. Clears flag in API	r_saic_uart_send_command
UART	uint8_t	g_uart_rx_end_flag	UART reception completion flag. Sets bit corresponding to e_uart_ch_t when reception is completed. Clears flag in API.	r_saic_uart_write_read r_saic_uart_send_command r_saic_uart_get_response R_SAIC_UART_ADC_GetResult R_SAIC_UART_ADC_GetReceive r_saic_uart_flash_read r_saic_uart_flash_write r_saic_uart_flash_all_erase r_saic_uart_all_flash_to_reg r_saic_uart_buffer_refresh R_SAIC_UART_FLASH_WRITE_01H R_SAIC_UART_FLASH_WRITE_1FH
SPI	uint8_t	g_csi_overrun_flag	SPI overrun flag. Sets the bit corresponding to e_csi_ch_t if an overrun occurs. Clears flag in API.	r_saic_spi_overrun_err_check

4.2.4 Width Specification Integer Types

The API uses C99-compliant width specification integer types (intN_t, uintN_t), which are declared in the macrodriver.h generated by the code generation tool, not in stdint.h. If width specification integer types are not supported by the compiler you are using (i.e. stdint.h is not defined), please add the definitions listed in Table 4-8.

Table 4-8 Width Specification Integer Types Used in API

Type	Description	Definition
int8_t	8-bit width signed integer type	typedef signed char int8_t;
uint8_t	8-bit width unsigned integer type	typedef unsigned char uint8_t;
int16_t	16-bit width signed integer type	typedef signed short int16_t;
uint16_t	16-bit width unsigned integer type	typedef unsigned short uint16_t;
int32_t	32-bit width signed integer type	typedef signed long int32_t;
uint32_t	32-bit width unsigned integer type	typedef unsigned long uint32_t;

4.2.5 RL78 MCU-Specific Description Corrections

The API functions use the descriptions created specifically for some RL78 MCUs and compilers and the definitions used in the code generation tool. When using SAIC101 with other MCUs, the user will need to replace some descriptions, accordingly.

Table 4-9 Specialized Definitions

Item	Function/Variable/Definition	**Description**
RL78 MCU-specific description	NOP()	Description of nop command execution for RL78 compiler. When using other MCUs, this description must be replaced according to the macro definition.
Definitions used in code generation tool	MD_STATUS	Definition of the return value type of the serial function. When using other MCUs, add the following definition: typedef unsigned short MD_STATUS; The return values used: MD_OK and MD_ARGERROR Declaration is as follows: #define MD_STATUSBASE (0x00U) /* register setting OK */ #define MD_OK (MD_STATUSBASE + 0x00U) /* Error list definition */ #define MD_ERRORBASE (0x80U) /* error argument input error */ #define MD_ARGERROR (MD_ERRORBASE + 0x01U)

4.3 Function Specifications

4.3.1 UART Communications

Specifications of functions required for UART communications are listed below. UART1 is used as an example in the explanations.

[Function Name] R_UART1_Start

Outline	UART1 operation start process
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h
Declaration	void R_ UART1_Start(void)
Description	Starts operations of channel corresponding to the serial array unit and goes to wait state
Argument	None
Return Value	None
Reference doc.	RL78/G13 Serial Array Unit (UART Communication) R01AN0459EJ
Additional Notes	When it is not necessary to start and stop serial communication functions in the API, this function does not need to be created. In this case, always register a dummy function in the corresponding location in the global variable to store serial module information.

