RX23W Group

Guide for Concurrent Use of Bluetooth Low Energy and USB

Summary

This document describes the things to keep in mind when you implement applications that uses Bluetooth Low Energy and USB 2.0 communications concurrently in the RX23W MCU. Also, this introduces the sample program that uses Bluetooth Low Energy and USB 2.0 communications concurrently.

The following contexts are described as the things to keep in mind.

- The implementation of Application and the settings of Bluetooth Low Energy operation to maintain the performance of Bluetooth LE and USB
- The overview and the priorities of tasks to control Bluetooth Low Energy and USB
- How to check the conditions for transitions to the low power consumption state of MCU and BLE module
- How to use API to use a clock output from CLKOUT_RF pin as a USB clock

Tested Device

Renesas Solution Starter Kit for RX23W (RTK5523W8AC00001BJ)

Table of Contents

1. Example of Assumed System Configuration

This document describes the things to keep in mind when you use Central or Peripheral operation of Bluetooth Low Energy (LE) and Peripheral operation of USB 2.0 communications concurrently in the RX23W MCU.

Figure 1-1 shows an example of the configuration of the system assumed by this document. The BLE FIT Module and the GATT Service code generated by QE for BLE are used for Bluetooth LE communication. The USB Basic Mini FIT Module and the USB Class FIT Modules such as PCDC (Peripheral Communication Device Class) are used for USB communication.

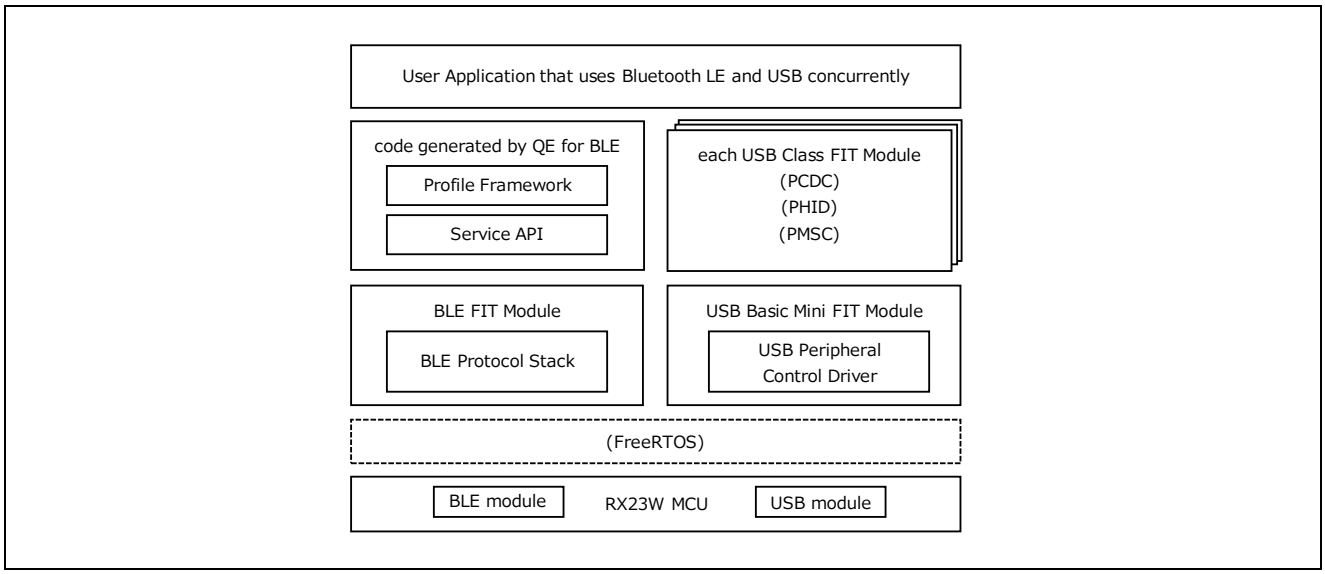

Figure 1-1 Example of Assumed System Configuration

For more information on the RX23W MCU, BLE FIT Module, QE for BLE, USB Basic FIT Module, and each USB Class PCDC FIT module, refer to the documents published in the following websites.

RX23W
https://www.renesas.com/rx23w

- RX23W Group User's Manual: Hardware (R01UH0823)

Bluetooth® Low Energy Protocol Stack for RX family
https://www.renesas.com/software-tool/bluetooth-low-energy-protocol-stack-rx-family

- RX23W Group BLE Module Firmware Integration Technology (R01AN4860)
- RX23W Group Bluetooth Low Energy Application Developer's Guide (R01AN5504)
- RX23W Group Bluetooth Low Energy Profile Developer's Guide (R01AN4553)

QE for BLE: Development Assistance Tool for Bluetooth® Low Energy
https://www.renesas.com/qe-ble

USB Drivers
https://www.renesas.com/software-tool/usb-drivers

- RX Family USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology (R01AN2166)

2. Things To Keep in Mind

2.1 Controlling Bluetooth LE and USB Communications

BLE Protocol Stack and USB Driver cannot complete the control of the peripheral modules with interrupt handlers only and needs additional control processing in a main loop or on an OS task. To maintain performance of Bluetooth LE and USB communications, application must be implemented not to delay the execution of these additional processing.

2.1.1 Operation Overview of BLE Protocol Stack and USB Driver

The BLE Protocol Stack manages the processing (tasks) that control the BLE module of the RX23W with a queue. When an application calls the BLE API or an interrupt is generated from the BLE module, a task is added to the queue. Tasks added to the queue are executed by the `R_BLE_Execute()` which is called by a main loop in an OS-less environment or which is called by a loop on the FreeRTOS task in a FreeRTOS environment.

Similarly, the USB Peripheral Control Driver manages the processing (tasks) that control the USB module of the RX23W with a queue. When an application calls the USB API or an interrupt is generated from the USB module, a task is added to the queue. Tasks added to the queue are executed by the `R_USB_GetEvent()` which is called by a main loop in an OS-less environment, or executed by the USB peripheral control driver task `usb_pstd_pcd_task()` in a FreeRTOS environment.

Table 2-1 Task Processing Function

Peripheral Module	Task Processing Function	
	OS-less	FreeRTOS
BLE module	`R_BLE_Execute()`	
USB module	`R_USB_GetEvent()`	`usb_pstd_pcd_task()`

Figure 2-1 shows an example of the operation of the BLE Protocol Stack and the USB Peripheral Control Driver. When an interrupt (BLEIRQ) is generated from the BLE module, an interrupt handler: `r_ble_bleirq_interrupt_func()` adds a task to the BLE control queue. The added task is executed by `R_BLE_Execute()`. When an interrupt (USBI) is generated from the USB module, an interrupt handler: `usb_cpu_usb_interrupt()` adds a task to the USB control queue. The added task is executed by `R_USB_GetEvent()`.

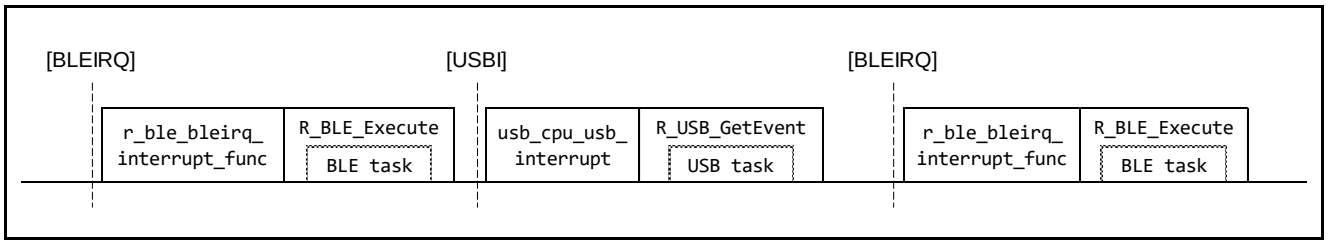

Figure 2-1 Example of the Operation of BLE Protocol Stack and USB Peripheral Control Driver

2.1.2 Influence of Long Processing Time of Application Running in Main Loop

RX23W performs BLE control, USB control, and other processing such as applications with single CPU. If an application running in the main loop occupies the CPU for a long time, it affects BLE control and USB control. Especially, Bluetooth LE communicates with adjusting the timing of transmission and reception, so a connection may be terminated if BLE control processing is not performed in time. To reduce execution delay for BLE control and USB control, when implementing applications and other processing, shorten the processing time of interrupt handler and the interrupt disabled period as much as possible. When OS is not used, shorten the processing time of applications in a main loop as much as possible.

Figure 2-2 shows an example of the operation of application and BLE control processing. When application processing: app_func() in a main loop consumes a long processing time, as shown in [A] in the figure, the delay time from the r_ble_breirq_interrupt_func() to the execution of R_BLE_Execute() is increased. Divide the application processing in the main loop into app_func_a() and app_func_b(), as shown in [B] and [C] in the figure, to shorten the execution delay to the execution of R_BLE_Execute(). Especially, it is recommended that the delay time is within a few milliseconds during a connection of Bluetooth LE. If the execution time of applications, consider using the FreeRTOS.

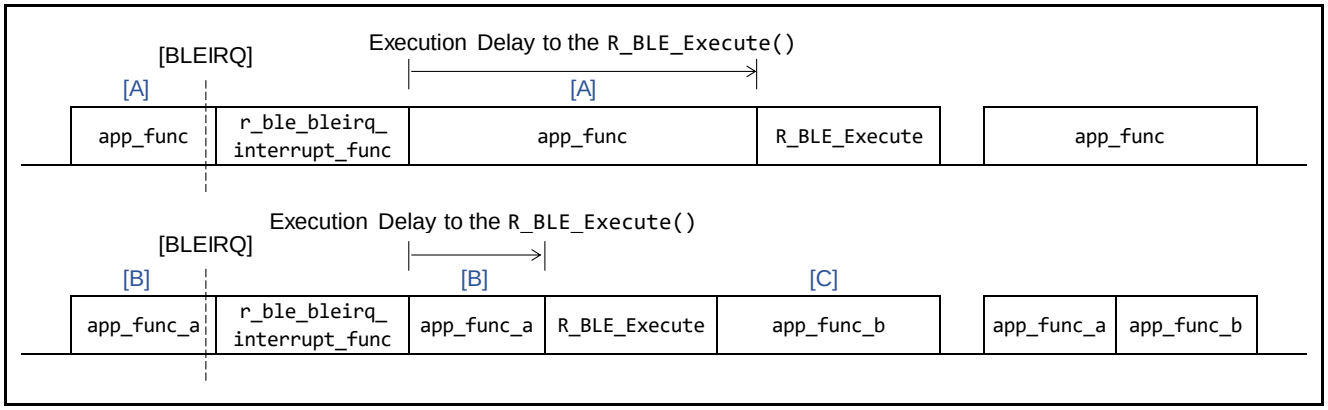

Figure 2-2 Example of the Operation of Application and BLE Control Processing

2.1.3 Cases Where the Processing Time of R_BLE_Execute() Becomes Longer

The R_BLE_Execute() in the BLE Protocol Stack executes the tasks in the BLE control queue sequentially. This function does not return control until all tasks in the BLE control queue have been executed and the queue is empty. Depending on the situation of tasks in the BLE control queue, the CPU may be occupied by R_BLE_Execute() for a long time, and subsequent processing such as USB control executed in the main loop may be delayed.

When data is received continuously from peer device and interrupts are generated, as shown in [A], [B], and [C] in Figure 2-3, tasks are added continuously to the BLE control queue and R_BLE_Execute() may occupy the CPU for a long time. To reduce long CPU occupancy by R_BLE_Execute(), adjust Scan Interval and Scan Window, Connection Interval and Maximum Connection Event Length (Max_CE_Length) respectively to reduce the frequency of Scan and Connection Events.

For example, in the case of Scan Interval = 60ms and Scan Window = 60ms, the duty cycle of Scan Event is 100%. By changing the Scan Window to 30ms, the duty cycle of Scan Event becomes 50%, and the frequency of interrupt by receiving Advertising packet and CPU occupancy by R_BLE_Execute() will be reduced.

Similarly, in the case of Connection Interval = 50ms and Maximum Connection Event Length = 50ms, the maximum duty cycle of Connection Event is 100%. By changing the Maximum Connection Event Length to 25ms, the duty cycle of Connection Event becomes 50%, and the frequency of interrupt by receiving data packet and CPU occupancy by R_BLE_Execute() will be reduced.

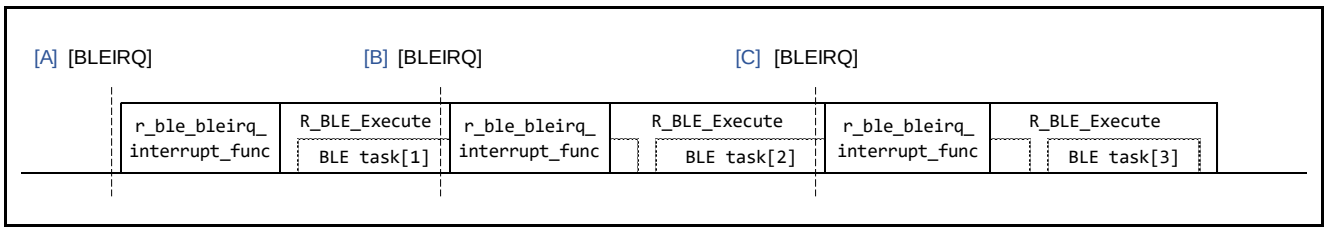

Figure 2-3 Bluetooth LE Communication Events with High Frequency

A callback function registered in BLE Protocol Stack is also executed by a task of R_BLE_Execute(). When the processing time of the callback function is long, as shown in [A] in Figure 2-4, subsequent tasks in the BLE control queue may be delayed and the performance of Bluetooth LE communication may be degraded. Shorten the processing time of the callback function as much as possible.

For example, in the case that 244 bytes of GATT Service data is received continuously during connection with 1M-PHY, transmission and reception operation is performed at intervals of 2.5ms. In this condition, callback function is called at intervals of 2.5ms, so it is recommended that the execution time of callback function is within 2.5ms.

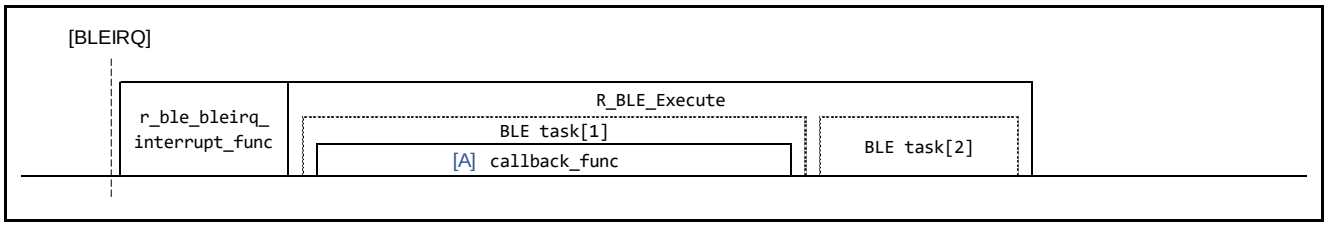

Figure 2-4 Bluetooth LE Event Callback Function with Long Processing Time

R_BLE_SetEvent() of BLE Protocol Stack allows any user function to be added to the BLE control queue as a task. The added user function is executed by R_BLE_Execute(). However, if the processing time of the added user function is long, as shown in [A] in Figure 2-5, subsequent tasks in the BLE control queue may be delayed and the performance of Bluetooth LE communication may be degraded. If the processing time of the user function is long, divide it into two or more functions and add them to the BLE control queue again using R_BLE_SetEvent(), as shown in [B] and [C] in the figure.

