Introduction

This Sample Program runs on Bluetooth® Low Energy microcontroller RL78/G1D device, and executes Advertising for providing information and executes connecting to smart phone in order to update configuration and data.

The Sample Program switches Beacon Application and Connect Application alternately. Beacon Application executes low power consumption Advertising. Connect Application works as GAP peripheral role, and executes pairing for establishing secure connection as well as communication with custom profile. As an example, the custom profile is implemented for updating configuration and data of beacon. It is not limited to this updating beacon but also possible to extend various usage.

Target Device

RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)

Related documents

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Document No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RL78/G1D</td>
<td></td>
</tr>
<tr>
<td>User's Manual: Hardware</td>
<td>R01UH0515E</td>
</tr>
<tr>
<td>RL78/G1D Evaluation Board</td>
<td>R30UZ0048E</td>
</tr>
<tr>
<td>User's Manual</td>
<td></td>
</tr>
<tr>
<td>E1 Emulator</td>
<td></td>
</tr>
<tr>
<td>User’s Manual</td>
<td>R20UT0398E</td>
</tr>
<tr>
<td>Renesas Flash Programmer V3.02 Flash memory programming software</td>
<td>R20UT3841E</td>
</tr>
<tr>
<td>User's Manual</td>
<td></td>
</tr>
<tr>
<td>CC-RL Compiler</td>
<td></td>
</tr>
<tr>
<td>User’s Manual</td>
<td>R20UT3123E</td>
</tr>
<tr>
<td>Bluetooth Low Energy Protocol Stack</td>
<td></td>
</tr>
<tr>
<td>User's Manual</td>
<td>R01UW0095E</td>
</tr>
<tr>
<td>API Reference Manual: Basics</td>
<td>R01UW0088E</td>
</tr>
<tr>
<td>RL78/G1D Beacon Stack</td>
<td></td>
</tr>
<tr>
<td>User's Manual</td>
<td>R01UW0171E</td>
</tr>
<tr>
<td>RL78 Family Flash Self-Programming Library Type01</td>
<td></td>
</tr>
<tr>
<td>User's Manual</td>
<td>R01US0050E</td>
</tr>
<tr>
<td>RL78 Family EEPROM Emulation Library Pack02</td>
<td></td>
</tr>
<tr>
<td>User’s Manual</td>
<td>R01US0068E</td>
</tr>
</tbody>
</table>
Contents

1. Overview ......................................................................................................................... 5
   1.1 Beacon Operation ....................................................................................................................... 6
   1.2 RF Evaluation Operation ............................................................................................................ 6

2. Environment ................................................................................................................... 7

3. File Composition ............................................................................................................ 8

4. Evaluation Procedure ................................................................................................... 10
   4.1 Getting Libraries ....................................................................................................................... 11
   4.2 Building Firmware ..................................................................................................................... 12
      4.2.1 Using CS+ for CC ........................................................................................................ 12
      4.2.2 Using Renesas e² studio .................................................................................................... 12
   4.3 Writing Firmware ....................................................................................................................... 13
   4.4 Evaluating with smart phone ................................................................................................... 15
      4.4.1 Confirming the transmission of advertising packet ........................................................ 16
      4.4.2 Updating the advertising packet ....................................................................................... 17
      4.4.3 Confirming the updated advertising packet ..................................................................... 21
   4.5 Evaluating RF characteristic .................................................................................................... 22
   4.6 Current Consumption Measurement ....................................................................................... 23
      4.6.1 Measurement Environment ................................................................................................ 23
      4.6.2 Evaluation Board Setting ................................................................................................... 24
      4.6.3 Measurement Procedure .................................................................................................... 24

5. Specification ................................................................................................................. 25
   5.1 Beacon Application ................................................................................................................... 25
      5.1.1 Non-connectable Advertising ............................................................................................ 25
   5.2 Connect Application ................................................................................................................. 27
      5.2.1 Connectable Advertising .................................................................................................... 27
      5.2.2 Pairing / Start Encryption ................................................................................................... 28
      5.2.3 Profile Communication ....................................................................................................... 29
   5.3 DTM Application ..................................................................................................................... 31
      5.3.1 Direct Test Mode ................................................................................................................. 31
   5.4 Accessing to Flash memory .................................................................................................... 32
      5.4.1 Accessing to Code Flash memory .................................................................................... 32
      5.4.2 Accessing to Data Flash memory ..................................................................................... 33
   5.5 Supporting Status of Protocol Stack Functions ....................................................................... 34
   5.6 Hardware Resources used .................................................................................................... 35
   5.7 Compiler ..................................................................................................................................... 36
   5.8 Memory Model ........................................................................................................................ 36
   5.9 Program Size .......................................................................................................................... 36
5.10 Address Map ...................................................................................................................... 37

6. Configuration ............................................................................................................................... 40
6.1 Hardware configuration .................................................................................................................. 40
  6.1.1 MCU main system clock frequency ........................................................................................ 41
  6.1.2 RF Operation ......................................................................................................................... 42
  6.1.3 RF on-chip DC-DC converter ................................................................................................ 42
  6.1.4 RF slow clock source ............................................................................................................... 42
  6.1.5 RF on-chip oscillator calibration .............................................................................................. 42
  6.1.6 RF base clock oscillation stabilization time ........................................................................... 43
  6.1.7 Maximum number of Simultaneous connection ...................................................................... 43
  6.1.8 HCI Monitoring ...................................................................................................................... 43
  6.1.9 System Configuration Address ............................................................................................. 44
  6.1.10 Switches on RL78/G1D Evaluation Board .......................................................................... 44
6.2 Application Configuration ............................................................................................................. 46
  6.2.1 System Configuration ............................................................................................................ 46
  6.2.2 Kernel Heap Memory Configuration ..................................................................................... 47
  6.2.3 Advertising Configuration ..................................................................................................... 48
  6.2.4 No Connection Timeout Time Configuration ........................................................................ 51
  6.2.5 Paring Configuration ............................................................................................................. 52
  6.2.6 Custom Profile ....................................................................................................................... 53
  6.2.7 RF Operation ......................................................................................................................... 58

7. Functions ....................................................................................................................................... 59
7.1 Function List ............................................................................................................................... 59
  7.1.1 Switching Application ........................................................................................................... 59
  7.1.2 Beacon Application .............................................................................................................. 59
  7.1.3 Connect Application ............................................................................................................ 59
  7.1.4 DTM Application .................................................................................................................. 59
7.2 Function Calling ........................................................................................................................... 60
  7.2.1 Function Calling of Beacon Operation ............................................................................... 60
  7.2.2 Function Calling of RF Evaluation Operation ....................................................................... 61

8. Operation ....................................................................................................................................... 62
8.1 State Transition ............................................................................................................................ 62
  8.1.1 Beacon Application .............................................................................................................. 62
  8.1.2 Connect Application ............................................................................................................ 63
  8.1.3 DTM Application .................................................................................................................. 64
8.2 Sequence ..................................................................................................................................... 65
  8.2.1 Beacon Application .............................................................................................................. 65
  8.2.2 Connect Application ............................................................................................................ 68
  8.2.3 DTM Application .................................................................................................................. 81
9. Appendix.......................................................................................................................... 83
  9.1 Device Address ............................................................................................................ 83
  9.2 Advertising Packet Format ....................................................................................... 84
  9.3 Attribute Packet Format .......................................................................................... 85
  9.4 Specification Changes ............................................................................................... 87
1. Overview

The Sample Program executes either Beacon Operation or RF Evaluation Operation. In the Beacon Operation, it is possible to transmit Advertising packets and update Advertising data with Custom Profile by using smart phone or other Bluetooth Low Energy device. In the RF Evaluation Operation, it is possible to evaluate RL78/G1D device RF characteristic by using RF Tester.

Figure 1-1 shows the architecture of the Sample Program. The Sample Program consists of Beacon Application, Connect Application, Direct Test Mode (DTM) Application, Beacon Stack, Bluetooth Low Energy (BLE) Protocol Stack, Code Flash Library, and Data Flash Library. The sample program works on RL78/G1D Evaluation Board.

Beacon Application executes transmitting Advertising packets by using Beacon Stack.

Connect Application executes connecting to peer device by using BLE Protocol Stack and writing data by using Code Flash Library and Data Flash Library. Beacon configuration is written to Code Flash memory and pairing information is written to Data Flash memory, so that the configuration and information data are stored after power off.

DTM Application executes Direct Test Mode for evaluating RF characteristic by using BLE Protocol Stack.

Beacon Stack provides the APIs for application to execute Advertising Function.

BLE Protocol Stack provides the APIs for applications to execute Bluetooth Low Energy Functions.

Regarding to the specification of Beacon Application, Connect Application, and DTM Application, refer to chapter 5 “Specification” in this document.

Regarding to the specification of Beacon Stack, refer to RL78/G1D Beacon Stack User's Manual (R01UW0171).


The Sample Program runs on RL78/G1D Evaluation Board. Regarding to the details about the evaluation board, refer to RL78/G1D Evaluation Board User's Manual (R30UZ0048).
1.1 Beacon Operation

Figure 1-2 shows the overview of Beacon Operation.

When set DIP switch SW6 position-1 to ON and then power up the evaluation board, Beacon Application starts running. Beacon Application uses Beacon Stack and it transmits Non-connectable undirected Advertising packet. When switch SW2 is pushed, Beacon Application stops and Connect Application starts running. Connect Application uses BLE Protocol Stack and it transmits Connectable Undirected Advertising packet for establishing connection with peer device. When switch SW2 is pushed again or connection is not established within 30 seconds, Connect Application stops and Beacon Application starts running again.

1.2 RF Evaluation Operation

Figure 1-3 shows the overview of RF Evaluation Operation.

When set DIP switch SW6 position-1 to ON and then power up the evaluation board, the Sample Program starts DTM Application. DTM Application executes Direct Test Mode corresponds to RF Test Commands, which are transferred by RF Tester through 2-wire UART. The application sends the Direct Test Mode results as RF Test Events to the RF Tester.
2. Environment

For compiling and evaluating the Sample Program, following are the necessary environment.

- Hardware Environment
  - Host
    - 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
  - RL78/G1D Evaluation Board (RTK0EN0001D01001BZ)
  - USB cable (A type male / mini-B type male)
  - iOS device or Android device

- Tool
  - Renesas On-chip Debugging Emulator E1 (R0E000010KCE00)

- Software Environment
  - Windows®7 Service Pack1
  - Renesas CS+ for CC V5.00.00 / Renesas CC-RL V1.04.00
    or Renesas e² studio Version 5.3.0.023 / Renesas CC-RL V1.04.00
  - Renesas Flash Programmer v3.02.01
  - Tera Term Pro (or Terminal software which can connect to serial port)
  - UART-USB conversion device driver

Note: It may be that device driver of UART-USB conversion IC "FT232RL" is requested is in the first connection with host. In this case, you can get the device driver from below website.

  - FTDI (Future Technology Devices International) - Drivers
    http://www.ftdichip.com/Drivers/D2XX.htm

- Software Library
  - Beacon Stack : RL78/G1D Beacon Stack V2.10
  - Code Flash Library : Flash Self Programming Library Type01 Ver2.21
  - Data Flash Library : EEPROM Emulation Library Pack02 for CC-RL Compiler Ver1.01

It is possible to download above software libraries from Renesas Website. Regarding to the details about downloading the libraries, refer to section 4.1 "Getting Libraries" in this document.
3. File Composition

The Sample Program includes Beacon Stack library and the source code of Beacon Application, Connect Application and DTM Application. However, the following libraries are not included. So, it is necessary to download the libraries and put in suitable folders for building firmware.

- BLE Protocol Stack library
- Code Flash Library
- Data Flash Library

File and folder composition of the release package of the Sample Program is as shown below.

```
RL78G1D_BeaconCombination
├── ROM_File
│   ├── R5F11AGJ_BcnCmb.hex  Sample Program - firmware file (R5F11AGJ)
│   └── R5F11AGJ_BcnCmb_no_sw.hex  Sample Program - firmware file (R5F11AGJ) (EVABOARD SWITCH_EN=0)
├── RUC_File
│   ├── r5f11agg_syscfg.ruc  System Configuration - unique code file (R5F11AGG)
│   ├── r5f11agh_syscfg.ruc  System Configuration - unique code file (R5F11AGH)
│   └── r5f11agj_syscfg.ruc  System Configuration - unique code file (R5F11AGJ)
└── Project_Source
    │   ├── library
    │   │      ├── r_arch.h  architecture - header file
    │   │      ├── r_compiler.h  compiler dependent part - header file
    │   │      ├── r_iodefine.h  SFR definition - header file
    │   │      └── r_port.h  port access - header file
    │   │   └── beacon
    │   │       ├── BLE_BEACON_CC.lib  Beacon Stack - library for CC-RL
    │   │       └── r_bcn_api.h  Beacon Stack API - header file
    │   ├── protocol
    │   │      ├── (empty)  (it is necessary to put BLE Protocol Stack Library files here)
    │   │      └── dummy
    │   │          └── types.h  dummy header file
    │   │   └── codeflash
    │   │       └── (empty)  (it is necessary to put Code Flash Library files here)
    │   └── dataflash
    │       └── (empty)  (it is necessary to put Data Flash Library files here)
    └── application
        ├── src
        │   ├── cstart.asm  start-up - assembly file for CC-RL
        │   │ └── r_config.h  configuration - header file
        │   └── r_main.c  entry point - header file
        └── beacon
            ├── r_beacon_main.c  Beacon Application main loop - code file
            ├── r_beacon_isr.c  Beacon Application interrupt - code file
            └── r_beacon.h  Beacon Application - header file
        └── connect
            └── r_connect_main.c  Connect Application main loop - code file
        └── r_connect.c  Connect Application - code file
        └── r_profile.h  Custom Profile - header file
        └── r_profile.c  Custom Profile - code file
        └── r_dtm_main.c  DTM Application main loop - code file
        └── r_dtm.h  DTM Application - header file
        └── r_dtm.c  DTM Application - code file
        └── resource
            └── r_rble_core.h  BLE Protocol Stack rBLE Core Layer resource - header file
            └── r_rble_core.c  BLE Protocol Stack rBLE Core Layer resource - code file
            └── r_gatt.h  BLE Protocol Stack GATT Layer resource - header file
```
r_gatt.c  BLE Protocol Stack GATT Layer resource - code file
r_host.h  BLE Protocol Stack Host Layer resource - header file
r_host.c  BLE Protocol Stack Host Layer resource - code file
r_controller.c  BLE Protocol Stack Controller Layer resource - code file
r_kernel.h  BLE Protocol Stack Kernel resource - header file
r_kernel.c  BLE Protocol Stack Kernel resource - code file
r_stack.h  BLE Protocol Stack - header file
optional
r_optional.c  BLE Protocol Stack optional functions - code file
r_reserved.c  BLE Protocol Stack reserved functions - code file
driver
codeflash
  r_codeflash.h  code flash driver - header file
  r_codeflash.c  code flash driver - code file
dataflash
  r_dataflash.h  data flash driver - header file
  r_dataflash.c  data flash driver - code file
  r_eel_descriptor_t02.h  data flash library EEPROM Emulation descriptor - header file
  r_eel_descriptor_t02.c  data flash library EEPROM Emulation descriptor - code file
  r_fdl_descriptor_t02.h  data flash library descriptor - header file
  r_fdl_descriptor_t02.c  data flash library descriptor - code file
input
  r_input.h  external interrupt input driver - header file
  r_input.c  external interrupt input driver - code file
plf
  r_plf.h  platform driver - header file
  r_plf.c  platform driver - code file
uart
  r_uart.h  UART driver - header file
  r_uart.c  UART driver - code file
project
cs_cc
  BLE_Software.mtpj  project file for CS+ for CC
  R5F11AGG_BcnCmb
    R5F11AGG_BcnCmb.mtsp  subproject file for CS+ for CC (R5F11AGG)
  R5F11AGH_BcnCmb
    R5F11AGH_BcnCmb.mtsp  subproject file for CS+ for CC (R5F11AGH)
  R5F11AGJ_BcnCmb
    R5F11AGJ_BcnCmb.mtsp  subproject file for CS+ for CC (R5F11AGJ)
e2_cc
  BLE_Software
    R5F11AGG_BcnCmb
      .project  project composition file for e² studio (R5F11AGG)
      .project  project configuration file for e² studio (R5F11AGG)
      .info  IDE information file for e² studio (R5F11AGG)
      .DefaultBuildlinker  linker configuration file for e² studio (R5F11AGG)
    R5F11AGH_BcnCmb
      .project  project composition file for e² studio (R5F11AGH)
      .project  project configuration file for e² studio (R5F11AGH)
      .info  IDE information file for e² studio (R5F11AGH)
      .DefaultBuildlinker  linker configuration file for e² studio (R5F11AGH)
    R5F11AGJ_BcnCmb
      .project  project composition file for e² studio (R5F11AGJ)
      .project  project configuration file for e² studio (R5F11AGJ)
      .info  IDE information file for e² studio (R5F11AGJ)
      .DefaultBuildlinker  linker configuration file for e² studio (R5F11AGJ)
4. Evaluation Procedure

This chapter describes evaluation procedure of the Sample Program. The evaluation procedure consists of six steps: Getting Libraries, Building Firmware, Writing Firmware, Evaluating with smart phone, Evaluating RF characteristic, and Current Consumption Measurement.

- **Section 4.1 "Getting Libraries"**
- **Section 4.2 "Building Firmware"**
  - Subsection 4.2.1 "Using CS+ for CC"
  - or
  - Subsection 4.2.2 "Using Renesas e2 studio"
- **Section 4.3 "Writing Firmware"**
- **Section 4.4 "Evaluating with smart phone"**
  - Subsection 4.4.1 "Confirming the transmission of advertising packet"
  - Subsection 4.4.2 "Updating the advertising packet"
    - Subsection 4.4.2(1) "Using iOS device"
    - or
    - Subsection 4.4.2(2) "Using Android device"
  - Subsection 4.4.3 "Confirming the updated advertising packet"
- **Section 4.5 "Evaluating RF characteristic"**
- **Section 4.6 "Current Consumption Measurement"**
  - Subsection 4.6.1 "Measurement Environment"
  - Subsection 4.6.2 "Evaluation Board Setting"
  - Subsection 4.6.3 "Measurement Procedure"
4.1 Getting Libraries

Before building the Sample Program firmware, it is necessary to download below libraries from Renesas Website.

- **BLE Protocol Stack:**
  - Bluetooth Low Energy Protocol Stack V1.20
    https://www.renesas.com/software-tool/bluetooth-low-energy-protocol-stack-rl78-family

- **Code Flash Library:**
  - Flash Self Programming Library Type01 Package Ver.3.00 for the RL78 Family [for the CA78K0R/CC-RL Compiler]

- **Data Flash 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

After downloading the libraries, copy respective released library files into specified folders of the Sample Program. Respective downloaded library paths are as shown below.

