RL78 Family

CubeSuite+ Startup Guide

Introduction

The purpose of this document is to help the user understand how to use the RL78 family sample code in CubeSuite+ and also understand basic operations of development tools for the RL78 family. The user can further deepen his or her understanding of these by referring to this document during actual operation of the development tools.

This document provides an explanation by using the RL78/G13 sample codes.

Target Device

RL78/G13

Contents

1. Preface

Audience
- This document is intended for customers who for the first time use the sample code and development tools for the RL78 family. Note that basic knowledge of how to operate Windows® is required for the use of the development tools.

2. Development Environment

This section describes a development procedure and development tools under the use of the RL78 family.

2.1 Relationship between the Development Procedure and Development Tools

The figure below shows the relationship between a product development procedure and development tools.
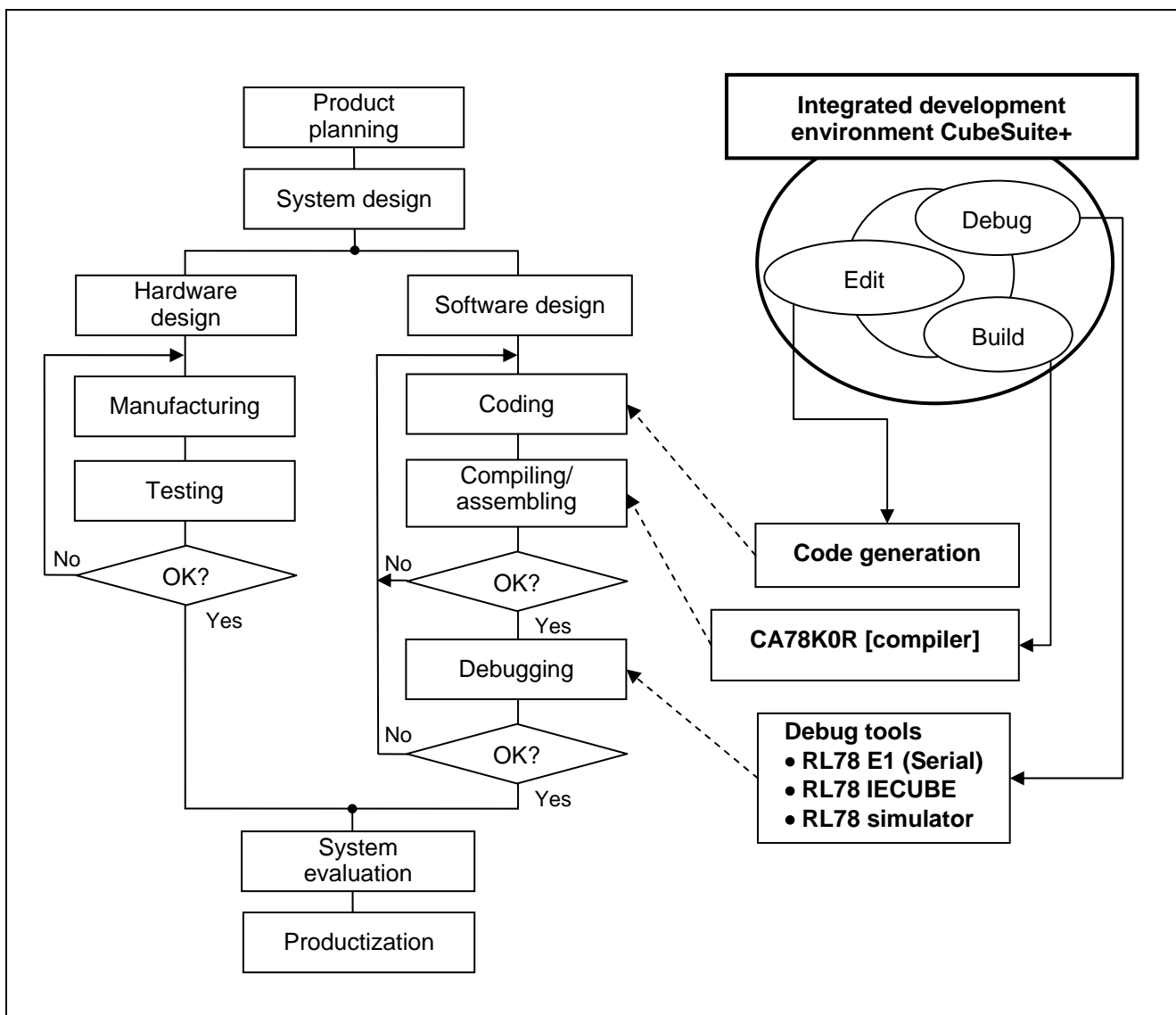

Figure 2-1   Development Procedure and Development Tools

2.2   Development Tools to be Used

To check operation of the sample code, the following development tools are used.

2.2.1 CubeSuite+ (Integrated Development Environment)

CubeSuite+ is an integrated development environment used on Windows. In combination with development tools such as editors, compilers, and debuggers, it allows efficient development. The following are provided as main tools.

a)   Device-dependent Information Files
     Files that contain device-specific information.

b)   CA78K0R (C Compiler)
     A highly versatile and portable C compiler, developed to describe in the C language a program embedded in the RL78 family. In order to use CA78K0R on Windows, CubeSuite+ is required.

c)   Debug Tools
   RL78 E1 (Serial)
     Performs on-chip debugging using an E1 emulator.

   RL78 IECUBE
     Performs debugging using a full-spec emulator IECUBE.

   CubeSuite+ RL78 Simulator
     Instruction simulator for the RL78 family.
     Capable of simulating operation of source code and interval timer functions of channels 0 to 3 of TAU on the host PC.

d)   Design Tools
   Pin Arrangement
     Function of configuring shared-pin settings on a pin arrangement diagram through GUI. Capable of inserting a pin arrangement dialog into a design document and outputting pin arrangement in Excel format.

   Code Generation
     Function capable of automatically generating according to GUI settings a device driver program that controls peripheral microcontroller functions (e.g., timer, serial I/F, A/D).

2.2.2 Typical CPU Evaluation Boards

   CPU board (e.g., QB-R5F100LE-TB (RL78/G13 with 64 pins and 64-kbyte ROM)
   Renesas Starter Kit (e.g., Renesas Starter Kit for RL78/G13)
   RL78G13 Stick starter kit (RL78G13-STICK)

3. Generating Object Module Files

Build the sample code using CubeSuite+ in order to generate object module files (HEX and LMF files) from the sample code.

This section describes a process from creating a project of the sample code to performing a build.

For details on how to create a CubeSuite+ project, refer to CubeSuite+ V1.02.00 Integrated Development Environment User's Manual: Start (R20UT0975).

Note: In the sample code for the RL78 family, it is impossible to change the device product name specified first in the project. Therefore, to change the device product name in the project, create a new project.

Remarks: 1. To use the downloaded original project of the sample code, double-click the file "<the name of the downloaded file> + <file extension> [*.mtpj]" to open the CubeSuite+ project. In this case, follow the procedure described in section 3.2, Setting Option Bytes.

2. A load module file (LMF) is an object file that includes debug information such as symbols.

3. A hex file is a hex-format object file generated from a LMF file and does not include debug information.

3.1   Creating a New Project and Reading the Sample Code

3.1.1   Preparing the Sample Code

Decompress the downloaded file and save it into a desired location (folder).

