Bluetooth® Low Energy Protocol Stack

Embedded Configuration Sample Program

R01AN3319EJ0120
Rev1.20
Jan 31, 2022

Introduction

This sample program provides an example implementation of the embedded configuration application. The embedded configuration application performs two behavior, Bluetooth® Low Energy behavior and application behavior, on RL78/G1D. The source codes can be used for the basement of your own application.

The developers can know the implementation method of the embedded configuration application by understanding the sample program. This document provides information needed to understand the sample program details, such as the sample program usage, structure, functional and detailed design.

There are two types of the sample programs, for Central role and for Peripheral role. The sample program works on RL78/G1D Evaluation Board and uses BLE protocol stack.

Target Device

RL78/G1D

Rerated Documents

Document Name	Document Number
Bluetooth Low Energy Protocol Stack	-
User's Manual	R01UW0095E
API Reference Manual: Basics	R01UW0088E
Quick Start Guide	R01AN2767E
Security Library	R01AN3777E
RL78/G1D	-
User's Manual: Evaluation Board	R30UZ0048E

Contents

1. Introduction

The sample program will provide an example implementation by using Sample Custom Profile, the simple example profile for this sample program. This sample program includes following implementation.

- The usage of fundamental GAP (Generic Access Profile) API for central and peripheral role
- Establish security by encryption
- The usage of RWKE (Renesas Wireless Kernel Extension) task and message
- Sample Custom Profile definition
- Multiple connections between a central role device and peripheral role devices
- The simple user application like push button switches and LED indicators

The sample program uses two or more RL78/G1D Evaluation Board (hereafter called as Evaluation Board). One board will be programmed with Hex file for central role (hereafter called simply as Central), and the others will be programmed with Hex file for peripheral role (hereafter called as Peripheral)

Section 2 guides you the sample program demo environment setup and operation.

Section 3 explains about the sample program internal structure and file/directory structure.

Section 4 explains about installation, working with each development environment such as e^2 studio and CS+.

Section 5 explain about the sample program functionality, API and configuration parameters.

2. Sample Program Demo

The following sub-sections will describe about how to evaluate this sample program. In the next two sub-section, it shows the environment and its setup procedure, and then next sub-section explains how to operate the demo.

2.1 Environment

Followings shows the environment needed to evaluate this sample program.

- Hardware Environment
  - Personal Computer
    - PC/AT™ compatible computer
    - Processor : at least 1.6GHz
    - Main Memory : at least 1Gbyte
    - Interface : USB2.0 (for connecting E1 emulator and RL78/G1D Evaluation Board)
  - Device
    - Two or more RL78/G1D Evaluation Boards (RTK0EN0001D01001BZ)
    - USB cable (A type male / mini-B type male)
  - Tool
    - Renesas On-chip Debugging Emulator E1 (R0E000010KCE00)
  - Software Environment
    - Windows®7 or later
    - e^2 studio V4.3.1.001 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
      or Renesas CS+ for CC V4.00.00 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
      or Renesas CS+ for CA, CX V3.02.00 / Renesas CA78K0R V1.72
    - Renesas Flash Programmer v3.02.00
    - Tera Term V4.89 (You can use any Terminal Software)
    - UART-USB Conversion Device Driver

      [Note] UART-USB device driver is required to connect an Evaluation Board with a PC. You can download the driver from "FTDI (Future Technology Devices International) – Drivers" web page.
      https://ftdichip.com/drivers/d2xx-drivers/

2.2 Environment Setup

Figure 2-1 shows the environment setup for typical test condition.

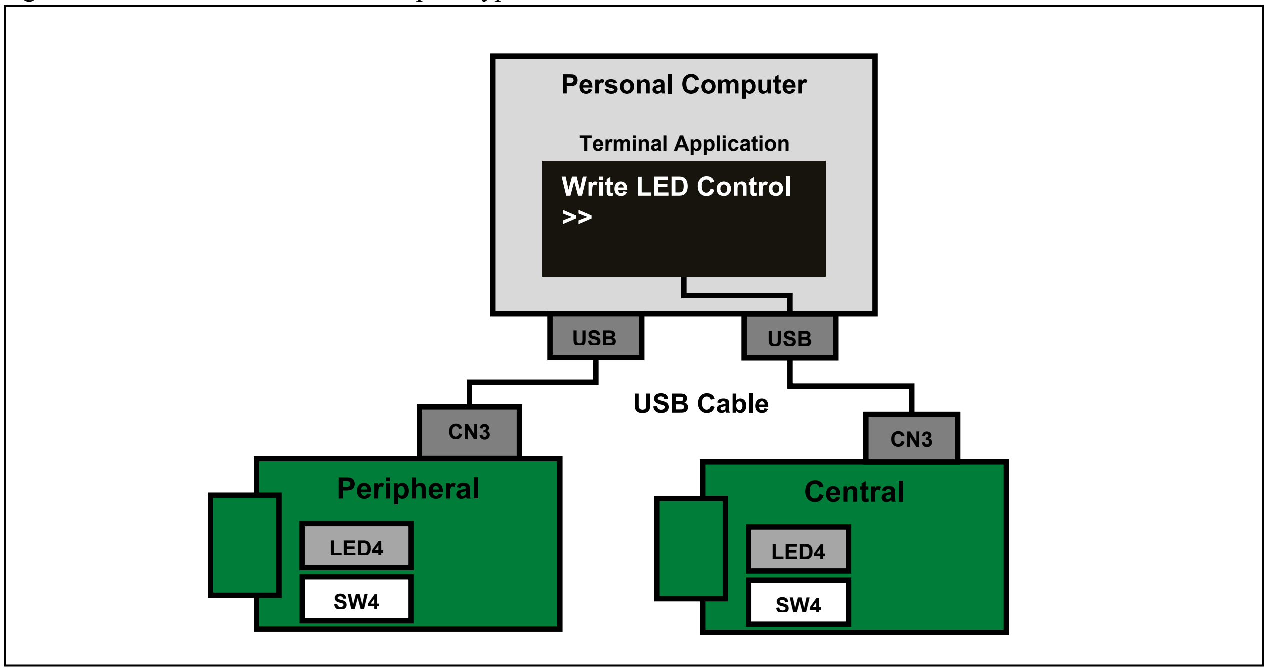


Figure 2-1 Environment Setup

Below show the steps for the demo environment setup.

1. Prepare two or more Evaluation boards. Use one Evaluation Board as Central and others as Peripherals. Central can connect multiple Peripherals concurrently (up to 8, the default is 4. Refer section 5.1.2 for the configuration). The number of Peripherals depend on your requirement for the multiple connections.

2. Adjust the switch settings of all Evaluation Boards for Central and Peripheral as shown in Table 2-1.

Table 2-1 Evaluation Board Switch Setting for Sample Program Demo

Switch	Position	Remark
SW7	OFF	Dot side is ON.
SW8	OFF	
SW9	OFF	
SW10	ON	
SW11	OFF	
SW12	OFF	
SW13	ON	

3. Program the Evaluation Boards with pre-compiled Hex file. Write the Hex file for Central to one Evaluation Board, and write the Hex file for Peripheral to other Evaluation Boards. To build the Hex file, refer section 4 in this document. Regarding to program the Hex file onto an Evaluation Board, refer “Quick Start Guide (R01AN2767)” “Section 5 Writing Programs”.
   - Hex file for Central:
     - ROM_File\cc_rl\Embedded\RL78_G1D_CCE(EMBSMP_Central).hex
   - Hex file for Peripheral
     - ROM_File\cc_rl\Embedded\RL78_G1D_CCE(EMBSMP_Peripheral).hex

4. Install Terminal Software onto the Personal Computer. For this demonstration, any preferable Terminal Software can be used. For testing this sample program, the demonstration will use the Teraterm V4.89.

5. Setup the Terminal Software with as shown in Table 2-2.

Table 2-2 Terminal Software Settings

Setting	Setting Value
New-line (Receive)	LF
New-line (Transmit)	CR
Baud rate	4,800 [bps]
Data Length	8 [bit]
Parity	none
Stop Bit	1 [bit]
Flow Control	none

6. First, connect Central to Personal Computer via USB cable. Then open the installed Terminal Software to connect with Central. Push the SW5 on Central to reset and then you will see the menu on Terminal Software.
   Central must communicate through Terminal Software. Peripheral also must connect to Terminal Software to display passkey.