- **Protocol Stack:**
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\renesas\lib\BLE_rBLE_lib_CCRL.lib
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\renesas\lib\BLE_HOST_lib_CCRL.lib
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\renesas\lib\BLE_CONTROLLER_LIB_CCRL.lib
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\rBLE\src\include\rble_api.h
  - BLE_Software_Ver_x_xx\RL78_G1D\Project_Source\rBLE\src\include\rble.h

- **Code Flash Library:**
  - FSLRL78 Type01\V2.21B\CCRL_V2.21\CCRL\V2.21\librl78\fsl.lib
  - FSLRL78 Type01\V2.21B\CCRL_V2.21\CCRL\V2.21\incrl78\fsl.h
  - FSLRL78 Type01\V2.21B\CCRL_V2.21\CCRL\V2.21\incrl78\fsl_types.h

- **Data Flash Library:**
  - EELRL78 Pack02\V.1.01\librl78\eel.lib
  - EELRL78 Pack02\V.1.01\librl78\fdl.lib
  - EELRL78 Pack02\V.1.01\incrl78\eel.h
  - EELRL78 Pack02\V.1.01\incrl78\eel_types.h
  - EELRL78 Pack02\V.1.01\incrl78\fdl.h
  - EELRL78 Pack02\V.1.01\incrl78\fdl_types.h

The above files that needed to copy into folders of the Sample Program are as shown below.

RL78G1D_BeaconCombination
  └ Project_Source
      └ library
          └ protocol
              BLE_rBLE_lib_CCRL.lib: Protocol Stack rBLE Layer - library file
              BLE_HOST_lib_CCRL.lib: Protocol Stack Host Layer - library file
              BLE_CONTROLLER_LIB_CCRL.lib: Protocol Stack Controller Layer - library file
              rble_api.h: Protocol Stack rBLE API - header file
              rble.h: Protocol Stack rBLE definitions - header file

          └ codeflash
              fsl.lib: Code Flash Library - library file
              fsl.h: Code Flash Library - header file
              fsl_types.h: Code Flash Library type definition - header file

          └ dataflash
              eel.lib: Data Flash Library EEPROM Emulation - library file
              eel.h: Data Flash Library EEPROM Emulation - header file
              eel_types.h: Data Flash Library EEPROM Emulation type definition - header file
              fdl.lib: Data Flash Library - library file
              fdl.h: Data Flash Library - header file
              fdl_types.h: Data Flash Library type definition - header file
4.2 Building Firmware

After adding the necessary libraries from aforementioned section 4.1, the project for the Sample Program is ready to build firmware. Building Sample Program firmware can be used either CS+ for CC or e² studio as IDE (Integrated Development Environment).

By building the Sample Program with default settings, the firmware R5F11AGJ_BcnCmB.hex file that is the same HEX file included in release package is generated.

If using HEX file included in release package for evaluation, you can skip below building procedures.

4.2.1 Using CS+ for CC

1. Start CS+ for CC and open the project "BLE_Software.mtpj" from menu bar [File]→[Open File]
   - Project_Source\application\project\cs_cc\BLE_Software\n
2. Select [Build]→[Rebuild project] and confirm that compiling is successful.

3. Confirm that the firmware R5F11AGJ_BcnCmB.hex is generated in the place of below path.
   - Project_Source\application\project\cs_cc\BLE_Software\R5G11AGJ_BcnCmB\DefaultBuild\n
4.2.2 Using Renesas e² studio

1. Start Renesas e² studio and select below path as a workspace.
   - Project_Source\n
2. Select [File]→[Import] in order to open Import dialog.

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

4. Select below path as a root folder and confirm R5F11AGJ_BcnCmB is selected in [Projects].
   - Project_Source\n
5. Click [Finish] in order to close Import dialog.

6. Close [Welcome].

7. Select R5F11AGJ_BcnCmB in the Project Explorer.

8. Select [Project]→[Build Project] and confirm that compiling is success.

9. Confirm that the firmware R5F11AGJ_BcnCmB.hex is generated in the place of below path.
   - Project_Source\application\project\e2_cc\BLE_Software\R5F11AGJ_BcnCmB\DefaultBuild\n
Note: By default, debugger setting of e² studio erases Flash memory before writing firmware.

When developing by using e² studio, change the debugger setting before starting debugging, to avoid erasing Shipping Checking Flag and Device Address which already been written in RL78/G1D Module. Disconnect E1 Emulator from RL78/G1D Module when changing the debugger setting.

   - Select [Debugger] tab in [Edit launch configuration properties] dialog, and set [No] in [Erase Flash ROM When Starting].
4.3 Writing Firmware

When writing the firmware of the Sample Program, Host machine and E1 Emulator is used. It is connected by USB cable between Host machine and E1 Emulator, between Host machine and the evaluation board. It is connected by User I/F cable between E1 Emulator and the evaluation board.

Regarding to the details of E1 Emulator, refer to E1 Emulator User's Manual (R20UT0398) and E1 Emulator Additional Document for User’s Manual (Notes on Connection of RL78) (R20UT1994).

![Figure 4-1 Evaluation Board Operation](image)

**Table 4-1** shows the slide switch settings for evaluating the Sample Program.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW7</td>
<td>2-3 connected (right) (default setting)</td>
<td>Power is supplied from the AC Power Supply Adapter through Power jack (J1) or USB interface (CN3) via a Regulator.</td>
</tr>
<tr>
<td>SW8</td>
<td>2-3 connected (right)</td>
<td>Power is supplied from USB. If it is necessary to supply from AC Power Supply Adapter, set 1-2 connected (left).</td>
</tr>
<tr>
<td>SW9</td>
<td>1-2 connected (left)</td>
<td>Connect to external extension interface.</td>
</tr>
<tr>
<td>SW10</td>
<td>1-2 connected (left) (default setting)</td>
<td>Power is supplied to the module.</td>
</tr>
<tr>
<td>SW11</td>
<td>2-3 connected (right) (default setting)</td>
<td>Power is supplied from a source other than the E1 Emulator</td>
</tr>
<tr>
<td>SW12</td>
<td>2-3 connected (right) (default setting) (Fixed)</td>
<td></td>
</tr>
<tr>
<td>SW13</td>
<td>2-3 connected (right)</td>
<td>USB interface is disconnected.</td>
</tr>
</tbody>
</table>
For writing Sample Program firmware, Renesas Flash Programmer (RFP) is used.

There is a Unique Code Embedding Function in RFP, so that it is possible to write same firmware with each different system configuration to each individual device. Regarding to the Unique Code Embedding Function, refer to subsection 2.3.6 "[Unique Code] Tabbed Page" in Renesas Flash Programmer V3.02 Flash memory programming software User's Manual (R20UT3841).

Procedure of writing the Sample Program firmware into RL78/G1D Evaluation Board is shown below.

1. Set the switches of Evaluation Board according to Table 4-1.
2. Connect E1 Emulator to Evaluation Board, and connect E1 Emulator to PC.
3. Connect Evaluation Board to PC or AC-USB Power Supply Adaptor, and start to supply power.
4. Start RFP, and create project in accordance with below procedures.
   - Note that after creating project once, below procedures can be skipped for next time.
     4-1. Select [File] ➔ [Create a new project].
     4-2. Select [RL78] as a Microcontroller, and input a project name, and click [Connect] in [Create New Project] dialog.
5. Select the firmware R5F11AGJ_BcnCmb.hex at [Program File].
6. Prevent erasing Block 254, 255 in Code Flash memory in accordance with below procedure.
   - Note that Shipping Check Flag is written in Block 254 and Device Address is written in Block 255 in the case of RL78/G1D Module.
     6-2. Select [Block Setting] tab, and uncheck each [Erase], [P.V] of Block 254, 255.
7. (Optional) If it is necessary to change System Configuration, set unique code in accordance with below procedure.
   - 7-1. Select [Unique Code] tab.
   - 7-2. Check [Enable].
   - 7-3. Select the below unique code file at [Unique Code File].
     - RUC_File\r5f11agj_syscfg.ruc
   - 7-4. Go back to [Operation] tab.
8. Click [Start] button to start writing the firmware, and confirm [Operation completed] message.
4.4 Evaluating with smart phone

To evaluate the Sample Program with smart phone, there are three steps: Confirming the transmission of advertising packet, Updating the advertising packet, and Confirming the updated advertising packet.

The Sample Program uses some switches (SW) and LED indicators on the evaluation board for user interface. Figure 4-2 shows the switches SW and the LED indicators used by Sample Program. The status of DIP switch SW6 position-1 is read when initialize the Sample Program, and SW2 is used external interrupt input for triggering user action Beacon Operation.

![Evaluation Board Operation Diagram](image)

**Figure 4-2 Evaluation Board Operation**
4.4.1 Confirming the transmission of advertising packet

This subsection describes procedure for enabling Beacon Application and confirming transmitted Advertising packet from the evaluation board to Smart Phone.

First, supply +5 DC power via Power jack (J1) or USB interface (CN3) to the evaluation board, which is programmed with the Sample Program firmware. Just after power on with setting OFF to DIP switch SW6 position-1, the Sample Program executes Beacon Application and turn on the LED4. The Beacon Application transmits Eddystone-URL packet by default. Using smart phone application, you can receive the advertising packet. The smart phone application procedure is very much similar for both iOS device and Android device.

1. In order to receive Eddystone-URL packet, install below application to smart phone.
   - for Android device, Physical Web - Google Play
   - for iOS device, Physical Web - App Store
     https://itunes.apple.com/app/physical-web/id927653608

2. Run the smart phone application and search the Eddystone beacons by pulling down the display.

3. When receive the Eddystone-URL packet from the Sample Program, below URL is displayed to link the web page.
   - Renesas Electronics
     https://www.renesas.com/
4.4.2 Updating the advertising packet

This subsection describes procedure for enabling Connect Application and updating Advertising packet by using Smart Phone.

Pushing SW2 on the evaluation board can change Beacon Application and Connect Application alternately when the Sample Program is running in Beacon Operation. When Connect Application works, not only LED4 but also LED3 are turned on. Connect Application transmits connectable advertising packet in order to connect peer device. By connecting Smart Phone to RL78/G1D device and accessing GATT, it is possible to update Advertising data Beacon Application transmits.

1. **Using iOS device**
   1. iOS device needs GATT Client application to access Custom Profile of Connect Application. As an example of GATT Client application, install and use below application.
      - GATTBrowser - App Store
        [https://itunes.apple.com/app/gattbrowser/id1163057977](https://itunes.apple.com/app/gattbrowser/id1163057977)
   2. Push SW2 on the evaluation board and confirm LED3 is on.
   3. Run the smart phone application and start scanning devices.
   4. Select the "RL78/G1D Beacon Updater" to establish connection.
   5. Select the Advertising Data characteristic of the Renesas Beacon Updater service.
   6. Tap "Read" button.
   7. Tap the value displayed below "Read" button.
   8. In Advertising Data dialog, key in below value as new Advertising data.
      - Advertising value, Eddystone-URL including shortened URL to [https://www.bluetooth.com/](https://www.bluetooth.com/) 1B0201060303AAFE1316AAFE10EE02676F6F2E676C2F3764694C547800000000
   9. Tap "OK" button in Advertising Data dialog.
   10. Tap "Write" button.
   11. Tap "Pair" button in pairing request dialog.
   12. Confirm LED1 on the evaluation board is on.
   13. Return to scanning view to disconnect.
   14. Confirm LED1 and LED2 on the evaluation board are off.
4. Tap the device to pair

5. Tap the Characteristic

6. Tap Read

7. Tap the value

8. Input new value

9. Tap OK

10. Tap Write

11. Select Pair

12. Tap to return

13. Tap to return

14. Tap to disconnect
(2) Using Android device

1. Android device needs GATT Client application to access Custom Profile of Connect Application. As an example of GATT Client application, install and use below application.
   - GATTBrowser - Google Play

2. Push SW2 on the evaluation board and confirm LED3 is on.

3. Run the smart phone application and start scanning devices.

4. Select the "RL78/G1D Beacon Updater" to establish connection.

5. Select the Advertising Data characteristic of the Renesas Beacon Updater service.

6. Tap "Read" button.

7. Tap the value displayed below "Read" button.

8. In Advertising Data dialog, key in below value as new Advertising data.
   - Advertising Data value, Eddystone-URL including shortened URL to https://www.bluetooth.com/
     0x1B0201060303AAFE1316AAFE10EE02676F6F2E676C2F3764694C5478

9. Tap "OK" button in Advertising Data dialog.

10. Tap "Write" button.

11. Confirm BONDED by pairing.

12. Confirm LED1 is on.

13. Tap "Write" button again.


15. Tap the DISCONNECT

16. Confirm LED1 and LED2 on the evaluation board is off.
4. Select the device

6. Tap Read

8. Input new value

10. Tap Write

13. Tap Write
4.4.3 Confirming the updated advertising packet

This subsection describes procedure for enabling Beacon Application again and confirming that Advertising packet is updated.

- Advertising data for Beacon Application is stored into Code Flash memory after updating the characteristic value of Custom Profile and disconnection. Again, you can confirm the updated advertising data with using Beacon Application and smart phone. Here, the procedure for both iOS device and Android device are very similar.

1. Restart Beacon Application, by pushing SW2 on the evaluation board, or reset the MCU, or power-cycling to evaluation board.

2. Run the smart phone application installed in subsection 4.4.1 and search the advertising packet by pulling down the display.

3. When receive new Eddystone-URL packet from the Sample Program, link to below new URL is displayed to link the web page.
   - Bluetooth Technology Website
     https://www.bluetooth.com/
4.5 Evaluating RF characteristic

This subsection describes procedure for enabling DTM Application and evaluating RF characteristic of RL78/G1D device.

Before supplying power to the evaluation board, turn on the DIP switch SW6 position-1. Then by supplying the power to the board, the Sample Program executes the DTM Application. Now, you can test RF characteristic of RL78/G1D device by using RF Tester. Below is the procedure for testing RF characteristic of RL78/G1D device.

1. Turn on the DIP switch SW6 position-1, which is on the evaluation board.
2. Connect UART TxD0 pin, RxD0 pin, and GND pin on the evaluation board to pins of RF Tester. If logic level of the signals is different between RL78/G1D device and RF Tester, connect through logic level converter.
3. Refer to respective manuals of RF Tester and set UART settings according to Table 4-2.
4. Refer to respective manuals of RF Tester and start Direct Test Mode.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baud rate</td>
<td>9600bps</td>
</tr>
<tr>
<td>Data bit length</td>
<td>8bit</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
</tr>
<tr>
<td>Stop bit length</td>
<td>1bit</td>
</tr>
<tr>
<td>Flow control</td>
<td>None</td>
</tr>
</tbody>
</table>
4.6 Current Consumption Measurement

This section describes current consumption measurement for using RL78/G1D Evaluation Board. Regarding to the details of RL78/G1D Evaluation Board (RTK0EN0001D01001BZ), refer to RL78/G1D Evaluation Board User's Manual (R30UZ0048).

4.6.1 Measurement Environment

Table 4-3 shows the necessary equipment for current consumption measurement. Regarding to the details on how to use each equipment, refer to respective manuals of each equipment.

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Role</th>
<th>Example Equipment</th>
</tr>
</thead>
</table>
| Power Source       | Supplying power to RL78/G1D               | Stabilized power supply or Battery
|                    | Note that supply voltage shall be in the range of the RL78/G1D operation voltage |
| Measurement        | Indicating and logging the result of measurement | Oscilloscope                                                                     |
| Measurement        | Voltage Detector                          | Oscilloscope                                                                      |
| Detector           | Detecting the operation voltage of RL78/G1D | Voltage Probe                                                                     |
| Detector           | Detecting the current consumption of RL78/G1D | Current Probe with clamp, or combination of Shunt Resistor and Voltage Probe
|                    | Note that recommended resistor is 10 ohm. |                                     |

Figure 4-3 shows the measurement environment which uses current probe as current detector. In this environment, the current consumption of RL78/G1D is the result of measuring between terminal TP7 and TP8 of the evaluation board by current probe.

![Figure 4-3 Measurement Environment which uses Current Probe](image)

Figure 4-4 shows the measurement environment which uses the combination of shunt resistor and voltage probe as a current detector. In this environment, the resistor is inserted between terminal TP7 and TP8 of the evaluation board, and voltage drop at the resistor is measured by using two voltage probes.

Voltage drop $dV$ by the resistor is difference of two voltages measured by individual voltage probe. The current consumption of RL78/G1D is the result of calculating with formula $I = dV / R$, where $I$ is current; $dV$ is voltage drop by the resistor; and $R$ is resistance value.

![Figure 4-4 Measurement Environment which uses the combination of Shunt Resistor and Voltage Probe](image)
4.6.2 Evaluation Board Setting

Table 4-4 shows the slide switch settings of the evaluation board for current consumption measurement.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW7</td>
<td>1-2 connected (left)</td>
<td>Power is directly supplied from external power source (not via a regulator). If it is necessary to supply from USB, set 2-3 connected (right).</td>
</tr>
<tr>
<td>SW8</td>
<td>1-2 connected (left)</td>
<td>Power is supplied from TP1, TP5 pin or AC Power Supply Adapter. If it is necessary to supply from USB, set 2-3 connected (right).</td>
</tr>
<tr>
<td>SW9</td>
<td>1-2 connected (left)</td>
<td>Connect to an external extension interface.</td>
</tr>
<tr>
<td>SW10</td>
<td>2-3 connected (right)</td>
<td>The power supply line is left open.</td>
</tr>
<tr>
<td>SW11</td>
<td>2-3 connected (right) (default setting)</td>
<td>Power is supplied from a source other than the E1 debugger.</td>
</tr>
<tr>
<td>SW12</td>
<td>2-3 connected (right) (default setting)</td>
<td>(Fixed).</td>
</tr>
<tr>
<td>SW13</td>
<td>2-3 connected (right)</td>
<td>USB interface is disconnected.</td>
</tr>
</tbody>
</table>

4.6.3 Measurement Procedure

Current consumption measurement procedures are described in below steps. Note that the procedure is reference for only measuring current consumption of Beacon Application with default setting.

Regarding to the details of how to set each equipment settings, refer to the respective manuals.

(1) Measuring Current Consumption in Periodic Packet Transmission

1. Start supplying power and start Beacon Application.
2. Set below settings to Oscilloscope by referring to Figure 4-5.
   - Capture Trigger : about 0.5mA in current consumption
   - Current Measurement Range : about 10mA
   - Measurement Period : about 10msec from capture trigger
3. Start measuring by Oscilloscope by detecting the current of periodic transmitting.

![Figure 4-5 Measuring Current Consumption in Periodic Packet Transmission]
5. Specification

5.1 Beacon Application

5.1.1 Non-connectable Advertising

Beacon Application loads the Advertising Information and the Advertising Data from system configuration, which stored in Code Flash memory. Then starts transmitting Non-connectable Undirected Advertising packet for broadcasting information. Peer device, like a Smart Phone, receives Advertising packet and provides each service related to the Advertising data. When request to exit the application, it stops Advertising and stops supplying power to RF unit.

![Image: Non-connectable Advertising](image)

**Figure 5-1 Non-connectable Advertising**

Regarding to the state transition of Beacon Application and the sequence of Non-connectable Advertising, refer to subsection 8.1.1 “Beacon Application” and subsection 8.2.1(1) “Initializing & Advertising & RF Powerdown Sequence” in this document respectively.

Regarding to the details about system configuration, refer to subsection 5.4.1 "Accessing to Code Flash memory” in this document.