3.1.2   Starting CubeSuite+ and Preparing for Creating a New Project

When you start CubeSuite+ and click [Start (S)] on the toolbar, the start panel is opened. After the start panel is opened, click the [GO] button below the "Create New Project", shown in figure 3-1, to create a new project.
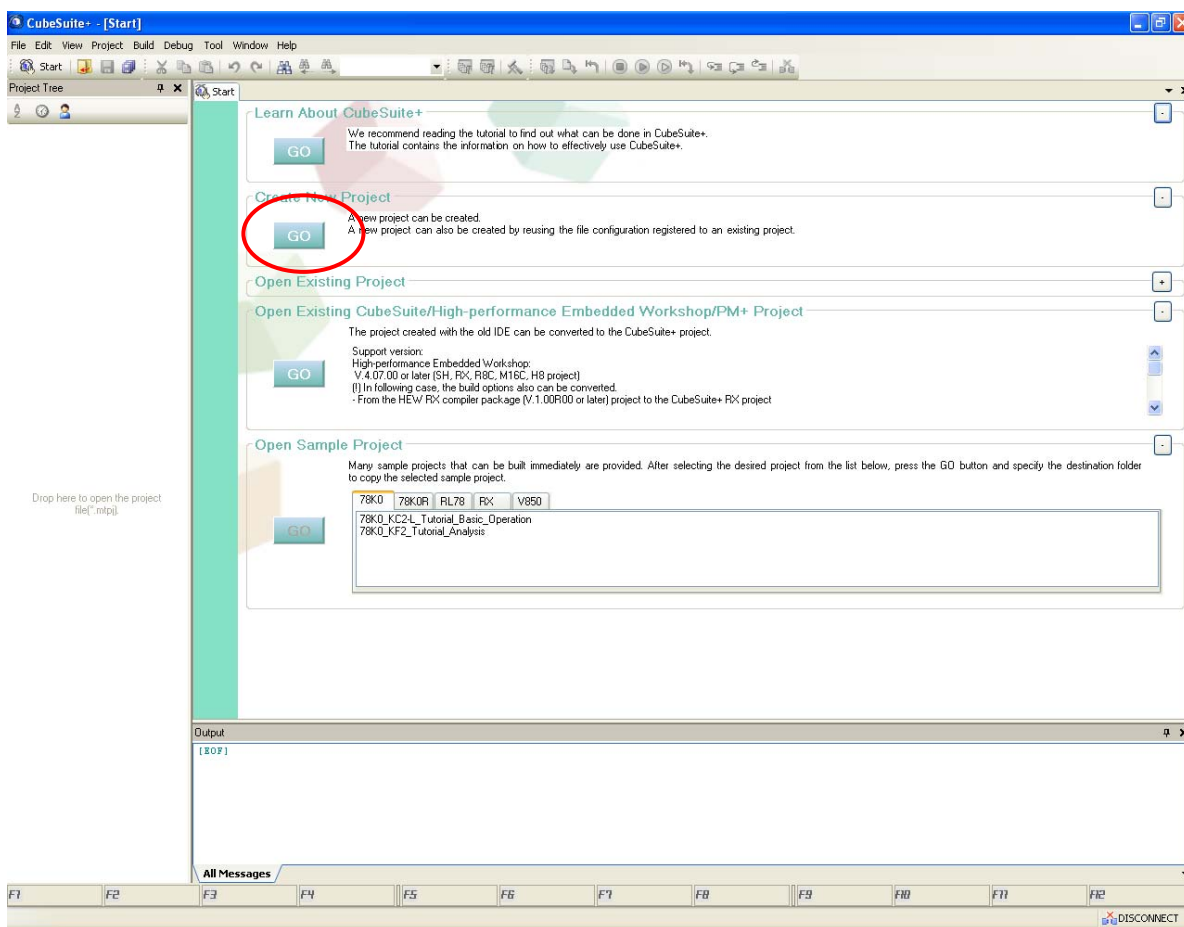

Figure 3-1   CubeSuite+ Start Panel

3.1.3 Specifying a Project

After the "Create Project" dialog, shown in figure 3-2, is displayed, configure the following settings.

(1)   Select the name of the device product to be used in the project. [Note]

(2)   Specify the project name and the location at which the project file is to be created.
(If you do not want to create a folder with the project name at the specified location, uncheck [Make the project folder].)

(3)   Check [Pass the file composition of an existing project to the new project], and in [Project to be passed:] specify the name of the project file of the sample code to be reused. By specifying it here, the source file to be reused is automatically read.

    To use the same file structure of the specified source project, check [Copy composition files in the diverted project folder to a new project folder].

(4)   Click the [Create] button to create a project.

Note: If the product type to be used does not exist in the selection list, install the device dependency information file.
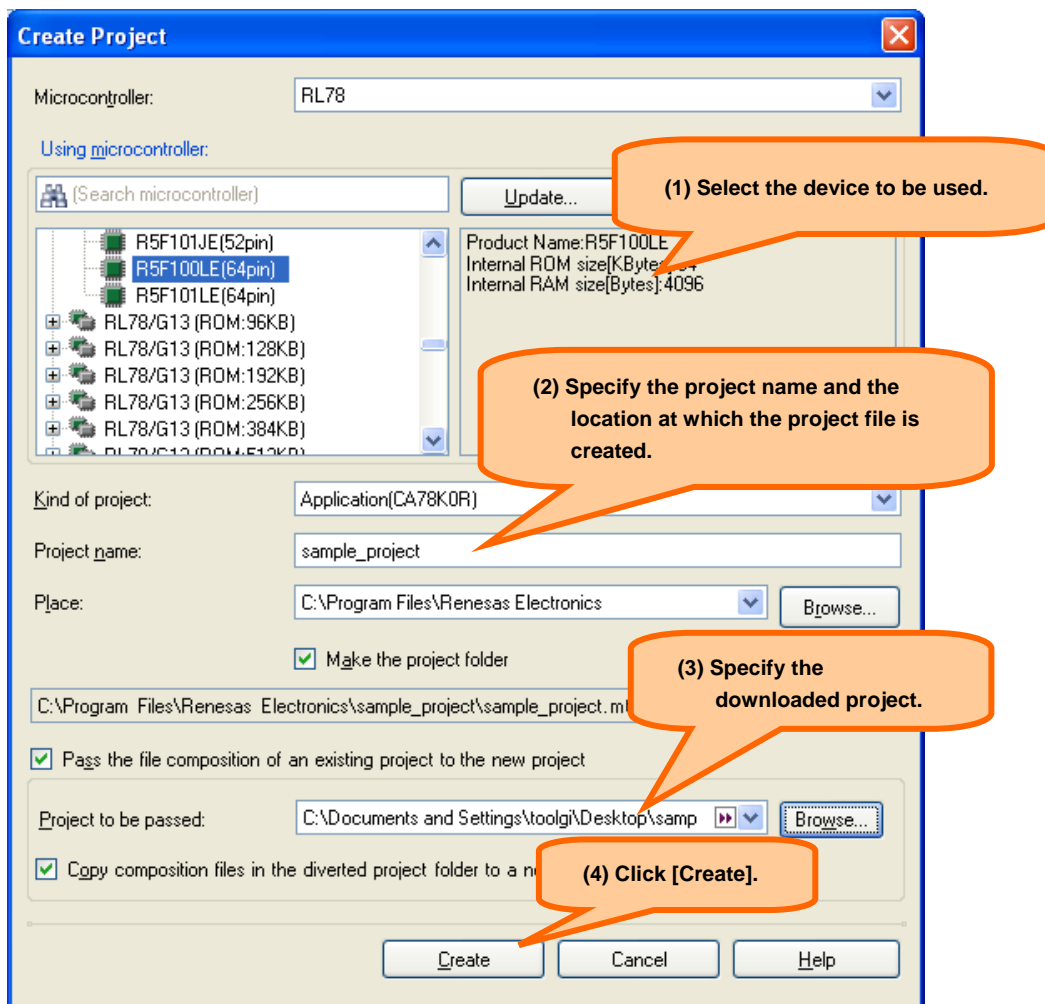

Figure 3-2   "Create Project" Dialog

3.2   Setting Option Bytes

The option bytes consist of a user option byte and an on-chip debug option byte. When the power is turned on, or at reset, the microcontroller automatically refers to the settings of the option bytes and starts operation.

