Smart Analog IC101

R21AN0012EJ0100
Rev.1.00
Nov 01, 2014

Tutorial for Sample Code Introduction and API Builder SAIC101 (RL78/L13)

Introduction

This tutorial introduces APIs and sample codes to control Smart Analog IC101 (RAA730101), with explanations on how to use the API Builder SAIC101 coding assistance tool to assist in implementing APIs and sample codes.

Note: Smart Analog IC101 is referred to as “SAIC101” throughout this document.

Target Devices

Smart Analog IC 101 (part name: RAA730101), RL78/L13 (part name: R5F10WMGAFB)

Contents

1. Outline

This document describes actual examples of implementing API and sample code to control SAIC101 UART transmission connections using API Builder SAIC101, a coding assistance tool. The environment combines use of Renesas Starter Kit for RL78/L13 and Smart Analog IC RSK Option Evaluation Board TSA-OP-IC101, which has an onboard Smart Analog IC.

The tutorial also explains API Builder SAIC101 specifications, as it is used to simplify the editing and integration of SAIC101 APIs and sample code into a project based on the user's development environment.

2. Conditions for Confirming Operations

Operations for the devices discussed in this document have been confirmed under the following conditions.

Table 2-1 Conditions for Confirming Operations

Item	Description
Evaluation boards	• Renesas Starter Kit for RL78/L13 [R0K5010WMS900BE] — Renesas Starter Kit for RL78/L13 CPU board Abbreviation: RSK CPU Board — Renesas Starter Kit LCD Application Board V2 Abbreviation: LCD extension board • Smart Analog IC RSK Option Evaluation Board [TSA-OP-IC101] Abbreviation: TSA-OP-IC101 board
MCU	R5F10WMGAFB (RL78/L13)
Coding Assistance Tool (API Builder SAIC101)	Ver1.00
Integrated Development Environment (CubeSuite+)	V2.02.01 [20 Jun 2014]
C Compiler (CubeSuite+)	CA78K0R V4.02.00.03 [16 Jan 2014]
RL78/L13 Code Library (CubeSuite+)	V1.02.01.02 [11 Jun 2014]Note1
Integrated Development Environment (e2studio)	V3.0.0.22
C Compiler (e2studio)	GNURL78 v14.01
RL78/L13 Code Library (e2studio)	V1.02.00.03 [11 Feb 2014]Note2

Note 1: The CubeSuite+ code library is included in the code generator plug-in. The environment described in this document has been confirmed with CubeSuite+ Code_Generator for RL78_78K V2.04.00.

Note 2: The e2studio code library is included with the e2studio product.

3. Usage Flow of Coding Assistance Tool API Builder SAIC101

The following flow shows the integration procedure for an API or sample code using API Builder SAIC101.

1. First create a new project in CubeSuite+ or e2studio. Using the code generation tool, set the MCU peripheral functions for the serial array unit or other modules, and generate code. When this is complete, close the project.
2. Read the project created in Step 1 with API Builder SAIC101.
3. On API Builder SAIC101, set the serial connection between the SAIC and MCU and the sample code output.
4. Based on the information set in Step 3, API Builder SAIC101 outputs a file with the API or sample code, and automatically integrates the information into the project file created in Step 1.
5. Build, run and confirm the project that now integrates the API or sample code from Step 4.

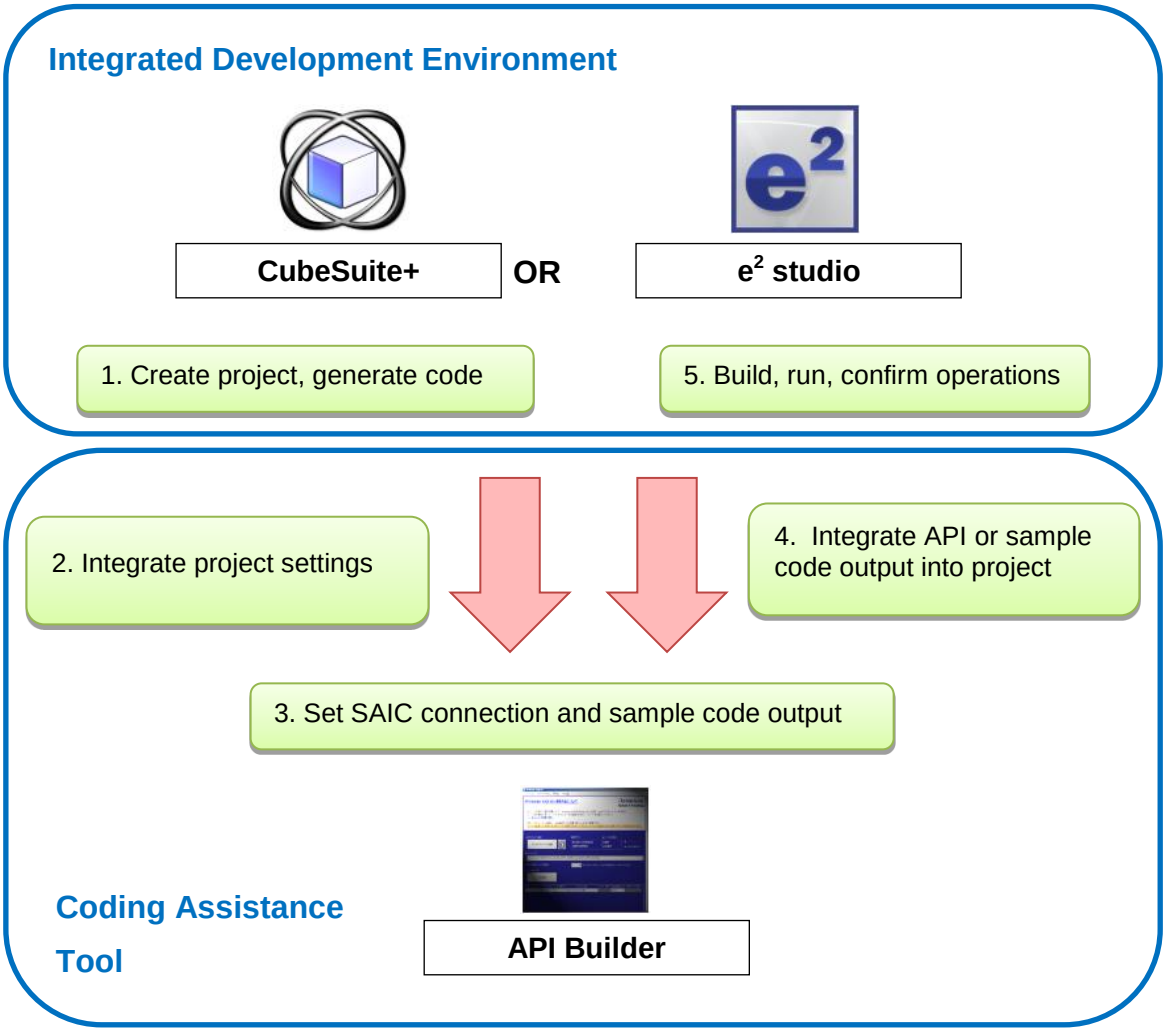

Figure 3-1 API Builder SAIC101 Usage Flow

4. SAIC101 API/Sample Code Integration Tutorial

This section describes how the CubeSuite+ code generation function and the API Builder SAIC101 are used to create and run sample code to operate the thermistor mounted on the Smart Analog IC RSK Option Evaluation Board [TSA-OP-IC101] (referred to as TSA-OP-IC101 board). The development environment[Note] consists of the TSA-OP-IC101 board, the Renesas Starter Kit for RL78/L13 CPU Board (referred to as RSK CPU board), and the Renesas Starter Kit LCD Application Board V2 (referred to as LCD extension board). The tutorial describes an example that uses a UART connection between the TSA-OP-IC101 board and the MCU.

Note: Refer to Smart Analog IC101: Smart Analog Easy Starter 101 Tutorial (RL78/L13) (R21AN0011EJ) for details on board connection settings.

4.1 Cautions for CubeSuite+ Usage

If **Code generation (design tool)** does not appear in the CubeSuite+ project tree, select **Tool (T) → Plug-in control (P)…** from the CubeSuite+ menu bar. In the **Plug-in control** window, go to the **Additional functions** tab and select **Code generation plug-in 1** and **Code generation plug-in 2**. Finally, reboot the CubeSuite+ program.

Carefully set the debug tool when confirming operations. The default setting of the debug tool is displayed in the CubeSuite+ project tree: **RL78 Simulator (debug tool)**. Bring your cursor over the default name and right click to select the emulator you are using from **Debug tools (D)**. For example, to select emulator E1, select **RL78 E1 (Serial) (L)**.

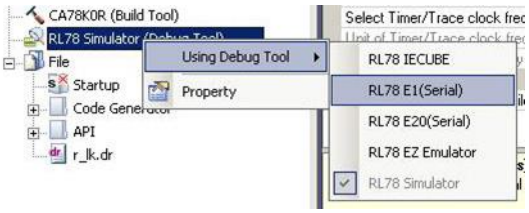

Figure 4-1 Setting the Debug Tool

4.2 Creating a Project and Setting Code Generation with CubeSuite+

Using the CubeSuite+ code generation function, first create a project template and set the MCU peripheral functions. Continue by following the procedure from 4.2.1on.

4.2.1 Create a Project

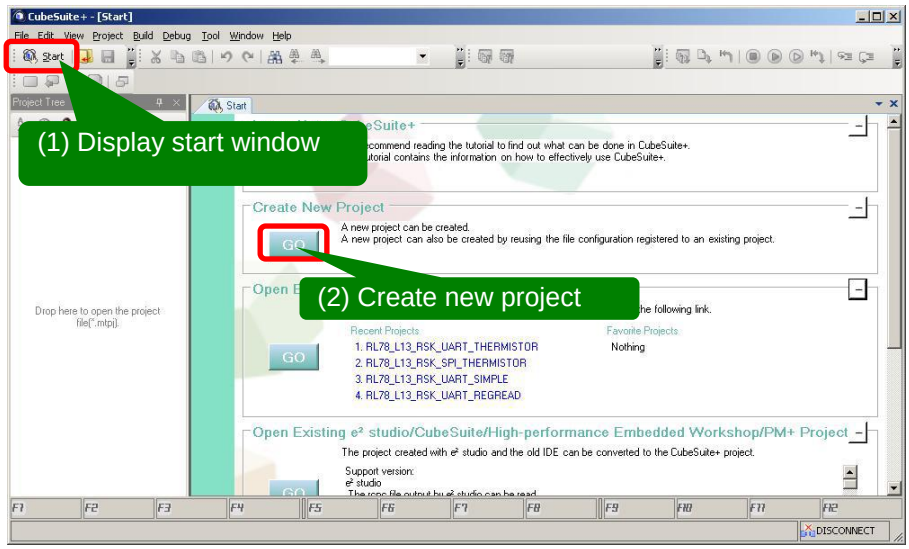

Figure 4-2 Create a Project

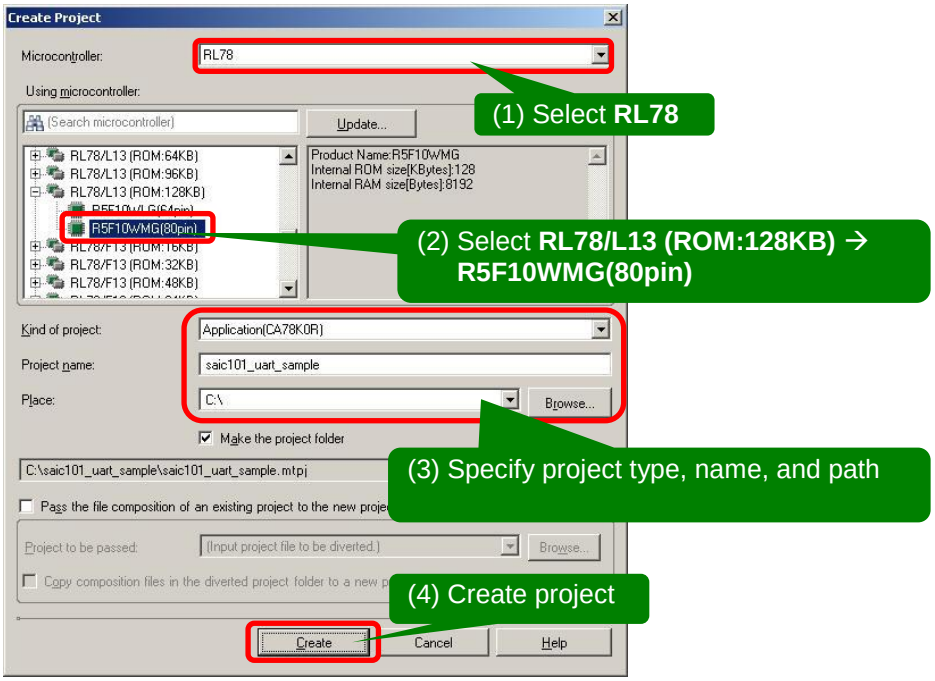

Figure 4-3 CubeSuite+ Project Creation Window

4.2.2 Code Generation Process (Design Tool): setting and execution

Set the peripheral functions of the code generation design tool as follows.