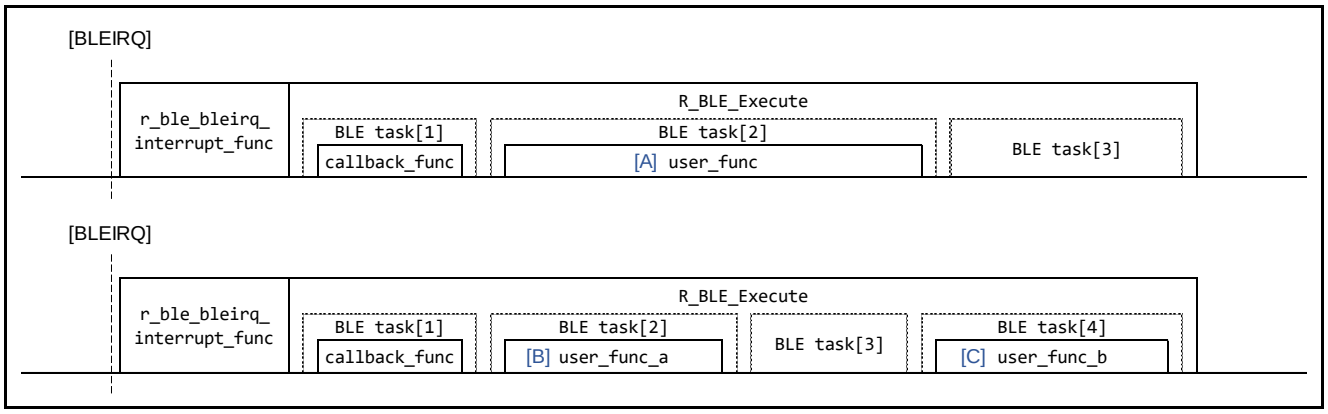

Figure 2-5 Operation of User Functions Added to BLE Control Queue

2.1.4 Influence of the Connection Interval of Bluetooth LE

Figure 2-6 shows an example of the transmission and reception operation of connected Bluetooth LE devices. After a connection of Bluetooth LE is established, Central device and Peripheral device perform transmission and reception operation alternately at each connection interval and can transmit data alternately at one connection event. Data which application requests to transmit at [A] or [B] in the figure is sent at the next connection event [C] in the figure.

The longer the Connection Interval (up to 4 sec), the less frequent the transmission and reception operations and the less CPU occupancy and power consumption, but the longer the delay between the data transmission request of the application to BLE Protocol Stack and the actual transmission.

The shorter the Connection Interval (a minimum of 7.5 msec), the shorter the transmission delay, but the higher the power consumption of the MCU and BLE module because it performs transmission and reception operation even when there is no data to transmit. Also, because BLE module control processing of the BLE Protocol Stack runs more frequently, the CPU time available for application is reduced.

Determine the Connection Interval suitable for each system requirements while keeping in mind the above. Also, the Connection Interval can be changed dynamically during a connection.

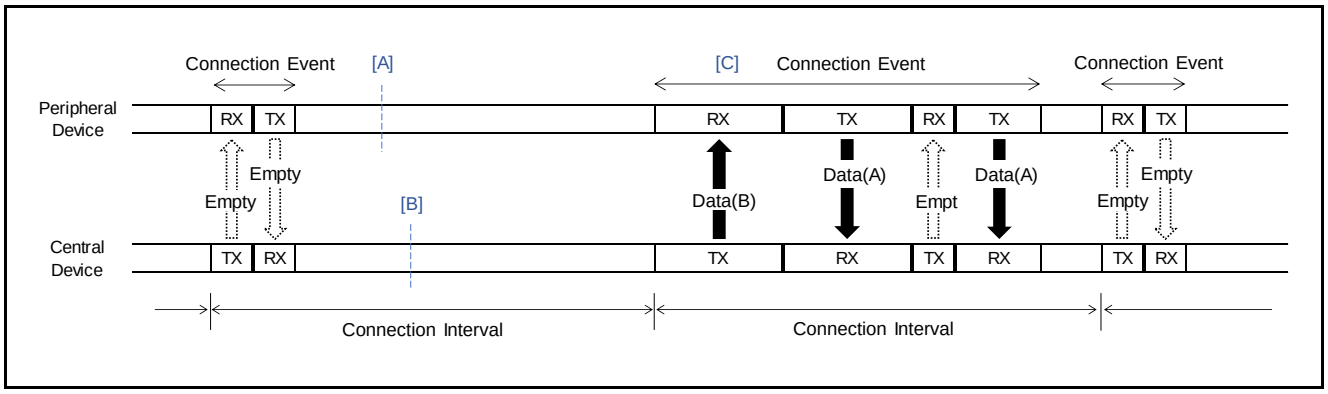

Figure 2-6 Example of Transmission and Reception of Connected Bluetooth LE Devices

In addition, Figure 2-7 shows an example of the transmission and reception operation of connected Bluetooth LE devices when Peripheral Latency is 2. The Peripheral Latency is a reduced number of Connection Events that is performed by Peripheral device. By setting the Peripheral Latency to the value other than zero, Peripheral device does not perform the number of connection events specified by the Peripheral Latency and is able to reduce power consumption. Also, Central device always performs connection events, so Peripheral device can perform connection events to transmit and/or receive data irrespective of the Peripheral Latency.

The [A] in the figure shows a data transmission request by an application of Central device. When Peripheral device does not perform a connection event according to the Peripheral Latency, data transmitted from Central device cannot be received by Peripheral device. Central device retransmit the data in each connection event until Peripheral device receives the data. Due to this reason, when the Peripheral Latency is set, the delay until Peripheral device completes to receive the data of Central device.

Determine the Peripheral Latency suitable for each system requirements while keeping in mind the above. Also, the Peripheral Latency can be changed dynamically during a connection.

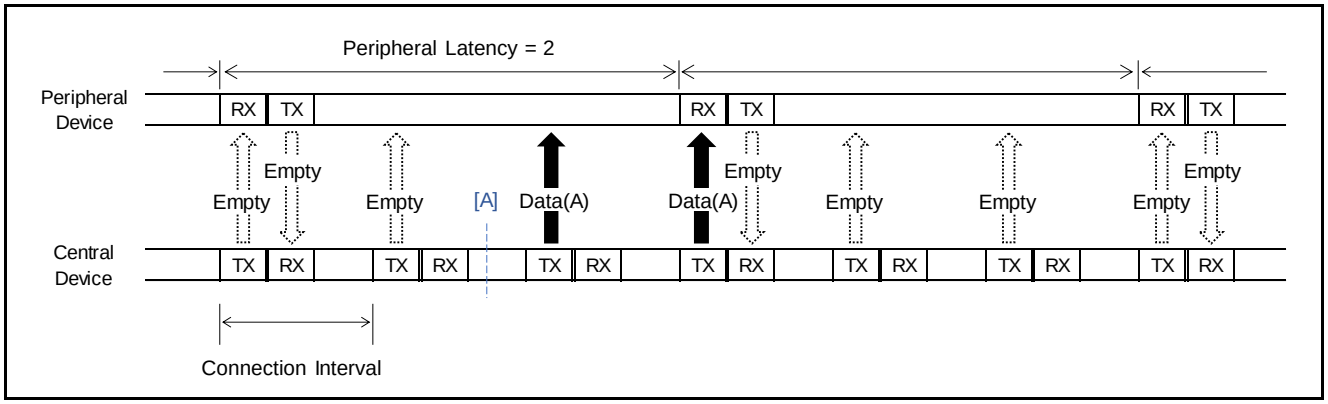

Figure 2-7 Example of Transmission and Reception of Connected Bluetooth LE Devices (Peripheral Latency = 2)

2.1.5 Necessity of Flow Control of USB Communication

Figure 2-8 shows an example of the transmission and reception operation of connected USB devices. The [A] to [D] in the figure shows data transmission requests by the application. USB communication (USB 2.0) is half-duplex and USB Host device controls reception or transmission. If the Host device continues to transmit data, as shown in [C] in the figure, the Peripheral device cannot transmit data. If necessary, implement flow control to the application of Host device side to prevent the Host device from being unable to receive data for a long time.

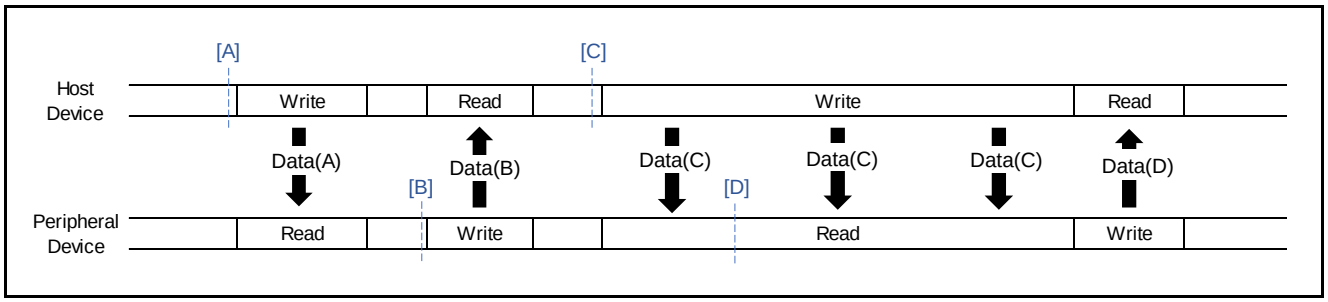

Figure 2-8 Example of Transmission and Reception of Connected USB Devices

2.1.6 Necessity of Data Retransmission of Bluetooth LE Communication and USB Communication

BLE Protocol Stack and USB Driver may miss the received data due to a failure of dynamic resource allocation. If necessary, it is recommended that the application verifies the integrity of the received data and implement a mechanism to request data retransmission if missing a received data is detected.

2.2 FreeRTOS Task Setting

Applications must be implemented to shorten the delay of performing BLE Protocol Stack or USB Driver. When the execution time of application will be long or multiple applications will be implemented, using FreeRTOS is recommended.

2.2.1 FreeRTOS Tasks for BLE Control and USB Control

In the case that BLE Protocol Stack is used on FreeRTOS, a task for controlling BLE module by repeatedly calling `R_BLE_Execute()`.

Because the callback function registered in the BLE Protocol Stack is executed by `R_BLE_Execute()`, if the processing time of application that uses the BLE event as a trigger is long, create a new task for the application.

When USB Peripheral Control Driver is used on FreeRTOS, the USB Peripheral Control Driver task (`usb_pstd_pcd_task`) for controlling USB module is generated automatically.

Table 2-2 Required Tasks on FreeRTOS

Task	Priority	Overview
BLE Task (`app_main`)	12 (recommended for concurrent use with USB)	Task for controlling the BLE module by calling `R_BLE_Execute()` repeatedly The callback functions that notify BLE events are executed in this context The BLE task is included in the code generated by QE for BLE
BLE GATT Application Tasks (Any task name)	0 to 7	Tasks for data transmission and reception with GATT services User implements.
USB Peripheral Control Driver Task (`usb_pstd_pcd_task`)	11 (default)	Task for controlling USB module A callback function that notifies USB events is executed in this context Task generated by the USB Peripheral Control Driver automatically
USB Application Task (Any task name)	0 to 7	Tasks for data transmission and reception with USB User implements.

The BLE task for BLE Protocol Stack must control BLE module according to timing parameters, such as Connection Interval. Therefore, the BLE task must have a higher priority than the BLE GATT application task.

The USB Peripheral Control Driver task requires high-speed control of reading and writing from/to the FIFO buffer to ensure transmission and reception capabilities of the USB module. Therefore, the USB Peripheral Control Driver task must have a higher priority than the USB Application task.

It is recommended to give a higher priority than the USB Peripheral Control Driver task to the BLE task to ensure the Bluetooth LE connection performance. Also, it is necessary to evaluate each application task priority in each use case.

2.3 Reduction of Power Consumption

When the state of MCU is changed to the low power consumption state, CPU and peripheral modules stop. To change the state of MCU to the low power consumption mode without losing functionality, it is necessary to check that all used peripheral modules meet the condition for the transition to the low power consumption mode. When also using Bluetooth LE and USB, application must check the BLE Protocol Stack and the USB Driver meet the condition for the transition.

2.3.1 Low Power Consumption mode of the MCU

Power consumption can be reduced by changing the state of RX23W MCU from the normal operation mode, which is the program operation state, to any one of the sleep mode, deep sleep mode, or software standby mode, which are low power consumption states. Operating conditions in each mode differ depending on peripheral modules. Before changing the state of MCU to any one of low power consumption modes, application must check that the BLE Protocol Stack and USB Peripheral Control Driver are in a condition that can be changed to the low power consumption modes.

Return from the sleep mode and deep sleep mode can be done by using non-maskable interrupts and all interrupts as a trigger, while return from software standby mode can be done by using only non-maskable interrupts and a part of interrupts as a trigger. BLEIRQ of the BLE module and USBR of the USB module are interrupts that can be used as a trigger to return from the software standby mode.

2.3.2 Transition to MCU Low Power Consumption Mode When Using FreeRTOS

FreeRTOS with default settings uses CMT module of RX23W uses for the system timer. CMT module and other peripheral modules stops their operations in the Software standby mode. To change the state of MCU to the Software standby mode in a FreeRTOS environment, it is necessary to verify stopping the system timer does not affect operations of software work on FreeRTOS.

2.3.3 Condition of BLE for Transition to MCU Low Power Consumption

In the case of using BLE Protocol Stack, when BLE control queue in the BLE Protocol Stack is empty, the state of MCU can be changed to the low power consumption mode. Whether the BLE control queue is empty can be checked with `R_BLE_IsTaskFree()`. `R_BLE_LPC_EnterLowPowerMode()` is also provided for stopping and restarting other peripheral modules and managing the low power consumption mode of the MCU. To prevent access to the stopped peripheral modules, disable the context switching during execution of `R_BLE_LPC_EnterLowPowerMode()` in a FreeRTOS environment.

After the transition to the low power consumption mode, the BLE Protocol Stack restores the state of MCU to normal operating mode by the interrupt generated by the BLE module.

2.3.4 Condition for Transition to RF Sleep Mode of BLE Module

The state of BLE module is independent of the mode of MCU, so BLE module continue the operation even if MCU is in the low power state. BLE protocol stack reduces power consumption by changing the state of BLE module to RF Sleep mode automatically. To change the state of BLE module to RF Sleep mode, sufficient RF suspended period is needed between the Bluetooth LE events.

Figure 2-9 shows the RF suspended period between the Bluetooth LE events. In each Bluetooth LE operation, it is necessary to adjust the Advertising Interval, Scan Interval, Scan Window, and Connection Interval so that RF suspended period becomes 80ms or longer. If the execution period of the connection event is extended, as shown in [A] in the figure, and the subsequent RF suspended period is less than 80 ms, as shown in [B] in the figure, the BLE module does not enter RF Sleep mode during this RF suspended period. Similarly, the BLE module does not enter RF sleep mode if the RF suspended period is shortened by establishing connections with multiple Bluetooth LE devices.

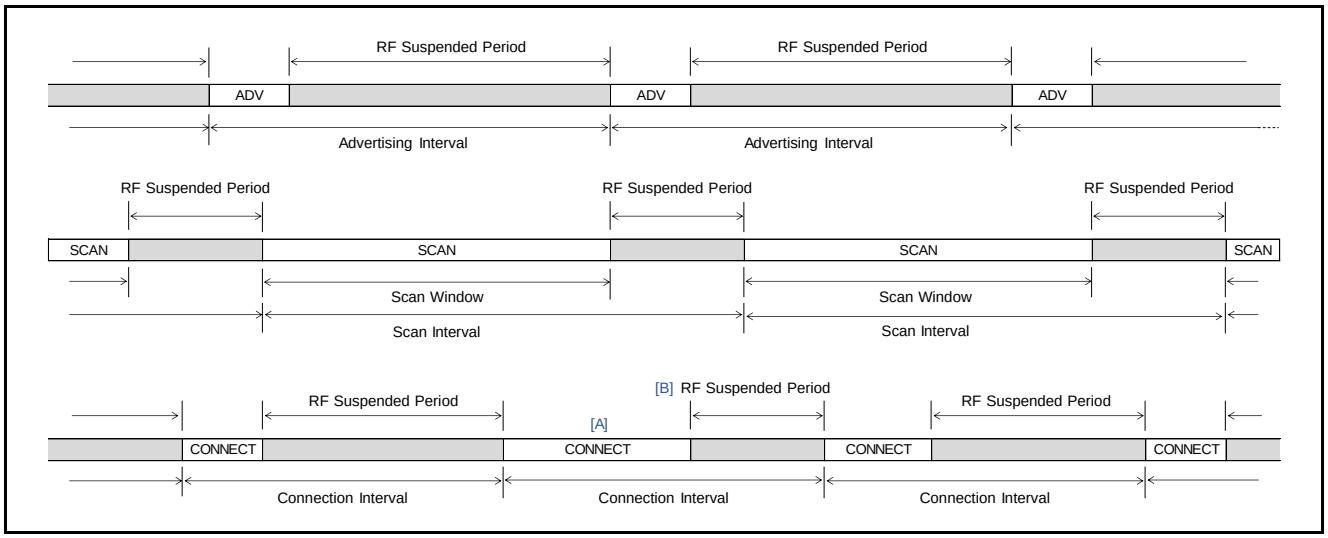

Figure 2-9 RF Suspended Period between the Bluetooth LE Events

2.3.5 Condition of USB for Transition to MCU Low Power Consumption

In the case of using USB Driver, the state of MCU can be changed to the low power consumption mode depending on the condition of USB module. Table 2-3 shows the MCU mode transition condition of which the sample program published in the following website.

