Renesas e^2 studio

Smart Configurator Application Examples: CMT, A/D, SCI, DMA, USB

Introduction

Smart Configurator (SC) is a GUI-based tool that has the functionalities of code generation and configuration for drivers, middleware and pins. SC generates suitable code for each Renesas MCU family and has the functionality to import code generated by FIT modules.

This application note guides user to use SC in the e^2 studio to configure drivers and generate codes.

Target Device and support compiler

Refer to the following URL for the range of supported devices:
https://www.renesas.com/smart-configurator

The following operating systems on the host computer are supported:

- Windows 7 32-bit / 64-bit
- Windows 8.1 32-bit / 64-bit
- Windows 10 32-bit / 64-bit

Software Components

Smart Configurator supports 2 types of software components: Code Generator (CG) and Firmware Integration Technology (FIT). Drivers and middleware supported by each software type are:

- Basic drivers: CG drivers (CMT, A/D Converter, SCI, etc.)
  FIT modules (CMT, DTC, DMAC, RSPI, SCIFA, etc.)
- Middleware: FIT modules (USB, Ethernet, Flash Memory (programming the on-chip flash memory), etc.)

The basic driver is a control program for peripheral functions of microcomputer such as CMT, A/D converter, SCI, etc. It is convenient to embed software component (CG driver) using code generation (CG) function.

In addition, FIT modules can be embedded for using middleware such as USB, Ethernet, and Flash memory (programming the on-chip flash memory) as software components.

List of abbreviations:

CMT: Compare Match Timer

DTC: Data Transfer Controller

DMAC: Direct Memory Access Controller

RSPI: Serial Peripheral Interface

SCIFA: FIFO Embedded Serial Communications Interface

Contents

1. Overview

1.1 Purpose

This document guides user to use Smart Configurator in few application examples.

1.2 Operating Environment

Target devices	RX64M Group RX65N, RX651 Group RX130 Group
Debugger	E1 /E2 Lite
IDE	e2 studio v.7.0.0 or above
Toolchains	Renesas C/C++ compiler package for RX family

1.3 Basic operation steps of Smart Configurator project

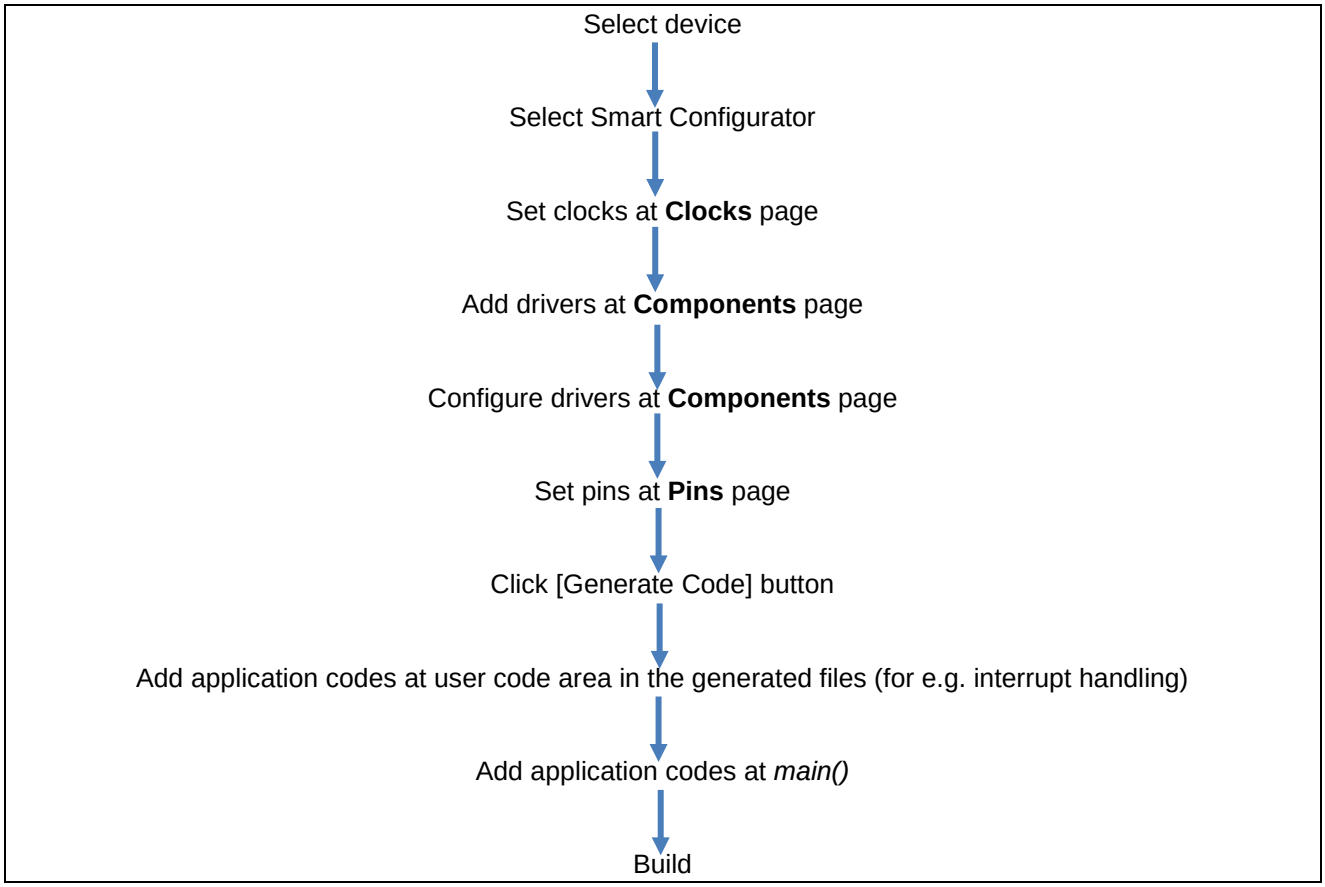


Figure 1-1-1 Basic operation

Refer to “Smart Configurator User Guide” document for detailed operations of Smart Configurator.

2. Application Example 1 (Creating a basic program using Smart Configurator)

This chapter describes how to create a LED blinking project using Code Generator drivers in Smart Configurator. The following resources are required for this example:

- IDE: e² studio v7.0 and above
- Toolchain: Renesas RXC Toolchain
- Board: Renesas Starter Kit+ for RX65N-2MB
- MCU: RX65N R5F565NEDxFC
- Debugger: E1 or E2 Lite

The schematic diagram of Renesas Starter Kit+ for RX65N 2MB is shown below. In this example, LED2 will be configured to blink every 5000 counts.

Figure 2-1 Schematic diagram of Renesas Starter Kit+ for RX65N 2MB

The table below shows CG drivers to be configured.

Drivers	Resource	Function
Ports	PORTG	Connected to LED2 (PG6) Output a low level '0' to turn on LED2
Compare Match Timer	CMT0	Toggle LEDs every 5000 counts with output compare interrupt handler

Based on the schematic below, RX65N main clock is connected to a 24MHz crystal resonator. This clock will be configured as main clock source in Chapter 2.7 on debug configuration setting.

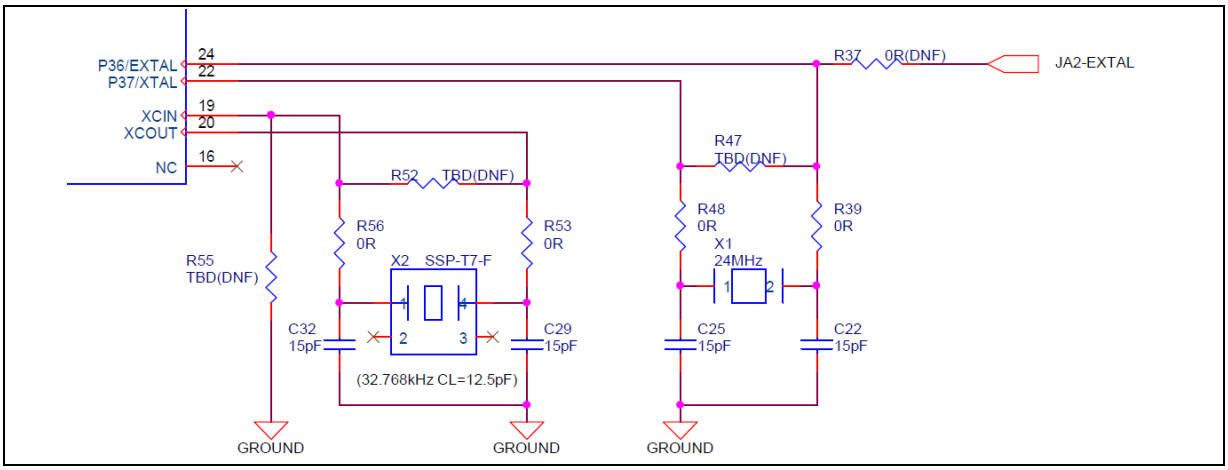


Figure 2-2 Schematic diagram of Renesas Starter Kit+ for RX65N-2MB

The program flowchart is shown as below:

a) Main function:

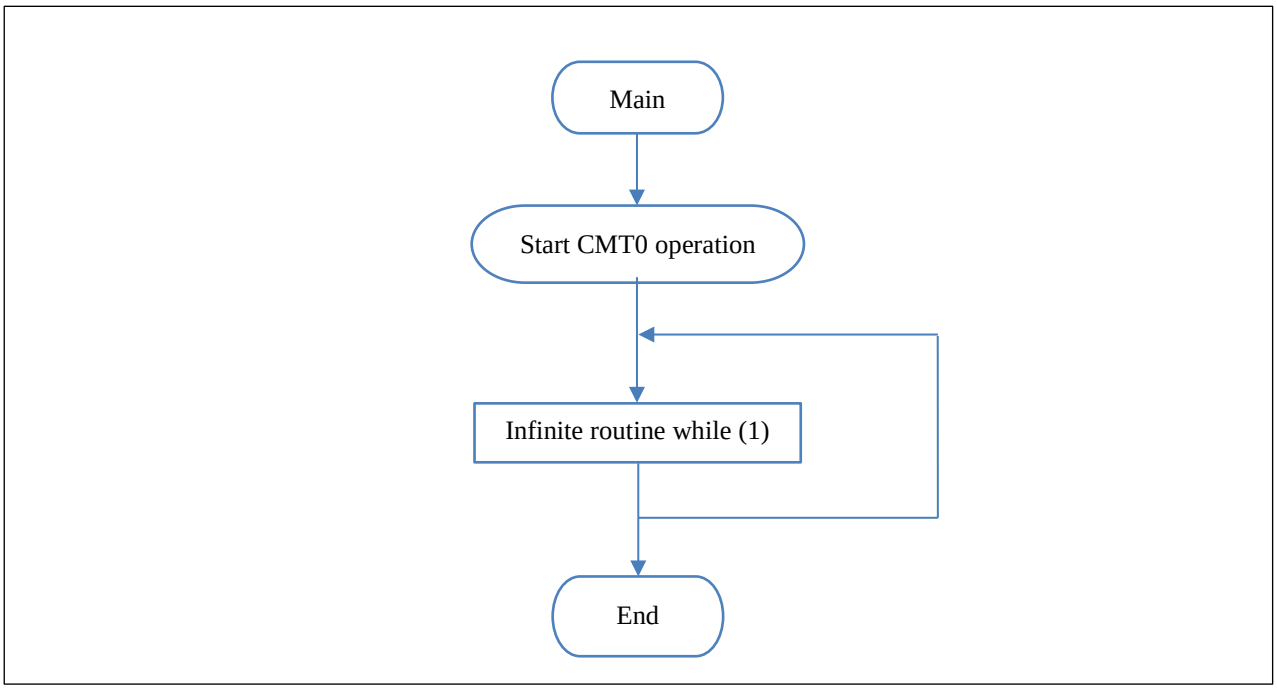


Figure 2-3 Main flowchart

b) CMT0 interrupt function:

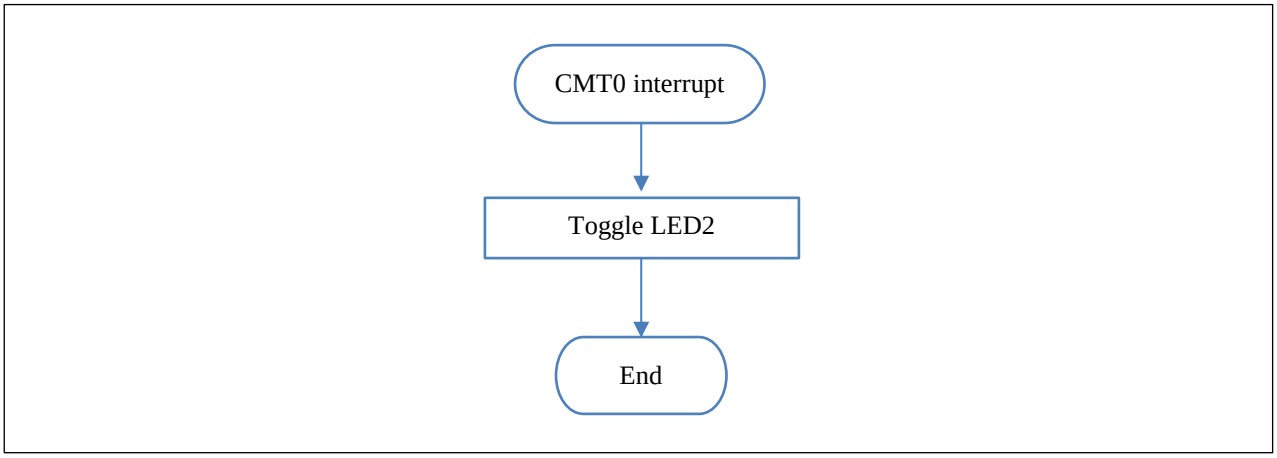


Figure 2-4 CMT0 interrupt flowchart

2.1 Creating a Workspace

1) Start e² studio from Windows® Start Menu. Use default workspace folder and click [OK].

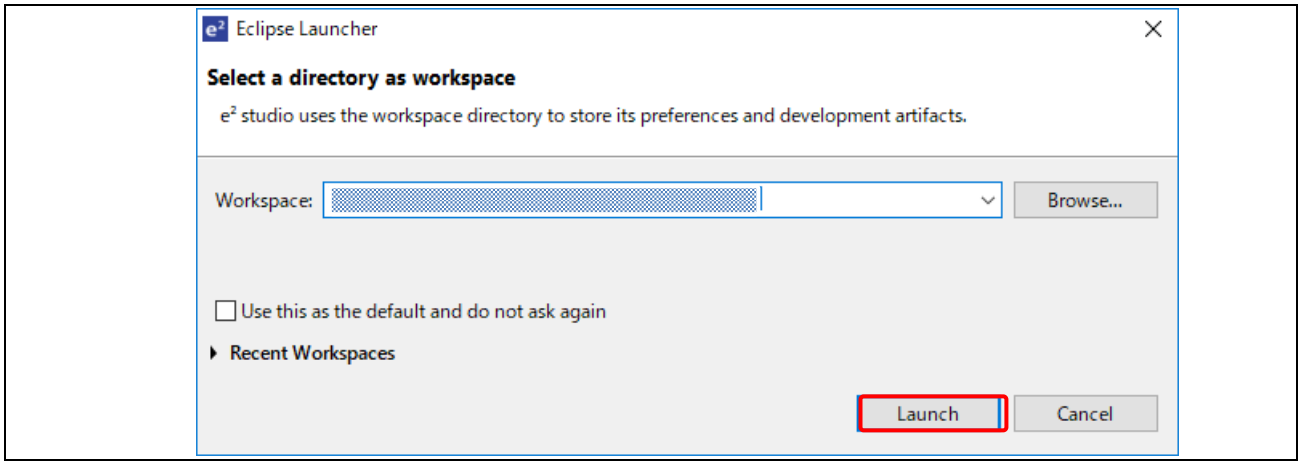


Figure 2-5 Workspace Launcher

2.2 Creating a Project

1) Create a new C project in e² studio.

   Go to [File] → [New] → [C/C++ Project] to start new project generation

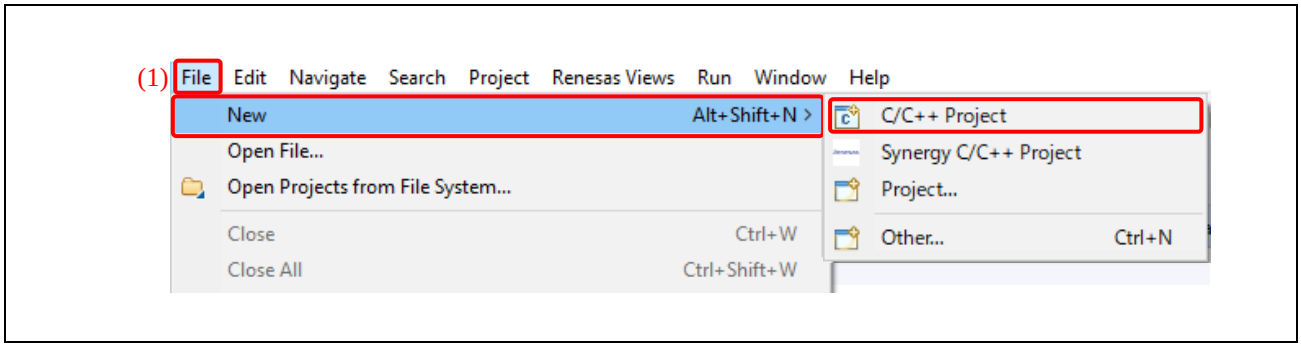


Figure 2-6 Creating project from File menu

2) Select [Renesas RX] → [Renesas CC-RX C/C++ Executable Project] → [Next]

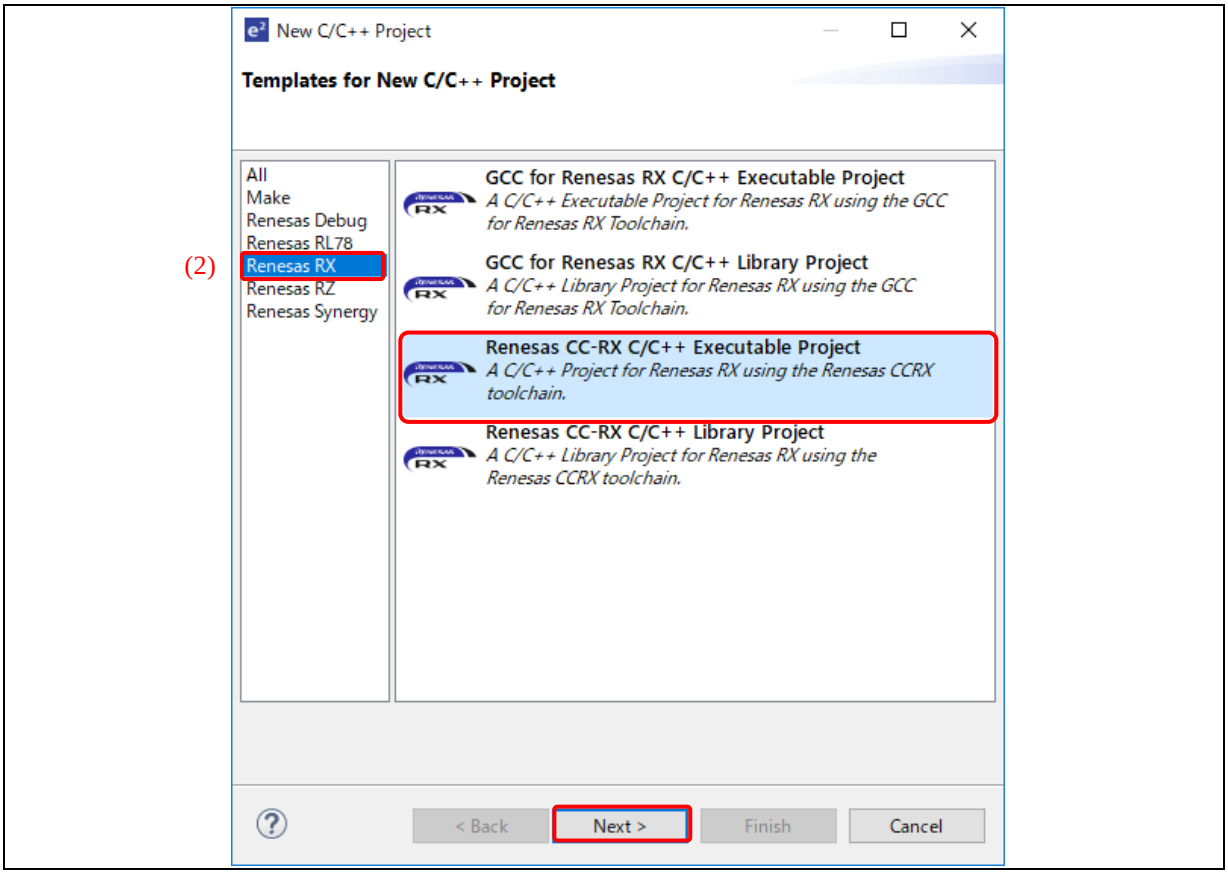


Figure 2-7 Creating project from File menu

3) Give an appropriate name to the project, for example “Smart_Configurator_Example” → [Next]

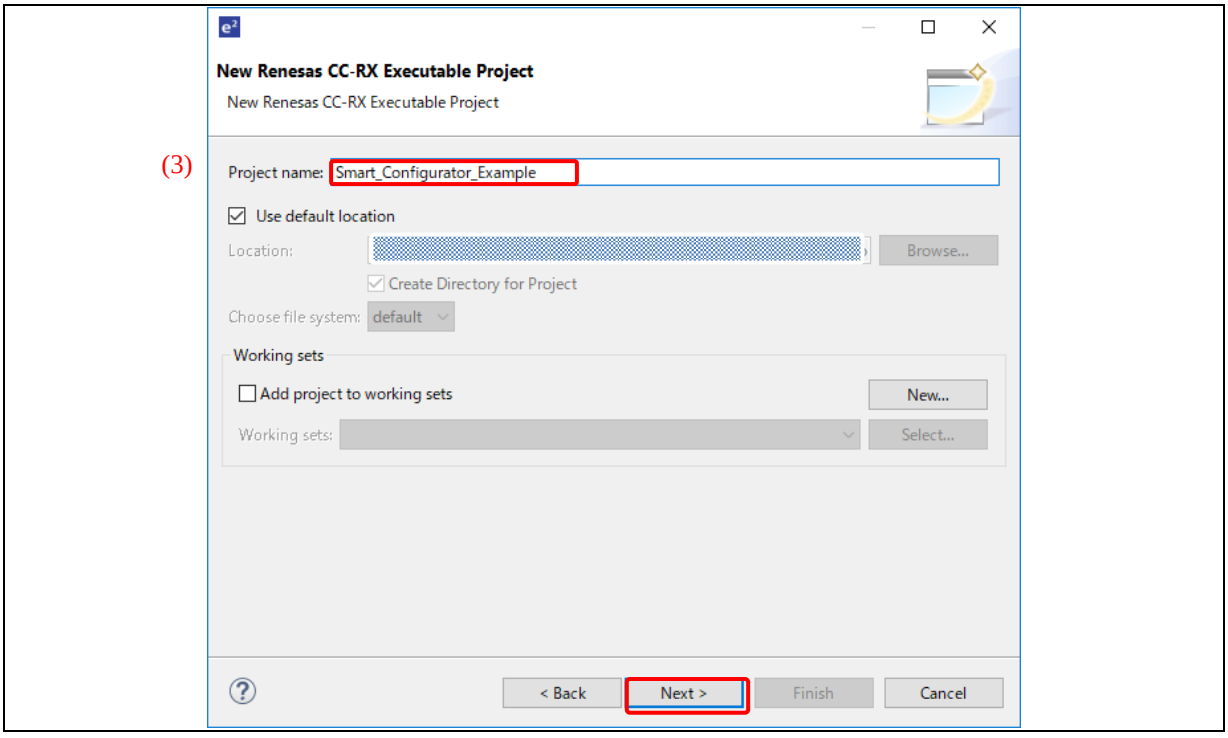


Figure 2-8 Creating project from File menu

4) Select “C” as Language
5) Select “Renesas CCRX” as Toolchain
6) Select Toolchain Version. For e.g. “v3.01.00”
7) Select Target Device: “RX600 > RX65N > RX65N - 176pin > R5F565NEDxFC”
8) Ensure [Create Hardware Debug Configuration] is ticked. Select emulator, for e.g. “E2 Lite(RX)”.
9) Click [Next]

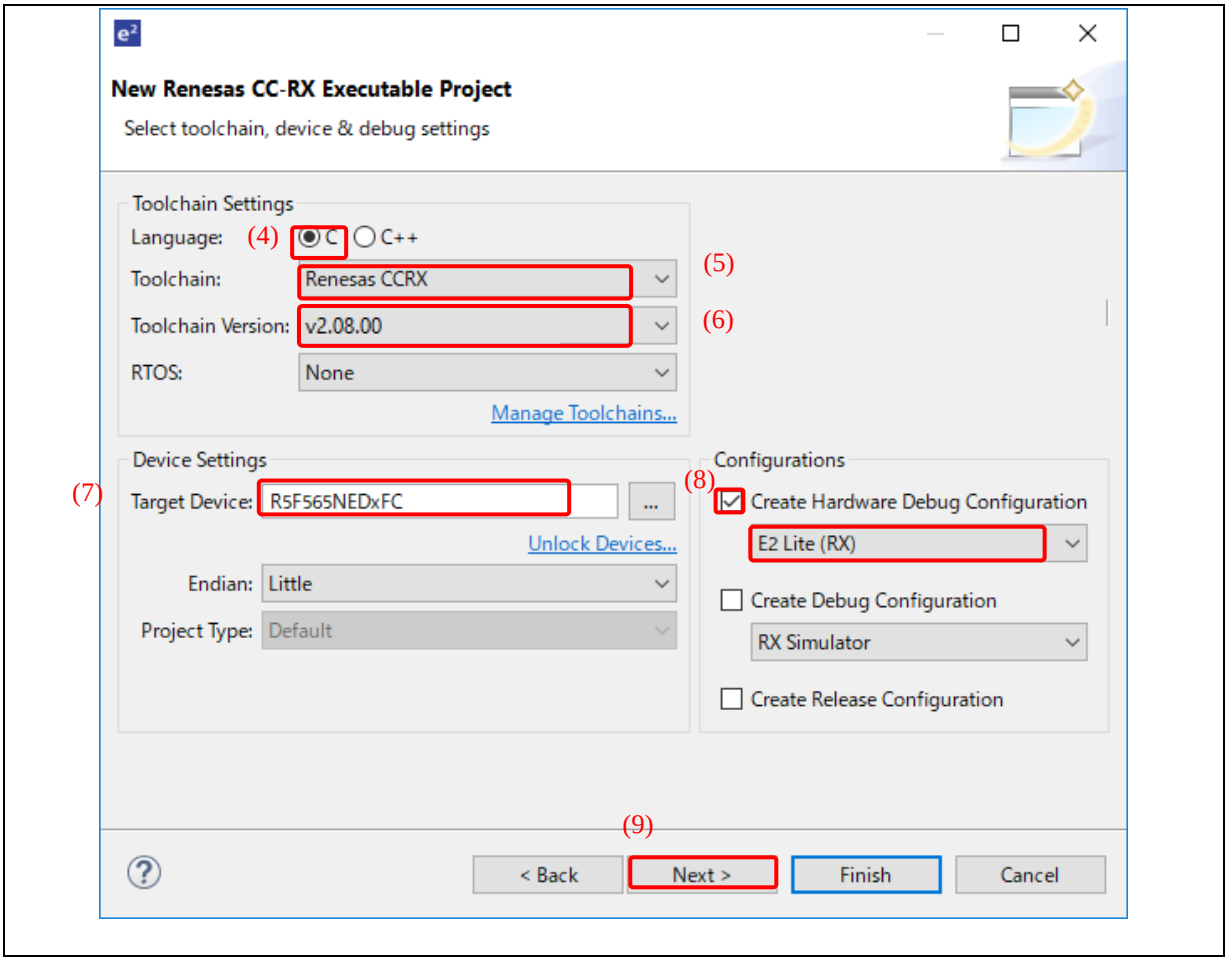


Figure 2-9 C Project - Target Specific Settings example with E2 Lite emulator

10) In the “Select Coding Assistant settings” dialog, select the checkbox of “Smart Configurator”

11) Click [Finish]

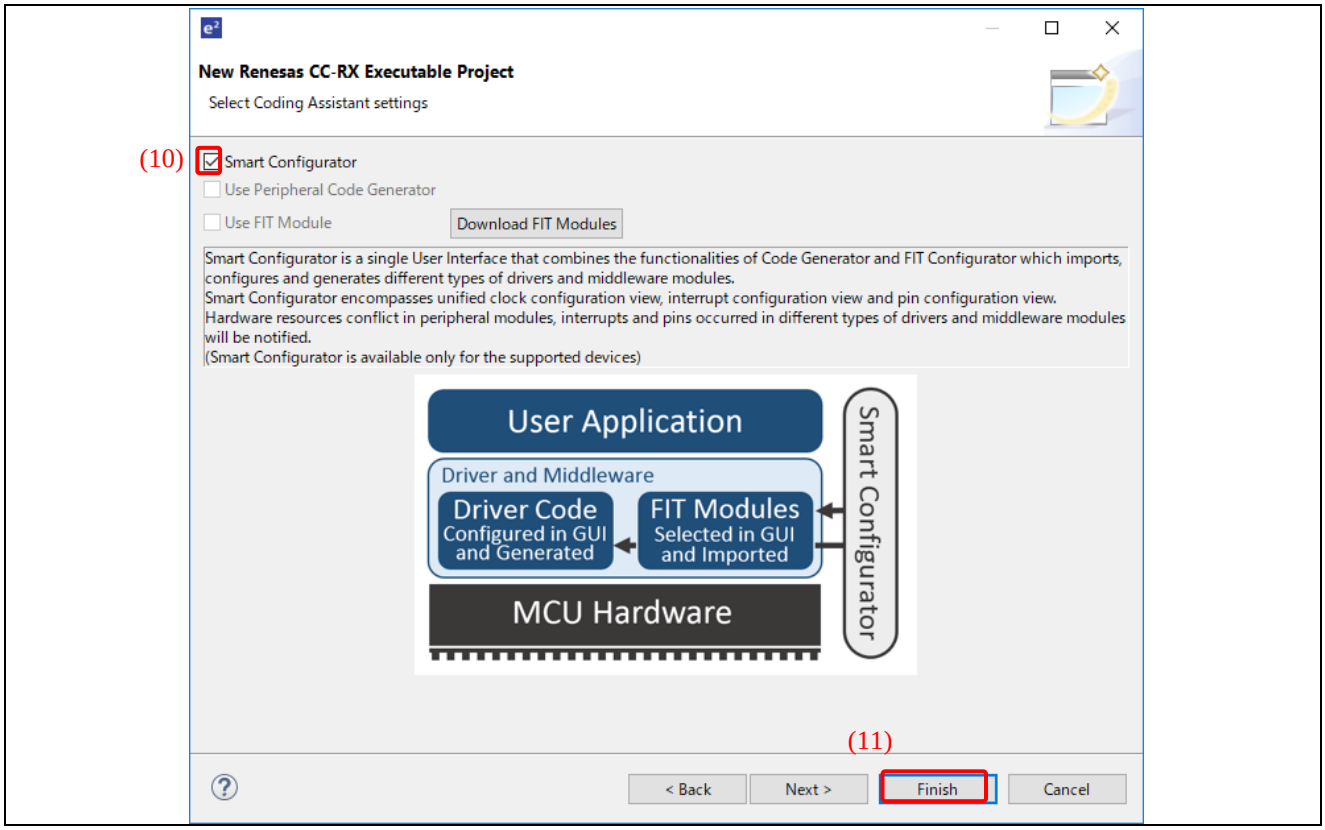


Figure 2-10 Select Coding Assistant Tool

2.3 Adding a peripheral driver

Smart Configurator perspective will be launched as shown below.

1) In Smart_Configurator_Example.scfg pane, click the [Components] tab

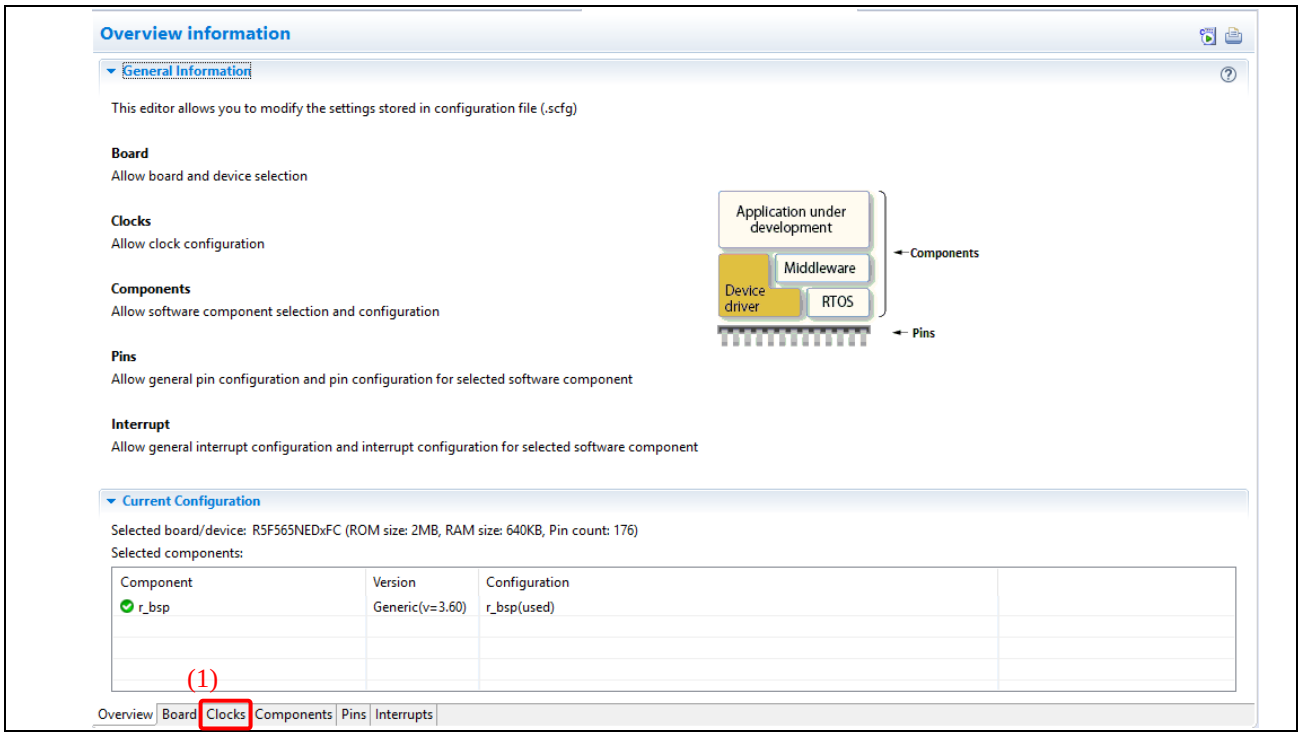


Figure 2-11 Smart Configurator perspective

2) Click  to add new component.

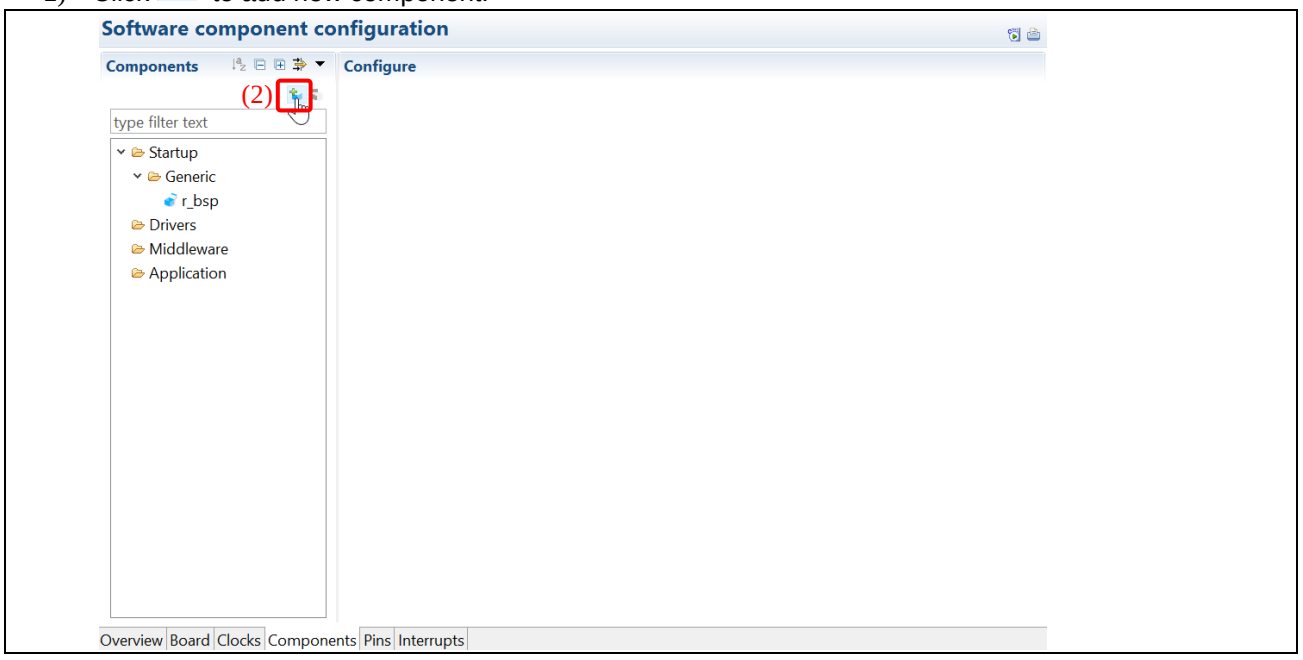


Figure 2-12 Software component configuration in Smart Configurator

3) Add *Compare Match Timer* driver into the project

a. Select “Timers (Drivers)” from the “Function” option
b. Navigate the component list and select *Compare Match Timer*
c. Click [Next] to continue

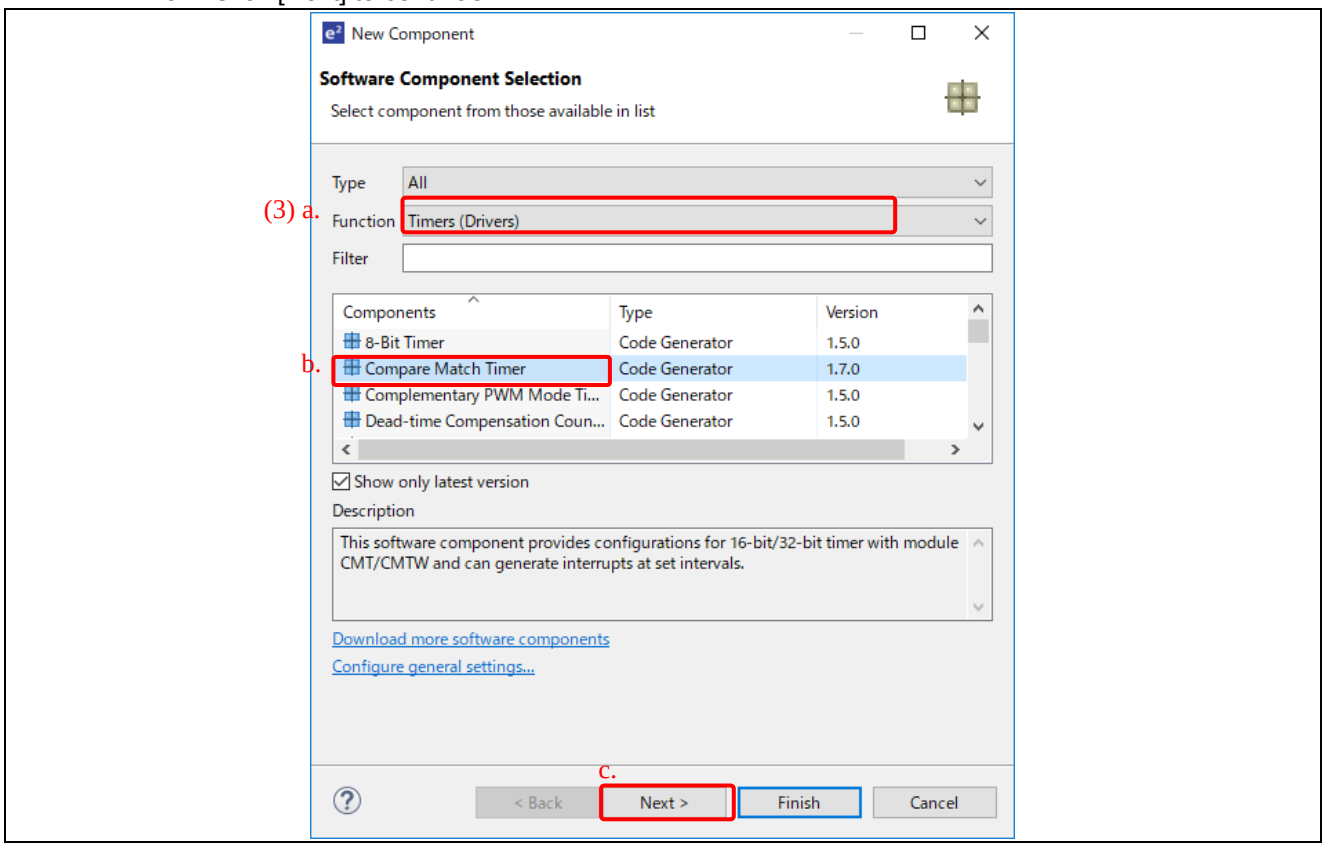


Figure 2-13 Select software component

d. Select CMT0 as resource
e. Keep the default configuration name. Source files and API will be generated based on this configuration name
f. Click [Finish]