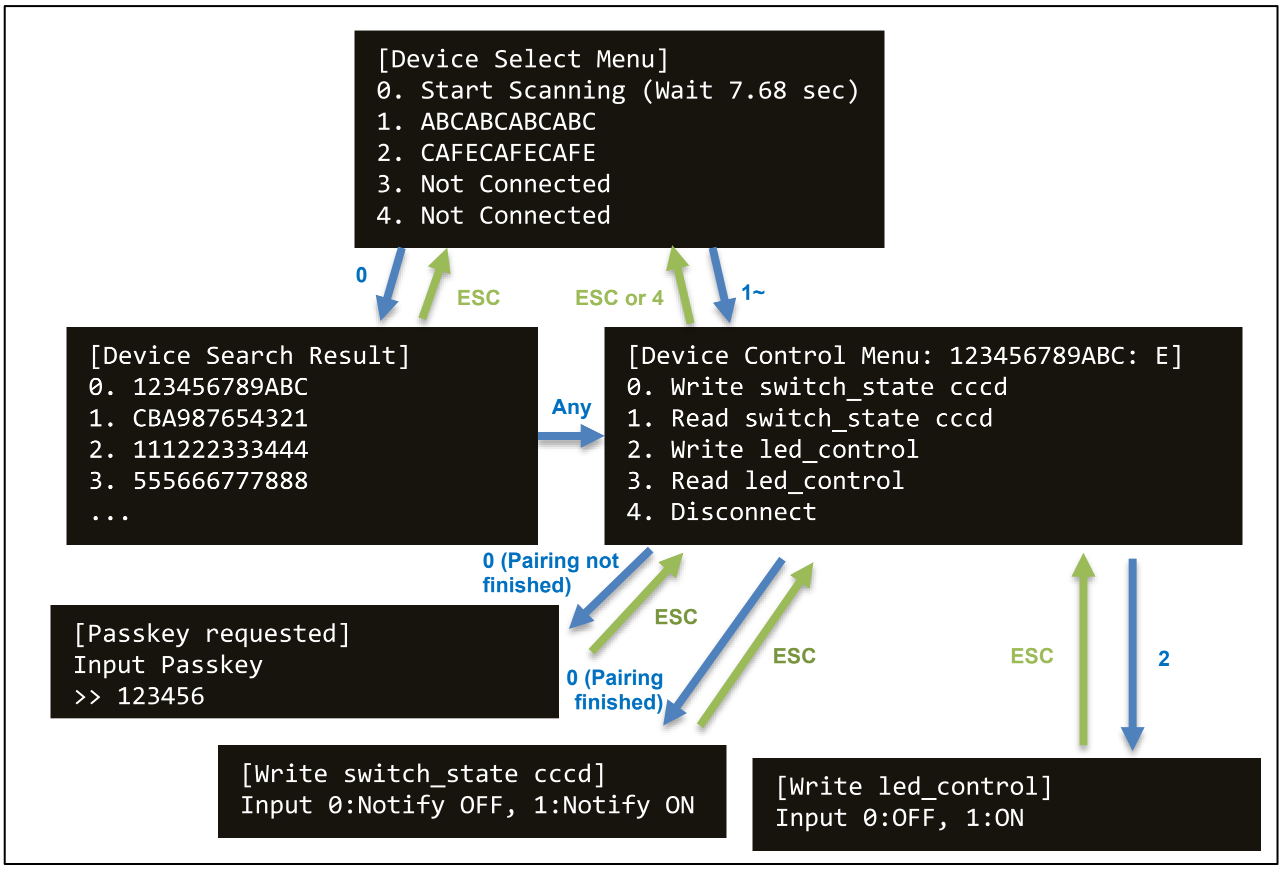
7. Now LED1 and LED2 are blinking on all Evaluation Boards. Due to the power saving mode, the blinking can be stopped or slow down.

2.3 Operation

Blow are the procedures to operate the demo.

1. After booting up, Peripheral automatically starts advertising.

2. On the Terminal Software, the terminal menu is displayed as shown in Figure 2-2. The figure also shows the terminal menu flow diagram. The menu items are selected by using number keys (0 to 9), and Esc key is used to exit current menu then move to the upper menu.



```
[Device Select Menu]
0. Start Scanning (Wait 7.68 sec)
1. ABCABCABCABC
2. CAFECAFECAFE
3. Not Connected
4. Not Connected
```

```
[Device Search Result]
0. 123456789ABC
1. CBA987654321
2. 111222333444
3. 555666777888
...
```

```
[Device Control Menu: 123456789ABC: E]
0. Write switch_state cccd
1. Read switch_state cccd
2. Write led_control
3. Read led_control
4. Disconnect
```

```
[Passkey requested]
Input Passkey
>> 123456
```

```
[Write switch_state cccd]
Input 0:Notify OFF, 1:Notify ON
```

```
[Write led_control]
Input 0:OFF, 1:ON
```

Figure 2-2 Terminal Menu Flow

3. Using “Start Scanning (Wait 7.68 sec)” in “Device Select Menu”, Central can start scanning the nearby Peripherals (This takes 7.68 sec. This is based on RBLE_GAP_Search_Device API). Then a list of Peripherals is shown in “Device Search Result” menu. To start a connection with one of the Peripheral listed in “Device Search Result”, select one by pressing respective number key.

4. After establishing the connection, the selected Peripheral can be communicated through “Device Control Menu”. “Device Control Menu” have Peripheral address and the security status with the peer device. When encryption is established with the peer device, “:E” sign on the menu. When encryption is not established, “: No Sec” sign on the menu. The menu has five commands from “0” to “4” as below.
   - **Write switch_state cccd**: In this menu, it can enable or disable the notification of Peripheral. Write 1 for the enable, write 0 for the disable. The notification is implemented with SW4 PUSH/RELEASE state in Peripheral. When the notification is enabled, the Peripheral checks its SW4 state either PUSH or RELEASE for every 500ms interval and send the current state to Central. With respect to the notified SW4 state from Peripheral, Central changes its LED4 ON/OFF state. See left side of Figure 2-3 for this operation. CCCD is an abbreviation of Client Characteristic Configuration Descriptor.
     This characteristic descriptor has Unauthenticated pairing permission, thus when an access is executed before pairing established, the access causes Error. When Central receives the error, it starts pairing. Passkey is displayed on Peripheral terminal software, then a user properly input the passkey to Central terminal software, pairing established and the communication encrypted. Security information exchanged between Central /

Peripheral is stored into RL78/G1D Data Flash by bonding feature. This information can be used subsequent communication.

- **Read switch_state cccd**: It is used to get the Peripheral's notification enable/disable setting from Central. "1" is read when it is enabled, "0" is read when it is disabled.

- **Write led_control**: In this menu, it does control Peripheral's LED4 ON/OFF state from Central. Write 1 for the enable, write 0 for the disable. See right side of Figure 2-3 for this operation.

- **Read led_control**: It is used to get the Peripheral's LED4 ON/OFF state from Central. "1" is read when it is ON, "0" is read when it is OFF.

- **Disconnect**: Selecting this option will disconnect between Central and the Peripheral. Then back to "Device Select Menu" in the terminal.

5. The procedures to establish multiple connections between Central and Peripherals are as below.
   1. Select "Start Scanning (wait 7.68 sec)" in "Device Select Menu" and establish a connection with a Peripheral.
   2. After establishing the connection, "Device Control Menu" is displayed. Then use Esc key to move up to "Device Select Menu".
   3. Re-select "Start Scanning (wait 7.68 sec)" in "Device Select Menu" to find another Peripheral and establish a connection with the Peripheral. After this, multiple connections are established.
   4. Currently connected Peripherals are displayed in "Device Select Menu". When a connection is disconnected, the Peripheral is removed from the "Device Select Menu".

   Note: The number of menu items in "Device Select Menu" except "Start Scanning (Wait 7.68 sec)" is the maximum number of concurrent connection between one Central and Peripherals. The example of Figure 2-2 shows that Central can establish four connections. Currently two connections have been established and two more connections can be established.

6. By pushing SW2, security information stored during pairing is deleted. The target of deletion is the information for non-connecting devices. The security information of connecting devices are not deleted. To delete all security information, disconnect with all devices and push SW2. When completing the deletion, "Bonding information have been deleted." is displayed on the terminal software.

