RL78/G24

Flexible Application Accelerator (FAA) Tool Guide: CS+

Introduction

This guide describes the options that must be set for the build process and debugger of the flexible application accelerator (FAA) contained in RL78/G24. It also describes how to operate the debugger.

Target Device

RL78/G24

RL78/G24 Fast Prototyping Board

Chapter Composition

Chapter 1: Overview of Flexible Application Accelerator (FAA)

This chapter describes the overview of the flexible application accelerator (FAA) and program creation.

Chapter 2: Overview of build process and debugger of Flexible Application Accelerator (FAA)

This chapter describes the new project creation procedure and the options that must be set for the build process and debugger of the flexible application accelerator (FAA). It also describes how to operate the debugger.

Chapter 3: Debugger operation using sample project

This chapter describes debugging operations for FAA programs using the sample code and the sample script.

Related Documents

- RL78/G24 User's Manual: Hardware (R01UH0961)
- RL78/G24 Fast Prototyping Board User's Manual (R20UT5091)

Contents

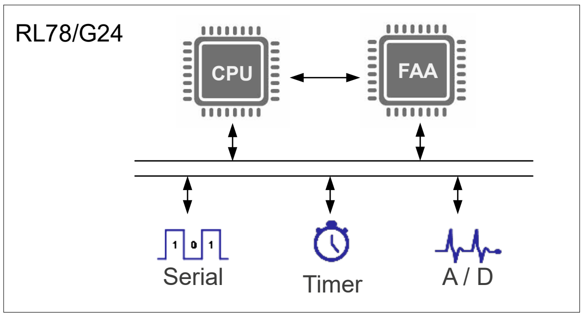
1. Overview

1.1 Flexible Application Accelerator (FAA)

The flexible application accelerator (FAA) contained in RL78/G24 is a Renesas original application accelerator with a Harvard architecture. It can execute 32-bit multiplication, addition, and subtraction in a single cycle.

FAA can access some peripheral functions directly by the address bus select function. Operations by the CPU and FAA can be combined to suit the application, it can improve operation efficiency of the system.

Figure 1-1 Image diagram of RL78/G24 FAA



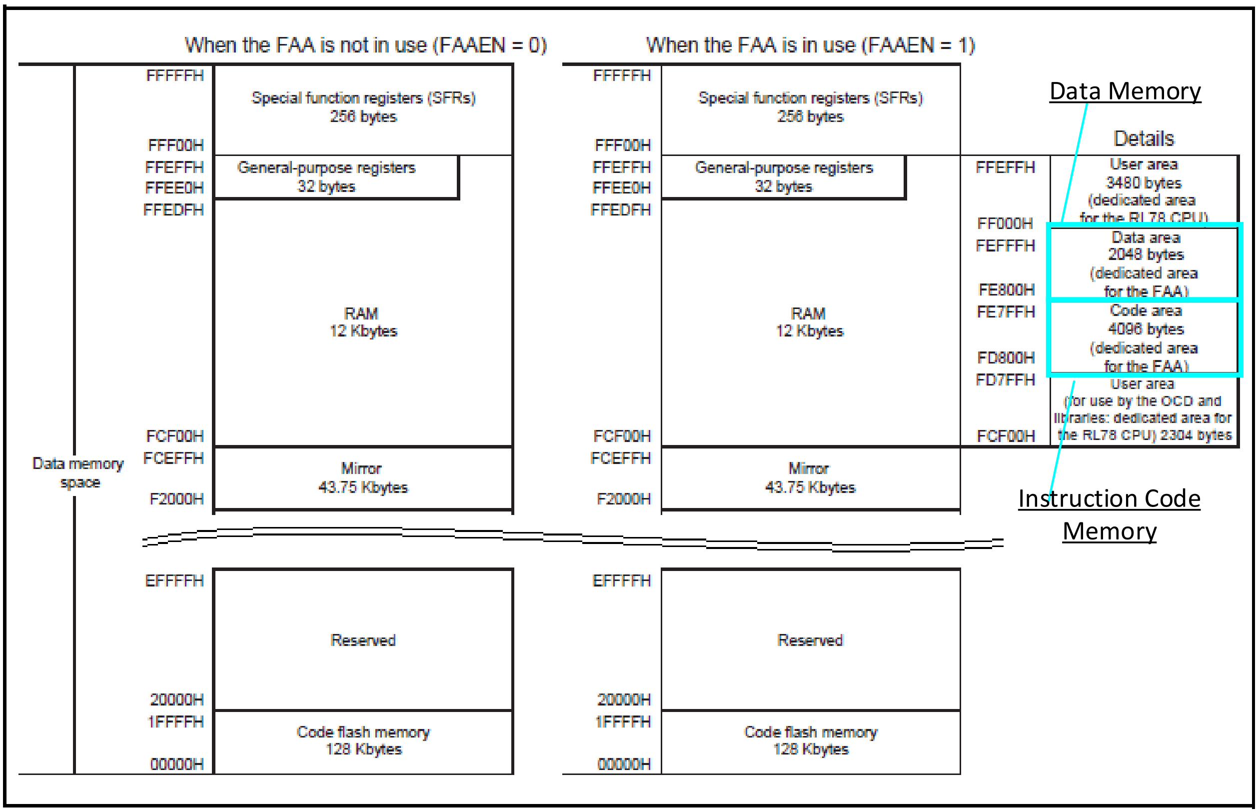
1.2 Internal Memory Space of FAA

When the FAA is in use, some of the RL78/G24's internal RAM is dedicated to the FAA.

Instruction Code Memory: Store the program for FAA

Data Memory: Store the data for FAA

Figure 1-2 Memory Map of the Instruction Code Memory and Data Memory



1.3 Program for RL78/G24

1.3.1 Program Structure

Programs for the CPU and programs for the FAA are coded in separate files. FAA programs use the FAA-dedicated instruction sets. CPU programs and FAA programs are built together in an object file (load module file) that can be executed in RL78/G24.

Figure 1-3 Program structure when FAA is in use

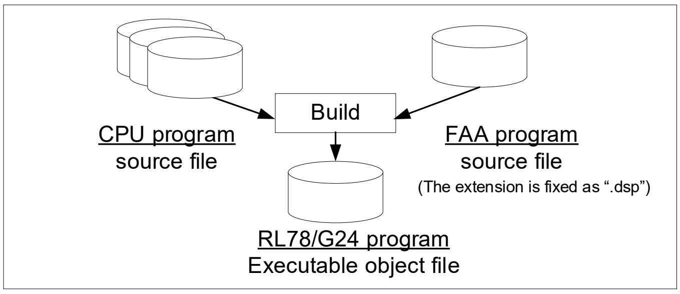
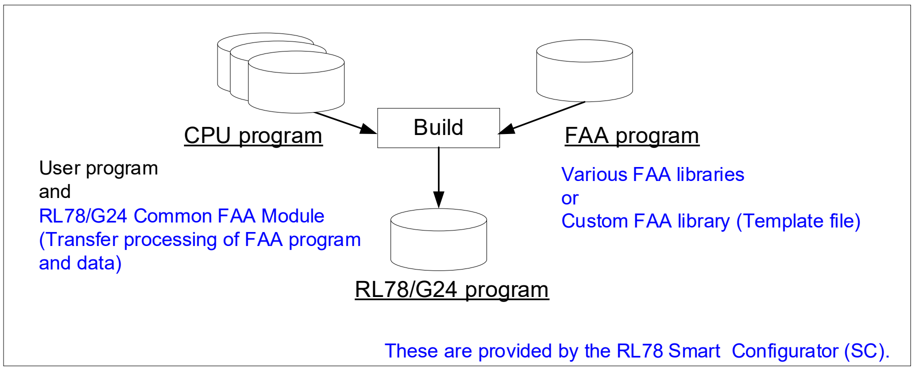


Figure 1-4 FAA related files provided by the RL78 Smart Configurator



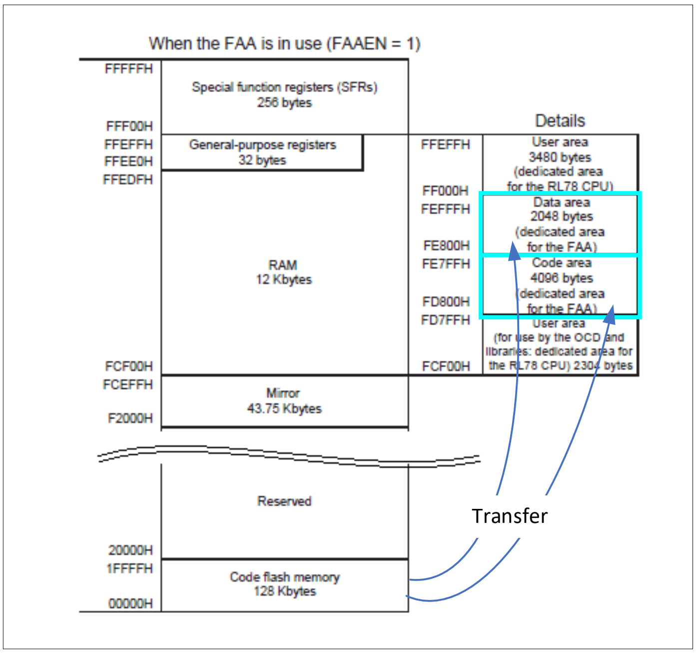
Remark. If you do not use the FAA program and data transfer processing, and template file provided by the SC, you will need to prepare your own program.

1.3.2 Transfer of Program and Data for FAA

An executable object file is written to the RL78/G24 code flash memory. However, FAA programs must be placed in the instruction code memory and FAA data must be placed in the data memory. Therefore, before executing an FAA program (before the FAA operation is enabled), the FAA program and data stored in the code flash memory must be transferred to the instruction code memory and data memory, respectively.

The transfer processing of FAA program and data is provided by the FAA component - FAA library "RL78/G24 Common FAA Module" in the Smart Configurator for RL78. For details about how to generate the transfer processing in the SC, see 2.3 Adding FAA Program.

Figure 1-5 Transfer of the program and data for FAA



1.3.3 FAA Program

You can create an FAA program by either of the following ways:

- Use a provided FAA library according to the purpose. The library is provided in a source file in which code cannot be changed. (FAA library of various function)
- Use a template file to code your own FAA program. (Template (Custom FAA library))

In both cases, add the FAA program to the program project by using the Smart Configurator (SC).

For details about how to use the Smart Configurator (SC) to output an FAA program file (library or template), see 2.3 Adding FAA Program.

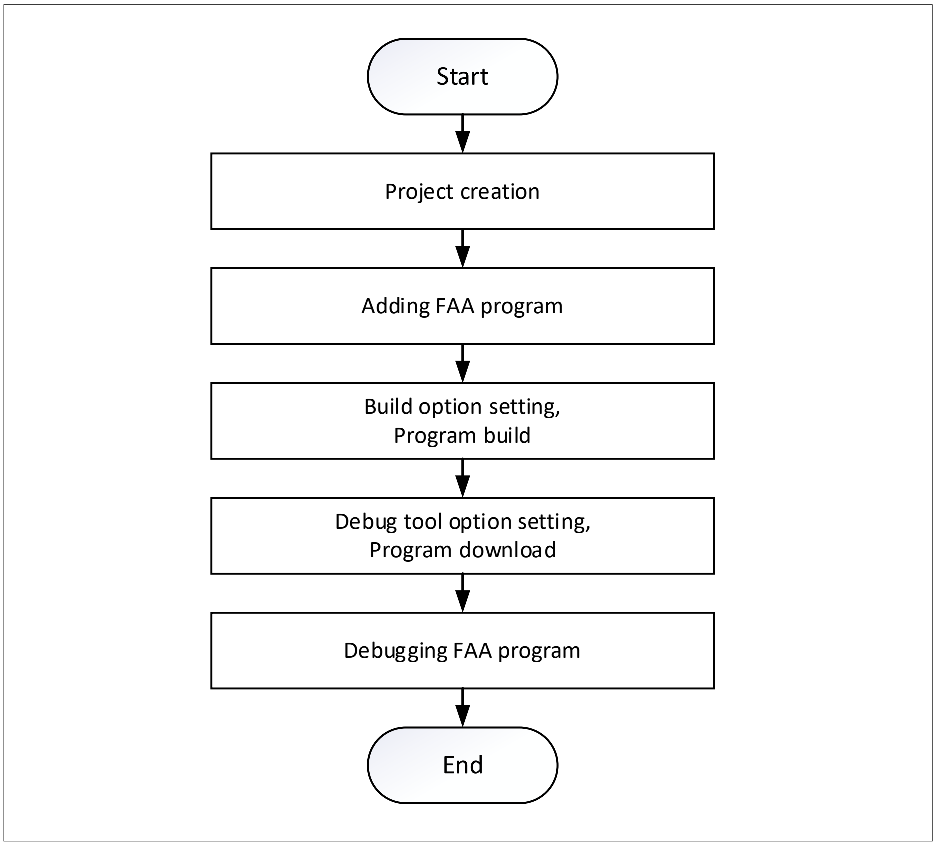
Remark. For instruction sets for FAA, refer to the chapter for FAA in RL78/G24 User's Manual: Hardware (R01UH0961).

1.3.4 Build Process and Debug of FAA Program

To build and debug FAA programs, some options must be set up. This guide describes the options that must be specified for the processing shown in Figure 1-6. It also describes how to use the debugger for debugging FAA programs.

Note that this guide requires the use of FAA programs (libraries or templates) generated by the Smart Configurator (SC).

Figure 1-6 Operating instruction in chapter 2 of this guide



2. Option Setting and Operation

This chapter explains the option settings and debugger operation required for building and debugging an FAA program in the CS+ for CC environment.

For options that are not described in this guide, set them if necessary. For details about the options and operations, see the help or documentation of CS+ for CC.

2.1 Operating Environment

This guide uses the following tools:

Table 2-1 Software tool

Integrated development environment	Item	version
CS+	CS+ for CC Manufactured by Renesas Electronics	V8.12.00
	CC-RL Manufactured by Renesas Electronics	V1.14.00
	DSPASM FAA/GREEN_DSP Structured Assembler Manufactured by Renesas Electronics	V1.05.00
	RL78 Smart Configurator Manufactured by Renesas Electronics	V1.11.0

Table 2-2 Hardware tool

Board / Emulator	Item
Board Note1	RL78/G24 Fast Prototyping Board Manufactured by Renesas Electronics
Emulator Note2	E2 emulator Lite Manufactured by Renesas Electronics
	E2 emulator Manufactured by Renesas Electronics

Note1. Jumper settings etc. differ depending on whether the board is connected to the emulator or using COM port debugging. Please see RL78/G24 Fast Prototyping Board User’s manual (R20UT5091) the instruction manual for details.

Note2. When the debugger and the RL78/G24 Fast Prototyping Board are connected via COM port, the emulator is not required.

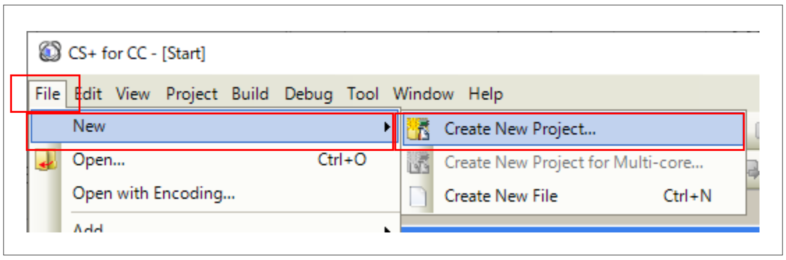
2.2 Project Creation

Select the RL78/G24 product as the microcontroller to be used and create a program project.

Procedure:

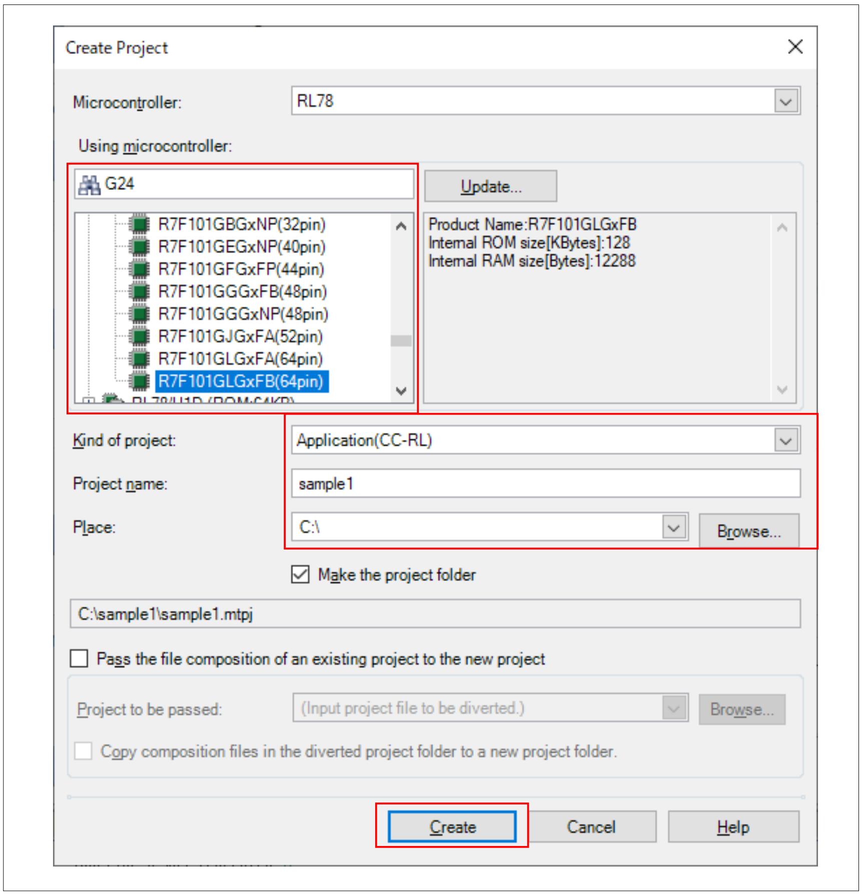
1. Launch the CS+.
2. Select [File] menu -> [New] -> [Create New Project] of CS+.

Figure 2-1 [File] menu -> [New] -> [Create New Project]



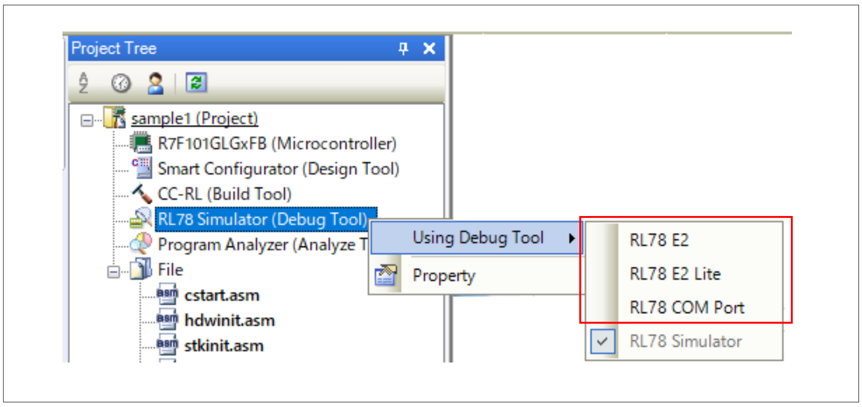
3. In the [Create Project] dialog, select the RL78/G24 products, input the project name and click the [Create].

Figure 2-2 [Create Project] dialog



4. After creating the project, change the debug tool you use. In 2.3 Adding FAA Program, the Smart Configurator (SC) sets some options for the debug tool, so you must first select the debug tool you want to use.

Figure 2-3 Select debug tool



2.3 Adding FAA Program

Use the Smart Configurator (SC) to add an FAA program (library or template) to your project.

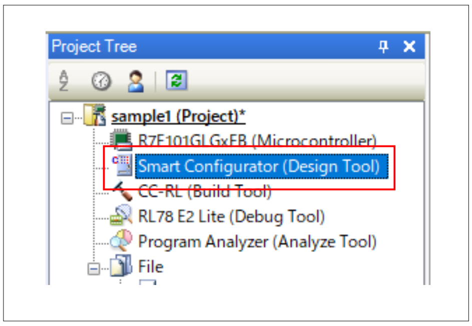
This guide only describes the procedure for adding an FAA program, [Clock], [System] and [Voltage detection] that need to be set in the CPU program. Please set other peripheral functions as appropriate to suit your system.

2.3.1 Adding FAA Component

Procedure:

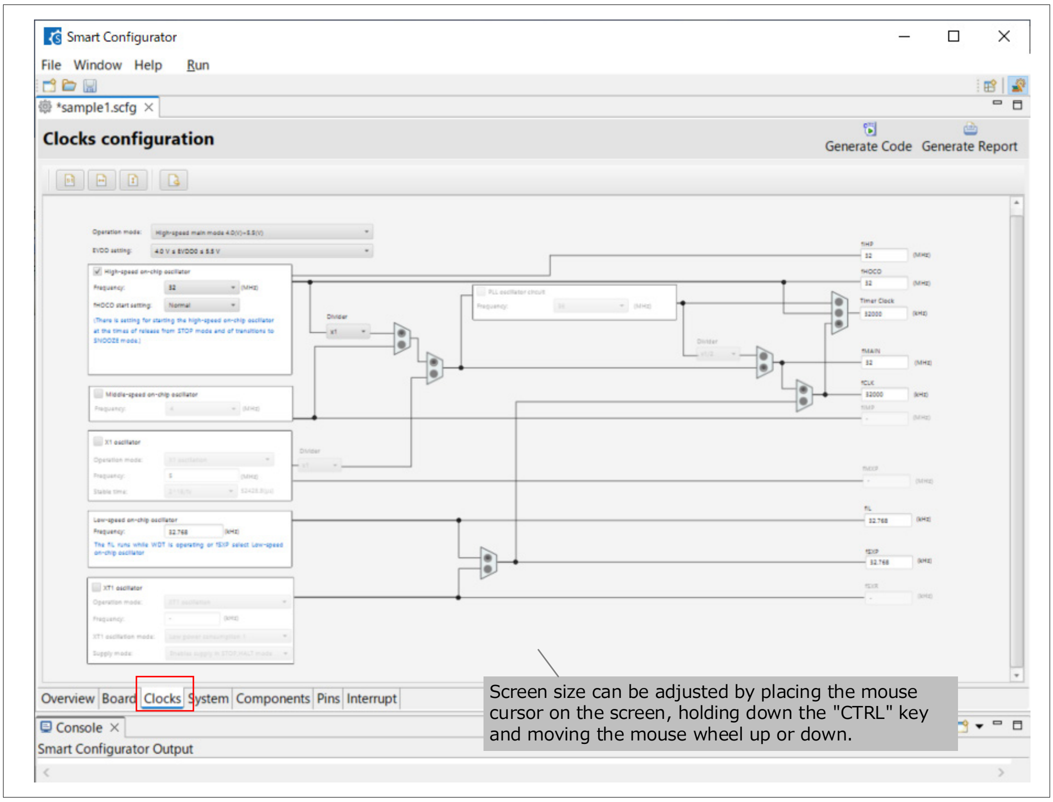
1. In the CS+ Project Tree, double-click [Smart Configurator (Design Tool)] to launch the RL78 Smart Configurator.