When only Tx is enabled as a RF Operation of Beacon Stack, Beacon Application transmits Non-connectable Undirected Advertising packet.

Table 5-1 shows the default Advertising configuration of Beacon Application.

**Table 5-1 the Advertising configuration of Beacon Application when only Tx is enabled**

<table>
<thead>
<tr>
<th>Advertiser Address</th>
<th>Public Device Address 12:34:56:78:9A:B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advertising Type</td>
<td>Non-connectable Undirected Advertising (ADV_NONCONN_IND)</td>
</tr>
<tr>
<td>Advertising Interval</td>
<td>100msec</td>
</tr>
<tr>
<td>Advertising Interval Delay</td>
<td>add random delay to Advertising interval</td>
</tr>
<tr>
<td>Advertising Channel Map</td>
<td>All channels (37,38,39ch)</td>
</tr>
<tr>
<td>Advertising Loop Count</td>
<td>transmitting indefinitely</td>
</tr>
<tr>
<td>Advertising Transmit Power</td>
<td>0dBm at ANT pin of RL78/G1D</td>
</tr>
<tr>
<td>Advertising Data Count</td>
<td>the number of Advertising Data is 1</td>
</tr>
<tr>
<td>Advertising Data [0]~[9]</td>
<td>Advertising Data[0] (ADV_NONCONN_IND payload data)</td>
</tr>
<tr>
<td>Length</td>
<td>2byte</td>
</tr>
<tr>
<td>AD Type</td>
<td>&lt;&lt;Flags&gt;&gt; (0x01)</td>
</tr>
<tr>
<td>AD Data</td>
<td>LE General Discoverable Mode (bit1)</td>
</tr>
<tr>
<td></td>
<td>BR/EDR Not Supported (bit2)</td>
</tr>
<tr>
<td>Length</td>
<td>3byte</td>
</tr>
<tr>
<td>AD Type</td>
<td>&lt;&lt;Complete List of 16-bit Service Class UUIDs&gt;&gt; (0x03)</td>
</tr>
<tr>
<td>AD Data</td>
<td>Eddystone (0xFEAA)</td>
</tr>
<tr>
<td>Length</td>
<td>19byte</td>
</tr>
<tr>
<td>AD Type</td>
<td>&lt;&lt;Service Data&gt;&gt; (0x16)</td>
</tr>
<tr>
<td>AD Data</td>
<td>Eddystone-URL: <a href="https://goo.gl/5wKkRK">https://goo.gl/5wKkRK</a></td>
</tr>
<tr>
<td>Advertising Data[1] to [9] are empty</td>
<td></td>
</tr>
<tr>
<td>Advertising Event Permission</td>
<td>notify All Advertising Event</td>
</tr>
<tr>
<td>Use White List</td>
<td>-</td>
</tr>
</tbody>
</table>
When both Tx and Rx are enabled as a RF Operation of Beacon Stack, Beacon Application transmits Scannable Undirected Advertising packet. And If receive Scan Request packet, Beacon Application transmits Scan Response packet.

Regarding to the setting of RF Operation, refer to subsection 6.1.2 "RF Operation".

Table 5-2 shows Advertising configuration of Beacon Application when both Tx and Rx are enabled.

<table>
<thead>
<tr>
<th>Advertiser Address</th>
<th>Public Device Address 12:34:56:78:9A:B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advertising Type</td>
<td>Scannable Undirected Advertising (ADV_SCAN_IND)</td>
</tr>
<tr>
<td>Advertising Interval</td>
<td>100msec</td>
</tr>
<tr>
<td>Advertising Interval Delay</td>
<td>add random delay to Advertising interval</td>
</tr>
<tr>
<td>Advertising Channel Map</td>
<td>All channels (37,38,39ch)</td>
</tr>
<tr>
<td>Advertising Loop Count</td>
<td>transmitting indefinitely</td>
</tr>
<tr>
<td>Advertising Transmit Power</td>
<td>0dBm at ANT pin of RL78/G1D</td>
</tr>
<tr>
<td>Advertising Data Count</td>
<td>the number of Advertising Data is 2</td>
</tr>
<tr>
<td>Advertising Data [0]~[9]</td>
<td>Advertising Data[0] (ADV_SCAN_IND payload data)</td>
</tr>
<tr>
<td></td>
<td>Length 2byte</td>
</tr>
<tr>
<td></td>
<td>AD Type &lt;&lt;Flags&gt;&gt; (0x01)</td>
</tr>
<tr>
<td></td>
<td>AD Data LE General Discoverable Mode (bit1) BR/EDR Not Supported (bit2)</td>
</tr>
<tr>
<td></td>
<td>Length 3byte</td>
</tr>
<tr>
<td></td>
<td>AD Type &lt;&lt;Complete List of 16-bit Service Class UUIDs&gt;&gt; (0x03)</td>
</tr>
<tr>
<td></td>
<td>AD Data Eddystone (0xFEAA)</td>
</tr>
<tr>
<td></td>
<td>Length 19byte</td>
</tr>
<tr>
<td></td>
<td>AD Type &lt;&lt;Service Data&gt;&gt; (0x16)</td>
</tr>
<tr>
<td></td>
<td>AD Data Eddystone-URL: <a href="https://goo.gl/5wKkRK">https://goo.gl/5wKkRK</a></td>
</tr>
<tr>
<td>Advertising Data[1] (SCAN_RSP payload data)</td>
<td>Length 24byte</td>
</tr>
<tr>
<td></td>
<td>AD Type &lt;&lt;Complete Local Name&gt;&gt; (0x09)</td>
</tr>
<tr>
<td></td>
<td>AD Data “Renesas RL78/G1D Beacon”</td>
</tr>
<tr>
<td>Advertising Data[2] to [9] are empty</td>
<td></td>
</tr>
<tr>
<td>Advertising Event Permission</td>
<td>notify All Advertising event</td>
</tr>
<tr>
<td>Use White List</td>
<td>not use White List</td>
</tr>
</tbody>
</table>

Regarding to the specification of Eddystone and Eddystone-URL, refer to below website.

- Specification for Eddystone, an open beacon format from Google https://github.com/google/eddystone
5.2 Connect Application

5.2.1 Connectable Advertising

First, Connect Application starts transmitting Connectable undirected advertising packet for establishing connection. Then peer device, like a smart phone, receives Advertising packet and establishes a connection with RL78/G1D device by transmitting Connection Request packet.

![Connectable Advertising](image)

**Figure 5-2 Connectable Advertising**

Regarding to the state transition of Connect Application and the sequence of Connectable Advertising, refer to subsection 8.1.2 "Connect Application" and subsection 8.2.2(1) "Initializing & Advertising & Slave Connection (Configurations) Sequence" in this document respectively.

Table 5-3 shows the default advertising configuration of Connect Application.

<table>
<thead>
<tr>
<th>Table 5-3  the default advertising configuration of Connect Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advertiser Address</strong></td>
</tr>
<tr>
<td><strong>Advertising Type</strong></td>
</tr>
<tr>
<td><strong>Advertising Interval Min</strong></td>
</tr>
<tr>
<td><strong>Advertising Interval Max</strong></td>
</tr>
<tr>
<td><strong>Advertising Channel Map</strong></td>
</tr>
<tr>
<td><strong>Advertising Data</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Scan Response Data</strong></td>
</tr>
</tbody>
</table>
5.2.2 Pairing / Start Encryption

After establishing connection, Connect Application executes paring sequence or starts encryption sequence by Master's request. When the pairing sequence is completed, the subsequent transmitted data packets are encrypted.

The pairing sequence is executed in first connection to the peer device. Device exchanges the pairing information and generates the encryption key. For encrypting data in subsequent connection, need the generated encryption key. Thus, the application stores the encryption key into the Data Flash memory, by using Data Flash Library.

The start encryption sequence is executed in the connection to the peer device, which has been already executed the pairing sequence before connection. The application loads encryption key from Data Flash memory after that start to encrypt data packets.

![Diagram of Pairing Sequence or Start Encryption Sequence]

**Figure 5-3** Pairing / Start Encryption

Regarding to the state transition of Connect Application and the sequence of the pairing / the start encryption, refer to subsection 8.1.2 "Connect Application", subsection 8.2.2(2) "Slave Connection (Pairing) Sequence", and subsection 8.2.2(3) "Slave Connection (Start Encryption) Sequence" in this document respectively.

**Table 5-4** shows the default pairing configuration of Connect Application.

<table>
<thead>
<tr>
<th>Bonding</th>
<th>Bondable Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Mode</td>
<td>Unauthenticated pairing with encryption</td>
</tr>
<tr>
<td>Pairing Method</td>
<td>Just Works</td>
</tr>
<tr>
<td>IO Capabilities</td>
<td>No Input No Output</td>
</tr>
<tr>
<td>OOB Flag</td>
<td>OOB Data not present</td>
</tr>
<tr>
<td>Authentication Requirements</td>
<td>No MITM Bonding</td>
</tr>
<tr>
<td>Encryption Key Size</td>
<td>128bit</td>
</tr>
<tr>
<td>Initiator Key Distribution</td>
<td>None</td>
</tr>
<tr>
<td>Responder Key Distribution</td>
<td>Encryption key</td>
</tr>
</tbody>
</table>
5.2.3 Profile Communication

In the connection, Connect Application communicates data according to the Custom Profile. First, GATT Client device gets the service composition and the characteristic composition of GATT Server device by Primary Service Discovery, Characteristic Discovery and Characteristic Descriptor Discovery.

![Figure 5-4 GATT Discovery](image)

Then, GATT Client device reads and writes the characteristic value of GATT Server device by Characteristic Value Read and Characteristic Value Write respectively.

![Figure 5-5 GATT Read / Write](image)

Regarding to the state transition of Connect Application and the sequence of GATT Access, refer to subsection 8.1.2 "Connect Application" and subsection 8.2.2(4) "Slave Connection (GATT Access) Sequence" in this document respectively.

Regarding to implementing and changing Custom Profile, refer to subsection 6.2.6 "Custom Profile" in this document.

The specification of Custom Profile implemented in the Sample Program is as shown below.

- **Custom Profile Role**
  - Role of Beacon device is GATT Server.
  - Role of device which connects to beacon device is GATT Client.
  - GATT Server has Custom Service.
  - GATT Client gets Characteristic Value of Custom Service by Characteristic Value Read, and updates Characteristic Value of Custom Service by Characteristic Value Write.
  - GATT Server does not inform data by Notification and Indication.

- **Custom Profile Scenarios**
  - GATT Client device updates Advertising Information and Advertising Data for Beacon Stack in beacon device by writing Characteristic Value of Custom Profile.
  - Advertising information and Advertising data is stored by Code Flash memory of beacon device.
  - GATT Client device gets the number of updating Code Flash memory and Data Flash memory in beacon device by reading Characteristic Value of Custom Profile.

*Figure 5-6 shows the default Custom Profile Role of Connect Application.*

![Figure 5-6 Custom Profile Role](image)
Table 5-5 shows the default Custom Service specification of Connect Application.

<table>
<thead>
<tr>
<th>Attribute Handle</th>
<th>Attribute Type</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;Custom Service&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000C</td>
<td>Primary Service Declaration (0x2800)</td>
<td>UUID: A7660001-4B1E-4D6E-91C4-997BA9B6FC07</td>
</tr>
<tr>
<td>&lt;&lt;Characteristic : Advertising Information&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000D</td>
<td>Characteristic Declaration (0x2803)</td>
<td>Properties: Read, Write (0x0A) Value Handle: 0x000E UUID: A7660002-4B1E-4D6E-91C4-997BA9B6FC07</td>
</tr>
<tr>
<td>0x000E</td>
<td>Advertising Information</td>
<td>Advertising Information structure defined by Beacon Stack API (18byte)</td>
</tr>
<tr>
<td>&lt;&lt;Characteristic : Advertising Data&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000F</td>
<td>Characteristic Declaration (0x2803)</td>
<td>Properties: Read, Write (0x0A) Value Handle: 0x0010 UUID: A7660003-4B1E-4D6E-91C4-997BA9B6FC07</td>
</tr>
<tr>
<td>0x0010</td>
<td>Advertising Data</td>
<td>Advertising Data structure defined by Beacon Stack API (32byte)</td>
</tr>
<tr>
<td>&lt;&lt;Characteristic : Scan Response Data&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0011</td>
<td>Characteristic Declaration (0x2803)</td>
<td>Properties: Read, Write (0x0A) Value Handle: 0x0012 UUID: A7660006-4B1E-4D6E-91C4-997BA9B6FC07</td>
</tr>
<tr>
<td>0x0012</td>
<td>Scan Response Data</td>
<td>Advertising Data structure defined by Beacon Stack API (32byte)</td>
</tr>
<tr>
<td>&lt;&lt;Characteristic : Code Flash Memory Updated Count&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0013</td>
<td>Characteristic Declaration (0x2803)</td>
<td>Properties: Read (0x02) Value Handle: 0x0014 UUID: A7660004-4B1E-4D6E-91C4-997BA9B6FC07</td>
</tr>
<tr>
<td>0x0014</td>
<td>Code Flash Memory Updated Count</td>
<td>Code Flash Memory Updated Count (2byte) Byte Order : Least Significant Byte First</td>
</tr>
<tr>
<td>&lt;&lt;Characteristic : Data Flash Memory Updated Count&gt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0015</td>
<td>Characteristic Declaration (0x2803)</td>
<td>Properties: Read (0x02) Value Handle: 0x0016 UUID: A7660005-4B1E-4d6e-91C4-997BA9B6FC07</td>
</tr>
<tr>
<td>0x0016</td>
<td>Data Flash Memory Updated Count</td>
<td>Data Flash Memory Updated Count (2byte) Byte Order : Least Significant Byte First</td>
</tr>
</tbody>
</table>

Regarding to the specification of Advertising Information structure and Advertising Data structure, refer to chapter 4 "API" in RL78/G1D Beacon Stack User's Manual (R01UW0171).
5.3 DTM Application

5.3.1 Direct Test Mode

DTM Application enables UART for communicating for RF Test commands and events. By receiving RF Test command from Tester, the application executes RF Transmitter Test and RF Receiver Test, then after the application transmits RF Test events back to Tester.

![Figure 5-7 Direct Test Mode](image)

Regarding to the state transition of DTM Application and the sequence of Direct Test Mode, refer to subsection 8.1.3 "DTM Application" and subsection 8.2.3(1) "Initializing & Transmitter Test & Receiver Test Sequence" in this document respectively.

Table 5-6 shows RF Test Commands for executing Direct Test mode.

<table>
<thead>
<tr>
<th>RF Test Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE_RESET</td>
<td>Control (ignored)</td>
</tr>
<tr>
<td>LE_RECEIVER_TEST</td>
<td>Frequency, Length, Packet Type</td>
</tr>
<tr>
<td>LE_TRANSMITTER_TEST</td>
<td>Frequency, Length, Packet Type</td>
</tr>
<tr>
<td>LE_TEST_END</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 5-7 shows RF Test Events for returning the result of Direct Test Mode.

<table>
<thead>
<tr>
<th>RF Test Event</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>LE_TEST_STATUS</td>
<td>Status (Success/Error)</td>
</tr>
<tr>
<td>LE_TEST_PACKET_REPORT</td>
<td>Packet Count</td>
</tr>
</tbody>
</table>

Regarding to the specification of Direct Test Mode, refer to [Vol. 6, Part F] Section 3.3, Bluetooth Core Specification v4.2.
5.4 Accessing to Flash memory

5.4.1 Accessing to Code Flash memory

Both Beacon Application and Connect Application uses a part of Code Flash memory, which is located outside of firmware, to store parameters as system configuration. System configuration is used to store parameters, which need to be different from each individual device.

Beacon Application only loads Device Address, Type, Advertising Information and Advertising Data from system configuration. Then start Advertising.

Connect Application loads Device Address, Type and Device Name from system configuration, and configures them to Protocol Stack. When Connect Application receives new Advertising Information or new Advertising Data from peer connected device, the application updates system configuration in Code Flash memory.

Table 5-8 shows the specification of system configuration in Code Flash memory. Regarding to the location of system configuration, refer to section 5.10 "Address Map" in this document.

<table>
<thead>
<tr>
<th>offset</th>
<th>data</th>
<th>size</th>
<th>read (YES:read, NO:not read)</th>
<th>write (YES:write, NO:not write)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Device Address (RBLE_BD_ADDR structure)</td>
<td>6 byte</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>0x06</td>
<td>Device Address Type 0x00: public, 0x01: random (uint8_t type)</td>
<td>1 byte</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>0x07</td>
<td>(reserved)</td>
<td>1 byte</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>0x08</td>
<td>Device Name (device_name structure)</td>
<td>66 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>namelen</td>
<td>1 byte</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>65 byte</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>0x4A</td>
<td>Advertising Information (RBLE_ADV_INFO structure)</td>
<td>18 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>interval</td>
<td>2 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>delay</td>
<td>1 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>ch_map</td>
<td>1 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>loop_cnt</td>
<td>1 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>tx_pwr</td>
<td>1 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>own_addr</td>
<td>6 byte</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>own_addr_type</td>
<td>1 byte</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>data_cnt</td>
<td>1 byte</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>2 byte</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>evt_permit</td>
<td>1 byte</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>use_wl</td>
<td>1 byte</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>0x5C</td>
<td>Non-connectable Undirected Advertising packet data (RBLE_ADV_DATA structure)</td>
<td>32 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>len</td>
<td>1 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>31 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>0x7C</td>
<td>Scannable Undirected Advertising packet data (RBLE_ADV_DATA structure)</td>
<td>32 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>len</td>
<td>1 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>data</td>
<td>31 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>0x9C</td>
<td>Scan Response packet data (RBLE_ADV_DATA structure)</td>
<td>32 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>len</td>
<td>1 byte</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
Connect Application uses Code Flash Library for updating the Code Flash memory. Regarding to the details about Code Flash Library, refer to RL78 Family Flash Self-Programing Library Type01 User's Manual (R01US0050).

### 5.4.2 Accessing to Data Flash memory

Connect Application uses Data Flash memory to save parameters, which needs to be stored when power shutdown. Thus, by using Data Flash Library, Connect Application reads and writes data in Data Flash memory.

Table 5-9 shows the stored data in Data Flash memory.

<table>
<thead>
<tr>
<th>Data ID</th>
<th>data</th>
<th>size</th>
<th>reading</th>
<th>writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x02</td>
<td>Paring Information (con_pairing_t structure)</td>
<td>In starting Connect Application, copy to the variable in RAM.</td>
<td>When disconnection in Connect Application, if pairing information is updated by connecting, write to Data Flash memory.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>peer device address</td>
<td>6 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>peer device address type</td>
<td>1 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>security status</td>
<td>1 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>encryption key information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDIV (Encrypted Diversifier)</td>
<td>2 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random Number</td>
<td>8 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LTK (Long Term Key)</td>
<td>16 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td>Flash memory updating count (con_flashcnt_t structure)</td>
<td></td>
<td></td>
<td>When disconnection in Connect Application, if either Pairing information, Advertising information, or Advertising Data is updated, write flash memory.</td>
</tr>
<tr>
<td></td>
<td>Code Flash memory updating count</td>
<td>2 byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data Flash memory updating count</td>
<td>2 byte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Connect Application uses Data Flash Library for reading and writing Data Flash memory. Regarding to the details about Data Flash Library, refer to RL78 Family EEPROM Emulation Library Pack02 User's Manual (R01US0068).

