RE01B Group

Bluetooth Low Energy sample code for batteryless beacon

Introduction

This application note describes the sample code of the Bluetooth® Low Energy Advertising packet transmission application (hereinafter referred to as the Beacon application) for batteryless environments using energy harvesting in the RE01B group.

Target Device

RE01B Group

Related Documents

- Bluetooth Core Specification (https://www.bluetooth.com)
- RE01B Group Product with 1.5-Mbyte Flash Memory User's Manual: Hardware (R01UH0903)
- Getting Started Guide to Development Using CMSIS Package (R01AN5310)
- e^2 studio Getting Started Guide (R20UT4204)
- RE01B Group Tuning procedure of Bluetooth dedicated clock frequency (R01AN5488)

Contents

1. Overview

This application note provides the sample code as the example to implement Beacon application. The Beacon application can be created using Beacon stack. Refer to “2 Sample code” as for the sample code, “1.1 Beacon stack” and “3 Beacon stack detail” as for Beacon stack, and “4 Create project” or later as for how to create Beacon application. The package (r01an5674xxYYYY-re01b-bcn.zip) attached to this application note consists of the directories and files shown in Table 1.1.

Note: The Beacon stack of RE01B group cannot be used together with the BLE protocol stack provided separately from this application note.

Table 1.1 Package contents

Directory / File	Description
ROM_Files\	Sample code’s binary files.
bcn_project_battless.zip	Beacon operation sample code for the EB-RE01B board. As for detail, refer to “2.7 Beacon sample code”.
r01an5674ejYYYY-re01b-bcn.pdf r01an5674jjYYYY-re01b-bcn.pdf	This document (English / Japanese). YYYY is the revision number.

1.1 Beacon stack

The Beacon stack is the driver and the middleware implemented in a project and providing the API in the form of a library.

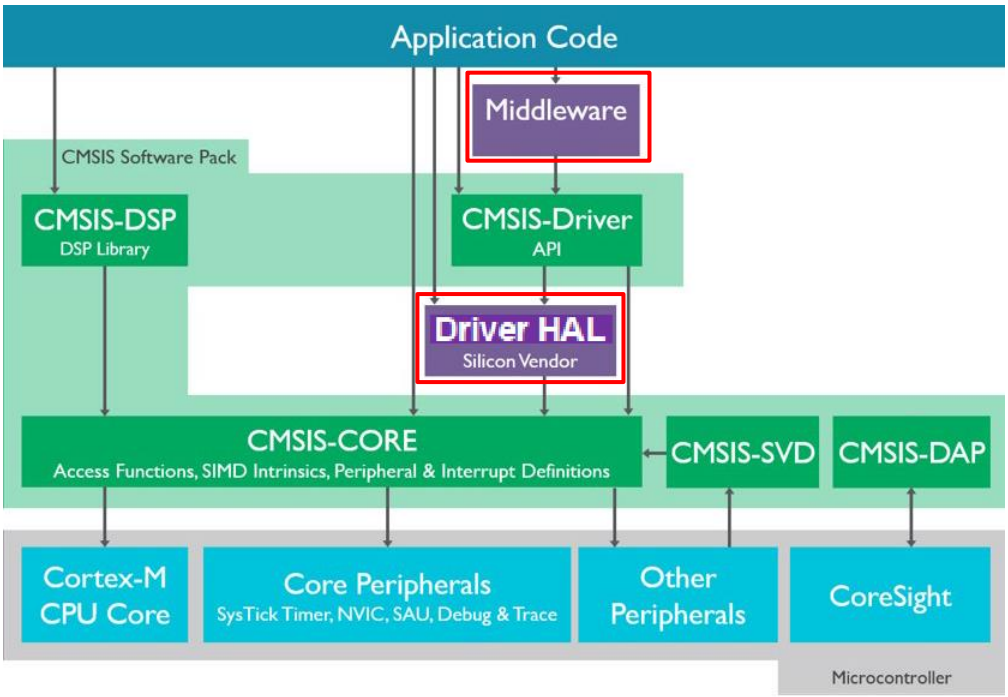

Figure 1.1 Positioning of Beacon stack

Beacon stack performs beacon operation for batteryless environment using Non-Connectable / Non-Scannable Advertising (ADV_NONCONN_IND) specified by Bluetooth SIG.

The Beacon stack can perform the following extended operations from the Advertising operation specifications specified by the Bluetooth SIG so that it can send Advertising packets with less power consumption

- It is possible to specify less than 20ms in the Advertising interval setting.
- Disables the random delay of 0ms to 10ms that is automatically added to the Advertising interval..

Beacon applications should only be used in batteryless environments, as the above extended behavior can result in more collisions with Advertising packets from nearby devices.

1.2 Software Structure

Figure 1.2 shows the software structure of this sample code.

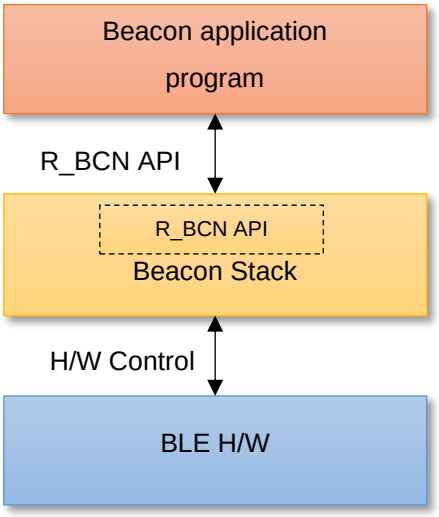

Figure 1.2 Software structure

Beacon applications can control the BLE function by calling the R_BCN API function provided by the Beacon stack. Beacon stacks only send Advertising packets, and Beacon applications cannot have a GATT profile.

1.3 Directory / File structure

Table 1.2 shows the directory / file structure of this sample code.

Table 1.2 Directory / File structure

Directory /File structure				Description
CMSIS\				CMSIS
config\	RE01B_1500KB_bcn.icf			Linker configuration file for IAR
Device\	BCN\	platform\		Driver for peripheral function
		lib\		Beacon stack Library (GCC ARM/EWARM)
	CMSIS_Driver\			CMSIS driver
	Config\	r_bcn_cfg.h		Configuration option file
	Driver\	Include\	r_bcn_api.h	API interface file All API calls and their supporting interface definition
qe_gen\	ble\	app_main.c		Beacon application main code
script\	RE01B_1500KB_bcn.ld			Linker script file for GCC
src\	main.c			Project main code
SVD\	RE01B_1500KB.svd			Register information of peripheral function
.cproject .project ble_project_xxx HardwareDebug.launch				Project file for GCC
bcn_project_xxx.ewd bcn_project_xxx.ewp bcn_project_xxx.eww				Project file for IAR

2. Sample code

2.1 Operating environment

Table 2.1 shows the hardware requirements for using this sample code.

Table 2.1 Hardware requirements

Hardware	Description
Host PC	Windows® 10 PC with USB interface.
MCU Board	EB-RE01B board
On-chip debugging emulators	GCC environment: Either of the following emulator. E2 emulator [RTE0T00020KCE00000R] E2 emulator Lite [RTE0T0002LKCE00000R] J-Link IAR environment: Either of the following emulator. I-jet® J-Link
USB cables	Used to connect to the MCU board. 1 USB A-microB cable

Table 2.2 shows the software requirements for using this sample code.