USB Drivers
https://www.renesas.com/software-tool/usb-drivers

- RX Family USB Peripheral Communications Device Class Driver (PCDC) Firmware Integration Technology (R01AN2296)

The status of VBUS can be checked with the VBSTS bit of the Interrupt Status Register (INTSTS0). The USB Device State can be checked with the DVSQ[2:0] bit of the Interrupt Status Register (INTSTS0). The USB Device State can be also checked with the value of `info.status` returned by `R_USB_GetInformation()`.

Table 2-3 USB Condition for MCU Mode Transition

MCU Mode	Transition Condition	
	VBUS	USB Device State (`info.status`)
Software standby mode	OFF	Powered (`USB_STS_DETACH`)
Sleep mode	ON	Suspend (`USB_STS_SUSPEND`)
Normal operating mode	ON	Other than Suspend (Other than `USB_STS_SUSPEND`)

2.4 Clock Setting

4MHz clock output from CLKOUT_RF pin can be used as a base clock for generating USB Clock (UCLK) that is required by USB module operation. In this configuration, an external resonator for Main clock oscillation is not needed, and that is why the BOM cost of your product will be reduced.

2.4.1 Setting for Using the CLKOUT_RF pin Output Clock as USB Clock

RX23W can output a clock of any of the frequencies of 1, 2, or 4 MHz based on the Bluetooth dedicated 32 MHz clock from the CLKOUT_RF pin. The output clock of the CLKOUT_RF pin can be input from the EXTAL pin and used as a Main clock. By using 4MHz clock output to CLKOUT_RF pin as the Main clock and multiplying it by 12 with USB dedicated PLL circuit, it can also be used as a 48 MHz USB clock. For an example of configuration of Smart Configurator, refer to Figure 3-2.

To permit clock output to the CLKOUT_RF pin, change the `BLE_CFG_RF_CLKOUT_EN` macro of the BLE FIT module. The clock output to the CLKOUT_RF pin starts by the execution of `R_BLE_Open()`.

When the output clock of the CLKOUT_RF pin is used as the main clock, set the `BSP_CFG_CLKOUT_RF_MAIN` macro of the BSP FIT module to 1. After the `R_BLE_Open()` is executed, the oscillation of the High-speed On-Chip Oscillator (HOCO) can be stopped.

The clock output to the CLKOUT_RF pin stops when `R_BLE_Close()` is executed. When the output clock to the CLKOUT_RF pin is used as the main clock, do not execute `R_BLE_Close()` after the HOCO oscillation stops. When the output clock of the CLKOUT_RF pin is also used as an USB clock, do not execute `R_BLE_Close()` while the USB module is in use.

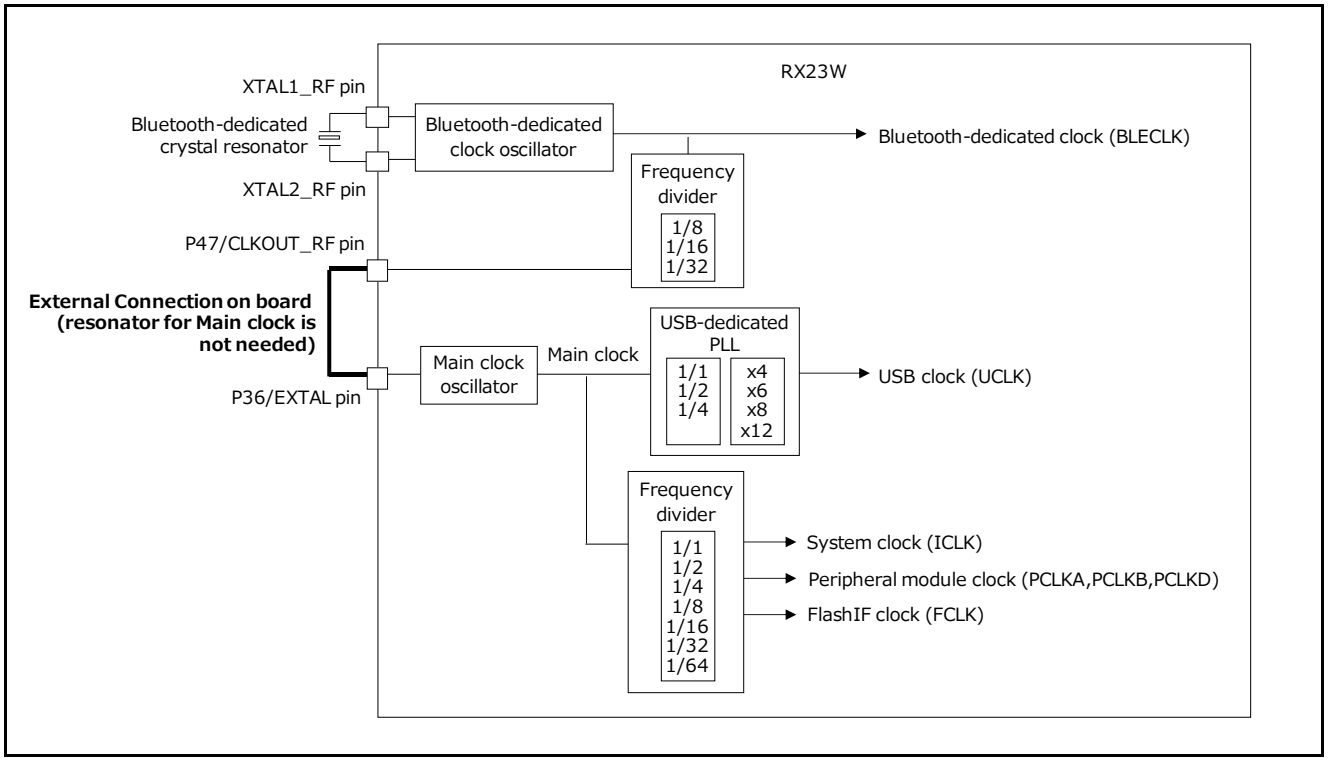

Figure 2-10 Example of using the CLKOUT_RF Pin Output Clock

3. Sample Program

3.1 Overview

This chapter describes an overview of the sample program of this application note. Table 3-1 shows the peripheral modules used by the sample program.

Table 3-1 Peripheral Modules used by the Sample Program

Peripheral Module	Use
Bluetooth Low Energy (BLE)	Bluetooth LE wireless communication with peer evaluation board
USB2.0 Host/Function Module (USB0)	USB communication with USB Host device (PC)
Serial Communication Interface (SCI8)	log output to the console on the PC by UART operation
I/O Ports (P41 to P44)	LED control on the evaluation board

Figure 3-1 shows the demo configuration of the sample program. Two evaluation boards are used in the demonstration. Two or even one PC can be used. PC and evaluation boards are connected by a USB cable for USB communication and a USB cable for UART-USB communication. Connections for USB communication and for UART-USB communication are recognized as different COM ports. In addition, the two evaluation boards establish a Bluetooth LE connection automatically.

A console for USB communication and a console for UART-USB communication are used on each PC. The console for UART-USB displays operation logs indicating the connection, disconnection, data transmission and reception of both the Bluetooth LE communication and USB communication.

When a character string is input to the console for USB communication, the character string data is transmitted from PC to the evaluation board. The evaluation board which received the data transmits the data to peer evaluation board by Bluetooth LE. Also, the evaluation board which received the data by Bluetooth LE transmits the data by USB and the character string is displayed on the console for USB communication.

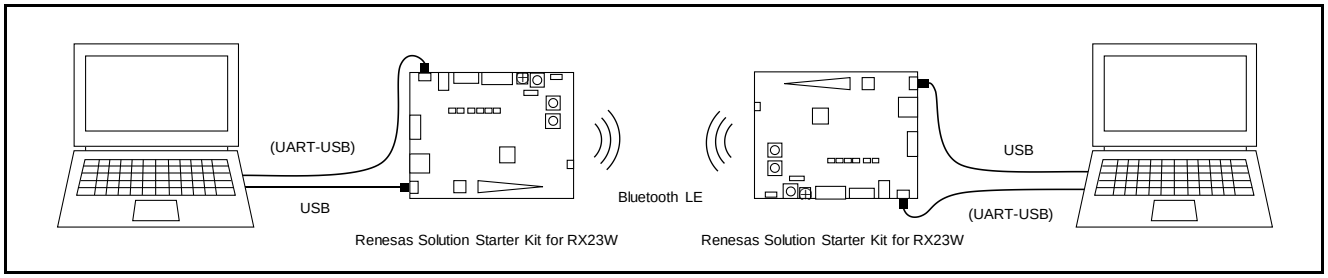

Figure 3-1 Demo Configuration of the Sample Program

3.2 Projects

Table 3-2 shows the project configuration of the sample program. The presence or absence of RTOS and Bluetooth LE operation differs for each project. There is no difference in USB operation for each project.

Table 3-2 Project Configuration of the Sample Program

Demo Project	RTOS	Bluetooth LE Operation	USB Operation
rsskrx23w_osless_gattclient_pcdc	OS-less	GAP Central Data I/O Service Client	USB Peripheral Communication Device Class (CDC)
rsskrx23w_osless_gattserver_pcdc		GAP Peripheral Data I/O Service Server	
rsskrx23w_freertos_gattclient_pcdc	FreeRTOS	GAP Central Data I/O Service Client	
rsskrx23w_freertos_gattserver_pcdc		GAP Peripheral Data I/O Service Server	

Table 3-3 shows the program size of the sample program.

Table 3-3 Program Size of the Sample Program

Demo Project	Section Size	
	ROMDATA+PROGRAM	RAMDATA
rsskrx23w_osless_gattclient_pcdc	212,962byte	44,978byte
rsskrx23w_osless_gattserver_pcdc	212,773byte	45,152byte
rsskrx23w_freertos_gattclient_pcdc	226,796byte	60,761byte
rsskrx23w_freertos_gattserver_pcdc	226,612byte	60,937byte

Table 3-4 shows the FIT Modules used by the sample program and their versions.

Table 3-4 FIT Module Configuration of the Sample Program

FIT Module	Version
RX23W Group BLE FIT Module (r_ble_rx23w)	2.30
RX Family Board Support Package Module (r_bsp)	6.21
RX Family BYTEQ Module (r_byteq)	1.90
RX Family CMT Module (r_cmt_rx)	5.00
RX Family LPC Module (r_lpc_rx)	2.03
RX Family Renesas FreeRTOS (FreeRTOS_Kernel)	1.0.102
RX Family Renesas FreeRTOS (FreeRTOS_Object)	1.0.104
RX Family GPIO Module (r_gpio_rx)	4.20
RX Family IRQ Module (r_irq_rx)	3.90
RX Family SCI Module (r_sci_rx)	4.00
RX Family USB Basic Mini Module (r_usb_basic_mini)	1.20
RX Family USB Peripheral Communication Device Class Driver (r_usb_pcdc_mini)	1.20

Table 3-5, Table 3-6, Table 3-7, and Table 3-8 shows the main settings of the BSP FIT Module, BLE FIT Module, USB Basic Mini FIT Module, and USB PCDC FIT Modules used by the sample program.

Table 3-5 BSP FIT Module Settings for the Sample Program (r_bsp_config.h)

Item	Value	Setting
BSP_CFG_CLOCK_SOURCE	1	System Clock Source: HOCO
BSP_CFG_USB_CLOCK_SOURCE	1	USB Clock Source: USB-dedicated PLL
BSP_CFG_MAIN_CLOCK_SOURCE	1	Main Clock Oscillator: External oscillator input
BSP_CFG_XTAL_HZ	4000000	Clock Frequency of External oscillator input: 4MHz
BSP_CFG_UPLL_DIV BSP_CFG_UPLL_MUL	1 12	Frequency Multiplication Factor: x12
BSP_CFG_HOCO_FREQUENCY	0	Frequency of HOCO: 32MHz
BSP_CFG_CLKOUT_RF_MAIN	1	Clock input source for EXTAL pin: CLKOUT_RF pin
BSP_CFG_MCU_VCC_MV	3300	VCC voltage: 3.3V

Table 3-6 BLE FIT Module Settings for the Sample Program (r_ble_rx23w_config.h)

Item	Value	Setting
BLE_CFG_LIB_TYPE	1	Type of the BLE Protocol Stack: Balance
BLE_CFG_RF_CONN_MAX	1	Maximum number of simultaneous connections: 1
BLE_CFG_RF_CONN_DATA_MAX	251	Maximum packet data length: 251byte
BLE_CFG_RF_ADV_DATA_MAX	31	Maximum Advertising Data length: 31byte
BLE_CFG_RF_DDC_EN	0	RF transceiver power-supply: linear regulator (DC-to-DC converter is disabled)
BLE_CFG_RF_EXT32K_EN	0	Bluetooth-dedicated low speed clock: Bluetooth-dedicated low speed on-chip oscillator
BLE_CFG_RF_SCA	250	Sleep Clock Accuracy(SCA) : 250ppm
BLE_CFG_RF_MAX_TX_POW	1	Maximum transmission power: +4dBm
BLE_CFG_RF_DEF_TX_POW	0	Default transmission power: +4dBm
BLE_CFG_RF_CLKOUT_EN	5	Clock output to CLKOUT_RF pin: 4MHz
BLE_CFG_RF_DEEP_SLEEP_EN	1	Transition to RF Sleep Mode: Enabled
BLE_CFG_GATT_MTU_SIZE	247	ATT_MTU size notified during Exchange MTU: 247byte
BLE_CFG_CMD_LINE_EN	1	Command Line function: Enabled
BLE_CFG_CMD_LINE_CH	8	SCI channel used for the Command Line function: 8
BLE_CFG_BOARD_LED_SW_EN	1	Board LED & Switch Control function: Enabled
BLE_CFG_BOARD_TYPE	2	Evaluation Board Type: RSSK for RX23W
BLE_CFG_ABS_API_EN	1	Abstraction API: Enabled
BLE_CFG_SOFT_TIMER_EN	1	Software Timer: Enabled
BLE_CFG_MCU_LPC_EN	1	MCU Low Power Consumption Control function: Enabled

Table 3-7 USB Basic Mini FIT Module Settings for the Sample Program (r_usb_basic_config.h)

Item	Value	Setting
USB_CFG_MODE	USB_CFG_PERI	USB Operating Mode: USB Peripheral Mode
USB_CFG_DEVICE_CLASS	USB_CFG_PCDC_USE	Device Class: Peripheral Communication Device Class
USB_CFG_DTC	USB_CFG_DISABLE	DTC: Not used
USB_CFG_DMA	USB_CFG_DISABLE	DMA: Not used
USB_CFG_BC	USB_CFG_DISABLE	Battery Charging function: Not used
USB_CFG_REGULATOR	USB_CFG_DISABLE	USB regulator: Not used

Table 3-8 USB PCDC FIT Module Settings for the Sample Program (r_usb_pcdc_mini_config.h)

Item	Value	Setting
USB_CFG_PCDC_BULK_IN	USB_PIPE1	1st VCOM CDC Data class Bulk In Pipe: PIPE1
USB_CFG_PCDC_BULK_OUT	USB_PIPE2	1st VCOM CDC Data class Bulk Out Pipe: PIPE2
USB_CFG_PCDC_INT_IN	USB_PIPE6	1st VCOM CDC Data class Interrupt In Pipe: PIPE6

Figure 3-2 shows the clock configuration in the Smart Configurator. The sample program uses 4MHz clock output to CLKOUT_RF pin as a Main clock and uses 48MHz clock as an USB clock by multiplying the Main clock by 12 with the USB-dedicated PLL.