[Function Name] R_UART1_Receive

Outline	UART1 reception status initialization function
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h
Declaration	MD_STATUS R_UART1_Receive(uint8_t *rxbuf, uint16_t rxnum)
Description	Initial setup of UART1 reception
Argument	uint8_t *rxbuf: [receive data buffer address] uint16_t rxnum: [receive data buffer size]
Return Value	If [MD_OK]: Reception setup completed If [MD_ARGERROR]: reception setup failure
Reference doc.	RL78/G13 Serial Array Unit (UART Communication) R01AN0459EJ
Additional Notes	When a register read or other API is called, the API calls this function, the UART communication function goes to the reception wait state and continues to receive data in the background until the specified number of bytes of data is received. The receive data storage buffer must be established by the user and passed by-pointer. After the number of receptions specified in argument rxnum is completed, execute the communication completion process described in section 4.2.2

[Function Name] R_UART1_Send

Outline	UART1 data transmission function
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h
Declaration	MD_STATUS R_UART1_Send(uint8_t* txbuf, uint16_t txnum)
Description	Initial setup of UART1 transmission and start data transmission.
Argument	uint8_t *txbuf: [transmit data buffer address] uint16_t txnum: [transmit data buffer size]
Return Value	If [MD_OK]: transmission setup completed If [MD_ARGERROR]: transmission setup failure
Reference doc.	RL78/G13 Serial Array Unit (UART Communication) R01AN0459EJ
Additional Notes	Almost all APIs call this function. When an API calls this function, the UART communication function goes to the transmission wait state and continues to transmit data in the background until the specified number of bytes of data is sent. The transmit data storage buffer must be established by the user and passed by-pointer. After the number of communications specified in argument txnum is completed, execute the communication completion process described in section 4.2.2.

[Function Name] R_UART1_Stop

Outline	UART1 operation stop process
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h
Declaration	void R_UART1_Stop (void)
Description	Stops operations of channel corresponding to the serial array unit.
Argument	None
Return Value	None
Reference doc.	RL78/G13 Serial Array Unit (UART Communication) R01AN0459EJ
Additional Notes	When it is not necessary to start and stop serial communication functions in the API, this function does not need to be created. In this case, always register a dummy function in the corresponding location in the global variable to store serial module information.

[Function Name] R_UART1_SettingChange

Outline	UART1 setting change process
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h, r_sa_uart_control_register.h
Declaration	void R_UART1_SettingChange(uint8_t setting)
Description	Changes settings of corresponding UART channel according to the value of the argument “setting”.
Argument	uint8_t setting
Return Value	None
Reference doc.	–
Additional Notes	Changes the communication format of the UART communication function according to the value of the argument “setting”.

Argument “setting” Value	UART Communication Setting
0	4800bps, Parity = None
1	4800bps, Parity = Odd
2	4800bps, Parity = Even
3	250000bps, Parity = None
4	250000bps, Parity = Odd
5	250000bps, Parity = Even

[Function Name] R_UART1_GetHeader

Outline	Header data acquisition function	
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h, r_sa_uart_control_register.h	
Declaration	uint8_t R_UART1_GetHeader(uint8_t *packet_data, uint8_t rx_buffer[], uint16_t read_pos)	
Description	If data is received, acquires one byte of header data for analysis.	
Argument	uint8_t *packet_data	: header storage pointer
	uint8_t rx_buffer[]	: receive data buffer address
	uint16_t read_pos	: receive data buffer read position
Return Value	uint8_t	: 0 = Invalid, 1 = Valid
Reference doc.	RL78/G13 Serial Array Unit (UART Communication) R01AN0459EJ	
Additional Notes	When the UART communication function goes to the reception wait state, this function is called according to the API reception data analysis process. If the size of the newly received data is more than one byte, the value of the oldest received one byte data in unread data from receive data buffer is assigned to the header storage pointer for analyzing header data. The function is called by loop processing until the header data matches the SAIC101 reception data type or it goes to time-out. Global variable g_uart1_rx_count is used in this function. This variable is declared as serial module file [r_cg_sau.c] or [r_cg_serial.c] by the code generation tool. The function initializes the value by calling UART1 reception status initialization function (R_UART1_Receive) and then adds the number of bytes received in the reception interrupt function.	

```
uint8_t R_UART1_GetHeader( uint8_t *packet_data, uint8_t rx_buffer[], uint16_t
read_pos )
{
    uint8_t ret = 0U;

    if (read_pos < g_uart1_rx_count)
    {
        *packet_data = rx_buffer[read_pos];
        ret = 1;
    }

    return (ret);
}
```

Comparison of receive data buffer reading position specified in argument and current receive data counter value (g_uart1_rx_count)

If received data is stored after the reading position, the data from the reading position is stored in the pointer.