Table 2.2 Software requirements

Software		Version	Description
GCC environment	e² studio	v21.1.0 or later	Integrated development environment (IDE) for Renesas devices.
	GCC ARM Embedded	v6.3.1 or later	C/C++ Compiler. (download from e² studio installer)
	CMSIS Driver Package	v1.0.0 or later	Software package for developing applications for the RE microcontroller series.
IAR environment	IAR Embedded Workbench for ARM (EWARM)	v8.50.5 or later	Integrated development environment (IDE) for ARM devices made by IAR Systems.
	IAR C/C++ Compiler for ARM	v8.50.5 or later	C/C++ Compiler made by IAR Systems.
	CMSIS Driver Package	v1.0.0 or later	Software package for developing applications for the RE microcontroller series.
Renesas Flash Programmer		v3.06.01 or later	Tool for programming the on-chip flash memory of Renesas microcontrollers.
Integer types			It uses ISO C99 “Exact width integer types”. These types are defined in stdint.h.
Endian			Little endian

2.2 Startup procedure

The EB-RE01B board supports USB power supply. Confirm that the board switches are as Table 2.3 and connect USB connector CN4 to the USB port of your PC (or other power supply) with a USB cable.

Table 2.3 Board switch settings

Board switch	Setting
JP1	Short (ON)
JP2	Short (ON)
JP3	Short (ON)
JP4 / JP5	2-3 (Regulator Out) / 1-2 (USB)
JP7 / JP8 / JP9	1-2 (Normal mode) / 1-2 (Normal mode) / 1-2 (Normal mode)
JP10 / JP11	2-3 (UART) / 2-3 (UART)
JP12	Open (OFF)
JP13	Short (ON)
SW3	1-2 (S_Chip)
SW4	1-2 (Normal)

After turning on the power, confirm that LED1 lights. If it does not light, check that that the board switches are as Table 2.3.

2.3 Using sample code

In case of programming and using the sample code's binary file, refer to "2.4 Programming firmware". In case of building and using the sample code, refer to "2.5 Importing sample code" and "2.6 Building and debugging".

2.4 Programming firmware

Use the Renesas Flash Programmer (Hereafter referred to as RFP) to program the firmware to this product.

Note: Use RFP v3.06.01 or later.

1. Change the board switches as Table 2.4 and connect your PC and CN4 connector with an A – micro B type USB cable.
   After connecting to the PC, press the RESET button to start the MCU in Boot mode.

Table 2.4 Board switch settings for using RFP

Board switch	Setting
JP1	Short (ON)
JP2	Short (ON)
JP3	Short (ON)
JP4 / JP5	2-3 (Regulator Out) / 1-2 (USB)
JP7 / JP8 / JP9	1-2 (Normal mode) / 1-2 (Normal mode) / 1-2 (Normal mode)
JP10 / JP11	1-2 (RFP) / 1-2 (RFP)
JP12	Open (OFF)
JP13	Short (ON)
SW3	1-3 (Boot)
SW4	1-2 (Normal)

2. Start RFP and select “File” → “New Project…”.

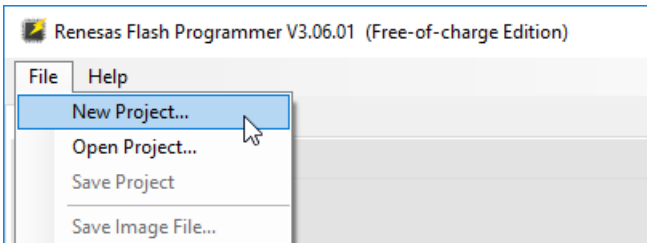

Figure 2.1 Creating New Project

3. In the “Create New Project” window, make the following settings and click the “Connect” button.
   - Microcontroller: RE
   - Project Name: Any name
   - Project Folder: Any folder
   - Communication Tool: COM port

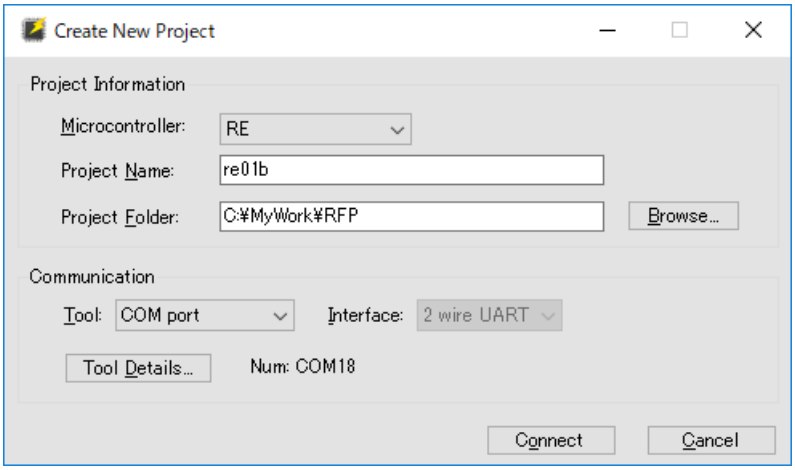

Figure 2.2 Project Setting

4. If the connection is successful, “**Operation completed.**” is displayed.
   If the connection fails, press the RESET button on the board and click the [Connect] button again.