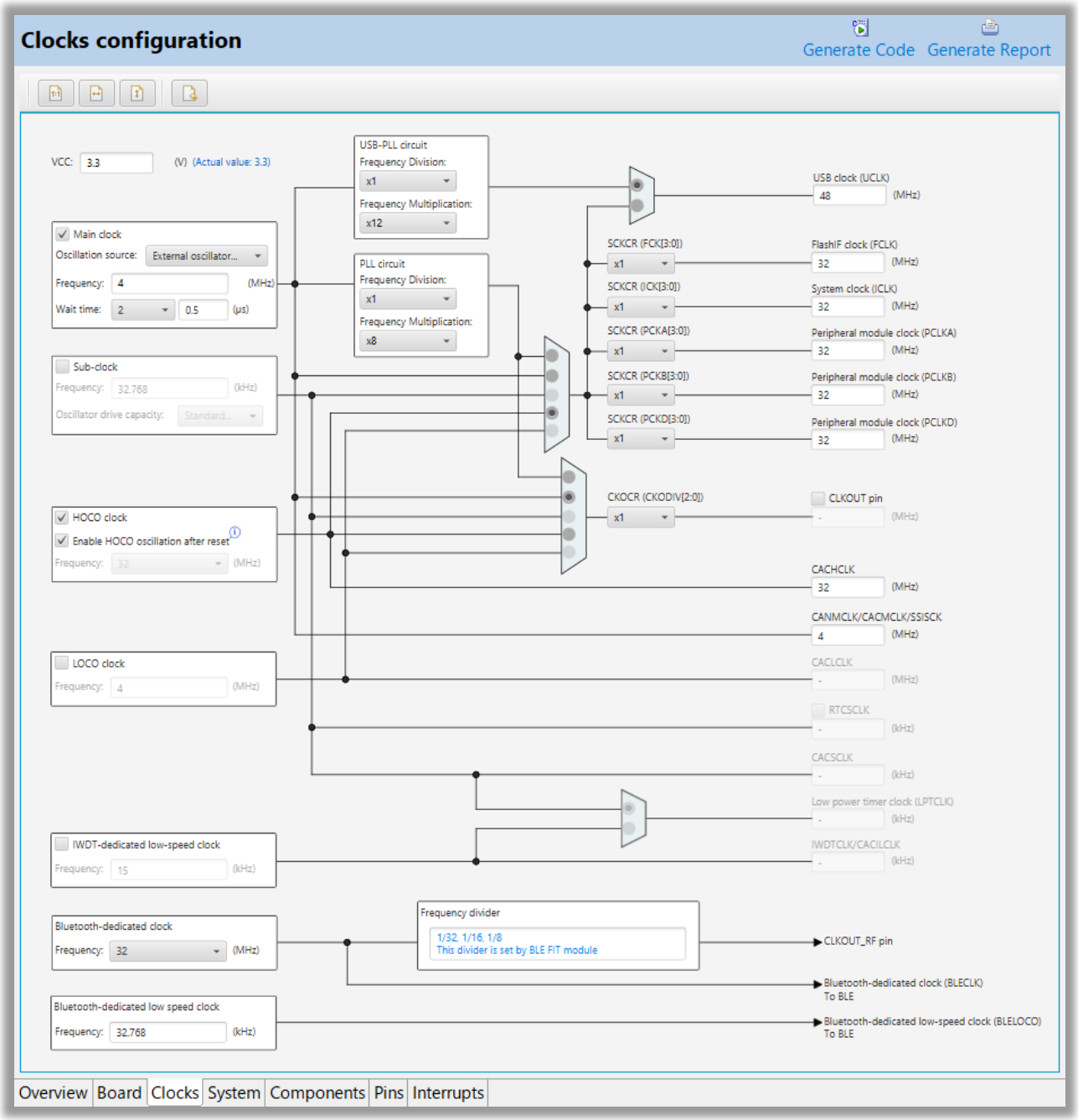

Figure 3-2 Clock Settings of the Sample Program

3.3 Operation Check Conditions

The sample program has been checked according to the operation check conditions shown in Table 3-9.

Table 3-9 Operation Check Conditions

Item	Condition
MCU	RX23W (F523W8ADBL) ROM size: 512Kbytes, RAM size: 64Kbytes, Number of Pins: 85
Operating Frequency	HOCO clock: Enable HOCO oscillation after reset Main clock: Use 4MHz clock output to CLKOUT_RF pin and input from EXTAL pin System clock (ICLK): 32MHz Peripheral module clock (PCLKA、PCLKB、PCLKD): 32MHz USB clock (UCLK): 48MHz FlashIF clock (FCLK): 32MHz Bluetooth-dedicated clock (BLECLK): 32MHz Bluetooth-dedicated low-speed clock (BLELOCO): 32.768kHz
Evaluation Board	Renesas Solution Starter Kit for RX23W (RTK5523W8AC00001BJ) Jumper Setting: Power Supply → USBCN0(FT234_5V) J3：1-2 Short J4：1-2 Short J5：1-2 Short J10：Short J13：Short Jumper Setting: P26 → USB0-VBUSEN J7：1-2 Short
Emulator	E2 Emulator Lite (RTE0T0002LKCE00000R)
IDE	e^2 studio 64-bit 2021-10
Compiler	C/C++ Compiler for RX Family (CC-RX) V2.08.01
Flash Programming Tool	Renesas Flash Programmer V3.09.00
Development Assistance Tool for Bluetooth Low Energy NOTE1	QE for BLE[RA,RE,RX] V1.4.0 (1.4.0.v20220120-803) QE for BLE[RA,RE,RX] Utility (1.4.0.v20220118-1104)
Endian	Little Endian
FreeRTOS	Renesas FreeRTOS (kernel only) 10.4.3-rx-1.0.1
Terminal Emulator	Tera Term Version 4.106
Serial Port Setting	Baud rate: 115,200 bps Data: 8 bits Parity: none Stop: 1 bit Flow Control: none

NOTE1:

For how to install the Development Assistance Tool for Bluetooth Low Energy QE for BLE[RA,RE,RX], refer to the following website.

QE: Tools for Particular Applications Information for Users
https://www.renesas.com/software-tool/qe-support

For how to update the QE for BLE[RA,RE,RX] Utility which is the plugin for QE for BLE[RA,RE,RX], refer to the following website.

QE for BLE: Development Assistance Tool for Bluetooth® Low Energy Information for Users
https://www.renesas.com/software-tool/qe-ble-development-assistance-tool-bluetooth-low-energy-information-users

3.4 Configuration and Operation

Figure 3-3 shows the software configuration of the sample program. The sample program includes BLE control application, USB control application, and demo application in addition to the FIT Modules and QE for BLE generation codes required to control the BLE modules and USB module of RX23W.

The BLE control application provides the R_BLECTRL interface for higher-layer applications to execute Bluetooth LE communication easily. The BLE control application uses the profile framework generated by QE for BLE. For details on the R_BLECTRL interface, refer to Section 3.6.

The USB control application provides the R_USBCTRL interface for higher-layer applications to execute USB communication easily. For details on the R_USBCTRL interface, refer to Section 3.7.

The demo application uses the R_BLECTRL interface and the R_USBCTRL interface to control Bluetooth LE communication and USB communication as well as low power consumption mode of MCU.

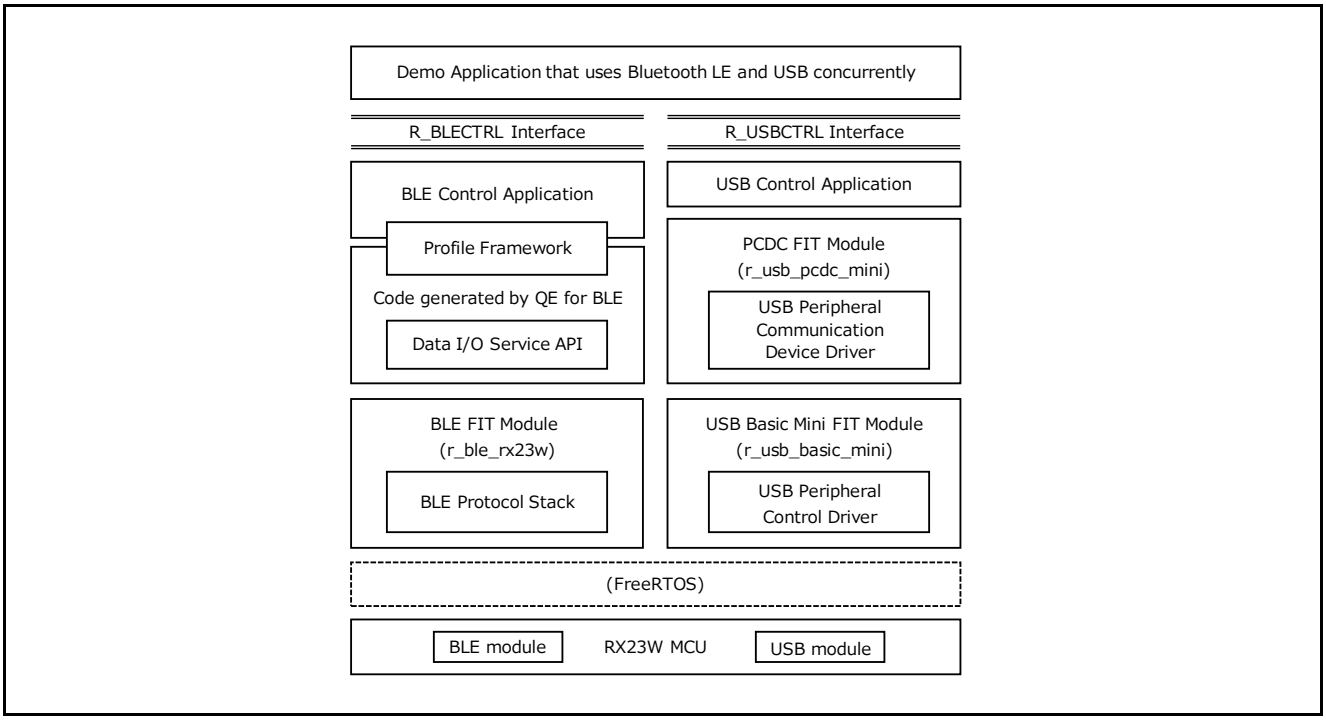

Figure 3-3 Software Configuration of the Sample Program

The sample program defines the following proprietary service. Two evaluation boards transmit and receive any data by using this service in Bluetooth LE communication. When working as a Central device, Data is transmitted to the Data In characteristic with the Write Without Response. When working as a Peripheral device, Data is transmitted from the Data Out characteristic with the Notify.

Table 3-10 Data I/O Service

Item	UUID	Properties Attribute Permission
Data I/O Service	7dbe3201-f5ab-498a-a012-676fef22f735	-
Data In Characteristic	7dbe3202-f5ab-498a-a012-676fef22f735	Write Without Response (max.244byte) No Authentication, No Authorization
Data Out Characteristic	7dbe3203-f5ab-498a-a012-676fef22f735	Notify (max.244byte)
Client Characteristic Configuration Descriptor	0x2902	Read, Write No Authentication, No Authorization

Figure 3-4 shows the LEDs mounted on the RSSK for RX23W, and Table 3-11 shows the LED Control of the sample program. When the power is supplied to the evaluation board on which the sample program is written, the BLE control application starts Bluetooth LE connection operation automatically. The LED0 is turned on when Bluetooth LE connection with peer evaluation board is established and data communication by Data I/O Service is enabled. The LED0 is turned off when the Bluetooth LE connection is terminated.

When the evaluation board is connected to PC with USB cable, the USB control application performs the processing to transition to the Configured state automatically. Then, when a terminal emulator on the PC opens COM port and CDC data communication is enabled, the LED1 is turned on. The LED1 is turned off when the USB connection to the PC is terminated.

The order of Bluetooth LE and USB connection is not restricted. You can either connect a USB cable after establishing a Bluetooth LE connection or establish a Bluetooth LE connection after connecting a USB cable.

Table 3-12 shows the control of the MCU low power consumption modes of the Sample program. LED2 is turned on when the RX23W MCU transitions to any one of the MCU low power consumption modes.

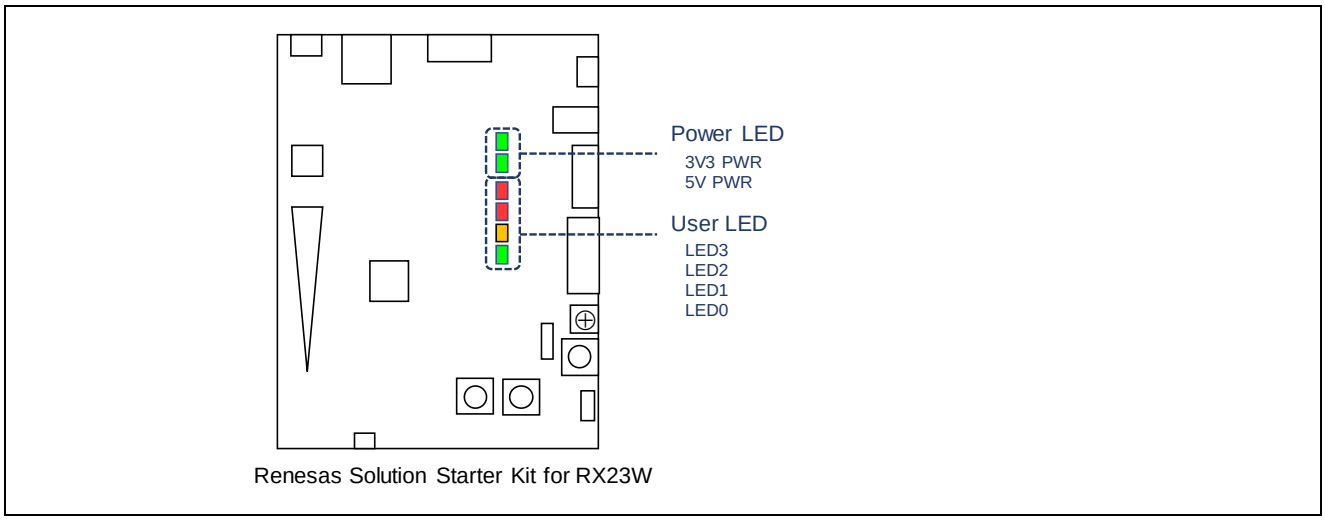

Figure 3-4 LEDs mounted on RSSK for RX23W

Table 3-11 LED Control of the Sample Program

User LED	Control Timing	Description
LED0 (green)	ON: `BLECTRL_EVT_SERVICE_ENABLED` OFF: `BLECTRL_EVT_DISCONNECTED`	It Indicates data communication using Data I/O Service of Bluetooth LE is possible.
LED1 (orange)	ON: `USBCTRL_EVT_CDC_ENABLED` OFF: `USBCTRL_EVT_DETACH`	It indicates data communication using CDC of USB is possible
LED2 (red)	ON: Just before `R_BLE_LPC_EnterLowPowerMode()` OFF: Just after `R_BLE_LPC_EnterLowPowerMode()`	It indicates RX23W MCU is in any one of low power consumption mode.
LED3 (red)	-	It is not used.

Table 3-12 Control of the MCU Low Power Consumption Mode of the Sample Program

BLE state	USB state	Control of the MCU Low Power Consumption Mode
Disconnected / Connected	Detach	OS-less: Transition to Software standby mode FreeRTOS: Transition to Deep sleep mode
	Suspend	Transition to Sleep mode
	other than the above	Normal Operation (Not transition to MCU low power consumption mode)

Table 3-13 shows the processing of data transmission and reception of the demo application, and Figure 3-5 shows the operation of data transmission and reception of the demo application. When both the BLE state and the USB state are disconnected, the demo application does not perform data transmission and reception. When the BLE or/and USB state is connected, the log is output to the console when the data is transmitted and received. If both the BLE and USB states are connected, the data received by USB is transmitted by the Bluetooth LE and the data received by Bluetooth LE is transmitted by the USB.