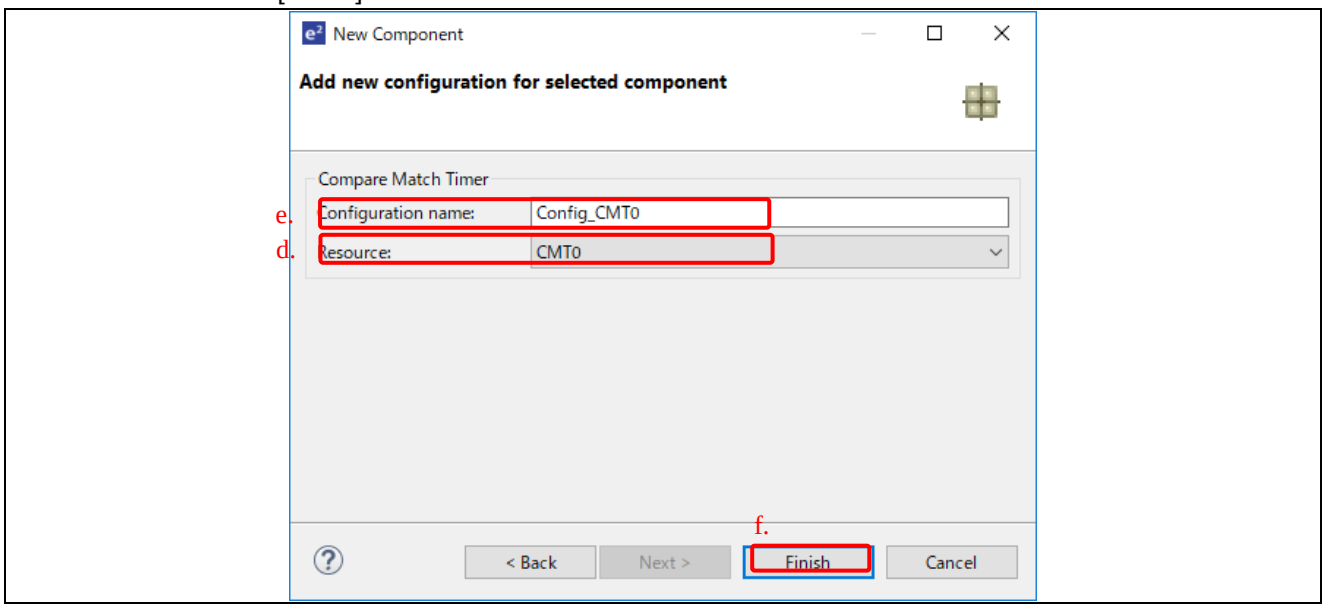


Figure 2-14 Add new configuration to the project

4) In the Configure panel of *Config_CMT0*,

a. Click Count clock setting as [PCLK/512] and set the Interval value to [5000 count]
b. Ensure the checkbox of “Enable compare match interrupt (CMI0)” is ticked

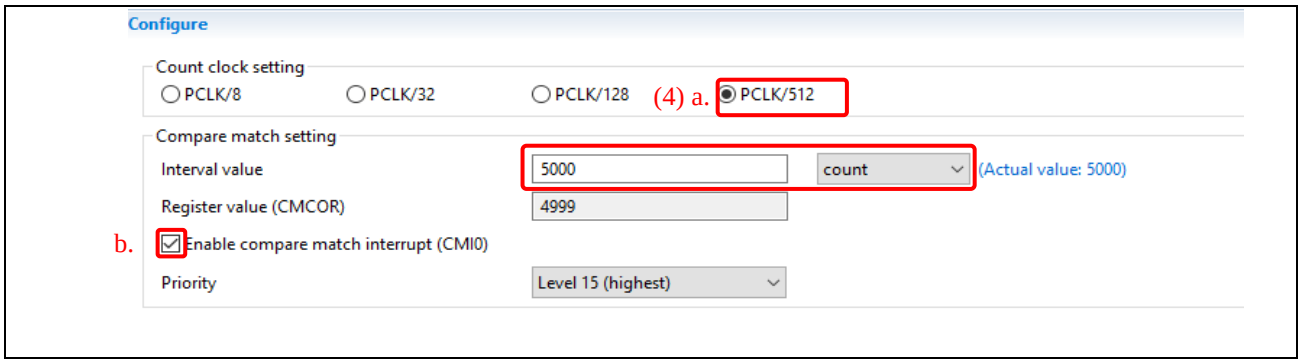


Figure 2-15 Compare Match Timer configuration

5) Add *Ports* driver into the project

a. At [Components] tab, click  to add *Ports*
b. Select “I/O Ports (Drivers)” from the “Function” option
c. Select “Ports” driver
d. Click [Next] to continue

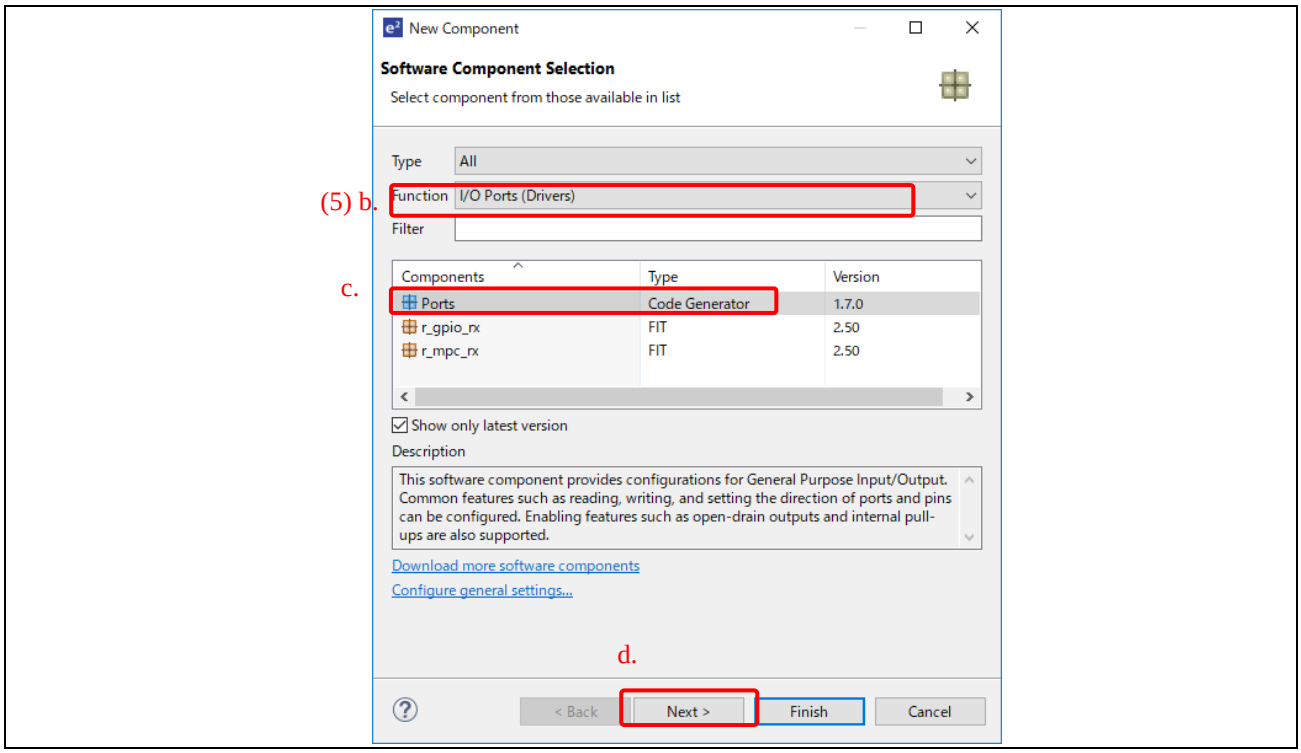


Figure 2-16 **Select software component**

e. Keep the default configuration name and click [Finish]

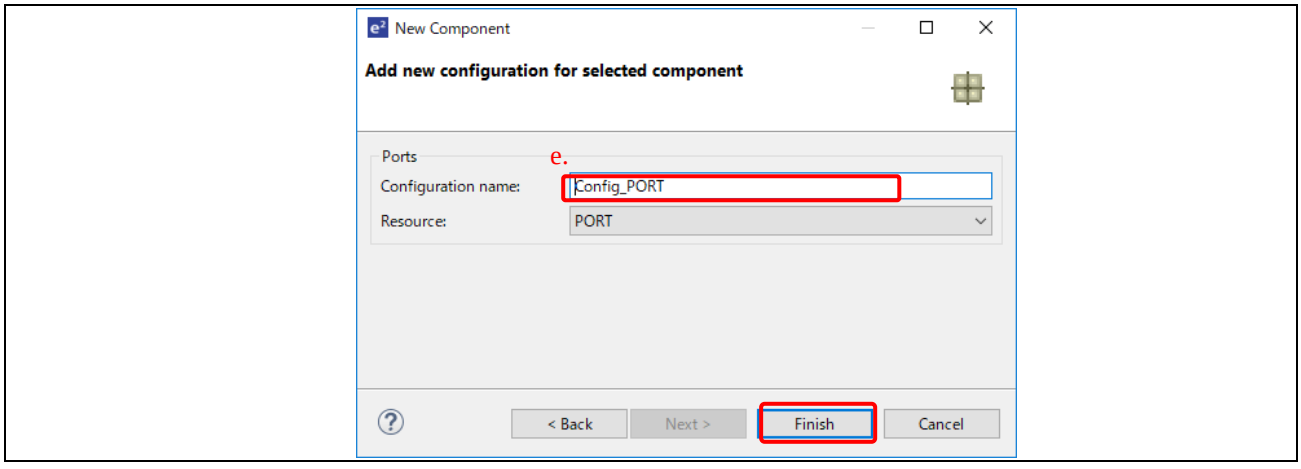


Figure 2-17 Add new configuration to the project

6) At [Port selection] tab, check PORTG checkbox.

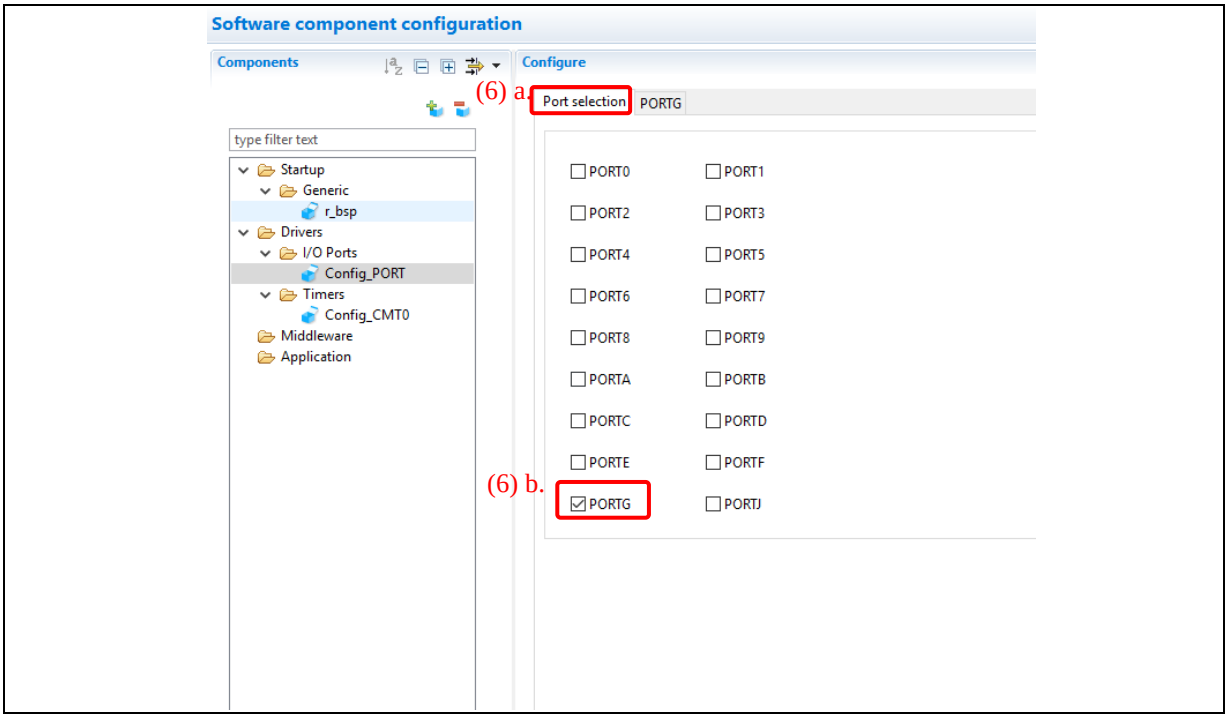


Figure 2-18 Port selection tab

7) At [PORTG] tab, set PG6 as output pin.

a. At PG6 section, select [Out].

b. Check [Output 1] checkbox to set LED2 to off status initially.

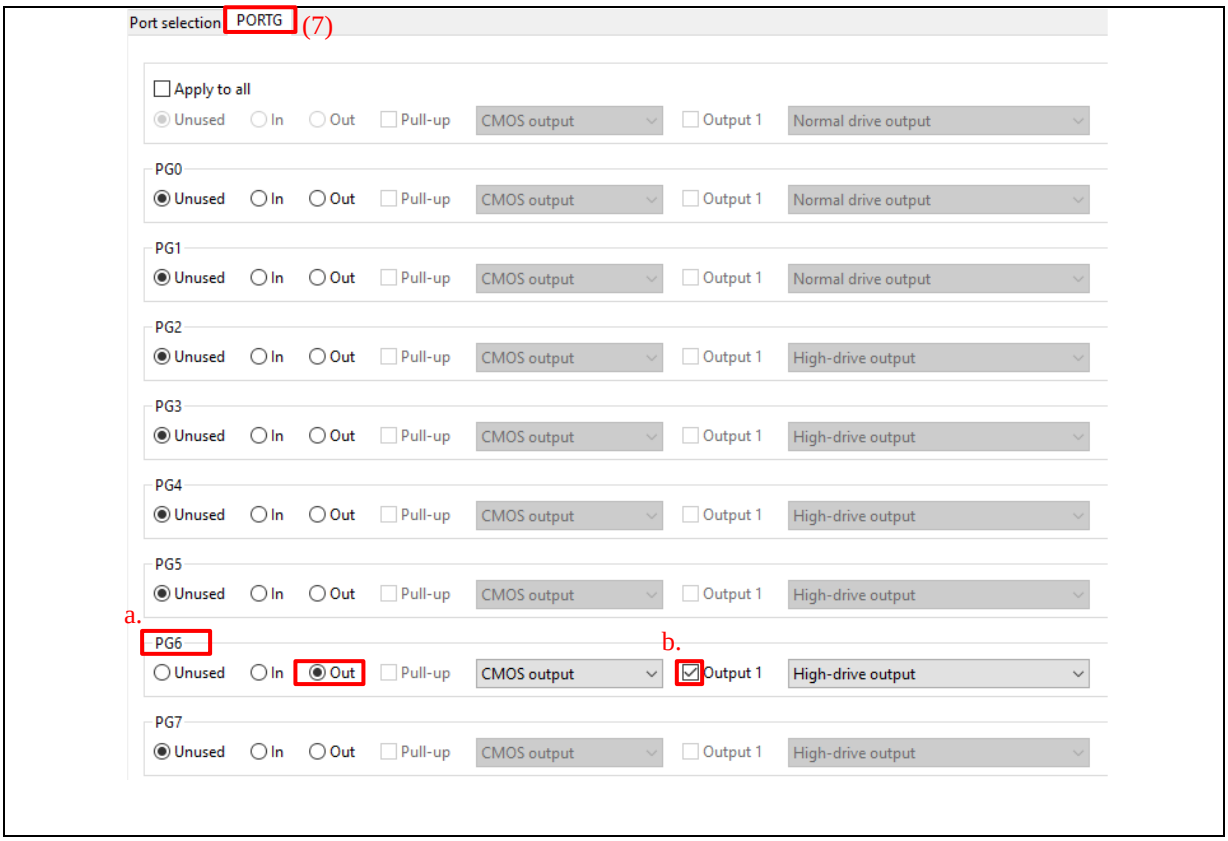


Figure 2-19 PORTG tab

8) At [Pins] tab, click  button to switch tree to Software Components view

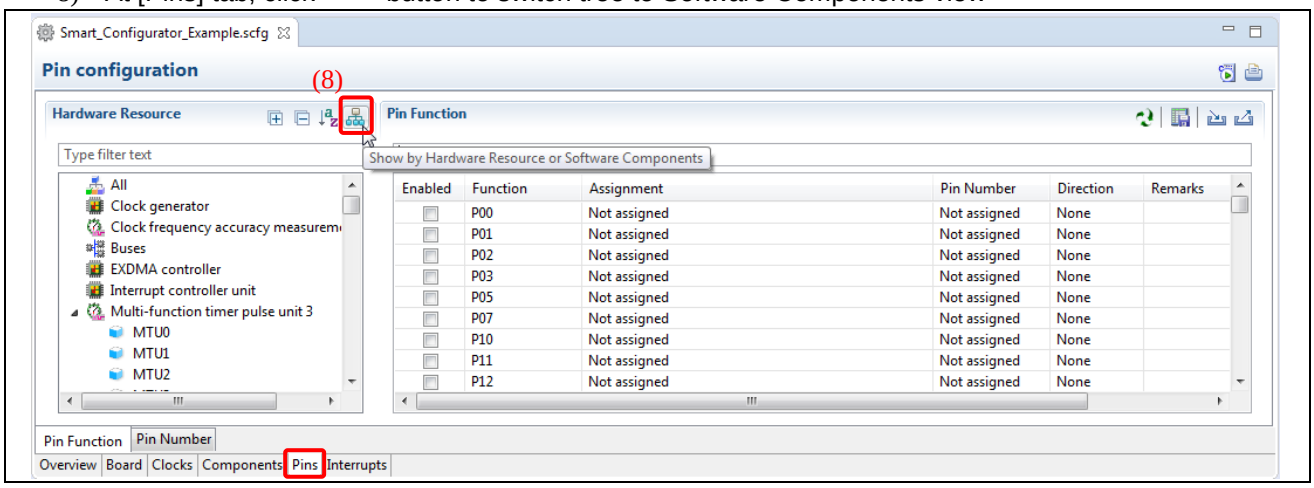


Figure 2-20 Pins tab

9) At the tree, click *Config_PORT* to view pin configurations
10) Type “PG” in the Pin Function filter
11) Ensure PG6 pin is enabled:

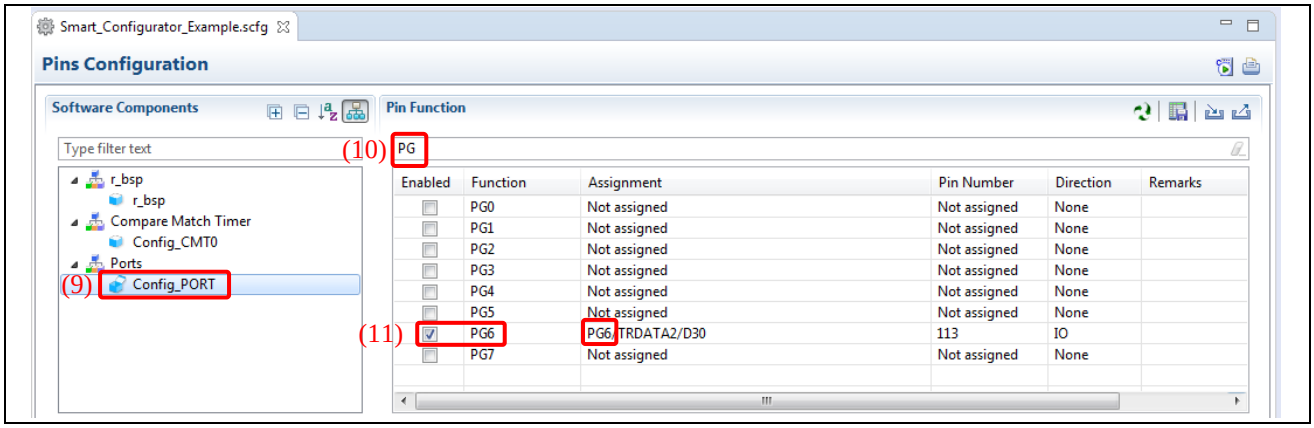


Figure 2-21 Pin configuration of *Config_PORT*

2.4 Generating codes

1) Click  to generate codes

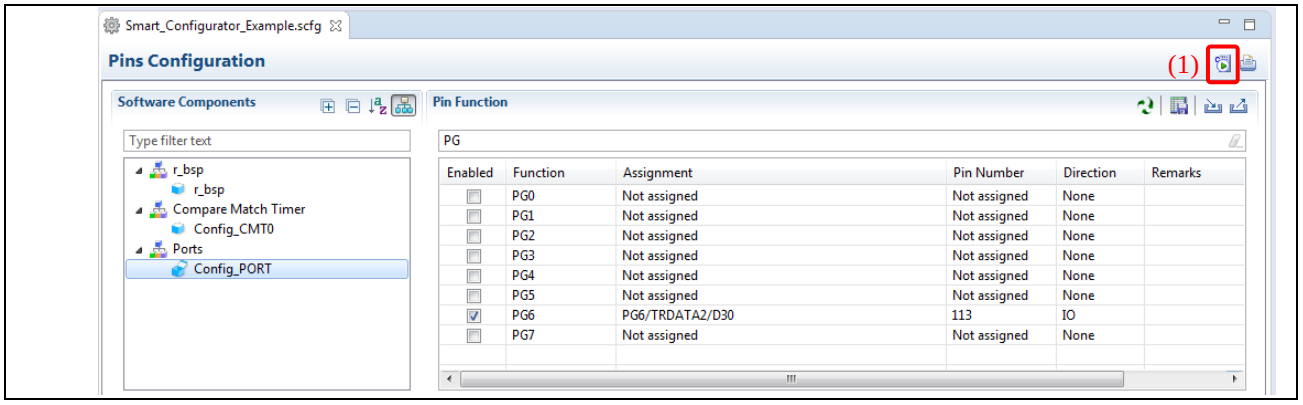


Figure 2-22 Generate codes

2) Message ‘Code generation is successful will be shown at Console

3) Files generated into \src\smc_gen folder of the project

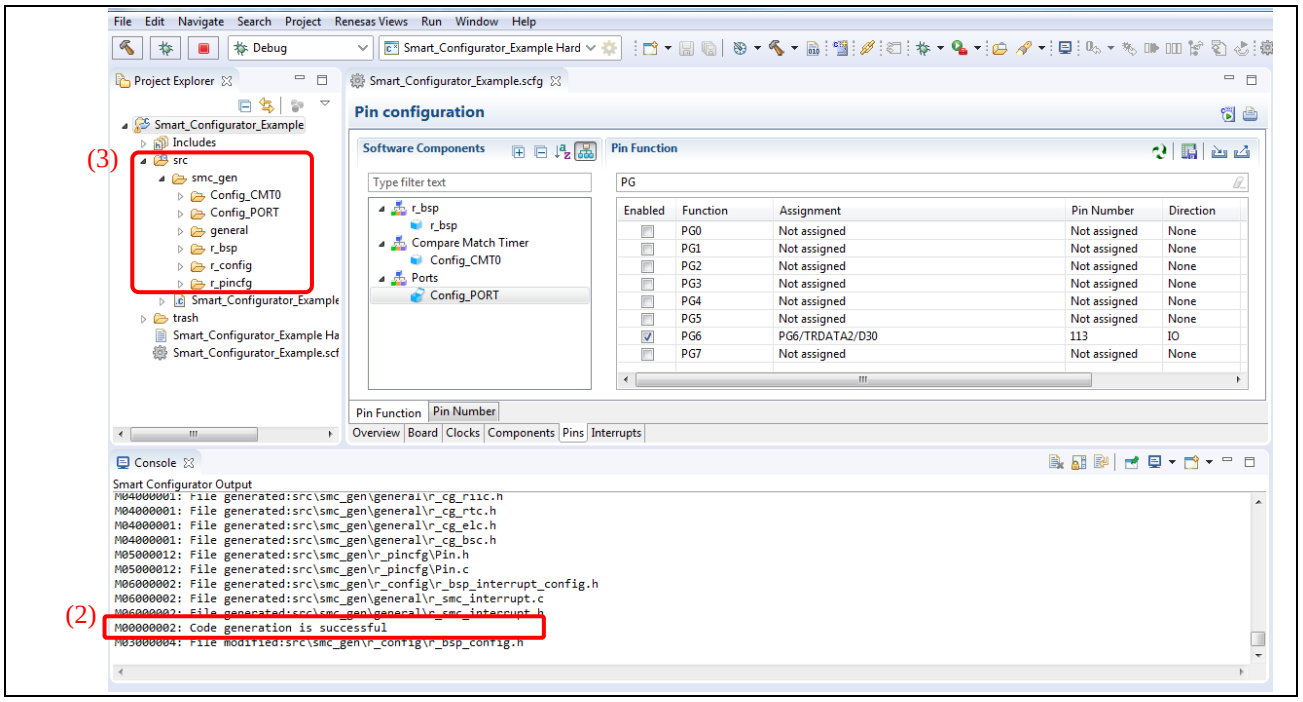


Figure 2-23 Successful code generation

2.5 Adding definition codes in “r_cg_userdefine.h” file

1) At the project tree, double-click to open “r_cg_userdefine.h” file in \src\smc_gen\general folder

   a. Add LED2 port setting in the part between the beginning and the end of the comment as shown in the figure below

```
/* LED port setting */
#define LED2            (PORTG.PODR.BIT.B6)
```

Your source file should look like this:

```
/***********************************************************************************
Macro definitions (Register bit)
***********************************************************************************/
/* Start user code for register. Do not edit comment generated here */

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

/***********************************************************************************
Macro definitions
***********************************************************************************/
/* Start user code for macro define. Do not edit comment generated here */
/* LED port settings */
#define LED2 (PORTG.PODR.BIT.B6)
/* End user code. Do not edit comment generated here */

/***********************************************************************************
Typedef definitions
***********************************************************************************/
/* Start user code for type define. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

/***********************************************************************************
Global functions
***********************************************************************************/
/* Start user code for function. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#endif
```

Figure 2-24 r_cg_userdefine.h

2.6 Adding application codes in Compare Match Timer driver

1) Open “Config_CMT0_user.c” in \src\smc_gen\Config_CMT0 folder of the project

   a. Toggle LED2 in *r_Config_CMT0_cmi0_interrupt (void)* function

```
/* Toggle LED2 whenever timer is up */
LED2 ^= 1U;
```

Your source file should look like this:

```
/***********************************************************************************************************
* Function Name: r_Config_CMT0_cmi0_interrupt
* Description  : This function is CMI0 interrupt service routine
* Arguments    : None
* Return Value : None
***********************************************************************************************************/

#if FAST_INTERRUPT_VECTOR == VECT_CMT0_CMI0
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0),fint)
#else
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0))
#endif
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_cmi0_interrupt. Do not edit comment generated here */
    /* Toggle LED2 whenever timer is up */
    LED2 ^= 1U;
    /* End user code. Do not edit comment generated here */
}
```

Figure 2-25 Config_CMT0_user.c

2) Start CMT0 operation.

a. Open “Smart_Configurator_Example.c” in \src folder, add below code into *main()* function:

```
/* Start CMT0 counter operation */
R_Config_CMT0_Start();

while(1U)
{
        nop();
}
```

Your source file should look like this:

```
*  FILE        : Smart_Configurator_Example.c
#include "r_smc_entry.h"

void main(void);

void main(void)
{
    /* Start CMT0 counter operation */
    R_Config_CMT0_Start();

    while(1U)
    {
        nop();
    }

}
```

Figure 2-26 Start CMT0 operation in main file

2.7 Build and run on hardware board

1) Build the project, click [Project] → [Build Project]

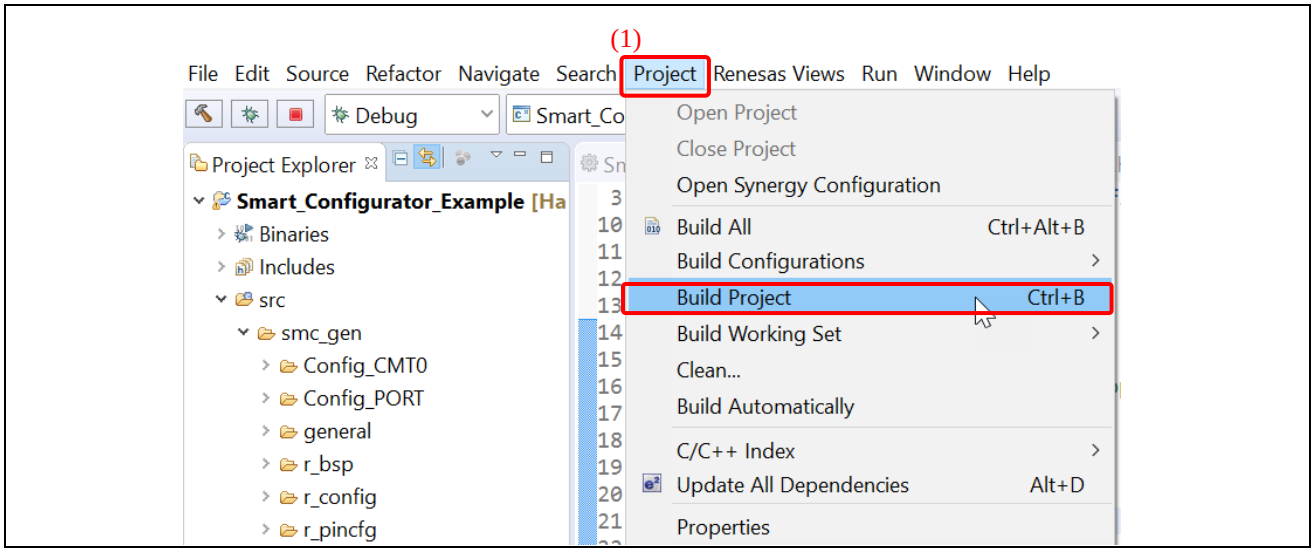


Figure 2-27 Build project

2) Debug the code

a. Connect RSK+ RX65N 2MB board to E1 / E2 Lite emulator and connect the E1 / E2 Lite emulator to PC

b. [Run] → [Debug Configurations…] or the downward arrow by the side of  icon → [Debug Configurations…] to open the “Debug Configurations” window

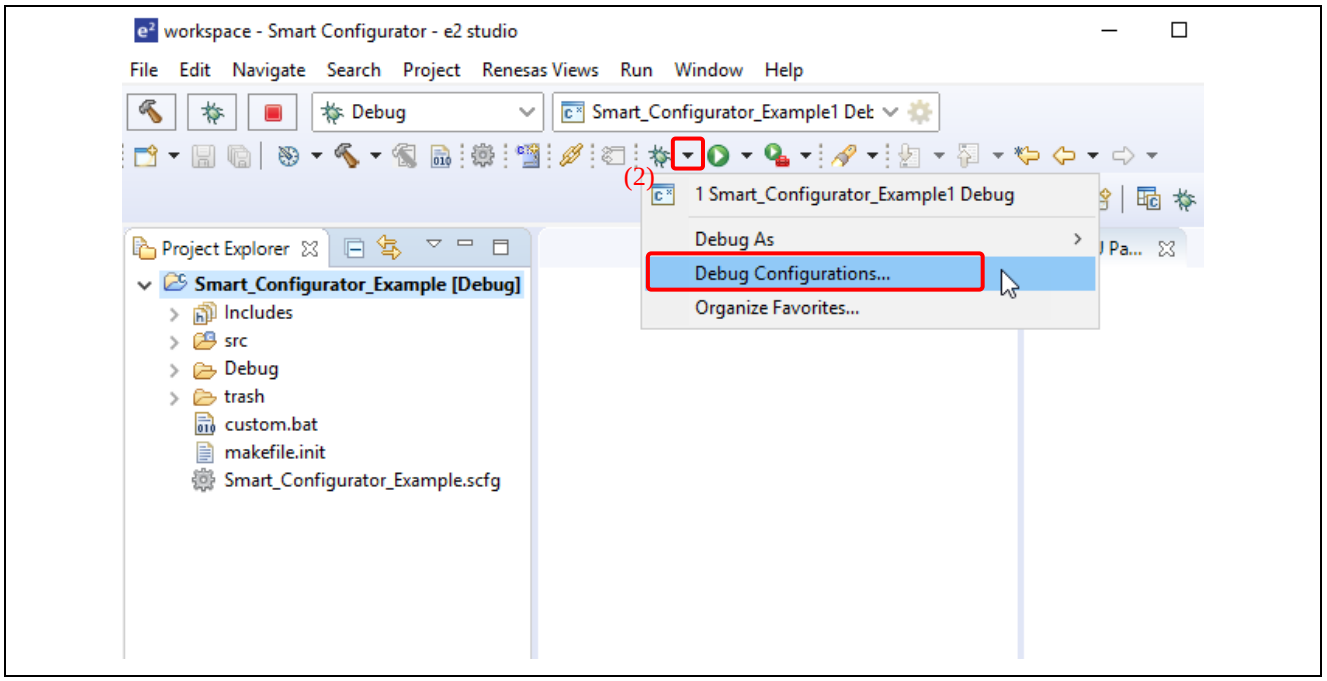


Figure 2-28 Open Debug Configurations window

3) Expand [Renesas GDB Hardware Debugging] and click on [Smart_Configurator_Example_Hardware Debug].Go to the [Debugger] Tab, click on [Connection Settings] tab, make sure:

   a. Debug Hardware: Select E1 (RX) or E2 Lite (RX)

   b. Target Device: R5F565NEDxFC

   c. Main Clock Source: EXTAL

   d. Extal Frequency[MHz]: 24MHz

   e. Permit Clock Source Change On Writing Internal Flash Memory: Yes

   f. Power Target From The Emulator (MAX 200mA): Yes

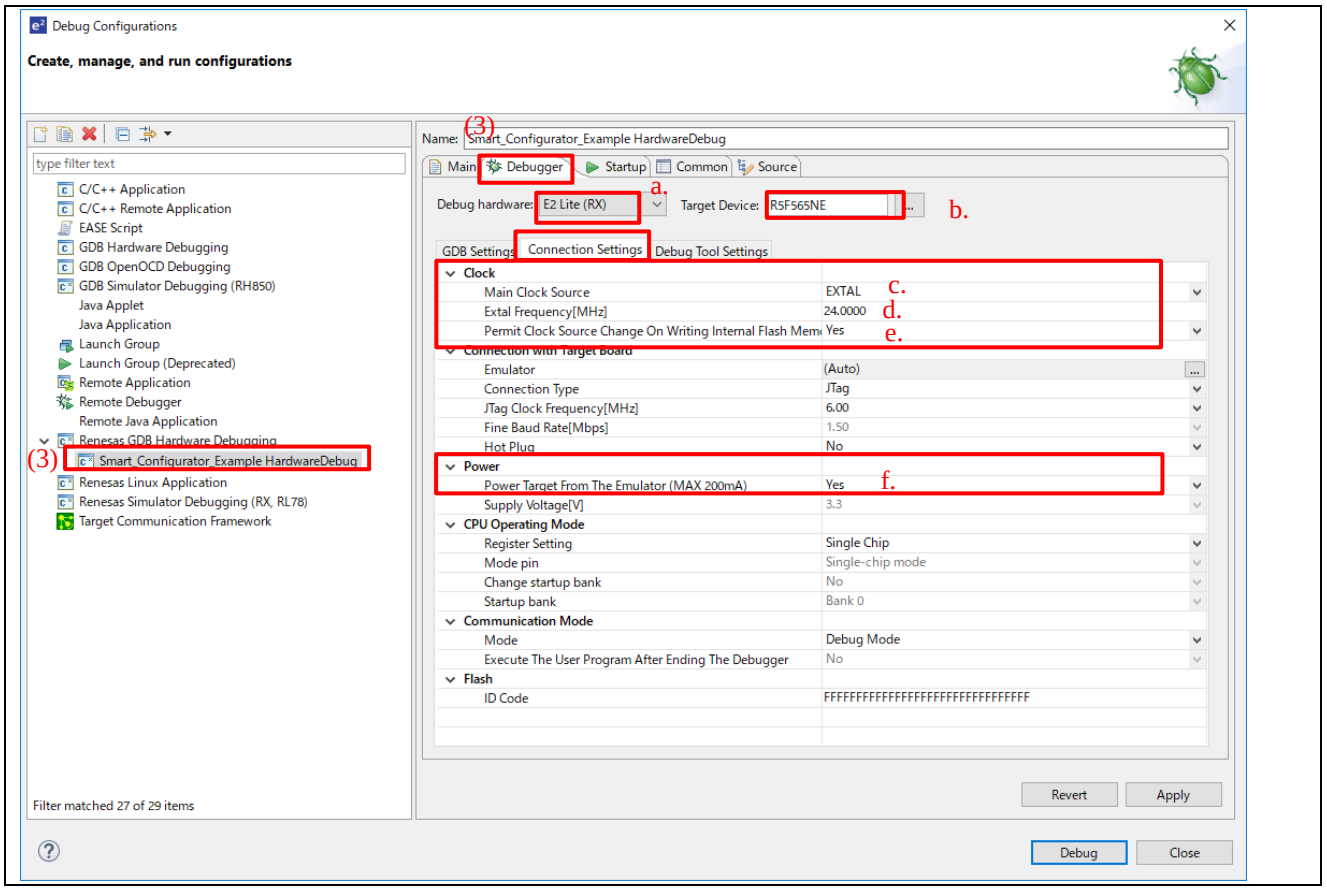


Figure 2-29 Debug Configurations example when connecting to E2 (Lite) emulator

4) At Startup, uncheck “Set breakpoint at: main”

5) Click [Debug] to start debugging.

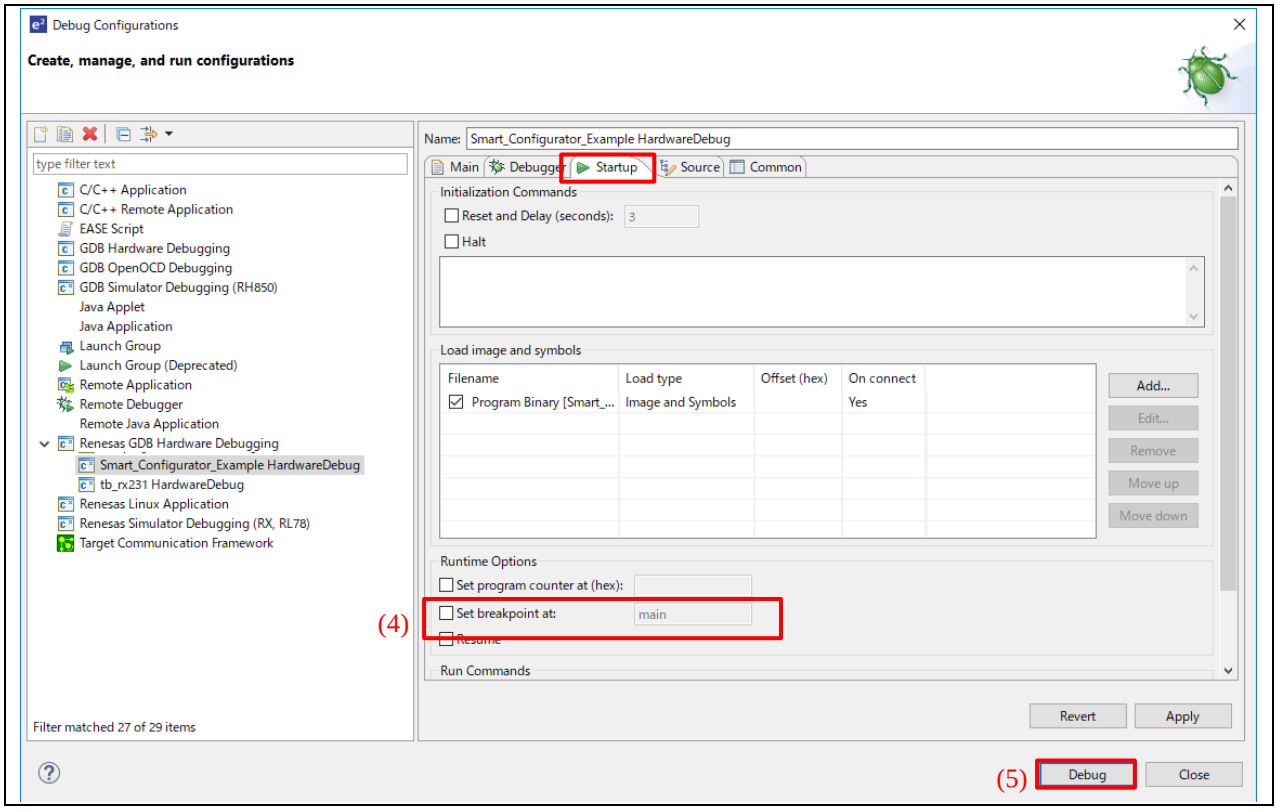


Figure 2-30 Remove breakpoint at main

6) ‘Confirm Perspective Switch’ dialog may pop up, click [Yes] to continue.

7) Click to start project execution.

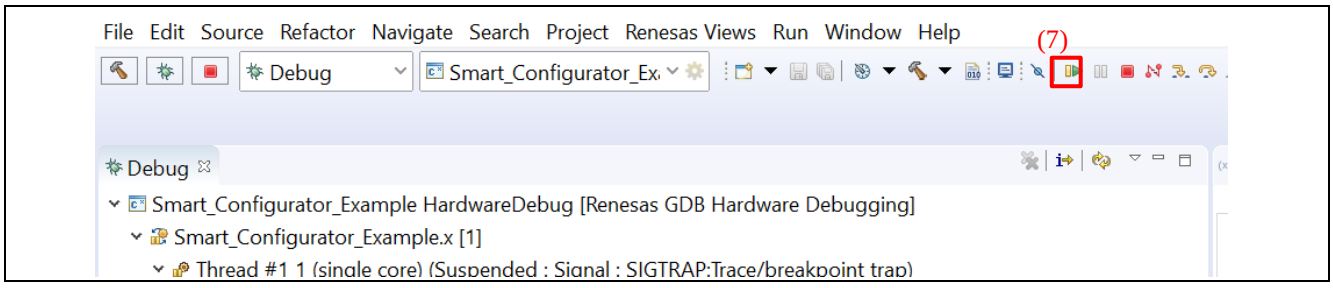


Figure 2-31 Start execution

8) Observed that LED2 is blinking. Refer to chapter 2.9 “Operation on board” for position of LED2.