Figure 2.3 Successful Connection

5. Click the “Browse…” button, select hex file in the ROM_Files folder, and click the “Open” button.

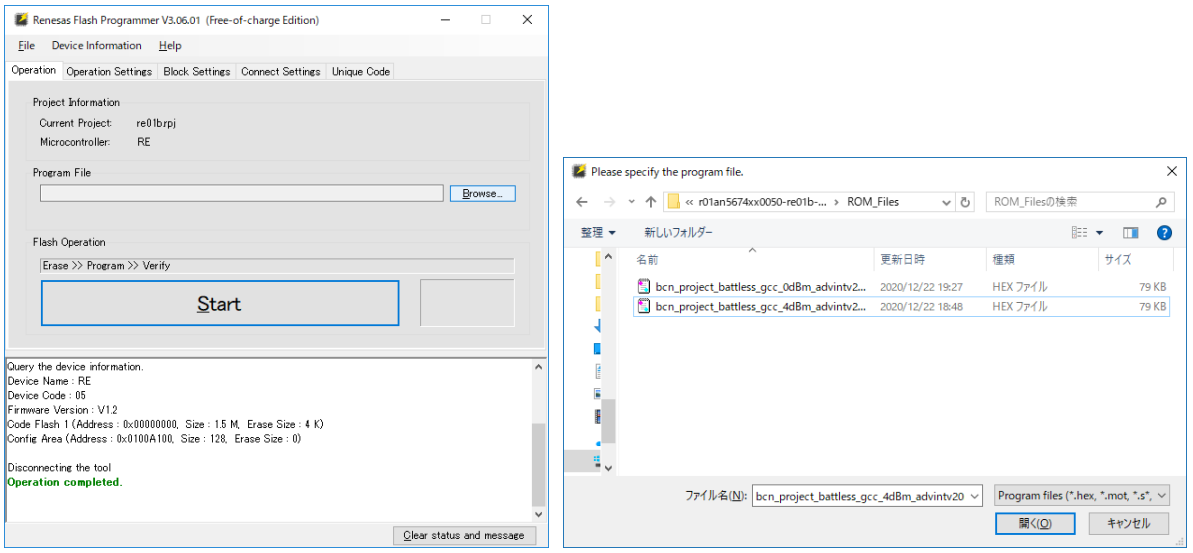

Figure 2.4 Selecting File

6. Click the “Start” button on the “Operation” tab to start programming the firmware.

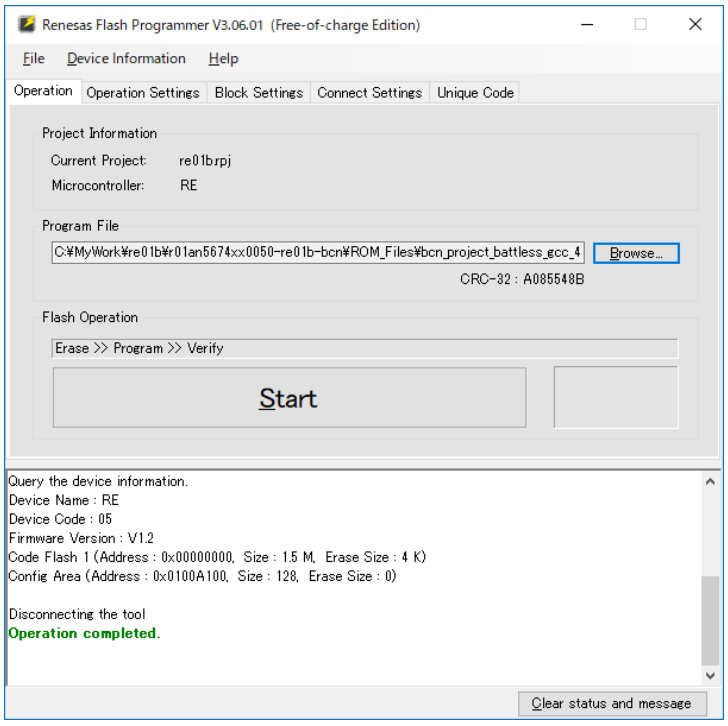

Figure 2.5 Programming Firmware

7. When programming is completed normally, “**Operation completed.**” And “**OK**” are displayed.
   If the programming fails, press the RESET button on the board and click the [Start] button again.

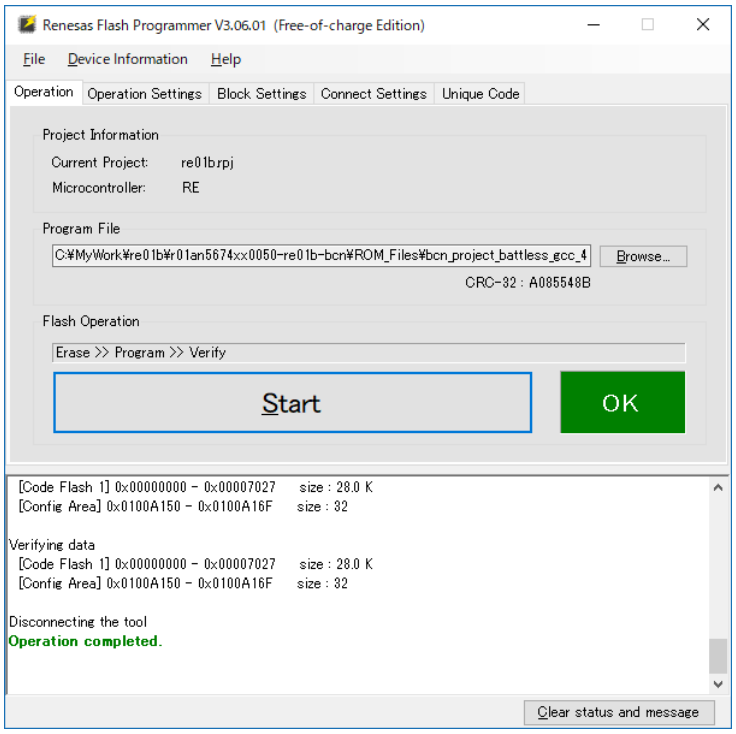

Figure 2.6 Programming Completion

8. After programming is complete, disconnect the USB cable that connected this product to your PC.

2.5 Importing sample code

How to import a sample code is below.

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

2.5.1 In case of e² studio

(1) Select “File” → ”Import”.

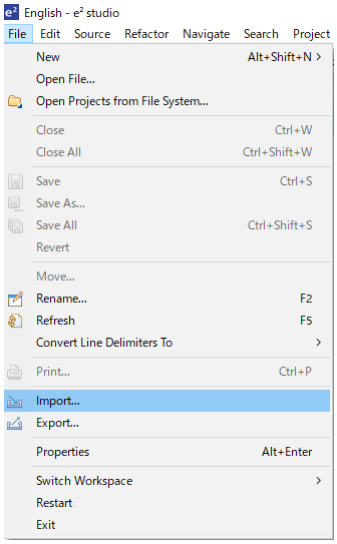

Figure 2.7 File menu

(2) Select “Existing Projects into Workspace” and click “Next” button.

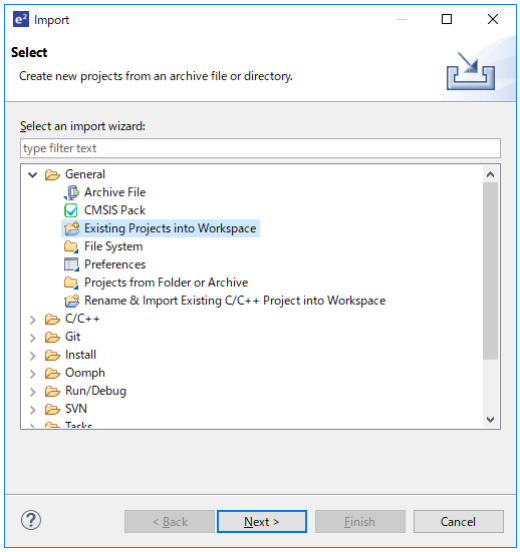

Figure 2.8 Select an import wizard

(3) Select “Select archive file”, click “Browse…” button and select the sample code archive file. Click “Finish” button and the sample code project is imported.

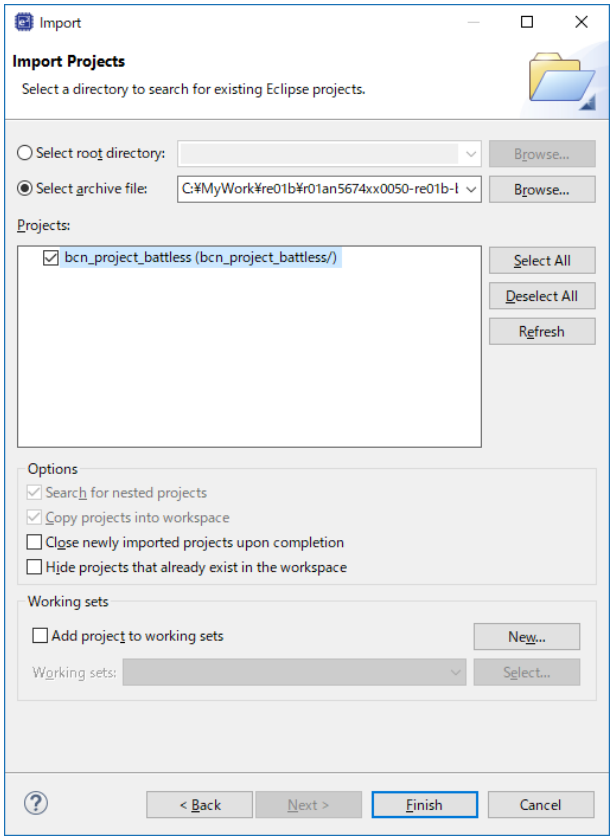

Figure 2.9 Import Projects

2.5.2 In case of EWARM

Click [Project]→[Add Existing Project…] and select the project file (.ewp).