Note: Specification of Stored Data in Flash memory is changed from the specification of Rev1.00. When evaluate this Sample Program Rev.1.10 by using RL78/G1D Evaluation Board which was used for evaluating the Sample Program Rev.1.00, it is necessary to erase Data Flash memory by using Renesas Flash Programmer (RFP), etc.

Regarding to erasing by using RFP, refer to subsection 2.3.2 "[Operation Setting] Tabbed Page" in Renesas Flash Programmer V3.02 Flash memory programming software User's Manual (R20UT3841).
5.5 Supporting Status of Protocol Stack Functions

This subsection shows supporting status of functions implemented in Protocol Stack for the Sample Program. The supported functions are listed in below tables.

<table>
<thead>
<tr>
<th>Table 5-10 Software Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Configuration</strong></td>
</tr>
<tr>
<td>Embedded configuration</td>
</tr>
<tr>
<td>Modem configuration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5-11 GAP Role</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GAP Role</strong></td>
</tr>
<tr>
<td>Broadcaster</td>
</tr>
<tr>
<td>Observer</td>
</tr>
<tr>
<td>Central</td>
</tr>
<tr>
<td>Peripheral</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5-12 Protocol Stack Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Protocol Stack Layer</strong></td>
</tr>
<tr>
<td>LL</td>
</tr>
<tr>
<td>GAP</td>
</tr>
<tr>
<td>SM</td>
</tr>
<tr>
<td>GATT</td>
</tr>
<tr>
<td>VS</td>
</tr>
<tr>
<td>Adopted Profile (Note1)</td>
</tr>
<tr>
<td>Custom Profile (Note2)</td>
</tr>
</tbody>
</table>

Note1: Adopted Profile
It is a GATT-based profile adopted by Bluetooth SIG.

Note2: Custom Profile
It is a profile defined uniquely by user.

Regarding to the details, refer to below website.
https://www.bluetooth.com/specifications/gatt

<table>
<thead>
<tr>
<th>Table 5-13 Optional Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optional Function</strong></td>
</tr>
<tr>
<td>RWKE</td>
</tr>
<tr>
<td>SLEEP</td>
</tr>
<tr>
<td>RSCIP</td>
</tr>
<tr>
<td>DTM 2Wire-UART</td>
</tr>
<tr>
<td>Adaptable</td>
</tr>
<tr>
<td>Peak current notification</td>
</tr>
<tr>
<td>FW update</td>
</tr>
<tr>
<td>HCI packet monitor</td>
</tr>
<tr>
<td>DataFlash read / write</td>
</tr>
<tr>
<td>CodeFlash write</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5-14 Hardware Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HW Configuration</strong></td>
</tr>
<tr>
<td>RF high-speed clock output</td>
</tr>
<tr>
<td>external clock MCU operation</td>
</tr>
<tr>
<td>RF External Power Amplifier</td>
</tr>
</tbody>
</table>
### 5.6 Hardware Resources used

Table 5-15 shows the hardware resources used by the Sample Program with default settings.

<table>
<thead>
<tr>
<th>Table 5-15 Hardware Resource used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RL78/G1D MCU Unit</strong></td>
</tr>
<tr>
<td><strong>Clock generator</strong></td>
</tr>
<tr>
<td>• use 8MHz from High-speed On-chip Oscillator as MCU main system clock</td>
</tr>
<tr>
<td>Common</td>
</tr>
<tr>
<td>• not use XT1 oscillator (use RF on-chip oscillator for generating RF slow clock)</td>
</tr>
<tr>
<td><strong>Clock output/buzzer output</strong></td>
</tr>
<tr>
<td>• not output clock generated XT1 oscillation from PCLBUZ0 pin</td>
</tr>
<tr>
<td><strong>Timer Array Unit</strong></td>
</tr>
<tr>
<td>• use TM00, and set operation clock CK00 to 1MHz</td>
</tr>
<tr>
<td><strong>Serial array unit</strong></td>
</tr>
<tr>
<td>• use CSI21</td>
</tr>
<tr>
<td>DTM Application</td>
</tr>
<tr>
<td>• use UART0</td>
</tr>
<tr>
<td><strong>DMA controller</strong></td>
</tr>
<tr>
<td>• use DMA2 and DMA3</td>
</tr>
<tr>
<td>DTM Application</td>
</tr>
<tr>
<td>• use DMA0 and DMA1</td>
</tr>
<tr>
<td><strong>Interrupt</strong></td>
</tr>
<tr>
<td>• use INTRF, INTDMA2, INTDMA3, and INTTM00</td>
</tr>
<tr>
<td>Beacon Application and Connect Application</td>
</tr>
<tr>
<td>• use INTP5</td>
</tr>
<tr>
<td>DTM Application</td>
</tr>
<tr>
<td>• use INTDMA0, INTDMA1, INTSR0, INTSRE0, and INTST0</td>
</tr>
<tr>
<td><strong>Port</strong></td>
</tr>
<tr>
<td>• use P10, for DIP switch SW6 position-1 input on the evaluation board</td>
</tr>
<tr>
<td>• use P16, for switch SW2 input on the evaluation board</td>
</tr>
<tr>
<td>• use P60, for controlling LED4 on the evaluation board</td>
</tr>
<tr>
<td>• use P120, P147, P03, and P60, for controlling LED1, 2, 3, and 4 on the evaluation board</td>
</tr>
<tr>
<td><strong>RL78/G1D RF Unit</strong></td>
</tr>
<tr>
<td><strong>DC-DC Converter</strong></td>
</tr>
<tr>
<td><strong>Oscillator for RF slow clock</strong></td>
</tr>
<tr>
<td><strong>GPIO00</strong></td>
</tr>
<tr>
<td><strong>GPIO1</strong></td>
</tr>
<tr>
<td><strong>GPIO2</strong></td>
</tr>
<tr>
<td><strong>GPIO3</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>RL78/G1D Evaluation Board</strong></td>
</tr>
<tr>
<td><strong>Input functions</strong></td>
</tr>
<tr>
<td>• use DIP switch SW6 position-1, for selecting application</td>
</tr>
<tr>
<td>Beacon Application and Connect Application</td>
</tr>
<tr>
<td>• use push switch SW2, for switching application</td>
</tr>
<tr>
<td><strong>Display</strong></td>
</tr>
<tr>
<td>• use LED4, for indicating that the Sample Program is started</td>
</tr>
<tr>
<td>Connect Application</td>
</tr>
<tr>
<td>• use LED1, for indicating that data is encrypted</td>
</tr>
<tr>
<td>• use LED2, for indicating that connection is established</td>
</tr>
<tr>
<td>• use LED3, for indicating that Connect Application is started</td>
</tr>
<tr>
<td>• use LED4, for indicating that the Sample Program is started</td>
</tr>
</tbody>
</table>
5.7 Compiler
The library of Beacon Stack is generated by below compiler. It is necessary to use CC-RL compiler for developing application which uses Beacon Stack.

Compiler : Renesas CC-RL V1.04.00

5.8 Memory Model
The memory model of Beacon Stack is medium model. It is necessary to set below option in the compile option of application which uses Beacon Stack.

Memory Model : -memory_model=medium

5.9 Program Size
Table 5-16 shows the total memory usage in the Sample Program.

Target Device : R5F11AGJ
Compiler : Renesas CC-RL V1.04.00
Compile Configuration : default configuration of Sample Program released

<table>
<thead>
<tr>
<th>ROM SIZE</th>
<th>119,998 byte PROGRAM SECTION + ROMDATA SECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM SIZE</td>
<td>10,439 byte RAMDATA SECTION (not included stack memory which program consumes for calling functions and allocating auto variables)</td>
</tr>
</tbody>
</table>

Regarding to the section specification, refer to chapter 6 "SECTION SPECIFICATIONS" in CC-RL Compiler User's Manual (R20UT3123).
## 5.10 Address Map

Figure 5-8 shows the address map of the Sample Program for RL78/G1D (R5F11AGG) device.

Under-lined values are different for R5F11AGH and R5F11AGJ.

<table>
<thead>
<tr>
<th>Address</th>
<th>Area Size</th>
<th>Section</th>
<th>Section Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>128byte</td>
<td>Code Flash memory</td>
<td>.vect</td>
</tr>
<tr>
<td>0x00080</td>
<td>64byte</td>
<td>Vector table area</td>
<td>.call0</td>
</tr>
<tr>
<td>0x000C0</td>
<td>4byte</td>
<td>Option byte area</td>
<td>.option_byte</td>
</tr>
<tr>
<td>0x000C4</td>
<td>10byte</td>
<td>Security ID setting area</td>
<td>.security_id</td>
</tr>
<tr>
<td>0x000CE</td>
<td>129,842byte</td>
<td>Program area (below sections are described in no particular order)</td>
<td></td>
</tr>
<tr>
<td>0x01FC0</td>
<td>156byte</td>
<td>OCD monitor</td>
<td>.monitor1, .monitor2</td>
</tr>
<tr>
<td>0x1FC9C</td>
<td>Reserved</td>
<td>Startup</td>
<td>BOOT0_TEXT</td>
</tr>
<tr>
<td>0x20000</td>
<td>Reserved</td>
<td>Runtime library</td>
<td>.RLIB</td>
</tr>
<tr>
<td>0x20000</td>
<td>Reserved</td>
<td>Standard library</td>
<td>.SLIB</td>
</tr>
<tr>
<td>0x30000</td>
<td>8192byte</td>
<td>CodeFlash library</td>
<td>FSL_FCD, FSL_RCD, FSL_BCD, FSL_BECED</td>
</tr>
<tr>
<td>0x30000</td>
<td>40,704byte</td>
<td>DataFlash library</td>
<td>EEL_CODE, FDL_CODE</td>
</tr>
<tr>
<td>0x30000</td>
<td>12,064byte</td>
<td>Beacon Stack</td>
<td>BCN_CONST, BCN_TEXT</td>
</tr>
<tr>
<td>0x40000</td>
<td>2048byte</td>
<td>Protocol Stack</td>
<td>RBL_CODE_f, HST_CODE_f, CNT_CODE_f</td>
</tr>
<tr>
<td>0x40000</td>
<td>32byte</td>
<td>Applications</td>
<td>.const, .constf, .data, .text, .textf</td>
</tr>
<tr>
<td>0x40000</td>
<td>12,064byte</td>
<td>Unused area</td>
<td>-</td>
</tr>
<tr>
<td>0x80000</td>
<td>2048byte</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0x80000</td>
<td>8192byte</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0x80000</td>
<td>40,704byte</td>
<td>Mirror area</td>
<td></td>
</tr>
<tr>
<td>0xE0000</td>
<td>32byte</td>
<td>System Configuration area</td>
<td></td>
</tr>
<tr>
<td>0xEE000</td>
<td>256byte</td>
<td>Unused area</td>
<td></td>
</tr>
<tr>
<td>0x00000</td>
<td>131,072byte</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5-8 Address Map (R5F11AGG)**
Figure 5-9 shows the address map of the Sample Program for RL78/G1D (R5F11AGH) device.

Under-lined values are different for R5F11AGG and R5F11AGJ.

<table>
<thead>
<tr>
<th>Address</th>
<th>Area Size</th>
<th>Section</th>
<th>Section Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>128byte</td>
<td>Code Flash memory</td>
<td>Code Flash memory</td>
</tr>
<tr>
<td>0x00080</td>
<td>64byte</td>
<td>Vector table area</td>
<td>.vect</td>
</tr>
<tr>
<td>0x000C0</td>
<td>4byte</td>
<td>CALLT table area</td>
<td>.callt0</td>
</tr>
<tr>
<td>0x000C4</td>
<td>10byte</td>
<td>Option byte area</td>
<td>.option_byte</td>
</tr>
<tr>
<td>0x000CE</td>
<td>195.378byte</td>
<td>Security ID setting area</td>
<td>.security_id</td>
</tr>
<tr>
<td>0x000CE</td>
<td>195.378byte</td>
<td>Program area (below sections are described in no particular order)</td>
<td></td>
</tr>
<tr>
<td>0x000F00</td>
<td>156byte</td>
<td>System Configuration area</td>
<td></td>
</tr>
<tr>
<td>0x000F400</td>
<td>204byte</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0x000F800</td>
<td>204byte</td>
<td>Special function register (2nd SFR)</td>
<td></td>
</tr>
<tr>
<td>0x000F800</td>
<td>8192byte</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0x000F840</td>
<td>36,608byte</td>
<td>DataFlash memory</td>
<td></td>
</tr>
<tr>
<td>0x000FBF00</td>
<td>16.160byte</td>
<td>Mirror area</td>
<td></td>
</tr>
<tr>
<td>0x000FBF00</td>
<td>32byte</td>
<td>Program Resource area (below sections are described in no particular order)</td>
<td></td>
</tr>
<tr>
<td>0x00FFEE0</td>
<td>32byte</td>
<td>General-purpose register</td>
<td></td>
</tr>
<tr>
<td>0x00FFFF00</td>
<td>256byte</td>
<td>Special function register (SFR)</td>
<td></td>
</tr>
<tr>
<td>0x00FFFF00</td>
<td>256byte</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-9  Address Map (R5F11AGH)
**Figure 5-10** shows the address map of the Sample Program for RL78/G1D (R5F11AGJ) device.

Under-lined values are different for R5F11AGG and R5F11AGH.

<table>
<thead>
<tr>
<th>Address</th>
<th>Area Size</th>
<th>Section Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>128byte</td>
<td>Vector table area</td>
</tr>
<tr>
<td>0x00080</td>
<td>64byte</td>
<td>CALLT table area</td>
</tr>
<tr>
<td>0x000C0</td>
<td>4byte</td>
<td>Option byte area</td>
</tr>
<tr>
<td>0x000C4</td>
<td>10byte</td>
<td>Security ID setting area</td>
</tr>
<tr>
<td>0x000CE</td>
<td>258,866byte</td>
<td>Program area (below sections are described in no particular order)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OCD monitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Startup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Runtime library</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard library</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CodeFlash library</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DataFlash library</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Beacon Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Protocol Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unused area</td>
</tr>
<tr>
<td>0x3F400</td>
<td>156byte</td>
<td>System Configuration area</td>
</tr>
<tr>
<td>0x3F49C</td>
<td></td>
<td>Unused area</td>
</tr>
<tr>
<td>0x3F800</td>
<td>512byte</td>
<td>Reserved area (RL78/G1D Module only)</td>
</tr>
<tr>
<td>0x3FC00</td>
<td>6byte</td>
<td>User Information area</td>
</tr>
<tr>
<td>0x3FC06</td>
<td></td>
<td>Unused area</td>
</tr>
<tr>
<td>0x40000</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0xF0000</td>
<td>2048byte</td>
<td>Special function register(2nd SFR)</td>
</tr>
<tr>
<td>0xF0800</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>0xF1000</td>
<td>8192byte</td>
<td>DataFlash memory</td>
</tr>
<tr>
<td>0xF3000</td>
<td>32,512byte</td>
<td>Mirror area</td>
</tr>
<tr>
<td>0xFAF00</td>
<td>1024byte</td>
<td>Self RAM area (R5F11AGJ only)</td>
</tr>
<tr>
<td>0xFB300</td>
<td>20,447byte</td>
<td>RAM area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program Resource area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Protocol Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Beacon Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DataFlash library</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unused area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stack area</td>
</tr>
<tr>
<td>0xFFEE0</td>
<td>32byte</td>
<td>General-purpose register</td>
</tr>
<tr>
<td>0xFFFF00</td>
<td>256byte</td>
<td>Special function register(SFR)</td>
</tr>
<tr>
<td>0xFFFFFFFF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5-10** Address Map (R5F11AGJ)
6. Configuration

This chapter describes the configurations for hardware and application of the Sample Program.

6.1 Hardware configuration

For using Protocol Stack and Beacon Stack, major hardware configurations are arranged to macro definitions in r_config.h. Regarding to the details about macro definitions, refer to following subsections.

Project_Source\application\src\r_config.h, line 34-86

```c
/*
 * CONFIGURATIONS (NEED TO CHANGE BELOW DEFINES AS NECESSARY)
*******************************************************************************
* MCU Main System Clock (either clock frequency of 4MHz,8MHz,16MHz,32MHz) *
* Note: It is necessary to set Option Bytes Value at Device Setting of Linker Option *
#define MCU_HOCO_CLK (8)
*
* RF Operation (0:enable both Tx and Rx, 1:enable Tx only) *
#define RF_TX_ONLY (0)
*
* RF DC-DC Converter (0:disable DC-DC, 1:enable DC-DC) *
#define RF_DCDC_EN (1)
*
* RF Slow Clock Source (0:RF On-Chip Oscillator, 1:MCU XT1 Oscillator) *
#define RF_SLK_XT1 (0)
*
* RF Slow Clock Calibration (0:not execute, 1:execute)                    *
* Note: This configuration is only for Beacon Stack                     *
*       : RF Slow Clock Calibration is only for RF-On_Chip_Oscillator     *
*       : Protocol Stack always execute RF on chip oscillator calibration *
#define RF_SLK_CAL (1)
*
* RF 32MHz Oscillation Stabilization Time (usec, at least 550usec)     *
* Note: This configuration is only for Beacon Stack                  *
#define RF_32MHZ_WAIT (1000)
*
* Maximum number of Simultaneous Connections (fixed 1)               *
* Note: This configuration is only for BLE Protocol Stack          *
#define MAX_CONNECTION (1)
*
* System Configuration Address in CodeFlash memory *
#if defined(_USE_R5F11AGG)
    /* System Configuration is located the last block */
    #define SYSCFG_ADDR (0x1FC00)
#elif defined(_USE_R5F11AGH)
    /* System Configuration is located the third last block */
    #define SYSCFG_ADDR (0x3FC00)
#elif defined(_USE_R5F11AGJ)
    /* System Configuration is located the last block */
    #define SYSCFG_ADDR (0x3FC00)
#else
    /* In the case of RL78/G1D Module (RY7011), Device Address is located the last block */
    #define MODCFG_ADDR (0x3FC00)
#endif
*/
```
6.1.1 MCU main system clock frequency

Clock generated by Hi-speed On-chip Oscillator is used as MCU main system clock, and selectable frequency of MCU main system clock is 4, 8, 16 and 32MHz. In the Sample Program, frequency of MCU main system clock is defined by the macro MCU_HOCO_CLK and Option Bytes. The default setting of clock frequency is 8 (MHz).

If changing the frequency of MCU main system clock, change the macro value to one of the values: 4 (MHz), 8 (MHz), 16 (MHz), 32 (MHz).

Option Bytes is set to the linker option "-user_opt_byte". Regarding to the value of Option Bytes, refer to Table 6-1.