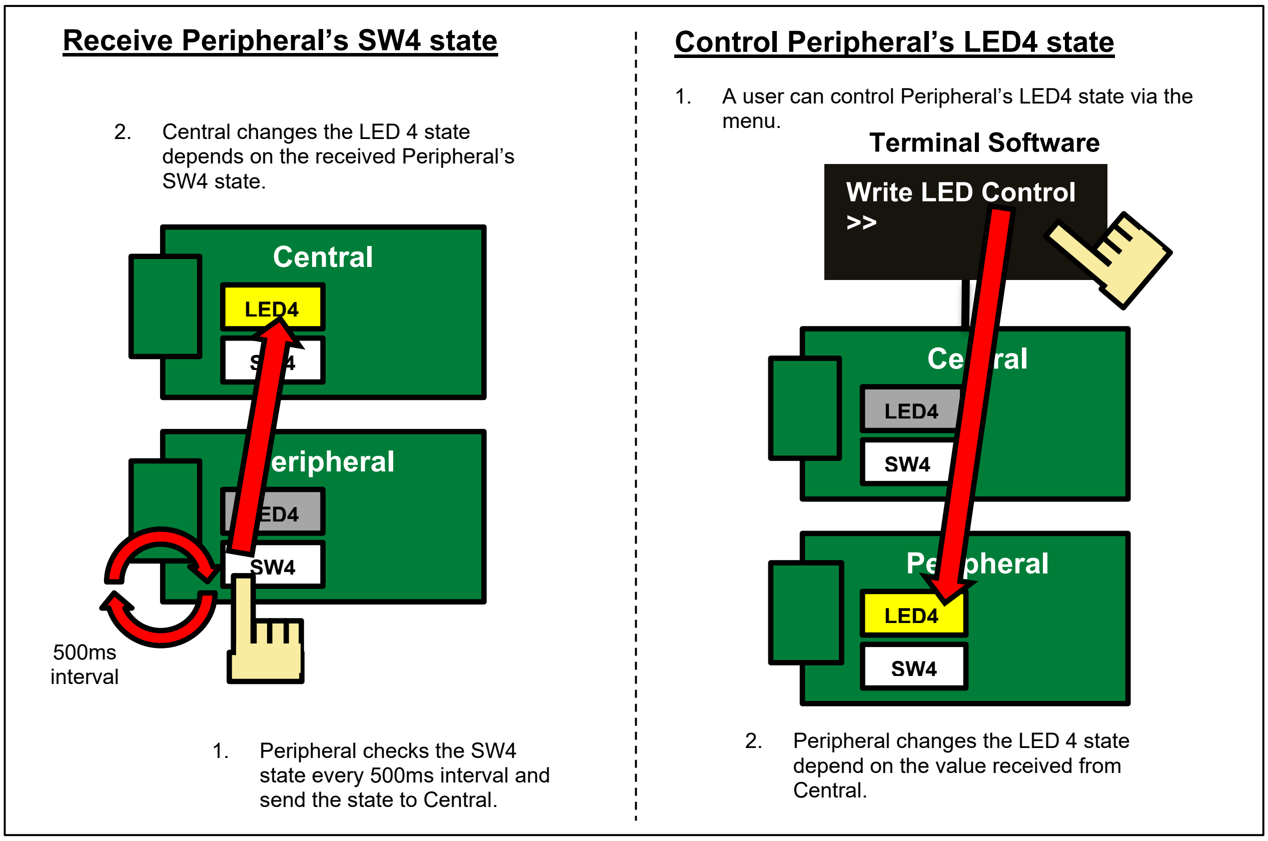

Receive Peripheral's SW4 state
2. Central changes the LED 4 state depends on the received Peripheral's SW4 state.
Central
LED4
SW4
Peripheral
LED4
SW4
500ms interval
1. Peripheral checks the SW4 state every 500ms interval and send the state to Central.
Control Peripheral's LED4 state
1. A user can control Peripheral's LED4 state via the menu.
Terminal Software
Write LED Control
>>
Central
LED4
SW4
Peripheral
LED4
SW4
2. Peripheral changes the LED 4 state depend on the value received from Central.

Figure 2-3 Peripheral Device Control

3. Structure

This section will explain about sample program structure for both Central and Peripheral as well as delivered file structure in the package.

3.1 Sample Software Structure

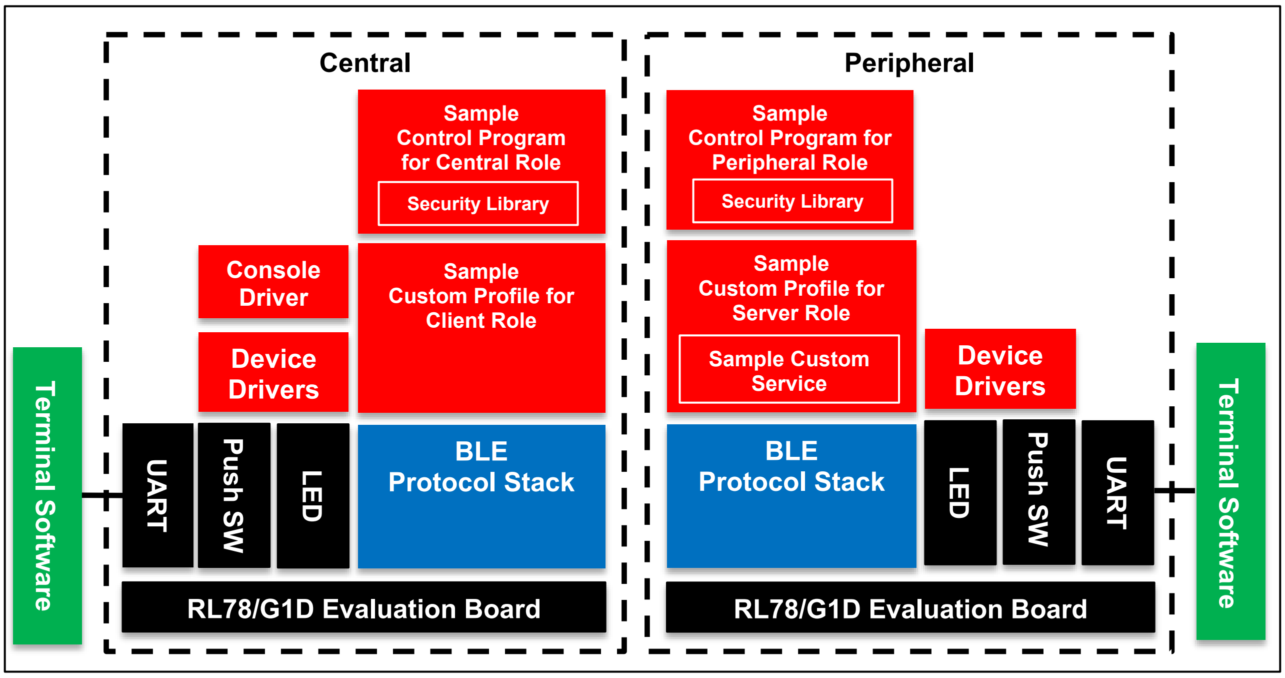


Figure 3-1 shows the software and hardware structure, which have three sections: Sample Program, BLE protocol stack, and Device. They are painted with Red, Blue and Black respectively. Since Central and Peripheral are associated with this demo, there are two types of the sample program, one for Central and the other for Peripheral. To enable security feature, Security Library (R01AN3777) is introduced for both of Central / Peripheral.

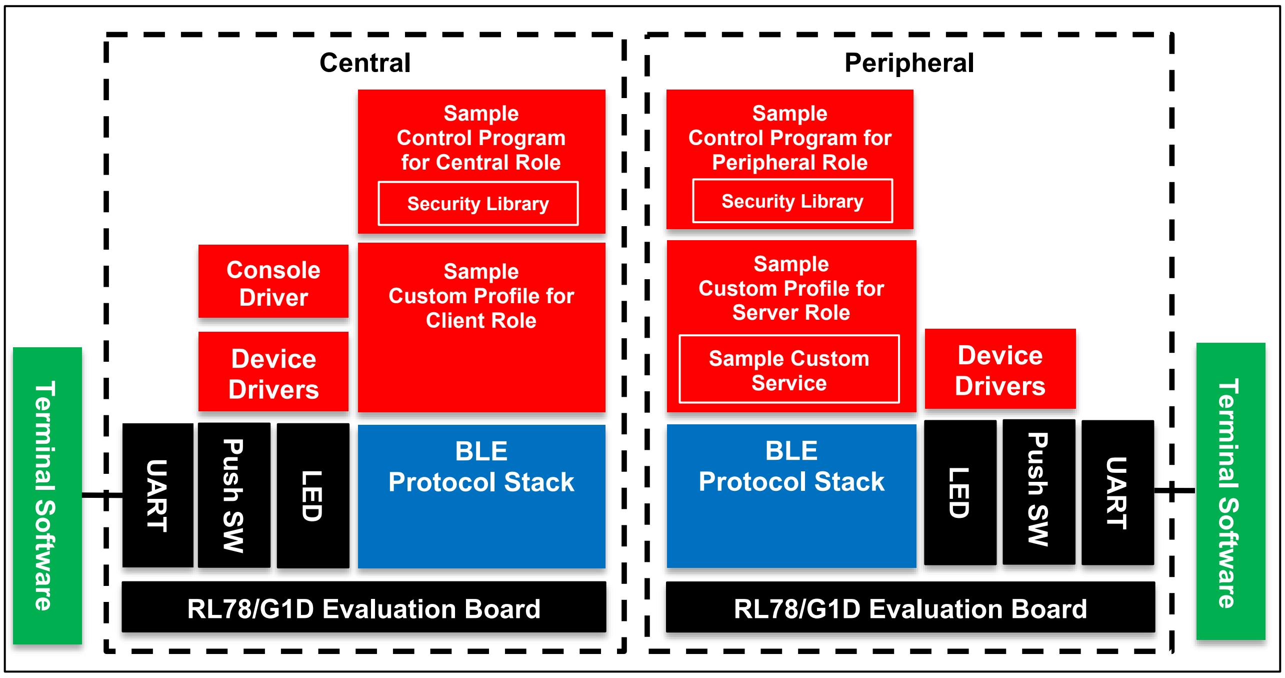


Figure 3-1 Software and Hardware Stack

For the sample program, the summaries of Central and Peripheral software are described as follows. Both of Central and Peripherals are implemented as the embedded configuration application.

- **Central**:
  - Sample Control Program for Central executes scanning to discover Peripheral and establishing a connection with Peripheral. After establishing the connection, it performs as GATT client role.
  - Sample Custom Profile for Client role provides APIs to ease of use Sample Custom Service.
  - This sample program includes the device drivers for Central, which LED (input/output port) and Console via UART.

- **Peripheral**:
  - Sample Control Program for Peripheral executes advertising after boot up and connection.
  - After establishing a connection, Peripheral performs as Server role and provides Sample Custom Service to Central.
  - Sample Custom Profile for Server role provides APIs to ease of use Sample Custom Service.
  - This sample program includes the device driver for Peripheral to support input/output port control for Push Switch and LED.

