User's Manual

Bluetooth® Low Energy Protocol Stack

User's Manual

Renesas MCU

Target Device

RL78/G1D

Renesas Electronics
www.renesas.com

Rev.1.19 Jan 2022

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

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.

   ¾ 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.

   ¾ 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.

   ¾ 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.

   ¾ 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.

   ¾ 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.

How to Use This Manual

1. Purpose and Target Readers

This manual describes setup method, organization, and features of the Bluetooth Low Energy protocol stack (BLE software), which is used to develop Bluetooth applications that incorporate the Renesas Bluetooth low energy microcontroller RL78/G1D. It is intended for users designing application systems incorporating this software. A basic knowledge of microcontrollers and Bluetooth low energy is necessary in order to use this manual.

Related documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Document Name	Document No.
Bluetooth Low Energy Protocol Stack	
User's Manual	This manual
API Reference Manual : Basics	R01UW0088E
API Reference Manual : FMP (Obsolete)	R01UW0089E
API Reference Manual : PXP (Obsolete)	R01UW0090E
API Reference Manual : HTP (Obsolete)	R01UW0091E
API Reference Manual : BLP (Obsolete)	R01UW0092E
API Reference Manual : HOGP (Obsolete)	R01UW0093E
API Reference Manual : ScPP (Obsolete)	R01UW0094E
API Reference Manual : HRP (Obsolete)	R01UW0097E
API Reference Manual : CSCP (Obsolete)	R01UW0098E
API Reference Manual : CPP (Obsolete)	R01UW0099E
API Reference Manual : GLP (Obsolete)	R01UW0103E
API Reference Manual : TIP (Obsolete)	R01UW0106E
API Reference Manual : RSCP (Obsolete)	R01UW0107E
API Reference Manual : ANP (Obsolete)	R01UW0108E
API Reference Manual : PASP (Obsolete)	R01UW0109E
API Reference Manual : LNP (Obsolete)	R01UW0113E
Application Note : Sample Program	R01AN1375E
Application Note : rBLE Command Specification	R01AN1376E

List of Abbreviations and Acronyms

Abbreviation	Full Form	Remark
ANP	Alert Notification Profile	
ANS	Alert Notification Service	
API	Application Programming Interface	
ATT	Attribute Protocol	
BAS	Battery Service	
BB	Base Band	
BD_ADDR	Bluetooth Device Address	
BLE	Bluetooth low energy	
BLP	Blood Pressure Profile	
BLS	Blood Pressure Service	
CPP	Cycling Power Profile	
CPS	Cycling Power Service	
CSCP	Cycling Speed and Cadence Profile	
CSCS	Cycling Speed and Cadence Service	
CSRK	Connection Signature Resolving Key	
CTS	Current Time Service	
DIS	Device Information Service	
EDIV	Encrypted Diversifier	
FMP	Find Me Profile	
GAP	Generic Access Profile	
GATT	Generic Attribute Profile	
GLP	Glucose Profile	
GLS	Glucose Service	
HCI	Host Controller Interface	
HID	Human Interface Device	
HIDS	HID Service	
HOGP	HID over GATT Profile	
HRP	Heart Rate Profile	
HRS	Heart Rate Service	
HTP	Health Thermometer Profile	
HTS	Health Thermometer Service	
IAS	Immediate Alert Service	
IRK	Identity Resolving Key	
L2CAP	Logical Link Control and Adaptation Protocol	
LE	Low Energy	
LL	Link Layer	

Abbreviation	Full Form	Remark
LLS	Link Loss Service	
LNP	Location and Navigation Profile	
LNS	Location and Navigation Service	
LTK	Long Term Key	
MCU	Micro Controller Unit	
MITM	Man-in-the-middle	
MTU	Maximum Transmission Unit	
OOB	Out of Band	
OS	Operating System	
PASP	Phone Alert Status Profile	
PASS	Phone Alert Status Service	
PXP	Proximity Profile	
RF	Radio Frequency	
RSCP	Running Speed and Cadence Profile	
RSCS	Running Speed and Cadence Service	
RSSI	Received Signal Strength Indication	
ScPP	Scan Parameters Profile	
ScPS	Scan Parameters Service	
SM	Security Manager	
SMP	Security Manager Protocol	
STK	Short Term Key	
TK	Temporary Key	
TPS	Tx Power Service	
UART	Universal Asynchronous Receiver Transmitter	
UUID	Universal Unique Identifier	

Abbreviation	Full Form	Remark
APP	Application	
CSI	Clocked Serial Interface	
IIC	Inter-Integrated Circuit	
RSCIP	Renesas Serial Communication Interface Protocol	
VS	Vendor Specific	

These commodities, technology or software, must be exported in accordance with the export administration regulations of the exporting country. Diversion contrary to the law of that country is prohibited.

Contents

1. Overview

This manual describes the API (Application Program Interface) of the basic features of the Bluetooth Low Energy protocol stack (BLE software), which is used to develop Bluetooth applications that incorporate Renesas Bluetooth low energy microcontroller RL78/G1D.

For details about the BLE software APIs, see *Bluetooth Low Energy Protocol Stack API Reference Manual*.

2. Applicability

The descriptions in this manual apply to Bluetooth Low Energy protocol stack Version 1.11 or later.

3. Restrictions

This section describes the restrictions that apply to BLE software.

4. Installing BLE Software

4.1 Components Included

The BLE software CD includes the followings:

- Documents
  - Bluetooth Low Energy Protocol Stack User's Manual (this document)
  - Bluetooth Low Energy Protocol Stack API Reference Manual
  - Bluetooth Low Energy Protocol Stack Sample Program Application Note
  - rBLE command specifications
- Project files used for creating the executable file
  - Executable file
  - BLE software library
  - Sample source code
  - Source code that configures parameters
  - e^2 studio project file
  - CS+ for CC project file
  - CS+ for CA,CX project file
- Sample applications for computer
  - Executable file
  - Source code
  - Microsoft Visual Studio Express 2013 project file
- HCI packet monitor application for computer
  - Executable file
  - INI file

4.2 BLE software build environment

The environment in which BLE software was built is shown below.

- Hardware environment
  - Host
    - PC/AT™-compatible computer
    - Processor : At least 1.6 GHz
    - Main memory : At least 1 GB
    - Display : 1024 x 768 or higher resolution and 65,536 colors
    - Interface : USB 2.0 (E1 and USB-serial conversion cable)
- Tools used
    - Renesas on-chip debugging emulator E1
- Software environment
    - Windows 7
    - Microsoft Visual Studio Express 2013 for Windows Desktop Update4
    - Microsoft .NET Framework 4 + language pack

- Renesas CS+ for CC V3.03.00/ RL78 Compiler CC-RL V1.02.00
  or e² studio 4.2.0.012/RL78 Family C Compiler Package V1 (without IDE) V1.02.00
  or Renesas CS+ for CA, CX V3.01.00/Renesas CA78K0R V1.71
- Renesas Flash Programmer v3.00.00
  (available from http://am.renesas.com/products/tools/flash_prom_programming/rfp/index.jsp)
  [Note] If you want to use the unique code embedding function for customer-specific information area, please use the Renesas Flash Programmer v2.05.02.

For details about the environment in which to run the sample application for computers, see Bluetooth Low Energy Protocol Stack Sample Program Application Note.

4.3 Installation Procedure

Copy the CD contents to any folder in your computer.

Also, download the EEPROM Emulation Library and Code Flash Library corresponding to your development Environment from Renesas website and copy to the following folder. The EEPROM Emulation Library and Code Flash Library are provided by Renesas Electronics Corporation. Refer to 4.4.2(5) and 4.4.2(6) in details.

Note: If using the e² studio, cannot be include multi-byte characters and blank in the BLE software installation folder path.

・EEPROM Emulation Library(CS+ for CC/e² studio (CC-RL))
\Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\dataflash\cc_rl

・EEPROM Emulation Library(CS+ for CA,CX)
\Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\dataflash\cs

・Code Flash Library(CS+ for CC/e² studio)
\Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\codeflash\cc_rl

・Code Flash Library(CS+ for CA,CX)
\Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\codeflash\cs

4.4 Folder Organization

The detail of file and folder after installation is shown below.

4.4.1 \Renesas\BLE_Software_Ver_X_XX\Manual\

This folder includes the manuals. Please read them before using the product.

4.4.2 \Renesas\BLE_Software_Ver_X_XX\RL78_G1D

(1) \ROM_File\

This folder contains the executable files (hex files) of the BLE software programs that run on the RL78/G1D. Write these programs to the RL78/G1D on-chip flash memory.

For how to write to the on-chip flash memory, see the Renesas Flash Programmer flash memory programming software User's Manual.

Also, if you have already written the BD address into on-chip data flash memory, change the operation mode of the RFP software configuration information list to "block (code flash)". If you don't, the written BD address might be erased.

The contents of the executable file that is stored in this folder are shown in Table 4-1. See a section 6 *Creating Executable Files* about Sample Custom profile. See *Bluetooth Low Energy Protocol Stack Application Note: Sample Program* about how to create an executable file.

Table 4-1 Overview of the executable file

Folder name		Name of the executable file	Content
\ca78k0r			Folder for CS+ for CA,CX
	\Embedded		Folder for Embedded configuration
		RL78_G1D_CE(SCP).hex	Executable file that corresponds to the Sample Custom Profile
	\Modem		Folder for Modem configuration
		RL78_G1D_CM(SCP).hex	Executable file that corresponds to the Sample Custom Profile
		RL78_G1D_CM(DTM_2WIRE).hex	Executable file that corresponds to 2-Wire UART Direct Test Mode
\ccrl			Folder for CC-RL
	\Embedded		Folder for Embedded configuration
		RL78_G1D_CE(SCP).hex	Executable file that corresponds to the Sample Custom Profile
	\Modem		Folder for Modem configuration
		RL78_G1D_CM(SCP).hex	Executable file that corresponds to the Sample Custom Profile

(2) \Project_Source\

This folder contains the BLE software library and sample source code required for building the executable files (hex files) of the software programs that run on the RL78/G1D.

(3) \Project_Source\renesas\tools\project\

This folder contains the project/workspace files for each development environment which are required for building the executable files (hex files) of the software programs that run on the RL78/G1D. In addition, projects/workspaces in the Embedded configuration and Modem configuration for each development environment are contained. Build the program by using these project/workspace files in your development environment to generate executable files.

For how to build the program, see *6.2*.

(4) \Project_Source\renesas\src\

This folder contains the files for configuring the parameters that can be changed by the user (source code). Change the parameter settings as required before building the program.

For details about the configurable parameters and how to change the settings, see *6.1*.

(5) \Project_Source\renesas\src\driver\dataflash\cc_rl\ or \cs\

Copy the EEPROM Emulation Library corresponding to your development environment to this folder. The EEPROM

Emulation Library is downloaded from Renesas website. For reference, shows how to obtain a tested version by the BLE software from Renesas website of the North America area. In addition, operating procedures might be changed without a notice by the renewals of the website.

Open the Renesas website (http://www.renesas.com) and select [Americas]. Select [Development Tools]-[Flash and PROM Programming]-[Flash Libraries]-[Data Flash Libraries]-[Download]. Set the following filter.

- Promotion Type – current tools
- µC Series – RL78
- Tool type – SW-Tool

Select [EEPROM_EMULATION_RL78]. Get the following two files and execute files for unzip the EEPROM Emulation Library.

- RENESAS_EEL_RL78_T01E_V1.20.zip / RENESAS_FDL_RL78_T01E_V1.20.zip (CS+ for CA,CX)
- RENESAS_EEL_RL78_T02E_V1.20.zip / RENESAS_FDL_RL78_T02E_V1.30.zip (CC-RL)

The files to be copied are shown below.

CS+ for CC or e^2 studio (CC-RL) version:

- RENESAS_EEL_RL78_T02E_V1.20.zip / RENESAS_FDL_RL78_T02E_V1.30.zip
  - eel.h
  - eel.lib
  - eel_types.h
  - fdl.h
  - fdl.lib
  - fdl_types.h

CS+ for CA, CX version:

- RENESAS_EEL_RL78_T01E_V1.20.zip / RENESAS_FDL_RL78_T01E_V1.20.zip
  - eel.h
  - eel.lib
  - eel_types.h
  - fdl.h
  - fdl.lib
  - fdl_types.h

(6) \Project_Source\renesas\src\driver\codeflash\cc_rl\ or cs\

Copy the Code Flash Library corresponding to your development environment to this folder. The Code Flash Library is downloaded from Renesas website. For reference, shows how to obtain a tested version by the BLE software from Renesas website of the North America area. In addition, operating procedures might be changed without a notice by the renewals of the website.

Open the Renesas website (http://www.renesas.com) and select [Americas]. Select [Development Tools]-[Flash and PROM Programming]-[Flash Libraries]-[Flash Self Programing Libraries]-[Download]. Set the following filter.

- Promotion Type – current tools
- µC Series – RL78
- Tool type – SW-Tool

Select [SelfFlib_RL78]. Get the following file and execute file for unzip the Code Flash Library.

- RENESAS_FSL_RL78_T01E_V1.20.zip (CS+ for CA,CX or CC-RL)

The files to be copied are shown below.

CS+ for CC or e^2 studio (CC-RL) version

- fsl.h
- fsl.lib
- fsl_types.h

CS+ for CA, CX version

- fsl.h
- fsl.lib
- fsl_types.h

4.4.3 \Renesas\BLE_Software_Ver_X_XX\BLE_Sample\

This folder contains the executable files of the BLE software sample program that runs on a computer when BLE software is used in the Modem configuration. About the detail of the sample program, see *Bluetooth Low Energy Protocol Stack Application Note: Sample Program*.

5. BLE Software Configuration

BLE software refers to the set of software that includes BLE stacks compliant with the Bluetooth Low Energy protocol (Bluetooth v4.2). The following section describes the BLE software configuration in detail.

5.1 Configuration

Figure 5-1 shows the BLE software configuration.

BLE software runs in a configuration in which the application is mounted on the RL78/G1D (hereafter referred to as the Embedded configuration) and in a configuration in which the application is mounted on another MCU (hereafter referred to as the Modem configuration). BLE software provides APIs which can use the same application in both configurations.

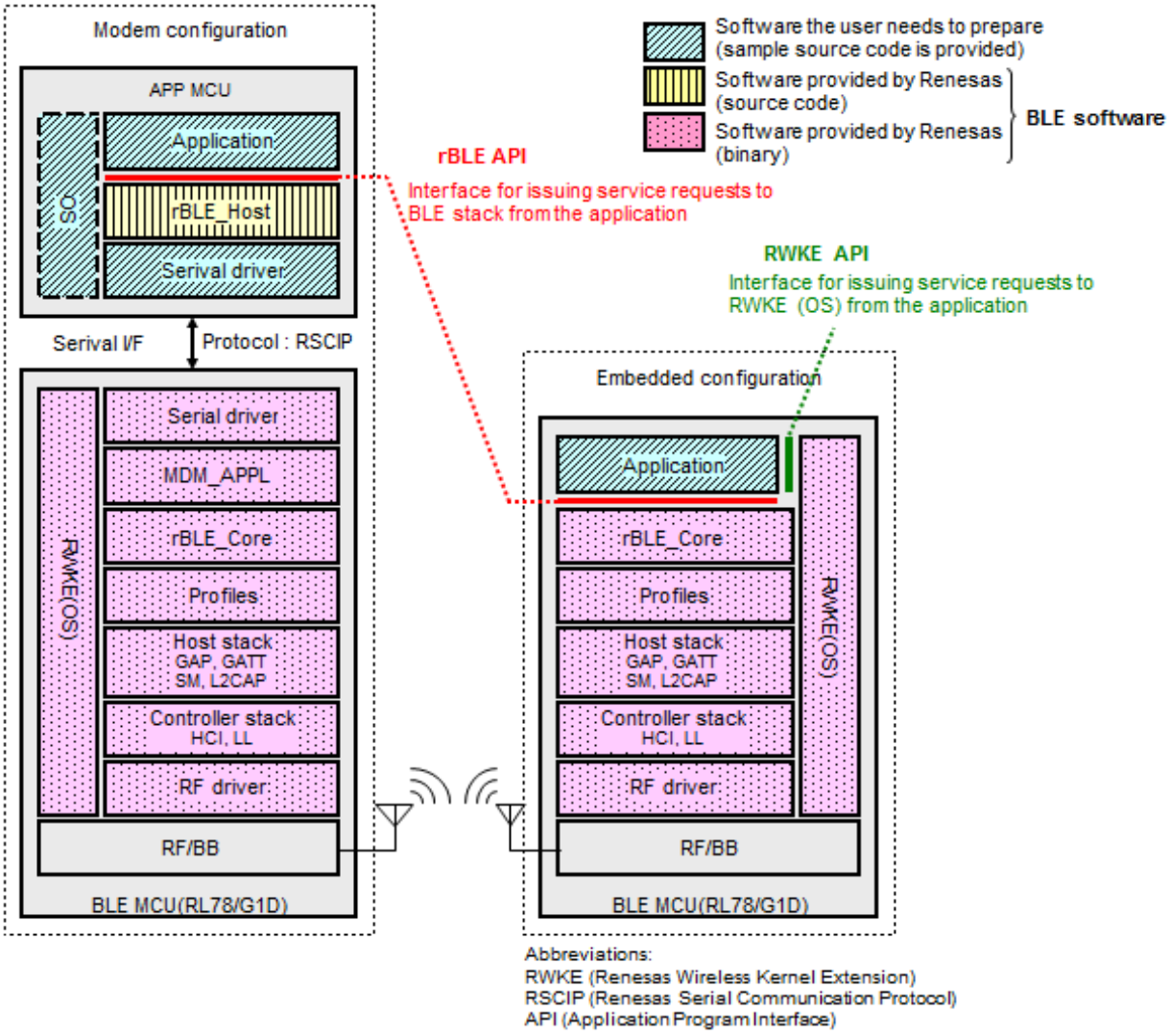


Figure 5-1 BLE Software Configuration

BLE software in the Modem configuration runs on two chips, the APP MCU and the BLE MCU (RL78/G1D). BLE software is configured of an rBLE_Host block that runs on the APP MCU (block in the figure), and software that runs on the BLE MCU (blocks in the figure).

The software to be prepared by the user (blocks in the figure) consists of the APP MCU's application block, serial

communication driver block, and OS block. However, if there is no OS in the APP MCU, software for the OS block does not have to be prepared because the rBLE_Host block does not use resources of the OS.

The application that runs on the APP MCU executes communication between the BLE MCU and BLE services via rBLE_Host. The APP MCU and BLE MCU are physically connected via UART or CSI or IIC, and communication is executed using RSCIP (Renesas Serial Communication Interface Protocol) under the control of rBLE_Host.

BLE software in the Embedded configuration runs on only a single chip, the BLE MCU (RL78/G1D). The software to be prepared by the user is only the application block and it should be implemented on the BLE MCU.

Table 5-1 gives an overview of the software blocks.

Table 5-1 Overview of Software Blocks

Block Name	Description
Application	Software the user needs to prepare
OS	Operating system the user needs to prepare
rBLE_Host	Builds command packets for MDM_APPL and analyzes event packets from MDM_APPL, enabling the issuance of rBLE APIs from the application.
Serial Communication driver (APP MCU)	Performs communication with the BLE MCU through UART or CSI or IIC. RSCIP is used as the communication protocol. Note: The user needs to prepare this driver.
Serial Communication driver (BLE MCU)	Performs communication with the APP MCU through UART or CSI or IIC. RSCIP is used as the communication protocol.
MDM APPL	Analyzes command packets from rBLE_Host and builds event packets for rBLE_Host, enabling the use of BLE stack services via rBLE_Core.
rBLE_Core	Provides an interface with the upstream modules for using the services of the main BLE stack (the part from the profile layer to the RF driver).
Profile layer	Main BLE stack Host stack : SM, L2CAP, GAP, and GATT Controller stack : LL and HCI
Host stack	
Controller stack	
RWKE (Renesas Wireless Kernel Extension)	Provides the basic functionality used commonly with the other modules and manages the entire BLE MCU.

5.2 rBLE API

The BLE software provides an API (rBLE API) that allows the application to use the services of the BLE stack on the BLE MCU from the application.

The Bluetooth layers that can be accessed by the APIs provided by the rBLE API are shown in the figure below.

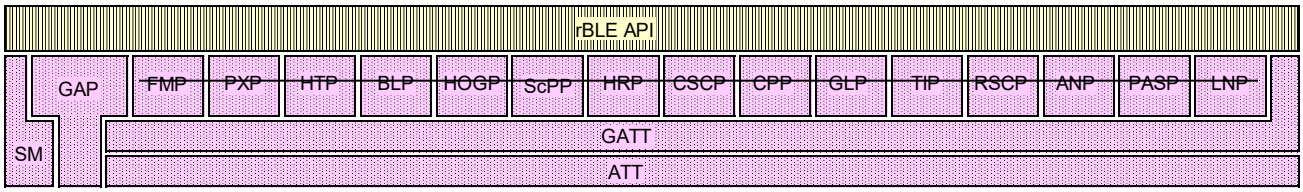


Figure 5-2 rBLE API and BLE Stack

Due to the deprecation and withdrawal plan of the profile version by Bluetooth SIG, each profile has been obsoleted because product registration using the profile (Note) supported by the BLE protocol stack is no longer possible.

For product registration, refer to "Bluetooth LE microcomputer/module Bluetooth qualification acquisition application note" (R01AN3177).

Note: FMP(Find Me), PXP(Proximity), HTP(Health Thermometer), BLP(Blood Pressure), HOGP(HID Over GATT), ScPP(Scan Parameters), HRP(Heart Rate), CSCP(Cycling Speed and Cadence), CPP(Cycling Power), GLP(Glucose), TIP (Time Profile), RSCP(Running Speed and Cadence), ANP(Alert Notification), PASP(Phone Alert Status), LNP(Location and Navigation)

The features supported by the various layers are listed in Table 5-2. For details about the various features, see 7 *Description of Features*.

Table 5-2 Bluetooth Features Supported by the rBLE API

Layer	Description	Supported Features
GAP (Generic Access Profile)	Executes access procedures according to the link management and security requirements for processes such as device discovery and peer device connection and disconnection.	• Four GAP roles (Central, Peripheral, Broadcaster, and Observer) • Broadcast and Scan • Discovery, Connection, and Bonding modes, and procedures • Security mode • Connection and disconnection of a link • Changing the connection parameters • Random and static addresses • Privacy feature
SM (Security Manager)	Executes pairing between two devices, communication encryption, and data signing to ensure security. Also executes information exchange between devices as needed for the above.	• Pairing procedure • Pairing algorithms (Passkey Entry, Just Works, OOB) • Pairing and key generation • Key distribution • Security implemented by authentication, encryption, and data signing

The rBLE API includes an interface for Direct Test Mode for performing RF evaluation of the BLE MCU, allowing transmission testing and reception testing.

5.3 RL78/G1D Hardware Resources used by the BLE Software

The BLE software uses the following RL78/G1D hardware resources. Therefor user program cannot use these hardware resources.

Table 5-3 Hardware Resources used by BLE Software

H/W Resources		Configuration		Purpose / Usage
		Modem	Embedded	
Data Flash Memory		Used	Used	Used to store the BD address.
12-bit Interval Timer		Used	Used	Used by Peak current consumption notification function, or Used by monitoring for RF slow clock for internal oscillation circuit Note (3)
Timer Array Unit		TI07/TO07	Not Used	Used by CSI or IIC Driver (Unit0 Channel7)
Clock Output/Buzzer Output		PCLBUZ0	PCLBUZ0	Used for clock output to RF transceiver (When not using RF slow clock for internal oscillation circuit.) Note (4)
Port Function		P21(output) P30(input)	Not Used	P21: Used by CSI or IIC Driver (SDIR or REQ signal) P30: Used by WAKEUP Driver for UART or CSI
Serial Interface (Serial Array Unit)		UART0 UART1	Not Used	Used by UART Driver
		CSI00 CSI20 CSI21	 CSI21	CSI00 or CSI20: Note (1) CSI21 : Note (2)
Serial Interface IICA		IICA0	Not Used	Used by IIC Driver
Multiplier, Divider/Multiply Accumulator		Used	Used	Used by the C Compiler Set the compiler option to use this hardware (CS+ for CA,CX and e^2 studio/CS+ for CC)
DMA Controller		DMA0, DMA1 DMA2, DMA3	 DMA2, DMA3	DMA0, DMA1 : Note (1) DMA2, DMA3 : Note (2)
Interrupt	External Pin	INTRF INTP3	INTRF	INTRF : Note (2) INTP3 : Note (1), used for WAKEUP signal
	DMA	INTDMA0 INTDMA1 INTDMA2 INTDMA3	INTDMA2 INTDMA3	INTDMA0, INTDMA1 : Note (1) INTDAM2, INTDMA3 : Note (2)
	Serial Array Unit	INTCSImn INTSTm INTSRm INTSREm	Not Used	INTCSImn : Note (1), for CSI (mn=00, 20) INTSTm, INTSRm, INTSREm : Note (1), for UART (m=0, 1)
	Serial Interface IICA	INTIICA0	Not Used	INTIICA0: Used by IIC Driver
	12-bit Interval Timer	INTIT	INTIT	Used by Peak current consumption notification function, or Used by monitoring for RF slow clock for internal oscillation circuit. Note (3)

Note (1): This hardware resource is used for communication interface between APP-MCU and BLE-MCU through UART0, UART1 or CSI00, CSI20 (in modem configuration)
Note (2): This hardware resource is used for communication interface between MCU and RF Transceiver
Note (3): User application can use the 12-bit interval timer when not using the Peak current consumption notification function and RF slow clock for internal oscillation circuit.
Note (4): User application can use the Clock Output/Buzzer Output when using RF slow clock for internal oscillation circuit.

5.4 Serial Communication in Modem Configuration

APP-MCU and BLE-MCU communicates through serial interface (UART or CSI or IIC), using the RSCIP (Renesas Serial Communication Interface Protocol) as the protocol.

The RSCIP is based on the SLIP (Serial Line Internet Protocol) defined in RFC 1055, and is extended. The RSCIP ensures the reliability of data communication using error recovery capabilities by retransmission.

The RSCIP driver, which is a part of rBLE_Host, performs protocol processing and control of the serial driver. The RSCIP driver provides packet based communication.

Refer to the Application Note: rBLE Command Specification for more details.

Serial communication provides communication through UART or CSI or IIC interface, and the following connection methods are available.

In addition, there is a connection method that cannot be selected in UART1 and CSI20.

Table 5-4 Connection methods for serial communication

Serial Interface	Connection method	Possible channel
UART	2-wire	UART0
	3-wire	UART0, UART1
	2-wire with branch	UART0, UART1
CSI	4-wire	CSI00
	5-wire	CSI00, CSI20
IIC	3-wire	IICA0

UART communication operates in the following settings.

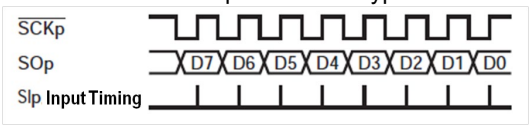
Table 5-5 Settings of UART

Setting	Setting value
Baud rate	(2-wire) 4,800bps ~ 250kbps (*1) (3-wire, 2-wire with branch) 4,800bps ~ 250kbps
Data length	8bit
Parity	None
Stop bit	1bit
Flow control	None

(*1) If you select the 2-wire connection method and set the baud rate greater than 4,800 bps, the Sleep function is disabled. The Sleep function is always enabled in other connection method. The Sleep function realizes the low current consumption of BLE MCU.

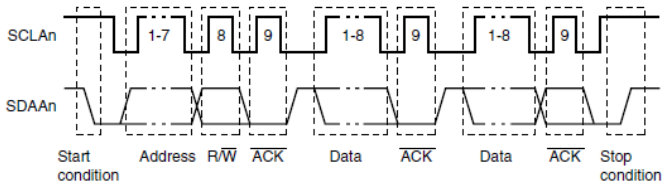
CSI communication operates in the following settings.