1. Set peripheral I/O redirection register (PIOR)

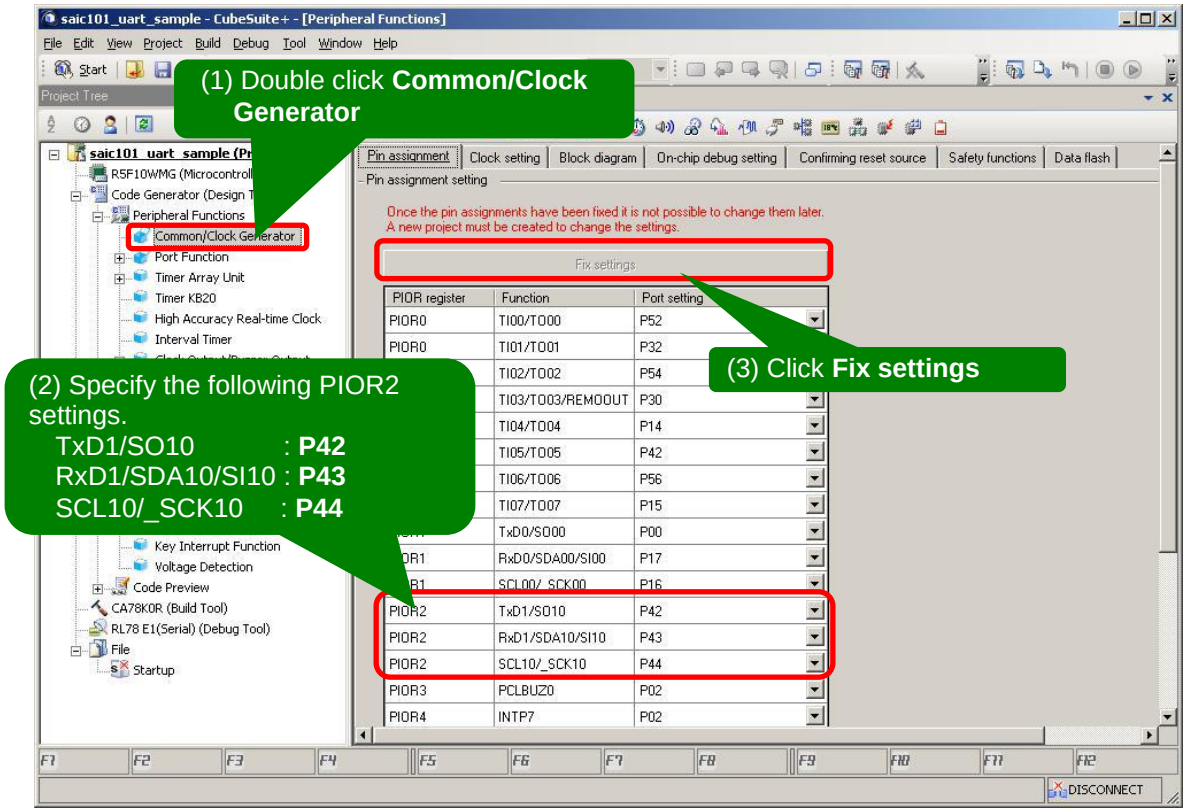

Figure 4-4 Peripheral Functions - Pin Assignments

2. Set serial array unit used for SAIC101 communications

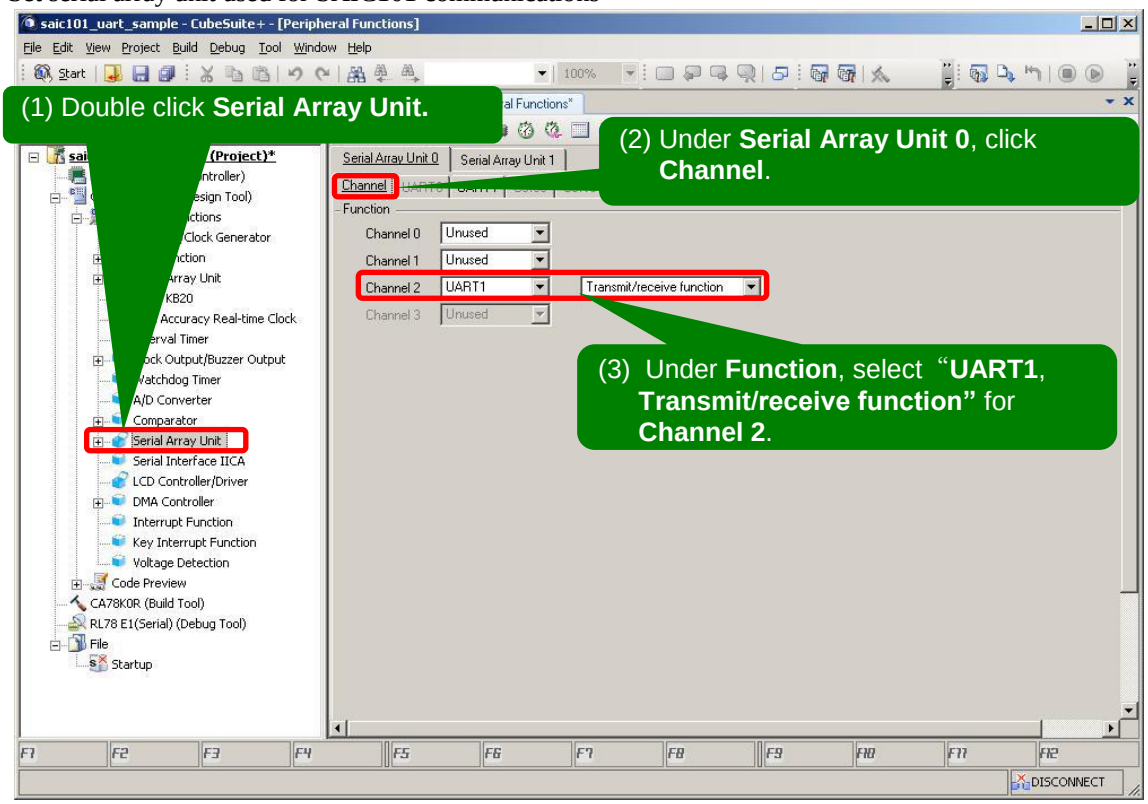

Figure 4-5 Serial Array Unit 0: Channel Setting

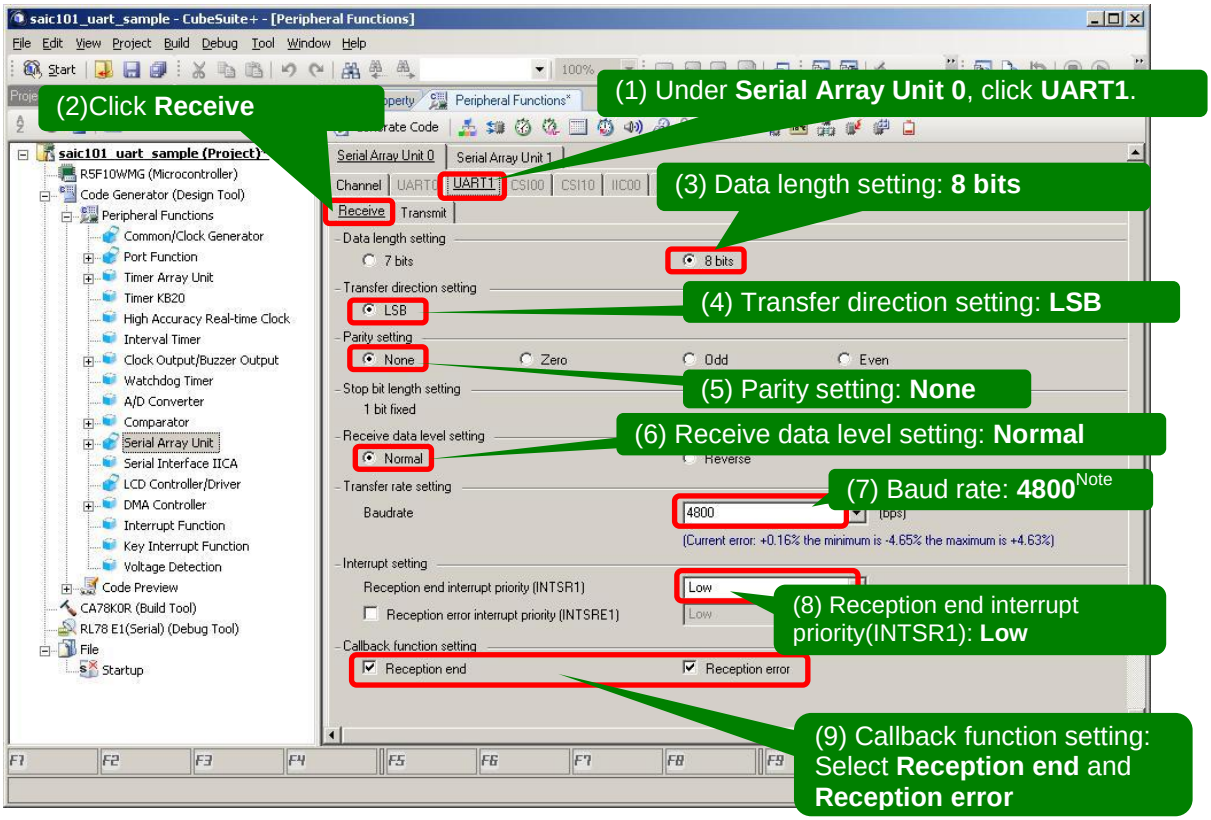

Figure 4-6 UART1 Setting, Receive Setting

Note: Specify baud rate manually; do not select value from the pull down menu.

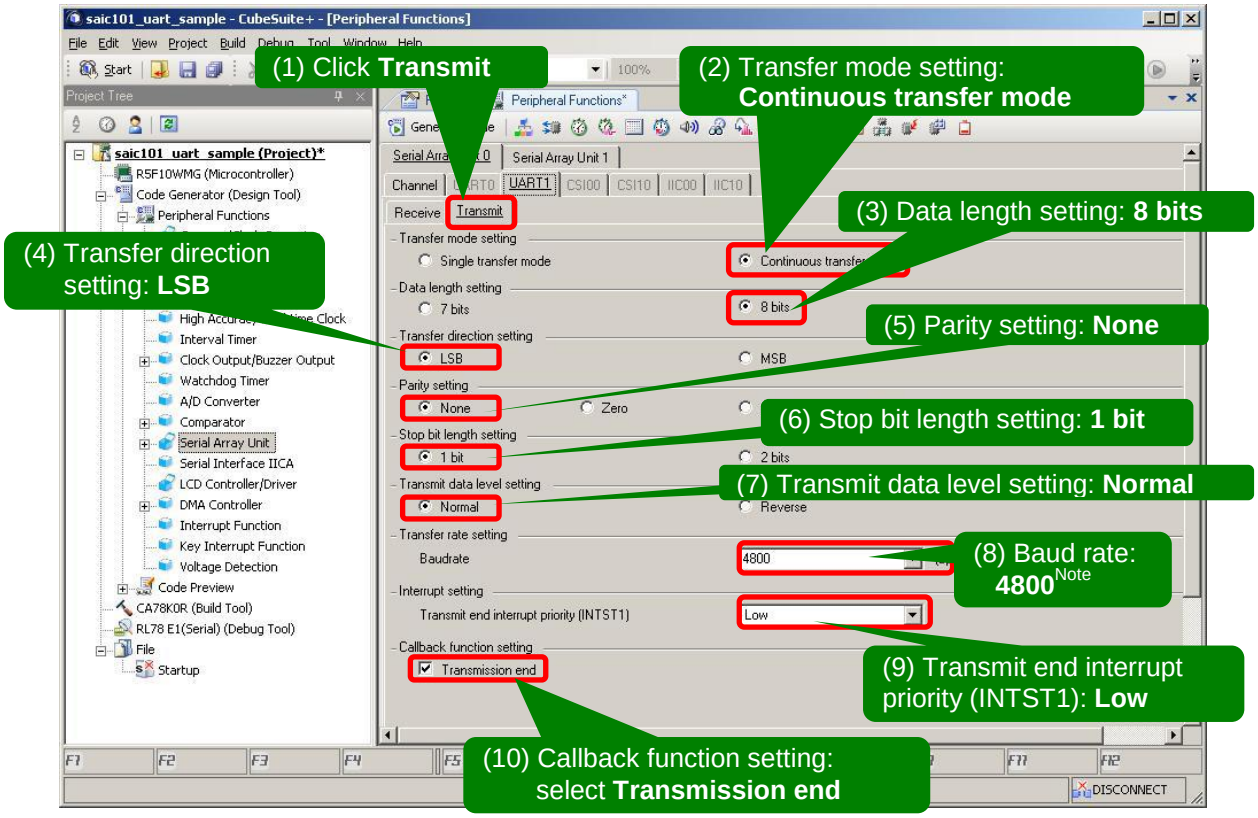

Figure 4-7 UART1 Setting, Send Setting

Note: Specify baud rate manually; do not select value from the pull down menu.