3.2 File and Directory Structure

The file and directory structure of the sample program is shown below. The package only includes the source codes or project files changed / added from BLE protocol stack V1.20 for the sample program.

```
¥---Project_Source
    +---rBLE
    |   ¥---src
    |       +---sample_app                               Sample Control Program
    |       |       console.c                            Console Drive
    |       |       console.h
    |       |       menu.c                               Menu Driver
    |       |       menu.h
    |       |       rble_sample_app_central.c            Sample Program for Central Role
    |       |       rble_sample_app_central.h
    |       |       rble_sample_app_peripheral.c         Sample Program for Peripheral Role
    |       |       rble_sample_app_peripheral.h
    |       |   ¥---seclib
    |       |           seclib.c
    |       |           seclib.h                         Security Library
    |       |           secdb.c
    |       |           secdb.h
    |       ¥---sample_profile
    |           ¥---sam                                  Sample Custom Profile
    |                   sam.h
    |                   samc.c                           Sample Custom Profile for Client Role
    |                   samc.h
    |                   sams.c                           Sample Custom Profile for Server Role
    |                   sams.h
    ¥---renesas
        +---src
        |   +---arch
        |   |   ¥---rl78
        |   |           arch_main.c                      BLE Software main loop
        |   |           db_handle.h                      GATT Service handle definition
        |   |           ke_conf.c                        Kernel task definition
        |   |           prf_config.c                     GATT Service definition
        |   |           prf_config.h
        |   |           prf_sel.h                        GATT Profile selection
        |   ¥---driver
        |       ¥---uart                                 UART 4800bps Configuration
        |               uart.c
        |       ¥---dataflash
        |               eel_descriptor_t02.c             Data Flash Library
        |               eel_descriptor_t02.h
        ¥---tools
            ¥---project
                +---CS_CCRL                              Project file for CS+ for CC
                |   +---BLE_Embedded_for_Peripheral
                |   ¥---BLE_Embedded_for_Central
                +---CubeSuite                            Project file for CS+ CA,CX
                |   +---BLE_Embedded_for_Peripheral
                |   ¥---BLE_Embedded_for_Central
                +---e2studio                             Project file for e$^{2}$ studio
                    +---BLE_Embedded_for_Peripheral
                    ¥---BLE_Embedded_for_Central
```

4. Building Sample Program

This section describes how to build the sample program. Using with one of the following development environment can build the sample program for demonstration.

- e^2 studio V4.3.1.001 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
- Renesas CS+ for CC V4.00.00 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
- Renesas CS+ for CA, CX V3.02.00 / Renesas CA78K0R V1.72

4.1 Common Procedure

Followings are the procedures to build the sample program.

1. You need BLE protocol stack and EEPROM emulation library. Download these from Renesas Web page.
    - BLE protocol stack
        - https://www.renesas.com/software-tool/bluetooth-low-energy-protocol-stack-rl78-family
    - EEPROM emulation library
        - EEPROM Emulation Library Pack02 Package Ver.2.00(for CA78K0R/CC-RL Compiler) for RL78 Family
            - https://www.renesas.com/software-tool/data-flash-libraries

    NOTE: The link address can be changed without notice.
2. Unzip BLE protocol stack. The path to unzip the package should not include white spaces or multi-byte characters. This manual uses “\BLE_Software_Ver_X_XX” as the target path.
3. Install EEPROM emulation library. Refer “Quick Start Guide [R01AN2767] section 4.2 Installing EEPROM Emulation Library” for the procedures.

4.2 Build Procedure

This section describes the build procedures of the sample program on each development environment, e2 studio, CS+.

4.2.1 e^2 studio

1. Launch e^2 studio.
2. Right click on “Project Explorer” and select “Import…” from the dropdown menu.
3. “Import” window is popped up and select “Existing Projects into Workspace” and click “Next >”.
4. Fill “Select root directory:” form with the project directory shown in Table 4-1.
5. Make sure that the project you selected is displayed in “Projects:” and click “Finish”. Then the window is closed.
6. Right click on the project just imported on “Project Explorer” and Select “Build Project” from the dropdown menu.
7. Refer Table 4-1 for the Hex file generate path.

4.2.2 CS+

1. Double click the project file shown in Table 4-1.
2. Right click on “BLE_Emb” in “Project Tree” and select “Build BLE_Emb” from the dropdown menu.
3. Refer Table 4-1 for the Hex file generate path.

Table 4-1 Project file and Hex File Location

e^2 studio with CC-RL	
Project Directory	**Peripheral:** e2studio\BLE_Embedded_for_Peripheral\rBLE_Emb **Central:** e2studio\BLE_Embedded_for_Central\rBLE_Emb
Hex File	**Peripheral:** e2studio\BLE_Embedded_for_Peripheral\rBLE_Emb\DefaultBuild\rBLE_Emb_CCRL.hex **Central:** e2studio\BLE_Embedded_for_Central\rBLE_Emb\DefaultBuild\rBLE_Emb_CCRL.hex
CS+ with CC-RL	
Project File	**Peripheral:** CS_CCRL\BLE_Embedded_for_Peripheral\BLE_Embedded_for_Peripheral.mtpj **Central:** CS_CCRL\BLE_Embedded_for_Central\BLE_Embedded_for_Central.mtpj
Hex File	**Peripheral:** CS_CCRL\BLE_Embedded_for_Peripheral\rBLE_Emb\DefaultBuild\rBLE_Emb_CCRL.hex **Central:** CS_CCRL\BLE_Embedded_for_Central\rBLE_Emb\DefaultBuild\rBLE_Emb_CCRL.hex
CS+ with CA78K0R	
Project File	**Peripheral:** CubeSuite\BLE_Embedded_for_Peripheral\BLE_Embedded_for_Peripheral.mtpj **Central:** CubeSuite\BLE_Embedded_for_Central\BLE_Embedded_for_Central.mtpj
Hex	**Peripheral:** CubeSuite\BLE_Embedded_for_Peripheral\BLE_Emb\DefaultBuild\BLE_Emb.hex **Central:** CubeSuite\BLE_Embedded_for_Central\BLE_Emb\DefaultBuild\BLE_Emb.hex

(Base Path: \BLE_Software_Ver_X_XX\RL78_G1D \Project_Source\renesas\tools\project\)

5. Sample Program Internals

This section describes the sample program component behavior, configuration and APIs.

5.1 Sample Control Program

Sample Control Program performs advertising (Peripheral), scanning and establishing a connection (Central) by using GAP API provided by BLE protocol stack. Regarding the API details and behavior, refer API Reference Manual: Basics (R01UW0088). And refer section 5.7 in this document for the API calling sequence of the sample program.

Following sub-sections describes about the Sample Control Program configurations.

5.1.1 Connection Configuration

After booting up or disconnection with Central, Peripheral performs advertising. Table 5-1 shows the advertising setting. This is defined in “rBLE\sample_app\rble_sample_peripheral.c: app_advertise_param”.

Table 5-1 Advertising Configuration

Advertising Type	Connectable Undirected Advertising (ADV_IND)
Advertising Interval Min	20 [ms]
Advertising Interval Max	30 [ms]
Advertising Channel Map	All Channels (37, 38, 39 [ch])
Advertising Data	-
<<Flags>> (0x01)	LE General Discoverable Mode BR/EDR Not Supported
<<Complete Local Name>> (0x09)	“Renesas-BLE”
<<Complete List of 128-bit Service Class UUIDs>> (0x07)	Sample Custom Service 128-bit UUID
Scan Response Data	none

Central performs scanning to discover the nearby Peripherals. After the scanning, Central displays a list of Peripherals discovered. This list does not include Peripherals which have no Sample Custom Profile UUID in its Advertising Packet.

Central establishes a connection with a Peripheral a user selected through the terminal menu. Table 5-2 shows the configuration of the connection. This is defined in “rBLE\src\sample_app\rble_sample_app_central.c: app_connection”.

Table 5-2 Connection Configuration

Scan Interval	60 [ms]
Scan Window Size	30 [ms]
Initiator Filter Policy	none
Minimum Connection Interval	30 [ms]
Maximum Connection Interval	50 [ms]
Connection Latency	0
Link Supervision Timeout	3 [s]
Minimum Connection Event Length	0 [ms]
Maximum Connection Event Length	50 [ms]

5.1.2 Central: Multiple Connection Configuration

Central can connect multiple Peripherals concurrently. The maximum number of the connections is defined by Table 5-3 macro definition. You can change the setting through IDE project setting.

Table 5-3 Macro Definition of the maximum number of connections

Macro Definition	Default Setting	Setting Range
CFG_CON	4	1~7 (Note)

NOTE: BLE Protocol stack allow to set 1~8 to CFG_CON. With this application, the range is 1~7. This is because the memory usage limitation.

5.1.3 Peripheral: SW4 PUSH/RELEASE state notification Interval

Peripheral sends SW4 PUSH/RELEASE state every specified interval. The interval configuration is determined by Table 5-4 macro definition. This is defined in "rBLE\src\sample_app\rble_sample_peripheral.c"

Table 5-4 Macro Definition of SW4 State Notification Interval

Macro Definition	Default Configuration
APP_SWITCH_STATE_CHECK_INTERVAL	50 (500ms interval)

5.1.4 Security Settings

Table 5-5 shows the security settings of Central and Peripheral. Refer Security Library (R01AN3777) for the details of these parameters.

Table 5-5 Security Settings

Security Parameters	Central Settings	Peripheral Settings
role	RBLE_MASTER	RBLE_SLAVE
auth_req	RBLE_AUTH_REQ_MITM_BOND	RBLE_AUTH_REQ_MITM_BOND
iocap	RBLE_IO_CAP_KB_ONLY	RBLE_IO_CAP_DISPLAY_ONLY
rpa_generate	TRUE	FALSE

5.1.5 The number of storable Bonding Information

The macro definition shown in Table 5-6 determines the number of storable bonding information into Data Flash. If you change the number, Data Flash Library also shall be checked. Refer Security Library (R01AN3777) for the detail.

Table 5-6 Macro definition of the number of bonding information

Macro Definition	Central	Peripheral
CFG_SECLIB_BOND_NUM	4	4

5.1.6 Security Settings

The definition shown in Table 5-7 determines enable or disable security settings, such as encryption or privacy. The default setting is enable. To disable the setting, delete the definition and remove Security Library files (seclib.c, secdb.c) from the build target.