This section shows settings of RL78/G13 as an example to describe the procedure for setting the option bytes in the [CA78K0R (build tool)] property of CubeSuite+.

3.2.1   Setting the On-chip Debug Option Byte

Set the on-chip debug option byte to enable the on-chip debug function of the microcontroller.

(1)   After the [CA78K0R Property] screen is displayed, select the [Link Options] tab on the bottom of the screen.

(2)   Open the [Device] category and select [Yes (-go)] in the [Use on-chip debug] property. The [Option byte values for OCD], [Debug monitor area start address], and [Debug monitor area size[byte]] properties are displayed.

[Option byte values for OCD] Property:

Specifies the control value of the on-chip debug option byte as a hexadecimal number without "0x".

(For RL78/G13, specify whether to disable on-chip debug operation (0x04) or enable on-chip debug operation (0x84/0x85).)

[Debug monitor area start address] Property:

Specifies the start address of the debug monitor area as a hexadecimal number without "0x".

(Default: the internal ROM end address - 1024 + 1)

[Debug monitor area size[byte]] Property:

Specifies the size of the debug monitor area as a decimal number. The range of specifiable values is 0 to 1024. (Default: 512)

3.2.2 Setting the User Option Byte

Set the user option byte to configure the watchdog timer (WDT) and the low voltage detector (LVD) and to specify the system reserved area.

The setting of the user option byte can also be specified in the [Device] category of the [Link Options] tab. Selecting [Yes (-go)] in the [Set user option byte] property displays the [User option byte value] property.

[User Option Byte Value] Property:

Specifies the user option byte value as a hexadecimal number without "0x". (For RL78/G13, specify whether to stop the WDT (0xEF) or use the LVD reset mode 2.81 V (0x7F) or set the operation clock to f_{IH} = 32 MHz (HS mode) (0xE8)).

Remark: For details on how to set the option bytes, refer to CubeSuite+ V1.01.00 Integrated Development Environment User's Manual: RL78, 78K0R Build (R20UT0730).

Note: The option bytes can be set by programming in source code as well as by configuring the link options. Note that if the option bytes are set simultaneously in both the ways, the value set by configuring the link options becomes valid.

3.3   Generating Object Module Files (Build)

Build the sample code using CubeSuite+ to generate object module files (HEX and LMF files) from the sample code.

There are the following types of build.

Type	Description
Build	Of the build target files, only the updated files are built.
Rebuild	All build target files are built.
Rapid build	A build is performed while build settings are changed.
Batch build	A build is performed in all build modes available for the project at a time.

Remark: For details on build methods, refer to CubeSuite+ V1.01.00 Integrated Development Environment User's Manual: RL78, 78K0R Build (R20UT0730).

This section describes "Build" as an example to show the procedure for creating a HEX file.

3.3.1 Specifying a HEX File

After opening the project, follow the procedure below to specify the HEX file name.

(1)   Double-click [CA78K0R (build tool)] (shown with a red line in figure 3-3) in "Project Tree" shown on the left to display the [CA78K0R Property] screen.

(2)   After the [CA78K0R Property] screen is displayed, select the [Object Convert Options] tab on the bottom of the screen.

(3)   Enter your desired file name with the file extension ".hex" in the [HEX file name] property of the [HEX File] category. (By default, the same name as the project file is shown.)
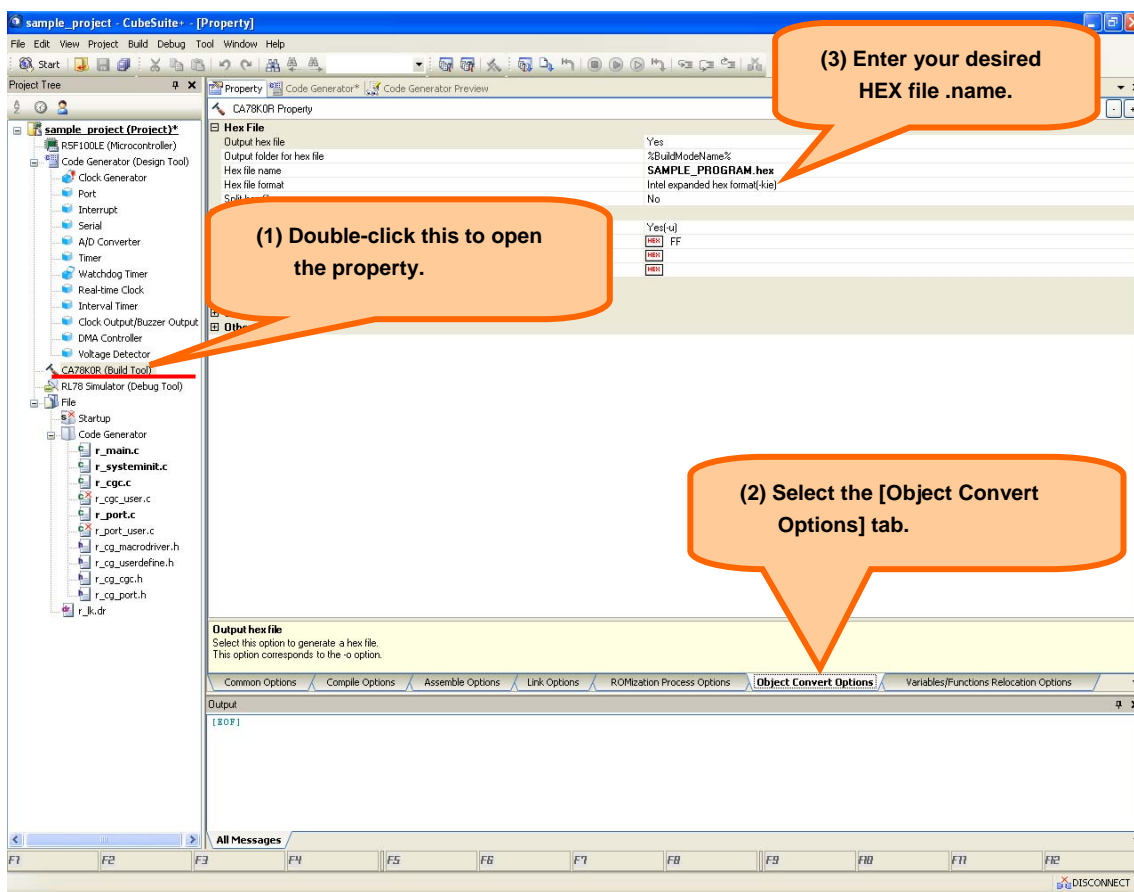

