RA4W1 Group

BLE sample application

Introduction

This document describes the accompanying sample application which controls the Bluetooth® Low Energy communication module. In this document, the module which performs Bluetooth® Low Energy communication is referred to as the BLE module.

Target Device

RA4W1 Group

Related Documents

Bluetooth Core Specification (https://www.bluetooth.com)

RA4W1 Group User's Manual: Hardware (R01UH0883)

Renesas Flexible Software Package User's Manual

e² studio Getting Started Guide (R20UT4204)

Renesas Flash Programmer User's manual (R01UT5757)

Tuning procedure of Bluetooth dedicated clock frequency (R01AN4887)

RA4W1 Group Bluetooth LE Profile API Document User's Manual (R11UM0154)

Bluetooth Low Energy Profile Developer's Guide (R01AN5428)

Host Controller Interface Firmware(R01AN5429)

Public BD Address writing tool(R01AN5439)

EK-RA4W1 Quick Start Guide (R20QS0015)

QE for BLE [RA, RE, RX] V1.5.0 Release Note (R20UT5145EJ)

Related Environments

Refer to section 2.1 Operating environment.

Contents

1. Overview

Demo projects accompanying this document are shown in Table 1. These projects are provided as BLE sample application using BLE module.

Table 1. Demo Projects

Demo Project	Description
ble_baremetal_ek_ra4w1	GATT Server demo project for EK-RA4W1 without RTOS.
ble_freertos_ek_ra4w1	GATT Server demo project for EK-RA4W1 using FreeRTOS and event group technique has been used for task synchronization. This project provides for backward compatibility with versions prior to FSP3.8.
ble_freertos_ek_ra4w1_semaphore	GATT Server demo project for EK-RA4W1 using FreeRTOS and semaphore give / take method has been used for task synchronization.
ble_azurertos_ek_ra4w1	GATT Server demo project for EK-RA4W1 using Azure RTOS.
ble_bearmetal_ek_ra4w1_client	GATT Client demo project for EK-RA4W1 without RTOS
ble_freertos_ek_ra4w1_client	GATT Client demo project for EK-RA4W1 using FreeRTOS and event group technique has been used for task synchronization. This project provides for backward compatibility with versions prior to FSP3.8.
ble_freertos_ek_ra4w1_client_semaphore	GATT Client demo project for EK-RA4W1 using FreeRTOS and semaphore give / take method has been used for task synchronization.
ble_azurertos_ek_ra4w1_client	GATT Client demo project for EK-RA4W1 using Azure RTOS

These projects can work on EK-RA4W1 board or user's custom board. GATT Server demo projects perform GATT Server role. They can change the blink rate of LED mounted on the board from remote device (e.g. smart phone) and send notification by pushing switch mounted on the board to remote device via BLE communication. LED and switch (e.g. push button) connected to RA4W1 GPIOs are necessary on user's custom board when this demo project running on user's custom board.

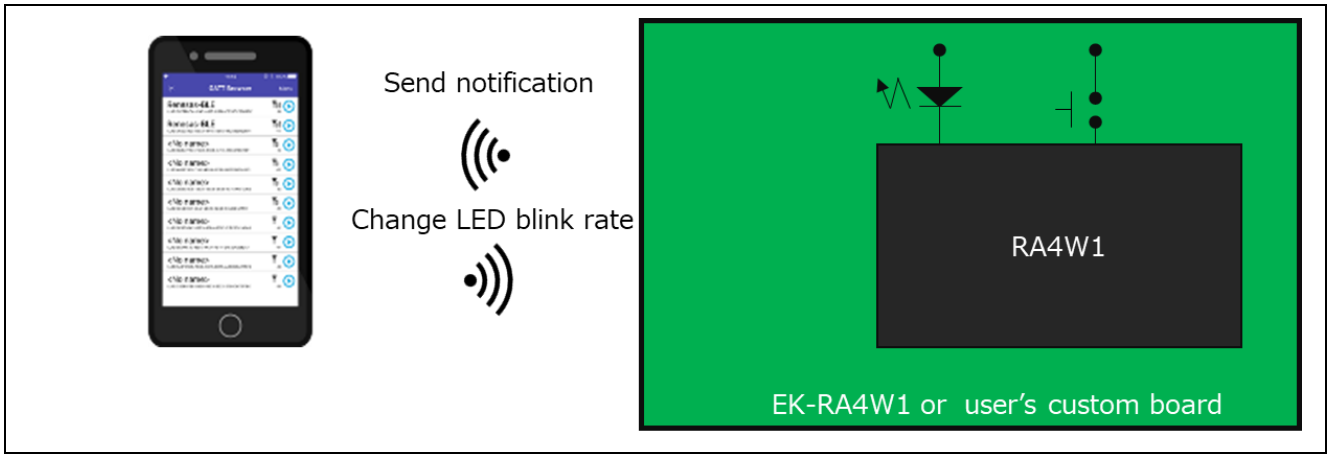

Figure 1. GATT Server demo project operating environment

GATT Client demo projects perform GATT Client role. They have the functionality of CLI (Command Line Interface) which can be accessed by the terminal emulator like Tera Term on PC connecting with EK-RA4W1 board via USB cable. They can perform various procedures in relation to GATT Client by receiving commands via CLI.

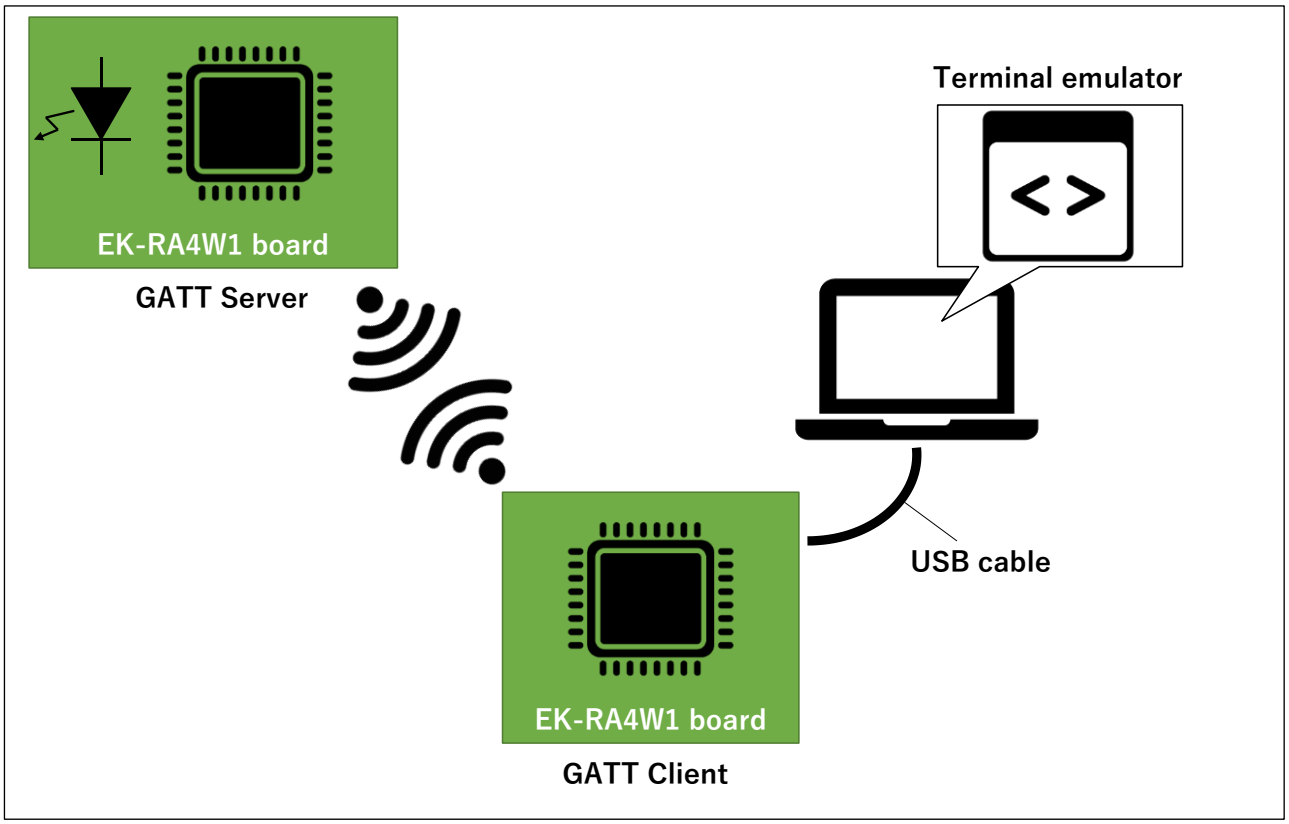

Figure 2. GATT Client demo project operating environment

1.1 BLE features

BLE module provides following BLE features which are compliant with Bluetooth version 5.0.

Bluetooth 5.0 Features

- LE 2M PHY

  Supports BLE communication with 2Msym/s PHY.

- LE Coded PHY

  Supports BLE communication with Coded PHY. Communication over long range than 1M PHY and 2M PHY is possible.

- LE Advertising Extensions

  An extension of Advertising. Up to 4 independent advertising can be performed simultaneously.

- LE Channel Selection Algorithm #2

  Selects a channel using the algorithm for selecting a hopping channel added in Bluetooth 5.0.

- High Duty Cycle Non-Connectable Advertising

  Supports non-connectable advertising with a minimum interval of 20 msec.

Bluetooth 4.2 Features

- LE Secure Connections

  Elliptic curve Diffie-Hellman key agreement method (ECDH) supports passive eavesdropping pairing.

- Link Layer Privacy

  Avoids tracking from other BLE devices by changing the BD Address periodically.

- Link Layer Extended Scanner Filter policies

  Resolvable private addresses as well as part of the filtering process.

- LE Data Packet Length Extension

  Expands the BLE data communication packet size up to 251bytes.

Bluetooth 4.1 Features

- LE L2CAP Connection Oriented Channel Support

  Supports communication using the L2CAP credit based flow control channel.

- Low Duty Cycle Directed Advertising

  Supports low duty cycle advertising for reconnection with known devices.

- 32-bit UUID Support in LE

  Supports GATT 32-bit UUID.

- LE Link Layer Topology

  Supports both Master and Slave roles and can operate as Master when connected to a remote device and as Slave when connected to another remote device.

- LE Ping

  After connection encryption, this feature checks whether connection is maintained by a packet transmission request including MIC field.

GAP Role

GAP Role supports the following.

- Central: A device that sends a connection request to a peripheral device.
- Peripheral: A device that accepts connection requests from Central and establishes a connection.
- Observer: A device that scans Advertising.
- Broadcaster: A device that sends Advertising.

GATT Role

GATT Role supports the following.

- Server: A device that prepares Characteristic provided by service in GATT Database and responds to requests from Client.
- Client: A device that makes request for services provided by Server.

1.2 BLE application software structure

Figure 3 shows software structure of BLE application in BareMetal environment.

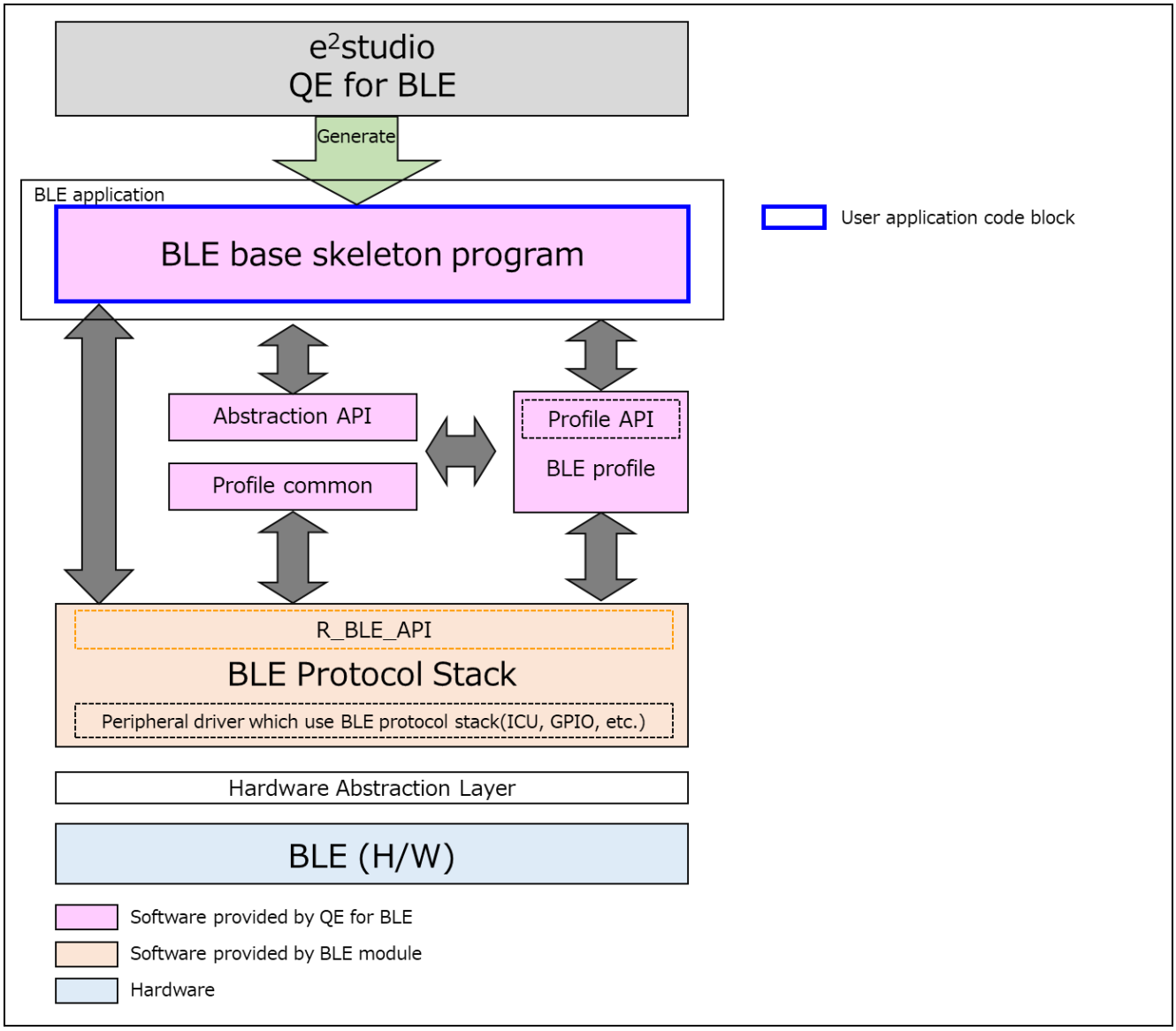

Figure 3. Software structure (BareMetal)

Figure 4 shows software structure of BLE application in case of FreeRTOS environment and event group technique has been used for task synchronization. BLE application is divided it into two or more tasks, BLE Core Task and GATT application tasks. BLE Core Task performs initialization and BLE related processing except profile event processing. GATT application task performs profile event processing.

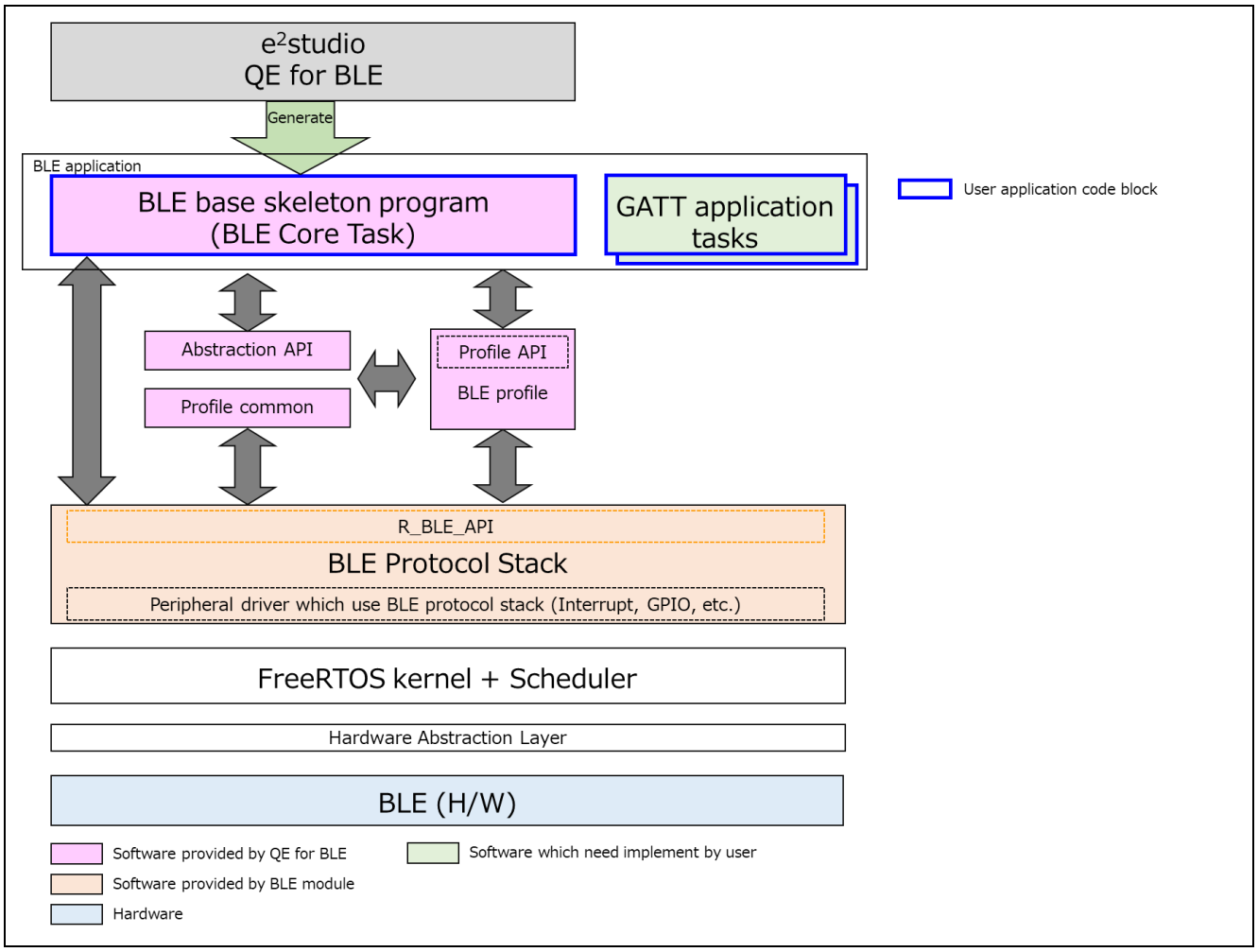

Figure 4. Software structure (FreeRTOS, Event group)

Figure 5 shows software structure of BLE application in case of FreeRTOS environment and semaphore give / take method has been used for task synchronization. BLE application is divided it into three or more tasks, BLE Core Task, BLE Execute Task and GATT application tasks. BLE Core Task performs initialization. BLE Execute Task performs BLE related processing except profile event processing. GATT application task performs profile event processing.

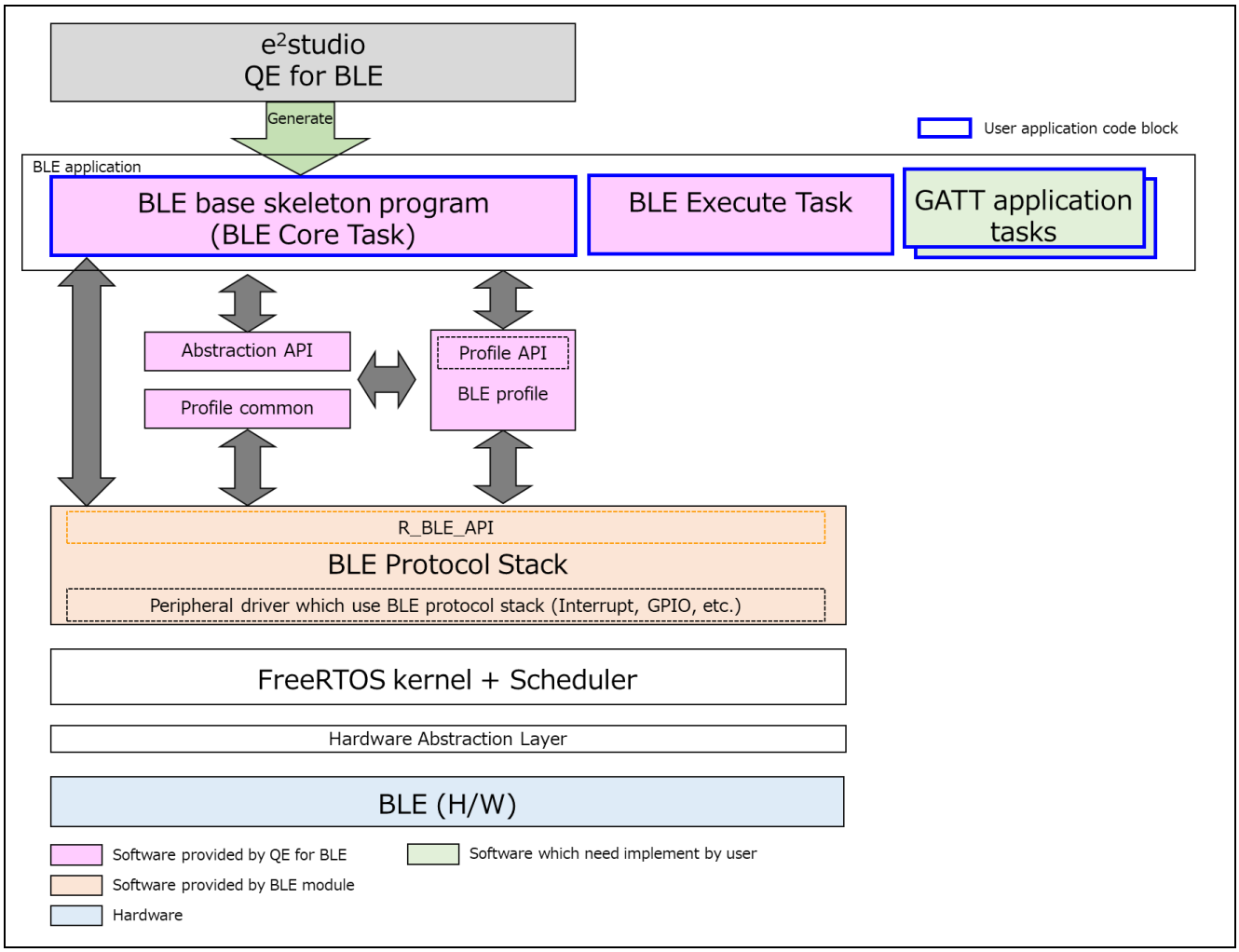

Figure 5. Software structure (FreeRTOS, Semaphore)

Task switching from interrupt context by using event group method should through FreeRTOS daemon task (Prior FSP3.8 environment). To reduce such overhead, made it possible to select semaphore synchronization method as *Synchronization Type* property of *BLE_Driver* FSP module from FSP4.0 or later.

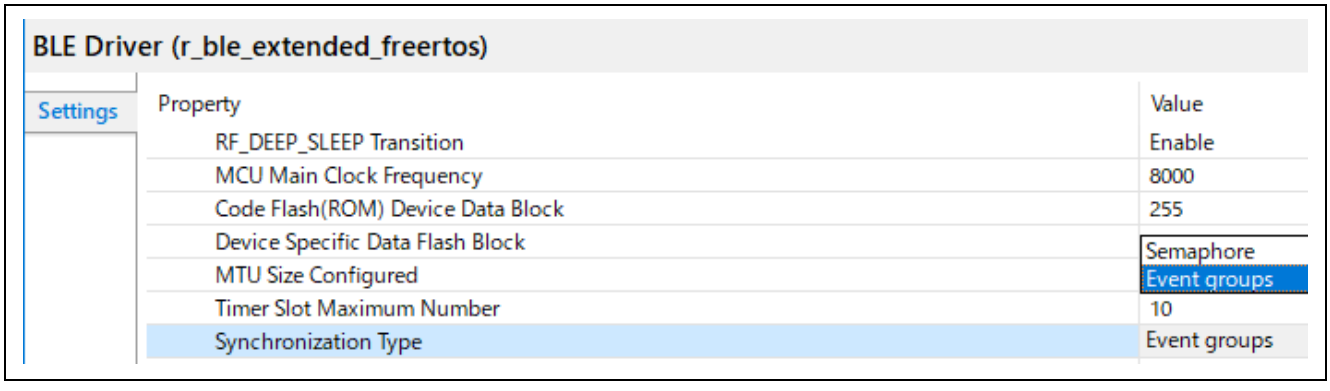

Figure 6. Synchronization Type

Figure 7 shows software structure of BLE application in Azure RTOS environment. BLE application is divided it into three or more tasks, BLE Core Task, BLE Execute Task and GATT application tasks. BLE Core Task performs initialization. BLE Execute Task performs BLE related processing except profile event processing. GATT application task performs profile event processing.

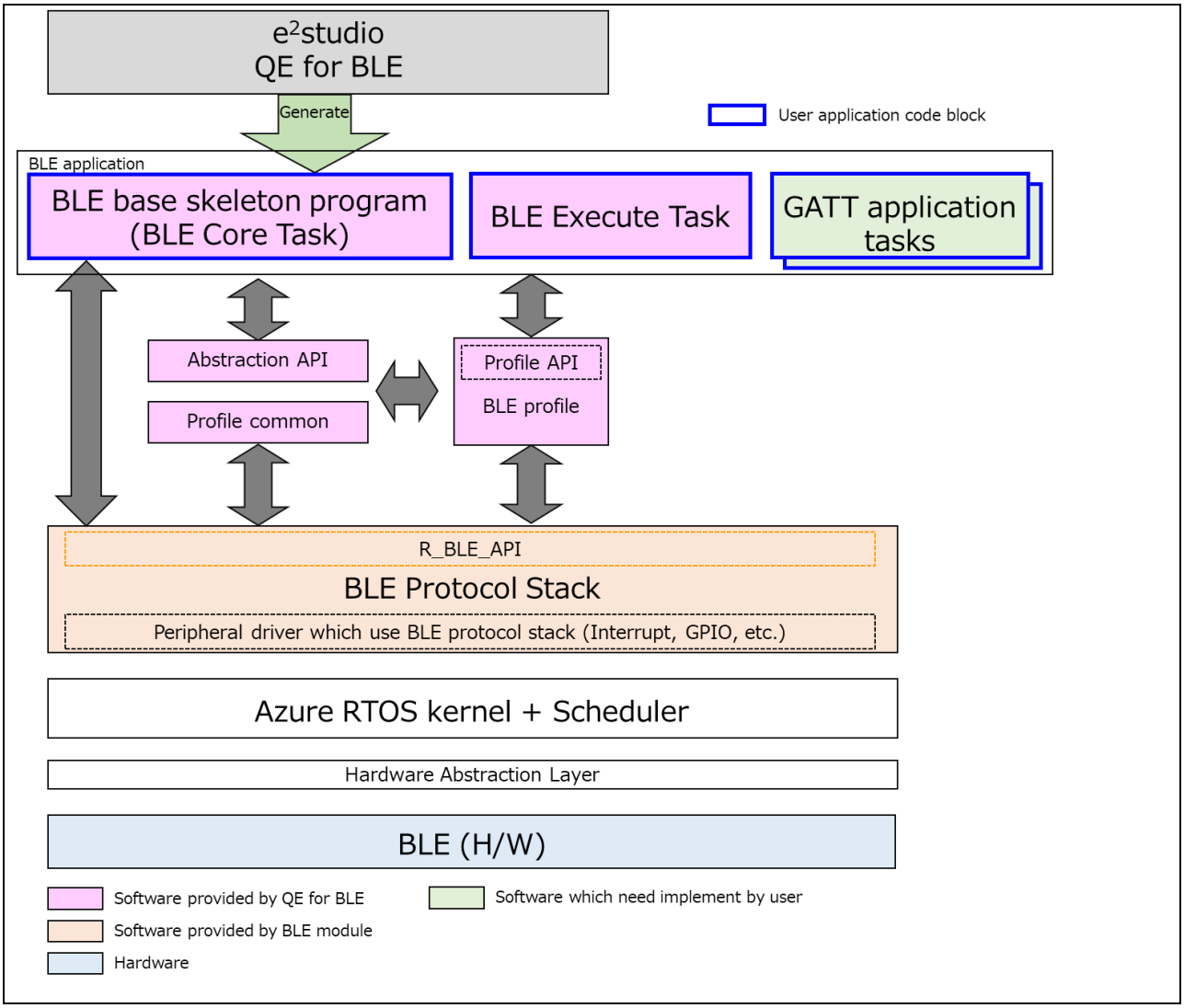

Figure 7. Software structure (Azure RTOS)

The QE for BLE tool generates C source codes of BLE base skeleton program for BLE application and BLE Profile. Renesas recommends using the QE for BLE tool when developing BLE application.

BLE application uses BLE functions via following APIs.

- **GAP API (R_BLE_GAP_XXX, R_BLE_L2CAP_XXX, R_BLE_VS_XXX)**

  To use BLE function. Refer to “Renesas Flexible Software Package User’s Manual” for details.

- **Discovery API (R_BLE_Disc_XXX)**

  To perform service discovery. These APIs are generated by QE for BLE. Refer to “Bluetooth Low Energy Profile Developer’s Guide(R01AN5428)” and “RA4W1 Group Bluetooth LE Profile API Document User's Manual (R11UM0154)” for the details of the API.

- **GATT Server API (R_BLE_GATTS_XXX)**
  To use GATT profile server function. These APIs are generated by QE for BLE. Refer to “Bluetooth Low Energy Profile Developer’s Guide(R01AN5428)” and “RA4W1 Group Bluetooth LE Profile API Document User's Manual (R11UM0154)” for the details of the API.
- **GATT Client API (R_BLE_GATTC_XXX)**
  To use GATT profile client function. These APIs are generated by QE for BLE. Refer to “Bluetooth Low Energy Profile Developer’s Guide(R01AN5428)” and “RA4W1 Group Bluetooth LE Profile API Document User's Manual (R11UM0154)” for the details of the API.

- **GATT service API (R_BLE_[GATT service abbreviation + S(Server) or C(Client)]_XXX)**
  Auxiliary functions available for the BLE Application. These APIs are generated by QE for BLE. Refer to “Bluetooth Low Energy Profile Developer’s Guide(R01AN5428)” and “RA4W1 Group Bluetooth LE Profile API Document User's Manual (R11UM0154)” for the details of the API.

- **Abstraction API (RM_BLE_ABS_XXX)**
  Makes it easy to use the frequently used BLE functions. Refer to “Renesas Flexible Software Package User’s Manual” for details.

APIs that can be called from BLE core task and GATT application task in FreeRTOS and Azure RTOS environment have the restrictions. Following categories of API can call only from BLE core task.

- GAP API (R_BLE_GAP_XXX, R_BLE_L2CAP_XXX, R_BLE_VS_XXX)
- Discovery API (R_BLE_Disc_XXX)
- GATT Server API (R_BLE_GATTS_XXX)
- GATT Client API (R_BLE_GATTC_XXX)
- Abstraction API (RM_BLE_ABS_XXX)

Following category of API can call from either BLE core task or GATT application task.

- GATT service API (R_BLE_[GATT service abbreviation + S(Server) or C(Client)]_XXX)

When BLE GATT Application task calls GATT service API, GATT communication is processed in BLE core task or BLE execute task. Figure 8 shows Bluetooth LE communication when two BLE GATT Application tasks control GATT services and send notification of the GATT service characteristic on FreeRTOS / Azure RTOS environment.

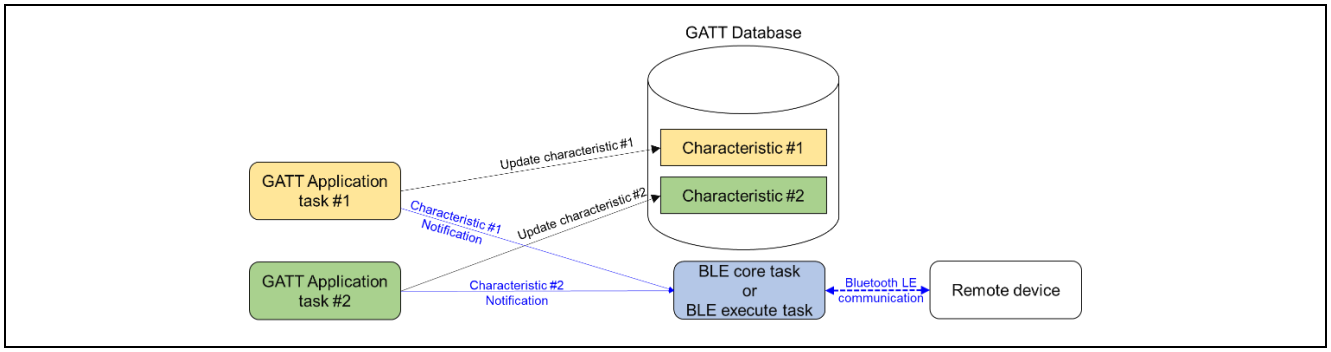

Figure 8. BLE core task and GATT Application task

Table 2 shows the directory / file structure of BLE application when using QE for BLE. Items shown in **bold** could add / modify according to use case.

Table 2. Directory / File structure

Directory/File structure				Description
qe_gen	ble	discovery		Service discovery related APIs
		profile_cmn		Profile common APIs
		app_main.c		**Main code** **C source code where user describe the BLE application.**
		gatt_db.c		GATT Database
		gatt_db.h		GATT Database
		r_ble_XXX.c		Profile API XXX depends on the included profile name.
		r_ble_XXX.h		Profile API XXX depends on the included profile name
ra	fsp	inc	api	BLE interface file r_ble_api.h rm_ble_abs_api.h
			instances	Abstraction API(GAP) rm_ble_abs.h
		lib	r_ble	BLE Protocol Stack See also section 1.3.
		src	rm_ble_abs	Abstraction API(GAP) rm_ble_abs.c
	aws	amazon-freertos		FeeRTOS kernel (Only FreeRTOS environment)
	microsoft	azure-rtos		Azure RTOS kernel (Only Azure RTOS environment)
ra_gen	---	---		RA configuration generated.
ra_cfg	fsp_cfg	r_ble_cfg.h		Configuration option file
		rm_ble_abs_cfg.h		Configuration option file
		azure	tx/tx_user.h	Azure RTOS configuration (Only Azure RTOS environment)
	aws	FreeRTOSConfig.h		FreeRTOS configuration (Only FreeRTOS environment)
src	**---**	**hal_entry.c**		**User code entry point.** **(BareMetal)**
		XXX_entry.c		**User task creation.** **XXX depends on task name which defined by user.** **(Only RTOS environment)**
		*****.c**		**User created C source codes**
		*****.h**		**User created header files**

1.3 BLE protocol stack

The Bluetooth protocol stack provides as static library. Customer can select “Extended”, “Balance” and “Compact” type according to the supported BLE features. Supported BLE features of each type are shown in Table 3.

Table 3. Features supported by each type of BLE Protocol Stack

BLE Features	Library type		
	Extended	Balance	Compact
GAP role	Central, Peripheral, Observer, Broadcaster	Central, Peripheral, Observer, Broadcaster	Peripheral, Broadcaster
GATT role	Server, Client	Server, Client	Server, Client
LE 2M PHY	Yes	Yes	No
LE Coded PHY	Yes	Yes	No
LE Advertising Extensions	Yes	No	No
LE Channel Selection Algorithm #2	Yes	Yes	No
High Duty Cycle Non-Connectable Advertising	Yes	Yes	Yes
LE Secure Connections	Yes	Yes	Yes
Link Layer privacy	Yes	Yes	Yes
Link Layer Extended Scanner Filter policies	Yes	Yes	No
LE Data Packet Length Extension	Yes	Yes	Yes
LE L2CAP Connection Oriented Channel Support	Yes	No	No
Low Duty Cycle Directed Advertising	Yes	Yes	Yes
LE Link Layer Topology	Yes	Yes	No
LE Ping	Yes	Yes	Yes
32-bit UUID support in LE	Yes	Yes	Yes

2. How to use demo project

This chapter describes how to use demo project with this document.

2.1 Operating environment

Table 4 shows the hardware requirements for building and debugging BLE software.

Table 4. Hardware requirements

Hardware	Description
Host PC	Windows® 10 PC with USB interface.
MCU Board	The MCU used must support BLE functions. EK-RA4W1 [RTK7EKA4W1S00000BJ]
On-chip debugging emulators	The EK-RA4W1 has an on-board debugger (J-Link OB), therefore it is not necessary to prepare an emulator.
E2 lite emulator	Needed if user wants to write device-specific data (refer to section 4.2) in custom board by using Renesas Flash Programmer.
USB cables	Used to connect to the MCU board. EK-RA4W1: 2 USB A-microB cable

Table 5 shows the software requirements for build and debug BLE software.

Table 5. Software requirements

Software		Version	Description
GCC environment	e² studio	2022-10	Integrated development environment (IDE) for Renesas devices.
	GCC ARM Embedded	10.3-2021.10	C/C++ Compiler. (Download from e² studio installer)
	Renesas Flexible Software Package (FSP)	V4.1.0	Software package for making applications for the RA microcontroller series.
	QE for BLE[RA]	V1.5.0	Generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile. Install latest QE for BLE and QE utility by referring release note on following link. https://www.renesas.com/us/en/software-tool/qe-ble-development-assistance-tool-bluetooth-low-energy
	QE utility [RA]	V1.5.0	
	SEGGER J-Flash	V7.80c	Tool for programming the on-chip flash memory of microcontrollers.
Header files			All API calls and their supporting interface definitions located in r_ble_api.h and rm_ble_abs_api.h.
Integer types			It uses ANSI C99 “Exact width integer types”. These types are defined in stdint.h.
Endian			Little endian

2.2 Importing demo project

Demo project provided with this document may be imported into e² studio using following steps in this section.

1. Select **File → Import**.

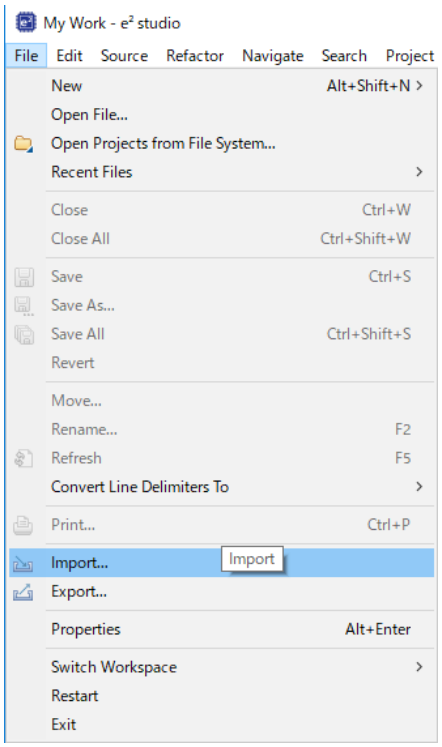

Figure 9. File menu

2. Select **Existing Projects into Workspace** and click **Next** button.

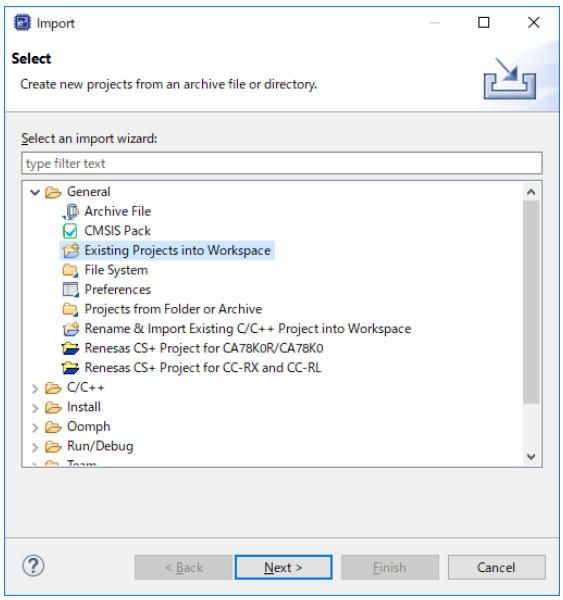

Figure 10. Select an import wizard

3. Select **Select archive file**, click **Browse…** button and select the demo project archive files. Click **Finish** button and the demo project is imported. Imported project include r01an5402.

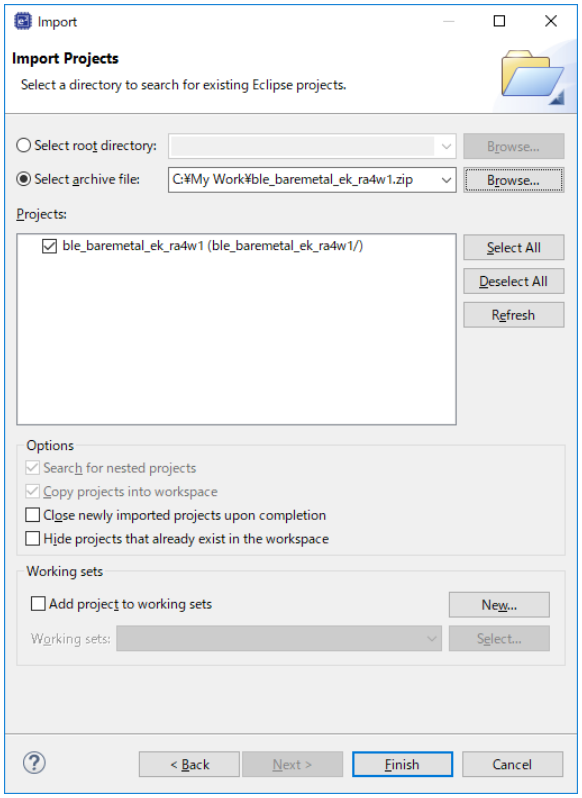

Figure 11. Import Project

4. Open FSP configuration by selecting **Project→Open FSP configuration.**

5. Press Generate Project Content button and then source code of related peripheral driver will automatically generate.

As a result of import, following file structure will appear in e² studio project explorer.

Table 6. File structure about demo project

Directory/File structure				Description
qe_gen	ble	discovery		Service discovery related APIs
		profile_cmn		Profile common APIs
		app_main.c		Main code
		gatt_db.c		GATT Database
		gatt_db.h		GATT Database
		r_ble_lss.c		Profile API
		r_ble_lss.h		Profile API
ra	fsp	Inc	api	BLE interface file r_ble_api.h rm_ble_abs_api.h
			instances	Abstraction API(GAP) rm_ble_abs.h
		lib	r_ble	BLE Protocol Stack (Extended type)
		src	rm_ble_abs	Abstraction API(GAP) rm_ble_abs.c
	aws	amazon-freertos		FreeRTOS kernel (Only FreeRTOS environment)
	microsoft	azure-rtos		Azure RTOS kernel (Only Azure RTOS environment)
ra_gen	---	---		RA configuration generated.
ra_cfg	fsp_cfg	r_ble_cfg.h		Configuration option file
		rm_ble_abs_cfg.h		Configuration option file
		azure	tx/tx_user.h	Azure RTOS configuration (Only Azure RTOS environment)
	aws	FreeRTOSConfig.h		FreeRTOS configuration (Only FreeRTOS environment)
src	---	hal_entry.c		User code entry point. (BareMetal)
		ble_core_task_entry.c		BLE task implementation (Only RTOS environment)
		lss_task.c		LED switch service task implementation (Only RTOS environment)
		task_function.h		LED switch service task header file (Only RTOS environment)
	app_lib	cli		CLI functionality provided by this demo project
		cmd		Commands of CLI provided by this project
		logger		Logger functionality provided by this demo project

2.3 Building and debugging

Refer to “e² studio Getting Started Guide (R20UT4204)”.

2.4 Demo project behavior

2.4.1 Preparation of demo

GATT Server demo projects can work by standalone. In case of making them work by standalone, refer to “EK-RA4W1 Quick Start Guide (R20QS0015)”. GATT Client demo projects and GATT Server demo projects with CLI can accept commands received via r_sci_uart. User can handle the communication between PC and EK-RA4W1 the same as COM ports by the terminal emulator like Tera Term because EK-RA4W1 board equips the USB-Serial converter IC. Setting of the terminal software for these demo projects is following table.

Table 7. Setting of the terminal software

New line (Receive)	LF
New line (Transmit)	CR
Terminal Mode	VT100
Baud rate	115200
Data bits	8bits
Parity	None
Stop bits	1bit
Flow control	None

2.4.2 GATT Server projects behavior

GATT Server demo projects provided with this document will work as mentioned next. Refer to “EK-RA4W1 Quick Start Guide (R20QS0015)” for the details of the EK-RA4W1 and the GATT Browser.

- When powered ON EK-RA4W1 or user’s custom board with the demo project programmed will start advertising.
- By scanning from remote device (e.g. smart phone with GATT browser), the remote device will detect EK-RA4W1 or user’s custom board as “TEST_RBLE” or “RBLE”.

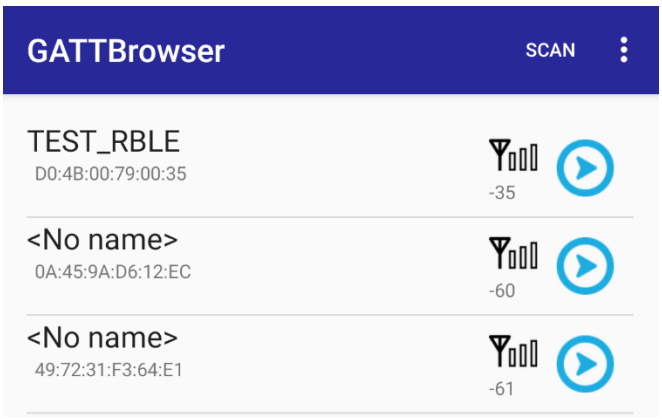

Figure 12. Scan result example

- When BLE connection is established between EK-RA4W1 or user’s custom board and the remote device, EK-RA4W1 or user’s custom board will stop advertising.