Figure 2-4 Launch Smart Configurator



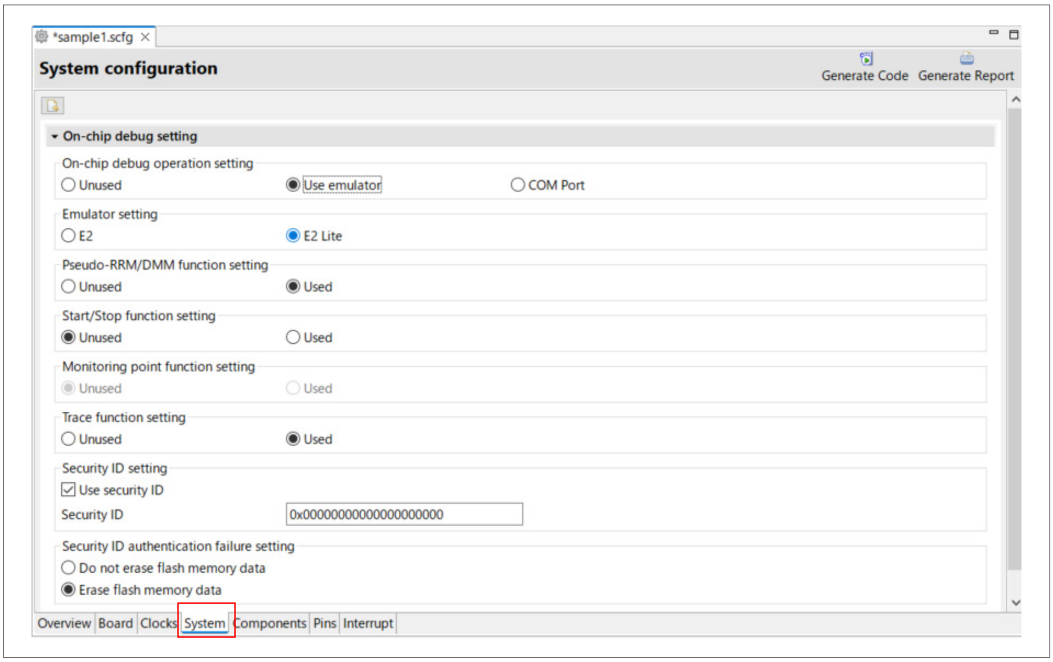
2. In the Smart Configurator (SC), click [Clock]. Set various clocks and the operation mode according to your system.

Figure 2-5 Smart Configurator: [Clock] tab



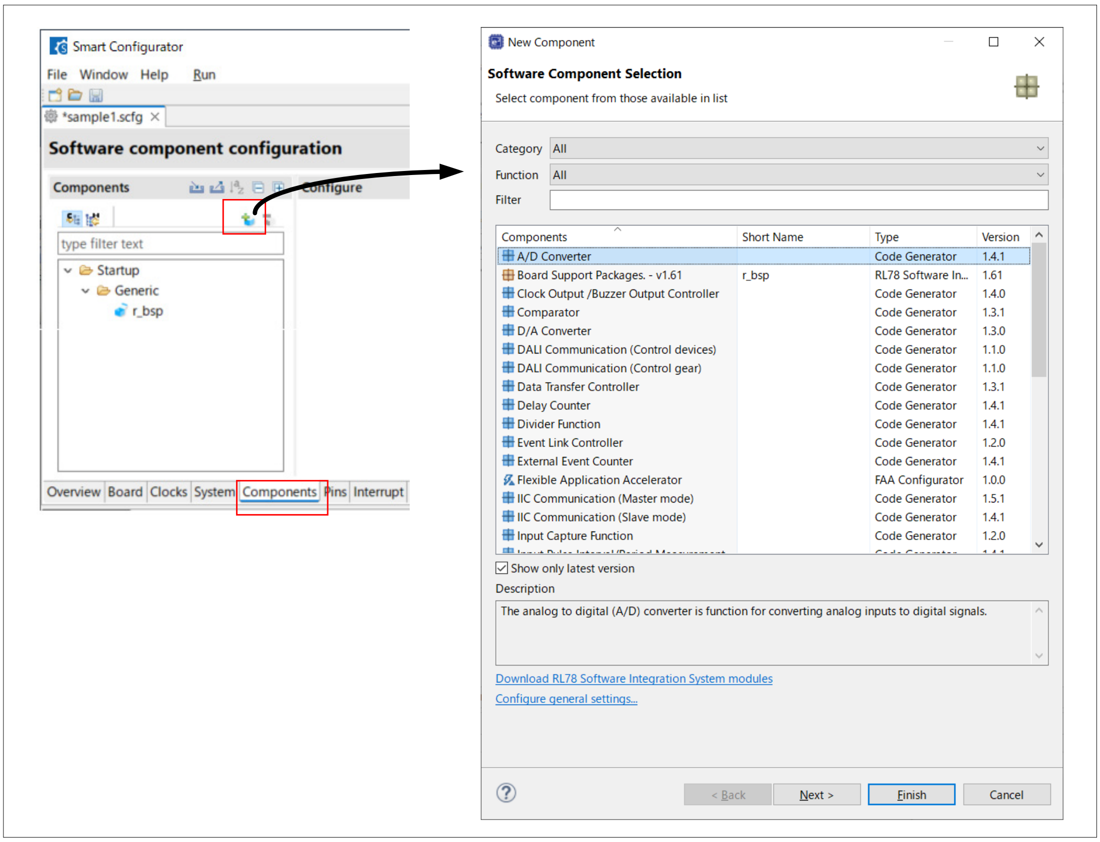
3. Click the [System]. In the [System] tab, set the debug tool and functions to be used, and security ID.

Figure 2-6 Smart Configurator: [System] tab



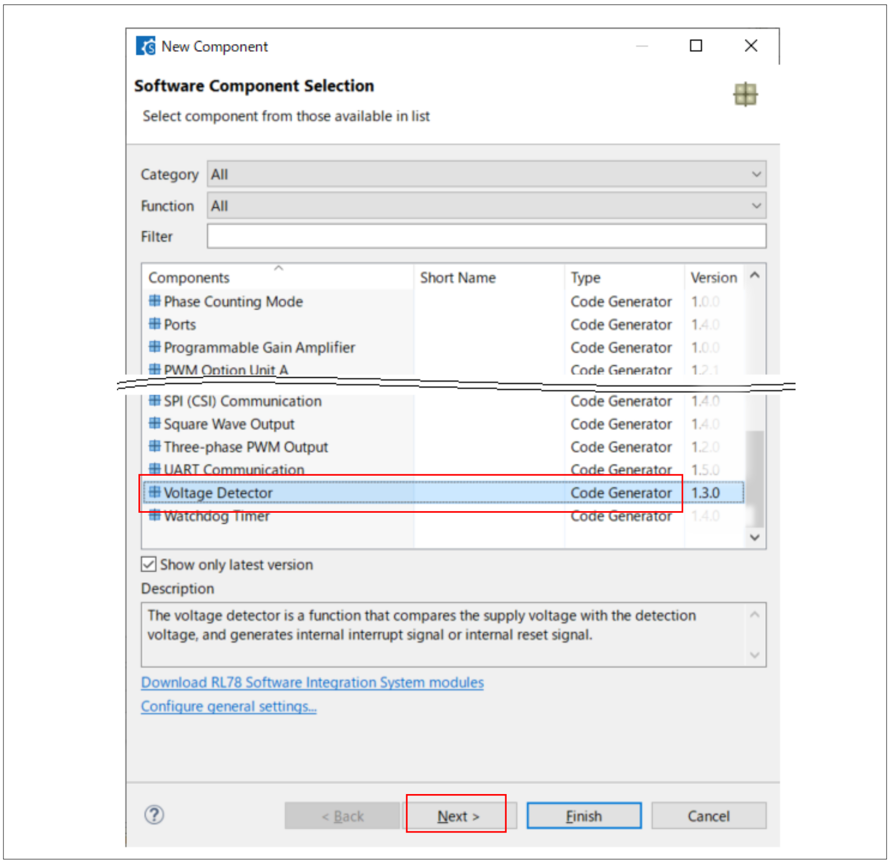
4. Click the [Component]. Next, click the [Add component] to open the [New Component] dialog.

Figure 2-7 Smart Configurator: [Component] tab



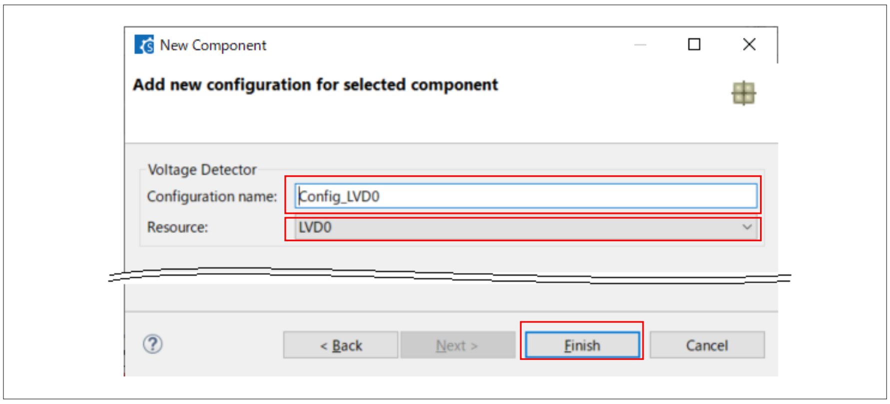
5. In the [New Component] dialog, select [Voltage Detector] and click the [Next].

Figure 2-8 Select [Voltage Detector]



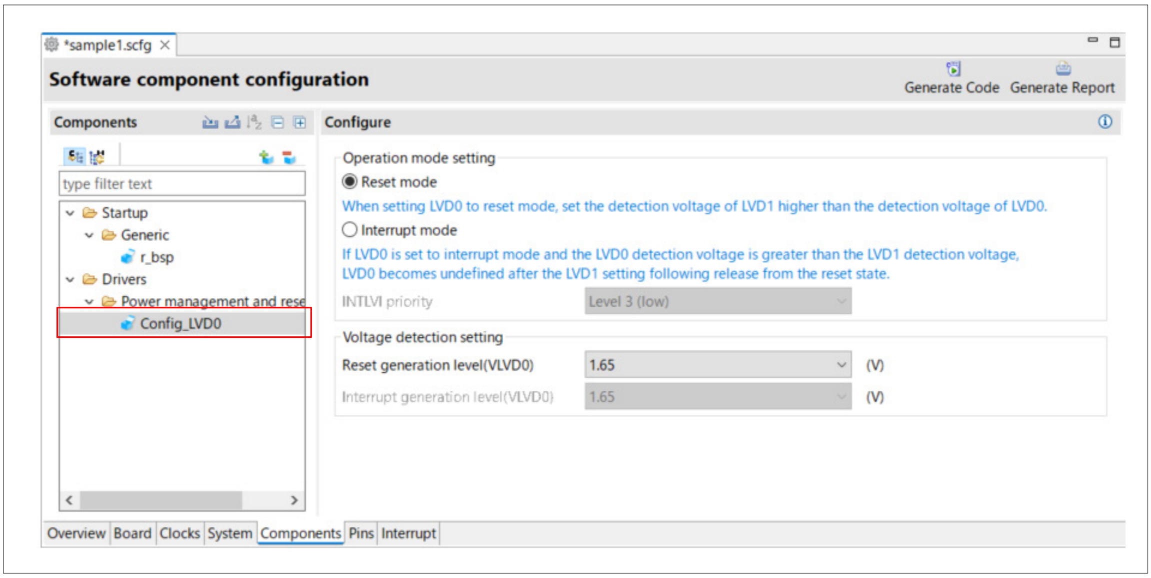
6. Select the [LVD0] at the [Resource]. Check the configuration name and click the [Finish]. (The configuration name can be changed to any name.)

Figure 2-9 Select resource and check configuration name [Voltage Detector]



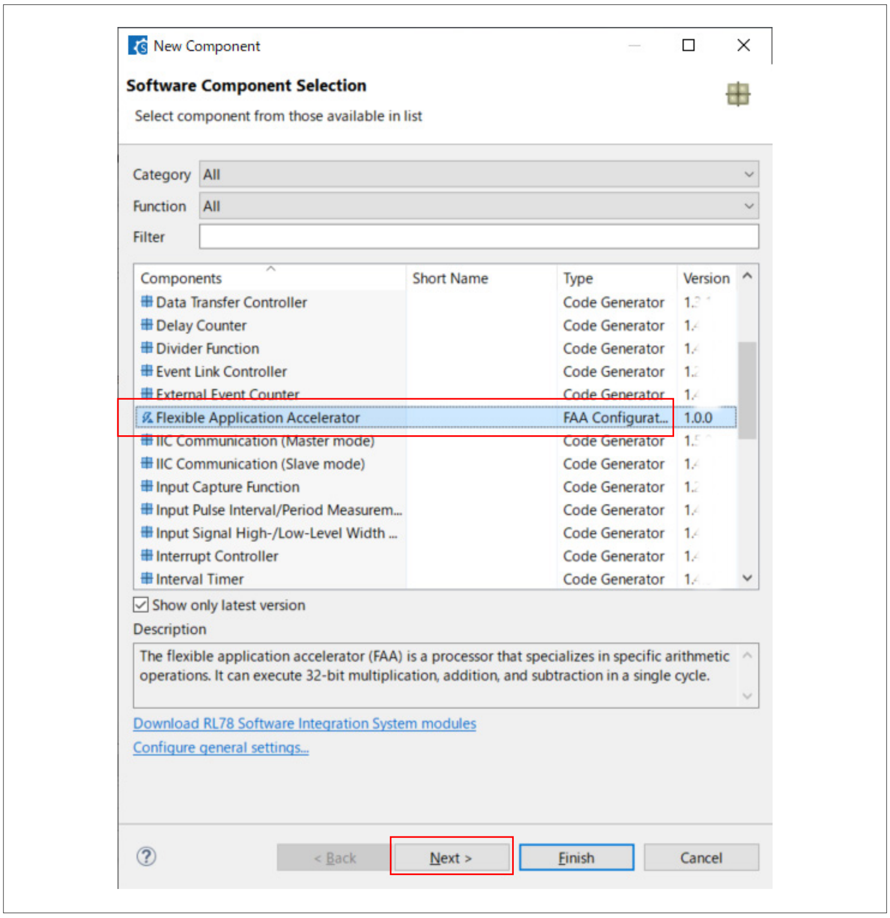
7. The Voltage Detector is added to the component tree. In the settings screen, set the Voltage Detector according to your system.

Figure 2-10 Smart Configurator: [Voltage Detector] setting screen



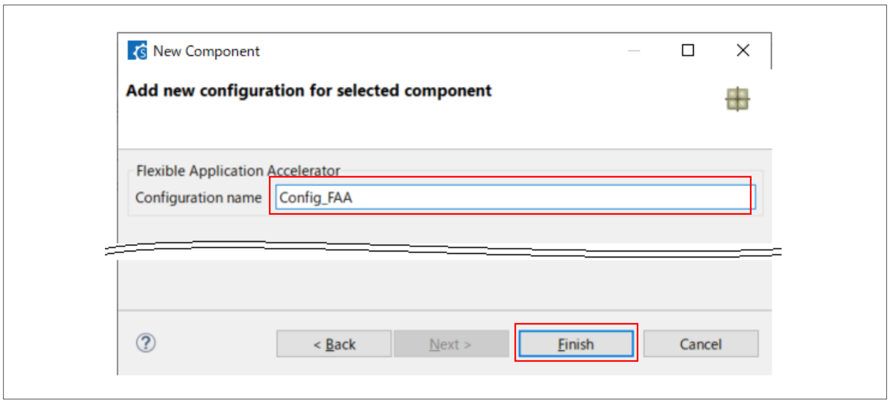
8. Open the [New Component] dialog again, select the [Flexible Application Accelerator] and click the [Next].

Figure 2-11 Select [Flexible Application Accelerator]



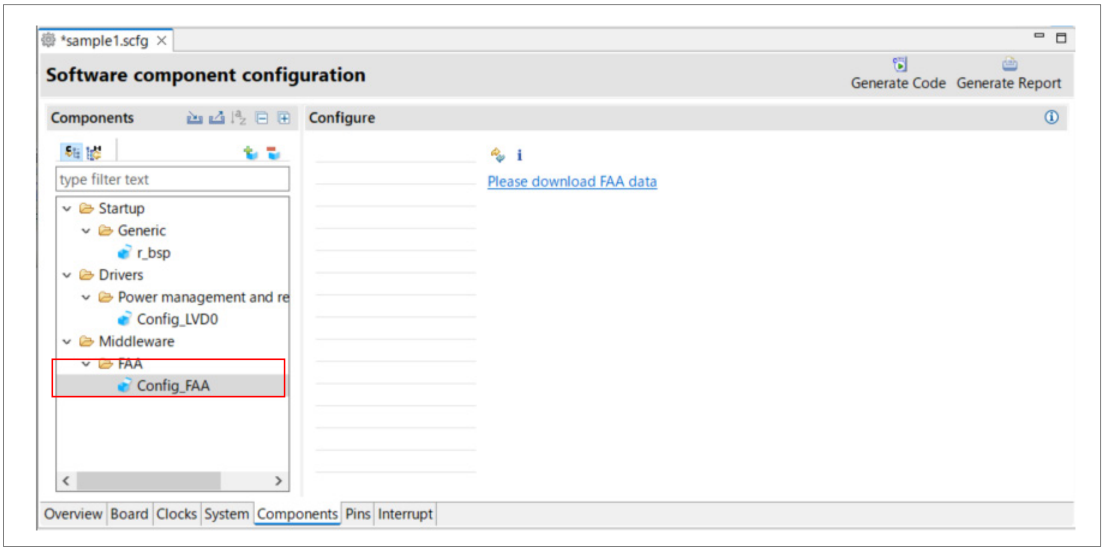
9. Check the configuration name and click the [Finish]. (The configuration name can be changed to any name.)

Figure 2-12 Select resource and check configuration name [Flexible Application Accelerator]



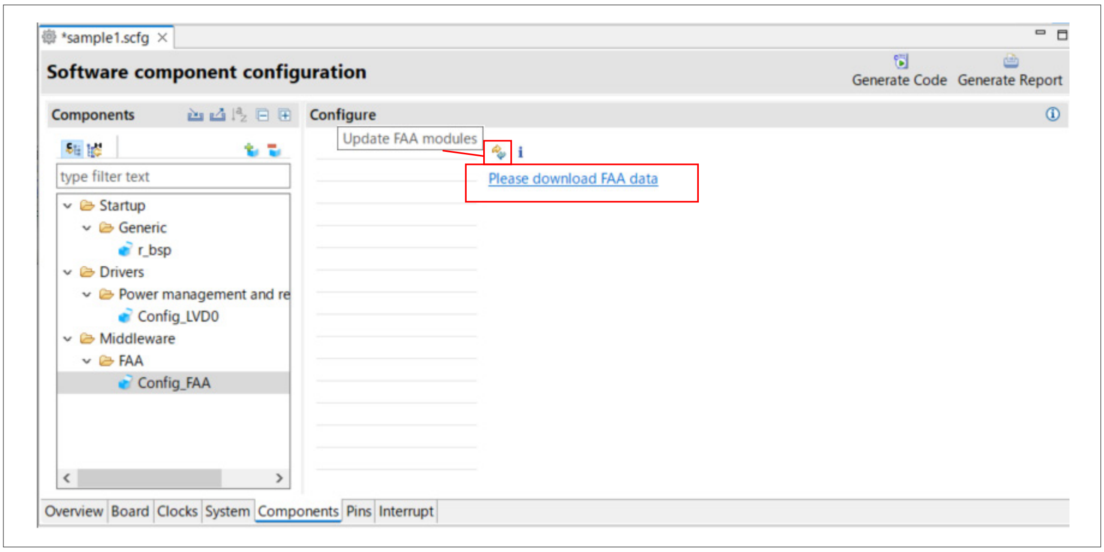
10. The Flexible Application Accelerator is added to the component tree.

Figure 2-13 Add [Flexible Application Accelerator] component



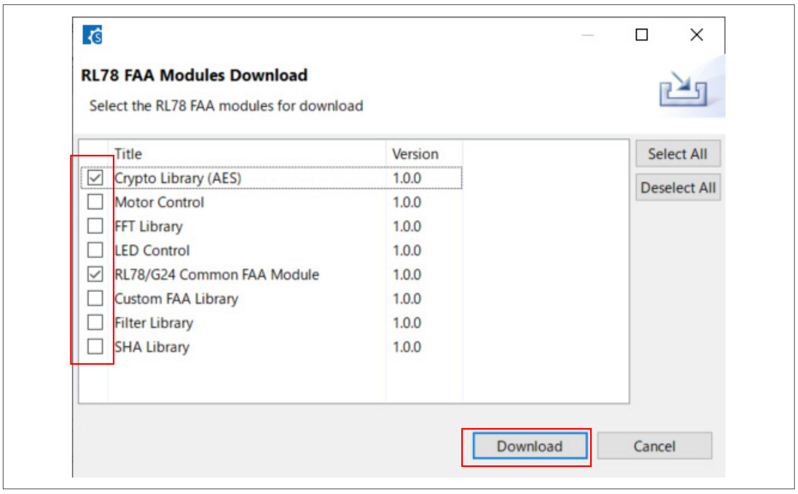
11. When the FAA component is used for the first time, the download of FAA libraries or template from the configurator's dedicated server is needed. Click the [Update FAA modules] or the [Please download FAA data] to download them. (Please use the [Update FAA modules] to check and obtain the latest version libraries as well.)

Figure 2-14 Update/Download FAA module (Library)



12. Select the library you want to download and click the [Download]. In the disclaimer dialog that follows, click the [Agree].

Figure 2-15 Download FAA module (Library)



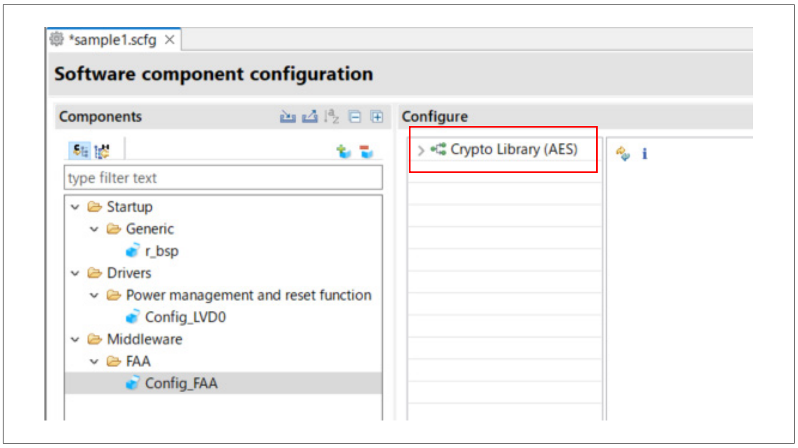
Remark. The content displayed on the actual download screen will differ.

Table 2-3 FAA library

Title	Overview
RL78/G24 Common FAA Library	The FAA program and data transfer routine described in 1.3.2 Transfer of Program and Data for FAA. When using FAA libraries/templates, this is always downloaded.
Custom Library	A template for writing FAA programs.
Others	FAA library of various function

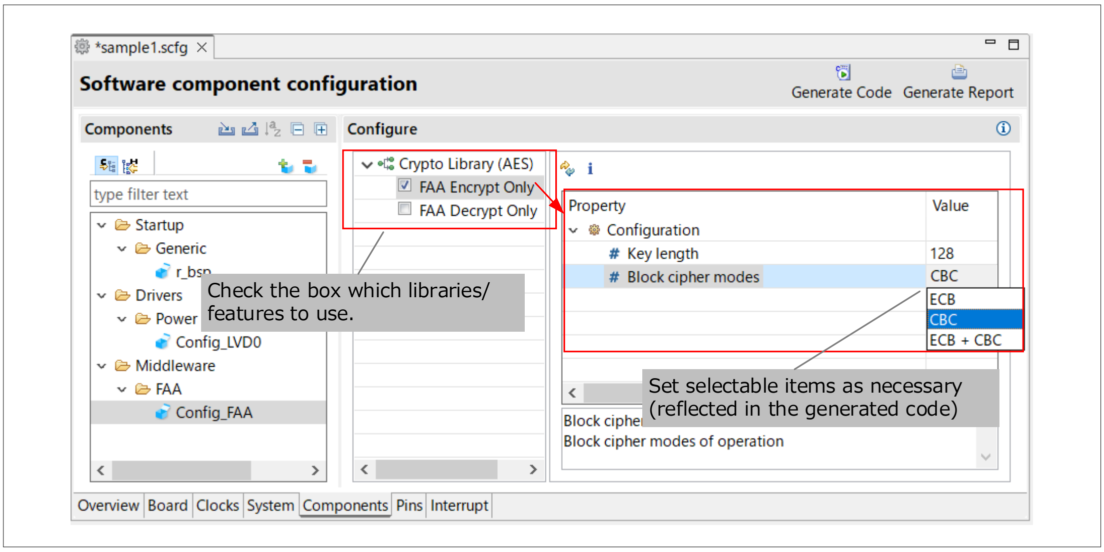
13. The downloaded libraries are added. (“RL78/G24 Common FAA Module” is not displayed.)