Table 5-7 Security settings

Macro Definition
USE_SECLIB

5.2 Console Driver

Console Driver provides following two functionalities. Console Driver uses UART Driver internally.

Console Input: Console Driver receives user input characters to Terminal Software via UART. And send the characters to the specified task line at a time.

Console Output: Console Driver output characters, such as the sample program log output or the terminal menu to Terminal Software via UART.

5.2.1 Console Input Handling

Console Driver receives characters from Terminal Software and sends the characters to the specified task line at a time. A line end with CR/LF character or Esc character. Console Driver stores a character to internal buffer and when it receives the CR/LR character or Esc character, sends stored characters to the specified task.

The line transmission to the specified task is performed by RWKE messaging functionality. Console Driver sends a message with CONSOLE_MSG_LINE_IN message ID (Refer section 5.2.3(a)), and the content is described with CONSOLE_MSG structure explained (Refer section 5.2.3(b)).

5.2.2 API

(a) **console_init**

void console_init(bool enable_in, ke_task_id_t task_id)			
- Initialize Console Driver			
Parameters:			
bool	enable_in	true	Use console input
		false	Not use console input
ke_task_id	task_id	The task console input notified	

(b) **console_enable_in**

void console_enable_in(void)
- Enable console input disabled by console_disable_in. - Do not call this API for the console initialized with enable_in as false.

(c) **console_disable_in**

void console_disable_in(void)
- This API disable console input temporary to avoid the unintentional user input. - Esc key can forcibly enable the console input.

(d) **Printf**

void Printf(const char_t *fmt, ...)
- This API provides the same functionality with printf.

5.2.3 Definition

(a) CONSOLE_MSG_LINE_IN

The message ID of the console input

```
#define CONSOLE MSG LINE IN (KE FIRST MSG(TASK CON APPL) + 2)
```

(b) CONSOLE_MSG

The message content structure. "len" is the length of "buf" array. The size of memory region for CONSOLE_MSG is determined by the "len". The sample program calculates the size by "sizeof(CONSOLE_MSG) + len", and dynamically allocates the region by ke_msg_alloc API.

```
typedef struct {
  uint8_t len;       // Message Data Length
  char_t  buf[1];    // Message Data
} CONSOLE MSG;
```

5.3 Menu Driver

Menu Driver controls the terminal menu displayed on Terminal Software.

5.3.1 Menu Structure

As shown in Figure 5-1, the menu has hierarchy structure.

- LIST has children (like a directory). A child menu is LIST or ITEM. When user select a LIST, the children are displayed.
- ITEM has specific behavior (like a file). When a user select an ITEM, it executes a handler bounded to the ITEM.
- SINGLE is a kind of LIST which have only one child menu. When a user select a SINGLE, it executes the child handler.
- TERMINATOR is the last child of LIST. TERMINATOR is needed due to the implementation purpose. TERMINATOR never shows to the menu on Terminal Software.

```
LIST
  |
  +---- LIST
  |     +---- ITEM
  |     +---- ITEM
  |     +---- TERMINATOR
  |
  +---- ITEM
  |
  +---- SINGLE
  |     +---- ITEM
  |     +---- TERMINATOR
  |
  +---- TERMINATOR
```

Figure 5-1 Menu Structure

5.3.2 Menu Control

Each menu has index. A user can select the menu by input the index and Enter key. Esc key is used to move up to the higher menu. During the menu selection, the console input is disabled to avoid unintentional user input.

5.3.3 API

(a) **menu_show**

void menu_show(MENU *menulist)		
- Display the menulist on Terminal Software.		
Parameters:		
MENU	menulist	LIST to display

(b) **menu_user_in**

```
int_t menu_user_in(ke_msg_id_t const msgid, void const *param,
                   ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

- A Handler for the console input kernel message.
- Register this handler for CONSOLE_MSG_LINE_IN.

5.3.4 Structure

(a) **MENU_TYPE**

MENU_TYPE enumeration is the type of menu element.

```
typedef enum {
  MENU_TYPE_TERMINATOR,
  MENU_TYPE_LIST,
  MENU_TYPE_SINGLE,
  MENU_TYPE_ITEM,
} MENU_TYPE;
```

(b) **MENU**

MENU structure represents one menu element. Each MENU_TYPE have different member to set.

```
typedef struct MENU_t {
  // for LIST, SINGLE, ITEM
  MENU_TYPE      type;                        // Menu Type
  char_t         title[MENU_TITLE_SIZE]; // Menu Title

  // for LIST, SINGLE
  struct MENU_t      *parent;    // Parent menu
  struct MENU_t      *children;  // Children menu
  MENU_UPDATE_HANDLER update;   // Handler to update LIST children dynamically
  MENU_CANCEL_HANDLER cancel;   // Handler which is called when Esc key pressed

  // for ITEM
  MENU_HANDLER       handler;     // ITEM Handler
} MENU;
```

(c) **MENU_HANDLER**

The handler executed when a user select ITEM.

```
typedef void (*MENU_HANDLER)(void *arg)
```

(d) **MENU_UPDATE_HANDLER**

The handler called to dynamically update LIST children. This is called just before displaying the LIST children. The sample program uses this handler to update the result of nearby Peripheral search.

```
typedef void (*MENU_UPDATE_HANDLER)(void)
```

(e) **MENU_CANCEL_HANDLER**

This handler called when Esc key is pressed. The sample program uses this handler to cancel passkey input.

```
typedef void (*MENU_CANCEL_HANDLER)(void)
```

5.4 Sample Custom Service Definition

Peripheral provides Sample Custom Service. Table 5-8 shows the definition of Sample Custom Service.

Table 5-8 Sample Custom Service Definition

Type		Value	Permission
Sample Custom Service			
	Primary Service Declaration (0x2800)	128-bit UUID	Read
Switch State Characteristic			
	Characteristic Declaration (0x2803)	Prop: Notification Type: 128-bit UUID	Read
	Characteristic Value	uint8_t	Notification
	Client Characteristic Configuration Descriptor (0x2902)	uint16_t	Read, Write
LED Control Characteristic			
	Characteristic Declaration (0x2803)	Prop: Read, Write Type: 128-bit UUID	Read
	Characteristic Value	uint8_t	Read, Write

- **Switch State Characteristic**: This characteristic is used to send Notification of SW PUSH/RELEASE state. Table 5-9 shows field/bit definition of the characteristic. This characteristic has Client Characteristic Configuration Descriptor to enable/disable Notification.

Table 5-9 Switch State Characteristic Field/Bit Definition

Bit	Definition	Key	Value
0	Switch State	0	RELEASE
		1	PUSH
1 to 7	Reserved For Future Use	-	-

- **LED Control Characteristic**: This characteristic is used to control LED ON/OFF state. Table 5-10 shows field/bit definition of this characteristic.

Table 5-10 LED Control Field/Bit Definition

Bit	Definition	Key	Value
0	LED State	0	OFF
		1	ON
1 to 7	Reserved For Future Use	-	-

5.5 Sample Custom Profile for Server role (SAMS)

SAMS provides API, Event and Structure by wrapping GATT API to ease the usage of Sample Custom Service. Refer section 5.7 for the GATT APIs SAMS internally called.

The prefix "SAMPLE_" means it is defined for the sample program. These definitions cannot be used outside of the sample program.

5.5.1 API

(a) **SAMPLE_Server_Enable**

RBLE_STATUS SAMPLE_Server_Enable (uint16_t conhdl, uint8_t con_type, SAMPLE_SERVER_PARAM *param, SAMPLE_SERVER_EVENT_HANDLER callback)			
- This API enables Sample Custom Profile Server Role. The completion of the API is informed by SAMPLE_SERVER_EVENT_ENABLE_COMP event. - con_type specifies the SAMS setting value initialization. If con_type is RBLE_PRF_NORMAL, SAMS setting value is initialized with param. If con_type is RBLE_PRF_DISCOVERY, it is not initialized. - param is the SAMS setting value. - callback is called when SAMS event happen.			
Parameters:			
uint16_t	conhdl	Connection Handle	
uint8_t	con_type	RBLE_PRF_NORMAL	Initialize SAMS setting value with param
		RBLE_PRF_DISCOVERY	Do not initialize SAMS setting value
SAMPLE_SERVER_PARAM *	param	the SAMS setting value	
SAMPLE_SERVER_EVENT_HANDLE	callback	SAMS Event Handler	
Return:			
RBLE_OK	Success		
RBLE_PARAM_ERR	Parameter Error		
RBLE_STATUS_ERROR	Status Error		

(b) **SAMPLE_Server_Disable**

RBLE_STATUS SAMPLE_Server_Disable (uint16_t conhdl)		
- This API disables SAMS. The completion of the API is informed by SAMPLE_SERVER_EVENT_DISABLE_COMP event.		
Parameters:		
uint16_t	conhdl	Connection Handle
Return:		
RBLE_OK	Success	
RBLE_PARAM_ERR	Parameter Error	
RBLE_STATUS_ERROR	Status Error	

(c) **SAMPLE_Server_Send_Switch_State**

RBLE_STATUS SAMPLE_Server_Send_Switch_State (uint16_t conhdl, uint8_t value)				
- This API is used to notify the Client of the switch state characteristic value. - value is the switch state to notify.				
Parameters:				
	uint16_t	conhdl	Connection Handle	
	uint8_t	value	SAMPLE_SWITCH_STATE_ON	Switch is PUSH state
			SAMPLE_SWITCH_STATE_OFF	Switch is RELEASE state
Return:				
	RBLE_OK	Success		
	RBLE_STATUS_ERROR	Status Error		

5.5.2 Event

(a) **SAMPLE_SERVER_EVENT_ENABLE_COMP**

SAMPLE_SERVER_ENABLE_COMP			
- The event to inform the completion of SAMS enabling.			
Parameters:			
	RBLE_STATUS	status	The result of the Server role enabling
	uint16_t	conhdl	Connection Handle

(b) **SAMPLE_SERVER_EVENT_DISABLE_COMP**

SAMPLE_SERER_DISABLE_COMP			
- The event to inform the completion of SAMS disabling.			
Parameters:			
	SAMPLE_SERVER_PARM	param	The SAMS setting value which is used before the disable
	uint16_t	conhdl	Connection Handle

(c) **SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND**

SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND				
- The event to inform the change of led_control characteristic value.				
Parameters:				
	uint8_t	value	SAMPLE_PRF_LED_CONTROL_ON	LED is ON
			SAMPLE_PRF_LED_CONTROL_OFF	LED is OFF

(d) **SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE**

SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE				
- The event to inform the change of switch sate characteristic value.				
Parameters:				
	uint16_t	char_code	SAMPLE_PRF_SWITCH_STATE_CCCD_CODE	The characteristic changed
	uint16_t	cccd_val	SAMPLE_PRF_STOP_NTFIND	Notification/Indication are disabled
			SAMPLE_PRF_START_NTF	Notification is enabled

5.5.3 Definition

(a) **SAMPLE_SERVER_EVENT_TYPE**

The definition of SAMS event IDs.