- The services and characteristics will be displayed after performing GATT service discovery from the remote device. This demo project includes following services.
  - LED Switch Service (UUID: 5883**1926**-5F05-4267-AB01-B4968E8EFCE0)
  - Switch State Characteristic (UUID: 5883**7F57**-5F05-4267-AB01-B4968E8EFCE0)
  - LED Blink Rate Characteristic (UUID: 5883**C32F**-5F05-4267-AB01-B4968E8EFCE0)

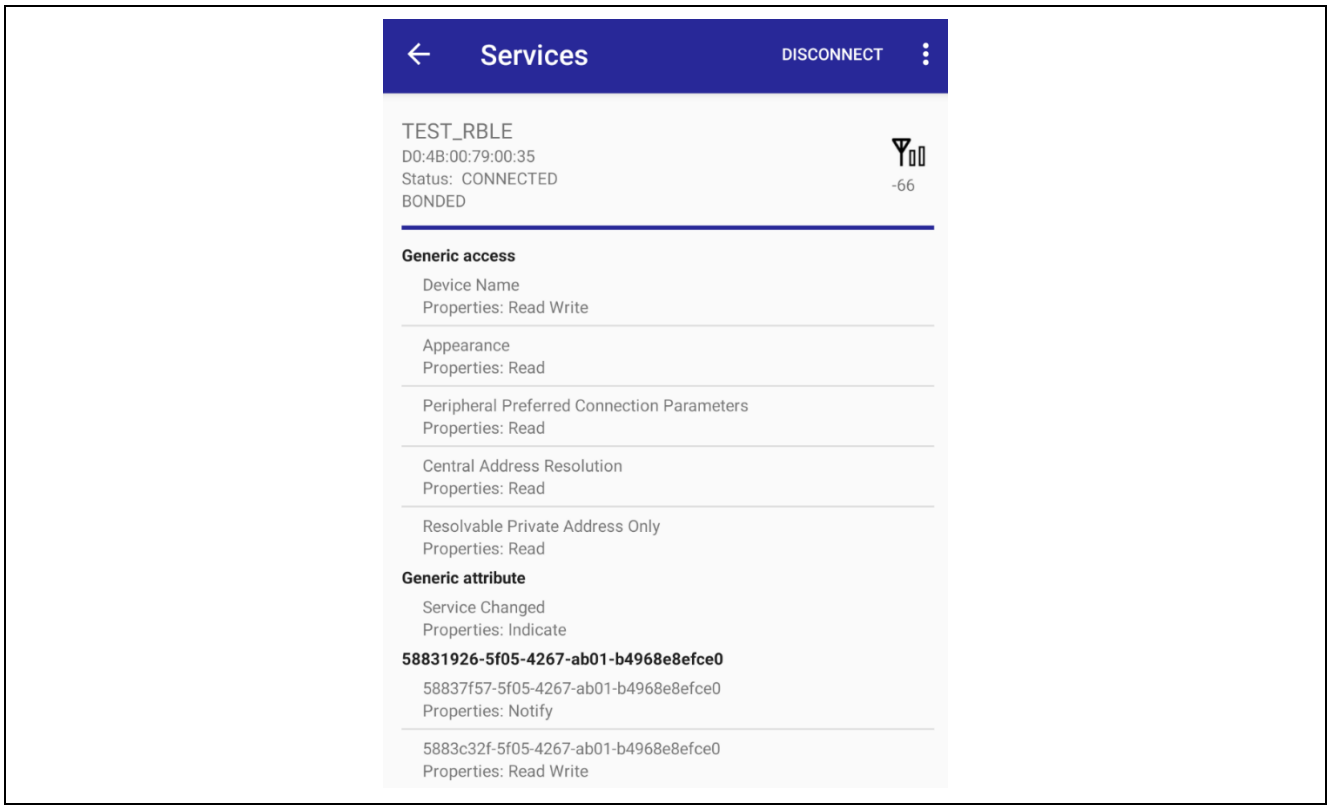

Figure 13. GATT Services

All included services are shown below. (WR : Write, RD : Read, IN : Indication, NT : Notification)

Table 8. GATT services

ATT Handle	ATT Type	Properties	ATT Values	Definition
GAP Service				
0x0001	0x28,0x00	RD	0x00,0x18	GAP Service Declaration
0x0002	0x28,0x03	RD	0x0A,0x03,0x00,0x00,0x2A	Device Name characteristic Declaration
0x0003	0x00,0x2A	RD,WR	0x00,0x00,0x00,0x00,0x00,0x00…	Device Name characteristic value
0x0004	0x28,0x03	RD	0x02,0x05,0x00,0x01,0x2A	Appearance characteristic Declaration
0x0005	0x01,0x2A	RD	0x00,0x00	Appearance characteristic value
0x0006	0x28,0x03	RD	0x02,0x07,0x00,0x04,0x2A	Peripheral Preferred Connection Parameters characteristic Declaration
0x0007	0x04,0x2A	RD	0x00,0x00,0x00,0x00,0x00,0x00…	Peripheral Preferred Connection Parameters characteristic value
0x0008	0x28,0x03	RD	0x02,0x09,0x00,0xA6,0x2A	Central Address Resolution characteristic Declaration
0x0009	0xA6,0x2A	RD	0x00	Central Address Resolution characteristic value
0x000A	0x28,0x03	RD	0x02,0x0B,0x00,0xC9,0x2A	Resolvable Private Address Only characteristic Declaration
0x000B	0xC9,0x2A	RD	0x00	Resolvable Private Address Only characteristic value
GATT Service				
0x000C	0x28,0x00	RD	0x01,0x18	GATT Service Declaration
0x000D	0x28,0x03	RD	0x20,0x0E,0x00,0x05,0x2A	Service Changed characteristic Declaration
0x000E	0x05,0x2A	IN	0x00,0x00,0x00,0x00	Service Changed characteristic value
0x000F	0x02,0x29	RD,WR	0x00,0x00	Client Characteristic Configuration descriptor
LED Switch Service(Custom Service)				
0x0010	0x28,0x00	RD	0xE0, 0xFC, 0x8E, 0x8E, 0x96, 0xB4, 0x01, 0xAB, 0x67, 0x42, 0x05, 0x5F, 0x26, 0x19, 0x83, 0x58	LED Switch Service(Custom Service) Declaration
0x0011	0x28,0x03	RD	0xE0, 0xFC, 0x8E, 0x8E, 0x96, 0xB4, 0x01, 0xAB, 0x67, 0x42, 0x05, 0x5F, 0x57, 0x7F, 0x83, 0x58	Switch State characteristic Declaration
0x0012	0xE0,0xFC,0x8E…	NT	0x00	Switch State characteristic value
0x0013	0x02,0x29	RD,WR	0x00,0x00	Client Characteristic Configuration descriptor
0x0014	0x28,0x03	RD	0xE0, 0xFC, 0x8E, 0x8E, 0x96, 0xB4, 0x01, 0xAB, 0x67, 0x42, 0x05, 0x5F, 0x2F, 0xC3, 0x83, 0x58	LED Blink Rate characteristic Declaration
0x0015	0xE0,0xFC,0x8E…	RD,WR	0x00	LED Blink Rate characteristic value

- If the LED Switch Service second parameter in the gs_gatt_service variable in the gatt_db.c is set to **BLE_GATT_DB_SER_SECURITY_UNAUTH**, the demo project will request pairing to access to the characteristic in the LED Switch Service.

```
static const st_ble_gatts_db_serv_cfg_t gs_gatt_service[] =
{
……
    /* LED Switch Service(Custom Service) */
    {
        /* Num of Services */
        {
            1,
        },
        /* Description */
        BLE_GATT_DB_SER_SECURITY_UNAUTH,
        /* Service Start Handle */
        0x0010,
……
};
```

Code 1. The security setting of the access to the LED Switch Service. (Necessary pairing case)

If the LED Switch Service second parameter in the gs_gatt_service variable in the gatt_db.c is set **0** is set, the demo project will not request pairing.

```
static const st_ble_gatts_db_serv_cfg_t gs_gatt_service[] =
{
……
    /* LED Switch Service(Custom Service) */
    {
        /* Num of Services */
        {
            1,
        },
        /* Description */
        0,
        /* Service Start Handle */
        0x0010,
……
};
```

Code 2. The security setting of the access to the LED Switch Service. (Not necessary pairing case)

- After enabling notification in the switch state characteristic, notification packet will send by pushing switch on EK-RA4W1 or user's custom board.
- LED on RA4W1 or user's custom board will blink according to numeric value which is written to the LED blink rate characteristic from the remote device. Note that the LED will turn off by writing 0x00 to the characteristic and remain on by writing 0xFF to the characteristic.
- When disconnected between EK-RA4W1 or user's custom board and the remote device, EK-RA4W1 or user's custom board will re-start advertising.

Figure 14 shows message sequence chart about behavior of demo projects accompanying this document.

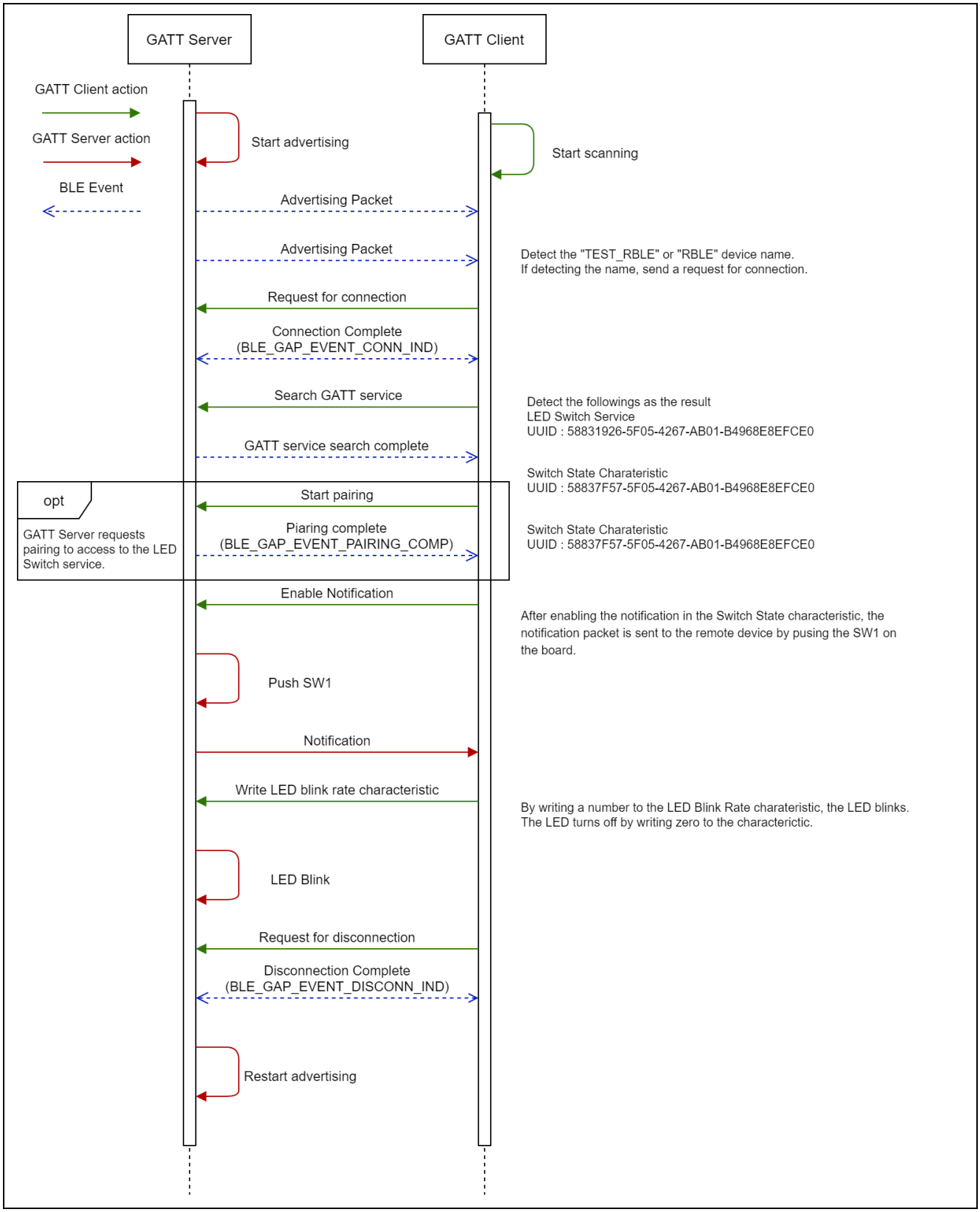

Figure 14. Usage example for demo projects

2.4.3 GATT Client demo projects behavior

GATT Client can perform various procedures in relation to GATT Client role by receiving commands via the terminal emulator. Some parts of following procedures can be also applied to GATT Server. Please refer to section 4.6. for details of each command.

(1) Scanning

GATT Client can start scanning procedure by following command. Then GATT Client can stop scanning procedure by Ctrl+C key input to terminal emulator.

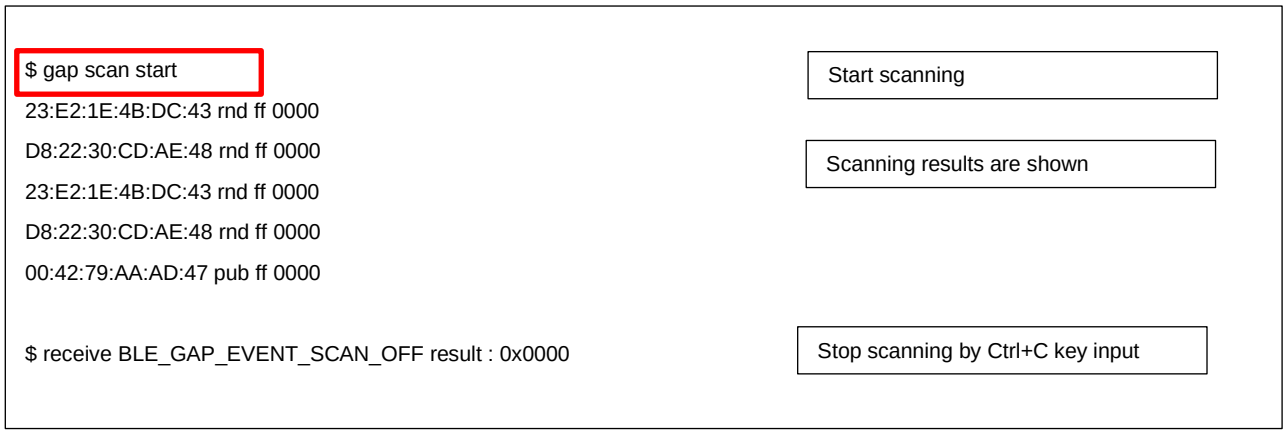

Figure 15. Scanning command and response

(2) Connection

GATT Client can try to create connection with GATT Server by following command.

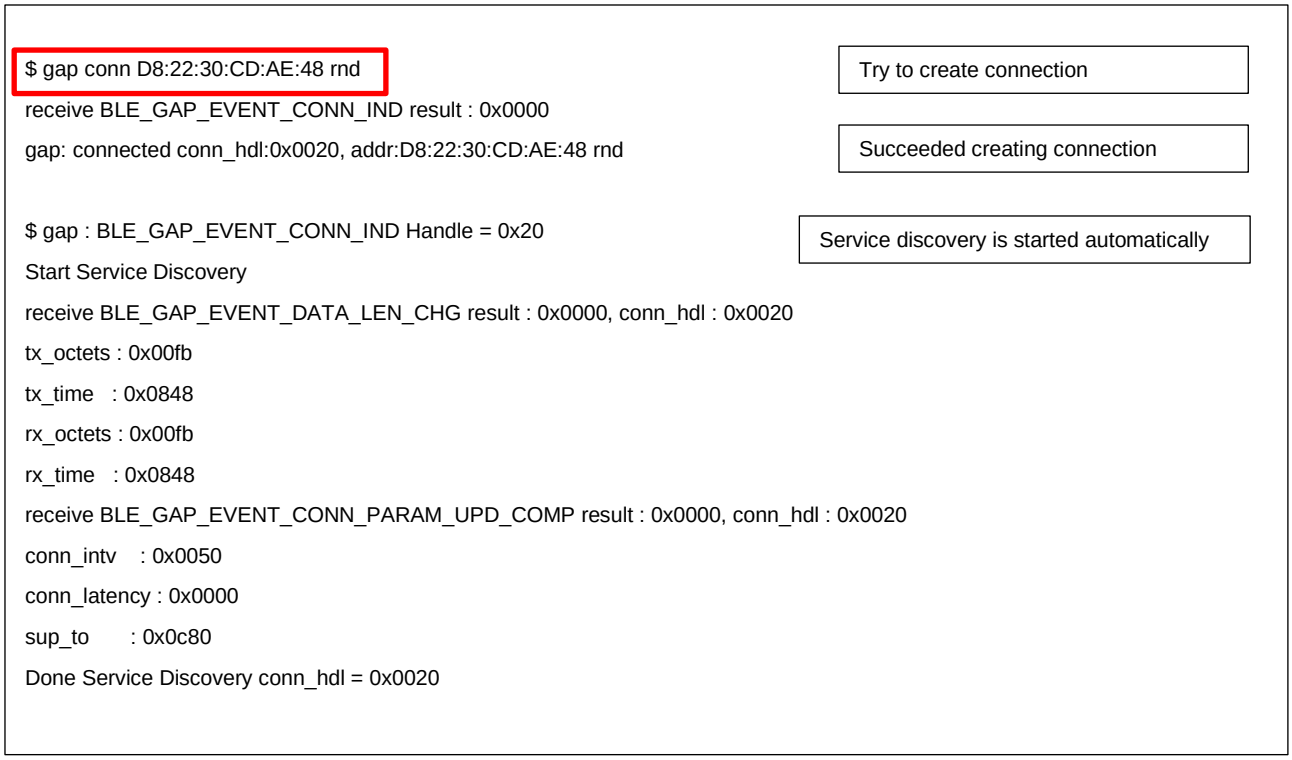

Figure 16. Creating connection command and response

(3) Paring (Option)

GATT Client and GATT Server can start paring procedure with the device which it is connecting with by following command.

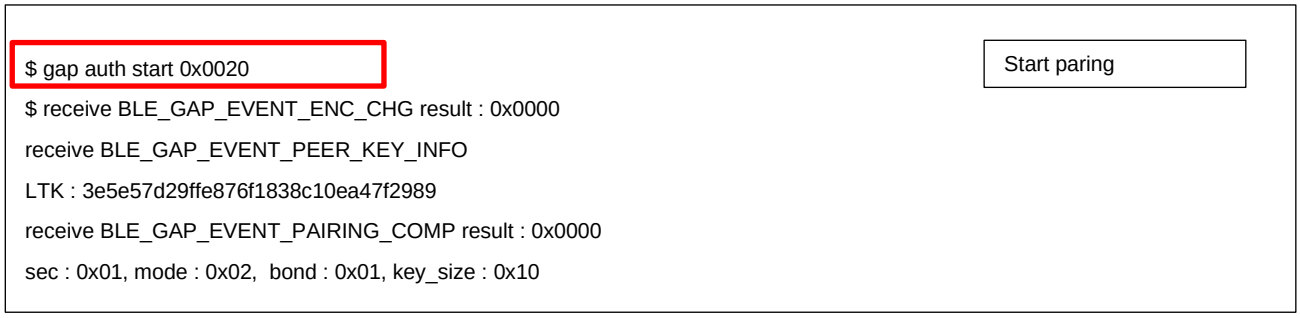

Figure 17. Paring command and response

(4) Disconnection

GATT Client and GATT Server can disconnect connection with the device which it is connecting with by following command.

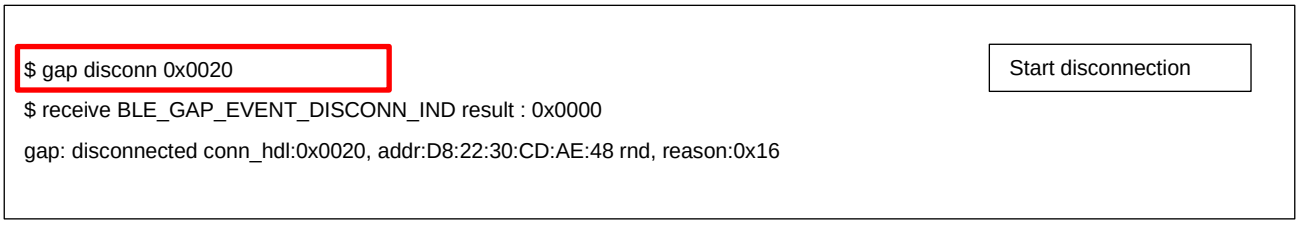

Figure 18. Disconnection command and response

(5) Entering Software Standby mode

GATT Client and GATT Server can enter Software Standby mode by receiving following command. Pressing SW1 on EK-RA4W1 board, when making EK-RA4W1 board exit from Software Standby mode.

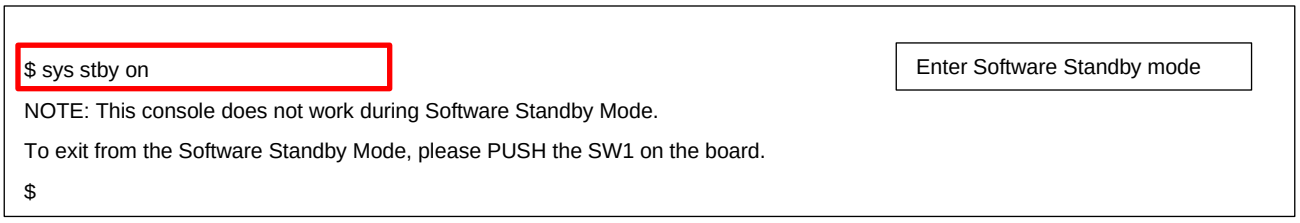

Figure 19. Standby command and response

(6) LED switch service

GATT Client enables receiving notifications from GATT Server by following command. Notifications can be sent by pressing SW on EK-RA4W1 board of GATT Server side. And GATT Client can write and read value LED blink rate of GATT Server by following commands. LED0 on EK-RA4W1 of GATT Server side blinks, turns on and off according to value written by GATT Client.

Figure 20. LED switch service commands and responses

3. Demo project implementation

This chapter describes demo project implementation.

3.1 BareMetal environment (Server)

BLE application implemented in app_main.c. The app_main.c includes initialization processing and implementation of the main loop.

Note: When using QE for BLE, the skeleton code of the app_main.c is automatically generated.

3.1.1 Entry point

Call *app_main()* in hal_entry.c as following.

```
void hal_entry(void) {
  /* TODO: add your own code here */
  app_main();
}
```

Code 3. Application entry point

3.1.2 Main loop

The *app_main()* includes initialization and main loop. Main loop of this demo project is following.

```
void app_main(void)
{
……
    /* Initialize Low Power Module */
    g_lpm0.p_api->open(g_lpm0.p_ctrl, g_lpm0.p_cfg);

    /* Initialize BLE and profiles */
    ble_init();

……
    R_BLE_CMD_SetResetCb((ble_event_cb_t)ble_init);
/* End user code. Do not edit comment generated here */

    /* main loop */
    while (1)
    {
……
        /* Process BLE Event */
        R_BLE_Execute();
……
/* Hint: Input process that should be done during main loop such as calling processing functions */
/* Start user code for process during main loop. Do not edit comment generated here */
……
        /* Disable IRQ */
        __disable_irq();

        /* UART reception on-going ? */
        if (false != get_uart_reception())
        {
            set_uart_reception(false);
            __enable_irq();
        }
        else
        {
            /* UART transmission on-going ? Allow enter software standby by sys stby command ? */
            if (true != g_inhibit_software_standby && true != get_uart_transmission() && true != g_led_blink_active)
            {
                /* Check whether there are executable BLE task or not */
                if (0 != R_BLE_IsTaskFree())
                {
                    /* There are no executable BLE task */
                    /* Terminate Command line */
                    R_BLE_CLI_Terminate();

                    /* Enter low power mode */
                    g_lpm0.p_api->lowPowerModeEnter(g_lpm0.p_ctrl);

                    /* Enable interrupt for processing interrupt handler after wake up */
                    __enable_irq();

                    /* Resume Command line */
                    R_BLE_CLI_Init();

                }
                else
                {
                    /* There is BLE related task */
                    __enable_irq();
                }
            }
            else
                __enable_irq();
        }

……
    /* Terminate BLE */
    RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
……
}
```

MCU low Power driver initialization.

BLE module initialization.

Main loop (Call *R_BLE_Execute*, Transition to MCU low power consumption state by *lowPowerModeEnter*)

Enter Software Standby mode

Code 4. app_main function

3.1.3 Initialization process

The *ble_init()* initializes the BLE module, and register callback function and GATT database. Initialization process of this demo project is following.

```
ble_status_t ble_init(void)
{
    ble_status_t status;
    fsp_err_t err;

    /* Initialize BLE */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
    if (FSP_SUCCESS != err)
    {
        return err;
    }

    /* Initialize GATT Database */
    status = R_BLE_GATTS_SetDbInst(&g_gatt_db_table);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT server */
    status = R_BLE_SERVS_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /*Initialize GATT client */
    status = R_BLE_SERVC_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Set Prepare Write Queue */
    R_BLE_GATTS_SetPrepareQueue(gs_queue, BLE_GATTS_QUEUE_NUM);

    /* Initialize LED Switch Service server API
    status = R_BLE_LSS_Init(lss_cb);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    return status;
}
```

BLE module initialization (*RM_ABS_BLE_Open*)

GATT database registration (*R_BLE_GATTS_SetDbInst*)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

GATT Server function initialization (*R_BLE_SERVS_Init*)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

GATT Client function initialization (*R_BLE_SERVC_Init*)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

Service initialization

Code 5. Initialization

Note: When using QE for BLE, the source code of the *ble_init()* function is automatically generated.

3.1.4 Register callback function

Registration of callback function is required to execute processing according to event from each layer of BLE protocol stack. Table 9 shows the callback registration API for each layer of BLE protocol stack.

Table 9. Callback function registration API

Function block	Callback registration API	Comment
GAP	*RM_BLE_ABS_Open* or *R_BLE_GAP_Init*	Registered callback function is called when receiving the result of *R_BLE_GAP_XXX* such as Advertising, Scan, Connection establishment and so on.
GATT Server (Profile common)	*RM_BLE_ABS_Open* or *R_BLE_GATTS_RegisterCb*	Registered callback function is called when accessed from GATT Client.
GATT Client (Profile common)	*RM_BLE_ABS_Open* or *R_BLE_GATTC_RegisterCb*	Registered callback function is called when accessed from GATT Server.
Service Discovery (Profile common)	*R_BLE_DISC_Start*	Registered callback function is called when completing Service Discovery.
Vendor Specific	*RM_BLE_ABS_Open* or *R_BLE_VS_Init*	Registered callback function is called when receiving the result of *R_BLE_VS_XXX*.
L2CAP	*R_BLE_L2CAP_RegisterCfPsm*	Registered callback function is called when receiving the result of *R_BLE_L2CAP_XXX* such as that the response of L2CAP Credit-Based Flow Control request is received. Note: Not code-generated by QE for BLE.
Server side profile API	*R_BLE_XXXS_Init* (XXX is Service name)	Registered callback function is called when accessed from Client.
Client side profile API	*R_BLE_XXXC_Init* (XXX is Service name)	Registered callback function is called when accessed from Server.

Note1: *RM_BLE_ABS_Open* can register GAP, GATT Server, GATT Client, and VS callback functions for each layer.

Note2: “XXX” included in name of callback registration API is “LSS” in demo project.

Note3: Callback registration API which is not used in demo project is also listed for reference.

3.1.5 Registering GATT database (R_BLE_GATTS_SetDbInst)

When creating a GATT service application, QE for BLE generates a service database code in the following file.

- gatt_db.c
- gatt_db.h

This GATT database is registered by *R_BLE_GATTS_SetDbInst()*.

3.1.6 Main loop and scheduler (R_BLE_Execute)

BLE protocol stack include a scheduler to process R_BLE API called by BLE application. Task to be performed are loaded to message queue in the scheduler when the API is called in BLE application. Call *R_BLE_Execute()* in main loop to start the scheduler and process the task in BLE protocol stack. Events that occur when a task is processed in BLE protocol stack are notified to callback function registered in section 3.1.4. Figure 21 shows typical sequence chart of BLE application.

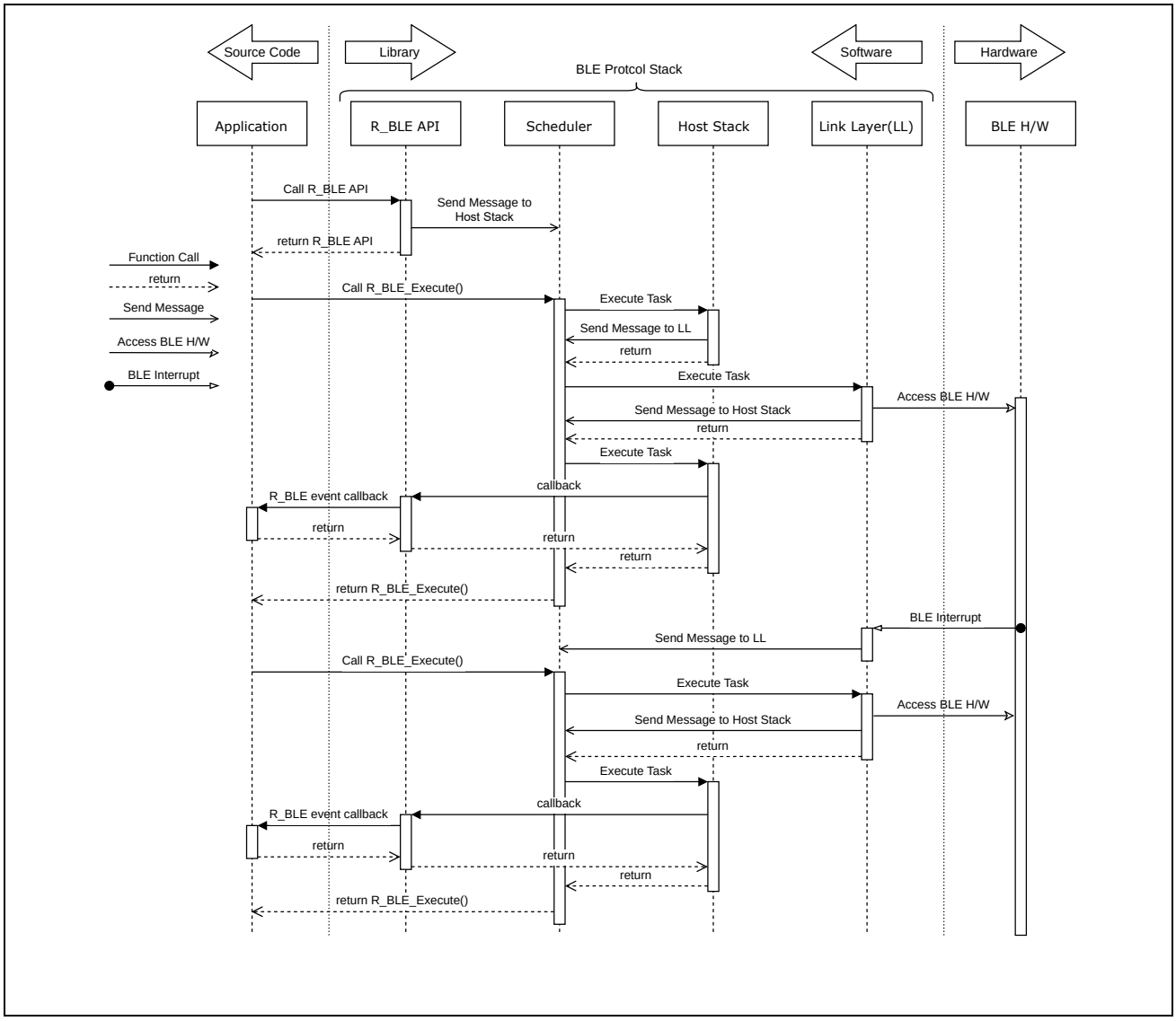

Figure 21. Basic sequence chart of BLE Protocol Stack

Relationship between callback function and event from each layer describe from the next section.

3.1.7 GAP event (gap_cb function)

GAP callback function receives following events.

```
enum   e_ble_gap_evt_t {
   BLE_GAP_EVENT_INVALID = 0x1001,
   BLE_GAP_EVENT_STACK_ON,
   BLE_GAP_EVENT_STACK_OFF,
   BLE_GAP_EVENT_LOC_VER_INFO,
   BLE_GAP_EVENT_HW_ERR,
   BLE_GAP_EVENT_CMD_ERR = 0x1101,
   BLE_GAP_EVENT_ADV_REPT_IND,
   BLE_GAP_EVENT_ADV_PARAM_SET_COMP,
   BLE_GAP_EVENT_ADV_DATA_UPD_COMP,
   BLE_GAP_EVENT_ADV_ON,
   BLE_GAP_EVENT_ADV_OFF,
   BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP,
   BLE_GAP_EVENT_PERD_ADV_ON,
   BLE_GAP_EVENT_PERD_ADV_OFF,
   BLE_GAP_EVENT_ADV_SET_REMOVE_COMP,
   BLE_GAP_EVENT_SCAN_ON,
   BLE_GAP_EVENT_SCAN_OFF,
   BLE_GAP_EVENT_SCAN_TO,
   BLE_GAP_EVENT_CREATE_CONN_COMP,
   BLE_GAP_EVENT_CONN_IND,
   BLE_GAP_EVENT_DISCONN_IND,
   BLE_GAP_EVENT_CONN_CANCEL_COMP,
   BLE_GAP_EVENT_WHITE_LIST_CONF_COMP,
   BLE_GAP_EVENT_RAND_ADDR_SET_COMP,
   BLE_GAP_EVENT_CH_MAP_RD_COMP,
   BLE_GAP_EVENT_CH_MAP_SET_COMP,
   BLE_GAP_EVENT_RSSI_RD_COMP,
   BLE_GAP_EVENT_GET_REM_DEV_INFO,
   BLE_GAP_EVENT_CONN_PARAM_UPD_COMP,
   BLE_GAP_EVENT_CONN_PARAM_UPD_REQ,
   BLE_GAP_EVENT_AUTH_PL_TO_EXPIRED,
   BLE_GAP_EVENT_SET_DATA_LEN_COMP,
   BLE_GAP_EVENT_DATA_LEN_CHG,
   BLE_GAP_EVENT_RSLV_LIST_CONF_COMP,
   BLE_GAP_EVENT_RPA_EN_COMP,
   BLE_GAP_EVENT_SET_RPA_TO_COMP,
   BLE_GAP_EVENT_RD_RPA_COMP,
   BLE_GAP_EVENT_PHY_UPD,
   BLE_GAP_EVENT_PHY_SET_COMP,
   BLE_GAP_EVENT_DEF_PHY_SET_COMP,
   BLE_GAP_EVENT_PHY_RD_COMP,
   BLE_GAP_EVENT_SCAN_REQ_RECV,
   BLE_GAP_EVENT_CREATE_SYNC_COMP,
   BLE_GAP_EVENT_SYNC_EST,
   BLE_GAP_EVENT_SYNC_TERM,
   BLE_GAP_EVENT_SYNC_LOST,
   BLE_GAP_EVENT_SYNC_CREATE_CANCEL_COMP,
   BLE_GAP_EVENT_PERD_LIST_CONF_COMP,
   BLE_GAP_EVENT_PRIV_MODE_SET_COMP,
   BLE_GAP_EVENT_PAIRING_REQ = 0x1401,
   BLE_GAP_EVENT_PASSKEY_ENTRY_REQ,
   BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ,
   BLE_GAP_EVENT_NUM_COMP_REQ,
   BLE_GAP_EVENT_KEY_PRESS_NTF,
   BLE_GAP_EVENT_PAIRING_COMP,
   BLE_GAP_EVENT_ENC_CHG,
   BLE_GAP_EVENT_PEER_KEY_INFO,
   BLE_GAP_EVENT_EX_KEY_REQ,
   BLE_GAP_EVENT_LTK_REQ,
   BLE_GAP_EVENT_LTK_RSP_COMP,
   BLE_GAP_EVENT_SC_OOB_CREATE_COMP
}
```

Code 6. GAP event

Reception condition of the frequently occurring events are shown below.

Table 10. Frequently use event of GAP callback

Event	Reception condition
BLE_GAP_EVENT_STACK_ON(0x1001)	Complete *R_BLE_GAP_Init*
BLE_GAP_EVENT_ADV_PARAM_SET_COMP(0x1003)	Complete *R_BLE_GAP_SetAdvParam*
BLE_GAP_EVENT_ADV_DATA_UPD_COMP (0x1004)	Complete *R_BLE_GAP_SetAdvSresData*
BLE_GAP_EVENT_ADV_ON (0x1005)	Start Advertising
BLE_GAP_EVENT_ADV_OFF (0x1006)	End Advertising
BLE_GAP_EVENT_ADV_REPT_IND (0x1102)	Received advertising report
BLE_GAP_EVENT_SCAN_ON (0x1111)	Start Scan
BLE_GAP_EVENT_SCAN_OFF (0x1112)	End Scan
BLE_GAP_EVENT_CONN_IND (0x1115)	Start Connection
BLE_GAP_EVENT_CONN_IND (0x1115)	End Connection
BLE_GAP_EVENT_DISCONN_IND (0x1116)	End Disconnection

GAP callback function in this demo project is following.

```
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GAP callback function common process. Do not edit comment generated here */

    R_BLE_CMD_AbsGapCb(type, result, p_data);

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

    switch(type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
            R_BLE_CLI_Printf("gap : BLE_GAP_EVENT_STACK_ON \n");

            /* Get BD address for Advertising */
            R_BLE_VS_GetBdAddr(BLE_VS_ADDR_AREA_REG, BLE_GAP_ADDR_RAND);
        } break;

        case BLE_GAP_EVENT_CONN_IND:
        {
……
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
……
        } break;

        case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
        {
……
        } break;

/* Hint: Add cases of GAP event macros defined as BLE_GAP_XXX */
/* Start user code for GAP callback function event process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
    }
}
```

In this demo project, many parts of processing when receiving events are also implemented in R_BLE_CMD_AbsGapCb().

Complete GAP initialization

Get BD address

Connection complete

Disconnection has happened

Connection parameter request come from client

Code 7. GAP callback function

QE for BLE generates skeleton code for GAP callback function. User can add their own code into the skeleton code.

3.1.8 GATTS event (gatts_cb function)

GATT server (GATTS) callback function receives following events.

```
enum   e_r_ble_gatts_evt_t {
   BLE_GATTS_EVENT_EX_MTU_REQ = 0x3002,
   BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP = 0x3009,
   BLE_GATTS_EVENT_READ_RSP_COMP = 0x300B,
   BLE_GATTS_EVENT_READ_BLOB_RSP_COMP = 0x300D,
   BLE_GATTS_EVENT_READ_MULTI_RSP_COMP = 0x300F,
   BLE_GATTS_EVENT_WRITE_RSP_COMP = 0x3013,
   BLE_GATTS_EVENT_PREPARE_WRITE_RSP_COMP = 0x3017,
   BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP = 0x3019,
   BLE_GATTS_EVENT_HDL_VAL_CNF = 0x301E,
   BLE_GATTS_EVENT_DB_ACCESS_IND = 0x3040,
   BLE_GATTS_EVENT_CONN_IND = 0x3081,
   BLE_GATTS_EVENT_DISCONN_IND = 0x3082,
   BLE_GATTS_EVENT_INVALID = 0x30FF
}
```

Code 8. GATTS event

Reception condition of frequently occurring events is shown below.

Table 11. Frequently use events of GATTS callback

Event	Reception condition
BLE_GATTS_EVENT_CONN_IND(0x3081)	Establish Connection
BLE_GATTS_EVENT_EX_MTU_REQ(0x3002)	Changing MTU is requested from GATT Client after Connection
BLE_GATTS_EVENT_DB_ACCESS_IND(0x3040)	Accessed to GATT database
BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP(0x3009)	Complete sending Read By Type Response
BLE_GATTS_EVENT_WRITE_RSP_COMP(0x3013)	Complete sending Write Response
BLE_GATTS_EVENT_HDL_VAL_CNF(0x301E)	Complete receiving Confirmation from GATT Client
BLE_GATTS_EVENT_DISCONN_IND(0x3082)	End Disconnection

GATTS callback function in this demo project is following.

```
void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Server callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVS_GattsCb(type, result, p_data);
    switch(type)
    {
/* Hint: Add cases of GATT Server event macros defined as BLE_GATTS_XXX */
/* Start user code for GATT Server callback function event process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
    }
}
```

Code 9. GATTS callback function

QE for BLE generates skeleton code for GATTS callback function. User can add their own code into the skeleton code.

3.1.9 GATTC event (gattc_cb function)

GATT client (GATTC) callback function receives following events.

```
enum   e_r_ble_gattc_evt_t {
   BLE_GATTC_EVENT_ERROR_RSP = 0x4001,
   BLE_GATTC_EVENT_EX_MTU_RSP = 0x4003,
   BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP = 0x4009,
   BLE_GATTC_EVENT_CHAR_READ_RSP = 0x400B,
   BLE_GATTC_EVENT_CHAR_PART_READ_RSP = 0x400D,
   BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP = 0x400F,
   BLE_GATTC_EVENT_CHAR_WRITE_RSP = 0x4013,
   BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP = 0x4017,
   BLE_GATTC_EVENT_HDL_VAL_NTF = 0x401B,
   BLE_GATTC_EVENT_HDL_VAL_IND = 0x401D,
   BLE_GATTC_EVENT_CONN_IND = 0x4081,
   BLE_GATTC_EVENT_DISCONN_IND = 0x4082,
   BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND = 0x40E0,
   BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND = 0x40E1,
   BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP = 0x40E2,
   BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP = 0x40E3,
   BLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND = 0x40E4,
   BLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND = 0x40E5,
   BLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP = 0x40E6,
   BLE_GATTC_EVENT_INC_SERV_16_DISC_IND = 0x40E7,
   BLE_GATTC_EVENT_INC_SERV_128_DISC_IND = 0x40E8,
   BLE_GATTC_EVENT_INC_SERV_DISC_COMP = 0x40E9,
   BLE_GATTC_EVENT_CHAR_16_DISC_IND = 0x40EA,
   BLE_GATTC_EVENT_CHAR_128_DISC_IND = 0x40EB,
   BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP = 0x40EC,
   BLE_GATTC_EVENT_CHAR_DISC_COMP = 0x40ED,
   BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND = 0x40EE,
   BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND = 0x40EF,
   BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP = 0x40F0,
   BLE_GATTC_EVENT_LONG_CHAR_READ_COMP = 0x40F1,
   BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP = 0x40F2,
   BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP = 0x40F3,
   BLE_GATTC_EVENT_RELIABLE_WRITES_COMP = 0x40F4,
   BLE_GATTC_EVENT_INVALID = 0x40FF
}
```

Code 10. GATTC event

Reception condition of frequently occurring events is shown below.

Table 12. Frequently use events of GATTC callback

Event	Reception condition
BLE_GATTC_EVENT_CONN_IND(0x4081)	Establish Connection
BLE_GATTC_EVENT_EX_MTU_RSP(0x4003)	Request Changing MTU to GATT Server after Connection and receive normal response
BLE_GATTC_EVENT_ERROR_RSP(0x4001)	Receive error response from GATT Server
BLE_GATTC_EVENT_HDL_VAL_NTF(0x401B)	Complete receiving Notification
BLE_GATTC_EVENT_HDL_VAL_IND(0x401D)	Complete receiving Indication
BLE_GATTC_EVENT_DISCONN_IND(0x4082)	End Disconnection

GATTC callback function is following.

```
void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Client callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVC_GattcCb(type, result, p_data);
    switch(type)
    {
/* Hint: Add cases of GATT Client event macros defined as BLE_GATTC_XXX */
/* Start user code for GATT Client callback function event process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
    }
}
```

Code 11. GATTC callback function

QE for BLE generates skeleton code for GATTC callback function. User can add their own code into the skeleton code.

3.1.10 VS event (vs_cb function)

Vender specific (VS) callback function receives following events.

```
enum   e_r_ble_vs_evt_t {
   BLE_VS_EVENT_SET_TX_POWER = 0x8001,
   BLE_VS_EVENT_GET_TX_POWER = 0x8002,
   BLE_VS_EVENT_TX_TEST_START = 0x8003,
   BLE_VS_EVENT_TX_TEST_TERM = 0x8004,
   BLE_VS_EVENT_RX_TEST_START = 0x8005,
   BLE_VS_EVENT_TEST_END = 0x8006,
   BLE_VS_EVENT_SET_CODING_SCHEME_COMP = 0x8007,
   BLE_VS_EVENT_RF_CONTROL_COMP = 0x8008,
   BLE_VS_EVENT_SET_ADDR_COMP = 0x8009,
   BLE_VS_EVENT_GET_ADDR_COMP = 0x800A,
   BLE_VS_EVENT_GET_RAND = 0x800B,
   BLE_VS_EVENT_TX_FLOW_STATE_CHG = 0x800C,
   BLE_VS_EVENT_FAIL_DETECT = 0x800D,
   BLE_VS_EVENT_SET_SCAN_CH_MAP = 0x800E,
   BLE_VS_EVENT_GET_SCAN_CH_MAP = 0x800F,
   BLE_VS_EVENT_INVALID = 0x80FF
}
```

Code 12. VS event

Reception condition of frequently occurring events are shown below.

Table 13. Frequently use events of VS callback

Event	Reception condition
BLE_VS_EVENT_SET_TX_POWER(0x8001)	Complete *R_BLE_VS_SetTxPower*
BLE_VS_EVENT_GET_TX_POWER(0x8002)	Complete *R_BLE_VS_GetTxPower*
BLE_VS_EVENT_SET_ADDR_COMP(0x8009)	Complete *R_BLE_VS_SetBdAddr*
BLE_VS_EVENT_GET_ADDR_COMP(0x800A)	Complete *R_BLE_VS_GetBdAddr*

VS callback function in this demo project is following.

```
void vs_cb(uint16_t type, ble_status_t result, st_ble_vs_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for vender specific callback function common process. Do not edit comment generated here */

    R_BLE_CMD_VsCb(type, result, p_data);

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

    R_BLE_SERVS_VsCb(type, result, p_data);
    switch(type)
    {
        case BLE_VS_EVENT_GET_ADDR_COMP:
        {
……
            RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl, &g_ble_advertising_parameter);
        } break;

/* Hint: Add cases of vender specific event macros defined as BLE_VS_XXX */
/* Start user code for vender specific callback function event process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
    }
}
```

In this demo project, many parts of processing when receiving events are also implemented in R_BLE_CMD_VsCb().