Table 3-13 Data Transmission and Reception Processing of the Demo Application

BLE state	USB state	Demo operation
Disconnected	Disconnected	Do nothing
Connected		Output log to console when data is received by Bluetooth LE
Disconnected	Connected	Output log to console when data is received by USB
Connected		Output log when data is transmitted and received by USB Output log when data is transmitted and received by Bluetooth LE Transmit the USB received data by Bluetooth LE Transmit the Bluetooth LE received data by USB

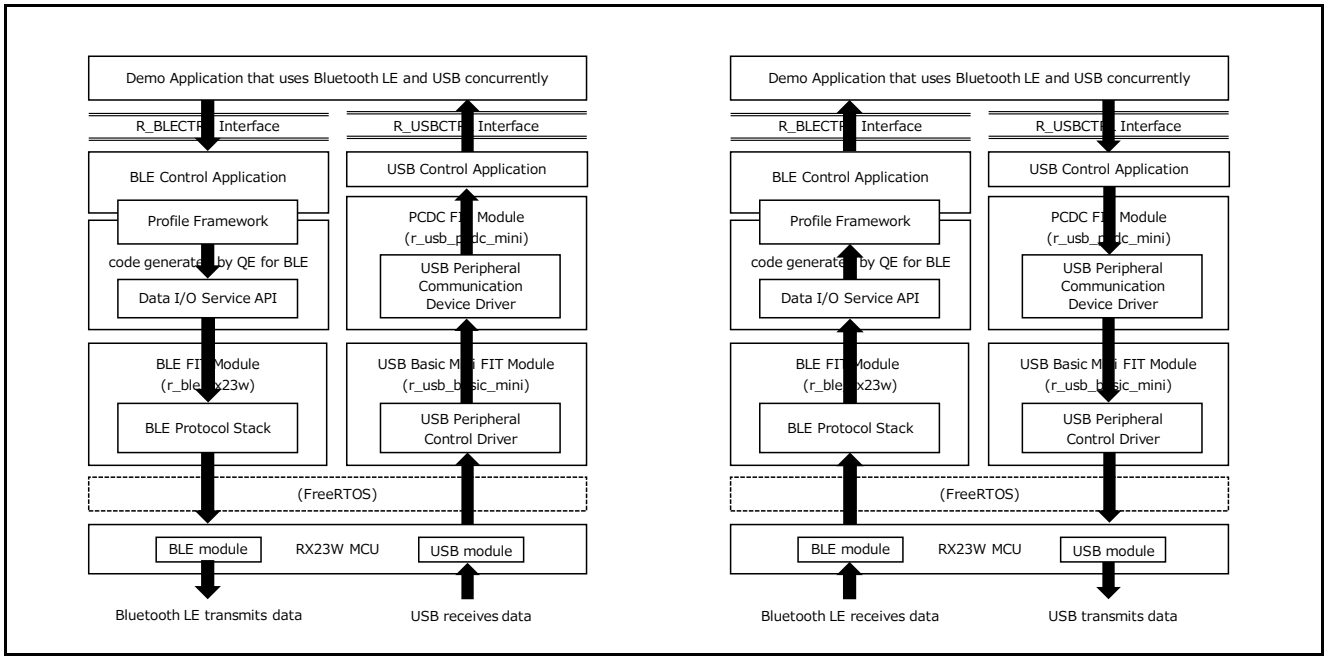

Figure 3-5 Data Reception and Transmission of the Sample Program (Left: USB→Bluetooth LE, Right: Bluetooth LE→USB)

3.5 Usage

3.5.1 Importing Projects to e² studio and Building

1. Launch the e² studio and select any folder as a workspace.

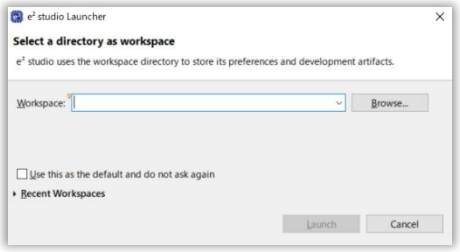

Figure 3-6 Selecting a Workspace

2. Select [File]→[Import…] in menu to display Import Dialog.

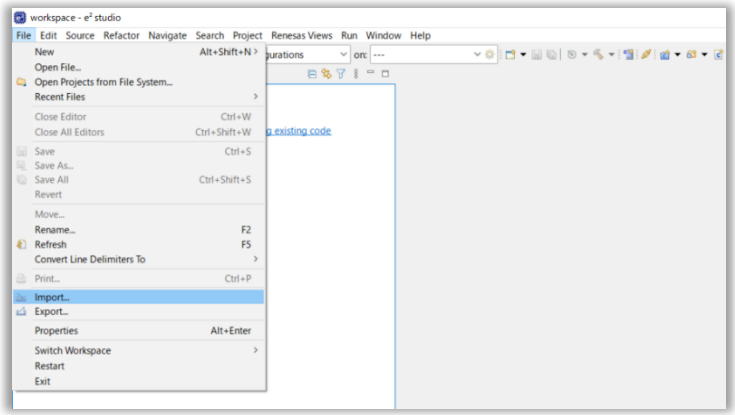

Figure 3-7 [File]→[Import…]

3. Select [General]→[Existing Projects into Workspace] and click [Next] button.

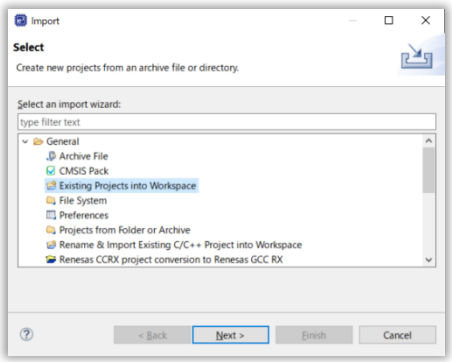

Figure 3-8 [General]→[Existing Projects into Workspace]

4. Select any project file (.zip) of the sample program in [Select archive file] and click [Finish] button.

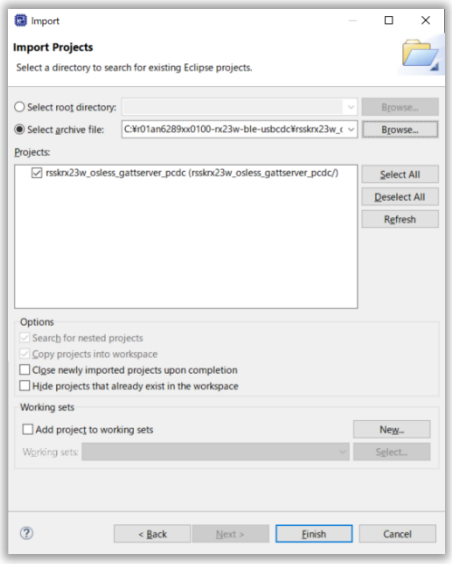

Figure 3-9 Selecting archive file

5. Repeat the steps 1 to 4 to import each project of the sample program.

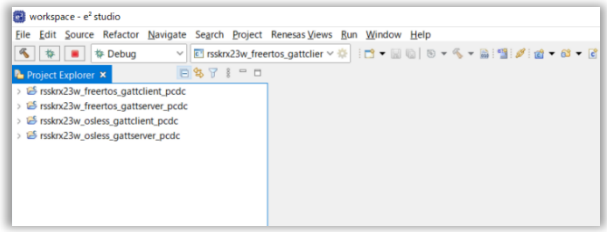

Figure 3-10 Completion of importing the projects of the sample program

6. After completion of the step 5, select [Project]→[Build All] in menu. "Build complete" is displayed after building the projects is completed. The program files (.mot) that was built are stored in Hardware Debug folder in each project folder.

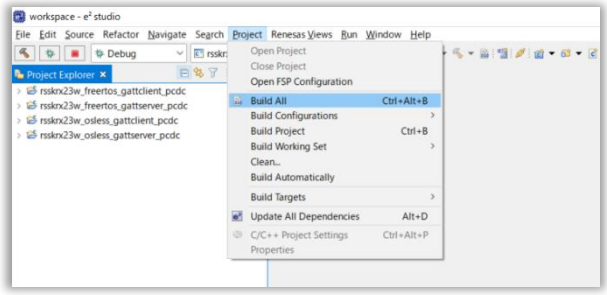

Figure 3-11 [Project]→[Build All]

NOTE: When the error indicating "No toolchain set or toolchain not integrated." occurs and building fails, open the project properties dialog and move to [C/C++ Build]→[Settings]→[Toolchain] tag, then set the toolchain.

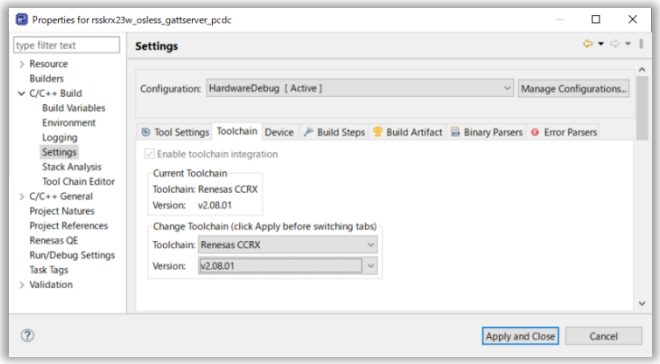

Figure 3-12 Selecting a Toolchain

3.5.2 Writing Program Files to the Evaluation Boards

1. Connect an emulator to the evaluation board and connect the emulator to PC. Then, connect USBCN0 on the evaluation board to PC with a cable. Power to the evaluation board is supplied from the USBCN0.

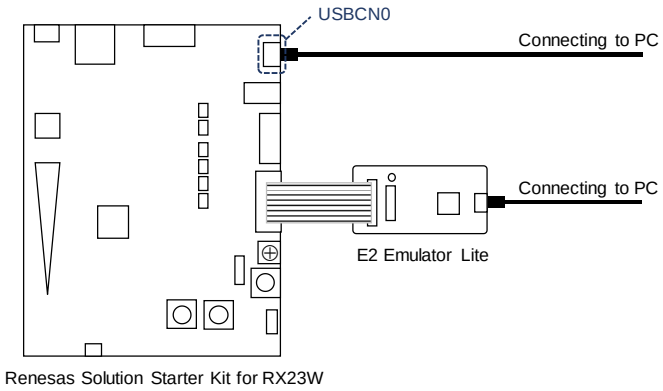

Figure 3-13 Connections of the Evaluation Board for Writing a Program File

2. Launch the Renesas Flash Programmer and select [File]→[New Project…] in menu. In the Create New Project dialog, enter any project name, select the Tool, set the Interface to FINE, and click [Connect]. Confirm that "Operation completed" is displayed.

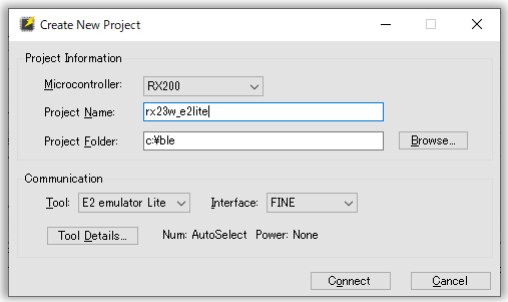

Figure 3-14 Creating New Project

3. Specify the program file (.mot) that was built by e^2 studio and click [Start]. If "Operation completed" is displayed, writing the program file is completed.

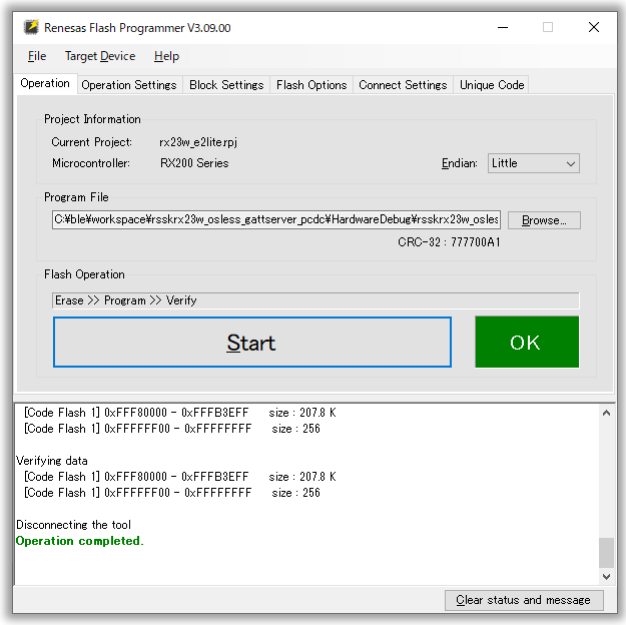

Figure 3-15 Writing a Program File

4. Repeat the steps 1 and 3 to write any of the program file combination shown in the Table 3-14 to the two evaluation boards.

Table 3-14 Combinations of the Program Files to be Written

Combination	Evaluation Board for Central	Evaluation Board for Peripheral
(1)	rsskrx23w_osless_gattclient_pcdc.mot	rsskrx23w_osless_gattserver_pcdc.mot
(2)	rsskrx23w_freertos_gattclient_pcdc.mot	rsskrx23w_freertos_gattserver_pcdc.mot
(3)	rsskrx23w_osless_gattclient_pcdc.mot	rsskrx23w_freertos_gattserver_pcdc.mot
(4)	rsskrx23w_freertos_gattclient_pcdc.mot	rsskrx23w_osless_gattserver_pcdc.mot

3.5.3 Evaluating Bluetooth LE and USB communications

1. Connect USBCN0 on the evaluation board to PC with a cable and connect USB0_2 to the PC. Power to the evaluation board is supplied from the USBCN0.

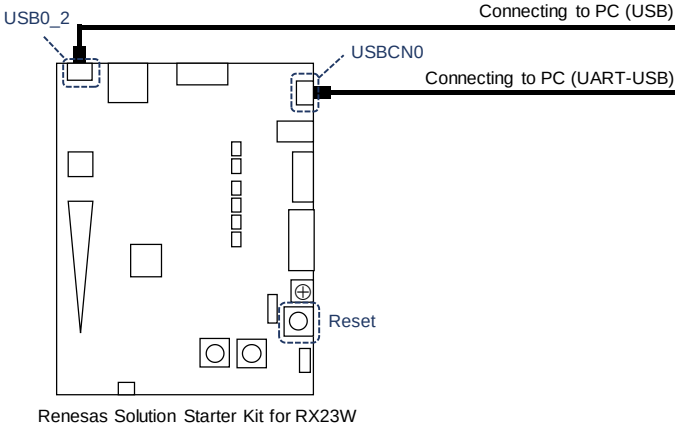

Figure 3-16 Connections of the Evaluation Board for Evaluating the Communications

2. When the two evaluation boards are connected to PC with a USB cable, each USB connection is recognized as a COM port. Launch the terminal emulators and open the COM ports. For serial port settings of the terminal emulator, refer to Table 3-9.

   Figure 3-17 is an example of the display of terminal emulators that open each COM port after connecting the two evaluation boards to PC.

   Upper left: rsskrx23w_osless_gattserver_pcdc, Operation log over UART-USB communication

   Lower left: rsskrx23w_osless_gattserver_pcdc, Received data over USB communication

   Upper right: rsskrx23w_osless_gattclient_pcdc, Operation log over UART-USB communication

   Lower right: rsskrx23w_osless_gattclient_pcdc, Received data over USB communication

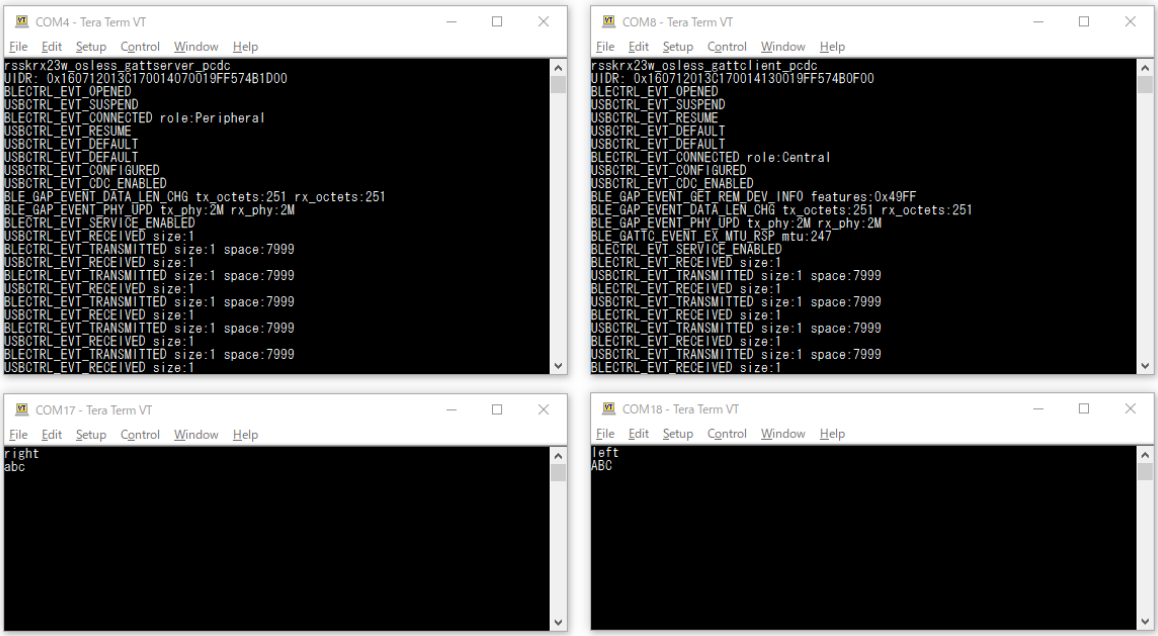

Figure 3-17 Example of the Display of Terminal Emulators

3. Pressing the reset button on the evaluation board restarts the program operation and then operation log is output to the console for UART-USB communication. For an example of the operation log, refer to Figure 3-18 that will be shown later.