Figure 2-16 Added FAA library



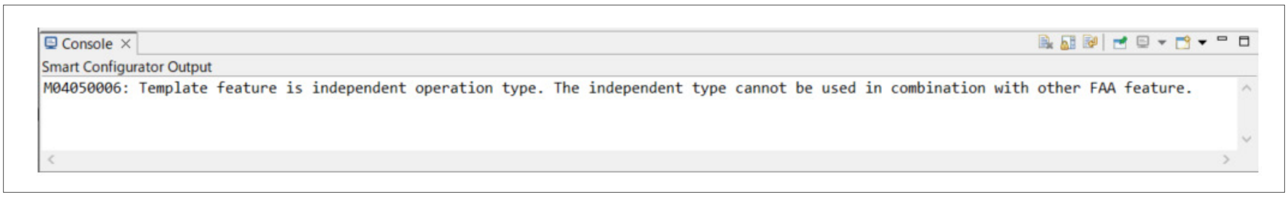
14. Check the box which libraries/functions you will actually use among the downloaded libraries. If there are any setting items in the properties of the checked function, set them as appropriate.

Figure 2-17 Select/set FAA library



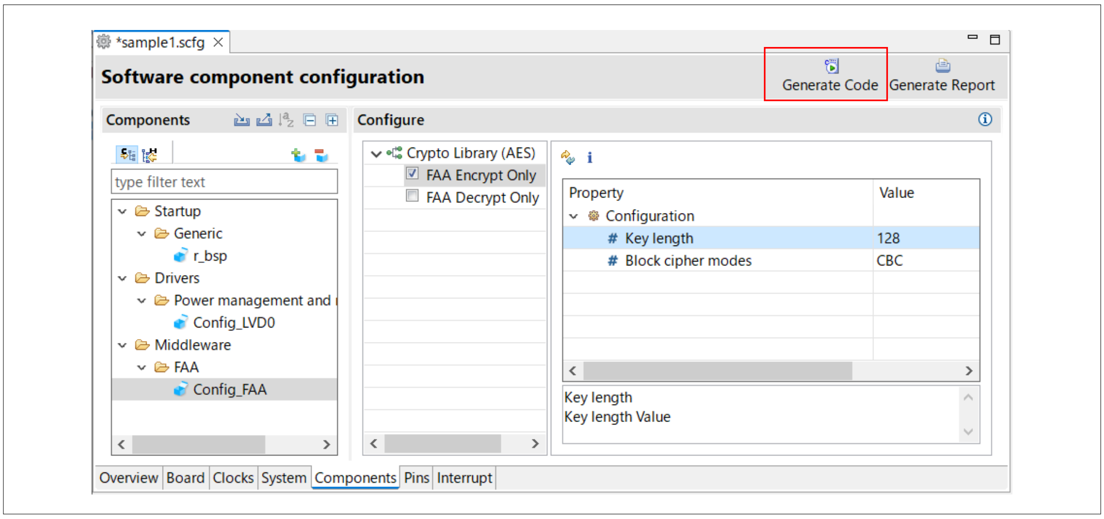
Remark. Two types of libraries and functions are provided: The subprocessor type, which can be used in conjunction with other functions, and the standalone type, which cannot. Do not use the standalone type simultaneously with any other library or function. When a standalone library or function is selected, selecting another library or function causes the following message to appear on the [Console] page.

Figure 2-18 Warning



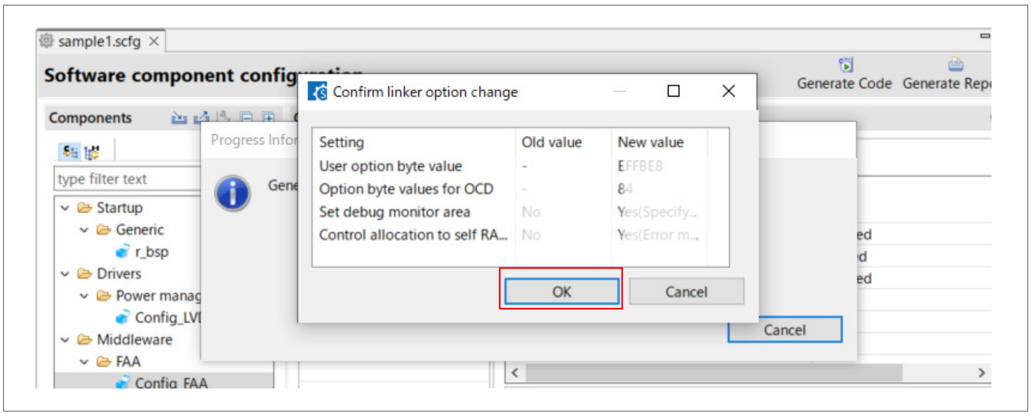
15. Click the [Generate Code] to generate source files of FAA library and added peripheral functions.

Figure 2-19 Generate Code



16. When the [Confirmation linker option change] dialog appears, click the [OK].

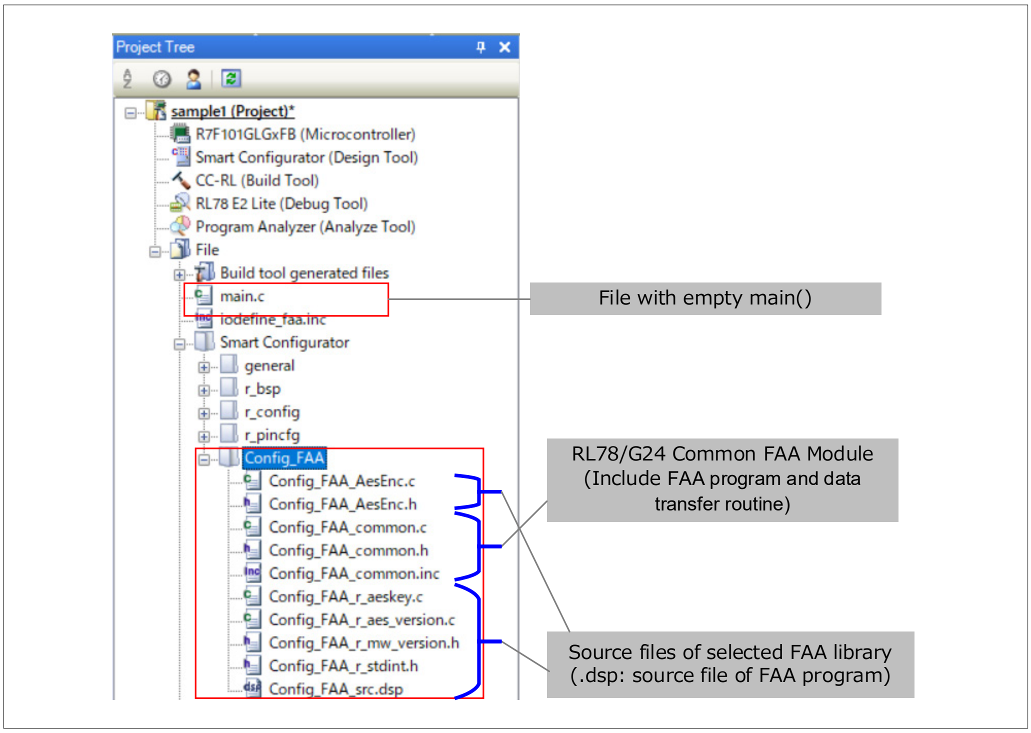
Figure 2-20 [Confirmation linker option change] dialog



Remark. Some items set in Smart Configurator's the [Clock], the [System] and the [Voltage Detector] (LVD0) are reflected in option settings of the build tool (CC-RL).

17. Source files of the FAA library and added peripheral functions are generated and registered in the project. The FAA library source files are shown below.

Figure 2-21 Registered FAA library source files



Remark 1. Some of the items set in “Clock”, “System” and “Voltage Detector (LVD0)” of the Smart Configurator (SC) are reflected in the Linker Option of Build Tool (CC-RL) .

Remark 2. For files other than the red frame above, refer to RL78 Smart Configurator User's Guide: CS+ (R20AN0580).

18. API functions to control the FAA are defined in the FAA library source file. Call these functions in the CPU program to operate the FAA. Create a CPU program according to your system.

For more information about API functions, see the documentation for each library.
https://www.renesas.com/rl78g24 (search for "documentation" on this page)

2.3.2 Overview of FAA library's File Structure

The overview of the FAA library file structure is shown below.

Table 2-4 Overview of FAA library's file structure

Library name	Files	Description
RL78/G24 Common FAA Library	<Config_FAA>_common.c <Config_FAA>_common.h	The transfer processing (R_Config_FAA_Create) and common functions to control the FAA are defined. The transfer processing is executed within the peripheral function initialization function (R_Systeminit) generated by SC, so there is no need to call it within the user program.
	<Config_FAA>_common.inc	SFRs for FAA are defined.
Custom FAA Library	<Config_FAA>_src.dsp	The template for the FAA source file.
Others	<Config_FAA>_XXX.c / asm / s <Config_FAA>_XXX.h /inc <Config_FAA>_src.dsp	FAA library of various functions. Refer to documents of each FAA library.

- <Config_FAA> is the configuration name set/checked in the step 9.
- "XXX" depends on each library.
- In the FAA source file (.dsp) provided by the FAA library and the template (Custom FAA Library), the code section name is defined as FAACODE and the data section name is defined as FAADATA.
- When using the Custom FAA Library, add your user code and data to the template. If you build the template as is, an error will occur.

2.4 Build Tool Option Setting

Before starting a build, set the build tool options required to build the FAA program. Some options are set by the Smart Configurator (SC) in 2.3.1 Adding FAA Component. Manually set the options for which “No” is indicated in the “Set by SC” column in Table 2-5.

For build tool options that are not described in this guide, set them if necessary.

How to open the build tool property:

Select the build tool node in the project tree, and then select the [View] menu -> [Property] or select the [Property] from the context menu.

Figure 2-5 shows the build tool options required to build the FAA program.

Table 2-5 Setting options of build tool

Tab	Category	Item	Description	Set by SC
FAA Assemble Options	Preprocess	Method for recognizing the text macro	Exact(-macro_identify exact)	Yes
Link Options	Section	Layout section automatically	Yes(-AUTO_SECTION_LAYOUT) or No	No
		Section start address	FAACODE,FAADATA/XXXX XXXX (hexadecimal number without “0x”) specifies an even address after address D8H in the code flash memory.	No
		ROM to RAM mapped section	FAACODE=FAACODER FAADATA=FAADATAR	Yes
		Allocate FAA memory area automatically	Yes or Yes(Automatically allocate sections by striding FAA memory area) Note2	Yes Note1

Note 1. SC sets “Yes”.

2. When the RAM size used by the user program (CPU program) is larger than 2304 bytes (the user RAM area before the FAA code area on RAM), manually set it to " Yes(Automatically allocate sections by striding FAA memory area) ". Also, when “Yes(Automatically allocate sections by striding FAA memory area)” is specified, set “Yes(-AUTO_SECTION_LAYOUT)” at “Layout section automatically”.

2.4.1 FAA Assemble Options

Figure 2-22 FAA Assemble Options

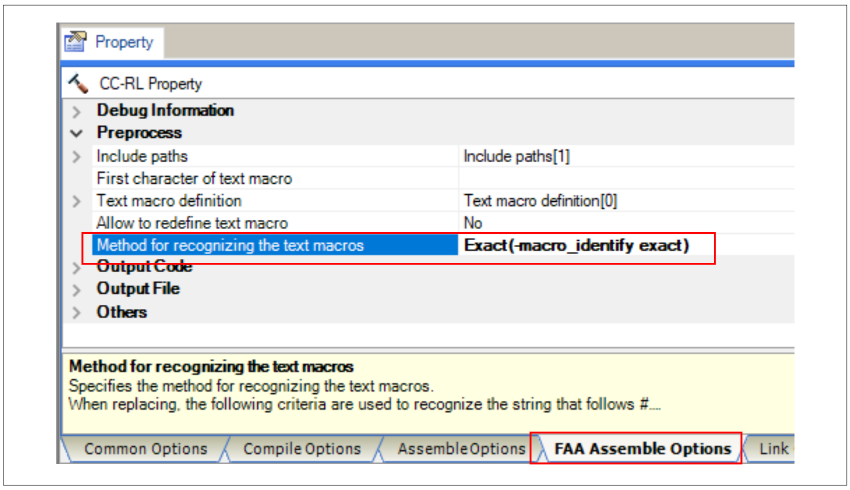


Table 2-6 FAA Assemble Options, Overview of settings

Category	Item	Description
Preprocess	Metho for recognizing the text macros	Set "Exact(-macro_identify exact)". A text macro is replaced in the FAA source file in units of tokens. Unless Exact is specified, replacement is performed even if the identifier to be replaced is included in another identifier.

2.4.2 Link Options

Figure 2-23 Link Options

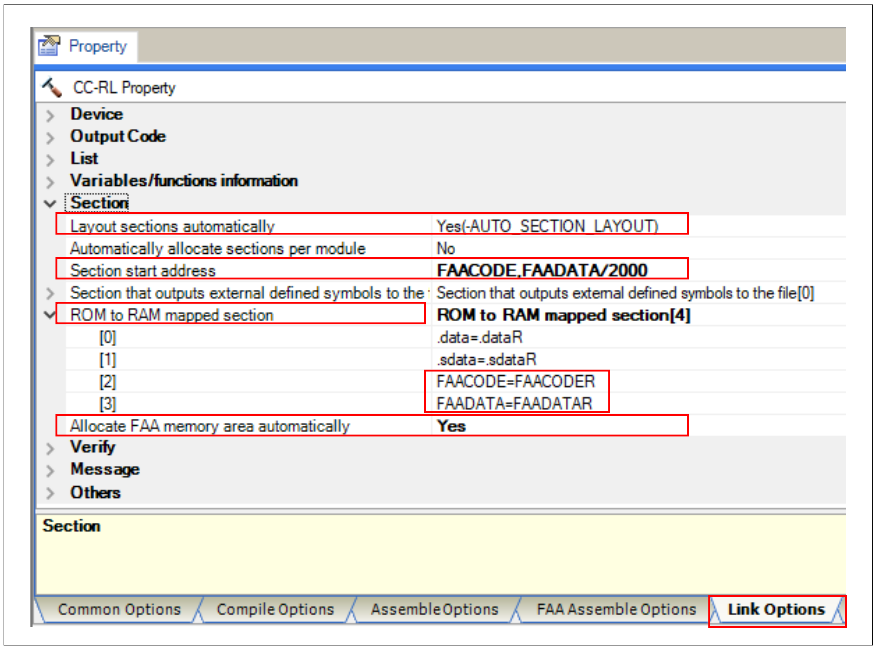


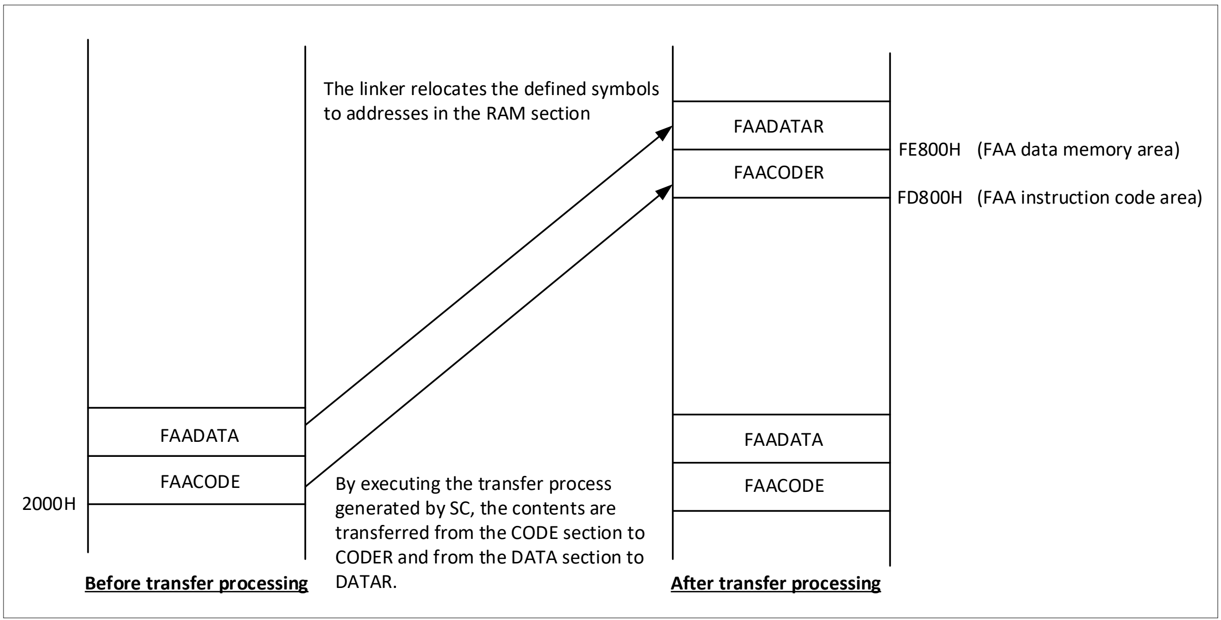
Table 2-7 Link Options, Overview of settings (1/2)

Category	Item	Description
Section	Layout sections automatically	Set "Yes(-AUTO_SECTION_LAYOUT)". Sections are automatically allocated based on information in the device file. When selecting "No", the address of each section used in the program need to be specified in "Section Start Address".
	Section start address	Set "FAACODE,FAADATA/address". Specify the address of code flash memory to store FAA programs and data. In the FAA program file (library or template) generated by the Smart Configurator (SC), the code section name is defined in FAACODE and the data section name is defined in FAADATA. Therefore, specify "FAACODE" and "FAADATA" as the section name. In addition, SC provides the processing (in Config_FAA_Common.c, generated by SC) to transfer the FAA program and data to the instruction code memory and data memory. The processing is performed in units of 2 bytes. Therefore, FAACODE and FAADATA must be aligned to the 2-byte boundary. specify an even number address after D8H. (at address 2000H in the example).

Table 2-8 Link Options, Overview of settings (2/2)

Category	Item	Description
Section	ROM to RAM mapped section	Set “FAACODE＝FAACODER,FAADATA=FAADATAR”. The definition symbols for the FAA program and data placed in the code flash memory will be relocated to the internal RAM (instruction code memory and data memory). If relocation is not performed, the addresses of the FAA program and data symbols will remain in the code flash memory area, and symbol information cannot be handled correctly during debugging. The left side specifies the FAA program and data sections located in code flash memory. The right side specifies the section of RAM to be transferred. In the processing to transfer the FAA program and data to the instruction code memory and data memory (in Config_FAA_Common.c generated by SC), FAACODER and FAADATAR is handled as the transfer destination RAM section, so the right side specifies FAACODER and FAADATAR.
	Allocate FAA memory area automatically	Set “Yes”. Reserve a dedicated area for FAA in the internal RAM. Variables for the CPU program will not be placed in the FAA instruction code memory (FD800H-FE7FFH) or data memory (FE800H-FEFFFH) in the internal RAM.

Figure 2-24 Memory image before and after transfer processing



2.4.3 Program Building

After setting the build tool options necessary to build the FAA program, build it. There are several ways to run a build. Two methods are described here.

- Select the [Build] menu -> [Build Project] (Figure 2-25)
- Click the [Builds the project] on the toolbar (Figure 2-26)

Figure 2-25 [Build] menu

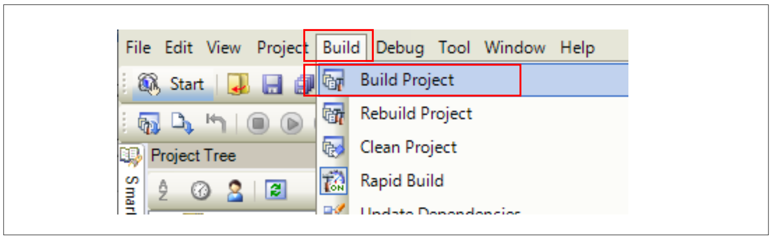
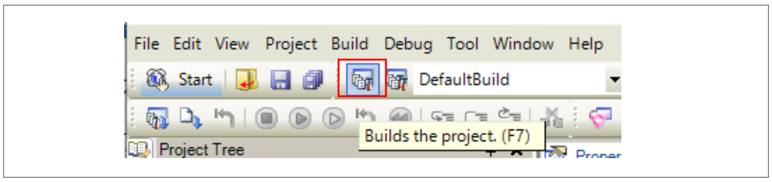


Figure 2-26 Build tool bar



2.5 Debug Tool Option Setting

Before downloading an executable object to the RL78/G24 Fast Prototyping Board, set the debug tool options required to debug an FAA program. Some options are set by the Smart Configurator (SC) in 2.3.1 Adding FAA Component. Manually set the options for which "No" is indicated in the "Set by SC" column in Table **2-9**. For debug tool options that are not described in this guide, set them if necessary.
After setting the required options, download the object.

How to open the debug tool property:

Select the debug tool node in the project tree, and then select the [View] menu -> [Property] or select the [Property] from the context menu.

Table 2-9 shows the build tool options required to build the FAA program.

Table 2-9 Setting options of debug tool

Tab	Category	Item	Description	Set by SC
Connect Settings	FAA	Debug FAA	Yes	Yes
Debug Tool Settings	Memory	FAA memory space (n) (n= 1 - 4)	Instruction code space or Data space	No
	Break	Stop FAA when stopping	No or Yes	No
Download File Settings	Download	Specify code section name defined in FAA source file	FAACODER	Yes
		Specify data section name defined in FAA source file	FAADATAR	Yes

2.5.1 Connect Settings

Figure 2-27 Connect Settings

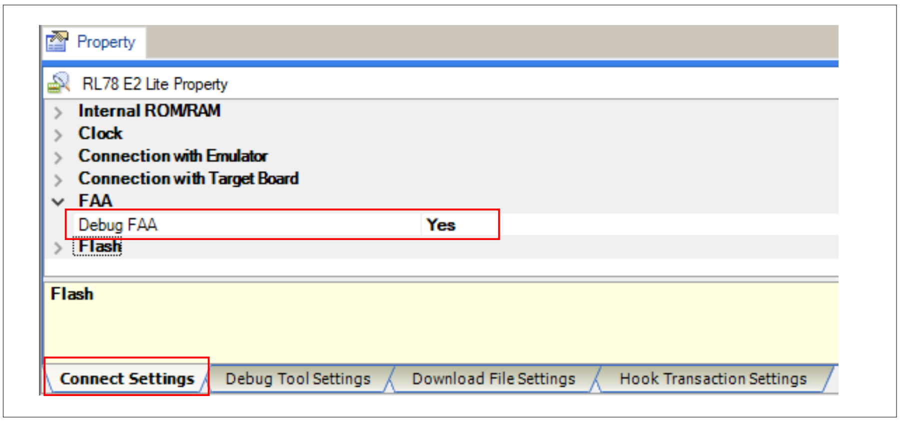


Table 2-10 Connect Settings, Overview of settings

Category	Item	Description
FAA	Debug FAA	Set "Yes". Enable source debugging of the FAA program.

2.5.2 Debug Tool Settings

Figure 2-28 Debug Tool Settings

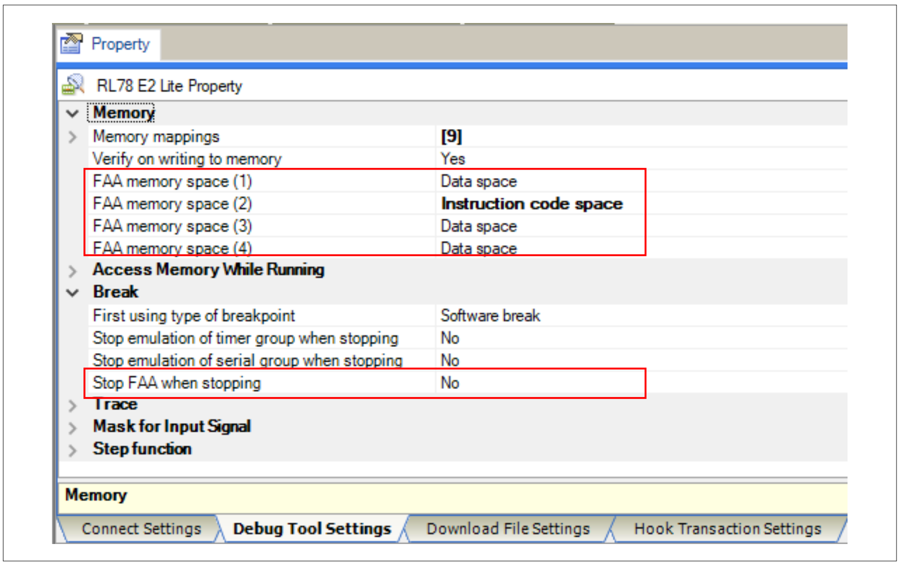


Table 2-11 Debug Tool Settings, Overview of settings

Category	Item	Description
Memory	FAA memory area(n) (n=1 - 4)	Set the FAA space corresponding to FAA memory space (n). The debugger can display up to four [Watch] panels and four [Memory] panels each. When debugging the FAA program, the space set here is displayed in each panel.
Break	Stop FAA when stopping	Set “No” or “Yes”. If the debug target is a CPU, select whether to stop the FAA program when the CPU program is stopped by the stop button.