Figure 3-3   Object Convert Options Screen

3.3.2 Performing a Build

Click the [button] button, shown in a frame in figure 3-4, to perform a build. After the build is performed, a HEX file (*.hex) is generated from the source file and stored in the project file folder. Upon completion of the build, the message "Build ended (Error: 0, Warning: 0)" is displayed on the output screen of the CubeSuite+ screen.
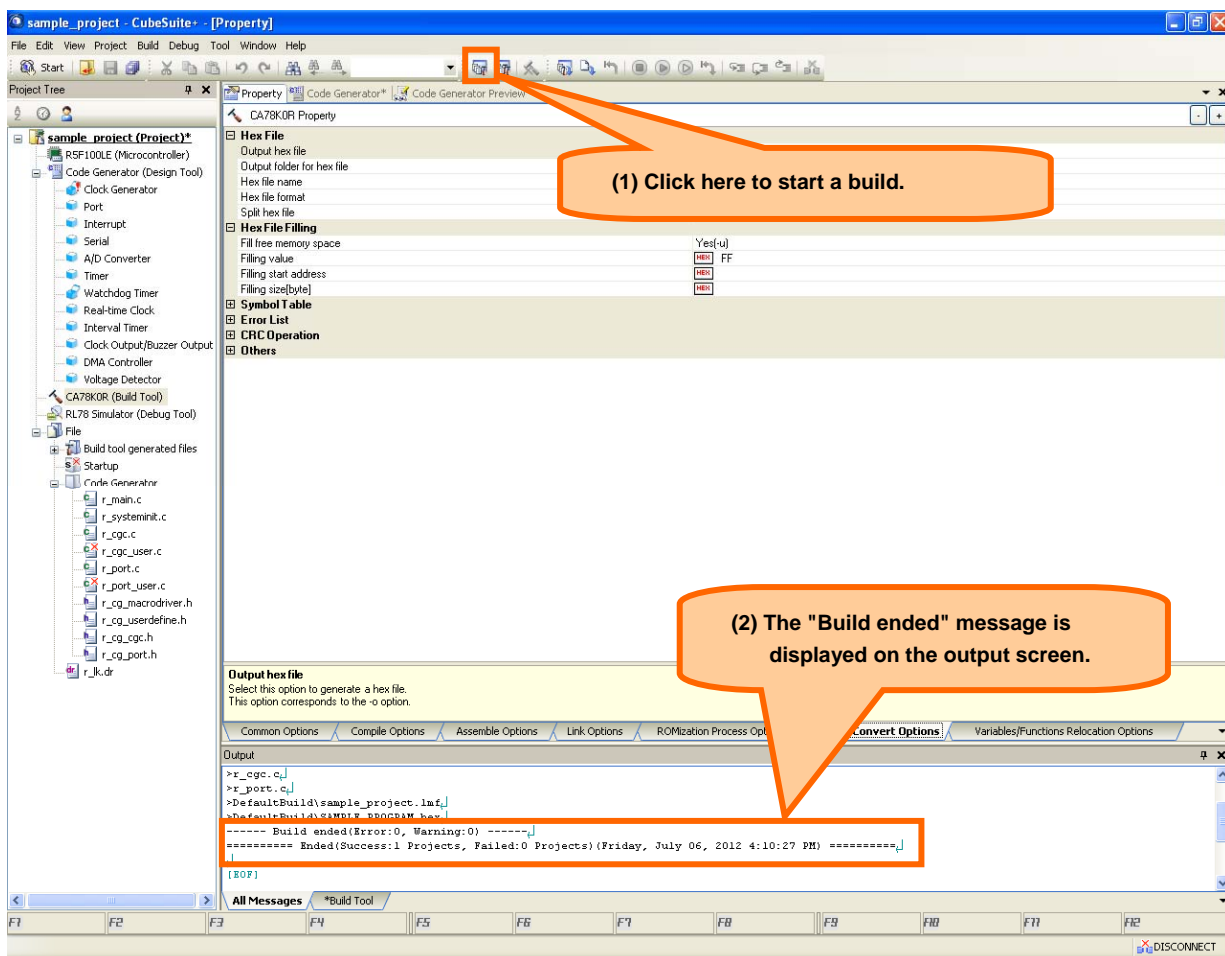

Figure 3-4   Build Execution Screen

Remark: The program in a HEX file (*.hex) can be written to the flash memory of the microcontroller by using a Renesas E1 emulator and flash programming software Renesas Flash Programmer (RFP). For details on flash programming, refer to Renesas Flash Programmer flash memory programming software User's Manual (R20UT0599).

4. On-chip Debugging with E1 Emulator

The use of debug functions provided by CubeSuite+ allows efficient debugging of programs developed for the RL78 family.

This section provides an example of configuring settings for on-chip debugging with a Renesas E1 emulator. For details on the debug functions provided by CubeSuite+, refer to CubeSuite+ V1.02.00 Integrated Development Environment User's Manual: RL78 Debug (R20UT0978).

4.1 Preparation for Debug

Connect the host machine, the E1 emulator, and the target board, if necessary, as shown in figure 4-1. For details on how to connect them, refer to E1/E20 Emulator Additional Document for User's Manual (Notes on Connecting RL78) (R20UT1994).
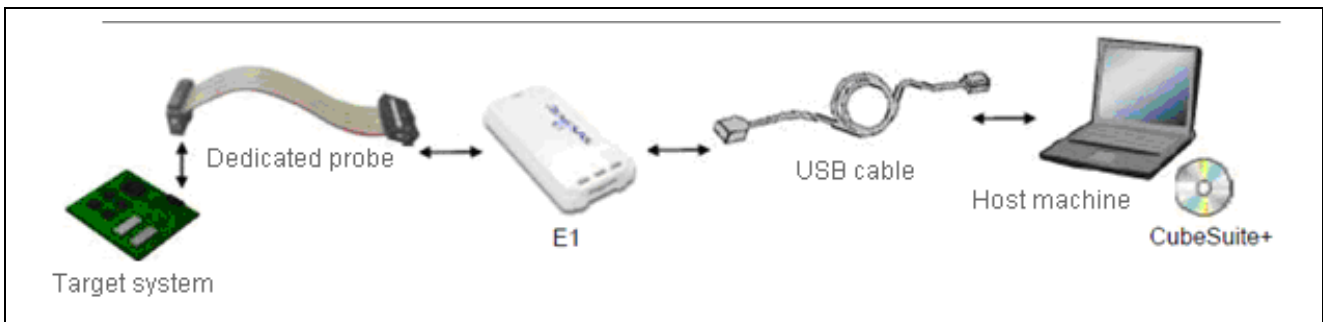
Note: Only serial communication is supported as the method of communication with the target system. (JTAG communication cannot be used.)