4. Check that `BLECTRL_EVT_SERVICE_ENABLED` and `USBCTRL_EVT_CDC_ENABLED` are displayed in the operation logs. When a character string is entered in the terminal emulator for the USB on the evaluation board, the character string is displayed in the terminal emulator for the USB on the other evaluation board.

   In the example of the display of the terminal emulators shown in Figure 3-17, "left" is displayed in the terminal of the lower right by entering the "left" to the terminal of the lower left. "right" is displayed in the terminal of the lower left by entering the "right" to the terminal of the lower right.

Figure 3-18 is an example of the operation log from the evaluation board on which rsskrx23w_osless_gattserver_pcdc is written.

```
 1  rsskrx23w_osless_gattserver_pcdc
 2  UIDR: 0x160712013C170014070019FF574B1D00
 3  BLECTRL_EVT_OPENED
 4  USBCTRL_EVT_SUSPEND
 5  USBCTRL_EVT_RESUME
 6  USBCTRL_EVT_DEFAULT
 7  USBCTRL_EVT_DEFAULT
 8  USBCTRL_EVT_CONFIGURED
 9  USBCTRL_EVT_CDC_ENABLED
10  BLECTRL_EVT_CONNECTED role:Peripheral
11  BLE_GAP_EVENT_DATA_LEN_CHG tx_octets:251 rx_octets:251
12  BLE_GAP_EVENT_PHY_UPD tx_phy:2M rx_phy:2M
13  BLECTRL_EVT_SERVICE_ENABLED
14  USBCTRL_EVT_RECEIVED size:1
15  BLECTRL_EVT_TRANSMITTED size:1 space:7999
16  USBCTRL_EVT_RECEIVED size:1
17  BLECTRL_EVT_TRANSMITTED size:1 space:7999
18  BLECTRL_EVT_RECEIVED size:1
19  USBCTRL_EVT_TRANSMITTED size:1 space:7999
20  BLECTRL_EVT_RECEIVED size:1
21  USBCTRL_EVT_TRANSMITTED size:1 space:7999
```

Figure 3-18 Example of the Operation Log of the Sample Program

L1. Program File Name

L2. UIDR: 16-byte Unique ID to identify RX23W, used as iSerialNumber of USB

Device instance path, which includes Device ID, Product ID, and iSerialNumber, can be found in [Ports (COM & LPT)]→[USB Serial Bus (COM **)] in the Device Manager of Windows OS

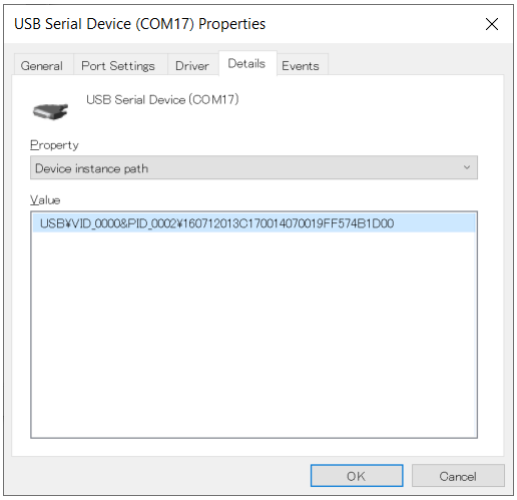

Figure 3-19 Display Example of the Device Instance Path

L8. `USBCTRL_EVT_CONFIGURED`: Completion of transition to Configured state of USB

L9. `USBCTRL_EVT_CDC_ENABLED`: Completion of opening COM port, data communication enabled

L10. `BLECTRL_EVT_CONNECTED`: Completion of Bluetooth LE connection establishment

L13. `BLECTRL_EVT_SERVICE_ENABLED`: Completion of enabling Data I/O Service, data communication enabled

L14. `USBCTRL_EVT_RECEIVED`: Completion of data reception by USB

L15. `BLECTRL_EVT_TRANSMITTED`: Completion of data transmission by Bluetooth LE

L18. `BLECTRL_EVT_RECEIVED`: Completion of data transmission by Bluetooth LE

L19. `USBCTRL_EVT_TRANSMITTED`: Completion of data reception by USB

3.6 R_BLECTRL Interface

This section describes the specification of R_BLECTRL Interface provided by the BLE Control Application.

3.6.1 R_BLECTRL_Open

R_BLECTRL_Open		
Initialize BLE Control processing		
Header	r_blectrl_if.h	
Definition	`e_blectrl_status_t R_BLECTRL_Open(void);`	
Parameter	None	
Return Value	`BLECTRL_SUCCESS`	Success
	`BLECTRL_ERR_STATE`	Failure: Initialized State
	`BLECTRL_ERR_API`	Failure: BLE Protocol Stack Initialization Failed
Supplement	The completion of initialization of the BLE Control processing is notified by the `BLECTRL_EVT_OPENED` event. When the CLKOUT_RF pin clock output is enabled, the clock output starts when this function is executed. When the CLKOUT_RF pin output clock is used as a reference clock of the USB clock (UCLK), this function shall be executed before the start of operation of the USB Module.	

3.6.2 R_BLECTRL_Close

R_BLECTRL_Close		
Finalize BLE Control projeccing		
Header	r_blectrl_if.h	
Definition	`e_blectrl_status_t R_BLECTRL_Close(void);`	
Parameter	None	
Return Value	`BLECTRL_SUCCESS`	Success
	`BLECTRL_ERR_STATE`	Failure: Uninitialized State or Closed State
	`BLECTRL_ERR_API`	Failure: BLE Protocol Stack Finalization Failed
Supplement	When this function is executed, the CLKOUT_RF pin clock output stops. When the clock output from the CLKOUT_RF pin is used as a Main clock, this function must not be executed because the operation of the USB module is stopped.	

3.6.3 R_BLECTRL_RegisterCb

R_BLECTRL_RegisterCb		
Register BLE Control Event callback function		
Header	r_blectrl_if.h	
Definition	`e_blectrl_status_t R_BLECTRL_RegisterCb(blectrl_evt_cb_t cb);`	
Parameter	(IN) `cb`	BLE Control Event callback function
Return Value	`BLECTRL_SUCCESS`	Success
	`BLECTRL_ERR_ARG`	Failure: argument `cb` is NULL
	`BLECTRL_ERR_STATE`	Failure: Uninitialized state or Communication operation state
Supplement	This function must be executed after the execution of `R_BLECTRL_Open()` and before the execution of `R_BLECTRL_StartPeripheral()` and `R_BLECTRL_StartCentral()`	

3.6.4 R_BLECTRL_RegisterTxBuffer

R_BLECTRL_RegisterTxBuffer			
Register BLE Transmission Buffer			
Header	r_blectrl_if.h		
Definition	`e_blectrl_status_t R_BLECTRL_RegisterTxBuffer(uint8_t *buffer, uint16_t size);`		
Parameter	(IN)	`buffer`	Transmission Buffer
	(IN)	`size`	Transmission Buffer Size
Return Value	`BLECTRL_SUCCESS`		Success
	`BLECTRL_ERR_ARG`		Failure: argument `buffer` is NULL or `buffer` is 0
	`BLECTRL_ERR_STATE`		Failure: Uninitialized state or Communication operation state
Supplement	This function must be executed after the execution of `R_BLECTRL_Open()` and before the execution of `R_BLECTRL_StartPeripheral()` and `R_BLECTRL_StartCentral()`		

3.6.5 R_BLECTRL_StartPeripheral

R_BLECTRL_StartPeripheral		
Start a Connection as a Peripheral Device		
Header	r_blectrl_if.h	
Definition	`e_blectrl_status_t R_BLECTRL_StartPeripheral(void);`	
Parameter	None	
Return Value	`BLECTRL_SUCCESS`	Success
	`BLECTRL_ERR_UNSUPPORT`	Failure: Peripheral operation is not supported
	`BLECTRL_ERR_STATE`	Failure: Uninitialized state
	`BLECTRL_ERR_API`	Failure: Advertising start failed
Return Value	The completion of connection to a Central device is notified by the `BLECTRL_EVT_CONNECTED` event.	
	After connecting, the completion of enabling GATT Service is notified by the `BLECTRL_EVT_SERVICE_ENABLED` event	
	After the completion of enabling GATT Service is enabled, data reception is notified by the `BLECTRL_EVT_RECEIVED` event.	
	The termination of the connection with the central device is notified by the `BLECTRL_EVT_DISCONNECTED` event	
Limitation	1. Only one Central device can be connected to simultaneously.	

3.6.6 R_BLECTRL_StartCentral

R_BLECTRL_StartCentral		
Start a Connection as a Central Device		
Header	r_blectrl_if.h	
Definition	`e_blectrl_status_t R_BLECTRL_StartCentral(void);`	
Parameter	None	
Return Value	`BLECTRL_SUCCESS`	Success
	`BLECTRL_ERR_UNSUPPORT`	Failure: Central operation is not supported
	`BLECTRL_ERR_STATE`	Failure: Uninitialized state
	`BLECTRL_ERR_API`	Failure: Scan start failed
Supplement	The completion of connection to a Peripheral device is notified by the `BLECTRL_EVT_CONNECTED` event. After connecting, the completion of enabling GATT Service is notified by the `BLECTRL_EVT_SERVICE_ENABLED` event. After the completion of enabling GATT Service is enabled, data reception is notified by the `BLECTRL_EVT_RECEIVED` event. The termination of the connection with the central device is notified by the `BLECTRL_EVT_DISCONNECTED` event.	
Limitation	1. Only one Peripheral device can be connected simultaneously.	

3.6.7 R_BLECTRL_Disconnect

R_BLECTRL_Disconnect			
Disconnect from the peer device			
Header	r_blectrl_if.h		
Definition	`e_blectrl_status_t R_BLECTRL_Disconnect(uint16_t conn_hdl);`		
Parameter	(IN)	`conn_hdl`	Connection Handle
Return Value	`BLECTRL_SUCCESS`		Success
	`BLECTRL_ERR_STATE`		Failure: Unconnected to peer device
	`BLECTRL_ERR_API`		Failure: Disconnect request failed
Supplement	The completion of disconnection is notified by the `BLECTRL_EVT_DISCONNECTED` event.		

3.6.8 R_BLECTRL_GetTxBufferSpace

R_BLECTRL_GetTxBufferSpace			
Get the empty size of BLE Transmission Buffer			
Header	r_blectrl_if.h		
Definition	`uint16_t R_BLECTRL_GetTxBufferSpace(uint16_t conn_hdl);`		
Parameter	(IN)	`conn_hdl`	Connection Handle
Return Value	Empty Size of Transmission Buffer [byte]		
Supplement	None		

3.6.9 R_BLECTRL_TransmitData

R_BLECTRL_TransmitData			
Transmit data to the connected peer device			
Header	r_blectrl_if.h		
Definition	`uint16_t R_BLECTRL_TransmitData(uint16_t conn_hdl, const uint8_t *data, uint16_t size);`		
Parameter	(IN)	`conn_hdl`	Connection Handle
	(IN)	`data`	Data to be Transmitted
	(IN)	`size`	Size of Data to be Transmitted [byte]
Return Value	Size of Data stored in Transmission Buffer [byte]		
Supplement	This function must be executed after the notification of the `BLECTRL_EVT_SERVICE_ENABLED` event. When working as a Central device, Data is transmitted to the Data In characteristic of the Data I/O service with the Write Without Response. When working as a Peripheral device, Data is transmitted from the Data Out characteristic of the Data I/O service with the Notify. Data Transmission is notified by the `BLECTRL_EVT_TRANSMITTED` event.		
Limitation	When using FreeRTOS 1. Do not call this function from an interrupt handler because this function uses `xTaskNotifyGive()`. 2. Do no call this function from multiple tasks because the transmission buffer operation in this function is not guarded as a critical section.		

3.6.10 blectrl_main

blectrl_main	
Main Context of the BLE Control Application	
Header	r_blectrl_if.h
Definition	`void blectrl_main(void);`
Parameter	None
Return Value	None
Supplement	When OS is not used, call this function in the main loop. When FreeRTOS is used, create a task for executing this function and call this function in a loop of the task.

3.6.1 blectrl_evt_cb_t

blectrl_evt_cb_t			
BLE Control Event Callback Function			
Header	r_blectrl_if.h		
Definition	`typedef void (*blectrl_evt_cb_t)(e_blectrl_evt_t type, st_blectrl_evt_param_t *param);`		
Parameter	(OUT)	`type`	BLE Control Event
	(OUT)	`param`	BLE Control Event Parameter
Return Value	None		
Supplement	Implement a BLE Control Event Callback Function in a upper application and register with `R_BLECTRL_RegisterCb()`.		

3.6.2 e_blectrl_status_t

e_blectrl_status_t

Execution Result of R_BLECTRL Interface Function

Header r_blectrl_if.h

Definition

```
typedef enum e_blectrl_status
{
    BLECTRL_SUCCESS = 0,
    BLECTRL_ERR_UNSUPPORT,      /* Unsupported API */
    BLECTRL_ERR_ARG,           /* Invalid Argument */
    BLECTRL_ERR_STATE,         /* Invalid State */
    BLECTRL_ERR_API,           /* RBLE API Error */
} e_blectrl_status_t;
```

3.6.3 e_blectrl_evt_t

e_blectrl_evt_t

BLE Control Events

Header r_blectrl_if.h

Definition

```
typedef enum e_blectrl_evt
{
    BLECTRL_EVT_NONE = 0,
    BLECTRL_EVT_OPENED,
    BLECTRL_EVT_CONNECTED,
    BLECTRL_EVT_DISCONNECTED,
    BLECTRL_EVT_SERVICE_ENABLED,
    BLECTRL_EVT_TRANSMITTED,
    BLECTRL_EVT_RECEIVED
} e_blectrl_evt_t;
```

Supplement BLE Control Events are notified by the Callback Function registered with `R_BLECTRL_RegisterCb()`.

For the notification timing of BLE Control Events, refer to Subsection 0.

3.6.4 st_blectrl_evt_param_t

st_blectrl_evt_param_t

BLE Control Event Parameters

Header r_blectrl_if.h

Definition

```
typedef struct st_blectrl_evt_conn
{
    /* role - 0x00: Central, 0x01: Peripheral */
    uint8_t role;
    /* device address type - 0x00: Public, 0x01: Random */
    uint8_t remote_addr_type;
    /* device address */
    uint8_t remote_addr[6];
} st_blectrl_evt_conn_t;

typedef struct st_blectrl_evt_disconn
{
    /* reason for disconnection */
    uint8_t reason;
} st_blectrl_evt_disconn_t;

typedef struct st_blectrl_evt_tx
{
    /* bytes of the data size transmitted */
    uint16_t size;
} st_blectrl_evt_tx_t;

typedef struct st_blectrl_evt_rx
{
    /* pointer to the data received */
    uint8_t *data;
    /* bytes of the data size received */
    uint16_t size;
} st_blectrl_evt_rx_t;

/* Event Parameter */
typedef struct st_blectrl_evt_param
{
    uint16_t conn_hdl;          /* Connection Handle */
    union
    {
        /* BLECTRL_EVT_CONNECTED event parameter */
        st_blectrl_evt_conn_t    conn;
        /* BLECTRL_EVT_DISCONNECTED event parameter */
        st_blectrl_evt_disconn_t disconn;
        /* BLECTRL_EVT_TRANSMITTED event parameter */
        st_blectrl_evt_tx_t      tx;
        /* BLECTRL_EVT_RECEIVED event parameter */
        st_blectrl_evt_rx_t      rx;
    } evt;
} st_blectrl_evt_param_t;
```

Supplement

BLE Control Event Parameters are notified by the Callback Function registered with `R_BLECTRL_RegisterCb()`.