2.5.3 Download File Settings

Figure 2-29 Download File Settings

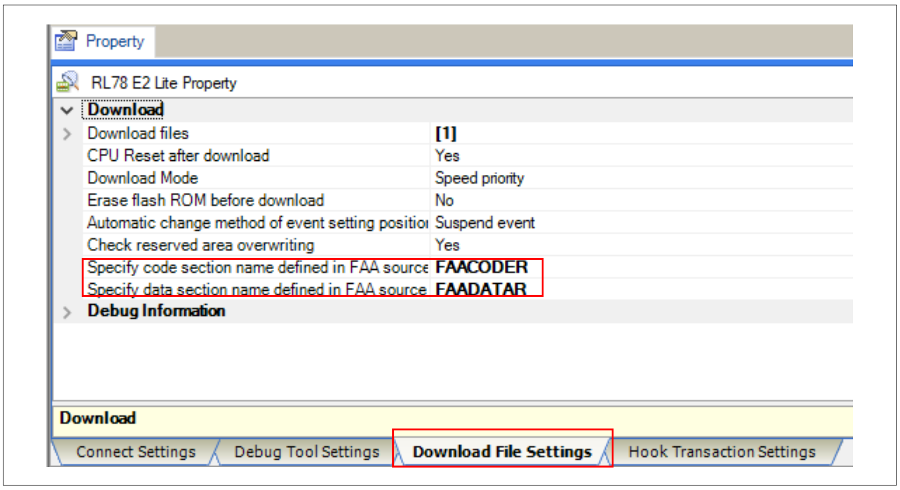


Table 2-12 Download File Settings, Overview of settings

Category	Item	Description
Download	Specify code section name defined in FAA source	Set “FAACODER”. In the FAA program file (library or template) generated by the Smart Configurator (SC), the code section name is defined in FAACODE. However, specify the section name FAACODER to be relocated to the RAM area. Reference: The link option “Section mapped from ROM to RAM”.
	Specify data section name defined in FAA source	Set “FAADATAR”. In the FAA program file (library or template) generated by the Smart Configurator (SC), the data section name is defined in FAADATA. However, specify the section name FAADATAR to be relocated to the RAM area. Reference: The link option “Section mapped from ROM to RAM”.

2.5.4 Program Download

After setting the debug tool options necessary to debug the FAA program, connect PC and RL78/G24 Fast Prototyping Board and then download the object. There are several ways to download. Two methods are described here.

- Select the [Debug] menu -> [Download] (Figure 2-30)
- Click the [Download] on the toolbar (Figure 2-31)

Caution1: Before downloading, check the power supply in the [Connect Settings] tab – [Connection with Target Board] of the debug tool option.

Caution2: The FAA program is not placed in the instruction code memory by simply downloading the object. You need to transfer the FAA program and data from the code flash memory to the instruction code memory and data memory by using the CPU program.

The RL78 Smart Configurator provides transfer processing function (R_Config_FAA_Create) as FAA components. The transfer processing function is executed in the initialization routine (R_Systeminit) before the main function is executed, and the transfer is performed.

Figure 2-30 [Debug] menu

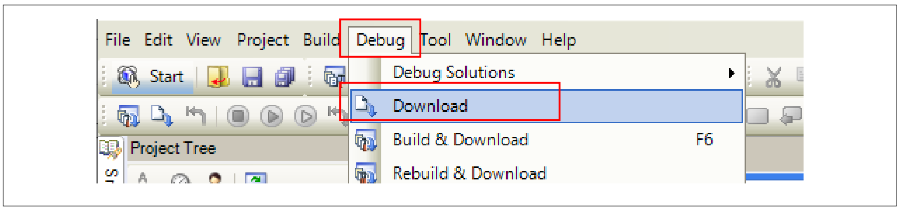
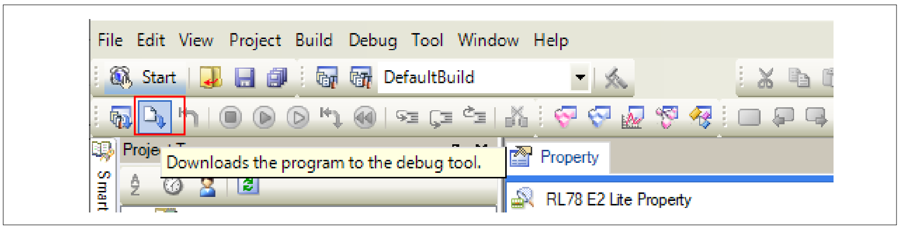


Figure 2-31 Debug tool bar



2.6 FAA Program Debug

2.6.1 Debug Target

When debugging the RL78/G24 program, select whether to debug the CPU or FAA. Select by using one of the following methods.

- Select the [View] menu -> [Debug Manager] to open [Debug Manager]. Select the debug target on it. (Figure 2-32)
- Select the target debug on the status bar. (Figure 2-33)

Figure 2-32 Open [Debug Manager]

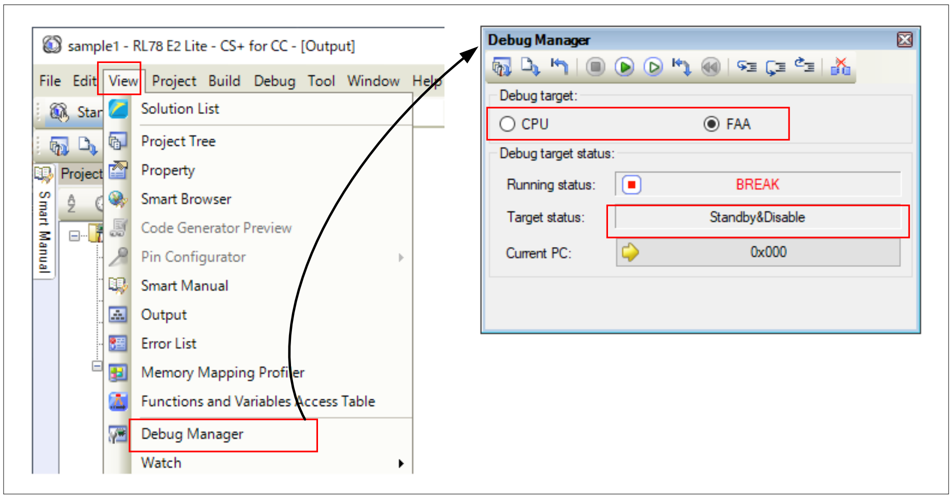
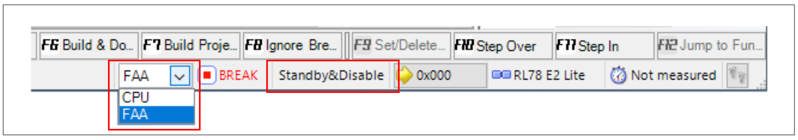


Figure 2-33 Status bar



Address information is displayed in the address area only for the source file to be debugged, and debugging operations such as step execution are possible at the source level.

It is possible to change the debug target in the following status.

Table 2-13 Change debug target

Current debug target	Status	Change from CPU to FAA	Change from FAA to CPU
CPU	CPU program stopping	Available	—
CPU	CPU program running	Not available	—
FAA	FAA program stopping	—	Available
FAA	FAA program running	—	Available

Additionally, CPU or FAA status that is debug target is displayed in the debug manager and status bar.
When FAA is the debug target, the status of FAA is as follows. If multiple statuses exist at the same time, the statuses are displayed separated by "&".

Table 2-14 FAA status

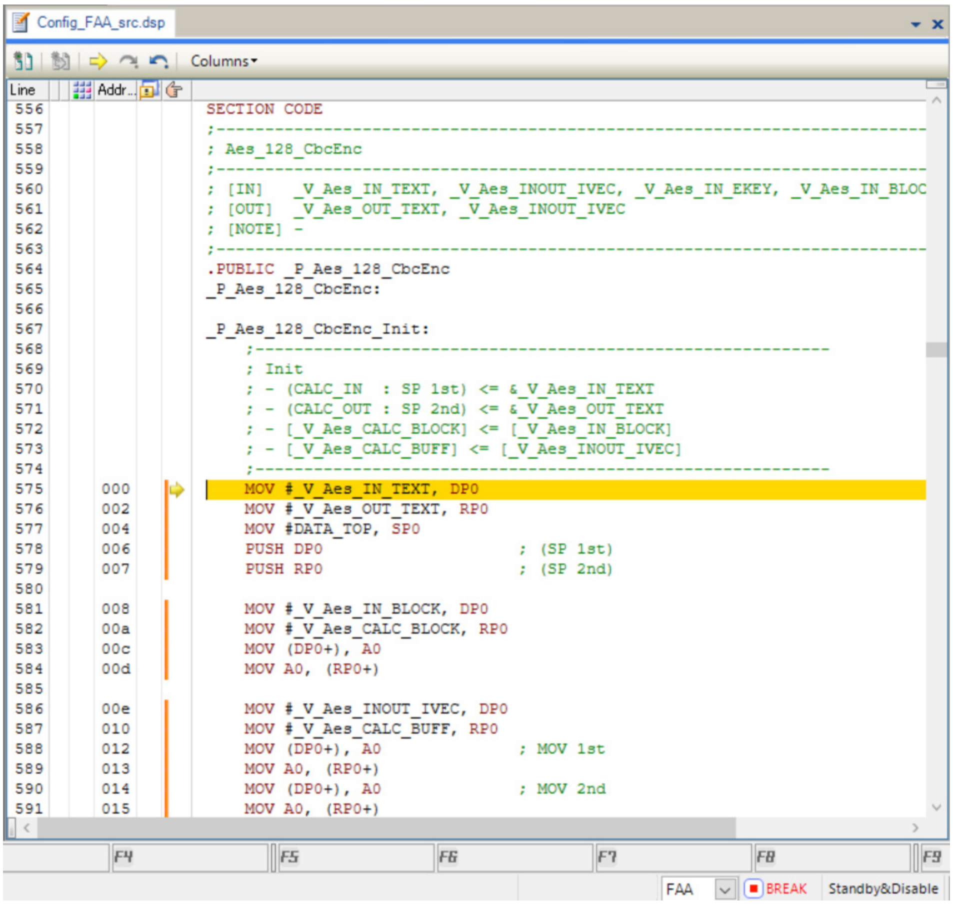
Status display	FAA status
Standby	Stops supply of an input clock to FAA. (FAAEN bit = 0)
Disable	Disables FAA operation. (ENB bit = 0)
Sleep	FAA in Low power consumption mode (SLP bit = 1 and EXE bit = 0)

2.6.2 Source File Display

After selecting the FAA as the debug target, display the .dsp file containing the FAA program on the [Editor] panel. The address information appears in the address area, and debug operations such as step execution can be performed at the FAA source level.

The address area indicates the addresses in the FAA instruction code memory space. The address area is not displayed when the debug target is CPU.

Figure 2-34 Source file display



2.6.3 Go/Stop

When selecting FAA as the debug target, FAA source debugging is enabled. There are several ways to go/stop FAA program. Two methods are described here.

- Select the [Debug] menu -> [Go] / [Stop]. (Figure 2-35)
- Click the [Go] / [Stop] on the toolbar. (Figure 2-36)
- Click the [Go] / [Stop] on the toolbar of the Debug Manager. (Figure 2-37)

Figure 2-35 [Debug] menu

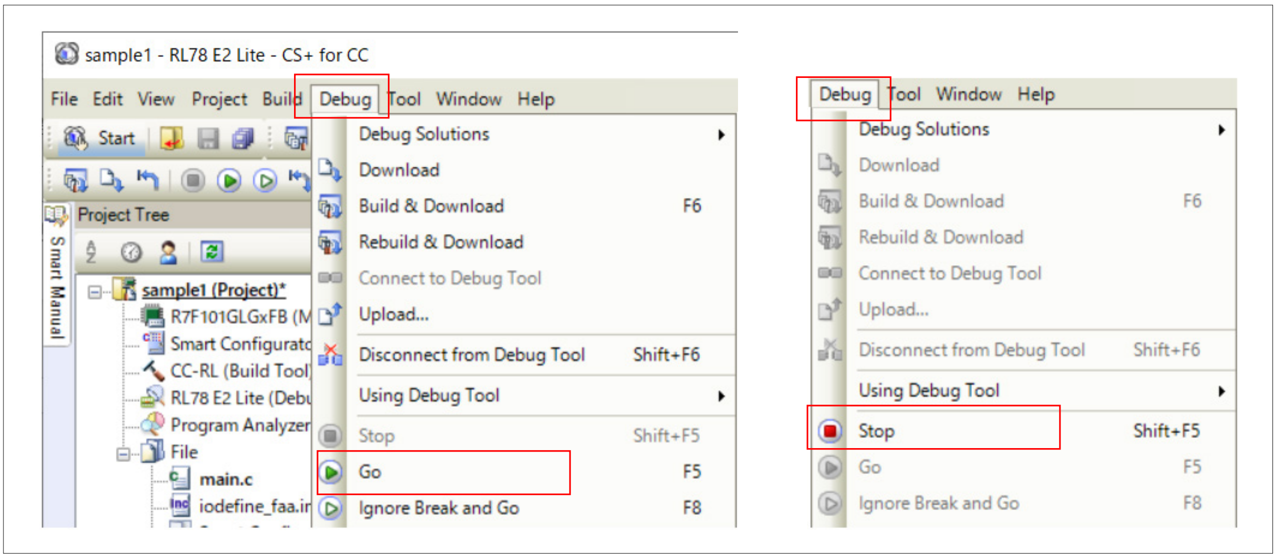


Figure 2-36 Debug tool bar

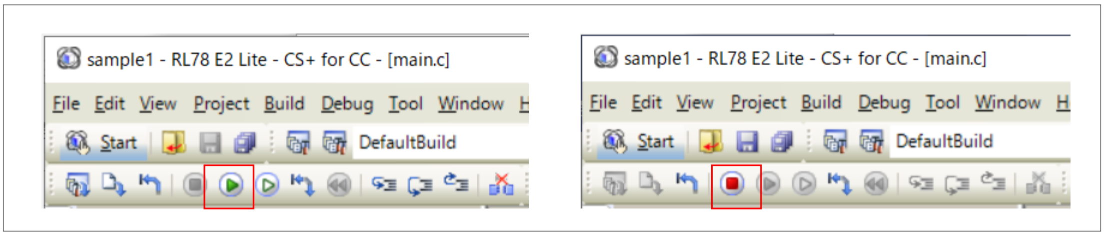
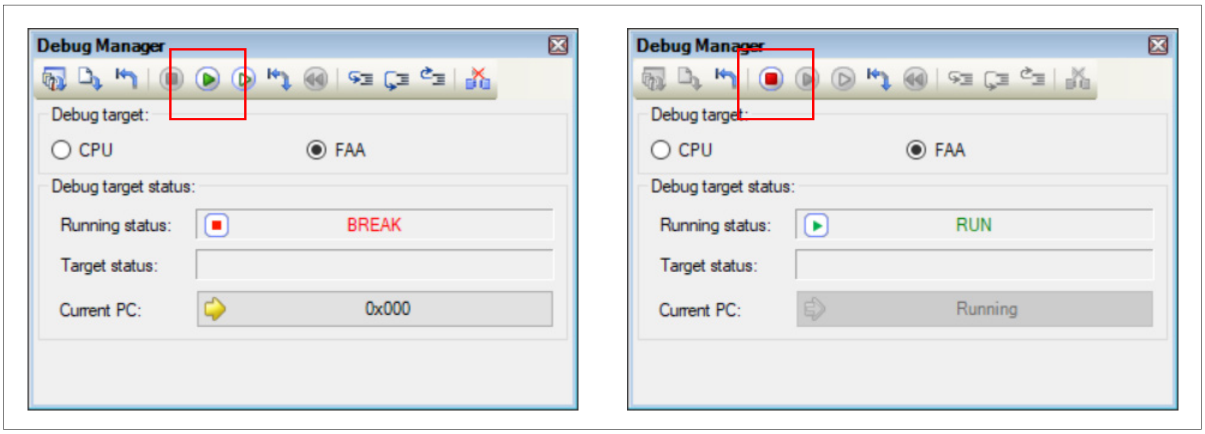


Figure 2-37 Debug Manager



The FAA program controls are as follows:

- ✓ If the FAA status is “Standby” or “Disable”, program execution cannot start and other debug operations such as step execution are also disabled. When using FAA libraries, FAA programs runs by calling the start function (that executes FAAEN=1, ENB=1) provided by each FAA library.
- ✓ When the debug target is FAA, the operation to execute or stop programs only executes or stops the FAA program. The CPU program is not executed or stopped in synchronization. However, you can use a debug tool option so that stopping a CPU program also stops the FAA program when the debug target is CPU. To do this, on the [Debug Tool Settings] tab, under the [Break] category, select [Yes] for [Stop the FAA when stopping the program].
- ✓ Step execution is applicable only to the FAA.
- ✓ Reset operation performs a software reset for the FAA. The whole MCU (CPU and peripheral functions) are not reset. When the debug target is CPU, the whole MCU (CPU and peripheral functions) are reset.
- ✓ Do not proceed with debugging of the FAA during execution of a CPU program that includes operations with the WIND register. Since the debugger temporarily rewrites the WIND register in the debugging operations for the FAA, the use of FAA debugging may make operation of the program being executed by the CPU incorrect.
- ✓ Disconnect the debug tool only when the CPU and FAA programs are stopped.

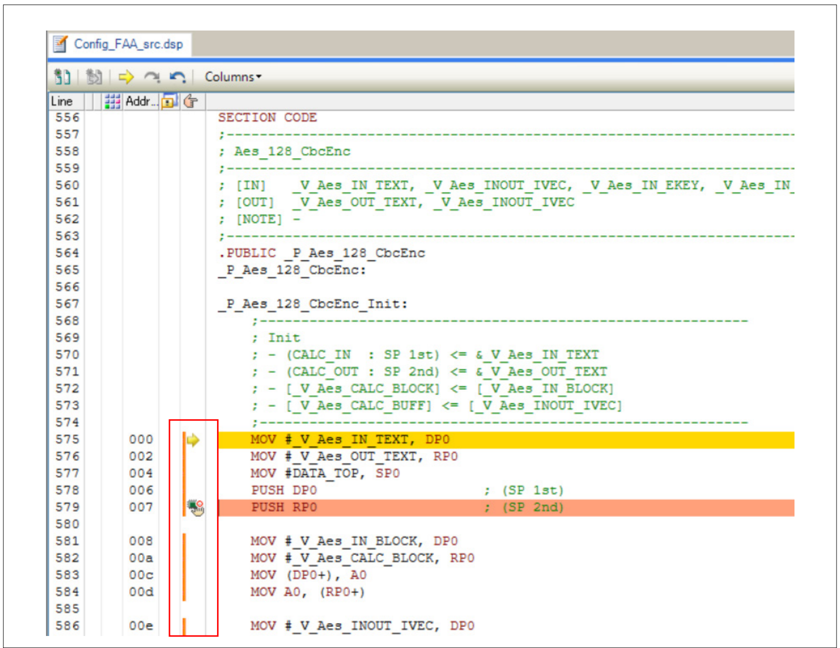
2.6.4 Breakpoint

After selecting the FAA as the debug target, display the FAA source on the [Editor] panel. You can set a breakpoint by clicking the main area on the row on which you want to set the breakpoint. To cancel a breakpoint, click the icon set for the breakpoint.

The breakpoint controls for the FAA program are as follows:

- ✓ 4 points hardware breaks are available. (Break after execution)
- ✓ If the FAA is stopped after detecting a hardware break, the CPU is not synchronously stopped.

Figure 2-38 FAA program, breakpoint setting



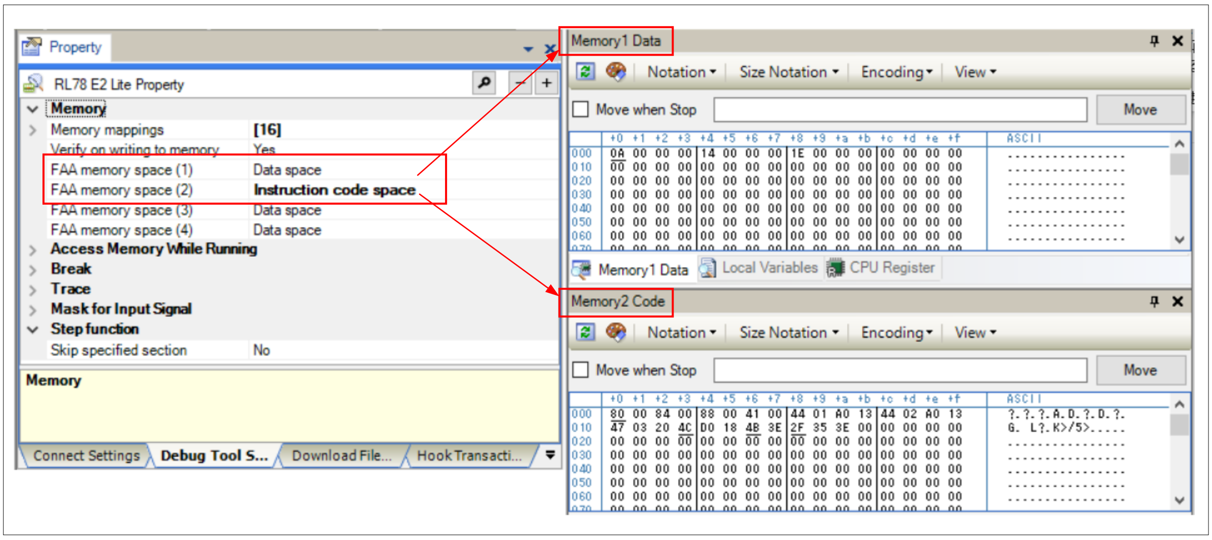
2.6.5 Memory

When selecting FAA as the debug target, FAA instruction code memory and data memory are displayed in the [Memory] panel.

The memory display control for the FAA are as follows:

- ✓ The [Memory] panel that specified in 2.5.2 Debug Tool Settings displays FAA instruction code memory and data memory.
- ✓ Address is the FAA space.
- ✓ When the debug target is CPU, CPU memory is displayed regardless of the settings in 2.5.2 Debug Tool Settings.
- ✓ The display cannot be updated while the FAA program is running.
- ✓ If the FAA status is “Disable” or “Standby”, the displayed content will be undefined.

Figure 2-39 [Memory] panel



2.6.6 Symbol (Label)

When selecting FAA as the debug target, the symbols (labels) defined in the FAA program are displayed in the [Watch] panel.

The watch display control for the FAA are as follows:

- ✓ Data for the FAA has 32-bit width. However, 8-bit width data is displayed when a symbol is registered on a [Watch] panel. Therefore, change the [Size Notation] setting to [4 Bytes]. (Figure **2-40**)
- ✓ Address is the FAA space address.
- ✓ If [Offset format] is specified for the watch-expression, the value is displayed every 4 bytes from the first symbol.
- ✓ If the debug target is CPU, the value column indicates a question mark (?).
- ✓ If the FAA status is “Standby” or “Disable”, the display contents are undefined.

Remark. To make a symbol accessible to the CPU program, it must be defined with a name starting with "_" and must be declared public in the FAA program.