3. Set watchdog timer

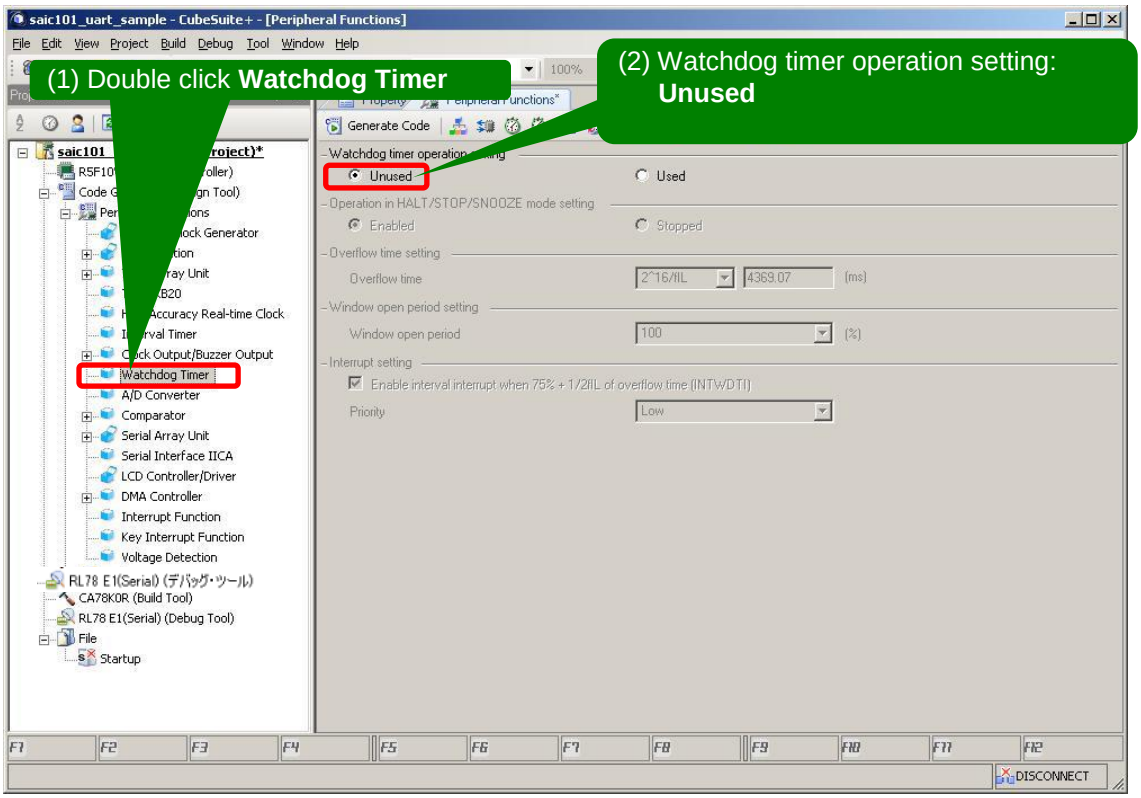

Figure 4-8 Watchdog Timer Setting

4. Set LCD control/driver

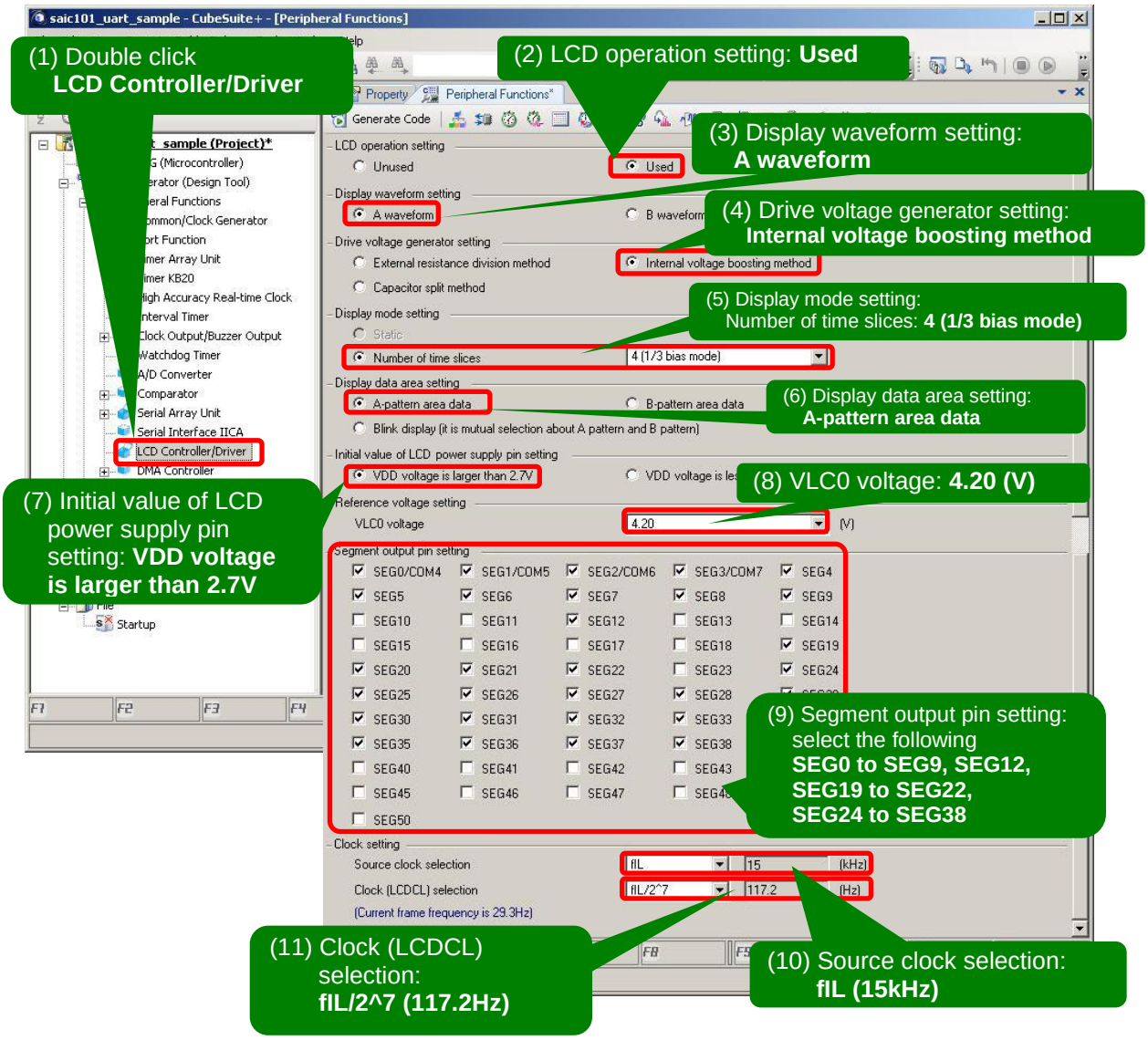

Figure 4-9 LCD Controller/Driver Settings

5. Generate code

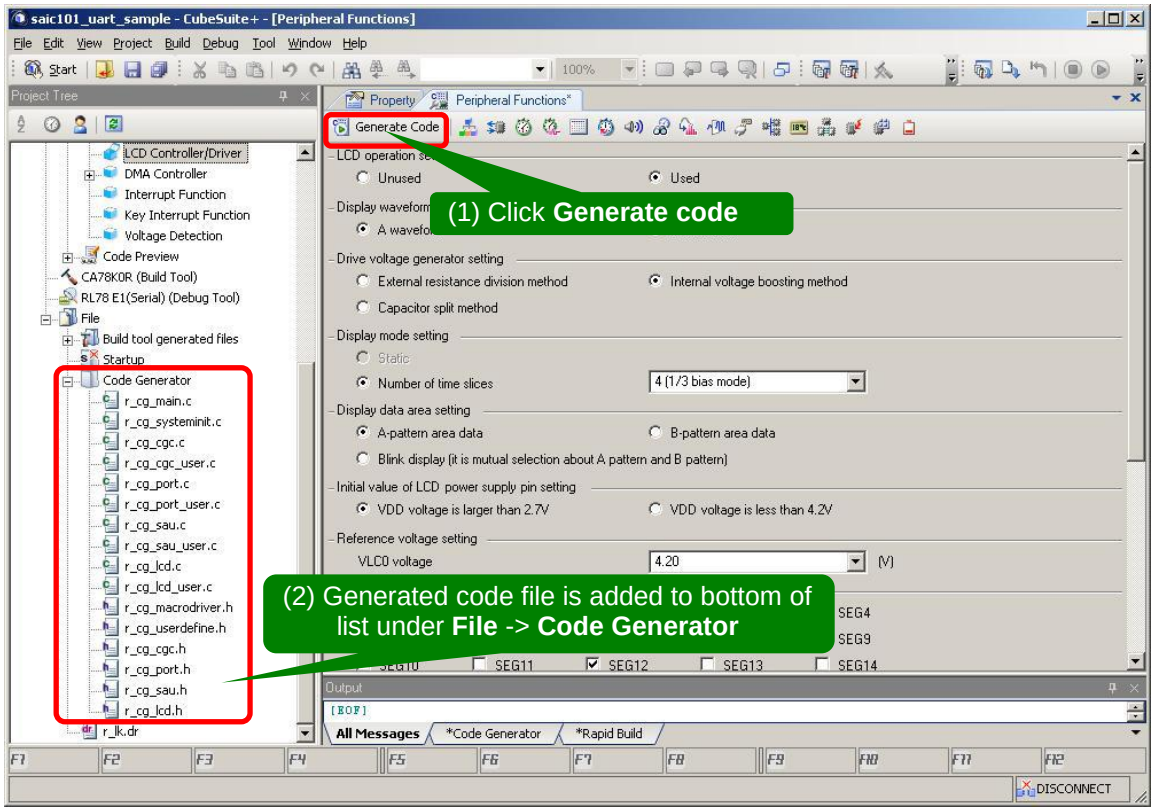

Figure 4-10 Code Generation Window

6. Save all changes made in CubeSuite+, close window.

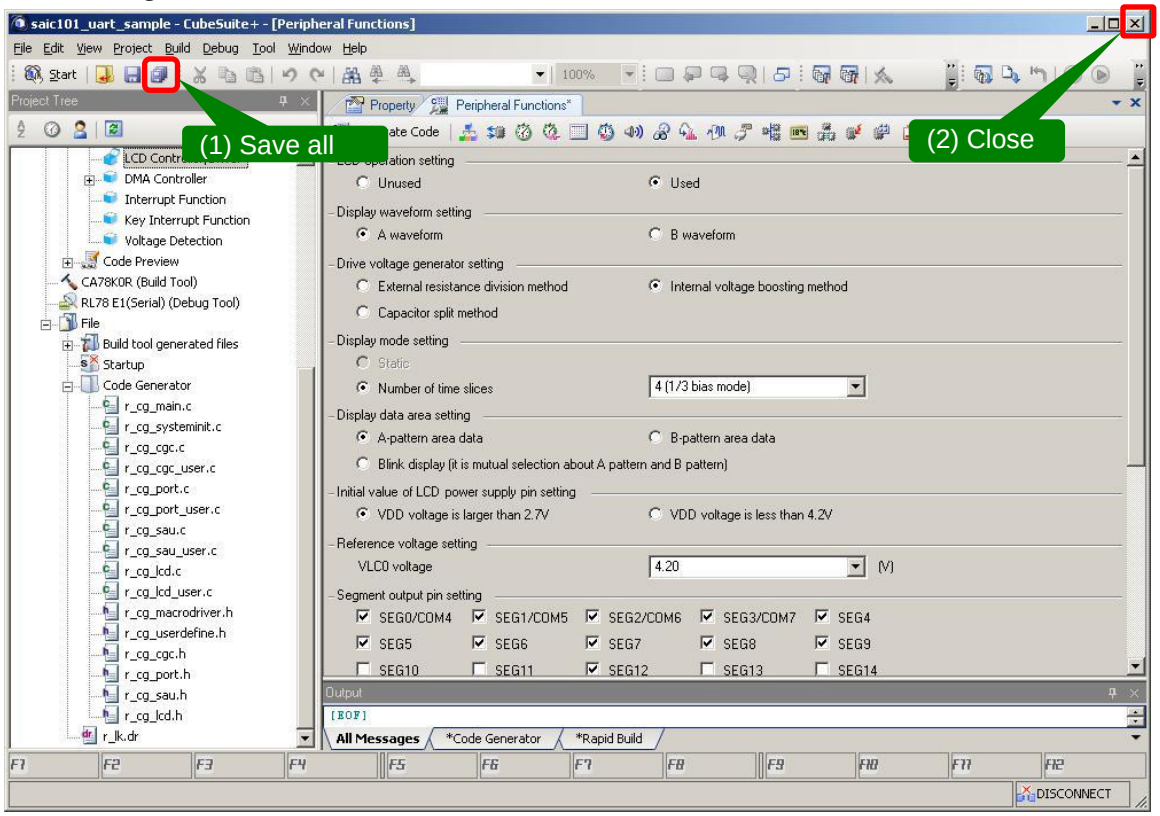

Figure 4-11 Save and Close

4.3 How to Use API Builder SAIC101

API Builder SAIC101 adds an API file and changes the source code for projects or source code created in 4.2.2. After implementing the initial settings to register the file, make sure the project file in CubeSuite+ is closed before executing the following process.