If received data is stored after the reading position, returns 1.

[Function Name] R_UART1_GetData

Outline	Specified receive data count complete check function	
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h, r_sa_uart_control_register.h	
Declaration	uint8_t R_UART1_Getdata(uint16_t rx_cnt)	
Description	Checks for data exceeding specified receive data count.	
Argument	uint16_t rx_cnt	: specified receive data count
Return Value	uint8_t	: 0 = Invalid, 1 = Valid
Reference doc.	–	
Additional Notes	When the UART communication function goes to the reception wait state and the header data is analyzed in the API reception data analysis process, the function checks if the number of reception data specified in argument rx_cnt is received at the time of SAIC101 reception data type 1 or 2. The function is called by loop processing until it receives the number of data that matches SAIC101 reception data type or it goes to time-out.	

```
uint8_t R_UART1_Getdata(uint16_t rx_cnt)
{
    uint8_t ret = 0U;

    if (rx_cnt <= g_uart1_rx_count)
    {
        ret = 1U;
    }

    return (ret);
}
```

Comparison of receive data buffer count specified in argument and current receive data counter value (g_uart1_rx_count)

If received data exceeds specified receive data count, returns 1.

4.3.2 SPI Communications

Specifications of functions required for SPI communications are listed below. CSI10 is used as an example in the explanations.

[Function Name] R_CSI10_Start

Outline	CSI10 operation start process
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h
Declaration	void R_CSI10_Start(void)
Description	Starts operations of the CSI corresponding to the serial array unit and goes to wait state.
Argument	None
Return Value	None
Reference doc.	RL78/G13 Serial Array Unit for 3-wire Serial I/O (SPI Master Transmission/Reception), for CubeSuite+, IAR, and e^2 studio (R01AN1367EJ)
Additional Notes	When it is not necessary to start and stop serial communication functions in the API, this function does not need to be created. In this case, always register a dummy function in the corresponding location in the global variable to store serial module information.

[Function Name] R_CSI10_Send_Receive

Outline	CSI10 data transmission/reception function
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h
Declaration	MD_STATUS R_CSI10_Send_Receive(uint8_t *txbuf, uint16_t txnum, uint8_t *rxbuf)
Description	Sets up data transmission/reception for CSI10.
Argument	uint8_t *txbuf: [transmit data buffer address] uint16_t txnum: [transmit data buffer size] uint8_t *rxbuf: [receive data buffer address]
Return Value	If [MD_OK]: transmission/reception setup completed If [MD_ARGERROR]: transmission/reception setup failure
Reference doc.	RL78/G13 Serial Array Unit for 3-wire Serial I/O (SPI Master Transmission/Reception), for CubeSuite+, IAR, and e^2 studio (R01AN1367EJ)
Additional Notes	The transmit/receive data storage buffer must be established by the user and passed by-pointer. After the number of communications specified in argument txnum is completed, execute the communication completion process described in section 4.2.2.

[Function Name] R_CSI10_Stop

Outline	CSI10 operation stop process
Header	r_cg_macrodriver.h, r_cg_sau.h, r_cg_userdefine.h
Declaration	void R_CSI10_Stop(void)
Description	Stops operations of corresponding CSI.
Argument	None
Return Value	None
Reference doc.	–
Additional Notes	When it is not necessary to start and stop serial communication functions in the API, this function does not need to be created. In this case, always register a dummy function in the corresponding location in the global variable to store serial module information.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
Rev.1.00	Feb 01, 2015	---	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0