Figure 2-40 [Watch] panel (Size change of symbol)

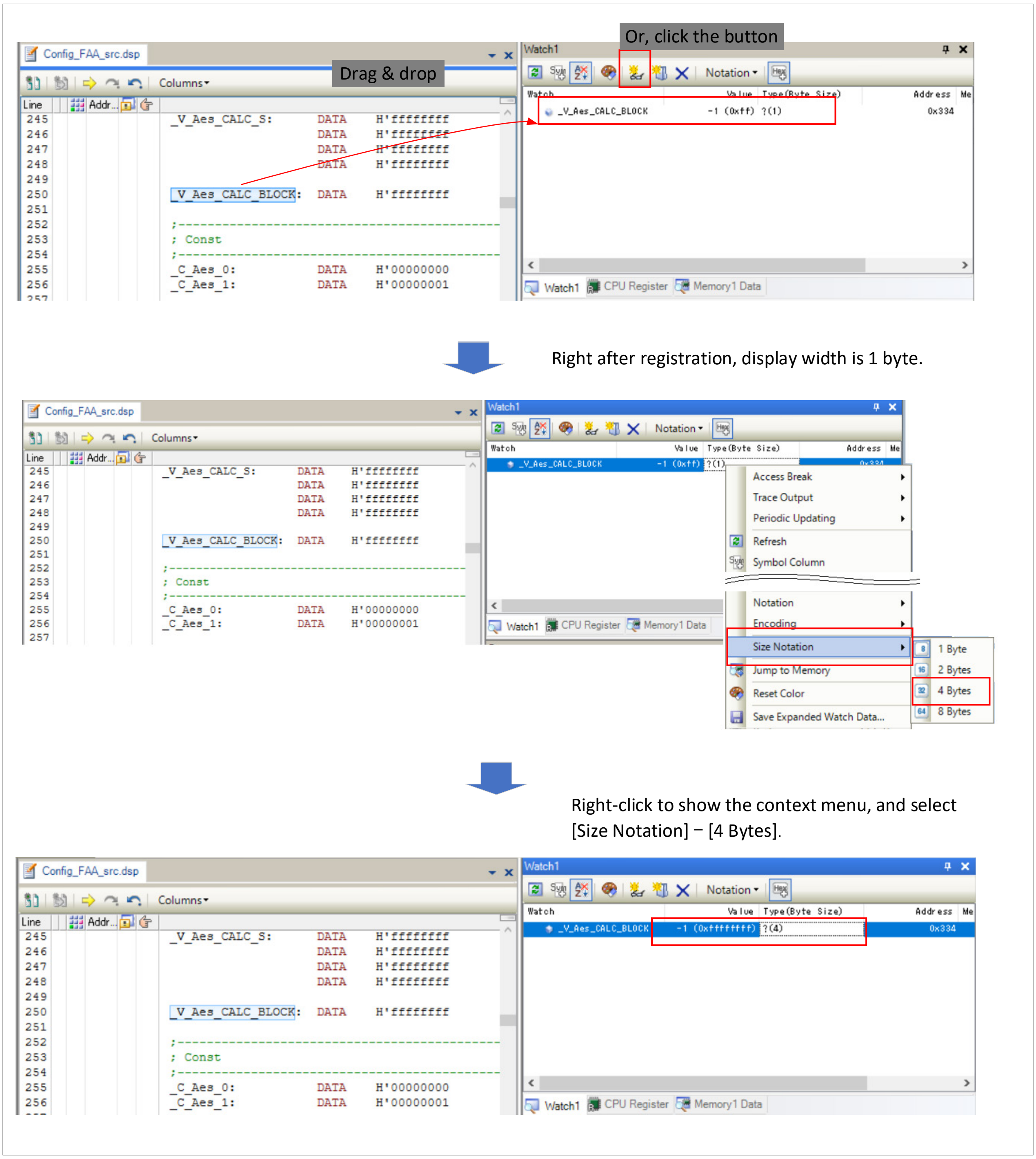
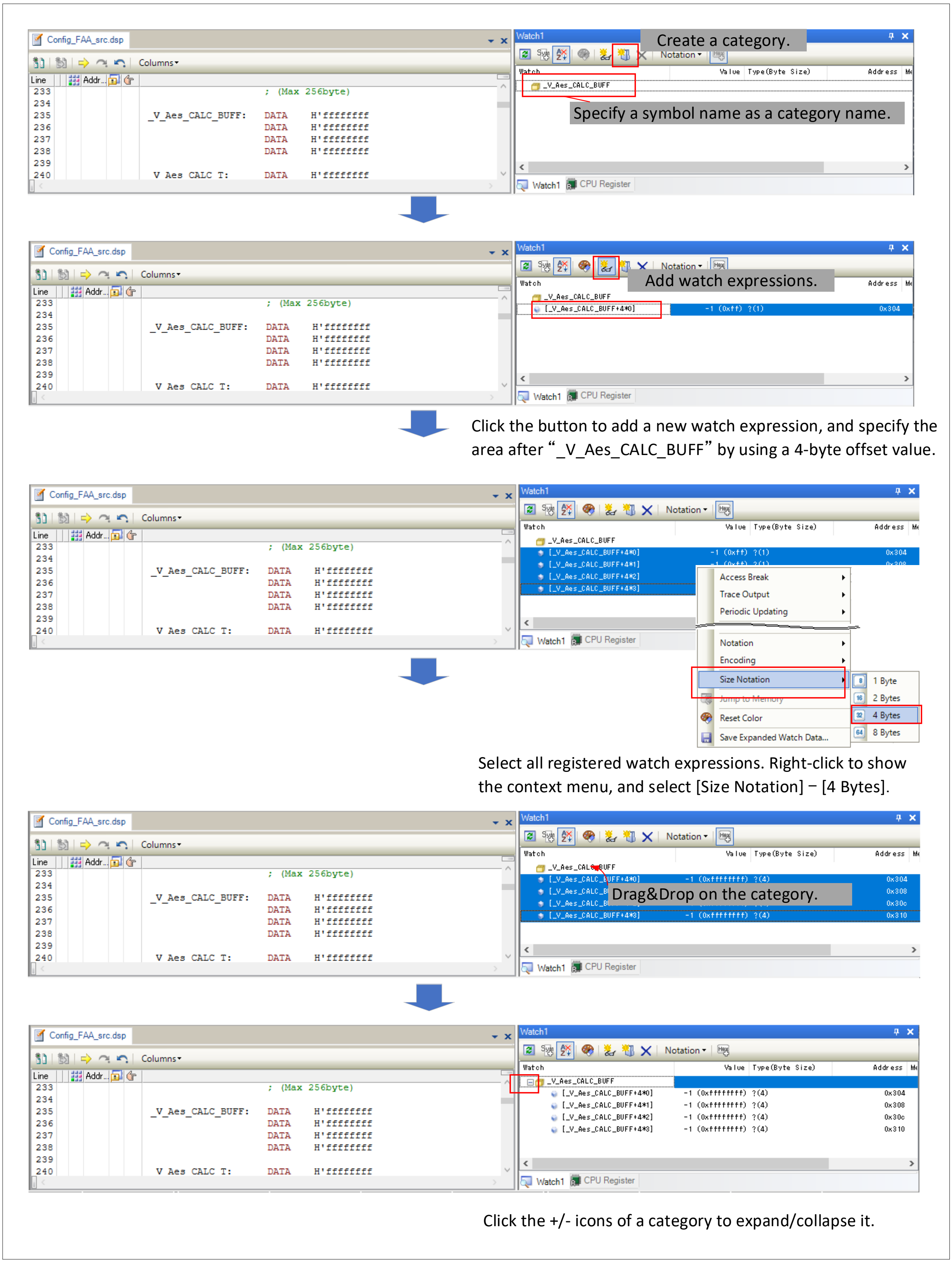


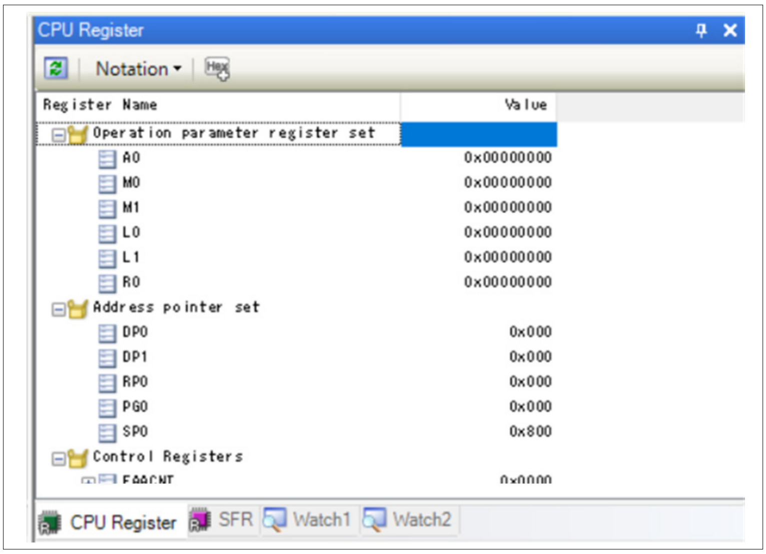
Figure 2-41 [Watch] panel (How to specify with [Offset format])



2.6.7 Register

When selecting FAA as the debug target, the operation parameter register set, address pointer set, the processor control register, etc. are displayed in the [CPU Register] panel.

Figure 2-42 [CPU Register] panel



2.6.8 SFR

When selecting FAA as the debug target, the [SFR] panel displays only SFRs (Special Function Register) that FAA can access. There are two types of SFRs that the FAA can access.

- SFRs of the FAA

  Registers that are not affected by the address bus select register (ADBSEL) settings and can be accessed via the FAA bus.

- Registers of the peripheral functions

  Registers that can be accessed via the FAA bus when "access from the FAA" is selected in the ADBSEL register.

  There are two different types of register access to the peripheral functions as described below.

  · Access to a peripheral function register through the FAA address map

  · Access to a peripheral function register by using the FAA address pointer (FAAAP)

  For the address bus select register (ADBSEL) and how to access, refer to RL78/G24 User's Manual: Hardware (R01UH0961).

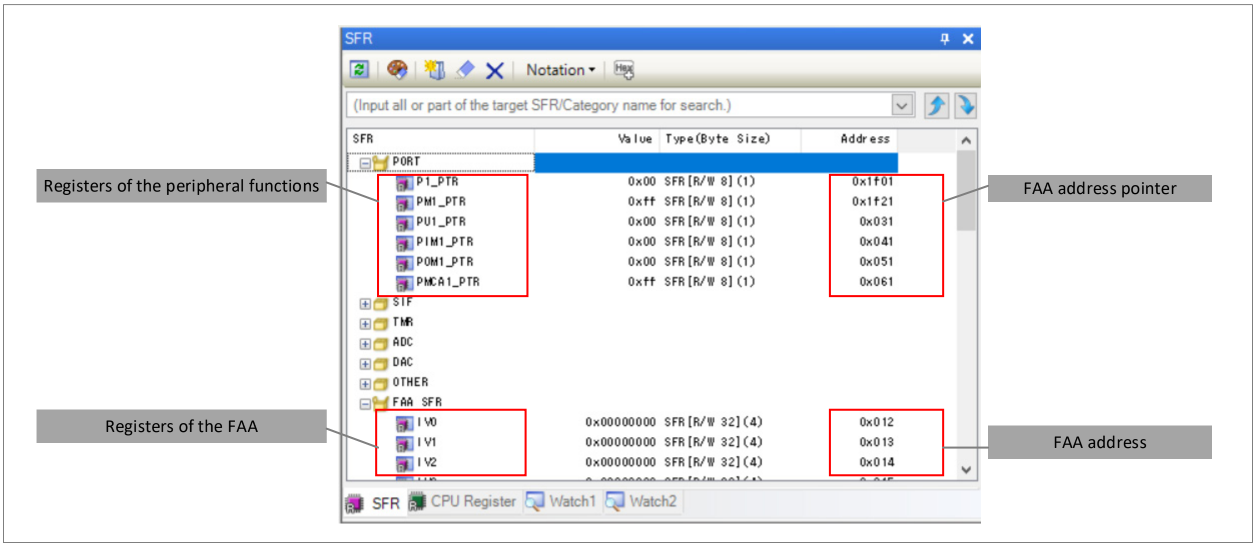
The SFR display control for the FAA are as follows:

✓ The address area for the FAA SFR displays the FAA addresses.

✓ Access to some peripheral function SFRs is enabled by using the address bus function to permit bus access from the FAA. For such SFRs, the display name is suffixed by "_PTR". The address displayed in the address field is the FAA address pointer values that be set in the FAA address pointer (FAAAP) when accessing using the FAAAP register.

✓ The debugger reads or writes peripheral function SFR values through bus access from the CPU. Therefore, it cannot access the peripheral function SFRs for which bus access from the FAA is selected by using the address bus selection function, and the displayed values for these SFRs are undefined. To display the values of the peripheral function SFRs for which bus access from the FAA is selected, see 3.5 Sample Script Specification.

Figure 2-43 [SFR] panel



3. Sample Project

This section describes how to display the SFR values of peripheral functions in the CS+'s [SFR] panel when debugging a FAA program using sample code and sample scripts.

3.1 Specifications

3.1.1 Specification Overview

This sample code uses a 16-bit timer KB30 (TKB30) to perform two PWM outputs.

PWM output is connected to LED1 and LED2. Initialize TKB30 using the CPU program, count the number of TKB30 timer interrupts (INTTKB00), create a fixed cycle (500ms) timing, and start FAA operation at a fixed cycle.

The FAA program controls the LED brightness by changing the duty ratio of the PWM output. After changing the duty ratio, the operation stops.

Table 3-1 Peripheral Functions and Their Usage

Peripheral	Usage
16-bit timer KB30 (TKB30)	Output PWM from TKBO00 pin and TKBO01 pin
Flexible application accelerator (FAA)	Change the duty ratio of PWM output from TKBO00 pin and KBO01 pin

Figure 3-1 Operation overview of PWM output

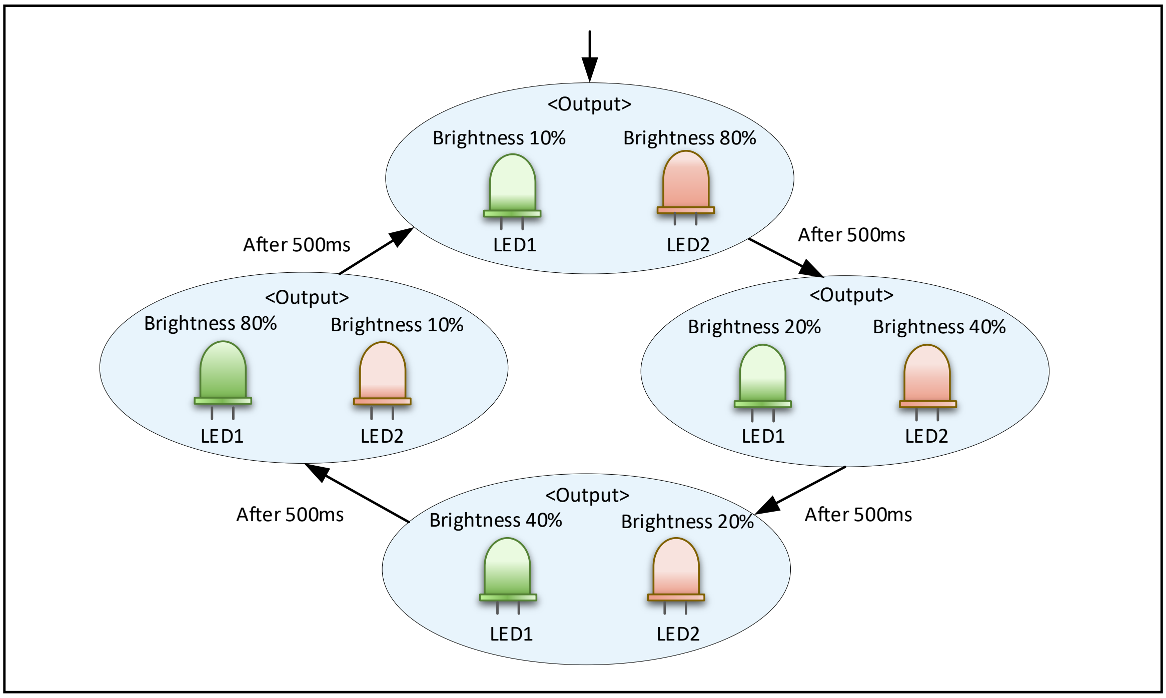


Table 3-2 Relationship between PWM output duty ratio and LED brightness

Duty ratio	Brightness
10%	10%
20%	20%
40%	40%
80%	80%

3.1.2 Operation Overview

In this sample code, 16-bit timer KB30 (TKB30) is used with the standalone mode (period controlled by the TKBCRn0 register), PWM signals are output from P12/TKBO00 and P13/TKBO01.

The PWM pulse period of TKB30 is 2ms, and the interrupts (INTTKB30) that occur in each period are counted 250 times. Start the FAA from the CPU every 500ms and change the duty ratio of PWM output with FAA.

1. [CPU program] Store the initial values of the TKBCR01 register and the TKBCR03 register in variables for checking the duty value.
2. [CPU program] Enable the TKB30 operation.
3. [CPU program] Set SFR access of the TKB30 to FAA bus.
4. [CPU program] Wait until the TKB30 interrupt occurs 250 times (500ms).
5. [CPU program] After the TKB30 starts the operation, the TKB30 interrupt occurs every 2ms.
6. [CPU program] Count the number of interrupt occurrences in the TKB30 interrupt (INTTKB30).
7. [CPU program] When TKB30 interrupt (INTTKB30) occurs 250 times (500ms), clock supply to the FAA is enabled and FAA operation is enabled.
8. [CPU program] Set the FAA stack pointer and the start address of the FAA program and start FAA operation. Then wait until the FAA program completes.
9. [FAA program] Update the compare register (TKBCR01) and change the duty ratio of TKBO00 output. And update the compare register (TKBCR03) and change the duty ratio of TKBO01 output. Every 500ms, the duty ratio of the TKBO00 output is updated by double in the order of 10% → 20% → 40% → 80%, and after the duty ratio reaches 80%, it is set to 10% again. The duty ratio of the TKBO01 output is updated by 1/2 in the order of 80% → 40% → 20% → 10%, and after the duty ratio is 10%, it is set to 80% again.
10. [FAA program] Store the updated duty ratio (values of the TKBCR01 register and the TKBCR03 register) in global variables and the FAA stops operating.
11. [CPU program] When FAA program execution is completed, clock supply to the FAA is stopped and FAA operation is disabled.
12. [CPU program] Store the updated duty ratio (values of the TKBCR01 register and the TKBCR03 register) in variables for duty value confirmation.
13. [CPU program] Return to step 4 and wait for TKB30 interrupts (INTTKB30) to occur 250 times (500ms) again.

3.2 Operation Confirmation Conditions

Table 3-3 Operation Confirmation Conditions

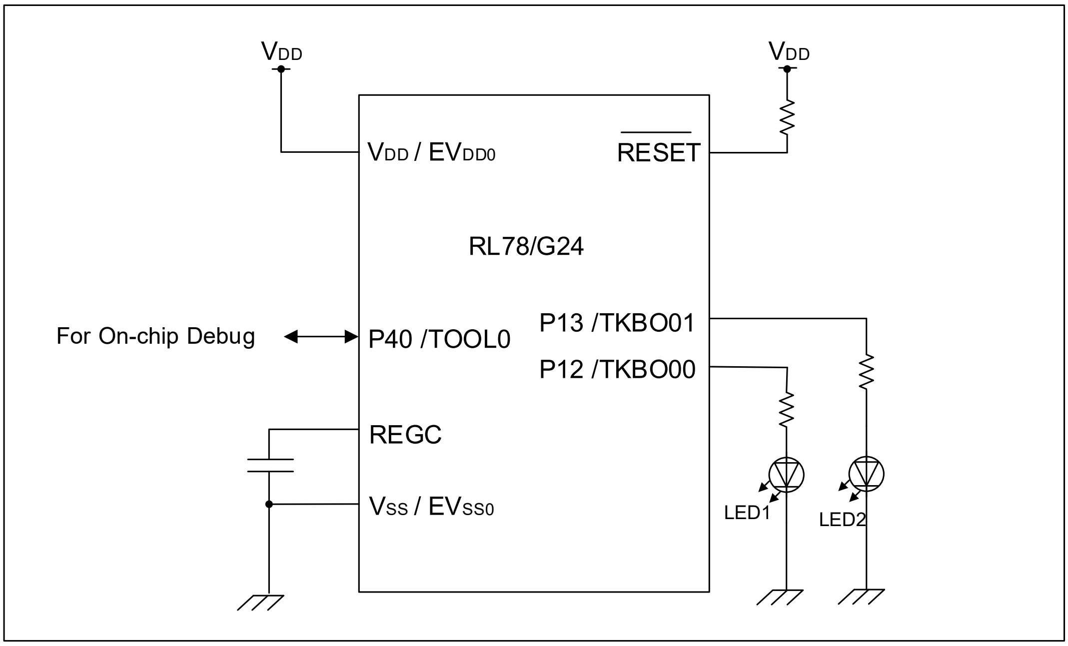
Item	Description
MCU	RL78/G24 (R7F101GLG)
Operating frequency	・ High-Speed On-Chip Oscillator Clock: 32MHz ・ CPU/Peripheral Hardware Clock: 32MHz
Operating voltage	・ 3.3V (Can operate between 2.7V to 5.5V) ・ LVD0 Operation (VLVD0): Reset Mode Rising edge = 2.97V Falling edge = 2.91V
Integrated development environment (CS+)	CS+ for CC V8.10.00 Manufactured by Renesas Electronics
C compiler (CS+)	CC-RL V1.12.01 Manufactured by Renesas Electronics
Smart Configurator (SC)	Manufactured by Renesas Electronics V1.7.0
Board Support Package (BSP)	Manufactured by Renesas Electronics V1.60
Emulator	E2 Emulator Lite
Board	RL78/G24 Fast Prototyping Board (RTK7RLG240C00000BJ)

3.3 Hardware Description

3.3.1 Example of Hardware Configuration

The example of the hardware configuration used in this sample code is shown below.

Figure 3-2 Example of Hardware Configuration



Note 1. This simplified circuit diagram was created to show an overview of connections only. When actually designing your circuit, make sure the design includes appropriate pin handling and meets electrical characteristic requirements (connect each input-only port to VDD or VSS through a resistor).

Note 2. Connect any pins whose name begins with EVSS to VSS, and any pins whose name begins with EVDD to VDD, respectively.

Note 3. VDD must not be lower than the reset release voltage (VLVD0) that is specified for the LVD0.

3.3.2 List of Used Pins

Table 3-1 shows the pins used and their function.

Table 3-4 Pins Used and their Functions

Pin name	I/O	Function
P12 / TKBO00	Output	PWM output (lighting control for LED1)
P13 / TKBO01	Output	PWM output (lighting control for LED2)

Caution. In this application note, only the used pins are processed. When actually designing your circuit, make sure the design includes sufficient pin processing and meets electrical characteristic requirements.

3.4 Software Description

3.4.1 Smart Configurator Setting

The Smart Configurator (SC) settings in this sample code are shown below. The items and settings in each SC settings table are explained using the description on the settings screen.

3.4.1.1 Clock

The clock settings used in this sample code are shown below.