2.6 Building and debugging

2.6.1 In case of e² studio

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

2.6.2 In case of EWARM

Refer to [Help]→[IDE Project Management and Building Guide] and [C-SPY Debugging Guide] on EWARM.

2.7 Beacon sample code

The EB-RE01B board on which the beacon application is written starts sending Advertising packets. Use a smartphone as an example of a remote device that receives Advertising packets.

Prepare a smartphone with the OS shown in Table 2.5 and the smartphone app “GATTBrowser” made by Renesas Electronics.

Table 2.5 Supported OS

OS	Version
iOS	9.0 or later
Android	5.0.1 or later

GATTBrowser	iOS version: https://apps.apple.com/app/gattbrowser/id1163057977 Android version: https://play.google.com/store/apps/details?id=com.renesas.ble.gattbrowser

- The beacon sample starts sending Advertising packets immediately after the power is turned on.
- When scanning from a remote device, the beacon sample is detected by default with the name "RE01B-BCN". Beacon samples use random addresses (static addresses) by default.

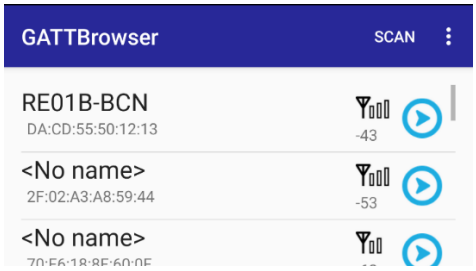

Figure 2.10 Scan result example

3. Beacon stack detail

3.1 R_BCN API

Table 3.1 shows the R_BCN API provided by the Beacon stack. The R_BCN API can be used by including r_bcn_api.h and linking the Beacon stack library.

Table 3.1 R_BCN API

API Name	Description
R_BCN_SystemInit	Initial settings for BLE peripheral functions. This API needs to be called in BoardInit() function.
R_BCN_Open	Open the beacon stack and start sending Advertising packets.
R_BCN_Close	Close the beacon stack and stop sending Advertising packets.
R_BCN_UpdateBeaconData	Update the beacon data (Advertising data).
R_BCN_LPC_EnterLowPowerMode	Transitions the MCU to a low power consumption state and waits for an interrupt.

3.1.1 API detail

3.1.1.1 R_BCN_SystemInit API

Table 3.2 R_BCN_SystemInit API detail

Format	void R_BCN_SystemInit(void);
Description	Initial settings for BLE peripheral functions.
Parameter	None
Return	None

Remarks	This API is provided by the library (lib_ble_bcn.a).
	This API needs to be called in BoardInit() function in main.c.
	
	[API call example]
	#include "r_bcn_api.h"
	void BoardInit(void)
	{
	  R_BCN_SystemInit();
	:
	}

3.1.1.2 R_BCN_Open API

Table 3.3 R_BCN_Open API detail

Format	bcn_status_t R_BCN_Open(const st_bcn_param_t *p_param);	
Description	Open the beacon stack and start sending Advertising packets.	
Parameter	const st_bcn_param_t * p_param: Beacon parameters Specifies the parameters of the Advertising packet to send. See Table 3.8 for the setting range of each element.	
Return	BCN_SUCCESS(0x0000)	Beacon stack open and Advertising packet start successful
	BCN_ERR_INVALID_PTR (0x0001)	Error due to invalid pointer specification - Specify NULL pointer for p_param - Specify NULL pointer for p_param-> p_data when p_param-> data_length is other than zero
	BCN_ERR_INVALID_ARG (0x0003)	Error due to invalid range specification.

Remarks	This API is provided by the library (lib_ble_bcn.a).
	
	[API call example]
	#include "r_bcn_api.h"
	/* beacon data (max 31bytes) */
	uint8_t g_bcn_data[] =
	{
	/* flags */
	0x02,0x01,0x06,
	/* Complete Name */
	0x0A,0x09,'R','E','0','1','B','-','B','C','N',
	/* Manufacture data */
	0x10,0xFF,0xFF,0xFF,0x11,0x22,0x33,0x44,0x55,
	0x66,0x77,0x88,0x99,0xAA,0xBB,0xCC,0xDD
	};
	
	/* beacon parameter */
	const st_bcn_param_t g_bcn_param =
	{
	.adv_intv = 0x0020,
	.o_addr_type = BCN_ADDR_RAND,
	.adv_ch_map = BCN_ADV_CH_ALL,
	.data_length = sizeof(g_bcn_data),
	.p_data = &g_bcn_data[0],
	};
	
	void app_main(void)
	{
	R_BCN_Open(&g_bcn_param);
	}	

3.1.1.3 R_BCN_Close API

Table 3.4 R_BCN_Close API detail

Format	bcn_status_t R_BCN_Close(void);	
Description	Closes the beacon stack and stops sending Advertising packets.	
Parameter	None	
Return	BCN_SUCCESS(0x0000)	Beacon stack closed successfully

Remarks	This API is provided by the library (lib_ble_bcn.a).
	
	[API call example]
	#include "r_bcn_api.h"
	void app_main(void)
	{
	  R_BCN_Close();
	}	

3.1.1.4 R_BCN_UpdateBeaconData API

Table 3.5 R_BCN_UpdateBeaconData API detail

Format	bcn_status_t R_BCN_UpdateBeaconData(uint8_t *p_data, uint8_t length);	
Description	Update the beacon data (Advertising data). When this API is called, Advertising is stopped once, and after updating the Advertising data, Advertising packet transmission is started again.	
Parameter	uint8_t * p_data: Beacon data pointer Specify the pointer of the beacon data to be updated. If you specify zero for length, you can specify a NULL pointer.	
	uint8_t length: Beacon data length Specifies the data length of the beacon data pointer. range: 0 to 31 [bytes]	
Return	BCN_SUCCESS(0x0000)	Beacon data update successful
	BCN_ERR_INVALID_PTR (0x0001)	Error due to invalid pointer specification - Specify NULL pointer for p_data when length is other than zero.
	BCN_ERR_INVALID_ARG (0x0003)	Error due to invalid range specification. - The value of length is out of range

Remarks	This API is provided by the library (lib_ble_bcn.a).
	
	[API call example]
	#include "r_bcn_api.h"
	uint8_t g_bcn_data2[] =
	{
	/* flags */
	0x02,0x01,0x06,
	/* Complete Name */
	0x0A,0x09,'R','E','0','1','B','-','B','C','N',
	/* Manufacture data */
	0x10,0xFF,0xFF,0xFF,0xAA,0xBB,0xCC,0xDD,0xEE,
	0xFF,0xAA,0xBB,0xCC,0xDD,0xEE,0xFF,0xAA
	};
	
	void app_main(void)
	{
	:
	R_BCN_UpdateBeaconData(&g_bcn_data2[0], sizeof(g_bcn_data2));
	}	

3.1.1.5 R_BCN_LPC_EnterLowPowerMode API

Table 3.6 R_BCN_LPC_EnterLowPowerMode API detail

Format	void R_BCN_LPC_EnterLowPowerMode(void);
Description	Transitions the MCU to a low power consumption state and waits for an interrupt.
Parameter	None
Return	None

Remarks	This API is provided in the source code (r_bcn_pf_lowpower.c).
	If you want to repeatedly shift the MCU to low power consumption even after an interrupt occurs, call this API in the main loop.
	