```
typedef enum {
  SAMPLE_SERVER_EVENT_ENABLE_COMP = 0,
  SAMPLE_SERVER_EVENT_DISABLE_COMP,
  SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND,
  SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE,
} SAMPLE_SERVER_EVENT_TYPE;
```

(b) **SAMPLE_CLIENT_PARAM**

The definition of SAMS setting value.

```
typedef struct {
  uint16_t switch_state_cccd;// The value of switch state CCCD
} SAMPLE_CLIENT_PARAM;
```

(c) **SAMPLE_SERVER_EVENT_HANDLER**

The definition of SAMS event handler.

```
typedef void (*SAMPLE_SERVER_EVENT_HANDLER)(SAMPLE_SERVER_EVENT *event);
```

(d) **SAMPLE_SERVER_EVENT**

The SAMS event structure.

```
typedef struct {
  SAMPLE_SERVER_EVENT_TYPE type;    // Event Type
  RBLE_STATUS status;               // SAMS execution status
  uint16_t conhdl;                  // The Connection Handle event happened

  union {
    struct {                        // SAMPLE_SERVER_EVENT_DISABLE_COMP
      SAMPLE_SERVER_PARAM param; // The configuration value of SAMS
    } disable_comp;

    struct {            // SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND
      uint8_t value;    // Updated Led Control Characteristic Value
    } change_led_control_ind;

    struct {            // SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE
      uint16_t cccd;    // Updated Switch State CCCD
    } write_char_resp;
  } param;
} SAMPLE_SERVER_EVENT;
```

5.6 Sample Custom Profile for Client Role (SAMC)

SAMC provides API, Event and Structure by wrapping GATT API to ease the usage of Sample Custom Service. Refer section 5.7 for the GATT API SAMC internally called.

The prefix "SAMPLE_" means the definition is defined in Sample Program. These definitions cannot be used outside of the sample program.

5.6.1 API

(a) **SAMPLE_Client_Enable**

```
RBLE_STATUS SAMPLE_Client_Enable (uint16_t conhdl,
  SAMPLE_CLIENT_CON_TYPE con_type,
  SAMPLE_CLIENT_CONTENT *content, SAMPLE_CLIENT_EVENT_HANDLE callback)
```

- This API enables SAMC. The completion of the enabling is informed by SAMPLE_CLIENT_EVENT_ENABLE_COMP.
- When con_type is RBLE_PRF_CON_DISCOVERY, this perform service discovery procedure. When con_type is RBLE_PRF_CON_NORMAL, this does not perform service discovery procedure instead uses handles specified by content.
- content is the service attribute information. This is used only when con_type is RBLE_PRF_CON_NORMAL.
- callback is called SAMC event happen.

Parameters:

uint16_t	conhdl	Connection Handle	
uint16_t	con_type	RBLE_PRF_NORMAL	perform service discovery
		RBLE_PRF_DISCOVERY	do not perform service discovery
SAMPLE_CLIENT_CONTENT *	content	service handles	
SAMPLE_CLIENT_EVENT_HANDLE	callback	SAMPLE Client Role Event Handler	

Return:

RBLE_OK	Success
RBLE_PARAM_ERR	Parameter Error
RBLE_STATUS_ERROR	Status Error

(b) **SAMPLE_Client_Disable**

```
RBLE_STATUS SAMPLE_Client_Disable (uint16_t conhdl)
```

- This API disables SAMC. The completion of the API is informed by SAMPLE_CLIENT_EVENT_DISABLE_COMP event.

Parameters:

uint16_t	conhdl	connection Handle

Return:

RBLE_OK	Success
RBLE_STATUS_ERROR	Status Error

(c) **SAMPLE_Client_Write_Led_Control**

RBLE_STATUS SAMPLE_Client_Write_Led_Control (uint8_t value)			
- This API write to LED control characteristic value.			
Parameters:			
uint8_t	value	SAMPLE_PRF_LED_CONTROL_ON	LED ON
		SAMPLE_PRF_LED_CONTROL_OFF	LED OFF
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Status Error		

(d) **SAMPLE_Client_Write_Char**

RBLE_STATUS SAMPLE_Client_Write_Char (uint16_t char_code, uint16_t cfg_val)			
- This API writes to Switch State CCCD (Client Characteristic Configuration Descriptor).			
Parameters:			
uint16_t	char_code	SAMPLE_CLIENT_WR_SWITCH_STATE_CCCD_CODE	Write target
uint16_t	cccd_val	RBLE_PRF_STOP_NTFIND	Disable Notification/Indication
		RBLE_PRF_START_NTF	Enable Notification
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Status Error		

(e) **SAMPLE_Client_Read_Char**

RBLE_STATUS SAMPLE_Client_Read_Char (uint16_t conhdl, uint8_t char_code)			
- This API reads the characteristic value specified by char_code.			
Parameters:			
uint16_t	conhdl	Connection Handle	
uint8_t	char_code	SAMPLE_CLIENT_RD_LED_CONTROL_CODE	Read LED Control characteristic
		SAMPLE_CLIENT_RD_SWITCH_STATE_CCCD_CODE	Read Switch state cccd characteristic
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Status Error		

5.6.2 Event

(a) **SAMPLE_CLIENT_EVENT_ENABLE_COMP**

SAMPLE_SERVER_ENABLE_COMP		
- The event to inform the completion of SAMC enabling.		
Parameters:		
RBLE_STATUS	status	The result of SAMPLE Client Role enabling
SAMPLE_CLIENT_CONTENT *	param	Sample Custom Service Attribute information
uint16_t	conhdl	Connection Handle

(b) **SAMPLE_CLIENT_EVENT_DISABLE_COMP**

SAMPLE_CLIENT_DISABLE_COMP		
- The event to inform the completion of SAMC disabling.		
Parameters:		
RBLE_STATUS	status	The result of SAMPLE Client Role disabling
uint16_t	conhdl	Connection Handle

(c) **SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND**

SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND		
- The event to inform the reception of switch state notification.		
Parameters:		
uint16_t	conhdl	Connection Handle
uint8_t	value	The switch state notified

(d) **SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE**

SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE		
- The event to inform the completion of Write command.		
Parameters:		
uint16_t	conhdl	Connection Handle
uint8_t	att_code	The result of the write command

(e) **SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE**

SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE		
- The event to inform the completion of Read command.		
Parameters:		
uint16_t	conhdl	Connection Handle
uint8_t	att_code	The result of the read command
RBLE_GATT_INFO_DATA	data	read value

5.6.3 Definition

(a) **SAMPLE_CLIENT_EVENT_TYPE**

The SAMC event IDs.

```
typedef enum {
  SAMPLE_CLIENT_EVENT_ENABLE_COMP,
  SAMPLE_CLIENT_EVENT_DISABLE_COMP,
  SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND,
  SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE,
  SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE,
} SAMPLE_CLIENT_EVENT_TYPE;
```

(b) **SAMPLE_CLIENT_EVENT_HANDLE**

The SAMC event handler.

```
typedef void (*SAMPLE_CLIENT_EVENT_HANDLE)(SAMPLE_CLIENT_EVENT *event);
```

(c) **SAMPLE_CLIENT_CONTENT**

The structure to save Attribute information acquired by discovery.