Table 5-6 Setting of CSI

Setting	Setting value
Communication	peer-to-peer, master-slave half-duplex, synchronous clock, master clock supply
Role	APP MCU : master BLE MCU : slave
Baud rate	4800bps ~ 250kbps (4-wire, 5-wire)
Data Length	8bit
Data clock phase	RL78/G1D phase mode type 1 

IIC communication operates in the following settings.

Table 5-7 Setting of IIC

Setting	Setting value
Communication	IIC bus mode half-duplex, synchronous clock, master clock supply
Role	APP MCU : master BLE MCU : slave
Transfer clock	100kbps ~ 400kbps
Data Length	8bit
Communication format	IIC Bus Serial Data Transfer format 

Note

* The serial communication driver on the APP MCU needs to be prepared by customer.
* With the requirements of the serial communication driver, refer to the Bluetooth Low Energy Protocol Stack Application Note: Sample Program.

5.4.1 UART 2-wire Connection

In this connection method, the APP MCU and the BLE MCU communicate using two signal lines (TxD and RxD).
If the baud rate greater than 4,800 bps is used, the Sleep function to reduce power consumption is disabled.
There is no handshake operation.

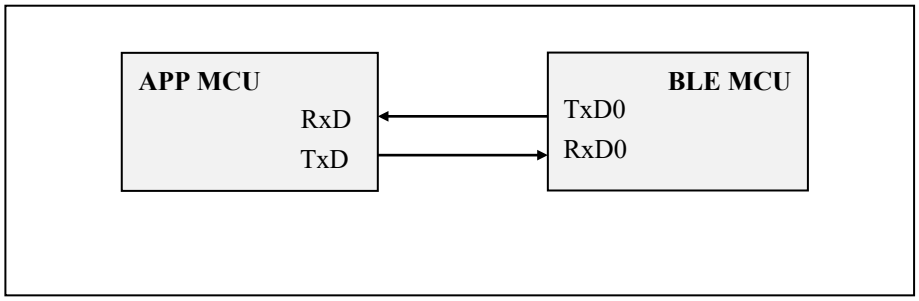


Figure 5-3 UART 2-wire connection method

BLE MCU Pin Name	Direction	Function
TxD0	BLE -> APP	Serial Output Data Signal
RxD0	APP -> BLE	serial input data signal

- In the timing chart described from now on, the terminal name of the BLE MCU side is described.

(1) Transmit Operation of APP MCU

This section shows the transmission sequence including rBLE_Host and serial communication driver function calls.

[At the start of transmission]
When the rBLE_Host calls the transmit function, the serial communication driver starts the transmission operation of the RSCIP packet.

[At the end of transmission]
The serial communications driver notifies rBLE_Host the transmission completion by calling the transmission completion notification function when the RSCIP packet transmission is completed.

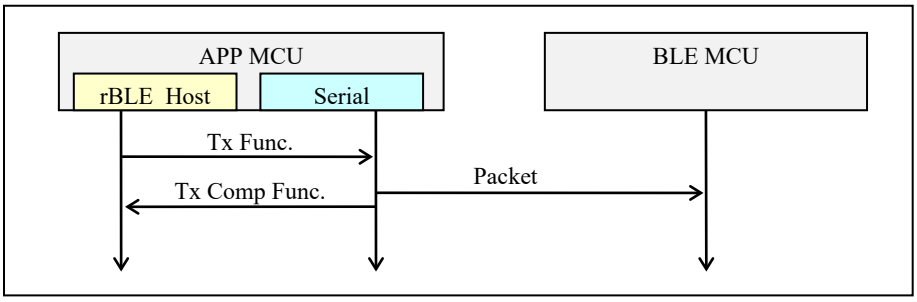


Figure 5-4 Transmit Sequence (APP MCU)

(2) APP MCU Receive Operation

The reception sequence including rBLE_Host and serial communications driver's function calls is shown. When one RSCIP packet is received, rBLE_Host calls the reception function two or more times because the RSCIP packet is variable-length.

[When beginning to receive it] RBLE_Host calls the reception function. As a result, the serial communications driver begins the reception operation of the RSCIP packet, and waits for the data reception.

[When the reception ends the packet on the way] The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after the reception ends. RBLE_Host calls the reception function again, and the serial communications driver restarts the reception.

[When the reception of the entire packet ends] The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after the reception ends. RBLE_Host calls the reception function again, and waits for the following RSCIP packet reception.

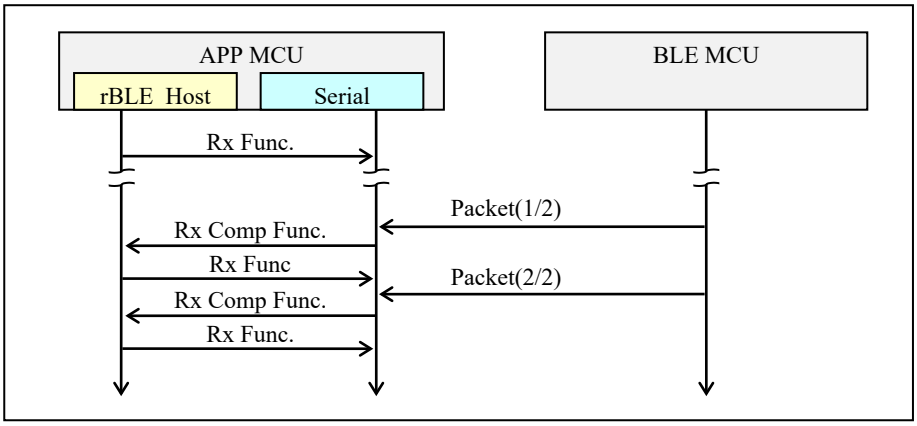


Figure 5-5 Receive Sequence (APP MCU)

5.4.2 UART 3-wire Connection

In this connected method, APP MCU and BLE MCU communicate by using control signal line WAKEUP to make BLE MCU get up when APP MCU in addition to TxD that is the data signal line of UART as shown in the following and RxD transmits data.

When transmitting from APP MCU, it is necessary to do handshaking though the full duplex transmission is possible. This is operation necessary to confirm BLE MCU completes the preparation for the reception. Moreover, please observe by the time-out to do a reliable communication at handshaking, and execute handshaking again when you generate the time-out.

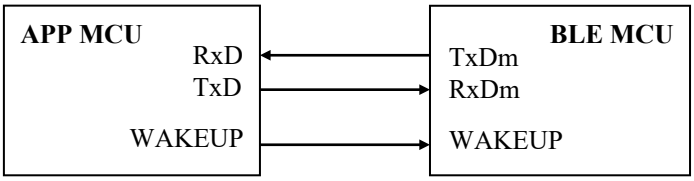


Figure 5-6 UART 3-wire connection

BLE MCU Pin Name	Direction	Function
TxDm (m=0,1)	BLE MCU-> APP MCU	Serial Output Data Signal
RxDm (m=0,1)	APP MCU-> BLE MCU	Serial Input Data Signal
WAKEUP(P30/INTP3) - Low Active	APP MCU-> BLE MCU	External Trigger Input Signal for Wakeup APP MCU is set at an active level at the transmission request. ACK byte (0x88) reception or data reception from BLE MCU is waited for, and it returns it to an inactive level.

- In the timing chart described from now on, the terminal name of the BLE MCU side is described.

(1) Transmit Operation (APP MCU)

The handshaking procedure when APP MCU transmits the RSCIP packet to BLE MCU is following T3.

T1: APP MCU makes the WAKEUP signal an active level for the transmission request.
T2: APP MCU detects ACK byte (0x88) from BLE MCU or the RSCIP packet by one byte.
T3: APP MCU makes the WAKEUP signal an inactive level.
T4: APP MCU transmits the RSCIP packet.

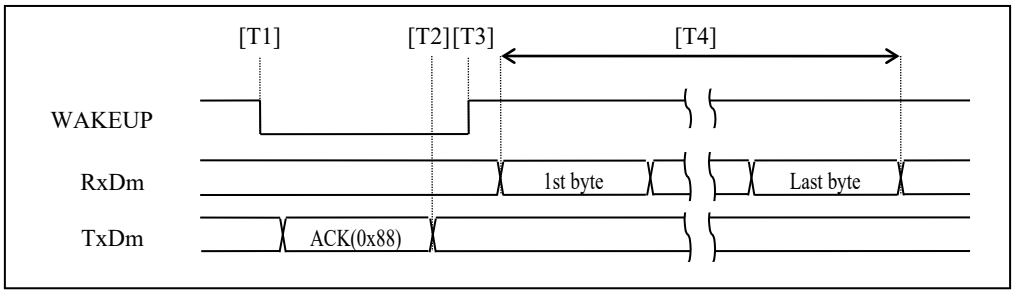


Figure 5-7 Transmit timing chart (APP MCU)

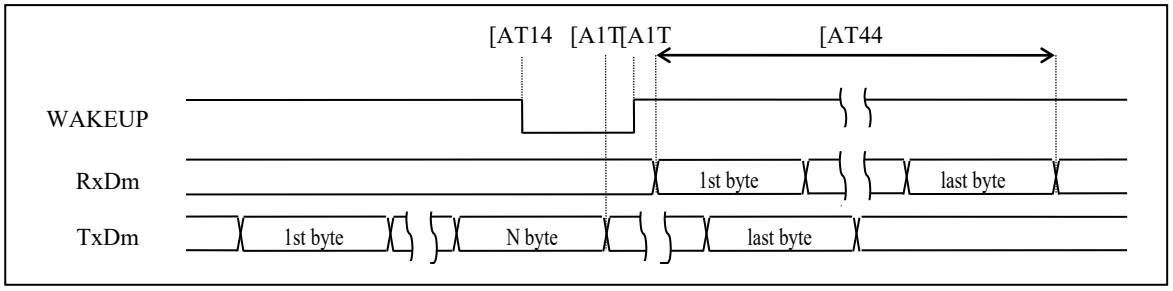


Figure 5-8 Transmit timing chart (APP MCU) (while BLE MCU transmitting)

After the transmission request, the serial communications driver begins the time-out watch. When the time-out is generated, the serial communications driver is T1 that returns the WAKEUP signal to an inactive level for the re-transmission demand once, and makes to an active level again. The recommended value at the timeout period is assumed to be 5msec.

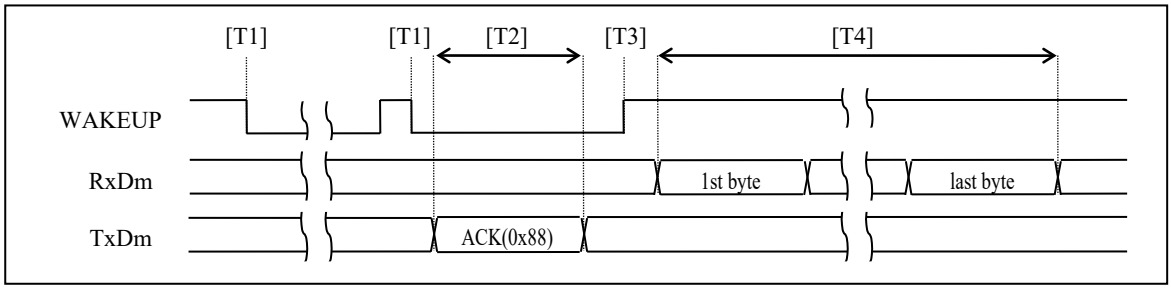


Figure 5-9 Transmit timing chart (APP MCU) (timeout occurs)

The transmission sequence including rBLE_Host and serial communications driver's function calls is shown.

[When beginning to transmit]: The serial communications driver is T1 according to the call of rBLE_Host of the transmission function that begins the transmission operation of the RSCIP packet, and makes the WAKEUP signal an active level for the transmission request.

[When the transmission ends]: The serial communications driver notifies rBLE_Host the transmission completion by calling the transmission completion notification function when RSCIP packet transmission T1- T4 is completed.

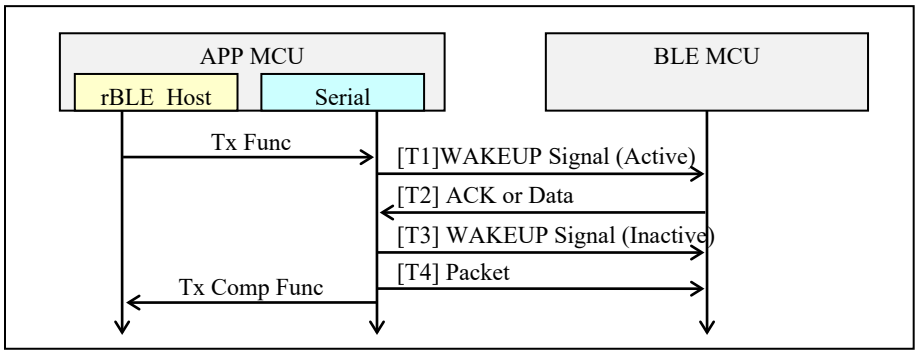


Figure 5-10 Transmit Sequence (APP MCU)

(2) Receive Operation (APP MCU)

The reception sequence including rBLE_Host and serial communications driver's function calls is shown. When one RSCIP packet is received, rBLE_Host calls the reception function two or more times because the RSCIP packet is variable-length.

[When beginning to receive it] RBLE_Host calls the reception function. As a result, the serial communications driver begins the reception operation of the RSCIP packet, and waits for the data reception.

[When the reception ends the packet on the way] The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after the reception ends. RBLE_Host calls the reception function again, and the serial communications driver restarts the reception.

[When the reception of the entire packet ends] The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after the reception ends. RBLE_Host calls the

reception function again, and waits for the following RSCIP packet reception.

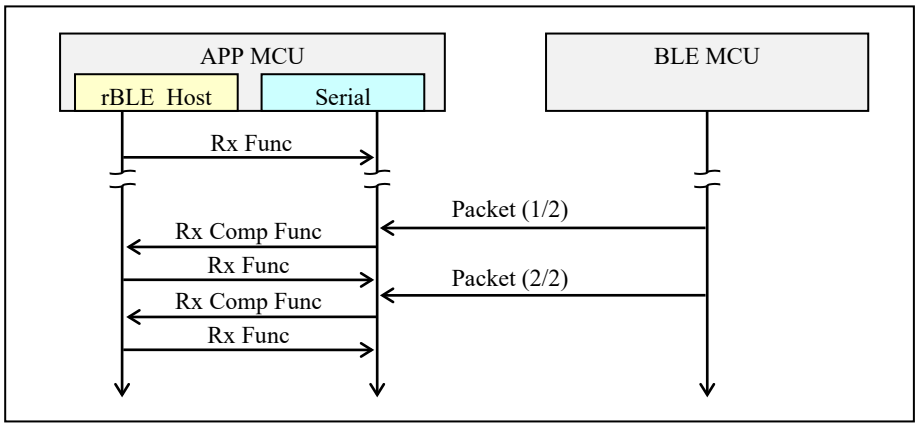


Figure 5-11 Receive Sequence (APP MCU)

5.4.3 UART 2-wire with Branch Connection

TxD of APP MCU diverges, is connected with WAKEUP of BLE MCU so that APP MCU and BLE MCU may make BLE MCU get up when APP MCU in addition to TxD that is the data signal line of UART as shown in the following and RxD transmits data in this connected method, and it communicates.

When transmitting from APP MCU, it is necessary to do handshaking though the full duplex transmission is possible. This is operation necessary to confirm BLE MCU completes the preparation for the reception. Moreover, please observe by the time-out to do a reliable communication at handshaking, and execute handshaking again when you generate the time-out.

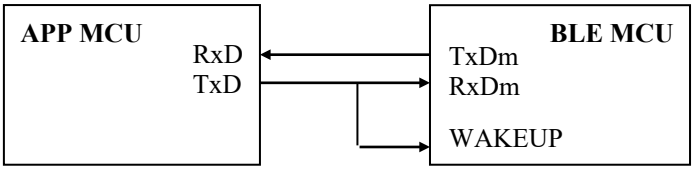


Figure 5-12 UART 2-wire with branch connection

BLE MCU Pin Name	Direction	Function
TxDm (m=0,1)	BLE MCU-> APP MCU	Serial Output Data Signal
RxDm (m=0,1)	APP MCU-> BLE MCU	Serial Input Data Signal
WAKEUP(P30/INTP3) - Low Active	APP MCU-> BLE MCU	External Trigger Input Signal for Wakeup APP MCU is set at an active level at the transmission request. ACK byte (0x88) reception or data reception from BLE MCU is waited for, and it returns it to an inactive level.

- In the timing chart described from now on, the terminal name of the BLE MCU side is described.

(1) Transmit Operation (APP MCU)

The handshaking procedure when APP MCU transmits the RSCIP packet to BLE MCU is following T3.

T1: APP MCU transmits REQ byte (0xC0) for the transmission request.
T2: APP MCU detects ACK byte (0x88) from BLE MCU or the RSCIP packet by one byte.
T3: APP MCU transmits the RSCIP packet.

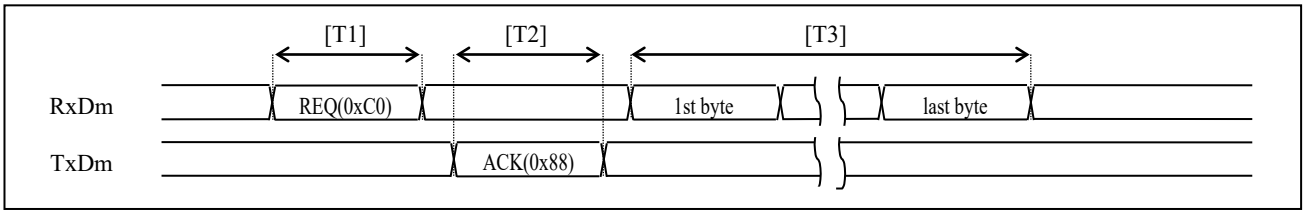


Figure 5-13 Transmit timing chart (APP MCU)

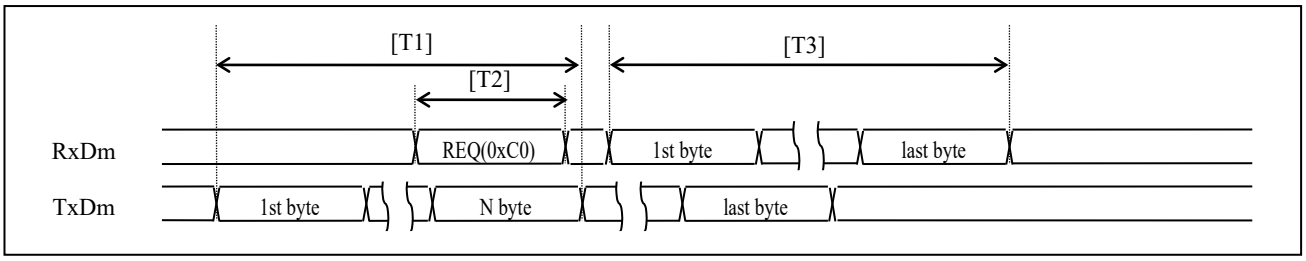


Figure 5-14 Transmit timing chart (APP MCU) (While BLE MCU transmitting)

After the transmission request, the serial communications driver begins the time-out watch. When the time-out is generated, the serial communications driver is T1 that transmits the REQ byte for the re-transmission demand. The recommended value at the timeout period is assumed to be 5msec.

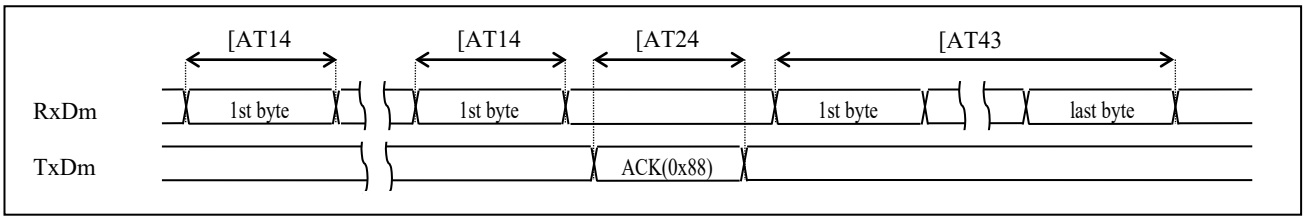


Figure 5-15 Transmit timing chart (APP MCU) (Timeout occurs)

The transmission sequence including rBLE_Host and serial communications driver's function calls is shown.

[When beginning to transmit]: The serial communications driver is T1 according to the call of rBLE_Host of the transmission function that begins the transmission operation of the RSCIP packet, and transmits the REQ byte for the transmission request.

[When the transmission ends]: The serial communications driver notifies rBLE_Host the transmission completion by calling the transmission completion notification function when RSCIP packet transmission T1- T3 is completed.

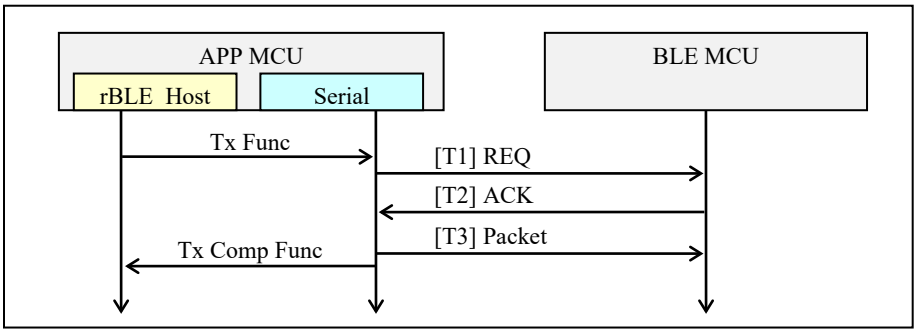


Figure 5-16 Transmit Sequence (APP MCU)

(2) Receive Operation (APP MCU)

The reception sequence including rBLE_Host and serial communications driver's function calls is shown. When one RSCIP packet is received, rBLE_Host calls the reception function two or more times because the RSCIP packet is variable-length.

[When beginning to receive it] RBLE_Host calls the reception function. As a result, the serial communications driver begins the reception operation of the RSCIP packet, and waits for the data reception.

[When the reception ends the packet on the way] The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after the reception ends. RBLE_Host calls the reception function again, and the serial communications driver restarts the reception.

[When the reception of the entire packet ends] The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after the reception ends. RBLE_Host calls the reception function again, and waits for the following RSCIP packet reception.

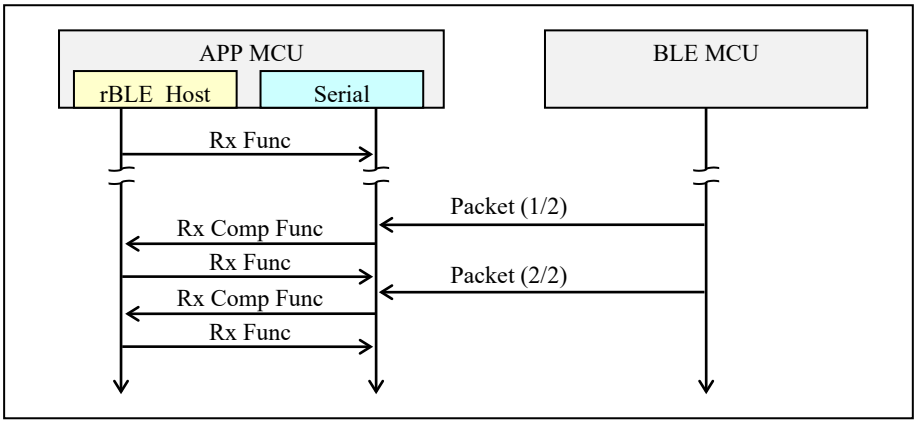


Figure 5-17 Receive Sequence (APP MCU)

5.4.4 CSI 4-wire Connection

In this connected method, APP MCU and BLE MCU communicate by using control signal line SDIR to control the direction of the communication and the communication timing of APP MCU and BLE MCU in addition to SO that is the data signal line of CSI as shown in the following, SI, and SCK.

The communication is half duplex, and when transmitting or receiving it, it is necessary to do handshaking. It is operation because it notifies APP MCU the transmission request from BLE MCU so that this may confirm BLE MCU completes the preparation for the reception or the transmission necessary to fix the direction of the communication in the half duplex transmission. Moreover, please observe by the time-out to do a reliable communication at handshaking, and execute handshaking again when you generate the time-out.

Time-out of reply from APP MCU is prepared on BLE MCU. For time-out, Timer Array Unit is used.

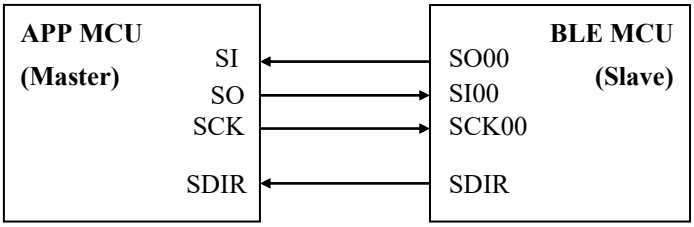


Figure 5-18 CSI 4-wire connection

BLE MCU Pin Name	Direction	Function
SO00	BLE MCU-> APP MCU	Serial Output Data Signal (MISO : Master-Input-Slave-Output)
SI00	APP MCU-> BLE MCU	Serial Input Data Signal (MOSI : Master-Output-Slave-Input)
SCK00	APP MCU-> BLE MCU	Data Communication Timing Clock Signal
SDIR(P21)	BLE MCU-> APP MCU	Communication Direction Control Signal / Response Control Signal Low: The direction where data is forwarded is BLE MCU->APP MCU High: The direction where data is forwarded is APP MCU->BLE MCU Pulse (High->Low->High): The pulse width is time of the BLE MCU operation clock ×4 The transmission request from communication permission response BLE MCU (From APP MCU, except for the transmission completion of one Packet head byte) from BLE MCU. (At Packet head one byte completion of transmission from APP MCU) Communication permission response from BLE MCU (From APP MCU, except for the transmission completion of one Packet head byte) Transmission request from BLE MCU

- In the timing chart described from now on, the terminal name of the BLE MCU side is described.

(1) Transmit Operation (APP MCU)

The handshaking procedure when APP MCU transmits the RSCIP packet to BLE MCU is following T3.

T1: APP MCU transmits one RSCIP packet head byte for the transmission request, and waits for the pulse of the SDIR signal.

- When the time-out is generated by the pulse waiting about the SDIR signal, it is necessary to retransmit one head byte.

T2: APP MCU detects the pulse of the SDIR signal of the communication permission response.

T3: APP MCU continuously transmits everything from the 2nd byte to the final byte of the RSCIP packet.

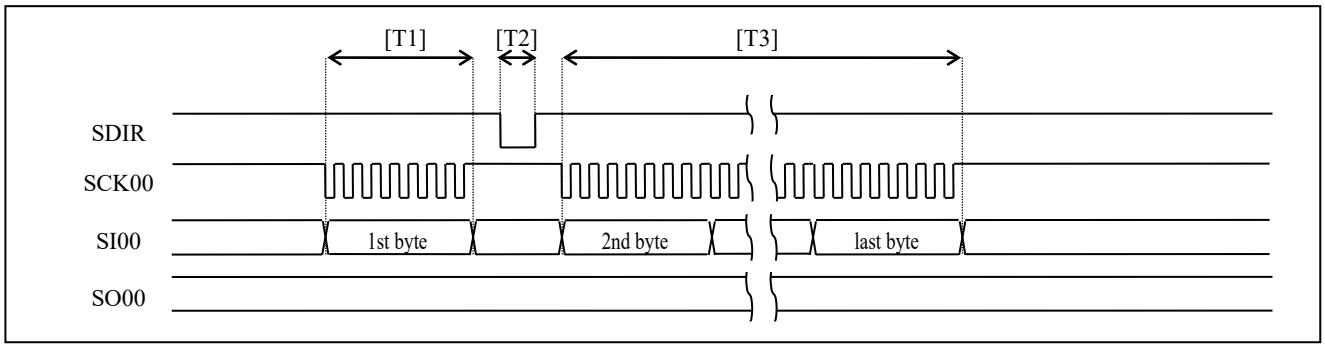


Figure 5-19 Transmit timing chart (APP MCU)

After the transmission request, the serial communications driver begins the time-out watch. When the time-out is generated, the serial communications driver is T1 that transmits one head byte of the RSCIP packet again for the re-transmission demand. The recommended value at the timeout period is assumed to be 5msec.