<table>
<thead>
<tr>
<th>Option Bytes setting</th>
<th>Clock frequency</th>
<th>Flash Operation Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(any)</td>
<td>(any)</td>
<td></td>
</tr>
<tr>
<td>2B</td>
<td>4MHz</td>
<td>low-voltage main mode</td>
</tr>
<tr>
<td>AA</td>
<td>8MHz</td>
<td>low-speed main mode</td>
</tr>
<tr>
<td>E9</td>
<td>16MHz</td>
<td>high-speed main mode</td>
</tr>
<tr>
<td>E8</td>
<td>32MHz</td>
<td></td>
</tr>
</tbody>
</table>

Regarding to the details about Option Bytes, refer to chapter 25 "OPTION BYTE" in RL78/G1D User's Manual: Hardware (R01UH0515). CPU operation voltage varies with respect to CPU clock frequency. Regarding to the operation voltage, refer to section 30.2 "Operating Voltage" in RL78/G1D User's Manual: Hardware (R01UH0515).

1) Using CS+ for CC

In the case of CS+ for CC about how to set Option Bytes, follow the below steps.

1. Right-click to [CC-RL] of the subproject "R5F11AGJ_BcnCmb" in the project tree.
2. Select [Property] in right click menu.
3. Set the Option Bytes at the [Device] [User option byte value] of [Link Options] tab.

2) Using e2 studio

In the case of e2 studio about how to set Option Bytes, follow the below steps.

1. Right-click to "R5F11AGJ_BcnCmb" project.
2. Select [Renesas Tool Settings] in right click menu.
3. Set the Option Bytes at the [Linker] [Device] [User option byte value] of [Tool Settings] tab.
6.1.2 RF Operation

It is possible to select whether to enable both Tx and Rx or only Tx when Beacon Stack works. When enabling only Tx is selected, RF initialization time is shortened. In the Sample Program, whether to enable both Tx and Rx or only Tx is defined by the macro RF_TX_ONLY. The default setting is 0, which means that RF operation is enabled both Tx and Rx.

If need to enable only Tx, change the macro value to 1.

```
Project_Source\application\src\r_config.h, line 42-44
```

6.1.3 RF on-chip DC-DC converter

In the Sample Program, whether to use RF on-chip DC-DC converter is defined by the macro RF_DCDC_EN. Thus, it is possible to select whether to use RF on-chip DC-DC converter or not. The default setting is 1, which means that RF on-chip DC-DC converter is used.

If not using RF on-chip DC-DC converter, change the macro value to 0.

```
Project_Source\application\src\r_config.h, line 46-47
```

6.1.4 RF slow clock source

RF slow clock is needed to RF unit for counting the period, and it is possible to select as a source of RF clock from either RF on-chip oscillator or MCU unit XT1 oscillator. In the Sample Program, RF slow clock source is defined by the macro RF_SLK_XT1. The default setting is 0, which means that RF on-chip oscillator is selected as a source for RF slow clock.

If changing RF slow clock source to MCU unit XT1 oscillator, change the macro value to 1. By changing the macro to 1, clock generated by MCU unit XT1 oscillator is supplied to RF unit via EXSLK_RF pin.

```
Project_Source\application\src\r_config.h, line 49-50
```

6.1.5 RF on-chip oscillator calibration

In the case of using RF on-chip oscillator as a source of RF slow clock, calibrating accuracy of clock generated by RF on-chip oscillator is always executed when Protocol Stack works. But it is possible to select whether to execute calibration or not when Beacon Stack works. In the Sample Program, whether to execute calibration is defined by the macro RF_SLK_CAL. The default setting is 1, which means that the calibration is executed.

Beacon Stack executes calibration only once, just after the end of the transmitting first advertising packet followed by RF initialization. By executing calibration, the accuracy of advertising interval is improved.

If not executing calibration, change the macro value to 0.

```
Project_Source\application\src\r_config.h, line 52-56
```
6.1.6 RF base clock oscillation stabilization time

In the Sample Program, the oscillation stabilization time is defined by the macro RF_32MHZ_WAIT. Thus, it is necessary to optimize the oscillation stabilization time of XTAL_RF oscillator for using RF base clock, which is depending on the 32MHz resonator connected to XTAL1_RF and XTAL2_RF pin. The default setting is 1000 (usec) which is suitable for the particular RL78/G1D Evaluation Board.

If changing the oscillation stabilization time, change the macro value to the time, as a minimum 550 (usec).

```
58:    /* RF 32MHz Oscillation Stabilization Time (usec, at least 550usec) */
59:    /* Note: This configuration is only for Beacon Stack */
60:    */    : Stabilization Time needs to be optimized for 32MHz resonator */
61:    #define RF_32MHZ_WAIT (1000)
```

Regarding to the details about RF base clock generator, refer to subsection 15.3.9 "RF clock generator circuit block" in RL78/G1D User's Manual: Hardware (R01UH0515).

6.1.7 Maximum number of Simultaneous connection

The Sample Program performs as Peripheral Role, so maximum number of simultaneous connection is fixed to 1. In the Sample Program, the maximum number is defined by the macro MAX_CONNECTION. The default setting is 1, which means that only one connection is established to peer Central Role device.

```
63:    /* Maximum number of Simultaneous Connections (fixed 1) */
64:    /* Note: This configuration is only for BLE Protocol Stack */
65:    */    : fixed 1, Connect Application behaves as peripheral device */
66:    #define MAX_CONNECTION (1)
```

6.1.8 HCI Monitoring

BLE Protocol Stack provides monitoring HCI sequence for debugging purpose. By enabling this function, you can monitor HCI log packet through UART1, and understand how Protocol Stack works. In the Sample Program, whether to enable HCI monitoring or not is defined by the macro PKTMON_EN. The default setting is 0, which means that HCI monitoring is disabled.

If enabling HCI monitoring, change the macro value to 1.

```
68:    /* Packet Monitoring (0:disable Packet Monitor, 1:enable Packet Monitor) */
69:    /* Note: This configuration is only for BLE Protocol Stack */
70:    */    : Packet Monitoring uses UART1 for using output HCI log */
71:    #define PKTMON_EN (0)
```

In order to confirm the contents of HCI log packets, PC and specific application software is needed.

Regarding to the details about how to use the HCI monitoring, refer to chapter 12 "HCI Packet Monitoring Feature" in Bluetooth Low Energy Protocol Stack User's Manual (R01UW0095).
6.1.9 System Configuration Address

In the Code Flash memory, it is possible to store information as system configuration outside of the firmware. By setting each different system configuration for different devices, it is possible to configure the information without rebuilding firmware. For example, this information includes device address, advertising data, and etc. In the Sample Program, the address of system configuration is defined by the macro SYSCFG_ADDR.

If needed to re-assign the address map, change the macro value to new address.

```
73: /* System Configuration Address in CodeFlash memory */
74: #if defined(_USE_R5F11AGG)
75:    /* System Configuration is located the last block */
76:    #define SYSCFG_ADDR (0x1FC00)
77: #elif defined(_USE_R5F11AGH)
78:    /* System Configuration is located the last block */
79:    #define SYSCFG_ADDR (0x2FC00)
80: #elif defined(_USE_R5F11AGJ)
81:    /* System Configuration is located the third last block */
82:    /* by taking into account the location of RL78/G1D module (RY7011) */
83:    #define SYSCFG_ADDR (0x3F400)
84:    /* In the case of RL78/G1D Module (RY7011), Device Address is located the last block */
85:    #define MODCFG_ADDR (0x3FC00)
86: #endif
```

Regarding to the details about System Configuration, refer to subsection 5.4.1 "Accessing to Code Flash memory" in this document.

6.1.10 Switches on RL78/G1D Evaluation Board

For switching application, the Sample Program uses switches on the evaluation board. DIP switch SW6 position-1 switches either Beacon Operation or RF Evaluation Operation. Switch SW2 switches either Beacon Application or Connect Application alternately. In the Sample Program, whether to use switches or not is defined by the macro EVB_SW. The default setting is 1, which means that switches are used.

If need not to use switches on the evaluation board, change the macro value to 0.

```
51: /* Switches on RL78/G1D Evaluation Board (0:not to use, 1:use) */
52: /* Operation: */
53: /* When use Switches: */
54: /* - After power up, Beacon Application starts running at first */
55: /* - It is possible to switch Beacon Application and Connect Application alternately */
56: /* When no Switches: */
57: /* - After power up, Connect Application starts running at first */
58: /* - If connection is not established within 30sec, Connect Application stops and */
59: /* Beacon Application starts running */
60: /* - It is not possible to switch from Beacon Application to Connect Application */
61: #define EVABOARD_SWITCH_EN (1)
```
Figure 6-1 shows application switch operation when the Sample Program uses switches on the evaluation board. The Sample Program uses Switch SW2 and SW6 position-1 for switching application. When switch SW6 position-1 is ON, the Sample Program executes DTM Application. When switch SW6 position-1 is OFF, the Sample Program executes Beacon Application and Connect Application. After power on, Beacon Application runs at first. Pushing switch SW2 can switch Beacon Application and Connect Application alternately.

![Application Switch Operation Diagram](image)

Figure 6-1 Application Switch Operation when the Sample Program uses switches on the evaluation board

Figure 6-2 shows application switch operation when the Sample Program uses no switch on the evaluation board. The Sample Program executes Beacon Application and Connect Application. After power on, Connect Application runs at first. If connection is not established within 30 seconds, Beacon Application runs. To execute Connect Application again, it is necessary to reset the MCU.

![Application Switch Operation Diagram](image)

Figure 6-2 Application Switch Operation when the Sample Program uses no switch on the evaluation board
6.2 Application Configuration

6.2.1 System Configuration

System configuration is located outside of firmware in the Code Flash memory. Thus it is possible to write System configuration and firmware at the same time, by using Unique Code Embedding Function of Renesas Flash Programmer.

As a sample, the release package includes unique code file for system configuration. Regarding to the details about system configuration defined by Sample Program, refer to subsection 5.4.1 "Accessing to Code Flash memory" in this document.

RUC_File\r5f11agj_syscfg.ruc, line 1-10

```
1:    //  =========================================================
2:    //  -- System Configuration for RL78/G1D BLE Protocol / Beacon Stack Combination Sample Program --
3:    //  --   Device Part Number : R5F11AGJ           --
4:    //  =========================================================
5:    format hex
6:    area user flash
7:    address 0x3f400
8:    size 156
9:    index data
10:    000001 B19A7856341200012524C37382F47314420426561636F6620303310000000000000000000000000000000000000000000
63676F6F2E676C2F3764694C54780000001B180952656653617320524C37382F47314420426561636F6E303100000000

(a): index of unique code data
(b): device address (6byte)
(c): device address type (1byte), padding (1byte)
(d): device name (66byte)
(e): advertising information (18byte)
(f): advertising data (32byte)
(g): scan response data (32byte)
```

The sample unique code file for R5F11AGJ describes below.

line 1-4 : The lines starting with // are comment line.
line 5  : specifies the format as hexadecimal format
line 6  : specifies the area as User area
line 7  : specifies the address as 0x3F400 (block 253)
line 8  : specifies the size 124 byte
line 9  : declares the unique code data starts at the next line
line10 : specifies the index and unique code
         (a): index of unique code data
         (b): device address (6byte)
         (c): device address type (1byte), padding (1byte)
         (d): device name (66byte)
         (e): advertising information (18byte)
         (f): advertising data (32byte)
         (g): scan response data (32byte)
6.2.2 Kernel Heap Memory Configuration

BLE Protocol Stack includes a kernel called RWKE. The kernel provides below functions.

- Event Management
- Message Communication Management
- Task State Management
- Timer Management
- Memory Management.

When execute kernel functions, the kernel allocates a partial area dynamically from an area predefined as heap memory. If there are a lot of load for the kernel, heap memory may be exhausted. When run short of heap memory by enhancing Connect Application, change the macro APP_HEAP_SIZE value, which defines heap memory size.

By the way, unused area of RAM is used as stack memory. If heap memory is too big, stack-overflow of stack memory may occur.

Project_Source\application\src\connect\resource\r_kernel.c, line 56-65

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>56:</td>
<td>#define APP_HEAP_SIZE (0)</td>
</tr>
<tr>
<td>57:</td>
<td>/* Note: When Kernel Heap size is not enough, it is necessary to increase APP_HEAP_SIZE */</td>
</tr>
<tr>
<td>58:</td>
<td></td>
</tr>
<tr>
<td>59:</td>
<td>#define BLE_HEAP_SIZE ((MAX_CONNECTION * 256) + 512 \</td>
</tr>
<tr>
<td>60:</td>
<td>+ BLE_HEAP_CONT \</td>
</tr>
<tr>
<td>61:</td>
<td>+ (BLE_HEAP_HOST * MAX_CONNECTION) \</td>
</tr>
<tr>
<td>62:</td>
<td>+ BLE_DB_SIZE \</td>
</tr>
<tr>
<td>63:</td>
<td>+ RBLE_TABLE_SIZE \</td>
</tr>
<tr>
<td>64:</td>
<td>+ APP_HEAP_SIZE \</td>
</tr>
<tr>
<td>65:</td>
<td>)</td>
</tr>
</tbody>
</table>

6.2.3 Advertising Configuration

Default Advertising configuration for Beacon Application is defined in r_beacon.c file.

If needed to change the default Advertising information or default Advertising data, modify the value of RBLE_ADV_INFO structure or RBLE_ADV_DATA structure. Regarding of these structure specification, refer to chapter 4 "API" in RL78/G1D Beacon Stack User's Manual (R01UW0171).

Project_Source\application\src\beacon\r_beacon.c, line 42-92

```c
42: /* Advertising Data Array */
43: static RBLE_ADV_DATA adv_data[] =
44: {
45:    /* Advertising Data[0] */
47:    { /* Advertising data length */
48:        27,
49:        /* Advertising data <<Flags>> */
50:        0x02, 0x01, 0x06,
51:        /* Advertising data <<Complete List of 16-bit Service Class UUIDs>> */
52:        0x03, 0x03, 0xAA, 0xFE,
53:        /* Advertising data <<Service Data>> */
54:        'g', 'o', 'o', '.', 'g', 'l', '/', '5', 'w', 'K', 'k', 'R', 'K'
55:    },
56:    #if !RF_TX_ONLY
57:    /* Scan Response Data[0] */
58:    { /* Scan Response data length */
59:        25,
60:        /* Scan Response data <<Complete local name>> */
61:        0x18, 0x09,
62:        'R', 'e', 'n', 'e', 's', 'a', 's', ' ', 'R', 'L', '7', '8', '/', 'G', '1', 'D',
63:        'B', 'e', 'a', 'c', 'o', 'n'
64:    },
65:    #endif
66:    },
67:    #if RF_TX_ONLY
68:    static const uint8_t adv_type = RBLE_PDU_ADV_NONCONN_IND;
69:    #else
70:    static const uint8_t adv_type = RBLE_PDU_ADV_SCAN_IND;
71:    #endif
72:    /* Advertising Information */
73: static RBLE_ADV_INFO adv_info =
74: {
75:    0x00A0, /* Advertising Interval */
76:    true, /* Advertising Interval Delay */
77:    RBLE_ADV_ALL_CHANNELS, /* Advertising Channel Map */
78:    0x00, /* Advertising Transfer Count */
79:    RBLE_TXPW_LVL9, /* Advertising Transfer Power */
80:    { 0xB0, 0x9A, 0x7B, 0x56, 0x34, 0x12 }, /* Own Device Address */
81:    RBLE_ADDR_PUBLIC, /* Own Device Address Type */
82:    sizeof(adv_data) / sizeof(RBLE_ADV_DATA), /* Advertising Data Count */
83:    &adv_data[0], /* Advertising Data */
84:    RBLE_EVT_PERMIT_ADV_ALL, /* Advertising Event Permission */
85:    false /* Use White List */
86:    }
87:};
```
If transmitting multiple Advertising data repeatedly, increase the number of RBLE_ADV_DATA structure array.

**Example Code for transmitting multiple Advertising data**

```c
/* Advertising Data Array */
static RBLE_ADV_DATA adv_data[] =
{
    /* Advertising data No.1 */
    {
        /* Advertising data length */
        ...
        /* Advertising data */
        ...
    },
    /* Advertising data No.2 */
    {
        /* Advertising data length */
        ...
        /* Advertising data */
        ...
    },
    ...
};
```
Default Advertising configuration for Connect Application is defined in r_connect.c file.

If needed to change the default Advertising information, default Advertising data and default Scan Response data, modify the value of RBLE_ADV_INFO structure. Regarding to the specification of its structure, refer to chapter 5 "Generic Access Profile" in Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

```c
Project_Source\application\src\connect\r_connect.c, line 148-181

148: /* Advertising Information for connection as a slave role */
149: /* Note : it is necessary to change configuration corresponds to each use case */
150: static RBLE_ADV_INFO broadcast_info =
151: {
152:     /* Advertising Parameter structure */
153:     {
154:         0x0030, /* Advertising Interval Min */
155:         0x0030, /* Advertising Interval Max */
156:         RBLE_GAP_ADV_CONN_UNDIR, /* Advertising Type */
157:         RBLE_ADDR_PUBLIC, /* Own Address Type */
158:         0x00, /* Direct Advertising Address Type */
159:         { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }, /* Direct Advertising Address */
160:         RBLE_ADV_ALL_CHANNELS, /* Advertising Channel Map */
161:         RBLE_ADV_ALLOW_SCAN_ANY_CON_ANY, /* Advertising Filter Policy */
162:         0x00, /* (reserved) */
163:     },
164:     /* Advertising Data structure */
165:     {
166:         /* Advertising data length (max 31byte) */
167:         3+25,
168:         /* Advertising data <<Flags>> */
169:         2, 0x01, 0x06,
170:         /* Advertising data <<Complete Local Name>> */
171:         24, 0x09,
172:         'R','L','7','8','/','G','1','D',' ','B','e','a','c','o','n',' ','U','p','d','a','t','e','r'
173:     },
174:     /* Scan Response Data structure */
175:     {
176:         /* Scan Response data length (max 31byte) */
177:         0,
178:         /* Scan Response data */
179:         0x00
180:     }
181: };
```

To specify peer device for establishing connection, set RBLE_ADV_ALLOW_SCAN_ANY_CON_WLST as the advertising filter policy of RBLE_ADV_INFO structure, in order to enable White List. Before starting Advertising, call RBLE_GAP_Add_To_White_List in order to add device address to the White List.

If specifying multiple peer device, call RBLE_GAP_Add_To_White_List for adding each peer device address.

**Example Code for adding device address to White List**

```c
uint9_t wl_cnt;
/* device address list for white list */
static RBLE_DEV_ADDR_INFO wl_info[] =
{
    {RBLE_ADDR_PUBLIC, { 0x01, 0x90, 0x78, 0x56, 0x34, 0x12 }},
    ...
};
/* set device address to white list */
/* it is necessary to call repeatedly until all device address of list is added */
if(wl_cnt < (sizeof(wl_info) / sizeof(RBLE_DEV_ADDR_INFO)))
{
    RBLE_GAP_Add_To_White_List(&wl_info[wl_cnt++]);
}
```
Note: White List should not be used for Resolvable Private address, so Resolvable Private address is changed regularly by generating with Identity Resolving Key.

6.2.4 No Connection Timeout Time Configuration

Connect Application finishes if connection is not established from either application start-up or previous disconnection within the no connection timeout time. The timeout time is defined by the macro CON_TIME_OUT. It is possible to set the timeout time in the range of 10 to 299,990 milli-seconds, and in increments of 10milli-seconds. The default setting is 30 seconds.