9) To suspend the execution, click suspend button 

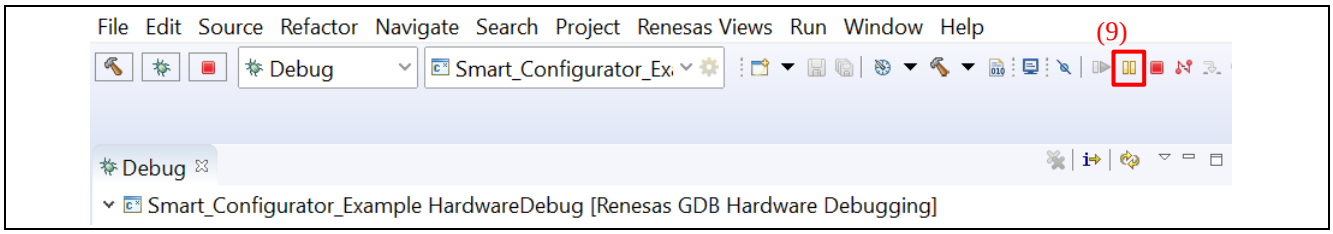


Figure 2-32 Suspend execution

10) To stop the execution, click disconnect button  to end debug session

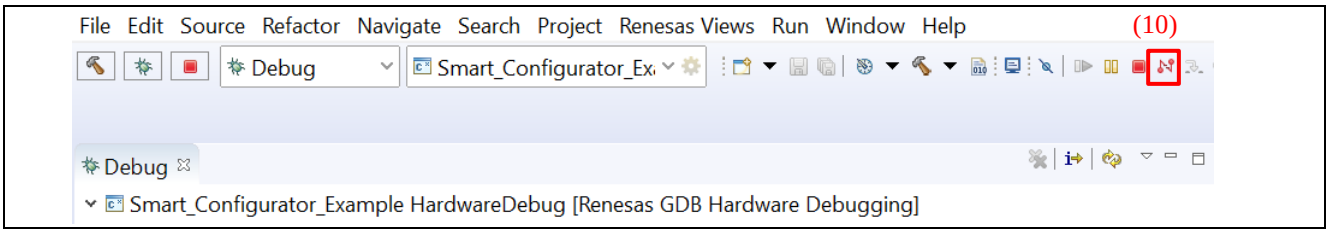


Figure 2-33 Stop debug

2.8 Changing LED2 blinking interval

1) Click  at the right hand side of the toolbar to switch to Smart Configurator perspective

Figure 2-34 Change to Smart Configurator perspective

2) In Smart_Configurator_Example.scfg pane, click the [Components] tab and select *Config_CMT0* from the tree
3) Change the Interval value to [50000 count] to slow down LED2 blinking speed
4) Click  on the toolbar to save changes and click  to regenerate codes

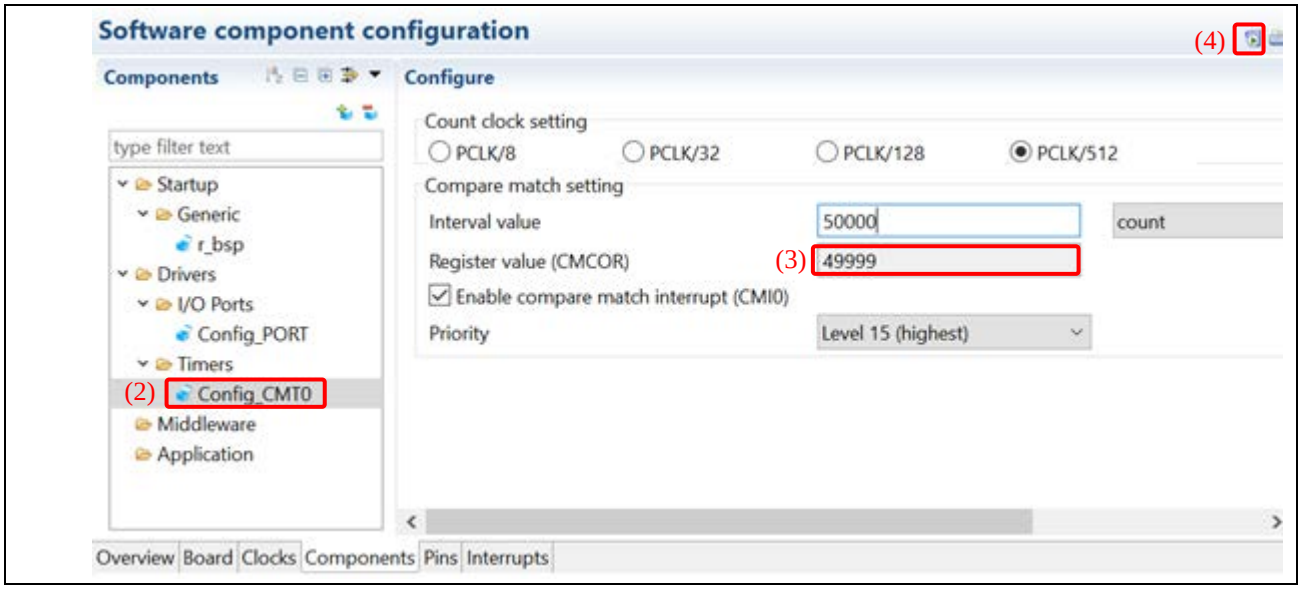


Figure 2-35 Compare Match Timer configuration

5) Click [Project] → [Build Project] to build the project
6) Click [Yes] to reload the updated project

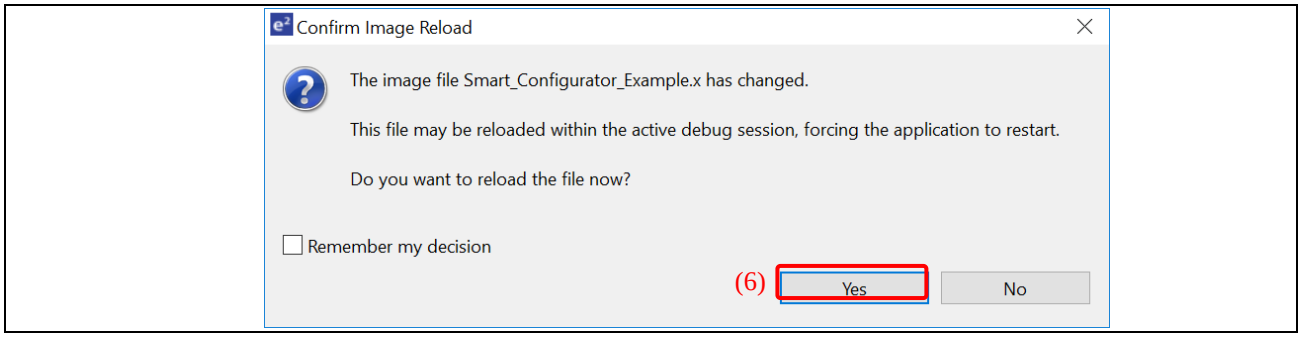


Figure 2-36 Confirm image reload

7) 'Confirm Perspective Switch' dialog may pop up, click [Yes] to continue.

8) Click icon to start the execution

LED2 is blinking at a lower speed. Refer to chapter 2.9 “Operation on board” for position of LED2.

2.9 Operation on board

On RSK+ RX65N 2MB board, LED2 is blinking.

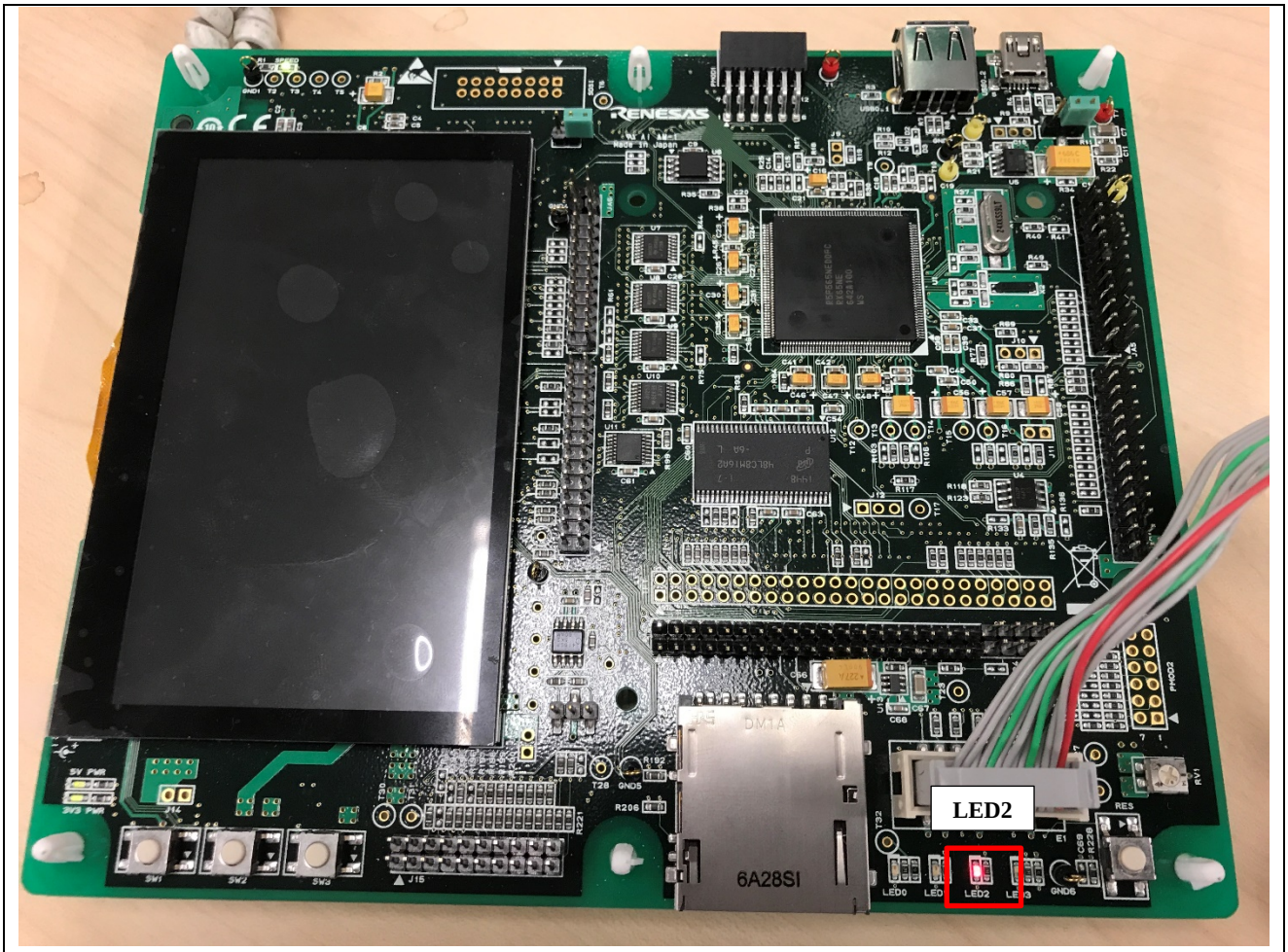


Figure 2-37 LEDs on RSK+ RX65N 2MB board

3. Application Example 2 (Adding a function using Smart Configurator)

This chapter describes how to use potentiometer to control the LED2 blinking rate.

The table below shows CG drivers to be configured in this example.

Drivers	Resource	Function
Single Scan Mode S12AD	S12AD	Read potentiometer value for controlling LED blinking rate

S12AD application codes will be added to the program as below:

a) Main function:

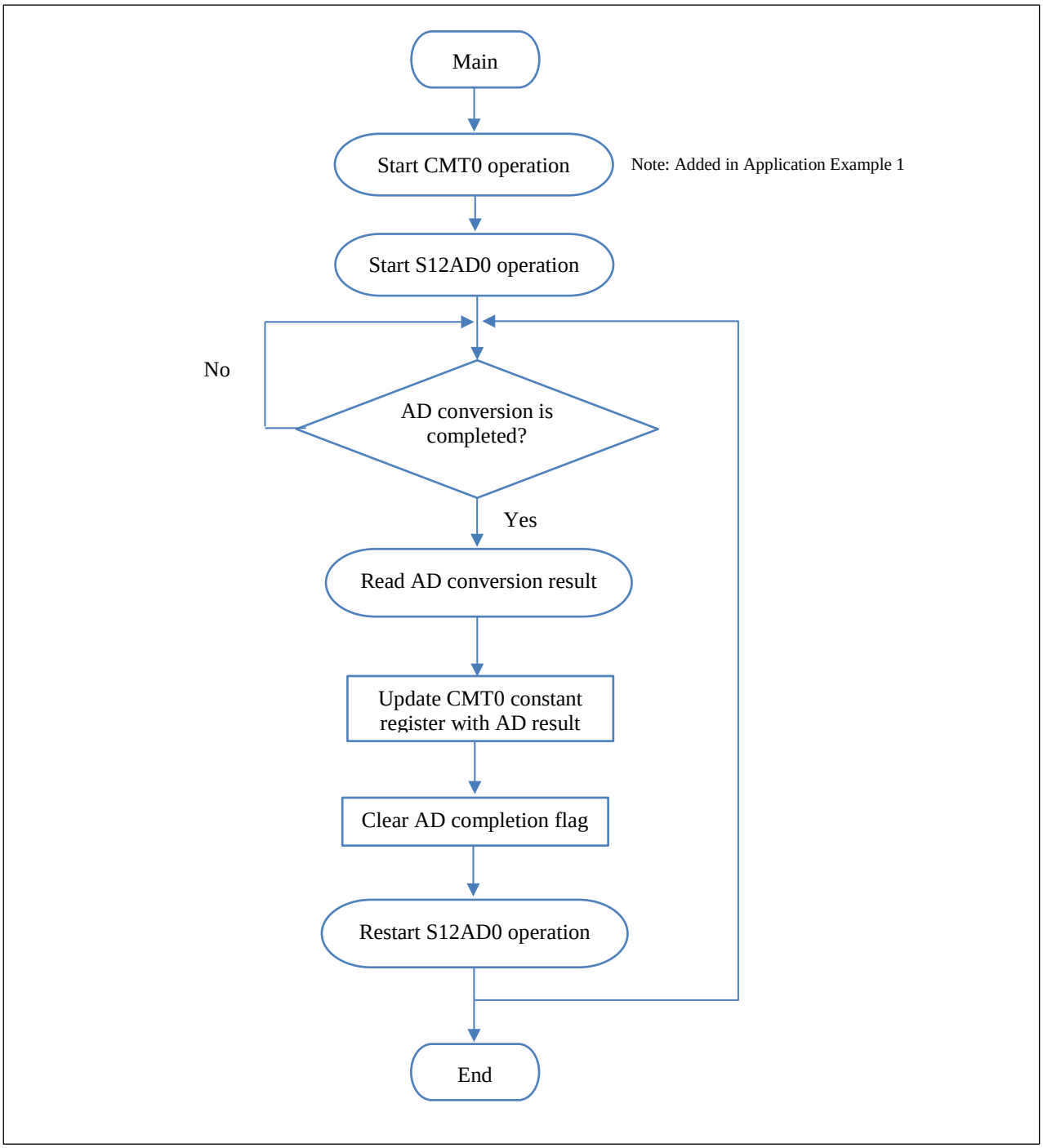


Figure 3-1 Main flowchart

b) S12AD0 user function:

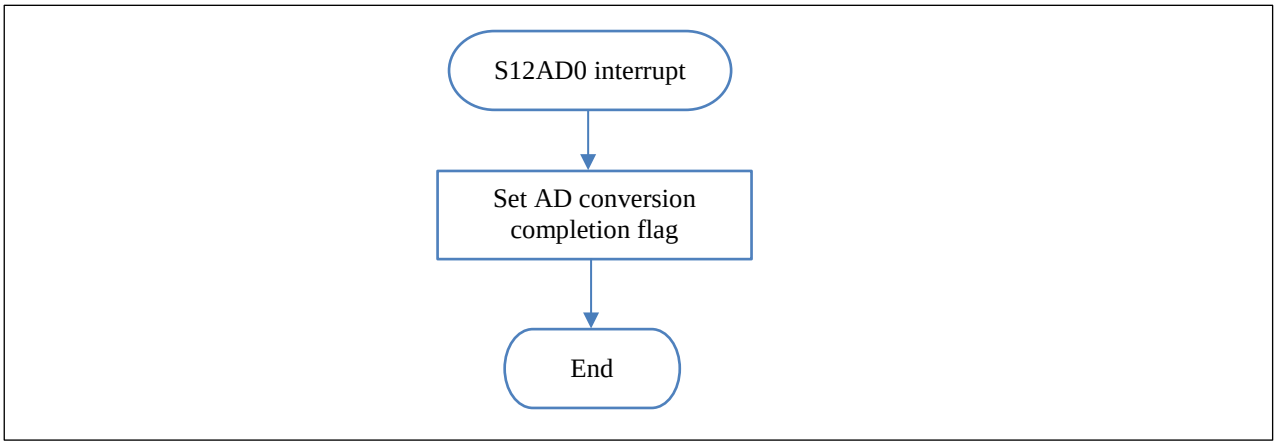


Figure 3-2 S12AD0 interrupt flowchart

3.1 Adding a peripheral driver

1) In Smart_Configurator_Example.scfg pane, click the [Components] tab
2) Click  to add new component

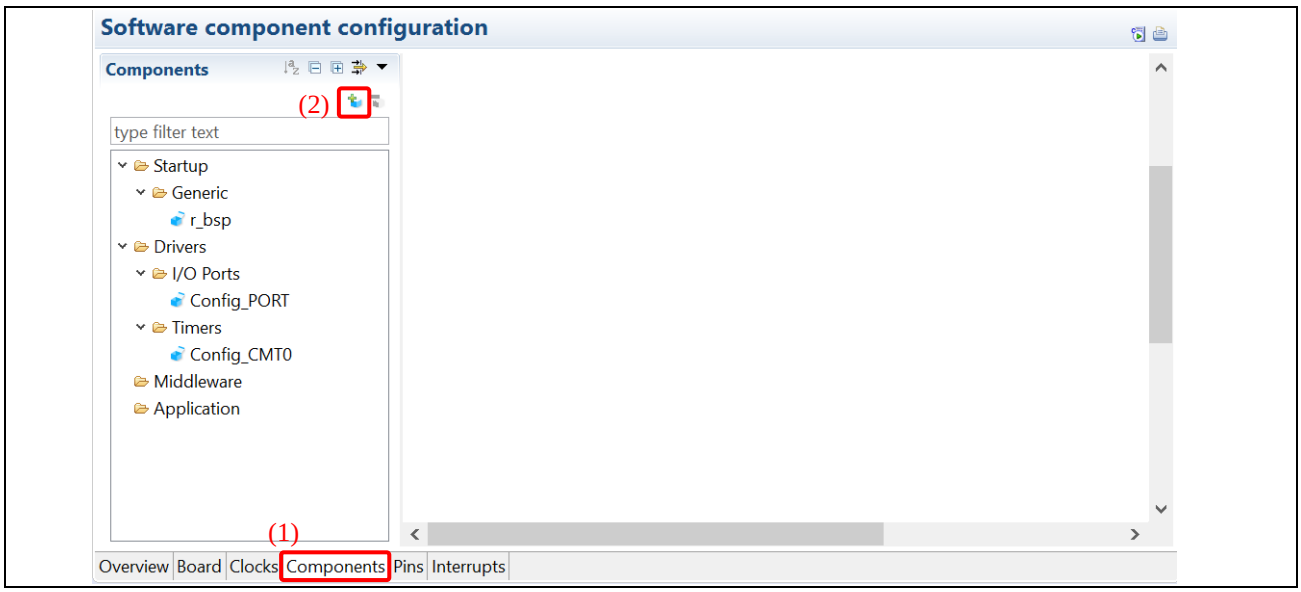


Figure 3-3 Software component configuration in Smart Configurator

3) Add *Single Scan Mode S12AD* driver into the project
   a. Select “A/D Converter (Drivers)” from the “Function” option
   b. Navigate the component list and select *Single Scan Mode S12AD*
   c. Click [Next] to continue

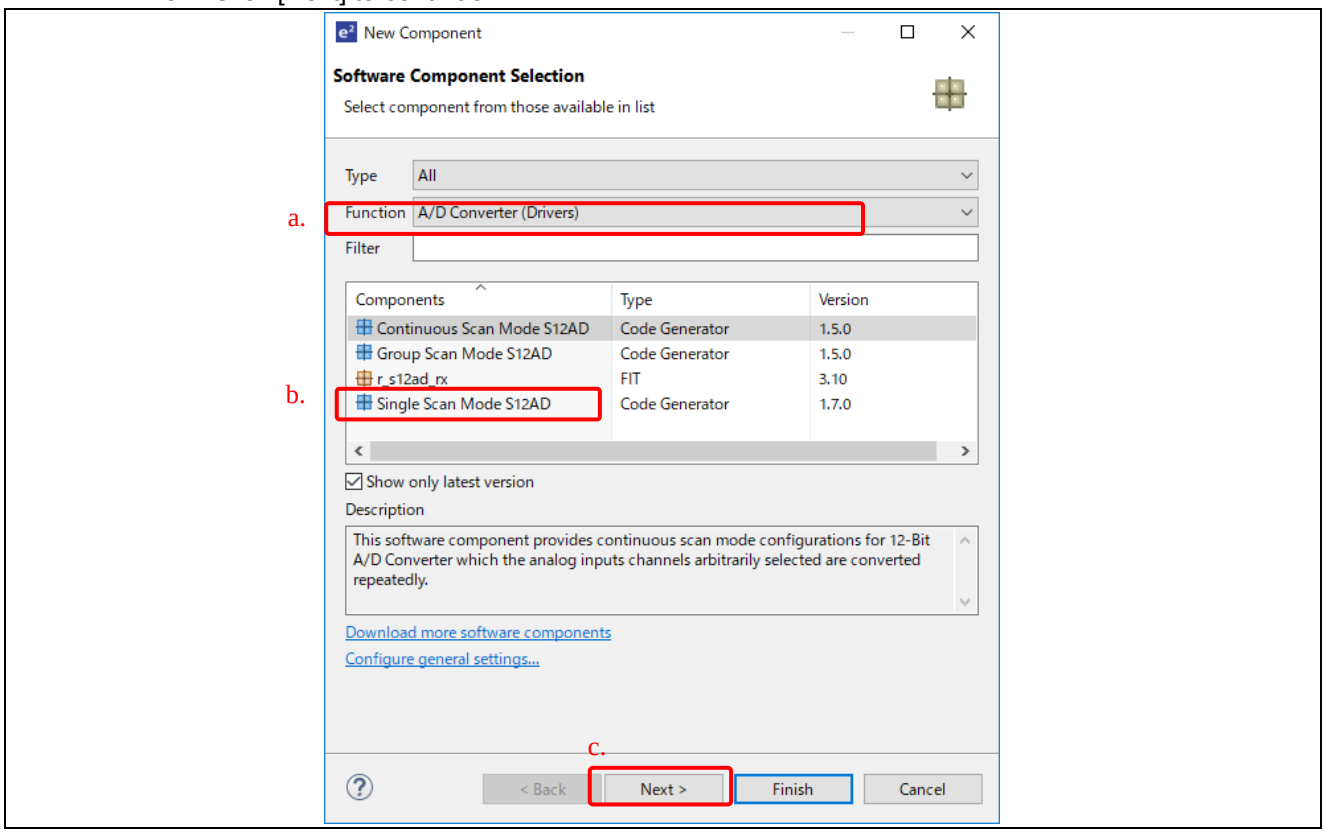


Figure 3-4 Select software component

d. Select S12AD0 as resource
e. Keep the default configuration name. Source files and API will be generated based on this configuration name
f. Click [Finish]

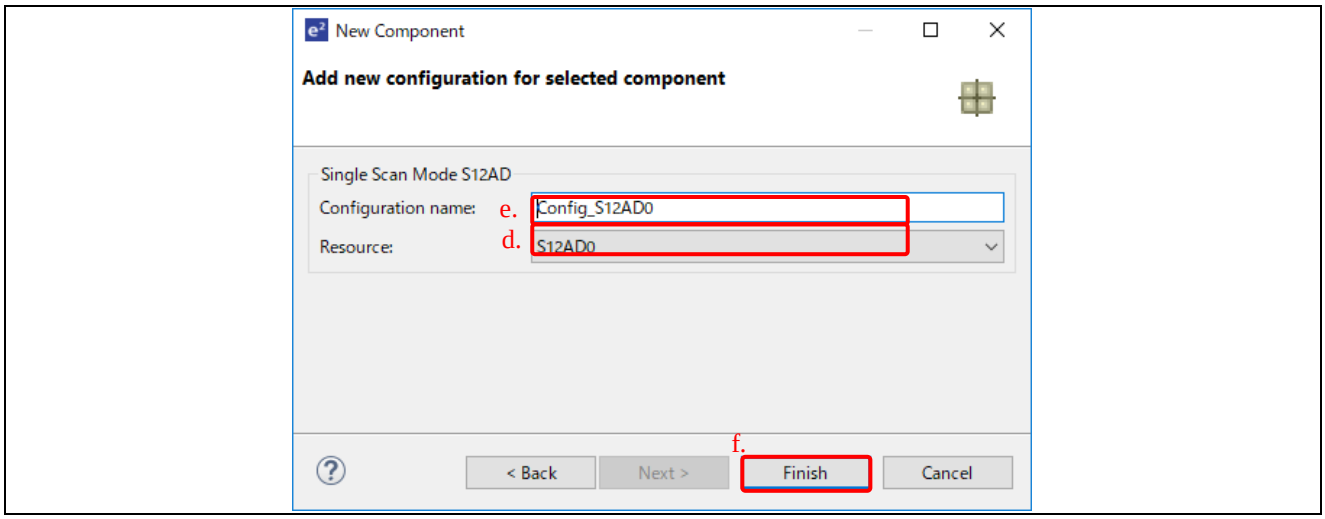


Figure 3-5 Add new configuration to the project

4) At Configure pane, select [AN000] at the Analog input channel setting

5) Ensure that the Start trigger source as [Software trigger] and the checkbox of [Enable AD conversion end interrupt (S12ADI)] is ticked

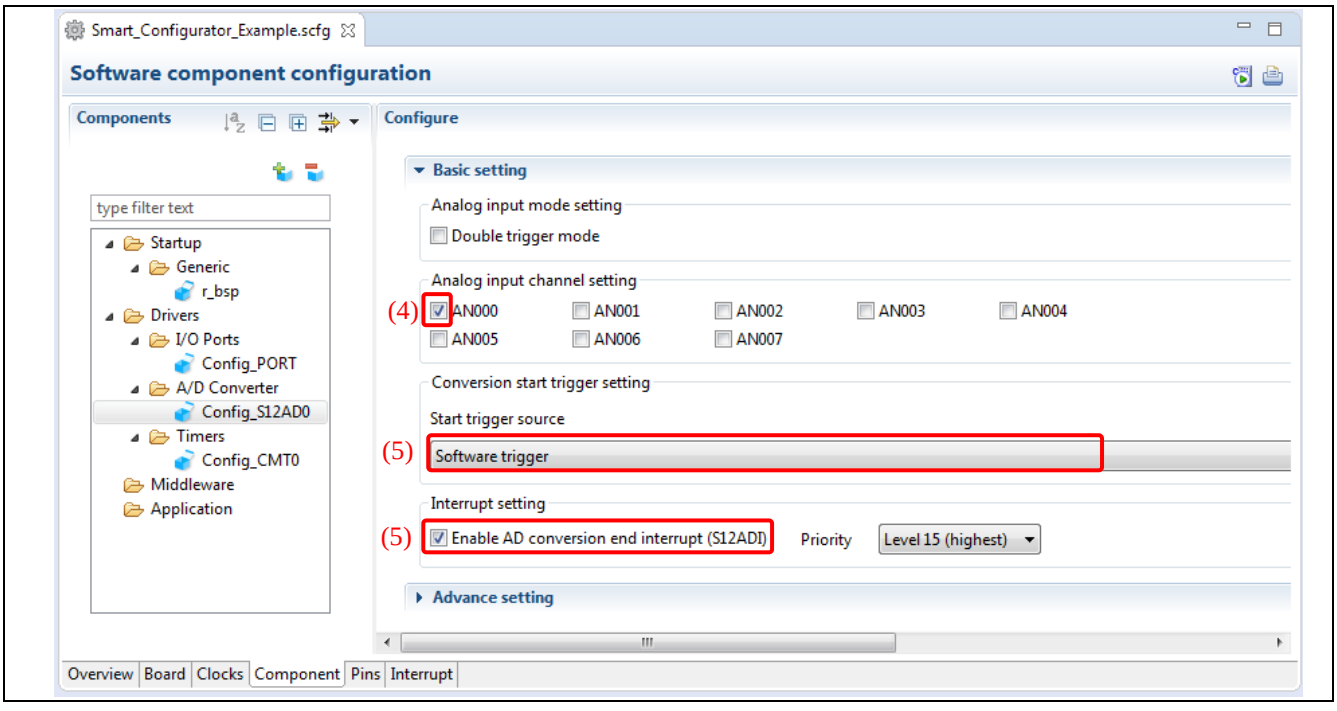


Figure 3-6 Single Scan Mode S12AD configuration

6) At [Pins] tab, click to select Config_S12AD0 from the tree
7) Clear typing in the Pin Function filter if there is any
8) Ensure AN000 pin is assigned to P40; Analog power pins (AVCC0, AVSS0, VREFH0, VREFL0) are enabled

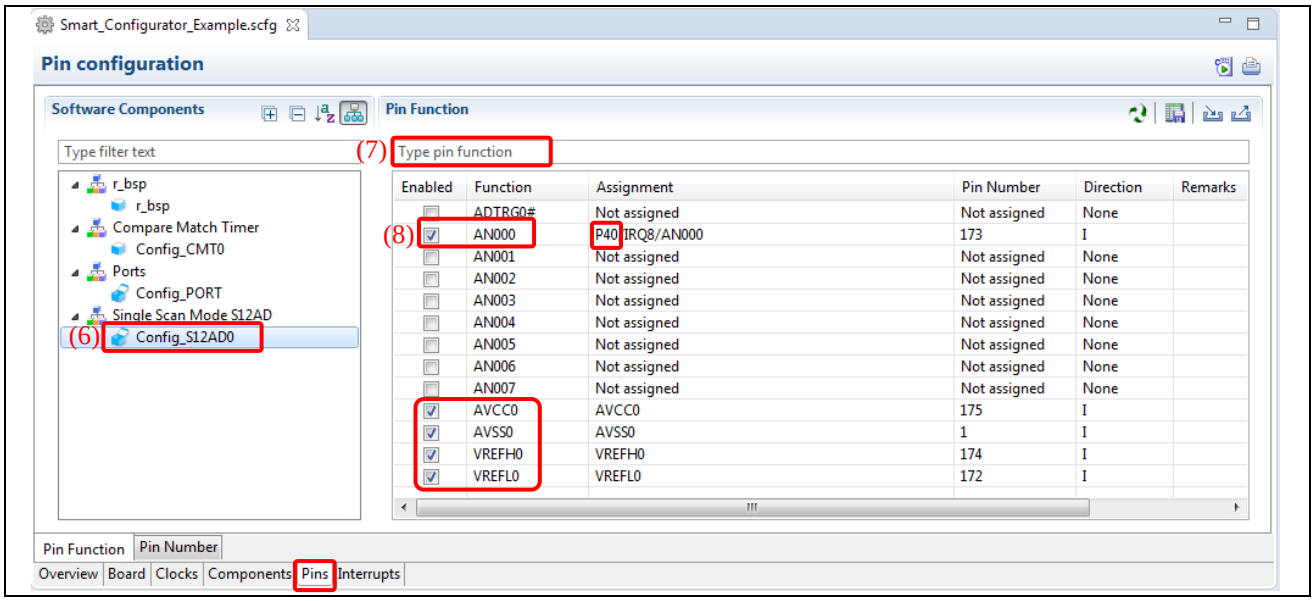


Figure 3-7 Pin configuration of *Config_S12AD0*

3.2 Generating codes

1) Click  to generate codes

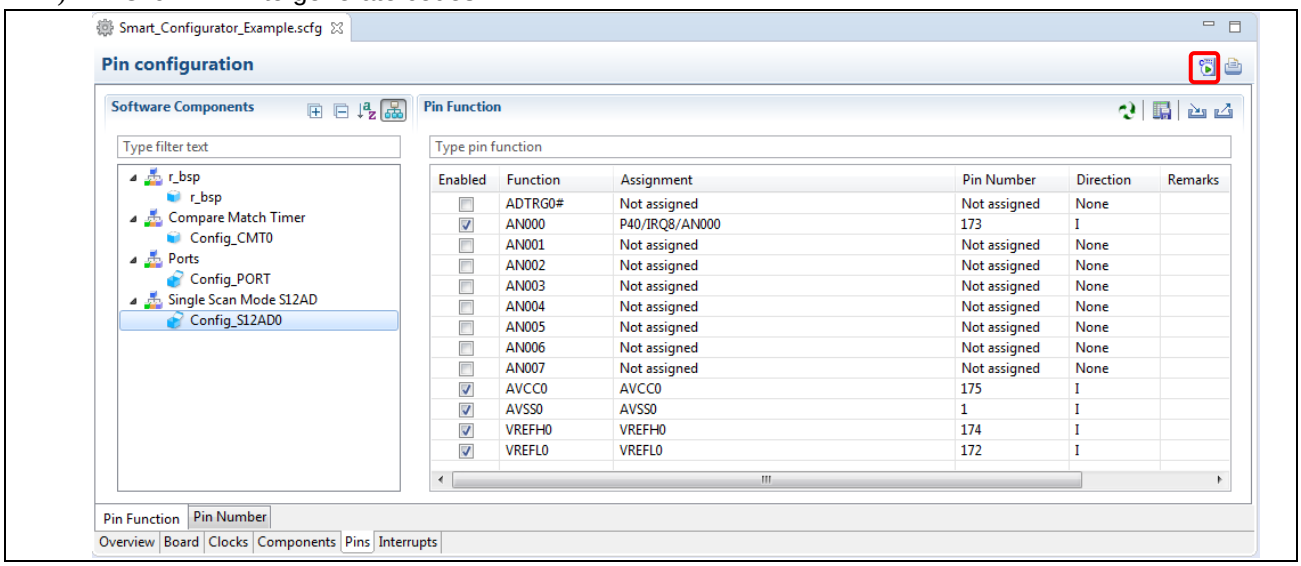


Figure 3-8 Generate codes

3.3 Adding application codes in Single Scan Mode S12AD driver

1) Open “Config_S12AD0_user.c” in \src\smc_gen Config_S12AD0 folder of the project

   a. Add declaration of global flag and variable at the user code area near the top of the file

```
/* Global flag to indicate A/D conversion operation is completed */
uint8_t g_adc_flag;
```

   b. Set A/D conversion completion flag in *r_Config_S12AD0_interrupt (void)* function

```
/* Set A/D conversion completion flag */
g_adc_flag = 1U;
```

Your source file should look like this:

```
* File Name    : Config_S12AD0_user.c

Pragma directive
/* Start user code for pragma. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

Includes
#include "r_cg_macrodriver.h"
#include "r_cg_userdefine.h"
#include "Config_S12AD0.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

Global variables and functions
/* Start user code for global. Do not edit comment generated here */
/* Global flag to indicate A/D conversion operation is completed */
uint8_t g_adc_flag;
/* End user code. Do not edit comment generated here */

* Function Name: R_Config_S12AD0_Create_UserInit

void R_Config_S12AD0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}

* Function Name: r_Config_S12AD0_interrupt

#if FAST_INTERRUPT_VECTOR == VECT_PERIB_INTB186
#pragma interrupt r_Config_S12AD0_interrupt(vect=VECT(PERIB,INTB186),fint)
#else
#pragma interrupt r_Config_S12AD0_interrupt(vect=VECT(PERIB,INTB186))
#endif
static void r_Config_S12AD0_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_interrupt. Do not edit comment generated here */
    /* Set A/D conversion completion flag */
    g_adc_flag = 1U;
    /* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
```

Figure 3-8 Config_S12AD0_user.c

3.4 Adding application codes in *main()*

1) Start CMT0 and S12AD0 operation

a. At the project tree, open “Smart_Configurator_Example.c” in \src folder

b. Add declarations of global variables at the user code area near the top of the file after

#include "r_smc_entry.h"

```
/* Global variable for changing CMT0 interval */
uint16_t interval_level;

/* Global variable for storing the A/D conversion result */
uint16_t g_adc_result;

/* Global flag to indicate A/D conversion operation is completed */
extern uint8_t g_adc_flag;
```

c. Add below code into *main()* function before the *while* loop to start S12AD0 operation:

```
/* Start performing A/D conversion */
R_Config_S12AD0_Start();
```

d. Read A/D conversion result and update into CMCOR constant register when A/D conversion is completed

Replace the *nop()* instruction in the *while* loop with the codes below:

```
/* A/D conversion is completed */
if (g_adc_flag == 1U)
{
        /* Read the A/D conversion result and store in variable */
        R_Config_S12AD0_Ge




t_ValueResult(ADCHANNEL0, &g_adc_result);

        /* Get new blink interval level from A/D conversion result */
        interval_level = g_adc_result / 500;

        /* Limit minimum level to 1 */
        if (interval_level < 1)
        {
              interval_level = 1;
        }
        else
        {
              /* Do nothing*/
        }

        /* Change blinking rate */
        CMT0.CMCOR = (uint16_t)(5000 * interval_level);

        /* Reset A/D conversion flag */
        g_adc_flag = 0U;

        /* Restart A/D conversion operation */
        R_Config_S12AD0_Start();
```

Your source file should look like this:

```
#include "r_smc_entry.h"

/* Global variable for changing CMT0 interval */
uint16_t interval_level;

/* Global variable for storing the A/D conversion result */
uint16_t g_adc_result;

/* Global flag to indicate A/D conversion operation is completed */
extern uint8_t g_adc_flag;

void main(void);


void main(void)
{
    /* Start CMT0 counter operation */
    R_Config_CMT0_Start();

    /* Start performing A/D conversion */
    R_Config_S12AD0_Start();

    while(1U)
    {
        /* A/D conversion is completed */
        if (g_adc_flag == 1U)
        {
            /* Read the A/D conversion result and store in variable */
            R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &g_adc_result);

            /* Get new blink interval level from A/D conversion result */
            interval_level = g_adc_result / 500;

            /* Limit minimum level to 1 */
            if (interval_level < 1)
            {
                interval_level = 1;
            }
            else
            {
                /* Do nothing*/
            }

            /* Change blinking rate */
            CMT0.CMCOR = (uint16_t)(5000 * interval_level);

            /* Reset A/D conversion flag */
            g_adc_flag = 0U;

            /* Restart A/D conversion operation */
            R_Config_S12AD0_Start();
        }
        else
        {
            /* Do nothing*/
        }
    }
}
```

Figure 3-9 Start CMT0 and S12AD0 operation in *main* code

3.5 Build and run on hardware board

Refer to *Chapter 2.7* to build and debug this project.

3.6 Operation on board

On the RSK+ RX65N 2MB board, LED2 blinking rate is changing when turning the potentiometer

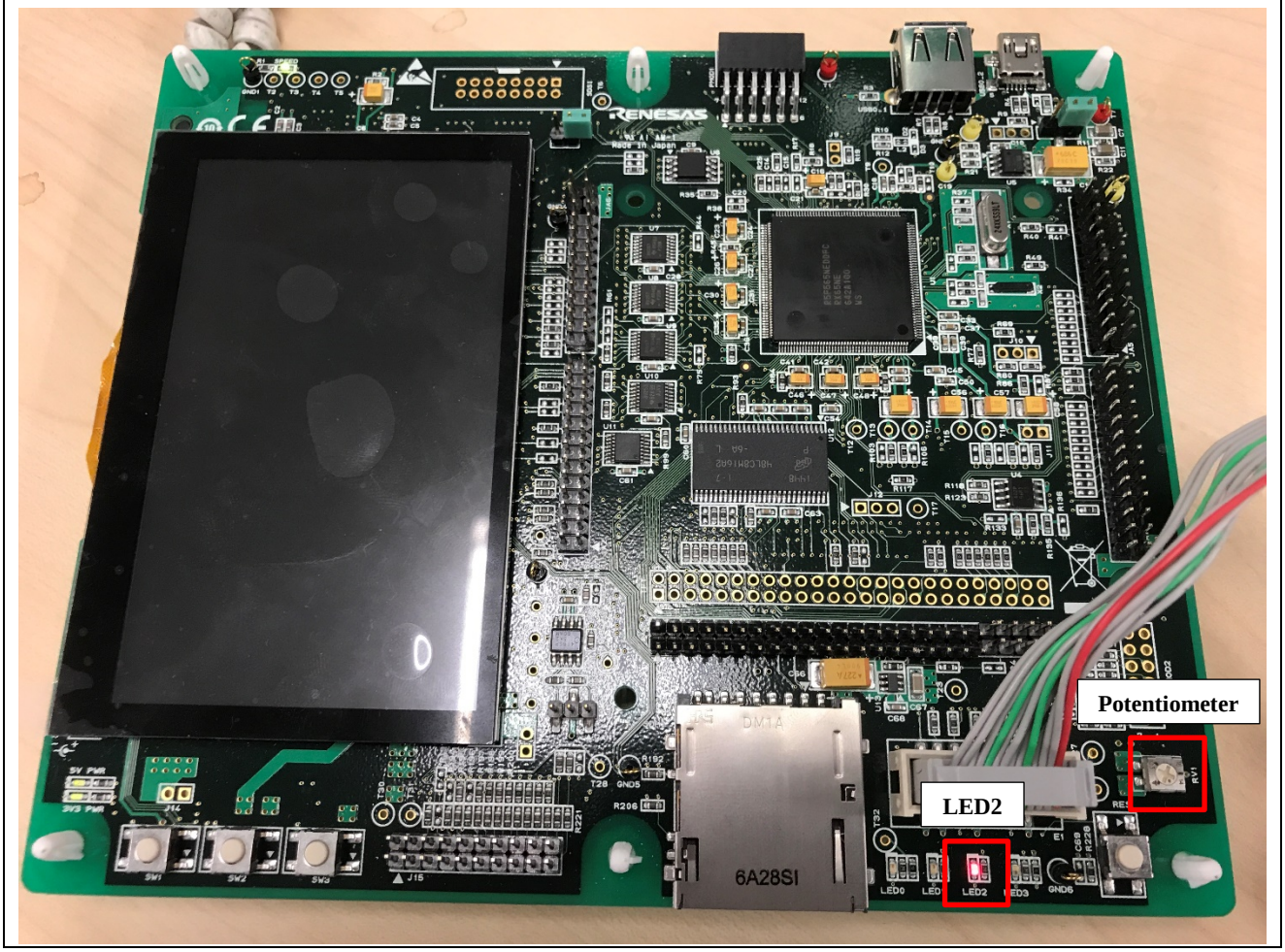


Figure 3-10 LEDs and Potentiometer on RSK+ RX65N 2MB board

4. Application Example 3 (Changing a function in Smart Configurator)

In previous chapter, potentiometer is used to control LED2 blinking rate.