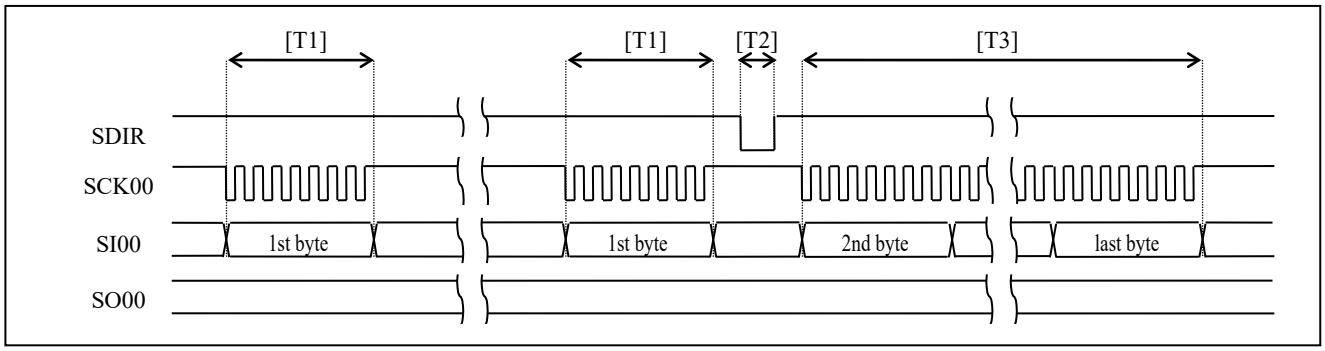


Figure 5-20 Transmit timing chart (APP MCU) (Timeout occurs)

The transmission sequence including rBLE_Host and serial communications driver's function calls is shown.

[When beginning to transmit]: The serial communications driver is T1 according to the call of rBLE_Host of the transmission function that begins the transmission operation of the RSCIP packet.

[When the transmission ends]: The serial communications driver notifies rBLE_Host the transmission completion by calling the transmission completion notification function when RSCIP packet transmission T1- T3 is completed.

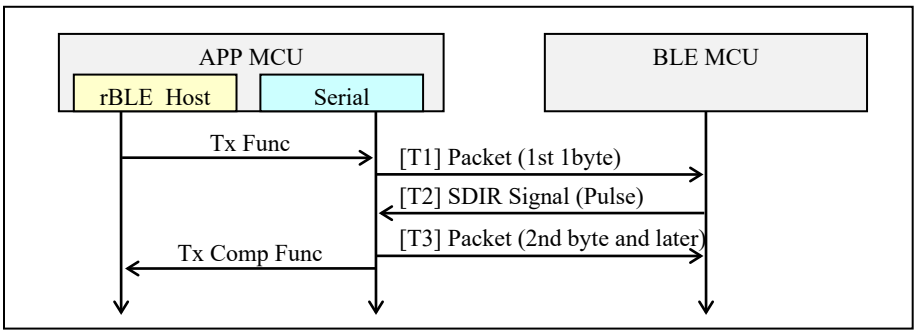


Figure 5-21 Transmit Sequence (APP MCU)

(2) Receive Operation (APP MCU)

The handshaking procedure when APP MCU receives the RSCIP packet from BLE MCU is following R5.

When R1 - R5 is executed, the transmission from APP MCU is assumed to be a prohibition for the half duplex transmission.

R1: APP MCU waits for the pulse of the SDIR signal of the transmission request.

R2: APP MCU transmits ACK byte (0x88) when the pulse of the SDIR signal is detected, and waits for Low of the SDIR signal.

R3: APP MCU detects Low of the SDIR signal.

R4: APP MCU supplies the clock, and receives the RSCIP packet.

R5: APP MCU detects High of the SDIR signal.

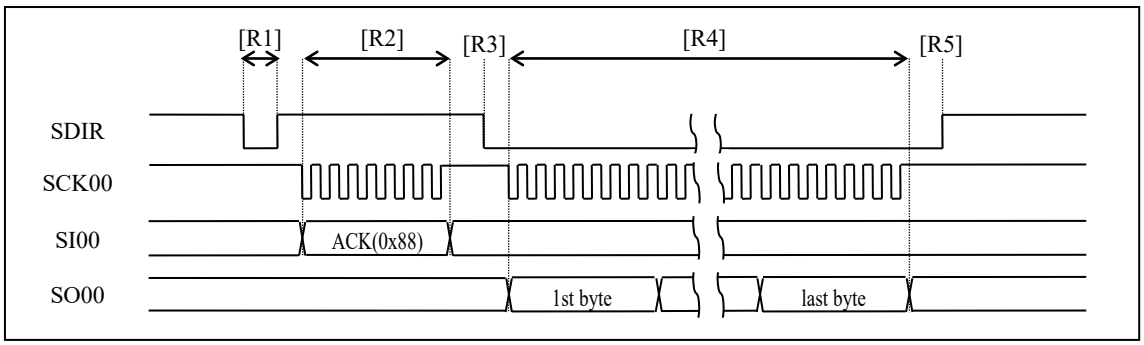


Figure 5-22 Receive timing chart (APP MCU)

When the transmission request of APP MCU collides with the transmission request of BLE MCU, BLE MCU postpones transmitting, and receives the RSCIP packet of APP MCU. After completing the RSCIP packet reception, BLE MCU outputs the pulse again by the SDIR signal.

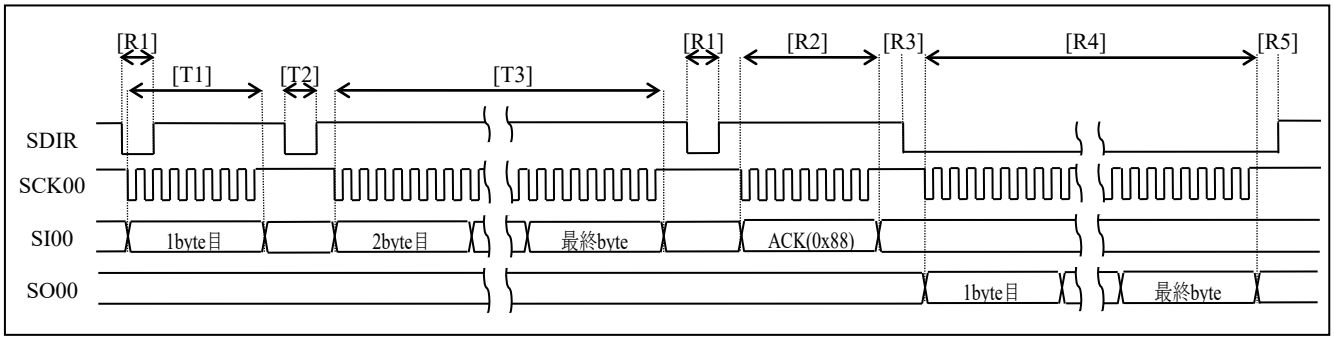


Figure 5-23 Transmit requested collision timing chart (APP MCU and BLE MCU)

The reception sequence including rBLE_Host and serial communications driver's function calls is shown. When one RSCIP packet is received, rBLE_Host calls the reception function two or more times because the RSCIP packet is variable-length.

[When beginning to receive it]: RBLE_Host calls the reception function. As a result, the serial communications driver is R1 that begins the reception operation of the RSCIP packet, and waits for the pulse of the SDIR signal.

[When the reception ends the packet on the way]: The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after ending R1 - R4, and rBLE_Host calls the

reception function again. The serial communications driver is R4 that confirms the reception state acquisition function, supplies the clock again while receiving the packet, and restarts the reception.

[When the reception of the entire packet ends]: The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after ending R4, and rBLE_Host calls the reception function again. The reception state acquisition function is confirmed, and if he or she is packet reception completion, the serial communications driver is R5 that waits for the SDIR signal to become High.

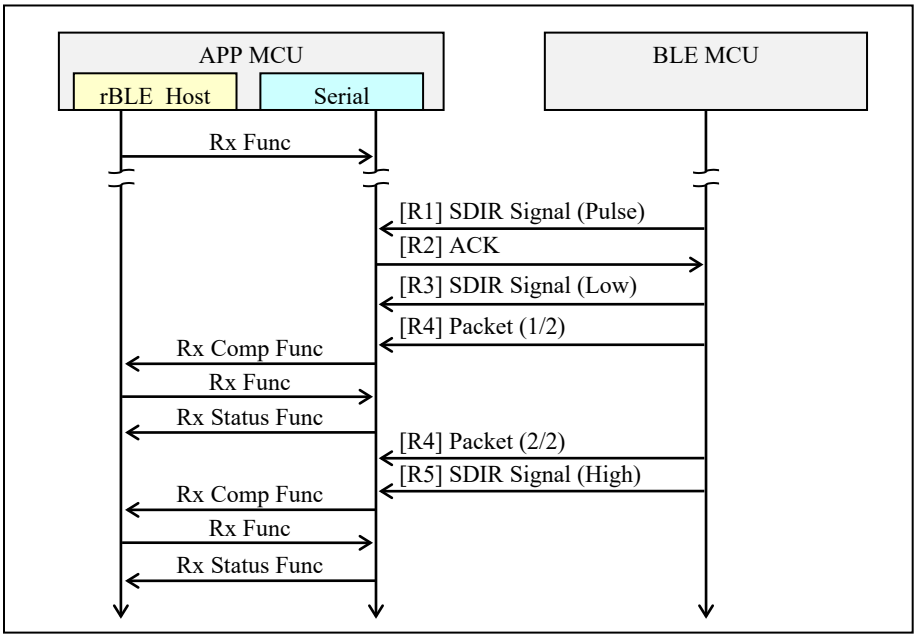


Figure 5-24 Receive Sequence (APP MCU)

5.4.5 CSI 5-wire Connection

In this connected method, APP MCU and BLE MCU communicate by using control signal line WAKEUP to make BLE MCU get up when control signal line SDIR and APP MCU to control the direction of the communication and the communication timing of APP MCU and BLE MCU in addition to SO that is the data signal line of CSI as shown in the following, SI, and SCK transmit data.

The communication is half duplex, and when transmitting or receiving it, it is necessary to do handshaking. It is operation because it notifies APP MCU the transmission request from BLE MCU so that this may confirm BLE MCU completes the preparation for the reception or the transmission necessary to fix the direction of the communication in the half duplex transmission. Moreover, please observe by the time-out to do a reliable communication at handshaking, and execute handshaking again when you generate the time-out.

Time-out of reply from APP MCU is prepared on BLE MCU. For time-out, Timer Array Unit is used.

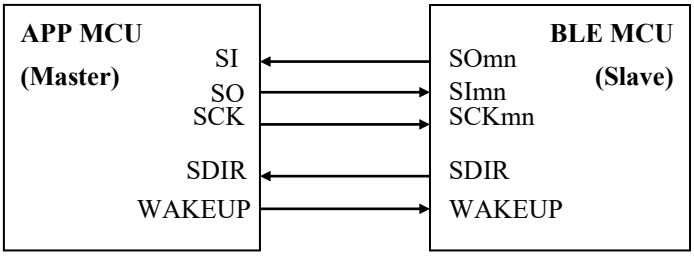


Figure 5-25 CSI 5-wire connection

BLE MCU Pin Name	Direction	Function
SOmn (mn=00,20)	BLE MCU-> APP MCU	Serial Output Data Signal (MISO : Master-Input-Slave-Output)
SImn (mn=00,20)	APP MCU-> BLE MCU	Serial Input Data Signal (MOSI : Master-Output-Slave-Input)
SCKmn (mn=00,20)	APP MCU-> BLE MCU	Data Communication Timing Clock Signal
SDIR(P21)	BLE MCU-> APP MCU	Communication Direction Control Signal / Response Control Signal Low : Data Direction is BLE MCU -> APP MCU High : Data Direction is APP MCU -> BLE MCU Pulse (High -> Low -> High) : pulse width = BLE MCU CPU clock x 4 (While WAKEUP signal is active) Clear-To-Send from BLE MCU (while WAKEUP signal is inactive) Request-To-Send from BLE MCU
WAKEUP(P30/INTP3) – Low Active	APP MCU-> BLE MCU	External Trigger Input Signal for Wakeup It sets it at an active level at the transmission request from APP MCU. - The response of SDIR from BLE MCU is waited for, and it returns it to an inactive level.

- In the timing chart described from now on, the terminal name of the BLE MCU side is described.

(1) Transmit Operation (APP MCU)

The handshaking procedure when APP MCU transmits the RSCIP packet to BLE MCU is following T4.

T1: APP MCU makes the WAKEUP signal an active level for the transmission request, and waits for the pulse of the SDIR signal.

- It is necessary to return the WAKEUP signal to an inactive level once when the time-out is generated by the pulse waiting about the SDIR signal, and to make it to an active level again.

T2: APP MCU detects the pulse of the SDIR signal of the communication permission response.

T3: The WAKEUP signal is returned to an inactive level.

T4: APP MCU continuously transmits everything from the first byte to the final byte of the RSCIP packet.

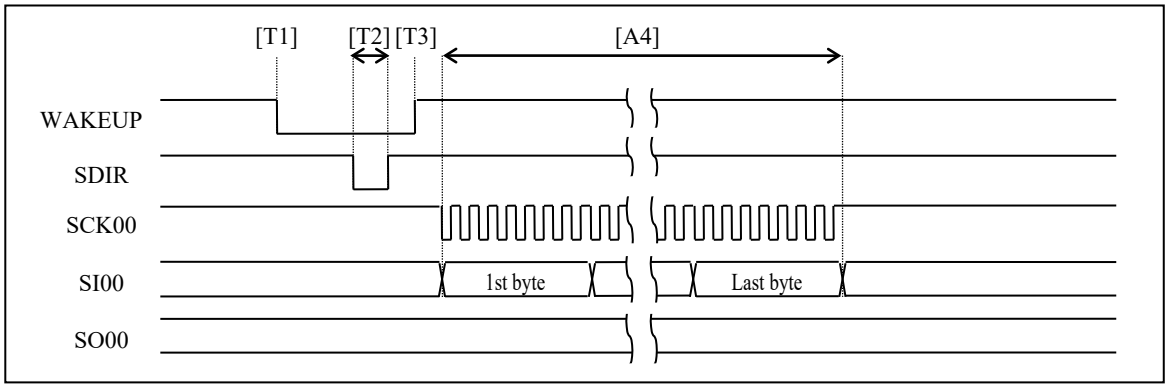


Figure 5-26 Transmit timing chart (APP MCU)

After the transmission request, the serial communications driver begins the time-out watch. When the time-out is generated, the serial communications driver is T1 that returns the WAKEUP signal to an inactive level for the re-transmission demand once, and outputs an active level again. The recommended value at the timeout period is assumed to be 5msec.

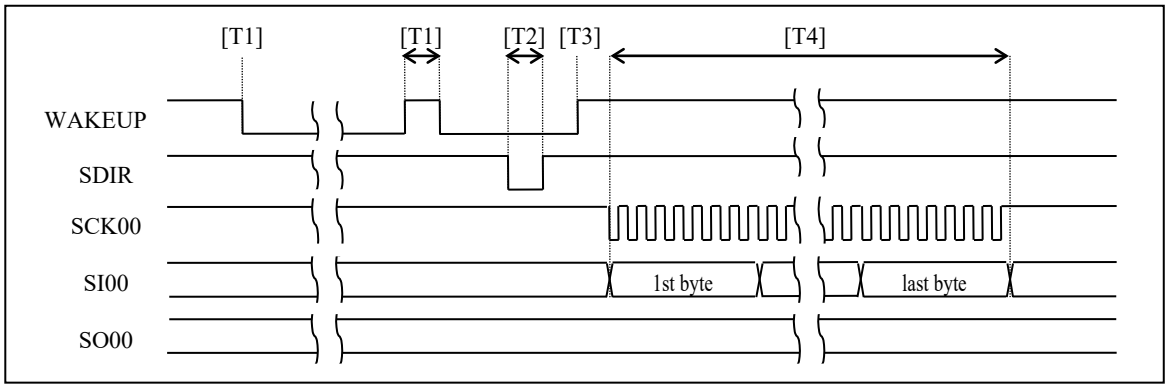


Figure 5-27 Transmit timing chart (APP MCU, timeout occurs)

The transmission sequence including rBLE_Host and serial communications driver's function calls is shown.

[When beginning to transmit]: The serial communications driver is T1 that begins because rBLE_Host calls the transmission function the transmission of the RSCIP packet, and makes the WAKEUP signal an active level.

[When the transmission ends]: The serial communications driver notifies the transmission completion by calling the transmission completion notification function of rBLE_Host when RSCIP packet transmission T1- T4 is completed.

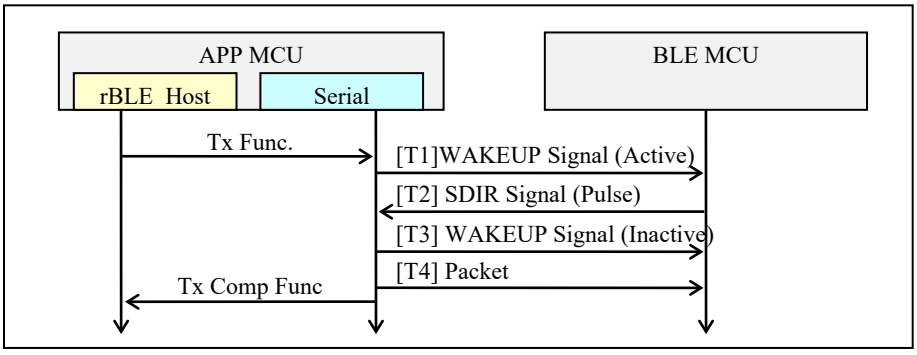


Figure 5-28 Transmit Sequence (APP MCU)

(2) Receive Operation (APP MCU)

The handshaking procedure when APP MCU receives the RSCIP packet from BLE MCU is following R5.

When R1 - R5 is executed, the transmission from APP MCU is assumed to be a prohibition for the half duplex transmission.

R1: APP MCU waits for the pulse of the SDIR signal of the transmission request.

R2: APP MCU transmits ACK byte (0x88) when the pulse of the SDIR signal is detected, and waits for Low of the SDIR signal.

R3: APP MCU detects Low of the SDIR signal.

R4: APP MCU supplies the clock, and receives the RSCIP packet.

R5: APP MCU detects High of the SDIR signal.

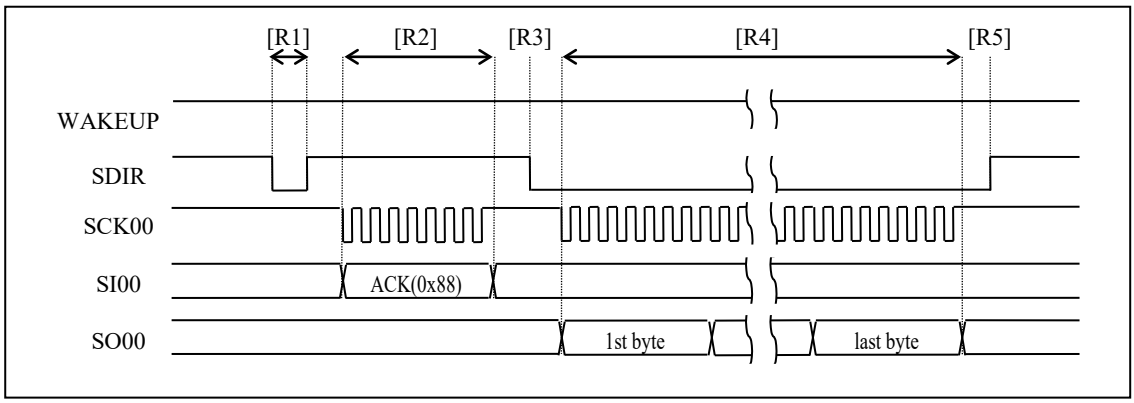


Figure 5-29 Receive timing chart (APP MCU)

When the transmission request of APP MCU collides with the transmission request of BLE MCU, BLE MCU postpones transmitting, and receives the RSCIP packet of APP MCU. After completing the RSCIP packet reception, BLE MCU outputs the pulse again by the SDIR signal.

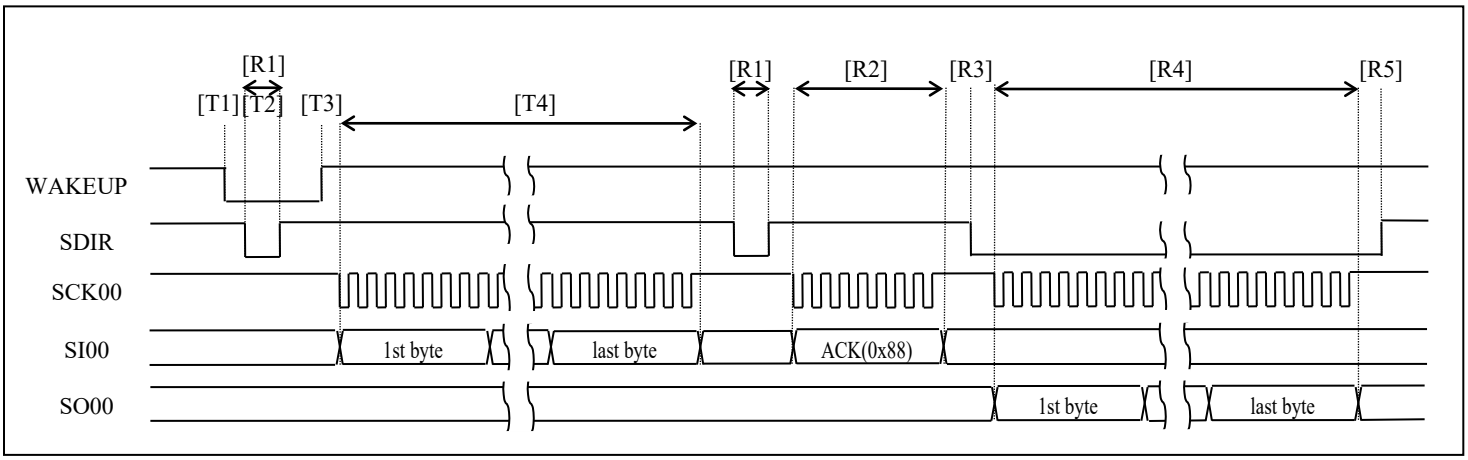


Figure 5-30 Transmit requested collision timing chart (APP MCU and BLE MCU)

The reception sequence including rBLE_Host and serial communications driver's function calls is shown. When one

RSCIP packet is received, rBLE_Host calls the reception function two or more times because the RSCIP packet is variable-length.

[When beginning to receive it]: RBLE_Host calls the reception function. As a result, the serial communications driver is R1 that begins the reception operation of the RSCIP packet, and waits for the pulse of the SDIR signal.

[When the reception ends the packet on the way]: The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after ending R1 - R4, and rBLE_Host calls the reception function again. The serial communications driver is R4 that confirms the reception state acquisition function, supplies the clock again while receiving the packet, and restarts the reception.

[When the reception of the entire packet ends]: The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after ending R4, and rBLE_Host calls the reception function again. The reception state acquisition function is confirmed, and if he or she is packet reception completion, the serial communications driver is R5 that waits for the SDIR signal to become High.

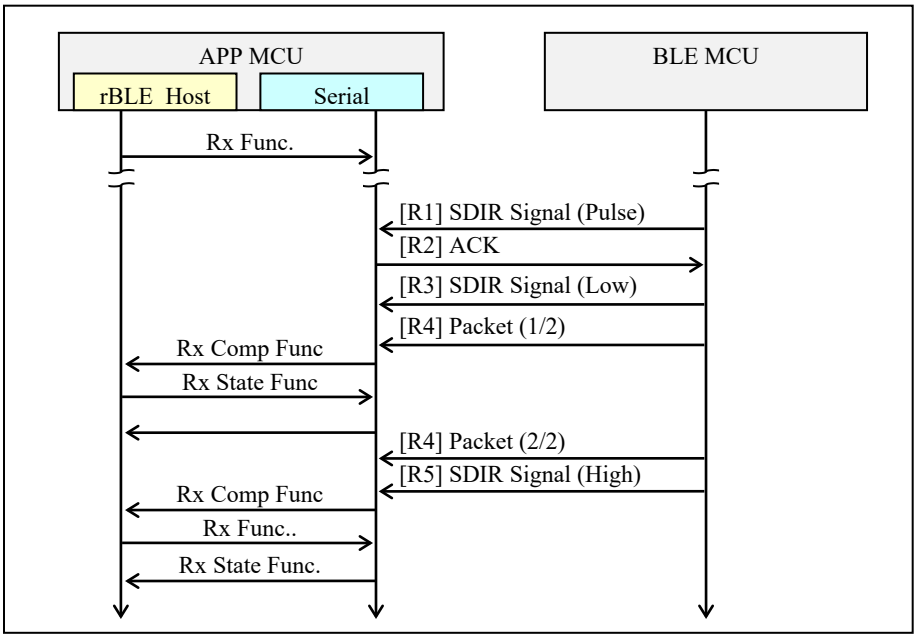


Figure 5-31 Receive Sequence (APP MCU)

5.4.6 IIC 3-wire Connection

In this connection method, APP MCU and BLE MCU use SDA and SCL signal line of the IIC as shown in the following. And IIC slave (BLE MCU) use the control signal line REQ for controlling the transmission request.

Since the SCL and SDA pins are used for open drain outputs, serial interface IIC requires pull-up resistors for the serial clock line and the serial data bus line.

The communication is half duplex. The transmission and reception by sending an IIC slave address of BLE MCU to identify the target. Moreover, please observe by the time-out to do a reliable communication, and execute communication again when you generate the time-out.

Time-out of reply from APP MCU is prepared on BLE MCU. For time-out, Timer Array Unit is used.

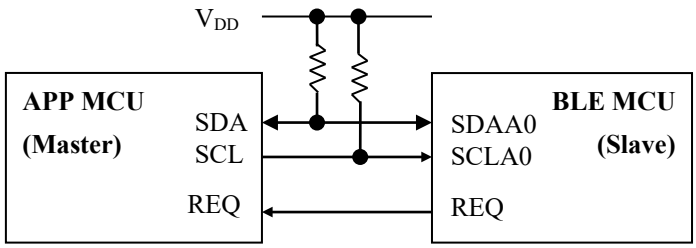


Figure 5-32 IIC 3-wire connection

BLE MCU Pin Name	Direction	Function
SDAA0	BLE MCU <-> APP MCU	IIC serial data bus
SCLA	APP MCU -> BLE MCU	IIC serial clock line
REQ(P21)	BLE MCU -> APP MCU	IIC slave data transmit request signal Low: data transmission request active High: data transmission request inactive

- In the timing chart described from now on, the terminal name of the BLE MCU side is described.

(1) Transmit Operation (APP MCU)

The communication procedure when APP MCU transmits the RSCIP packet to BLE MCU is following T4.

T1: APP MCU generates a start condition.

T2: APP MCU sends the slave address of BLE MCU (7 bits) and transfer direction Write (1 bit), detects the ACK of BLE MCU.

T3: APP MCU continuously transmits from the first byte to the last byte of the RSCIP packet.

T4: APP MCU generates stop condition and terminates communication.