1. Start up API Builder SAIC101

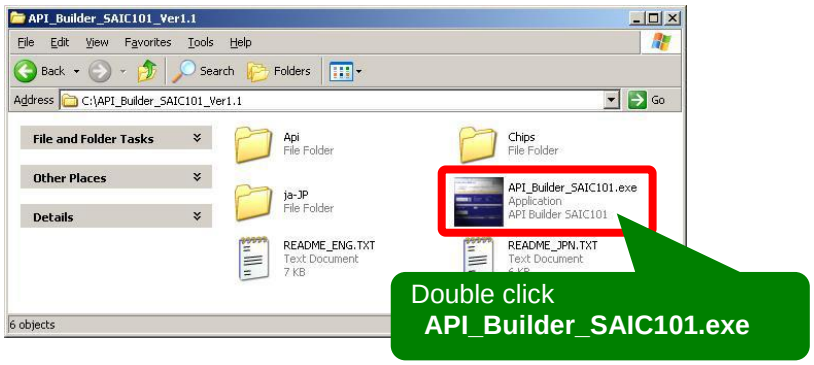

Figure 4-12 Icon Screen

2. Read and display information for project to be integrated in the API.

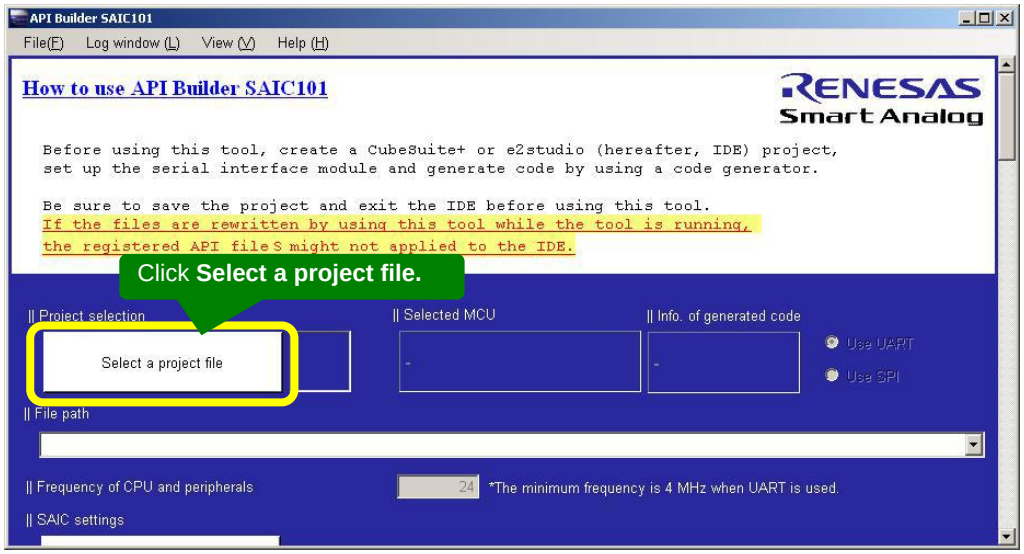

Figure 4-13 Project File Read Operation

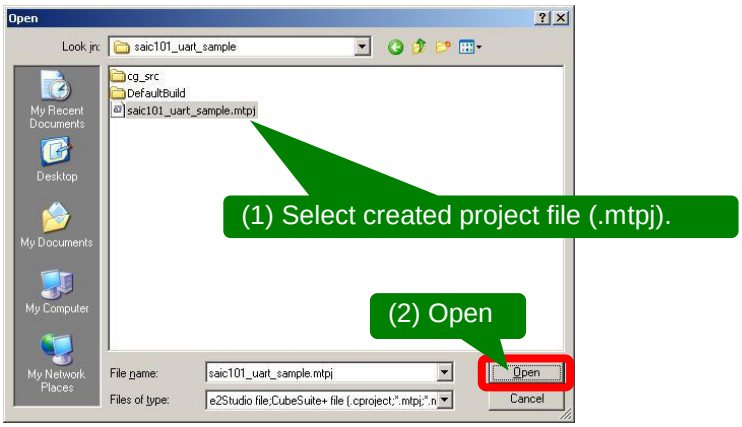

Figure 4-14 File Open Dialog

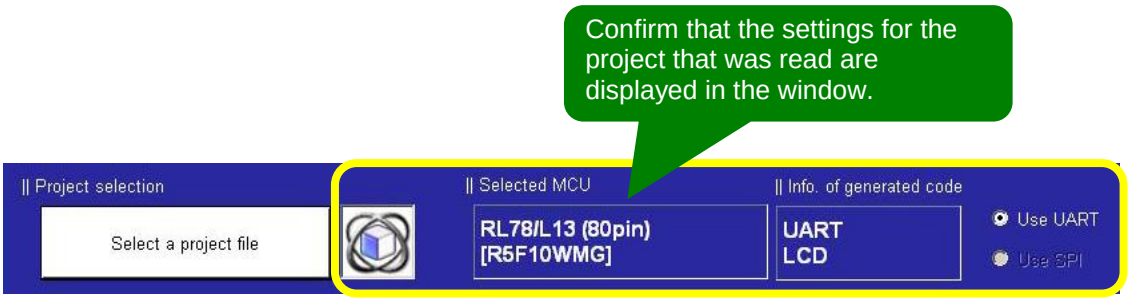

Figure 4-15 Project Information Confirmation

3. Set SAIC101 connection, set sample code output, execute file output

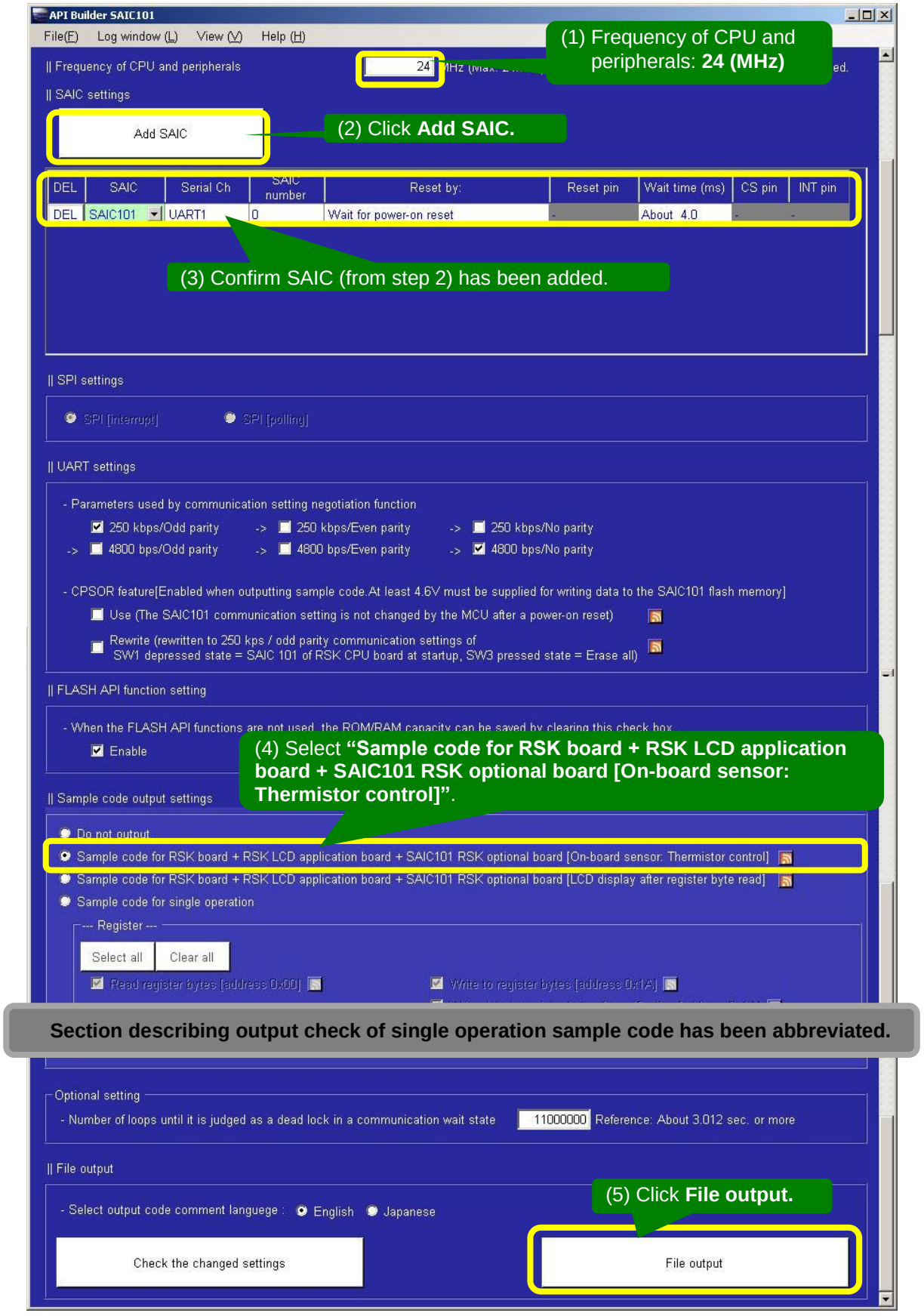

Figure 4-16 Tool Setting Window

4. Start up CubeSuite+ to confirm that the API files have been integrated

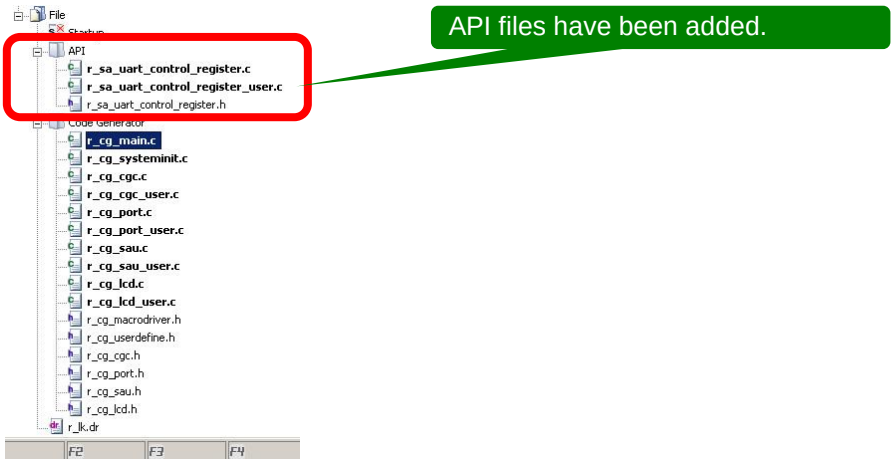

Figure 4-17 API File Integration Confirmation

5. Confirm that the thermistor control sample has been added to the **r_cg_main.c** in **main** function.

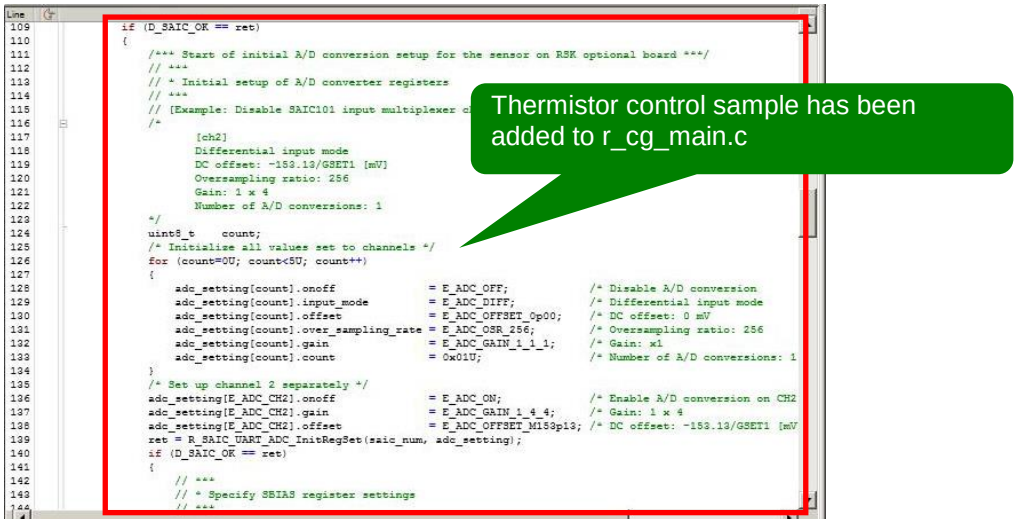

Figure 4-18 Thermistor Control Sample Code Output Confirmation (r_cg_main.c)