This chapter describes how to control LED2 blinking rate with terminal program on PC via serial communication. In this application example, Serial Communication Interface module (SCI8) of RX65N is connected to PC via a USB-to-serial converter (implemented in RL78/G1C) on RSK+ board.

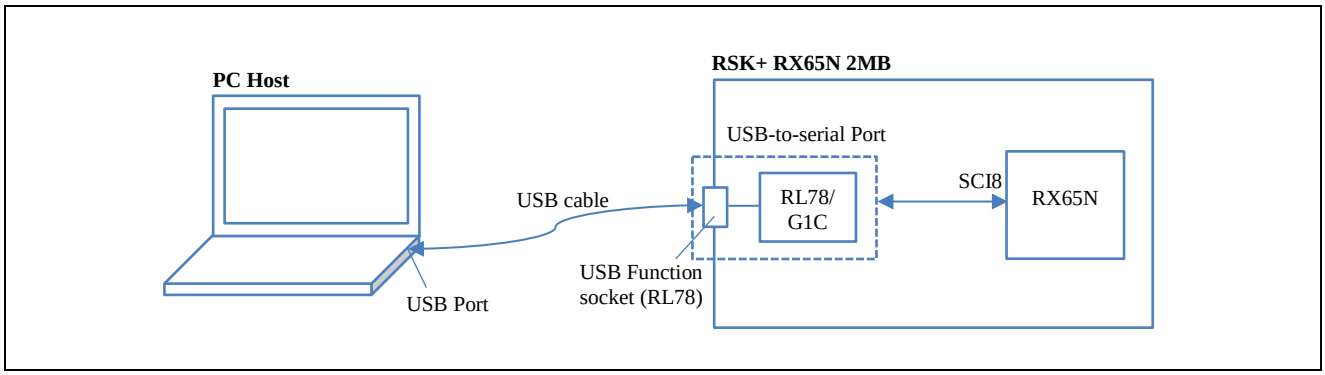


Figure 4-1 Control LED blinking rate by terminal program via serial communication

The schematic diagram of Renesas Starter Kit+ for RX65N 2MB is shown below. In this example, SCI8 will be configured to communicate with RL78/G1C.

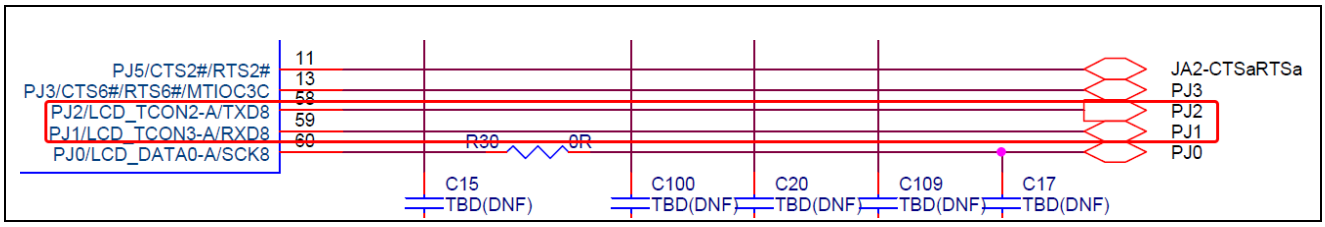


Figure 4-2 Schematic diagram of Renesas Starter Kit+ for RX65N 2MB

The table below shows changes of the driver selection:

	Drivers	Resource	Function
1. Remove driver	Single Scan Mode S12AD	S12AD	Read potentiometer value for controlling LED blinking rate
2. Add new driver	SCI/SCIF Asynchronous Mode	SCI8	Communication between RX65N and PC terminal via an on-board USB-to-serial converter (RL78/G1C) Data received from PC terminal is used to control LED blinking rate

SCI/SCIF Asynchronous Mode application codes will be added to the program as below:

a) Main function:

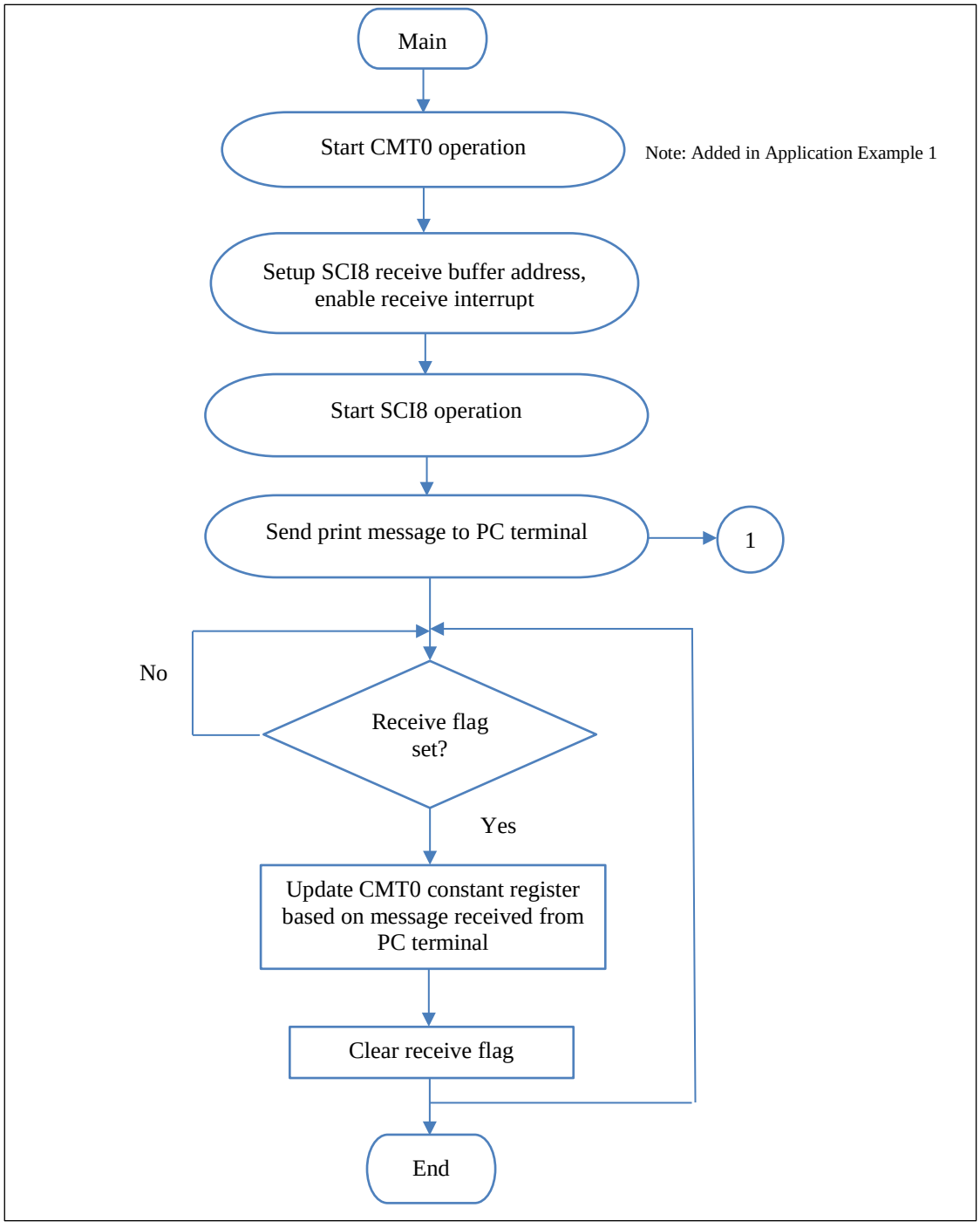


Figure 4-3 Main flowchart

b) SCI8 user functions:

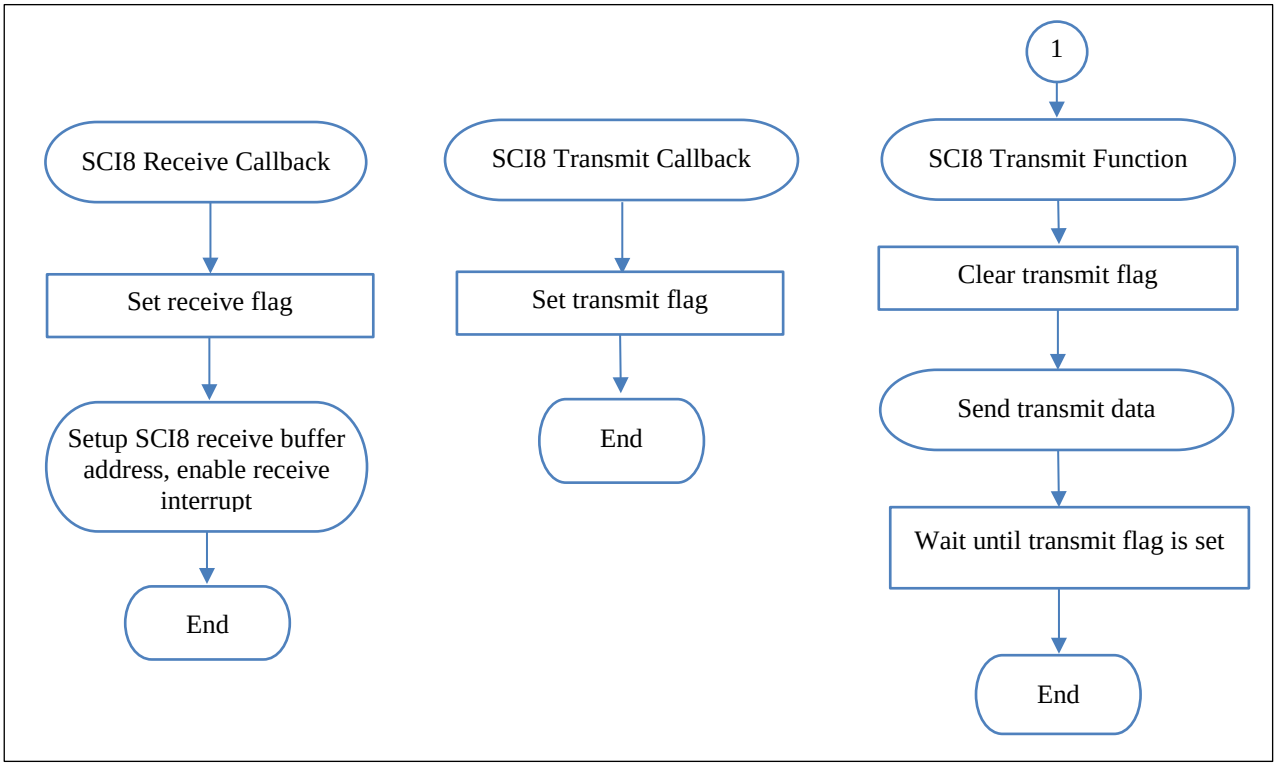


Figure 4-4 SCI user functions flowchart

4.1 Removing S12AD driver and related application codes

1) In Smart_Configurator_Example.scfg pane, click the [Components] tab
2) At Components tree, click to select *Config_S12AD0* configuration
3) Click  to remove this configuration

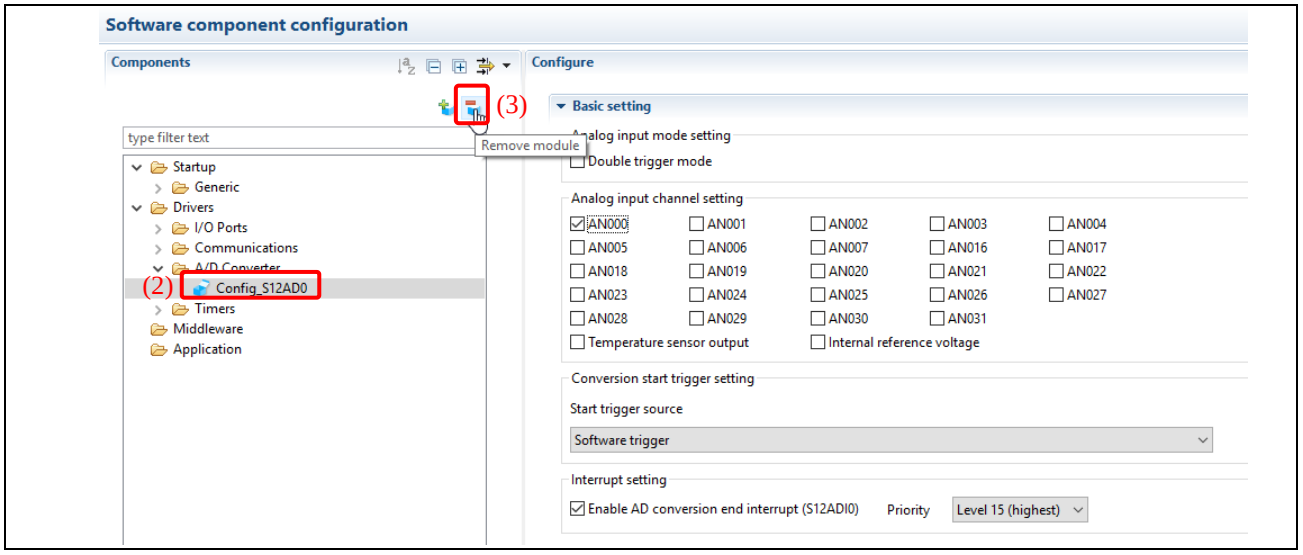


Figure 4-5 Software component configuration in Smart Configurator

4) Click  to generate codes
The source folder *Config_S12AD0* will be moved to project \trash folder before the code generation operation
5) At the project tree, open “Smart_Configurator_Example.c” in \src folder
6) Remove the application codes which are related to S12AD0
   a. Remove declarations for global variables

```
/* Global variable for changing CMT0 interval */
uint16_t interval_level;

/* Global variable for storing the A/D conversion result */
uint16_t g_adc_result;

/* Global flag to indicate A/D conversion operation is completed */
extern uint8_t g_adc_flag;
```

   b. Remove all codes after *R_Config_CMT0_Start* function

   After deleted the codes related to S12AD0, the main file will look like following:

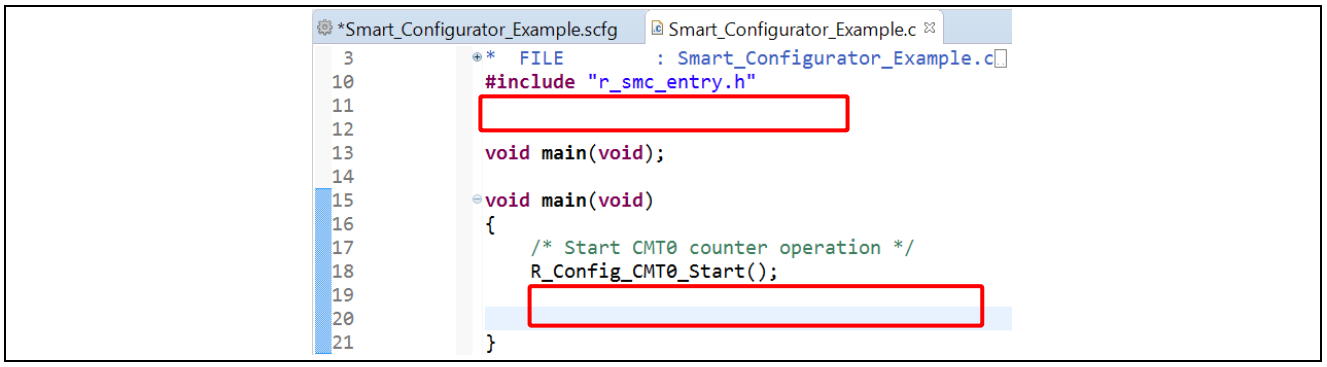


Figure 4-6 Main file after deleted codes related to S12AD0

4.2 Adding a peripheral driver

1) In the Smart_Configurator_Example.scfg pane, click [Components] tab
2) Click  to add new component

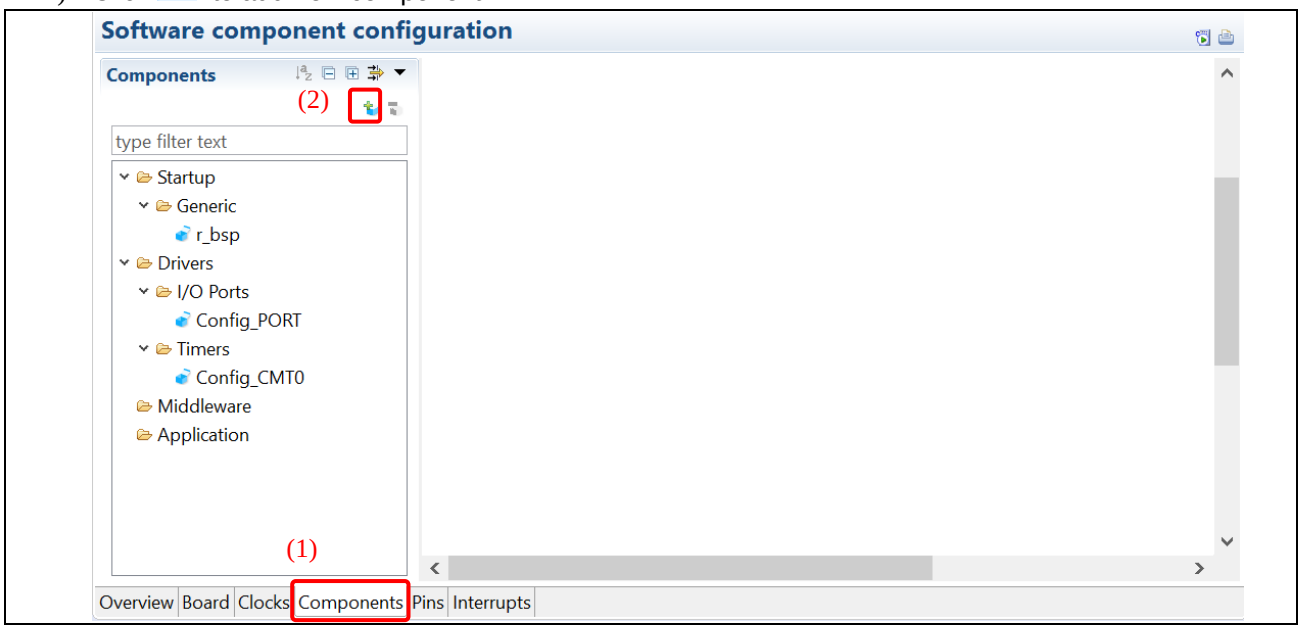


Figure 4-7 Software component configuration in Smart Configurator

3) Add *Serial Communication Interface* driver into the project
   - Select “Communications (Drivers)” from the “Function” option
   - Navigate the component list and select *SCI/SCIF Asynchronous Mode* driver
   - Click [Next] to continue

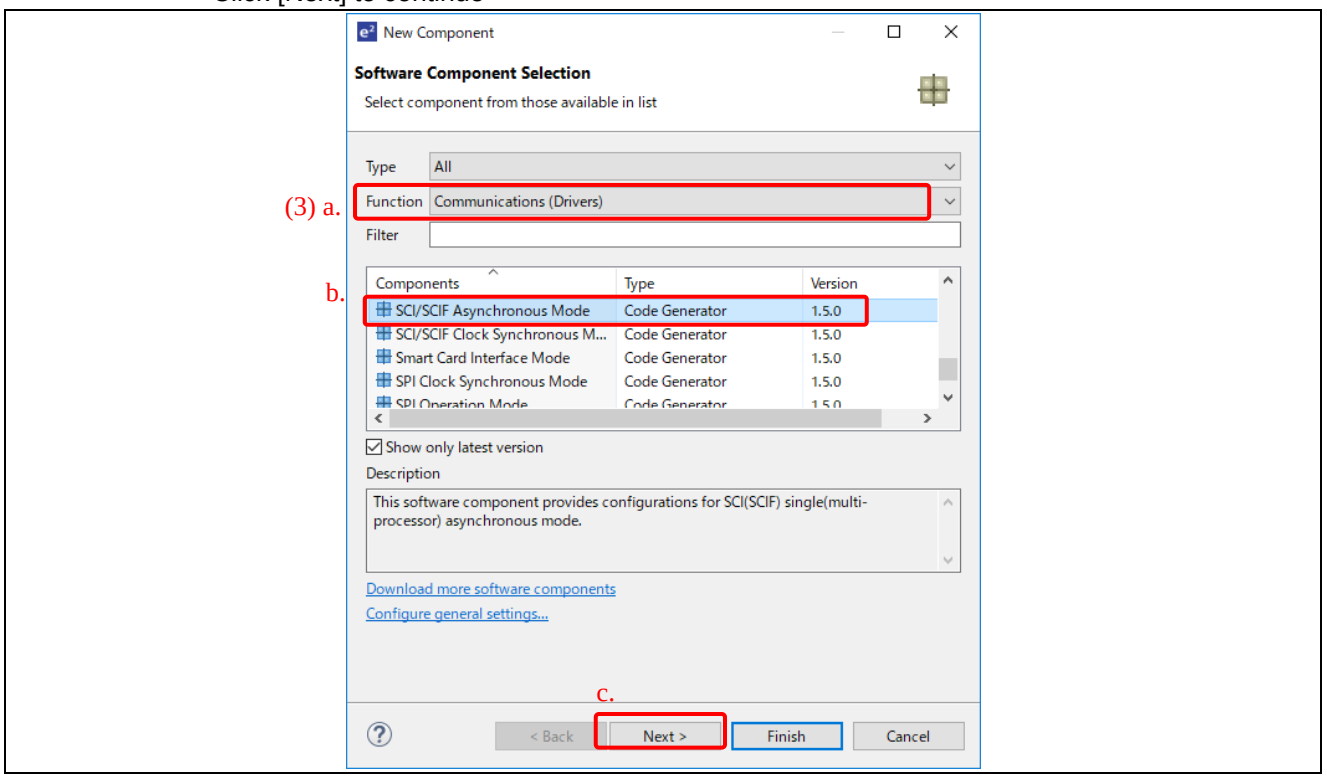


Figure 4-8 Select software component

- Select SCI8 as resource
- Select Work mode as “Transmission/Reception”
- Keep the default configuration name. Source files and API will be generated based on this configuration name
- Click [Finish]

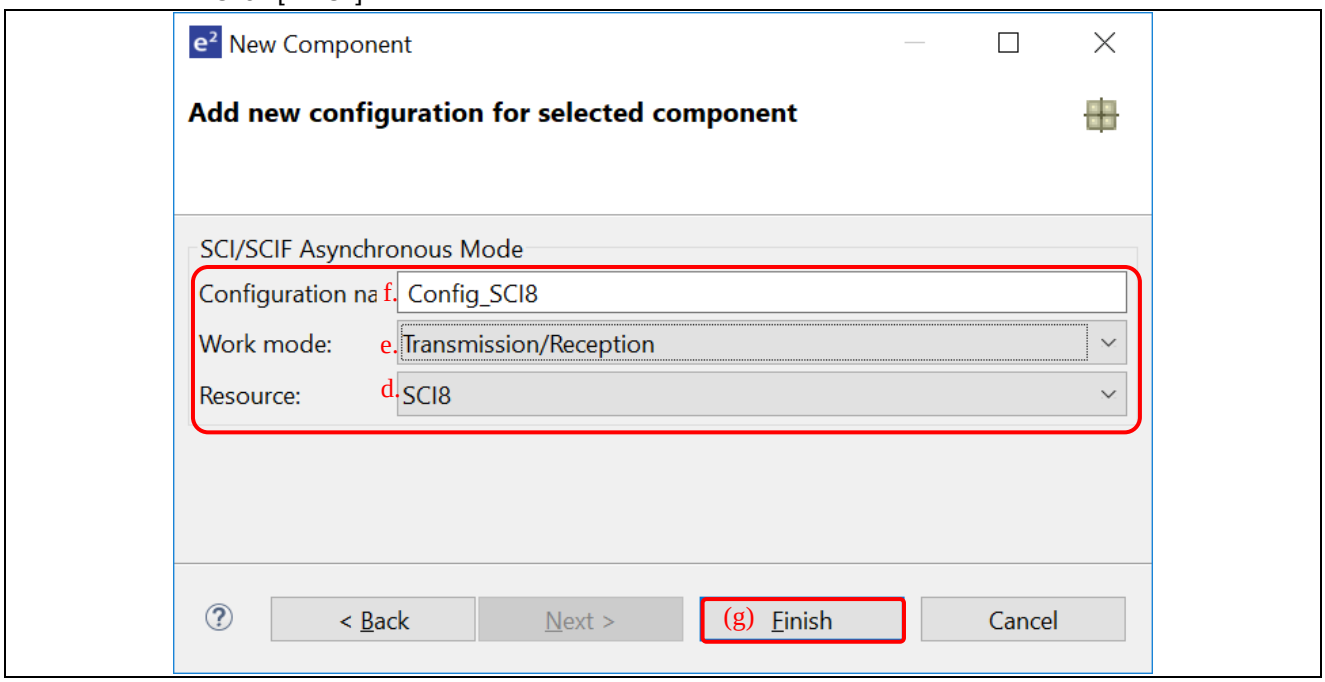


Figure 4-9 Add new configuration to the project

4) At Configure pane, ensure settings as the following:
   - Start bit edge detection setting: Falling edge on RXD8 pin
   - Data length setting: 8 bits
   - Parity setting: None
   - Stop bit length setting: 1 bit
   - Bit rate: 9600 bps

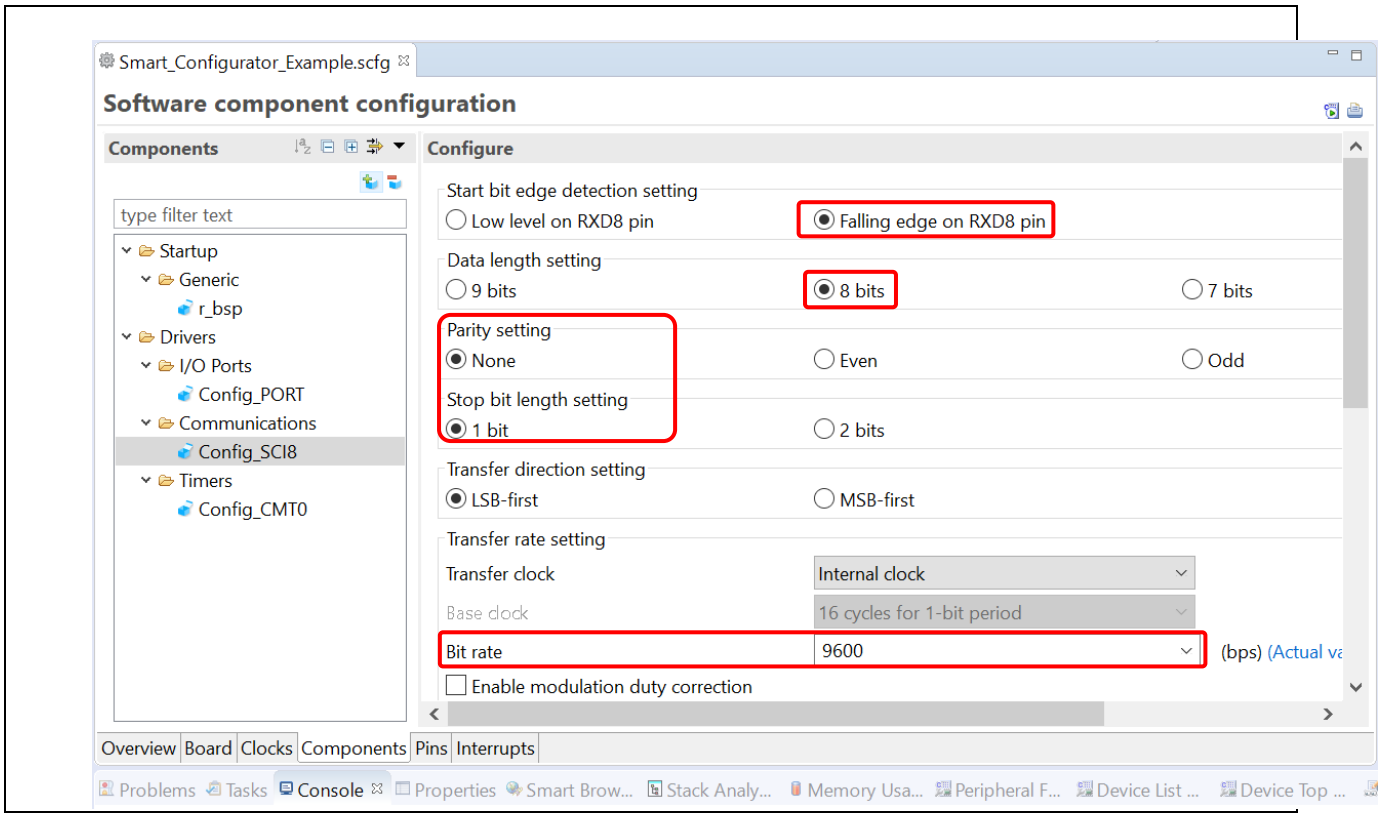


Figure 4-10 SCI/SCIF Asynchronous Mode driver configuration

5) At [Pins] tab, click  button to switch tree to Software Components view

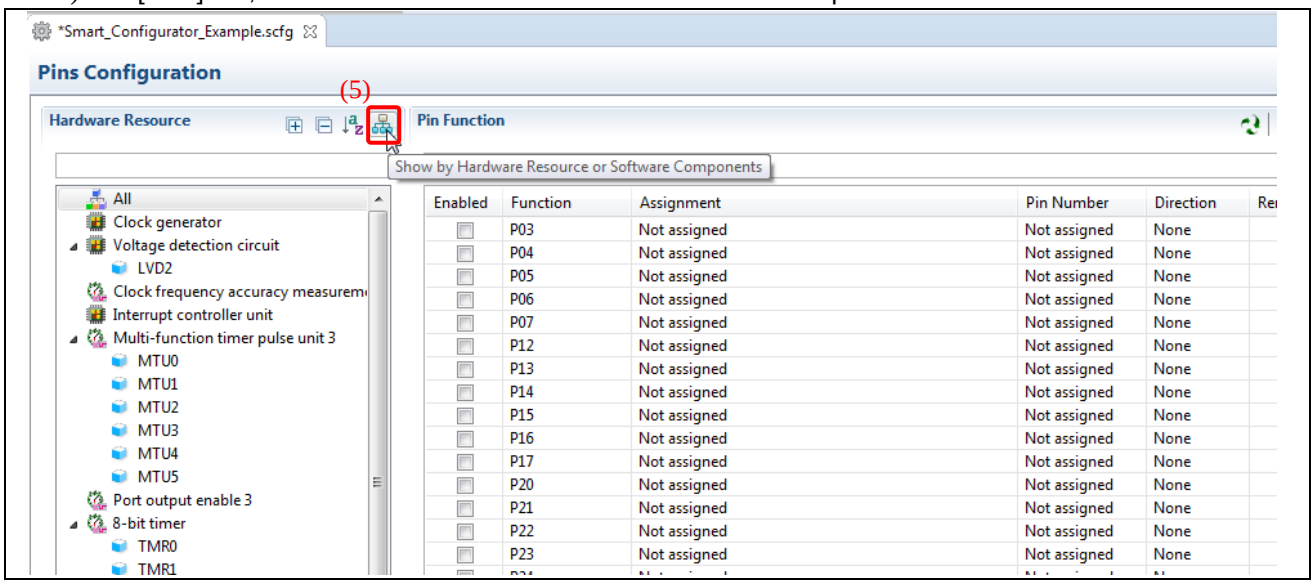


Figure 4-11 Pins tab

6) At the tree, click Config_SCI8 to view pin configurations
7) Ensure RXD8 is assigned to PJ1 and TXD8 is assigned to PJ2:

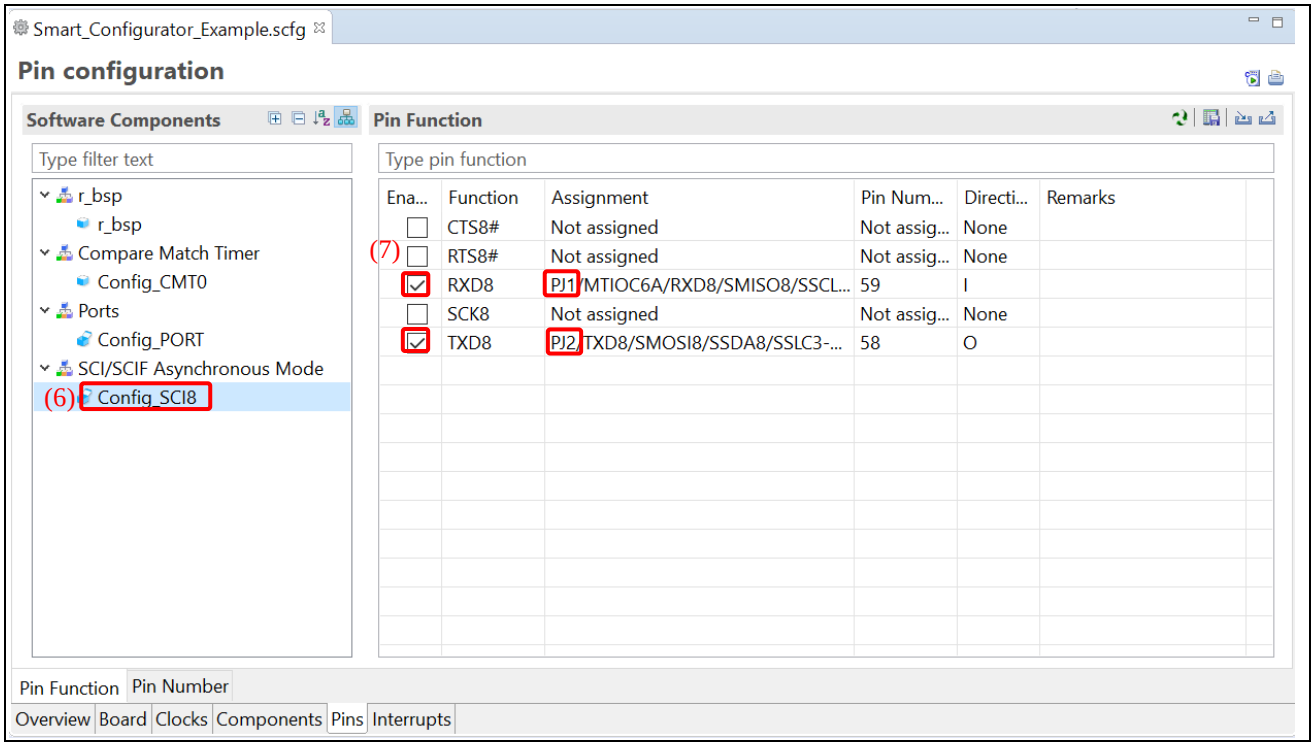


Figure 4-12 Pins configuration of *Config_SCI8*

4.3 Generating codes

1) At [Software Component Configuration] tab, click  to generate codes

4.4 Adding application codes in SCI/SCIF Asynchronous Mode driver

1) At Project Explore tree, open “Config_SCI8.h” in \src\smc_gen\Config_SCI8 folder. Add the declarations below at the user code area at the bottom of the file:

```
/* Sends SCI8 data and waits for transmit end flag */
MD_STATUS R_SCI8_AsyncTransmit(uint8_t * const tx_buf, const uint16_t
tx_num);
```

2) At Project Explore tree, open “Config_SCI8_user.c” in \src\smc_gen\Config_SCI8 folder

   a. Add the declarations below at the user code area near the top of the file:

```
/* Global variable used to receive a character from PC terminal */
volatile uint8_t g_rx_char;

/* Flag used to detect whether data is received */
volatile uint8_t g_rx_flag;

/* Flag used to detect completion of transmission */
static volatile uint8_t SCI8_txdone;
```

   b. Set the transmission completion flag in *r_Config_SCI8_callback_transmitend (void)* function

```
/* Set transmit completion flag */
SCI8_txdone = 1U;
```

   c. Set the reception completion flag in *r_Config_SCI8_callback_receiveend (void)* function and restart reception

```
/* Set receive completion flag */
g_rx_flag = 1U;

/* Set SCI8 receive buffer address and restart reception */
R_Config_SCI8_Serial_Receive((uint8_t *)&g_rx_char, 1);
```

d. Add a function to send SCI8 data and waits for the transmit end flag at user code area at the bottom of the file

```
/***********************************************************************
*********
* Function Name: R_SCI8_AsyncTransmit
* Description  : This function sends SCI8 data and waits for the
transmit end flag
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************
********/
MD_STATUS R_SCI8_AsyncTransmit (uint8_t * const tx_buf, const uint16_t
tx_num)
{
    MD_STATUS status = MD_OK;

    /* Clear transmit completion flag before transmission */
    SCI8_txdone = 0U;

      /* Set SCI8 transmit buffer address and start transmission */
    status = R_Config_SCI8_Serial_Send(tx_buf, tx_num);

    /* Wait for transmit end flag */
    while (0U == SCI8_txdone)
    {
        /* Wait */
    }
    return (status);
}
/***********************************************************************
*********
* End of function R_SCI8_AsyncTransmit
      ***********************************************************************
**************/
```

Your source file should look like this:

```
* File Name     : Config_SCI8_user.c

Pragma directive
/* Start user code for pragma. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

Includes
#include "r_cg_macrodriver.h"
#include "Config_SCI8.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

Global variables and functions
extern volatile uint8_t * gp_sci8_tx_address;               /* SCI8 transmit buffer address */
extern volatile uint16_t  g_sci8_tx_count;                  /* SCI8 transmit data number */
extern volatile uint8_t * gp_sci8_rx_address;               /* SCI8 receive buffer address */
extern volatile uint16_t  g_sci8_rx_count;                  /* SCI8 receive data number */
extern volatile uint16_t  g_sci8_rx_length;                 /* SCI8 receive data length */
/* Start user code for global. Do not edit comment generated here */
/* Global variable used to receive a character from PC terminal */
volatile uint8_t g_rx_char;

/* Flag used to detect whether data is received */
volatile uint8_t g_rx_flag;

/* Flag used to detect completion of transmission */
static volatile uint8_t SCI8_txdone;
/* End user code. Do not edit comment generated here */
```

```
/***********************************************************************************************************
* Function Name: r_Config_SCI8_callback_transmitend
* Description  : This function is a callback function when SCI8 finishes transmission
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
static void r_Config_SCI8_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI8_callback_transmitend. Do not edit comment generated here */
    /* Set transmit completion flag */
    SCI8_txdone = 1U;
    /* End user code. Do not edit comment generated here */
}

/***********************************************************************************************************
* Function Name: r_Config_SCI8_callback_receiveend
* Description  : This function is a callback function when SCI8 finishes reception
* Arguments    : None
* Return Value : None
***********************************************************************************************************/
static void r_Config_SCI8_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI8_callback_receiveend. Do not edit comment generated here */
    /* Set receive completion flag */
    g_rx_flag = 1U;

    /* Set SCI8 receive buffer address and restart reception */
    R_Config_SCI8_Serial_Receive((uint8_t *)&g_rx_char, 1);

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

```
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************
* Function Name: R_SCI8_AsyncTransmit
* Description  : This function sends SCI8 data and waits for the transmit end flag
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************************/
MD_STATUS R_SCI8_AsyncTransmit (uint8_t * const tx_buf, const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* Clear transmit completion flag before transmission */
    SCI8_txdone = 0U;

    /* Set SCI8 transmit buffer address and start transmission */
    status = R_Config_SCI8_Serial_Send(tx_buf, tx_num);

    /* Wait for transmit end flag */
    while (0U == SCI8_txdone)
    {
        /* Wait */
    }
    return (status);
}
/***********************************************************************************
* End of function R_SCI8_AsyncTransmit
***********************************************************************************/

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

Figure 4-13 Config_SCI8_user.c

4.5 Adding application codes in *main()*

1) In Smart_Configurator_Example.c, add header files and declarations near the top of the file

```
#include <stdio.h>
#include <string.h>

/* Global variable for changing CMT0 interval */
volatile uint16_t interval_level = 5;

/* String used to print message at PC terminal */
static char print_str[80];

/* Flag used to detect whether data is received from PC terminal */
extern volatile uint8_t g_rx_flag;

/* Global variable used for storing data received from PC terminal */
extern volatile uint8_t g_rx_char;
```

2) Add below codes into *main()* function after *R_Config_CMT0_Start()* to start communication with terminal program:

```
/* Set SCI8 receive buffer address and enable receive interrupt */
R_Config_SCI8_Serial_Receive((uint8_t *)&g_rx_char, 1);

/* Enable SCI8 operation */
R_Config_SCI8_Start();

/* Print instruction onto PC terminal */
sprintf(print_str, "\r\n Press keyboard to control the blinking rate of LED2 (not case sensitive)\r\n");
R_SCI8_AsyncTransmit((uint8_t *)print_str, (uint16_t)strlen(print_str));
sprintf(print_str, "\r\n a ----> Slower\n");
R_SCI8_AsyncTransmit((uint8_t *)print_str, (uint16_t)strlen(print_str));
sprintf(print_str, "\r\n b ----> Faster\n");
R_SCI8_AsyncTransmit((uint8_t *)print_str, (uint16_t)strlen(print_str));

while (1U)
{
        /* Get new blink interval level from PC terminal */
        if (g_rx_flag == 1U)
        {
                /* Instruction to slow down blinking rate is received */
                if (('A' == g_rx_char) || ('a' == g_rx_char))
                {
                        R_Config_SCI8_Serial_Receive((uint8_t *)&g_rx_char, 1);

                        /* Notify the character received and inform next action */
                        sprintf(print_str, "\r\n Character 'a' or 'A' is received. LED2 will blink at
                slower rate.\r\n");
                        R_SCI8_AsyncTransmit((uint8_t *)print_str, (uint16_t)strlen(print_str));

                        /* Get new blink interval level. Maximum level is 9. */
                        if (interval_level < 10)
                        {
                                interval_level++;
                        }
                        else
                        {
                                sprintf(print_str, "\r\n This is minimum. Please press 'b' or 'B' to
                        blink faster. \r\n");
                                R_SCI8_AsyncTransmit((uint8_t *)print_str,
                        (uint16_t)strlen(print_str));
                        }
                }
                /* Instruction to increase blinking rate is received */
                else if (('B' == g_rx_char) || ('b' == g_rx_char))
                {
                        R_Config_SCI8_Serial_Receive((uint8_t *)&g_rx_char, 1);

                        /* Notify the character received and inform next action */
                        sprintf(print_str, "\r\n Character 'b' or 'B' is received. LED2 will blink at
                faster rate.\r\n");
                        R_SCI8_AsyncTransmit((uint8_t *)print_str, (uint16_t)strlen(print_str));

                        /* Get new blink interval level. Minimum level is 1 */
                        if (interval_level > 1)
                        {
                                interval_level--;
                        }
                        else
                        {
                                sprintf(print_str, "\r\n This is maximum. Please press 'a' or 'A' to
                        blink slower. \r\n");
                                R_SCI8_AsyncTransmit((uint8_t *)print_str,
                        (uint16_t)strlen(print_str));
                        }
                }
                else
                {
                        /* Do nothing */
                }

                /* Change blinking rate */
                CMT0.CMCOR = (uint16_t)(5000 * interval_level);

                /* Reset SCI8 reception flag*/
                g_rx_flag = 0U;
        }
        else
        {
        /* Do nothing */
        }
}
```

Your source file should look like this:

```
 *  FILE        : Smart_Configurator_Example.c
#include "r_smc_entry.h"
#include <stdio.h>
#include <string.h>

/* Global variable for changing CMT0 interval */
volatile uint16_t interval_level = 5;

/* String used to print message at PC terminal */
static char print_str[80];

/* Flag used to detect whether data is received from PC terminal */
extern volatile uint8_t g_rx_flag;

/* Global variable used for storing data received from PC terminal */
extern volatile uint8_t g_rx_char;
```

```
void main(void);

void main(void)
{
    /* Start CMT0 counter operation */
    R_Config_CMT0_Start();

    /* Set SCI8 receive buffer address and enable receive interrupt */
    R_Config_SCI8_Serial_Receive((uint8_t *) &g_rx_char, 1);

    /* Enable SCI8 operation */
    R_Config_SCI8_Start();

    /* Print instruction onto PC terminal */
    sprintf(print_str, "\r\n Press keyboard to control the blinking rate of LED2 (not case sensitive)\r\n");
    R_SCI8_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
    sprintf(print_str, "\r\n a ----> Slower\n");
    R_SCI8_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
    sprintf(print_str, "\r\n b ----> Faster\n");
    R_SCI8_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

    while (1U)
    {
        /* Get new blink interval level from PC terminal */
        if (g_rx_flag == 1U)
        {
            /* Instruction to slow down blinking rate is received */
            if (('A' == g_rx_char) || ('a' == g_rx_char))
            {
                R_Config_SCI8_Serial_Receive((uint8_t *) &g_rx_char, 1);

                /* Notify the character received and inform next action */
                sprintf(print_str, "\r\n Character 'a' or 'A' is received. LED2 will blink at slower rate.\r\n");
                R_SCI8_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

                /* Get new blink interval level. Maximum level is 9. */
                if (interval_level < 10)
                {
                    interval_level++;
                }
                else
                {
                    sprintf(print_str, "\r\n This is minimum. Please press 'b' or 'B' to blink faster. \r\n");
                    R_SCI8_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
                }
            }
```

```
            /* Instruction to increase blinking rate is received */
            else if (('B' == g_rx_char) || ('b' == g_rx_char)) {
                R_Config_SCI8_Serial_Receive((uint8_t *) &g_rx_char, 1);

                /* Notify the character received and inform next action */
                sprintf(print_str,
                        "\r\n Character 'b' or 'B' is received. LED2 will blink at faster rate.\r\n");
                R_SCI8_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

                /* Get new blink interval level. Minimum level is 1 */
                if (interval_level > 1)
                {
                    interval_level--;
                }
                else
                {
                    sprintf(print_str, "\r\n This is maximum. Please press 'a' or 'A' to blink slower. \r\n");
                    R_SCI8_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
                }
            } else
            {
                /* Do nothing */
            }

            /* Change blinking rate */
            CMT0.CMCOR = (uint16_t) (5000 * interval_level);

            /* Reset SCI8 reception flag*/
            g_rx_flag = 0U;
        }
        else
        {
            /* Do nothing */
        }
    }
}
```

Figure 4-14 Asynchronous application codes in *main* function

4.6 Build and run on hardware board

After build the project, perform the following steps before debug the project.