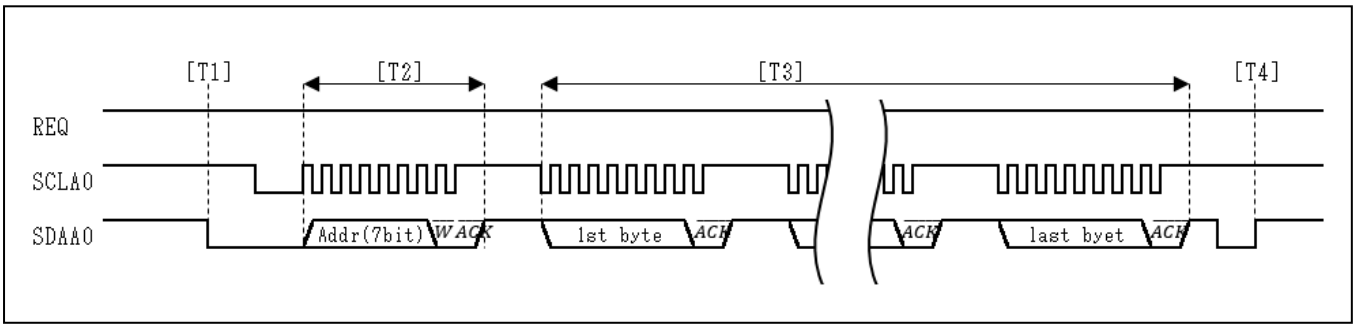


Figure 5-33 Transmit timing chart (APP MCU)

The transmission sequence including rBLE_Host and serial communications driver's function calls is shown.

[When beginning to transmit]: The serial communications driver is T1 according to the call of rBLE_Host of the transmission function that begins the transmission operation of the RSCIP packet.

[When the transmission ends]: The serial communications driver notifies rBLE_Host the transmission completion by

calling the transmission completion notification function when RSCIP packet transmission T1- T4 is completed.

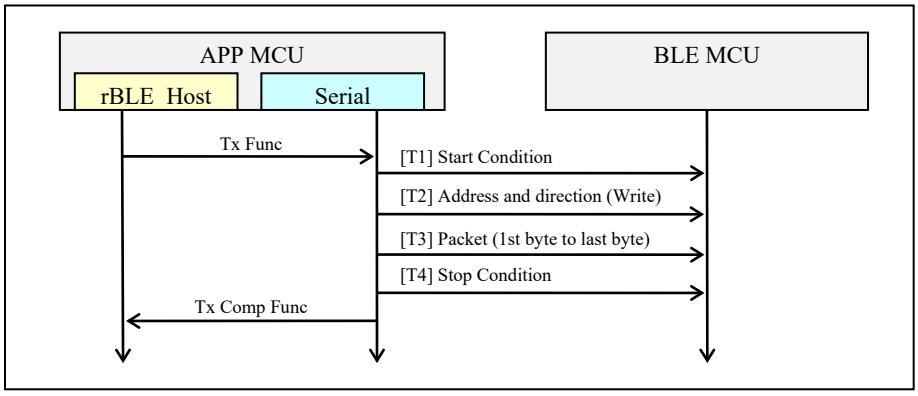


Figure 5-34 Transmit Sequence (APP MCU)

(2) Receive Operation (APP MCU)

The communication procedure when APP MCU receives the RSCIP packet from BLE MCU is following R7.

When R1 - R7 is executed, the transmission from APP MCU is assumed to be a prohibition for the half duplex transmission.

R1: APP MCU waits for the Low level of the REQ signal of the transmission request.

R2: APP MCU generates a start condition after detected falling edge of REQ signal.

R3: APP MCU sends the slave address of BLE MCU (7 bits) and transfer direction Read (1 bit), detects the ACK of BLE MCU.

R4: APP MCU generates the clock, receives the RSCIP packet, responds the ACK.

R5: BLE MCU change the REQ signal to High level before the last byte transmit.

R6: APP MCU responds a NACK indicating the last byte when detected High of the REQ signal.

R7: APP MCU generates stop condition and terminates communication.

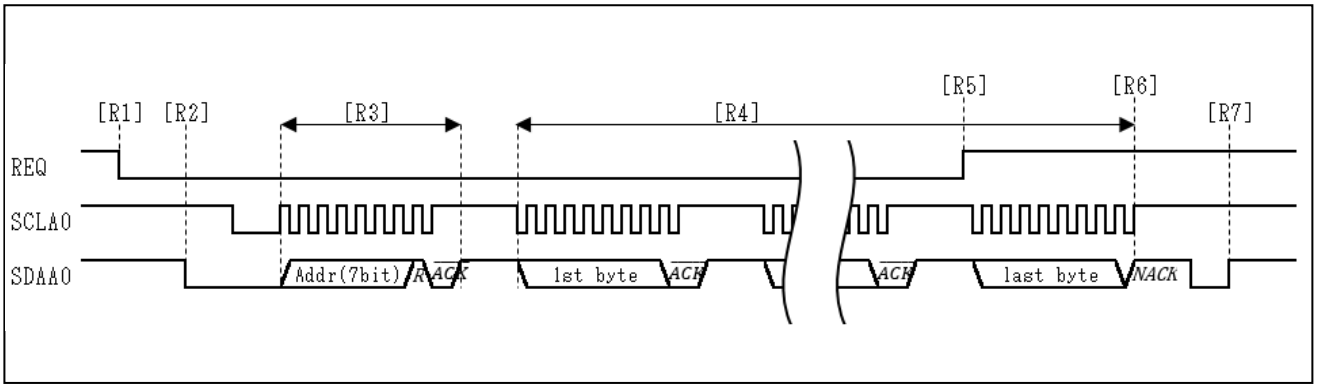


Figure 5-35 Receive timing chart (APP MCU)

When the transmission request of APP MCU collides with the transmission request of BLE MCU, BLE MCU postponed transmission by the REQ signal to High level [R8], and receives the RSCIP packet of APP MCU. After completing the RSCIP packet reception, BLE MCU outputs High level again by the REQ signal.

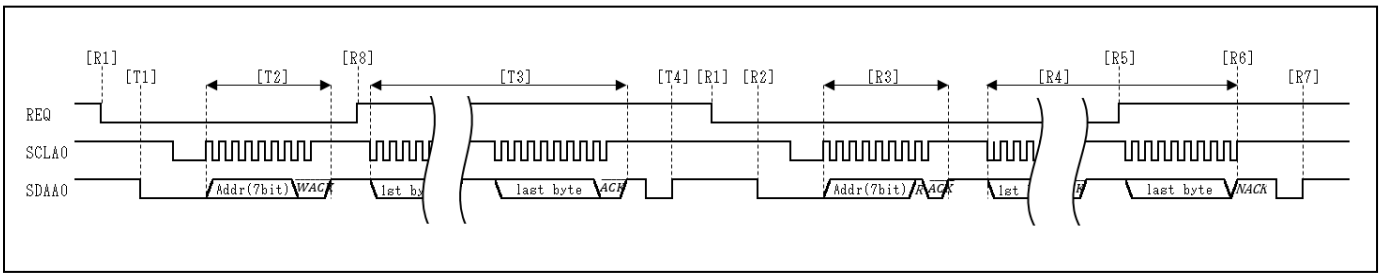


Figure 5-36 Transmit requested collision timing chart (APP MCU and BLE MCU)

The reception sequence including rBLE_Host and serial communications driver's function calls is shown. When one RSCIP packet is received, rBLE_Host calls the reception function two or more times because the RSCIP packet is variable-length.

[When beginning to receive it]: RBLE_Host calls the reception function. As a result, the serial communications driver is R1 that begins the reception operation of the RSCIP packet, and waits for the Low level of the SDIR signal.

[When the reception ends the packet on the way]: The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function during R4 when receive buffer full, and rBLE_Host calls the reception function again. The serial communications driver restarts the reception.

[When the reception of the entire packet ends]: The serial communications driver notifies rBLE_Host the reception completion by calling the reception completion notification function after ending R7, and rBLE_Host calls the reception function again. The serial communications driver waits the next Low level of REQ signal.

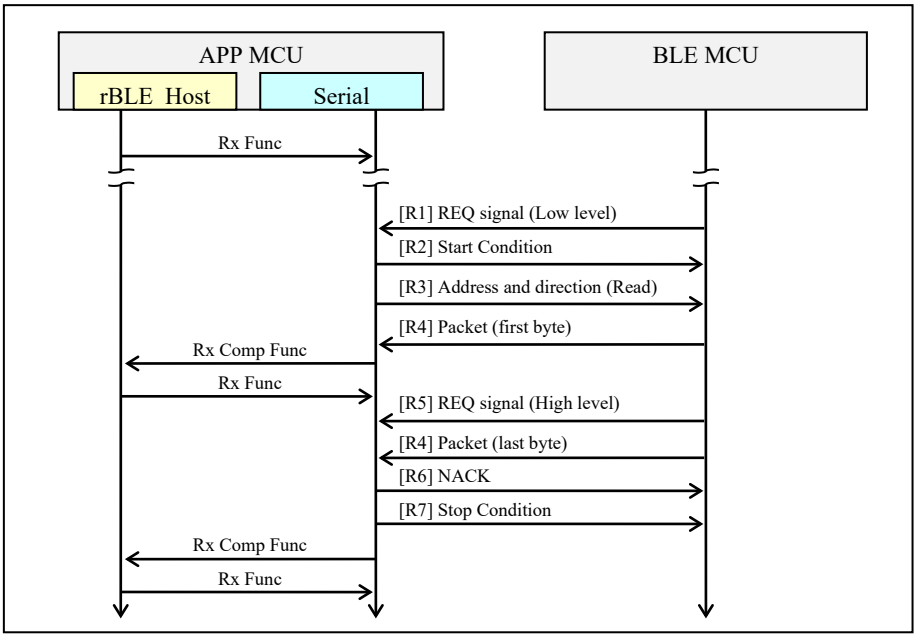


Figure 5-37 Receive Sequence (APP MCU)

5.5 Customer-specific information

BLE software to use the first 512 bytes of Code Flash last block as customer-specific information area. Customer-specific information to be referenced in the BLE software is shown in Table 5-8.

The BD address by writing to the area, it is possible to set different BD address for each BLE MCU. This BD address, since lower priority than BD address on the Data Flash, BD address changes after customer-specific information writing is possible.

Device name written in this area, it will be exposed to the peer device as the device name characteristic values of the GAP. When the valid device name isn't written in this area, the default value of the GATT database is used as GAP device name characteristic value.

Table 5-8 Customer-specific information to be referenced in the BLE software

Information	address	size	Notes
BD address	0x3FC00 Note (1)	6 bytes	Bluetooth Device Address - Address for identifying the devices
Device Name	0x3FC06 Note (1)	66 bytes	Bluetooth Device Name - User-friendly name for identifying the devices 0x3FC06: Length of Device Name (1 to 65) 0x3FC07 to 0x3FC48: Device Name (string of UTF-8)

Note (1): This address is the value of Code Flash 256KB. Address of the last block depends on the size of Code Flash.

Customer-specific information and BLE software executable file, it is necessary to write separately Code Flash. For how to write to the on-chip flash memory, see the Renesas Flash Programmer flash memory programming software User's Manual (R20UT2906EJ0202).

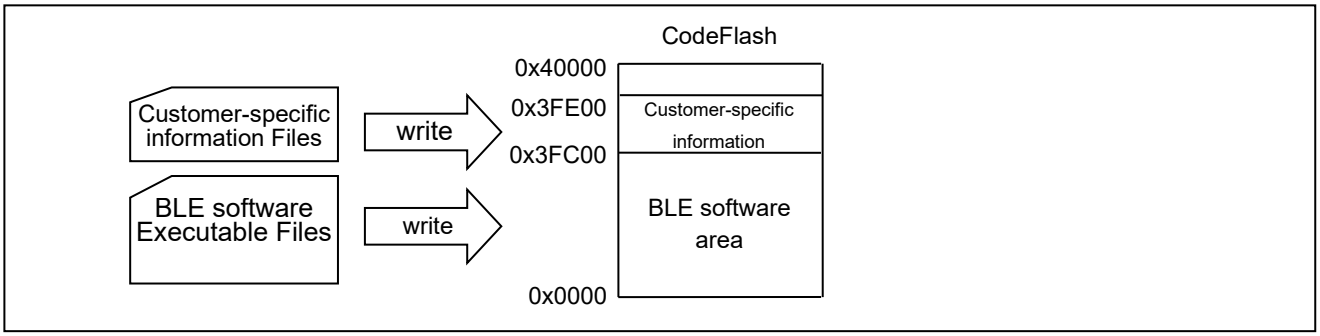


Figure 5-38 Write area of BLE software and customer-specific information

5.6 Selection of own Bluetooth Device address

The BLE software, search for the three types of BD address setting area in the following order, to determine its own BD address. When the BD address not 0xFF values are set even one, to determine a valid BD address.

1. Data Flash area (refer to 7.20.4.1)
2. Customer-specific information on Code Flash area (refer to 5.5)
3. CFG_TEST_BDADDR defined values of config.h

Selection of the BD address is performed BLE software startup.When writing the BD address on the Data Flash, it is necessary to restart the BLE MCU.

6. Creating Executable Files

This section describes how to create the executable files (hex files) of the BLE software programs that run on the RL78/G1D.

6.1 Changing the Configuration Parameters

The items that can be configured by the user when creating an executable file are listed in Table 6-1, and the configuration method is described from *6.1.2*.

Table 6-1 Changing the Configuration Parameters

User-Configurable Items		Specifiable Value
Maximum number of simultaneous connections		1 to 8
Heap area size		The size to be used by the user application can be added
Operating frequency setting		Main System Clock In case of high-speed on-chip oscillator : 4MHz, 8MHz, 16 MHz, 32 MHz In case of X1 oscillator (X1,X2): 4MHz, 8MHz, 16MHz In case of external clock(RF part or APP MCU) 4MHz, 8MHz, 16MHz Subsystem Clock XT1 clock oscillator (XT1,XT2) external clock (EXCLKS)
MCU part initialization setting		clock output setting
RF part initialization setting		RF part user options - external power amplifier setting - on-chip DC-DC converter setting - RF slow clock setting - hi-speed clock output setting - hi-speed clock setting - Sleep Clock Accuracy setting (20ppm to 500ppm)
Baud rate or Transfer clock of serial communication	UART	4,800 bps to 250 kbps
	CSI	4,800 bps to 250 kbps
	IIC	100 kbps to 400 kbps
Peak current consumption notification*1	Enable/disable	Enabled or disabled
	Notification start time	1 ms before, 2 ms before, 4 ms before
	Peak notification	Processing following peak notification
	Peak end notification	Processing following peak end notification
HCI packet monitoring feature *2		Enable or disable
Whether to enable/disable of each profile *3		Enable or disable
GAP parameters	Device Search parameters	Search time Scan interval Scan window

User-Configurable Items		Specifiable Value
	Limited Discoverable mode parameters	Discoverable time
	Auto / Selective connection parameters	Scan interval Scan window Connection interval Slave latency Supervision timeout
	Privacy parameters	Private address change interval
GAP characteristics	Device Name	UTF-8 String
	Appearance	Category value
	Peripheral Preferred Connection Parameters	Connection interval Slave latency Supervision timeout
GATT characteristics	Service Changed	Start of Affected Attribute Handle Range End of Affected Attribute Handle Range

Note:

*1: For details about peak current consumption notification, see *7.20.1*.

*2: For details about HCI packet monitoring feature, see *12*.

*3: For details about how to create custom profile, see *7.4.2*.

6.1.1 Maximum Number of Simultaneous Connections

When operating as Master device, the number of Slave devices that can be connected at the same time can be specified in the range of 1-8.

Since allocating memory from the Heap area required for the connection, the amount of required RAM can be reduced by limiting the number of simultaneous connections.

When operating as Slave device only, set 1 to the number of simultaneous connections.

The maximum number of simultaneous connections can be changed by the value of following macro.

Macro name: CFG_CON

Note: In all of the connection parameters, connections up to a maximum number of simultaneous connections are not guaranteed. The maximum number of simultaneous connections might be limited by the Heap size.

6.1.2 Allocating the Heap Area

The heap area used by BLE software is allocated by the ke_mem_heap array. The current setting (BLE_HEAP_SIZE) is the minimum memory capacity for BLE software to run. When the user program runs on the BLE MCU and ke_malloc is used in the Embedded configuration, for example, memory for the amount required by the user program should be added.

The heap memory size can be changed in the following source file.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\arch\rl78

File name: config.h

6.1.3 Changing the Operating Frequency

In the BLE software, some processing depends on the operating frequency of the BLE MCU. Therefore, when changing the operating frequency of the BLE MCU, define the corresponding macro listed below to inform the operating frequency

to the BLE software. Define one of the following macros as the compiler option in your project setting.

When select the high-speed on-chip oscillator, the available operating frequencies are 4MHz, 8MHz, 16MHz and 32MHz. When select the X1 oscillator or external clock, the available operation frequencies are 4MHz, 8MHz and 16MHz.

Table 6-2 Macro names to specify operating frequency

Macro name	Corresponding operating frequency
CLK_HOCO_4MHZ	4 MHz using high-speed on-chip oscillator
CLK_HOCO_8MHZ	8 MHz using high-speed on-chip oscillator (default)
CLK_HOCO_16MHZ	16 MHz using high-speed on-chip oscillator
CLK_HOCO_32MHZ	32 MHz using high-speed on-chip oscillator
CLK_X1_4MHZ	4 MHz using X1 oscillator
CLK_X1_8MHZ	8 MHz using X1 oscillator
CLK_X1_16MHZ	16 MHz using X1 oscillator
CLK_EX_RF_4MHZ	4 MHz using external clock of RF part
CLK_EX_RF_8MHZ	8 MHz using external clock of RF part
CLK_EX_RF_16MHZ	16 MHz using external clock of RF part
CLK_EX_4MHZ	4 MHz using external clock of APP MCU
CLK_EX_8MHZ	8 MHz using external clock of APP MCU
CLK_EX_16MHZ	16 MHz using external clock of APP MCU

In addition, use the same operating frequency as that is specified by the user option byte.

To specify the operating frequency to the user option byte, set in the following way. Refer to the documentation of your development environment. By default, CLK_HOCO_8MHZ is used.

- For CS+(CS+ for CA, CX and CS+ for CC common)

  Click the right mouse button on "CA78K0R (build tool)" (If CS+ for CC is “CC-RL (build tool)”) in the "Project Tree", then select "Properties", and change the "User Option Byte Value" in the "device" of the "Link Options" tab. Refer to about the setting of user option byte that corresponds to the operating frequency.

Table 6-3 Setting of User Option Byte

Setting of user option byte			Operating frequency	Operating mode
000C0	000C1	000C2		
(arbitrary)	(arbitrary)	2B	4MHz	LV (low voltage main) mode
		AA	8MHz	LS (low speed main) mode
		E9	16MHz	HS (high speed main) mode
		E8	32MHz	

- For e^2 studio

  Click the right mouse button in the “Project Explorer”, and select the “Renesas Tool Settings”. Change the “User Option Byte Value” in the “C/C++ Build” -> “Settings” and” Tool Settings” -> “Linker” -> “Device”.

When changing the subsystem clock of the BLE MCU and RF part, define the corresponding macro listed below to inform the subsystem clock to the BLE software. Define one of the following macros as the compiler option in your project setting.

Table 6-4 Macro names to specify subsystem clock

Macro name	Corresponding subsystem clock
CLK_SUB_XT1	MCU part: 32.768kHz using XT1 clock oscillator input (XT1,XT2)

	RF part: 16.384kHz using PCLBUZ0 input (default)
CLK_SUB_EX_OT	MCU part: 32.768kHz using external clock input (EXCLKS) RF part: 32.768kHz using RF slow clock for internal oscillation circuit
No defined	MCU part: 15kHz using MCU Low-speed On-chip oscillator RF part: 32.768kHz using RF slow clock for internal oscillation circuit

6.1.4 Setting MCU part initialization

MCU part of RL78/G1D is initialized by executing plf_init() function in arch_main.c. It is possible to change below setting by plf_init() function. When not using RF slow clock for internal oscillation circuit, it is necessary to supply clock(16.384kHz) from MCU part to RF part. Clock output setting is specified by argument "plf_flg" of plf_init() function. Also, it is possible to use the macro shown in Table 6-5 for the clock output setting.

- clock output setting(whether to output 16.384kHz or 32.768kHz clock from PCLBUZ0 or not)

Note: For details about output clock (PCLBUZ0) of MCU part, see *RL78/G1D User's Manual: Hardware*.

Table 6-5 MCU part initialization setting macros

Macro name	Value	Setting Item	Description
PLF_PCLBUZ_NONE	0x00	clock output (select one)	Clock(16.384kHz/32.768kHz) is not output.
PLF_PCLBUZ_16KHZ	0x01		Clock(16.384kHz) is output. (default)
PLF_PCLBUZ_32KHZ	0x02		Clock(32.768kHz) is output.

Table 6-6 MCU part initialization function

Function name	void plf_init(const uint8_t plf_flg);	
Overview	Initializes MCU part of RL78/G1D	
Description	This function initializes hi-speed on-chip oscillator, port s, interrupt registers related to RF part. In addition, if argument plf_flg is PLF_PCLBUZ_16KHZ or PLF_PCLBUZ_32KHz, initializes MCU to output 16.384kHz or 32.768kHz clock from PCLBUZ0.	
Parameters	const uint8_t plf_flg	MCU part initialization setting Refer to Table 6-5 "MCU part initialization setting macros".
Return value	none	

You can select one of the settings shown in Table 6-7 by specifying the compile option of the BLE Software as the argument of plf_init function. If you use the other settings, specify one of the macros in Table 6-5 as the argument.

Table 6-7 RF slow clock output

Compile options		Selected arguments
RF slow clock output		
	CLK_SUB_XT1 (default)	PLF_PCLBUZ_16KHZ
	CLK_SUB_EX_OT	PLF_PCLBUZ_NONE
	None specified	PLF_PCLBUZ_NONE

6.1.5 Setting RF part initialization

RF part of RL78/G1D is initialized by executing rf_init() and rwble_init() function of arch_main.c.

It is possible to change below settings by rf_init() function. Below settings are specified by argument "rf_flg" of rf_init()

function. Also, it is possible to use the macro shown in Table 6-8 for the settings.

- external power amplifier setting (whether to output control signal from TXSELH_RF,TXSELL_RF or not)
- on-chip DC-DC converter setting(whether to use on-chip DC-DC converter or not)
- RF slow clock setting(whether to use RF slow clock for internal oscillation circuit or not, when not using it, it is necessary to be supplied clock from MCU to EXSLK_RF)
- hi-speed clock setting(selecting clock from Not output/16MHz/8MHz/4MHz)

It is possible to change below settings by rwble_init() function.

- Sleep Clock Accuracy setting (20ppm to 500ppm)

Note: For details about user options of RF part, see *RL78/G1D User's Manual: Hardware*.

Table 6-8 RF part initialization setting macros

Macro name	Value	Setting Item	Description
RF_EXPA_OFF	0x0000	external power amplifier (select one)	External power amplifier is not used. (default)
RF_EXPA_ON	0x0001		External power amplifier is used.
RF_DCDC_ON	0x0000	DC-DC converter (select one)	On-chip DC-DC converter is used. (default)
RF_DCDC_OFF	0x0002		On-chip DC-DC converter is not used.
RF_INT_32KHZ	0x0000	RF slow clock (select one)	Internal oscillation circuit is used.
RF_EXT_32KHZ	0x0020		EXSLK_RF 32kHz is used.
RF_EXT_16KHZ	0x0040		EXSLK_RF 16kHz is used. (default)
RF_CLK_NONE	0x0000	high-speed clock output (select one)	Hi-speed clock from RF part is not output. (default)
RF_CLK_16MHZ	0x0300		16MHz
RF_CLK_8MHZ	0x0400		8MHz
RF_CLK_4MHZ	0x0500		4MHz

Table 6-9 RF part initialization function

Function name	bool rf_init(const uint16_t rf_flg);	
Overview	Initializes RF part of RL78/G1D	
Description	This function initializes RF part in accordance with the setting of argument rf_flg. By executing this function, the operation mode of RF part is changed from POWEROFF to STANDBY_RF.	
Parameters	const uint16_t rf_flg	RF part initialization setting Refer to Table 6-8 "RF part initialization setting macros" and set OR operation value of each setting value.
Return value	true	initialization succeeds
	false	initialization fails

Table 6-10 Sleep Clock Accuracy setting macros

Macro name	Value	Description
SCA_500PPM	0	500ppm
SCA_250PPM	1	250ppm
SCA_150PPM	2	150ppm
SCA_100PPM	3	100ppm

SCA_75PPM	4	75ppm
SCA_50PPM	5	50ppm (default)
SCA_30PPM	6	30ppm
SCA_20PPM	7	20ppm

Table 6-11 Sleep Clock Accuracy setting function

Function name	void rwble_init(const struct bd_addr *bd_addr, const uint8_t sca);	
Overview	Bluetooth Device Address and SCA setting	
Description	This function sets the Bluetooth Device Address and Sleep Clock Accuracy (SCA) to BLE software.	
Parameters	const struct bd_addr *bd_addr	Bluetooth Device Address setting Refer to section 5.6 *Selection of own Bluetooth Device address* and section 7.20.4.1 *Bluetooth device address write.*
	const uint8_t sca	Sleep Clock Accuracy setting Refer to Table 6-10 Sleep Clock Accuracy setting macros".
Return value	None	

You can select one of the settings shown in Table 6-12 by specifying the compile option of the BLE Software as the arguments of rf_init or rwble_init function. If you use the other settings, specify one of the macros in Table 6-8 or Table 6-10 as the arguments.

Table 6-12 Clock setting combination

Compile options		Selected arguments
RF slow clock		
	CLK_SUB_XT1 (default)	RF_EXT_16KHZ / SCA_50PPM
	CLK_SUB_EX_OT	RF_INT_32KHZ / SCA_250PPM
	None specified	RF_INT_32KHZ / SCA_250PPM
High-speed clock output		
	CLK_EX_RF_4MHZ	RF_CLK_4MHZ
	CLK_EX_RF_8MHZ	RF_CLK_8MHZ
	CLK_EX_RF_16MHZ	RF_CLK_16MHZ
	None specified (default)	RF_CLK_NONE

6.1.6 Selecting the serial communication method

When using the Modem configuration, Communication hardware and connection method is selectable.
To select communication hardware, refer to 1 below. To select connection method, refer to 2 below.

1. Communication Hardware

- To select the UART

Add the following files in the following folder to your development environment project.
If the files csi.h and csi.c and iic_slave.c and iic_slave.h have been added to your project, remove them.
By default, the following files have been added to your project.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\uart
Files: uart.c, uart.h

UART channel is defined by the following macro in uart.c.

Table 6-13 UART channel setting macro

Macro name	Value	Description
UART_CHANNEL	0	UART0 (default)
	1	UART1

・ To select the CSI

Add the following files in the following folder to your development environment project.

If the files uart.h and uart.c and iic_slave.c and iic_slave.h have been added to your project, remove them.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\csi

Files: csi.c, csi.h

CSI channel is defined by the following macro in csi.c.

Table 6-14 CSI channel setting macro

Macro name	Value	Description
CSI_CHANNEL	0	CSI00 (default)
	1	CSI20

・ To select the IIC

Add the following files in the following folder to your development environment project.

If the files uart.h and uart.c and csi.c and csi.h have been added to your project, remove them.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\iic

Files: iic_slave.c, iic_slave.h

IIC slave address is defined by the following macro in iic_slave.h.

Table 6-15 IIC slave address setting macro

Macro name	Range	Default Value
IIC_SLAVE_ADDRESS	0x00 to 0x7F	0x50

2. Connection Method

By changing the constant value of the macro that is defined in the file serial.h, you can select the connection method. If you choose UART above, you can select the 2-wire, 3-wire or 2-wire with branch connection method. If you choose to CSI above, you can select the 4-wire and 5-wire connection method. If you choose to IIC above, you can select the 3-wire connection method.

With reference to Table 6-16, change the value of macro corresponding to communication hardware and connection method, which you want to use, from to (1). Also change the values of other macros to (0).