Get BD address event

Code 13. VS callback function

QE for BLE generates skeleton code for VS callback function. User can add their own code into the skeleton code.

3.1.11 Server-side Profile API event ([service_name]s_cb function)

Callback function of the server side Profile API receives following events.

```
enum   e_ble_servs_event_t {
   BLE_SERVS_WRITE_REQ = 0x00,
   BLE_SERVS_WRITE_CMD = 0x01,
   BLE_SERVS_WRITE_COMP = 0x02,
   BLE_SERVS_READ_REQ = 0x03,
   BLE_SERVS_HDL_VAL_CNF = 0x04
}

enum   e_ble_[service name]s_event_t {
   BLE_[service name]S_EVENT_[characteristic name]_WRITE_REQ = 0xXX00,
   BLE_[service name]S_EVENT_[characteristic name]_WRITE_CMD= 0xXX01,
   BLE_[service name]S_EVENT_[characteristic name]_WRITE_COMP = 0xXX02,
   BLE_[service name]S_EVENT_[characteristic name]_READ_REQ = 0xXX03,
   BLE_[service name]S_EVENT_[characteristic name]_HDL_VAL_CNF = 0xXX04,
   BLE_[service name]S_EVENT_[characteristic name]_[descriptor name]_WRITE_REQ = 0xYY00,
   BLE_[service name]S_EVENT_[characteristic name]_[descriptor name]_READ_REQ = 0xYY03,
   :
   :
}
```

Code 14. Server-side Profile API event

Note1: The 10th to 15th bits are serial numbers that distinguish attributes (characteristics and descriptors). XX and YY are 00, 04, 08, 10, ..., FC.

Note2: [service name] is “LS” in this demo project.

Reception condition of frequently occurring events are shown below.

Table 14. Frequently use events of Profile Server callback

Event	Reception condition
XXX_WRITE_REQ (0xXXX0)	Complete receiving Write Request
XXX_WRITE_CMD (0xXXX1)	Complete receiving Write Without Response
XXX_WRITE_COMP (0xXXX2)	Complete sending Write Response
XXX_READ_REQ (0xXXX3)	Complete receiving Read Request
XXX_HDL_VAL_CNF (0xXXX4)	Complete receiving Confirmation

Note3: ”XXX” is “LSS” in this demo project.

Callback function of server side profile API in this demo project is following.

```
static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
    switch(type)
    {
……
        case BLE_LSS_EVENT_BLINK_RATE_WRITE_REQ:
        {
            g_current_blinky_interval = *(uint8_t *)p_data->p_param;

            if (g_current_blinky_interval == 0x00)
            {
                /* LED OFF */
                g_ioport.p_api->pinWrite(g_ioport.p_ctrl, BSP_IO_PORT_01_PIN_06, BSP_IO_LEVEL_HIGH);
                g_led_blink_active = false;
            }
            else if (g_current_blinky_interval == 0xFF)
            {
                /* LED ON */
                g_ioport.p_api->pinWrite(g_ioport.p_ctrl, BSP_IO_PORT_01_PIN_06, BSP_IO_LEVEL_LOW);
                g_led_blink_active = false;
            }
            else
            {
                g_led_blink_active = true;
                g_interval_update_flag = true;
            }
        } break;

        default:
            break;
/* End user code. Do not edit comment generated here */    }
}
```

Change LED blink rate

Code 15. Profile Server callback function

QE for BLE generates skeleton code for Profile Server callback function. User can add their own code into the skeleton code.

3.1.12 L2CAP event

L2CAP callback function receives following events.

```
enum   e_r_ble_l2cap_cf_evt_t {
   BLE_L2CAP_EVENT_CF_CONN_CNF = 0x5001,
   BLE_L2CAP_EVENT_CF_CONN_IND = 0x5002,
   BLE_L2CAP_EVENT_CF_DISCONN_CNF = 0x5003,
   BLE_L2CAP_EVENT_CF_DISCONN_IND = 0x5004,
   BLE_L2CAP_EVENT_CF_RX_DATA_IND = 0x5005,
   BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND = 0x5006,
   BLE_L2CAP_EVENT_CF_TX_CRD_IND = 0x5007,
   BLE_L2CAP_EVENT_CF_TX_DATA_CNF = 0x5008,
   BLE_L2CAP_EVENT_CMD_REJ = 0x5009
}
```

Code 16. L2CAP event

L2CAP callback function is following.

```
static void l2cap_cb(uint16_t type, ble_status_t result, st_ble_l2cap_cf_evt_data_t *p_data)
{
    switch (type)
    {
    Note: Add processing when an event is received here.
     }
}
```

Code 17. L2CAP callback function

QE for BLE does not generate skeleton code for L2CAP callback function. Users have to define / implement L2CAP callback function and register it by using *R_BLE_L2CAP_RegisterCfPsm()* at *app_main()* when user needs to use l2cap function.

3.1.13 Event notification and exiting from Software Standby mode

Event notification can be added to scheduler in BLE application by using *R_BLE_SetEvent()* API. If an event has occurred, the corresponding callback function will execute at the next call of the *R_BLE_Execute()*. The *R_BLE_Set_Event()* API should be used address the following cases.

- To perform time-consuming application processing within an interrupt service routine.
- To control program flow of a function which cannot be executed from an interrupt service routine.

Event notification use case in this document is following.

```
static void sw_cb(void)
{
    uint8_t state = 1;
    R_BLE_LSS_NotifySwitchState(g_conn_hdl, &state);
    g_inhibit_software_standby = true;
}


void Callback_ble_sw_irq(external_irq_callback_args_t *p_args)
{
    FSP_PARAMETER_NOT_USED(p_args);
    R_BLE_SetEvent(sw_cb);

}
```

Send notification

Clear flag to exit Software Standby mode

Interrupt callback function for push switch

Register sw_cb function as callback function

Code 18. Event notification

In above use case, BLE module will send notification to remote device when the user operates tactile switch which connected with RA4W1 GPIO.

In this demo project, IRQ4 assigned SW1 on EK-RA4W1 is designated as Wakeup Source of Low Power Module. When SW1 on EK-RA4W1 under Software Standby mode is pressed, Software Standby mode is exited then *Callback_ble_sw_irq()* function is executed because it is registered as callback function of IRQ4 interrupt.

3.1.14 CLI (Command Line Interface)

This demo project provides the functionality of CLI (Command Line Interface). CLI can be access with the terminal emulator like Tera Term on PC connecting EK-RA4W1 board via USB cable. Each command of CLI is registered to gsp_cmds structure in app_main.c like following. User defined commands can be added to gsp_cmds structure by the same scheme depending on the necessity. Refer to section 4.6.6 when user wants to create new command.

```
/* CommandLine parameters */
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
     &g_abs_cmd,
     &g_vs_cmd,
     &g_sys_cmd,
     &g_ble_cmd
};
```

Code 19. gsp_cmds

CLI is initialized by the following procedure in *app_main()* function in app_main.c.

```
void app_main(void)
{
……
    /* Initialize BLE and profiles */
    ble_init();

/* Hint: Input process that should be done before main loop such as calling initial function or
   variable definitions */
/* Start user code for process before main loop. Do not edit comment generated here */
……
    /* Configure CommandLine */
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, sizeof(gsp_cmds)/sizeof(gsp_cmds[0]));
    R_BLE_CLI_RegisterEventCb(NULL);
    R_BLE_CMD_SetResetCb((ble_event_cb_t)ble_init);
/* End user code. Do not edit comment generated here */

     /* main loop */
    while (1)
    {……
```

Code 20. Initialization of CLI

The processing of CLI is executed by *R_BLE_CLI_Process()* API in main loop in app_main.c.

```
    /* main loop */
    while (1)
    {
        /* Process BLE Event */
        R_BLE_Execute();
……
/* Hint: Input process that should be done during main loop such as calling processing functions */
/* Start user code for process during main loop. Do not edit comment generated here */


        /* Process Command Line */
        R_BLE_CLI_Process();
……
/* End user code. Do not edit comment generated here */
```

Code 21. Executing the processing of CLI

The processing of the event occurred as a result of calling *R_BLE_CLI_Process()* API is described as shown in the top part of Code 7 (The description of calling *R_BLE_CMD_***** API).

3.2 FreeRTOS environment (Server, EventGroup as Synchronization Type case)

In case of selected *Event Group* as *Synchronization Type* property of *BLE Driver* FSP module, BLE application is divided it into two or more tasks, BLE Core Task and GATT application tasks. BLE Core Task performs initialization and BLE related processing except GATT related event processing. The BLE Core task should be highest priority. In this demo project, BLE Core Task implemented in app_main.c and GATT application task (LED switch service) implemented in lss_task.c. This section describes BLE related task creation, task switching between BLE related task and implementation each task in following sections.

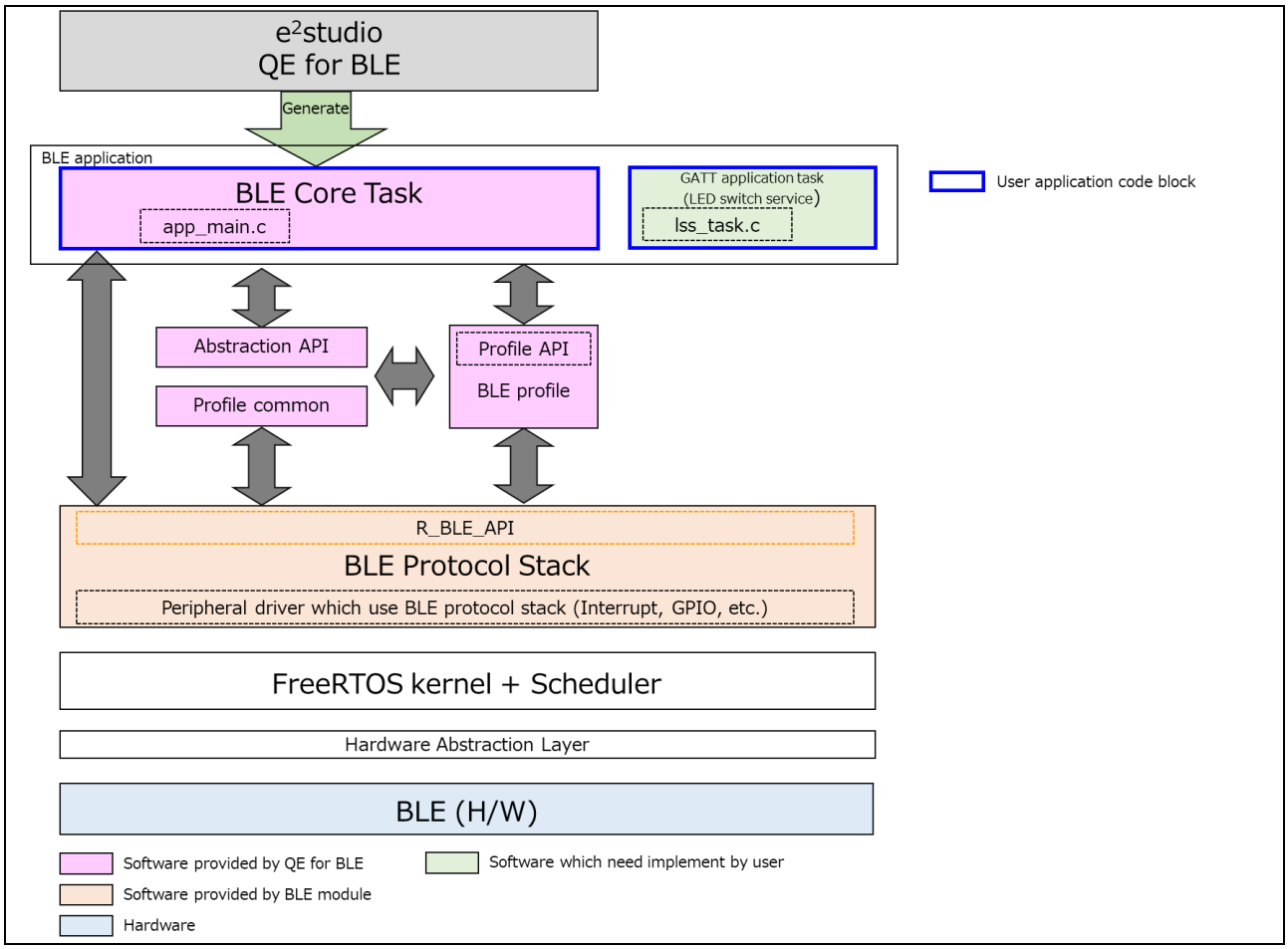

Figure 22. Software structure (FreeRTOS, EventGroup as Synchronization Type case)

Note1: When using QE for BLE, source code of the app_main function is automatically generated.

Note2: QE for BLE does not generate source code of the lss_task. User needs to define and the functionality for the lss_task.c. The user may this document and sample code for reference.

3.2.1 Create / delete task

- **Include ble_core_task.h**

Add the description of including “ble_core_task.h” as following to app_main.c.

```
/**********************************************************************************
 User file includes
**********************************************************************************/
/* Start user code for file includes. Do not edit comment generated here */
#include "ble_core_task.h"
/* End user code. Do not edit comment generated here */
```

Code 22. app_main.c

- **BLE Core task**

Initialization and main loop of BLE core task included in *app_main()*. Call the *app_main()* in ble_core_task_entry.c as following.

```
void ble_core_task_entry(void *pvParameters)
{
    FSP_PARAMETER_NOT_USED (pvParameters);

    /* TODO: add your own code here */
    app_main();

    while (1)
    {
        vTaskDelay (1000 / portTICK_PERIOD_MS);
    }
}
```

Code 23. app_main entry point

- **GATT application task**

GATT server event processing of GATT application task included in *lss_task_entry()*. The GATT application task is created when remote device connects to the RA4W1. And the task is deleted when the remote device disconnects from the RA4W1. This task creation/deletion is performed by GATT server callback function (gatts_cb) in app_main.c.

```
void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Server callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVS_GattsCb(type, result, p_data);
    switch(type)
    {
/* Hint: Add cases of GATT Server event macros defined as BLE_GATTS_XXX */
/* Start user code for GATT Server callback function event process. Do not edit comment generated here */

        case BLE_GATTS_EVENT_CONN_IND:                          Create GATT application task.
        {
            ……
            /* task create */
            xTaskCreate(lss_task_entry, "lss_task", 128, &g_conn_hdl, 4, &g_lss_task);

        }break;

        case BLE_GATTS_EVENT_DISCONN_IND:                       Delete GATT application task.
        {
      ……
            /* Delete Task */
            delete_lss_task_rsrc();
        }break;

        default:
            /* Do Nothing */
            break;
/* End user code. Do not edit comment generated here */
    }
}
```

Code 24. LED switch service task creation

3.2.2 Task switching between BLE core task and GATT application task

If event notified by scheduler, part of the BLE protocol stack, is an event for GATT application task, BLE core task activates GATT application task and provides a notification of the event by using event group setting and cleaning technique. In this demo project, event group bit defined in task_functon.h as following.

Table 15. Defined event group bit

Macro name (Value)	Usage
LSS_WAIT_EN_CCCD (0x0001)	Enable CCCD in LED Switch Service
LSS_WAIT_DIS_CCCD (0x0002)	Disable CCCD in LED Switch Service
LSS_WAIT_PUSH_SW (0x0004)	Notify push switch
LSS_WAIT_WR_BLINK (0x0008)	Change LED blink rate

Function which sets event group bit need to implement by the user. The function in this demo project is as follows.

```
void set_lss_event(EventBits_t uxBitsToSet)
{
    R_BLE_LSS_GetSwitchStateCliCnfg(gs_conn_hdl, &cccd);

    switch(uxBitsToSet)
    {
        case LSS_WAIT_EN_CCCD:                                          Enable CCCD in LED Switch Service
            xEventGroupClearBits(xLssEvent, (LSS_WAIT_DIS_CCCD | LSS_WAIT_PUSH_SW));
            xEventGroupSetBits(xLssEvent, uxBitsToSet);
            break;

        case LSS_WAIT_DIS_CCCD:                                         Disable CCCD in LED Switch Service
            uxBitsToSet = LSS_WAIT_DIS_CCCD;
            xEventGroupClearBits(xLssEvent, (LSS_WAIT_EN_CCCD | LSS_WAIT_PUSH_SW));
            xEventGroupSetBits(xLssEvent, uxBitsToSet);
            break;

        case LSS_WAIT_PUSH_SW:                                          Notify push switch
            if(BLE_GATTS_CLI_CNFG_NOTIFICATION == cccd)
            {
                xEventGroupSetBits(xLssEvent, uxBitsToSet);
            }
            break;

        case LSS_WAIT_WR_BLINK:                                         Change LED blink rate
            xEventGroupSetBits(xLssEvent, uxBitsToSet);
            break;
    }
}
```

Code 25. Set event group bit

3.2.3 Main loop of BLE core task

The *app_main()* includes initialization and main loop of BLE Core task. The program flow of this demo project is following.

```
void app_main(void)
{
……
    /* Create Event Group */
    g_ble_event_group_handle = (void *)xEventGroupCreate();
    assert(g_ble_event_group_handle);
……

    /* Initialize BLE and profiles */
    ble_init();

/* Hint: Input process that should be done before main loop such as calling initial function or variable
definitions */
/* Start user code for process before main loop. Do not edit comment generated here */
……
    R_BLE_CMD_SetResetCb((ble_event_cb_t)ble_init);
/* End user code. Do not edit comment generated here */

    /* main loop */
    while (1)
    {
……
        /* Process BLE Event */
        R_BLE_Execute();

        if(0 != R_BLE_IsTaskFree())
        {
            /* If the BLE Task has no operation to be processed, it transits block state until the event from
               RF transciever occurs. */
            xEventGroupWaitBits((EventGroupHandle_t)g_ble_event_group_handle,
                                (EventBits_t)BLE_EVENT_PATTERN,
                                pdTRUE,
                                pdFALSE,
                                portMAX_DELAY);
        }
……
    }

……
    /* Terminate BLE */
    RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
}
```

Create event group for transition task status.

BLE module initialization.

Main loop (Call *R_BLE_Execute*, Transition to Block state by `xEventGroupWaitBits`)

Code 26. app_main function

3.2.4 Main loop of GATT application task

The *lss_task_entry()* includes main loop of GATT application task. The program flow of this demo project is following.

```
void lss_task_entry(void *pvParameters)
{
    ble_status_t retval;
    uint8_t push_state;
    EventBits_t event;

    gs_conn_hdl = *(uint16_t *)pvParameters;                    Store connection handle
    xLssEvent = xEventGroupCreate();                            Create event group for transition task status.

    xBlinkTimerHdl = xTimerCreate("Blink" , 1000, pdTRUE , 0, blink_timer_cb);

    R_BLE_LSS_GetSwitchStateCliCnfg(gs_conn_hdl, &cccd);
    if(BLE_GATTS_CLI_CNFG_INDICATION != cccd)
    {
        cccd = BLE_GATTS_CLI_CNFG_DEFAULT;
    }

    wait_event = LSS_WAIT_EN_CCCD | LSS_WAIT_DIS_CCCD | LSS_WAIT_WR_BLINK | LSS_WAIT_PUSH_SW;

    while (1)
    {
        event = xEventGroupWaitBits(                            Wait for event from BLE core task
                                    xLssEvent,
                                    wait_event,
                                    pdTRUE,
                                    pdFALSE,
                                    portMAX_DELAY);

        if(LSS_WAIT_EN_CCCD & event)                            Wait for enable CCCD event from BLE core task
        {
……
        }
        else if(LSS_WAIT_DIS_CCCD & event)                      Wait for disable CCCD event from BLE core task
        {
……
        }
        else if((LSS_WAIT_PUSH_SW & event) && (BLE_GATTS_CLI_CNFG_NOTIFICATION == cccd))
        {                                                       Wait for push switch event from BLE core task
……
        }
        else if(LSS_WAIT_WR_BLINK & event)                      Wait for LED blink rate change event from BLE core task
        {
……
        }
    }
    vTaskDelete(NULL);
    /* End user code. Do not edit comment generated here */
}
```

Code 27. lss_task_entry function

3.2.5 Initialization process

Same as section 3.1.3.

3.2.6 Register callback function

Same as section 3.1.4.

3.2.7 Registering GATT database (R_BLE_GATTS_SetDbInst)

Same as section 3.1.5.

3.2.8 Main loop and scheduler (R_BLE_Execute)

The operation of the main loop and scheduler is similar to the description in section 3.1.6. The difference from BareMetal environment is that, if the event notified by scheduler which include BLE protocol stack is an event for GATT application task, BLE core task activates GATT application task and notify the event by using event group technique. Figure 23 shows the typical sequence chart of BLE module.

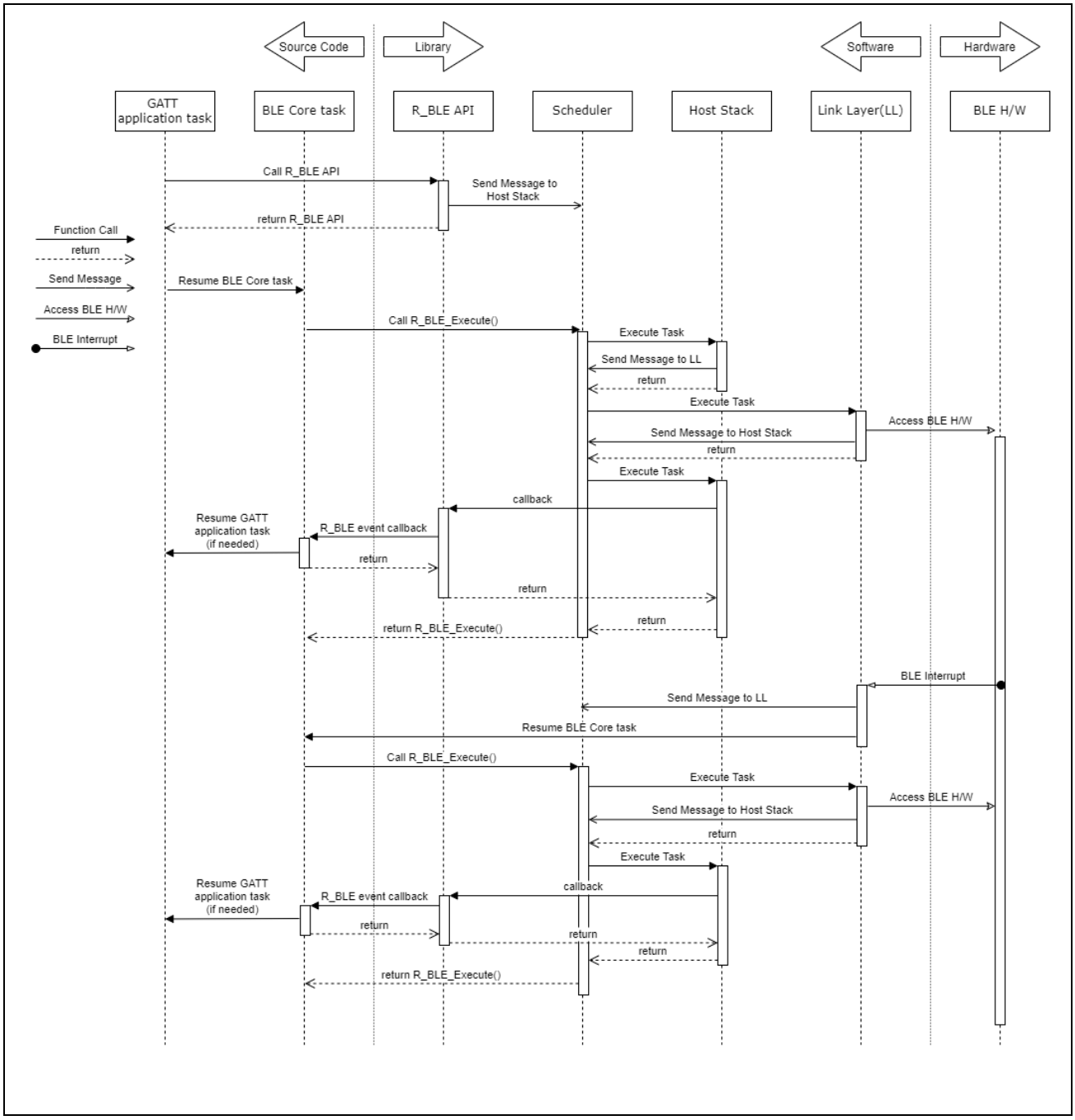

Figure 23. Basic sequence chart of BLE Protocol Stack

3.2.9 GAP event (gap_cb function)

Same as section 3.1.7.

3.2.10 GATTS event (gatts_cb function)

Almost the same as section 3.1.8. The difference from BareMetal environment is that,

- GATT application task is created when connection is established with the client.
- GATT application task is deleted when client disconnects.

Implementation of this demo project is following.

```
void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Server callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVS_GattsCb(type, result, p_data);
    switch(type)
    {
/* Hint: Add cases of GATT Server event macros defined as BLE_GATTS_XXX */
/* Start user code for GATT Server callback function event process. Do not edit comment generated here */

        case BLE_GATTS_EVENT_CONN_IND:
        {
.....                                                    Create GATT application task
            /* task create */
            /* LED Switch */
            xTaskCreate(lss_task_entry, "lss_task", 128, &g_conn_hdl, 4, &g_lss_task);
        }break;

        case BLE_GATTS_EVENT_DISCONN_IND:                Delete GATT application task
        {
            /* Delete GATT Application Task */
            delete_lss_task_rsrc();
        }break;

        default:
            /* Do Nothing */
            break;
/* End user code. Do not edit comment generated here */
    }
}
```

Code 28. GATTS callback function

3.2.11 GATTC event (gattc_cb function)

Almost the same as section 3.1.9. The difference from BareMetal environment is that,

- GATT Application task is created when connection is established with the server.
- GATT Application task is deleted when upon disconnecting from the server.

3.2.12 VS event (vs_cb function)

Same as section 3.1.10.

3.2.13 Server-side Profile API event ([service_name]s_cb function)

Almost the same as section 3.1.11. The difference from BareMetal environment is that event group bits are adjusted according to the data received from server-side profile API event. As a result, the GATT Application task is activated per the function definition provided in Section 3.2.2. Implementation of this demo project is following.

```
static void lss_cb(uint16_t type, ble_status_t result, st_ble_servs_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for LED Switch Service(Custom Service) Server callback function common process. */
    /* Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    uint16_t    data;

    switch(type)
    {
/* Hint: Add cases of LED Switch Service(Custom Service) server events defined in e_ble_lss_event_t */
/* Start user code for LED Switch Service(Custom Service) Server callback function event process. Do not edit
comment generated here */
        case BLE_LSS_EVENT_SWITCH_STATE_CLI_CNFG_WRITE_COMP :
        {
            R_BLE_LSS_GetSwitchStateCliCnfg(p_data->conn_hdl, &data);
            if (data)
                set_lss_event(LSS_WAIT_EN_CCCD);
            else
                set_lss_event(LSS_WAIT_DIS_CCCD);
        } break;

        case BLE_LSS_EVENT_BLINK_RATE_WRITE_COMP:
        {
            set_lss_event(LSS_WAIT_WR_BLINK);
        } break;

        default:
        {
            /* Do nothing. */
        } break;
/* End user code. Do not edit comment generated here */
    }
}
```

Change CCCD event.

Change LED blink rate

Code 29. Profile Server callback function

3.2.14 L2CAP event

Same as section 3.1.12.

3.2.15 Event notification

Event notification use case for FreeRTOS is following.

```
static void sw_cb(void)
{
    set_lss_event(LSS_WAIT_PUSH_SW);
}

void Callback_Sw_1(external_irq_callback_args_t *p_args)
{
    FSP_PARAMETER_NOT_USED(p_args);
    R_BLE_SetEvent(sw_cb);
}
```

Send notification

Interrupt callback function for push switch

Register sw_cb function as callback function

Code 30. Event notification

3.2.16 CLI (Command Line Interface)

Same as section 3.1.14.

3.3 FreeRTOS environment (Server, Semaphore case)

In case of selected *Semaphore* as *Synchronization Type* property of *BLE Driver* FSP module, BLE application is divided it into three or more tasks, BLE Core Task, Execute task and GATT application tasks. BLE Core Task performs initialization and BLE related processing except GATT related event processing. Execute task periodically calls *R_BLE_Execute* API. The execute task should be highest priority. In this demo project, BLE Core Task and Execute task implemented in app_main.c. GATT application task (LED switch service) implemented in lss_task.c. This section describes BLE related task creation, task switching between BLE related task and implementation each task in following sections.

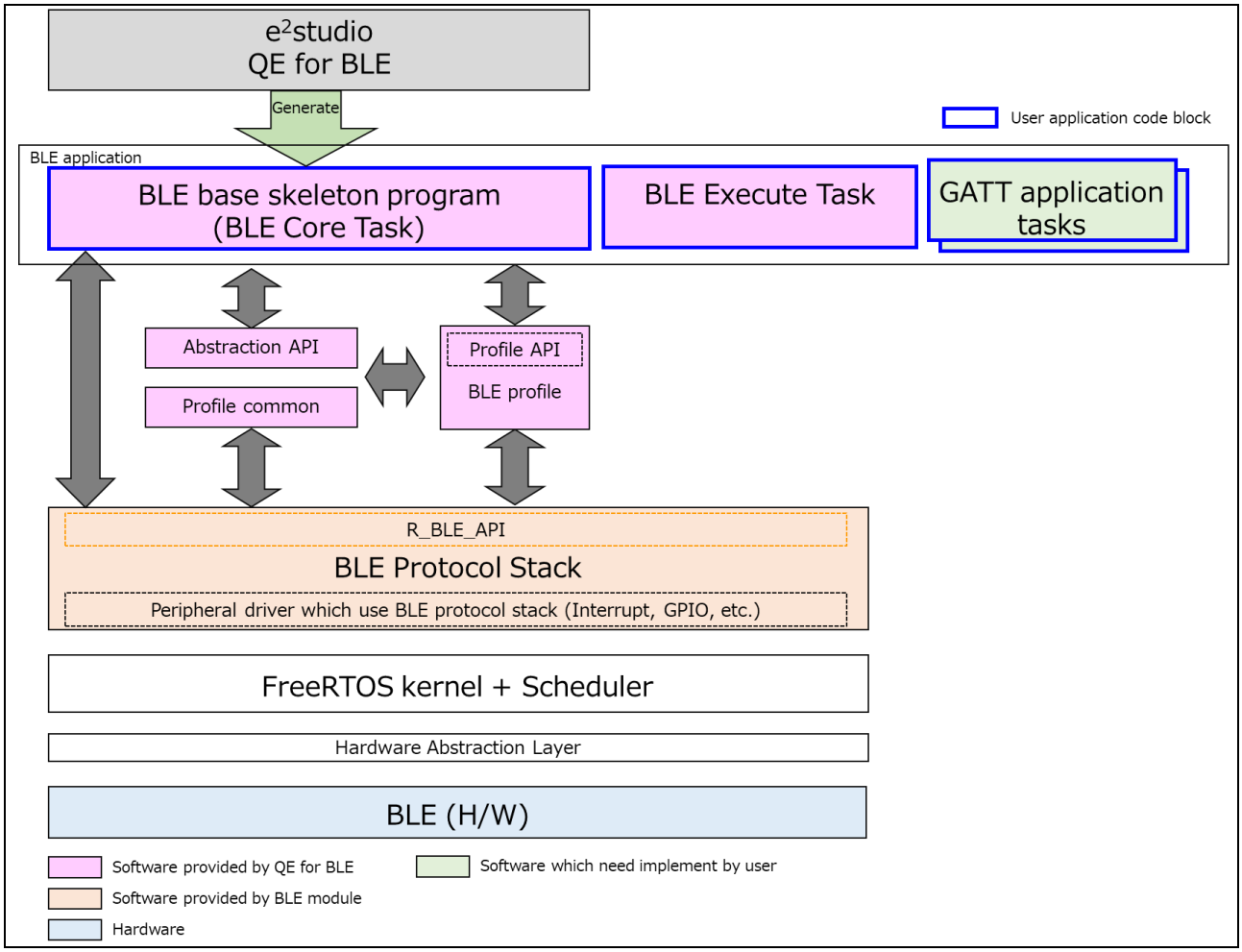

Figure 24. Software structure (FreeRTOS, Semaphore as Synchronization Type case)

Note1: When using QE for BLE, source code of the app_main function is automatically generated.

Note2: QE for BLE does not generate source code of the lss_task. User needs to define and the functionality for the lss_task.c. The user may this document and sample code for reference.

3.3.1 Create / delete task

- **Include ble_core_task.h**

Add the description of including “ble_core_task.h” as following to app_main.c.

```
User file includes
#include "ble_core_task.h"
/* End user code. Do not edit comment generated here */#
```

Code 31. app_main.c

- **BLE Core task**

Initialization and main loop of BLE core task included in *app_main()*. Call the *app_main()* in ble_core_task_entry.c as following.

```
void ble_core_task_entry(void *pvParameters)
{
    FSP_PARAMETER_NOT_USED (pvParameters);

    /* TODO: add your own code here */
    app_main();

    while (1)
    {
        vTaskDelay (1000 / portTICK_PERIOD_MS);
    }
}
```

Code 32. app_main entry point

- **Execute task**

Execute task will be created in app_main() as following. QE for BLE generated skeleton code includes the task creation and implementation.

```
void app_main(void)
{
……
    /* Get Current Task handle */
    gs_ble_core_task_ptr = xTaskGetCurrentTaskHandle();

    /* Create Execute Task */
    xTaskCreate(ble_execute_task_func, "execute_task", EXECUTE_STACK_SIZE, NULL, configMAX_PRIORITIES-1,
                &gs_ble_execute_task);
……
    While(1)
    {
        if(0 != R_BLE_IsTaskFree())
            vTaskSuspend(NULL);
        else
            xSemaphoreGive(gs_ble_exe_smpr);
    }
……
}
……
static void ble_execute_task_func(void *pvParameters)
{
   while(1)
   {
      xSemaphoreTake(gs_ble_exe_smpr, portMAX_DELAY);
      while(0 == R_BLE_IsTaskFree())
         R_BLE_Execute();

      vTaskResume(gs_ble_core_task_ptr);
   }
}
```

Code 33. Execute task creation and implementation

- **GATT application task**

Same as section 3.2.1.

3.3.2 Task switching between BLE core task and GATT application task

Same as section 3.2.2.

3.3.3 Main loop of BLE core task

The *app_main()* includes initialization and main loop of BLE Core task. The program flow of this demo project is following.

```
void app_main(void)
{
……
    gs_ble_exe_smpr = xSemaphoreCreateBinary();
    assert(gs_ble_exe_smpr);
    g_ble_event_group_handle = (void *)gs_ble_exe_smpr;
……
    /* Initialize BLE and profiles */
    ble_init();
……
    /* Get Current Task handle */
    gs_ble_core_task_ptr = xTaskGetCurrentTaskHandle();

    /* Create Execute Task */
    xTaskCreate(ble_execute_task_func, "execute_task", EXECUTE_STACK_SIZE, NULL, configMAX_PRIORITIES-1,
                &gs_ble_execute_task);
……
/* Hint: Input process that should be done before main loop such as calling initial function or variable
definitions */
/* Start user code for process before main loop. Do not edit comment generated here */
……
    R_BLE_CMD_SetResetCb((ble_event_cb_t)ble_init);
/* End user code. Do not edit comment generated here */

    /* main loop */
    while (1)
    {
……
        if(0 != R_BLE_IsTaskFree())
        {
            vTaskSuspend(NULL);
        }
        else
        {
            xSemaphoreGive(gs_ble_exe_smpr);
        }
……
    }

……
    /* Terminate BLE */
    RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
}
```

Create semaphore for transition task status.

BLE module initialization.

Create execute task.

Main loop (Give semaphore when scheduler queue does not empty)

Code 34. app_main function

3.3.4 Main loop of BLE execute task

Execute tasks by calling *R_BLE_Execute* API until running out all of them in queue once semaphore has been given.

```
static void ble_execute_task_func(void *pvParameters)
{
    while(1)
    {
        xSemaphoreTake(gs_ble_exe_smpr, portMAX_DELAY);
        while(0 == R_BLE_IsTaskFree())
            R_BLE_Execute();

        vTaskResume(gs_ble_core_task_ptr);
    }
}
```

Waiting for the semaphore forever until success to get it.

Call R_BLE_Execute API once semaphore has been given.

Resume BLE Core Task after completing to execute all BLE tasks.

Code 35. ble_execute_task_func function

3.3.5 Main loop of GATT application task

Same as section 3.2.4.

3.3.6 Initialization process

Same as section 3.1.3.

3.3.7 Register callback function

Same as section 3.1.4.

3.3.8 Registering GATT database (R_BLE_GATTS_SetDbInst)

Same as section 3.1.5.

3.3.9 Main loop and scheduler (R_BLE_Execute)

The operation of the BLE Core task, execute task and GATT application task are shown in Figure 25.

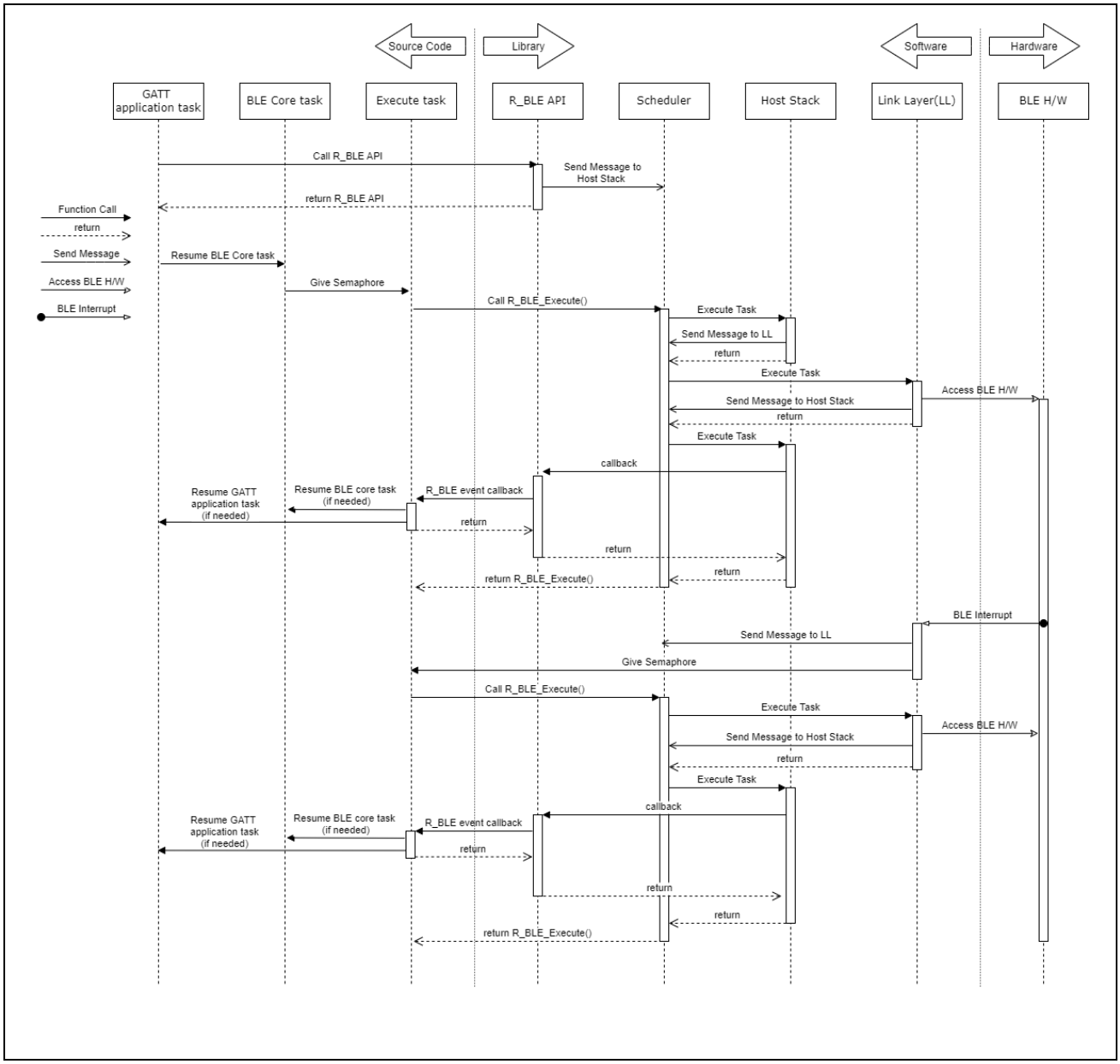

Figure 25. Basic sequence chart of BLE Protocol Stack

3.3.10 GAP event (gap_cb function)
Same as section 3.1.7.

3.3.11 GATTS event (gatts_cb function)
Same as section 3.2.10.

3.3.12 GATTC event (gattc_cb function)
Same as section 3.2.11.

3.3.13 VS event (vs_cb function)
Same as section 3.1.10.

3.3.14 Server-side Profile API event ([service_name]s_cb function)
Same as section 3.2.13.

3.3.15 L2CAP event
Same as section 3.1.12.

3.3.16 Event notification
Same as section 3.2.15.

3.3.17 CLI (Command Line Interface)
Same as section 3.1.14.

3.4 Azure RTOS environment (Server)

BLE application is divided it into three or more tasks, BLE Core Task, Execute task and GATT application tasks. BLE Core Task performs initialization and BLE related processing except GATT related event processing. Execute task periodically calls *R_BLE_Execute* API. The execute task should be highest priority. In this demo project, BLE Core Task and Execute task implemented in app_main.c. GATT application task (LED switch service) implemented in lss_task.c. This section describes BLE related task creation, task switching between BLE related task and implementation each task in following sections.

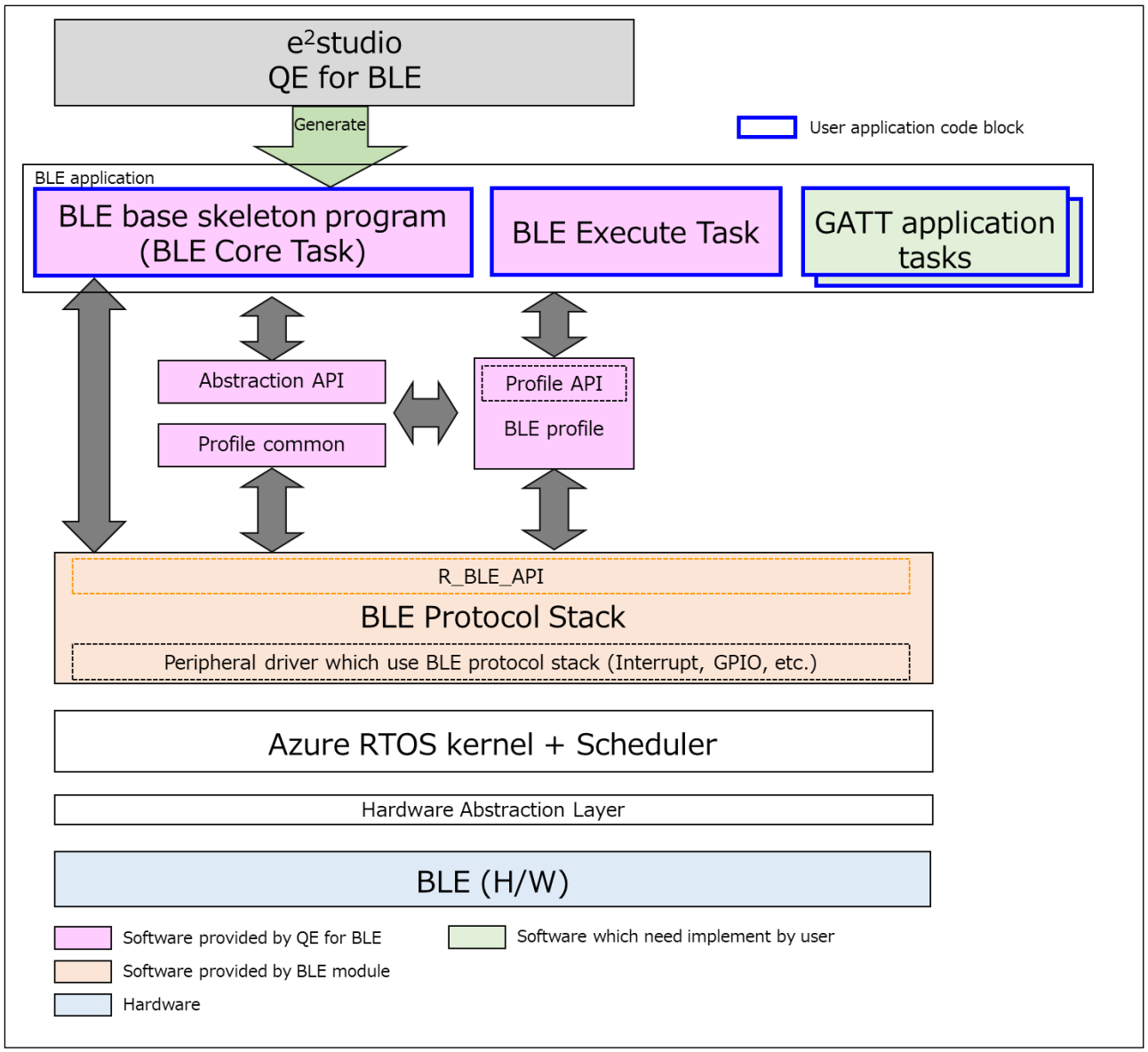

Figure 26. Software structure (Azure RTOS)

Note1: When using QE for BLE, source code of the app_main function is automatically generated.

Note2: QE for BLE does not generate source code of the lss_task. User needs to define and the functionality for the lss_task.c. The user may this document and sample code for reference.