If changing the timeout time, change the CON_TIME_OUT macro value in the range of 1 to 29,999.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.</td>
<td>/* Connect Application Exit Timeout Time (unit: 10msec) */</td>
<td></td>
</tr>
<tr>
<td>49.</td>
<td>/* When connection is not established within this time, Connect Application exits. */</td>
<td></td>
</tr>
<tr>
<td>50.</td>
<td>#define CON_EXIT_TIME (3000)</td>
<td></td>
</tr>
</tbody>
</table>

To monitor the timeout time, the application uses the kernel timer of BLE Protocol Stack. Regarding to the specification of the kernel timer, refer to section 9.5 “Timer Management” in Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).
6.2.5 Paring Configuration

Default pairing configuration is defined in r_connect.c file. The Sample Program executes Just Works pairing method in pairing sequence.

When changing pairing method from Just Works to Passkey Entry, it is necessary to provide Passkey from the application. Thus set RBLE_IO_CAP_DISPLAY_ONLY as the IO capabilities and set RBLE_AUTH_REQ_MITM_BOND as the authentication requirements of RBLE_BOND_RESP_PARAM structure.

When executing paring sequence with Passkey Entry, RBLE_SM_TK_REQ_IND event occurs. Therefore, the application is needed to call RBLE_SM_Tk_Req_Resp in order to respond Passkey as the Temporary Key. Below example code generates Passkey with rand function of standard library. After generating, it is necessary to display the passkey to user on the display like LCD screen.

Example Code for responding Temporary Key

```
uint32_t passkey;
uint8_t* byteptr = (uint8_t*)&passkey;
uint8_t idx;

/* TK(Temporary Key) buffer */
RBLE_KEY_VALUE tk =
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

/* generate Passkey (range:000,000 - 999,999) */
passkey = (uint32_t)rand();
passkey |= (uint32_t)rand() << 16;
passkey %= 1000000;

/* copy Passkey to TK(Temporary Key) buffer */
for(idx = 0; idx < sizeof(uint32_t); idx++)
{  tk.key[RBLE_KEY_LEN - 1 - idx] = byteptr[idx];
}

RBLE_SM_Tk_Req_Resp(ccn_idx, RBLE_OK, &tk);
```
6.2.6 Custom Profile

In order to implement or change Custom Profile, it is necessary to implement below definitions, resources, and processing. Detail will be explained in the following.

- **definitions:**
  - UUIDs of Custom Profile service and Characteristics, for identifying them.
  - Attribute indexes of Custom Profile Service and Characteristics, for operating by Protocol Stack.
  - Attribute handles of Custom Profile Service and Characteristics, for announcing to Client device

- **resources:**
  - variables of Custom Profile Characteristics, for storing each Characteristic Value
  - descriptors of Custom Profile Service and characteristics, for setting features to Attribute database.
  - descriptors of Custom Profile Characteristics, for setting Characteristic variable to Attribute database.
  - Attribute database, which is accessed by Protocol Stack.

- **processing:**
  - enabling GATT after establishing connection, and registering event callback function.
  - updating Characteristic Value by Write Request from client device, and sending Write Response.

Note1: It is not necessary to implement the processing for responding Characteristic Value to the Read Request from Client device. The response is executed by Protocol Stack automatically.

Note2: If it is necessary to inform Characteristic Value in the timing that determined by server device, the processing of sending Notification or Indication is needed. In the Sample Program, processing for Notification or Indication is not implemented. If sending Notification or Indication, use RBLE_GATT_Notify_Request or RBLE_GATT_Indicate_Request of Protocol Stack API respectively.

Regarding to the specification of those functions, refer to subsection 7.2.9 "RBLE_GATT_Notify_Request" and 7.2.10 "RBLE_GATT_Indicate_Request" in Bluetooth Low Energy Protocol Stack API Reference Manual: Basics (R01UW0088).

(1) Definitions

UUIDs are defined in r_profile.h file.

It is possible to generate randomly numbers UUID by using UUIDGEN Linux command. It is necessary to set UUID value in LSB order and UUIDs value are set to each descriptor of Service and Characteristic.

```
#define PRF_UUID_SERVICE
{0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x01,0x00,0x66,0xA7}
#define PRF_UUID_CHAR_BCNINFO
{0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x02,0x00,0x66,0xA7}
#define PRF_UUID_CHAR_BCNDATA
{0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x03,0x00,0x66,0xA7}
#define PRF_UUID_CHAR_CFLCNT
{0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x04,0x00,0x66,0xA7}
#define PRF_UUID_CHAR_DFLCNT
{0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x05,0x00,0x66,0xA7}
#define PRF_UUID_CHAR_RSPDATA
{0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x06,0x00,0x66,0xA7}
```
Attribute indexes and Attribute handles are defined in r_gatt.h file.

It is possible to add or delete Service or Characteristic of Custom Profile.

The Attribute index values are set directly to Attribute database, and Attribute handle values are set to each descriptor of Service and Characteristic.

```
37:    /* Attribute index */
38:    enum
39:    {
60:        /* offset index for Custom Profiles */
61:        ATT_IDX_CUSTOM = 0x0200,
65:        /* Custom Profile Service */
66:        PRF_IDX_SVC,
68:        PRF_IDX_BCNINFO_CHAR,
69:        PRF_IDX_BCNINFO_VAL,
70:        PRF_IDX_BCNDATA_CHAR,
71:        PRF_IDX_BCNDATA_VAL,
72:        PRF_IDX_RSPDATA_CHAR,
73:        PRF_IDX_RSPDATA_VAL,
74:        PRF_IDX_CFCLNT_CHAR,
75:        PRF_IDX_CFCLNT_VAL,
76:        PRF_IDX_DFLCNT_CHAR,
79:        PRF_IDX_DFLCNT_VAL;
81:    };
82:    /* Custom Profile Service */
83:    enum
99:        PRF_HDL_SVC                   = 0x000C,
101:       PRF_HDL_BCNINFO_CHAR        = 0x000D,
102:       PRF_HDL_BCNINFO_VAL         = 0x000E,
103:       PRF_HDL_BCNDATA_CHAR        = 0x000F,
104:       PRF_HDL_BCNDATA_VAL         = 0x0010,
105:       PRF_HDL_RSPDATA_CHAR        = 0x0011,
106:       PRF_HDL_RSPDATA_VAL         = 0x0012,
107:       PRF_HDL_CFCLNT_CHAR         = 0x0013,
108:       PRF_HDL_CFCLNT_VAL          = 0x0014,
109:       PRF_HDL_DFLCNT_CHAR         = 0x0015,
110:      PRF_HDL_DFLCNT_VAL          = 0x0016;
```

(2) Resources

Variables for storing Characteristic Value are defined in r_profile.c file.

Initial Characteristic Value are set by application. After that, Characteristic Value which has the write permission is updated by Client device. Characteristic value variables are set to each descriptor of Characteristic.

```
64:    /* Custom Profile characteristic variables */
65:    PRF_ADV_INFO  prf_bcninfo_val; /* Advertising Information */
66:    PRF_ADV_DATA  prf_bcndata_val; /* Advertising Data */
67:    PRF_ADV_DATA  prf_rspdata_val; /* Scan Response Data */
68:    uint16_t prf_cfclnt_val; /* Code Flash Updated Count */
69:    uint16_t prf_dfclnt_val; /* Data Flash Updated Count */
```
Descriptors of Custom Profile Service and Characteristic are defined in r_gatt.c file.

It is necessary to define descriptors in order to set the features like UUID, Attribute handle, Attribute permissions and the variables of Characteristic Values to the Attribute database.

```c
103:    /* Custom Service */
104:    static const uint8_t custom_svc[RABLE_GATT_128BIT_UUID_OCTET] = PRF_UUID_SERVICE;
105:
106:    /* Advertising Information */
107:    static const struct atts_char128_desc prf_bcninfo_char =
108:    {
109:        ...
110:    };
111:
112:    /* Advertising Data */
113:    static const struct atts_char128_desc prf_bcndata_char =
114:    {
115:        ...
116:    };
117:
118:    /* Scan Response Data */
119:    static const struct atts_char128_desc prf_rspdata_char =
120:    {
121:        ...
122:    };
123:
124:    /* Code Flash Updated Count */
125:    static const struct atts_char128_desc prf_cflcnt_char =
126:    {
127:        ...
128:    };
129:
130:    /* Data Flash Updated Count */
131:    static const struct atts_char128_desc prf_dflcnt_char =
132:    {
133:        ...
134:    };
135:
```

Project_Source\application\src\connect\resource\r_gatt.c, line 103-179
Attribute database is defined in r_gatt.c file.

Attribute database is needed in order to set Service and Characteristics to Protocol Stack. Attribute database consists of Service descriptors and Characteristic descriptors.

Project_Source\application\src\connect\resource\r_gatt.c, line 213-243

```c
213:    /* Attribute Database */
214:    const struct atts_desc atts_desc_list_prf[] =
215:        {  
216:            /**************************************************************************/
217:            /* Custom Service */
218:            /**************************************************************************/
219:            {RBLE_DECL_PRIMARY_SERVICE, sizeof(custom_svc), sizeof(custom_svc),  },
220:            /* Advertising Information */
221:            { RBLE_DECL_CHARACTERISTIC, sizeof(prf_bcninfo_char), sizeof(prf_bcninfo_char),  },
222:            { DB_TYPE_128BIT_UUID, sizeof(PRF_ADV_INFO), sizeof(PRF_ADV_INFO),    },
223:            /* Advertising Data */
224:            { RBLE_DECL_CHARACTERISTIC, sizeof(prf_bcndata_char), sizeof(prf_bcndata_char),  },
225:            { DB_TYPE_128BIT_UUID, sizeof(PRF_ADV_DATA), sizeof(PRF_ADV_DATA),    },
226:            /* Scan Response Data */
227:            if RF_TX_ONLY
228:                { RBLE_DECL_CHARACTERISTIC, sizeof(prf_rspdata_char), sizeof(prf_rspdata_char),  },
229:                { DB_TYPE_128BIT_UUID, sizeof(PRF_ADV_DATA), sizeof(PRF_ADV_DATA),    },
230:            else
231:                { RBLE_DECL_CHARACTERISTIC, sizeof(prf_rspdata_char), sizeof(prf_rspdata_char),  },
232:                { DB_TYPE_128BIT_UUID, sizeof(PRF_ADV_DATA), sizeof(PRF_ADV_DATA),    },
233:            endif
234:            /* Code Flash Updated Count */
235:            { RBLE_DECL_CHARACTERISTIC, sizeof(prf_cflcnt_char), sizeof(prf_cflcnt_char),  },
236:            { DB_TYPE_128BIT_UUID, sizeof(uint16_t), sizeof(uint16_t),    },
237:            /* Data Flash Updated Count */
238:            { RBLE_DECL_CHARACTERISTIC, sizeof(prf_dflcnt_char), sizeof(prf_dflcnt_char),  },
239:            { DB_TYPE_128BIT_UUID, sizeof(uint16_t), sizeof(uint16_t),    },
240:            /* zero terminator */
241:            {0,0,0,0,0,0}
242:        };
```
(3) Processing

Processing for enabling GATT and updating Characteristic Values are implemented in r_profile.c file.

RBLE_GATT_Enable function is called to enable GATT and register GATT event callback function. When updating Characteristic Values is requested by Write Request from Client device, RBLE_GATT_EVENT_WRITE_CMD_IND event occurs. After updating the Characteristic Value, the application calls RBLE_GATT_Write_Response in order to send Write Response.

Project_Source\application\src\connect\r_profile.c, line 92-341

```c
92:    RBLE_STATUS PRF_Server_Enable(uint16_t conhdl, PRF_EVT_HANDLER callback)
93:    {
101:            result = RBLE_GATT_Enable(prf_gatt_callback);
113:    }
215:    static void prf_gatt_callback(RBLE_GATT_EVENT* evt)
216:    {
224:        switch(evt->type)
225:        {
226:            case RBLE_GATT_EVENT_WRITE_CMD_IND:
227:                /* reach here when client device requests to write characteristic */
235:                switch(att_hdl)
236:                {
237:                    /* Advertising information (18byte fixed) is written */
243:                        /* update characteristic value */
256:                        break;
257:
258:                    /* Advertising data (2byte - 32byte variable) is written */
259:                    /* Note: when requested size is over than the size of single write request, */
260:                    /* characteristic value is transferred separately per 18byte */
261:                    case PRF_HDL_BCNDATA_VAL:
268:                                    /* update characteristic value */
304:                        break;
305:
306:                        /* send the write response to client device */
307:                        if(evt->param.write_cmd.ind.resp)
308:                        {
309:                            prf_send_wr Resp(att_hdl, result);
310:                        }
311:                        break;
315:                }
316:            }
317:        }
325:    static void prf_send_wr Resp(uint16_t att_hdl, RBLE_STATUS result)
326:    {
340:        RBLE_GATT_Write_Response(&wr_resp);
341:    }
```
6.2.7 RF Operation

If change RF Operation of Beacon Stack, the application is changed. Changes of Beacon Application and Connect Application are shown in Table 6-2.

Regarding to the detail of RF Operation, refer to chapter 6.1.2 “RF Operation” in this document.

<table>
<thead>
<tr>
<th>Beacon Application</th>
<th>Scannable Undirected Advertising</th>
<th>Non-connectable Undirected Advertising</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advertising Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(r_beacon.c)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connect Application</td>
<td>Characteristics</td>
<td>Characteristics</td>
</tr>
<tr>
<td></td>
<td>Advertising Information</td>
<td>Advertising Information</td>
</tr>
<tr>
<td></td>
<td>Advertising Data</td>
<td>Advertising Data</td>
</tr>
<tr>
<td></td>
<td>Scan Response Data</td>
<td>Code Flash Memory Updated Count</td>
</tr>
<tr>
<td></td>
<td>Code Flash Memory Updated Count</td>
<td>Data Flash Memory Updated Count</td>
</tr>
<tr>
<td></td>
<td>Data Flash Memory Updated Count</td>
<td></td>
</tr>
</tbody>
</table>

Note that it is impossible to access Scan Response Data Characteristic.

iOS device stores services and characteristics constitution of connected device. Changing RF operation causes a difference between constitution of Connect Application and the stored constitution.

If change RF Operation, to clear the services and characteristics constitution stored by iOS device, disable and enable Bluetooth in iOS Settings.
7. Functions

This chapter describes major functions implemented in the Sample Program.

7.1 Function List

7.1.1 Switching Application

Table 7-1 shows the functions for switching application.

<table>
<thead>
<tr>
<th>file</th>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_main.c</td>
<td>main</td>
<td>initializes MCU and executes Applications</td>
</tr>
<tr>
<td></td>
<td>input_callback</td>
<td>calls application exit function</td>
</tr>
<tr>
<td>r_input.c</td>
<td>R_INPUT_Init</td>
<td>initializes external interrupt input</td>
</tr>
<tr>
<td></td>
<td>intp5_interrupt</td>
<td>handler of external interrupt input</td>
</tr>
<tr>
<td>r_plf.c</td>
<td>R_PLF_Init</td>
<td>initializes MCU (ports and clock)</td>
</tr>
</tbody>
</table>

7.1.2 Beacon Application

Table 7-2 shows the functions of Beacon Application.

<table>
<thead>
<tr>
<th>file</th>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_beacon_main.c</td>
<td>R_BEACON_Main</td>
<td>main loop of Beacon Application</td>
</tr>
<tr>
<td>r_beacon.c</td>
<td>R_BEACON_Start</td>
<td>starts Beacon Application</td>
</tr>
<tr>
<td></td>
<td>R_BEACON_Exit</td>
<td>exits Beacon Application</td>
</tr>
<tr>
<td></td>
<td>R_BEACON_EventHandler</td>
<td>event handler</td>
</tr>
</tbody>
</table>

7.1.3 Connect Application

Table 7-3 shows the function of Connect Application.

<table>
<thead>
<tr>
<th>file</th>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_connect_main.c</td>
<td>R_CONNECT_Main</td>
<td>main loop of Connect Application</td>
</tr>
<tr>
<td>r_connect.c</td>
<td>R_CONNECT_Start</td>
<td>starts Connect Application</td>
</tr>
<tr>
<td></td>
<td>R_CONNECT_Exit</td>
<td>exits Connect Application</td>
</tr>
<tr>
<td>con_rble_callback</td>
<td>event callback function : RBLE</td>
<td></td>
</tr>
<tr>
<td>con_gap_callback</td>
<td>event callback function : Generic Access Profile</td>
<td></td>
</tr>
<tr>
<td>con_sm_callback</td>
<td>event callback function : Security Manager</td>
<td></td>
</tr>
<tr>
<td>con_profile_callback</td>
<td>event callback function : Custom Profile</td>
<td></td>
</tr>
<tr>
<td>con_vs_callback</td>
<td>event callback function : Vendor Specific</td>
<td></td>
</tr>
<tr>
<td>con_exit_timer_task</td>
<td>exits Connect Application when no connection within 30sec</td>
<td></td>
</tr>
</tbody>
</table>

7.1.4 DTM Application

Table 7-4 shows the functions of DTM Application.

<table>
<thead>
<tr>
<th>module</th>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_dtm_main.c</td>
<td>R_DTM_Main</td>
<td>main loop of DTM Application</td>
</tr>
<tr>
<td>r_dtm.c</td>
<td>R_DTM_Start</td>
<td>starts DTM Application</td>
</tr>
<tr>
<td></td>
<td>dtm_rble_callback</td>
<td>event callback function : RBLE</td>
</tr>
<tr>
<td></td>
<td>dtm_gap_callback</td>
<td>event callback function : Generic Access Profile</td>
</tr>
<tr>
<td></td>
<td>dtm_sm_callback</td>
<td>event callback function : Security Manager</td>
</tr>
<tr>
<td></td>
<td>dtm_vs_callback</td>
<td>event callback function : Vendor Specific</td>
</tr>
</tbody>
</table>
7.2 Function Calling

7.2.1 Function Calling of Beacon Operation

Figure 7-1 shows the function calling graph of Beacon Operation. If DIP switch SW6 position-1 on the evaluation board is OFF, main function calls R_BEACON_Main function or R_CONNECT_Main function alternately.

R_BEACON_Main function starts Beacon Application by calling R_BEACON_Start function and executes main loop, then the main loop breaks by calling R_BEACON_Break function.

R_CONNECT_Main function starts Connect Application by calling R_CONNECT_Start function and executes main loop, then main loop exits by calling R_CONNECT_Break function.
7.2.2 Function Calling of RF Evaluation Operation

Figure 7-2 shows the function calling graph of RF Evaluation Operation. If DIP switch SW6 position-1 on the evaluation board is ON, main function calls R_DTM_Main function. R_DTM_Main function starts DTM Application by calling R_DTM_Start function and executes main loop. The main loop of DTM Application never break.

![Function Calling Graph of RF Evaluation Operation](image-url)
8. Operation

8.1 State Transition

There are three applications in the Sample Program: Beacon Application, Connect Application, and DTM Application. This section describes the state transition of each application.

8.1.1 Beacon Application

Figure 8-1 shows the state transition of Beacon Application.