* By default, UART and 2-wire connection method is selected.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\serial

File: serial.h

Table 6-16 Connection method of serial communication

Hardware	Connection Method	Corresponding Macros
UART	2-wire	SERIAL_U_2WIRE
	3-wire	SERIAL_U_3WIRE
	2-wire with branch	SERIAL_U_DIV_2WIRE
CSI	4-wire	SERIAL_C_4WIRE

	5-wire	SERIAL_C_5WIRE
IIC	3-wire	SERIAL_I_3WIRE

```
39 :     /*
40 :      * DEFINES
41 :      ************************************************
42 :      */
43 :     #define SERIAL_U_2WIRE      (1)
44 :     #define SERIAL_U_3WIRE      (0)
45 :     #define SERIAL_U_DIV_2WIRE  (0)
46 :     #define SERIAL_C_4WIRE      (0)
47 :     #define SERIAL_C_5WIRE      (0)
48 :     #define SERIAL_I_3WIRE      (0)
```

Figure 6-1 Sample Code for Selecting the serial communication method

3. Setting the use or non-use of the WAKEUP signal

With reference to Table 6-16, change the value of macro USE_WAKEUP_SIGNAL_PORT corresponding to connection method, which you want to use.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\wakeup

File: wakeup.c

Table 6-17 Connection method

Serial	Connection Method	USE_WAKEUP_SIGNAL_PORT
UART	2-wire	0
	3-wire	1
	2-wire with branch	1
CSI	4-wire	0
	5-wire	1
IIC	3-wire	0

```
22 :     #ifndef CONFIG_EMBEDDED
23 :      #define USE_WAKEUP_SIGNAL_PORT  (0) /* Modem Setting */
24 :     #else
25 :      #define USE_WAKEUP_SIGNAL_PORT  (0) /* Embedded Setting */
26 :     #endif
```

Figure 6-2 Sample code for Setting the use or non-use of the WAKEUP signal

6.1.7 Setting the UART baud rate

The UART0 baud rate can be specified by using the following source file:

Folder : \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\uart

File: uart.c
Function: void serial_init(void)

To change the baud rate, change the values of the registers below that are specified by the UART0 initialization function (serial_init). The MCU operating clock divided by the factor specified in serial clock selection register 0 (SPS0) is used as the UART0 operating clock. The baud rate can be changed by setting the values calculated based on the UART0 operating clock and the desired baud rate to the serial data registers (SDR00 and SDR01).

For details about the values to set to each register, see *6.1.7.1 Selecting the Serial Clock* and *6.1.7.2 Calculating the baud rate*.

Table 6-18 Baud Rate Setting Register

Register Symbol	Register Name	Description
SPS0	Serial clock selection register 0	UART0 operating clock setting
SDR00	Serial data register 00	UART0 transmission baud rate setting
SDR01	Serial data register 01	UART0 reception baud rate setting

Note: For details about the baud rate settings, see *RL78/G1D User's Manual: Hardware*.

6.1.7.1 Selecting the Serial Clock

The UART0 operating clock (f_{mck}) can be specified by using the PRS00[3 : 0] bits of serial clock selection register 0 (SPS0).

Table 6-19 Values Set to Serial Clock Selection Register

PRS 003	PRS 002	PRS 001	PRS 000		Selection of Operating Clock (CK00)				
					f_{clk} = 8MHz	f_{clk} = 12MHz	f_{clk} = 16MHz	f_{clk} = 24MHz	f_{clk} = 32MHz
0	0	0	0	f_{clk}	8MHz	12MHz	16MHz	24MHz	32MHz
0	0	0	1	$f_{clk}/2$	4MHz	6MHz	8MHz	12MHz	16MHz
0	0	1	0	$f_{clk}/2^2$	2MHz	3MHz	4MHz	6MHz	8MHz
0	0	1	1	$f_{clk}/2^3$	1MHz	1.5MHz	2MHz	3MHz	4MHz
0	1	0	0	$f_{clk}/2^4$	500kHz	750kHz	1MHz	1.5MHz	2MHz
0	1	0	1	$f_{clk}/2^5$	250kHz	375kHz	500kHz	750kHz	1MHz
0	1	1	0	$f_{clk}/2^6$	125kHz	188kHz	250kHz	375kHz	500kHz
0	1	1	1	$f_{clk}/2^7$	62.5kHz	93.8kHz	125kHz	188kHz	250kHz
1	0	0	0	$f_{clk}/2^8$	31.25kHz	46.9kHz	62.5kHz	93.8kHz	125kHz
1	0	0	1	$f_{clk}/2^9$	15.62kHz	23.4kHz	31.3kHz	46.9kHz	62.5kHz
1	0	1	0	$f_{clk}/2^{10}$	7.81kHz	11.7kHz	15.6kHz	23.4kHz	31.3kHz
1	0	1	1	$f_{clk}/2^{11}$	3.91kHz	5.86kHz	7.81kHz	11.7kHz	15.6kHz
1	1	0	0	$f_{clk}/2^{12}$	1.95kHz	2.93kHz	3.91kHz	5.86kHz	7.81kHz
1	1	0	1	$f_{clk}/2^{13}$	977Hz	1.46kHz	1.95kHz	2.93kHz	3.91kHz
1	1	1	0	$f_{clk}/2^{14}$	488Hz	732Hz	977Hz	1.46kHz	1.95kHz
1	1	1	1	$f_{clk}/2^{15}$	244Hz	366Hz	488Hz	732Hz	977Hz
Other than above				Setting prohibited					

6.1.7.2 Calculating the baud rate

The value to be set to bits 15 to 9 of the serial data registers (SDR00 and SDR01) can be obtained by using the formula below.

- SDRmn[15 : 9] = (UART0 operating clock (f_{mck}) ÷ 2 ÷ baud rate) – 1

To set 4800bps, when fclk = 8MHz and SPS0 = 0x0003, SDR01 and SDR00 is 0xCE00.
To set 250kbps, when fclk = 8MHz and SPS0 = 0x0002, SDR01 and SDR00 is 0x0600.

6.1.7.3 Sample code for changing the baud rate

The source code includes a sample code that changes the baud rate to 250 kbps. The default baud rate in Modem configuration is 4800 bps. This baud rate can be changed to 250 kbps by setting the constant for the #if statement in the 387th line in the source code to 0.

```
387:  #if (1)
388:    #ifndef CONFIG_EMBEDDED
389:     /* MCK = fclk/n = 1MHz */
390:     write_sfr(SPS0L, (uint8_t)((read_sfr(SPS0L) | UART_VAL_SPS_1MHZ)));
391:
392:     /* baudrate 4800bps(when MCK = 1MHz) */
393:     write_sfrp(UART_TXD_SDR, (uint16_t)0xCE00U);
394:     write_sfrp(UART_RXD_SDR, (uint16_t)0xCE00U);
395:     #else /*CONFIG_EMBEDDED*/
396:     /* MCK = fclk/n = 2MHz */
397:     write_sfr(SPS0L, (uint8_t)((read_sfr(SPS0L) | UART_VAL_SPS_2MHZ)));
398:
399:     /* baudrate 250000bps(when MCK = 2MHz) */
400:     write_sfrp(UART_TXD_SDR, (uint16_t)0x0600U);
401:     write_sfrp(UART_RXD_SDR, (uint16_t)0x0600U);
402:     #endif /*CONFIG_EMBEDDED*/
403:    #else
404:     /* MCK = fclk/n = 2MHz */
405:     write_sfr(SPS0L, (uint8_t)((read_sfr(SPS0L) | UART_VAL_SPS_2MHZ)));
406:     /* baudrate 250000bps(when MCK = 2MHz) */
407:     write_sfrp(UART_TXD_SDR, (uint16_t)0x0600U);
408:     write_sfrp(UART_RXD_SDR, (uint16_t)0x0600U);
409:    #endif
```

Figure 6-3 Sample Code for Changing the Baud Rate

If and only if you select the 2-wire connection method, you need to set the STOP-enable-flag that switches enable/disable of the Sleep function. The Sleep function realizes the low current consumption of BLE MCU.

If you want to set the baud rate to 4800bps, set the operand of "# if"-statement to (0). As the result, the STOP enable flag is set to true. If you want to set the baud rate to a value larger than 4800bps, set the operand of "# if"-statement to (0). As the result, the STOP enable flag is set to false.

```
414:    /* set stop permission */
415:    #if SERIAL_U_2WIRE
416:    #if (1)
417:      #ifndef CONFIG_EMBEDDED
418:      /* if baudrate is 4800bps, set enable */
419:      stop_flg = true;
420:      #else /*CONFIG_EMBEDDED*/
421:      /* if baudrate is over than 4800bps, set disable */
422:      stop_flg = false;
423:      #endif /*CONFIG_EMBEDDED*/
424:    #else
425:    /* if baudrate is over than 4800bps, set disable */
426:    stop_flg = false;
427:    #endif
428:    #else
```

Figure 6-4 Sample Code for Changing the Baud Rate

6.1.8 Setting the CSI baud rate

In the modem configuration, if you select the CSI as the communication hardware between APP MCU and BLE MCU, the baud rate is determined by the APP MCU. Therefore, you need to do nothing.

6.1.9 Setting the IIC transfer clock

In the modem configuration, if you select the IIC as the communication hardware between APP MCU and BLE MCU, the transfer clock is determined by the APP MCU. Therefore, you need to do nothing.

6.1.10 Wait for the time Sub Clock is stabled

The user need to change the waiting time for Sub Clock is stabled for matching the system. Change the number of times for execution of clk_waitfunc(). Initial setting is 1 second by 1000 times execution.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\driver\plf

File: plf.c

Function: plf_init()

6.1.11 Setting the Profile Service

The user configurable parameters for the profiles and services, you can change them by using variables and macros in the following source file.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\arch\rl78

File: prf_config.c, prf_config.h

Folder: \ Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\arch\rl78

File: prf_sel.h

Note: GATT database structure of existing profiles that are defined in the above file shall not be changed. Therefore it isn't possible to changes order or add/delete elements. If you do not want to expose the optional characteristics, please use the *RBLE_GATT_PERM_HIDE* permission.

6.1.11.1 Profile Enable / Disable Setting

Due to the deprecation and withdrawal plan of the profile version by the Bluetooth SIG, product registration using this profile is no longer possible, so this section has been obsoleted.

For product registration, refer to "Bluetooth LE microcomputer/module Bluetooth qualification acquisition application note" (R01AN3177).

6.1.11.2 GAP Parameters Setting

Parameters related to the GAP modes/procedures can be set by the value of the defined macros in Table 6-20. These macros are defined in source file.

Table 6-20 Macro name for GAP parameters

Macro name	Description	Note
GAP_DEV_SEARCH_TIME	Device Search time	Parameters for device search (Limited / General discovery procedure).
GAP_DEV_SEARCH_SCAN_INTV	Scan interval	
GAP_DEV_SEARCH_SCAN_WINDOW	Scan window	
GAP_LIM_ADV_TIMEOUT	Discoverable time	Parameters for limited discoverable mode.
GAP_SCAN_FAST_INTV	Scan interval	Parameters for Auto / Selective connection procedure.
GAP_SCAN_FAST_WINDOW	Scan window	
GAP_INIT_CONN_MIN_INTV	Max connection interval	
GAP_INIT_CONN_MAX_INTV	Min connection interval	
GAP_CONN_SLAVE_LATENCY	Slave latency	
GAP_DEV_SUPERVISION_TIMEOUT	Supervision timeout	
GAP_RESOLVABLE_PRIVATE_ADDR_INTV	Private address change interval	Set the favorite value according to the product use.

6.1.11.3 GAP Characteristic Setting

The following GAP characteristic value can be set.

- Device Name
- Appearance
- Peripheral Preferred Connection Parameters

It is possible to set the initial value of the Device Name characteristic value, which indicates a user-friendly name to identify a device, can be set by the macro definitions shown in Table 6-21.

Table 6-21 Device Name setting macro

Macro name	Description	Note
GAP_DEV_NAME	Initial value of the Device Name characteristic value	Set the device name as an UTF-8 string.

Note: Device name that was written in the Code Flash last block (see 5.5) is priority than this definition value. Also, it is possible to change the device name using API after a BLE software start-up.

It is possible to set the Appearance characteristic value, which indicates outward appearance of device, can be set by the variables shown in Table 6-22.

Table 6-22 Appearance setting variable

Variable name	Description	Note
static const uint16_t iconval	Externals setting variable of device	Set it according to the product, referring to the following. http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.gap.appearance.xml

For more information about the GAP Appearance characteristic, refer to Bluetooth Core Specification v4.2 [Vol. 3], the Part C Section 12.2.

The Peripheral Preferred Connection Parameters characteristic value, which is desired by the peripheral device connection parameters, can be set by the macro shown in Table 6-23.
These macros are defined in source file.

Table 6-23 Peripheral Preferred Connection Parameters setting macro

Macro name	Description	Note
GAP_PPCP_CONN_INTV_MAX	Maximum connection interval	Set the favorite value according to the product use.
GAP_PPCP_CONN_INTV_MIN	Minimum connection interval	
GAP_PPCP_SLAVE_LATENCY	Slave latency	
GAP_PPCP_STO_MULT	Supervision timeout	

For more information about the GAP Peripheral Preferred Connection Parameters characteristic, please refer to Bluetooth Core Specification v4.2 [Vol. 3], the Part C Section 12.3.

6.1.11.4 GATT Characteristic Setting

It is possible to set handle range of the Service Changed characteristic value, which indicates that services have changed, can be set by the macro definitions shown in Table 6-24.

Table 6-24 Service Changed setting macro

Macro name	Description	Note
GATT_SERVICE_CHANGED_START_HDL	Start of Affected Attribute Handle Range	By default, Service Changed characteristic does not expose.
GATT_SERVICE_CHANGED_END_HDL	End of Affected Attribute Handle Range	

Note: If there is a possibility to change the GATT database structure, please expose the Service Changed characteristic. And, if the client has enabled indication of this characteristic, please indicate using the rBLE API when you have changed the GATT database structure.

For more information about the GATT Service Changed characteristic, refer to Bluetooth Core Specification v4.2 [Vol. 3], the Part G Section 7.1.

6.1.11.5 Blood Pressure Service Characteristic Setting

Due to the deprecation and withdrawal plan of the profile version by the Bluetooth SIG, product registration using this profile is no longer possible, so this section has been obsoleted.

For product registration, refer to "Bluetooth LE microcomputer/module Bluetooth qualification acquisition application note" (R01AN3177).

6.1.11.6 HID Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.7 Battery Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.8 Specifying Device Information service product information

Refer to "6.1.11.5".

6.1.11.9 Heart Rate Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.10 Cycling Speed and Cadence Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.11 Cycling Power Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.12 Glucose Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.13 Current Time Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.14 Running Speed and Cadence Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.15 Alert Notification Service Characteristic Setting

Refer to "6.1.11.5".

6.1.11.16 Location and Navigation Service Characteristic Setting

Refer to "6.1.11.5".

6.2 Building a Project

Follow the procedure below to build a project for creating an executable file.

1. See *6.1* for how to change the parameters to accord with your environment.
2. To open a file in each environment, double-click in the following folder a project file or workspace file which is shown in Table 6-25 and matches the development environment and BLE software configuration in use.
   `\Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\tools\project\`
3. For CS+, click the **Build** menu and then select **Build Project**. For e^2 studio, click the **Project** menu and then select **Build Project**.
4. When build is finished, an executable file is output to the executable file creation folder shown in Table 6-25.

Table 6-25 Correspondence between Development Environment and Build Environment

Development Environment	Configu-ration	Project File or Workspace File	Executable File Creation Folder
CS+ for CA, CX (CA78K0R)	Modem	CubeSuite\BLE_Modem\BLE_Modem.mtpj	rBLE_emb\DefaultBuild
	Embedded	CubeSuite\BLE_Embedded\BLE_Embedded.mtpj	BLE_Emb\DefaultBuild
CS+ for CC (CC-RL)	Modem	CS_CCRL\BLE_Modem\BLE_Modem.mtpj	rBLE_Mdm\DefaultBuild
	Embedded	CS_CCRL\BLE_Embedded\BLE_Embedded.mtpj	rBLE_Emb\DefaultBuild
e^2 studio (CC-RL)	Modem	e2studio\BLE_Modem\rBLE_Mdm\ (*1)	rBLE_Mdm\DefaultBuild
	Embedded	e2studio\BLE_Embedded\rBLE_Emb\ (*1)	rBLE_Emb\DefaultBuild

*1 : e^2 studio project need to import for workspace in e^2 studio.

6.3 Additional Note

BLE software provides multiple libraries, so it is possible that the BLE software does not work properly due to the wrong combination of library even if the build succeeds. To avoid this situation, the BLE software has a function to check the combination of the library and to notify by the completion status of the GAP reset show in in Table 6-26 Library Management Status.

Also, for example, if the profile of the prototype has been added to the BLE software, there is a case to provide a version of the library for evaluation. Also in this case, it is can be determined by the status of the in Table 6-26 Library Management Status at the completion of GAP reset.

Note that if you have the evaluation version, do not apply to your product and please wait for the official release.

Table 6-26 Library Management Status

Status	Value	Description
RBLE_VERSION_FAIL	0xF7	Library combination error
RBLE_TEST_VERSION	0xF8	BLE software is test version

In addition, BLE software requires the stack size shown in Table 6-27.

Table 6-27 BLE software stack size

Required stack size	824bytes

The stack size of the caller of API and callback function of the application which BLE software calls are not contained in the above stack size. Please ensure enough stack size, when developing application.

7. Description of Features

This section describes the features of the BLE software.

7.1 Controller Stack

The controller stack includes the Host Controller Interface (HCI) and Link Layer (LL), and is used to control the RF/BB according to requests from the Host stack, and perform packet processing such as advertising and scanning.

7.1.1 Advertising

Advertising is used to establish a connection or periodically provide user data to the scanning device. During advertising, packets are transmitted on the advertising channel and the response from the scanning device is received and responded to. A device in this state is called an Advertiser.

When an Advertiser receives a connection request from another device and connects to that device, it operates as a slave device.

There are four types of advertising, and the relationships between the allowable responses for each advertising type are indicated in the following table.

Table 7-1 Advertising Event Types

Advertising Event Type		Allowable Response	
		SCAN_REQ	CONNECT_REQ
Connectable Undirected Event	ADV_IND	YES	YES
Connectable Directed Event	ADV_DIRECT_IND	NO	YES
Non-connectable Undirected Event	ADV_NONCONN_IND	NO	NO
Scannable Undirected Event	ADV_SCAN_IND	YES	NO

SCAN_REQ: Requests additional information.

CONNECT_REQ: Starts connection establishment.

Advertising uses any channel out of the advertising channels (channels 37, 38, or 39). The period during which advertising data is transmitted is called an advertising event, and the interval between advertising events is calculated as follows as *T_advEvent*.

T_advEvent = *advInterval* + *advDelay*

- *advInterval* : An integral multiple of 0.625 ms in the range of 20 ms to 10.24 s

(In the case of a scannable undirected event type or non-connectable undirected event type, this interval is not less than 100 ms. In the case of a connectable undirected event type, this interval is 20 ms or greater.)

- *advDelay* : Pseudorandom value in the range of 0 ms to 10 ms

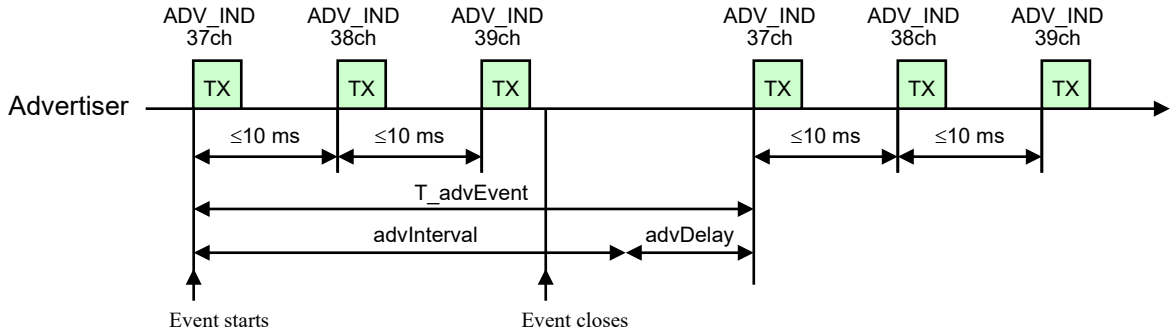


Figure 7-1 Advertising Event (ADV_IND)

For details about Advertising, see *Bluetooth Core Specification v4.2 [Vol. 6], Part B Section 4.4.2*.

7.1.2 Scanning

Scanning is used to receive data broadcasts from an Advertiser. A device that waits for packets from an Advertiser on an advertising channel is called a scanner. There two types of scanning : passive scanning and active scanning.

7.1.2.1 Passive Scanning

During passive scanning, the scanner only receives packets and does not transmit any packets.

7.1.2.2 Active Scanning

During active scanning, the scanner waits for advertising packets from the Advertiser and responds according to the advertising event type. Upon receiving an ADV_IND packet or ADV_SCAN_IND packet, the scanner can obtain additional information by sending a SCAN_REQ packet to the Advertiser.

An example of the operation of a scanner that receives an ADV_IND packet on channel 38 during active scanning is shown below.

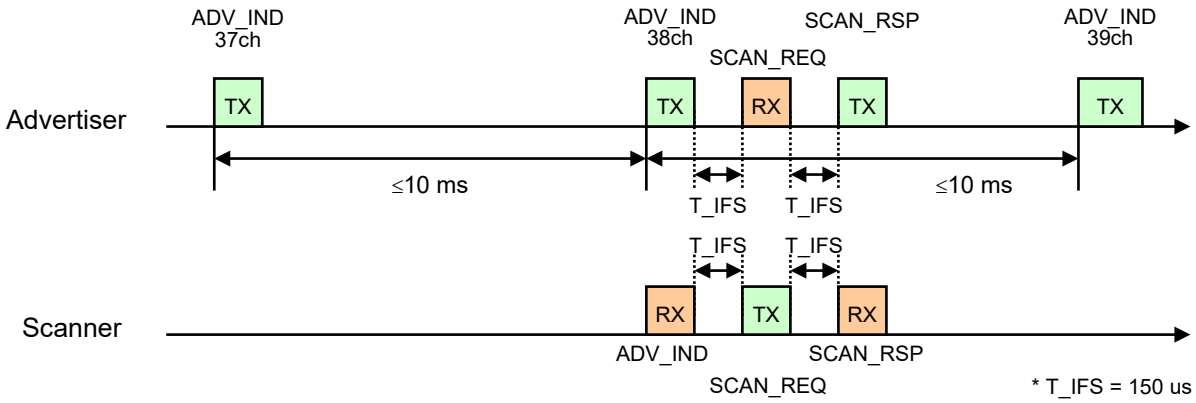


Figure 7-2 Active Scanning (ADV_IND)

For details about Scanning, see *Bluetooth Core Specification v4.2 [Vol. 6], Part B Section 4.4.3*.

7.1.3 Initiating

Initiating is used to establish a connection with another device. A device that waits for an advertising packet on an advertising channel in order to connect to another device is called an Initiator. An Initiator that receives an ADV_IND packet or ADV_DIRECT_IND packet operates as a master upon the end of initiation triggered by sending a CONNECT_REQ packet.

The first packet following transmission of a CONNECT_REQ packet is transmitted within *transmitWindowSize* that starts after 1.25ms + *transmitWindowOffset*.

- *transmitWindowOffset* : A multiple of 1.25 ms in the range of 0 ms to *connInterval*
- *transmitWindowSize* : A multiple of 1.25 ms in the range of 1.25 ms to 10 ms

While connected, the master and slave send and receive packets to each other alternately using *connInterval*.

- *connInterval* : A multiple of 1.25 ms in the range of 7.5 ms to 4.0 s

An example of the operation from when the initiator receives an ADV_IND packet from the Advertiser until it becomes the master device is shown below.

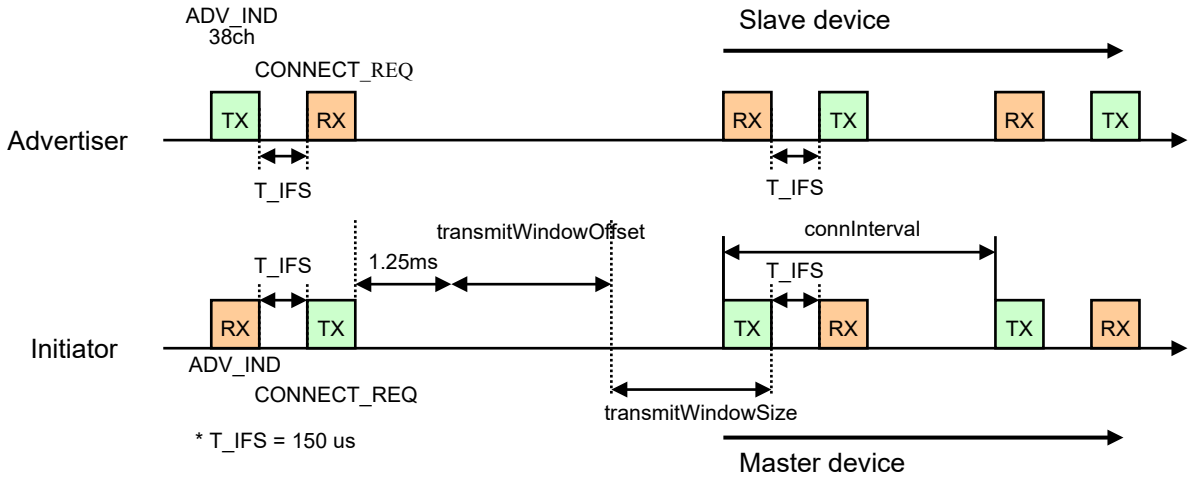


Figure 7-3 Connection Setup

For details about Initiating, see *Bluetooth Core Specification v4.2 [Vol. 6], Part B Section 4.4.4*.

7.1.4 White List

The White List is used to filter devices from which advertising packets, scan packets, and connection requests are allowed to be received. This list includes device addresses and address types (public address or random address).

The White List is managed in the Link Layer block of the Controller stack. In the Reset state, the White List is empty and is set by the application.

For details about the White List, see *Bluetooth Core Specification v4.2 [Vol. 6], Part B Section 4.3.1*.

7.1.4.1 Advertising filter policy

The advertising filter policy determines how scan and connection requests are processed. When connectable directed advertising is used, the advertising filter policy is ignored. At all other times, one of the following advertising filter policies set by the application is used.

- The White List is not used and scan and connection requests from all devices are processed (default state).
- Only scan and connection requests from devices registered to the White List are processed.
- Scan requests from all devices are processed, but only connection requests from devices on the White List are processed.
- Connection requests from all devices are processed, but only scan requests from devices on the White List are processed.

For details about the Advertising filter policy, see *Bluetooth Core Specification v4.2 [Vol. 6], Part B Section 4.3.2*.

7.1.4.2 Scanner filter policy

The scanner filter policy determines how advertising packets are processed. One of the following scanner filter policies set by the Host stack is used.

- The White List is not used and advertising packets from all devices are processed (default state).
- Only advertising packets from devices registered to the White List are processed.

For details about the Scanner filter policy, see *Bluetooth Core Specification v4.2 [Vol. 6], Part B Section 4.3.3*.

7.1.4.3 Initiator filter policy

The initiator filter policy determines how advertising packets are processed. One of the following initiator filter policies set by the application is used.

- Connectable advertising packets from all devices registered to the White List are processed.
- The White List is ignored and the connectable advertising packets from a single device that is specified are processed.

For details about the Initiator filter policy, see *Bluetooth Core Specification v4.2 [Vol. 6], Part B Section 4.3.4*.

7.2 Generic Access Profile

The Generic Access Profile (GAP) executes access procedures according to the link management and security requirements for processes such as device discovery and peer device connection and disconnection.

7.2.1 GAP roles

The BLE software supports all four of the roles prescribed in the GAP listed in Table 7-2.

Table 7-2 GAP Roles

GAP Roles	Description
Broadcaster	Transmits advertising events. In the Link Layer, it is called the Advertiser.
Observer	Receives advertising events. In the Link Layer, it is called the Scanner.
Central	Establishes a physical link. In the Link Layer, it is called the Master.
Peripheral	Accepts the establishment of a physical link. In the Link Layer, it is called the Slave.