6. Confirm source code build

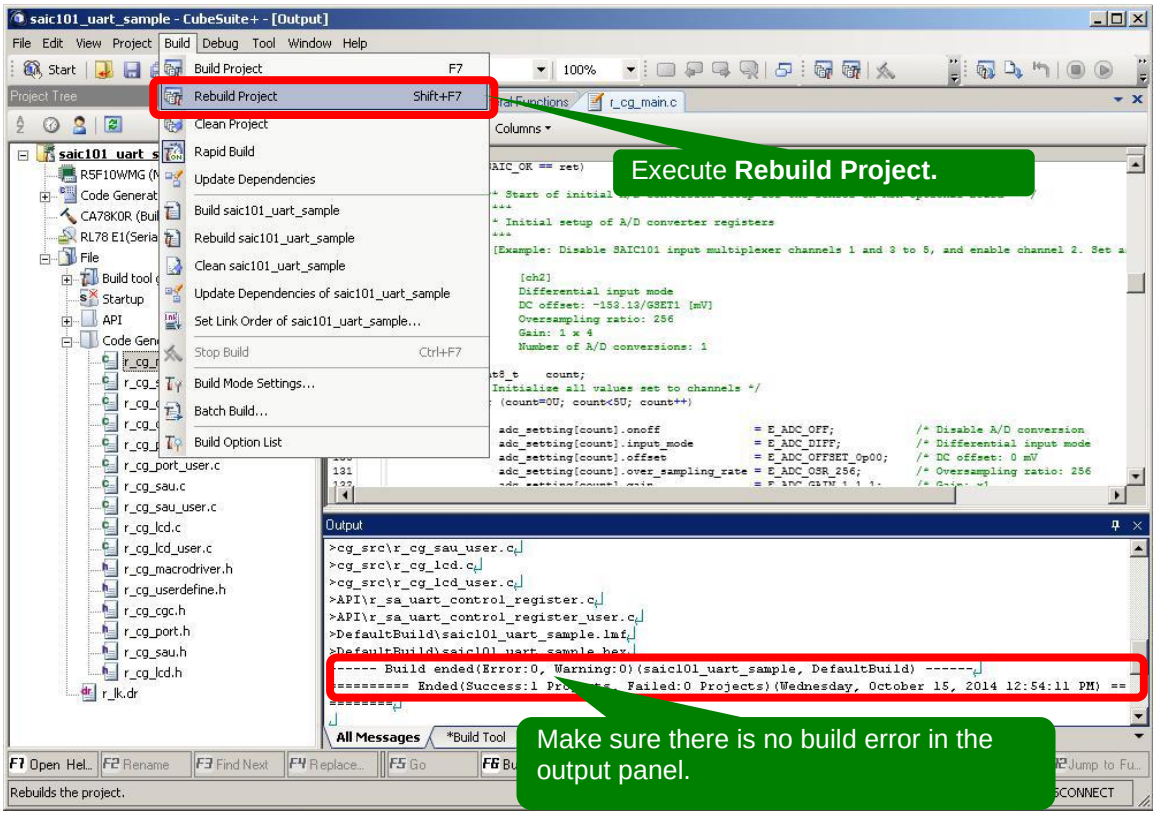

Figure 4-19 CubeSuite+ Rebuild Project Result Window

7. Connect target board, download program, then execute sample code

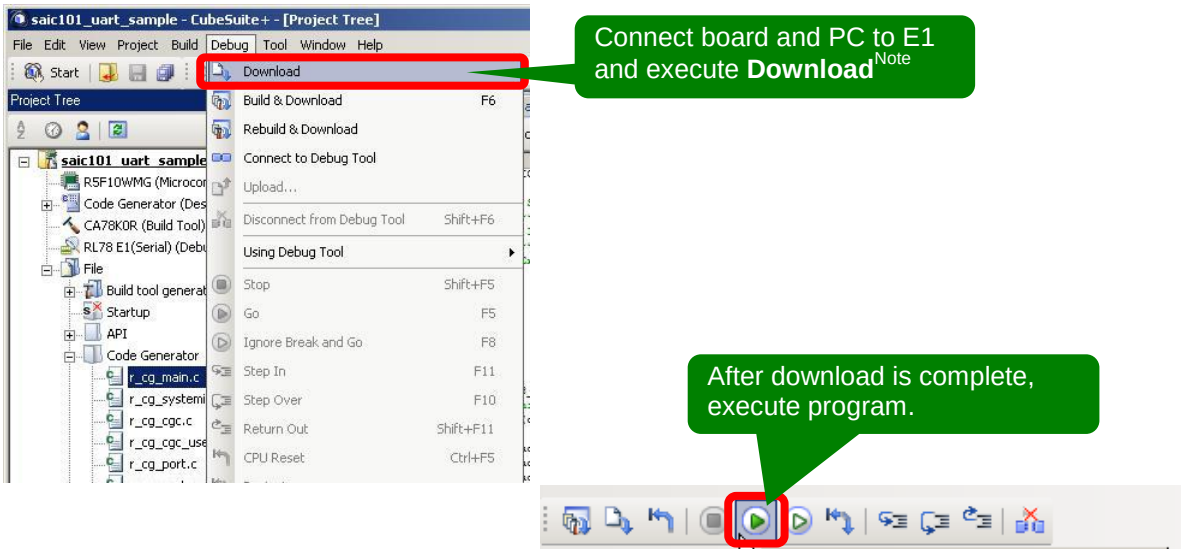

Figure 4-20 CubeSuite+ Code Download/Execute Window

Note: To supply power from the emulator, set the following before implementing the process in Step 7: Go to RL78 E1 (Serial) (Debug tool) → Property → Target board connection, set Supply power from emulator (max 200mA) to Yes and Supply voltage to 5.0V.

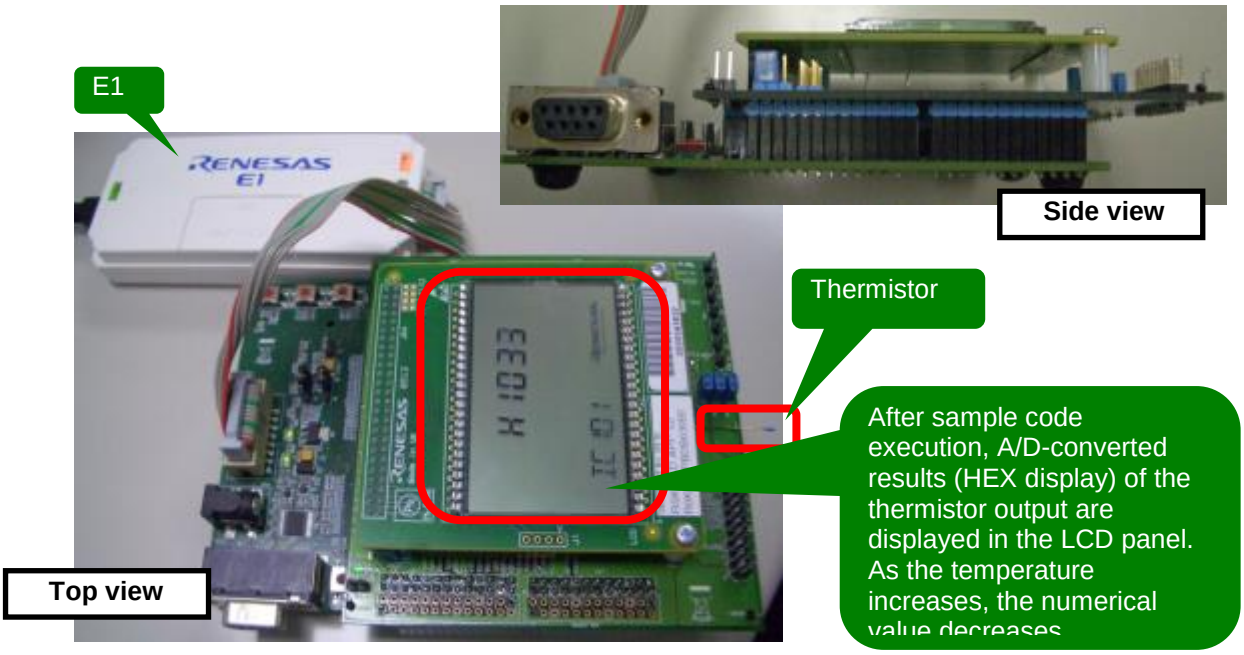

Figure 4-21 Confirming Operations

5. API Builder SAIC101

5.1 Outline

The API Builder SAIC101 is a coding assistance tool that helps developers configure an API for Smart Analog IC101, generate the API sample code, and integrate the code into the user project. User-friendly GUI enables operations, such as editing API to meet user environments, selecting sample code, and outputting data to the debugger.

5.2 System Configuration

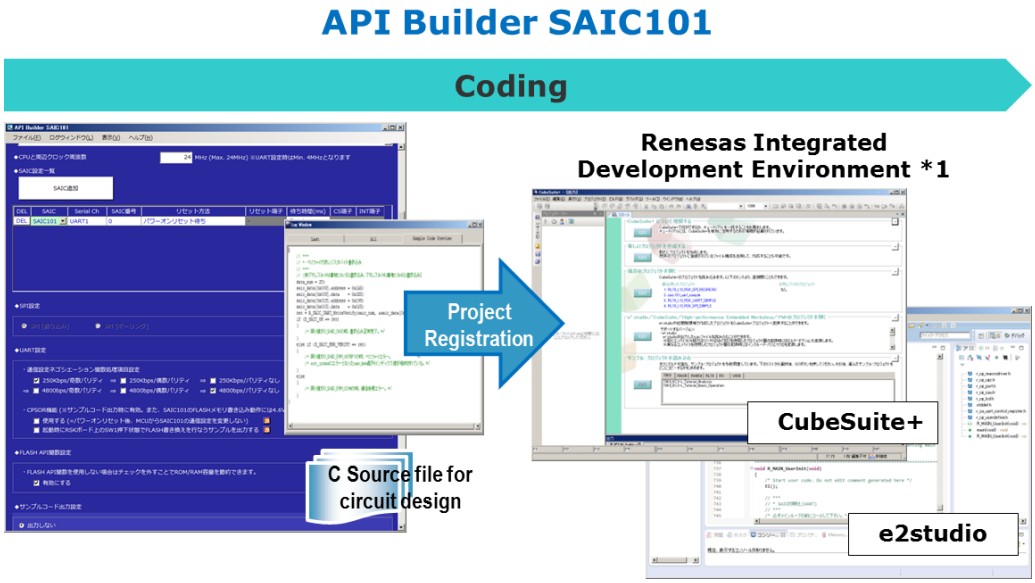

Figure 5-1 System Configuration

Note1: Currently supported integrated development environments: Renesas Electronics' CubeSuite+ and e2studio.

5.3 Major Functions

- Auto-integration of SAIC101 sample code to project file created in CubeSuite+ or e2 studio.
- Easy editing and integration of developer-specific serial interface with user-friendly GUI.
- Select SAIC101 sample code and generate sample C source code for Renesas Starter Kit.[Note]

Note: API Builder SAIC101 currently only supports RL78/L13.

5.4 Support Environment

Table 5-1 Support Environment

Supported OS	• Windows® 8 (32-bit, 64-bit versions) • Windows® 7 (32-bit, 64-bit versions)
Essential software environment in addition to Windows OS	.Net Framework 4 and later

5.5 Target MCUs

- RL78/L13

  R5F10WLA
  R5F10WLC
  R5F10WLD
  R5F10WLE
  R5F10WLF
  R5F10WLG
  R5F10WMA
  R5F10WMC
  R5F10WMD
  R5F10WME
  R5F10WMF
  R5F10WMG

5.6 Window Configuration Explanation

The API Builder SAIC101 windows are configured as follows.