3.4.1 Create / delete task

- **Include ble_core_task.h**

Add the description of including “ble_core_task.h” as following to app_main.c.

```
User file includes

/* Start user code for file includes. Do not edit comment generated here */
#include "ble_core_task.h"
/* End user code. Do not edit comment generated here */
```

Code 36. app_main.c

- **BLE Core task**

Initialization and main loop of BLE core task included in *app_main()*. Call the *app_main()* in ble_core_task_entry.c as following.

```
void ble_core_task_entry(void)
{
    /* TODO: add your own code here */
    app_main();
    while (1)
    {
        tx_thread_sleep (1);
    }
}
```

Code 37. app_main entry point

- **Execute task**

Execute task will be created in app_main() as following. QE for BLE generated skeleton code includes the task creation and implementation.

```
void app_main(void)
{
……
    /* Create Semaphore */
    tx_semaphore_create(&gs_ble_exe_smpr, "BLE_CORE_TASK_SEMAPHOR", TX_NO_INHERIT);

    /* Get Own thread handle */
    gs_ble_core_task_ptr = tx_thread_identify();

    /* Create BLE Execute Task */
    tx_thread_create(&gs_ble_execute_task, (CHAR*) "BLE_EXECUTE_TASK", ble_execute_task_func, (ULONG) NULL,
                           &gs_ble_execute_task_stack, EXECUTE_STACK_SIZE, 1, 1, TX_NO_TIME_SLICE,
                           TX_AUTO_START);
……
    While(1)
    {
        if(0 != R_BLE_IsTaskFree())
            tx_thread_suspend(gs_ble_core_task_ptr);
        else
            tx_semaphore_put(&gs_ble_exe_smpr);
    }
}
……
static void ble_execute_task_func(void *pvParameters)
{
   while(1)
   {
      tx_semaphore_get(&gs_ble_exe_smpr, TX_WAIT_FOREVER);
      while(0 == R_BLE_IsTaskFree())
         R_BLE_Execute();

      tx_thread_resume(gs_ble_core_task_ptr);
   }
}
```

Code 38. Execute task creation and implementation

- **GATT application task**

GATT server event processing of GATT application task included in *lss_task_entry()*. The task is created when remote device connects to the RA4W1. And the task is deleted when the remote device disconnects from the RA4W1. This task creation/deletion is performed by GATT server callback function (gatts_cb) in app_main.c.

```
void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Server callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVS_GattsCb(type, result, p_data);
    switch(type)
    {
/* Hint: Add cases of GATT Server event macros defined as BLE_GATTS_XXX */
/* Start user code for GATT Server callback function event process. Do not edit comment generated here */
        case BLE_GATTS_EVENT_CONN_IND:
        {
            /* Enable Notification SW */
            g_external_irq_sw1.p_api->enable(g_external_irq_sw1.p_ctrl);

            /* Create LED Switch Service Task */
            tx_thread_create(&ble_lss_task, (CHAR*) "BLE_LSS_TASK", lss_task_entry, g_conn_hdl,
                             &ble_lss_task_stack, 512, 4, 4, TX_AUTO_START);
        } break;

        case BLE_GATTS_EVENT_DISCONN_IND:
        {
            delete_lss_task_rsrc();

            /* Disable Notification SW */
            g_external_irq_sw1.p_api->disable(g_external_irq_sw1.p_ctrl);

            /* LED turn OFF */
            g_ioport.p_api->pinWrite(g_ioport.p_ctrl, BSP_IO_PORT_01_PIN_06, BSP_IO_LEVEL_HIGH);

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

Create GATT application task.

Delete GATT application task.

Code 39. LED switch service task creation

3.4.2 Task switching between BLE core task and GATT application task

If event notified by scheduler, part of the BLE protocol stack, is an event for GATT application task, BLE core task activates GATT application task and provides a notification of the event by using event flags setting and cleaning technique. In this demo project, event flag bit defined in task_functon.h as following.

Table 16. Defined event group bit

Macro name (Value)	Usage
LSS_WAIT_EN_CCCD (0x0001)	Enable CCCD in LED Switch Service
LSS_WAIT_DIS_CCCD (0x0002)	Disable CCCD in LED Switch Service
LSS_WAIT_PUSH_SW (0x0004)	Notify push switch
LSS_WAIT_WR_BLINK (0x0008)	Change LED blink rate

Function which sets event flag bit need to be implemented by the user. The function in this demo project is as following.

```
void set_lss_event(unsigned long uxBitsToSet)
{
    R_BLE_LSS_GetSwitchStateCliCnfg(gs_conn_hdl, &cccd);

    switch(uxBitsToSet)
    {
        case LSS_WAIT_EN_CCCD:
            tx_event_flags_set(&xLssEvent, uxBitsToSet, TX_OR);
            break;
        case LSS_WAIT_DIS_CCCD:
            uxBitsToSet = LSS_WAIT_DIS_CCCD;
            tx_event_flags_set(&xLssEvent, uxBitsToSet, TX_OR);
            break;
        case LSS_WAIT_PUSH_SW:
            if(BLE_GATTS_CLI_CNFG_NOTIFICATION == cccd)
            {
                tx_event_flags_set(&xLssEvent, uxBitsToSet, TX_OR);
            }
            break;
        case LSS_WAIT_WR_BLINK:
            tx_event_flags_set(&xLssEvent, uxBitsToSet, TX_OR);
            break;

        default:
            break;
    }
}
```

Code 40. Set event flags

.

3.4.3 Main loop of BLE core task

The *app_main()* includes initialization and main loop of BLE Core task. The program flow of this demo project is following.

```
void app_main(void)
{
……
    /* Initialize BLE and profiles */
    ble_init();

/* When this BLE application works on the Azure RTOS */
#if (BSP_CFG_RTOS == 1)

    /* Create Semaphore */
    tx_semaphore_create(&ble_exe_smpr, "BLE_CORE_TASK_SEMAPHOR", TX_NO_INHERIT);

    /* Get Own thread handle */
    ble_core_task_ptr = tx_thread_identify();

    /* Create BLE Execute Task */
    tx_thread_create(&ble_execute_task, (CHAR*) "BLE_EXECUTE_TASK", ble_execute_task_func, (ULONG) NULL,
                     &ble_execute_task_stack, EXECUTE_STACK_SIZE, 1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);
#endif

/* Hint: Input process that should be done before main loop such as calling initial function or variable
definitions */
/* Start user code for process before main loop. Do not edit comment generated here */
……
    R_BLE_CMD_SetResetCb((ble_event_cb_t)ble_init);

    /* Open external interrupt */
    g_external_irq_sw1.p_api->open(g_external_irq_sw1.p_ctrl, g_external_irq_sw1.p_cfg);
/* End user code. Do not edit comment generated here */

    /* main loop */
    while (1)
    {
……
#if (BSP_CFG_RTOS == 1)
        if(0 != R_BLE_IsTaskFree())
        {
            tx_thread_suspend(ble_core_task_ptr);
        }
        else
        {
            tx_semaphore_put(&ble_exe_smpr);
        }
……
#endif

/* Hint: Input process that should be done during main loop such as calling processing functions */
/* Start user code for process during main loop. Do not edit comment generated here */

        /* Process Command Line */
        R_BLE_CLI_Process();

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

/* Hint: Input process that should be done after main loop such as calling closing functions */
/* Start user code for process after main loop. Do not edit comment generated here */
    g_external_irq_sw1.p_api->close(g_external_irq_sw1.p_ctrl);
/* End user code. Do not edit comment generated here */

    /* Terminate BLE */
    RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
}
```

Create semaphore for BLE Execute Task

Create BLE Execute Task

When there are no BLE Task, suspend BLE Core Task

When there are BLE Tasks, put semaphore for BLE Execute Task

Code 41. app_main function

3.4.4 Main loop of BLE execute task

Execute tasks by calling *R_BLE_Execute* API until running out all of them in queue once semaphore has been given.

```
static void ble_execute_task_func(unsigned long Parameters)
{
    FSP_PARAMETER_NOT_USED(Parameters);
    while(1)
    {
        tx_semaphore_get(&ble_exe_smpr, TX_WAIT_FOREVER);
        while(0 == R_BLE_IsTaskFree())
            R_BLE_Execute();

        tx_thread_resume(ble_core_task_ptr);
    }
}
```

Waiting for the semaphore forever until success to get it.

Call R_BLE_Execute API once semaphore has been given.

Resume BLE Core Task after completing to execute all BLE tasks.

Code 42. ble_execute_task_func function

3.4.5 Main loop of GATT application task

The *lss_task_entry()* includes main loop of GATT application task. The program flow of this demo project is following.

```
void lss_task_entry(unsigned long Parameters)
{
    uint8_t push_state;
    unsigned long event;

    gs_conn_hdl = (uint16_t)Parameters;
    tx_event_flags_create(&xLssEvent, "LSS_EVENT_FLAG");

    /* Create Timer for LED blink */
    tx_timer_create(&xBlinkTimerHdl, "Blink", blink_timer_cb, NULL, 0x00000001, 0x00000001, TX_NO_ACTIVATE);

    R_BLE_LSS_GetSwitchStateCliCnfg(gs_conn_hdl, &cccd);
    if(BLE_GATTS_CLI_CNFG_INDICATION != cccd)
    {
        cccd = BLE_GATTS_CLI_CNFG_DEFAULT;
    }

    wait_event = LSS_WAIT_EN_CCCD | LSS_WAIT_DIS_CCCD | LSS_WAIT_WR_BLINK | LSS_WAIT_PUSH_SW;

    while (1)
    {
        tx_event_flags_get(&xLssEvent, wait_event, TX_OR_CLEAR, &event, TX_WAIT_FOREVER);

        if(LSS_WAIT_EN_CCCD & event)
        {
.....

        }
        else if(LSS_WAIT_DIS_CCCD & event)
        {
.....
        }
        else if((LSS_WAIT_PUSH_SW & event) && (BLE_GATTS_CLI_CNFG_NOTIFICATION == cccd))
        {
.....
        }
        else if(LSS_WAIT_WR_BLINK & event)
        {
.....
        }
    }
}
```

Store connection handle

Create event flags for transition task

Wait for event from BLE core task

Wait for enable CCCD event from BLE core task

Wait for disable CCCD event from BLE core task

Wait for push switch event from BLE core task

Wait for LED blink rate change event from BLE

Code 43. lss_task_entry function

3.4.6 Initialization process

Same as section 3.1.3.

3.4.7 Register callback function

Same as section 3.1.4.

3.4.8 Registering GATT database (R_BLE_GATTS_SetDbInst)

Same as section 3.1.5.

3.4.9 Main loop and scheduler (R_BLE_Execute)

Same as section 3.3.9.

3.4.10 GAP event (gap_cb function)

Same as section 3.1.7.

3.4.11 GATTS event (gatts_cb function)

Almost the same as section 3.1.8. The difference from BareMetal environment is that,

- GATT application task is created when connection is established with the client.
- GATT application task is deleted when client disconnects.

Implementation of this demo project is following.

```
void gatts_cb(uint16_t type, ble_status_t result, st_ble_gatts_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Server callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVS_GattsCb(type, result, p_data);
    switch(type)
    {
/* Hint: Add cases of GATT Server event macros defined as BLE_GATTS_XXX */
/* Start user code for GATT Server callback function event process. Do not edit comment generated here */
        case BLE_GATTS_EVENT_CONN_IND:
        {
            /* Enable Notification SW */
            g_external_irq_sw1.p_api->enable(g_external_irq_sw1.p_ctrl);

            /* Create LED Switch Service Task */
            tx_thread_create(&ble_lss_task, (CHAR*) "BLE_LSS_TASK", lss_task_entry, g_conn_hdl,
                             &ble_lss_task_stack, 512, 4, 4, 4, TX_AUTO_START);
        } break;

        case BLE_GATTS_EVENT_DISCONN_IND:
        {
            delete_lss_task_rsrc();

            /* Disable Notification SW */
            g_external_irq_sw1.p_api->disable(g_external_irq_sw1.p_ctrl);

            /* LED turn OFF */
            g_ioport.p_api->pinWrite(g_ioport.p_ctrl, BSP_IO_PORT_01_PIN_06, BSP_IO_LEVEL_HIGH);

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

Create GATT application task

Delete GATT application task

Code 44. GATTS callback function

3.4.12 GATTC event (gattc_cb function)

Same as section 3.2.11.

3.4.13 VS event (vs_cb function)

Same as section 3.1.10.

3.4.14 Server-side Profile API event ([service_name]s_cb function)

Same as section 3.2.13.

3.4.15 L2CAP event

Same as section 3.1.12.

3.4.16 Event notification

Same as section3.2.15..

3.4.17 CLI (Command Line Interface)

Same as section 3.1.14.

3.5 BareMetal environment (Client)

3.5.1 Entry point

Same as section 3.1.1.

3.5.2 Main loop

The *app_main()* includes initialization and main loop. Main loop of this demo project is following.

```
void app_main(void)
{
……
    /* Initialize Low Power Module */
    g_lpm0.p_api->open(g_lpm0.p_ctrl, g_lpm0.p_cfg);

    /* Initialize BLE and profiles */
    ble_init();
……
    R_BLE_CMD_SetResetCb((ble_event_cb_t)ble_init);

    g_ble_sw_irq.p_api->open(g_ble_sw_irq.p_ctrl, g_ble_sw_irq.p_cfg);
    g_ble_sw_irq.p_api->enable(g_ble_sw_irq.p_ctrl);
/* End user code. Do not edit comment generated here */

    /* main loop */
    while (1)
    {
        /* Process BLE Event */
        R_BLE_Execute();
……
        /* Disable IRQ */
        __disable_irq();

        /* UART reception on-going ? */
        if (false != get_uart_reception())
        {
            set_uart_reception(false);
            __enable_irq();
        }
        else
        {
            /* UART transmission on-going ? Allow enter software standby by sys stby command ? */
            if (true != g_inhibit_software_standby && true != get_uart_transmission())
            {
                /* Check whether there are executable BLE task or not */
                if (0 != R_BLE_IsTaskFree())
                {
                    /* There are no executable BLE task */
                    /* Terminate Command line */
                    R_BLE_CLI_Terminate();

                    /* Enter low power mode */
                    g_lpm0.p_api->lowPowerModeEnter(g_lpm0.p_ctrl);

                    /* Enable interrupt for processing interrupt handler after wake up */
                    __enable_irq();

                    /* Resume Command line */
                    R_BLE_CLI_Init();

                }
                else
                {
                    /* There is BLE related task */
                    __enable_irq();
                }
            }
            else
                __enable_irq();
        }
……
```

MCU low Power driver initialization.

BLE module initialization.

Main loop (Call *R_BLE_Execute*, Transition to MCU low power consumption state by *lowPowerModeEnter*)

Enter Software Standby mode

Code 45. app_main function

3.5.3 Initialization process

The *ble_init()* initializes the BLE module, and register callback function and GATT database. Initialization process of this demo project is following.

```
ble_status_t ble_init(void)
{
    ble_status_t status;
    fsp_err_t err;

    /* Initialize BLE */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
    if (FSP_SUCCESS != err)
    {
        return err;
    }

    /* Initialize GATT Database */
    status = R_BLE_GATTS_SetDbInst(&g_gatt_db_table);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize GATT server */
    status = R_BLE_SERVS_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /*Initialize GATT client */
    status = R_BLE_SERVC_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Set Prepare Write Queue */
    R_BLE_GATTS_SetPrepareQueue(gs_queue, BLE_GATTS_QUEUE_NUM);
    /* Initialize GATT Discovery Library */
    status = R_BLE_DISC_Init();
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    /* Initialize LED Switch Service client API */
    status = R_BLE_LSC_Init(lsc_cb);
    if (BLE_SUCCESS != status)
    {
        return BLE_ERR_INVALID_OPERATION;
    }

    return status;
}
```

BLE module initialization (*RM_ABS_BLE_Open*)

GATT database registration (*R_BLE_GATTS_SetDbInst*)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

GATT Server function initialization (*R_BLE_SERVS_Init*)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

GATT Client function initialization (*R_BLE_SERVC_Init*)
Note: Code-generated when GATT role is set as whichever Server and Client by QE for BLE.

Service initialization

Code 46. ble_init function

3.5.4 Register callback function

Same as section 3.1.4.

3.5.5 Registering GATT database (R_BLE_GATTS_SetDbInst)

Same as section 3.1.5.

3.5.6 Main loop and scheduler (R_BLE_Execute)

Same as section 3.1.6.

3.5.7 GAP event (gap_cb function)

Refer to section 3.1.7. for details of GAP events which callback function receives. GAP callback function in this demo project is following.

```
void gap_cb(uint16_t type, ble_status_t result, st_ble_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GAP callback function common process. Do not edit comment generated here */
    R_BLE_CMD_AbsGapCb(type, result, p_data);
/* End user code. Do not edit comment generated here */

    switch(type)
    {
        case BLE_GAP_EVENT_STACK_ON:
        {
……
        } break;

        case BLE_GAP_EVENT_CONN_IND:
        {
……
            }
        } break;

        case BLE_GAP_EVENT_DISCONN_IND:
        {
……
        } break;

        case BLE_GAP_EVENT_CONN_PARAM_UPD_REQ:
        {
……
        } break;

        case BLE_GAP_EVENT_ADV_REPT_IND:
        {
……
        } break;

        case BLE_GAP_EVENT_SCAN_OFF:
        {
……
        } break;

/* Hint: Add cases of GAP event macros defined as BLE_GAP_XXX */
/* Start user code for GAP callback function event process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
    }
}
```

In this demo project, many parts of processing when receiving events are also implemented in R_BLE_CMD_AbsGapCb().

Complete GAP initialization

Connection complete

Disconnection has happened

Connection parameter request come from server

Notification of receiving advertising reports from server

Stop scanning

Code 47. GAP callback function

3.5.8 GATTS event (gatts_cb function)

Same as section 3.1.8.

3.5.9 GATTC event (gattc_cb function)

Refer to section 3.1.9. for details of GATTC events which callback function receives. GATTC callback function in this demo project is following.

```
void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Client callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVC_GattcCb(type, result, p_data);
    switch(type)
    {
        case BLE_GATTC_EVENT_CONN_IND:
        {
            R_BLE_CLI_Printf("Start Service Discovery\n");
            /* Start discovery operation after connection established */
            R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries, ARRAY_SIZE(gs_disc_entries), disc_comp_cb);
        } break;

/* Hint: Add cases of GATT Client event macros defined as BLE_GATTC_XXX */
/* Start user code for GATT Client callback function event process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
    }
}
```

Complete connection with GATT server

Code 48. GATTC callback function

3.5.10 VS event (vs_cb function)

Same as section 3.1.10.

3.5.11 Client side Profile API event ([service_name]c_cb function)

Callback function of the client side profile API receives following events.

```
enum   e_ble_servc_event_t {
   BLE_SERVC_WRITE_RSP,
   BLE_SERVC_READ_RSP,
   BLE_SERVC_HDL_VAL_NTF,
   BLE_SERVC_HDL_VAL_IND
}

enum   e_ble_[service name]c_event_t {
   BLE_[service name]C_EVENT_[characteristic name]_WRITE_RSP = 0xXX00,
   BLE_[service name]C_EVENT_[characteristic name]_READ_RSP= 0xXX01,
   BLE_[service name]C_EVENT_[characteristic name]_HDL_VAL_NTF = 0xXX02,
   BLE_[service name]C_EVENT_[characteristic name]_HDL_VAL_IND = 0xXX03,
   BLE_[service name]C_EVENT_[characteristic name]_[descriptor name]_WRITE_RSP = 0xYY00,
   BLE_[service name]C_EVENT_[characteristic name]_[descriptor name]_READ_RSP = 0xYY01,
   :
   :
}
```

Code 49. Client-side Profile API event

Note: The 10th to 15th bits are serial numbers that distinguish attributes (characteristics and descriptors). XX and YY are 00, 04, 08, 10, ..., FC.

Reception condition of the frequently occurring events are shown below.

Table 17. Frequently use events of Profile Client callback

Event	Reception condition
XXX_WRITE_RSP (0xXXX0)	Complete receiving Write Response
XXX_READ_RSP (0xXXX1)	Complete receiving Read Response
XXX_HDL_VAL_NTF (0xXXX2)	Complete receiving Notification
XXX_HDL_VAL_IND (0xXXX3)	Complete receiving Indication

Callback function of client-side profile API is following. (Example of LED switch service)

```
static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for LED Switch Service Client callback function common process. Do not edit comment
generated here */
/* End user code. Do not edit comment generated here */

    switch(type)
    {
/* Hint: Add cases of LED Switch Service client events defined in e_ble_lsc_event_t */
/* Start user code for LED Switch Service Client callback function event process. Do not edit comment
generated here */
        case BLE_LSC_EVENT_SWITCH_STATE_HDL_VAL_NTF:
        {
            if (BLE_SUCCESS == result)
                R_BLE_CLI_Printf("lsc : Recieve Notification from Server \n");
        } break;

        case BLE_LSC_EVENT_BLINK_RATE_READ_RSP:
        {
            if (BLE_SUCCESS == result)
                R_BLE_CLI_Printf("lsc : LED blink rate = 0x%X \n", *(uint8_t *)((st_ble_lsc_evt_data_t
*)(p_data)->p_param));
        } break;

        default:
            break;
/* End user code. Do not edit comment generated here */
    }
}
```

Code 50. Client side profile API callback function

QE for BLE generates skeleton code for Profile client callback function. User can add their own code into the skeleton code.

3.5.12 L2CAP event

Same as section 3.1.12.

3.5.13 Exiting from Software Standby mode

In this demo project, IRQ4 assigned SW1 on EK-RA4W1 is designated as Wakeup Source of Low Power Module. When SW1 on EK-RA4W1 under Software Standby mode is pressed, Software Standby mode is exited then *Callback_ble_sw_irq()* function is executed because it is registered as callback function of IRQ4 interrupt.

```
static void sw_cb(void)
{
    g_inhibit_software_standby = true;
}
……
void Callback_ble_sw_irq(external_irq_callback_args_t *p_args)
{
    FSP_PARAMETER_NOT_USED(p_args);
    R_BLE_SetEvent(sw_cb);
}
```

Clear flag to manage state of Software Standby mode

Interrupt callback function for push switch

Code 51. Callback function of IRQ4 interrupt

3.5.14 CLI (Command Line Interface)

This section is almost same as section 3.1.14. LSC command is registered to gsp_cmds structure in GATT Client demo projects.

```
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
     &g_abs_cmd,
     &g_vs_cmd,
     &g_sys_cmd,
     &g_lsc_cmd,
     &g_ble_cmd
};
```

Code 52. gsp_cmds structure

3.6 FreeRTOS environment (Client, EventGroup as Synchronization Type case)

3.6.1 Create / delete task

Same as section 3.2.1.

3.6.2 Task switching between BLE core task and GATT application task

If event notified by scheduler, part of the BLE protocol stack, is an event for GATT application task, BLE core task activates GATT application task and provides a notification of the event by using event group setting and cleaning technique. In this demo project, event group bit defined in task_functon.h as following.

Table 18. Defined event group bit

Macro name (Value)	Usage
LSC_WAIT_EN_CCCD (0x0001)	Enable CCCD in LED Switch Service
LSC_WAIT_DIS_CCCD (0x0002)	Disable CCCD in LED Switch Service
LSC_WAIT_RECV_NTF (0x0004)	Receive notification from server
LSC_WAIT_WR_BLINK (0x0008)	Change LED blink rate
LSC_WAIT_RD_BLINK (0x0010)	Read LED blink rate

Function which sets event group bit need to implement by the user. The function in this demo project is as follows.

```
void set_lsc_event(EventBits_t uxBitsToSet)
{
    switch(uxBitsToSet)
    {
        case LSC_WAIT_EN_CCCD:
        {
            xEventGroupClearBits(xLscEvent, LSC_WAIT_DIS_CCCD);
        } break;
        case LSC_WAIT_DIS_CCCD:
        {
            xEventGroupClearBits(xLscEvent, LSC_WAIT_EN_CCCD);
        } break;
        default:
            /* Do Nothing */
            break;
    }

    xEventGroupSetBits(xLscEvent, uxBitsToSet);
}
```

Code 53. Set event group bit

3.6.3 Main loop of BLE core task

Same as section 3.2.3.

3.6.4 Main loop of GATT application task

The *lsc_task_entry()* includes main loop of GATT application task. The program flow of this demo project is following.

```
void lsc_task_entry(void * pvParameters)
{
    FSP_PARAMETER_NOT_USED (pvParameters);

    ble_status_t    retval;
    EventBits_t     event;

    gs_conn_hdl = *(uint16_t *)pvParameters;
    xLscEvent = xEventGroupCreate();
    wait_event = LSC_WAIT_EN_CCCD | LSC_WAIT_DIS_CCCD | LSC_WAIT_RECV_NTF | LSC_WAIT_WR_BLINK | LSC_WAIT_RD_BLINK;
    while (1)
    {
        event = xEventGroupWaitBits(
                                    xLscEvent,
                                    wait_event,
                                    pdTRUE,
                                    pdFALSE,
                                    portMAX_DELAY);

        if (LSC_WAIT_EN_CCCD & event)
        {
            retval = R_BLE_LSC_WriteSwitchStateCliCnfg(gs_conn_hdl, (uint16_t *)&g_lsc_ntf_value);
            if (BLE_SUCCESS == retval)
            {
                wait_event = LSC_WAIT_DIS_CCCD | LSC_WAIT_RECV_NTF | LSC_WAIT_WR_BLINK | LSC_WAIT_RD_BLINK;
                r_ble_wake_up_task((void *)g_ble_event_group_handle);
            }
        }
        else if (LSC_WAIT_DIS_CCCD & event)
        {
            retval = R_BLE_LSC_WriteSwitchStateCliCnfg(gs_conn_hdl, (uint16_t *)&g_lsc_ntf_value);
            if (BLE_SUCCESS == retval)
            {
                wait_event = LSC_WAIT_EN_CCCD | LSC_WAIT_RECV_NTF | LSC_WAIT_WR_BLINK | LSC_WAIT_RD_BLINK;
                r_ble_wake_up_task((void *)g_ble_event_group_handle);
            }
        }
        else if (LSC_WAIT_WR_BLINK & event)
        {
            R_BLE_LSC_WriteBlinkRate(gs_conn_hdl, &g_blink_rate);
        }
        else if (LSC_WAIT_RD_BLINK & event)
        {
            retval = R_BLE_LSC_ReadBlinkRate(gs_conn_hdl);
            if (BLE_SUCCESS == retval){
                r_ble_wake_up_task((void *)g_ble_event_group_handle);
            }
        }
    }

    vTaskDelete(NULL);
}
```

Store connection handle

Create event group for transition task status.

Wait for event from BLE core task

Wait for enable CCCD event from BLE core task

Wait for disable CCCD event from BLE core task

Wait for write LED blink rate event from BLE core task

Wait for read LED blink rate event from BLE core task

Code 54. lsc_task_entry function

3.6.5 Initialization process

Same as section3.5.3.

3.6.6 Register callback function

Same as section 3.1.4

3.6.7 Registering GATT database (R_BLE_GATTS_SetDbInst)

Same as section 3.1.5.

3.6.8 Main loop and scheduler (R_BLE_Execute)

Same as section 3.2.8.

3.6.9 GAP event (gap_cb function)

Same as section 3.5.7.

3.6.10 GATTC event (gattc_cb function)

Almost the same as section 3.5.9. The difference from BareMetal environment is that,

- GATT application task is created when connection is established with the client.
- GATT application task is deleted when client disconnects.

Implementation of this demo project is following.

```
void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Client callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVC_GattcCb(type, result, p_data);
    switch(type)
    {
        case BLE_GATTC_EVENT_CONN_IND:
        {
            /* Start discovery operation after connection established */
            R_BLE_CLI_Printf("Start Service Discovery\n");

            R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries, ARRAY_SIZE(gs_disc_entries), disc_comp_cb);

            /* Create GATT application task */
            xTaskCreate(lsc_task_entry, "lsc_task", 128, &g_conn_hdl, 4, &g_lsc_task);

        } break;

        case BLE_GATTC_EVENT_DISCONN_IND:
        {

            /* task delete */
            delete_lsc_task_rsrc();

        } break;

        default:
            /* Do nothing */
            break;

/* Hint: Add cases of GATT Client event macros defined as BLE_GATTC_XXX */
/* Start user code for GATT Client callback function event process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
    }
}
```

Code 55. GATTC callback function

3.6.11 VS event (vs_cb function)

Same as section 3.1.10.

3.6.12 Client side Profile API event ([service_name]c_cb function)

Almost the same as section 3.5.11. The difference from BareMetal environment is that event group bits are adjusted according to the data received from client-side profile API event. As a result, the GATT Application task is activated per the function definition provided in Section 3.6.2.

```
static void lsc_cb(uint16_t type, ble_status_t result, st_ble_servc_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for LED Switch Service(Custom Service) Client callback function common process. */
/*Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    switch(type)
    {
/* Hint: Add cases of LED Switch Service(Custom Service) client events defined in e_ble_lsc_event_t */
/* Start user code for LED Switch Service(Custom Service) Client callback function event process. Do not edit
comment generated here */

        case BLE_LSC_EVENT_SWITCH_STATE_HDL_VAL_NTF:
        {
            Note: Add processing when an event is received here.
        } break;

        case BLE_LSC_EVENT_BLINK_RATE_READ_RSP:
        {
            Note: Add processing when an event is received here.
        } break;

        default:
            break;

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

Receive notification from server-side

Change LED blink rate

Code 56. Profile Client callback function

3.6.13 L2CAP event

Same as section 3.1.12.

3.6.14 CLI (Command Line Interface)

Same as section 3.5.14.

3.7 FreeRTOS environment (Client, Semaphore as Synchronization Type case)

3.7.1 Create / delete task
Same as section 3.3.1.

3.7.2 Task switching between BLE core task and GATT application task
Same as section 3.6.2.

3.7.3 Main loop of BLE core task
Same as section 3.3.3.

3.7.4 Main loop of BLE execute task
Same as section 0.

3.7.5 Main loop of GATT application task
Same as section 3.6.4.

3.7.6 Initialization process
Same as section3.5.3.

3.7.7 Register callback function
Same as section 3.1.4.

3.7.8 Registering GATT database (R_BLE_GATTS_SetDbInst)
Same as section 3.1.5.

3.7.9 Main loop and scheduler (R_BLE_Execute)
Same as section 3.3.9..

3.7.10 GAP event (gap_cb function)
Same as section 3.5.7.

3.7.11 GATTC event (gattc_cb function)
Same as section 3.6.12.

3.7.12 VS event (vs_cb function)
Same as section 3.1.10.

3.7.13 Client side Profile API event ([service_name]c_cb function)
Same as section 3.6.12.

3.7.14 L2CAP event
Same as section 3.1.12.

3.7.15 CLI (Command Line Interface)

Same as section 3.5.14.

3.8 Azure RTOS environment (Client)

3.8.1 Create / delete task

Same as section 3.4.1.

3.8.2 Task switching between BLE core task and GATT application task

Same as section 3.4.2.

3.8.3 Main loop of BLE core task

Same as section 3.4.3.

3.8.4 Main loop of BLE execute task

Same as section 3.4.4.

3.8.5 Main loop of GATT application task

The *lsc_task_entry()* includes main loop of GATT application task. The program flow of this demo project is following.

```
void lsc_task_entry(unsigned long Parameters)
{

    ble_status_t    retval;
    unsigned long event;

    gs_conn_hdl = (uint16_t)Parameters;
    tx_event_flags_create(&xLscEvent, "LSC_EVENT_FLAG");
    wait_event = LSC_WAIT_EN_CCCD | LSC_WAIT_DIS_CCCD | LSC_WAIT_RECV_NTF | LSC_WAIT_WR_BLINK | LSC_WAIT_RD_BLINK;

    while (1)
    {
        tx_event_flags_get(&xLscEvent, wait_event, TX_OR_CLEAR, &event, TX_WAIT_FOREVER);

        if (LSC_WAIT_EN_CCCD & event)
        {
            retval = R_BLE_LSC_WriteSwitchStateCliCnfg(gs_conn_hdl, (uint16_t *)&g_lsc_ntf_value);
            if (BLE_SUCCESS == retval)
            {
                wait_event = LSC_WAIT_DIS_CCCD | LSC_WAIT_RECV_NTF | LSC_WAIT_WR_BLINK | LSC_WAIT_RD_BLINK;
                r_ble_wake_up_task((void *)g_ble_event_group_handle);
            }
        }
        else if (LSC_WAIT_DIS_CCCD & event)
        {
            retval = R_BLE_LSC_WriteSwitchStateCliCnfg(gs_conn_hdl, (uint16_t *)&g_lsc_ntf_value);
            if (BLE_SUCCESS == retval)
            {
                wait_event = LSC_WAIT_EN_CCCD | LSC_WAIT_RECV_NTF | LSC_WAIT_WR_BLINK | LSC_WAIT_RD_BLINK;
                r_ble_wake_up_task((void *)g_ble_event_group_handle);
            }
        }
        else if (LSC_WAIT_WR_BLINK & event)
        {
            retval = R_BLE_LSC_WriteBlinkRate(gs_conn_hdl, &g_blink_rate);
            if (BLE_SUCCESS == retval)
            {
                wait_event = LSC_WAIT_EN_CCCD | LSC_WAIT_RECV_NTF | LSC_WAIT_WR_BLINK | LSC_WAIT_RD_BLINK;
                r_ble_wake_up_task((void *)g_ble_event_group_handle);
            }
        }
        else if (LSC_WAIT_RD_BLINK & event)
        {
            retval = R_BLE_LSC_ReadBlinkRate(gs_conn_hdl);
            if (BLE_SUCCESS == retval){
                r_ble_wake_up_task((void *)g_ble_event_group_handle);
            }
        }
    }
}
```

Store connection

Create event flags for transition task status.

Wait for event from BLE core task

Wait for enable CCCD event from BLE core task

Wait for disable CCCD event from BLE core task

Wait for write LED blink rate event from BLE core task

Wait for read LED blink rate event from BLE core task

Code 57. lsc_task_entry function

3.8.6 Initialization process

Same as section 3.5.3.

3.8.7 Register callback function

Same as section 3.1.4

3.8.8 Registering GATT database (R_BLE_GATTS_SetDbInst)

Same as section 3.1.5.

3.8.9 Main loop and scheduler (R_BLE_Execute)

Same as section 3.3.9.

3.8.10 GAP event (gap_cb function)

Same as section 3.5.7.

3.8.11 GATTC event (gattc_cb function)

Almost the same as section 3.5.9. The difference from BareMetal environment is that,

- GATT application task is created when connection is established with the client.
- GATT application task is deleted when client disconnects.

Implementation of this demo project is following.

```
void gattc_cb(uint16_t type, ble_status_t result, st_ble_gattc_evt_data_t *p_data)
{
/* Hint: Input common process of callback function such as variable definitions */
/* Start user code for GATT Client callback function common process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

    R_BLE_SERVC_GattcCb(type, result, p_data);
    switch(type)
    {
        case BLE_GATTC_EVENT_CONN_IND:
        {
            R_BLE_CLI_Printf("Start Service Discovery\n");
            /* Start discovery operation after connection established */
            R_BLE_DISC_Start(p_data->conn_hdl, gs_disc_entries, ARRAY_SIZE(gs_disc_entries), disc_comp_cb);

            /* Create LED Switch Service Task */
            tx_thread_create(&ble_lsc_task, (CHAR*) "BLE_LSC_TASK", lsc_task_entry, g_conn_hdl,
                             &ble_lsc_task_stack, 512, 4, 4, 4, TX_AUTO_START);
        } break;

/* Hint: Add cases of GATT Client event macros defined as BLE_GATTC_XXX */
/* Start user code for GATT Client callback function event process. Do not edit comment generated here */
        case BLE_GATTC_EVENT_DISCONN_IND:
        {
            /* task delete */
            delete_lsc_task_rsrc();