For details about the GAP roles, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 9*.

7.2.2 GAP modes and procedures

This section describes the GAP modes and the GAP procedures supported by the BLE software.

Advertising event types are used according to each mode and procedure, as shown in Table 7-3.

Table 7-3 Advertising Event Type

Advertising Event Type	Description
Connectable Undirected	Can respond to CONNECT_REQ or SCAN_REQ (connectable).
Connectable Directed	Only connectable with specified device.
Scannable Undirected	Can respond to SCAN_REQ (non-connectable).
Non-connectable Undirected	Only information sent from Advertiser (non-connectable)

7.2.2.1 Broadcast mode and Observation procedure

The broadcast mode and observation procedure allow two devices to communicate with each other without establishing connection between them.

A device in the broadcast mode is called the Broadcaster. It broadcasts data during advertising events. All data sent by a device in the broadcast mode is considered unreliable since there is no acknowledgment from any device that may have received the data. The advertising event types that can be transmitted are non-connectable undirected events and scannable undirected events. The AD type flag of the advertising data must be set to 0 both for the LE General Discoverable Mode and LE Limited Discoverable Mode.

The device executing the observation procedure is called the Observer. It receives advertising events.

The rBLE API provides APIs for executing the broadcast mode and observation procedure. The advertising data in the broadcast mode can be set freely by the user.

For details about the Broadcast mode and Observation procedure, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 9.1*.

7.2.2.2 Discovery mode and procedure

Peripheral device discovery is possible in the discovery modes and by using the discovery procedures. The discovery modes are modes that allow discovery from remote devices by transmitting advertising data. The discovery procedures are procedures for receiving advertising data from scanning and discovering peripheral devices. They are executed by a central device.

Table 7-4 lists the relationships between the discovery modes, transmittable advertising event types, and AD type flag setting values of the advertising data.

Table 7-4 Discovery Modes

Discovery Mode	Transmittable Advertising Event Types	Flags AD Type		Description
		LE General Discoverable Mode	LE Limited Discoverable Mode	
Non-Discoverable	• Non-connectable Undirected • Scannable Undirected • Do not transmit	0	0	Not discoverable by any device performing either the general discovery procedure or the limited discovery procedure.
Limited Discoverable	• Non-connectable Undirected • Scannable Undirected • Connectable Undirected	0	1	Discoverable for a limited period of time by other devices performing the limited or general device discovery procedure.
General Discoverable	• Non-connectable Undirected • Scannable Undirected • Connectable Undirected	1	0	Discoverable by devices performing the general discovery procedure.

The discovery procedures are outlined in Table 7-5 below.

Table 7-5 Discovery Procedure

Discovery Procedure	Description
Limited Discovery	Only devices in limited discoverable mode can be discovered.
General Discovery	Devices in general or limited discoverable mode can be discovered.
Name Discovery	The device names of connectable remote devices are retrieved by using GATT.

The rBLE API provides APIs for executing discovery mode, peripheral device discovery, and device name retrieval. It is also possible to discover only existing devices by using peripheral device discovery. Any advertising data can be set by the user in discovery mode. The AD type flag of the advertising data must be set according to the mode to be executed.

For details about each mode and procedures, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 9.2*.

7.2.2.3 Connection mode and procedure

The connection modes and procedures can be used to establish connections with other devices. The connection modes are modes that allow connection from remote devices by sending advertising data. These modes are executed by peripheral devices. The connection procedures are procedures for establishing a connection with a peripheral device. They are executed by a central device. (The terminate connection procedure, which cuts off a link can be executed from both a central and peripheral devices.)

Table 7-6 shows the relationships among the connection modes and transmittable advertising event types that can be sent.

Table 7-6 Connection Modes

Connection Mode	Transmittable Advertising Event Types	Description
Non-connectable	• Non-connectable Undirected • Scannable Undirected	Connection not allowed.
Directed connectable	• Connectable Directed	Connection is possible only from known devices that execute the auto connection establishment procedure or general connection establishment procedure.
Undirected connectable	• Connectable Undirected	Connection is possible from devices that execute the auto connection establishment procedure or general connection establishment procedure.

The connection procedures are outlined in Table 7-7 below.

Table 7-7 Connection Procedure

Connection Procedure	Description
Auto connection establishment	Automatically establishes a connection with devices in the directed connectable mode or undirected connectable mode, using the White List of the Initiator.
General connection establishment	Establishes a connection with known devices in the directed connectable mode or undirected connectable mode.
Selective connection establishment	Establishes a connection with devices on the White List using the connection configuration parameters selected by the host.
Direct connection establishment	Establishes a connection with one known device using the connection configuration parameters selected by the host.
Connection parameter update	Changes the connection parameters of an established connection.
Terminate connection	Terminates the connection with a peer device.

The rBLE API provides APIs for executing the connection modes and connection procedures. In a connection mode, advertising data can be set freely by the user. A connection mode can be executed in combination with a discovery mode.

For details about each mode and procedures, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 9.3*.

7.2.2.4 Bonding mode and procedure

Bonding allows two connected devices to exchange and store security and identity information to create a trusting relationship. Security and identity information is called bonding information. When devices store bonding information, they are said to have bonded.

There are two bonding modes, as shown in Table 7-8 below.

Table 7-8 Bonding Modes

Bonding Mode	Description
Non-Bondable	Bonding with peer devices is not allowed.
Bondable	Bonding with peer devices is allowed in the bondable mode.

The bonding procedure is executed when a device that is not bonded accesses a service that requires bonding. Pairing is used for bonding.

The rBLE API provides APIs for executing bonding and responding to bonding requests.

For details about bonding, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 9.4*.

7.2.3 Security

This section describes BLE security as defined in the GAP.

7.2.3.1 Security mode

The security requirements of a device, a service, or a service request are expressed in terms of a security mode and security level. Each service or service request may have its own security requirement. The device may also have a security requirement. Pairing is required in order to satisfy the various security requirements. There are two types of pairing, authenticated pairing in which the paired devices are protected from MITM (man-in-the-middle) attacks, and unauthenticated pairing in which the paired devices are not protected from MITM. The security mode level is determined by device pairing, encryption, and the use or non-use of data signing. Table 7-9 lists the security modes and security levels defined in the BLE standard.

Table 7-9 Security Modes and Levels

Security Mode	Security Level	Description
LE Security Mode 1	1	No security (no authentication and no encryption)
	2	Unauthenticated pairing with encryption
	3	Authenticated pairing with encryption
	4	Authenticated LE Secure Connections pairing with encryption
LE Security Mode 2	1	Unauthenticated pairing with data signing
	2	Authenticated pairing with data signing

Note: BLE software is not supported LE security mode 1 level 4 (LE Secure Connections).

LE security mode 1 level 2 satisfies the security requirements for LE security mode 1 level 1.

LE security mode 1 level 3 satisfies the security requirements for LE security mode 1 level 2. LE security mode 1 level 3 satisfies the security requirements for LE security mode 2.

If LE security mode 1 and LE security mode 2 level 2 are required for a given physical link, then LE security mode 1 level 3 is used.

If LE security mode 1 level 3 and LE security mode 2 are required for a given physical link, then LE security mode 1

level 3 is used.

If LE security mode 1 level 2 and LE security mode 2 level 1 are required for a given physical link, then LE security mode 1 level 2 is used.

The rBLE API provides an API for setting the security mode. The security mode can be set for each profile.

For details about the security modes, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 10.2*.

7.2.3.2 Authentication procedure

The authentication procedure describes how the required security is established when a device initiates a service request to a remote device and when a device receives a service request from a remote device. The authentication procedure covers both LE security mode 1 and LE security mode 2. The authentication procedure is only initiated after a connection has been established.

If security is not required, or the security requirements are already satisfied, service requests will continue. If security is required, pairing will be necessary.

The rBLE API allows the execution of pairing by executing bonding.

For details about the Authentication procedure, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 10.3*.

7.2.3.3 Data Signing

Data signing is used for transferring authenticated data between two devices in an unencrypted connection. The data signing method is used by services that require fast connection setup and fast data transfer.

If a service request specifies LE security mode 2, the connection data signing procedure is used.

In the BLE software, data signing is used when sending the Signed Write command of ATT (*Bluetooth Core Specification v4.2[Vol. 3], Part F Section 3.4.5.4*). The CSRK key for data signing must be set in advance to the rBLE API. The validity of the received signed data is verified internally by the BLE software. The CSRK of the remote device that is required at that time is requested from the application by the rBLE API. Management of the CSRK distributed from remote devices and generation of the CSRK of the local device must be performed by the application.

For details about the data signature, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 10.4*.

7.2.3.4 Privacy features

The privacy feature is possible to prevent to be tracked and identified from an attacker by using random address.

The rBLE API provides an API that enables the privacy feature, and random addresses are generated according to the role by setting IRK in advance. The IRK must be generated and managed by the application.

For details about the privacy features, see *Bluetooth Core Specification v4.1 [Vol. 3], Part C Section 10.7*.

7.2.4 Bluetooth Device Address

Each Bluetooth device shall be allocated a unique 48-bit Bluetooth device address (BD_ADDR) to identify the Bluetooth devices. The BLE specification, two types of public address and a random address has been defined as a Bluetooth device address.

7.2.4.1 Public address

Public address shall be created in accordance with IEEE 802-2001 standard, and using 24bit OUI (Organizationally

Unique Identifier). Public address is uniquely given to each of the device. A device shall not change its public address value during the lifetime of the device.

This address shall be obtained from the IEEE Registration Authority.

7.2.4.2 Random address

Random addresses may be of the sub-types listed in Table 7-10 below.

Table 7-10 Random Address

Type		Description
Static address		When not needing registration to IEEE Registration Authority, it's possible to use as substitution of the public address. A device may choose to initialize its static address to a new value after each power cycle. It's also possible to use the same address during the lifetime. A device shall not change its static address value once initialized until the device is power cycled.
Private address		Private addresses are used for privacy, it is possible to make it difficult to track and identify from an attacker by periodically changing the address.
	Non-resolvable private address	Bluetooth Core Specification v4.1 or later, this address shall not use for the connection.
	Resolvable private address	This type of address is generated from an IRK and a 24-bit random number. Only devices which is shared IRK can identify the device by Resolvable Private Address Resolution procedure.

Note: Bluetooth Specification does not support random device address collision avoidance or detection. And therefore, random addresses have a very small chance of being in conflict. When using private address, reconnection takes time because Resolvable Private Address Resolution procedure is required.

- Interpretation of resolvable private address

  Using the Resolvable Private Address Resolution procedure, the host can resolve the resolvable private addresses of all peer devices that have an IRK. Once a resolvable private address is resolved, the host can link this address to the peer device. If the host stores multiple IRKs, this procedure is repeated for each stored IRK until the corresponding address is successfully resolved.

The BLE software can use the address type other than Non-resolvable. Private addresses are generated automatically by enabling the privacy feature using the IRK specified from the application. Resolvable private addresses are resolved internally by the BLE software. The IRK of the remote device required at that time is requested from the application by the rBLE API. The IRKs of remote devices must be managed by the application.

For details about the random address, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 10.8*.

7.2.5 Advertising and Scan response data formats

Advertising and Scan Response data are created in the format shown in Figure 7-4.

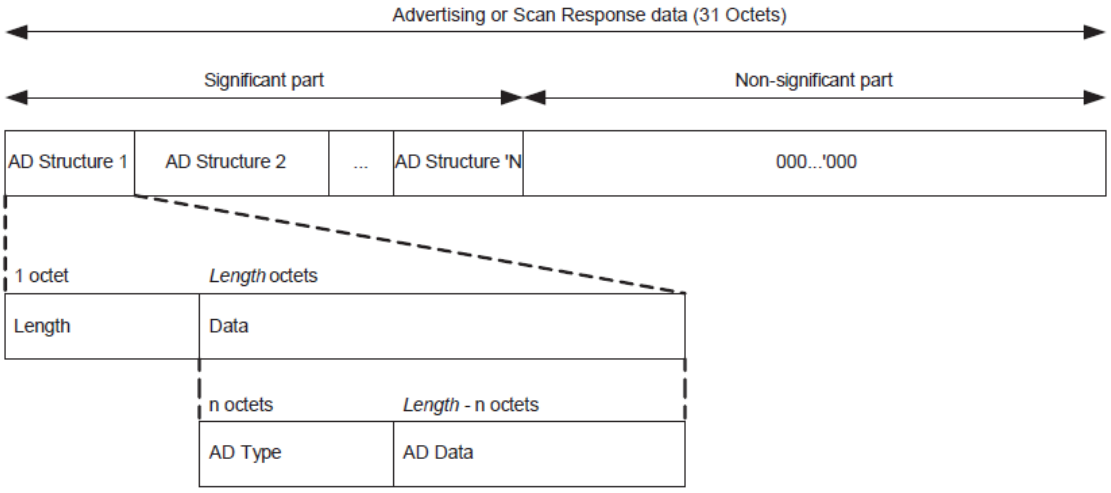


Figure 7-4 Advertising and Scan Response Data Formats

The Advertising and Scan Response data has the following characteristics:

- Total data size of 31 octets
- Consists of multiple AD structures
- Each AD structure consists of 1 octet of length information and the data of the length octet.
- The data of the length octet consists of an AD type of n octets and AD data of length – n octets.
- If the total size of all the AD structures is less than 31 octets, it is padded with 0s.
- Data consisting of all 0s is used only to allow early termination of Advertising or Scan Response.
- Only the significant part of the Advertising or Scan Response data is transmitted over the air.
- The Advertising and Scan Response data is transmitted in advertising events.
- Advertising data is placed in the AdvData field of the ADV_IND, ADV_NONCONN_IND, and ADV_SCAN_IND packets.
- The Scan Response data is transmitted in the ScanRspData field of the SCAN_RSP packet.

The definitions of the AD types that can be used for AD structures and the AD data format are shown in Table 7-11.

Table 7-11 AD Type Definitions and AD Data Format

<table>
<tr><th>AD Type</th><th>AD Type Value</th><th>AD Data Description</th></tr>
<tr><td>Flags</td><td>0x01</td><td>Flags are configured of the following bits :
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>LE Limited Discoverable Mode</td></tr>
<tr><td>1</td><td>LE General Discoverable Mode</td></tr>
<tr><td>2</td><td>BR/EDR not supported</td></tr>
<tr><td>3</td><td>Simultaneous LE and BR/EDR operation supported (Controller)</td></tr>
<tr><td>4</td><td>Simultaneous LE and BR/EDR operation supported (Host)</td></tr>
</table>
• The Flags AD type must not include Scan Response data.
• Advertising data can include only one Flags AD type.</td></tr>
<tr><td rowspan="2">Service</td><td>0x02</td><td>Usable 16-bit service UUID</td></tr>
<tr><td>0x03</td><td>Usable 16-bit service UUID (complete list)</td></tr>
</table>

<table>
<tr><th>AD Type</th><th>AD Type Value</th><th>AD Data Description</th></tr>
<tr><td></td><td>0x06</td><td>Usable 128-bit service UUID</td></tr>
<tr><td></td><td>0x07</td><td>Usable 128-bit service UUID (complete list)</td></tr>
<tr><td>Local Name</td><td>0x08</td><td>Short local device name</td></tr>
<tr><td></td><td>0x09</td><td>Complete local device name</td></tr>
<tr><td>TX Power Level</td><td>0x0A</td><td>Advertising packet transmission power level (1 byte)
0xXX : −127 to +127 dBm</td></tr>
<tr><td>OOB Data</td><td>0x0D</td><td>Class of device (3 bytes)</td></tr>
<tr><td></td><td>0x0E</td><td>Simple Pairing Hash C (16 bytes)</td></tr>
<tr><td></td><td>0x0F</td><td>Simple Pairing Randomizer R (16 bytes)</td></tr>
<tr><td>TK Value</td><td>0x10</td><td>TK (Temporary Key) used for pairing (128 bits)</td></tr>
<tr><td>OOB Flags</td><td>0x11</td><td>The OOB Flags field consists of the following bits :
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>OOB Flags field
(0 = OOB data not present, 1 = OOB data present)</td></tr>
<tr><td>1</td><td>LE supported by host</td></tr>
<tr><td>2</td><td>Simultaneous LE and BR/EDR operation supported (Host)</td></tr>
<tr><td>3</td><td>Address type
(0 = public address, 1 = random address)</td></tr>
</table>
</td></tr>
<tr><td>Slave Connection Interval Range</td><td>0x12</td><td>Includes the connection interval requested by a Peripheral for all the logical links. A Central should use the data of this AD type in the Peripheral.
The first 2 bytes indicate the minimum connection interval.
N = 0x0006 to 0x0C80 (Time = N * 1.25 ms)
The next 2 bytes indicate the maximum connection interval.
N = 0x0006 to 0x0C80 (Time = N * 1.25 ms)
0xFFFF : don’t care</td></tr>
<tr><td>Service Solicitation</td><td>0x14</td><td>16-bit service UUID list</td></tr>
<tr><td></td><td>0x15</td><td>128-bit service UUID list</td></tr>
<tr><td>Service Data</td><td>0x16</td><td>Additional service data that follows the 16-bit service UUID</td></tr>
<tr><td>Manufacturer Specific Data</td><td>0xFF</td><td>Manufacturer specific data
The company ID is included in the first 2 bytes.</td></tr>
</table>

Table 7-12 shows an advertising data setting example. In this example, Flags is set as the AD type, along with the complete device name and 16-bit UUID.

Table 7-12 Advertising Data Example

Value	Description
0x02	Data length of this AD structure (2 octets)
0x01	AD type = Flags
0x06	LE Limited Discoverable Flag bit and BR/EDR not supported bit set
0x08	Data length of this AD structure (8 octets)
0x09	AD type = Complete local device name
0x52	'R'
0x65	'e'
0x6E	'n'
0x65	'e'
0x73	's'
0x61	'a'
0x73	's'
0x07	Data length of this AD structure (7 octets)
0x03	AD type = Usable 16-bit service UUID (complete list)
0x02 0x18	Immediate Alert service (UUID : 0x1802)
0x03 0x18	Link Loss service (UUID : 0x1803)
0x04 0x18	Tx Power service (UUID : 0x1804)

The Advertising and Scan Response data can be set freely by the user. This data must be set as appropriate for the use case according to the above format.

For details about the Advertising Scan Response data formats, see *Bluetooth Core Specification v4.2 [Vol. 3], Part C Section 11*. For details about the AD Type, see *Supplement to the Bluetooth Core Specification v5, Part A*.

7.3 Security Manager

The Security Manager (SM) is in charge of ensuring secure BLE communication, including pairing, encryption, private address resolution and data signing.

Pairing is performed to generate the key used for link encryption and data signing. The device that initiates pairing is called the Initiator, and the device that responds is called the Responder. Pairing is executed in the phases listed below and shown in Figure 7-5.

- Phase 1: Pairing feature exchange
- Phase 2: STK generation. The STK generation method is based on the information that was exchanged during phase 1.
- Phase 3: Distribution of the generated key. This is done via a link encrypted by using the key generated in phase 2.

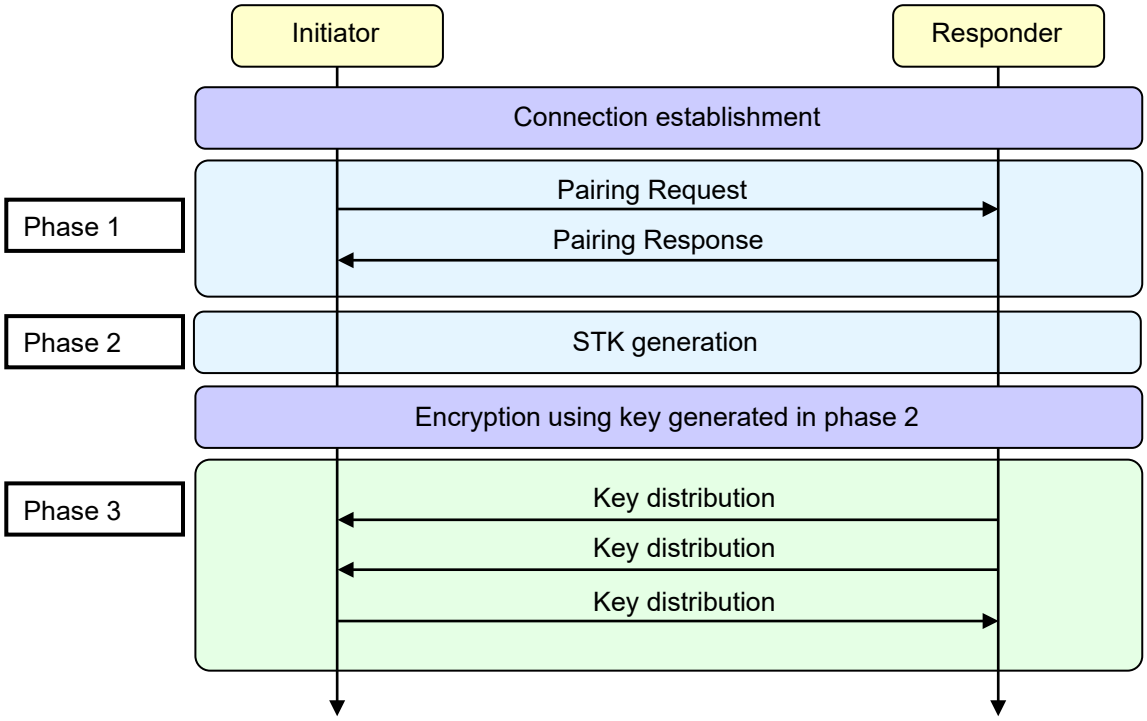


Figure 7-5 LE Pairing Phases

The keys used for pairing, encryption, private address resolution, and data signing are shown in the table below.

Table 7-13 Key Definitions

Key Type	Description	Generated by
IRK (Identity Resolving Key)	128-bit key used to generate and resolve random addresses	The application
CSRK (Connection Signature Resolving Key)	128-bit key used to create signatures and verify the signatures of received data	The application
LTK (Long Term Key)	128-bit key used to generate the session key for encryption (partially used according to the agreed upon key size)	The application
EDIV (Encrypted Diversifier)	16-bit key used to identify the LTK. A new EDIV is generated each time an LTK is distributed.	The application

Key Type	Description	Generated by
Rand (Random Number)	64-bit key used to identify the LTK. A new Rand is generated each time an LTK is distributed.	The application
STK (Short Term Key)	128-bit key generated in pairing phase 2 by using TK and used to encrypt the link after phase 2	The BLE software by using the TK passed from the application
TK (Temporary Key)	128-bit key used in pairing phase 2 to generate the STK	The application

For details about generating each key, see *Bluetooth Core Specification v4.2 [Vol. 3], Part H Section 2.4.1 and 2.4.2*.

7.3.1 Pairing feature exchange

In pairing phase 1, the Initiator and Responder perform a pairing feature exchange. The fields of the features that are exchanged are described below. The pairing method used in phase 2 is determined based on the information that is exchanged here.

- IO Capability

  This field indicates the input and output capabilities of the device. IO Capability (Table 7-16) is the combination of the Input Capability (Table 7-14) and Output Capability (Table 7-15).

Table 7-14 Input Capability

Input Capability	Description
No input	The device does not have the capability to indicate ‘yes’ or ‘no’.
Yes / No	The device has at least two buttons that can indicate ‘yes’ and ‘no’, or a mechanism that allows ‘yes’ or ‘no’ indication. (Example : ‘yes’ is indicated by pressing a button within the time limit, and ‘no’ is displayed upon timeout.)
Keyboard	The device has a numeric keyboard that can be used to input numbers ‘0’ through ‘9’. The device also has at least two buttons that can indicate ‘yes’ and ‘no’, or a mechanism that allows ‘yes’ or ‘no’ indication.

Table 7-15 Output Capability

Output Capability	Description
No output	The device does not have the capability to display or communicate a 6-digit number.
Numeric output	The device has the capability to display or communicate a 6-digit number.

The input capability (Table 7-14) and output capability (Table 7-15) are mapped to a single IO capability shown in Table 7-16, which is used for the pairing feature exchange.

Table 7-16 IO Capability Mapping

Input \ Output	No output	Numeric output
No input	NoInputNoOutput	DisplayOnly
Yes / No	NoInputNoOutput[1]	DisplayYesNo
Keyboard	KeyboardOnly	KeyboardDisplay

Note: The combination of Yes/No and No Output is regarded as NoInputNoOutput.

- Validity of OOB authentication data
  The data required for authentication of a remote device that is using a communication method other than Bluetooth is called OOB (Out Of Band) data. This field indicates the presence/non-presence of the OOB authentication data required for remote device authentication.

- Authentication requirement
  This field indicates the following security properties used for authentication :
  - Protection from MITM (man-in-the-middle) attacks required/not required
  - Perform/don't perform bonding

- Encryption key size
  This field indicates the size of the key to be used for link encryption. The smaller of the key sizes indicated by the two devices is used for encryption. The size can be specified in 1-byte units from 7 bytes (56 bits) to 16 bytes (128 bits).

- Key distribution
  This field indicates the keys whose distribution in phase 3 is requested by the pairing Initiator and Responder. The distribution of multiple keys can be requested.
  - EncKey (LTK, EDIV, and Rand)
  - IdKey (IRK)
  - Sign (CSRK)

The rBLE API allows the above-described pairing features to be freely set during bonding execution or in the response API. Set the appropriate pairing feature according to the functions and/or purpose of the local device.

For details about each feature, see *Bluetooth Core Specification v4.2 [Vol. 3], Part H Section 2.3.1 to 2.3.4*.

7.3.2 STK generation

In phase 2, the pairing method (STK generation method) is determined using the information exchanged between the devices in phase 1. The TK required for STK generation is determined by pairing. The BLE pairing methods and characteristics are shown below.

- Just Works
  This method provides no protection against eavesdroppers or man-in-the-middle attacks during the pairing process. If no eavesdropping or MITM attack occurs during the pairing process, security is ensured by encryption during future connections.
  No action on the user's part is required during the pairing process.
  The TK used by both devices is 0x00000000000000000000000000000000.

- Passkey Entry
  This is a method whereby a six-digit number is input by each device to the other, or a six-digit (random) number is displayed by one of the devices and that number is input to the other device.
  This 6-digit number (decimal: 000,000 to 999,999) is used as the TK.
  This method provides protection against man-in-the-middle attacks. (The success rate of MITM attacks is just 0.000001.) Since the TK range is limited, protection from eavesdropping during the pairing process is limited. If there is no eavesdropping during the pairing process, security is ensured by encryption during future connections.

- Out of Band
  This is a method whereby a 128-bit random number exchanged prior to the pairing process is used as the TK value. If the method used for TK exchange is robust against man-in-the-middle attacks, pairing using OOB is also protected against such attacks.
  No restrictions apply as long as the value of the TK is within 128 bits, so this method can be said to be more secure than Just Works and Passkey Entry. However, both devices need to have interfaces that allow TK exchange.

If both devices have OOB data, pairing via OOB is performed. If neither of the devices requires protection against MITM attacks, pairing is performed by using the Just Works method. Other pairing methods are determined from the mutual IO capability indicated by the information exchanged in phase 1. If the key generated as the result of the pairing process does not satisfy the authentication requirements, pairing fails.

Table 7-17 shows the pairing methods determined from the IO capability of the Initiator and Responder, the Passkey Entry roles and the keys generated during the pairing process.

Table 7-17 Mapping of Pairing Methods According to IO Capability

Initiator / Responder	DisplayOnly	DisplayYesNo	KeyboardOnly	NoInput NoOutput	Keyboard Display
DisplayOnly	Just Works Unauthenticated	Just Works Unauthenticated	Passkey Entry Initiator inputs and Responder displays Authenticated	Just Works Unauthenticated	Passkey Entry Initiator inputs and Responder displays Authenticated
DisplayYesNo	Just Works Unauthenticated	Just Works Unauthenticated	Passkey Entry Initiator inputs and Responder displays Authenticated	Just Works Unauthenticated	Passkey Entry Initiator inputs and Responder displays Authenticated
KeyboardOnly	Passkey Entry Initiator displays and responder inputs Authenticated	Passkey Entry Initiator displays and responder inputs Authenticated	Passkey Entry Initiator and Responder input Authenticated	Just Works Unauthenticated	Passkey Entry Initiator displays and Responder inputs Authenticated
NoInputNoOutput	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated	Just Works Unauthenticated
KeyboardDisplay	Passkey Entry Initiator displays and Responder inputs Authenticated	Passkey Entry Initiator displays and Responder inputs Authenticated	Passkey Entry Initiator inputs and Responder displays Authenticated	Just Works Unauthenticated	Passkey Entry Initiator displays and Responder inputs Authenticated

Authenticated: Keys protected from MITM attacks are generated.