1. Project selection and information display

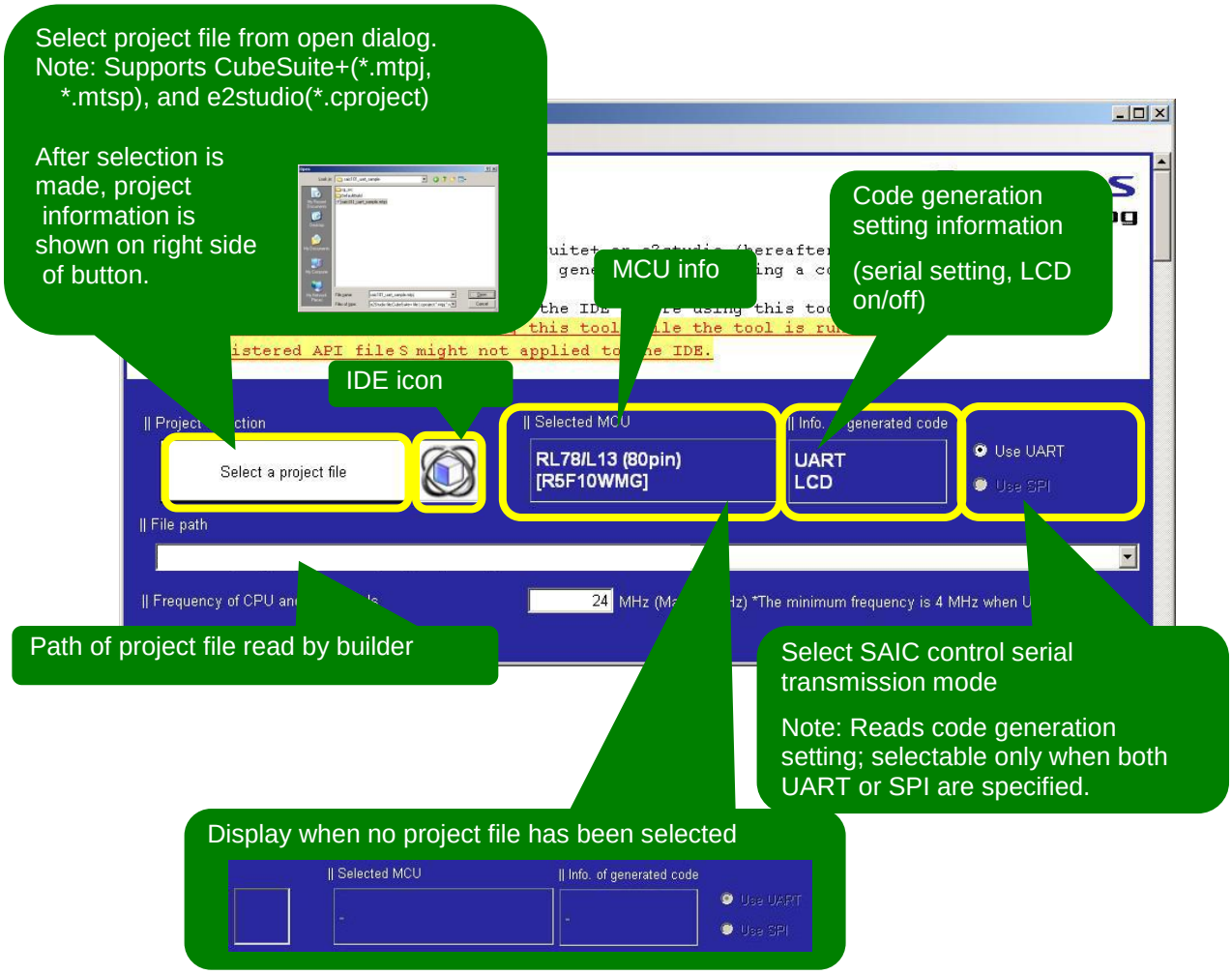

Figure 5-2 File Read/Display Update Confirmation

2. Set hardware-dependent areas and select sample code output

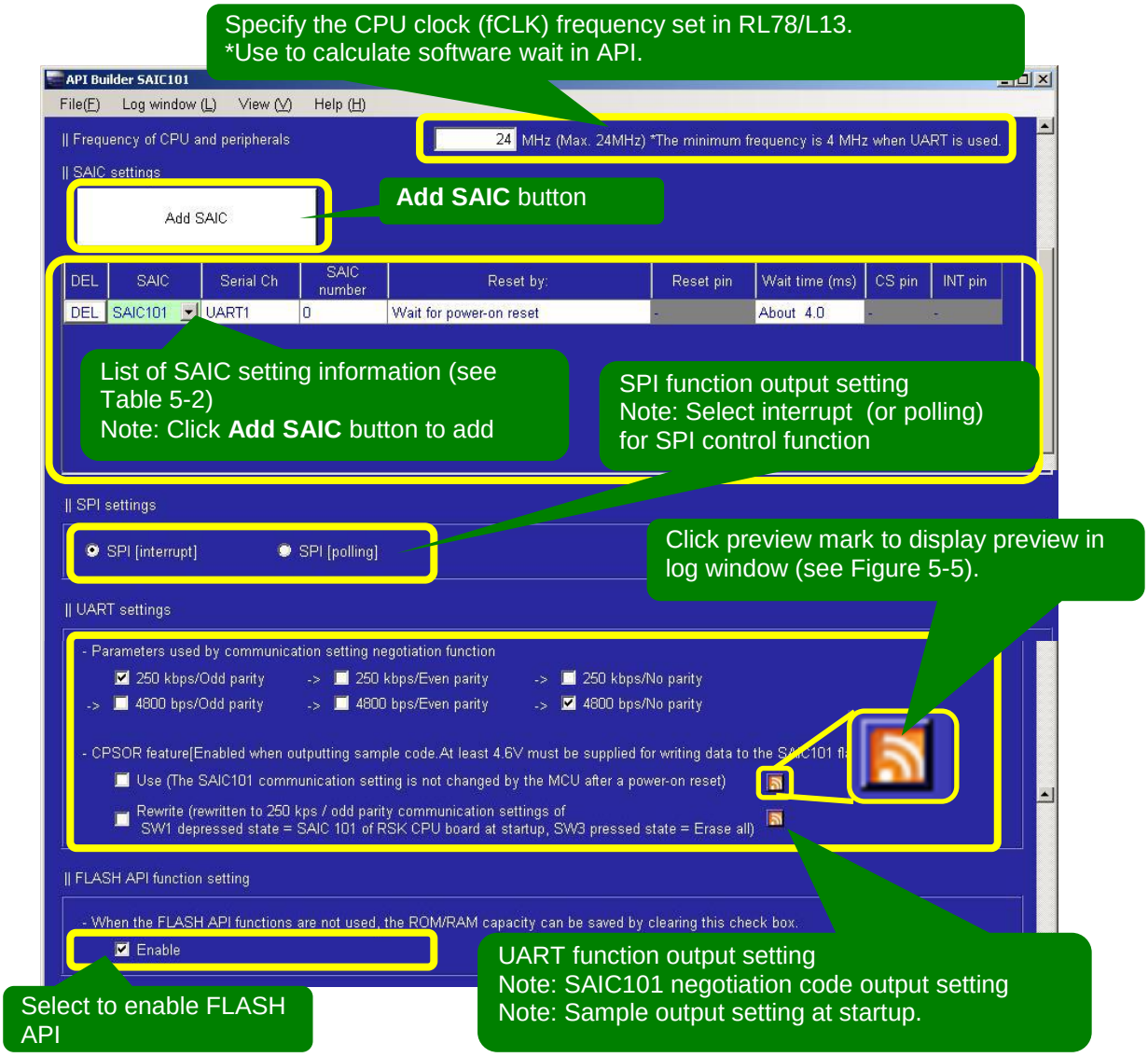

Figure 5-3 Hardware-dependent Setting and Sample Code Output Selection (1/2)

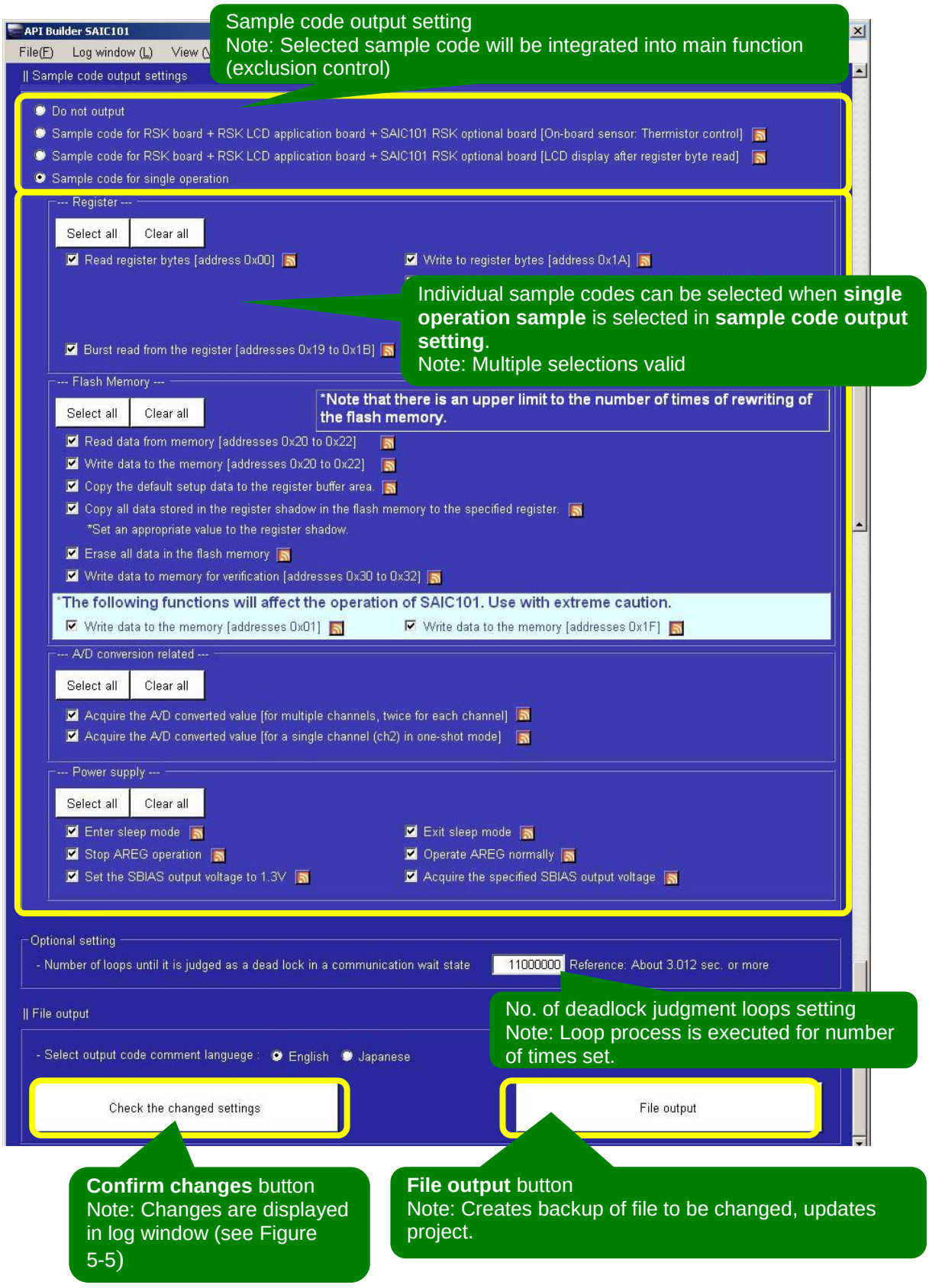

Figure 5-4 Hardware-dependent Setting and Sample Code Output Selection (2/2)

Table 5-2 SAIC Setting Information

Item	Description	Notes
DEL	Deletes a line that corresponds to the list of SAIC setting information	—
SAIC	Specifies SAIC part name to be connected to board	For SPI communications: select SAIC101, SAIC300, SAIC301, SAIC500, SAIC501, or SAIC502 from pull-down menu. For UART communications: only SAIC101 can be selected
Serial Ch	Specifies serial channel	Select serial channels from pull-down menu that have been set during code generation (See 4.2.2)
SAIC number	Number specified as argument when calling API function	Can be set within unsigned char type range (0 to 255)[Note1]
Reset by:	Specifies reset process at SAIC startup	Select the following from the pull-down menu. • Power-on reset wait[Note2] Waits for the period specified by **"Wait time (ms)"** • External reset (RESET port = L)[Note3] Outputs period L specified by **"Wait time (ms)"** to the port specified in **"Reset pin**". • Internal reset (RESET register = 1)[Note3] Set 1 to the RESET bit of the SAIC reset control register (RC) to clear it.
Reset pin[Note3]	Specifies the port connected to the RESET pin	The RESET pin can be specified when external reset has been selected in "**Reset by:".**. The RESET pin must be set when using external reset; otherwise an error will occur when a file is output.
Wait time (ms)	Specifies the wait period length when power-on reset wait is selected in **"Reset by:".** When external reset is selected, specify a period that sets the RESET pin to L.	The period can be set within the float-type range.
CS pin	Specifies the port connected to the CS pin.	Can only be set when using SPI communications. The CS pin must be set when using SPI; otherwise an error will occur when a file is output.
INT pin	Specifies the port connected to the INT pin.	Can only be set when using SPI communications and SAIC101.