It starts with Initializing state and then follow by Advertising state. In the Advertising state, the application executes Advertising. If receive exit request, next go to RF Powerdown state and finally exit from Beacon Application.

![Figure 8-1 State Transition of Beacon Application](image-url)
8.1.2 Connect Application

Figure 8-2 shows the state transition of Connect Application.

It starts with Initializing state and the follow by Advertising state. In the Advertising state, the application executes Advertising.

If receive connection request from peer device, go to Slave Connection state. In the Slave Connection state, the application enables GATT and checks about security status. If receive either Pairing Request or Start Encryption Request from peer connected device, the application executes pairing or start encryption respectively. And in this state, the application executes GATT Access. If disconnect from peer connected device, go back to the Advertising state.

In the Advertising state, if receive exit request or if not connect within 30seconds, the application stops Advertising and next go to RF Powerdown state and finally exit from Connect Application.

In the Slave Connection state, if receive exit request, the application requests disconnection to peer device and next go to RF Powerdown state and finally exit from Connect Application.

---

**Figure 8-2 State Transition of Connect Application**
8.1.3 DTM Application

Figure 8-3 shows the state transition of DTM Application.

It starts with Initializing state and the follow by Idling state. Then perform either RF Transmitter Test or RF Receiver Test. Respective test executes according the request from Tester and return to Idling state when complete the test.

![State Transition Diagram of DTM Application](image-url)
8.2 Sequence

8.2.1 Beacon Application

(1) Initializing & Advertising & RF Powerdown Sequence

Figure 8-4 shows the sequence in Initializing state, Advertising state and RF Powerdown state of Beacon Application. Beacon Stack API is used in sequence between Beacon Application and Beacon Stack in the Sample Program.

![Figure 8-4 Initializing & Advertising & RF Powerdown Sequence of Beacon Application]

Regarding to the specification of Beacon Stack API, refer to chapter 4 "API" in RL78/G1D Beacon Stack User's Manual (R01UW0171).
Beacon Application calls R_RF_PowerUp and R_RF_Init in order to enable RF unit. After that, the application calls R_BLE_Init in order to initialize Beacon Stack.

```c
63:    void R_BEACON_Main(void)
64:    {
72:        /*
73:        ********************************************************************************
74:        * Beacon Stack Initialization
75:        ********************************************************************************
76:        */
77:        if (RBLE_OK != R_RF_PowerUp(BCN_RF_CFG, RF_32MHZ_WAIT))
78:        {
81:            mcu_reset();
82:        }
83:
84:        if (RBLE_OK != R_RF_Init())
85:        {
88:            mcu_reset();
89:        }
90:
91:        /* Initialize Beacon Stack */
92:        interrupt_init();
93:        R_BLE_Init();
129:    }
```
The application calls R_BLE_StartAdvertising in order to start Advertising for providing information.

RBLE_EVT_ADV_TX_IND event occurs after every transmitting of advertising packets.

The application calls R_BLE_StopAdvertising in order to request stopping Advertising. After stopping Advertising RBLE_EVT_ADV_STOP_CMP event occurs.

```c
bool R_BEACON_Start(void)
{
    if(RBLE_OK != R_BLE_StartAdvertising(adv_type, &adv_info))
    {
        return false;
    }
}

void R_BEACON_Exit(void)
{
    R_BLE_StopAdvertising();
}

void R_BEACON_EventHandler(void)
{        
    RBLE_EVT* evt = R_BLE_GetEvent();
    while (evt != NULL)
    {        
        switch (evt->type)
        {            
            case RBLE_EVT_ADV_TX_IND: /* reach here after transmitting Advertising packet */
                bcn_adv_tx_eventhandler(evt);
                break;
            case RBLE_EVT_SCANREQ_RX_IND: /* reach here after receiving scan request packet */
                bcn_scanreq_rx_eventhandler(evt);
                break;
            case RBLE_EVT_ADV_STOP_CMP: /* reach here when advertising is stopped */
                bcn_adv_stop_eventhandler(evt);
                break;
            default:
                break;
        }
        evt = R_BLE_GetEvent();
    }
}
```
8.2.2 Connect Application

(1) Initializing & Advertising & Slave Connection (Configurations) Sequence

Figure 8-5 shows the sequence in Initialization state, Advertising state and Slave Connection (Configurations) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

First of all, Connect Application calls `RBLE_Init` in order to activate the rBLE_Core of BLE Protocol Stack.

The application calls `RBLE_GAP_Reset` in order to initialize GAP layer of Protocol Stack and GAP event callback function and SM event callback function.

```
Project_Source\application\src\connect\r_connect.c, line 268-490

268: bool R_CONNECT_Start(void)
269: { /* initialize rBLE */
270:     if(RBLE_OK != RBLE_Init(&con_rble_callback))
271:         return false;
272: }
273:
293:
374: static void con_rble_callback(RBLE_MODE mode)
375: { switch(mode)
376:     { case RBLE_MODE_ACTIVE:
377:         /* reach here when activating rBLE is completed after calling RBLE_Init */
378:         con_rble_active_eventhandler();
379:         break;
380:     }
381:     
392:     { case RBLE_GAP_EVENT_RESET_RESULT:
393:         /* reach here after RBLE_GAP_Reset is called */
394:         con_gap_reset_eventandler(evt);
395:         break;
396:     }
397:     
409:     { static void con_gap_callback(RBLE_GAP_EVENT* evt)
410:         { switch(evt->type)
411:             { case RBLE_GAP_EVENT_RESET_RESULT:
412:                 /* reach here after RBLE_GAP_Reset is called */
413:                 con_gap_reset_eventandler(evt);
414:                 break;
415:             case RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP:
416:                 /* reach here after RBLE_GAP_Set_Random_Address is called */
417:                 con_gap_set_random_address_eventhandler(evt);
418:                 break;
419:             case RBLE_GAP_EVENT_SET_BONDING_MODE_COMP:
420:                 /* reach here after RBLE_GAP_Set_Bonding_Mode is called */
421:                 con_gap_set_bonding_mode_eventhandler(evt);
422:                 break;
423:             case RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP:
424:                 /* reach here after RBLE_GAP_Set_Security_Request is called */
425:                 con_gap_set_security_request_eventhandler(evt);
426:                 break;
427:             case RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP:
428:                 /* reach here after RBLE_GAP_Broadcast_Enable is called */
429:                 con_gap_broadcast_enable_eventhandler(evt);
430:                 break;
431:             case RBLE_GAP_EVENT_CONNECTION_COMP:
432:                 /* reach here when connection occurred */
433:                 con_gap_connection_eventhandler(evt);
434:                 break;
435:             }
436:         }
437:     }
438: }
If Random Device Address is used, the application calls RBLE_GAP_Set_Random_Address in order to set random address to Protocol Stack.

To execute pairing sequence to the peer device, the application calls RBLE_GAP_Set_Bonding_Mode and RBLE_GAP_Set_Security_Request.

After above initializing is completed, the application calls RBLE_GAP_Broadcast_Enable to start Advertising for establishing connection as Slave.

Project_Source\application\src\connect\r_connect.c, line 498-609

498: static void con_gap_reset_eventandler(RBLE_GAP_EVENT* evt)
499: { /* reach here after RBLE_GAP_Reset is called */
500:     if(own_type == RBLE_ADDR_RAND)
501:     {
502:         /* Set Random Device Address */
503:         RBLE_GAP_Set_Random_Address(&own_addr);
504:     }
505:     else
506:     {
507:         /* Set Bonding Mode */
508:         RBLE_GAP_Set_Bonding_Mode(RBLE_GAP_BONDABLE);
509:     }
510: }

566: static void con_gap_set_random_address_eventhandler(RBLE_GAP_EVENT* evt)
567: { /* reach here after RBLE_GAP_Set_Random_Address is called */
568:     /* Set Bonding Mode */
569:     RBLE_GAP_Set_Bonding_Mode(RBLE_GAP_BONDABLE);
570: }

583: static void con_gap_set_bonding_mode_eventhandler(RBLE_GAP_EVENT* evt)
584: { /* reach here after RBLE_GAP_Set_Bonding_Mode is called */
585:     /* Set Security Request */
586:     RBLE_GAP_Set_Security_Request(RBLE_GAP_SEC1_NOAUTH_PAIR_ENC);
587: }

600: static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
601: { /* reach here after RBLE_GAP_Set_Security_Request is called */
602:     /* Start Broadcast for the First Connection */
603:     RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE, … );
604: }

633: static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
634: { /* reach here after RBLE_GAP_Set_Security_Request is called */
635:     /* Start Broadcast for the First Connection */
636:     RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE, … );
637: }

660: static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
661: { /* reach here after RBLE_GAP_Set_Security_Request is called */
662:     /* Start Broadcast for the First Connection */
663:     RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE, … );
664: }

717: static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
718: { /* reach here after RBLE_GAP_Set_Security_Request is called */
719:     /* Start Broadcast for the First Connection */
720:     RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE, … );
721: }

782: static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
783: { /* reach here after RBLE_GAP_Set_Security_Request is called */
784:     /* Start Broadcast for the First Connection */
785:     RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE, … );
786: }

857: static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
858: { /* reach here after RBLE_GAP_Set_Security_Request is called */
859:     /* Start Broadcast for the First Connection */
860:     RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE, … );
861: }

927: static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
928: { /* reach here after RBLE_GAP_Set_Security_Request is called */
929:     /* Start Broadcast for the First Connection */
930:     RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE, … );
931: }
After establishing connection, RBLE_GAP_EVENT_CONNECTION_COMP event occurs.

If peer’s Device Address Type is not Resolvable Private Address, RBLE_SM_CHK_BD_ADDR_REQ event occurs. The application should call RBLE_SM_Chk_Bd_Addr_Req to respond security status in previous connection with peer device. In order to execute Pairing again if peer device forgets Pairing information, the application always respond that there is no security status.

If peer’s Device Address Type is Resolvable Private Address, RBLE_SM_IRK_REQ_IND occurs. The application should call RBLE_SM_Irk_Req_Resp to respond security status and Identity Resolving Key (IRK) for resolving address. But in order to execute Pairing again if peer device forgets Pairing information, the application always respond that there are no security status and no IRK.

```
815: static void con_sm_callback(RBLE_SM_EVENT* evt)
816: {
824:     switch(evt->type)
825:     {
826:         case RBLE_SM_CHK_BD_ADDR_REQ:
827:             /* reach here when connection is established to peer device that address is 
828:                public address or random address except resolvable private address */
829:                 con_sm_bdaddr_check_request_eventhandler(evt);
830:                 break;
831:         case RBLE_SM_IRK_REQ_IND:
832:             /* reach here when connection is established to peer device that address is 
833:                resolvable private address */
834:             /* IRK is requested for resolving peer’s resolvable private address */
835:                 con_sm_irk_request_eventhandler(evt);
836:                 break;
837:         }
859:     }
867:     static void con_sm_bdaddr_check_request_eventhandler(RBLE_SM_EVENT* evt)
868:     {
869:         /* reach here when connection is established to peer device that address is 
870:            public address or random address except resolvable private address */
871:         RBLE_SM_Chk_Bd_Addr_Req_Resp(evt->param.chk_bdaddr.idx, 
872:                                         0, 
873:                                         false, 
874:                                         RBLE_SMP_SEC_NONE, 
875:                                         NULL);
882:     }
890:     static void con_sm_irk_request_eventhandler(RBLE_SM_EVENT* evt)
891:     {
892:         /* reach here when connection is established to peer device that address is 
893:            resolvable private address */
894:         /* IRK is requested for resolving peer’s resolvable private address */
895:         RBLE_SM_Irk_Req_Resp(evt->param.irk_req.idx, 
896:                               RBLE_ERR, 
897:                             &con_env.con_addr, 
898:                             NULL, 
899:                             RBLE_SMP_SEC_NONE);
906:     }
```
Slave Connection (Pairing) Sequence

Figure 8-6 shows the sequence in Slave Connection (Pairing) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

When pairing request is sent after establishing connection to the unpaired device, RBLE_GAP_BONDING_REQ_IND event occurs. The application calls RBLE_GAP_Bonding_Response in order to respond the pairing features. Pairing method is determined by exchanging pairing feature. In the Sample Program, Just Works is executed as pairing method. In the case of Just Works, it is not necessary to set Temporary Key to Protocol Stack. Following pairing process is encrypted with Short Term Key, which is generated by BLE Protocol Stack.

```
414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
415:    {
440:        switch(evt->type)
441:        {
477:            case RBLE_GAP_EVENT_BONDING_REQ_IND:
478:                /* reach here when bonding is requested */
479:                /* in the middle of PHASE1: PAIRING FEATURE EXCHANGE in pairing sequence */
480:                con_gap_bonding_request_eventhandler(evt);
481:                break;
482:            case RBLE_GAP_EVENT_BONDING_COMP:
483:                /* reach here bonding is completed */
484:                /* at the end of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
485:                con_gap_bonding_eventhandler(evt);
486:                break;
489:        }
490:    }
776:    static void con_gap_bonding_request_eventhandler(RBLE_GAP_EVENT* evt)
777:    {
778:        /* reach here when bonding is requested */
779:        /* in the middle of PHASE1: PAIRING FEATURE EXCHANGE in pairing sequence */
780:        /* Reply Bonding Response */
783:        RBLE_GAP_Bonding_Response(&bond_info);
784:    }
```
After starting encryption with Short Term Key, RBLE_SM_LTK_REQ_IND event occurs. The application generates Long Term Key (LTK) and calls RBLE_SM_Req_Resp in order to respond LTK. LTK is used as encryption key for encrypting following connection.

After providing LTK to Master device, RBLE_SM_KEY_IND event occurs and encryption key is provided from Master device, which is specified by the bonding response.

When pairing sequence is completed, RBLE_GAP_EVENT_BONDING_COMP event occurs.

```
815: static void con_sm_callback(RBLE_SM_EVENT* evt)
816: {
824:     switch(evt->type)
825:     {
837:         case RBLE_SM_LTK_REQ_IND:
838:             /* reach here when LTK is requested */
839:             /* in the first of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
840:             con_sm_ltk_request_eventhandler(evt);
841:             break;
842:         case RBLE_SM_KEY_IND:
843:             /* reach here when peer device's encryption information are provided */
844:             /* in the middle of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
845:             con_sm_key_eventhandler(evt);
846:             break;
858:     }
859: }
914: static void con_sm_ltk_request_eventhandler(RBLE_SM_EVENT* evt)
915: {
916:     /* reach here when LTK is requested */
917:     /* in the first of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
918:     /
933:         /* Reply LTK(Long Term Key) */
934:         RBLE_SM_Ltk_Req_Resp(evt->param.ltk_req.idx,
935:             RBLE_OK,
936:             RBLE_SMP_KSEC_NONE,
937:             pair_info.enc_key.ediv,
938:             &pair_info.enc_key.nb ,
939:             &pair_info.enc_key.ltk );
940: }```
(3) Slave Connection (Start Encryption) Sequence

Figure 8-7 shows the sequence in Slave Connection (Start Encryption) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

```
Peer Device

<table>
<thead>
<tr>
<th>Connect Application</th>
<th>Protocol Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt [MASTER has pairing information]</td>
<td></td>
</tr>
<tr>
<td>RBLE_SM_LTK_REQ_FOR_ENC_IND</td>
<td>LL_ENC_REQ</td>
</tr>
<tr>
<td>RBLE_SM_Ltk_Req_Resp</td>
<td>LL_ENC_RSP</td>
</tr>
<tr>
<td>LL_ENC_RSP</td>
<td>LL_START_ENC_REQ</td>
</tr>
<tr>
<td>LL_START_ENC_RSP</td>
<td>LL_START_ENC_RSP</td>
</tr>
</tbody>
</table>

Renesas BLE Microcontroller - RL78/G1D

RL_ENC_RSP

Protocol Stack

SLAVE_CONNECTION

MASTER CONNECTION
```

Figure 8-7 Slave Connection (Start Encryption) Sequence of Connect Application

When the encryption request is sent after establishing connection to the paired device, RBLE_SM_LTK_REQ_FOR_ENC_IND event occurs. The application calls RBLE_SM_Ltk_Req_Resp in order to respond Long Term Key (LTK). LTK is generated and exchanged already in before connection, and used as encryption key for encrypting this connection.

When start encryption sequence is completed, RBLE_SM_ENC_START_IND event occurs.

```
815:    static void con_sm_callback(RBLE_SM_EVENT* evt)
816:    {
824:        switch(evt->type)
825:        {
847:            case RBLE_SM_LTK_REQ_FOR_ENC_IND:
848:                /* reach here when LTK is requested */
849:                /* in the first of start encryption sequence */
850:                con_sm_ltk_request_for_enc_eventhandler(evt);
851:                break;
852:            case RBLE_SM_ENC_START_IND:
853:                /* reach here when start encryption sequence is executed */
854:                con_sm_enc_start_eventhandler(evt);
855:                break;
858:        }
859:    }
```

```
860:    static void con_sm_ltk_request_for_enc_eventhandler(RBLE_SM_EVENT* evt)
861:    {
862:        /* reach here when LTK is requested */
863:        /* in the first of Start Encryption sequence */
864:        /* Reply LTK(Long Term Key) */
878:            RBLE_SM_Ltk_Req_Resp(evt->param.ltk_req_for_enc.idx,
879:                              status,
880:                              pair_info.sec_prop,
881:                              pair_info.enc_key.ediv,
882:                              $pair_info.enc_key.nb ,
883:                              $pair_info.enc_key.ltk );
884:    }
```
(4) **Slave Connection (GATT Access) Sequence**

Figure 8-8 shows the sequence in Slave Connection (GATT Access) state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

When Service Discovery, Characteristic Discovery and Characteristic Descriptor Discovery are requested from Client device, response are executed by Protocol Stack automatically.

Similarly, when Read Request is sent from Client device, Read Response is sent by Protocol Stack automatically.

When updating characteristic values is requested by Write Request from Client device, \texttt{RBLE\_GATT\_EVENT\_WRITE\_CMD\_IND} event occurs. After updating characteristic value, the application calls \texttt{RBLE\_GATT\_Write\_Response} in order to send Write Response.

```
215: static void prf_gatt_callback(RBLE_GATT_EVENT* evt)
216: {  
217:     switch(evt->type)
218:     {  
219:         case RBLE_GATT_EVENT_WRITE_CMD_IND:
220:             /* reach here when client device requests to write characteristic */
221:             switch(att_hdl)
222:             {  
223:                 /* Advertising information (18byte fixed) is written */
224:                     case PRF_HDL_BCNINFO_VAL:
225:                         /* update characteristic value */
226:                         break;
227:                 /* Advertising data (2byte - 32byte variable) is written */
228:                     case PRF_HDL_BCNDATA_VAL:
229:                         /* update characteristic value */
230:                         break;
231:                     /* Scan Response data (2byte - 32byte variable) is written */
232:                     case PRF_HDL_RSPDATA_VAL:
233:                         /* update characteristic value */
234:                         break;
235: #if !RF_TX_ONLY
236:                     case PRF_HDL_RSPDATA_VAL:
237:                         /* update characteristic value */
238:                         break;
239: #endif
240:             }
241:             /* send the write response to client device */
242:             if(evt->param.write_cmd_ind.resp)
243:             {  
244:                 prf_send_wr_resp(att_hdl, result);
245:             }
246:             break;
247:         }
248:     }
249: }
250:
251: static void prf_send_wr_resp(uint16_t att_hdl, RBLE_STATUS result)
252: {
253:     RBLE_GATT_Write_Response(&wr_resp);
254: }
```
(5) Disconnection & RF Powerdown Sequence

Figure 8-9 shows the sequence in Disconnection state and RF Powerdown state of Connect Application. rBLE API is used in sequence between Connect Application and BLE Protocol Stack in the Sample Program.