Operation mode: High-speed main mode 2.7(V)~5.5(V)

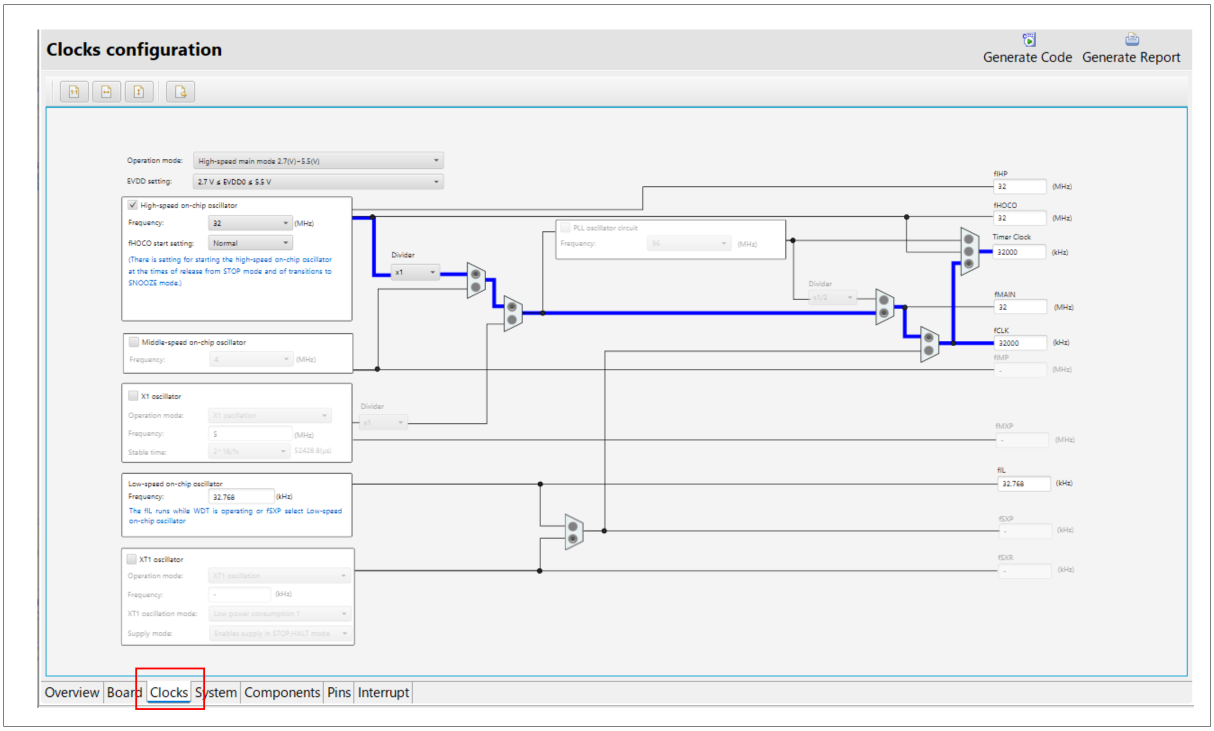
EVDD setting: 2.7 V ≤ EVDD0 ≤ 5.5V

High-speed on-chip oscillator: 32MHz

fCLK: 32000kHz

Timer Clock: 32000kHz

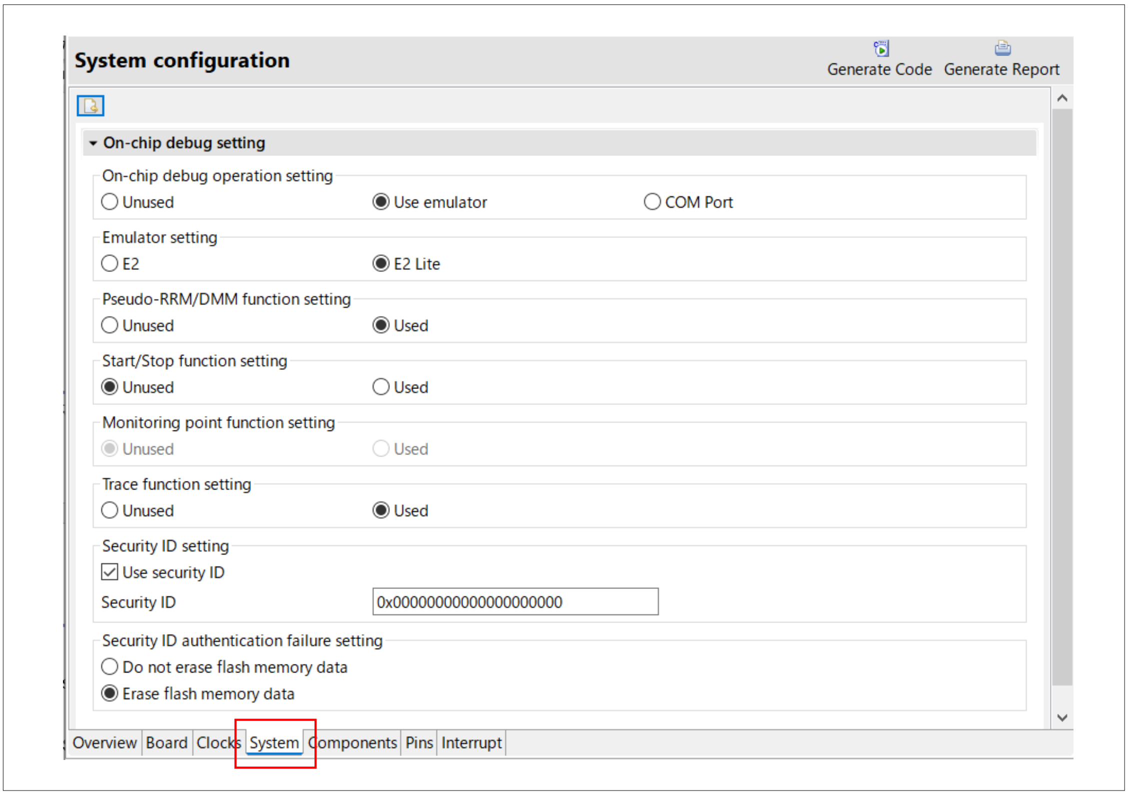
Figure 3-3 Clock Settings



3.4.1.2 System

The system settings used in this sample code are shown below.

Figure 3-4 System Settings



3.4.1.3 Component

The component settings used in this sample code are shown below.

Table 3-5 Component settings (LVD0)

Item	Description
Component	Voltage Detector
Configuration name	Config_LVD0
Resource	LVD0

Figure 3-5 LVD0 Settings

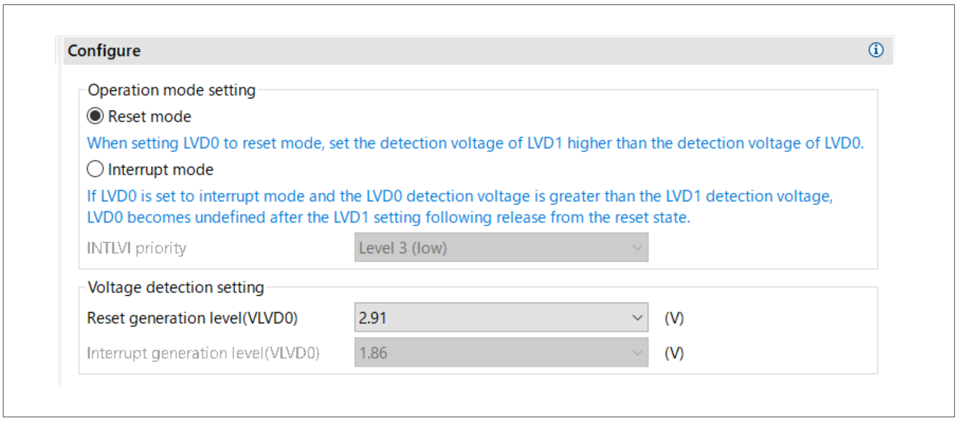


Table 3-6 Component settings (TKB30)

Item	Description
Component	PWM Output
Operation	Standalone mode (Period controlled by the TKBCRn0 register)
Configuration name	Config_TKB0
Resource	TKB0

Figure 3-6 TKB30 Settings

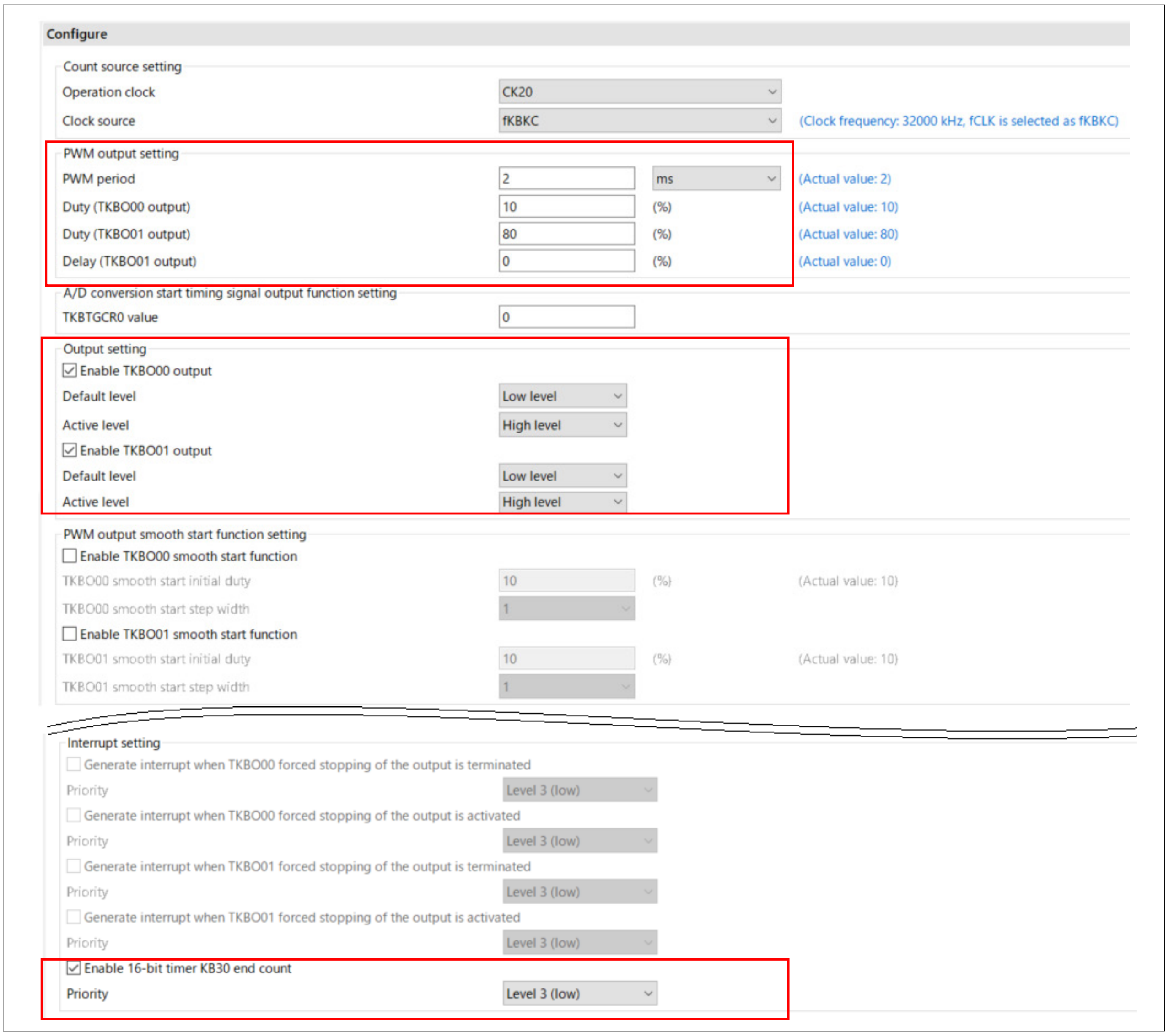
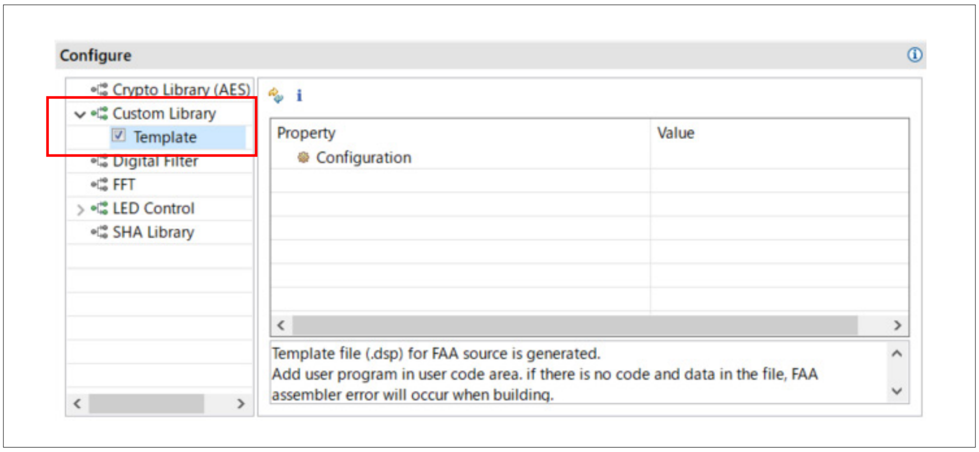


Table 3-7 Component settings (FAA)

Item	Description
Component	Flexible Application Accelerator
Configuration name	Config_FAA

Figure 3-7 FAA Settings



Remark. If any FAA library is not displayed after the sample project is opened, refer to step 11 in 2.3.1 Adding FAA Component to download FAA libraries.

3.4.2 Folder Structure

Table 3-8 shows the structure of the source files/header files used in the sample project.

Table 3-8 Folder Structure

Folder, File name	Description	Generated by SC
\sample_project<DIR>	Sample project folder	
main.c	Sample source file	
sample_project.py	(File for loading sample script)	
sample_script.py	(Sample script)	
\src<DIR>	Program storage folder	√
\smc_gen<DIR>	Smart Configurator generated folder	√
\Config_FAA<DIR>	FAA program storage folder	√
Config_FAA_common.c	Common FAA module source file	√
Config_FAA_common.h	Common FAA module header file	√
Config_FAA_common.inc	Include file for FAA assembly source file	√
Config_FAA_src.dsp	FAA assembly source file	√ Note 1
\Config_TKB0<DIR>	TKB30 program storage folder	√
Config_TKB0.c	TKB30 source file	√
Config_TKB0.h	TKB30 header file	√
Config_TKB0_user.c	TKB30interrupt source file	√ Note 2
¥general<DIR>	Initialization and common program storage folder	√
¥r_bsp<DIR>	BSP program storage folder	√
¥r_config<DIR>	Configuration header storage folder	√

Note. “<DIR>” indicates a directory.

Note 1. This sample project uses the Custom Library of FAA library. Therefore, file content is only a template and no code right after the file is generated. Sample code has been added.

Note 2. Sample code has been added in the user code area of SC.

3.4.3 Option Byte Settings

Table 3-9 shows the option byte settings.

Table 3-9 Option Byte Settings

Address	Setting value	Description
000C0H/040C0H	1110 1111B (EFH)	Watchdog Timer stopped operation (Count stops after reset release)
000C1H/040C1H	1111 1011B (FBH)	LVD0 reset mode. Detection voltage: Rising 2.97V / Falling 2.91V
000C2H/040C2H	1110 1000B (E8H)	lash operation mode: High-speed main mode. High-speed on-chip oscillator frequency: 32MHz
000C3H/040C3H	1000 0100B (84H)	On-chip debug operation enabled

3.4.4 List of Constants

Table 3-10 and Table 3-11 show constants used in the sample code.

Table 3-10 Constants (CPU program)

Constant name	Value	Description	Function that uses the constant
FAA_BUS_ACCESS	0200H	Enable to access TKB30 register from FAA. (ADBSEL setting value)	main

Table 3-11 Constans (FAA program)

Constant name	Value	Description
_C_TKBO00_DUTY_INIT	1900H	Initial duty ratio for TKBO00 output (TKBCR01 setting value)
_C_TKBO01_DUTY_INIT	C800H	Initial duty ratio for TKBO01 output (TKBCR03 setting value)
_C_TKBTRG_TKBRDT_REQ	1H	Batch overwrite request of TKB30 compare register (TKBRDT0 setting value)

3.4.5 List of Variables

Table 3-12 and Table 3-13 show variables used in the sample code.

Table 3-12 Variables (CPU program)

Type	Variable name	Description	Function that uses the variable
uint32_t	g_work_tkbo00	Variable to check the current duty ratio for TKBO00 output (Value of TKBCR01)	main
uint32_t	g_work_tkbo01	Variable to check the current duty ratio for TKBO01 output (Value of TKBCR03)	main
uint8_t	g_tkb_interrupt_flag	500ms elapsed flag	r_Config_TKB0_end _count_interrupt

Table 3-13 Variables (FAA program)

Size	Variable name	Description
4 bytes	_V_TKBO00_DUTY	Storage the updated duty ratio for TKBO00 output (TKBCR01 setting value)
4 bytes	_V_TKBO01_DUTY	Storage the updated duty ratio for TKBO01 output (TKBCR03 setting value)

3.4.6 List of Functions

Table 3-14 and Table 3-15show functions and processing used in the sample code. However, functions generated by the Smart Configurator that have not been modified are excluded.

Table 3-14 Functions (CPU program)

Function name	Description	Source file
main	main process	main.c
r_Config_TKB0_end_count_interrupt	TKB30 interrupt processing (Count the number of INTTKB00 occurrences)	Config_TKB0_user.c

Table 3-15 Processing (FAA program)

Label name	Description	Source file
_P_TKB_PWM	Change the duty ratio of TKBO00 and TKBO01 output	Config_FAA_src.dsp

3.4.7 Function Specification

The function specifications of the sample code are shown below.

CPU program

[FUnction name] main()

Outline	main process
Header	r_smc_entry.h、Config_TKB0.h
Declaration	void main(void)
Description	Start operation of the Timer TKB30, and start operation of the FAA every 500ms.
Argument	-
Return value	-

CPU program

[Function name] r_Config_TKB0_end_count_interrupt()

Outline	Timer TKB30 interrupt processing
Header	r_cg_macrodriver.h、r_cg_userdefine.h、Config_TKB0.h
Declaration	static void __near r_Config_TKB0_end_count_interrupt(void)
Description	Count INTTMKB30 occurrences and set the 500ms elapsed flag every 250 interrupts (500ms elapsed).
Argument	-
Return value	-

FAA program

[Label name] _P_TKB_PWM

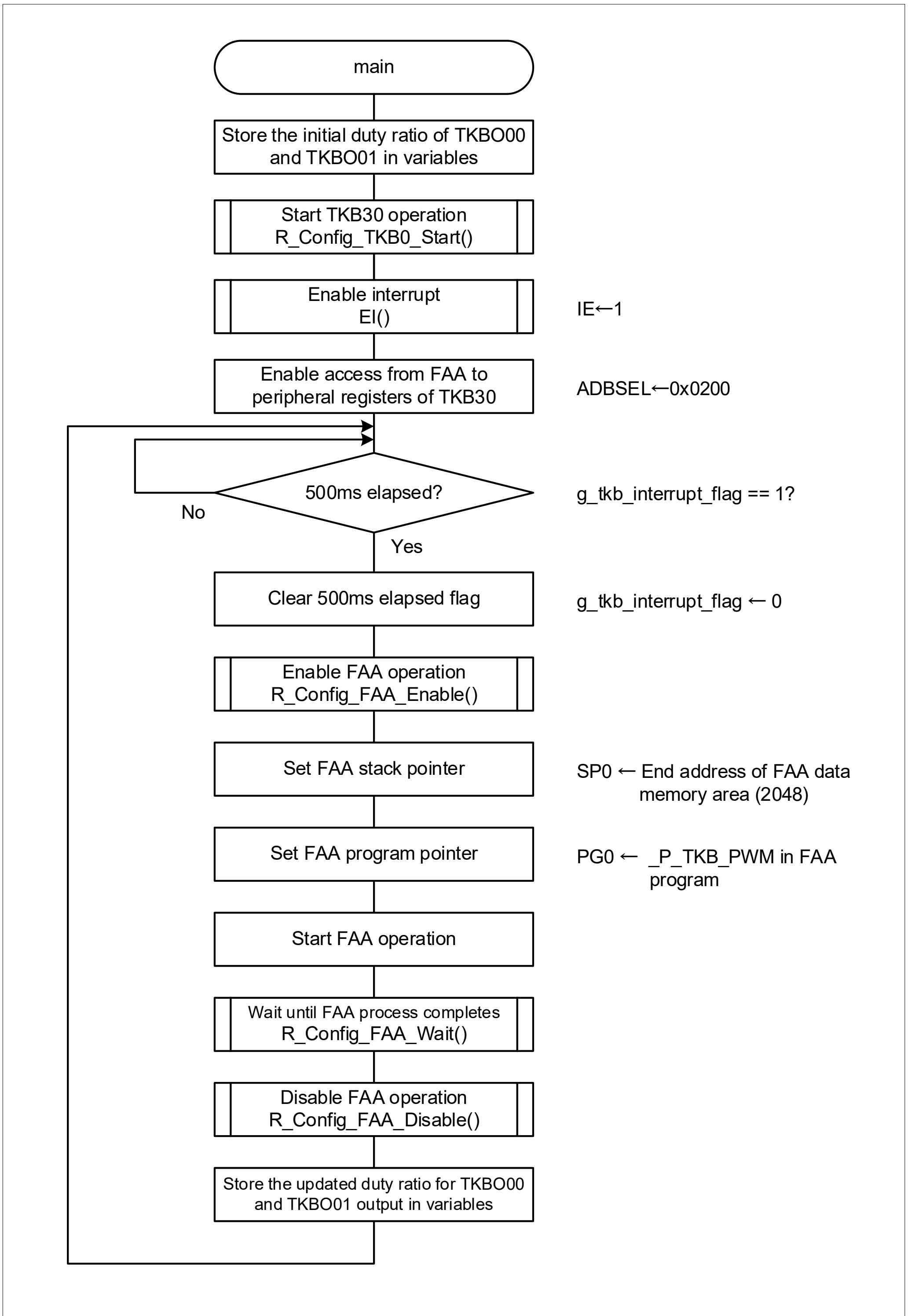
Outline	Change processing of the duty ratio for TKBO00 and TKBO01 output
Header	Config_FAA_common.inc
Declaration	-
Description	Change the duty ratio for TKBO00 and TKBO01 output.
Argument	-
Return value	-

3.4.8 Flowchart

3.4.8.1 Main Process

Figure 3-8 shows the flowchart for the main process.

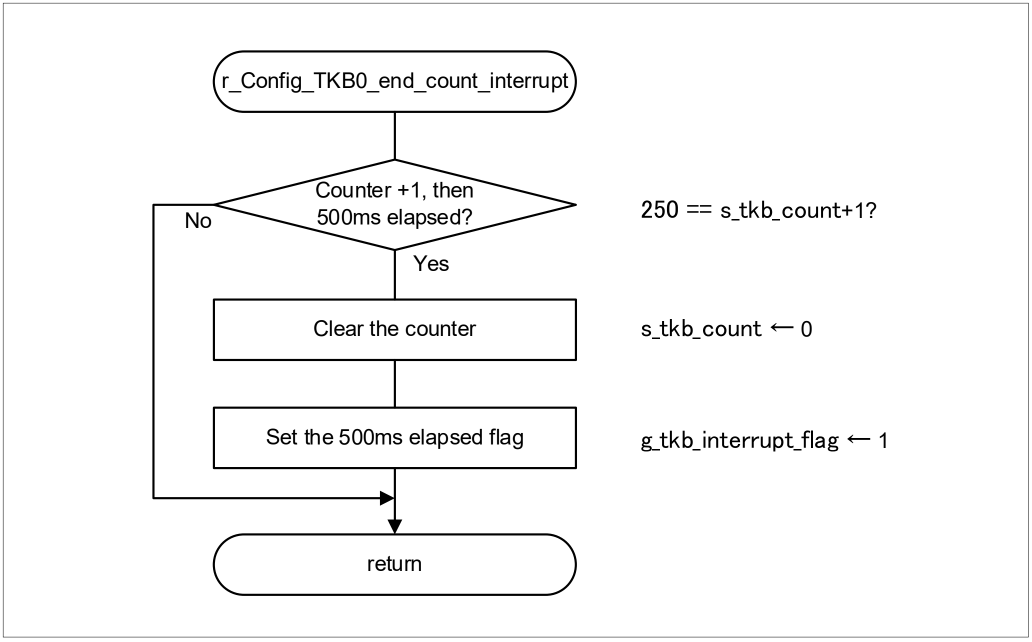
Figure 3-8 Main process



3.4.8.2 r_Config_TKB0_end_count_interrupt Function

Figure 3-9 shows the flowchart of the r_Config_TKB0_end_count_interrupt function.