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

/* Start user code for GATT Client callback function closing process. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
}
```

Code 58. GATTC callback function

3.8.12 VS event (vs_cb function)

Same as section 3.1.10.

3.8.13 Client side Profile API event ([service_name]c_cb function)

Same as section 3.6.12.

3.8.14 L2CAP event

Same as section 3.1.12.

3.8.15 CLI (Command Line Interface)

Same as section 3.5.14.

4. Appendix

4.1 How to make and configure new project

This section describes required configuration to create a project for BLE application.

4.1.1 Create a new project

1. Launch e^2 studio and select **File→New→C/C++ Project**. In **New C/C++ Project** dialog, select **Renesas RA** and **Renesas RA C Executable Project** and click on the **Next** button.

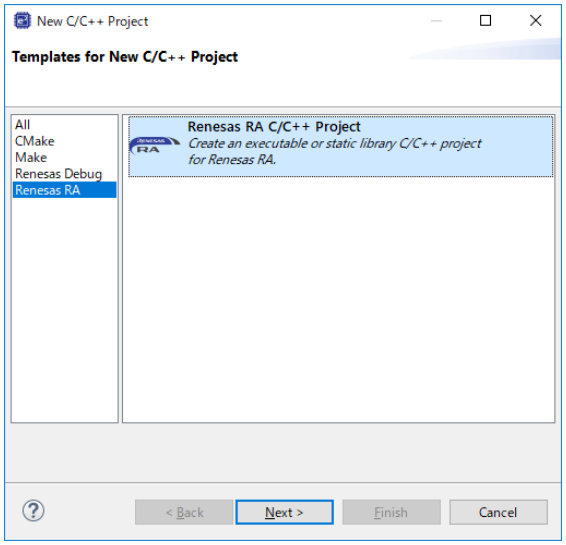

Figure 27. Templates for New C/C++ Project

2. Enter project name and click on **Next** button. The project named **SampleAppl** in this document.

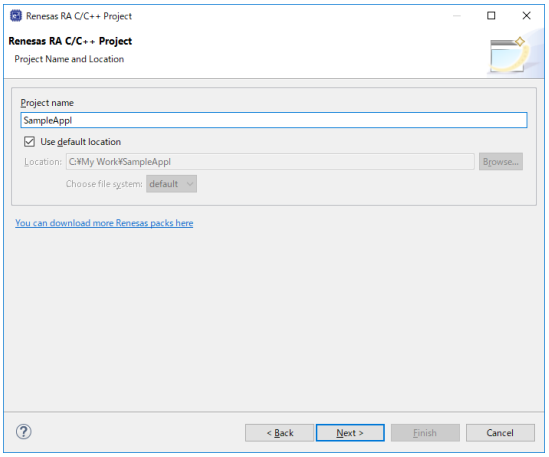

Figure 28. New Renesas Executable Project

3. Select the **Custom User Board (Any Device)** from **Board**, **R7FA4W1AD2CNG** from **Device**.

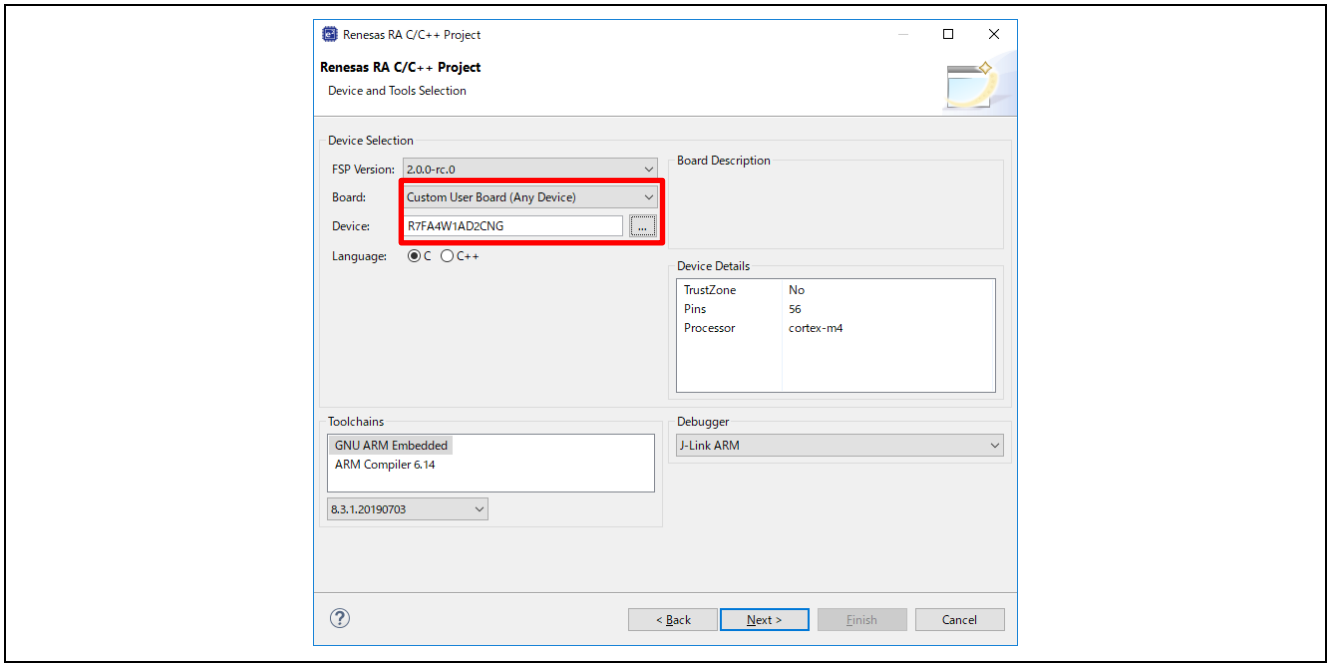

Figure 29. Project Configuration (Board and Device)

4. When making BLE application on BareMetal environment, choose **No RTOS**. When making the application on FreeRTOS environment, choose **FreeRTOS**. When making the application on Azure RTOS environment, choose **Azure RTOS ThreadX**.

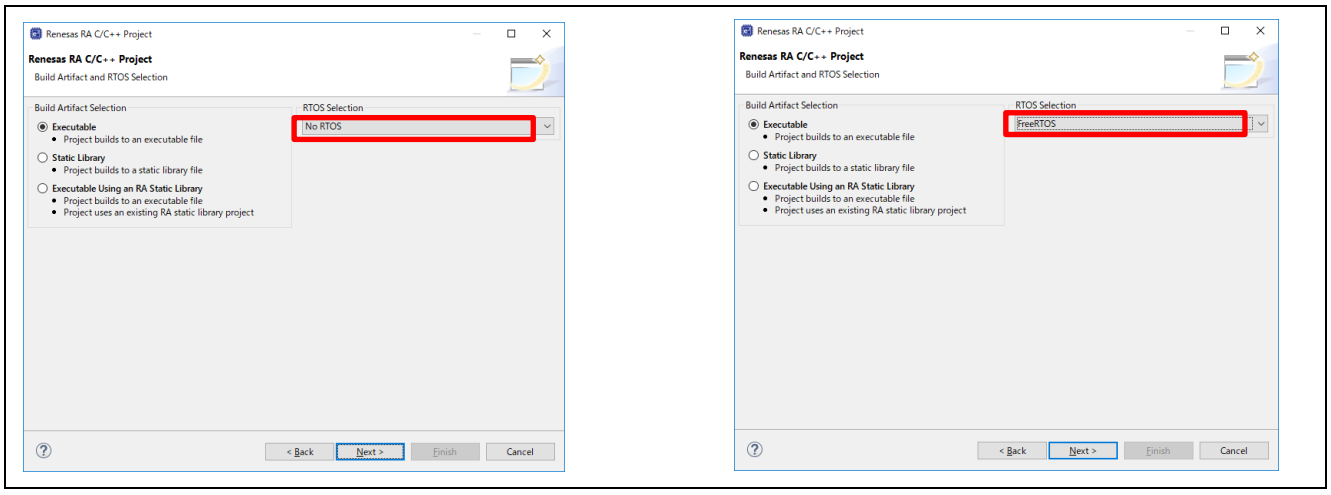

Figure 30. Project Configuration

5. Click **Next** button.

6. When making BLE application on BareMetal environment, choose **BareMetal -Minimal**. When making the application with FreeRTOS environment, choose **FreeRTOS -Minimal- Static Allocation**. When making the application with Azure RTOS environment, choose **Azure RTOS ThreadX – Minimal**.

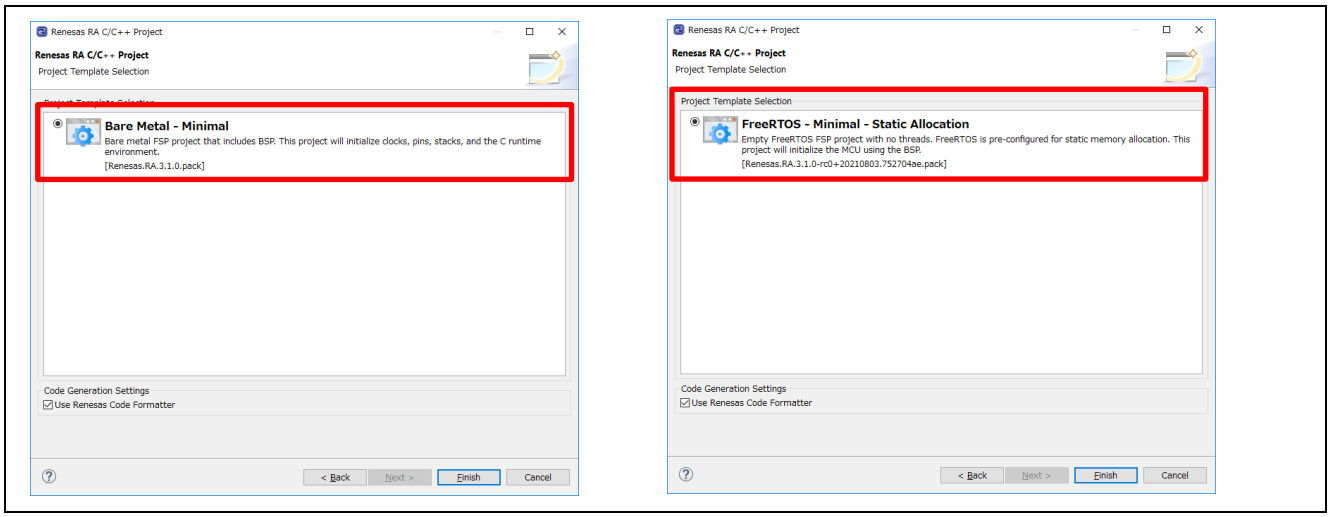

Figure 31. Project Configuration (Select Template)

7. Click **Finish** button. After a while, project will be created

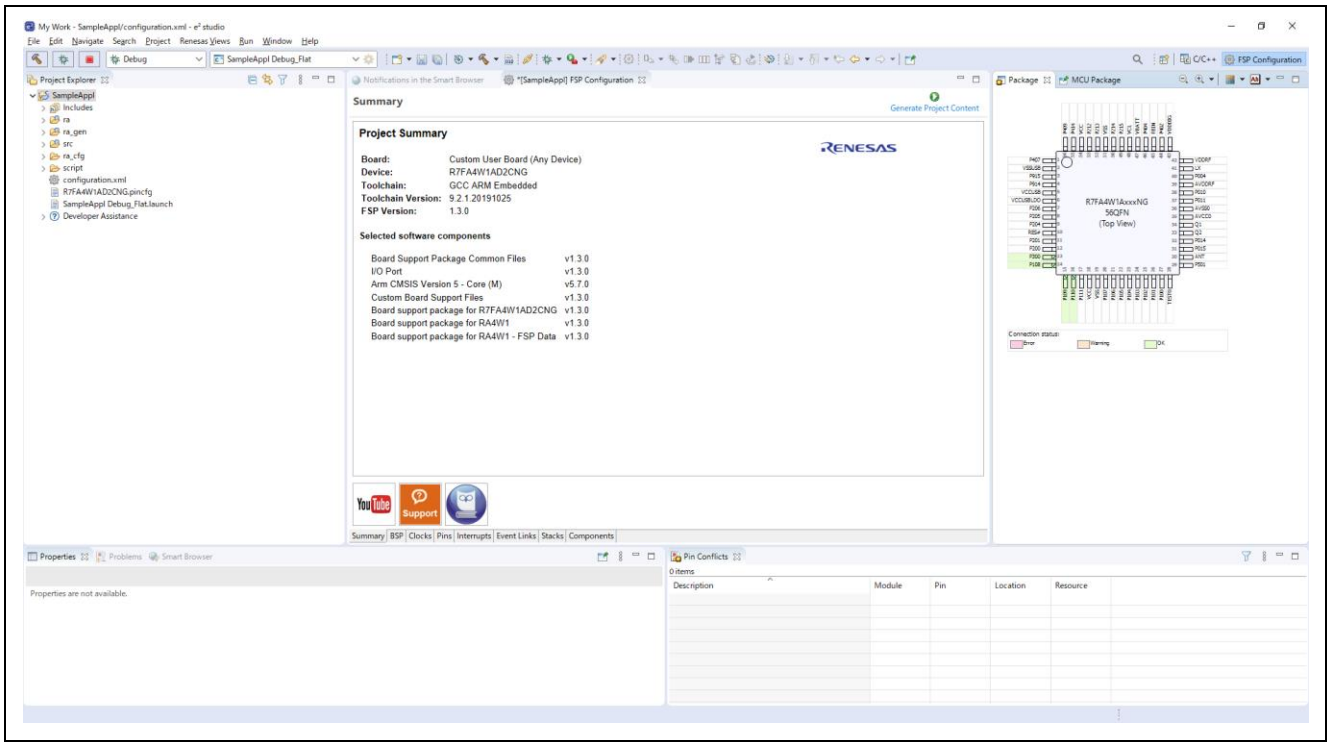

Figure 32. Project Overview

4.1.2 Heap and Stack configuration

Set heap and stack configuration as following on FSP configuration **BSP** tab.

- [RA Common]→[Main stack size (bytes)] : 0x1000
- [RA Common]→[Heap size (bytes)] : 0x1000

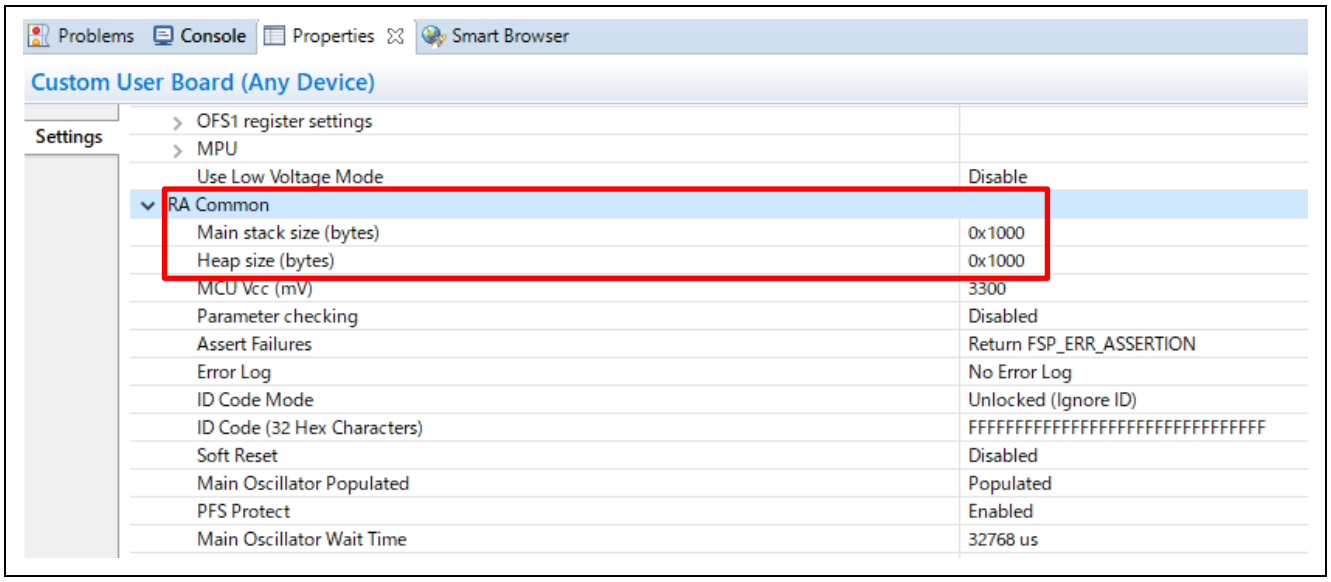

Figure 33. BSP configuration

If the Properties tab is not visible, choose **Window→Show View→Properties** on e² studio menu bar.

4.1.3 Clocks configuration

Set clock frequencies as following on FSP configuration **Clocks** tab.

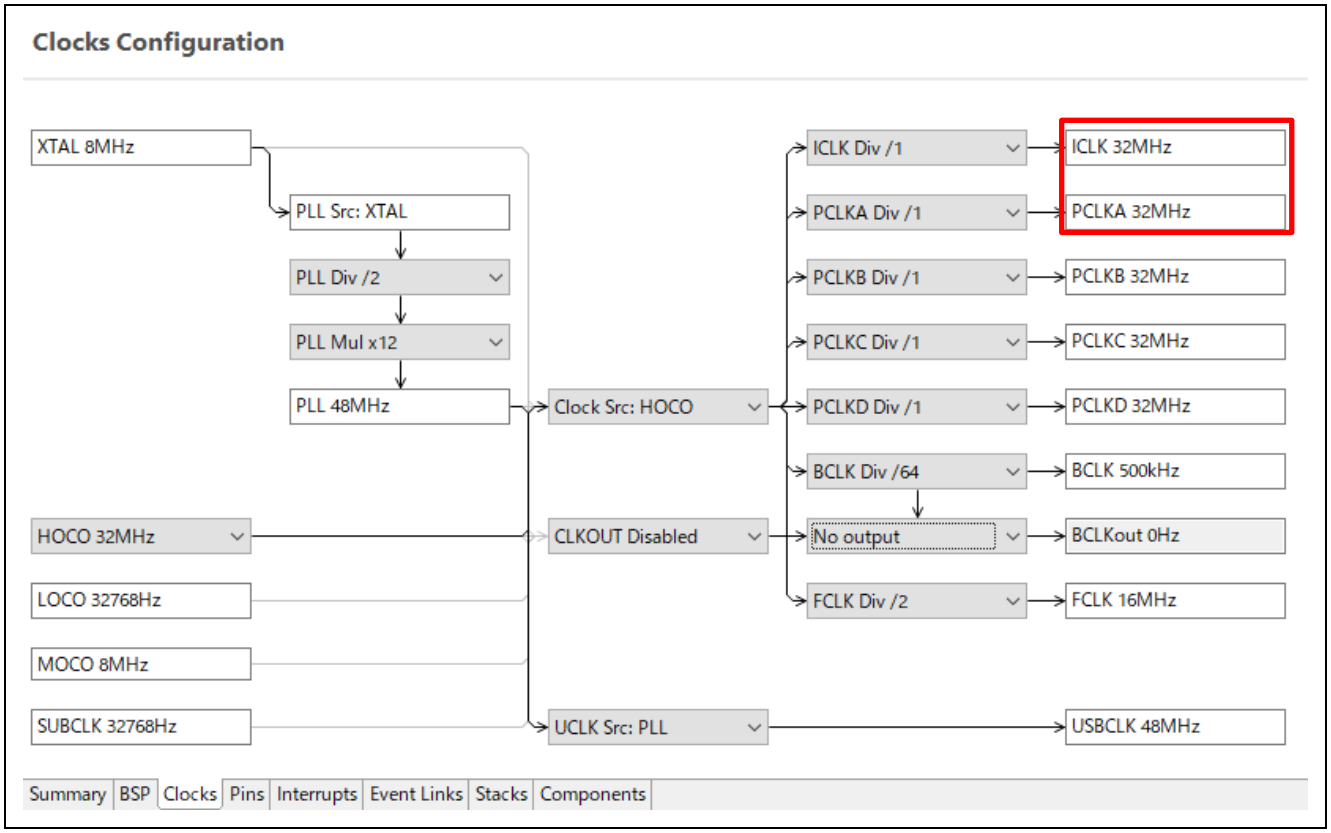

Figure 34. Clocks configuration

The minimum clock frequency for BLE module is following.

- System clock (ICLK) : 8MHz
- Peripheral module clock A (PCLKA) : 8MHz

However, the BLE module is optimized to operate with ICLK = 32MHz and PCLKA=32MHz.Therefore, Renesas recommends configuring frequency of ICLK and PCLKA to 32MHz for maximizing BLE performance.

4.1.4 Add and configure BLE module

This section describes how to add / configure BLE module into BLE application. Click **configuration.xml** in the project and add / configure BLE module on FSP configuration **Stacks** tab. Procedure about adding BLE module is different for BareMetal, FreeRTOS and Azure RTOS environment. Section 4.1.4.1 describes the procedure for BareMetal environment. Section 4.1.4.2 describes the procedure for FreeRTOS environment. Section 4.1.4.3 describes the procedure for Azure RTOS. And BLE module configuration is common to BareMetal, FreeRTOS and Azure RTOS environment. The configuration is described in detail in section 4.1.4.4 and 4.1.4.5.

4.1.4.1 Add BLE module in BareMetal environment

1. Click **New Stack** and add **Middleware→BLE Abstraction Driver on rm_ble_abs** to **HAL/Common**.

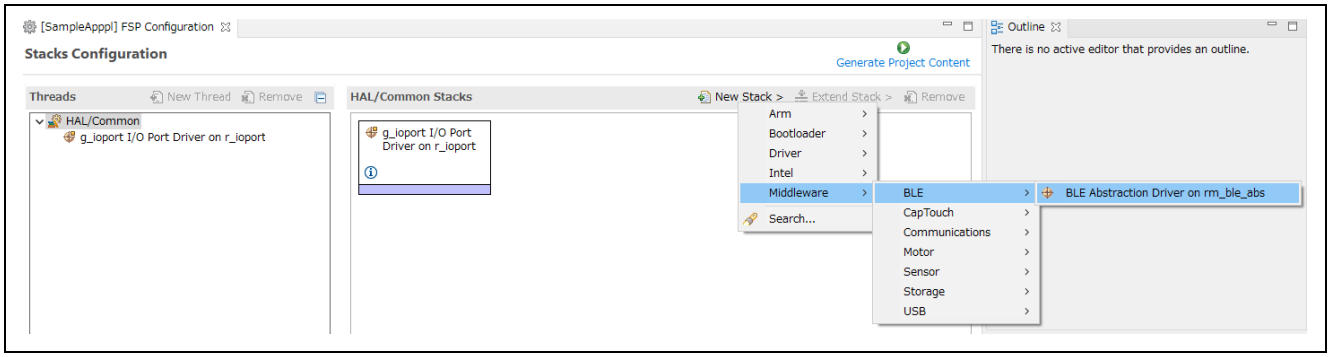

Figure 35. Add BLE module

2. Click **Add BLE Library for Network** box and select **New→Network Driver on r_ble_XXX.**

   “Extended”, “Balance”, and “Compact” can be selected for XXX according to the supported BLE features. Refer to section 1.3 about supported BLE features of each library type.

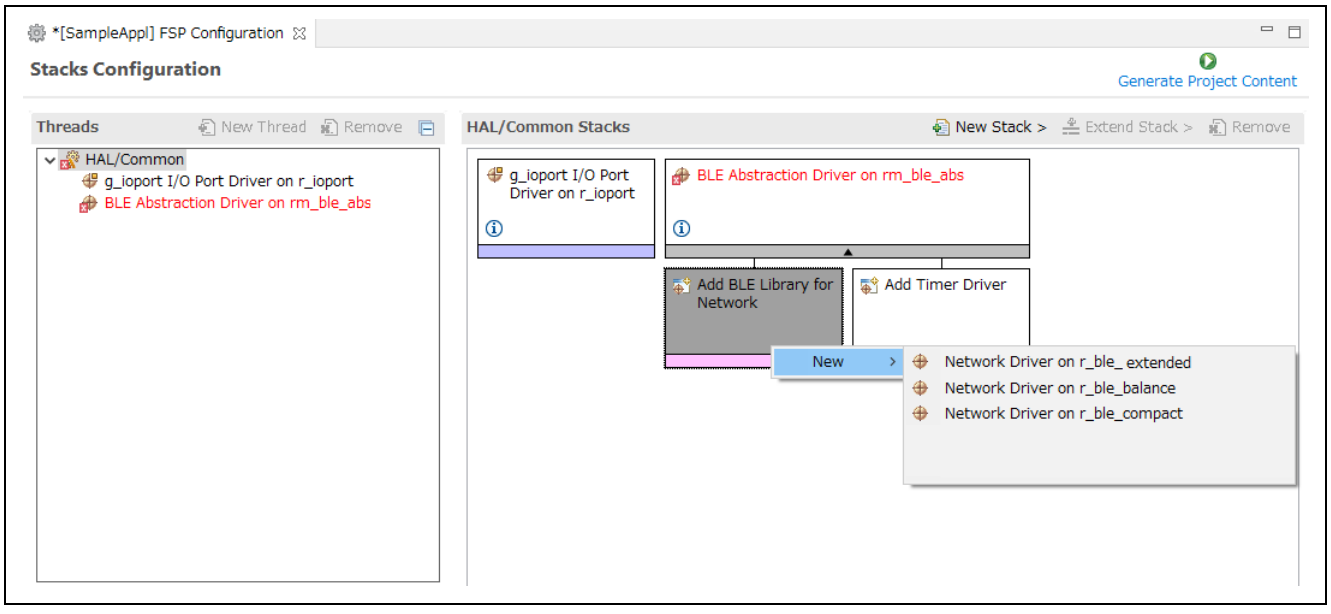

Figure 36. Select module type

The BLE FSP module has properties which may change according to user scenario. Refer to section 4.1.4.4 about description of the properties. And The driver includes some peripheral driver. Configuration for these peripherals describes in section 4.1.4.5.

4.1.4.2 Add BLE module in FreeRTOS environment

1. Click **New Thread** on Thread area and add New Thread. In this example, the New Thread is named BLE Core Task. Note that the symbol of the New Thread should be “ble_core_task” in case of using QE for BLE, because QE for BLE expects so.

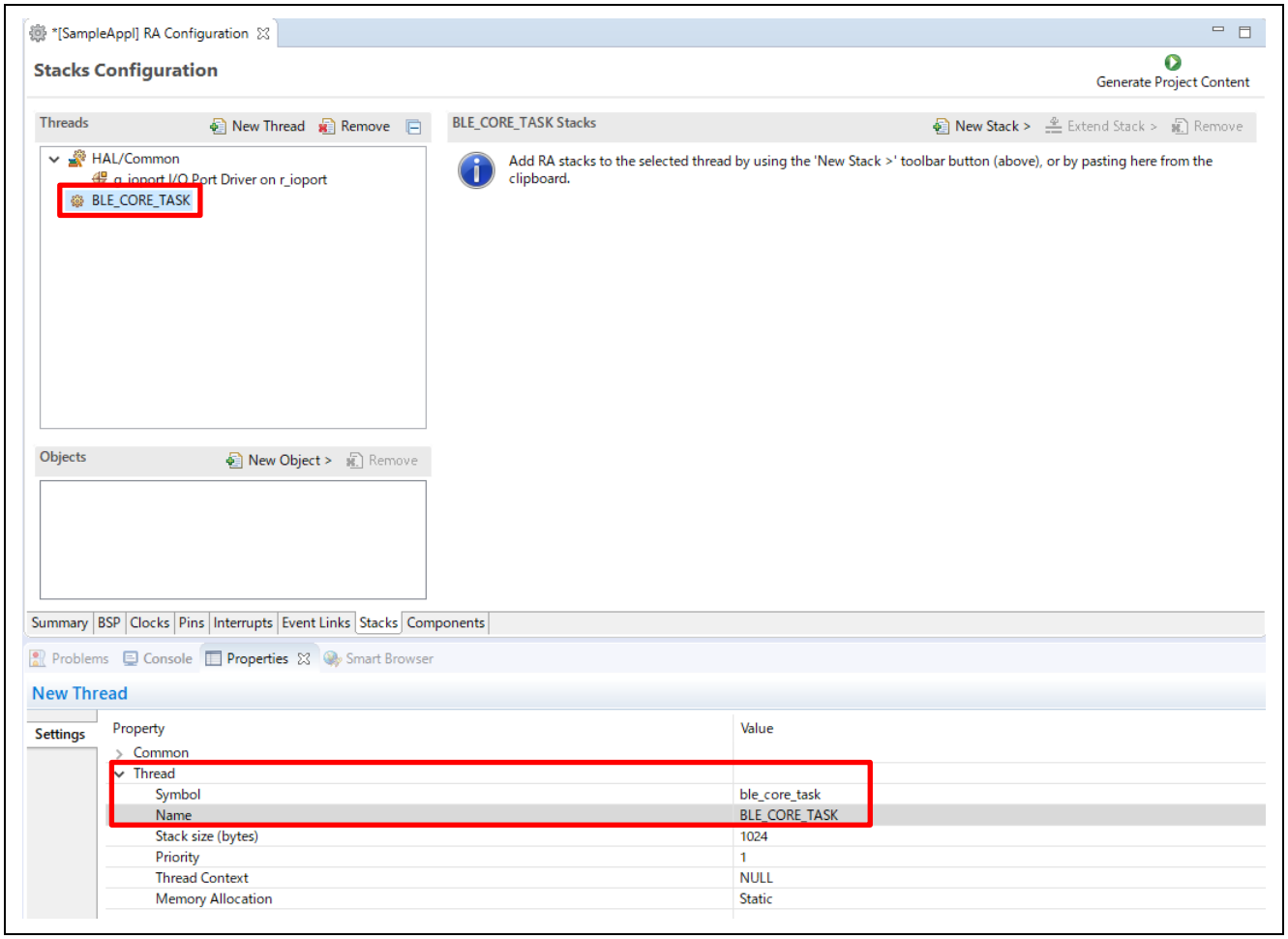

Figure 37. Add BLE Core Task

2. Change Stack size as 2048[bytes]. The BLE stack included in this application requires 1.5 [KB] of memory space to use. And the profile itself included in this application requires 0.4[KB] memory space to use.

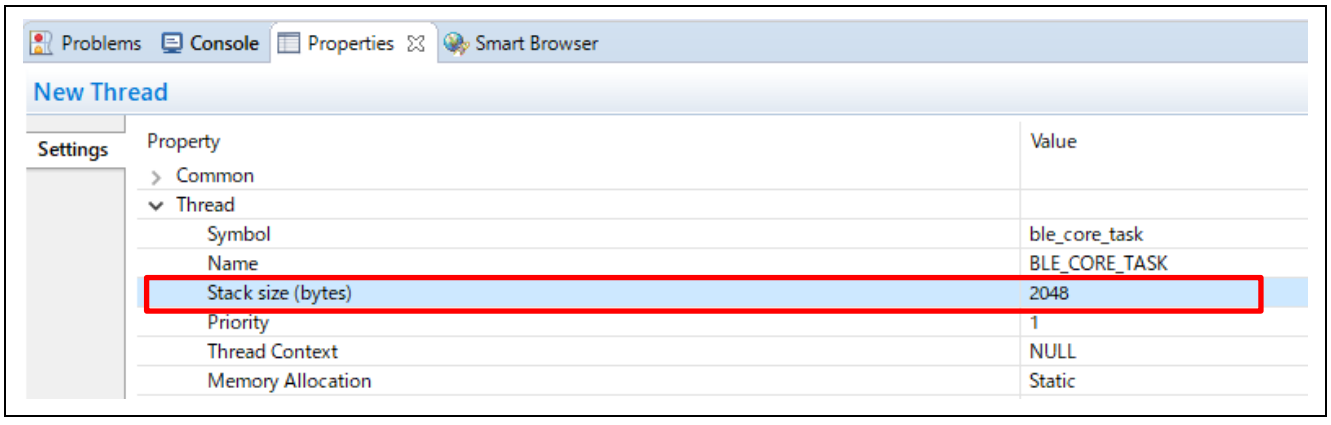

Figure 38. Stack size of BLE Core Task

3. BLE_CORE_TASK priority depends on following *BLE_Driver* FSP module property.

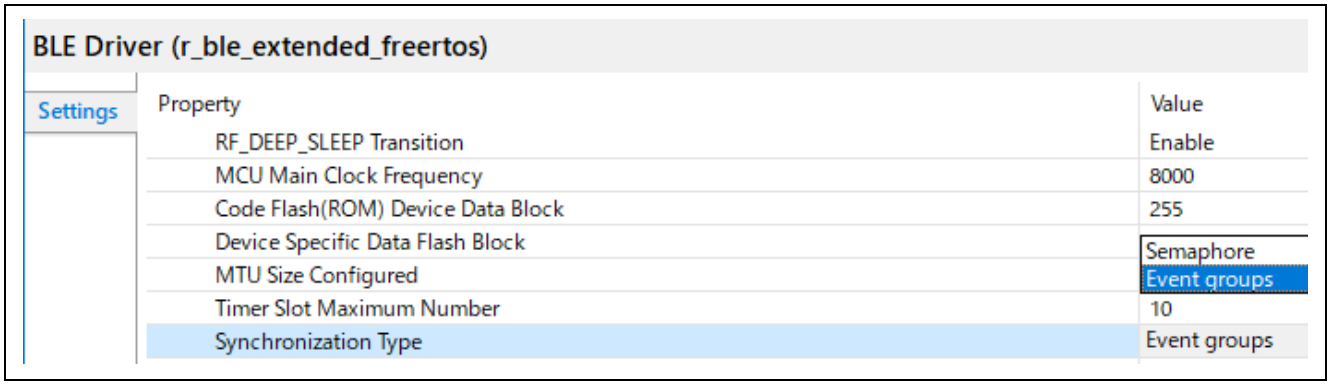

Figure 39. Synchronization Type

In case of choose *Event group*, priority of BLE Core Task should be highest priority (configMAX_PRIORITIES-1).

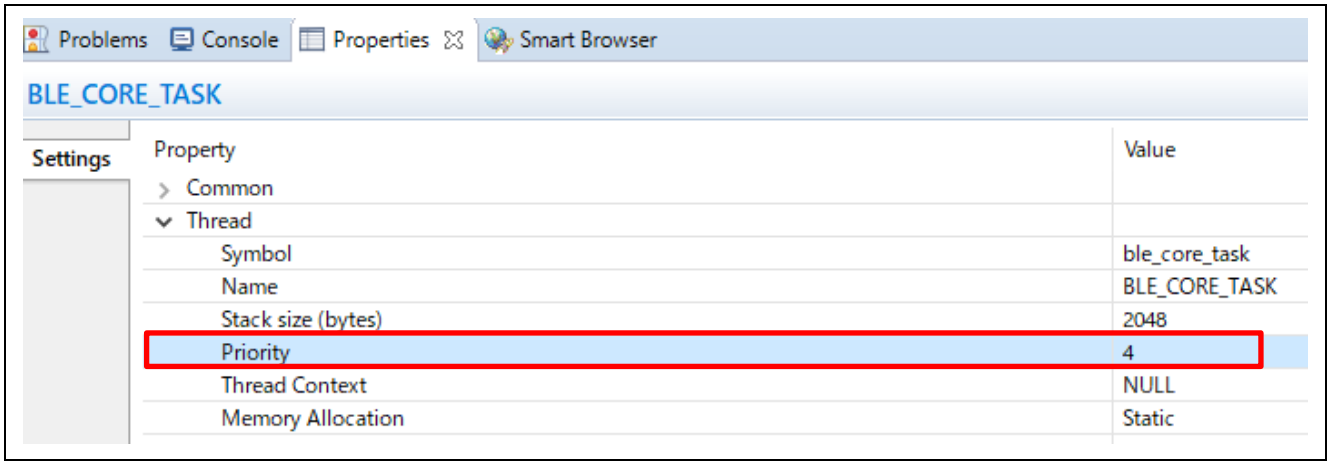

Figure 40. Priority of BLE Core Task (Event Group case)

In case of choose *Semaphore*, priority of BLE Core Task should NOT be highest priority. In demo project, priority of the task configured as 2.

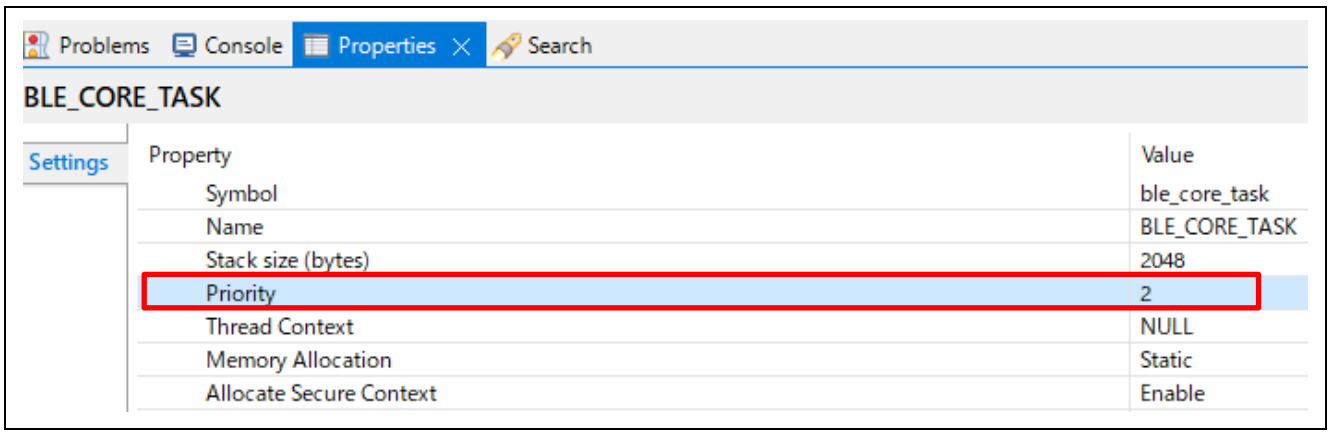

Figure 41. Priority of BLE Core Task (Semaphore case)

4. Change FreeRTOS configurations as following on BLE Core task **Properties** tab.

Table 19. FreeRTOS configuration

Item	Changed Value	Default Value
Common > General > Use Mutexes	Enabled	Disabled
Common > General > Use Recursive Mutexes	Enabled	Disabled
Common > Memory Allocation > Support Dynamic Allocation	Enabled	Disabled
Common > Memory Allocation > Total Heap Size	4096	0
Common > Optional Functions > *xTimerPendingFunctoinCall()* Function	Enabled	Disabled

5. Click **New Stack** and add **Middleware→BLE Abstraction Driver on rm_ble_abs** to **BLE Core task**.

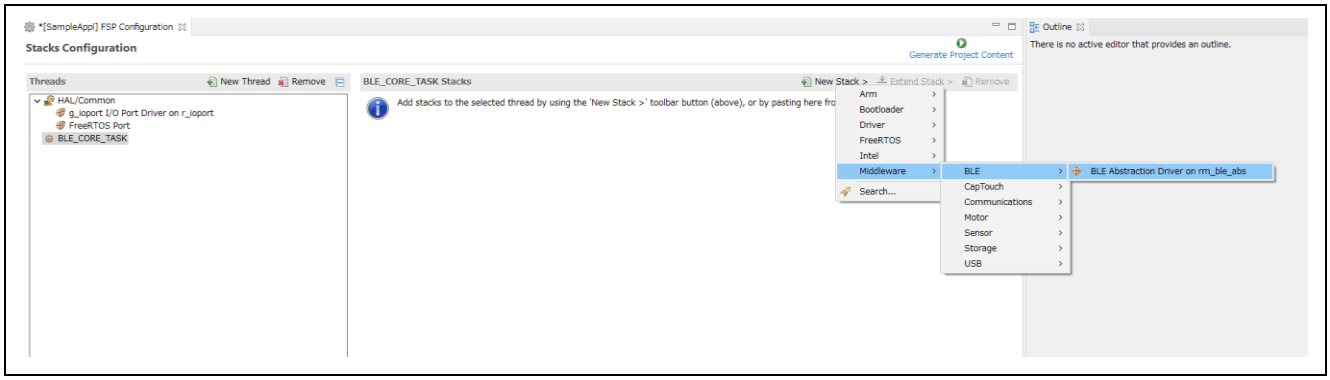

Figure 42. Add BLE module

6. Click **Add BLE Library for Network** box and select **New→Network Driver on r_ble_XXX_freertos.**

   “Extended”, “Balance”, and “Compact” can be selected for XXX according to the supported BLE features. Number of supported BLE features decreases in the order of “Extended”, “Balance”, “Compact”. Refer to section 1.3 about supported BLE features of each library type.

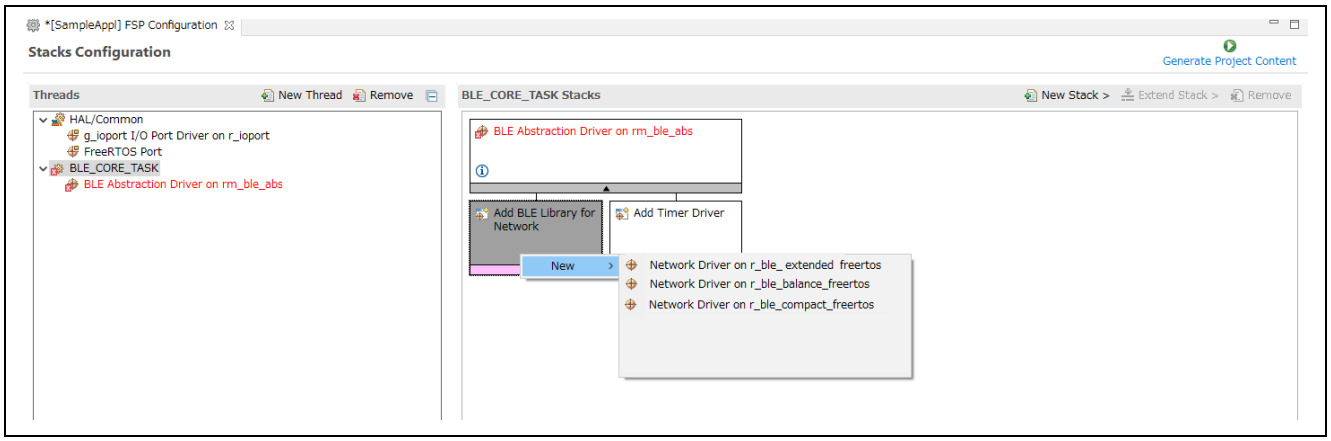

Figure 43. Select module type

The driver includes some peripheral driver. Configuration for these peripherals describes in section 4.1.4.5.

6. Add **Heap4** module to **HAL/Common**.

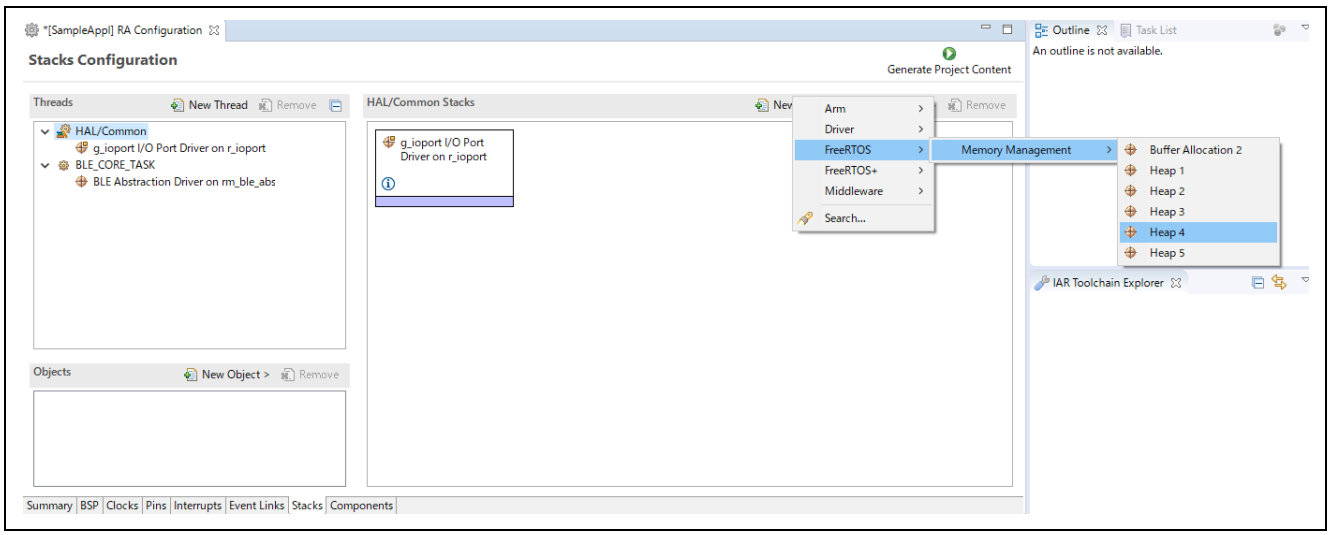

Figure 44. Add Heap4 module

4.1.4.3 Add BLE module in Azure RTOS environment

1. Click **New Thread** on Thread area and add New Thread. In this example, the New Thread is named BLE Core Task. Note that the symbol of the New Thread should be “ble_core_task” in case of using QE for BLE, because QE for BLE expects so.

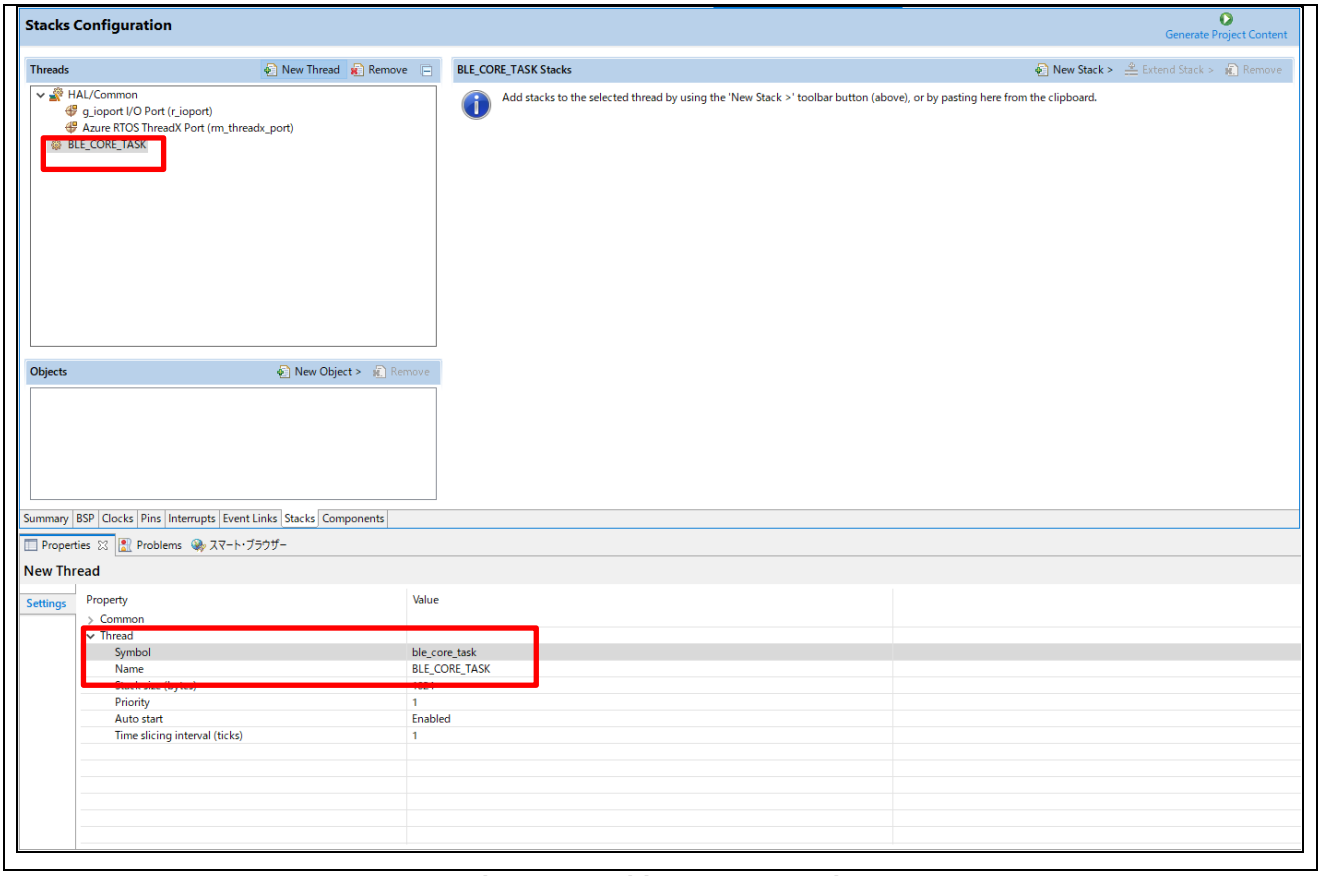

Figure 45. Add BLE Core Task

2. Change Stack size as 2048[bytes]. The BLE stack included in this application requires 1.5 [KB] of memory space to use. And the profile itself included in this application requires 0.4[KB] memory space to use.

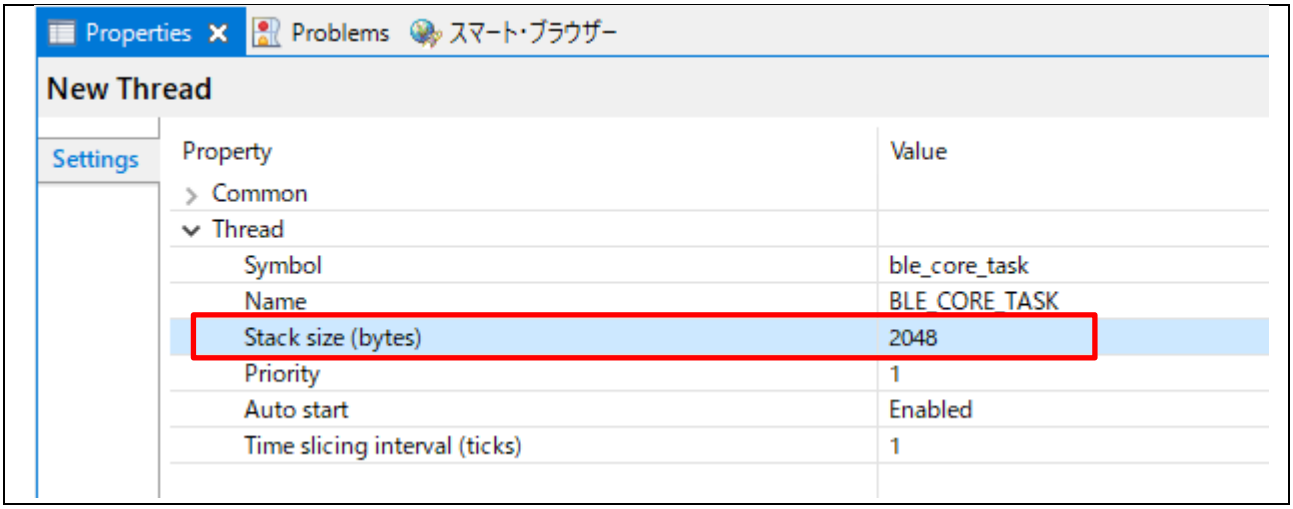

Figure 46. Change Stack Size

3. Priority of BLE Core Task should NOT be highest priority. In demo project, priority of the task configured as 3.

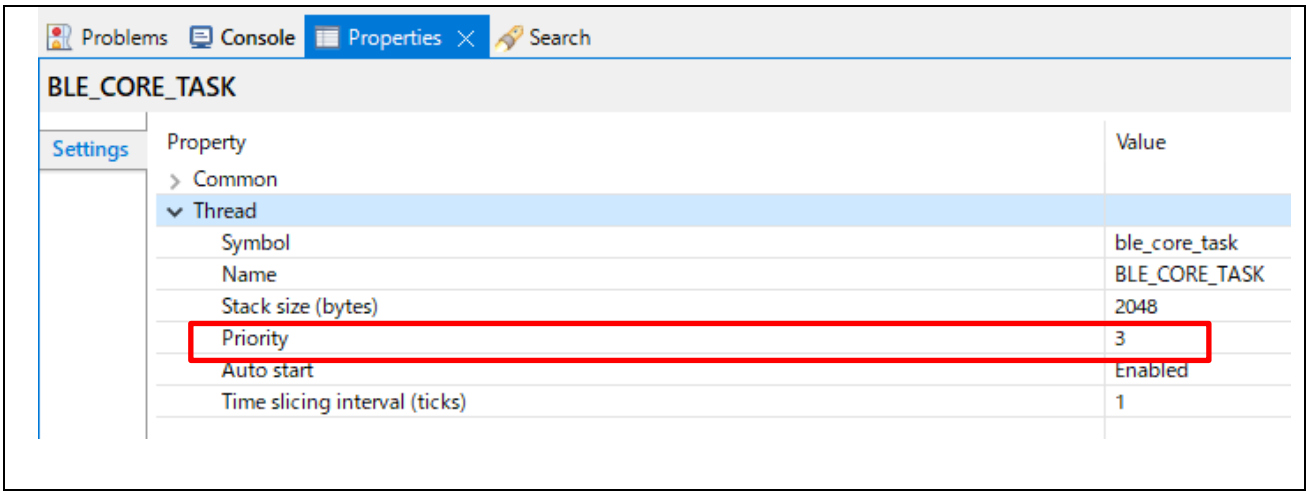

Figure 47. Change Priority and Time slicing interval (ticks)

4. Change Azure RTOS configurations as following on BLE Core task **Properties** tab.

Table 20. AzureRTOS configuration

Item	Changed Value	Default Value
Common > Timer > Timer Ticks Per Second	1000	100
Common > Timer > Timer Thread Priority	2	0

5. Click **Add BLE Library for Network** box and select **New→BLE Driver on r_ble_XXX_threadx.**

   “Extended”, “Balance”, and “Compact” can be selected for XXX according to the supported BLE features. Number of supported BLE features decreases in the order of “Extended”, “Balance”, “Compact”. Refer to section 1.3 about supported BLE features of each library type.

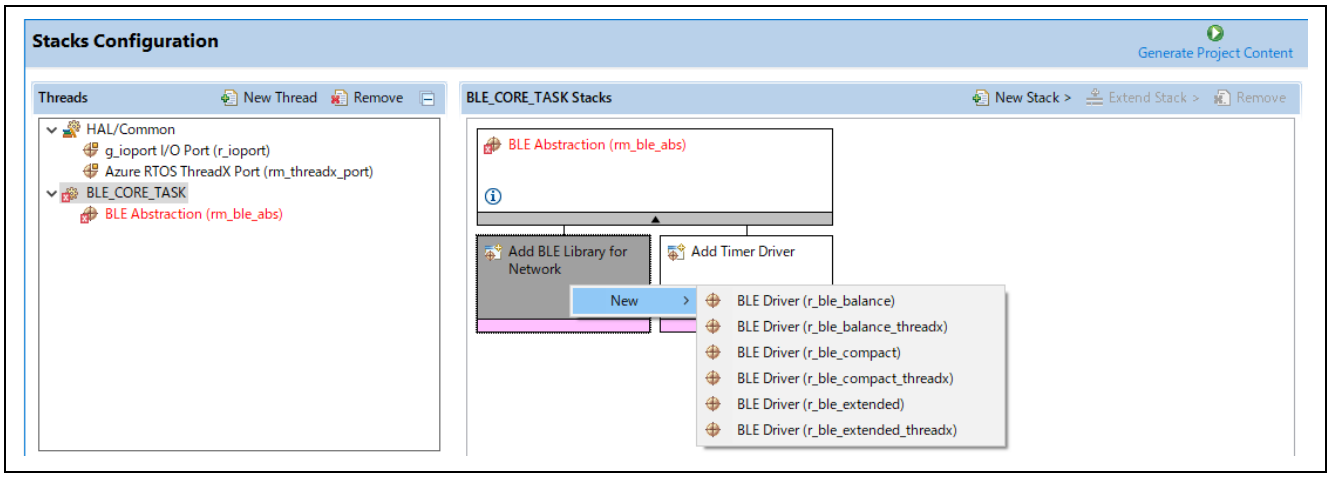

Figure 48. Select module type

The BLE FSP module has properties which may change according to user scenario. Refer to section 4.1.4.4 about description of the properties. And The driver includes some peripheral driver. Configuration for these peripherals describes in section 4.1.4.5.

4.1.4.4 BLE module configurations

This section describes BLE module configuration options and related modules. BLE module include following configuration categories. About each category will describe from following.

- Common
- Module BLE Abstraction Driver on rm_ble_abs

1. Common options

The BLE module can change BD address etc. by modifying common options on FSP configuration. **BLE Abstraction (rm_ble_abs) FSP module and BLE Driver (r_ble_xxxx) module have same properties. Users need to enter the same values for both modules.** The changed options are automatically reflected to the r_ble_cfg.h when generating code.

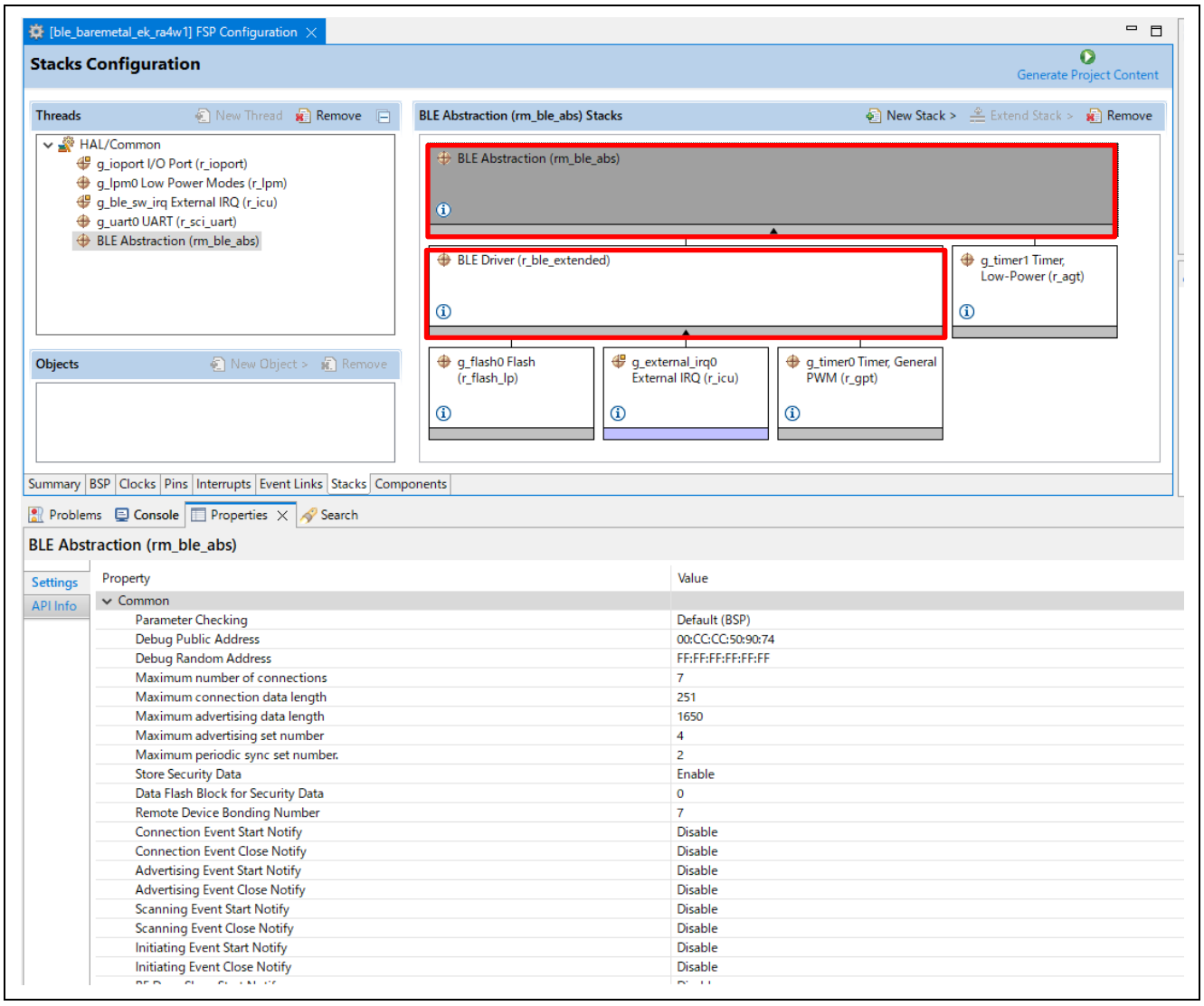

Figure 49. Common options

Option names and setting values in the configuration are listed following. Items shown in **bold may be modified** according to user's environment.

Table 21. Common options

Configuration options	
Debug Public Address Default: {0xFF,0xFF,0xFF,0x50,0x90,0x74}	Initial Public Address. If the public addresses in Code Flash and Data Flash are all 0x00 or 0xFF, the demo project will use this value as public address. Refer to section 4.2 for details.
Debug Random Address Default: {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}	Initial Static Address. If the static addresses in the Code Flash and the Data Flash are all 0x00 or 0xFF, the demo project will use this value as static address. Refer to section 4.2 for details.
Maximum number of connections Default: 7	Maximum number of simultaneous connections. Range : 1 to 7
Maximum connection data length Default: 251	Maximum packet data length (bytes). Range: 27 to 251
Maximum advertising data length Default: 1650	Maximum advertising data length (bytes). Range: 31 to 1650 This configuration will be ignored and fix at 31 bytes when "balance" or "compact" is selected for BLE library.
Maximum advertising set number Default: 4	Maximum number of the advertising set. Range: 1 to 4 This configuration will be ignored and fix at 1 when "balance" or "compact" is selected for BLE library.
Maximum periodic sync set number Default: 2	Maximum number of simultaneous synchronizations against periodic advertising. Range: 1 to 2 This configuration will be ignored and fix at 1 when "balance" or "compact" is selected for BLE library.
Store Security Data Default: Disable	Enable or disable the security data management. Range: Enable or Disable Bonding information is stored in Data Flash block when this property set to Enable. And the bonding information will be stored to Data Flash block which specified by **Data Flash Block for Security Data** option. Refer to section 4.3.1 for details.
Data Flash Block for Security Data Default: 0	Specify Data Flash block which stores the bonding information. Range: 0 to 7 Choose a different block from **Device Specific Data** Refer to section 4.3.1 for details.

Configuration options	
Remote Device Bonding Number Default: 7	Maximum number of the bonding information stored in the Data Flash. Range : 1 to 7 This value should be set same as **Maximum number of connections**. Refer to section 4.3.1 for details.
Connection Event Start Notify Default: Disable notify	Enable or disable start connection event start interrupt notification. Range: Disable notify or Enable notify This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).
Connection Event Close Notify Default: Disable notify	Enable or disable close connection event interrupt notification. Range: Disable notify or Enable notify This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).
Advertising Event Start Notify Default: Disable notify	Enable or disable start advertising event interrupt notification. Range: Disable notify or Enable notify The notification event occurs at the following timings. - Start Primary Advertising channel. - Start Secondary Advertising Channel - Start Periodic Advertising. (When Extended Advertising is enabled.) This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).
Advertising Event Close Notify Default: Disable notify	Enable or disable close advertising event interrupt notification. Range: Disable notify or Enable notify The notification occurs at the following timings. - Complete Primary Advertising channel. - Complete Secondary Advertising Channel - Complete Periodic Advertising. (When the Extended Advertising is enabled.) This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).
Scanning Event Start Notify Default: Disable notify	Enable or disable start scan interrupt notification. Range: Disable notify or Enable notify This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).

Configuration options	
Scanning Event Close Notify Default: Disable notify	Enable or disable close scan interrupt notification. Range: Disable notify or Enable notify This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).
Initiating Event Start Notify Default: Disable notify	Enable or disable notification that scan start interrupt has occurred in sending a connection request. Range: Disable notify or enable notify This notification will not occur when scan interval and scan window is equal. This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).
Initiating Event Close Notify Default: Disable notify	Enable or disable notification that scan complete interrupt has occurred in sending a connection request. Range: Disable notify or enable notify This notification will not occur when scan interval and scan window is equal. This notification event occurs after actual RF event because this notification event is triggered by the interrupt from BLE(H/W).
RF Deep Sleep Start Notify Default: Disable notify	Enable or disable notification event when BLE(H/W) enter deep sleep. Range: Disable notify or enable notify
RF Deep Sleep Wakeup Notify Default: Disable notify	Enable or disable notification event when BLE(H/W) wake up from deep sleep. Range: Disable notify or enable notify
Bluetooth dedicated clock Default: 6	Load capacitance adjustment value for 32MHz BLE dedicated crystal. Adjust this value so that the crystal oscillates at the frequency closest to 32MHz. Range: 0 to 15 Refer to "Tuning procedure of Bluetooth dedicated clock frequency(R01AN4887)" for details.
DC-DC Converter Default: Disable DC-DC Converter	Enable or disable the DC-DC on BLE(H/W). Range: Disable DC-DC Converter or Enable DC-DC Converter. Refer to “RA4W1 Group User’s Manual: Hardware (R01UH0883)" for details.