Figure 8-9 Disconnection & RF Powerdown Sequence of Connect Application

When exiting the application is requested in Advertising, the application calls RBLE_GAP_Broadcast_Disable in order to stop Advertising.

When exiting the application is requested in establishing connection, the application calls RBLE_GAP_Disconnect in order to disconnect.

When RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP event or RBLE_GAP_EVENT_DISCONNECT_COMP event occurs, the application calls RBLE_VS_Enable and RBLE_VS_RF_Control in order to power down the RF unit.

Project Source\application\src\connect\r_connect.c, line 414-1085

```c
414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
415:    {
440:        switch(evt->type)
441:        {
462:            case RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP:
463:                /* reach here after RBLE_GAP_Broadcast_Disable is called */
464:                con_gap_broadcast_disable_eventhandler(evt);
465:                break;
470:            case RBLE_GAP_EVENT_DISCONNECT_COMP:
471:                /* reach here when disconnection occurred */
472:                con_gap_disconnection_eventhandler(evt);
473:                break;
489:        }
490:    }

641:    static void con_gap_broadcast_disable_eventhandler(RBLE_GAP_EVENT* evt)
642:    {
643:        /* reach here after RBLE_GAP_Broadcast_Disable is called */
644:        RBLE_VS_Enable(con_vs_callback);
645:        RBLE_VS_RF_Control(RBLE_VS_RFCNTL_CMD_POWDOWN);
650:    }

685:    static void con_gap_disconnection_eventhandler(RBLE_GAP_EVENT* evt)
686:    {
687:        /* reach here when disconnection occurred */
688:        RBLE_VS_Enable(con_vs_callback);
689:        RBLE_VS_RF_Control(RBLE_VS_RFCNTL_CMD_POWDOWN);
768:    }

1053:    static void con_vs_callback(RBLE_VS_EVENT* evt)
1054:    {
1076:        switch(evt->type)
1077:        {
1078:            case RBLE_VS_EVENT_RF_CONTROL_COMP:
1079:                /* reach here after RBLE_VS_RF_Control is called */
1080:                con_vs_rf_control_eventhandler(evt);
1081:                break;
1084:        }
1085:    }
```
8.2.3 DTM Application

(1) Initializing & Transmitter Test & Receiver Test Sequence

Figure 8-10 shows the sequence in Initializing state, Transmitter Test state and Receiver Test state of DTM Application. rBLE API is used in sequence between DTM Application and BLE Protocol Stack in the Sample Program.

DTM Application executes RF Test by receiving RF test command through UART. The application reports the result by transmitting RF test event through UART.

When LE_RECEIVER_TEST command is received, the application calls RBLE_VS_Test_Rx_Start in order to start receiving RF test packets from RF Tester.

In the same way, when LE_TRANSMITTER_TEST command is received, the application calls RBLE_VS_Test_Tx_Start in order to start transmitting RF test packets to RF Tester.

When LE_TEST_END command is received, the application calls RBLE_VS_Test_End in order to stop RF test.

```
Project_Source\application\src\connect\r_dtm.c, line 193-384

193: static void dtm_cmdhandler(uint16_t cmd)
194: {
201:     switch(val16 & DTM_CMD_MASK)
202:     {
203:         case DTM_CMD_RESET:
204:             if(RBLE_OK != RBLE_GAP_Reset(&dtm_gap_callback, &dtm_sm_callback))
205:                 dtm_send_error();
206:             break;
207:         case DTM_CMD_RX_START:
208:             if(RBLE_OK != RBLE_VS_Test_Rx_Start(DTM_GET_FREQ(val16)))
209:                 dtm_send_error();
210:             break;
211:         case DTM_CMD_TX_START:
212:             if(RBLE_OK != RBLE_VS_Test_Tx_Start(DTM_GET_FREQ(val16), DTM_GET_LENGTH(val16), ... )
213:                 dtm_send_error();
214:             break;
215:         case DTM_CMD_TX_END:
216:             if(RBLE_OK != RBLE_VS_Test_End())
217:                 dtm_send_error();
218:             break;
219:         } case DTM_CMD_END:
220:             if(RBLE_OK != RBLE_VS_Test_End())
221:                 dtm_send_error();
222:             break;
223:         }
224:     }
225:     switch(evt->type)
226:     {
229:         case RBLE_VS_EVENT_RX_START_COMP:
230:             /* reach here when RBLE_VS_Test_Rx_Start is called */
232:                 break;
233:         case RBLE_VS_EVENT_TX_START_COMP:
236:             /* reach here when RBLE_VS_Test_Tx_Start is called */
237:                 break;
238:         case RBLE_VS_EVENT_TX_END_COMP:
241:             /* reach here when RBLE_VS_Test_Tx_End is called */
242:                 break;
243:         case RBLE_VS_EVENT_RX_END_COMP:
246:             /* reach here when RBLE_VS_Test_Rx_End is called */
247:                 break;
248:         }```

9. Appendix

9.1 Device Address

Device Address is 48-bit value for identifying each device. Device Address Types defined by Bluetooth Core Specification are shown as below.

- Public Device Address
  - Public device address shall be created in accordance with section "48-bit universal LAN MAC addresses" of the IEEE 802-2001 standard and using a valid Organizationally Unique Identifier (OUI) obtained from the IEEE Registration Authority.

- Random Device Address
  - Static Device Address
    - Static Device Address is a 48-bit randomly generated. Device may choose to initialize its address to a new value after each power cycle. And device shall not change its address value once initialized until the device is power cycled.

- Private Device Address
  - Non-resolvable Private Address
    - Non-resolvable Private Address is a 48-bit randomly generated. Its address should be changed over a period of time (recommended value of Bluetooth Core Specification is 15mins) for reducing the ability to track by other device.
  - Resolvable Private Address
    - Resolvable Private Address contains 24-bit randomly generated number and 24-bit hash generated with randomly generated number and Identity Resolving Key (IRK). Its address should be changed over a period of time (recommended value of Bluetooth Core Specification is 15mins) for reducing the ability to track by other device.

Figure 9-1 Public Device Address format

Figure 9-2 Static Device Address format

Figure 9-3 Non-resolvable Private Address format

Figure 9-4 Resolvable Private Address format

Regarding to the specification of Device Address, refer to [Vol. 6, Part B] Section 1.3, Bluetooth Core Specification v4.2.
### 9.2 Advertising Packet Format

Beacon Application transmits non-connectable undirected advertising packet, and Connect Application transmits connectable undirected advertising packet in non-connected state. The packet format is common, and it is shown in Figure 9-5.

<table>
<thead>
<tr>
<th>Advertising channel Packet</th>
<th>Preamble Access Address</th>
<th>PDU (Protocol Data Unit)</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 byte)</td>
<td>(4 byte)</td>
<td>(max 37 byte)</td>
<td></td>
</tr>
<tr>
<td>Advertising channel PDU</td>
<td>Header Payload</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6 byte)</td>
<td>(max 31 byte)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advertising channel PDU Payload</td>
<td>AdvA AdvData</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(max 31 byte)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advertising Data AD Structure 1 AD Structure 2 ⋯ AD Structure N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 byte)</td>
<td>(Length byte)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD Structure</td>
<td>Length Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n byte)</td>
<td>(Length-n byte)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AD Type</td>
<td>AD Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9-5 Advertising packet format**

The specification of advertising packet is as shown below.

- advertising channel packet
  - Preamble : fixed 10101010b
  - Access Address : fixed 0x8E89BED6
  - Advertising channel PDU : Header and Payload
  - CRC : 24bits

Below fields are set by application.

- advertising channel PDU Payload
  - AdvA : Advertiser's Address is placed in
  - AdvData (Advertising data) : multiple AD structures are placed in, and maximum size is 31 bytes
    - AD structure : 1byte part of Length information and Length bytes part of Data
    - Data : n bytes part of AD Type and (Length-n) bytes part of AD Data

Regarding to the details, refer to below specifications respectively.

- advertising packet format : [Vol. 6, Part B] Section 2.1, Bluetooth Core Specification v4.2
- advertising channel PDU format : [Vol. 6, Part B] Section 2.3, Bluetooth Core Specification v4.2
- advertising data format : [Vol. 3, Part C] Section 11, Bluetooth Core Specification v4.2
- AD Type : Part A, Supplement to the Bluetooth Core Specification v7.

Regarding to the definitions of AD Type, refer to below website.

- Bluetooth SIG Home > Specification > Assigned Numbers > Generic Access Profile
  [https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile](https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile)
9.3 Attribute Packet Format

Connect Application transmits Attribute packet in connected state. Attribute packet format which is used in Characteristic Value Read and Characteristic Value Write is shown in Figure 9-6.

<table>
<thead>
<tr>
<th>Data channel Packet Preamble</th>
<th>Access Address</th>
<th>Protocol Data Unit</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2 byte)</td>
<td>(2 byte)</td>
<td>(max 33 byte)</td>
<td>(3 byte)</td>
</tr>
</tbody>
</table>

- **Preamble**: either 10101010b or 01010101b.
- **Access Address**: determined by Master device when establishing connection.
- **Data channel PDU**: consists of Header, Payload and Message Integrity Check (MIC). MIC is added if data is encrypted.
- **CRC**: 24bits

---

<table>
<thead>
<tr>
<th>Attribute Protocol PDU</th>
<th>Opcode</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 byte)</td>
<td>(2 byte)</td>
<td>(max 22 byte)</td>
</tr>
</tbody>
</table>

**Attribute Protocol - Write Request**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Attribute Handle</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x0A)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Read Request**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Attribute Handle</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x0B)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Read Blob Request**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Attribute Handle</th>
<th>Value Offset</th>
<th>Part Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x0C)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Read Blob Response**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Attribute Handle</th>
<th>Part Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x0D)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Write Response**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Attribute Handle</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x12)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Prepare Write Request**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Attribute Handle</th>
<th>Value Offset</th>
<th>Part Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x16)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Prepare Write Response**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Attribute Handle</th>
<th>Value Offset</th>
<th>Part Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x17)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Execute Write Request**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x18)</td>
<td></td>
</tr>
</tbody>
</table>

**Attribute Protocol - Execute Write Response**

<table>
<thead>
<tr>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x19)</td>
</tr>
</tbody>
</table>

The specification of Attribute packet is as shown below.

- Data channel packet
  - Preamble: either 10101010b or 01010101b.
  - Access Address: determined by Master device when establishing connection.
  - Data channel PDU: consists of Header, Payload and Message Integrity Check (MIC).
  - CRC: 24bits

---

**Figure 9-6 Attribute packet format**
- L2CAP PDU
  - consists of Length, Channel ID (CID) and Information payload.
  - CID is 0x0004 (Attribute Protocol)
- Attribute Protocol PDU
  - consists of Attribute Opcode and Attribute Parameters
  - Attribute Opcode specifies Attribute operation
  - Attribute Parameters is different from each Attribute operation

Regarding to the details, refer to below specification
- Data channel PDU: [Vol. 6, Part B] Section 2.4, Bluetooth Core Specification v4.2
- Attribute Protocol PDU: [Vol. 3, Part F] Section 3.4, Bluetooth Core Specification v4.2
- GATT Features: [Vol. 3, Part G] Chapter 4, Bluetooth Core Specification v4.2
9.4 Specification Changes

Table 9-1 shows major specification changes.

<table>
<thead>
<tr>
<th>Item</th>
<th>Rev.1.00</th>
<th>Rev1.10</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Version</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC-RL</td>
<td>V1.02.00</td>
<td>V1.04.00</td>
<td>Chapter2</td>
</tr>
<tr>
<td>CS+ for CC</td>
<td>V3.03.00</td>
<td>V5.00.00</td>
<td>Chapter2</td>
</tr>
<tr>
<td>e² studio</td>
<td>Version 4.3.0.008</td>
<td>Version 5.2.0.023</td>
<td>Chapter2</td>
</tr>
<tr>
<td>BLE Protocol Stack</td>
<td>V1.11</td>
<td>V1.20</td>
<td>Chapter2</td>
</tr>
<tr>
<td>Beacon Stack</td>
<td>V1.00</td>
<td>V2.10</td>
<td>Chapter2</td>
</tr>
<tr>
<td>RF Operation</td>
<td>RF Transmission only</td>
<td>RF Transmission only</td>
<td>Subsection 6.1.2</td>
</tr>
<tr>
<td>Advertising Type</td>
<td>Non-connectable Undirected Advertising</td>
<td>Non-connectable Undirected Advertising</td>
<td>Subsection 5.1.1</td>
</tr>
<tr>
<td>Connect Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pairing Initiator Key Distribution</td>
<td>ID Resolving key</td>
<td>None</td>
<td>Subsection 5.2.2</td>
</tr>
<tr>
<td>Custom Service</td>
<td>Advertising Information</td>
<td>Advertising Information</td>
<td>Subsection 5.2.3</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Advertising Data</td>
<td>Advertising Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code Flash Memory Updated Count</td>
<td>Data Flash Memory Updated Count</td>
<td></td>
</tr>
<tr>
<td>Application Switching</td>
<td>Beacon Application starts, and then either Beacon Application and Connect Application switches alternately when switch is pushed.</td>
<td>Beacon Application starts, and then either Beacon Application and Connect Application switches alternately when switch is pushed. Connect Application starts, and then switch to Beacon Application after 30 seconds.</td>
<td>Subsection 6.1.10</td>
</tr>
<tr>
<td>Stored Data in Flash memory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Flash</td>
<td>Device Address</td>
<td>Device Address</td>
<td>Subsection 5.4.1</td>
</tr>
<tr>
<td></td>
<td>Device Address Type</td>
<td>Device Address Type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Device Name</td>
<td>Device Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Advertising Information</td>
<td>Advertising Information</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non-connectable Undirected Advertising</td>
<td>Non-connectable Undirected Advertising</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Advertising Data</td>
<td>Advertising Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scan Response data</td>
<td>Code Flash Memory Updated Count</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code Flash Updated Count</td>
<td>Data Flash Memory Updated Count</td>
<td></td>
</tr>
<tr>
<td>Data Flash</td>
<td>Pairing Information</td>
<td>Pairing Information</td>
<td>Subsection 5.4.2</td>
</tr>
<tr>
<td></td>
<td>Peer Device Address</td>
<td>Peer Device Address</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Peer Device Address Type</td>
<td>Peer Device Address Type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security Status</td>
<td>Security Status</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Local Encryption Keys</td>
<td>Local Encryption Keys</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remote Encryption Keys</td>
<td>Remote Encryption Keys</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random Seed Value</td>
<td>Random Seed Value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code Flash Updated Count</td>
<td>Code Flash Updated Count</td>
<td></td>
</tr>
</tbody>
</table>
Website and Support

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

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

All trademarks and registered trademarks are the property of their respective owners.
## Revision History of Preceding Editions

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Jul. 14, 2016</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
| 1.10 | Jun. 30, 2017 | Chapter 1 Overview             
  - P.5 figure of Beacon System and figure of RF Evaluation System are merged into Figure 1-1  
  - P.6 Figure 1-2 and Figure 1-3 are changed  
  - content about the evaluation board operation is moved to Chapter 4 |
<p>|      |             | Chapter 2 Environment                                                                 |
|      |             | P.7 Software Environment and Software Library versions are updated             |
|      |             | Chapter 3 File Composition                                                                 |
|      |             | P.8 R5F11AGJ_BcnCmb_no_sw.hex is added in file composition                      |
|      |             | Chapter 4 Evaluation Procedure                                                                 |
|      |             | P.10 Current Consumption Measurement is added in evaluation procedure          |
|      |             | P.11 Library download URLs are changed                                          |
|      |             | P.13 content about the evaluation board operation and Figure 4-1 are added     |
|      |             | P.15 content about the evaluation board operation and Figure 4-2 are added     |
|      |             | P.15 flow chart using smart phone is merged into flow chart of P.10         |
|      |             | P.16 figure of Confirming the transmission of advertising packet is added     |
|      |             | P.17 figure of Updating the advertising packet is added                       |
|      |             | P.17 procedure is changed for evaluating with iOS GATTBrowser                |
|      |             | P.19 procedure is changed for evaluating with Android GATTBrowser            |
|      |             | P.21 figure of Confirming the updated advertising packet is added             |
|      |             | P.22 figure of Evaluating RF characteristic is added                         |
|      |             | P.23 Section 4.6 is added                                                   |
|      |             | Chapter 5 Specification                                                                 |
|      |             | P.25 Operation when both Tx and Rx are enabled is added                       |
|      |             | P.26 Table 5-2 is added                                                      |
|      |             | P.28 Initiator Key Distribution is changed to None in Table 5-4               |
|      |             | P.30 Scan Response Data Characteristic is added in Table 5-5                 |
|      |             | P.30 Attribute Handle are changed in Table 5-5                               |
|      |             | P.31 RF Test Commands /Events Table is separated into Table 5-6 and Table 5-7 |
|      |             | P.31 Combinations of RF Test Commands / Events Table is removed              |
|      |             | P.32 format of Table 5-8 is changed                                           |
|      |             | P.32 Scannable Advertising Data and Scan Response Data are added in Table 5-8 |
|      |             | P.33 Structure of Pairing Information and Flash memory count are changed in Table 5-9 |
|      |             | P.33 Random Seed is removed in Table 5-9                                      |
|      |             | P.35 Section 5.6 is added                                                    |
|      |             | P.36 Compiler version is updated                                              |
|      |             | P.37 size of System Configuration is changed in Figure 5-8, Figure 5-9, and Figure 5-10 |
|      |             | Chapter 6 Configuration                                                                 |
|      |             | P.44 Subsection 6.1.10 is added                                               |
|      |             | P.46 Scan Response Data is added in unique code file                          |
|      |             | P.47 Subsection 6.2.2 is added                                                |
|      |             | P.51 Subsection 6.2.4 is added                                                |
|      |             | P.58 Subsection 6.2.7 is added                                                |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Chapter</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.03.30</td>
<td>5</td>
<td>Specification</td>
<td>p.34 Supplementation of Table 5-12 is changed.</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Appendix</td>
<td>p.84 Description in Section 9.2 is modified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p.85 Description in Section 9.3 is modified.</td>
</tr>
</tbody>
</table>

Chapter 9 Appendix
- content about building Android application is removed
- Section 9.4 is added

Overall
- "Beacon System" is changed into "Beacon Operation"
- "RF Evaluation System" is changed into "RF Evaluation Operation"
- Related documents to be referred are added
- typos of term, symbol name, and other are modified
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.
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 limb 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 applicable environmental and health regulations. Renesas Electronics disclaims any and all liability for any losses and damages 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 reproduced, reprinted 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.

(Rev.4.0-1 November 2017)