	[API call example]
	#include "r_bcn_api.h"
	void app_main(void)
	{
	  while(1)
	  {
	  R_BCN_LPC_EnterLowPowerMode();
	  }
	}

3.1.2 Macro/Enum definition

Table 3.7 shows the macro/enum definitions provided by the beacon stack.

Table 3.7 Macro/Enum definition

Name	Value	Description
BCN_ADDR_PUBLIC	(0x00)	Set public address
BCN_ADDR_RAND	(0x01)	Set random address
BCN_ADV_CH_37	(0x01)	Use 37 channel
BCN_ADV_CH_38	(0x02)	Use 38 channel
BCN_ADV_CH_39	(0x04)	Use 39 channel
BCN_ADV_CH_ALL	(0x07)	Use all channel (37 to 39ch)
BCN_SUCCESS	0x0000	Success
BCN_ERR_INVALID_PTR	0x0001	NULL specified for pointer
BCN_ERR_INVALID_DATA	0x0002	Invalid data specified
BCN_ERR_INVALID_ARG	0x0003	A value outside the range was specified
BCN_ERR_INVALID_FUNC	0x0004	NULL specified in the callback pointer
BCN_ERR_INVALID_CHAN	0x0005	A channel that does not exist is specified
BCN_ERR_INVALID_MODE	0x0006	Executed in invalid mode
BCN_ERR_UNSUPPORTED	0x0007	This API is not supported
BCN_ERR_INVALID_STATE	0x0008	API call status is invalid
BCN_ERR_INVALID_OPERATION	0x0009	Called API without permission
BCN_ERR_ALREADY_IN_PROGRESS	0x000A	Performed an action that was already performed
BCN_ERR_CONTEXT_FULL	0x000B	Context is full and cannot be newly registered
BCN_ERR_MEM_ALLOC_FAILED	0x000C	Memory allocation failed
BCN_ERR_NOT_FOUND	0x000D	The specified one cannot be found
BCN_ERR_INVALID_HDL	0x000E	Invalid handle specified
BCN_ERR_DISCONNECTED	0x000F	Disconnection occurred during processing
BCN_ERR_LIMIT_EXCEEDED	0x0010	Resource exceeded limit
BCN_ERR_RSP_TIMEOUT	0x0011	Response wait timed out
BCN_ERR_NOT_YET_READY	0x0012	Not ready to work
BCN_ERR_UNSPECIFIED	0x0013	An unknown error has occurred

3.1.3 Structure definition

3.1.3.1 st_bcn_param_t structure

Beacon parameter structure used by R_BCN_Open API.

Table 3.8 st_bcn_param_t structure

<table>
<tr><th>Member Name</th><th>Type</th><th>Description</th></tr>
<tr><td>adv_intv</td><td>uint16_t</td><td>Specify the Advertising interval.
Interval time (ms) = adv_intv * 0.625

The valid range depends on the number of channels set by the adv_ch_map parameter.

<table>
<tr><th>adv_ch_map</th><th>Valid range</th></tr>
<tr><td>0x01,0x02,0x04 (Use one channel)</td><td>0x0001 to 0x4000</td></tr>
<tr><td>0x03,0x05,0x06 (Use two channels)</td><td>0x0002 to 0x4000</td></tr>
<tr><td>0x07 (Use three channels)</td><td>0x0003 to 0x4000</td></tr>
</table>
</td></tr>
<tr><td>o_addr_type</td><td>uint8_t</td><td>Specify the BD address type of the local device.
BCN_ADDR_PUBLIC (0x00) : Public address
BCN_ADDR_RAND (0x01) : Random address</td></tr>
<tr><td>adv_ch_map</td><td>uint8_t</td><td>Specifies the channel to send Advertising packets.
It can be specified by the logical sum of the following macros.
BCN_ADV_CH_37 (0x01) : Use 37ch
BCN_ADV_CH_38 (0x02) : Use 38ch
BCN_ADV_CH_39 (0x04) : Use 39ch
BCN_ADV_CH_ALL (0x07) : Use 37,38,39ch</td></tr>
<tr><td>reserved</td><td>uint8_t</td><td>Reserved area.</td></tr>
<tr><td>data_length</td><td>uint8_t</td><td>Specify the Advertising data length.
Valid range: 0 to 31 [bytes]</td></tr>
<tr><td>*p_data</td><td>uint8_t</td><td>Specifies a pointer to the Advertising data.
If data_length is zero, this element is ignored.</td></tr>
</table>

3.2 Configuration options

The configuration options of the Beacon stack can be configured in r_bcn_cfg.h. The option names and setting values are listed in the table shown as follows.

Table 3.9 Configuration options

Configuration options (r_bcn_cfg..h)	
BCN_CFG_RF_DBG_PUB_ADDR Default : "{0xFF,0xFF,0xFF,0x50,0x90,0x74}"	Initial Public Address. If the public addresses in the Code Flash Memory are all 0x00 or 0xFF, the device adopts this public address. If all 0x00 or 0xFF is set, the device uses 74:90:50:FF:FF:FF as public address. Refer to "5.1 Device-specific Data Management" for details.
BCN_CFG_RF_DBG_RAND_ADDR Default : “{0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}"	Initial Random Address. If the static addresses in the Code Flash Memory are all 0x00 or 0xFF, the device adopts this random address. If all 0x00 or 0xFF is set, the device uses the value generated with the MCU specific value as the static address. Refer to "5.1 Device-specific Data Management" for details.
BCN_CFG_RF_CLVAL Default : "7"	Adjustment value of the 32MHz crystal oscillator. Set this option according to the board environment. Range : 0 to 15 Refer to "Tuning procedure of Bluetooth dedicated clock frequency(R01AN5488)" for details.
BCN_CFG_RF_DDC_EN Default : "1"	Enable or disable the DC-DC on the RF. 0: Disable 1: Enable
BCN_CFG_RF_MAX_TX_POW Default : "0"	Maximum transmit power configuration. Range : 0 to 1 0: max +0dBm 1: max +4dBm
BCN_CFG_RF_DEF_TX_POW Default : "0"	Default transmit power level. Range : 0 to 2 This option depends on the BCN_CFG_RF_MAX_TX_POW option. If the BCN_CFG_RF_MAX_TX_POW option is 0(0dBm), this option is as follows. 0(High) : 0dBm 1(Mid) : 0dBm 2(Low) : -18dBm If the BCN_CFG_RF_MAX_TX_POW option is 1(+4dBm), this option is as follows. 0(High) : +4dBm 1(Mid) : 0dBm 2(Low) : -20dBm
BCN_CFG_DEV_DATA_CF_BLOCK Default : "383"	The Code Flash(ROM) block stored the device specific data. Range : -1 to 383 If this option is set to -1, the device specific data in the Code Flash isn't used. The blocks from "0" to "15" are the Start-Up Program Protection block. If the Start-Up Program Protection is used, don’t use the blocks from "0" to "15". Refer to "5.1 Device-specific Data Management" for details.

3.3 MCU low power consumption

Beacon stack provides the control API of MCU Low Power Consumption Function (LPC) in the source code (r_bcn_pf_lowpower.c).