`evt.conn` must be referred when `BLECTRL_EVT_CONNECTED` event is notified.

`evt.disconn` must be referred when `BLECTRL_EVT_DISCONNECTED` event is notified.

`evt.tx` must be referred when `BLECTRL_EVT_TRANSMITTED` event is notified.

`evt.rx` must be referred when `BLECTRL_EVT_RECEIVED` event is notified.

3.7 R_USBCTRL Interface

This section describes the specification of R_USBCTRL Interface provided by the USB Control Application.

3.7.1 R_USBCTRL_Open

R_USBCTRL_Open		
Initialize USB Control processing		
Header	r_usbctrl_if.h	
Definition	`e_usbctrl_status_t R_USBCTRL_Open(void);`	
Parameter	None	
Return Value	`USBCTRL_SUCCESS`	Success
	`USBCTRL_ERR_API`	Failure: USB Driver Initialization Failed
Supplement	After the insertion of USB, the transition to Configured state is notified by the `USBCTRL_EVT_CDC_ENABLED` event. The enabling of Communication Device Class is notified by the `USBCTRL_EVT_CDC_ENABLED` event. After the removal of USB, the transition to Powered state is notified by the `USBCTRL_EVT_DETACH` event. When the CLKOUT_RF pin output clock is used as a reference clock of the USB clock (UCLK), `R_BLECTRL_Open()` shall be executed before the execution of this function.	

3.7.2 R_USBCTRL_Close

R_USBCTRL_Close		
Finalize USB Control processing		
Header	r_usbctrl_if.h	
Definition	`e_usbctrl_status_t R_USBCTRL_Close(void);`	
Parameter	None	
Return Value	`USBCTRL_SUCCESS`	Success
	`USBCTRL_ERR_API`	Failure: USB Driver Finalization Failed
Supplement	None	

3.7.3 R_USBCTRL_RegisterCb

R_USBCTRL_RegisterCb		
Register USB Control Event Callback Function		
Header	r_usbctrl_if.h	
Definition	`e_usbctrl_status_t R_USBCTRL_RegisterCb(usbctrl_evt_cb_t cb);`	
Parameter	(IN) `cb`	USB Control Event Callback Function
Return Value	`USBCTRL_SUCCESS`	Success
	`USBCTRL_ERR_ARG`	Failure: argument `cb` is NULL
Supplement	None	

3.7.4 R_USBCTRL_RegisterTxBuffer

R_USBCTRL_RegisterTxBuffer			
Register USB Transmission Buffer			
Header	r_usbctrl_if.h		
Definition	`e_usbctrl_status_t R_USBCTRL_RegisterTxBuffer(uint8_t *buffer, uint16_t size);`		
Parameter	(IN)	`buffer`	Transmission Buffer
	(IN)	`size`	Transmission Buffer Size
Return Value	`USBCTRL_SUCCESS`		Success
	`USBCTRL_ERR_ARG`		Failure: argument `buffer` is NULL or `buffer` is 0
	`USBCTRL_ERR_STATE`		Failure: Uninitialized state or Communication operation state
Supplement	None		

3.7.5 R_USBCTRL_SuspendRx

R_USBCTRL_SuspendRx		
Resume USB Reception		
Header	r_usbctrl_if.h	
Definition	`e_usbctrl_status_t R_USBCTRL_SuspendRx(void);`	
Parameter	None	
Return Value	`USBCTRL_SUCCESS`	Success
	`USBCTRL_ERR_STATE`	Failure: CDC Disabled state
Supplement	This function must be executed after the notification of the `USBCTRL_EVT_CDC_ENABLED` event. USB Reception is resumed by executing `R_USBCTRL_ResumeRx()`. USB Reception state is canceled by the removal of USB.	

3.7.6 R_USBCTRL_ResumeRx

R_USBCTRL_ResumeRx		
Resume USB Reception		
Header	r_usbctrl_if.h	
Definition	`e_usbctrl_status_t R_USBCTRL_ResumeRx(void);`	
Parameter	None	
Return Value	`USBCTRL_SUCCESS`	Success
	`USBCTRL_ERR_STATE`	Failure: Communication Device Class(CDC) is disabled
Supplement	None	

3.7.7 R_USBCTRL_GetTxBufferSpace

R_USBCTRL_GetTxBufferSpace	
Get the empty size of USB Transmission Buffer	
Header	r_usbctrl_if.h
Definition	`uint16_t R_USBCTRL_GetTxBufferSpace(void);`
Parameter	None
Return Value	Empty Size of Transmission Buffer [byte]
Supplement	None

3.7.8 R_USBCTRL_TransmitData

R_USBCTRL_TransmitData

Transmit data to the Host device

Header	r_usbctrl_if.h		
Definition	`uint16_t R_USBCTRL_TransmitData(const uint8_t *data, uint16_t size);`		
Parameter	(IN)	`data`	Data to be Transmitted
	(IN)	`size`	Size of Data to be Transmitted [byte]
Return Value	Size of Data stored in Transmission Buffer [byte]		
Supplement	This function must be executed after the notification of the `USBCTRL_EVT_CDC_ENABLED` event. Data Transmission is notified by the `USBCTRL_EVT_TRANSMITTED` event.		
Limitation	When using FreeRTOS 1. Do not call this function from an interrupt handler because this function uses `xTaskNotifyGive()`. 2. Do no call this function from multiple tasks because the transmission buffer operation in this function is not guarded as a critical section.		

3.7.9 R_USBCTRL_GetAllowedLpcMode

R_USBCTRL_GetAllowedLpcMode

Get the Low Power Consumption mode that can be switched

Header	r_usbctrl_if.h
Definition	`lpc_low_power_mode_t R_USBCTRL_GetAllowedLpcMode(void);`
Parameter	None
Return Value	Low Power Consumption mode that can be switched

Return Value	Low Power Consumption Mode		
	SLEEP	DEEP_SLEEP	SW_STANDBY
`LPC_LP_INVALID_MODE`	×	×	×
`LPC_LP_SLEEP`	○	×	×
`LPC_LP_DEEP_SLEEP`	Not Returned by this function		
`LPC_LP_SW_STANDBY`	○	○	○

Supplement	None

3.7.10 usbctrl_main

usbctrl_main

Main Context of the USB Control Application

Header	r_usbctrl_if.h
Definition	`void usbctrl_main(void);`
Parameter	None
Return Value	None
Supplement	When OS is not used, call this function in the main loop. When FreeRTOS is used, create a task for executing this function and call this function in a loop of the task.

3.7.11 usbctrl_evt_cb_t

usbctrl_evt_cb_t

USB Control Event Callback Function

Header	r_usbctrl_if.h		
Definition	`typedef void (*usbctrl_evt_cb_t)(e_usbctrl_evt_t type, st_usbctrl_evt_param_t *param);`		
Parameter	(OUT)	`type`	USB Control Event
	(OUT)	`param`	USB Control Event Parameter
Return Value	None		
Supplement	Implement a USB Control Event Callback Function in a upper application and register with `R_USBCTRL_RegisterCb()`.		

3.7.12 e_usbctrl_status_t

e_usbctrl_status_t

Execution Result of R_USBCTRL Interface Function

Header: r_usbctrl_if.h

Definition:

```
typedef enum e_usbctrl_status
{
    USBCTRL_SUCCESS = 0,
    USBCTRL_ERR_UNSUPPORT,      /* Unsupported API */
    USBCTRL_ERR_ARG,           /* Invalid Argument */
    USBCTRL_ERR_STATE,         /* Invalid State */
    USBCTRL_ERR_API,           /* USB API Error */
} e_usbctrl_status_t;
```

3.7.13 e_usbctrl_evt_t

e_usbctrl_evt_t

BLE Control Events

Header: r_usbctrl_if.h

Definition:

```
typedef enum e_usbctrl_evt
{
    USBCTRL_EVT_NONE = 0,
    USBCTRL_EVT_SUSPEND,
    USBCTRL_EVT_RESUME,
    USBCTRL_EVT_DETACH,
    USBCTRL_EVT_DEFAULT,
    USBCTRL_EVT_CONFIGURED,
    USBCTRL_EVT_CDC_ENABLED,
    USBCTRL_EVT_TRANSMITTED,
    USBCTRL_EVT_RECEIVED
} e_usbctrl_evt_t;
```

Supplement: BLE Control Events are notified by the Callback Function registered with `R_USBCTRL_RegisterCb()`.

For the notification timing of USB Control Events, refer to Subsection 3.7.16.

3.7.14 st_usbctrl_evt_param_t

st_usbctrl_evt_param_t

USB Control Event Parameters

Header r_usbctrl_if.h

Definition

```
typedef struct st_usbctrl_evt_cdc
{
    /* dte(data terminal equipment) rate in bps */
    uint32_t rate;
    /* data bits */
    uint8_t  bits;
} st_usbctrl_evt_cdc_t;

typedef struct st_usbctrl_evt_tx
{
    /* transmitted data size in bytes */
    uint16_t size;
} st_usbctrl_evt_tx_t;

typedef struct st_usbctrl_evt_rx
{
    /* pointer to the data received */
    uint8_t *data;
    /* bytes of the data size received */
    uint16_t size;
} st_usbctrl_evt_rx_t;

typedef struct st_usbctrl_evt_param
{
    union
    {
        /* USBCTRL_EVT_CDC_ENABLED event parameter */
        st_usbctrl_evt_cdc_t    cdc;
        /* USBCTRL_EVT_TRANSMITTED event parameter */
        st_usbctrl_evt_tx_t     tx;
        /* USBCTRL_EVT_RECEIVED event parameter */
        st_usbctrl_evt_rx_t     rx;
    } evt;
} st_usbctrl_evt_param_t;
```

Supplement

USB Control Event Parameters are notified by the Callback Function registered with `R_USBCTRL_RegisterCb()`.

`evt.cdc` must be referred when `USBCTRL_EVT_CDC_ENABLED` event is notified.

`evt.tx` must be referred when `USBCTRL_EVT_TRANSMITTED` event is notified.

`evt.rx` must be referred when `USBCTRL_EVT_RECEIVED` event is notified.

3.7.15 State Transition of BLE Control Processing

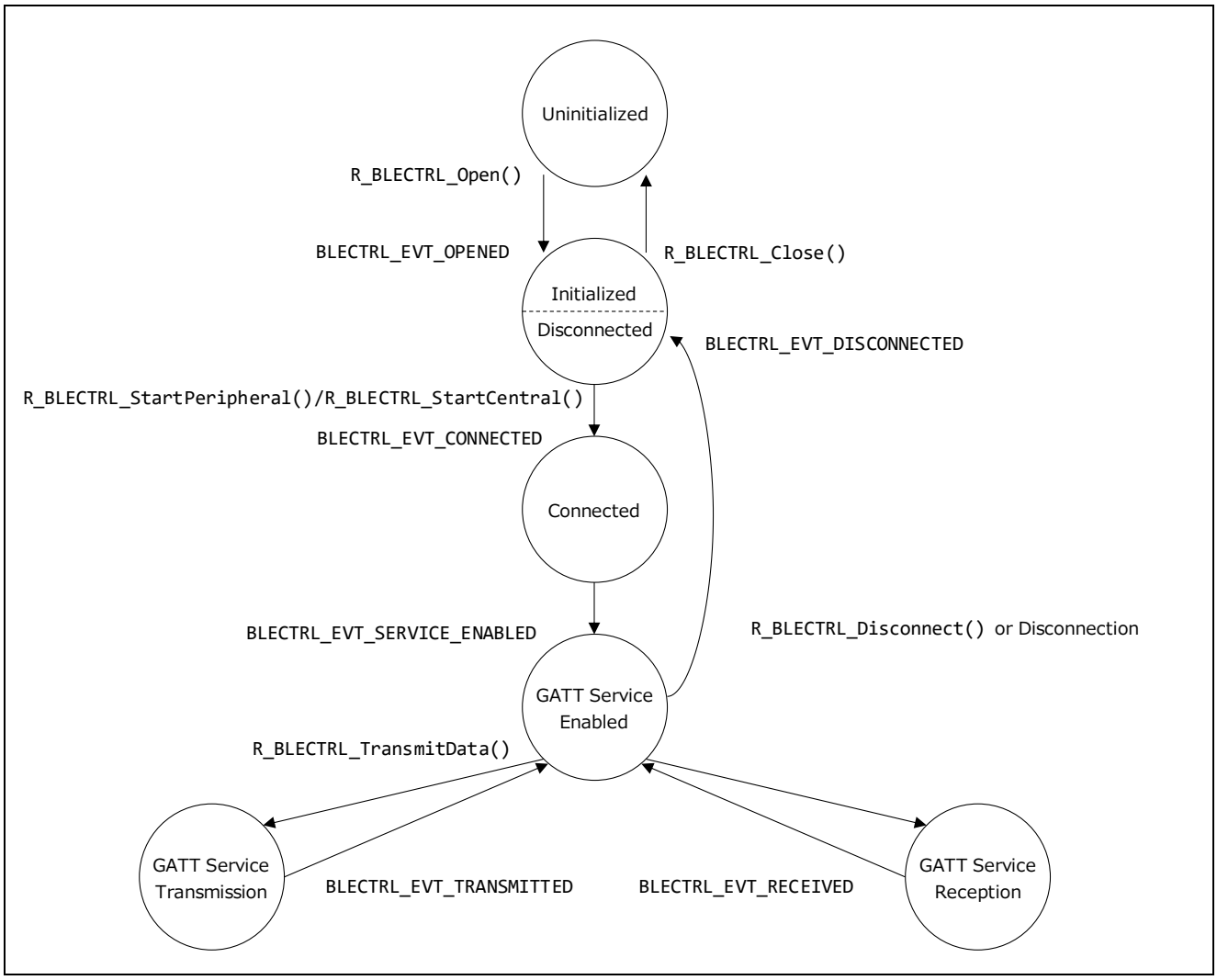

Figure 3-20 State Transition of BLE Control Processing

3.7.16 State Transition of USB Control Transition

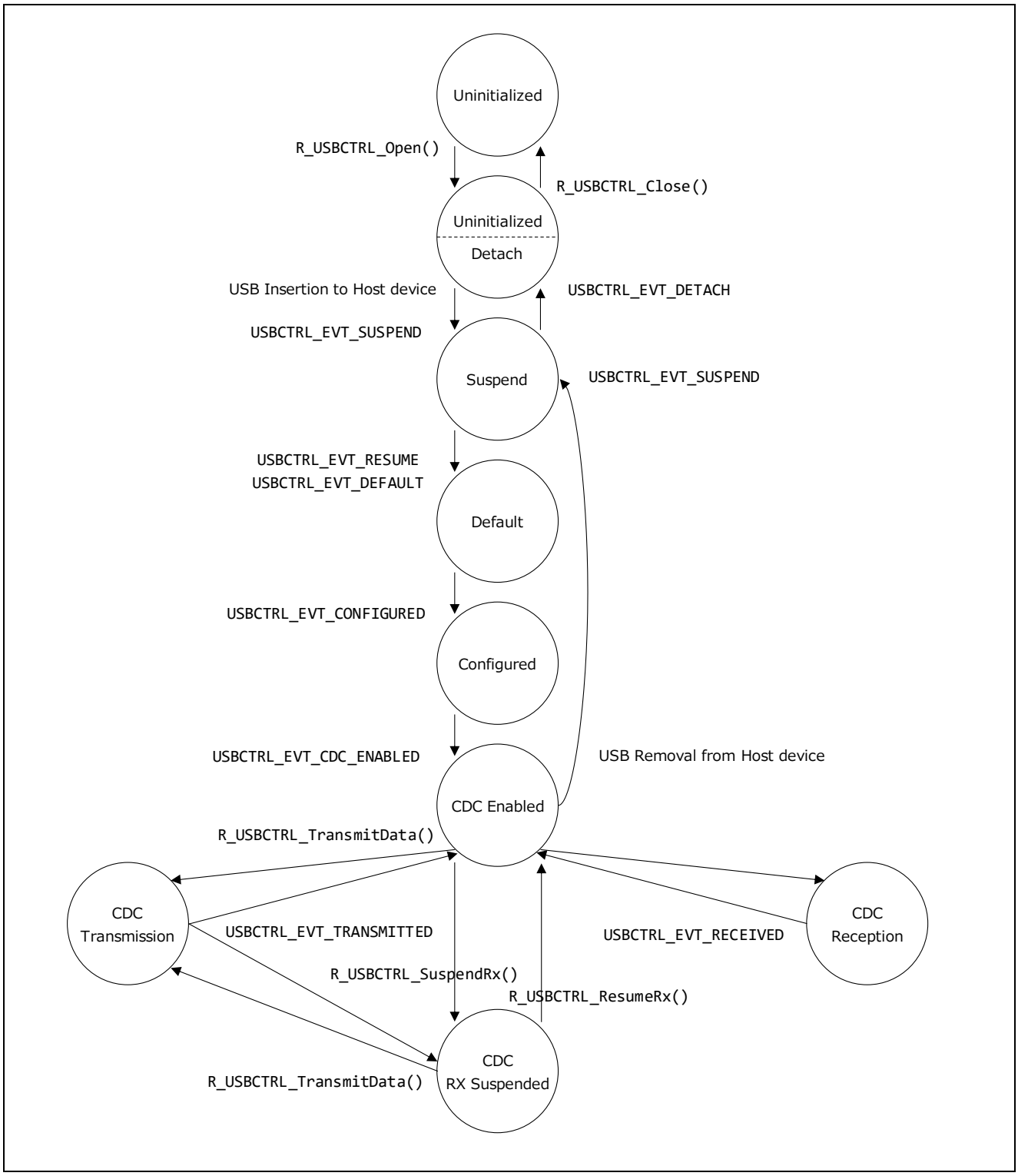

Figure 3-21 State Transition of USB Control Processing

3.8 How to Customize and Notes for Final Products

3.8.1 Changing the Service Configuration of the BLE Control Application

The sample program generated the Data I/O Service by using QE for BLE[RA, RE, RX]. For the default configuration of the Data I/O Service, refer to Table 3-10. By using the QE for BLE to generate source code, you can extend the functionality of the Data I/O Service and define new services. For more information on the QE for BLE, refer to the following website.