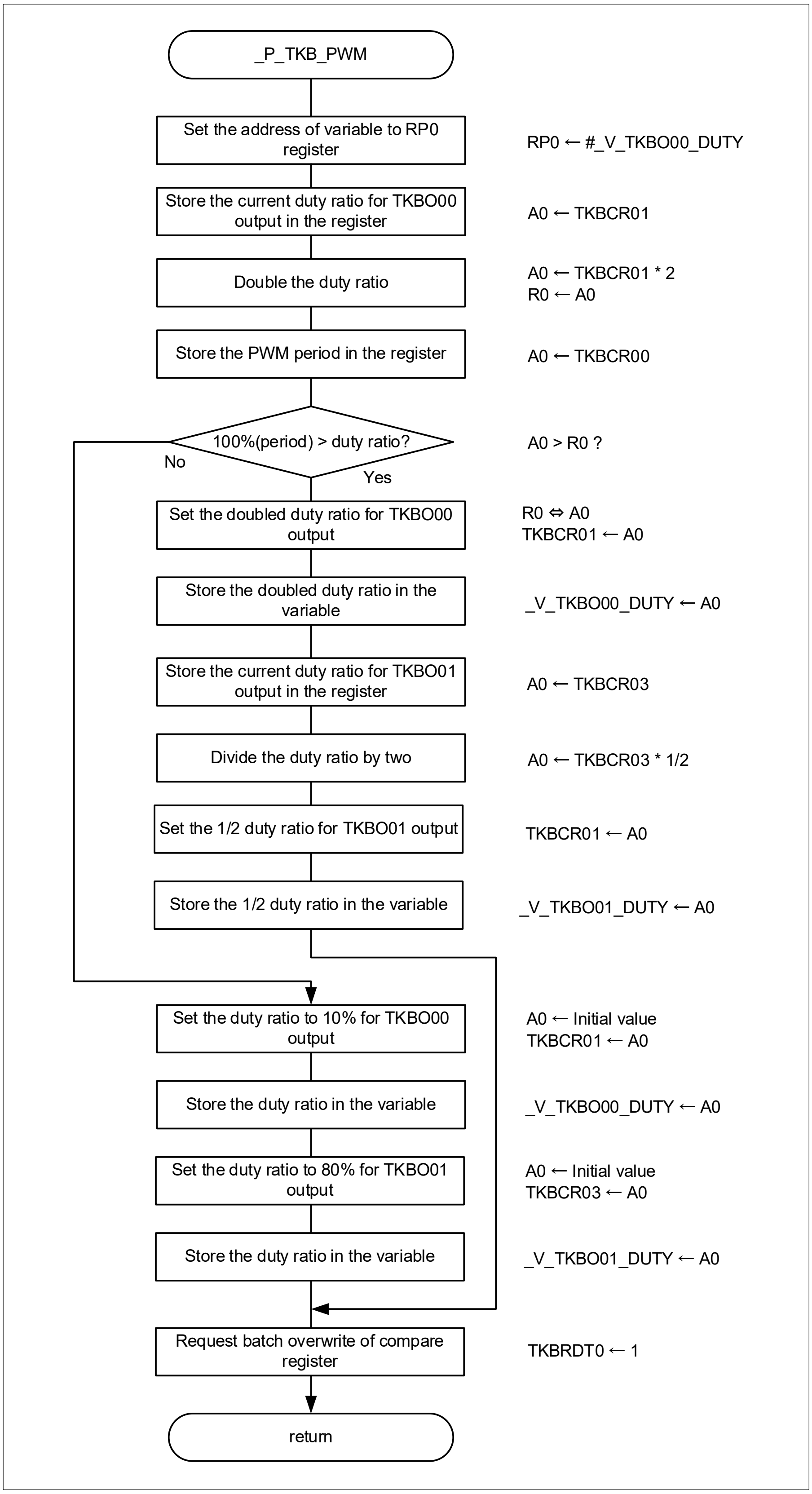
Figure 3-9 r_Config_TKB0_end_count_interrupt function



3.4.8.3 FAA Processing

Figure 3-10 shows the flowchart of the r_Config_TKB0_end_count_interrupt function.

Figure 3-10 FAA processing



3.5 Sample Script Specification

This sample project includes the sample script that manipulates the value of the address bus selection register (ADBSEL) to display peripheral function SFRs on the [SFR] panel in CS+ when debugging an FAA program. (sample_script.py in the sample project)

CS+ can be controlled by using a script language IronPython (Python that runs on .NET Framework) and the CS+ Python function. For details about the functions, see the help or documentation of CS+ for CC.

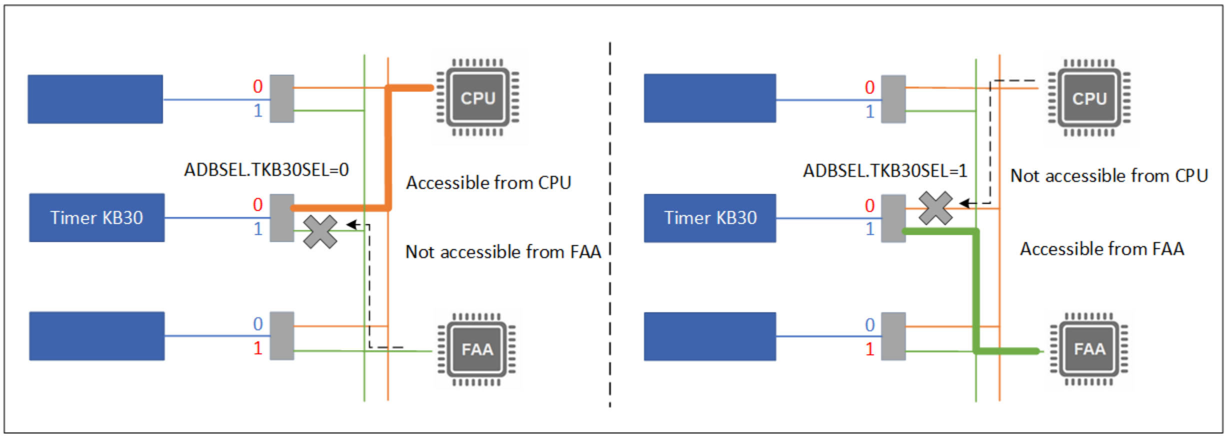
3.5.1 SFR Display Overview

For some peripheral functions of RL78/G24, access from the CPU or from the FAA can be selected with the address bus selection register (ADBSEL). For the address bus select register (ADBSEL), refer to RL78/G24 User's Manual: Hardware (R01UH0961).

The debugger reads or writes peripheral function SFR values through bus access from the CPU. It cannot access the peripheral function SFRs for which bus access from the FAA is selected with the address bus select register (ADBSEL). Therefore, reading from or writing to these peripheral function SFRs cannot be performed on the debugger's [SFR] panel.

To enable read and write on the debugger's [SFR] panel for the peripheral function SFRs for which bus access from the FAA is selected when the debug target is FAA, use the script to manipulate the ADBSEL register value.

Figure 3-11 Image diagram of address bus select function



3.5.2 Operation Overview

When the debug target is FAA, after the FAA program is stopped by using the stop button, step execution, or breakpoint, the script assigns the XORed value to the current setting of the ADBSEL register. This temporality permits access from the CPU (the debugger) for the peripheral function SFRs for which access from the FAA is selected. In addition, before the FAA program is executed by using the execution button or step execution, the script assigns the original setting to the ADBSEL register to return the setting to permit access from the FAA.

This allows access from the FAA to the relevant SFRs during execution of the FAA program and, after the FAA program stops, allows the debugger to access the relevant SFRs and read or write values on the [SFR] panel.

Figure 3-12 Image of sample script

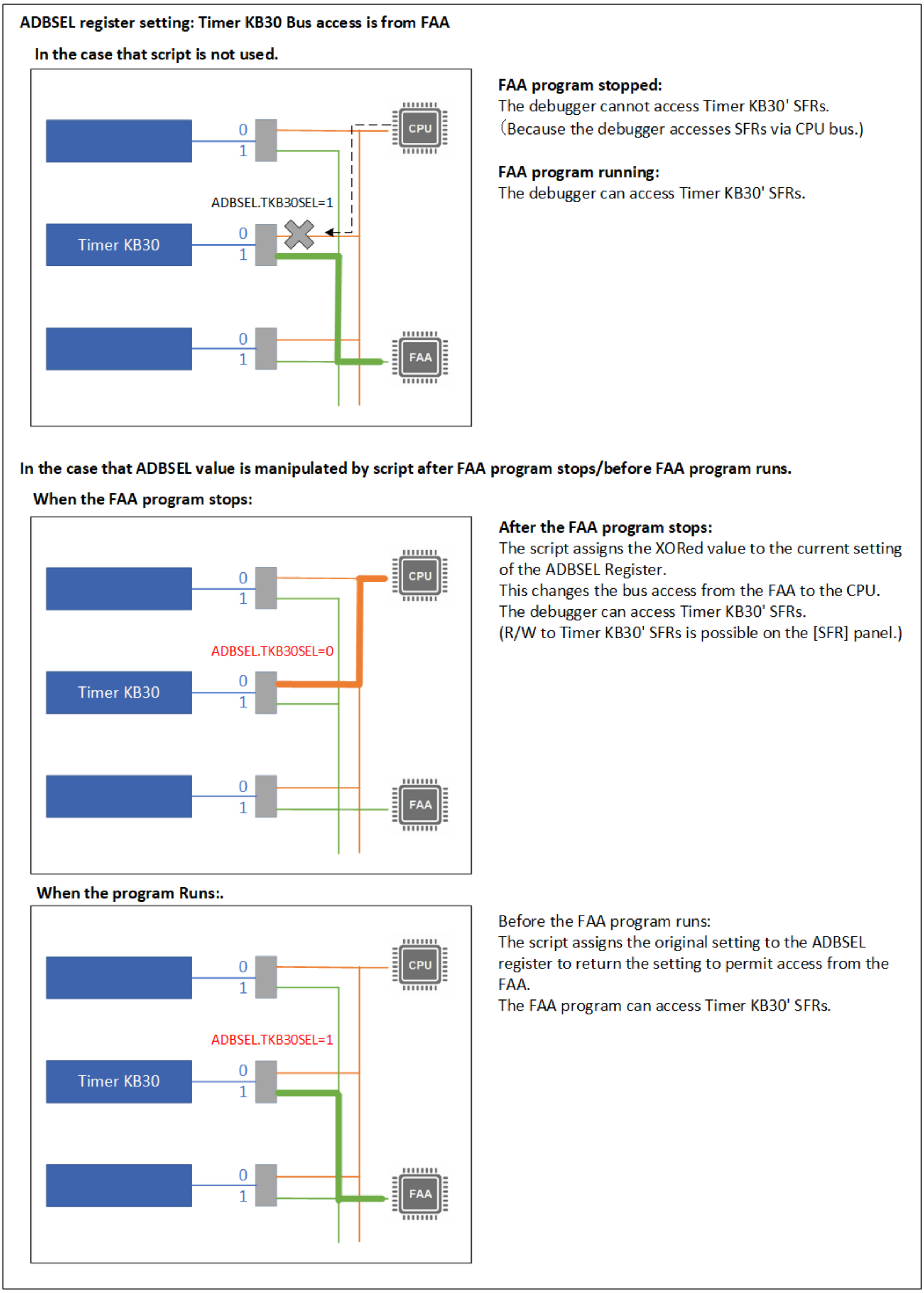
Sample script file (.py)

```
Variable initialization

def BeforeCpuRun():
   processing

def AfterCpuStop():
   processing

def AfterCpuReset():
   processing
```

- In addition to the functions and control statements supported by the IronPython language, use additional Python functions for CS+ to create operations to control CS+.
- Register CS+ hook functions to be executed before the program starts running and after it stops running.
- Write the process to change ADBSEL register values in each hook function.

The script file for this sample project is sample_script.py.

Figure 3-13 Image diagram of changing ADBSEL register values by script



3.5.3 List of Functions

(1) Hook Functions

The sample script uses the CS+ hook functions to change the ADBSEL register value within a hook function that is called when an event occurs. Table 3-16 lists the hook functions used in the script and provides an overview of processing.

Table 3-16 Hook functions used in the sample script and processing overview

Hook function name	Event	overview
AfterCpuReset	After CPU reset	Initialize variables used in the sample script.
BeforeCpuRun	Before execute	Write the original value that CPU sets to ADBSEL register to the ADBSEL register.
AfterCpuStop	After break	Write the XORed value of the original value to the ADBSEL register.

(2) CS+ Python Functions

Table 3-17 lists the CS+ Python functions used in the script and provides an overview of processing.

Table 3-17 CS+ Python functions used in the sample script and processing overview

Function name	Overview
debugger.DebugTool.GetType	This function displays information about the debug tool.
debugger.Watch.SetValue	This function sets a variable (SFR) value.
debugger.Watch.GetValue	This function refers to a variable (SFR) value.

3.5.4 List of Variables

(1) CS+ Python Property

Table 3-18 lists the CS+ Python property used in the script and provides an overview of processing.

Table 3-18 CS+ Python property used in the sample script and processing overview

Property name	Overview
debugger.ProcessorElement	This property sets or refers to the PE of multiple cores with the name. [Value] 1: CPU 2: FAA

(2) Others

Table 3-19 lists the variables other than CS+ Python property used in the script and provides an overview of processing.

Table 3-19 Other variables used in the sample script and processing overview

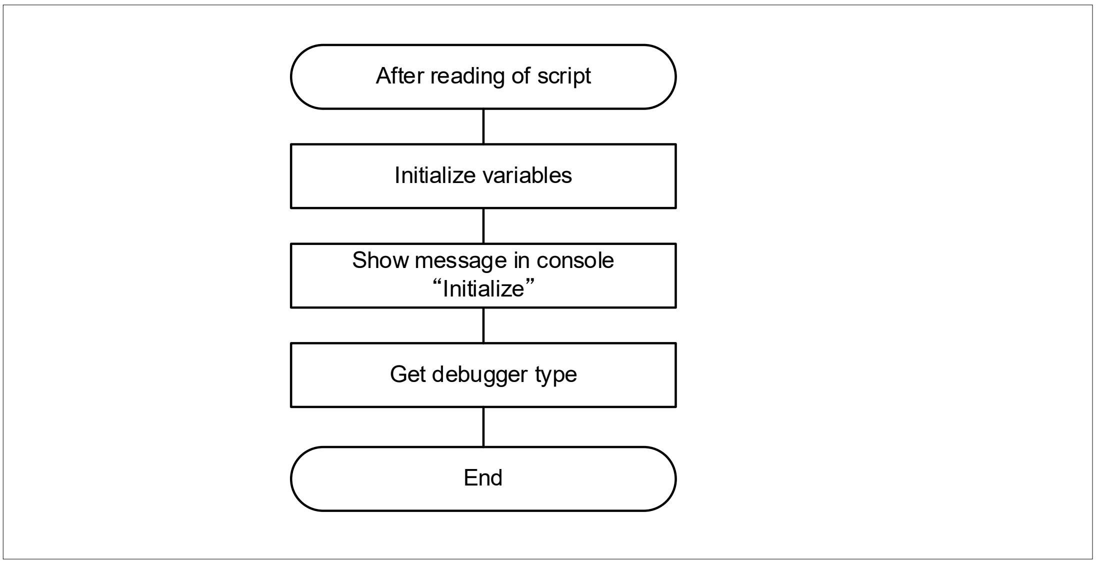
Variable name	Overview
FaaStatus	The FAA program operation status (Set when Go/Stop button is pressed). [Value] RUNNING: FAA program is running STOPPING: FAA program is stopping
previousPe	The debug target just before pressing the Go/Stop button. [Value] 1: CPU 2: FAA
adbsel_value_cpu	ADBSEL register's value set by the CPU program
number_of_command	The number of times the hook function was executed.

3.5.5 Flowchart

(1) Initialization Process

Figure 3-14 shows the flowchart of the initialization process that is executed after loading the sample script (.py).

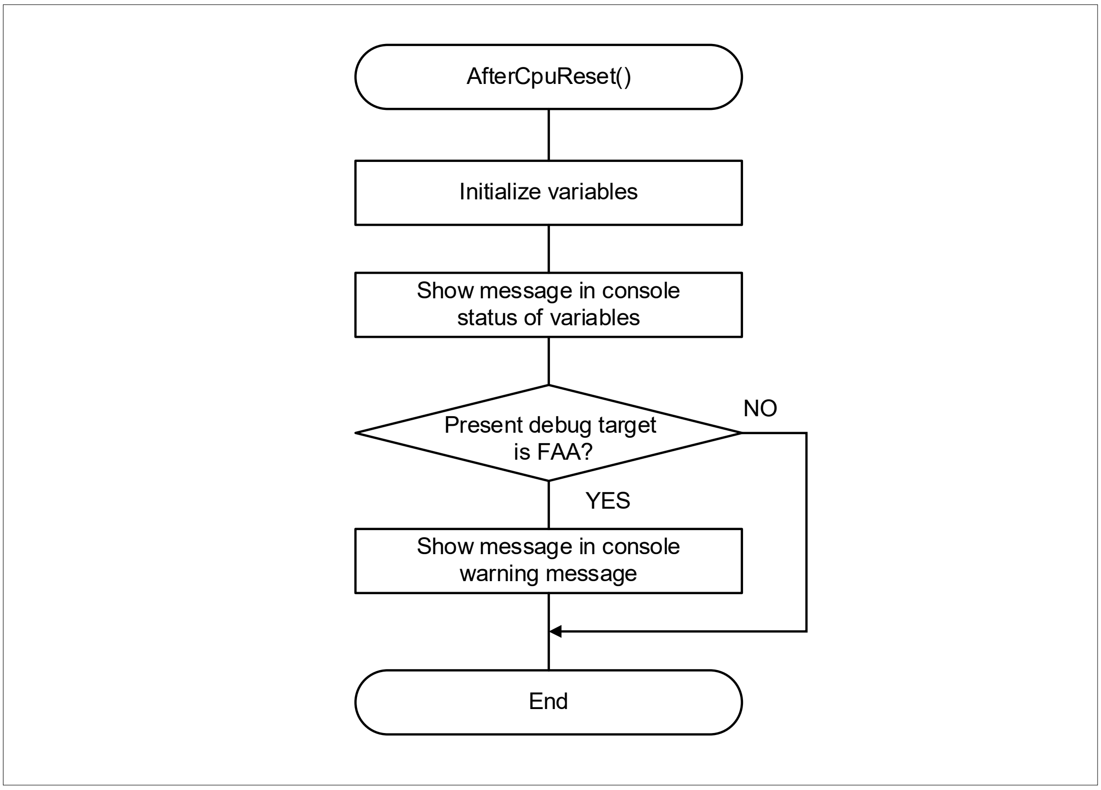
Figure 3-14 Initialization process



(2) AfterCpuReset Process

Figure 3-15 shows the flowchart of the AfterCpuReset process.

Figure 3-15 AfterCpuReset process



(3) BeforeCpuRun Process

Figure 3-16 and Figure 3-17 show the flowchart of the BeforeCpuRun process.

Figure 3-16 BeforeCpuRun process (1/2)

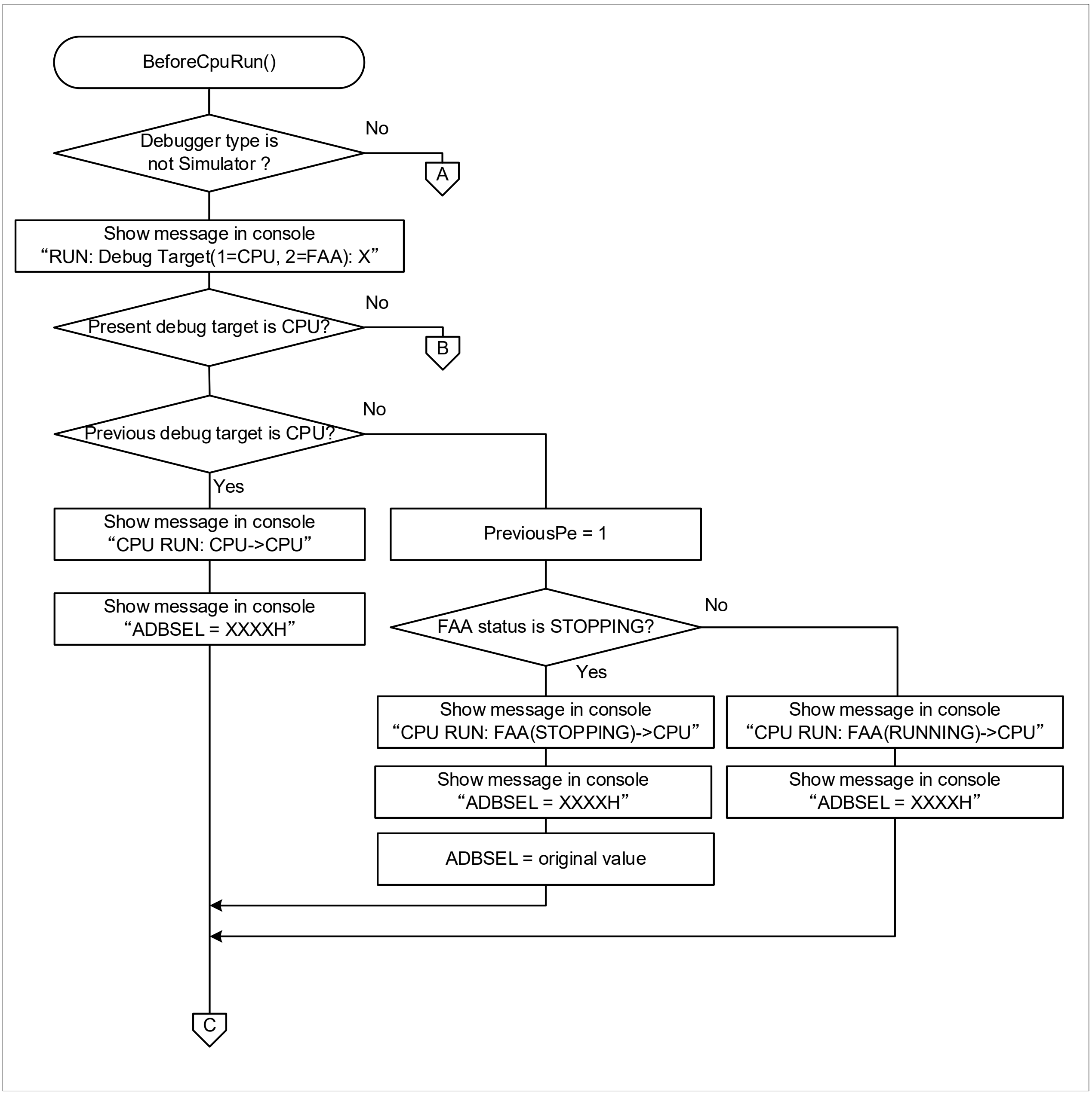
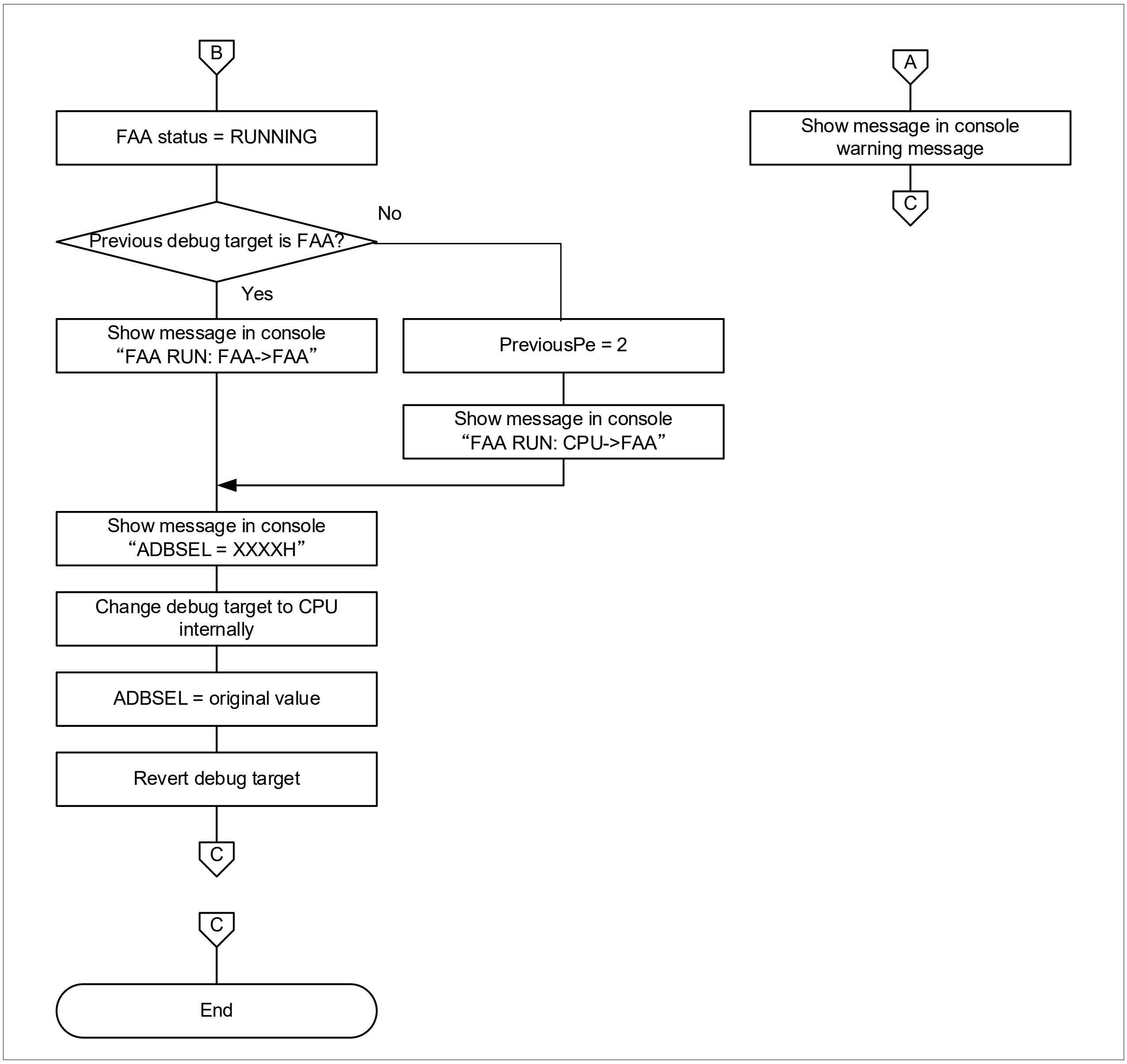


Figure 3-17 BeforeCpuRun process (2/2)



(4) AfterCpuStop Process

Figure 3-18 and Figure 3-19 show the flowchart of the AfterCpuStop process.