- Use *R_BCN_LPC_EnterLowPowerMode()* to enter to low power consumption state.
  This function is called inside the main loop to do the following:
  - Disable MCU interrupts
  - Check that there is no problem even if each component shifts to Low power consumption state
  - Execute the transition processing to Low power consumption state of each component
  - Enter MCU to Low power consumption state..
  - After MCU wakes-up from Low power consumption state, resume each component to the normal state.

3.4 Code size

Table 3.10 shows the beacon stack code size. This code size does not include the size of application code or other CMSIS driver code.

Table 3.10 Beacon stack code size

Device	Compiler	Category	Beacon stack size
RE01B	GCC	ROM	24,531 bytes
		RAM	2,589 bytes
	IAR	ROM	21,900 bytes
		RAM	2,586 bytes

4. Create project

This section describes how to create a project to develop a Beacon application . For the details about creating a project, see “Getting Started Guide to Development Using CMSIS Package (R01AN5310)”.

4.1 Import a project

Referring “2.5 Importing sample code”, import a sample code.

4.2 Stack/Heap size configuration

In case of e² studio, set the memory size needed for BLE to the following file in the project.

Device\startup_RE01B_1500KB.c

```
#define SYSTEM_CFG_STACK_SIZE  (0x1000)
#define SYSTEM_CFG_HEAP_SIZE   (0x400)
```

In case of EWARM, set the memory size needed for BLE to [Project]→[Options…]→[Linker]→[Config] tab→[Linker configuration file]→[Edit…] button→[Stack/Heap Sizes] tab in the project.

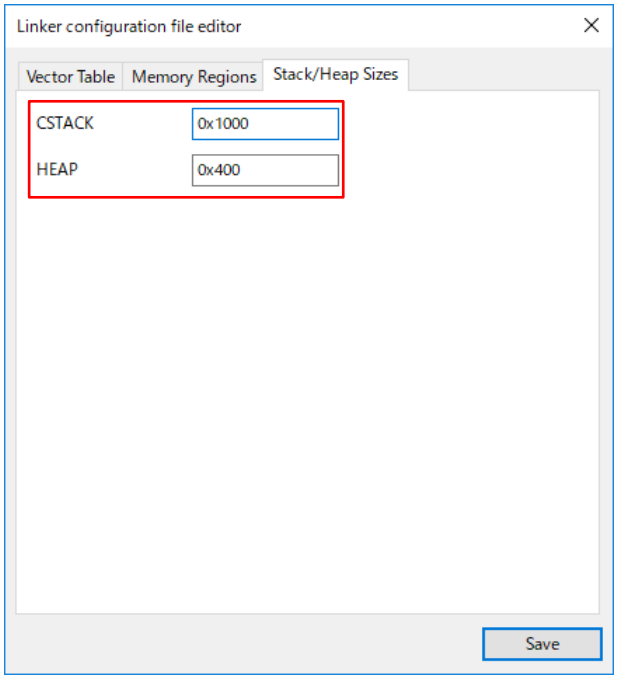

Figure 4.1 Scack/Heap size configuration on EWARM

4.3 Clocks configuration

Referring “Getting Started Guide to Development Using CMSIS Package (R01AN5310) 6.1.2 Setting Clock/Power Control Modes on Start of Operation”, configure the clocks to the following file in the project.

Device\Config\r_core_cfg.h

```
#define SYSTEM_CFG_HOCO_ENABLE              (1)
#define SYSTEM_CFG_HOCO_FREQUENCY           (1)
#define SYSTEM_CFG_LOCO_ENABLE              (1)
#define SYSTEM_CFG_CLOCK_SOURCE             (0)
#define SYSTEM_CFG_ICK_PCKA_DIV             (0)
#define SYSTEM_CFG_PCKB_DIV                 (0)
#define SYSTEM_CFG_POWER_CONTROL_MODE       (1)
```

If the Beacon stack is used, set the clocks within the following range.

- System clock / Peripheral module clock A (ICLK/PCLKA): more than 32MHz
- Peripheral module clock B (PCLKB): more than 32MHz

The Beacon stack is optimized with ICLK/PCLKA, PCLKB: 32MHz.

Renesas prerequires configuring the frequency of ICLK and PCLKA to 32MHz to maximize the Beacon stack performance.

4.4 Linker configurations

The Beacon stack is provided as static library. Libraries shown in Table 4.1 are included in the Device\BCN\lib directory.

Table 4.1 Beacon stack libraries

Environment	Library
GCC environment	/lib_gcc_sotb/lib_ble_bcn.a
IAR environment	/lib_iar_sotb/lib_ble_bcn.a

Configure the followings to make the Beacon application project available to the Beacon stack library.

(1) Library configuration

In case of e2 studio, Click [Project] → [Properties].
Select “C/C++ Build” → “Settings” → “Tool Settings” → “Cross ARM C Linker” → “Libraries”.
Confirm that the followings are added (Figure 4.2). If not, add the followings.

User defined archive (library) files (-l): “:lib_ble_bcn.a”
User defined archive search directories (-L): "${ProjDirPath}/Device/BCN/lib/lib_gcc_sotb"

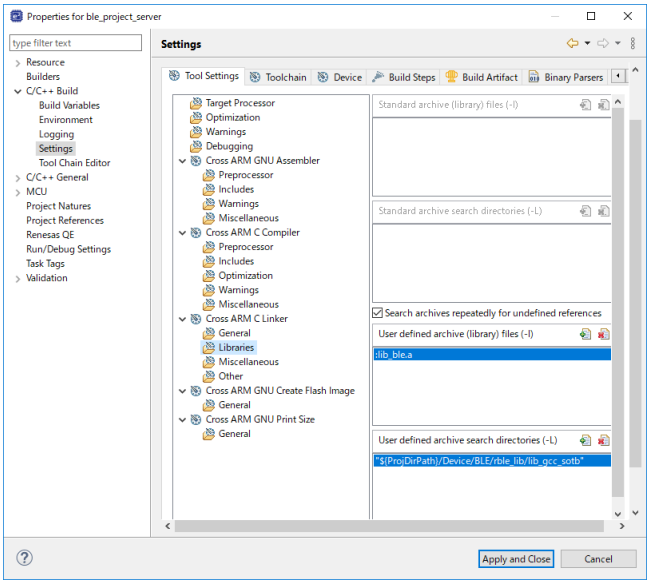

Figure 4.2 Linker configuration

In case of EWARM, Select [Project]→[Options…]→[Linker]→[Library].

Confirm that the [Automatic runtime library selection] is checked (Figure 4.3).

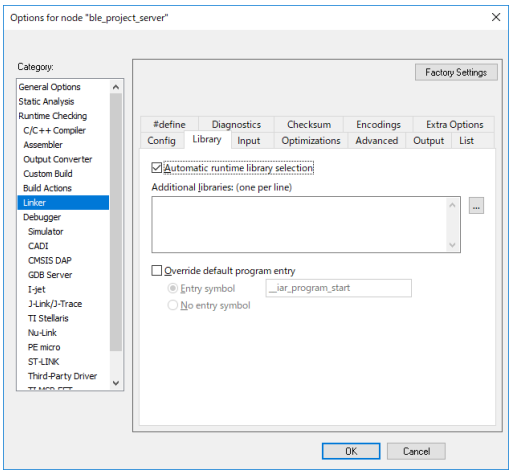

Figure 4.3 Linker configuration

5. Optional function

5.1 Device-specific Data Management

Bluetooth Device Address (hereinafter referred to as BD address) used by Beacon stack can be written as device-specific data in the code flash memory. This allows user to set different BD address for multiple devices using the same firmware.