Step1. Use a USB to mini B cable to connect PC COM port and USB Serial connector (G1CUSB0) on Renesas Starter Kit+ for RX65N 2MB (refer to Chapter 4.7 for the serial port location)

Step 2. Open a terminal program in PC (for e.g. Tera Term). Perform the following steps:

- Select USB Serial Device

  Note: Port name may vary depends on driver in PC

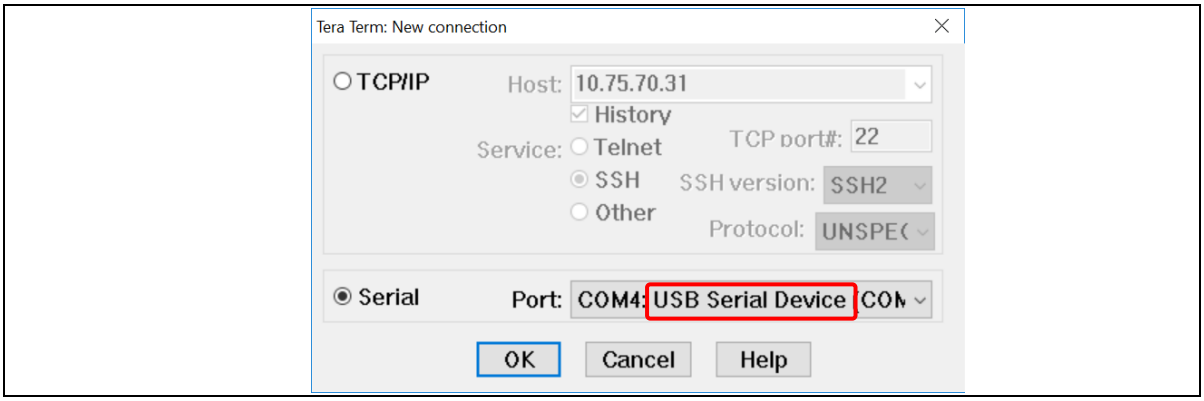


Figure 4-15 USB Serial Port

- Click [Setup] from the menu and select [Serial port…]

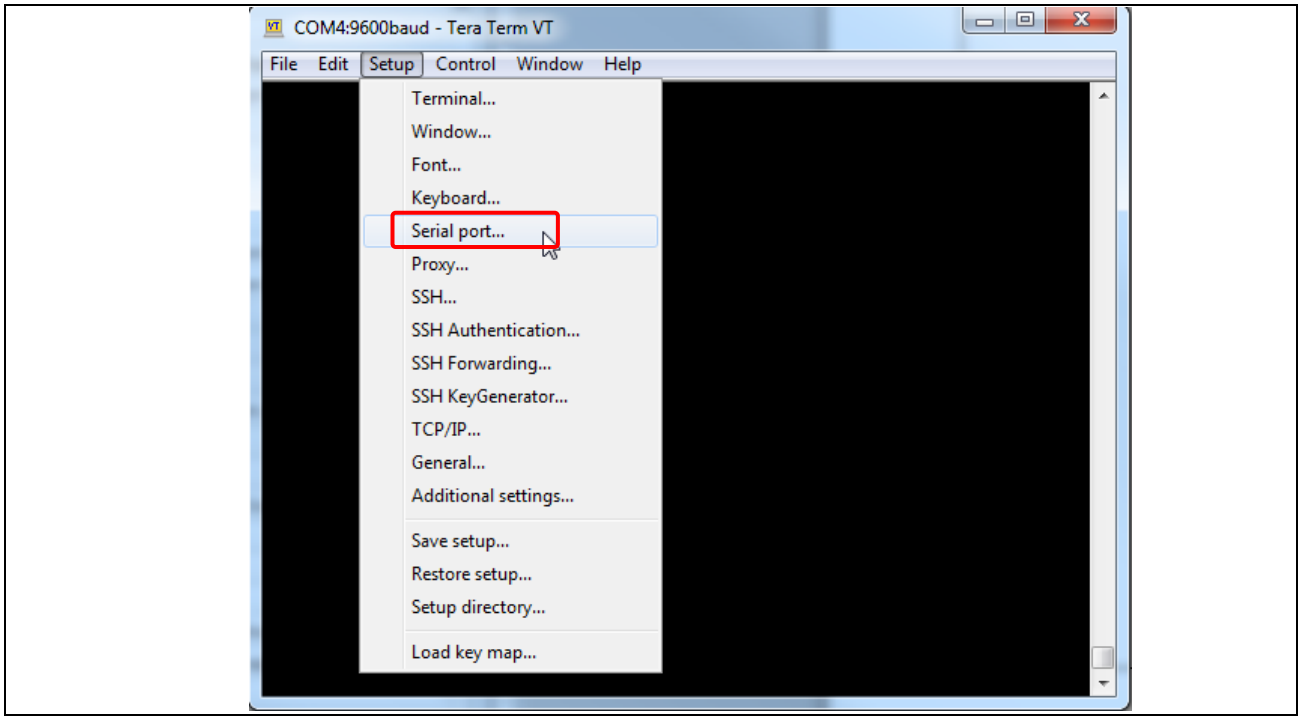


Figure 4-16 Terminal window in PC

- In [Serial port setup] popup window, ensure the settings below match the setting in *SCI/SCIF Asynchronous Mode* driver and click [OK] to proceed
    - Data length setting: 8 bits
    - Parity setting: None
    - Stop bit length setting: 1 bit
    - Bit rate: 9600 bps

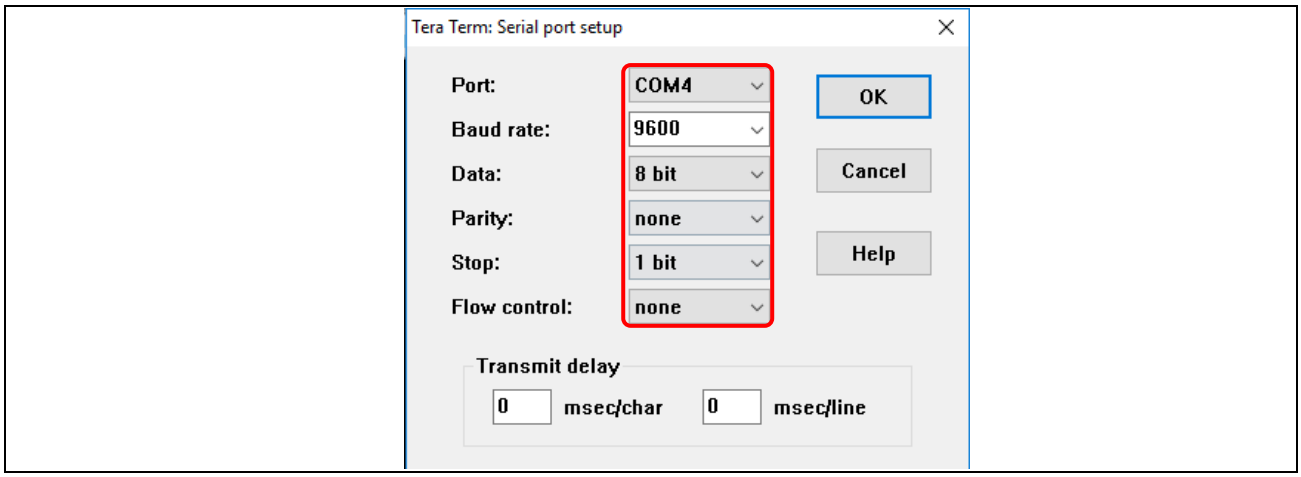


Figure 4-17 Serial port setup

Step 3. At e^2 studio, click debug  icon

Step 4. At debug perspective, click resume icon icon to execute the project

The PC terminal window will show the following message. Follow the message to increase or decrease LED2 blinking speed.

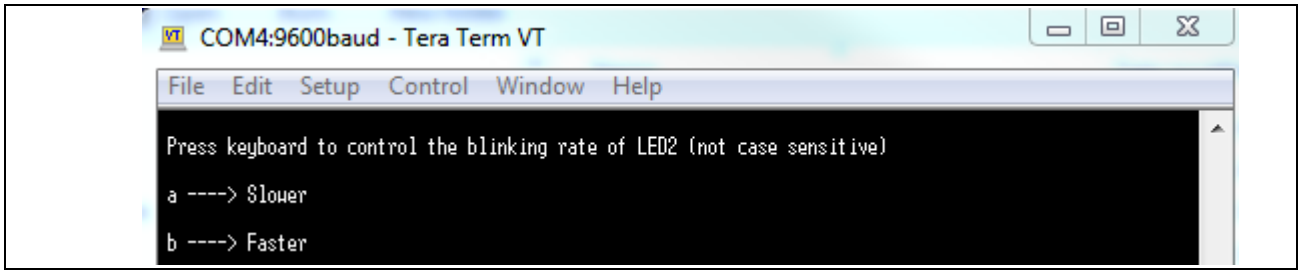


Figure 4-18 Terminal window in PC

These messages will be shown if RSK board receives an input from PC.

Figure 4-19 Response from RSK board when an input is received from PC

4.7 Operation on board

On the Renesas Starter Kit+ for RX65N 2MB board:

Blinking rate of LED2 is changed when key in character ‘a’ or ‘b’ at the PC terminal:

- a : Blinking of LED2 will slow down
- b : Blinking of LED2 will become faster

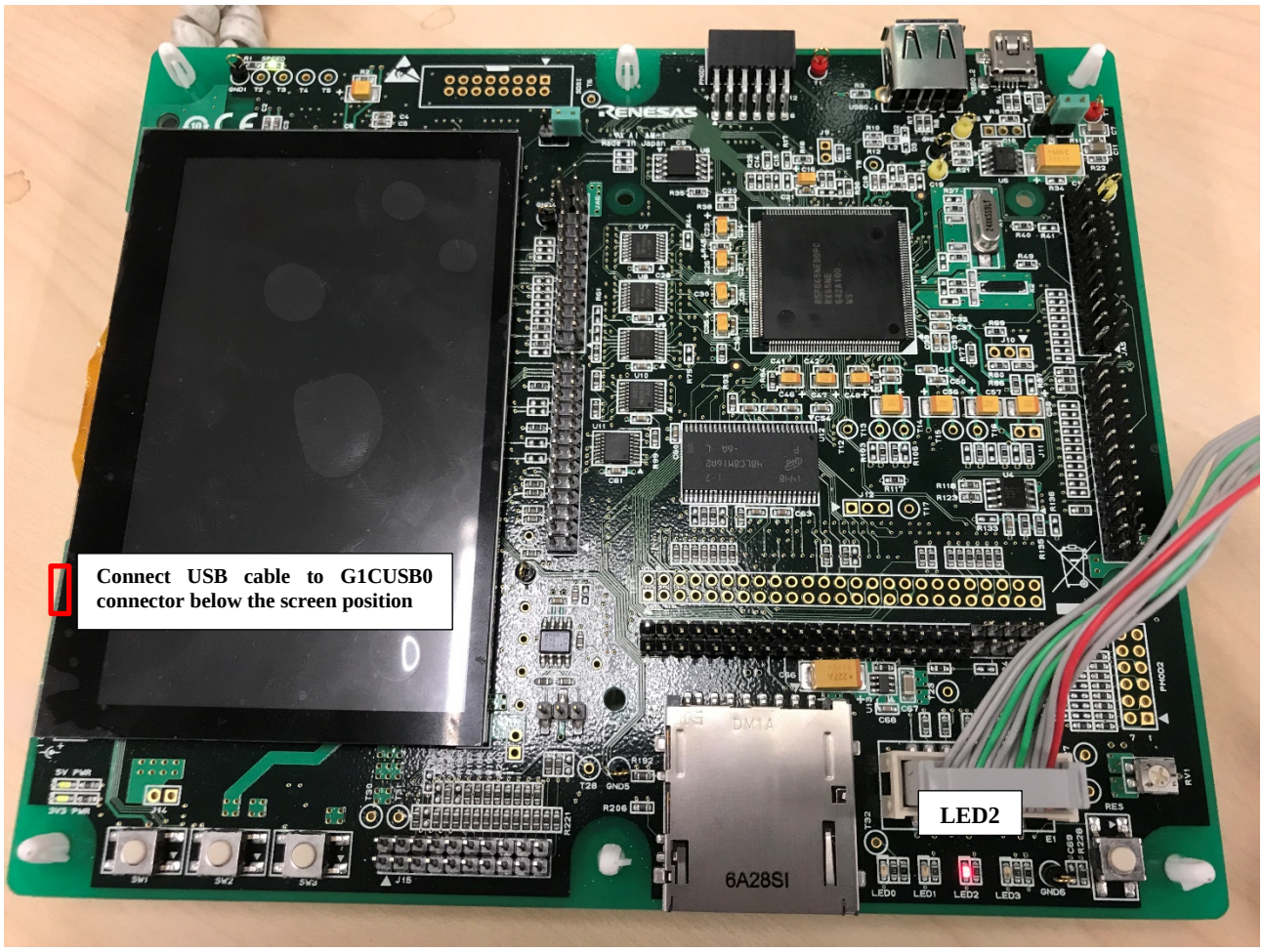


Figure 4-20 LED2 and Serial Port on RX65N 2MB board

5. Application Example 4 (Transferring data using DMA)

In the previous chapter, we use interrupt service routine to transfer the data between SCI8 and on-chip RAM. The data transfer is handled by CPU.

In this chapter, we introduce DMA controller to transfer the data. The memory operation is speeded up and the CPU is freed from data transfer.

Note:

Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

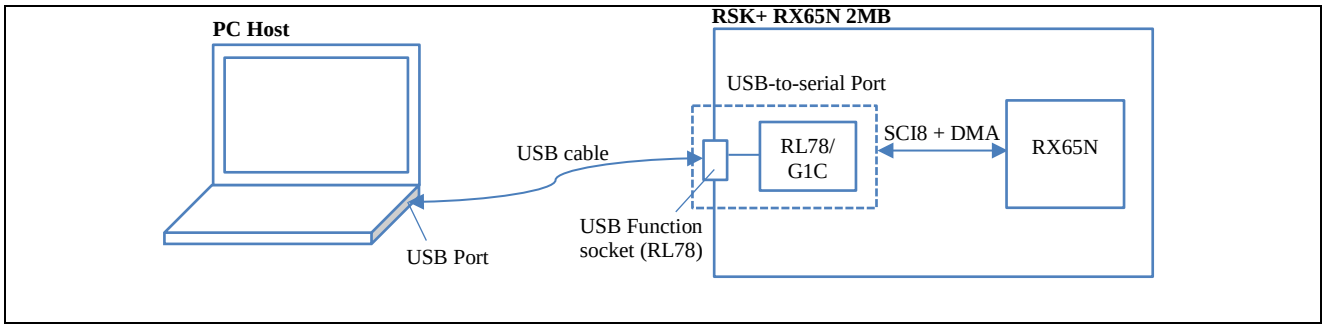


Figure 5-1 Data handled by DMAC

The table below shows changes of the driver selection:

	Drivers	Resource	Function
1. Add new driver	DMA Controller	DMAC0	Transfer data from SCI8 to RAM (Received)
		DMAC1	Transfer data from RAM to SCI8 (Transmit)
2. Modify driver	SCI/SCIF Asynchronous Mode	SCI8	Communication between PC terminal and target board for controlling LED blinking rate

The SCI-DMA application codes will be added to the program as below:

a) Main function:

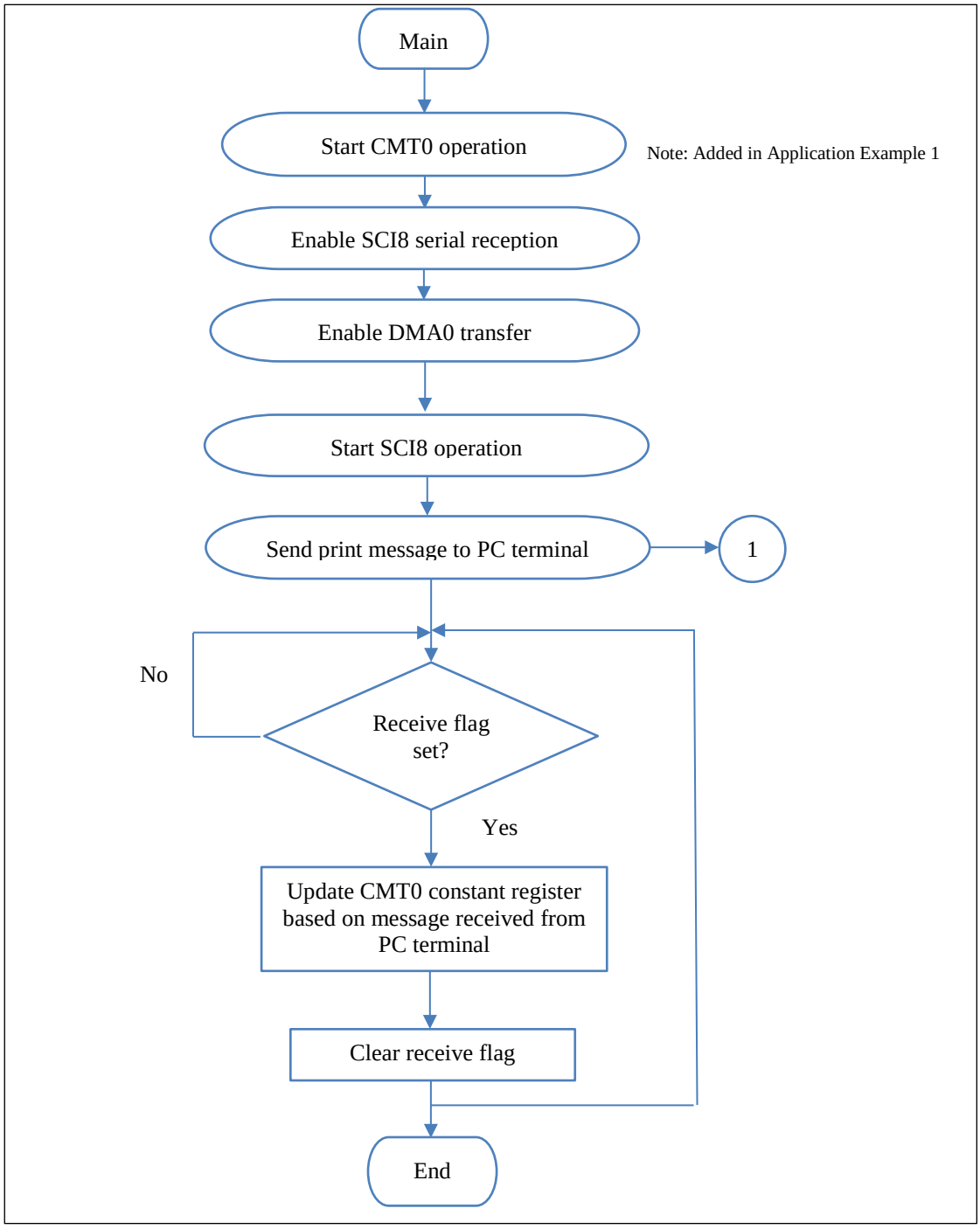


Figure 5-2 Main flowchart

b) SCI8 user functions:

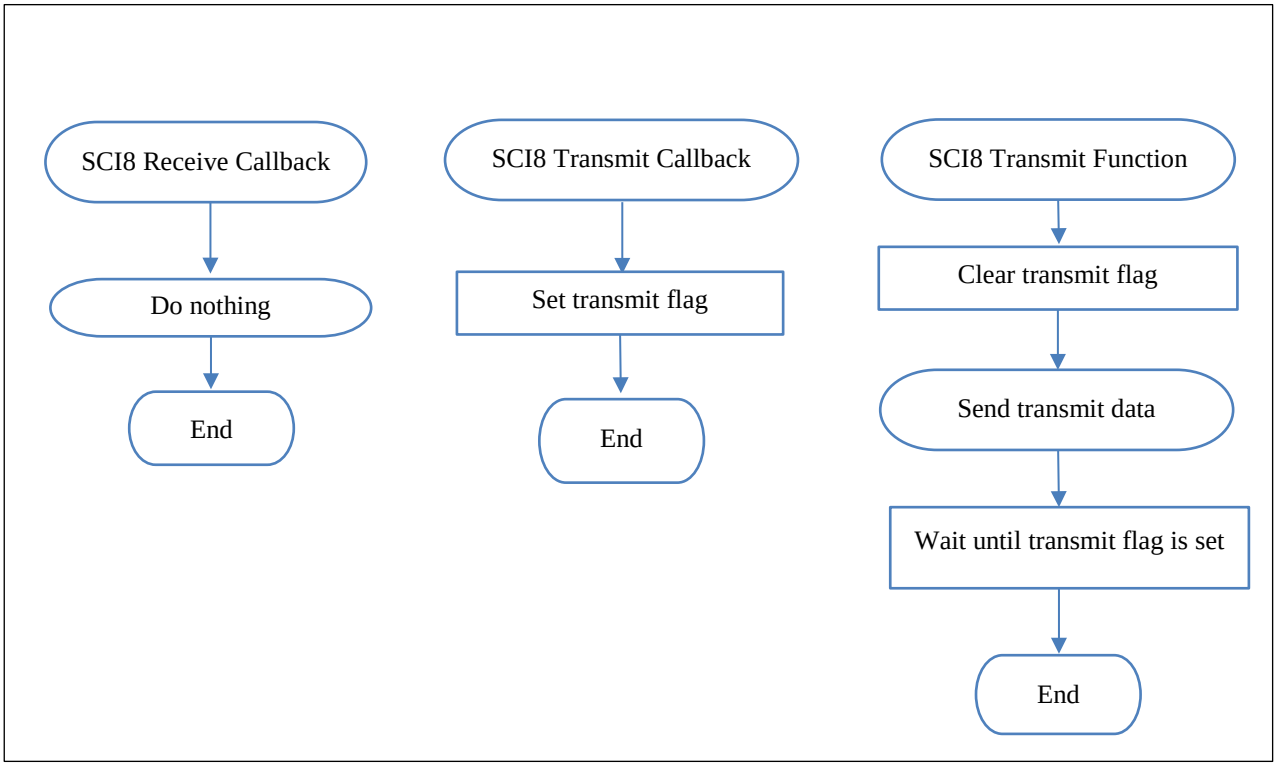


Figure 5-3 SCI user functions flowchart

c) DMAC0 user functions:

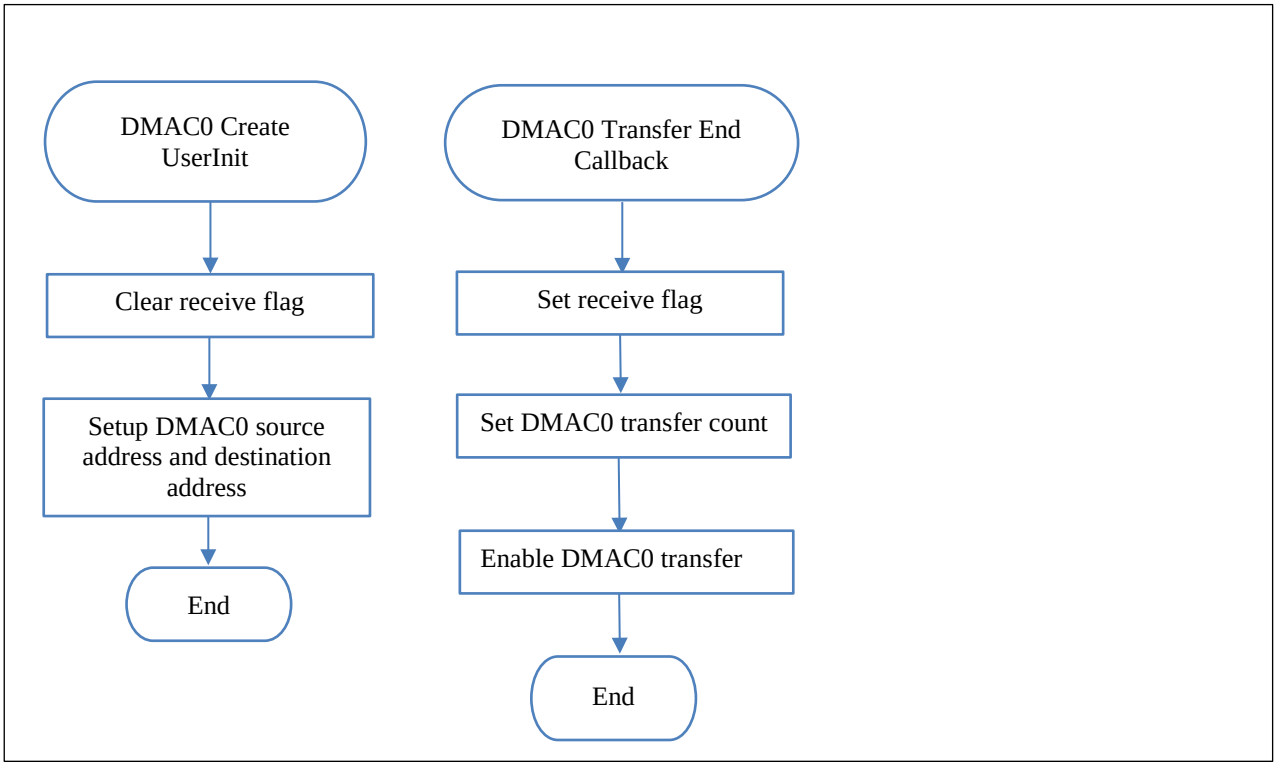


Figure 5-4 DMAC0 user functions flowchart

d) DMAC1 user functions:

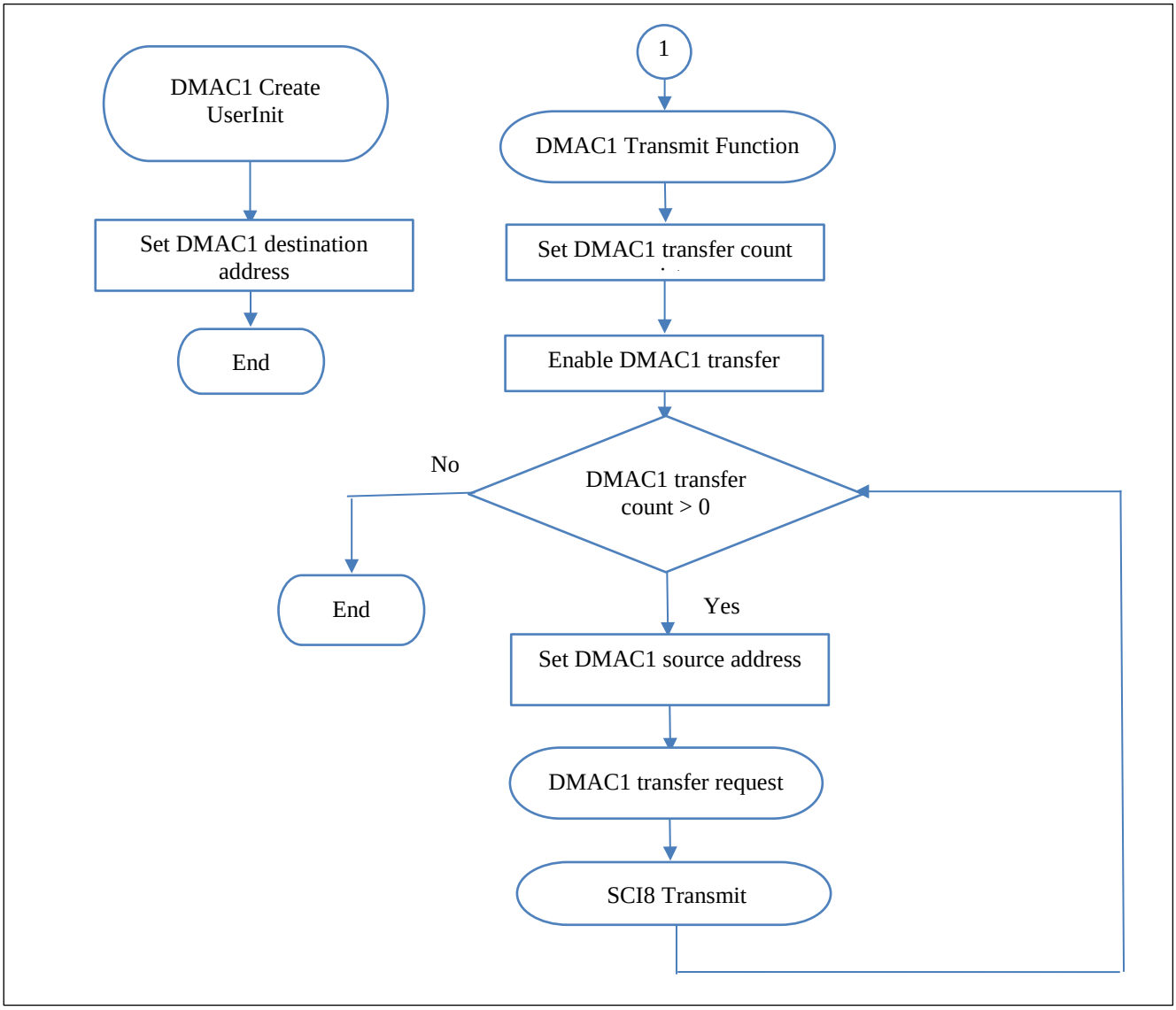


Figure 5-5 DMAC1 user functions flowchart

5.1 Adding and modifying peripheral drivers

5.1.1 Add DMAC0 driver

1) In the Smart_Configurator_Example.scfg pane, click [Components] tab
2) Click  to add new component

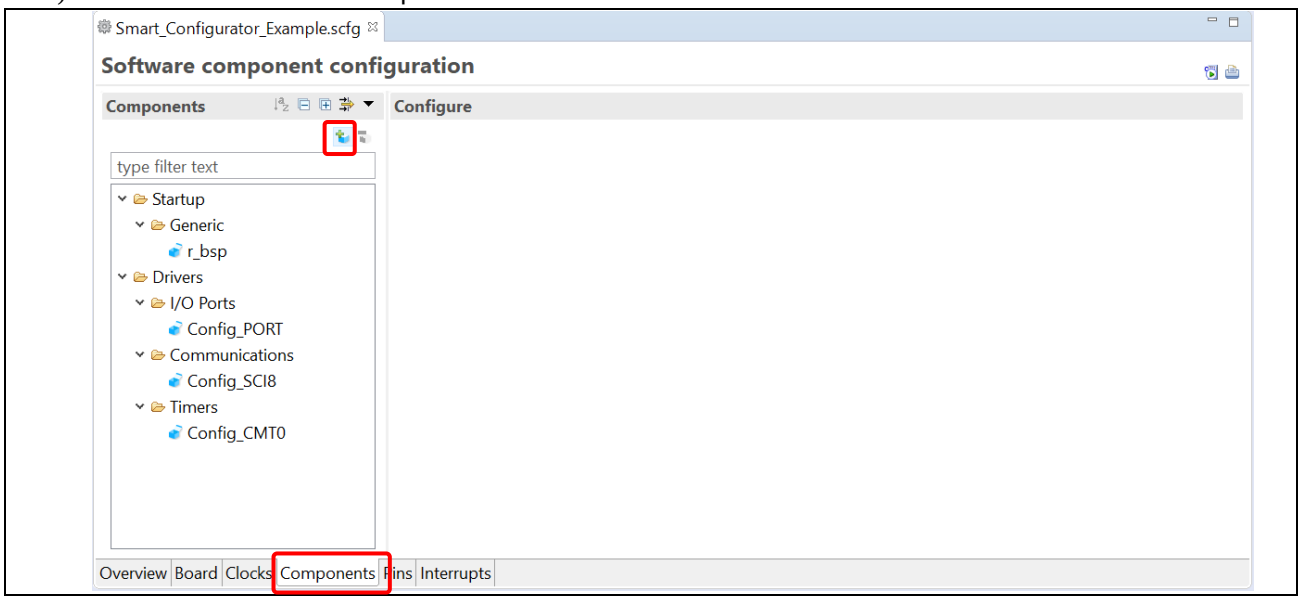


Figure 5-6 Software component configuration in Smart Configurator

3) Add *DMA Controller* driver into the project
   - Type “DMA” to the “Filter” box.
   - Navigate the component list and select *DMA Controller* driver
   - Click [Next] to continue

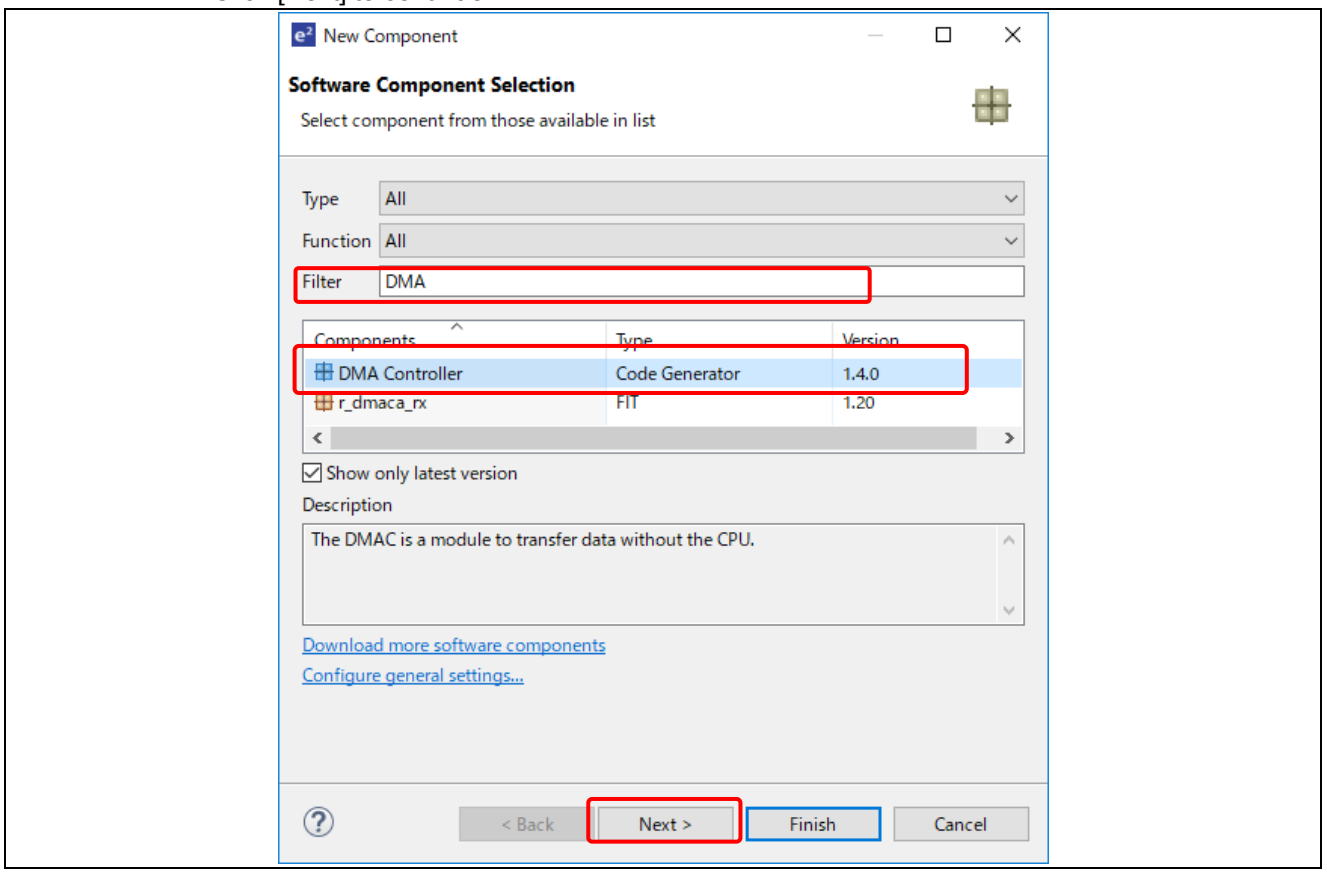


Figure 5-7 Select software component

4) In the [New Component] dialog box
   - Select *DMAC0* as resource
   - Keep the default configuration name. Source files and API will be generated based on this configuration name
   - Click [Finish]