Figure 3-18 AfterCpuStop process (1/2)

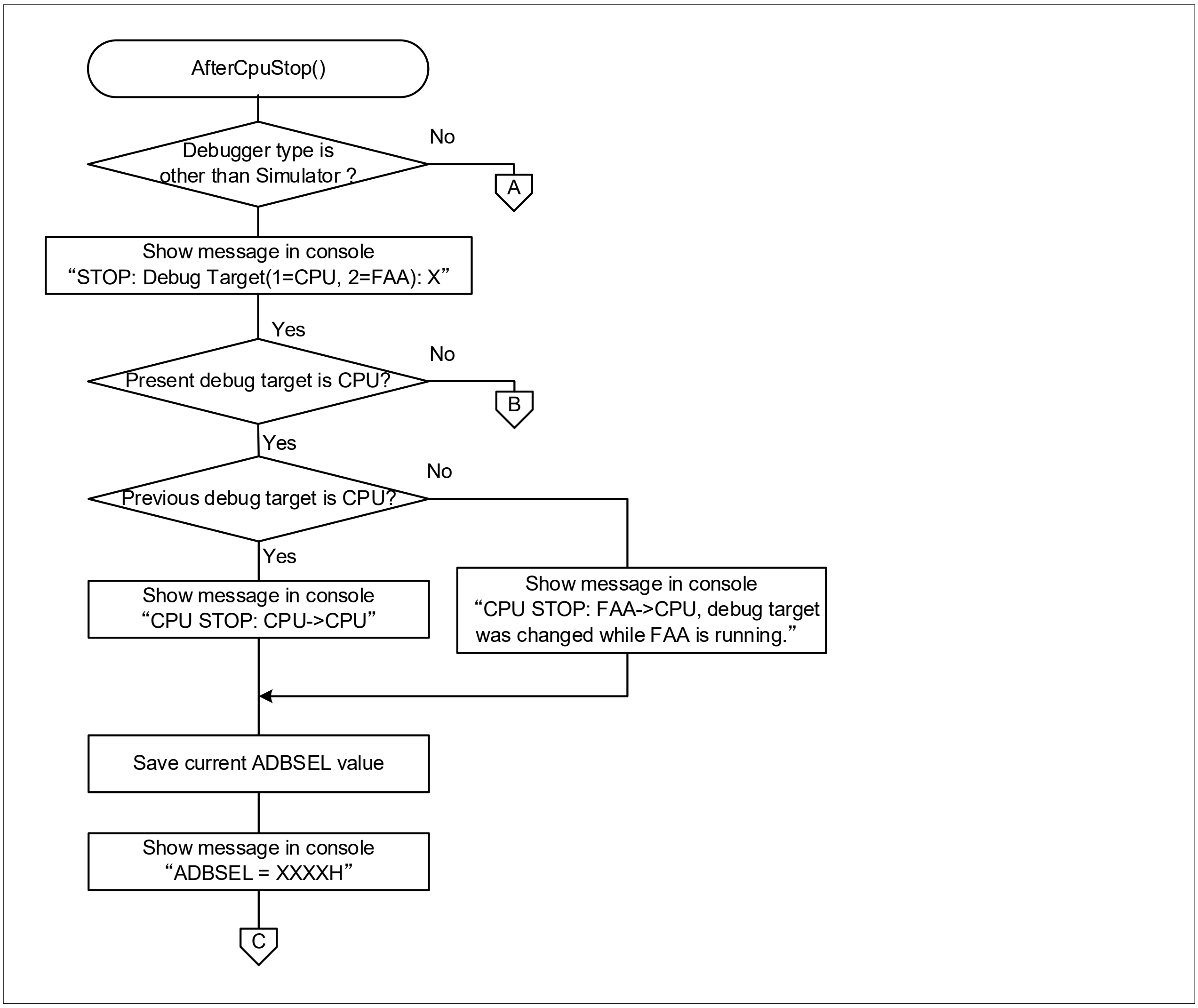
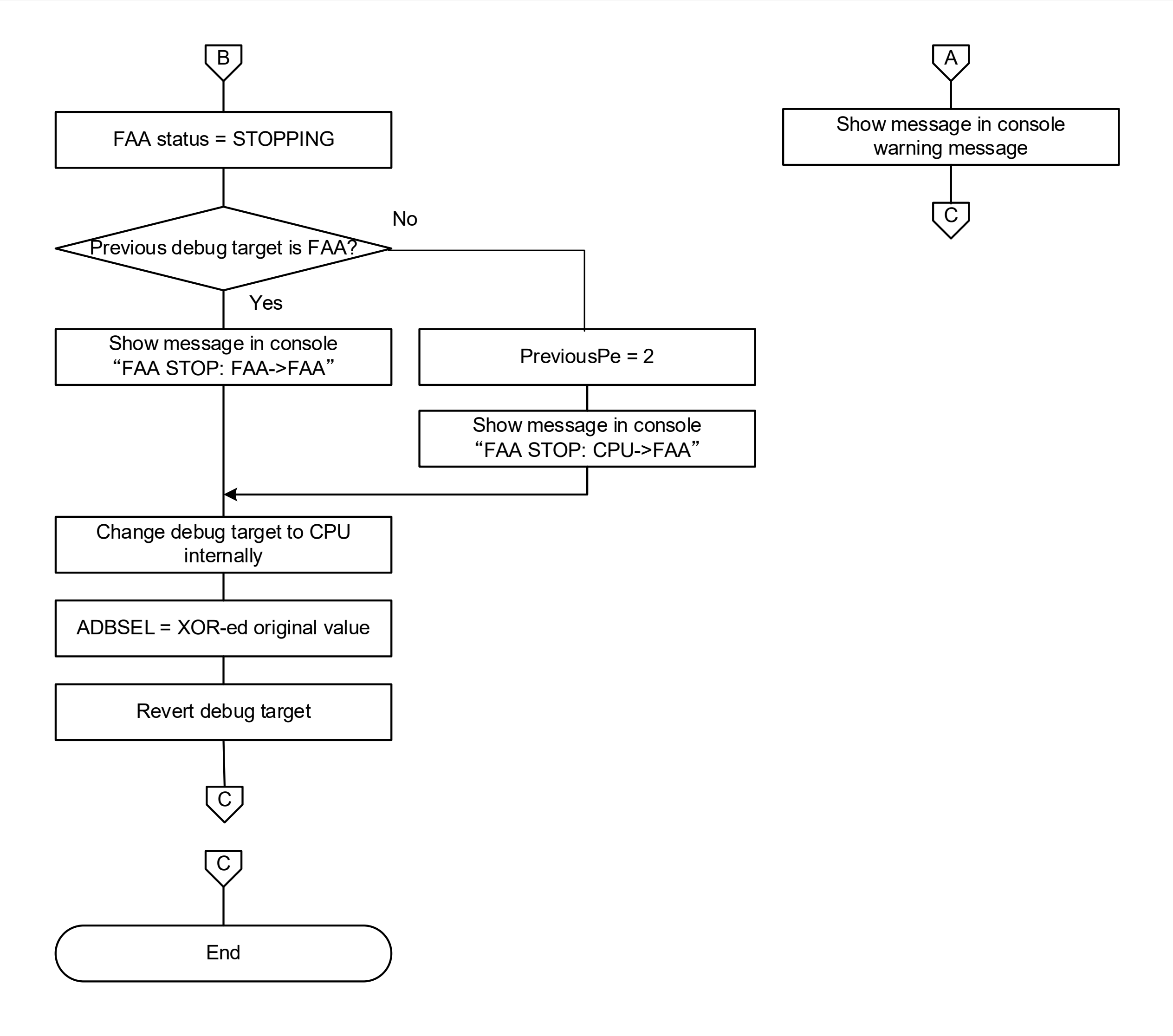


Figure 3-19 AfterCpuStop process (2/2)



3.5.6 Script Execution

There are several ways to execute a script and register hook functions.

- When loading the project file (*project-file-name*.py)

  If there is a file in the same folder as the project file, and with the same name as the project file but with the "py" extension, then that file is executed automatically when the project file is loaded. The active project will be processed.

- When downloading the download file (*download-file-name*.py)

  If there is a file in the same folder as the download file, and with the same name as the download file but with the "py" extension, then that file is executed automatically after downloading.

- Execute in the CS+ [Python Console] panel

  Execute the ".py" file by the CS+ Python function: "Source".

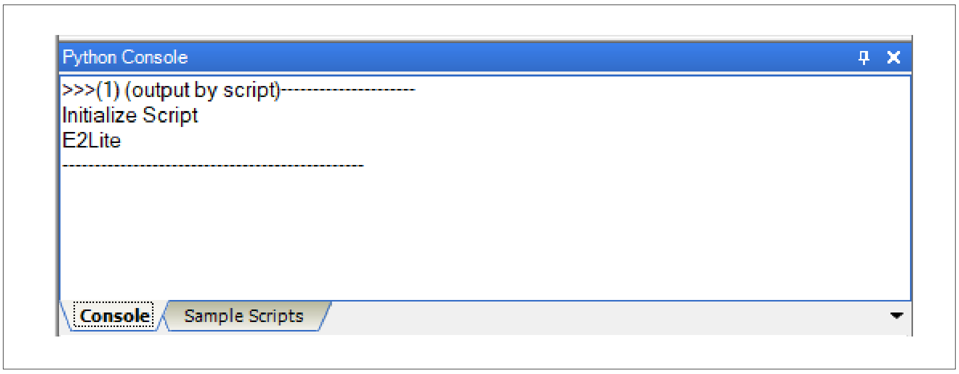
In this sample project, it is executed when the project file is loaded.

The hook functions are declared in the sample_script.py. Also, there is the sample_project.py with the same name as the sample project "sample_project.mtpj", and the sample_project.py hooks the sample_script.py and registers hook functions declared in the sample_script.py. The sample_project.py is executed automatically when the project file is loaded.

Procedure:

1. Load the sample_project.mtpj to CS+.
2. Select the CS+ [View] menu -> [Python Console].
3. In the [Python Console] panel, confirm that the script executes.

Figure 3-20 Python Console



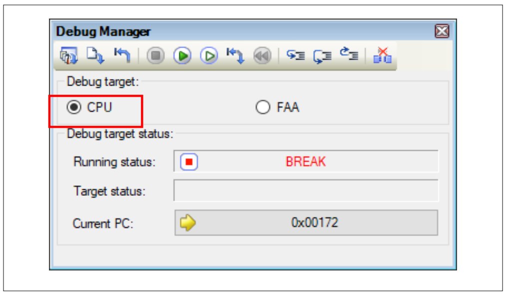
3.5.7 Basic debug operations

This section explains the basic operations of debugging a FAA program using sample code and sample scripts.

Procedure:

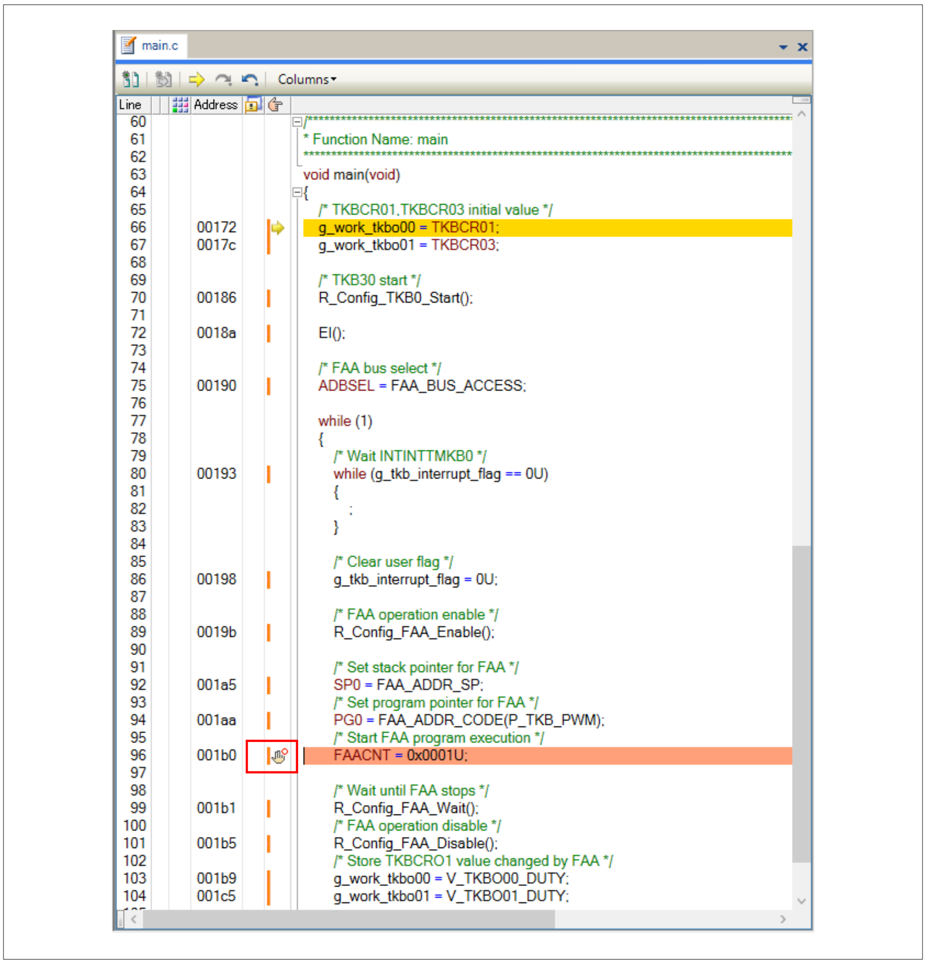
1. Connect the RL78/G24 Fast Prototyping Board (with the emulator or via COM port) to the PC.
2. Select the [Debug] menu -> [Rebuild & Download].
3. Select the [View] menu -> [Debug Manager] and select the CPU as debug target.

Figure 3-21 Debug Manager



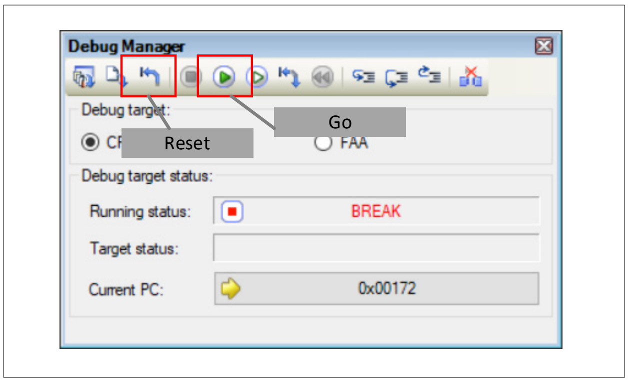
4. Open the main.c. Click the main area of "FAACNT = 0x0001U;" to set the breakpoint (Software break).

Figure 3-22 main.c (Debug target: CPU)



5. Click the reset button and then click the execution button in the [Debug Manager].

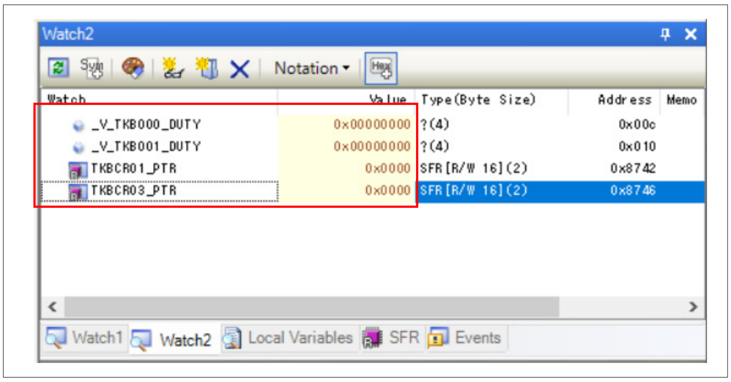
Figure 3-23 Debug Manager



6. After the program stopped by the breakpoint, change the debug target to the FAA on the [Debug Manager]. To debug FAA programs, the FAA must be enabled (FAAEN=1, ENB=1). In the sample code, "R_Config_FAA_Enable()" enables the FAA. Therefore, the FAA has been enabled at the breakpoint.

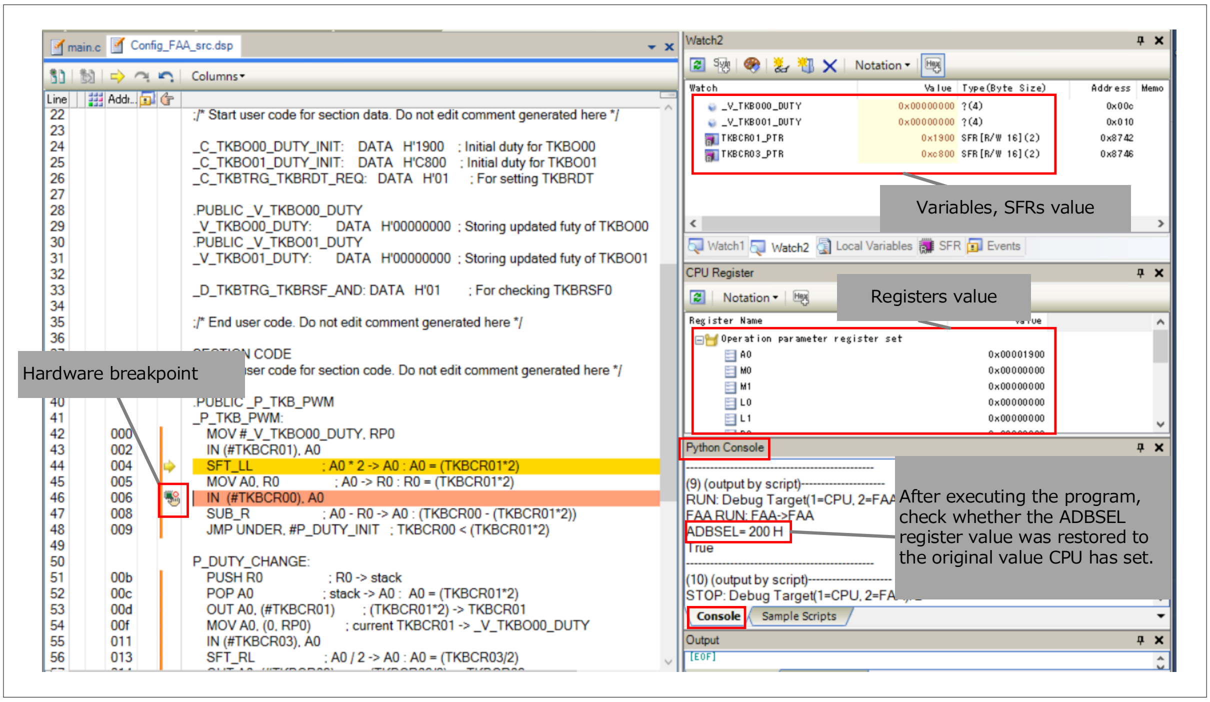
7. Register variables (_V_TKBO00_DUTY, _V_TKBO01_DUTY) and SFRs (TKBCR01_PTR, TKBCR03_PTR PPP, RRR) whose values are changed in the FAA program to the [Watch] panel.
   - After registering the variable, change it to 4-byte notation. (Refer to 2.6.6 Symbol (Label)
   - SFRs can also be displayed in the [SFR] panel.
   - The [Watch] panel to display the variables must be set to the FAA data space.(Refer to 2.5.2 Debug Tool Settings)

Figure 3-24 [Watch] panel



8. Step-execute/execute the FAA program and debug while checking the values of variables, SFRs, and registers.
   - Breakpoints can be set by clicking in the main area of the FAA program source. (Refer to 2.6.4 Breakpoint)
   - After running the program, check in the [Python Console] panel whether the ADBSEL register value is the value set in the CPU program.
     (Remark: ADBSEL register is only accessible by the CPU, so the value of the ADBSEL register cannot be displayed in the [SFR] panel while debugging the FAA.)

Figure 3-25 Example of FAA program debugger screen



3.5.8 Cautions When Using the Sample Script

✓ To disable this script (initialize Python), enter the following in the [Console] tab of the [Python console] panel.

common.PythonInitialize()

Alternatively, if you want to re-enable the sample script without reloading the sample project, enter the following in the [Console] tab of the [Python console] panel.

import os

Source (os.path.join(os.path.dirname(project.Path), 'sample_script.py'))

Remark. “os.path.join(os.path.dirname(project.Path)” is a description to get the full path of the file.

✓ The operation of sample code is not guaranteed. And the operation of this sample script is not guaranteed with all application programs and debugging operations.

✓ This sample script assists in displaying SFRs when debugging FAA programs. After completing debugging, thoroughly evaluate your system without using the sample script.

4. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

5. Reference Documents

RL78/G24 User’s Manual: Hardware (R01UH0961)

RL78 family User's Manual: Software (R01US0015)

DSPASM FAA/GREEN_DSP Structured Assembler User’s Manual (R20UT3911)

RL78/G24 Fast Prototyping Board User’s Manual (R20UT5091)

RL78 Smart Configurator User’s Gude: CS+ (R20AN0580)

CS+ V8.10.00 User’s Manual: RL78 Debug Tool (R20UT5301)

(The latest version can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News

(The latest version can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov. 14. 23	-	First edition
1.10	Mar. 28. 25	5	Figure1-4: Modification 1.3.2: Addition of description
		6	1.3.3: Addition of Remark
		8	Table2-1: Modification of tool version Table2-2: Addition of Note1
		16	Figure2-15: Modification
		17	Figure2-16: Modification Figure2-17: Modification Figure2-18: Modification
		18	Figure2-19: Modification
		19	Figure2-21: Modification, Addition of Remark1 18: Addition of Link
		20	Table2-4: Addition of function name of transfer processing
		21	Note2: Modification of description
		29	Caution2: Addition of function name of transfer processing
		32	Figure2-34: Modification
		34	The FAA program controls: Addition of description Figure2-38: Modification
		35	2.6.6: Modification of description for watch expression
		36	Figure2-40: Modification
		37	Figure2-41: Modification

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.