Device-specific data is placed in a different area from the firmware program area. If the device-specific data is not deleted when rewriting the firmware, the same BD address can be used continuously. If the device-specific data is deleted, determine the BD address according to “5.1.4 BD address adoption flow”.

5.1.1 Specifying Device-specific data location block

The block number of the code flash memory where device-specific data is located can be specified with the BCN_CFG_DEV_DATA_CF_BLOCK configuration options.

The block number of the code flash memory is block 0 at beginning of address (0x00000000) and block 383 at end of address (0x0017F000).

Figure 5.1 shows the code flash memory block configuration.

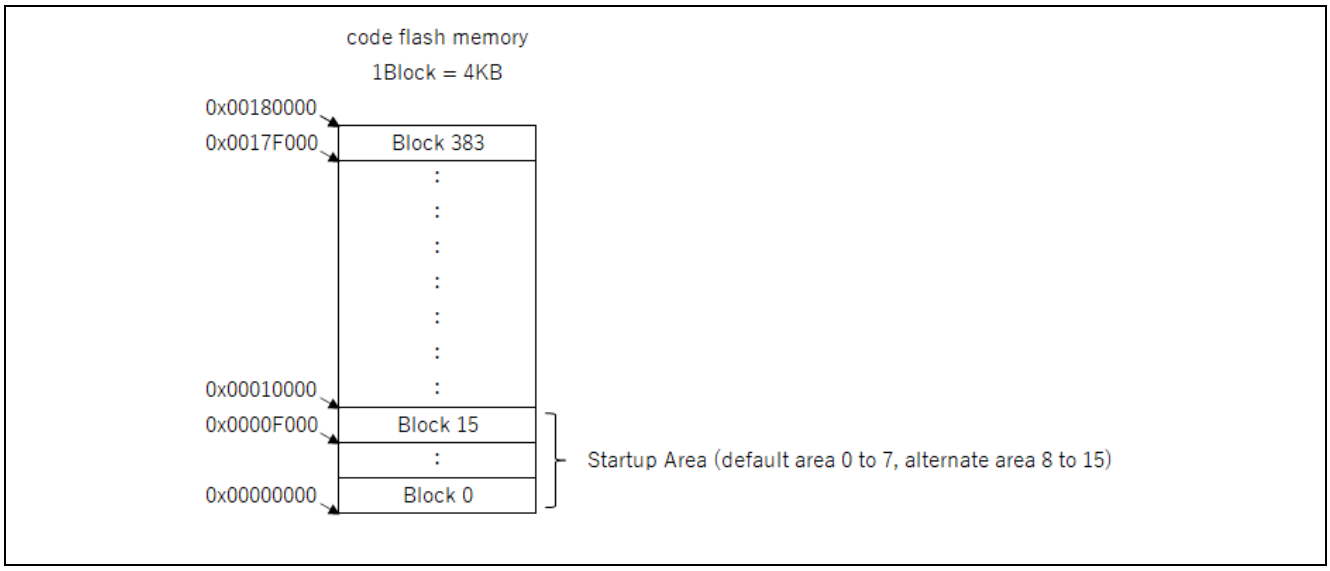

Figure 5.1 Flash memory block configuration

When placing device-specific data in the code flash memory, it is necessary to specify blocks that are not used in program code. In addition, it is necessary to write device-specific data to the top address in specified user area block.

When using MCU Startup Area Select function, do not place device-specific data in blocks 0 to 15 of the code flash memory.

5.1.2 Device-specific data format

Table 5.1 shows the device-specific data format.

Table 5.1 device-specific data format

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

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

Figure 5.2 shows an example of device-specific data flash memory layout.

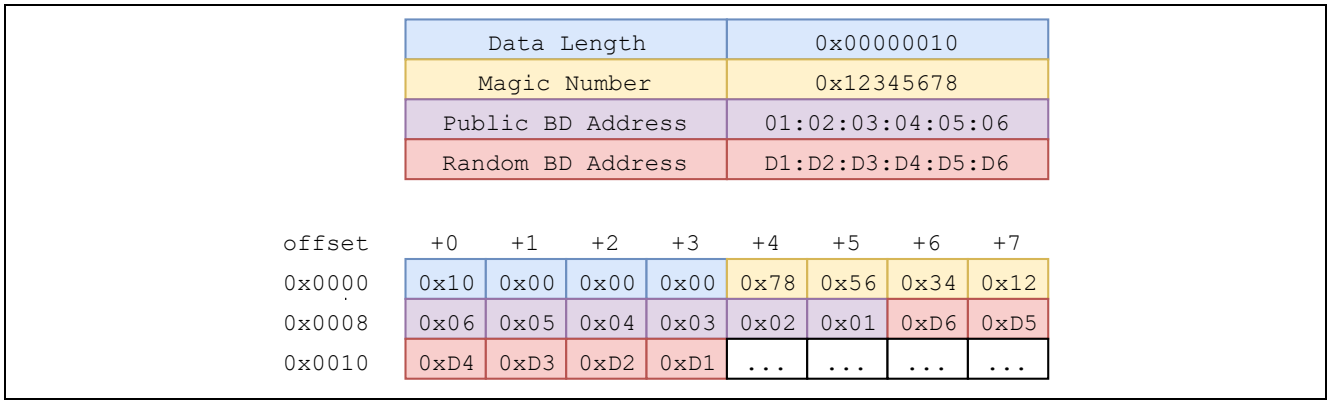

Figure 5.2 Device-specific data flash memory layout

5.1.3 Writing to the code flash memory

To write device-specific data to the code flash memory, use Renesas Flash Programmer (RFP) unique code function to write to the code flash memory at the same time as firmware program data.

Figure 5.3 shows an overview of writing device-specific data using RFP.

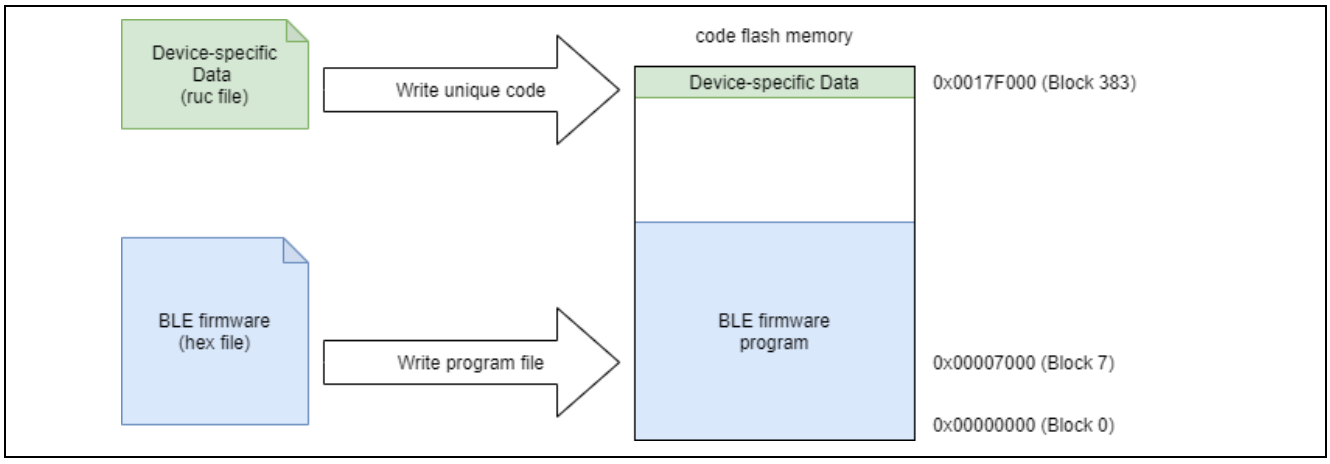

Figure 5.3 Writing device-specific data using RFP

Figure 5.4 shows an example of setting device-specific data for RFP Unique Code (ruc) file.