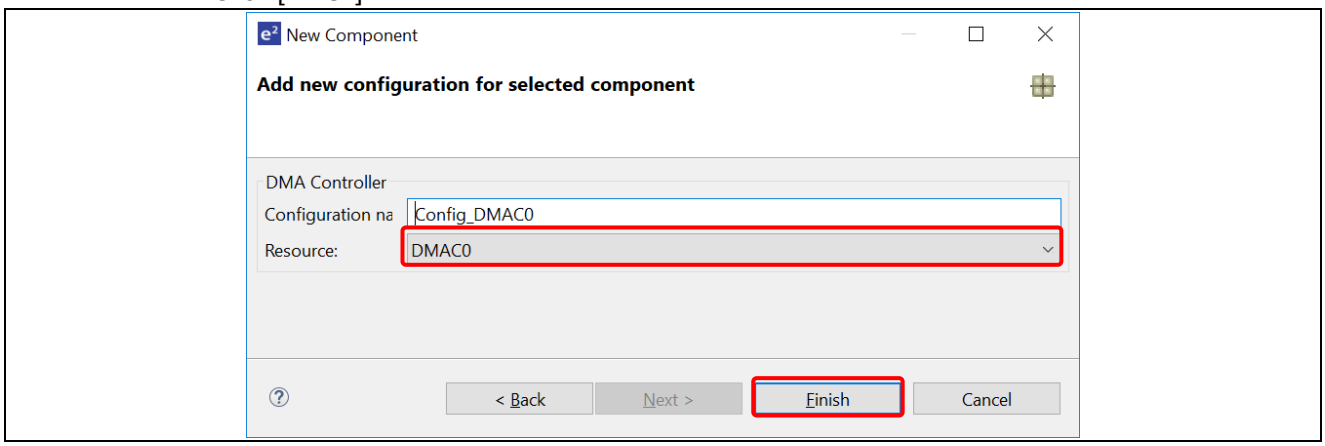


Figure 5-8 Add new configuration to the project

5) At Configure pane, ensure settings as the following:
   - Activation source: SCI8(RXI8)
   - Enable interrupt on transfer end: Checked
   - Keep other default settings

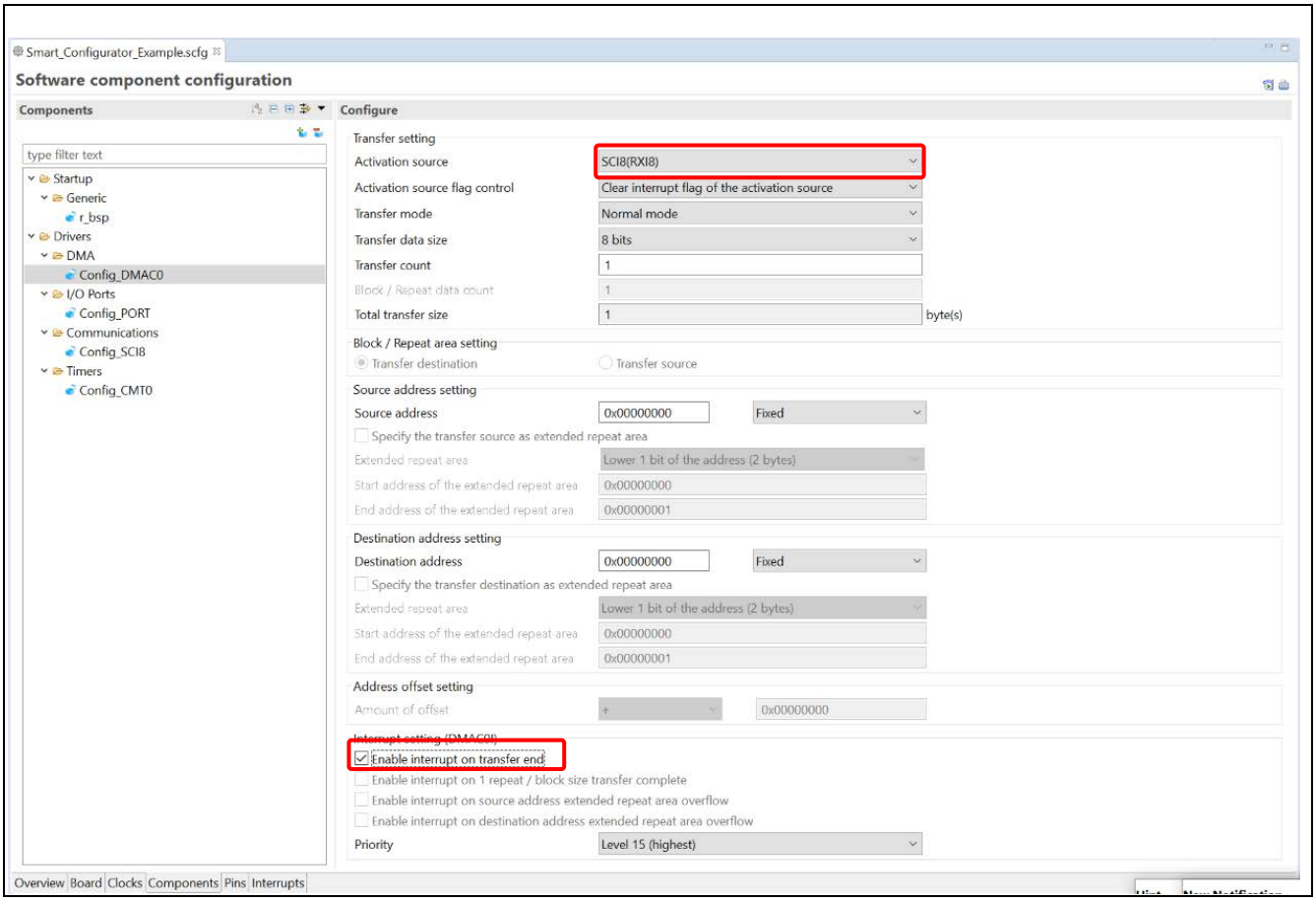


Figure 5-9 DMAC0 driver configuration

5.1.2 Add DMAC1 driver

1) In the Smart_Configurator_Example.scfg pane, click [Components] tab
2) Click to add new component

Figure 5-10 Software component configuration in Smart Configurator

3) Add *DMA Controller* driver into the project
   - Type “DMA” to the “Filter” box.
   - Navigate the component list and select *DMA Controller* driver
   - Click [Next] to continue

Figure 5-11 Select software component

4) In the [New Component] dialog box
   - Select *DMAC1* as resource
   - Keep the default configuration name. Source files and API will be generated based on this configuration name
   - Click [Finish]

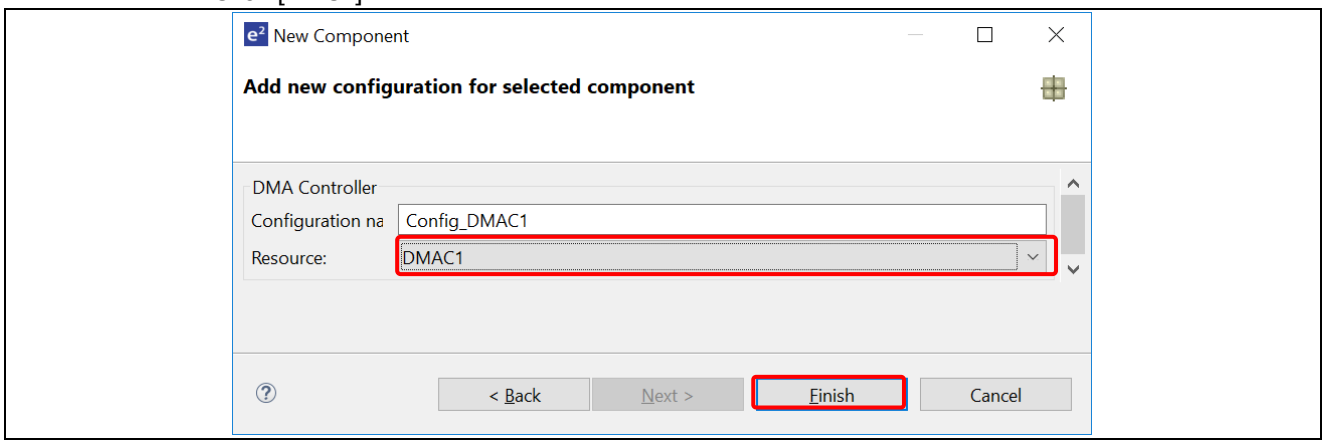


Figure 5-12 Add new configuration to the project

5) At Configure pane, ensure settings as the following:
   - Activation source: Software trigger
   - Keep other default settings

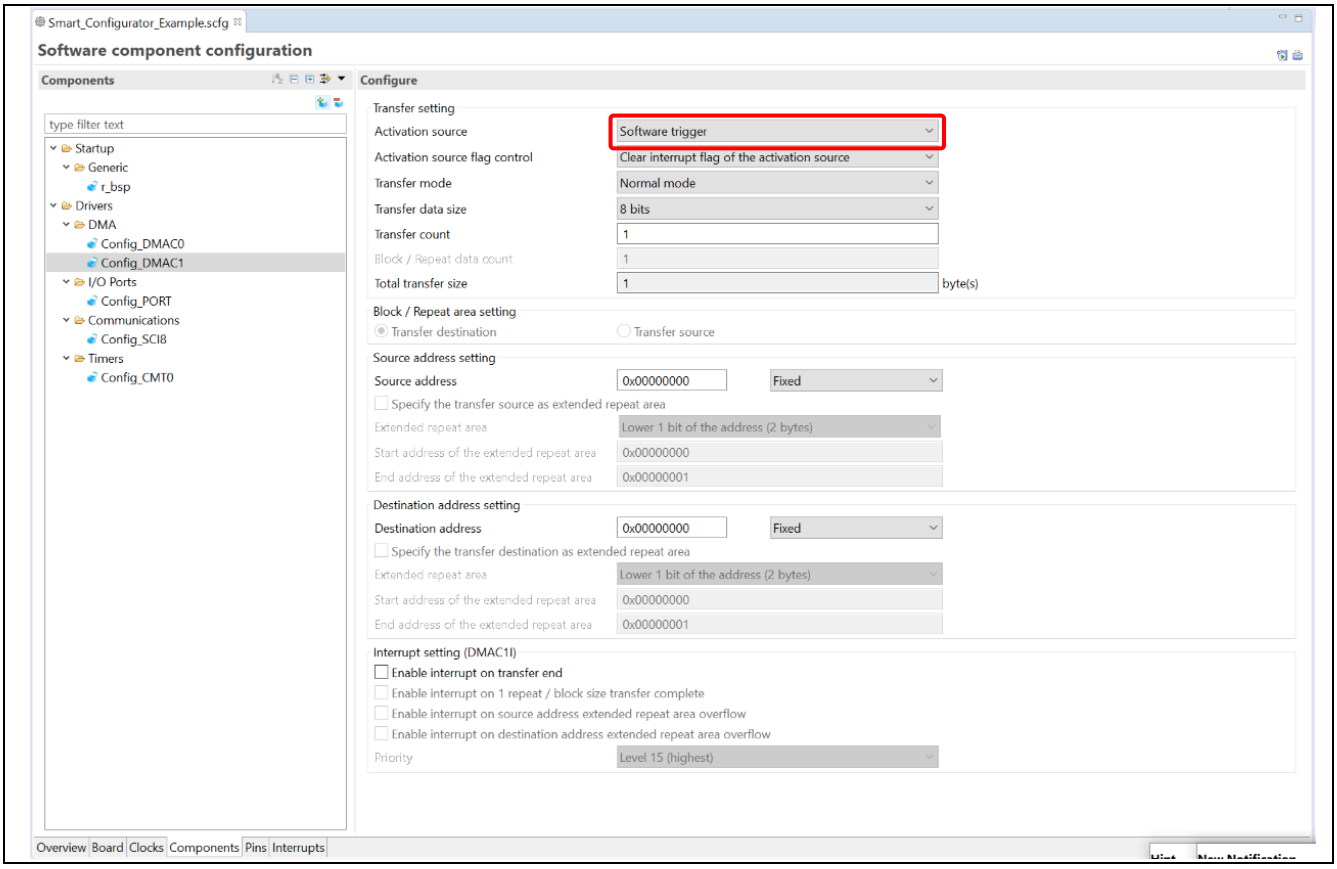


Figure 5-13 DMAC1 driver configuration

5.1.3 Modify SCI8 driver

1) In the Smart_Configurator_Example.scfg pane, click [Components] tab
2) Select *Config_SCI8* in the [Component] panel
3) At Configure pane, modify settings as the following:
   - Transmit data handling: Data handled by DMAC
   - Receive data handling: Data handled by DMAC
   - Keep other settings

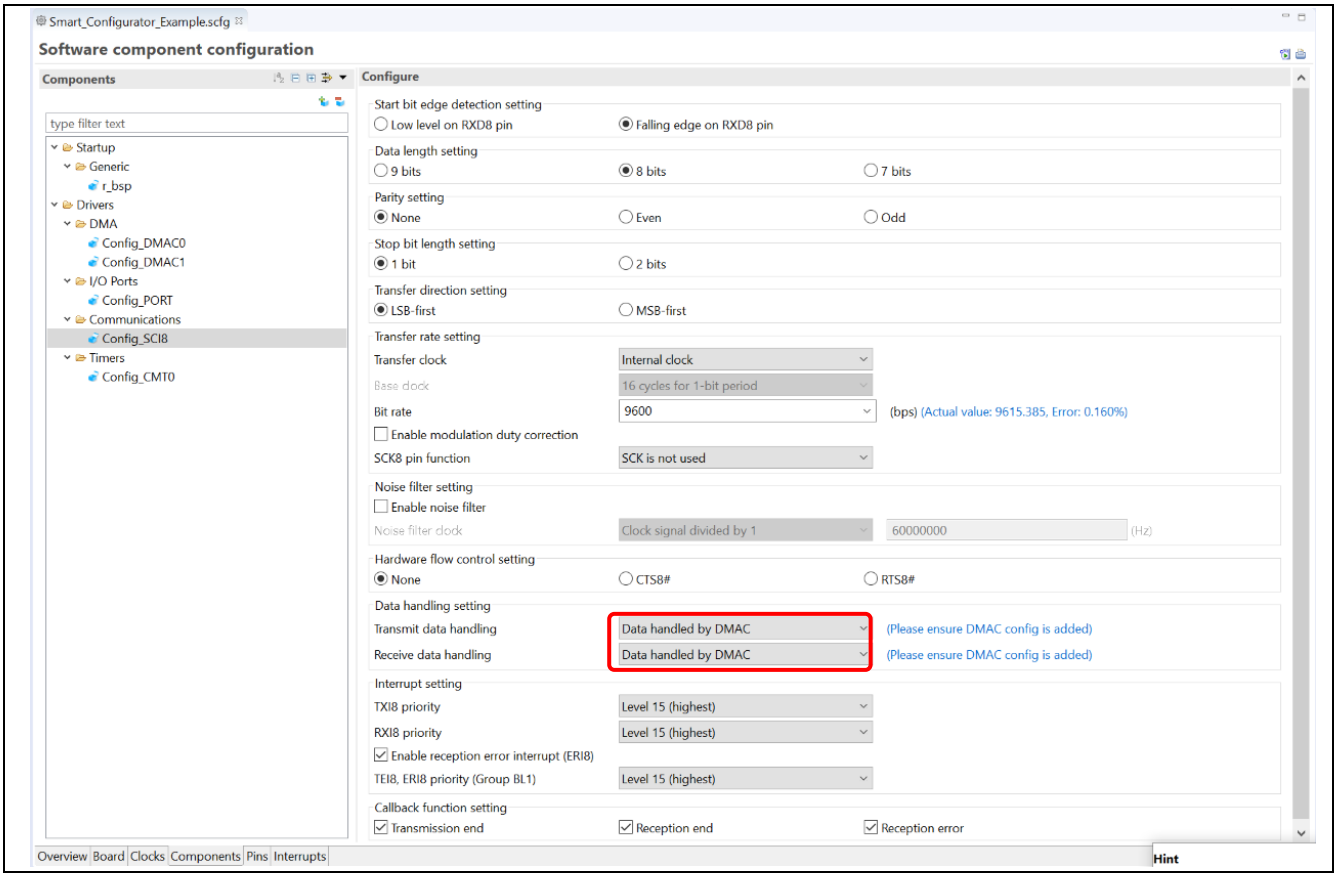


Figure 5-14 SCI8 driver configuration

5.2 Generating codes

At [Software Component Configuration] tab, click  to generate codes

5.3 Adding application codes in DMAC0 driver

At Project Explore tree, open “Config_DMAC0_user.c” in \src\smc_gen\Config_DMAC0 folder

a. Add the declarations below at the user code area near the top of the file:

```
/* Global variable used to receive a character from SCI8 */
volatile uint8_t g_rx_char_dmac;

/* Flag used to detect whether data is received */
volatile uint8_t g_rx_flag_dmac;
```

b. Add code to set the initial value for the reception completion flag, DMAC0 source address and DMAC0 destination address in *R_Config_DMAC0_Create_UserInit(void)* function

```
/* Clear receive flag */
g_rx_flag_dmac = 0U;

/* Set DMAC0 source address */
DMAC0.DMSAR = (void *)&SCI8.RDR;

/* Set DMAC0 destination address */
DMAC0.DMDAR = (void *)&g_rx_char_dmac;
```

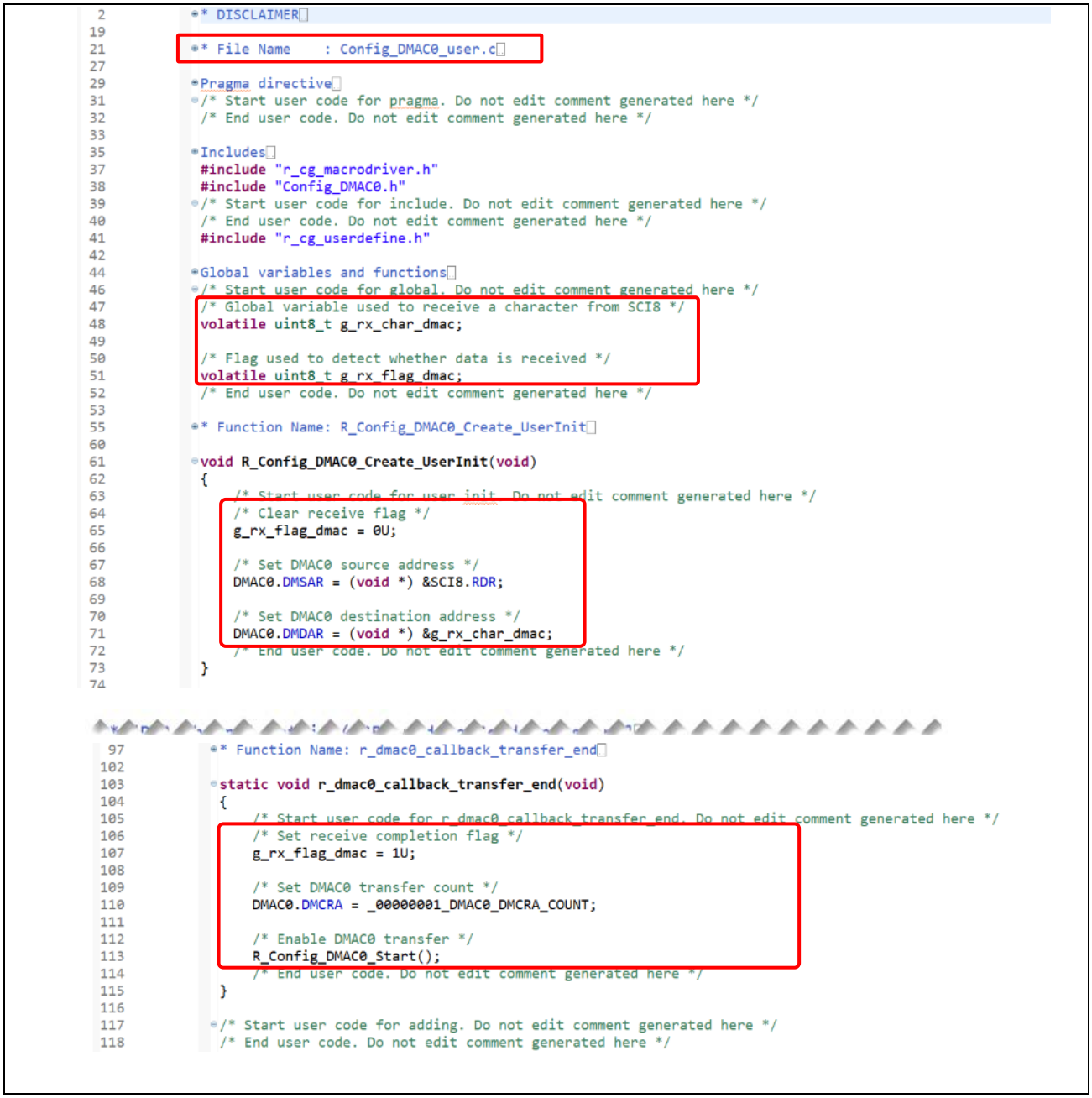
c. In *r_dmac0_callback_transfer_end(void)* function:

   i. Set the reception completion flag

   ii. Restart reception by setting the transfer count and enable DMAC0 transfer

```
/* Set receive completion flag */
g_rx_flag_dmac = 1U;

/* Set DMAC0 transfer count */
DMAC0.DMCRA = _00000001_DMAC0_DMCRA_COUNT;

/* Enable DMAC0 transfer */
R_Config_DMAC0_Start();
```

Your source file should look like this:



```
 2          * DISCLAIMER
19
21          * File Name    : Config_DMAC0_user.c
27
29          Pragma directive
31          /* Start user code for pragma. Do not edit comment generated here */
32           /* End user code. Do not edit comment generated here */
33
35          Includes
37           #include "r_cg_macrodriver.h"
38           #include "Config_DMAC0.h"
39          /* Start user code for include. Do not edit comment generated here */
40           /* End user code. Do not edit comment generated here */
41           #include "r_cg_userdefine.h"
42
44          Global variables and functions
46          /* Start user code for global. Do not edit comment generated here */
47          /* Global variable used to receive a character from SCI8 */
48          volatile uint8_t g_rx_char_dmac;
49
50          /* Flag used to detect whether data is received */
51          volatile uint8_t g_rx_flag_dmac;
52           /* End user code. Do not edit comment generated here */
53
55          * Function Name: R_Config_DMAC0_Create_UserInit
60
61          void R_Config_DMAC0_Create_UserInit(void)
62          {
63              /* Start user code for user init. Do not edit comment generated here */
64              /* Clear receive flag */
65              g_rx_flag_dmac = 0U;
66
67              /* Set DMAC0 source address */
68              DMAC0.DMSAR = (void *) &SCI8.RDR;
69
70              /* Set DMAC0 destination address */
71              DMAC0.DMDAR = (void *) &g_rx_char_dmac;
72              /* End user code. Do not edit comment generated here */
73          }
74
97          * Function Name: r_dmac0_callback_transfer_end
102
103         static void r_dmac0_callback_transfer_end(void)
104         {
105             /* Start user code for r_dmac0_callback_transfer_end. Do not edit comment generated here */
106             /* Set receive completion flag */
107             g_rx_flag_dmac = 1U;
108
109             /* Set DMAC0 transfer count */
110             DMAC0.DMCRA = _00000001_DMAC0_DMCRA_COUNT;
111
112             /* Enable DMAC0 transfer */
113             R_Config_DMAC0_Start();
114             /* End user code. Do not edit comment generated here */
115         }
116
117         /* Start user code for adding. Do not edit comment generated here */
118          /* End user code. Do not edit comment generated here */
```

Figure 5-15 Config_DMAC0_user.c

5.4 Adding application codes in DMAC1 driver

At Project Explore tree, open “Config_DMAC1_user.c” in \src\smc_gen\Config_DMAC1 folder

a. Add header file at the user code area near the top of the file:

```
#include "Config_SCI8.h"
```

b. Add code to set the initial value for DMAC1 destination address in *R_Config_DMAC1_Create_UserInit(void)* function

```
/* Set DMAC1 destination address */
DMAC1.DMDAR = (void *)&SCI8.TDR;
```

c. Add a function to transfer data to SCI8 and call SCI8 function to send data to PC at user code area at the bottom of the file

```
/***********************************************************************
*********************************************
* Function Name: R_DMAC1_AsyncTransmit
* Description  : This function transfer data to SCI8 and call SCI8
function to send data to PC.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************
*********************************************/
MD_STATUS R_DMAC1_AsyncTransmit(uint8_t * const tx_buf, const uint16_t
tx_num)
{
    MD_STATUS status = MD_OK;

    uint8_t * source_address = tx_buf;

       // Set DMAC1 transfer count
       DMAC1.DMCRA = tx_num;

       // Enables DMA transfer
       R_Config_DMAC1_Start();

       while(DMAC1.DMCRA > 0)
       {
              // Set DMAC1 source address
              DMAC1.DMSAR = (void *) source_address;
              source_address++;

              // DMA transfer request
              R_Config_DMAC1_Set_SoftwareTrigger();

              // Sends SCI8 data
              R_SCI8_AsyncTransmit(NULL, 1);
       }

       return (status);
}
/***********************************************************************
*********
* End of function R_DMAC1_AsyncTransmit
       ***********************************************************
**************/
```

Your source file should look like this:

Config_DMAC1_user.c

```
 2  * DISCLAIMER
19
21  * File Name     : Config_DMAC1_user.c
27
29  Pragma directive
31  /* Start user code for pragma. Do not edit comment generated here */
32  /* End user code. Do not edit comment generated here */
33
35  Includes
37  #include "r_cg_macrodriver.h"
38  #include "Config_DMAC1.h"
39  /* Start user code for include. Do not edit comment generated here */
40  #include "Config_SCI8.h"
41  /* End user code. Do not edit comment generated here */
42  #include "r_cg_userdefine.h"
43
45  Global variables and functions
47  /* Start user code for global. Do not edit comment generated here */
48  /* End user code. Do not edit comment generated here */
49
51  * Function Name: R_Config_DMAC1_Create_UserInit
56
57  void R_Config_DMAC1_Create_UserInit(void)
58  {
59      /* Start user code for user init. Do not edit comment generated here */
60      /* Set DMAC1 destination address */
61      DMAC1.DMDAR = (void *)&SCI8.TDR;
62      /* End user code. Do not edit comment generated here */
63  }
64
```

```
65  /* Start user code for adding. Do not edit comment generated here */
66  /***********************************************************************************************************************
67  * Function Name: R_DMAC1_AsyncTransmit
68  * Description  : This function transfer data to SCI8 and call SCI8 function to send data to PC.
69  * Arguments    : tx_buf -
70  *                    transfer buffer pointer
71  *                tx_num -
72  *                    buffer size
73  * Return Value : status -
74  *                    MD_OK or MD_ARGERROR
75  ***********************************************************************************************************************/
76  MD_STATUS R_DMAC1_AsyncTransmit(uint8_t * const tx_buf, const uint16_t tx_num)
77  {
78      MD_STATUS status = MD_OK;
79
80      uint8_t * source_address = tx_buf;
81
82      // Set DMAC1 transfer count
83      DMAC1.DMCRA = tx_num;
84
85      // Enables DMA transfer
86      R_Config_DMAC1_Start();
87
88      while(DMAC1.DMCRA > 0)
89      {
90          // Set DMAC1 source address
91          DMAC1.DMSAR = (void *) source_address;
92          source_address++;
93
94          // DMA transfer request
95          R_Config_DMAC1_Set_SoftwareTrigger();
96
97          // Sends SCI8 data
98          R_SCI8_AsyncTransmit(NULL, 1);
99      }
100
101     return (status);
102 }
103 /***********************************************************************************************************************
104 * End of function R_DMAC1_AsyncTransmit
105 ***********************************************************************************************************************/
106 /* End user code. Do not edit comment generated here */
107
```

Figure 5-16 Config_DMAC1_user.c

5.5 Modifying application codes in SCI8 driver

1) At Project Explore tree, open “Config_SCI8_user.c” in \src\smc_gen\Config_SCI8 folder

a. Remove the declarations “g_rx_char” and “g_rx_flag” at the user code area near the top of the file:

```
/* Global variable used to receive a character from PC terminal */
volatile uint8_t g_rx_char;

/* Flag used to detect whether data is received */
volatile uint8_t g_rx_flag;

/* Flag used to detect completion of transmission */
static volatile uint8_t SCI8_txdone;
```

b. Remove code in *r_Config_SCI8_callback_receiveend (void)* function

```
/* Set receive completion flag */
g_rx_flag = 1U;

/* Set SCI8 receive buffer address and restart reception */
R_Config_SCI8_Serial_Receive((uint8_t *)&g_rx_char, 1);
```

Your source file should look like this:

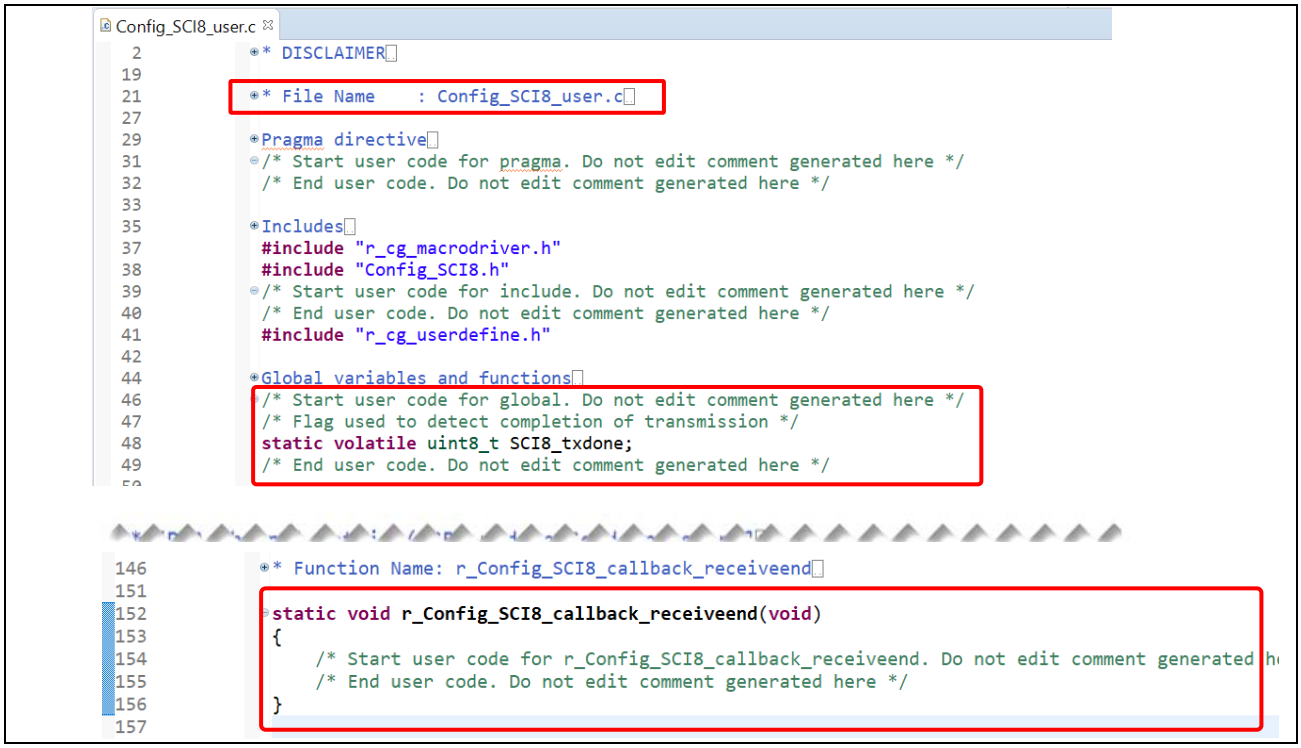


Figure 5-17 Config_SCI8_user.c

5.6 Adding application codes in *main()*

1) Near the top of the file Smart_Configurator_Example.c, there are 2 external declarations (g_rx_char and g_rx_flag) which are used for SCI8 and are defined in file Config_SCI8_user.c. Because we use DMAC instead of SCI8, they become obsolete now. Remove them.

```
/* Flag used to detect whether data is received from PC terminal */
extern volatile uint8_t g_rx_flag;

/* Global variable used for storing data received from PC terminal */
extern volatile uint8_t g_rx_char;
```

Add new external declarations for DMAC0 variables. (Defined in DMAC Config_DMAC0_user.c in section 5.3)

```
/* Flag used to detect whether data is received from PC terminal */
extern volatile uint8_t g_rx_flag_dmac;

/* Global variable used for storing data received from PC terminal */
extern volatile uint8_t g_rx_char_dmac;
```

2) Replace all codes inside *main*() function with below codes:

```
            /* Start CMT0 counter operation */
            R_Config_CMT0_Start();

            /* Set SCI8 receive buffer address and enable receive interrupt */
            R_Config_SCI8_Serial_Receive(NULL, 0);

            /* Enable DMAC0 operation */
            R_Config_DMAC0_Start();

            /* Enable SCI8 operation */
            R_Config_SCI8_Start();

            /* Print instruction onto PC terminal */
            sprintf(print_str, "\r\n Press keyboard to control the blinking rate of LED2 (not case sensitive)\r\n");
            R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
            sprintf(print_str, "\r\n a ----> Slower\n");
            R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
            sprintf(print_str, "\r\n b ----> Faster\n");
            R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

            while (1U)
            {
                        /* Get new blink interval level from PC terminal */
                        if (g_rx_flag_dmac == 1U)
                        {
                                    /* Instruction to slow down blinking rate is received */
                                    if (('A' == g_rx_char_dmac) || ('a' == g_rx_char_dmac))
                                    {
                                                R_Config_SCI8_Serial_Receive(NULL, 0);

                                                /* Notify the character received and inform next action */
                                                sprintf(print_str, "\r\n Character 'a' or 'A' is received. LED2 will blink at slower
rate.\r\n");
                                                R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

                                                /* Get new blink interval level. Maximum level is 9. */
                                                if (interval_level < 10)
                                                {
                                                            interval_level++;
                                                }
                                                else
                                                {
                                                            sprintf(print_str, "\r\n This is minimum. Please press 'b' or 'B' to blink faster.
\r\n");
                                                            R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
                                                }
                                    }
                                    /* Instruction to increase blinking rate is received */
                                    else if (('B' == g_rx_char_dmac) || ('b' == g_rx_char_dmac))
                                    {
                                                R_Config_SCI8_Serial_Receive(NULL, 0);

                                                /* Notify the character received and inform next action */
                                                sprintf(print_str, "\r\n Character 'b' or 'B' is received. LED2 will blink at faster
rate.\r\n");
                                                R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

                                                /* Get new blink interval level. Minimum level is 1 */
                                                if (interval_level > 1)
                                                {
                                                            interval_level--;
                                                }
                                                else
                                                {
                                                            sprintf(print_str, "\r\n This is maximum. Please press 'a' or 'A' to blink slower.
\r\n");
                                                            R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
                                                }
                                    } else
                                    {
                                                /* Do nothing */
                                    }

                                    /* Change blinking rate */
                                    CMT0.CMCOR = (uint16_t) (5000 * interval_level);

                                    /* Reset SCI8 reception flag*/
                                    g_rx_flag_dmac = 0U;
                        }
                        else
                        {
                                    /* Do nothing */
                        }

            }
```

Your source file should look like this:

```
 *  FILE        : Smart_Configurator_Example.c
#include "r_smc_entry.h"
#include <stdio.h>
#include <string.h>

/* Global variable for changing CMT0 interval */
volatile uint16_t interval_level = 5;

/* String used to print message at PC terminal */
static char print_str[80];

/* Flag used to detect whether data is received from PC terminal */
extern volatile uint8_t g_rx_flag_dmac;

/* Global variable used for storing data received from PC terminal */
extern volatile uint8_t g_rx_char_dmac;

void main(void);

void main(void)
{
    /* Start CMT0 counter operation */
    R_Config_CMT0_Start();

    /* Set SCI8 receive buffer address and enable receive interrupt */
    R_Config_SCI8_Serial_Receive(NULL, 0);

    /* Enable DMAC0 operation */
    R_Config_DMAC0_Start();

    /* Enable SCI8 operation */
    R_Config_SCI8_Start();

    /* Print instruction onto PC terminal */
    sprintf(print_str, "\r\n Press keyboard to control the blinking rate of LED2 (not case sensitive)\r\n");
    R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
    sprintf(print_str, "\r\n a ----> Slower\n");
    R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
    sprintf(print_str, "\r\n b ----> Faster\n");
    R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

    while (1U)
    {
        /* Get new blink interval level from PC terminal */
        if (g_rx_flag_dmac == 1U)
        {
            /* Instruction to slow down blinking rate is received */
            if (('A' == g_rx_char_dmac) || ('a' == g_rx_char_dmac))
            {
                R_Config_SCI8_Serial_Receive(NULL, 0);

                /* Notify the character received and inform next action */
                sprintf(print_str, "\r\n Character 'a' or 'A' is received. LED2 will blink at slower rate.\r\n");
                R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));

                /* Get new blink interval level. Maximum level is 9. */
                if (interval_level < 10)
                {
                    interval_level++;
                }
                else
                {
                    sprintf(print_str, "\r\n This is minimum. Please press 'b' or 'B' to blink faster. \r\n");
                    R_DMAC1_AsyncTransmit((uint8_t *) print_str, (uint16_t) strlen(print_str));
                }
            }
            /* Instruction to increase blinking rate is received */
            else if (('B' == g_rx_char_dmac) || ('b' == g_rx_char_dmac))
            {
                R_Config_SCI8_Serial_Receive(NULL, 0);

                /* Notify the character received and inform next action */
                sprintf(print_str, "\r\n Character 'b' or 'B' is received. LED2 will blink at faster rate.\r\n");
                R_DMAC1_AsyncTransmit((uint8_t *) print_str,    (uint16_t) strlen(print_str));

                /* Get new blink interval level. Minimum level is 1 */
                if (interval_level > 1)
                {
                    interval_level--;
                }
                else
                {
                    sprintf(print_str, "\r\n This is maximum. Please press 'a' or 'A' to blink slower. \r\n");
                    R_DMAC1_AsyncTransmit((uint8_t *) print_str,    (uint16_t) strlen(print_str));
                }
            } else
            {
                /* Do nothing */
            }

            /* Change blinking rate */
            CMT0.CMCOR = (uint16_t) (5000 * interval_level);

            /* Reset SCI8 reception flag*/
            g_rx_flag_dmac = 0U;
        }
        else
        {
            /* Do nothing */
        }
    }
}
```

5.7 Build and run on hardware board

Refer to section 4.6

5.8 Operation on board

Refer to section 4.7

6. Application Example 5 (Integration with USB function using Smart Configurator and FIT module)

In previous chapter, serial communication (asynchronous mode) is used to control LED2 blinking rate.

This chapter describes how to control LED2 blinking rate with terminal program on PC via USB serial communication. In this application example, two FIT modules will be integrated to the project to emulate a serial COM port on RX65N for the communication with PC:

R01AN2025: USB Basic Host and Peripheral Driver
R01AN2030: USB Peripheral Communications Device Class Driver (PCDC)

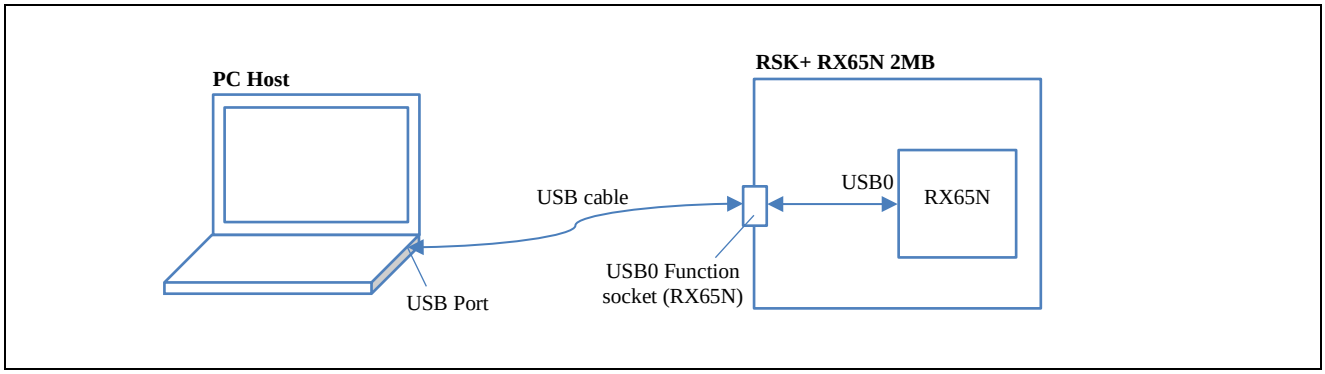


Figure 6-1 Control LED blinking rate by terminal program via USB serial communication

The schematic diagram of Renesas Starter Kit+ for RX65N 2MB is shown below. In this example, USB0_VBUS (P16) will be configured before using USB0 function. Ensure J7 Pin2-Pin3 are shorted (refer to Chapter 5.8 for J7 location).