Figure 4-1   Example of Connecting the Host Machine and Debug Tools [E1]

4.2   Procedure for Configuring On-chip Debug Settings

4.2.1 Selecting/Changing a Debug Tool

Select the debug tool to be used in the project. To select or change the debug tool, use the context menu shown by right-clicking the [RL78 debug tool name (Debug Tool)] node in the project tree panel. (Refer to figure 4-2.)
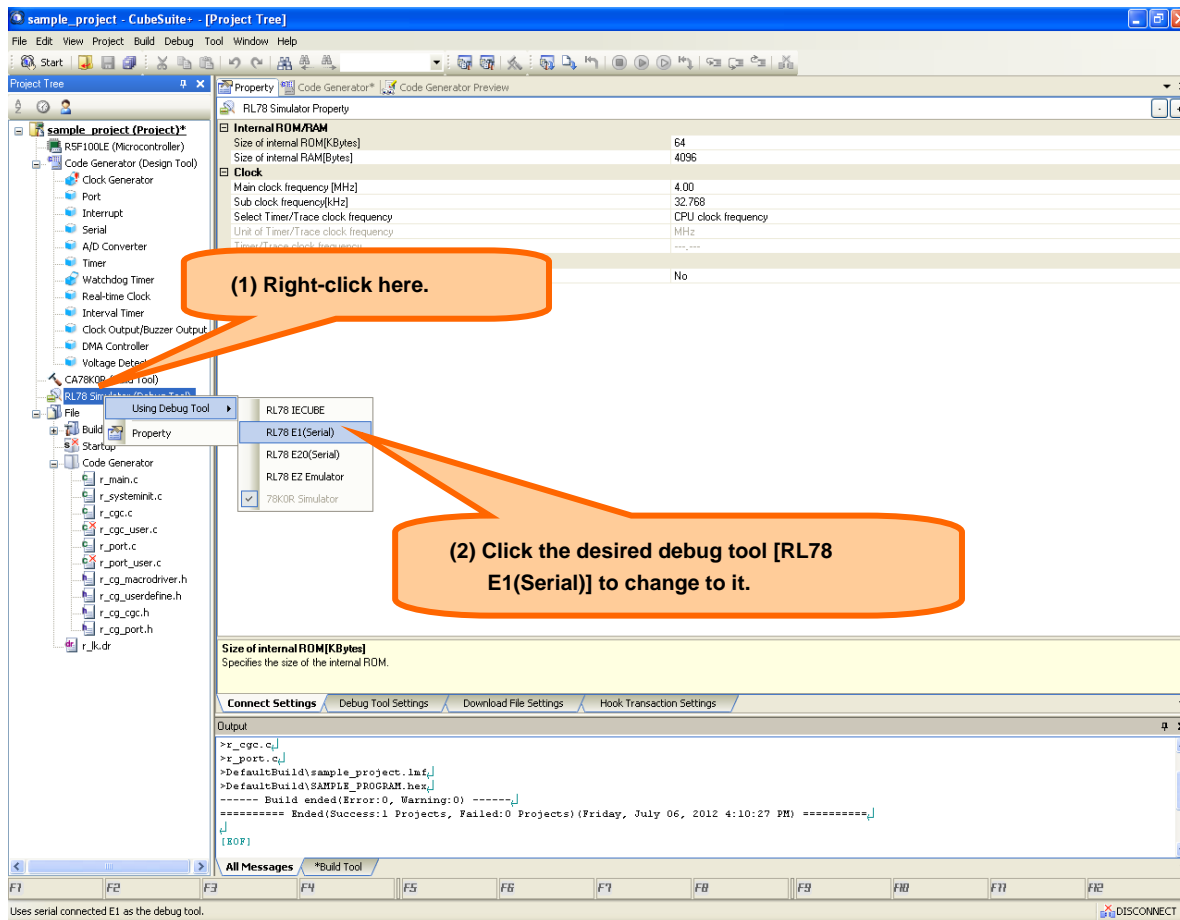

Figure 4-2   Selecting/Changing a Debug Tool [E1]

4.2.2 Setting up the On-chip Debug Environment

Use the property panel, shown in figure 4-3, to set up the operation environment for the E1 emulator. This section shows a process for configuring the settings in "Connection with Target Board". For configuring other settings, refer to CubeSuite+ V1.02.00 Integrated Development Environment User's Manual: RL78 Debug (R20UT0978).
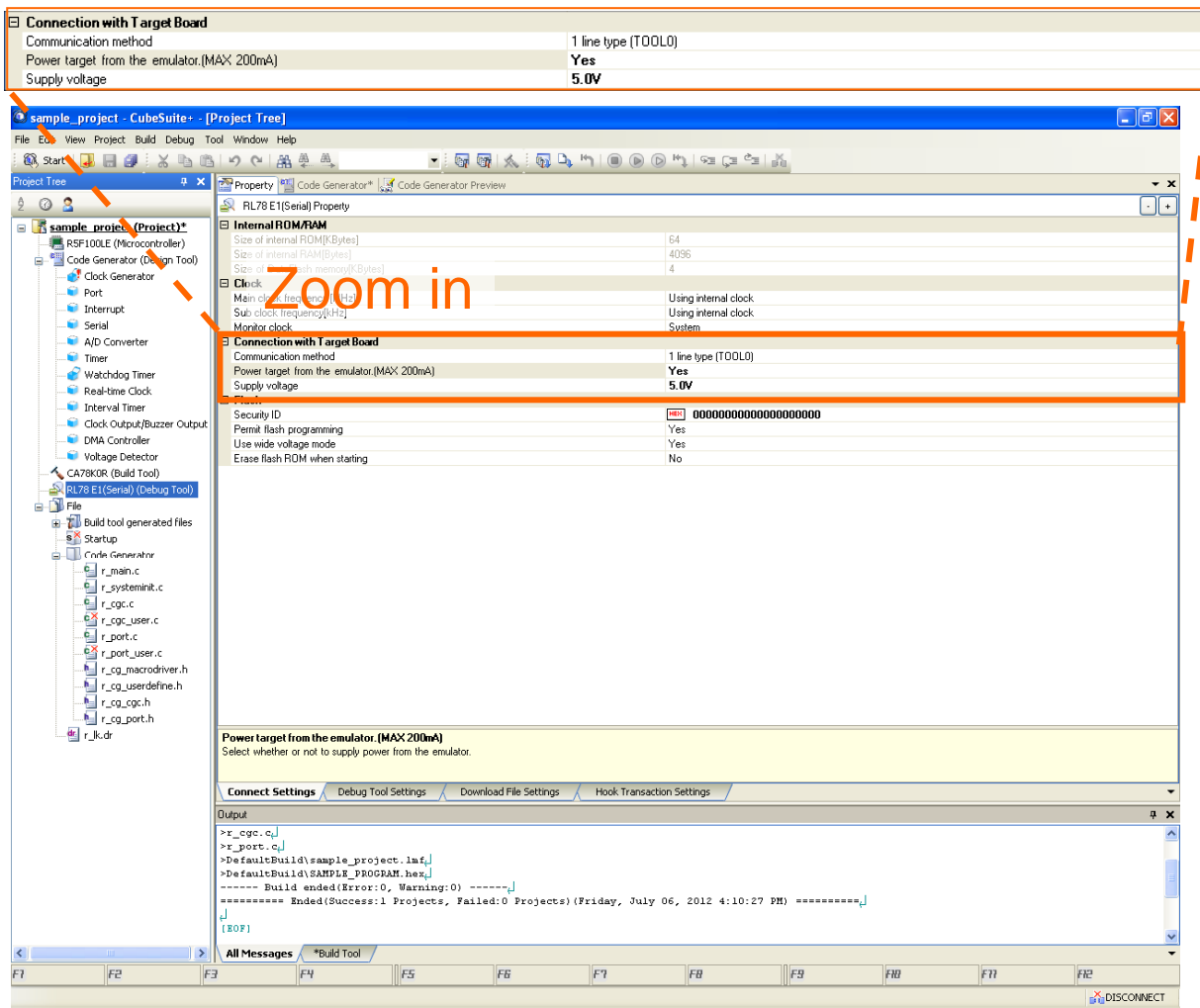
**Figure 4-3   Operation Environment Setting [E1] (Property Panel)**

a. Communication method
Specifies the communication method by which the E1 emulator establishes a serial communication with the RL78/G13 in the target system. Select 1 line type (TOOL0).

b. Power target from the emulator (MAX 200 mA)
Specifies whether to supply power to the target system from the E1 emulator. To supply power to it, select [Yes] ([No] is selected by default). To choose not to supply power from the emulator, select [No] and supply power externally.

c. Supply Voltage
This property is displayed only if [Yes] is selected in the [Power target from the emulator (MAX 200 mA)] property. It specifies the voltage supplied to the target system.