Slow Clock Source Default: Use RF_LOCO	Slow clock source for BLE (H/W) Range: Use RF_LOCO or Use External 32.768kHz. Do NOT change.

Configuration options	
MCU CLKOUT Port Default: P109	Port of the MCU CLKOUT. Range: P109 or P205 This option will be ignored if the Slow Clock Source option is Use RF_LOCO.
MCU CLKOUT Frequency Output Default: MCU CLKOUT Frequency 32.768kHz	Output frequency from the MCU CLKOUT Port. Range : MCU CLKOUT frequency 32.768kHz or MCU CLKOUT frequency 16.384kHz This option will be ignored if the Slow Clock Source option is Use RF_LOCO.
Sleep Clock Accuracy (SCA) Default: 250	Clock Accuracy (SCA) of Slow clock source for BLE(H/W). Range: 0 to 500 ppm Value of this option is fixed to more than 250ppm when Slow Clock Source option is Use RF_LOCO.
Transmission Power Maximum Value Default: max +4dBm	Maximum transmit power configuration. Range: max +4dBm or max 0dBm.
Transmission Power Default Value Default: High	Actual BLE air packet transmit power. Range: High or Mid or Low This option depends on the **Transmission Power Maximum Value** configuration. If the **transmission Power Maximum Value** option set to 0dBm, relationship this option and Actual BLE air packet transmit power is as follows. High : 0dBm Mid : 0dBm (same as High) Low : -18dBm If the **transmission Power Maximum Value** option set to +4dBm, relationship this option and Actual BLE air packet transmit power is as follows. High : +4dBm Mid : 0dBm Low : -20dBm
CLKOUT_RF Output Default: No output	Specify CLKOUT_RF(P414) output frequency. Range: No output 4MHz output 2MHz output 1MHz output
RF_DEEP_SLEEP Transition Default: Enable	Enable or disable BLE(H/W) Deep Sleep. Range: Disable or Enable

Configuration options	
MCU Main Clock Frequency Default: 8000	MCU main clock frequency (kHz). This option needs to be configured according to System Clock Source configuration. If the HOCO is used as System Clock Source, this option is ignored. If the Main Clock is used as System Clock Source, set a value within the range between 1000 and 20000. If the PLL Circuit is used as System Clock Source, set a value within the range between 4000 and 12500. Set clock frequency according to user's system clock source configuration.
Code Flash (ROM) Device Data Block Default: 255	Specify Code Flash (ROM) block which stored the device specific data (e.g. BD address, etc.) Range: -1 to 255 If this option is set to -1, Code Flash will not use for this purpose. Refer to section 4.2 for details.
Device Specific Data Flash Block Default: -1	Specify Data Flash (RAM) block which stored the device specific data (e.g. BD address, etc.) Range: -1 to 7 If this option is set to -1, Data Flash will not use for this purpose. Specify a different block from Data Flash Block for Security Data. Refer to section 4.2 for details.
MTU size configured Default: 247	MTU size (bytes) for the GATT communication. Range: 23 to 247
Timer Slot Maximum Number Default: 10	N/A Do NOT change.
Synchronization Type **Default : Event groups**	This property is available for FreeRTOS only. The property is specified task synchronization method in FreeRTOS environment. Also refer to section 1.2. Range: Event groups or Semaphore
Parameter Checking Default: Default (BSP)	Enable or disable the validity check of the parameters for BLE module. Range: Default (BSP) or Enabled or Disabled

When **RF_DEEP_SLEEP Transition** option is set to enable, when there is no task to be executed by the BLE protocol stack, and when there is a time of 80ms or more before the start of the next RF event time, transition to RF sleep mode to reduce the current consumption of the RF part. This time does not mean the "interval time" of an RF event, but the "RF idle time" between the completion of one RF event and the start of the next RF event. Therefore, it is necessary to set the RF event interval to 100ms or more in consideration of the processing time of each layer in order to shift the RF part to sleep mode. The BLE protocol stack performs RF sleep processing and RF wake-up processing to transition the RF part to sleep mode. Figure 50 shows MCU/RF operation overview with RF sleep.

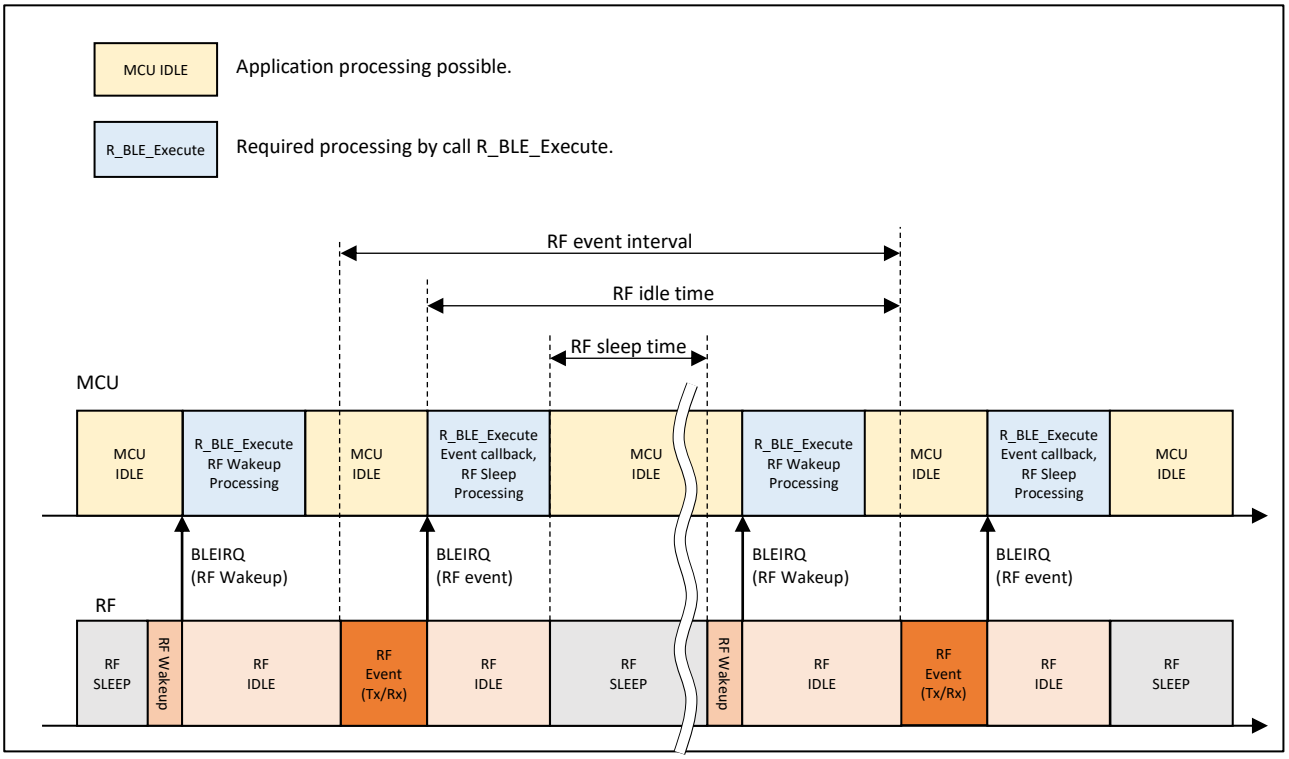

Figure 50. MCU/RF operation overview with RF sleep

While the MCU is idle, it is possible to transition the MCU to the low power consumption mode or execute application processing. However, if the RF wakeup process by *R_BLE_Execute* is not performed before the RF event starts, the RF event cannot be executed. Therefore, application processing must be implemented so as not to interfere with the *R_BLE_Execute* call.

When **RF_DEEP_SLEEP Transition** option is set to disable, or when **RF_DEEP_SLEEP Transition** option is set to enable but the RF sleep transition condition is not satisfied (e.g. RF event interval < 100 msec), the BLE protocol stack does not transition RF part to sleep mode. In this case, the current consumption during RF idle time increases, but the MCU idle time that can be used by the application increases because RF sleep processing and RF wakeup processing are not performed. Figure 51 shows MCU/RF operation without RF sleep.

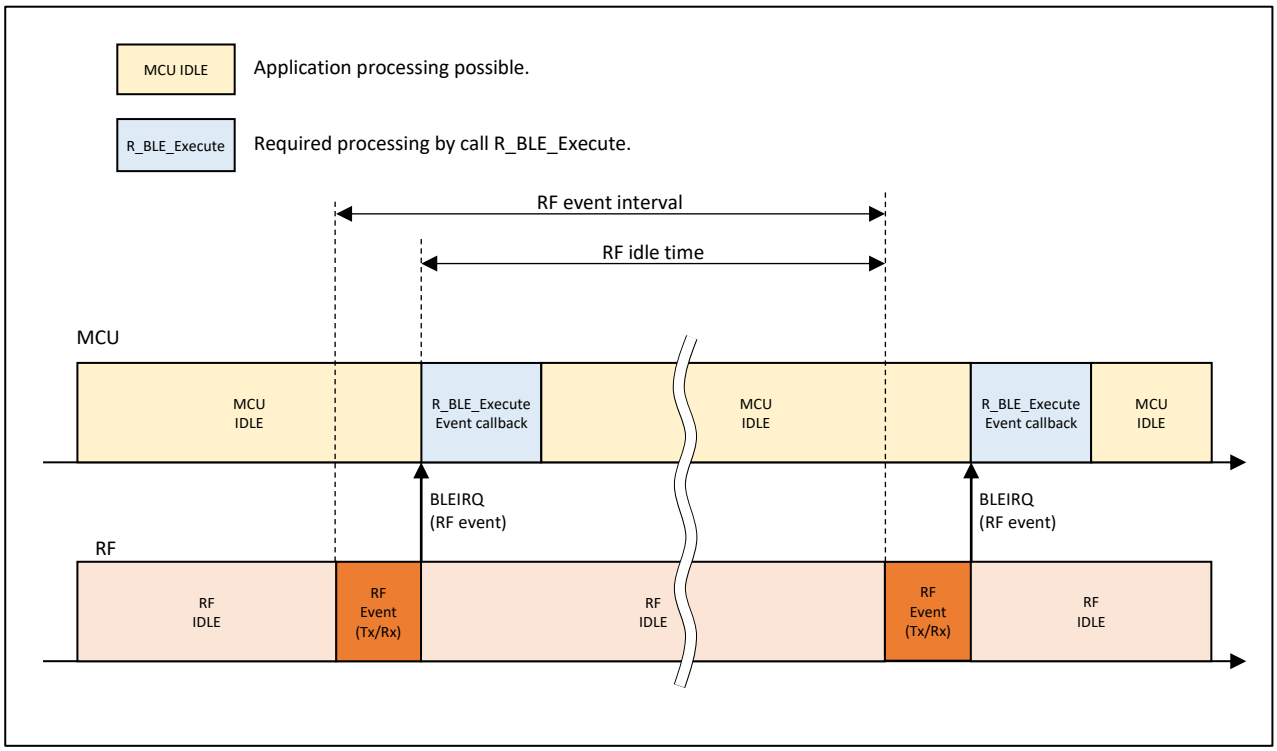

Figure 51. MCU/RF operation overview without RF sleep

Regardless of the RF sleep state, if the application process continuously occupies the MCU and *R_BLE_Execute* is not called, the connection may not be maintained. Therefore, it is recommended that the application processing is active for a short time or Task performing R_BLE_Execute is given an appropriate priority to allow periodic execution.

2. BLE Abstraction Driver on rm_ble_abs options

The BLE module can change IO capability on local device etc. by modifying Module **BLE Abstraction Driver on rm_ble_abs** options on FSP configuration. The changed options are automatically reflected to the rm_ble_abs_cfg.h when generating code.

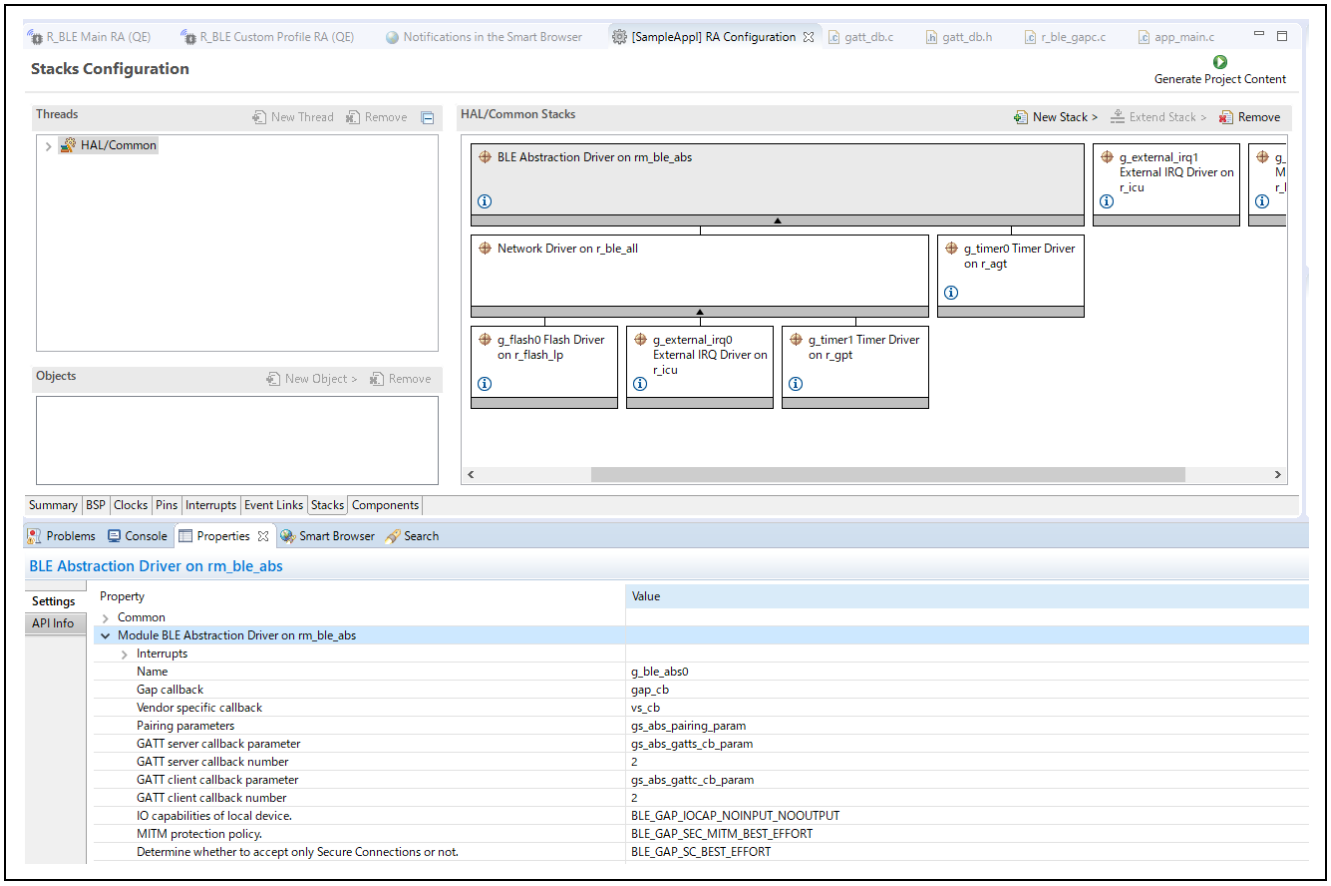

Figure 52. BLE Abstraction Driver on rm_ble_abs options

Option names and setting values in the configuration are listed following. Items shown in **bold may be modified** according to user's environment.

Table 22. Module BLE Abstraction Driver on rm_ble_abs options

Configuration options	
Interrupts > Callback provided when an ISR occurs Default: NULL	Do NOT change.
Name Default: g_ble_abs0	Do NOT change.
Gap callback Default: gap_cb	Do NOT change.

Configuration options	
Vendor specific callback Default: vs_cb	Do NOT change.
Pairing parameters Default: gs_abs_pairing_param	Do NOT change.
GATT server callback parameter Default: gs_abs_gatts_cb_param	Do NOT change.
GATT server callback number Default: 2	Do NOT change.
GATT client callback parameter Default: gs_abs_gattc_cb_param	Do NOT change.
GATT client callback number Default: 2	Do NOT change.
IO capabilities of local device Default: BLE_GAP_IOCAP_NOINPUT_NOOUTPUT	IO capabilities. Range: Select one of the following. • **BLE_GAP_IOCAP_DISPLAY_ONLY** Output: local device has ability to display 6 digits decimal number. Input: None. • **BLE_GAP_IOCAP_DISPLAY_YESNO** Output: local device has ability to display 6 digits decimal number. Input: local device has ability to indicate ‘yes’ or ‘no’. • **BLE_GAP_IOCAP_KEYBOARD_ONLY** Output: None. Input: local device has ability to input the number ‘0’ – ‘9’. • **BLE_GAP_IOCAP_NOINPUT_NOOUTPUT** Output: None. Input: None. • **BLE_GAP_IOCAP_KEYBOARD_DISPLAY** Output: local device has ability to display 6 digits decimal number. Input: local device has ability to input the number ‘0’ – ‘9’.
MITM protection policy Default: BLE_GAP_SEC_MITM_BEST_EFFORT	MITM protection policy. Range: Select one of the following. • **BLE_GAP_SEC_MITM_BEST_EFFORT** MITM Protection not required. • **BLE_GAP_SEC_MITM_STRICT** MITM Protection required.

Configuration options	
Determine whether to accept only Secure Connections or not Default: BLE_GAP_SC_BEST_EFFORT	Determine whether to accept only Secure Connections or not. Range: Select one of the following. • **BLE_GAP_SC_BEST_EFFORT** Accept Legacy pairing and Secure Connections. • **BLE_GAP_SC_STRICT** Accept only Secure Connections.
Type of keys to be distributed from local device Default: BLE_GAP_KEY_DIST_ENCKEY	Type of keys to be distributed from local device. This field is set to a bitwise OR of the following values. • **BLE_GAP_KEY_DIST_ENCKEY** Distribute LTK. • **BLE_GAP_KEY_DIST_IDKEY** Distribute IRK and Identity address. • **BLE_GAP_KEY_DIST_SIGNKEY** Distribute CSRK.
Type of keys which local device requests a remote device to distribute Default: BLE_GAP_KEY_DIST_ENCKEY	Type of keys which local device requests a remote device to distribute. This field is set to a bitwise OR of the following values. • **BLE_GAP_KEY_DIST_ENCKEY** Distribute LTK. In case of Secure Connections, LTK is notified even if this bit is not set. • **BLE_GAP_KEY_DIST_IDKEY** Distribute IRK and Identity address. • **BLE_GAP_KEY_DIST_SIGNKEY** Distribute CSRK.
Maximum LTK size Default: 16	The maximum LTK size(byte) to be requested to a remote device. Range: 7 – 16 When the LTK size of a remote device is less than this configuration size, the pairing fails.

4.1.4.5 Add and configure related peripherals for BLE module

BLE module used below following peripherals to perform BLE communication.

Table 23. Related peripherals

Item	Usage
Flash Driver on r_flash_lp	Store Bonding information etc.
External IRQ driver on r_icu	Interrupt from BLE(H/W)
GPT Driver	Timer for BLE protocol stack
Timer Driver	Timer for BLE abstraction API

This section describes how to configure related peripherals (timers, interrupt) for BLE module which added previous section. Procedure describes in this section is common to BareMetal, FreeRTOS and Azure RTOS environment.

1. Click **Add GPT Driver** box and select **New→Timer Driver on r_gpt**.

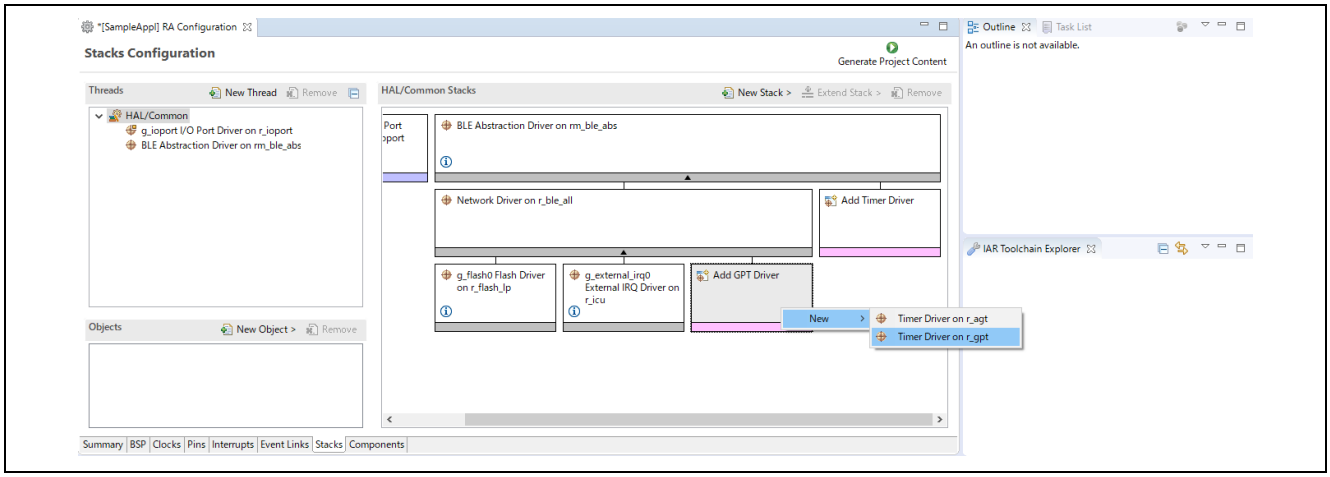

Figure 53. Add GPT Driver

2. Set **Overflow/Crest Interrupt Priority** of **g_timer0 Timer Driver on r_gpt** as **Priority 2** on **Properties** tab.

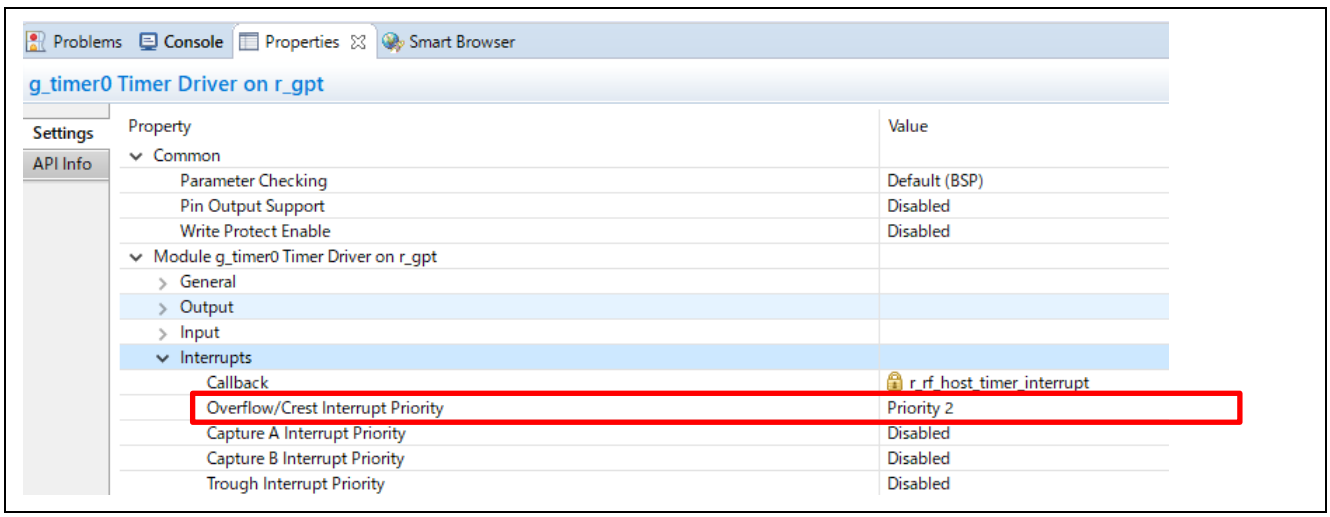

Figure 54. GPT Driver configuration

3. Click **Add Timer Driver** box and select **New→Timer Driver on r_agt**.

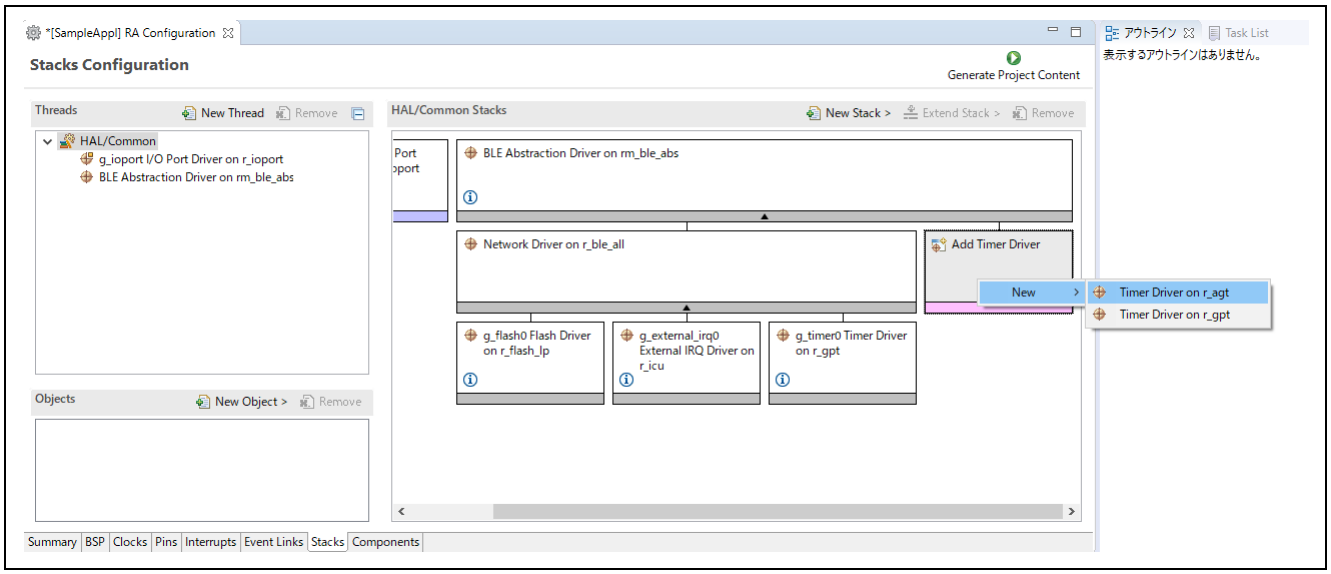

Figure 55. Add AGT Driver

4. Set **Underflow Interrupt Priority** of **g_timer1 Timer Driver on r_agt** as **Priority 7** on **Properties** tab.

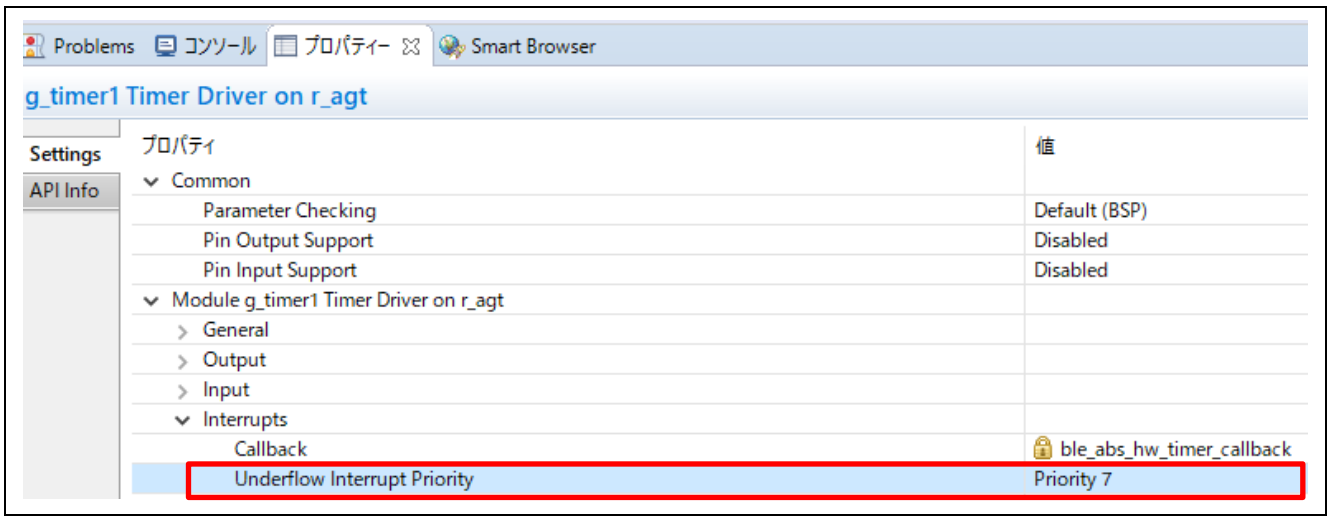

Figure 56. AGT Driver configuration

5. Set **Pin Interrupt Priority of g_external_irq0 External IRQ Driver** on **r_icu** as,

- **BareMetal environment**
  Priority 0 on Priority.

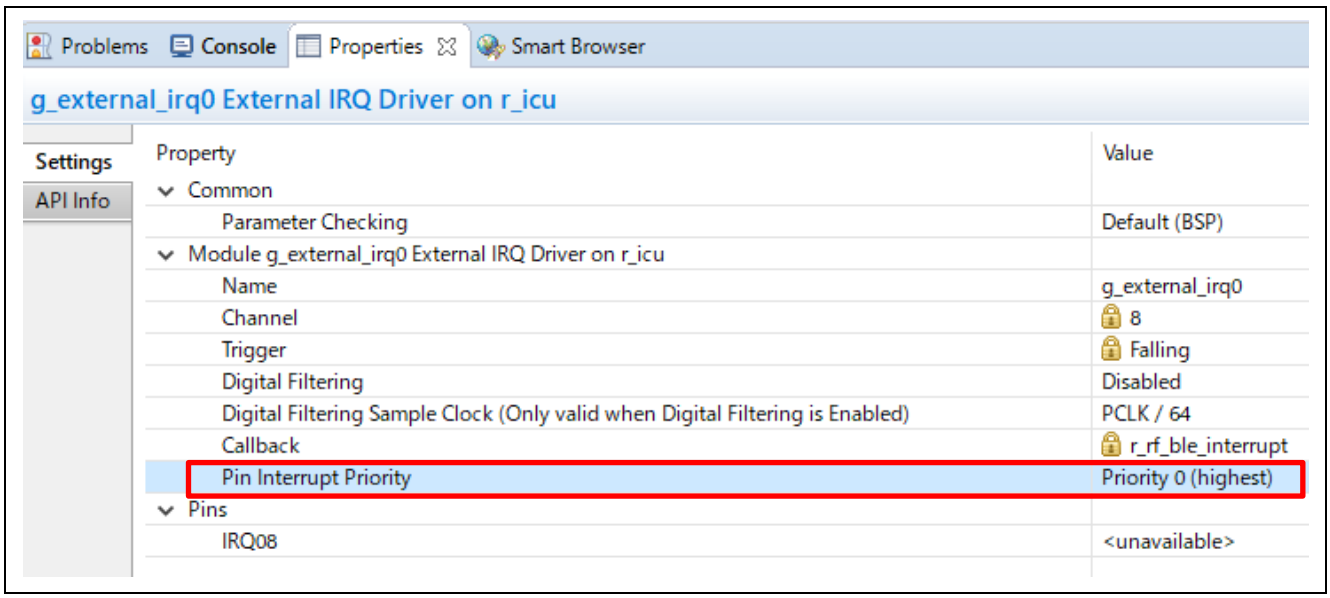

Figure 57. ICU Driver configuration (BareMetal Environment)

- **FreeRTOS and Azure RTOS environment**
  Priority 1 on Priority. Because the highest priority used FreeRTOS and Azure RTOS kernel.

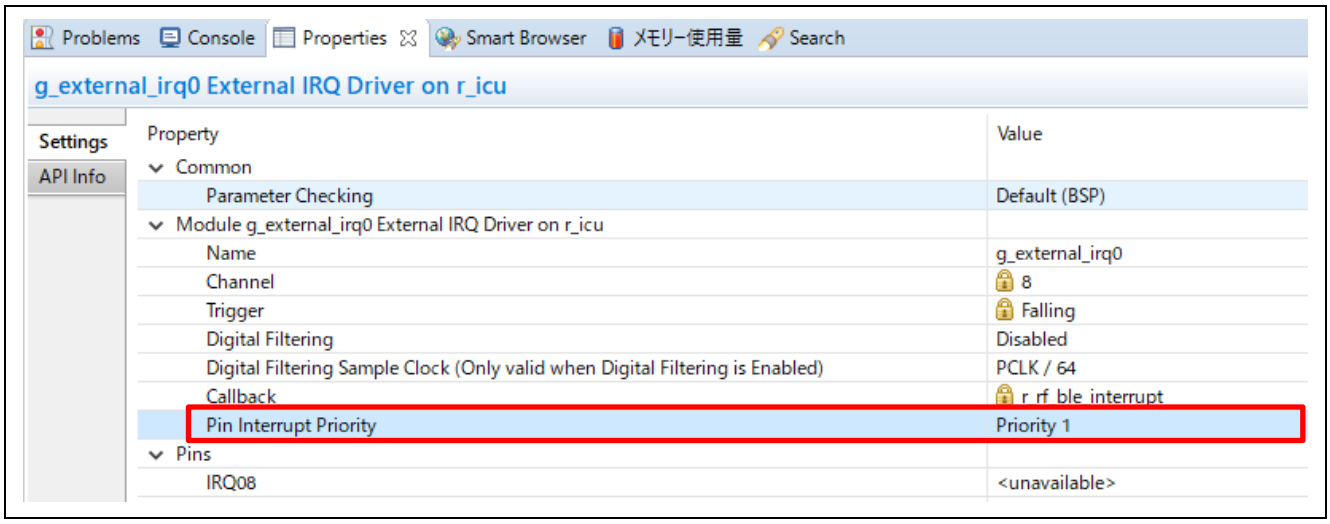

Figure 58. ICU Driver configuration (FreeRTOS and Azure RTOS Environment)

4.1.5 Low Power Mode

Software standby mode, which is one of MCU's Low Power mode feature, can be used to reduce power consumption. It is necessary to add *Low Power Mode (r_lpm)* module to your project and configure *BLEIRQ* as Wake Sources.

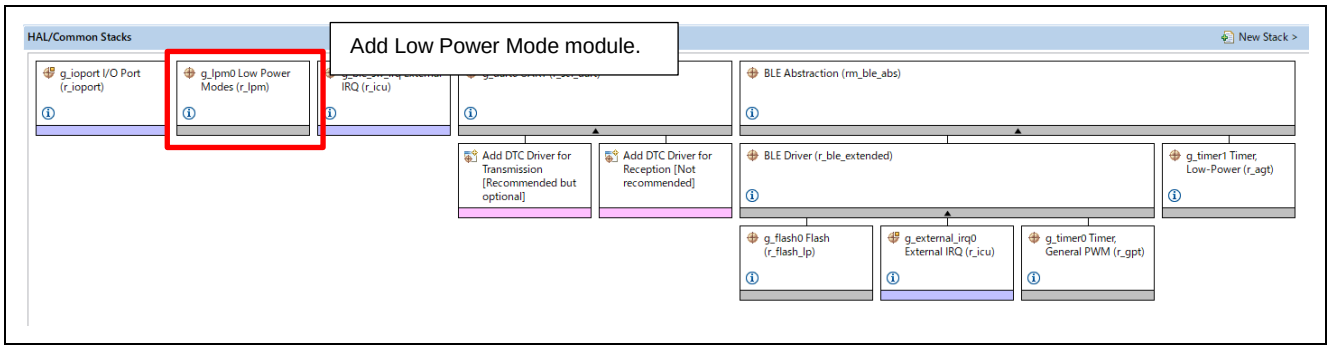

Figure 59. Add Low Power Mode

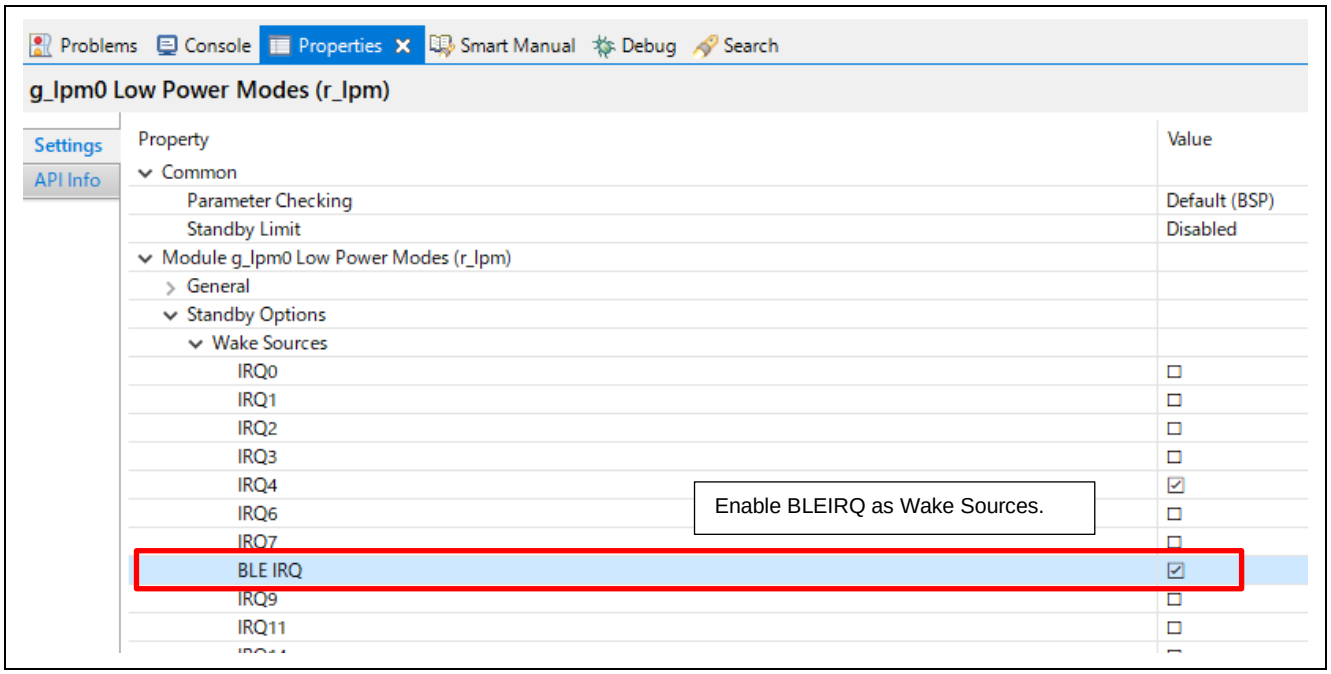

Figure 60. Wake Sources

Refer to section 3.1.2 for how to use the low power mode in your application.

4.1.6 Make profile and BLE application skeleton code

QE for BLE can generate profile and BLE application skeleton code. And user can modify these codes according to use case. Refer to *Bluetooth Low Energy Profile Developer's Guide(R01AN5428)* about usage of QE for BLE.

4.2 Device-specific Data Management

Bluetooth Device Address (hereinafter referred to as BD address) used by BLE Protocol Stack can be written to Data Flash area and Code Flash area as device-specific data. This allows user to set different BD address for multiple devices using the same firmware. Device-specific data is placed in a different area from the firmware program area. If the device-specific data is not deleted when rewriting the firmware, the same BD address can be used continuously.

4.2.1 Specify device-specific data location block

1. Data flash area

The block number of data flash area where device-specific data is located can be specified with **Data Flash (RAM) Device Data Block** configuration options. Relationship between block number and address is following.

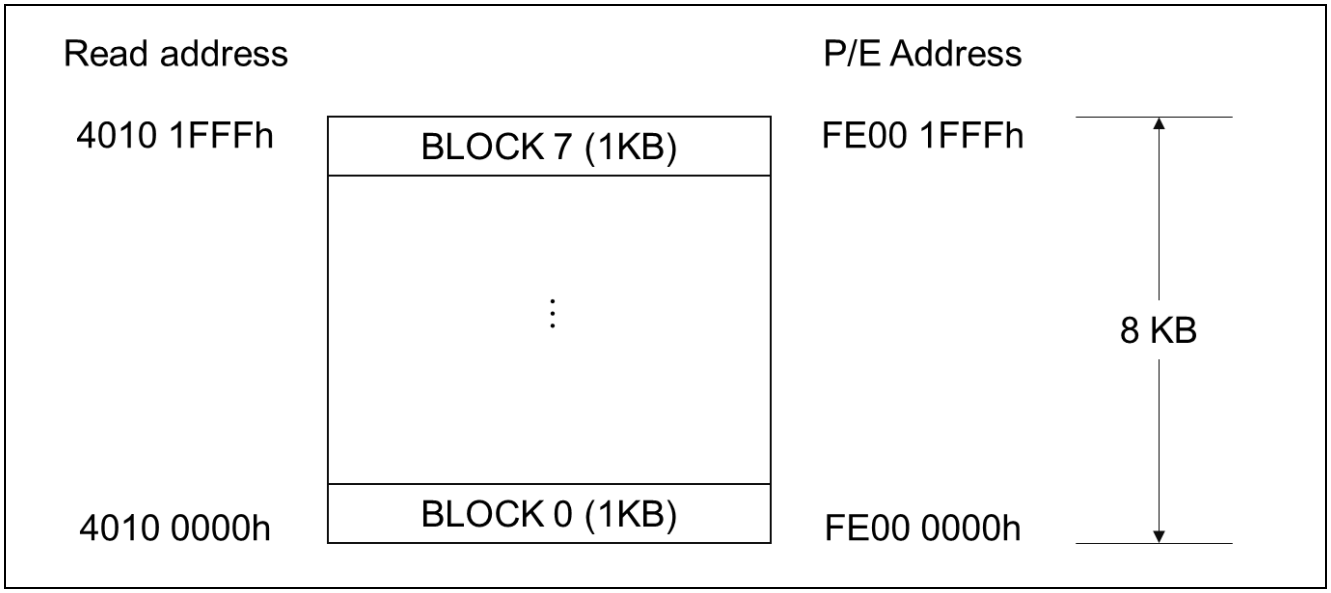

Figure 61. Data flash memory block configuration

Device-specific data is written at the top of the block specified by Device Specific Data Flash Block option. Do not write other data to the block where device-specific data is placed.

2. Code flash area

The block number of code flash where device-specific data is located can be specified with **Code Flash (ROM) Device Data Block** configuration options. Relationship between block number and address is following.

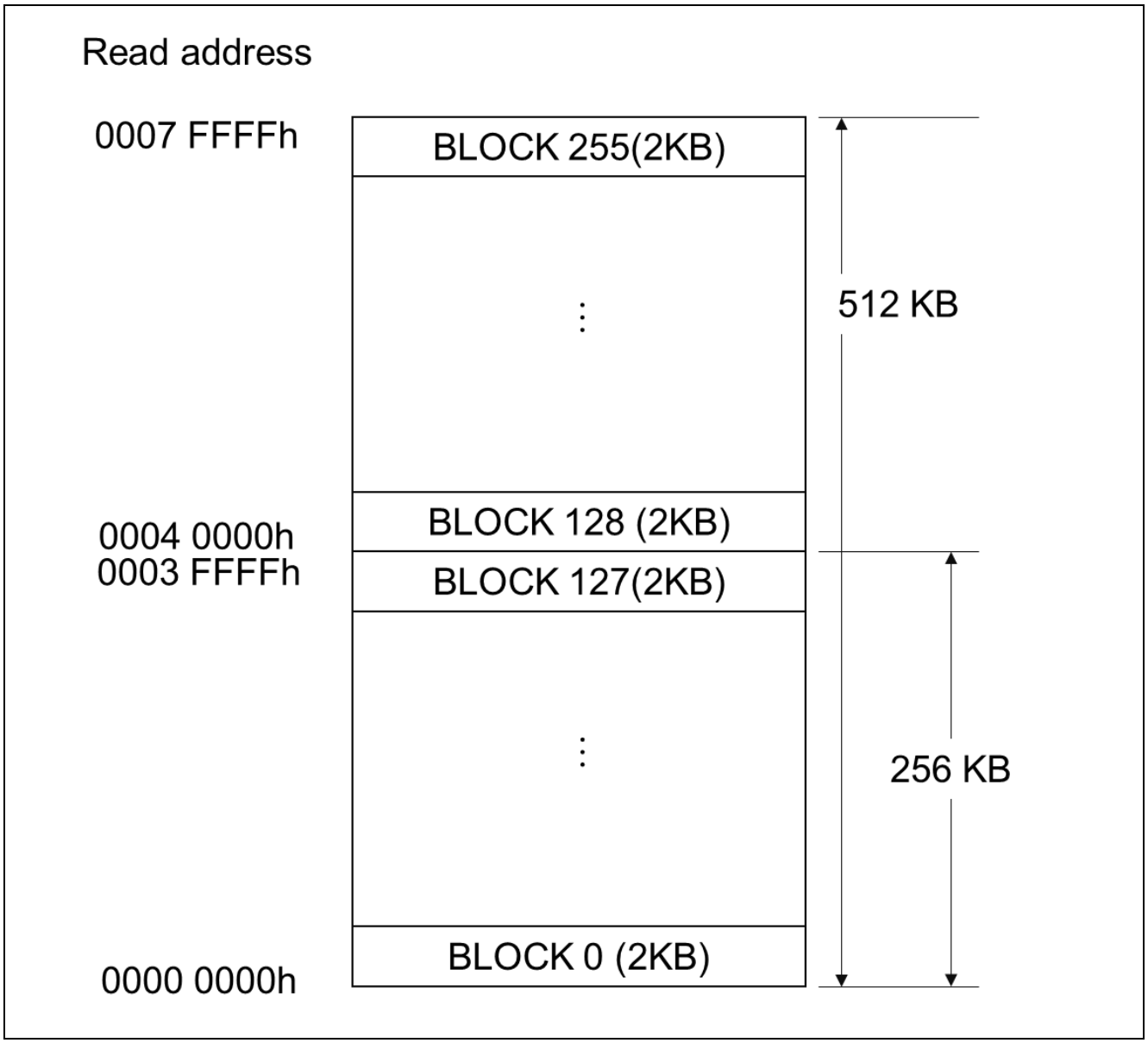

Figure 62. Code flash memory block configuration

When placing device-specific data in code flash area, it is necessary to specify blocks that are not used in program code. In addition, device-specific data is written at the top of the block specified by **Code Flash (ROM) Device Data Block** configuration.

4.2.2 Device-specific data format

Table 24 shows the device-specific data format.

Table 24. device-specific data format

Offset	Size[bytes]	Type	Description
0	4	uint32_t	Data length after magic number (fixed to 0x00000010)
4	4	uint32_t	Magic number (fixed 0x12345678)
8	6	uint8_t [6]	Public BD address
14	6	uint8_t [6]	Random BD address

Each data must be written in little endian. For example, if BD address is “01:02:03:04:05:06”, write to the flash memory in the order of 0x06,0x05,0x04,0x03,0x02,0x01. Figure 63 shows an example of device-specific data flash memory layout.

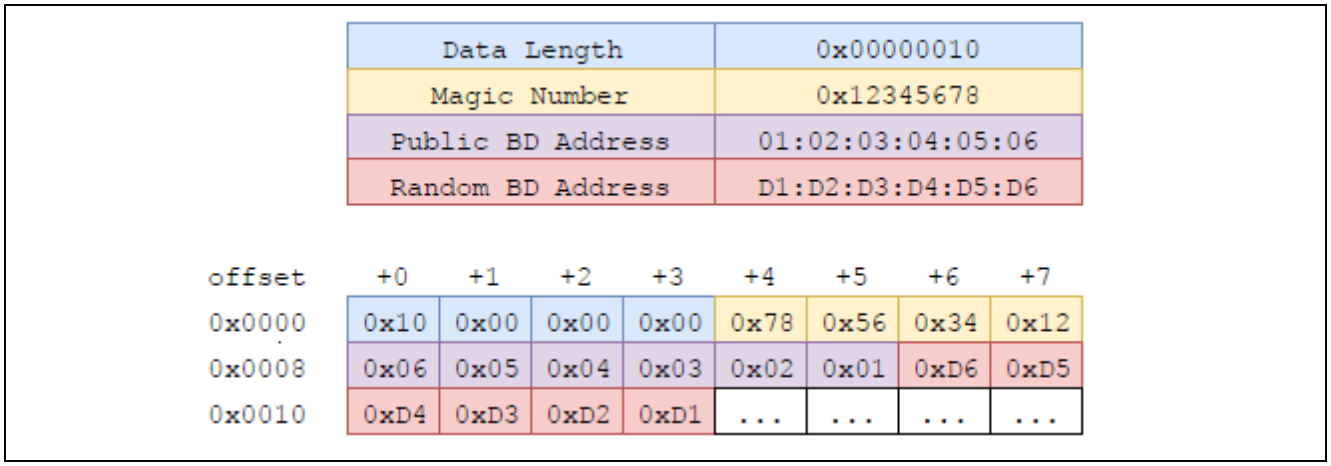

Figure 63. Device-specific data flash memory layout

4.2.3 How to write device-specific data

User can write device specific data by following way. When device-specific data is written to the data flash area, the written BD address is adopted after reset of MCU.

1. Write to data flash area using R_BLE API

Use *R_BLE_VS_SetBdAddr()* API to write device-specific data to data flash area. When device-specific data is written to the data flash area, BD address written by reboot once RA4W1 is adopt. Refer to “Renesas Flexible Software Package User’s Manual” for details of the API.

2. Write to data area using BD address writing tool

User can write Public BD address to data area by using Public BD address writing tool for the RA4W1 device with HCI mode firmware. Refer to “Host Controller Interface Firmware(R01AN5429)” and “Public BD Address writing tool(R01AN5439)”.

Note: The BD address writing tool does not support Random BD address writing.

3. Write to code flash area

To write device-specific data to code flash area, use Renesas Flash Programmer V3.1.0 (RFP) unique code Function. The unique code function is functionality to write the device specific data to user area at the same time as firmware program data. Refer to Renesas Flash Programmer User's manual (R01UT5757) about usage of RFP.

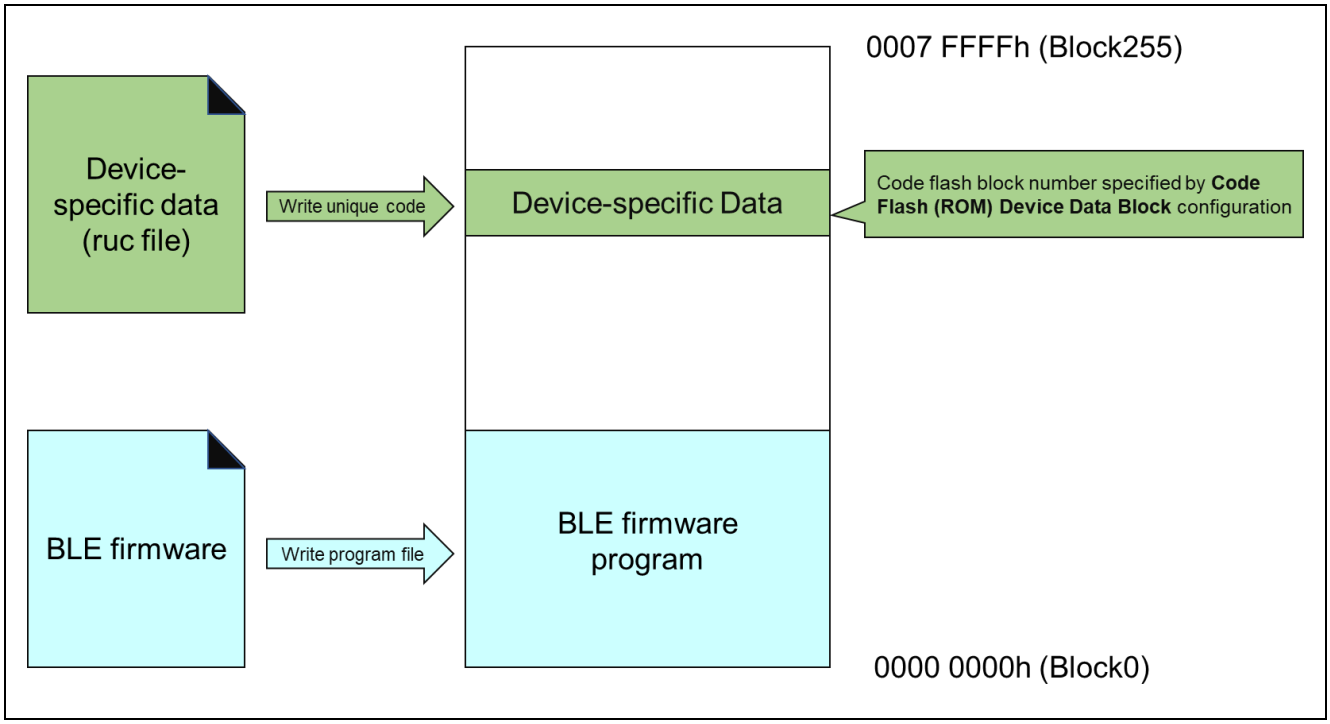

Figure 64. writing device-specific data by using RFP

Code 59 shown an example of setting device-specific data for RFP unique code (*.ruc) file.