Note1: When using the sample code, set to 0, as the SAIC number is fixed as 0 when calling API functions.

Note2: The power-on reset function is only supported for SAIC101 and SAIC502.

Note3: Cannot be set when SAIC101 is specified in the SAIC field.

5.7 Log Window

This section describes the log window.

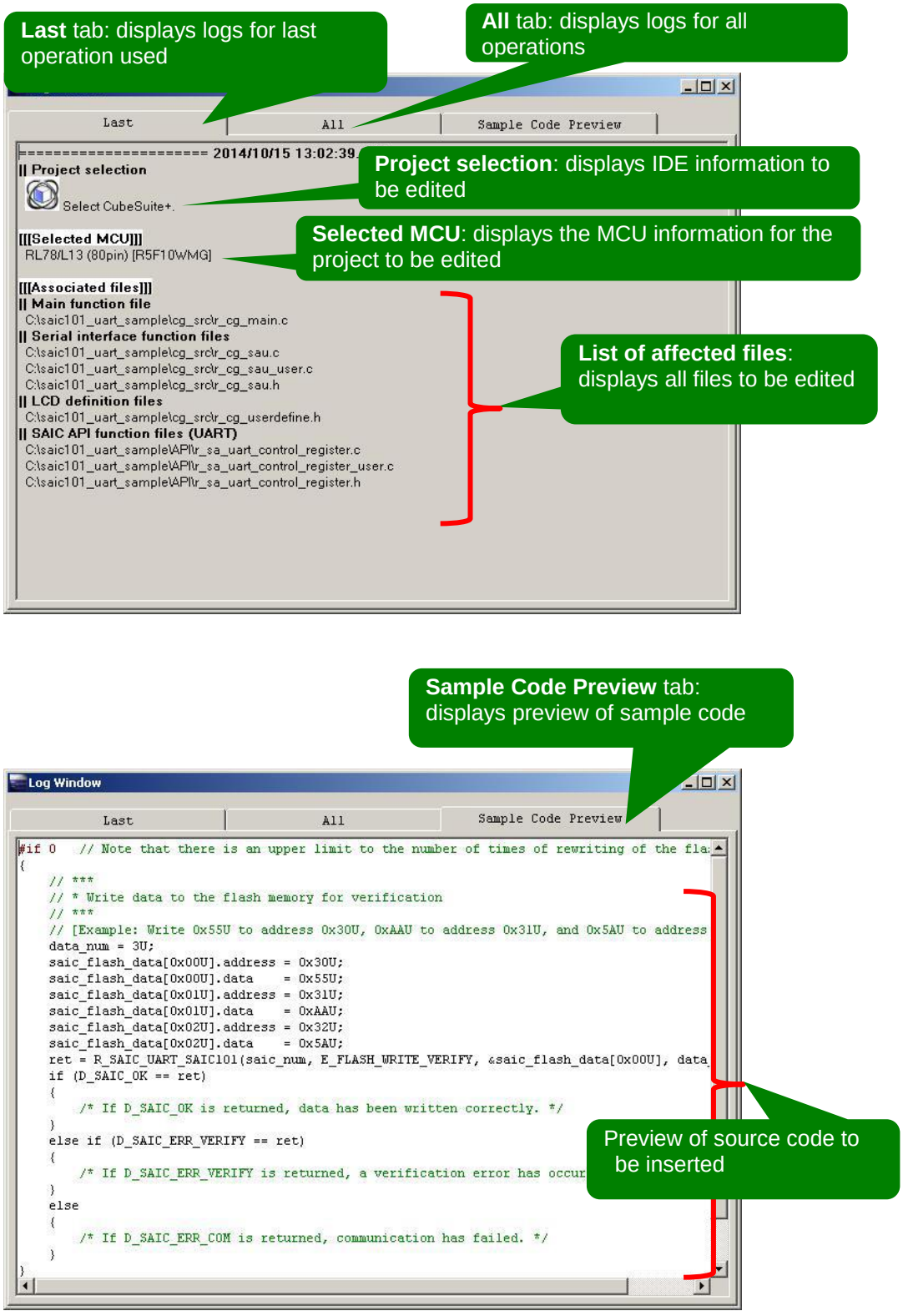

Figure 5-5 Log Window

5.8 Error Messages

Table 5-3 provides descriptions and notes for each error message. When a fatal error occurs, the configuration file may be corrupted. If this happens, reinstall the file from the API Builder SAIC101 ZIP file provided by Renesas.

Table 5-3 Error Messages

No	Error Window	Notes
1	[Fatal Error] The CSV file format is not correct. There is a possibility that the configuration file is corrupted.	This tool stores MCU-specific information in a CSV file. An error indicates that the CSV file format read for the MCU is incorrect.
2	[Error] The serial interface setting has not been generated.	This API uses functions output by the code generation tool. Before executing the tool, set the serial interface (UART/SPI) in the code generation tool, then generate the code.
3	[Error] There is no configuration file for the corresponding MCU.	An MCU information file for this tool is not available. Please check the Chips folder to make sure there is an MCU information file selected in the current project. If no file exists, the MCU is not supported by the tool.
4	[Error] There is no SAIC setting.	Displayed during file output if no SAICs are selected. Make sure you specify at least one SAIC.
5	[Error] Write error has occurred. [Affected write file name]	Displayed when an error occurs during the file write operation. The name of the file triggering the error is displayed.
6	[Error] The SAIC number specified for SPI is duplicated. The SAIC number specified for UART is duplicated.	Displayed when the same number is used for two or more SAICs. Always make sure unique numbers are set for each SAIC.
7	[Error] Port: PX.X is duplicated.	Displayed when more than one pin is assigned as the CS pin or the RESET pin. Carefully confirm the circuit settings on the hardware and specify the correct port number.
8	[Error] The CS pin is not set.	Displayed when the CS pin is not set. SPI communications require the CS pin to be set to ensure normal operations.
9	[Error] The reset time is not correct.	Displayed when an invalid value is set as the wait period (ms). Please enter a correct value.
10	[Error] The reset pin setting is not correct.	Displayed when the RESET pin is not set correctly. Please set the correct pin name.
11	[Error] Multiple SAICs are specified for one UART.	Only one SAIC can be set for each UART channel. Make sure you only set one SAIC to each channel.
12	[Input Error] Enter a value from 0 to 255.	The valid range for SAIC numbers is 0 to 255. Please set a value from within that range.

6. Integrating API Functions without Using API Builder SAIC101

Table 6-1 provides a list of files used for setting API Builder SAIC101when integrating APIs using the combined RSK CPU board and TSA-OP-IC101 board environment. If you are not using API Builder SAIC101, the source codes shown in this section will need to be changed manually.

This section describes integrating APIs for UART communications. The same descriptions can be applied to SPI communications by simply replacing uart/UART with spi/SPI. In addition, this example uses UART1 for UART and CSI10 SPI communications.

Table 6-1 Files Required for Setting API Integration

File Name	Notes
r_cg_main.c	Code generation file
r_cg_sau.c	Code generation file
r_cg_sau.h	Code generation file
r_cg_sau_user.c	Code generation file
r_sa_uart_control_register.c	API file[Note]
r_sa_uart_control_register.h	API file[Note]
r_sa_uart_control_register_user.c	API file[Note]

Note: Copy the API files from the sample code. The sample code can be downloaded from the following URL. UART files are stored in the " \an_r21an0014jj0100_saic_usefulexample\UART\source" folder; and SPI files are stored in the "\an_r21an0014jj0100_saic_usefulexample\SPI\source" folder.

Sample code download URL:
http://www.renesas.com/products/smart_analog/smart_analog_ic/smart_analog_ic_101/Application_Notes.jsp

File name: an_r21an0014ej0100_saic_usefulexample.zip

The following shows the contents of each file.

— r_cg_main.c modifications

- [UART/SPI] Add include definitions for API functions.

```
#include <stddef.h>
#include "r_sa_uart_control_register.h"
```

- [UART/SPI] Add API initialization function to R_MAIN_UserInit function.

```
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();

    // ***
    // * SAIC Initialization(UART)
    // ***
    /* Be sure to call this function prior to the main loop.*/
    R_SAIC_UART_Init();    /* SmartAnalogIC Initialize. */

    /* End user code. Do not edit comment generated here */
}
```

— r_cg_sau_user.c modifications

- [UART/SPI] Add include definition for API functions.

```
#include "r_sa_uart_control_register.h"
```

- [UART/SPI] Add global variables in bit control file.

```
static const uint8_t  gs_bit_tbl[] =
{
    0x01U, 0x02U, 0x04U, 0x08U, 0x10U, 0x20U, 0x40U, 0x80U,
};
```

- [UART only] Add global variables across flag files used by API function.

```
uint8_t g_uart_tx_end_flag = 0U;
uint8_t g_uart_rx_end_flag = 0U;
```

- [SPI only] Add global variable across flag files used by API function.

```
uint8_t g_csi_overrun_flag = 0U;
```

- [UART only] Add flag update process in r_uart1_callback_receiveend function.

```
static void r_uart1_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    g_uart_rx_end_flag |= gs_bit_tbl[E_UART1];
    /* End user code. Do not edit comment generated here */
}
```

- [SPI only] (when using communication module interrupt) Add CS=H or communication end process to r_csi10_callback_receiveend function.

```
static void r_csi10_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    R_SAIC_SPI_CSDisable(E_CSI10);
    R_CSI10_Stop();
    /* End user code. Do not edit comment generated here */
}
```

- [SPI only] (when using communication module interrupt) Add flag update, CS=H or communication end process to r_csi10_callback_error function.

```
static void r_csi10_callback_error(uint8_t err_type)
{
    /* Start user code. Do not edit comment generated here */
    g_csi_overrun_flag |= gs_bit_tbl[E_CSI10];
    R_SAIC_SPI_CSDisable( E_CSI10 );
    R_CSI10_Stop();
    /* End user code. Do not edit comment generated here */
}
```

- [UART only] Add flag update process to r_uart1_callback_sendend function.

```
static void r_uart1_callback_sendend(void)
{
    /* Start user code. Do not edit comment generated here */
    g_uart_tx_end_flag |= gs_bit_tbl[E_UART1];
    /* End user code. Do not edit comment generated here */
}
```

- [UART only] Add R_UART1_SettingChange function called from API.

```
void R_UART1_SettingChange(uint8_t setting)
{
    e_uart_setting_t uart_setting = (e_uart_setting_t)setting;
    switch (uart_setting)
    {
        /*   4800bps, Parity=None */
        case E_UART_4800bps_None:
            SPS0 = 0x0055U;             /* Serial clock selection register */
            SDR02 = 0x9A00U;            /* Baud rate setting                */
            SDR03 = 0x9A00U;            /* Baud rate setting                */
            SCR02 &= ~0x0300U;          /* Parity setting                   */
            SCR03 &= ~0x0300U;          /* Parity setting                   */
            break;
        /*   4800bps, Parity=Odd  */
        case E_UART_4800bps_Odd:
            SPS0 = 0x0055U;             /* Serial clock selection register */
            SDR02 = 0x9A00U;            /* Baud rate setting                */
            SDR03 = 0x9A00U;            /* Baud rate setting                */
            SCR02 |= 0x0300U;           /* Parity setting                   */
            SCR03 |= 0x0300U;           /* Parity setting                   */
            break;
        /*   4800bps, Parity=Even  */
        case E_UART_4800bps_Even:
            SPS0 = 0x0055U;             /* Serial clock selection register */
            SDR02 = 0x9A00U;            /* Baud rate setting                */
            SDR03 = 0x9A00U;            /* Baud rate setting                */
            SCR02 &= ~0x0300U;          /* Parity setting                   */
            SCR03 &= ~0x0300U;          /* Parity setting                   */
            SCR02 |= 0x0200U;           /* Parity setting                   */
            SCR03 |= 0x0200U;           /* Parity setting                   */
            break;
        /* 250000bps, Parity=None */
        case E_UART_250kbps_None:
            SPS0 = 0x0000U;             /* Serial clock selection register */
            SDR02 = 0x5E00U;            /* Baud rate setting                */
            SDR03 = 0x5E00U;            /* Baud rate setting                */
            SCR02 &= ~0x0300U;          /* Parity setting                   */
            SCR03 &= ~0x0300U;          /* Parity setting                   */
            break;
```

```
        /* 250000bps, Parity=Odd  */
        case E_UART_250kbps_Odd:
            SPS0 = 0x0000U;             /* Serial clock selection register */
            SDR02 = 0x5E00U;            /* Baud rate setting                */
            SDR03 = 0x5E00U;            /* Baud rate setting                */
            SCR02 |= 0x0300U;           /* Parity setting                   */
            SCR03 |= 0x0300U;           /* Parity setting                   */
            break;
        /* 250000bps, Parity=Even  */
        case E_UART_250kbps_Even:
            SPS0 = 0x0000U;             /* Serial clock selection register */
            SDR02 = 0x5E00U;            /* Baud rate setting                */
            SDR03 = 0x5E00U;            /* Baud rate setting                */
            SCR02 &= ~0x0300U;          /* Parity setting                   */
            SCR03 &= ~0x0300U;          /* Parity setting                   */
            SCR02 |= 0x0200U;           /* Parity setting                   */
            SCR03 |= 0x0200U;           /* Parity setting                   */

            break;
    }
}
```