QE for BLE: Development Assistance Tool for Bluetooth® Low Energy
https://www.renesas.com/qe-ble

The procedure to start QE for BLE[RA, RE, RX] and update profile configuration is shown below.

1. To start QE for BLE[RA, RE, RX], select [Renesas Views]→[Renesas QE]→[R_BLE Custom Profile RA, RE, RX (QE)] in the menu of e² studio.

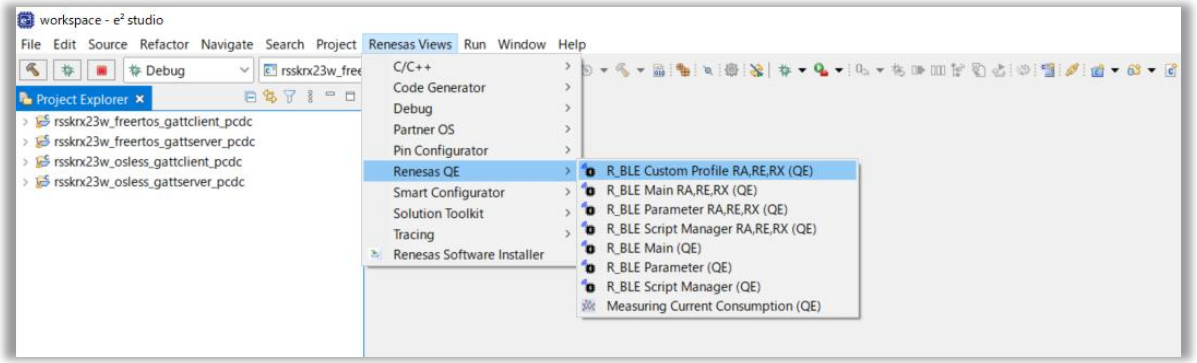

Figure 3-22 [Renesas Views]→[Renesas QE]→[R_BLE Custom Profile RA,RE,RX (QE)]

2. Select the project in [R_BLE Custom Profile RA, RE, RX (QE)].

3. The current profile configuration is displayed in [R_BLE Custom Profile RA, RE, RX (QE)].

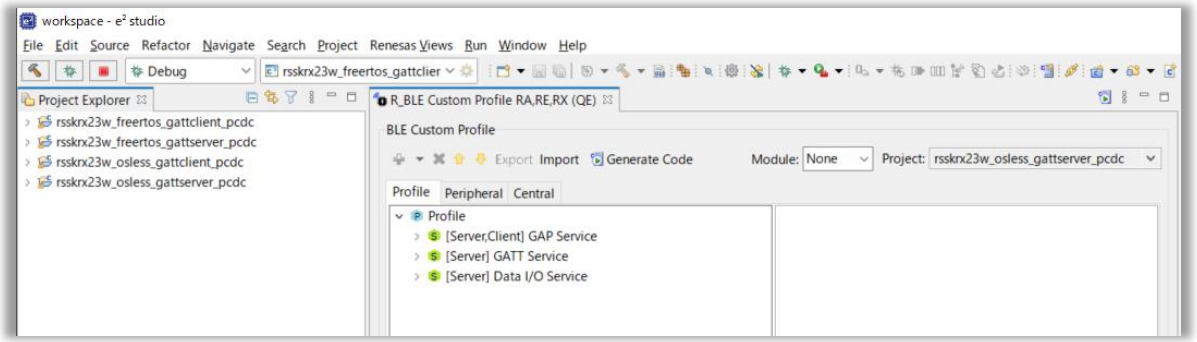

Figure 3-23 R_BLE Custom Profile RA,RE,RX (QE)

4. Refer to the following document to design the profile configuration and implement your program to the source code generated by QE for BLE.

RX23W Group Bluetooth Low Energy Profile Developer's Guide (R01AN4553)
https://www.renesas.com/document/apn/rx23w-group-bluetooth-low-energy-profile-developers-guide-rev110

3.8.2 Operation Setting for enhancing Throughput of the BLE Control Application

The sample program adds the following processing to the app_main.c generated by QE for BLE to enhance throughput during a Bluetooth LE connection.

- When operating as a Central device, `R_BLE_GAP_SetPhy()` is called to change to 2M PHY
- When operating as a Central device, `R_BLE_GATTC_ReqExMtu()` to expand ATT MTU size from 23byts to 247bytes.

3.8.3 Security features of BLE Control Application

The sample program does not perform pairing, encryption, encryption, or privacy of Bluetooth LE. To use these security features, refer to the following document.

RX23W Group Bluetooth Low Energy Application Developer's Guide (R01AN5504)
https://www.renesas.com/document/apn/rx23w-group-bluetooth-low-energy-application-developers-guide-rev110

→Chapter 9 "Security"

3.8.4 Permission of Software Standby Mode of BLE Control Application

The sample program allows MCU to enter to Software standby mode of Low Power Consumption state by calling `R_BLE_LPC_SetInhibitSoftwareStandby(false)` of the BLE FIT module. In the Software standby mode, SCI is stopped. The sample program uses the SCI8 to send an operation log to the console only. However, if the SCI8 is also used to receive a string from the console, the transition to the Software standby mode must be prohibited.

3.8.5 Bluetooth SIG Qualification

Final products using Bluetooth technology must be qualified by Bluetooth SIG. For information on how to acquire Bluetooth qualification for the final products using the Bluetooth LE MCU or module manufactured by Renesas, refer to the following document.

Bluetooth LE microcomputer / module Bluetooth qualification acquisition (R01AN3177)
https://www.renesas.com/document/apn/bluetooth-le-microcomputer-module-bluetooth-qualification-acquisition-application-note-rev140

3.8.6 Compliance with the Radio Laws and Acquisition of Certification

Final products using wireless technology must comply with the radio laws of the countries where they are sold. For certification, refer to the documents issued by each country's certification organization or certification agency.

The RX23W module (R5F523W8CDLN/ R5F523W8DDLN) has acquired Radio Law (Japan), FCC/ISED (North America), and CE (Europe) certifications.

3.8.7 DTC Transfer of DMA Transfer of USB Driver

The USB Basic Mini FIT Module can use DTC transfer or DMA transfer. The procedure for using DTC transfer or DMA transfer is shown below.

1. Double-click the smart configurator configuration file (.scfg) in [Project Explorer] of e² studio to open the smart configurator.

2. Click [Add component] button (green + icon) in [Software component configuration] to add the DTC FTT Module (r_dtc_rx) or the DMA FIT Module (r_dmaca_rx) in New Component dialog.

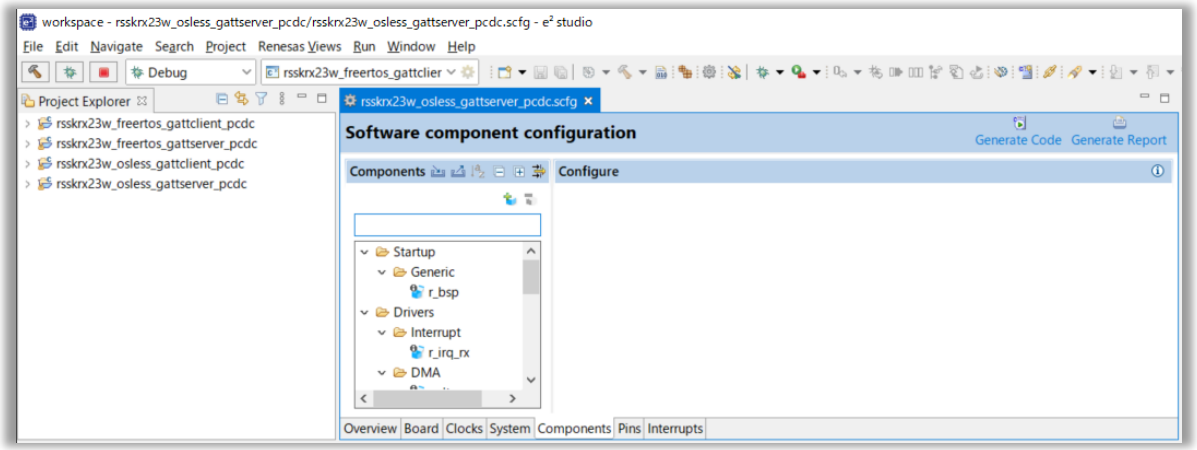

Figure 3-24 Software Component Configuration of the Smart Configurator

3. Select the USB Basic Mini FIT Module (r_usb_basic_mini) in the Software component configuration. When using DTC transfer, set [DTC use setting] (USB_CFG_DTC) to "Using DTC". When using DMA transfer, set [DMA use setting] (USB_CFG_DMA) to "Using DMA".

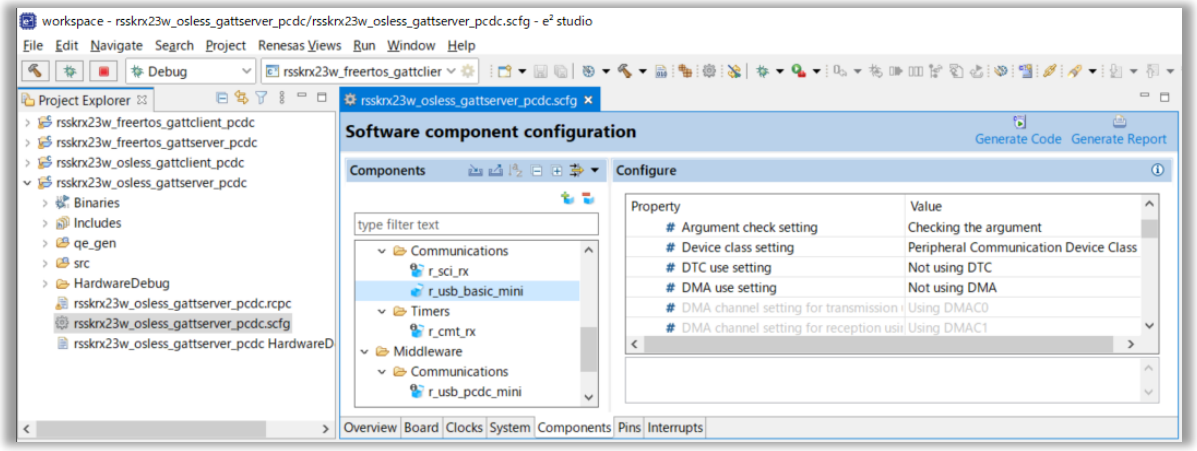

Figure 3-25 Configuration of USB Basic Mini FIT Module (r_usb_basic_mini)

4. Click [Generate Code] button of [Software component configuration] to generate the source code of the software components. When using DMA transfer, no further steps are needed.

5. When using DTC transfer, the heap area must be expanded to allow the DTC FIT Module to allocate the DTC vector table area dynamically. For the heap size required by the DTC FIT Module, refer to the DTC_VECTOR_TABLE_SIZE_BYTES macro in the following code file generated by the Smart Configurator. The following is the macro definition for the DTC FIT Module V3.80.

   {Project Folder}¥src¥smc_gen¥r_dtc_rx¥src¥targets¥rx23w¥r_dtc_rx_target.h

```
/* Size of DTC Vector table (in byte units) */
#define DTC_VECTOR_TABLE_SIZE_BYTES (0x3E8 + 0x400)
```

   Select the Board Support Package Module (r_bsp) in the Software component configuration. Add the size specified by the DTC_VECTOR_TABLE_SIZE_BYTES to the heap size required by the application or system and set to [Heap size] (BSP_CFG_HEAP_BYTES).

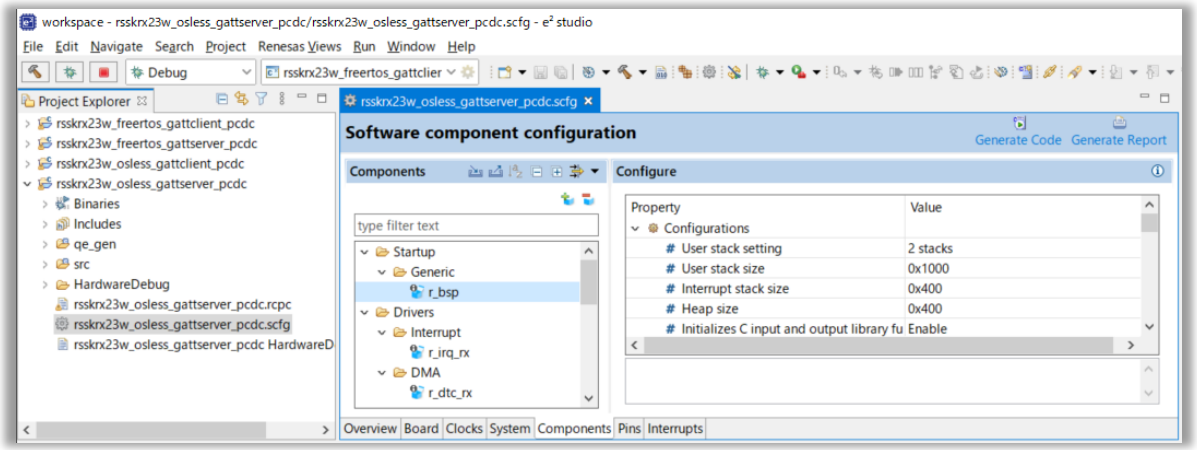

Figure 3-26 Configuration of Board Support Package Module (r_usb_basic_mini)

6. Click [Generate Code] button of [Software component configuration] to generate the source code of the software components.

3.8.8 Getting a USB Vendor ID

Vendor ID for USB products must be acquired by the customer (vendor) who sells them as the final product. For information on acquiring Vendor IDs, refer to the USB Implementers Forum (USB-IF) website.

Getting a Vendor ID - USB Implementers Forum, Inc.
https://www.usb.org/developers/vendor/

Set the acquired Vendor ID as well as the Product ID, Release Number, etc. that can be assigned by the vendor in the following code file.

{Project Folder}¥src¥usbctrl¥r_usb_pcdc_descriptor.c

```
/* Vendor ID */
#define USB_VENDORID                    (0x0000u)
/* Product ID */
#define USB_PRODUCTID                   (0x0002u)
/* Release Number */
#define USB_RELEASE                     (0x0200u)
```

Figure 3-27 Macro Definitions of Vendor ID, Product ID, and Release Number

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 04, 2022	-	First Issue

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

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/.