```
format hex
address 0x0007F800
size 20
index data
000001 10000000 78563412 060504030201 D6D5D4D3D2D1
```

ROM address of the block specified by **Code Flash (ROM) Device Data Block** Configuration.

Index

Data Length (Fixed)

Magic Number (Fixed)

Public BD Address (User can define)

Random BD Address (User can define)

Code 59. Setting device-specific data for RFP unique code

This code writes the following configurations at the top of block 255 in code flash area.

- Public address : 0x01:02:03:04:05:06
- Random address : 0xD1:D2:D3:D4:D5:D6

4.2.4 BD address adoption flow

BLE Protocol Stack adopts initial value of BD address in following priority order in *RM_BLE_ABS_Open()* API.

(1) Data flash specified block

(2) Code flash specified block

(3) Firmware initial value (**Debug Public Address** or **Debug Random Address** configuration)

For Random BD address, if BD addresses for all areas are not specified, static address is generated from Unique ID of MCU. Generated static address can be obtained with the *R_BLE_VS_GetBdAddr()* API.

Even after BD address is adopted, the BD address can be changed again with *R_BLE_VS_SetBdAddr()* API.

Note: The generated static address is a fixed value that does not change when the MCU power off or reset.

Note: A static address consists of random numbers. The possibility of duplicate values with other devices is near zero.

Figure 65 shows BD address adoption flow of BLE Protocol Stack.

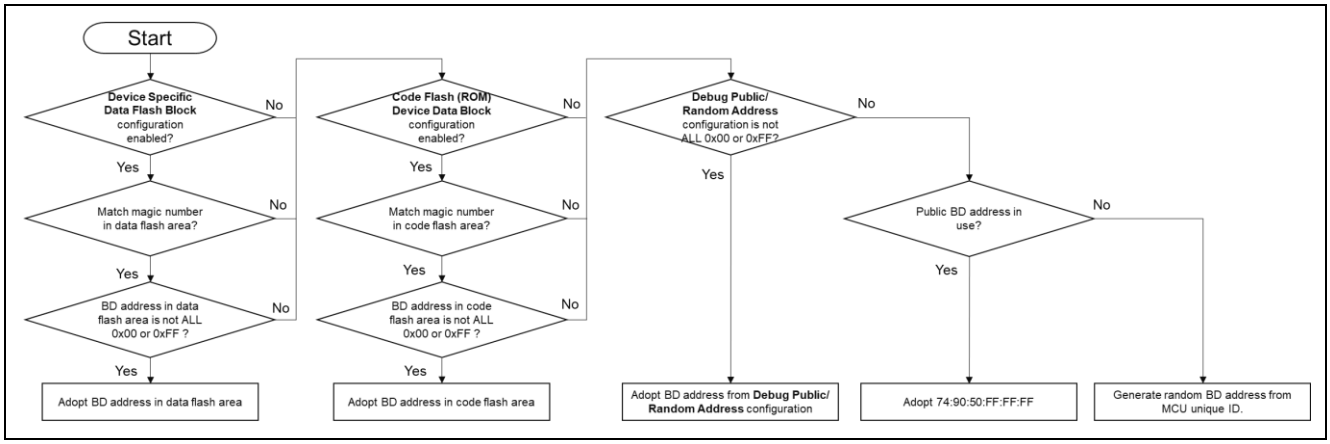

Figure 65. BD address adoption flow of BLE Protocol Stack

Since BLE Protocol Stack does not check format of BD address written in each area (1)-(3), when setting static address, set value that matches the format shown in Figure 66.

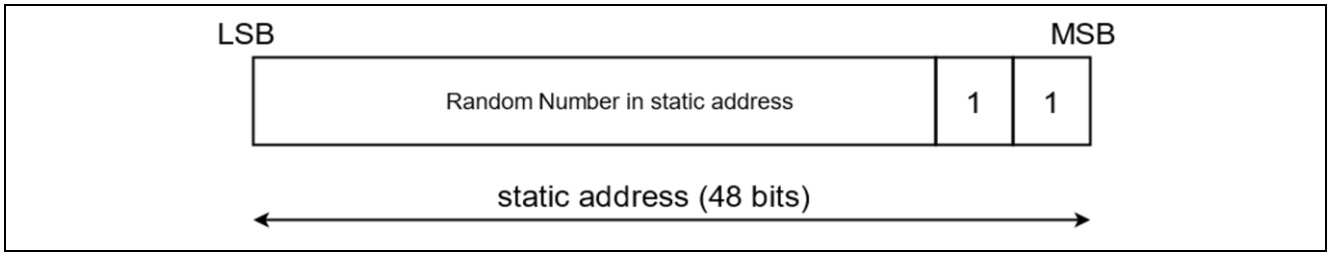

Figure 66. Static address format

4.3 Security Data Management

The security data management function read / write the following data in the data flash area.

- Local device key to distribute during pairing
- Key and information obtained from the remote device during pairing

The local device key and remote device key storage in the data flash is configurable in the BLE Protocol Stack using the security data management API. The Abstraction API uses the security data management API to manage security data for local and remote devices. The security data management function is set using the configuration options shown in Table 25.

Table 25. Security data management configuration options

Configuration Options	Description
Store Security Data Default: Disable	Enable or disable the security data management. Range: Enable or Disable Bonding information is stored in Data Flash block when this configuration set to Enable. And the bonding information will store Data Flash block which specified by **Data Flash Block for Security Data** option.
Data Flash Block for Security Data Default: 0	Specify Data Flash block which store the bonding information. Range: 0 to 7 Choose a different block from **Device Specific Data** Refer to section 4.3.1 for details.
Remote device bonding number Default: 7	Maximum number of the bonding information stored in the Data Flash. Range : 1 to 7 This value should be set same as **Maximum number of connections**. Refer to section 4.3.1 for details.

The security data management function manages security data management information, local device security data, and remote device security data. The memory map in the data flash is as shown in Figure 67.

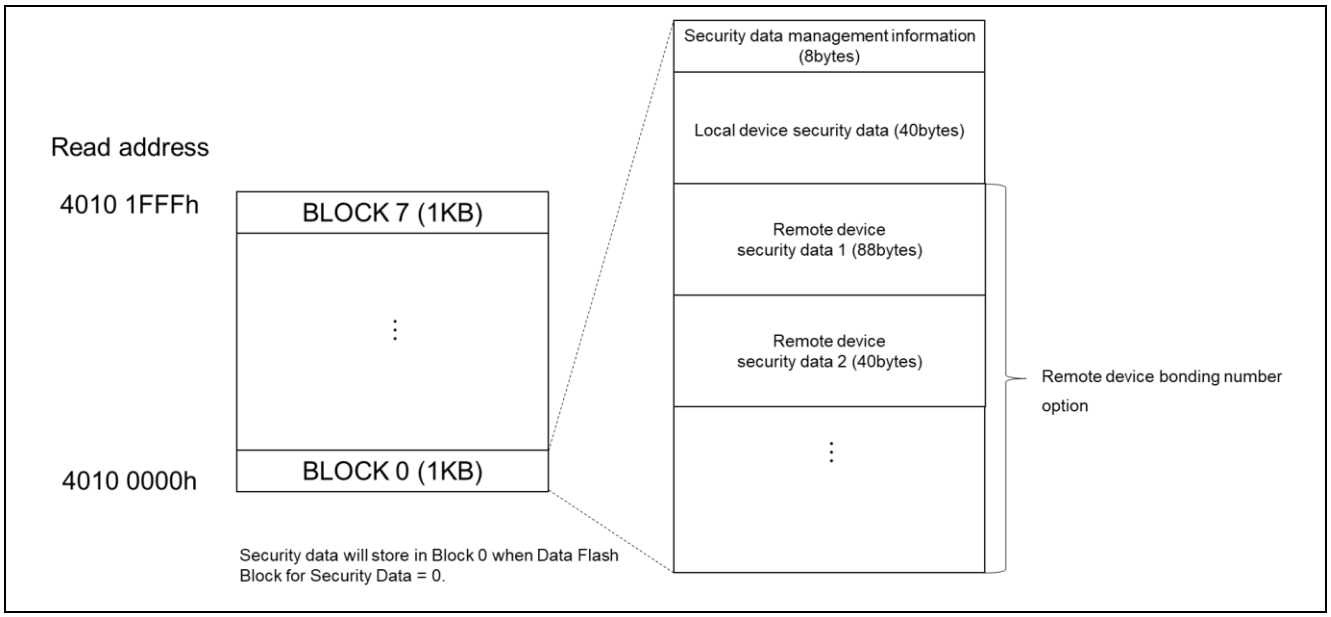

Figure 67. Memory map of security data in data flash

Each data information is described in following section.

4.3.1 Security data management information

The structure and structure elements of security data management information are shown in Figure 68 and Table 26. This data is handled internally by the security data management function and does not need to be updated by the user application.

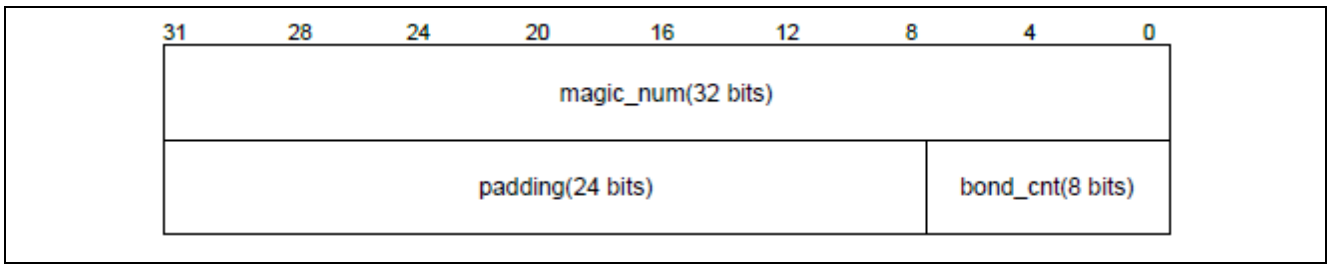

Figure 68. Security data management information structure

Table 26. Security management information structure elements

Type	Element Name	size [bytes]	Description
uint32_t	magic_num	4	Magic number of security data. Check whether security data is written. Fixed to 0x12345678. 0xFFFFFFFF when not written.
uint8_t	bond_cnt	1	Number of bonding information stored.
uint8_t	padding[3]	3	Padding

4.3.2 Local device security data

The security data structure and structure elements of the local device are shown in Figure 69 and Table 27.

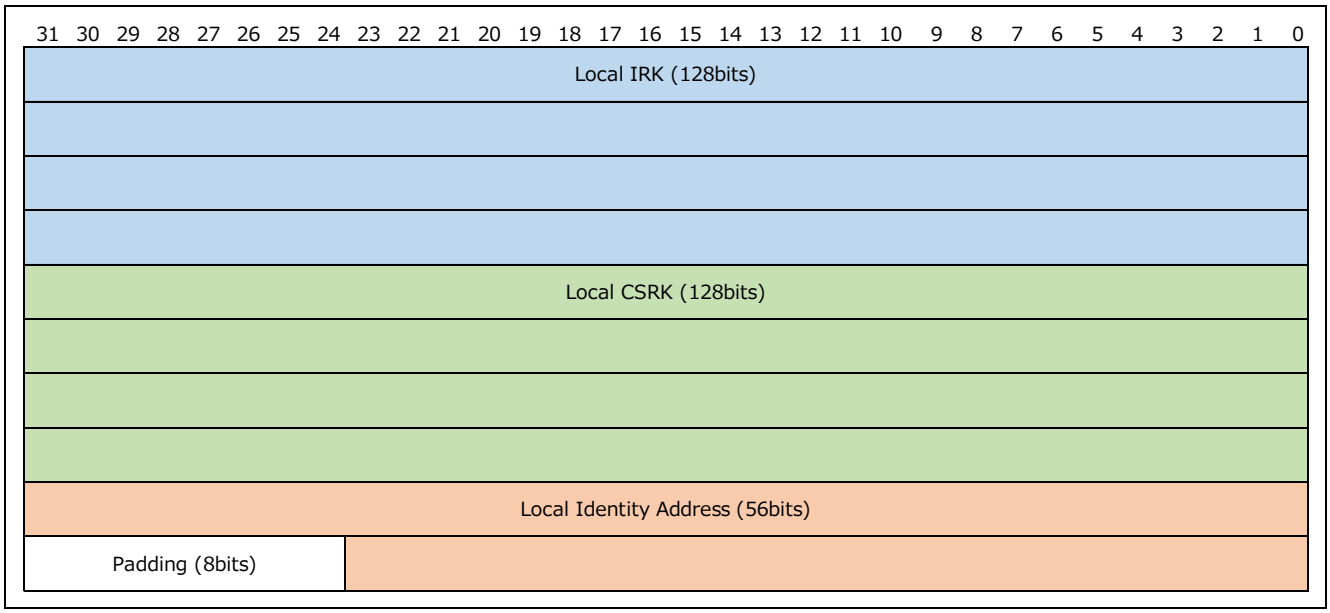

Figure 69. Local device security data structure

Table 27. Local device security data structure elements

Element Name	size [bytes]	Description
Local Identity Resolving Key (IRK)	16	IRK distributed to remote devices during pairing. Resolvable Private Address (RPA) is used when generating by Privacy feature.
Local Connection Signature Resolving Key (CSRK)	16	CSRK distributed to remote devices during pairing. Used when sending with signed data.
Local Identity Address	7	The local device identity address that informs the remote device during pairing.
Padding	1	Padding

The following describes security data settings for local devices.

- IRK and CSRK generate and set a 16-byte random number.
- To set BLE Protocol Stack, use *R_BLE_GAP_SetLocIdInfo()* (IRK, Identity Address) and *R_BLE_GAP_SetLocCsrk()* (CSRK).

By using API of security data management function, the generated security data can be written to data flash. It is possible to reconfigure to BLE Protocol Stack after reboot device. Figure 70 shows an example of local device security data setting processing that is performed when the BLE Protocol Stack is started.

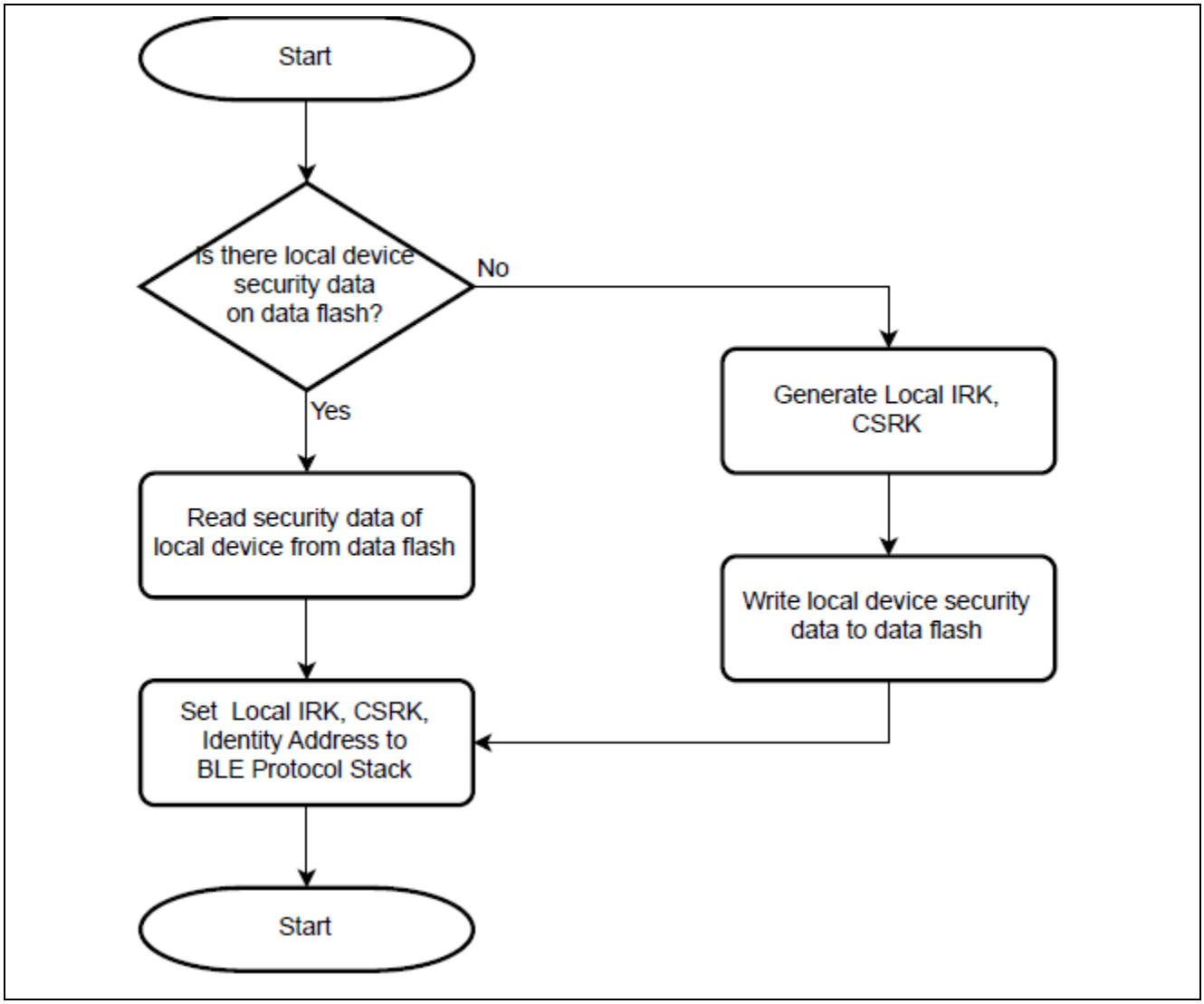

Figure 70. Example of setting local device security data

4.3.3 Remote device security data

The structure and structural elements of the remote device security data are shown in Figure 71 and Table 28.

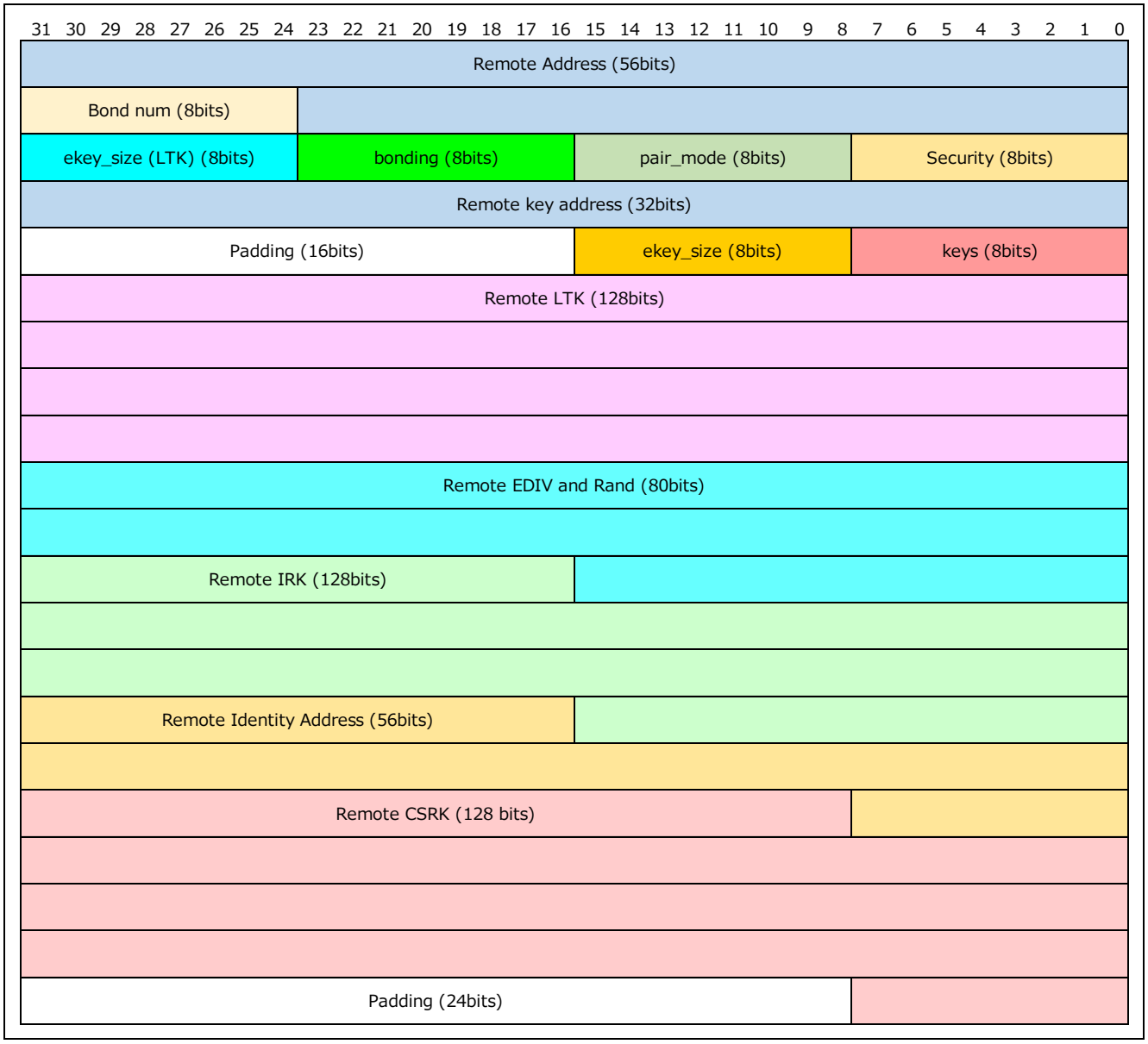

Figure 71. Remote device security data structure

Table 28. Remote device security data structure elements

Element Name	size [bytes]	Description
Remote Address	7	BD address used by remote device during pairing.
Bond num	1	Bonding serial number.
security	1	Security level of the pairing performed. 0x01: Perform pairing with Unauthenticated pairing. 0x02: Perform pairing with Authenticated pairing
pair_mode	1	Type of pairing performed. 0x01: Perform pairing with Legacy Pairing. 0x02: Perform pairing with Secure Connections.
bonding	1	Bonding policy of remote device. 0x00: Indicates that remote device does not bonding performed. 0x01 Indicates that remote device is bonding performed.
ekey_size (LTK)	1	Size of LTK.
remote key address	4	Start address of the data flash to store the remote device key (Remote LTK to Remote CSRK).
keys	1	Type of key distributed by remote device.
ekey_size	1	Negotiated LTK size.
Remote LTK	16	LTK distributed by remote device. Used for connection encryption.
Remote EDIV and Rand	10	EDIV and Random number distributed by the remote device. Used for connection encryption.
Remote IRK	16	IRK distributed by remote device. Used for address resolution when the remote device uses the privacy feature.
Remote Identity Address	7	Identity address of remote device. Used for address resolution when remote device uses the privacy feature.
Remote CSRK	16	CSRK distributed by remote device. Used when receiving signed data.

The following describes security data settings for remote devices.

- The remote device security data is received during pairing.
- security, pair_mode, bonding, and ekey_size in Table 28 are written to data flash at the BLE_GAP_EVENT_PAIRING_COMP event. Other data is written to the data flash at the BLE_GAP_EVENT_PEER_KEY_INFO event.
- Initializing process at the BLE_GAP_EVENT_STACK_ON event reads the remote device security data from data flash and calls *R_BLE_GAP_SetBondInfo()* to set remote device security data in the BLE Protocol Stack.
- If number of data written to data flash exceeds number specified by **Remote device bonding number** option, oldest security data entry is overwritten.

By using security data management function, the received remote device security data during pairing can be written to data flash. It is possible to reconfigure to BLE Protocol Stack after reboot device.

4.4 Data Flash Block

If your application holds data in Data Flash, use the block except the following.

- Data Flash (RAM) Device Data Block
- Data Flash Block for Security Data

4.5 Importing CLI (Command Line Interface) to user's project

4.5.1 Related source files

Related source files to CLI are installed under app_lib in this demo project. User can add the CLI functionality by copying / adding path app_lib directory from this demo project to their own project.

4.5.2 Configurations of SCI

Please open FSP configuration of user's project and select **Stacks** tab. Add **New Stack → Driver → Connectivity → UART Driver on r_sci_uart** to **HAL/Common**. And modify configuration of the added **r_sci_uart** as following.

- [Interrupts]→[Callback] : user_uart_callback_ble_cli

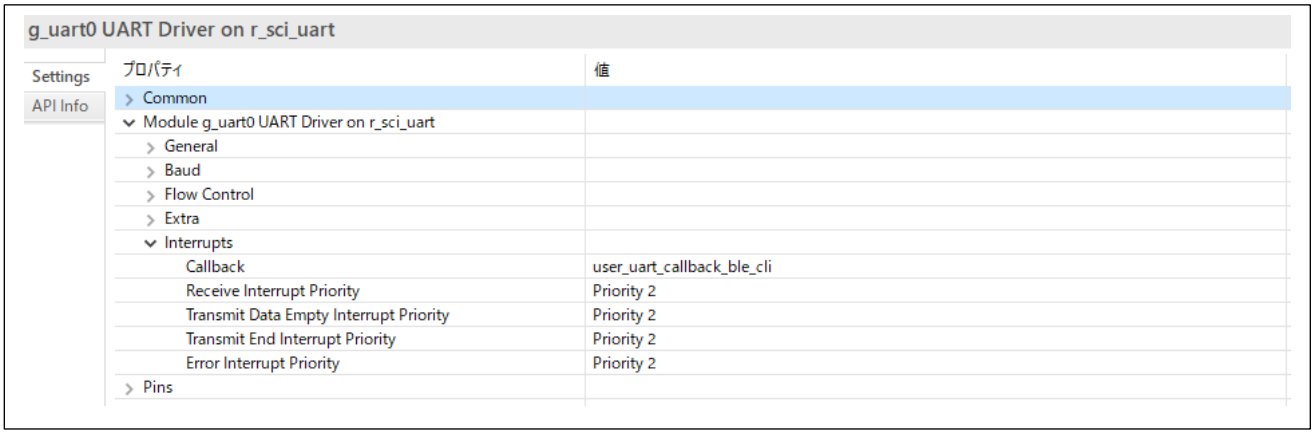

Figure 72. Interrupts of r_sci_uart

4.5.3 Designating module name

Edit value of BLE_UART_INSTANCE macro in app_lib / r_ble_console.c according to the module name of r_sci_uart which user named.