- [UART only] Add R_UART1_GetHeader function called from API.

```
/***********************************************************************
* Function Name: R_UART1_GetHeader
* Description  : This function returns the process header data
received by the UART1.
* Arguments    : uint8_t *packet_data -
*                    Header data
*              : uint8_t rx_buffer[] -
*                    Receive buffer
*              : uint16_t read_pos -
*                    Buffer read position
* Global Value : g_uart1_rx_count
*                    Number of received data in the UART1
* SFR          : None
* Return Value : uint8_t -
*                    0=Invalid, 1=Valid
***********************************************************************/
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);
}
```

- [UART only] Add R_UART1_Getdata function called from API.

```
/***********************************************************************
* Function Name: R_UART1_Getdata
* Description  : This function check of bytes of data that is not
less than the number specified has been received.
* Arguments    : uint16_t rx_cnt -
*                    number of bytes specified data
* Global Value : g_uart1_rx_count
*                    Number of received data in the UART1
* SFR          : None
* Return Value : uint8_t -
*                    0=Invalid, 1=Valid
***********************************************************************/
uint8_t R_UART1_Getdata(uint16_t rx_cnt)
{
    uint8_t ret = 0U;

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

    return (ret);
}
```

- [SPI only] (when using polling) Add R_CSI10_MaskStart function called from API.

```
/***********************************************************************
* Function Name: R_CSI10_MaskStart
* Description  : This function starts the CSI10 module operation.
* Arguments    : None
* Return Value : None
***********************************************************************/
void R_CSI10_MaskStart(void)
{
    SO0 |= _0400_SAU_CH2_CLOCK_OUTPUT_1;/* CSI10 clock initial level */
    SO0 &= ~_0004_SAU_CH2_DATA_OUTPUT_1;    /* CSI10 SO initial level */
    SOE0 |= _0004_SAU_CH2_OUTPUT_ENABLE;    /* enable CSI10 output */
    SS0 |= _0004_SAU_CH2_START_TRG_ON;     /* enable CSI10 */
    CSIIF10 = 0U;    /* clear INTCSI10 interrupt flag */
    CSIMK10 = 1U;    /* disable INTCSI10 interrupt */
}
```

— r_cg_sau.h modifications

- [UART only] Add extern declaration of the function added to r_cg_sau.c to enable reference from API.

```
extern void    R_UART1_SettingChange(uint8_t setting);
extern uint8_t R_UART1_GetHeader(uint8_t *packet_data, uint8_t
rx_buffer[], uint16_t read_pos);
extern uint8_t R_UART1_Getdata(uint16_t rx_cnt);
```

- [SPI only] Add extern declaration of the function added to r_cg_sau.c to enable reference from API.

```
extern uint8_t g_csi_overrun_flag;
```

- [SPI only] (when using polling) Add extern declaration of the function added to r_cg_sau.c to enable reference from API.

```
extern void R_CSI10_MaskStart(void);
```

— r_sa_uart_control_register.h modifications

- [SPI only] Use either communications module interrupt or polling function by commenting out one of the two. (The polling setting is disabled in UART so the interrupt usage is fixed.)

```
#define D_SPI_OPERATION D_SPI_USE_INTERRUPT         /* Use of interrupts by
communication modules    */
//#define D_SPI_OPERATION D_SPI_REGISTER_POLLING      /* No use of interrupts by
communication modules */
```

- [UART only] Set UART negotiation function process items. Comment out unnecessary items.

```
#define D_UART_NEGOTIATION_250KBPS_PARITY_ODD      /* UART baudrate=250000bps, Parity=Odd  */
#define D_UART_NEGOTIATION_250KBPS_PARITY_EVEN     /* UART baudrate=250000bps, Parity=Even */
#define D_UART_NEGOTIATION_250KBPS_PARITY_NONE     /* UART baudrate=250000bps, Parity=None */
#define D_UART_NEGOTIATION_4800BPS_PARITY_ODD      /* UART baudrate=4800bps, Parity=Odd    */
#define D_UART_NEGOTIATION_4800BPS_PARITY_EVEN     /* UART baudrate=4800bps, Parity=Even   */
#define D_UART_NEGOTIATION_4800BPS_PARITY_NONE     /* UART baudrate=4800bps, Parity=None   */
```

- [UART/SPI] Enable/disable Flash-related API. To disable, comment out.

```
#define D_SAIC_FLASH_API_VALID                 /* Enabling FLASH API functions */
```

- [UART/SPI] Set number of loops to judge deadlock during communication wait with SAIC101.

```
#define D_DEADLOCK_CNT      (11000000L)     /* Number of loops
until it is judged as a dead lock in a communication wait state */
```

- [UART only] Set UART ch definition. Modify to match MCU's number of UART module channels.

```
typedef enum
{
    E_UART0 = 0x00U,                    /* UART0 */
    E_UART1,                             /* UART1 */
    E_UART2,                             /* UART2 */
    E_UART3,                             /* UART3 */
    E_UART_MAX                          /* Maximum value judgment */

} e_uart_ch_t;
```

- [SPI only] Set CSI ch definition. Modify to match MCU's number of SPI module channels.

```
typedef enum
{
    E_CSI00 = 0x00U,                    /* CSI00        */
    E_CSI01,                            /* CSI01        */
    E_CSI10,                            /* CSI10        */
    E_CSI11,                            /* CSI11        */
    E_CSI20,                            /* CSI20        */
    E_CSI21,                            /* CSI21        */
    E_CSI30,                            /* CSI30        */
    E_CSI31,                            /* CSI31        */
    E_CSI_MAX                           /* Maximum value judgment */

} e_csi_ch_t;
```

— r_sa_uart_control_register_user.c modifications

- [UART/SPI] Set include definition for code generation serial file. Modify code generation tool output file name if necessary.

```
#include "r_cg_sau.h"
```

- [UART/SPI] Set CPU CLK (MHz). Modify as needed based on MCU setting.

```
#define D_CPU_CLK_MHZ                ( 24.0F)     /* Operation clock(MHz) */
```

- [UART/SPI] Set (ms) power-on reset period. Modify period as needed.

```
#define D_WAIT_PON_RST_TIME_MS        ( 4.00F)    /* Wait time (ms)        */
```

- [UART/SPI] Set NOP count calculated from power-on reset period.

```
#define D_PON_RST_NOP_CNT
((uint32_t)((D_WAIT_PON_RST_TIME_MS/(1.0F/D_CPU_CLK_MHZ))*1000.0F/7.0F))
```

- [UART only] Set global variables to store SAIC information. The variable array's index number corresponds to the SAIC number used in each API. Register the connected channel and SAIC with the ENUM value set in r_sa_uart_control_register.h.

```
const uart_saic_t g_uart_saic_data_tbl[] =
{
//  { UART_ch,    sa_type,       }, /* format                                    */
    { E_UART1,    E_SAIC101,     }, /* Information of SAIC whose SAIC number is 0 */

};/* Global variable that stores SAIC information */
```

- [SPI only] Set global variables to store SAIC information. The variable array's index number corresponds to the SAIC number used in each API. Register the connected channel and SAIC with the ENUM value set in r_sa_spi_control_register.h. Register the addresses and bits for the CS pin and INT pin as shown below. When not using the INT pin, register NULL as the pin address.

```
const spi_saic_t g_spi_saic_data_tbl[] =
{
//  { csi_ch,    sa_type,       p_cs_addr, cs_bit_num, p_int_addr, int_bit_num, }, /* format                                      */
    { E_CSI10, E_SAIC101,       &P0,       6U,         &P0,        7U,          }, /* Information of SAIC whose SAIC number is 0 */
//  { E_CSI21, E_SAIC300,       &P7,       3U,         &P7,        4U,          }, /* Information of SAIC whose SAIC number is 1 */

}; /* Global variable that stores SAIC information */
```

- [UART only] Set global variables to store serial module information. The variable array's index number corresponds to the SAIC connection channel. Register any related functions.

```
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, },
    { NULL,          NULL,         NULL,            NULL,         NULL,              NULL,            NULL,                 },
    { R_UART1_Start, R_UART1_Stop, R_UART1_Receive, R_UART1_Send, R_UART1_GetHeader, R_UART1_Getdata, R_UART1_SettingChange, },
    { NULL,          NULL,         NULL,            NULL,         NULL,              NULL,            NULL,                 },
    { NULL,          NULL,         NULL,            NULL,         NULL,              NULL,            NULL,                 },

#elif D_UART_OPERATION==D_UART_REGISTER_POLLING
    /* not supported */

#endif

}; /* global variables to store serial module information */
```

- [SPI only] Set global variables to store serial module information. The variable array's index number corresponds to the SAIC connection channel. Register the functions needed to use of the communication module interrupt. When using polling, register the communication register addresses and bits, and any related functions.

```
const spi_serial_t g_spi_serial_data_tbl[] =
{
#if D_SPI_OPERATION==D_SPI_USE_INTERRUPT
//  { CSI_Start,     CSI_Stop,     CSI_Send_Receive,     }, /* format  */
    { NULL,          NULL,         NULL,                 }, /* CSI00   */
    { NULL,          NULL,         NULL,                 }, /* CSI01   */
    { R_CSI10_Start, R_CSI10_Stop, R_CSI10_Send_Receive, }, /* CSI10   */
    { NULL,          NULL,         NULL,                 }, /* CSI11   */
    { NULL,          NULL,         NULL,                 }, /* CSI20   */
    { NULL,          NULL,         NULL,                 }, /* CSI21   */
    { NULL,          NULL,         NULL,                 }, /* CSI30   */
    { NULL,          NULL,         NULL,                 }, /* CSI31   */

#elif D_SPI_OPERATION==D_SPI_REGISTER_POLLING
    { NULL,               NULL,   NULL,  0U, NULL,               NULL,               NULL,               NULL,         }, /* CSI00    */
    { NULL,               NULL,   NULL,  0U, NULL,               NULL,               NULL,               NULL,         }, /* CSI01    */
    { (uint16_t *)&SMR02,&SIO10,&IF1L,1U,(uint16_t*)&SSR02,(uint16_t*)&SIR02,R_CSI10_MaskStart,R_CSI10_Stop,}, /* CSI10    */
    { NULL,               NULL,   NULL,  0U, NULL,               NULL,               NULL,               NULL,         }, /* CSI11    */
    { NULL,               NULL,   NULL,  0U, NULL,               NULL,               NULL,               NULL,         }, /* CSI20    */
    { NULL,               NULL,   NULL,  0U, NULL,               NULL,               NULL,               NULL,         }, /* CSI21    */
    { NULL,               NULL,   NULL,  0U, NULL,               NULL,               NULL,               NULL,         }, /* CSI30    */
    { NULL,               NULL,   NULL,  0U, NULL,               NULL,               NULL,               NULL,         }, /* CSI31    */

#endif
}; /* global variables to store serial module information */
```

- [UART/SPI] Set global variables to store RESET information. Set the Reset method. Only power-on reset can be selected for SAIC101. For the SAIC number, register the index number of the global variable array that stores the SAIC information.

```
const uart_reset_t g_uart_reset_data_tbl[] =
{
    //process,                  Port address, Bit num, nop_cnt,               spi_saic_t number,},
    { E_SAIC_POWERON_RESET,     NULL,         0U,      D_PON_RST_NOP_CNT,     0U,                },

};/* Global variable that stores RESET information */
```

Website and Support

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

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

All trademarks and registered trademarks are the property of their respective owners.

Revision History <revision history,rh>

		Description	
Rev.	**Date**	**Page**	**Summary**
Rev.1.00	Nov 01, 2014	---	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

© 2014 Renesas Electronics Corporation. All rights reserved.

Colophon 4.0