Example: 5.0 V … A voltage of 5.0 V is supplied to the target system.

Note: The properties in this category cannot be changed while the E1 emulator is connected.

4.2.3 Connecting the Debug Tool to CubeSuite+

Click the [icon] button on the debug toolbar to download the LMF file after connecting the debug tool.

(1) Click here to download the program to the debug tool.
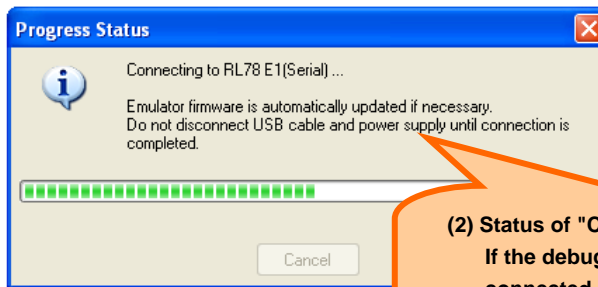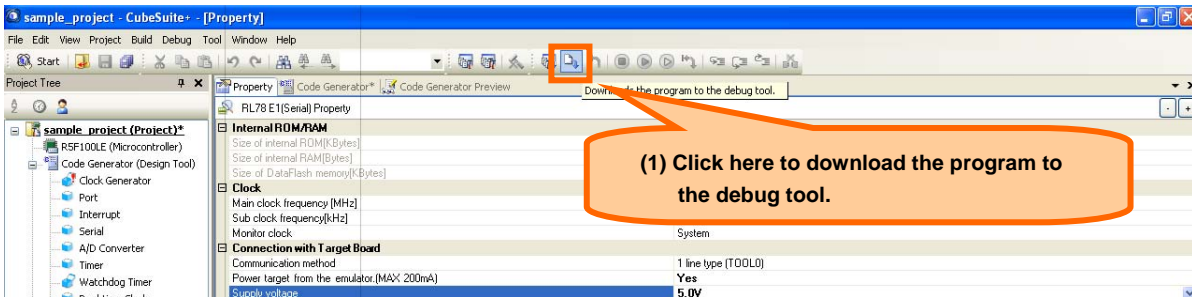
(2) Status of "Connecting to RL78 E1".
If the debug tool is successfully connected, the status bar in the main window changes as shown below.
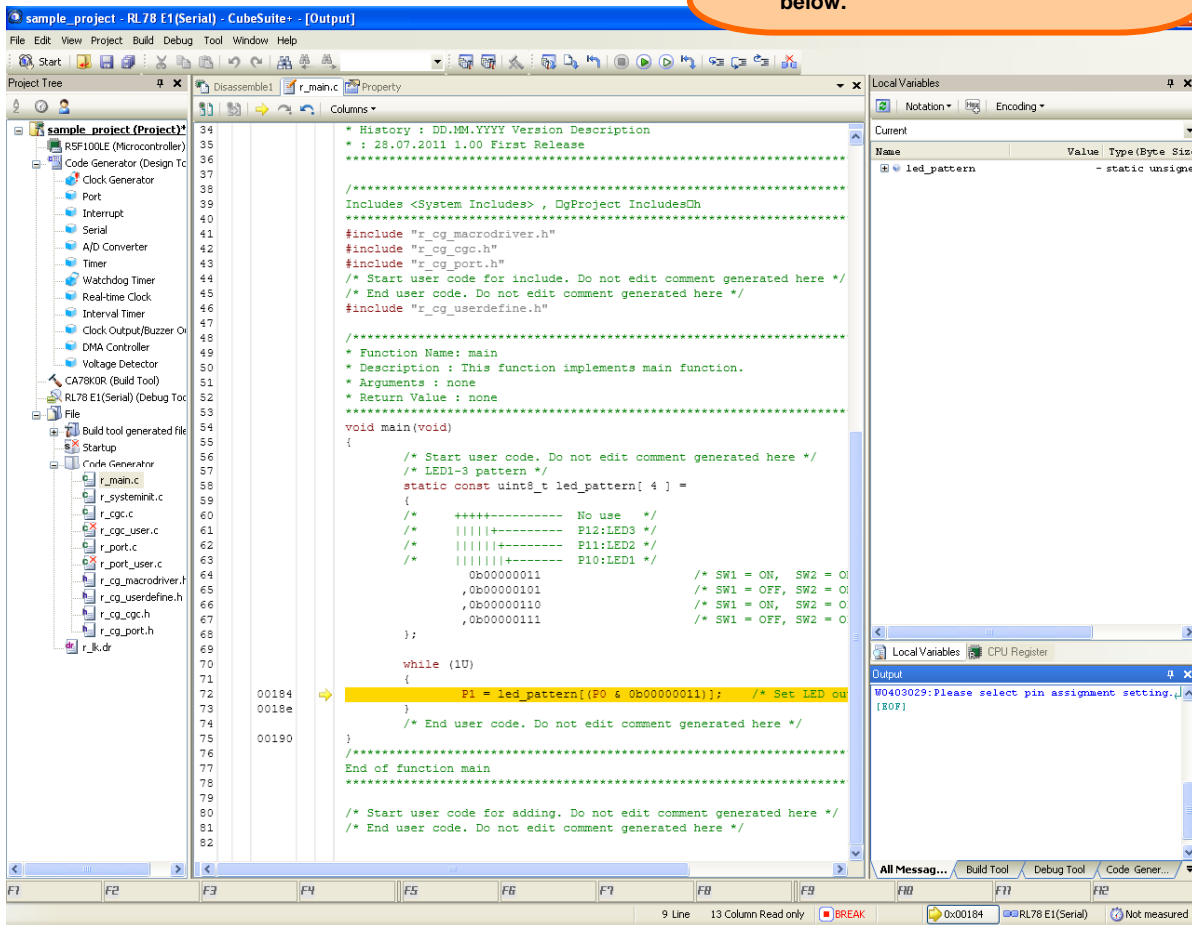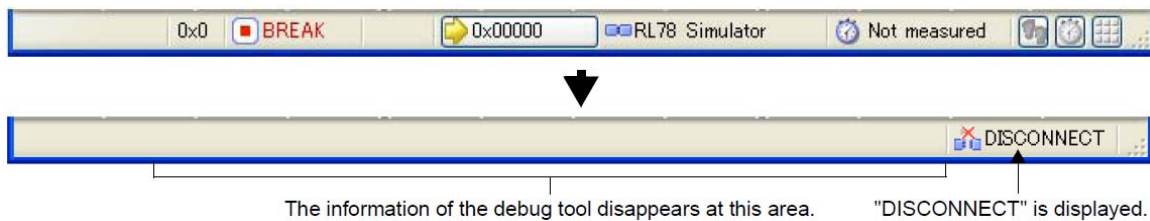
Figure 4-4   Example of Downloading the Program to the Debug Tool

4.2.4 Disconnecting the Debug Tool from CubeSuite+

Click the [ ] button in the debug toolbar to terminate the communication with the currently connected debug tool. After the communication is terminated, the status bar on the main window changes as shown in figure 4-5.