```
/**********************************************************************************
 Macro definitions
**********************************************************************************/

#define BLE_TX_BUFSIZ           (80)
#define BLE_UART_INSTANCE       (g_uart0)
```

Code 60. BLE_UART_INSTANCE macro

4.5.4 Serial data output of UART

Serial data output of UART can be performed by *R_BLE_CLI_Printf()* function. *R_BLE_CLI_Printf()* function can generate formatted character lines by the way like *printf()* function.

Table 29. Syntax of *R_BLE_CLI_Printf()*

Function Name	R_BLE_CLI_Printf	
Format	void R_BLE_CLI_Printf(const char *format, …);	
Return	void	-
Arguments	const char *format	Designate a constant character line including formats
	…	Variable number of arguments represented by formats can be designated.

4.6 Command List

4.6.1 GAP command

(1) Advertising command

adv command		
Format :	gap adv [adv_type] [operation]	
	Start, stop, or remove advertising.	
Parameters :	[adv_type]	Select one of the followings as the type of advertising. legacy : legacy advertising ext : extended advertising non-conn : non-connectable advertising periodic : periodic advertising
	[operation]	Start or stop advertising. start : start advertising stop : stop advertising. remove : remove advertising set specified by adv_type.
Example :	gap adv legacy start Start legacy advertising. The local device address is a static address. gap adv ext stop Stop extended advertising.	

Other parameters related to Advertising that cannot be set from this command are set in the Advertising parameter variables of gs_legacy_adv_param, gs_ext_adv_param, gs_non_conn_adv_param, and gs_periodic_adv_param in app_lib\cmd\r_ble_cmd_abs.c. Changing these variables will change the setting of Advertising parameters.

Table 30. legacy advertising parameter: gs_legacy_adv_param

Parameter Structure st_ble_abs_legacy_adv_param_t		gs_legacy_adv_param	
Type	**Field Name**	**Description**	**Default Value**
st_ble_dev_addr_t *	p_addr	Specify the remote address registered in the Resolving List. When o_addr_type is BLE_GAP_ADDR_RPA_ID_PUBLIC (0x02) or BLE_GAP_ADDR_RPA_ID_STATIC (0x03), specify the remote address registered in the Resolving List in p_addr. If o_addr_type is BLE_GAP_ADDR_PUBLIC (0x00) or BLE_GAP_ADDR_RAND (0x01), specify NULL for p_addr.	NULL
uint8_t *	p_adv_data	Specify Advertising Data. If NULL is specified, Advertising Data is not set.	gs_adv_data
uint8_t *	p_sres_data	Specify Scan Response Data. If NULL is specified, Scan Response Data is not set.	gs_sres_data
uint32_t	fast_adv_intv	Advertising is performed at the interval specified by fast_adv_intv for the period specified by the fast_period parameter. Time (ms) = fast_adv_intv * 0.625. Ignored if fast_period is 0. The range is 0x00000020-0x00FFFFFF.	0x00000100
uint32_t	slow_adv_intv	After the time specified by the fast_period parameter elapses, advertising is performed at the interval specified by slow_adv_intv for the period specified by the slow_period parameter. Time (ms) = adv_intv_max * 0.625 The range is 0x00000020-0x00FFFFFF.	0x00000200
uint16_t	fast_period	Specify the period for advertising in fast_adv_intv. Time = duration * 10ms. When the time specified in duration elapses, the BLE_GAP_EVENT_ADV_OFF event notifies that Advertising has stopped. Range : 0x0000-0xFFFF. If 0x0000 is specified, fast_period is ignored.	0x0100
uint16_t	slow_period	Specify the period for performing Advertising with slow_adv_intv. Time = duration * 10ms. When the time specified in duration elapses, the BLE_GAP_EVENT_ADV_OFF event notifies that Advertising has stopped. The range is 0x0000-0xFFFF. If 0x0000 is specified, slow_period is ignored.	0x0000
uint16_t	adv_data_length	Specify Advertising Data size (byte). For Legacy Advertising PDU, the range is 0 to 31. If 0 is specified, Advertising Data is not set.	sizeof(gs_adv_data)
uint16_t	sres_data_length	Specify the size (in bytes) of Scan Response Data. For Legacy Advertising PDU, the range is 0 to 31. If 0 is specified, Scan Response Data is not set.	sizeof(gs_sres_data)
uint8_t	adv_ch_map	Specify the channel to be used for advertising packet transmission. It is possible to specify by the logical sum of the following macros. **BLE_GAP_ADV_CH_37 (0x01)** 37 CH is used. **BLE_GAP_ADV_CH_38 (0x02)** 38 CH is used. **BLE_GAP_ADV_CH_39 (0x04)** 39 CH is used. **BLE_GAP_ADV_CH_ALL (0x07)** 37-39 CH is used.	BLE_GAP_ADV_CH_ALL

Parameter Structure st_ble_abs_legacy_adv_param_t		gs_legacy_adv_param	
Type	**Field Name**	**Description**	**Default Value**
uint8_t	filter	Specify Advertising Filter Policy. When p_addr parameter is NULL, advertising is performed according to the filter policy. This parameter is ignored if the remote device address is specified in the p_addr parameter. **BLE_ABS_ADV_ALLOW_CONN_ANY (0x00)** Accepts Connection Requests from all devices. **BLE_ABS_ADV_ALLOW_CONN_WLST (0x01)** Only devices registered in the White List will accept Connection Requests.	BLE_ABS_ADV_ALLOW_CONN_ANY
uint8_t	o_addr_type	Specify Own BD Address Type. **BLE_GAP_ADDR_PUBLIC (0x00)** Indicates a public address. **BLE_GAP_ADDR_RAND(0x01)** Indicates a static address. **BLE_GAP_ADDR_RPA_ID_PUBLIC (0x02)** Indicates that RPA is to be used. If there is no IRK on the Resolving List, use the public address. **BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)** Indicates that RPA is to be used. If there is no IRK on the Resolving List, use the static address.	BLE_GAP_ADDR_PUBLIC
uint8_t	o_addr[6]	When o_addr_type is BLE_GAP_ADDR_RAND (0x01) or BLE_GAP_ADDR_RPA_ID_RANDOM (0x03), specify the Random Address to be set in the Advertising Set. This parameter is ignored when using the Balance or Compact library.	Not set because o_addr_type is BLE_GAP_ADDR_PUBLIC.

Table 31. Extended advertising parameter: gs_ext_adv_param

Parameter Structure st_ble_abs_ext_adv_param_t		gs_ext_adv_param	
Type	**Field Name**	**Description**	**Default Value**
st_ble_dev_addr_t *	p_addr	Specify the remote address registered in the Resolving List. When o_addr_type is BLE_GAP_ADDR_RPA_ID_PUBLIC (0x02) or BLE_GAP_ADDR_RPA_ID_STATIC (0x03), specify the remote address registered in the Resolving List in p_addr. If o_addr_type is BLE_GAP_ADDR_PUBLIC (0x00) or BLE_GAP_ADDR_RAND (0x01), specify NULL for p_addr.	NULL
uint8_t *	p_adv_data	Specify Advertising Data. If NULL is specified, Advertising Data is not set.	gs_adv_data
uint32_t	fast_adv_intv	Advertising is performed at the interval specified by fast_adv_intv for the period specified by the fast_period parameter. Time (ms) = fast_adv_intv * 0.625. Ignored if fast_period is 0. The range is 0x00000020-0x00FFFFFF.	0x00000100
uint32_t	slow_adv_intv	After the time specified by the fast_period parameter elapses, advertising is performed at the interval specified by slow_adv_intv for the period specified by the slow_period parameter. Time (ms) = adv_intv_max * 0.625 The range is 0x00000020-0x00FFFFFF.	0x00000200
uint16_t	fast_period	Specify the period for advertising in fast_adv_intv. Time = duration * 10ms. When the time specified in duration elapses, the BLE_GAP_EVENT_ADV_OFF event notifies that Advertising has stopped. The range is 0x0000-0xFFFF. If 0x0000 is specified, fast_period is ignored.	0x0300
uint16_t	slow_period	Specify the period for performing Advertising with slow_adv_intv. Time = duration * 10ms. When the time specified in duration elapses, the BLE_GAP_EVENT_ADV_OFF event notifies that Advertising has stopped. The range is 0x0000-0xFFFF. If 0x0000 is specified, slow_period is ignored.	0x0000
uint16_t	adv_data_length	Specify Advertising Data size (byte). The range is from 0 to 229. If 0 is specified, Advertising Data will not be set.	sizeof(gs_adv_data)
uint8_t	adv_ch_map	Specify the channel to be used for advertising packet transmission. It is possible to specify by the logical sum of the following macros. **BLE_GAP_ADV_CH_37 (0x01)** 37 CH is used. **BLE_GAP_ADV_CH_38 (0x02)** 38 CH is used. **BLE_GAP_ADV_CH_39 (0x04)** 39 CH is used. **BLE_GAP_ADV_CH_ALL (0x07)** 37-39 CH is used.	BLE_GAP_ADV_CH_ALL
uint8_t	filter	Specify Advertising Filter Policy. When p_addr parameter is NULL, advertising is performed according to the filter policy. This parameter is ignored if the remote device address is specified in the p_addr parameter. **BLE_ABS_ADV_ALLOW_CONN_ANY (0x00)** Accepts Connection Requests from all devices. **BLE_ABS_ADV_ALLOW_CONN_WLST (0x01)** Only devices registered in the White List will accept Connection Requests.	BLE_ABS_ADV_ALLOW_CONN_ANY

Parameter Structure st_ble_abs_ext_adv_param_t		gs_ext_adv_param	
Type	**Field Name**	**Description**	**Default Value**
uint8_t	o_addr_type	Specify Own BD Address Type. **BLE_GAP_ADDR_PUBLIC (0x00)** Indicates a public address. **BLE_GAP_ADDR_RAND(0x01)** Indicates a static address. **BLE_GAP_ADDR_RPA_ID_PUBLIC (0x02)** Indicates that RPA is to be used. If there is no IRK on the Resolving List, use the public address. **BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)** Indicates that RPA is to be used. If there is no IRK on the Resolving List, use the static address.	BLE_GAP_ADDR_PUBLIC
uint8_t	adv_phy	Specify Primary ADV PHY. In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified. **BLE_GAP_ADV_PHY_1M (0x01)** 1M PHY is used as Primary Advertising PHY. **BLE_GAP_ADV_PHY_CD (0x03)** Use Coded PHY as Primary Advertising PHY. Coding scheme is the contents set by *R_BLE_VS_SetCodingScheme()*.	BLE_GAP_ADV_PHY_1M
uint8_t	sec_adv_phy	Specify Secondary ADV Phy. **BLE_GAP_ADV_PHY_1M (0x01)** 1M PHY is used for Secondary Advertising PHY. **BLE_GAP_ADV_PHY_2M (0x02)** 2M PHY is used for Secondary Advertising PHY. **BLE_GAP_ADV_PHY_CD (0x03)** Use Coded PHY for Secondary Advertising PHY. Coding scheme is the contents set by *R_BLE_VS_SetCodingScheme()*.	BLE_GAP_ADV_PHY_1M

Table 32. Non-Connectable advertising parameter: gs_non_conn_adv_param

Parameter Structure st_ble_abs_non_conn_adv_param_t		gs_non_conn_adv_param	
Type	**Field Name**	**Description**	**Default Value**
st_ble_dev_addr_t *	p_addr	For the remote address specified by p_addr Direct non-connectable advertising. If p_addr is NULL, Undirect Non-Connectable Advertising is performed.	NULL
uint8_t *	p_adv_data	Specify Advertising Data. If NULL is specified, Advertising Data is not set.	gs_adv_data
uint32_t	adv_intv	Advertising is performed at the interval specified by adv_intv for the period specified by the duration parameter. Time (ms) = adv_intv * 0.625 When duration is 0x0000, the interval advertisement specified by adv_intv is continued. The range is 0x00000020-0x00FFFFFF.	0x000000a0
uint16_t	duration	Specify the period for performing Advertising in adv_intv. Time = duration * 10ms. When the time specified in duration elapses, a BLE_GAP_EVENT_ADV_OFF event occurs. The range is 0x0000-0xFFFF. If 0x0000 is specified, duration is ignored.	0x0000
uint16_t	adv_data_length	Specify Advertising Data size (byte). If BLE_ABS_ADV_PHY_LEGACY (0x00) is specified in the adv_phy parameter, the range is 0-31. Otherwise, it is 0-1650. If 0 is specified, Advertising Data is not set.	sizeof(gs_adv_data)
uint8_t	adv_ch_map	Specify the channel to be used for advertising packet transmission. It is possible to specify by the logical sum of the following macros. **BLE_GAP_ADV_CH_37 (0x01)** 37 CH is used. **BLE_GAP_ADV_CH_38 (0x02)** 38 CH is used. **BLE_GAP_ADV_CH_39 (0x04)** 39 CH is used. **BLE_GAP_ADV_CH_ALL (0x07)** 37-39 CH is used.	BLE_GAP_ADV_CH_ALL
uint8_t	o_addr_type	Specify Own BD Address Type. **BLE_GAP_ADDR_PUBLIC (0x00)** Indicates a public address. **BLE_GAP_ADDR_RPA_ID_PUBLIC (0x02)** Indicates that RPA is to be used. If there is no IRK registered in the Resolving List, use Public Address.	BLE_GAP_ADDR_PUBLIC
uint8_t	adv_phy	Specify Primary ADV PHY. In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified. **BLE_GAP_ADV_PHY_1M (0x01)** 1M PHY is used as Primary Advertising PHY. **BLE_GAP_ADV_PHY_CD (0x03)** Use Coded PHY as Primary Advertising PHY. Coding scheme is the contents set by *R_BLE_VS_SetCodingScheme()*.	BLE_GAP_ADV_PHY_1M

Parameter Structure st_ble_abs_non_conn_adv_param_t		gs_non_conn_adv_param	
Type	**Field Name**	**Description**	**Default Value**
uint8_t	sec_adv_phy	Specify Secondary ADV Phy. **BLE_GAP_ADV_PHY_1M (0x01)** 1M PHY is used for Secondary Advertising PHY. **BLE_GAP_ADV_PHY_2M (0x02)** 2M PHY is used for Secondary Advertising PHY. **BLE_GAP_ADV_PHY_CD (0x03)** Use Coded PHY for Secondary Advertising PHY. Coding scheme is the contents set by *R_BLE_VS_SetCodingScheme()*.	BLE_GAP_ADV_PHY_1M

Table 33. Periodic advertising parameter: gs_periodic_adv_param

Parameter Structure st_ble_abs_perd_adv_param_t		gs_periodic_adv_param	
Type	**Field Name**	**Description**	**Default Value**
st_ble_abs_non_conn_adv_param_t	param	Specify the non-connectable advertising parameter.	gs_non_conn_adv_param (*1)
uint8_t *	p_perd_adv_data	Specify Periodic Advertising Data. If NULL is specified, Periodic Advertising Data is not set.	gs_adv_data
uint16_t	perd_intv	Specify Periodic Advertising Interval. Time (ms) = perd_intv * 1.25. The range is 0x0006-0xFFFF.	0x0040
uint16_t	perd_adv_data_length	Specify the size (bytes) of Periodic Advertising Data. The range is 0-1650. If 0 is specified, Periodic Advertising Data is not set.	sizeof(gs_adv_data)

*1: It is set in *exec_abs_adv()* of app_lib\cmd\r_ble_cmd_abs.c.

(2) Scan command

scan command		
Format :	gap scan (operation) (filter_ad_type) (filter_data) (addr_type) (-wl)	
	Start scan. It is not necessary to specify (operation) when starting scan. When scan stops, input [ctrl] + [c].	
Parameters :	(operation)	Specify operation for scan stop : stop scan.
	(filter_ad_type)	The AD type for filtering. Refer to Bluetooth SIG Assigned Numbers for generic access profile for the definition of the AD type. If the filter is not used, this parameter can be omitted.
	(filter_data)	The data for filtering. Specify the data for the filter_ad_type. If the filter is not used, this parameter can be omitted. If the filter_ad_type is not used, this parameter is ignored.
	(addr_type)	Specify the address type of scan request. When this parameter is omitted, static address is selected. pub : Public Address rnd : Static Address
	(-wl)	Specify this parameter when using white list. If white list is not used, this parameter is can be omitted.
Example :	gap scan Start scan. gap scan 2 0x01,0x29 Search the advertising report which of the AD Type : Incomplete List of 16-bit Service Class UUIDs(0x02) and the service UUID : 0x2901.	

Other parameters related to Scan that cannot be set from this command are set in the scan parameter variables of gs_phy_param_1m and gs_scan_param in app_lib\cmd\r_ble_cmd_abs.c. Changing these variables will change the scan parameter settings.

Table 34. Scan parameter: gs_phy_param_1m

Parameter Structure st_ble_abs_scan_phy_param_t		gs_phy_param_1m	
Type	**Field Name**	**Description**	**Default Value**
uint16_t	fast_intv	Specify Fast Scan interval. Fast Scan interval (ms) = fast_intv * 0.625 The range is 0x0004-0xFFFF.	0x0200 (320ms)
uint16_t	slow_intv	Specify the Slow Scan interval. Slow Scan interval (ms) = slow_intv * 0.625 The range is 0x0004-0xFFFF.	0x0800 (1.28s)
uint16_t	fast_window	Specify Fast Scan window. Fast Scan window (ms) = fast_window * 0.625 The range is 0x0004-0xFFFF.	0x0100 (160ms)
uint16_t	slow_window	Specify Slow Scan window. Slow Scan window (ms) = slow_window * 0.625 The range is 0x0004-0xFFFF.	0x0100 (160ms)
uint8_t	scan_type	Specify Passive Scan / Active Scan as the scan type. **BLE_GAP_SCAN_PASSIVE (0x00)** Indicates that a passive scan is to be performed. **BLE_GAP_SCAN_ACTIVE (0x01)** Indicates that Active Scan is to be performed.	BLE_GAP_SCAN_PASSIVE

Table 35. Scan parameter: gs_scan_param

Parameter Structure st_ble_abs_scan_param_t		gs_scan_param	
Type	**Field Name**	**Description**	**Default Value**
st_ble_abs_scan_phy_param_t *	p_phy_param_1M	Specify the Scan parameter for 1M PHY. Specify NULL when not scanning with 1M PHY. Specify scan parameter for either p_phy_param_1M or p_phy_param_coded.	&gs_phy_param_1M
st_ble_abs_scan_phy_param_t *	p_phy_param_coded	Specify the Scan parameter for Coded PHY. Specify NULL when not scanning with Coded PHY. Specify scan parameter for either p_phy_param_1M or p_phy_param_coded.	NULL
uint8_t *	p_filter_data	Specify the data to be filtered. Data included in a single Advertising Data PDU is targeted. Filtering is not performed for data indicated by multiple Advertising Data PDUs. When NULL is specified or when 0 is specified for filter_data_length, filtering is not performed.	gs_filt_data
uint16_t	fast_period	Specify the scanning time in Fast scan interval / Fast scan window. Time (ms) = fast_period * 10. The range is 0x0000-0xFFFF. When 0x0000 is specified, scanning by Fast scan interval / Fast scan window is not performed. When the time specified in fast_period elapses, a BLE_GAP_EVENT_SCAN_TO event occurs.	0x0100
uint16_t	slow_period	Specify the scan time in Slow scan interval / Slow scan window. Time (ms) = slow_period * 10. The range is 0x0000-0xFFFF. When 0x0000 is specified, scanning with Slow scan interval / Slow scan window continues. When the time specified by slow_period elapses, a BLE_GAP_EVENT_SCAN_TO event occurs.	0x0000
uint16_t	filter_data_length	Specifies the size of the filtering data indicated by the p_filter_data parameter. If 0 is specified, or p_filter_data is NULL, no filtering is performed. Up to 16 bytes can be specified.	0
uint8_t	dev_filter	Specify the Scan Filter Policy. Set one of the following values. **BLE_GAP_SCAN_ALLOW_ADV_ALL (0x00)** All Advertising PDUs and Scan Response PDUs are accepted. **BLE_GAP_SCAN_ALLOW_ADV_WLST (0x01)** Only Advertising PDUs and Scan Response PDUs of devices registered in the White List are accepted. **BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED (0x02)** All Advertising PDUs and Scan Response PDUs are accepted, except when the Directed Advertising PDU destination is not the Scanner identity address. Directed Advertising PDUs are accepted even if the destination is the RPA of the local device. **BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST (0x03)** Except for the following cases, all advertising, scan response PDUs are accepted. • The address included in the Direct Advertising PDU is not the Scanner identity address. • The Advertiser Identity Address is not registered in the White List.	BLE_GAP_SCAN_ALLOW_ADV_ALL

Parameter Structure st_ble_abs_scan_param_t		gs_scan_param	
Type	**Field Name**	**Description**	**Default Value**
uint8_t	filter_dups	Specify the presence or absence of duplicates filter to filter duplicate advertising packet notifications. The number of devices that can be filtered is eight. The duplicate filter is disabled for the ninth and subsequent devices. **BLE_GAP_SCAN_FILT_DUPLIC_DISABLE (0x00)** Disable duplicate filter. **BLE_GAP_SCAN_FILT_DUPLIC_ENABLE (0x01)** Enable duplicate filter.	BLE_GAP_SCAN_FILT_DUPLIC_DISABLE
uint8_t	filter_ad_type	Specify the AD type of the filtering data indicated by the p_filter_data parameter. For details on AD type, refer to Assigned Numbers for generic access profile of Bluetooth SIG.	—

(3) Connection command

conn command		
Format :	gap conn [addr] [addr_type]	
	Send a connection request. In case of stopping connection request, input [ctrl] + [c].	
Parameters :	[addr]	Remote device address.
	[addr_type]	Specify the followings as remote device address type. pub : Public Address rnd : Random Address
Example :	gap conn 74:90:50:00:95:a8 pub Send a connection request to the remote device whose public address is 74:90:50:00:95:a8. gap conn d8:19:e3:30:92:21 pub Send a connection request to the remote device whose random address is d8:19:e3:30:92:21.	

Other parameters related to Connection that cannot be set from this command are set in the connection parameter variables of gs_conn_phy_1m and gs_conn_param in app_lib\cmd\r_ble_cmd_abs.c. Changing these variables will change the connection parameter settings.

Table 36. Connection parameter: gs_conn_phy_1m

Parameter Structure		**gs_conn_phy_1m**	
st_ble_abs_conn_phy_param_t			
Type	**Field Name**	**Description**	**Default Value**
uint16_t	conn_intv	Specify the Connection interval. Time (ms) = conn_intv * 1.25. The range is 0x0006-0x0C80.	0x00A0 (200ms)
uint16_t	conn_latency	Specify Slave latency. The range is 0x0000-0x01F3.	0x0000
uint16_t	sup_to	Specify Supervision timeout. Time (ms) = sup_to * 10 The range is 0x000A-0x0C80.	0x03E8 (10s)

Table 37. Connection parameter: gs_conn_param

Parameter Structure st_ble_abs_conn_param_t		gs_conn_param	
Type	**Field Name**	**Description**	**Default Value**
uint8_t	filter	Specify how to select a remote device to establish a connection and the address type of a local device. **BLE_ABS_CONN_USE_ADDR_PUBLIC** **(BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_PUBLIC << 4))** Establish a connection with the remote device specified by p_addr. Local device uses public address. **BLE_ABS_CONN_USE_WLST_PUBLIC** **(BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_PUBLIC << 4))** Establish a connection with a remote device registered in the White List. Local device uses public address. **BLE_ABS_CONN_USE_ADDR_STATIC** **(BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_RAND << 4))** Establish a connection with the remote device specified by p_addr Local device uses static address. **BLE_ABS_CONN_USE_WLST_STATIC** **(BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_RAND << 4))** Establish a connection with a remote device registered in the White List. Local device uses static address. **BLE_ABS_CONN_USE_ADDR_RPA_PUBLIC** **(BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))** Establish a connection with the remote device specified by p_addr Use RPA. If the IRK is not registered in the Resolving List, public address is used. **BLE_ABS_CONN_USE_WLST_RPA_PUBLIC** **(BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_RPA_ID_PUBLIC << 4))** Establish a connection with a remote device registered in the White List. Use RPA. If the IRK is not registered in the Resolving List, public address is used. **BLE_ABS_CONN_USE_ADDR_RPA_STATIC** **(BLE_GAP_INIT_FILT_USE_ADDR | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))** Establish a connection with the remote device specified by p_addr Use RPA. If the IRK is not registered in the Resolving List, static address is used. **BLE_ABS_CONN_USE_WLST_RPA_STATIC** **(BLE_GAP_INIT_FILT_USE_WLST | (BLE_GAP_ADDR_RPA_ID_RANDOM << 4))** Establish a connection with a remote device registered in the White List.Use RPA. If the IRK is not registered in the Resolving List, static address is used.	BLE_ABS_CONN_USE_ADDR_PUBLIC
uint8_t	conn_to	Specify the time (s) from when the connection establishment request is issued until cancellation. The range is 0 <= conn_to <= 10. If 0 is specified, no cancellation is performed.	7(s)
st_ble_abs_conn_phy_param_t *	p_conn_1M	Specify 1M PHY connection parameters. When NULL is specified, connection with 1M PHY is not performed.	&gs_conn_phy_1m

Parameter Structure st_ble_abs_conn_param_t		gs_conn_param	
Type	**Field Name**	**Description**	**Default Value**
st_ble_abs_conn_phy_param_t *	p_conn_2M	Specify 2M PHY connection parameters. If NULL is specified, 2M PHY connection is not performed.	NULL
st_ble_abs_conn_phy_param_t *	p_conn_coded	Specify the connection parameters for Coded PHY. If NULL is specified, connection with Coded PHY is not performed.	NULL
st_ble_dev_addr_t *	p_addr	Specify the address of the remote device to be connected. This parameter is ignored if the filter parameter is BLE_GAP_INIT_FILT_USE_WLST (0x01).	&gs_conn_bd_addr (*1)

*1: Use the address entered on the command line.

(4) Disconnection command

disconn command		
Format :	gap disconn [conn_hdl] Disconnect the connection.	
Parameters :	[conn_hdl]	Connection handle of which the connection is disconnected.
Example :	gap disconn 0x0020 Disconnect the connection with connection handle 0x0020.	

(5) Device command

device command	
Format :	gap device Display the addresses of the connected devices.
Parameters :	None
Example :	gap device Display the addresses of the connected devices.

(6) Privacy command

priv command		
Format :	gap priv [operation] (IRK) [priv_mode] gap priv [operation] [addr] [addr_type] gap priv [operation]	
	Operate the local device's privacy.	
Parameters :	[operation]	Select one of the followings as the operation of privacy. set : Register the IRK of the local device in the resolving list and turn on the address generation function. It is used when the local device uses RPA in the advertising command and connection command. remove : Delete the remote device registered in the resolving list.
	{params, …}	[operation] : set (IRK) : The local device's IRK which is registered in the resolving list. If this parameter is omitted, the IRK is generated with the random generation function. [priv_mode] : Privacy mode and the address type of local device. Select one of the followings. net : network privacy mode. Static address is used as identity address. dev : device privacy mode. Static address is used as identity address. [operation] : remove [addr] : Specify the address (6 bytes) of the remote device registered in the Resolving list. [addr_type] : Specify the address type of the remote device registered in the Resolving list.
Example :	gap priv set 000102030405060708090a0b0c0d0e0f dev Register IRK : 0x0f0e0d0c0b0a09080706050403020100 and set the privacy mode to "device privacy mode". Static address is used as identity address. gap priv set net IRK is generated by the random number generation . The privacy mode is set to "network privacy mode". Static address is used as identity address. gap priv remove 12:34:56:78:9a:bc pub Delete the 12:34:56:78:9a:bc (public) remote device registered in the resolving list.	

(7) Connection config command

<table>
<tr><td colspan="3">conn_cfg command</td></tr>
<tr><td>Format :</td><td colspan="2">gap conn_cfg [operation] {params, ...}
Connection configuration command.</td></tr>
<tr><td rowspan="2">Parameters :</td><td>[operation]</td><td>Type of connection configuration. Select one of the followings.
update : Connection parameter update.
phy : Set PHY.
def_phy : Set default phy.
data_len : Set data packet length or data transmit time.</td></tr>
<tr><td>{params, ...}</td><td>[operation] : update
Parameter1 : Connection handle.
Parameter2 : Connection interval. Time(ms) = Parameter2 x 1.25.
Valid range is 0x0006-0x0C80.
Parameter3 : Slave latency. Valid range is 0x0000-0x01F3.
Parameter4 : Supervision timeout. Time(ms) = Parameter4 x 10.
Valid range is 0x000A-0x0C80.
Input Parameter2-4 to meet the following condition.
Parameter4 x 10 >= (1 + Parameter3) x Parameter2 x 1.25
[operation] : phy
Parameter1 : Connection handle
Parameter2 : Transmitter PHY. Parameter2 is set to a bitwise OR of the following values.
bit0 : 1M PHY
bit1 : 2M PHY
bit2 : Coded PHY
Parameter3 : Receiver PHY. Parameter3 is set to a bitwise OR of the following values.
bit0 : 1M PHY
bit1 : 2M PHY
bit2 : Coded PHY
Parameter4 : Coding scheme of Coded PHY. Select one of the following.
0x00 : The controller's preferred value.
0x01 : S=2 Coding scheme.
0x02 : S=8 Coding scheme.
[operation] : def_phy
Parameter1 : Transmitter PHY preferences which a remote device may change.
Parameter1 is set to a bitwise OR of the following values.
bit0 : 1M PHY
bit1 : 2M PHY
bit2 : Coded PHY
Parameter2 : Receiver PHY preferences which a remote device may change.
Parameter2 is set to a bitwise OR of the following values.
bit0 : 1M PHY
bit1 : 2M PHY
bit2 : Coded PHY
[operation] : data_len
Parameter1 : Connection handle
Parameter2 : Maximum transmit packet data length (in bytes).
Valid range is 0x001B-0x00FB.
Parameter3 : Maximum transmit time (us). Valid range is 0x0148-0x4290.</td></tr>
</table>

Example :	gap conn_cfg update 0x0026 0x0100 0 0x0100 Change the connection parameters of the connection handle : 0x0026 to the following values. connection interval : 0x0100 slave latency : 0 supervision timeout : 0x0100 gap conn_cfg phy 0x0026 2 2 0 Change the PHY of the connection (connection handle : 0x0026) Transmitter PHY : 2M Receiver PHY : 2M gap conn_cfg def_phy 7 7 Accept the following change request. Transmitter PHY : 1M, 2M and Coded PHY. Receiver PHY : 1M, 2M and Coded PHY. gap conn_cfg data_len 0x0026 0x00FB 0x4290 Change the following transmit packet length or transmit time Max transmit packet length : 251 bytes Max transmit time : 0x4290 us

(8) White List command

<table>
<tr><td colspan="3">wl command</td></tr>
<tr><td rowspan="2">Format :</td><td colspan="2">gap wl [operation] {params, ...}</td></tr>
<tr><td colspan="2">White List operation command.</td></tr>
<tr><td rowspan="2">Parameters :</td><td>[operation]</td><td>White List operation. Select one of the followings.
reg : Register a device specified with the {params, ...} on the White List.
del : Delete the device specified with the {params, ...} on the White List.
clear : Clear the White List.</td></tr>
<tr><td>{params, ...}</td><td>[operation] : reg
Parameter1 : Address of a device to be registered on the White List.
Parameter2 : Address type of a device to be registered on the White List.
pub : Public Address
rnd : Random Address

[operation] : del
Parameter1 : Address of a device to be deleted on the White List.
Parameter2 : Address type of a device to be deleted on the White List.
pub : Public Address
rnd : Random Address

[operation] : clear
Not used.</td></tr>
<tr><td>Example :</td><td colspan="2">gap wl reg 74:90:50:00:95:a8 pub
Register the device whose public address is 74:90:50:00:95:a8 on the White List.

gap wl del 74:90:50:00:95:a8 pub
Delete the device whose public address is 74:90:50:00:95:a8 on the White List.

gap wl clear
Clear the White List.</td></tr>
</table>

(9) Authentication command

<table>
<tr><td colspan="3">auth command</td></tr>
<tr><td rowspan="2">Format :</td><td colspan="2">gap auth [operation] {params, ...}</td></tr>
<tr><td colspan="2">Pairing or encryption command.</td></tr>
<tr><td rowspan="2">Parameters :</td><td>[operation]</td><td>Security operation.
start : Start pairing or encryption.
passkey : Input 6-digit number(decimal) to be required in passkey entry pairing.
numcmp : Return the result of a numeric comparison.
del : Delete the pairing keys.</td></tr>
<tr><td>{params,...}</td><td>[operation] : start
Parameter1 : Connection handle identifying the connection which local device starts pairing or encryption.

[operation] : passkey
Parameter1 : 6 digit passkey (decimal)

[operation] : numcmp
Parameter1 : Result of a numeric comparison.("yes" or "no")
Return "yes" if both devices display same number, otherwise "no".
[operation] : del
Parameter1 : Type of key to be deleted.
local : keys which local device distributes.
remote : keys distributed from the remote devices.
all : the above two types of keys.

Parameter2: Type of the remote device key deletion.
addr : Delete the keys specified by the Parameter3, 4.
all : Delete all the keys distributed from remote devices.

Parameter3 : Address of the remote device whose keys to be deleted.

Parameter4 : Address type of the remote device whose keys to be deleted.
pub : Public Address
rnd : Random Address</td></tr>
</table>

Example :	gap auth start 0x0026 Start pairing or encryption with the connection (connection handle : 0x0026). gap auth passkey 123456 Input "123456" as a passkey. gap auth numcmp yes Return "yes" as a result of numeric comparison. gap auth del remote all Delete all the keys distributed from the remote devices.

(10) Synchronization command

<table>
<tr><td colspan="3">sync command</td></tr>
<tr><td rowspan="2">Format :</td><td colspan="2">gap sync [operation] {params...}</td></tr>
<tr><td colspan="2">Create or Terminate a periodic sync.</td></tr>
<tr><td rowspan="2">Parameters :</td><td>[operation]</td><td>Periodic sync operation.
create : Create a periodic sync with the device whose address is specified by the {params...}. Scanning runs until a periodic sync is established.
In case of stopping creating periodic sync, input [ctrl] + [c].

term : Terminate the periodic sync whose sync_hdl is specified by the {params...}.</td></tr>
<tr><td>{params,...}</td><td>[operation] : create
Parameter1 : Address of the advertiser.
Parameter2 : Address type of the advertiser.

[operation] : term
Parameter1 : Sync handle identifying the periodic sync to be terminated.
If no parameters are given, all the established periodic syncs are terminated.</td></tr>
<tr><td>Example :</td><td colspan="2">gap sync create 74:90:50:00:95:a8 pub
Establish a periodic sync with the advertiser whose public address is 74:90:50:00:95:a8.

gap sync term 0x01
Terminate the periodic sync (sync handle : 0x01).</td></tr>
</table>

(11) Version command

ver command	
Format :	gap ver
	Get the following BLE Protocol Stack version information. - Link Layer - HCI - Host Stack - Manufacturer ID
Parameters :	None
Example :	gap ver Get the version information. Result sample : Link Layer / HCI Version HCI version : 0x09 *1 HCI revision : 0x000b Link Layer version : 0x09 *1 Link Layer subversion : 0x1908 Manufacturer ID : 0x0036 Host stack Version major version : 0x0d minor version : 0x19 subminor version : 0x08

*1 : The version number defined by Bluetooth SIG. The version number 0x09 shows Bluetooth 5.0 .

4.6.2 Vendor Specific (VS) command

(1) Tx Power command

txp command		
Format :	vs txp [operation] [conn_hdl] {params,...}	
	Set / Get the transmit power.	
Parameters :	[operation]	Transmit power operation. set : Set the transmit power. get : Get the transmit power.
	[conn_hdl]	Connection handle identifying the connection whose transmit power to be set or retrieved. Inputting 0xFFFF sets / gets the transmit power in the non-connected state.
	{params,...}	[operation] : set Parameter1 : Tx power level to be set. 0 : High 1 : Middle 2 : Low [operation] : get Not used.
Example :	vs txp set 0xFFFF 0 Set the non-connected state transmit power to the High level. vs txp get 0x0026 Get the transmit power of the connection (connection handle : 0x0026).	

(2) Coded Scheme command

scheme command		
Format :	vs scheme [type]	
	Set the coding scheme of the Coded PHY.	
Parameters :	[type]	Coding scheme for Primary advertising PHY, Secondary advertising PHY, request for connection establishment. This parameter is set to a bitwise OR of the following values. By default, S=8 coding scheme is enabled. bit0 : Coding scheme for Primary Advertising PHY(0:S=8/1:S=2). bit1 : Coding scheme for Secondary Advertising PHY(0:S=8/1:S=2). bit2 : Coding scheme for Connection(0:S=8/1:S=2).
Example :	vs scheme 7 Set coding scheme for Primacy Advertising, for Secondary Advertising, and for Connection to S=2.	

(3) Extended Direct Test Mode(DTM) command

<table>
<tr><td colspan="3">test command</td></tr>
<tr><td rowspan="2">Format :</td><td colspan="2">vs test [operation] {params, ...}</td></tr>
<tr><td colspan="2">DTM test command.</td></tr>
<tr><td rowspan="2">Parameters :</td><td>[operation]</td><td>DTM test operation. Select one of the followings.
tx : Start DTM transmitter test.
Set "channel", "length", "payload", "phy", "tx_power", "option" and "number of packet" to {params, ..,}.
rx : Start DTM receiver test.
Set "channel" and "phy" to {params, ...}.
end : Terminate DTM test.
No parameter.</td></tr>
<tr><td>{params, ...}</td><td>[operation] : tx
Parameter1 : Channel used in Tx test.
Valid range is 0 to 39.
Frequency range is 2402 MHz to 2480 MHz.
Parameter2 : Length(in bytes) of the packet used in Tx Test.
Valid range is 0 to 255.
Parameter3 : Packet Payload.
Valid range is 0x00-0x07.
If the Parameter6 is set to "non-modulation", this parameter is ignored.

<u>Payload type:</u>
0x00 : PRBS9 sequence '11111111100000111101..'.
0x01 : Repeated '11110000'
0x02 : Repeated '10101010'
0x03 : PRBS15 sequence
0x04 : Repeated '11111111'
0x05 : Repeated '00000000'
0x06 : Repeated '00001111'
0x07 : Repeated '01010101'
Parameter4 : Transmitter PHY used in test. Select one of the following.
If the Parameter6 is set to "non-modulation", this parameter is ignored.
If the Parameter6 is configured to "modulation" and "continuous transmission",
0x03 : Coded PHY (S=8) and 0x04 : Coded PHY (S=2) are not supported.

0x01 : 1M PHY
0x02 : 2M PHY
0x03 : Coded PHY (S=8)
0x04 : Coded PHY (S=2)</td></tr>
</table>

Parameters :	{params, ...}	Parameter5 : Tx Power Level used in DTM Tx Test. Select one of the following. 0x00 : High 0x01 : Middle 0x02 : Low Parameter6 : The test option configuration. This parameter is set to a bitwise OR of the following bits. bit0 : 0:modulation, 1:non-modulation bit1 : 0:packet transmission, 1:continuous transmission Parameter7 : The number of packets to be sent. Valid range is 0x0000-0xFFFF. If the Parameter6 is configured to "continuous transmission", this parameter is ignored. If this parameter is set to 0x0000, the packets are continuously transmitted until test end command is issued. [operation] : rx Parameter1 : Channel used in the test. Valid range is 0 to 39. Frequency range is 2402 MHz to 2480 MHz. Parameter2 : Receiver PHY used in the test. Select one of the following. 0x01 : 1M PHY 0x02 : 2M PHY 0x03 : Coded PHY The coding scheme (S=8/S=2) doesn't need to be specified in the receiver test. [operation] : end Not used.
Example :	vs test tx 39 251 1 3 1 0 1 Start DTM transmitter test. CH : 39ch Packet length : 251 bytes payload : Repeated '11110000' sequence phy : Coded PHY(S=8) tx_power : Middle option : modulation packet transmission num_of_packet : 1 vs test rx 39 2 Start DTM receiver test. CH : 39ch phy : 2M PHY vs test end Terminate DTM test.	

(4) BD Address command

<table>
<tr><td colspan="3">addr command</td></tr>
<tr><td rowspan="2">Format :</td><td colspan="2">vs addr [operation] [area] {params...}</td></tr>
<tr><td colspan="2">Set/Get the address of the local device.</td></tr>
<tr><td rowspan="3">Parameters :</td><td>[operation]</td><td>Address operation. Select one of the followings.
set : Set an address to the local device.
Set address type and address to {params...} .
If [area] is "df", the address is enabled after reset.
get : Get the address of the local device.
Set the address type to {params...}.</td></tr>
<tr><td>[area]</td><td>The area where the address is stored.
curr : The temporary area storing the address.
df : The area storing the address in the Data Flash.</td></tr>
<tr><td>{params...}</td><td>[operation] : set
Parameter1 : Address type
pub : Public Address
rnd : Random Address
Parameter2 : Address
[operation] : get
Parameter1 : Address type
pub : Public Address
rnd : Random Address</td></tr>
<tr><td>Example :</td><td colspan="2">vs addr set df pub 78:90:50:00:95:a8
Set the public address : 78:90:50:00:95:a8 to the Data Flash.
vs addr get curr pub
Get the current public address.</td></tr>
</table>

(5) Random Number generation command

rand command		
Format :	vs rand [rand_size]	
	Generate a random number.	
Parameters :	[rand_size]	Specify the size of the random number to be generated. Range: 4 to 16 [bytes].
Example :	vs rand 16 Generate a 16 bytes random number.	

(6) Scan Channel command

scan_ch_map command		
Format :	vs scan_ch_map [operation] {params,...}	
	Set/Get the scan channel map.	
Parameter :	[operation]	Scan Channel operation. Select one of the followings. set : Set the channel map specified by {params,...} as scan channel. get : Get the current scan channel map.
	{params,...}	[operation] : set Parameter 1 : The channel map to be set. It is a bitwise OR of the following values. bit 0 : 37 ch bit 1 : 38 ch bit 2 : 39 ch other than the above : reserved [operation] : get “Get” operation does not use parameter.
Example :	vs scan_ch_map set 7 Set 37, 38, 39ch as scan channel. vs scan_ch_map get Get the current scan channel map.	

4.6.3 SYS command

(1) MCU Software Standby command

stby command		
Format :	sys stby [operation]	
	Control the software standby mode.	
Parameters :	[operation]	Software standby operation. Select one of the followings. on : Enter the software standby mode. off : Come back from the software standby mode. get : Get the current software standby status.
Example :	sys stby on Enter the software standby mode.	

4.6.4 BLE command

(1) BLE protocol stack Reset command

stby command	
Format :	ble reset
	Reset the BLE protocol stack.
Parameters :	None
Example :	ble reset

(2) BLE protocol stack Close command

stby command	
Format :	ble close
	Terminate the BLE protocol stack. To restart the BLE protocol stack, execute “ble reset” command.
Parameters :	None
Example :	ble close

4.6.5 LSC command

(1) Set switch state notification command

set_switch_state_ntf command		
Format :	lsc set_switch_state_ntf [conn_hdl] [enable]	
	Enable receiving notification from GATT server.	
Parameters :	[conn_hdl]	Connection handle identifying the connection whose receive notification from GATT server.
	[enable]	Designate if receiving notification is enable. 0 : Disable 1 : Enable
Example :	lsc set_switch_state_enable 0x0020 1 Enable receiving notification from GATT server of 0x0020	

(2) Write led blink rate command

write_led_blink_rate command		
Format :	lsc write_led_blink_rate [conn_hdl] [blink rate]	
	Write value of LED blink rate of GATT server.	
Parameters :	[conn_hdl]	Connection handle identifying the connection with GATT server which is written its LED blink rate.
	[blink rate]	Designate LED blink rate 0x00 : LED turns off 0x01 – 0xFE : LED blinks in the frequency based on this value 0xFF : LED turns on
Example :	lsc write_led_blink_rate 0x0020 0xA0 Write 0xA0 as value of LED blink rate of GATT server of 0x0020	

(3) Read led blink rate command

read_led_blink_rate command		
Format :	lsc read_led_blink_rate [conn_hdl] [blink rate]	
	Read value of LED blink rate of GATT server written.	
Parameters :	[conn_hdl]	Connection handle identifying the connection with GATT server which is read its LED blink rate.
Example :	lsc read_led_blink_rate 0x0020 Read value of LED blink rate of GATT server of 0x0020	

4.6.6 Command creation procedure

In the command line interface feature, user can create their own commands by defining commands in the st_ble_cli_cmd_t type variable. This section describes an example of creating a new command to operate the custom profile LED Switch service Client (hereafter “lsc”) provided in the demo project.

(1) Command definition

Defines command name, subcommand group, number of subcommands, and the message string output by “help” command. For “lsc” command, define a command structure variables as following.

```
const st_ble_cli_cmd_t g_lsc_cmd =
{
    .p_name      = "lsc",
    .p_cmds      = lsc_sub_cmds,
    .num_of_cmds = ARRAY_SIZE(lsc_sub_cmds),
    .p_help      = "Sub Command: set_switch_state_ntf, write_led_blink_rate, read_led_blink_rate\n"
                   "Try 'lsc sub-cmd help' for more information",
};
```

Code 61. Command definition example

(2) Subcommand definition

Defines subcommand. For “lsc” command, define a subcommand structure variables as following.

If user wants to create a command such as the "Connection command” or "Scan command” that manually abort the process, user needs to set a abort handler.

During execution of a command for which the abort handler is set, no other command input will be accepted until the command execution is aborted by pressing Ctrl+C key.

```
static const st_ble_cli_cmd_t lsc_set_switch_state_ntf_cmd =
{
    .p_name = "set_switch_state_ntf",
    .exec   = cmd_lsc_set_switch_state_ntf,
    .p_help = "Usage: lsc set_switch_state_ntf conn_hdl value",
};
……
static const st_ble_cli_cmd_t lsc_read_led_blink_rate_cmd =
{
    .p_name = "read_led_blink_rate",
    .exec   = cmd_lsc_read_led_blink_rate,
    .p_help = "Usage: lsc read_led_blink_rate conn_hdl",
};

……
static const st_ble_cli_cmd_t lsc_write_led_blink_rate_cmd =
{
    .p_name = "write_led_blink_rate",
    .exec   = cmd_lsc_write_led_blink_rate,
    .p_help = "Usage: lsc write_led_blink_rate conn_hdl blink_rate",
};
……
static const st_ble_cli_cmd_t * const lsc_sub_cmds[] =
{
    &lsc_set_switch_state_ntf_cmd,
    &lsc_write_led_blink_rate_cmd,
    &lsc_read_led_blink_rate_cmd,
};
```

Code 62. Subcommand definition example

(3) Subcommand function definition

Define the function to be processed when the subcommand is executed.

For “lsc” command, define a subcommand function as following.

```
/*---------------------------------------------------------------
    lsc set_switch_state_ntf command
---------------------------------------------------------------*/
static void cmd_lsc_set_switch_state_ntf(int argc, char *argv[])
{
    if (argc != 3)
    {
        pf("lsc %s: unrecognized operands\n", argv[0]);
                return;
    }

    uint16_t conn_hdl;
    conn_hdl = (uint16_t)strtol(argv[1], NULL, 0);

    long value = strtol(argv[2], NULL, 0);

    ble_status_t ret;
    ret = R_BLE_LSC_WriteSwitchStateCliCnfg(conn_hdl, (uint16_t *)&value);
    if (ret != BLE_SUCCESS)
    {
        pf("lsc %s: failed with 0x%04X\n", argv[0], ret);
                return;
        }
}
```

Code 63. Subcommand function example

(4) Registering commands

After defining the command and subcommand, register the command using *R_BLE_CLI_RegisterCmds()* API as following so that it can be used as an application-specific command.

```
static const st_ble_cli_cmd_t * const gsp_cmds[] =
{
     &g_abs_cmd,
     &g_vs_cmd,
     &g_sys_cmd,
     &g_lsc_cmd,
     &g_ble_cmd
};

……
void app_main(void)
{

……
    R_BLE_CLI_Init();
    R_BLE_CLI_RegisterCmds(gsp_cmds, sizeof(gsp_cmds)/sizeof(gsp_cmds[0]));
    R_BLE_CLI_RegisterEventCb(NULL);

……
}
```

Code 64. Command register example

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar.31.2020	—	First edition issued.
1.01	Oct.07.2020	—	Add Chapter 4. Move “3. BLE Module Detail” to “4. Appendix”. Update attached sample project.
1.02	Mar.03.2021	—	In this revision, GATT client demo projects were newly added and this document was also revised with accompanying it. Following items were revised or added in this revision. • Revised “1 Overview” • Revised “2.1 Operating environment” • Revised “2.2 Importing demo project” • Revised “2.4 Demo project behavior” • Revised “3.1 BareMetal environment (Server)” • Revised “3.2 FreeRTOS environment (Server, EventGroup as Synchronization Type case)” • Added “3.5 BareMetal environment (Client)” • Added “3.6 FreeRTOS environment (Client, EventGroup as Synchronization Type case)” • Added “4.5 Importing CLI (Command Line Interface) to user’s project” • Added “4.6 Command List” Following GATT server demo projects were updated. • ble_baremetal_ek_ra4w1 • ble_freertos_ek_ra4w1 Following GATT client demo projects were newly added. • ble_baremetal_ek_ra4w1_client • ble_freertos_ek_ra4w1_client
1.03	Aug.31.2021	—	• Add section 1.3 • Add section 2.2 item 4 and 5. • Add explanation about “extended”, “balance” and “compact” configuration in section 4.1.4. • Update attached sample application for FSP3.2.
1.04	Feb.25.2022	—	• Add section 3.4, section 3.8. • Add attached sample application for Azure RTOS. • Update attached sample application for FSP3.6.
1.0.5	Apr.27.2022	---	• Update Table 5. • Add section 4.1.5.
1.0.6	Oct.06.2022	---	• Add explanation for task synchronization method for FreeRTOS in section 1.2. • Add section 3.3 and 3.7. • Correction of typo about security data structure in section 4.3.3. • Update attached sample project for FSP4.0.

Rev.	Date	Description	
		Page	Summary
1.0.7	Oct.26.2022	120	• Updated how to write device specific data to code flash area by using Renesas Flash Programmer. • Update attached sample project for FSP4.1.

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/.