Unauthenticated: Keys not protected from MITM attacks are generated.

An STK is generated in the BLE software by determining the pairing method from the pairing features passed from the application and remote device. For pairing implemented by Passkey Entry and OOB, the TK required for STK generation is requested from the application.

For details about each pairing method, see *Bluetooth Core Specification v4.2 [Vol. 3], Part H Section 2.3.5*.

7.3.3 Key distribution

In phase 3, key distribution is performed as needed. The key distribution method is the same regardless of the pairing method in phase 2.

The following keys may be distributed from the slave to the master :

- LTK, EDIV, and Rand
- IRK
- CSRK

The following keys may be distributed from the master to the slave :

- LTK, EDIV, and Rand
- IRK
- CSRK

Before all the keys are distributed, the links must be encrypted by using the STK generated in phase 2. (Link encryption using the STK is performed automatically by the BLE software.)

The BLE software carries out key distribution based on the pairing feature passed from the application and remote device. In the case of IRK or CSRK distribution, distribution is performed automatically using the key specified from the application beforehand. In the case of LTK distribution, an LTK request is issued to the application and the LTK passed from the application is distributed. All the keys distributed from a remote device are reported to the application.

Execute link encryption from the application by using the LTK after pairing.

Note: Key management (recording keys, linking with devices) is not performed by the BLE software. Execute these actions from the application.

7.4 Generic Attribute Profile

The Generic Attribute Profile (GATT) specifies the service framework using the Attribute Protocol (ATT). This framework defines the format of the services and service characteristics and the procedures.

GATT has server and client roles. A server exposes data called data called characteristics that are grouped within a service as shown in Figure 7-6. The exposed services or characteristics are identified by an identifier called UUID. On the server, data are managed with attribute handles, which are the basis of ATT.

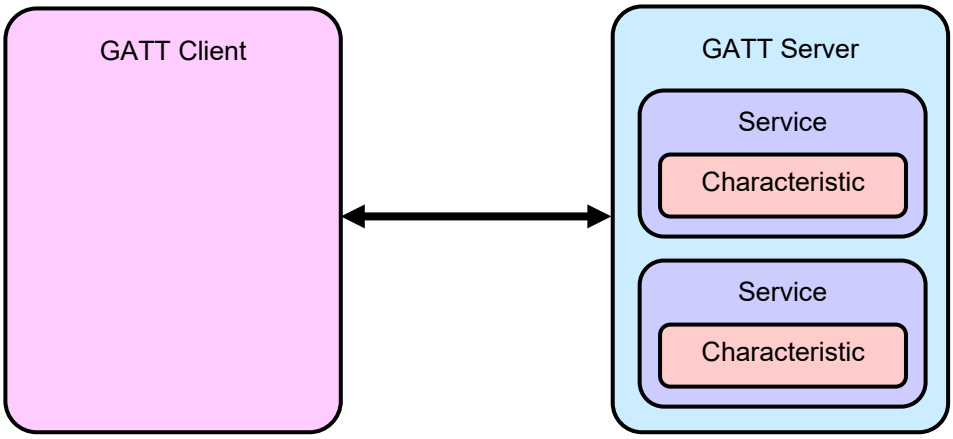


Figure 7-6 Generic Attribute Profile

The properties listed in the following table are set for characteristics exposed on the GATT server. The set properties are prescribed for each service.

Table 7-18 Properties of Characteristics

Property	Value	Description
Broadcast	0x01 (RBLE_GATT_CHAR_PROP_BCAST)	The server broadcasts characteristic values using the characteristic configuration descriptor
Read	0x02 (RBLE_GATT_CHAR_PROP_RD)	Permits reading of characteristic values from the client
Write Without Response	0x04 (RBLE_GATT_CHAR_PROP_WR_NO_RESP)	Permits writing of characteristic values from the client (no server response in relation to write operation)
Write	0x08 (RBLE_GATT_CHAR_PROP_WR)	Permits characteristic values can be written from the client (server response in relation to write operation)
Notify	0x10 (RBLE_GATT_CHAR_PROP_NTF)	Permits notification about characteristic values issued from the server to the client (no verification of reception from the client to the server)
Indicate	0x20 (RBLE_GATT_CHAR_PROP_IND)	Permits indication of characteristic values from the server to the client (verification of reception from the client to the server)
Authenticated Signed	0x40	Permits signed writing of characteristic

Property	Value	Description
Writes	(RBLE_GATT_CHAR_PROP_AUTH)	values from the client
Extended Properties	0x80 (RBLE_GATT_CHAR_PROP_EXT_PROP)	Extended properties

In GATT, the following procedures are defined for exchanging data exposed on the server.

- Discover services exposed by the server from a client
- Discover characteristics exposed by the server from a client
- Discover characteristic descriptors exposed by the server from a client
- Read characteristic values exposed by the server from a client (Read)
- Write characteristic values exposed by the server from a client (Write)
- Read the characteristic configuration descriptor exposed on the server from a client (Read)
- Write the characteristic configuration descriptor on the server from a client (Write)
- Indicate characteristic values from the server to a client (Indication) (with verification of reception from the client to the server)
- Notification of characteristic values from the server to a client (Notification)

In the rBLE API, the APIs are available to perform above procedures. You can add a new profile using these APIs.

In each profile, the services/characteristics discovery is performed automatically and the APIs to read/write characteristic values are provided.

Each profile is based on the GATT, and its functions are implemented using the mechanism described above.

For details about the GATT, see *Bluetooth Core Specification v4.2 [Vol. 3], Part G*.

7.4.1 GATT Database

The data exposed by the GATT Server is called GATT database. Each profile implements the use case by exchanging data in the GATT database. The data registered in the GATT database is defined by each service.

7.4.1.1 Database structure

The database consists of "elements", that are "services" or "properties" used in the profile. All of the "elements" are included in "attributes". The "attributes" are used as a container for carrying the data of the profile service by the ATT. The "Attributes" are managed by "attribute handles".

In order to realize the use cases, the profile which is located on top of the GATT requires one or more "services". The "services" consists of references to the "characteristics" or other "services". Each characteristic includes "characteristic values" or "characteristic descriptors", which contain the service data.

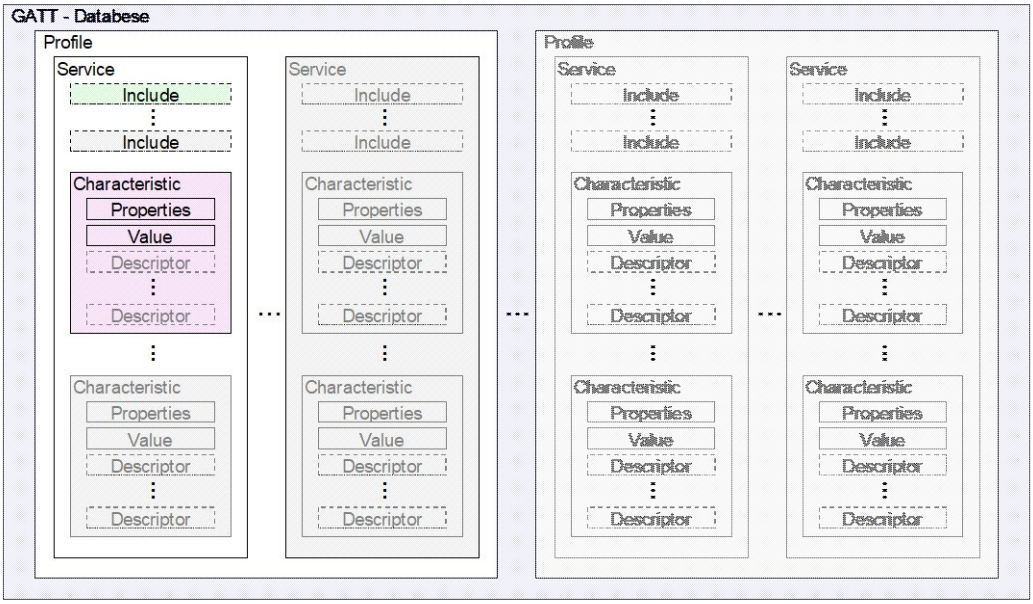


Figure 7-7 GATT database structure

In the BLE software, the GATT database is built by the following variables in the source file.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\arch\rl78

File Name: prf_config_host.c, prf_config.c

Variable Name: struct atts_desc atts_desc_list_host [], struct atts_desc atts_desc_list_prf []

The GATT database is expanded to the RAM of the RL78/G1D. The database structure when built is not changed during operation.

Each element of the array atts_desc_list corresponds to "attribute". In the BLE software, the index of the array is managed as an "attribute handle". Each attribute consist of the following items.

Table 7-19 Component of attribute

Member name	Description
type	Attribute type UUID
maxlength	Maximum length of attribute value
length	Current length of attribute value
taskid	The most significant 6 bits : Profile task ID that attribute belongs to The least significant 10 bits : Index to identify attribute
perm	Permission of attribute
*value	Pointer to the storage of attribute value

7.4.1.2 Attribute Type

The attribute consists of service definition, include definition and characteristic definition shown in Figure 7-8 below.

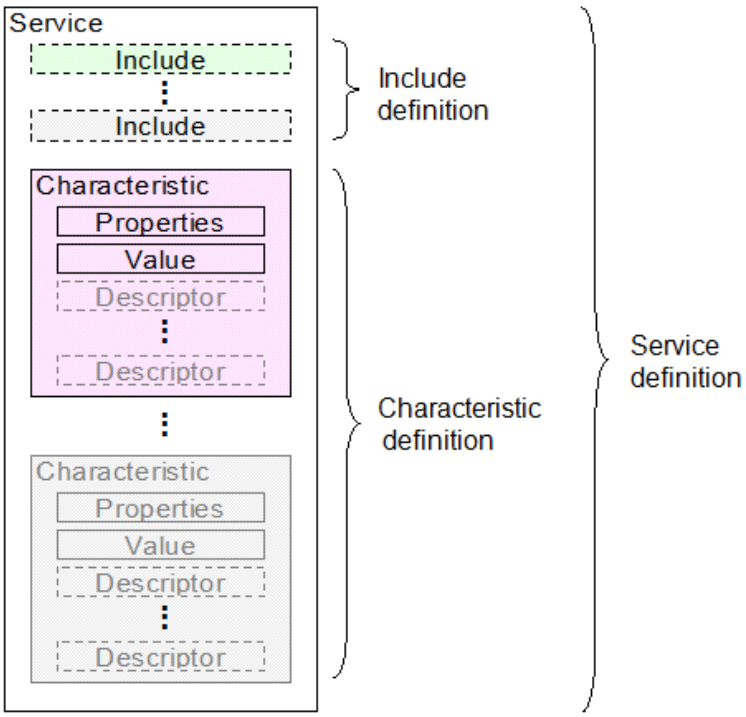


Figure 7-8 Attribute structure

Permissions shown in Table 7-20 are set for each attribute, the access from GATT client can be restricted based on reading, writing, security requirements and so on.

Table 7-20 Attribute permission

Permission value	Description
RBLE_GATT_PERM_NONE	No permission
RBLE_GATT_PERM_RD	Readable
RBLE_GATT_PERM_RD_UNAUTH	Unauthenticated pairing required to read
RBLE_GATT_PERM_RD_AUTH	Authenticated pairing required to read
RBLE_GATT_PERM_RD_AUTZ	Authorization requited to read
RBLE_GATT_PERM_WR	Writable
RBLE_GATT_PERM_WR_UNAUTH	Unauthenticated pairing required to write
RBLE_GATT_PERM_WR_AUTH	Authenticated pairing required to write
RBLE_GATT_PERM_WR_AUTZ	Authorization required to write
RBLE_GATT_PERM_NI	Able to be notified / indicated
RBLE_GATT_PERM_NI_UNAUTH	Unauthenticated pairing required for notification / indication
RBLE_GATT_PERM_NI_AUTH	Authenticated pairing required for notification / indication
RBLE_GATT_PERM_NI_AUTZ	Authorization required for notification / indication
RBLE_GATT_PERM_EKS	Encryption by key of sufficient length Required
RBLE_GATT_PERM_HIDE	Unexposed (hidden)
RBLE_GATT_PERM_ENC	Encryption required
RBLE_GATT_PERM_NOTIFY_COMP_EN	Indicate the notification completion event

(1) Service Definition

The Service definition begins with the service declarations and contains include definitions and characteristic definitions. The service definition ends when the next service definition starts or when the attribute handles have reached the maximum. Include definitions and characteristic definitions placed in the service definition are considered as a part of the service definition.

The following table shows the attribute structure in the service declaration.

Table 7-21 Service declaration

Attribute type UUID	Attribute values	Permissions
0x2800 : Primary Service (RBLE_DECL_PRIMARY_SERVICE) or 0x2801: Secondary Service (RBLE_DECL_SECONDARY_SERVICE)	16bit or 128bit service UUID	Read Only (RBLE_GATT_PERM_RD)

For more information on service definition and service declaration, see *Bluetooth Core Specification v4.2 [Vol. 3], the Part G Section 3.1*.

(2) Include Definition

The include definition is defined when referring other service, and contains only one include declaration.

The following table shows the attribute structure in the include definition.

Table 7-22 Include declaration

Attribute type UUID	Attribute values	Permissions
0x2802 : Include (RBLE_DECL_INCLUDE)	• Included Service attribute handle • group end handle • service UUID	Read Only (RBLE_GATT_PERM_RD)

For more information on include definition and include declaration, see *Bluetooth Core Specification v4.2 [Vol. 3], the Part G Section 3.2*.

(3) Characteristic Definition

The characteristic definition begins with the characteristic declaration, and contains characteristic value declarations and characteristic descriptor declarations. The characteristic definition ends when the next service declaration or next characteristic declaration starts or when the attribute handles have reached the maximum. Each declaration is contained in individual attribute.

The characteristic definition requires both characteristic declarations and characteristic value declarations. The characteristic value declarations should be placed immediate after the characteristics declarations. The optional characteristics descriptor declarations are placed after the characteristic value declarations.

The following table shows the attribute structure in the characteristic definition.

Table 7-23 Characteristic Declaration

Attribute type UUID	Attribute values	Permissions
0x2803 : Characteristic (RBLE_DECL_CHARACTERISTIC)	• characteristic property (defined by higher profile. Refer to Table 7-18.) • characteristic value handle • characteristic UUID	Read Only (RBLE_GATT_PERM_RD)

Table 7-24 shows attribute structure of characteristic value declaration. All of the characteristic definitions contain characteristic value declarations.

Table 7-24 Characteristic value declaration

Attribute type UUID	Attribute values	Permissions
Characteristic UUID	Characteristic value	Defined by higher profile, or implementation dependent

The characteristic descriptor declaration is defined to include information corresponding to the characteristic value. GATT defines standard characteristic descriptors shown in Table 7-25, that may be used in the upper layers.

Table 7-25 Characteristic descriptor

Characteristic descriptor name	Description
Characteristic Extended Properties	Defines additional characteristic properties.
Characteristic User Description	Defines a user textual description of the characteristic value.
Client Characteristic Configuration	Defines how the characteristic may be configured by a specific client.
Server Characteristic Configuration	Defines how the characteristic may be configured for the server.
Characteristic Presentation Format	Defines characteristic value format
Characteristic Aggregate Format	List of presentation format

Table 7-26 shows attribute structure of characteristic descriptor declaration.

Table 7-26 Characteristic descriptor declaration

Attribute type UUID	Attribute values	Permissions
0x2900 : Characteristic Extended Properties (RBLE_DESC_CHAR_EXT_PROPERTIES)	Extended Property Bit	Read Only (RBLE_GATT_PERM_RD)
0x2901 : Characteristic User Description (RBLE_DESC_CHAR_USER_DESCRIPTION)	(UTF-8)	Defined by higher profile, or implementation dependent
0x2902 : Client Characteristic Configuration (RBLE_DESC_CLIENT_CHAR_CONF)	Characteristic Configuration Bit	Read (RBLE_GATT_PERM_RD) Write(RBLE_GATT_PERM_WR) Requirement of authentication / Authorization depends on higher layer.
0x2903 : Server Characteristic Configuration (RBLE_DESC_SERVER_CHAR_CONF)	Characteristic Configuration Bit	
0x2904 : Characteristic Format (RBLE_DESC_CHAR_PRESENTATION_FMT)	• Format • Index • Unit • Namespace	Read Only (RBLE_GATT_PERM_RD)

Attribute type UUID	Attribute values	Permissions
	• Description	
0x2905 : Characteristic Aggregate Format (RBLE_DESC_CHAR_AGGREGATE_FMT)	Handle List of Presentation Format Declaration Handle	Read Only (RBLE_GATT_PERM_RD)

For more information on characteristic definition, see *Bluetooth Core Specification v4.2 [Vol. 3], the Part G Section 3.3.*

7.4.1.3 Example of Database Construction

From the first to the third elements in the GATT database array atts_desc_list_host[ ] (prf_config_host.c) construct the device name characteristic of the GAP service.

In addition to the contents defined in the specification, the current length of attribute value, the maximum length of attribute value, the task ID and so on are added to the database array.

Table 7-27 Device Name Characteristic of GAP service

Handle	Attribute type	Attribute UUID	Permissions	Attribute value
0x0001	Service declaration	0x2800 (Primary Service)	Read Only	0x1800 (Generic Access)
0x0002	Characteristic declaration	0x2803 (Characteristic)	Read Only	Property: Read, Write Characteristic handle: 0x0003 Characteristic UUID: 0x2A00
0x0003	Characteristic value declaration	0x2A00 (Device Name)	Read Write (Unauthenticated pairing required)	Device name (Renesas-BLE)

7.4.2 Creating a User Profile

In the BLE software, creating user profiles is possible. This section describes how to create a user profile.

7.4.2.1 GATT Client Role

The profile role that is the GATT client can discover services or characteristics and can read or write characteristic values or characteristic descriptors using rBLE API. In addition, the indications or notifications from the server are notified to the registered callback function as events. The response to the indication from the server is performed by the BLE software automatically.

Procedures and requirements, such as service discovery or characteristic discovery, refer to the specifications for each profile.

7.4.2.2 GATT Server Role

The profile role that is the GATT server can be realized by constructing the GATT database. The BLE software refers the

GATT database and sends response automatically according to permissions and properties for the service/characteristic discovery or read characteristic value request from the remote GATT client.

For the write characteristic value request, the BLE software checks permissions and properties of the corresponding characteristic value, and notifies to the application if it is writable through rBLE API.

The BLE software doesn't update the attribute value in the database. Confirm the validity of the write request value by the application, and update the database by the application. Also, the response to the write request is required, send it by the application. The rBLE API can send the notification and/or indication of characteristic value.

How to build GATT database is described below. Refer to 7.4.1 about the construction of the database.

It should be noted that the attributes and characteristics of the service to be registered in the database, refer to the specifications for each service profile.

(1) Definition of Attribute Identification Number

Add the index of the attributes "XXXX_IDX_XXXX" that are required by the user profile at the end of the attribute index enumeration declarations in the file prf_config.h. This index is used to identify the database elements by the BLE software.

Note

* Don't change the value of predefined enumerations in the file prf_config.h. The BLE software doesn't work correctly due to the inconsistency between the database and the BLE software.

(2) Definition of Attribute Handle

Add the handle of the attributes "XXXX_HDL_XXXX" that are required by the user profile at the end of the attribute handle enumeration declarations in the file db_handle.h. This enumeration should be equal to the index of database array.

Application can identify attributes using this enumeration. Also, if attribute handle is required to define attribute value, e.g., in the characteristic declarations, this enumeration can be used.

Note that the attribute handle enumeration should be equal to the index of database array.

(3) Constructing Database Array

Add the service definition (7.4.1.2(1)), include definition (7.4.1.2(2)) and characteristic definition (7.4.1.2(3)) required by the user profile at the end of the GATT database array atts_desc_list[ ] in the file prf_config.c.

The index of each array element should be equal to the enumerated value defined in (2). Each member of the database structure is explained below.

- type
  Set the UUID of attribute type. Refer to 7.4.1.2.
  If the attribute type is 128bit UUID, use DB_TYPE_128BIT_UUID macro for definition (defined in prf_config.h). (The *value below contains 128bit UUID.)

- length and maxlength
  Set the current length and maximum length of the attribute value.

- taskid
  Set the calculated value using "TASK_ATTID" macro defined in the file prf_config.c, with the task ID "TASK_RBLE" and enumerated value of attribute defined in (1).

- perm
  Set the appropriate permissions of attribute depending on the service specification. The BLE software restricts the access to the corresponding attribute from the remote GATT client according to these permissions.

- *value
  In case that attribute type is 16bit UUID:
  Set the pointer to the attribute value. For attribute value, set proper value according to the service specification. To set the attribute value of include declaration, the "ATTS_INCL" macro defined in the file prf_config.c is available. To set the attribute value of characteristic declaration, the "ATTS_CHAR" macro defined in the file prf_config.c is available.
  In the characteristic declaration, if the attribute to be declared is 16bit UUID, use struct atts_char_desc type for the attribute value (character declaration structure for 16bit UUID). If the attribute to be declared is 128bit UUID, use struct atts_char128_desc type for the attribute value (character declaration structure for 128bit UUID).

  In case that attribute type is 128bit UUID:
  Set the pointer to the structure whose type is struct atts_elmt_128.
  Set the UUID, UUID length and pointer to the attribute value to the members of this structure.

Note:

1. To remove unused existing profile role, use macro definition (6.1.11.1).
2. Don't change the attribute configuration of GAP and GATT.
3. Use TASK_RBLE as a task ID for user profile.
4. Set the attribute permissions and attribute value properly according to the service specification
5. Depending on the future version up, the attribute identification number, enumerations for handles and database array may increase or decrease its elements.
6. Set 128bit UUID in the least significant byte first and left justify.

7.5 Find Me Profile (Obsolete)

Due to the deprecation and withdrawal plan of the profile version by the Bluetooth SIG, product registration using this profile is no longer possible, so this section has been obsoleted.

For product registration, refer to "Bluetooth LE microcomputer/module Bluetooth qualification acquisition application note" (R01AN3177).

7.6 Proximity Profile (Obsolete)

Refer to "7.5".

7.7 Health Thermometer Profile (Obsolete)

Refer to "7.5".

7.8 Blood Pressure Profile (Obsolete)

Refer to "7.5".

7.9 HID over GATT Profile (Obsolete)

Refer to "7.5".

7.10 Scan Parameters Profile (Obsolete)

Refer to "7.5".

7.11 Heart Rete Profile (Obsolete)

Refer to "7.5".

7.12 Cycling Speed and Cadence Profile (Obsolete)

Refer to "7.5".

7.13 Cycling Power Profile (Obsolete)

Refer to "7.5".

7.14 Glucose Profile (Obsolete)

Refer to "7.5".

7.15 Time Profile (Obsolete)

Refer to "7.5".

7.16 Running Speed and Cadence Profile (Obsolete)

Refer to "7.5".

7.17 Alert Notification Profile (Obsolete)

Refer to "7.5".

7.18 Phone Alert Status Profile (Obsolete)

Refer to "7.5".

7.19 Location and Navigation Profile (Obsolete)

Refer to "7.5".

7.20 Vendor Specific

Vendor Specific (VS) provides the original extended features offered by Renesas.

7.20.1 Peak current consumption notification

7.20.1.1 Overview

The BLE MCU's current consumption increases during data transmission and reception. The peak current consumption notification feature provides functionality to report in advance when the current consumption increases in a system that includes the BLE software. Using this feature, other processing can be stopped momentarily in situations when the BLE MCU's current consumption rises to a high level, preventing multiple loads on the system.

Note: When using the RF slow clock (32kHz) for internal oscillation circuit, cannot be use this function.

7.20.1.2 Specifications

The peak current consumption notification feature is implemented by calling the peak current consumption notification and peak current consumption notification end callback functions from the BLE software. This feature can be enabled and disabled. If this feature is enabled, setting the value of following macro to 1.

Macro name: CFG_USE_PEAK

Specify the settings for this feature before the end of BLE initialization within the main function of arch_main.c, which is the main function of the BLE software. The times that can be set are 1 ms before, 2 ms before, and 4 ms before the current consumption reaches its peak. This feature uses a 12-bit interval timer for managing the set time. When using this feature, do not use the 12-bit interval timer with other features. If, in the case of multiple connections, data transmission and reception overlaps, callback functions might be called multiple times. As an example, callback functions might be called in the sequence of peak_start, peak_start, peak_end, and peak_end.

Also, when this feature is enabled, the power consumption of the entire system may increase.

7.20.1.3 Function specifications

The peak current consumption notification feature uses the following three functions:

(1) Peak current consumption notification setting function

This function is used to specify the settings for the peak current consumption notification feature. The notification feature can be enabled and disabled, and the notification time can be specified. Notification is not performed if this function is not called.

Settings cannot be made if this function is called after BLE initialization, and PEAK_ERROR_STATE is returned.

Table 7-28 Peak current consumption notification setting function

Function name	uint8_t peak_init (uint16_t peak_time)		
Overview	Peak current consumption notification setting function		
Description	Specifies the settings for the peak current consumption notification feature.		
Parameters	uint16_t peak_time	PEAK_TIME_OFF	Don't execute notification.
		PEAK_TIME_1	Start notification 1 ms before peak.
		PEAK_TIME_2	Start notification 2 ms before peak.
		PEAK_TIME_4	Start notification 4 ms before peak.
Return values	PEAK_OK	Success	
	PEAK_ERROR_PARM	Parameter error	
	PEAK_ERROR_STATE	Setting not possible	

(2) Peak current consumption notification function

This function is called from the BLE software at the time set by the peak current consumption notification setting function. The processing of this function must be defined by the user. Other MCUs can be notified by executing external port output from within this function. However, a limitation of the BLE software requires that the least possible amount of processing be executed within this function. If not using The peak current consumption notification, make this function empty.

Table 7-29 Peak current consumption notification function

Function name	void peak_start (void)
Overview	Peak current consumption notification function
Description	This function is called from the BLE software at the time set by the peak current consumption notification setting function. (Callback function)
Parameters	None
Return values	None

(3) Peak current consumption end notification function

This function is called from the BLE software upon completion of the transmission/reception that corresponds to the current consumption peak. The processing of this function must be defined by the user. Use this function to perform actions such as stopping the output started by the peak current consumption notification function. However, like the peak current consumption notification function, a limitation of the BLE software requires that the least possible amount of processing be executed within this function. If not using the peak current consumption notification, make this function empty.

Table 7-30 Peak current consumption end notification function

Function name	void peak_end (void)
Overview	Peak current consumption end notification function
Description	This function is called from the BLE software when the current consumption peak is exceeded. (Callback function)
Parameters	None
Return values	None

7.20.2 Sleep

7.20.2.1 Overview

The sleep feature is used to place the BLE MCU in a low power consumption state when the BLE MCU is idle, such as during breaks between sending and receiving data, for the purpose of lowering the power consumption. There are two states of lowered power consumption, the Sleep state, during which the MCU block and RF block in the BLE MCU are placed in STOP mode and HALT mode, respectively, and the DeepSleep state, during which both the MCU block and RF block are placed in the STOP mode.

7.20.2.2 Operation overview

In a state in which operations such as data transmission and reception are not being performed, the MCU and RF modules both enter an idle state. In this state, unnecessary power gets consumed, so the MCU and RF modules are placed in a low power consumption state. This transition to a low power consumption state is made on the judgment of the BLE software. Whether the Sleep mode or DeepSleep mode is selected is determined based on the BLE MCU's idle time information. Therefore, state transition processing does not need to be executed by the user application.