```
typedef struct {
  uint16_t start_hdl;      // Service Start Handle
  uint16_t end_hdl;        // Service End Handle
  uint16_t switch_state_char_hdl; // Switch State Characteristic Handle
  uint16_t switch_state_val_hdl;  // Switch State Value Char Handle
  uint16_t switch_state_prop;     // Switch State Characteristic Property
  uint16_t switch_state_cccd_hdl; // Switch State Characteristic CCCD
  uint16_t led_control_char_hdl;  // LED Control Characteristic Handle
  uint16_t led_control_val_hdl;   // LED Control Value Characteristic Handle
  uint16_t led_control_prop;      // LED Control Characteristic Property
} SAMPLE_CLIENT_CONTENT;
```

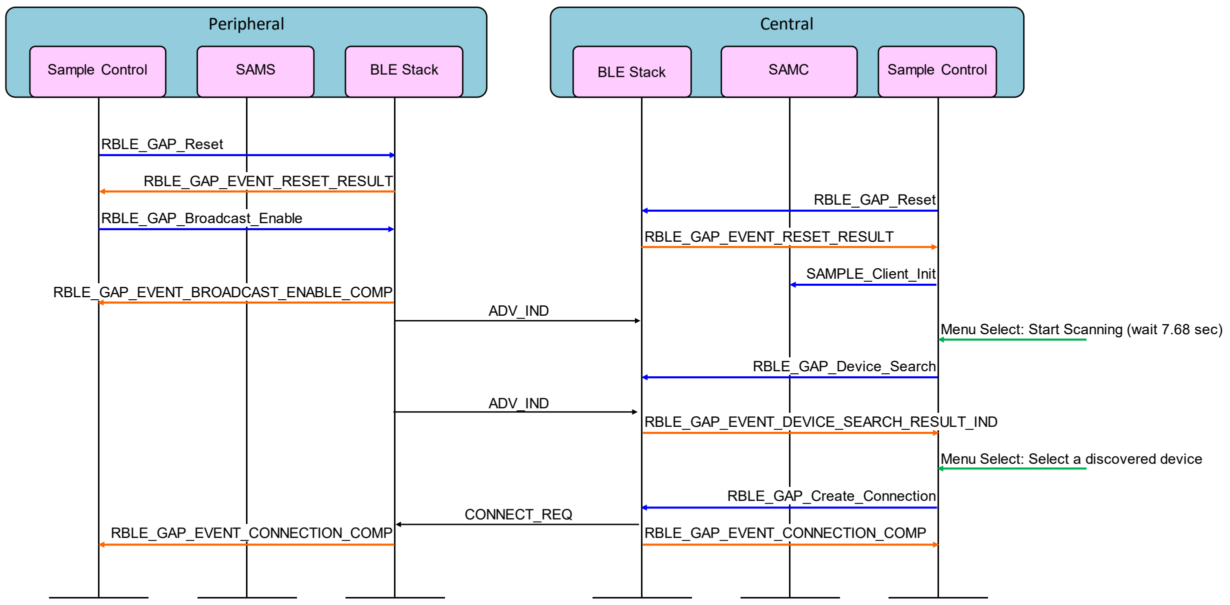
(d) **SAMPLE_CLIENT_EVENT**

The SAMC event structure.

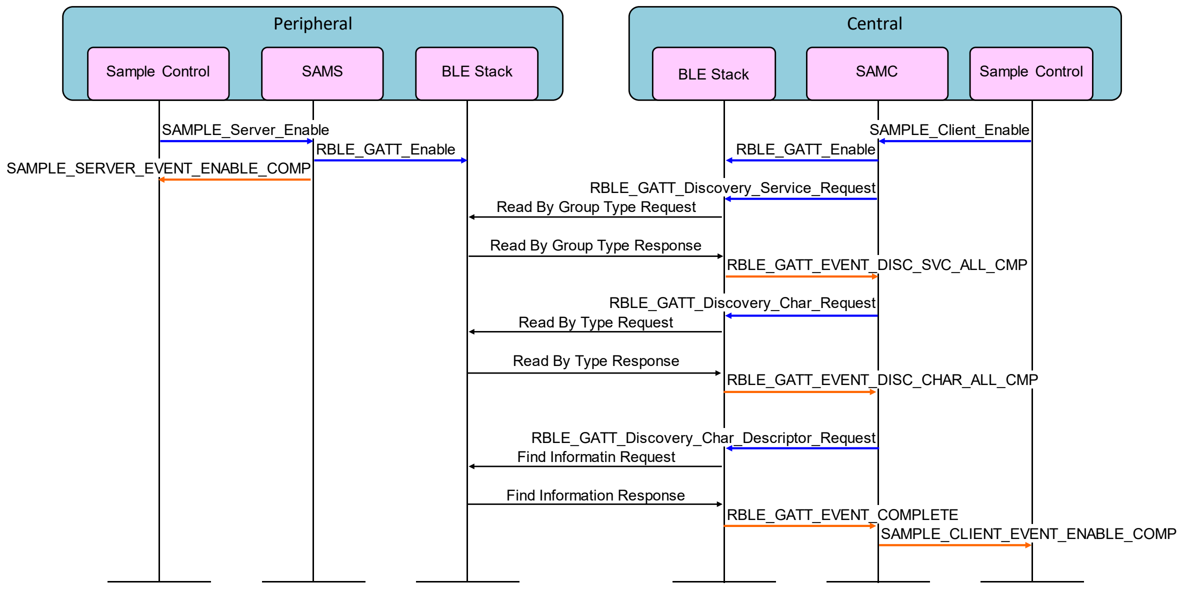
```
typedef struct {
  SAMPLE_CLIENT_EVENT_TYPE type;    // Event Type
  RBLE_STATUS status;               // SAMC execution Result
  uint16_t conhdl;                  // The Connection Handle event happened
  union {
    struct {                        // SAMPLE_CLIENT_EVENT_ENABLE_COMP
      SAMPLE_CLIENT_CONTENT samc;   // SAMC Discovery Result
    } enable_comp;
    struct {            // SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND
      uint8_t value;    // The value notified
    } switch_state_ind;
    struct {            // SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE
      uint8_t value;    // The result of Read command
    } read_char_resp;
  } param;
} SAMPLE_CLIENT_EVENT;
```

5.7 Sequence Chart

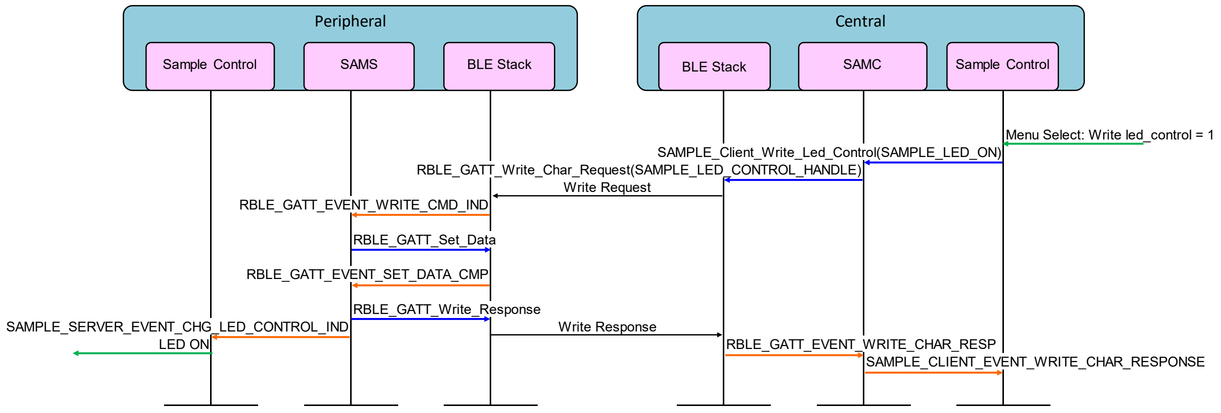
- Booting up to establishing a connection


Peripheral
Sample Control
SAMS
BLE Stack
Central
BLE Stack
SAMC
Sample Control
RBLE_GAP_Reset
RBLE_GAP_EVENT_RESET_RESULT
RBLE_GAP_Broadcast_Enable
RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP
RBLE_GAP_Reset
RBLE_GAP_EVENT_RESET_RESULT
SAMPLE_Client_Init
ADV_IND
Menu Select: Start Scanning (wait 7.68 sec)
RBLE_GAP_Device_Search
ADV_IND
RBLE_GAP_EVENT_DEVICE_SEARCH_RESULT_IND
Menu Select: Select a discovered device
RBLE_GAP_Create_Connection
CONNECT_REQ
RBLE_GAP_EVENT_CONNECTION_COMP
RBLE_GAP_EVENT_CONNECTION_COMP

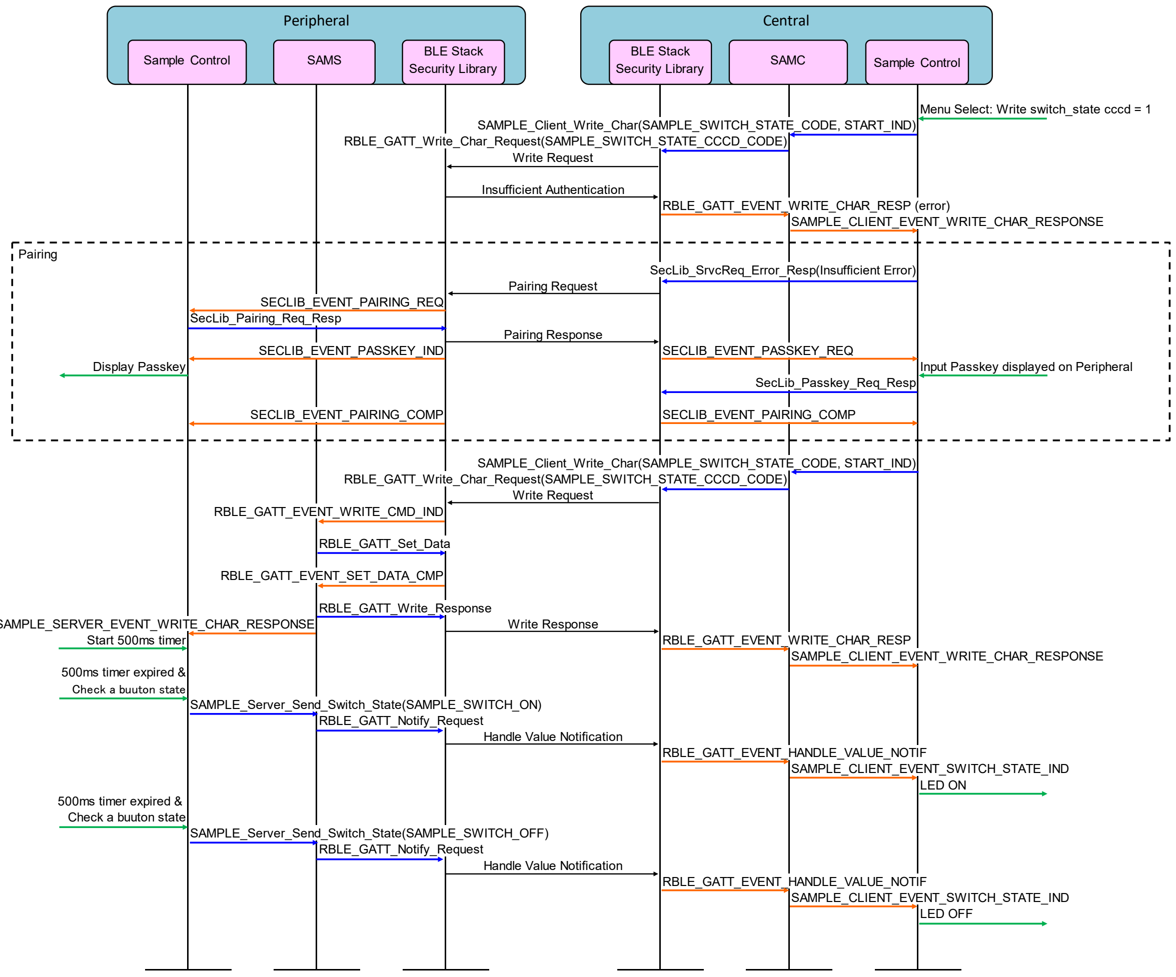
- Enabling of Sample Custom Profile


Peripheral
Sample Control
SAMS
BLE Stack
Central
BLE Stack
SAMC
Sample Control
SAMPLE_Server_Enable
RBLE_GATT_Enable
SAMPLE_SERVER_EVENT_ENABLE_COMP
SAMPLE_Client_Enable
RBLE_GATT_Enable
RBLE_GATT_Discovery_Service_Request
Read By Group Type Request
Read By Group Type Response
RBLE_GATT_EVENT_DISC_SVC_ALL_CMP
RBLE_GATT_Discovery_Char_Request
Read By Type Request
Read By Type Response
RBLE_GATT_EVENT_DISC_CHAR_ALL_CMP
RBLE_GATT_Discovery_Char_Descriptor_Request
Find Informatin Request
Find Information Response
RBLE_GATT_EVENT_COMPLETE
SAMPLE_CLIENT_EVENT_ENABLE_COMP

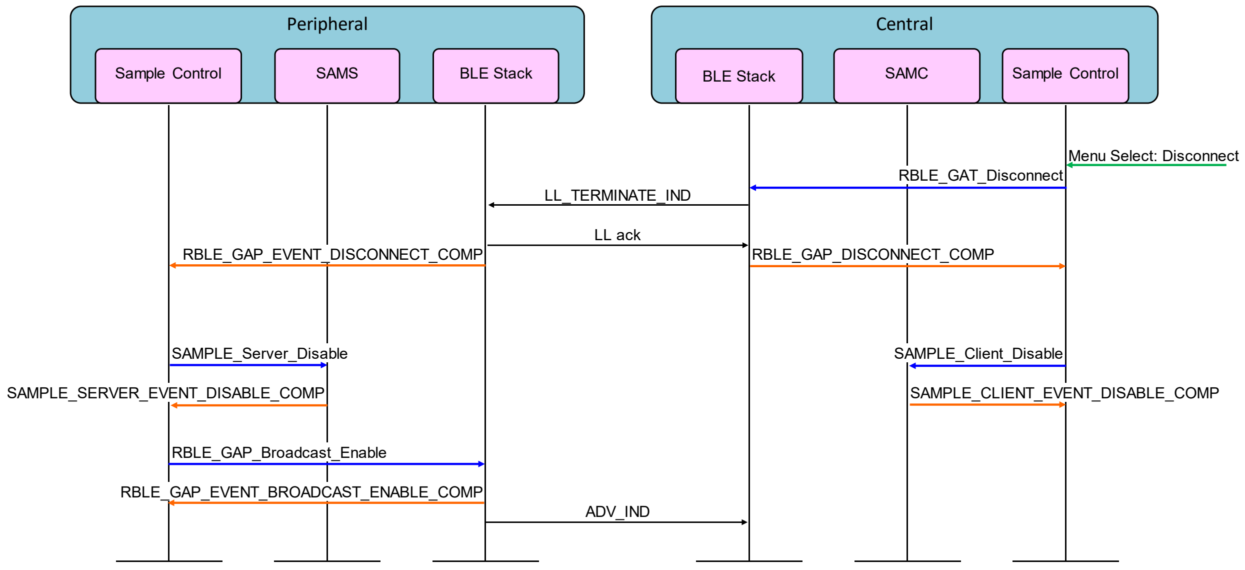
- LED4 Control


Peripheral
Sample Control
SAMS
BLE Stack
Central
BLE Stack
SAMC
Sample Control
Menu Select: Write led_control = 1
SAMPLE_Client_Write_Led_Control(SAMPLE_LED_ON)
RBLE_GATT_Write_Char_Request(SAMPLE_LED_CONTROL_HANDLE)
Write Request
RBLE_GATT_EVENT_WRITE_CMD_IND
RBLE_GATT_Set_Data
RBLE_GATT_EVENT_SET_DATA_CMP
RBLE_GATT_Write_Response
SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND
LED ON
Write Response
RBLE_GATT_EVENT_WRITE_CHAR_RESP
SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE

- SW4 state Notification