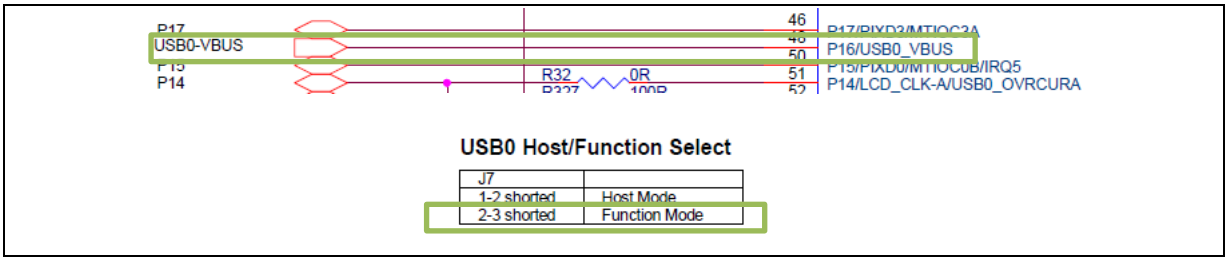


Figure 6-2 Schematic diagram of Renesas Starter Kit+ for RX65N 2MB

The table below shows changes of the driver selection:

	Drivers	Resource	Function
1. Remove driver	SCI/SCIF Asynchronous Mode	SCI8	Communication between PC terminal and target board for controlling LED blinking rate
	DMA Controller	DMAC0	Transfer data from SCI8 to RAM (Received)
		DMAC1	Transfer data from RAM to SCI8 (Transmit)
2. Add new driver	r_usb_basic	USB0	USB basic host and peripheral firmware Performs USB hardware control
	r_usb_pcdc	USB0	Operates as a USB peripheral communications device class driver (PCDC) Enables communication with a USB host

USB PCDC Mode application codes will be added to the program as below:

a) Main function:

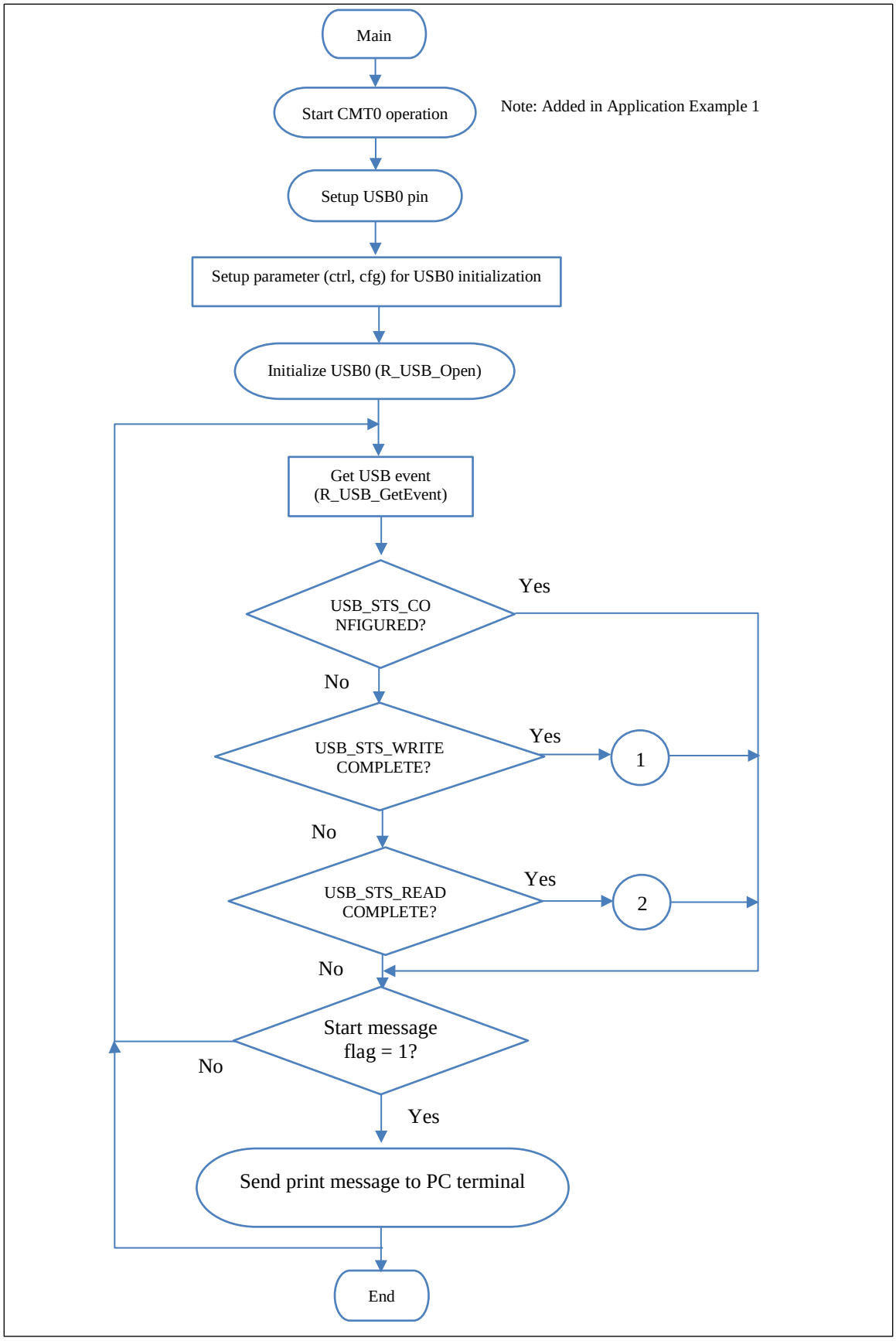


Figure 6-3 Main flowchart

b) Process when USB_STS_WRITE_COMPLETE is detected:

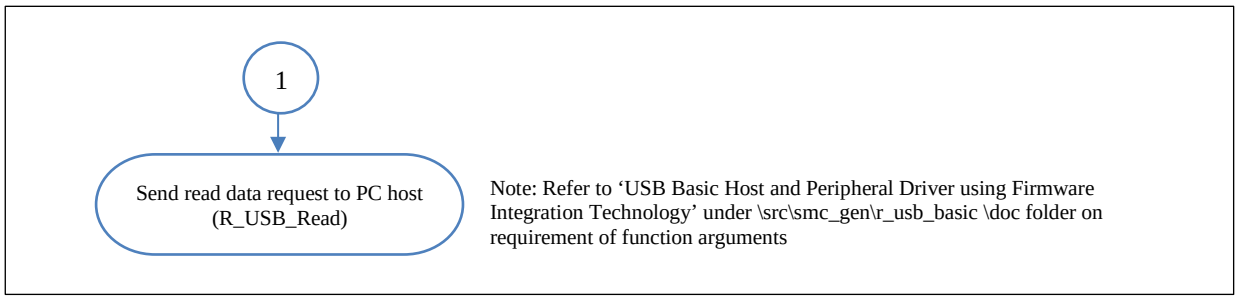


Figure 6-4 Process when USB_STS_WRITE_COMPLETE is detected

c) Process when USB_STS_READ_COMPLETE is detected:

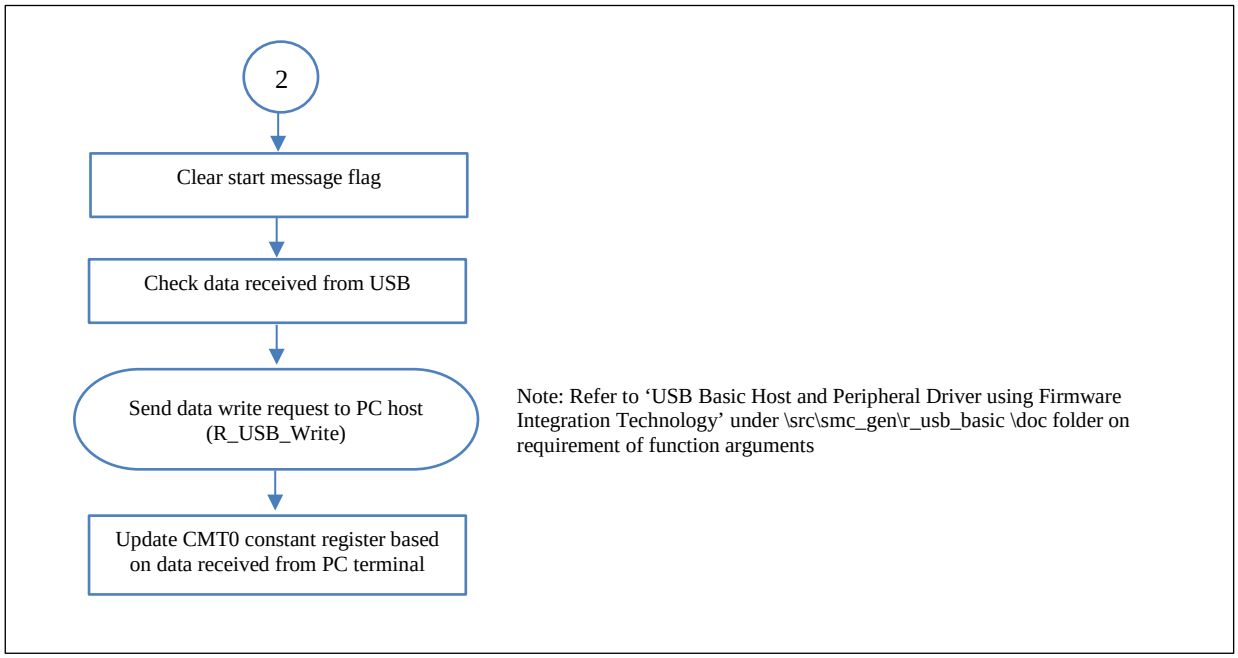


Figure 6-5 Process when USB_STS_READ_COMPLETE is detected

6.1 Removing SCI/SCIF Asynchronous Mode driver and related application codes

1) In Smart_Configurator_Example.scfg pane, click the [Components] tab

2) At Components tree, click to select *Config_SCI8* configuration. Click  to remove this configuration. Repeat the same steps to remove Config_DMAC0 and Config_DMAC1.

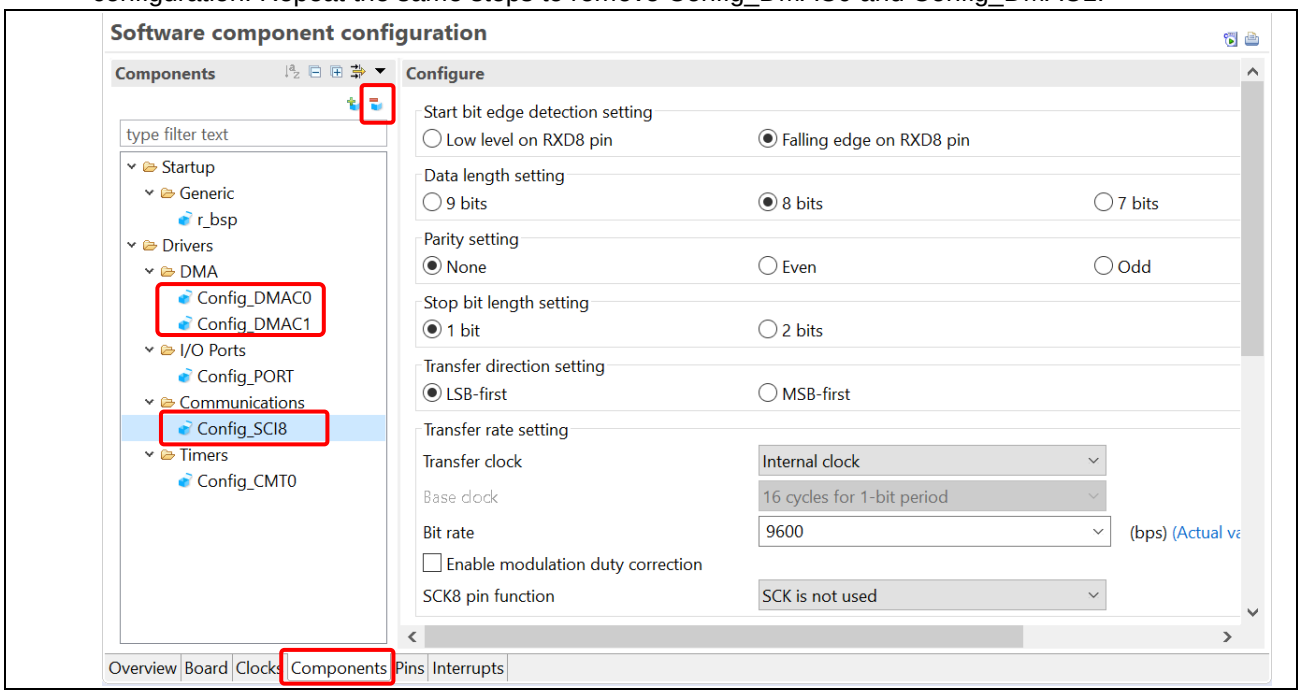


Figure 6-6 Software component configuration in Smart Configurator

3) Click  to generate codes
The source folder *Config_SCI8, Config_DMAC0* and *Config_DMAC1* will be moved to project \trash folder before the code generation operation

4) At the project tree, open “Smart_Configurator_Example.c” in \src folder

5) Remove the application codes which are related to previous example

a. Remove declarations for variables

```
#include <stdio.h>
#include <string.h>

/* Global variable used for changing CMT0 interval */
volatile uint16_t interval_level = 5;

/* String used to print message at PC terminal */
static char print_str[70];

/* Flag used to detect whether data is received from PC terminal */
extern volatile uint8_t g_rx_flag_dmac;

/* Global variable used for storing data received from PC terminal */
extern volatile uint8_t g_rx_char_dmac;
```

b. Remove all codes after *R_Config_CMT0_Start* function

After deleted the codes related to SCI8, the main file will look like following:

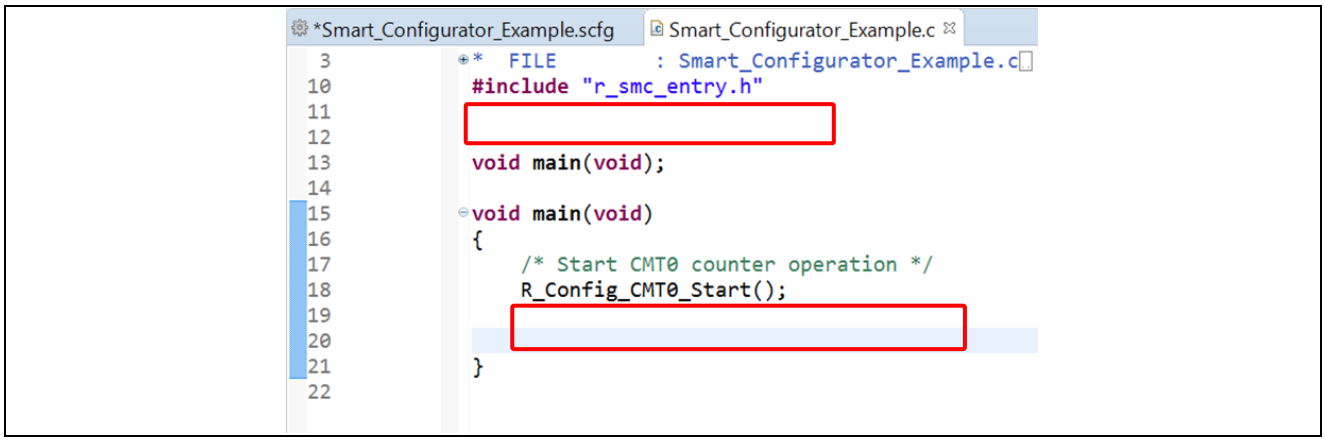


Figure 6-7 Main file after deleted codes related to SCI8

6.2 Adding a peripheral driver

1) In the Smart_Configurator_Example.scfg pane, click [Components] tab
2) Click  to add new component

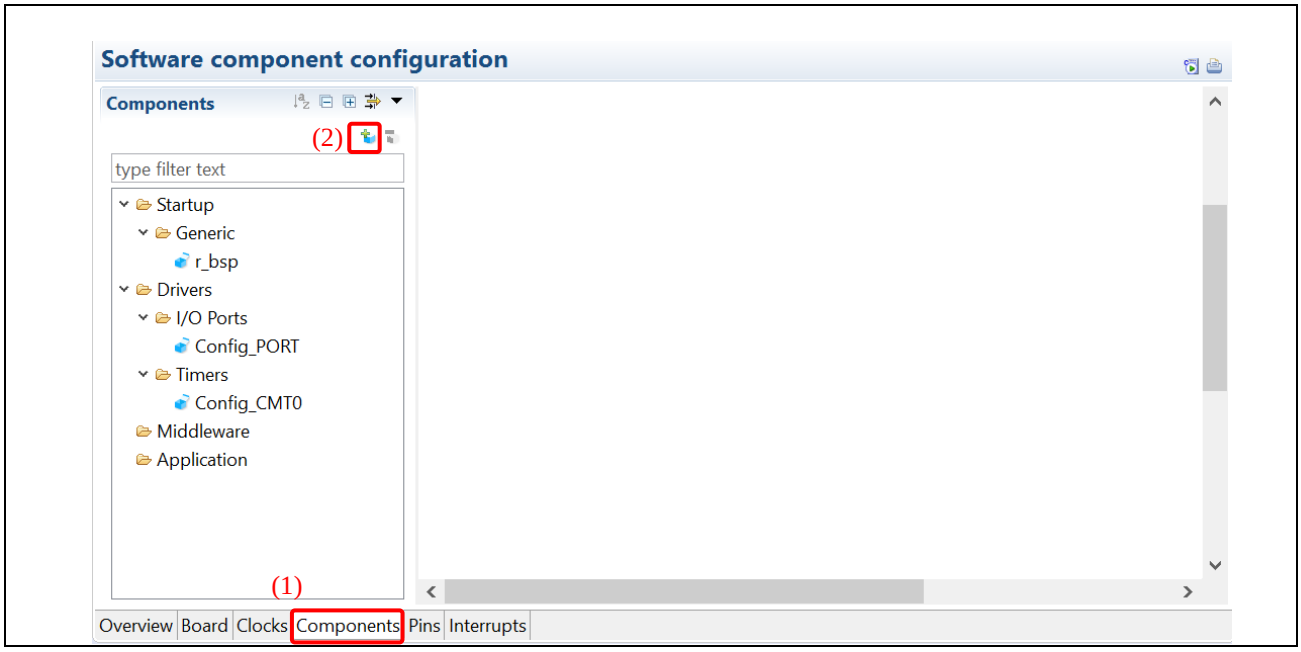


Figure 6-8 Software component configuration in Smart Configurator

3) Add *USB Basic* and *USB PCDC* drivers into the project
   a. Enter “usb” in the “Filter” column
   b. Navigate the component list and select *r_usb_basic* driver
   c. Press and hold Ctrl key, click on *r_usb_pcdc* driver
      Note: If above drivers were not available in the list, click [Download more software components] to download the FIT modules.
   d. Click [Finish]

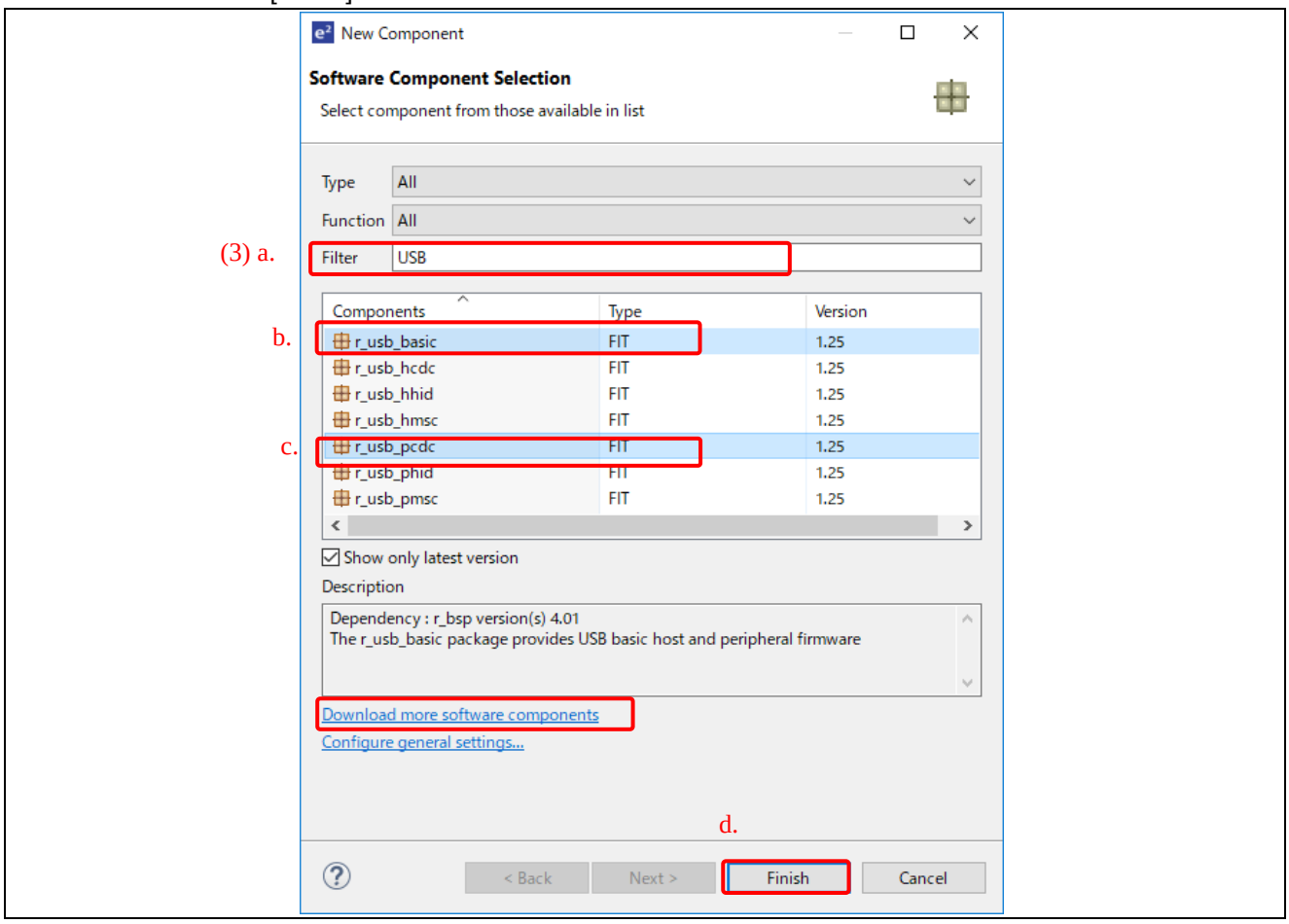


Figure 6-9 Select software component

4) In the component page
   a. Click *r_usb_basic* to open the configure tab
   b. Under “Configuration”, set [Setting whether the received class request is supported.] to “Not supporting the class request”.
   c. Tick the checkbox for [USB0_PERI] & [USB0_VBUS Pin]

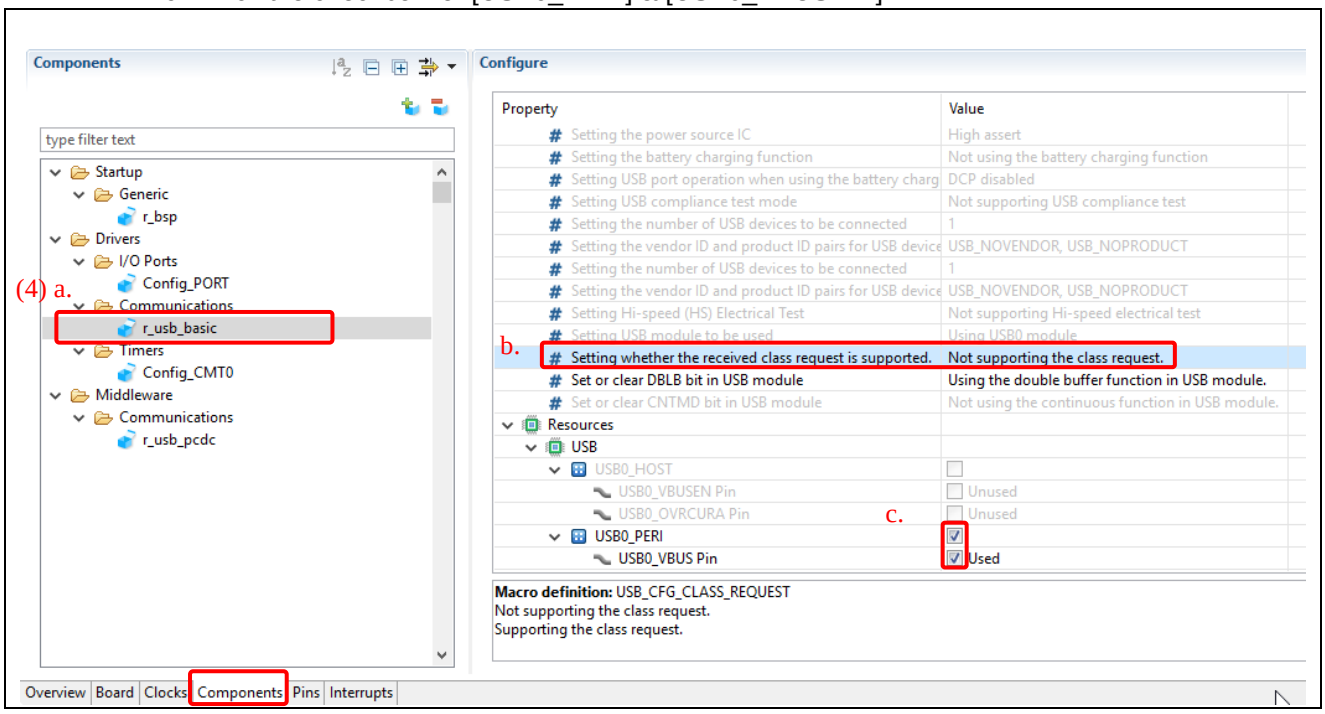


Figure 6-10 configure software component

5) In the pins page
   a. Click
   b. Click *r_usb_basic* to open the corresponding Pin Function tab
   c. Tick the checkbox for [USB0_VBUS]
   d. Click on the corresponding [Assignment] row to assign P16 to function [USB0_VBUS]

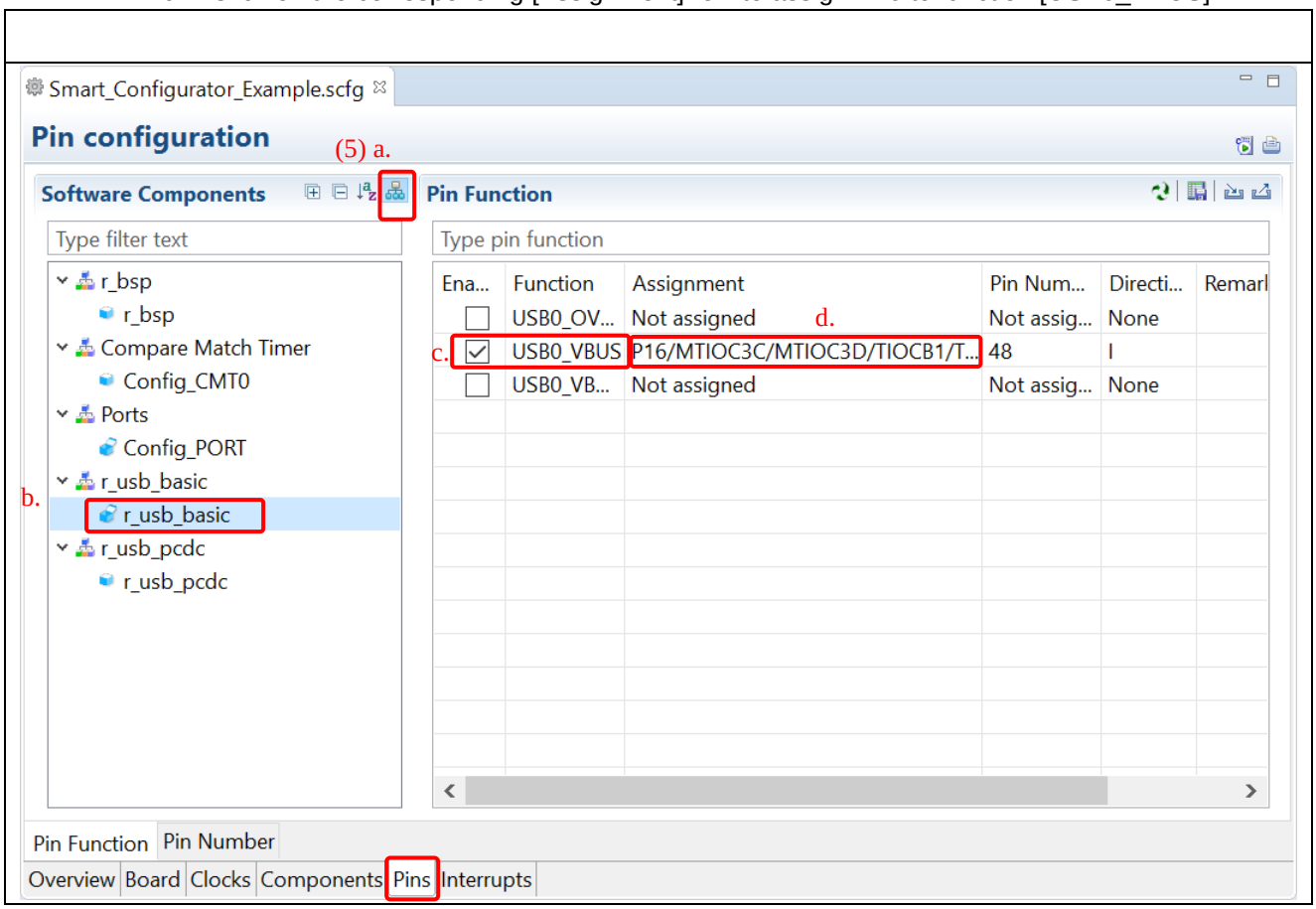


Figure 6-11 configure software component

6.3 Generating codes

1) At [Software Component Configuration] tab, click  to generate codes

2) *USB Basic Host and Peripheral Driver* and *USB PCDC Driver* codes are generated into three folders under \src\smc_gen:

Folder	Sub-folder/File	Description
r_usb_basic	**doc**	Application Note of *USB Basic Host and Peripheral Driver (r_usb_basic)*
	ref	Reference of configuration file
	src	Source files and header files
	r_usb_basic_if.h	List of all API calls and interface definitions of *r_usb_basic* Refer to this file for operation in application
r_usb_pcdc	**doc**	Application Note of *USB PCDC Driver (r_usb_pcdc)*
	ref	Reference of configuration file
	src	Source files and header files
	utilities	Contains system definition file (CDC_Demo_Win7.inf) Install this driver for Windows 7 PC host
	r_usb_pcdc_if.h	List of all API calls and interface definitions of *r_usb_pcdc* Refer to this file for operation in application
r_config	*r_usb_basic_config.h*	Configuration of *r_usb_basic* Configure this file as the requirement of application Refer to Application Note of *r_usb_basic* before modifying this file
	r_usb_pcdc_config.h	Configuration of *r_usb_pcdc* Configure this file as the requirement of application Refer to Application Note of *r_usb_basic* and *r_usb_pcdc* before modifying this file

3) In this application example:

   a. USB driver is configured to peripheral mode (device)

   b. A header file is created for construction of descriptor: r_usb_pcdc_descriptor.h

   c. API functions that are called in application codes: *R_USB_Open, R_USB_Read, R_USB_Write*

   d. USB driver status that are checked for operation in application codes: *USB_STS_WRITE_COMPLETE, USB_STS_READ_COMPLETE*

6.4 Adding header file of USB PCDC driver

1) At Project Explorer tree, right click [src] folder, select [New] → [Header file]

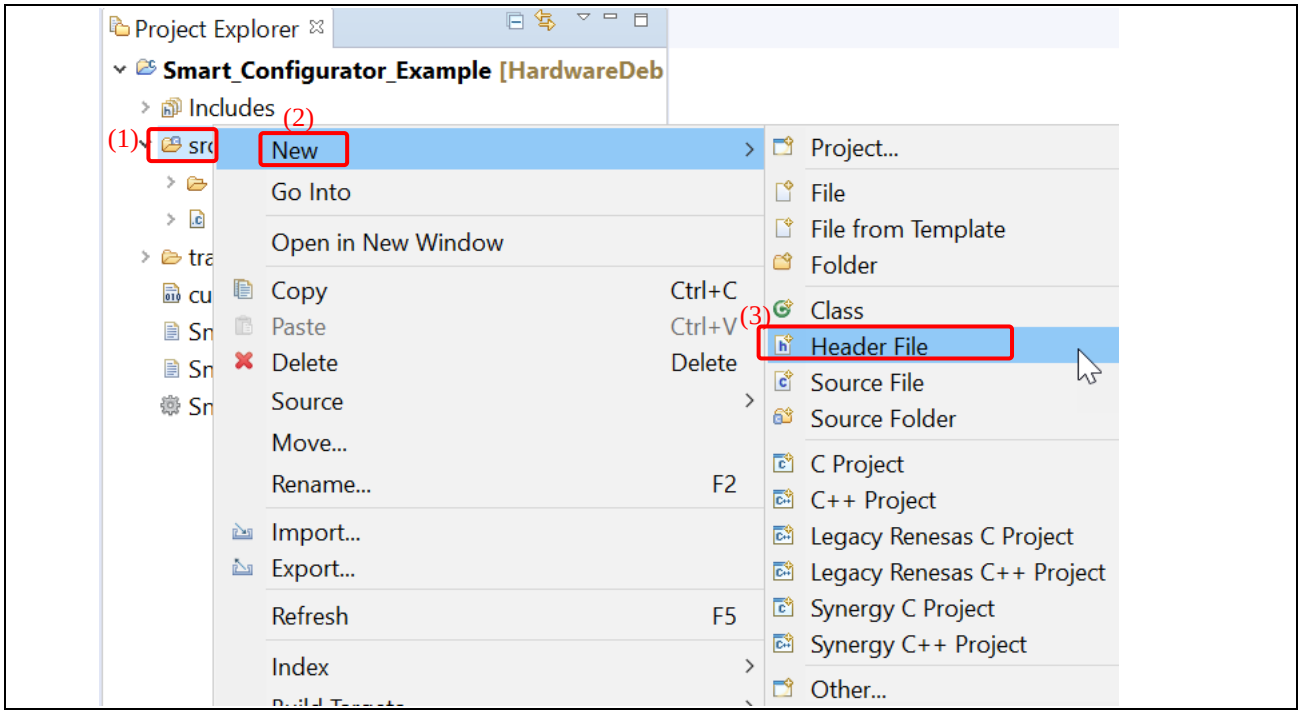


Figure 6-12 Adding head file

2) Input header file name (e.g. *r_usb_pcdc_descriptor.h*), click [Finish]

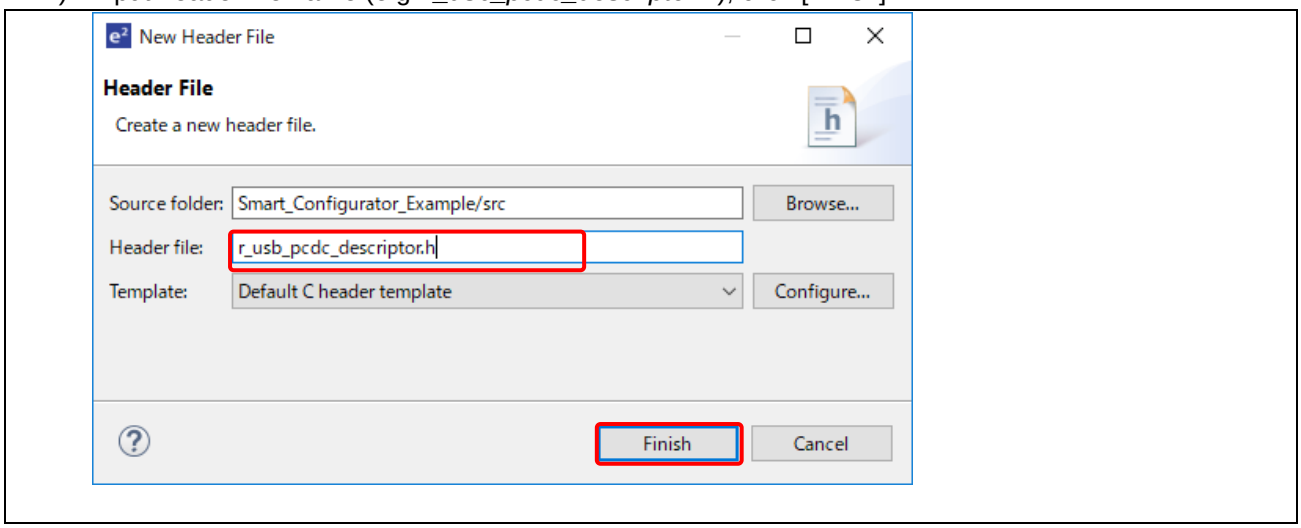


Figure 6-13Adding header file

3) Open “r_usb_pcdc_descriptor.h” file in \src folder

Refer to the application note of ‘USB Basic Host and Peripheral Driver using Firmware Integration Technology’ under \src\smc_gen\r_usb_basic \doc folder or USB-IF website (http://www.usb.org) for more information on USB 2.0 specification

Add below codes in “r_usb_pcdc_descriptor.h” to construct the descriptor for this application example:

Note:

Please be sure to specify your vendor ID and product ID in the device descriptor and INF file.

```
#include "r_usb_basic_if.h"
/******************************************************************************
 Macro definitions
 ******************************************************************************/
/* bcdUSB */
#define USB_BCDNUM       (0x0200u)       /* 0x0100 (USB1.0), 0x0110 (USB1.1), 0x0200 (USB2.0) */
/* Release Number */
#define USB_RELEASE      (0x0200u)       /* 0x0100 (USB1.0), 0x0110 (USB1.1), 0x0200 (USB2.0) */
/* DCP max packet size */
#define USB_DCPMAXP      (64u)           /* Max packet size for endpoint 0. 8/16/32/64 */
/* Configuration number */
#define USB_CONFIGNUM    (1u)            /* Total number of possible configurations for the device */
/* Vendor ID */
#define USB_VENDORID     (0x0000u)       /* Must be obtained from USB-IF */
/* Product ID*/
#define USB_PRODUCTID    (0x0002u)       /* Assigned by the manufacturer */

/* Class-Specific Configuration Descriptors */
#define     USB_PCDC_CS_INTERFACE        (0x24u) /* Assigned by USB-IF*/

/* bDescriptor SubType in Communications Class Functional Descriptors */
/* Header Functional Descriptor */
#define     USB_PCDC_DT_SUBTYPE_HEADER_FUNC                     (0x00u) /* Assigned by USB-IF */
/* Call Management Functional Descriptor. */
#define     USB_PCDC_DT_SUBTYPE_CALL_MANAGE_FUNC                (0x01u) /* Assigned by USB-IF */
/* Abstract Control Management Functional Descriptor. */
#define     USB_PCDC_DT_SUBTYPE_ABSTRACT_CTR_MANAGE_FUNC        (0x02u) /* Assigned by USB-IF */
/* Union Functional Descriptor */
#define     USB_PCDC_DT_SUBTYPE_UNION_FUNC                      (0x06u) /* Assigned by USB-IF */

/* Communications Class Subclass Codes */
#define     USB_PCDC_CLASS_SUBCLASS_CODE_ABS_CTR_MDL     (0x02u) /*SubClass Code assigned by USB-IF */

/* USB Class Definitions for Communications Devices Specification
 release number in binary-coded decimal. */
#define     USB_PCDC_BCD_CDC             (0x0110u) /* Assigned by USB-IF */

/* Descriptor length */
#define USB_PCDC_DD_LEN                  (18u)
#define USB_PCDC_QD_LEN                  (10u)
#define USB_PCDC_CD1_LEN                 (67u)
#define STRING_DESCRIPTOR0_LEN           (4u)
#define STRING_DESCRIPTOR1_LEN           (16u)
#define STRING_DESCRIPTOR2_LEN           (8u)
#define STRING_DESCRIPTOR3_LEN           (22u)
#define STRING_DESCRIPTOR4_LEN           (22u)
#define STRING_DESCRIPTOR5_LEN           (18u)
#define STRING_DESCRIPTOR6_LEN           (28u)

/* Standard Device Descriptor */
uint8_t g_apl_device[USB_PCDC_DD_LEN + ( USB_PCDC_DD_LEN % 2)] =
{
    USB_PCDC_DD_LEN,                                    /* 0:bLength */
    USB_DT_DEVICE,                                      /* 1:bDescriptorType */
    (USB_BCDNUM & (uint8_t) 0xffu),                     /* 2:bcdUSB_lo */
    ((uint8_t) (USB_BCDNUM >> 8) & (uint8_t) 0xffu),    /* 3:bcdUSB_hi */
    USB_IFCLS_CDCC,                                     /* 4:bDeviceClass */
    0,                                                  /* 5:bDeviceSubClass */
    0,                                                  /* 6:bDeviceProtocol */
    (uint8_t) USB_DCPMAXP,                              /* 7:bMAXPacketSize(for DCP) */
    (USB_VENDORID & (uint8_t) 0xffu),                   /* 8:idVendor_lo */
    ((uint8_t) (USB_VENDORID >> 8) & (uint8_t) 0xffu),  /* 9:idVendor_hi */
    ((uint16_t) USB_PRODUCTID & (uint8_t) 0xffu),       /* 10:idProduct_lo */
    ((uint8_t) (USB_PRODUCTID >> 8) & (uint8_t) 0xffu), /* 11:idProduct_hi */
    (USB_RELEASE & (uint8_t) 0xffu),                    /* 12:bcdDevice_lo */
     ((uint8_t) (USB_RELEASE >> 8) & (uint8_t) 0xffu),  /* 13:bcdDevice_hi */
    1,                                                  /* 14:iManufacturer */
    2,                                                  /* 15:iProduct */
    6,                                                  /* 16:iSerialNumber */
    USB_CONFIGNUM                                       /* 17:bNumConfigurations */
};
```

```
/***********************************************************
 *  Configuration or other Speed Configuration Descriptor   *
 ***********************************************************/
/* For Full-Speed */
uint8_t g_apl_configuration[USB_PCDC_CD1_LEN + ( USB_PCDC_CD1_LEN % 2)] =
{
        9,                                              /* 0:bLength */
        USB_SOFT_CHANGE,                                /* 1:bDescriptorType */
        USB_PCDC_CD1_LEN % 256,                         /* 2:wTotalLength(L) */
        USB_PCDC_CD1_LEN / 256,                         /* 3:wTotalLength(H) */
        2,                                              /* 4:bNumInterfaces */
        1,                                              /* 5:bConfigurationValue */
        0,                                              /* 6:iConfiguration */
        USB_CF_RESERVED | USB_CF_SELFP,                 /* 7:bmAttributes */
        (10 / 2),                                       /* 8:MAXPower (2mA unit) */

        /* Interface Descriptor */
        9,                                              /* 0:bLength */
        USB_DT_INTERFACE,                               /* 1:bDescriptor */
        0,                                              /* 2:bInterfaceNumber */
        0,                                              /* 3:bAlternateSetting */
        1,                                              /* 4:bNumEndpoints */
        USB_IFCLS_CDCC,                                 /* 5:bInterfaceClass */
        USB_PCDC_CLASS_SUBCLASS_CODE_ABS_CTR_MDL,       /* 6:bInterfaceSubClass */
        1,                                              /* 7:bInterfaceProtocol */
        0,                                              /* 8:iInterface */

        /* Communications Class Functional Descriptors */
        5,                                              /* 0:bLength */
        USB_PCDC_CS_INTERFACE,                          /* 1:bDescriptorType */
        USB_PCDC_DT_SUBTYPE_HEADER_FUNC,                /* 2:bDescriptorSubtype */
        USB_PCDC_BCD_CDC % 256,                         /* 3:bcdCDC_lo */
        USB_PCDC_BCD_CDC / 256,                         /* 4:bcdCDC_hi */

        /* Communications Class Functional Descriptors */
        4,                                              /* 0:bLength */
        USB_PCDC_CS_INTERFACE,                          /* 1:bDescriptorType */
        USB_PCDC_DT_SUBTYPE_ABSTRACT_CTR_MANAGE_FUNC,   /* 2:bDescriptorSubtype */
        2,                                              /* 3:bmCapabilities */

        /* Communications Class Functional Descriptors */
        5,                                              /* 0:bLength */
        USB_PCDC_CS_INTERFACE,                          /* 1:bDescriptorType */
        USB_PCDC_DT_SUBTYPE_UNION_FUNC,                 /* 2:bDescriptorSubtype */
        0,                                              /* 3:bMasterInterface */
        1,                                              /* 4:bSlaveInterface0 */

        /* Communications Class Functional Descriptors */
        5,                                              /* 0:bLength */
        USB_PCDC_CS_INTERFACE,                          /* 1:bDescriptorType */
        USB_PCDC_DT_SUBTYPE_CALL_MANAGE_FUNC,           /* 2:bDescriptorSubtype */
        /* D1:1-Device can send/receive call management
         information over a Data Class interface. */
        /* D0:1-Device handles call management itself. */
        3,                                              /* 3:bmCapabilities */
        1,                                              /* 4:bDataInterface */

        /* Endpoint Descriptor 0 */
        7,                                              /* 0:bLength */
        USB_DT_ENDPOINT,                                /* 1:bDescriptorType */
        USB_EP_IN | USB_EP3,                            /* 2:bEndpointAddress */
        USB_EP_INT,                                     /* 3:bmAttribute */
        16,                                             /* 4:wMAXPacketSize_lo */
        0,                                              /* 5:wMAXPacketSize_hi */
        0x10,                                           /* 6:bInterval */
```

```
        /* Interface Descriptor */
        9,                                      /* 0:bLength */
        USB_DT_INTERFACE,                       /* 1:bDescriptor */
        1,                                      /* 2:bInterfaceNumber */
        0,                                      /* 3:bAlternateSetting */
        2,                                      /* 4:bNumEndpoints */
        USB_IFCLS_CDCD,                         /* 5:bInterfaceClass */
        0,                                      /* 6:bInterfaceSubClass */
        0,                                      /* 7:bInterfaceProtocol */
        0,                                      /* 8:iInterface */

        /* Endpoint Descriptor 0 */
        7,                                      /* 0:bLength */
        USB_DT_ENDPOINT,                        /* 1:bDescriptorType */
        USB_EP_IN | USB_EP1,                    /* 2:bEndpointAddress */
        USB_EP_BULK,                            /* 3:bmAttribute */
        64,                                     /* 4:wMAXPacketSize_lo */
        0,                                      /* 5:wMAXPacketSize_hi */
        0,                                      /* 6:bInterval */

        /* Endpoint Descriptor 1 */
        7,                                      /* 0:bLength */
        USB_DT_ENDPOINT,                        /* 1:bDescriptorType */
        USB_EP_OUT | USB_EP2,                   /* 2:bEndpointAddress */
        USB_EP_BULK,                            /* 3:bmAttribute */
        64,                                     /* 4:wMAXPacketSize_lo */
        0,                                      /* 5:wMAXPacketSize_hi */
        0,                                      /* 6:bInterval */
};

/***********************************
 *    String Descriptor             *
 ***********************************/
/* UNICODE 0x0409 English (United States) */
uint8_t g_cdc_string_descriptor0[STRING_DESCRIPTOR0_LEN + (STRING_DESCRIPTOR0_LEN % 2)] =
{
        STRING_DESCRIPTOR0_LEN,                 /* 0:bLength */
        USB_DT_STRING,                          /* 1:bDescriptorType */
        0x09, 0x04                              /* 2:wLANGID[0] */
};

/* iManufacturer */
uint8_t g_cdc_string_descriptor1[STRING_DESCRIPTOR1_LEN + (STRING_DESCRIPTOR1_LEN % 2)] =
{
        STRING_DESCRIPTOR1_LEN,                 /* 0:bLength */
        USB_DT_STRING,                          /* 1:bDescriptorType */
        'R', 0x00,                              /* 2:wLANGID[0] */
        'E', 0x00,
        'N', 0x00,
        'E', 0x00,
        'S', 0x00,
        'A', 0x00,
        'S', 0x00,
};

/* iProduct */
uint8_t g_cdc_string_descriptor2[STRING_DESCRIPTOR2_LEN + (STRING_DESCRIPTOR2_LEN % 2)] =
{
        STRING_DESCRIPTOR2_LEN,                 /* 0:bLength */
        USB_DT_STRING,                          /* 1:bDescriptorType */
        'U', 0x00,
        'S', 0x00,
        'B', 0x00,
};
```

```
/* iInterface */
uint8_t g_cdc_string_descriptor3[STRING_DESCRIPTOR3_LEN + (STRING_DESCRIPTOR3_LEN % 2)] =
{
        STRING_DESCRIPTOR3_LEN,                         /* 0:bLength */
        USB_DT_STRING,                                  /* 1:bDescriptorType */
        'C', 0x00,
        'o', 0x00,
        'm', 0x00,
        ' ', 0x00,
        'D', 0x00,
        'e', 0x00,
        'v', 0x00,
        'i', 0x00,
        'c', 0x00,
        'e', 0x00,
};

/* iConfiguration */
uint8_t g_cdc_string_descriptor4[STRING_DESCRIPTOR4_LEN + (STRING_DESCRIPTOR4_LEN % 2)] =
{
        STRING_DESCRIPTOR4_LEN,                         /* 0:bLength */
        USB_DT_STRING,                                  /* 1:bDescriptorType */
        'F', 0x00,                                      /* 2:wLANGID[0] */
        'u', 0x00,
        'l', 0x00,
        'l', 0x00,
        '-', 0x00,
        'S', 0x00,
        'p', 0x00,
        'e', 0x00,
        'e', 0x00,
        'd', 0x00
};


uint8_t *g_apl_string_table[] =
{
        g_cdc_string_descriptor0,
        g_cdc_string_descriptor1,
        g_cdc_string_descriptor2,
        g_cdc_string_descriptor3,
        g_cdc_string_descriptor4,
};

/***********************************************************************************
 Renesas Abstracted Peripheral Communications Devices Class Driver API functions
 ***********************************************************************************/
```

Your source file should look like this:

```
 * r_usb_pcdc_descriptor.h

#ifndef R_USB_PCDC_DESCRIPTOR_H_
#define R_USB_PCDC_DESCRIPTOR_H_
#include "r_usb_basic_if.h"
/***********************************************************************
 Macro definitions
 ***********************************************************************/
/* bcdUSB */
#define USB_BCDNUM                      (0x0200u)/* 0x0100 (USB1.0), 0x0110 (USB1.1), 0x0200 (USB2.0)*/
/* Release Number */
#define USB_RELEASE                     (0x0200u)/* 0x0100 (USB1.0), 0x0110 (USB1.1), 0x0200 (USB2.0)*/
/* DCP max packet size */
#define USB_DCPMAXP                     (64u)  /*Max packet size for endpoint 0.Must be 8, 16, 32 or 64*/
/* Configuration number */
#define USB_CONFIGNUM                   (1u)/*Specifies the total number of possible configurations for the device.*/
/* Vendor ID */
#define USB_VENDORID                    (0x0000u)/*must be obtained from USB-IF*/
/* Product ID*/
#define USB_PRODUCTID                   (0x0002u)/*assigned by the manufacturer */

/* Class-Specific Configuration Descriptors */
#define     USB_PCDC_CS_INTERFACE                               (0x24u) /*assigned from USB-IF*/

/* bDescriptor SubType in Communications Class Functional Descriptors */
/* Header Functional Descriptor */
#define     USB_PCDC_DT_SUBTYPE_HEADER_FUNC                     (0x00u)/*assigned from USB-IF*/
/* Call Management Functional Descriptor. */
#define     USB_PCDC_DT_SUBTYPE_CALL_MANAGE_FUNC                (0x01u)/*assigned from USB-IF*/
/* Abstract Control Management Functional Descriptor. */
#define     USB_PCDC_DT_SUBTYPE_ABSTRACT_CTR_MANAGE_FUNC        (0x02u)/*assigned from USB-IF*/
/* Union Functional Descriptor */
#define     USB_PCDC_DT_SUBTYPE_UNION_FUNC                      (0x06u)/*assigned from USB-IF*/

/* Communications Class Subclass Codes */
#define     USB_PCDC_CLASS_SUBCLASS_CODE_ABS_CTR_MDL            (0x02u) /*SubClass Code assigned by USB-IF*/

/* USB Class Definitions for Communications Devices Specification
 release number in binary-coded decimal. */
#define     USB_PCDC_BCD_CDC          (0x0110u) /*assigned from USB-IF*/

/* Descriptor length */
#define USB_PCDC_DD_LEN             (18u)
#define USB_PCDC_QD_LEN             (10u)
#define USB_PCDC_CD1_LEN            (67u)
#define STRING_DESCRIPTOR0_LEN      (4u)
#define STRING_DESCRIPTOR1_LEN      (16u)
#define STRING_DESCRIPTOR2_LEN      (8u)
#define STRING_DESCRIPTOR3_LEN      (22u)
#define STRING_DESCRIPTOR4_LEN      (22u)
#define STRING_DESCRIPTOR5_LEN      (18u)
#define STRING_DESCRIPTOR6_LEN      (28u)

/* Standard Device Descriptor */
uint8_t g_apl_device[USB_PCDC_DD_LEN + ( USB_PCDC_DD_LEN % 2)] =
{
    USB_PCDC_DD_LEN,                                    /*  0:bLength */
    USB_DT_DEVICE,                                      /*  1:bDescriptorType */
    (USB_BCDNUM & (uint8_t) 0xffu),                     /*  2:bcdUSB_lo */
    ((uint8_t) (USB_BCDNUM >> 8) & (uint8_t) 0xffu),    /*  3:bcdUSB_hi */
    USB_IFCLS_CDCC,                                     /*  4:bDeviceClass */
    0,                                                  /*  5:bDeviceSubClass */
    0,                                                  /*  6:bDeviceProtocol */
    (uint8_t) USB_DCPMAXP,                              /*  7:bMAXPacketSize(for DCP) */
    (USB_VENDORID & (uint8_t) 0xffu),                   /*  8:idVendor_lo */
    ((uint8_t) (USB_VENDORID >> 8) & (uint8_t) 0xffu),  /*  9:idVendor_hi */
    ((uint16_t) USB_PRODUCTID & (uint8_t) 0xffu),       /* 10:idProduct_lo */
    ((uint8_t) (USB_PRODUCTID >> 8) & (uint8_t) 0xffu), /* 11:idProduct_hi */
    (USB_RELEASE & (uint8_t) 0xffu),                    /* 12:bcdDevice_lo */
    ((uint8_t) (USB_RELEASE >> 8) & (uint8_t) 0xffu),   /* 13:bcdDevice_hi */
    1,                                                  /* 14:iManufacturer */
    2,                                                  /* 15:iProduct */
    6,                                                  /* 16:iSerialNumber */
    USB_CONFIGNUM                                       /* 17:bNumConfigurations */
};
```

```
 *  Configuration Or Other_Speed_Configuration Descriptor   *
/* For Full-Speed */
uint8_t g_apl_configuration[USB_PCDC_CD1_LEN + ( USB_PCDC_CD1_LEN % 2)] =
{
    9,                              /*  0:bLength */
    USB_SOFT_CHANGE,                /*  1:bDescriptorType */
    USB_PCDC_CD1_LEN % 256,         /*  2:wTotalLength(L) */
    USB_PCDC_CD1_LEN / 256,         /*  3:wTotalLength(H) */
    2,                              /*  4:bNumInterfaces */
    1,                              /*  5:bConfigurationValue */
    0,                              /*  6:iConfiguration */
    USB_CF_RESERVED | USB_CF_SELFP, /*  7:bmAttributes */
    (10 / 2),                       /*  8:MAXPower (2mA unit) */

    /* Interface Descriptor */
    9,                              /*  0:bLength */
    USB_DT_INTERFACE,               /*  1:bDescriptor */
    0,                              /*  2:bInterfaceNumber */
    0,                              /*  3:bAlternateSetting */
    1,                              /*  4:bNumEndpoints */
    USB_IFCLS_CDCC,                 /*  5:bInterfaceClass */
    USB_PCDC_CLASS_SUBCLASS_CODE_ABS_CTR_MDL,   /*  6:bInterfaceSubClass */
    1,                              /*  7:bInterfaceProtocol */
    0,                              /*  8:iInterface */

    /* Communications Class Functional Descriptors */
    5,                          /*  0:bLength */
    USB_PCDC_CS_INTERFACE,      /*  1:bDescriptorType */
    USB_PCDC_DT_SUBTYPE_HEADER_FUNC,    /*  2:bDescriptorSubtype */
    USB_PCDC_BCD_CDC % 256,     /*  3:bcdCDC_lo */
    USB_PCDC_BCD_CDC / 256,     /*  4:bcdCDC_hi */

    /* Communications Class Functional Descriptors */
    4,                          /*  0:bLength */
    USB_PCDC_CS_INTERFACE,      /*  1:bDescriptorType */
    USB_PCDC_DT_SUBTYPE_ABSTRACT_CTR_MANAGE_FUNC,   /*  2:bDescriptorSubtype */
    2,                          /*  3:bmCapabilities */

    /* Communications Class Functional Descriptors */
    5,                          /*  0:bLength */
    USB_PCDC_CS_INTERFACE,      /*  1:bDescriptorType */
    USB_PCDC_DT_SUBTYPE_UNION_FUNC, /*  2:bDescriptorSubtype */
    0,                          /*  3:bMasterInterface */
    1,                          /*  4:bSlaveInterface0 */

    /* Communications Class Functional Descriptors */
    5,                          /*  0:bLength */
    USB_PCDC_CS_INTERFACE,      /*  1:bDescriptorType */
    USB_PCDC_DT_SUBTYPE_CALL_MANAGE_FUNC,   /*  2:bDescriptorSubtype */
    /* D1:1-Device can send/receive call management
     information over a Data Class interface. */
    /* D0:1-Device handles call management itself. */
    3,                          /*  3:bmCapabilities */
    1,                          /*  4:bDataInterface */

    /* Endpoint Descriptor 0 */
    7,                          /*  0:bLength */
    USB_DT_ENDPOINT,            /*  1:bDescriptorType */
    USB_EP_IN | USB_EP3,        /*  2:bEndpointAddress */
    USB_EP_INT,                 /*  3:bmAttribute */
    16,                         /*  4:wMAXPacketSize_lo */
    0,                          /*  5:wMAXPacketSize_hi */
    0x10,                       /*  6:bInterval */

    /* Interface Descriptor */
    9,                              /*  0:bLength */
    USB_DT_INTERFACE,               /*  1:bDescriptor */
    1,                              /*  2:bInterfaceNumber */
    0,                              /*  3:bAlternateSetting */
    2,                              /*  4:bNumEndpoints */
    USB_IFCLS_CDCD,                 /*  5:bInterfaceClass */
    0,                              /*  6:bInterfaceSubClass */
    0,                              /*  7:bInterfaceProtocol */
    0,                              /*  8:iInterface */

    /* Endpoint Descriptor 0 */
    7,                          /*  0:bLength */
    USB_DT_ENDPOINT,            /*  1:bDescriptorType */
    USB_EP_IN | USB_EP1,        /*  2:bEndpointAddress */
    USB_EP_BULK,                /*  3:bmAttribute */
    64,                         /*  4:wMAXPacketSize_lo */
    0,                          /*  5:wMAXPacketSize_hi */
    0,                          /*  6:bInterval */
```

```
        /* Endpoint Descriptor 1 */
        7,                              /*  0:bLength */
        USB_DT_ENDPOINT,                /*  1:bDescriptorType */
        USB_EP_OUT | USB_EP2,           /*  2:bEndpointAddress */
        USB_EP_BULK,                    /*  3:bmAttribute */
        64,                             /*  4:wMAXPacketSize_lo */
        0,                              /*  5:wMAXPacketSize_hi */
        0,                              /*  6:bInterval */
};
/**********************************
 *    String Descriptor             *
 **********************************/
/* UNICODE 0x0409 English (United States) */
uint8_t g_cdc_string_descriptor0[STRING_DESCRIPTOR0_LEN + ( STRING_DESCRIPTOR0_LEN % 2)] =
{
    STRING_DESCRIPTOR0_LEN,     /*  0:bLength */
    USB_DT_STRING,              /*  1:bDescriptorType */
    0x09, 0x04                  /*  2:wLANGID[0] */
};
/* iManufacturer */
uint8_t g_cdc_string_descriptor1[STRING_DESCRIPTOR1_LEN + ( STRING_DESCRIPTOR1_LEN % 2)] =
{
    STRING_DESCRIPTOR1_LEN,     /*  0:bLength */
    USB_DT_STRING,              /*  1:bDescriptorType */
    'R', 0x00,                  /*  2:wLANGID[0] */
    'E', 0x00,
    'N', 0x00,
    'E', 0x00,
    'S', 0x00,
    'A', 0x00,
    'S', 0x00,
};
/* iProduct */
uint8_t g_cdc_string_descriptor2[STRING_DESCRIPTOR2_LEN + ( STRING_DESCRIPTOR2_LEN % 2)] =
{
    STRING_DESCRIPTOR2_LEN, /*  0:bLength */
    USB_DT_STRING,          /*  1:bDescriptorType */
    'U', 0x00,
    'S', 0x00,
    'B', 0x00,
};
/* iInterface */
uint8_t g_cdc_string_descriptor3[STRING_DESCRIPTOR3_LEN + ( STRING_DESCRIPTOR3_LEN % 2)] =
{
    STRING_DESCRIPTOR3_LEN, /*  0:bLength */
    USB_DT_STRING,          /*  1:bDescriptorType */
    'C', 0x00,
    'o', 0x00,
    'm', 0x00,
    ' ', 0x00,
    'D', 0x00,
    'e', 0x00,
    'v', 0x00,
    'i', 0x00,
    'c', 0x00,
    'e', 0x00,
};
/* iConfiguration */
uint8_t g_cdc_string_descriptor4[STRING_DESCRIPTOR4_LEN + ( STRING_DESCRIPTOR4_LEN % 2)] =
{
    STRING_DESCRIPTOR4_LEN, /*  0:bLength */
    USB_DT_STRING,          /*  1:bDescriptorType */
    'F', 0x00,              /*  2:wLANGID[0] */
    'u', 0x00,
    'l', 0x00,
    'l', 0x00,
    '-', 0x00,
    'S', 0x00,
    'p', 0x00,
    'e', 0x00,
    'e', 0x00,
    'd', 0x00
};
uint8_t *g_apl_string_table[] =
{
    g_cdc_string_descriptor0,
    g_cdc_string_descriptor1,
    g_cdc_string_descriptor2,
    g_cdc_string_descriptor3,
    g_cdc_string_descriptor4,
};
/***********************************************************************
 Renesas Abstracted Peripheral Communications Devices Class Driver API functions
 ***********************************************************************/
#endif /* R_USB_PCDC_DESCRIPTOR_H_ */
```

Figure 6-14 r_usb_pcdc_descriptor.h

6.5 Adding application codes in *main()*

1) In Smart_Configurator_Example.c, add header files includes and declarations near the top of the file

```
#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"
#include "r_usb_pcdc_descriptor.h"
#include <stdio.h>
#include <string.h>

void R_USB_PinSet_USB0_PERI();          /* Initialize USB0_VBUS pin */
static uint8_t  g_buf[1];               /* Variable to store input
character from PC terminal */
volatile unsigned int flag_start = 1;   /* Flag to print start message
*/
volatile uint16_t interval_level = 1;   /* Variable to change CMT0
interval */
static char print_str[120];             /* String to print message at
PC terminal */
volatile uint32_t slength;              /* String length */

extern uint8_t g_apl_device[];
extern uint8_t g_apl_configuration[];
extern uint8_t *g_apl_string_table[];
const static usb_descriptor_t usb_descriptor =
{
    g_apl_device,                   /* Pointer to the device descriptor
*/
    g_apl_configuration,            /* Pointer to the configuration
descriptor for Full-speed */
    USB_NULL,                       /* Pointer to the configuration
descriptor for Hi-speed */
    USB_NULL,                       /* Pointer to the qualifier
descriptor */
    g_apl_string_table              /* Pointer to the string descriptor
table */
};
```

2) Add below codes into *main()* function before *R_Config_CMT0_Start()*

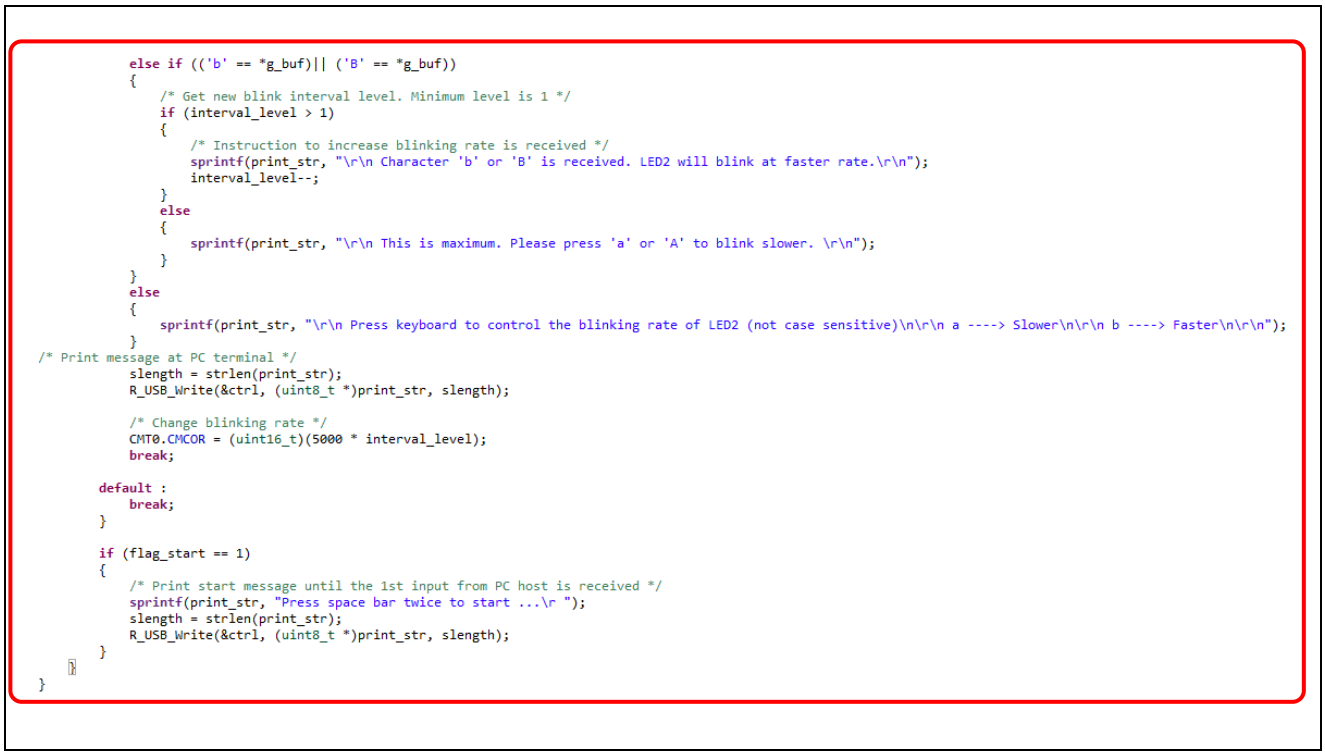
```
    usb_ctrl_t  ctrl;
    usb_cfg_t   cfg;
```

3) Add below codes in the same function after *R_Config_CMT0_Start()* to initialize USB0 and start the communication with terminal program on the PC host:

```
R_USB_PinSet_USB0_PERI();     /* USB MCU pin setting */

    ctrl.module     = USB_IP0;    /* USB0 module */
    ctrl.type       = USB_PCDC;   /* Peripheral Communication Device Class*/
    cfg.usb_speed   = USB_FS;     /* USB_HS/USB_FS */
    cfg.usb_mode    = USB_PERI;
    cfg.p_usb_reg   = (usb_descriptor_t *)&usb_descriptor;
    R_USB_Open(&ctrl, &cfg);      /* Initializes the USB module */

         /* Loop back between PC Terminal and USB MCU */
         while (1)
         {
                  switch (R_USB_GetEvent(&ctrl))
                  {
                  case USB_STS_CONFIGURED :
                           break;

                  case USB_STS_WRITE_COMPLETE :
                           /* Read the input from PC terminal*/
                           R_USB_Read(&ctrl, g_buf, 1);
                           break;

                  case USB_STS_READ_COMPLETE :
                           /* Clear start message flag*/
                           flag_start = 0;
                           /* Instruction to slow down blinking rate is received */
                           if (('a' == *g_buf)|| ('A' == *g_buf))
                           {
                                    /* Get new blink interval level. Maximum level is 9. */
                                    if (interval_level < 10)
                                    {/* Notify the character received and inform next action */
                                    sprintf(print_str, "\r\n Character 'a' or 'A' is received. LED2 will blink
at slower rate.\r\n");
                                    interval_level++;
                                    }
                                    else
                                    {
                                    sprintf(print_str, "\r\n This is minimum. Please press 'b' or 'B' to blink
faster. \r\n");
                                    }
                           }
                           else if (('b' == *g_buf)|| ('B' == *g_buf))
                           {
                                    /* Get new blink interval level. Minimum level is 1 */
                                    if (interval_level > 1)
                                    {
                                             /* Instruction to increase blinking rate is received */
                                    sprintf(print_str, "\r\n Character 'b' or 'B' is received. LED2 will blink
at faster rate.\r\n");
                                    interval_level--;
                                    }
                                    else
                                    {
                                    sprintf(print_str, "\r\n This is maximum. Please press 'a' or 'A' to blink
slower. \r\n");
                                    }
                           }
                           else
                           {
                           sprintf(print_str, "\r\n Press keyboard to control the blinking rate of LED2 (not
case sensitive)\n\r\n a ----> Slower\n\r\n b ----> Faster\n\r\n");
                           }
                           /* Print message at PC terminal */
                           slength = strlen(print_str);
                           R_USB_Write(&ctrl, (uint8_t *)print_str, slength);

                           /* Change blinking rate */
                           CMT0.CMCOR = (uint16_t)(5000 * interval_level);
                           break;

                  default :
                           break;
                  }

                  if (flag_start == 1)
                  {
                           /* Print start message until the 1st input from PC host is received */
                           sprintf(print_str, "Press space bar twice to start ...\r ");
                           slength = strlen(print_str);
                           R_USB_Write(&ctrl, (uint8_t *)print_str, slength);
         }
}
```

Your source file should look like this:

```
#include "r_smc_entry.h"

#include "r_usb_basic_if.h"
#include "r_usb_pcdc_if.h"
#include "r_usb_pcdc_descriptor.h"
#include <stdio.h>
#include <string.h>

void R_USB_PinSet_USB0_PERI();          /* Initialize USB0_VBUS pin */
static uint8_t  g_buf[1];               /* Variable to store input character from PC terminal */
volatile unsigned int flag_start = 1;   /* Flag to print start message */
volatile uint16_t interval_level = 1;   /* Variable to change CMT0 interval */
static char print_str[120];             /* String to print message at PC terminal */
volatile uint32_t slength;              /* String length */

extern uint8_t g_apl_device[];
extern uint8_t g_apl_configuration[];
extern uint8_t *g_apl_string_table[];
const static usb_descriptor_t usb_descriptor =
{
    g_apl_device,                   /* Pointer to the device descriptor */
    g_apl_configuration,            /* Pointer to the configuration descriptor for Full-speed */
    USB_NULL,                       /* Pointer to the configuration descriptor for Hi-speed */
    USB_NULL,                       /* Pointer to the qualifier descriptor */
    g_apl_string_table              /* Pointer to the string descriptor table */
};

[redacted]

void main(void)
{
    usb_ctrl_t  ctrl;
    usb_cfg_t   cfg;

    /* Start CMT0 counter operation */
    R_Config_CMT0_Start();

    usb_pin_setting();              /* USB MCU pin setting */

    ctrl.module     = USB_IP0;      /* USB0 module */
    ctrl.type       = USB_PCDC;     /* Peripheral Communication Device Class*/
    cfg.usb_speed   = USB_FS;       /* USB_HS/USB_FS */
    cfg.usb_mode    = USB_PERI;
    cfg.p_usb_reg   = (usb_descriptor_t *)&usb_descriptor;
    R_USB_PinSet_USB0_PERI();       /* Initializes the USB module */

    /* Loop back between PC Terminal and USB MCU */
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
        case USB_STS_CONFIGURED :
            break;

        case USB_STS_WRITE_COMPLETE :
            /* Read the input from PC terminal*/
            R_USB_Read(&ctrl, g_buf, 1);
            break;

        case USB_STS_READ_COMPLETE :
            /* Clear start message flag*/
            flag_start = 0;
            /* Instruction to slow down blinking rate is received */
            if (('a' == *g_buf)|| ('A' == *g_buf))
            {
                /* Get new blink interval level. Maximum level is 9. */
                if (interval_level < 10)
                {
                    /* Notify the character received and inform next action */
                    sprintf(print_str, "\r\n Character 'a' or 'A' is received. LED2 will blink at slower rate.\r\n");
                    interval_level++;
                }
                else
                {
                    sprintf(print_str, "\r\n This is minimum. Please press 'b' or 'B' to blink faster. \r\n");
                }
            }
```



```
            else if (('b' == *g_buf)|| ('B' == *g_buf))
            {
                /* Get new blink interval level. Minimum level is 1 */
                if (interval_level > 1)
                {
                    /* Instruction to increase blinking rate is received */
                    sprintf(print_str, "\r\n Character 'b' or 'B' is received. LED2 will blink at faster rate.\r\n");
                    interval_level--;
                }
                else
                {
                    sprintf(print_str, "\r\n This is maximum. Please press 'a' or 'A' to blink slower. \r\n");
                }
            }
            else
            {
                sprintf(print_str, "\r\n Press keyboard to control the blinking rate of LED2 (not case sensitive)\n\r\n a ----> Slower\n\r\n b ----> Faster\n\r\n");
            }
/* Print message at PC terminal */
            slength = strlen(print_str);
            R_USB_Write(&ctrl, (uint8_t *)print_str, slength);

            /* Change blinking rate */
            CMT0.CMCOR = (uint16_t)(5000 * interval_level);
            break;

        default :
            break;
        }

        if (flag_start == 1)
        {
            /* Print start message until the 1st input from PC host is received */
            sprintf(print_str, "Press space bar twice to start ...\r ");
            slength = strlen(print_str);
            R_USB_Write(&ctrl, (uint8_t *)print_str, slength);
        }
    }
}
```

Figure 6-15 USB PCDC application codes in *main* function

6.6 Build and run on hardware board

After build the project, click debug  icon to debug the project.

Step1. Use a USB to mini B cable to connect PC host and USB function connector on Renesas Starter Kit+ for RX65N 2MB (refer to Chapter 5.8 for the USB function port location)

Step 2. At e2 studio Debug perspective, click to execute the project, the project status will show in Running status  at the left bottom of IDE.

Step 3. Open a terminal program in PC (for e.g. Tera Term). Perform the following steps:

- Select Serial radio button
- In the Port option, select COM that is connected to *USB Serial Device*
- Click [OK]

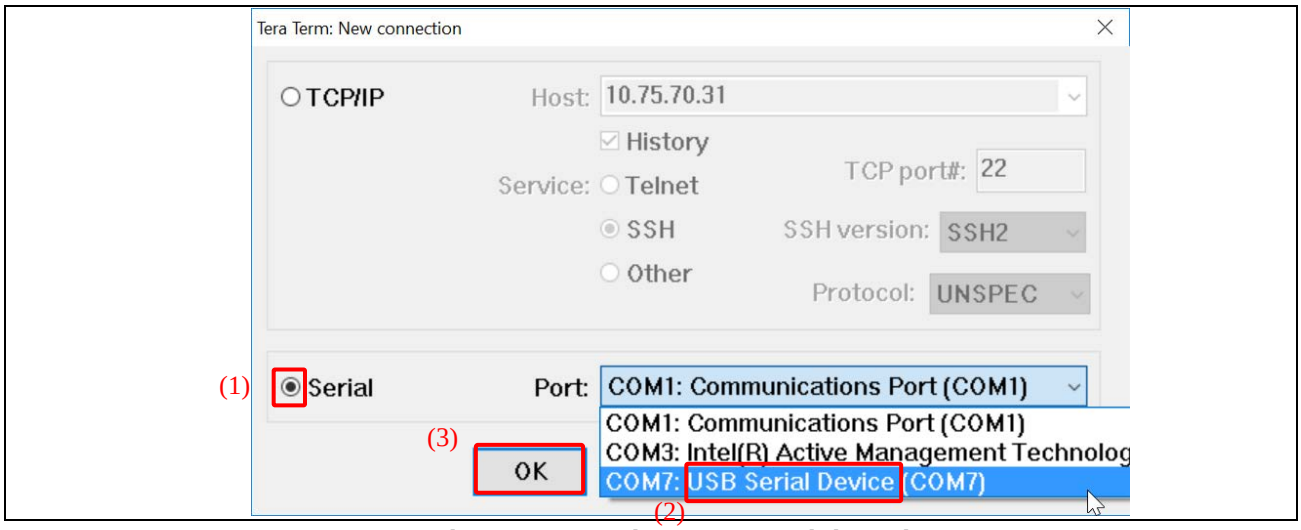


Figure 6-16 Select USB Serial Device

Note:

For Windows 7 PC, device driver software may not be installed successfully.

Install the system definition file separately from \src\smc_gen\r_usb_pcdc\utilities\CDC_Demo_Win7.inf

After installation, terminal program will display COM as "CDC USB Demonstration".

Step 4. Follow the message on the terminal program to increase or decrease LED2 blinking rate:

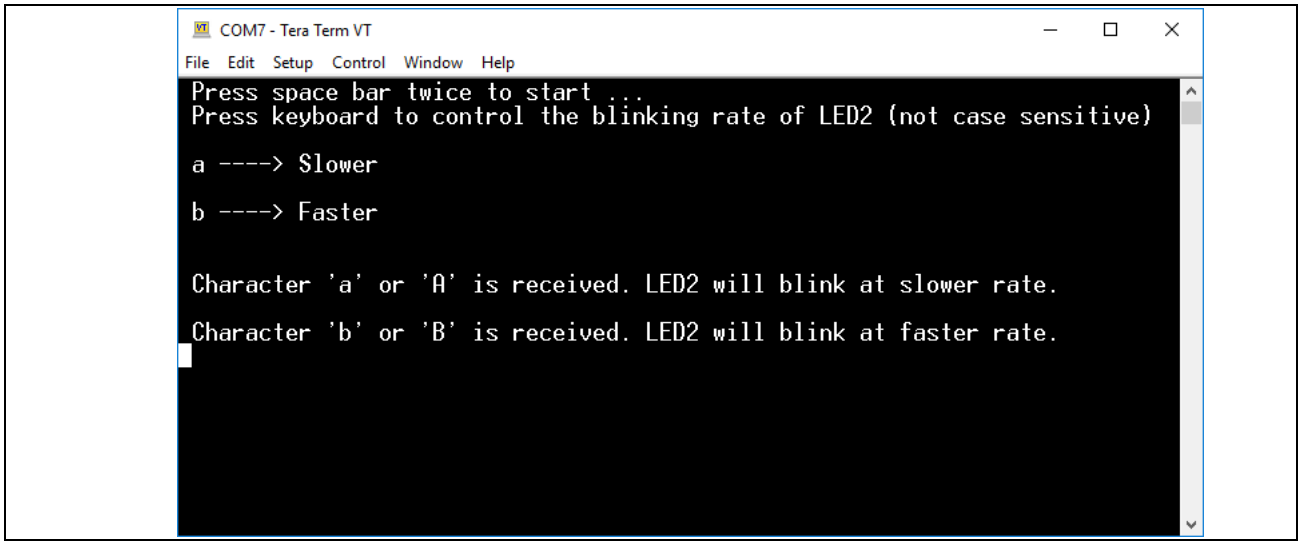


Figure 6-17 Terminal window in PC

6.7 Operation on board

On the Renesas Starter Kit+ for RX65N 2MB board:

Blinking rate of LED2 is changed when key in character ‘a’ or ‘b’ at the PC terminal:

- a : Blinking of LED2 will slow down
- b : Blinking of LED2 will become faster

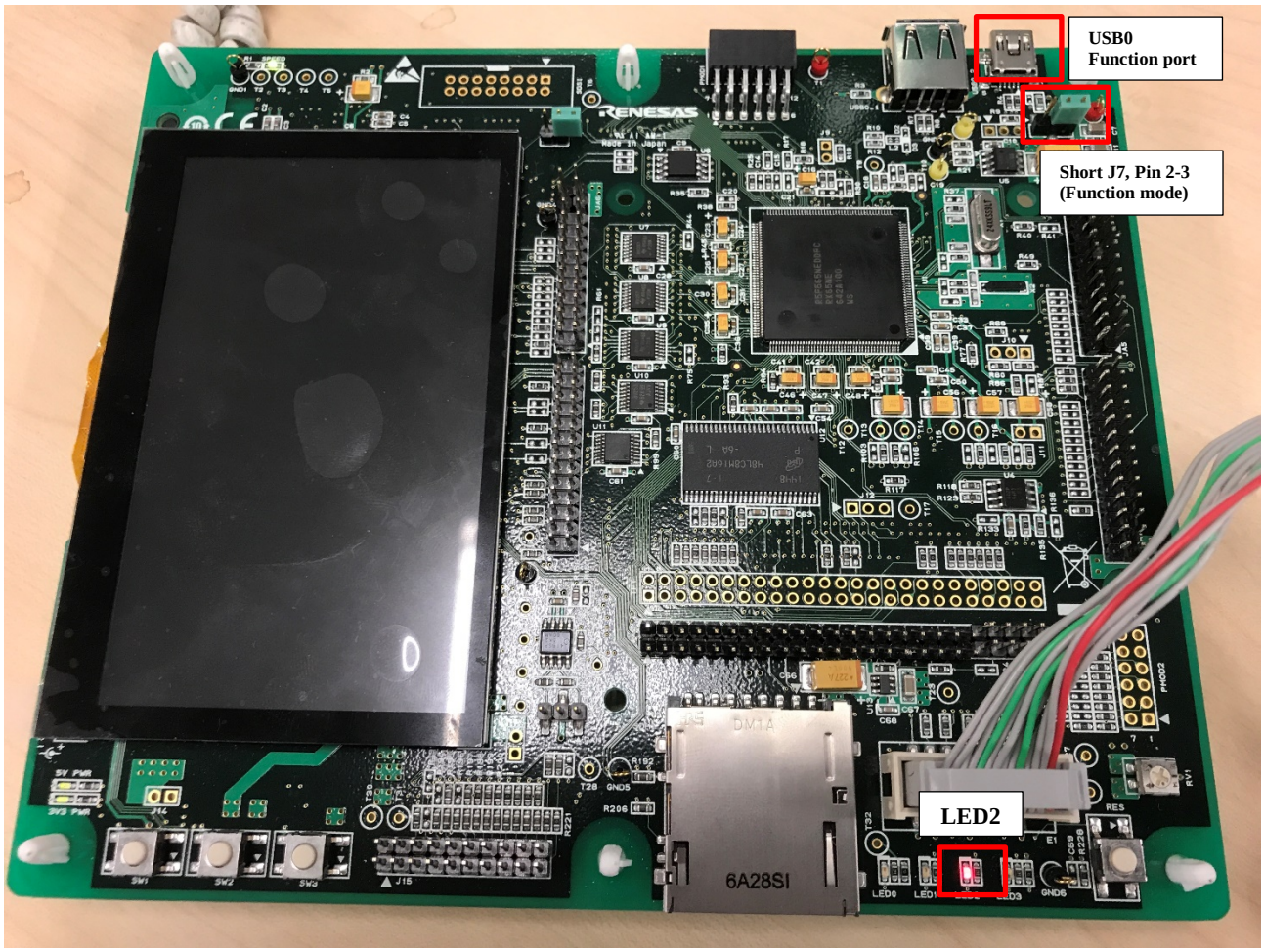


Figure 6-18 LED2 and USB Function Port on RX65N 2MB board

6.8 Additional debugging assistance tool (QE)

Renesas has a range of Quick and Effective Tool Solutions (QEs) as development tools for particular applications to assist and improve efficiency during development. Specific to this example of integrating USB function, QE for USB is recommended as it has several features to assist in debugging application based on USB function.

Please refer to the following link for more information on QE and the type of supported applications for Renesas IDE:

QE: https://www.renesas.com/qe

QE for USB: https://www.renesas.com/qe-usb

Website and Support

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

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

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	July 31, 2017	-	First creation
1.10	Mar 30,2018	1	Document title, changed
		10	Updated based on e^{2}studio V6.2 Figure 2-9 C Project - Target Specific Settings example with E2 Lite emulator, changed
		13	Updated based on e^{2}studio V6.2 Figure 2-13 Select software component, changed
		15	Updated based on e^{2}studio V6.2 Figure 2-16 Select software component, changed
		29	Updated based on e^{2}studio V6.2 Figure 3-4 Select software component, changed
		40	Updated based on e^{2}studio V6.2 Figure 4-8 Select software component, changed
		61	Updated based on e^{2}studio V6.2 Figure 6-9 Select software component, changed
		62	Updated based on e^{2}studio V6.2 Figure 6-10 configure software component, added
		63	Updated based on e^{2}studio V6.2 Figure 6-11 configure software component, added
		65	5.4 Modifying the default configuration, deleted
		74	Updated based on e^{2}studio V6.2 Revise code to adjust to new smart configurator
1.20	Jun 20,2019	36	Updated Application example 3 to use SCI8
		54	Added Application 4 – Transferring data using DMA

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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

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/.