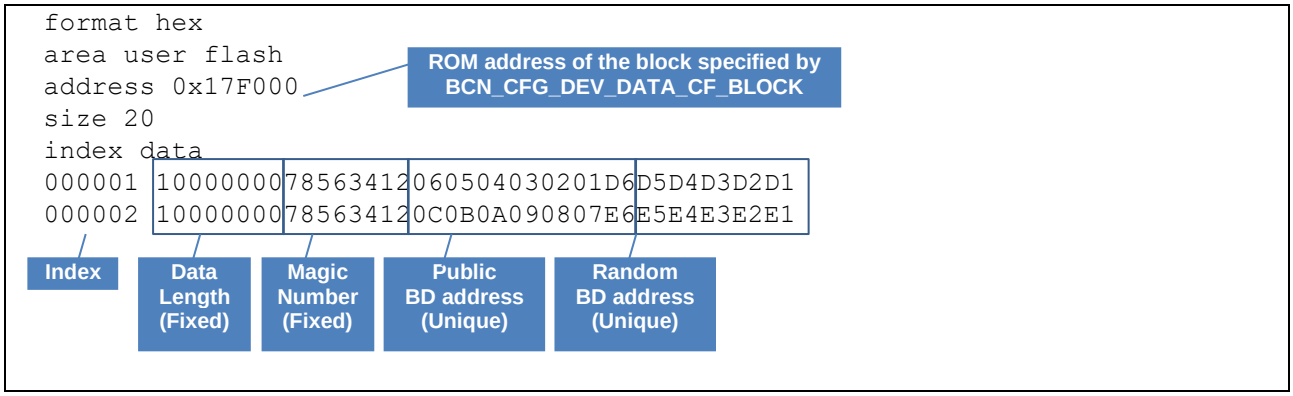

Figure 5.4 Setting device-specific data for RFP Unique Code

5.1.4 BD address adoption flow

Beacon stack adopts initial value of BD address in following priority order in *R_BCN_Open()* API.

(1) Code flash memory specified block

(2) Firmware initial value (BCN_CFG_RF_DEB_PUB_ADDR or BCN_CFG_RF_DEB_RAND_ADDR)

For Random BD address, if BD addresses for all areas are not specified, static address is generated from Unique ID of MCU.

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

Note: A static address consists of random numbers. The possibility of duplicate values with other devices is not zero.

Figure 5.5 shows BD address adoption flow of Beacon stack.

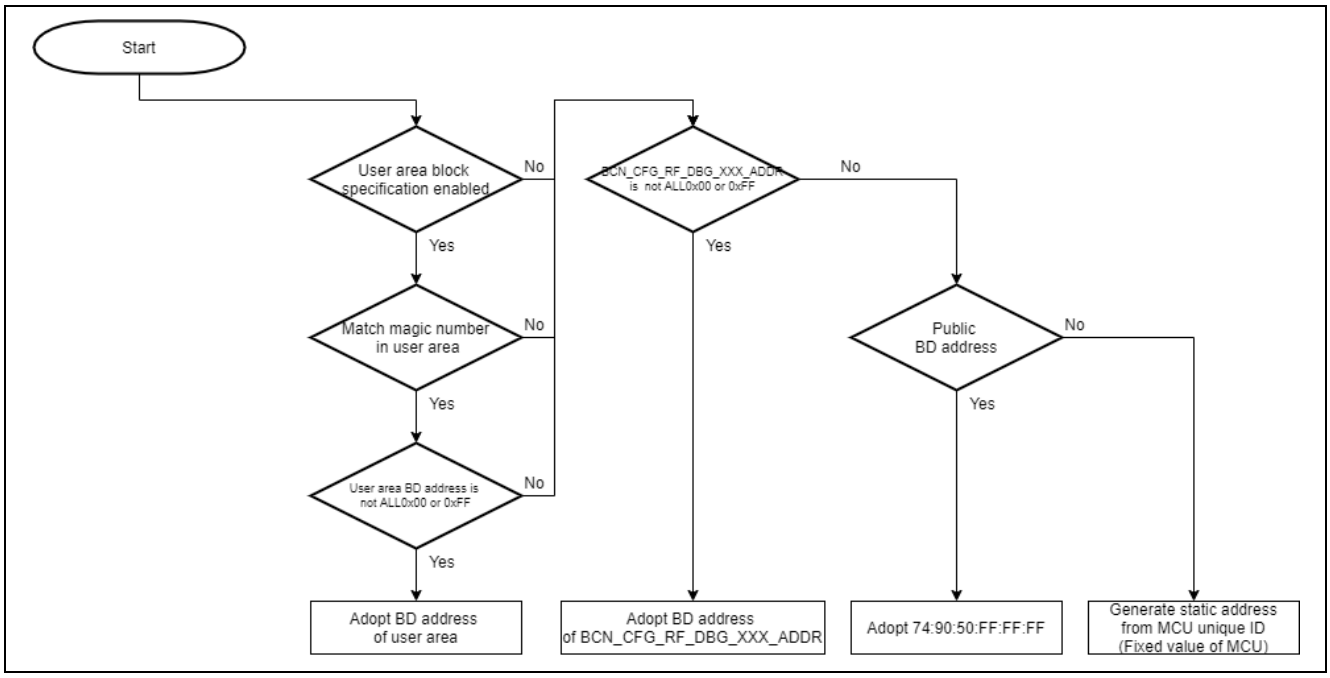

Figure 5.5 BD address adoption flow of Beacon stack

Since Beacon stack does not check format of BD address written in each area (1)-(2), when setting static address, set value that matches the format shown in Figure 5.6.

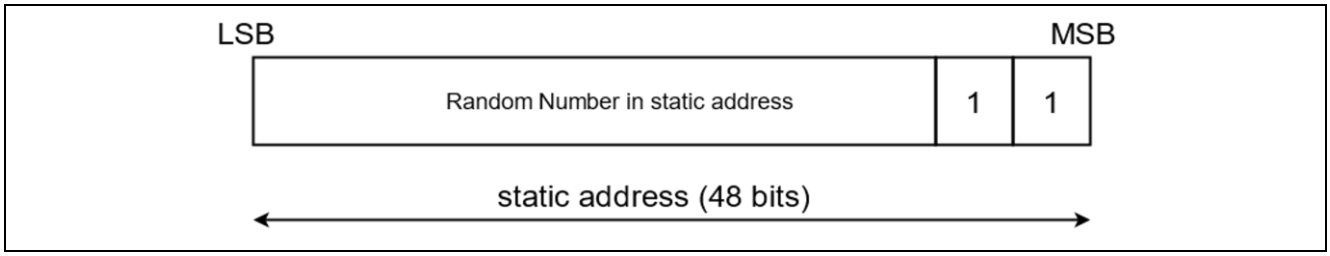

Figure 5.6 Static address format

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Feb.10.2021	—	First edition issued.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.