Peripheral
Sample Control
SAMS
BLE Stack
Security Library
Central
BLE Stack
Security Library
SAMC
Sample Control
Menu Select: Write switch_state ccd = 1
SAMPLE_Client_Write_Char(SAMPLE_SWITCH_STATE_CODE, START_IND)
RBLE_GATT_Write_Char_Request(SAMPLE_SWITCH_STATE_CCCD_CODE)
Write Request
Insufficient Authentication
RBLE_GATT_EVENT_WRITE_CHAR_RESP (error)
SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE
Pairing
SecLib_SrvcReq_Error_Resp(Insufficient Error)
Pairing Request
SECLIB_EVENT_PAIRING_REQ
SecLib_Pairing_Req_Resp
Pairing Response
SECLIB_EVENT_PASSKEY_IND
SECLIB_EVENT_PASSKEY_REQ
Display Passkey
Input Passkey displayed on Peripheral
SecLib_Passkey_Req_Resp
SECLIB_EVENT_PAIRING_COMP
SECLIB_EVENT_PAIRING_COMP
SAMPLE_Client_Write_Char(SAMPLE_SWITCH_STATE_CODE, START_IND)
RBLE_GATT_Write_Char_Request(SAMPLE_SWITCH_STATE_CCCD_CODE)
Write Request
RBLE_GATT_EVENT_WRITE_CMD_IND
RBLE_GATT_Set_Data
RBLE_GATT_EVENT_SET_DATA_CMP
RBLE_GATT_Write_Response
SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE
Write Response
Start 500ms timer
RBLE_GATT_EVENT_WRITE_CHAR_RESP
SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE
500ms timer expired &
Check a buuton state
SAMPLE_Server_Send_Switch_State(SAMPLE_SWITCH_ON)
RBLE_GATT_Notify_Request
Handle Value Notification
RBLE_GATT_EVENT_HANDLE_VALUE_NOTIF
SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND
LED ON
500ms timer expired &
Check a buuton state
SAMPLE_Server_Send_Switch_State(SAMPLE_SWITCH_OFF)
RBLE_GATT_Notify_Request
Handle Value Notification
RBLE_GATT_EVENT_HANDLE_VALUE_NOTIF
SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND
LED OFF

- Connection Disconnection and Disabling Sample Custom Profile



6. Appendix

6.1 ROM size, RAM size

The ROM and RAM size used by the sample software are shown in Figure 6-1. These sizes are measured with the multiple connection configuration to 4.

Figure 6-1 ROM and RAM size

Compiler	Central		Peripheral	
	ROM size	RAM size	ROM size	RAM size
RL78 Family C Compiler Package V1 V1.03.00	160,524	12,739	154,400	7,307
Renesas CA78K0R V1.72	131,719	12,699	127,178	7,219

(in bytes)

Website and Support

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

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul 14, 2016	-	Initial revision
1.10	Oct 10, 2016	5	2.1 Environment : Add supported development environments descriptions. Remove Software Library description due to the duplication with Section 4.1.
		15	
		33	Build Procedure : Add IARv2 description.
			6.1 ROM size, RAM size : Add IARv2 ROM/RAM size.
1.20	Mar 1, 2017		All changes related to introduction of security feature.
		4	1 Introduction
		6	2.2 Environment Setup
		8	2.3 Operation
		11	3 Structure
		13	3.2 File and Directory Structure
		17	5.1.4 Security Settings
		17	5.1.5 The number of storable Bonding Information
		30	5.7 Sequence Chart
		33	6.1 ROM size, RAM size
1.20	Jan 31, 2022	-	Fixed due to the end of IAR support in Bluetooth Low Energy Protocol Stack.

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. 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 LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
     In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
     In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the 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.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141