Figure 4-5   Status Bar Indicating Disconnection with the Debug Tool

5. Appendix: Code Generation

5.1 How to Perform Code Generation

Code generation with CubeSuite+ outputs source code (device driver program) according to settings that are configured through the CubeSuite+ panel and are necessary to control peripheral functions (e.g., functions of clock generators and ports) provided by the RL78 family.

This section describes the procedure for code generation. For details on code generation, refer to CubeSuite+ V1.02.00 Integrated Development Environment User's Manual: RL78 Design (R20UT0976).

5.1.1 Configuration on the Code Generation Panel

Open the code generation panel for the peripheral functions to be used (e.g., functions of clock generators and ports), and configure necessary settings.
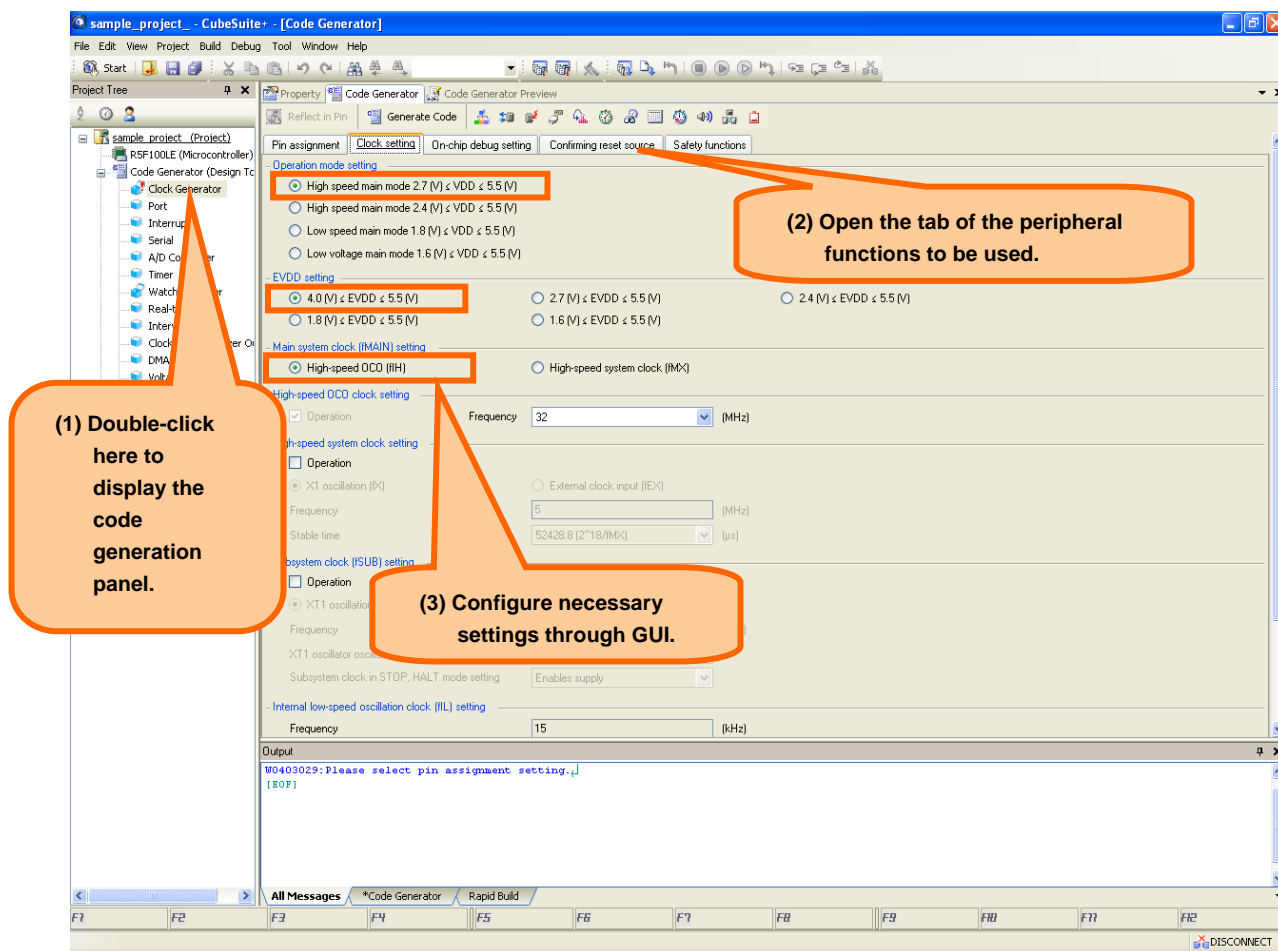
Figure 5-1   Example of Configuration on the Code Generation Panel

Remark: If, on the code generation panel, settings are wrongly configured or necessary settings are not configured, or if pins to be configured have already been used for other functions, a ⬤ icon is displayed indicating that the relevant information is incorrect. When the mouse cursor is moved onto the ⬤ icon, information on this warning (hints about solution) is shown as a pop-up message.

5.1.2 Checking the Source Code

Selecting [Code Generation Preview] allows you to, on the opened code generation preview panel, check the source code to be generated according to settings configured on the code generation panel.

5.1.3 Outputting the Source Code

Click the [Generate Code] button on the code generation panel to output the source code.
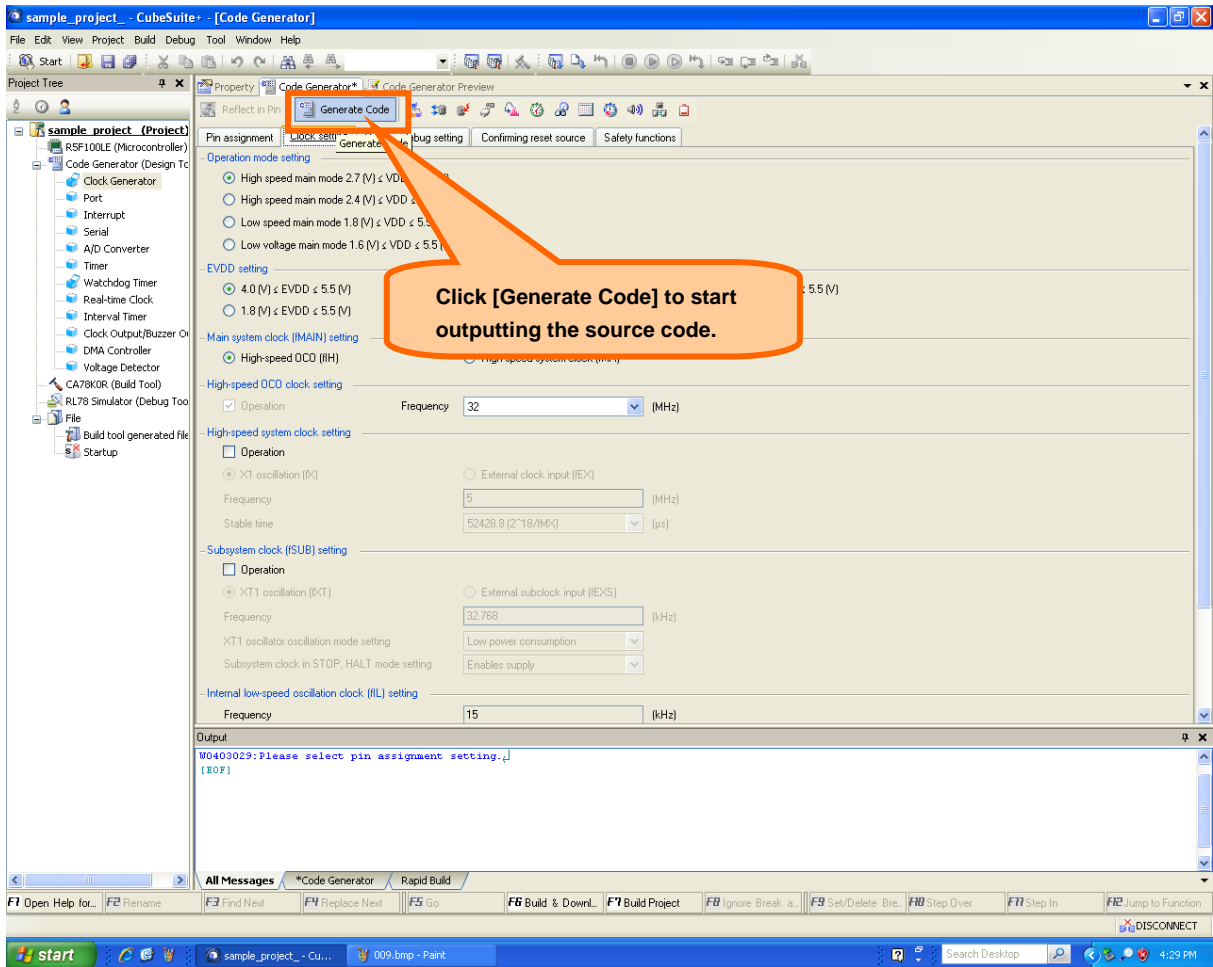

Figure 5-2   Example of Outputting Source Code

5.2   Rules on Source Code Output through Code Generation

5.2.1 Structure of Source Code

Listed below are the files output through code generation.

Table 5.1   List of the Files Output through Code Generation

Output Unit	File Name	Description
Each peripheral function	r_cg_*peripheral function name*.c	Initialization functions, API functions. [Note]
	r_cg_*peripheral function name*_user.c	Interrupt functions, callback functions.
	r_cg_*peripheral function name*.h	Defines macros for the values assigned to registers.
Project	r_main.c	main function
	r_systeminit.c	Calls the initialization function of each peripheral function. Calls R_CGC_Get_ResetSource.
	r_cg_macrodriver.h	Defines the macros used in all source files.
	r_cg_userdefine.h	Empty files (For user definition.)
	r_lk.dr	Link directive

Note: For details on API function, refer to CubeSuite+ V1.02.00 Integrated Development Environment User's Manual: RL78 Design (R20UT0976).

5.2.2 Rules on Source Code

This section shows the source code in RL78/G13 A/D Converter (Software Trigger and Sequential Conversion Modes) (R01AN0452) as an example to describe naming rules.

● main.c

```
<List omitted>
/***************************************************************************
Exported global variables and functions (to be accessed by other files)
***************************************************************************/
/* Start user code for global. Do not edit comment generated here */

uint16_t        result_buffer;              /* AD converter result buffer */          You can define global
                                                                                        variables between these.
/* End user code. Do not edit comment generated here */
/***************************************************************************
* Function Name: main
* Description : This function implements main function.
* Arguments : none
* Return Value : none
***************************************************************************/
void main(void)
{                                                                                       You can write a user program
/* Start user code. Do not edit comment generated here */                               between these.
uint8_t count;                               /* Loop counter */

result_buffer = 0U;                          /* Initialize result buffer */

DI();                                        /* Disable interrupt */
R_ADC_Set_OperationOn();                     /* Enable comparator operation */

/* ---- stabilization wait time(about 1us) ---- */
for (count=0U; count<3U; count++)
{                                            API function [refer to r_adc.c.]
NOP();
}
R_ADC_Start();                               /* Start AD converter */

<List omitted>
}
/* End user code. Do not edit comment generated here */
}
```

Remark: Do not change character strings in these comments.

RENESAS

- r_adc.c

```
<List omitted>
void R_ADC_Create(void)                    Binary constant
{
        ADCEN = 1U;        /* supply AD clock */           The value to be set to the register (the
                                                           ADM0 register) is the logical sum of the
<List omitted>                                             underlined bytes.
        /* Set ANI0 pin as analog input */
        PM2 |= 0x01U;
        ADM0 = _28_AD_CONVERSION_CLOCK_5 | _00_AD_TIME_MODE_NORMAL_1 | _00_AD_OPERMODE_SELECT;
        ADM1 = _00_AD_TRIGGER_SOFTWARE | _00_AD_CONVMODE_CONSELECT;
<List omitted>
/************************************************************************
* Function Name: R_ADC_Start
* Description : This function starts the AD converter.
* Arguments : none
* Return Value : none
************************************************************************/
void R_ADC_Start(void)
{
        ADIF = 0U;                          /* clear INTAD interrupt flag */
        ADMK = 0U;                          /* enable INTAD interrupt */
        ADCS = 1U;                          /* enable AD conversion */
}
/************************************************************************
End of function R_ADC_Start
************************************************************************/
```

Remark: Do not change character strings in these comments.

a. Defining Global Variables

Global variables must be defined after "/* Start user code for global. Do not edit comment generated here */". Note that the lines before "/* End user code. Do not edit comment generated here */" are not changed by code generation.

> Note: If a global variable is defined in lines other than these, it is overwritten by CubeSuite+ during code generation.

b. Writing a User Program

A user program must be written after "/* Start user code. Do not edit comment generated here */". Note that the lines before "/* End user code. Do not edit comment generated here */" are not changed by code generation.

> Note: If a user program is written in lines other than these, it is overwritten by CubeSuite+ during code generation.

c. API Functions

API functions must be called by the user program. Its calling them configures registers for each peripheral function and performs other operations, according to the generated code.

For details on API functions, refer to CubeSuite+ V1.02.00 Integrated Development Environment User's Manual: RL78 Design (R20UT0976).

d. Binary Constants
    To directly write data to bits or registers, it must be written as a binary constant in the source code for code
    generation.

Table 5.2   List of Binary Constant Types

Binary number with no additional code	int, unsigned int, long int, unsigned long int
With an additional code "u" or "U"	unsigned int, unsigned long int
With an additional code "l" or "L"	long int, unsigned long int
With additional codes "u" or "U" and "l" or "L"	unsigned long int

e. Register Setting
    In the generated code, registers are assigned the logical sum of variables automatically defined in
    "r_cg_*peripheral function name*.h".

Website and Support

Renesas Electronics Website
- http://www.renesas.com/index.jsp

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

Revision Record	RL78 Family CubeSuite+ Startup Guide

Rev.	Date	Description	
		Page	Summary
1.00	Aug. 24, 2012	—	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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
    Handle unused pins in accord 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 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 one with a different part number, confirm that the change will not lead to problems.
    — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, 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: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

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: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141