However, in order to operate the user application, you do not want the BLE MCU to enter into Sleep or DeepSleep state, return false in the Check sleep OK function. This Check sleep OK function is called immediately before the BLE software entering into Sleep state. If this function returns false, the BLE software stops entering into Sleep state.

The specification of this function is shown below.

Table 7-31 Check sleep OK function

Function name	bool sleep_check_enable (void)	
Overview	Checks if state transition to Sleep state is allowed	
Description	Checks if state transition to Sleep state is allowed	
Parameters	None	
Return values	true	State transition to Sleep state is allowed
	false	State transition to Sleep state is not allowed

7.20.2.3 Cautions on application implementation

The BLE software calculates idle time based on the data transmission and reception frequency and the frequency of communications to establish connections. Therefore, when data is transmitted and received or connections are established frequently, the BLE MCU is not placed in the Sleep state or the DeepSleep state. As a result, the system's power consumption may increase in some cases.

The sleep_cont functions in arch_main.c should be used without change. Since these functions are important for operating the Sleep function, the Sleep function may not operate normally when they are altered.

7.20.3 Reset processing

If there is a memory shortage or a hardware error occurs, the BLE software executes a hardware reset.

If the reset cause occurs, the function indicated below is called. Currently, this function forcibly resets the hardware by executing an illegal instruction.

Folder: \Renesas\BLE_Software_Ver_X_XX\RL78_G1D\Project_Source\renesas\src\arch\rl78

File: arch_main.c

Function: void platform_reset(uint32_t error)

7.20.4 Original features provided by rBLE API

This section describes the original Renesas features offered in the rBLE API.

7.20.4.1 Bluetooth device address write

The Bluetooth device address (BD address) write feature allows BD addresses to be written by accessing the non-volatile data flash area. BD addresses refers to addresses used to uniquely identify Bluetooth devices. The BD addresses used in BLE consist of public addresses and random addresses, but this feature has been designed with public addresses in mind.

Access to the data flash requires a data flash driver. The data flash driver is provided as sample code.

The operation during BD address writing is described below.

1. Set access start to the data flash by the rBLE API.
2. Set the BD address to be written by the rBLE API.
3. The BLE stack calls the BD address write feature.
4. The BD address write feature writes the BD address to the data flash.
5. Set access stop to the data flash by the rBLE API

7.20.4.2 Direct Test Mode

The Direct Test Mode is a feature prescribed in the BLE standard and provides functionality for testing the transmission/reception capacity of the BLE MCU. It has the following main features:

- A reception/transmission Direct Test Mode start command and Direct Test Mode stop command are provided by the rBLE API.
- When launched, the reception Direct Test Mode executes packet reception in all 625 us slots.
- When launched, the transmission Direct Test Mode executes packet transmission in all 625 us slots.
- The reception/transmission Direct Test Mode continues operating until the Direct Test Mode stop command is received.
- Upon reception of the stop command, Direct Test Mode is terminated and completion is reported by an event.
- If a reception Direct Test Mode operation was in progress when the stop command was received, the number of receive packets is returned by an event parameter.

For details about Direct Test Mode, see *Bluetooth Core Specification v4.2 [Vol. 6], Part F*.

7.20.4.3 Extended Direct Test Mode

The extended Direct Test Mode offers functionality for the smooth operation of Direct Test Mode. It has the following main features:

- A parameter setting command that defines the Direct Test Mode operation is provided by the rBLE API.
- The number of packets received/sent in Direct Test Mode can be specified by using a parameter of the parameter setting command.
- Following the start of Direct Test Mode, Direct Test Mode terminates automatically when the specified number of packets is reached, and completion is reported by an event
- If the specified number of packets is 0, reception/transmission continues until a command to stop Direct Test Mode is received, per the BLE standard.

- Even if the specified number of packets is other than 0, the operation can be aborted by using the Direct Test Mode end command.
- The number of packets to be received or sent specified by using the parameter setting command is held even after Direct Test Mode execution ends.

* In the DirectTestMode operation, it is possible to report a little bit more number of received packets than specified, because it is possible to receive extra packets while processing received packets.

7.20.4.4 Burst transfer

Burst transfer provides functionality to continuously receive/send data for devices that are designed with current consumption measurement in mind. It has the following main features:

- The command for specifying burst transfer is the same as the extended Direct Test Mode's parameter setting command.
- During the execution of reception burst transfer, the Direct Test Mode's standby time becomes infinite and thus the device receives data continuously.
- During the execution of transmission burst transfer, the packet payload length of Direct Test Mode becomes infinite and thus the device transmits continuously.
- Reception/transmission burst transfer is started by execution of the reception/transmission Direct Test Mode start command.
- Burst transfer continues until the Direct Test Mode stop command is received.
- Even if the specified number of packets in Direct Test Mode is other than 0, burst transfer continues until the Direct Test Mode stop command is received.

7.20.4.5 Continuous carrier wave (CW) output

Continuous carrier wave (CW) output provides functionality to output continuous carrier waves (CWs) in accord with the test items of the technology conformance inspection based on the Radio Law. It has the following main features:

- The command for specifying continuous carrier wave (CW) output is the same as the extended Direct Test Mode's parameter setting command.
- During the execution of continuous carrier wave (CW) output, the packet payload length of Direct Test Mode becomes infinite and thus the device transmits continuously.
- Continuous carrier wave (CW) output is started by execution of the transmission Direct Test Mode start command.
- Continuous carrier wave (CW) output continues until the Direct Test Mode stop command is received.
- Even if the specified number of packets in Direct Test Mode is other than 0, continuous carrier wave (CW) output continues until the Direct Test Mode stop command is received.

7.20.4.6 RSSI reading in reception Direct Test Mode

RSSI reading in reception Direct Test Mode allows the RSSI value to be acquired while reception Direct Test Mode is being executed. It has the following main features:

- The RSSI acquisition command for use in reception Direct Test Mode is provided by the rBLE API.
- The RSSI value acquired during reception Direct Test Mode is reported by an event.
- The RSSI value can be acquired in the period from when reception Direct Test Mode starts to immediately before a

normal packet is received after exiting Direct Test Mode.

7.20.4.7 Tx power selection

Tx power selection allows the Tx power setting to be changed. The Tx power can be set separately for the advertising/scanning/initiating handle or connection handle. It has the following main features:

- The Tx power setting command is provided by the rBLE API.
- The Tx power can be set separately for each connection handle when connected or during advertising/scanning/initiating.

The initial setting is RBLE_VS_TXPW_HIGH (0dBm). It is possible to change Tx power in Advertising/Scanning/Initiating before execution. And it is possible to change Tx power in Master/Slave connection after a connection.

Refer to Bluetooth Low Energy protocol stack API reference manual: Basics.

7.20.4.8 GPIO of RF part

The GPIO[3:0] pins of RF part can be accessed as General Purpose Input/Output.

When the RF part has changed in Deep Sleep mode, GPIO[2:0] is reset to the input, and GPIO[3] is reset to alternate function output, the output value cannot be maintained. When the RF part to wake-up from Deep Sleep mode, recover the output value set in this function.

Note: GPIO[3] pin is reset to the output when the RF part is change to the Deep Sleep mode. Please be careful if you want to use the GPIO[3] pin as the input port.

7.20.4.9 Adaptable mode

The signal strength measured at the packet reception, and changes to the optimal power mode. This feature is only available in the Slave-Role.

This feature has three modes of the RF low-power mode (signal strength is large) and the RF normal mode (signal strength is medium) and the RF high-performance mode (signal strength is small). If this feature is enabled each mode will automatically change state

- RF low-power mode: When the signal strength of reception packet is large, reduces the peak current consumption. The distance between the master and slave is near-range or middle-range, this mode is suitable for applications such as maintain a high level signal strength.
- RF normal mode: When the signal strength of reception packet is medium, well-balanced operation mode between the RF low-power and the RF high-performance. The distance between the master and slave is near-range or middle-range, this mode is suitable for applications such as maintain a certain level signal strength. When adaptable feature is disabled, will operate in this mode.
- RF high-performance mode: When the signal strength of reception packet is small, operation to maximize the RF characteristics. The distance between the master and slave is middle-range or far-range, this mode is suitable for applications such as a low level signal strength.

You can control enabling or disabling the adaptable mode feature by BLE Software. Also, you can select inform the mode change or not when the adaptable mode feature is set enabling.

The default setting of the adaptable mode feature is disabled. If you want to use, please enable this feature. However, this feature shall disabled in the Master-Role.

For more setting information, refer to the Bluetooth Low Energy protocol stack API reference manual: Basics.

7.20.4.10 Power Control of RF part

The power supplied in RF part can be controlled. The power control commend is as follows.

- RF power supply OFF: The power supply to RF part will be stopped. If the clock output of 16.384 kHz or 32.768 kHz from PCLBUZ0 is enable, the clock output will be stopped during RF power-off.
- RF power supply ON (DC-DC enable): The on-chip DC-DC converter will be enable. Then, the power supply to RF part will be started.
- RF power supply ON (DC-DC disable): The on-chip DC-DC converter will be disable. Then, the power supply to RF part will be started.

Note: In use of this command, please note the following points.

- If the power control command is executed regardless of the ON / OFF setting, kernel events, messages and timer queue of RWKE are initialized.
- The following features cannot be used during RF power-off.
  - ✧ The high-speed clock output from RF part
  - ✧ GPIO of RF part
  - ✧ The timer management functionality of RWKE
  - ✧ The error recovery capabilities of RSCIP
- GAP reset is needed after RF power-on. In addition, only GAP reset is acceptable during RF power-off. The previous RF power-on setting for the use of the on-chip DC-DC converter will be inherited.

8. EEPEOM Emulation Library

8.1 About the EEPROM Emulation Library

The BLE software uses the EEPROM Emulation Library in order to store the BLE-MCU BD address into the data flash. BLE software does not guarantee behaviors of all functions and all versions of the EEPROM Emulation Library. Following version is used for the BLE software test. When another version of the EEPROM Emulation Library used or user function added, you may change the data flash driver.

Version: EEPROM Emulation Library RL78 EEL-T01 V1.13(CS+ for CA,CX)/ RL78 EEL-T02 V1.01(CC-RL)

This Version of the EEPROM Emulation Library is made from the following two files.

Note: Different file to be downloaded by the development environment and compiler.

- RENESAS_EEL_RL78_T01E_V1.20.zip / RENESAS_FDL_RL78_T01E_V1.20.zip (CS+ for CA,CX)
- RENESAS_EEL_RL78_T02E_V1.20.zip / RENESAS_FDL_RL78_T02E_V1.30.zip (CC-RL)

The EEPROM Emulation Library is not part of the BLE software and is a separate product. Please check the supplied conditions before use.

8.2 About setting for the EEPROM emulation library

In case of using the EEPROM Emulation Library, this library restricts to the RAM area. You must set for the RAM area with the linker file. Refer to User Manual: EEPROM Emulation Library EEL-T01(R01US0128ED0101) and User Manual: Data Flash Access Library FDL-T01(R01US0034ED0101) for more information. When using the RL78 EEL-T02, refer to User Manual: EEPROM Emulation Library Type T02 (Tiny) (R01US0070ED0105) and User Manual: Data Flash Access Library Type T02 (Tiny) (R01US0061ED0120) for more information.

8.3 Notes on using the EEPROM emulation library

The user application can use the data flash area, using the EEPROM emulation library. To learn how to use it, refer to the application note of EEPROM emulation library.

Because writing to or reading from the data flash using EEPROM emulation library occupies the long processing time, it is possible to affect other processes. Thus, design your system to do the process using EEPROM emulation library in the period in which communication is not performed (e.g., immediate after the power on)..

9. Code Flash Library

9.1 About the Code Flash Library

The BLE software uses the Code Flash Library in order to write the software code into the code flash for FW update function. BLE software does not guarantee behaviors of all functions and all versions of the Code Flash Library. Following version is used for the BLE software test. When another version of the Code Flash Library used or user function added, you may change the code flash driver.

Version: Selfprog Library RL78 T01 V1.20

This Version of the Code Flash Library is made from the following file.

- RENESAS_FSL_RL78_T01E_V1.20.zip

The Code Flash Library is not part of the BLE software and is a separate product. Please check the supplied conditions before use.

[Note] For FW update, please refer to Bluetooth Low Energy Protocol Stack Sample Program Application Note.

9.2 About setting for the Code Flash library

In case of using the Code Flash Library, this library restricts to the RAM area. You must set for the RAM area with the linker file. Refer to Application Note: Flash Self-programming Library Type T01(R01US0016ED0105) for more information.

9.3 Notes on using the Code Flash library

To learn how to use it, refer to the application note of Code Flash library.

10. Note on Writing User Application

10.1 Note on RWKE Timer Management Function

RWKE is the basic software that is designed to operate a BLE protocol stack. When a user creates an application on the BLE-MCU, you can use the features of RWKE. However, if you want to use the RWKE timer management function, use on the RWKE task. If you want to use from interrupt service routine, etc., the operation is not guaranteed. For more information, refer to the chapter of RWKE in the Bluetooth Low Energy Protocol Stack API Reference Manual: Basic.

10.2 Interrupt disabled time of the task and the interrupt handler

Please shorten the processing time of the task and the interrupt handler because the communications processing of BLE may be affected. The recommended value is within 1msec.

10.3 Data transmission of large size data

When BLE software is required transmission of multiple data in an interval, it performs two or more transmission in one event. When two or more transmission and reception are performed in 1 time of an event, connection of other devices is influenced and it may become impossible to maintain connection. Therefore, please do not make two or more transfer requests within an interval.

When two or more transmission and reception are performed in 1 time of an event, set up the interval value different from other connection devices. For example when the interval of connection which does not perform two or more transmission and reception is set to 500 ms, the interval of the connection which may transmit two or more times is set as 520 ms. Please set it so that a common multiple with 500ms becomes the big value. In this way, it is possible to reduce the number of communication failure.

10.4 Performance of BLE MCU

If data is received between a master device and a slave device, a memory will be dynamically allocated using the memory control function of RWKE. In order to prevent memory shortage, please design to complete processing by the next interval.

10.4.1 Modem Configuration

In the case of the Modem Configuration, bottleneck of data processing is serial communication. When a transfer rate is 4800bps and the operating frequency of BLE-MCU is 8MHz, the processing performance of BLE-MCU is about 100 bytes in 1 second. Therefore, the total data volume between a Master device and Slave devices are less than 100 bytes in 1 second. The example of a setting of an interval is shown below.

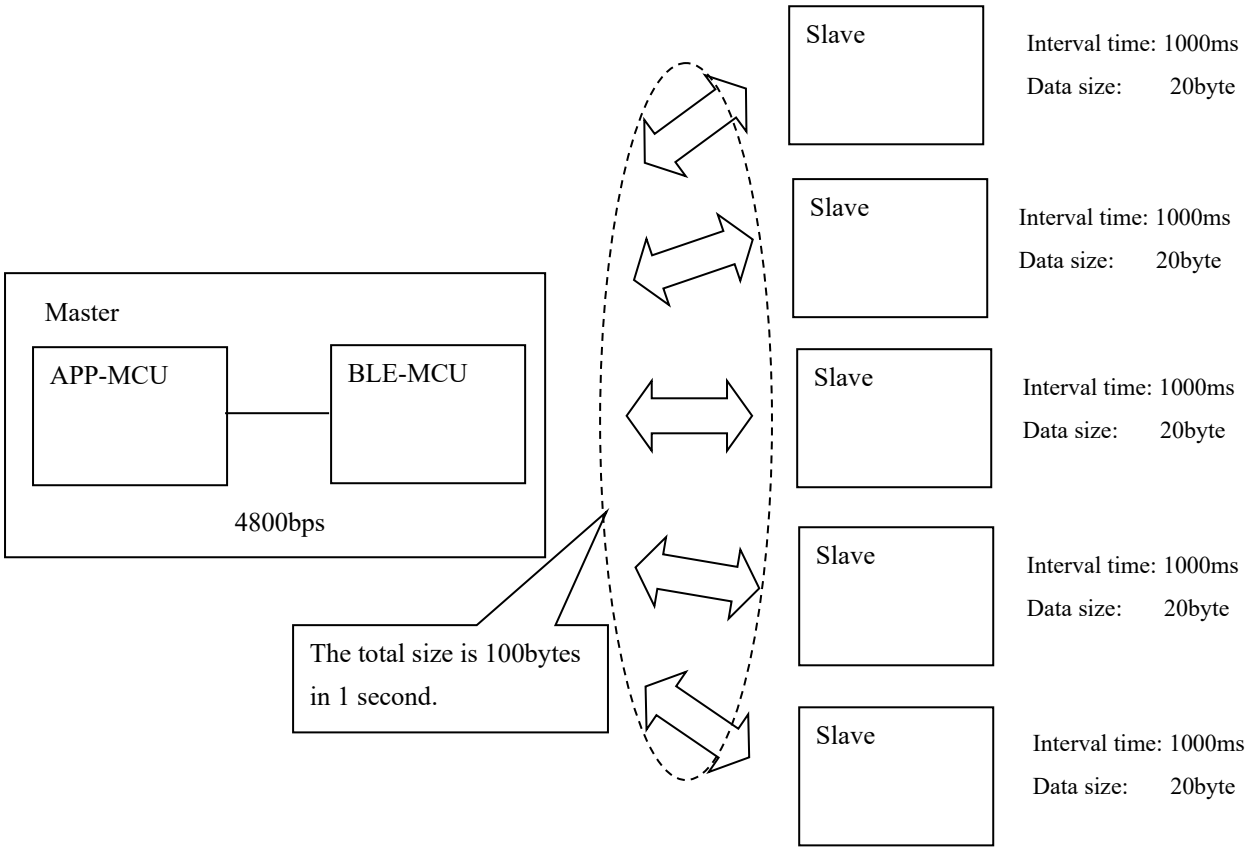


図 10-1 In case of the interval time where 5 connections of everything is same.

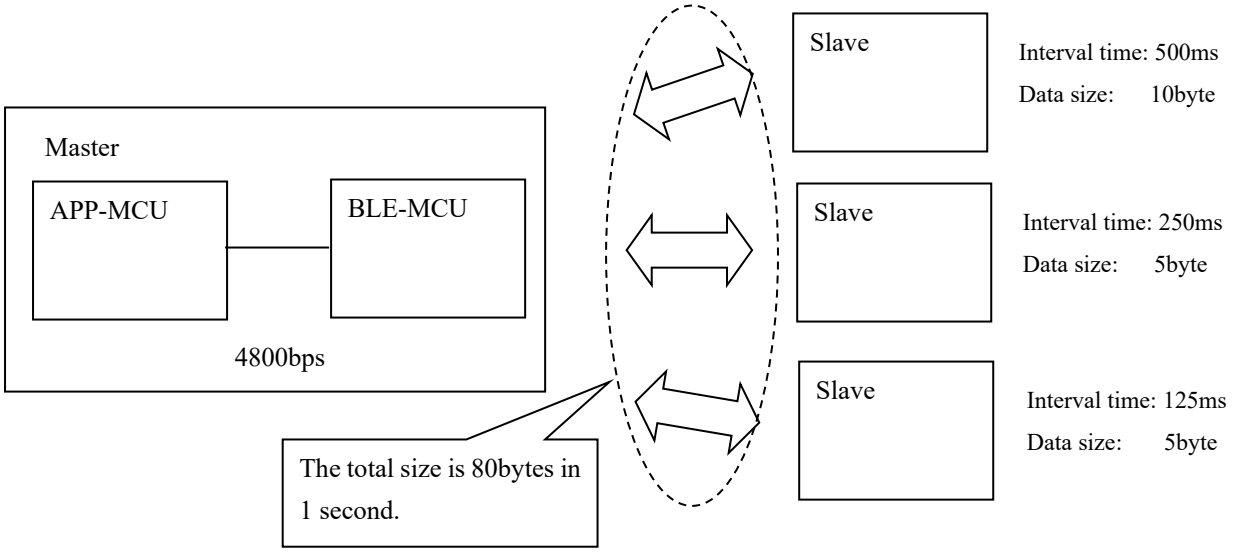


図 10-2 In case of the interval time where 3 connections of everything is different.

11. Implementation of FW Update Feature

11.1 The FW Update Feature

The FW Update feature can be updated a firmware of the following layer through the BLE radio.

- GATT based Profile
- User application

Considerations for implementation of the FW update feature is described hereinafter.

Note: The following descriptions are used in this chapter.

Receiver device: A device that receives updated data

Sender device: A device that sends update data

11.2 Function required for FW Update

The FW Update feature requires implementation of the following functions.

➢ For Receiver device

- Writing function to the code flash
- Data transmission and reception profile
- Application for update control

➢ For Sender device

- Data transmission and reception profile
- Application for update control

The detail and implementation example are described in the following.

11.2.1 Writing function to the code flash

This function is required on the Receiver device. This controls the code flash in order to write the update data that has been received from the Sender device.

The control of the code flash requires the code flash library. Please obtain according to ‘9.Code Flash Library’.

Update the firmware by repeating the erasing and writing in one block (1024byte) unit according to the code flash specifications.

11.2.2 Data transmission and reception profile

This is a custom profile for sending update data from Sender device to Receiver device.

Please implement the functions required for product system with reference to the examples and sample source code shown in Figure 11-1.

Data transmission and reception profile cannot be updated by the FW Update feature unlike GATT based profiles. Also cannot be updated the functions that operate during the update.

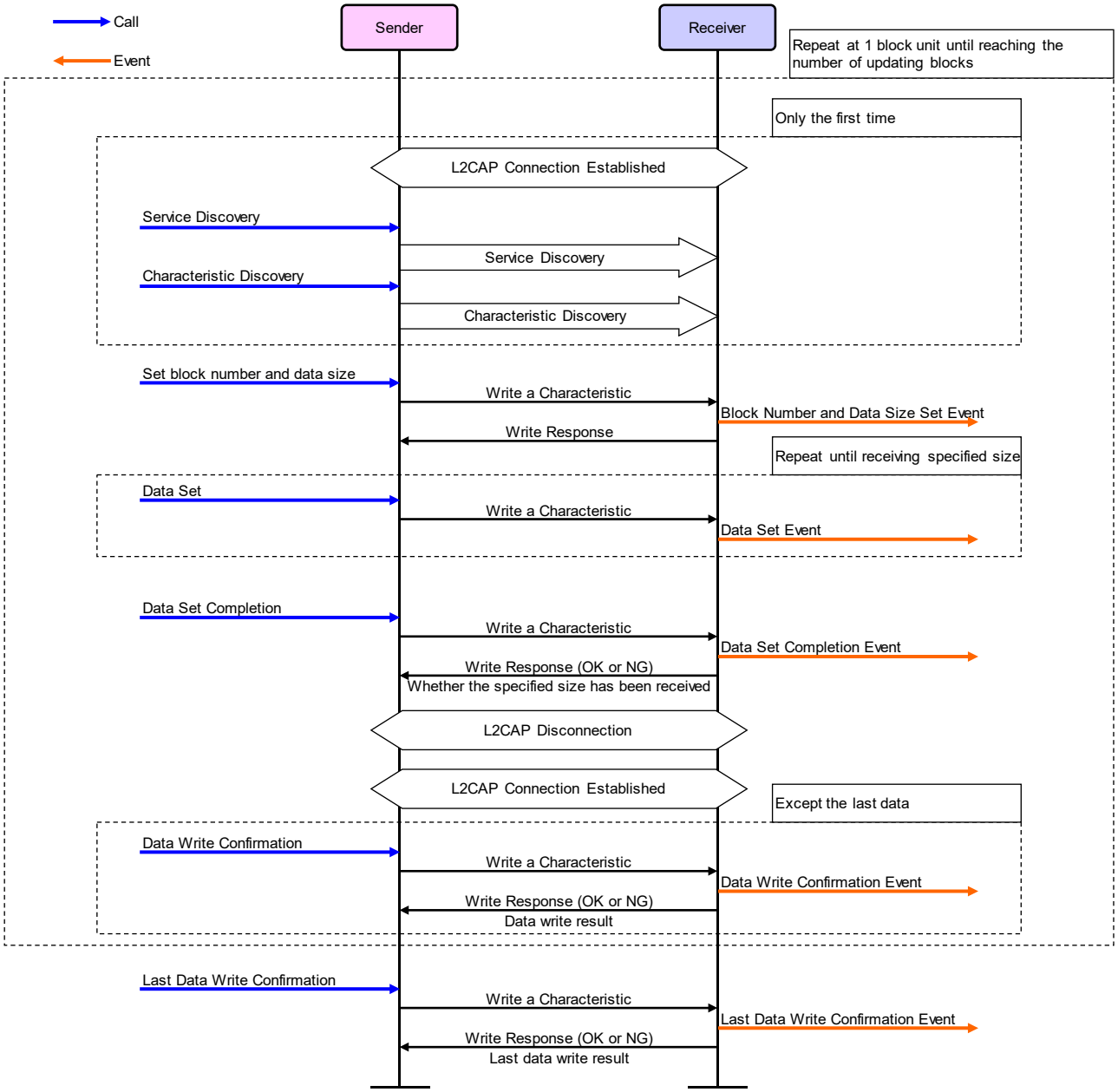


Figure 11-1 Example of update data transmission and reception sequence

11.2.3 Application for update control (for Receiver device)

This function is required on the Receiver device. This controls the receiving update data and writing to code flash.

This application cannot be updated by the FW Update feature as is the case with the data transmission and reception profile.

The FW Update feature uses the boot swap function of RL78/G1D.
Refer to the User's Manual: Hardware (R01UH0515) for this function.

The implemented processing is described in the following.

(1). Erase the code flash block 4 to 7 (0x01000 to 0x01FFF) when starting the update.
(2). Reset the RL78/G1D by a reset function (FSL_ForceReset) in the code flash library.
(3). When writing data to the block 7, write to 0x01FFE by adding 1 to the value of 0x00FFE. The number of updates are managed by using the 0x00FFE and 0x01FFE.
(4). The block 42 (0x0A800 ~ 0x0ABFF) and block 43 (0x0AC00 ~ 0x0AFFF) is used to switch the area, depending on the number of updates. Refer to 11.3.1 for details.
(5). Run the boot flag switching function (FSL_InvertBootFlag) and reset function (FSL_ForceReset) after writing all the update data in order to switch the boot cluster.

An example of the FW Update outline processing flow and data writing flow in the Receiver device are shown in Figure 11-2 and Figure 11-3.

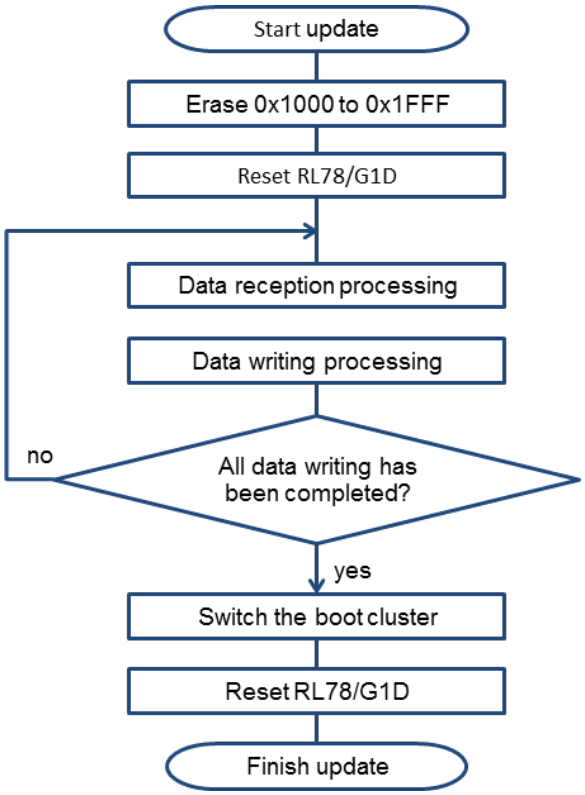


Figure 11-2 Example of the FW Update outline processing flow

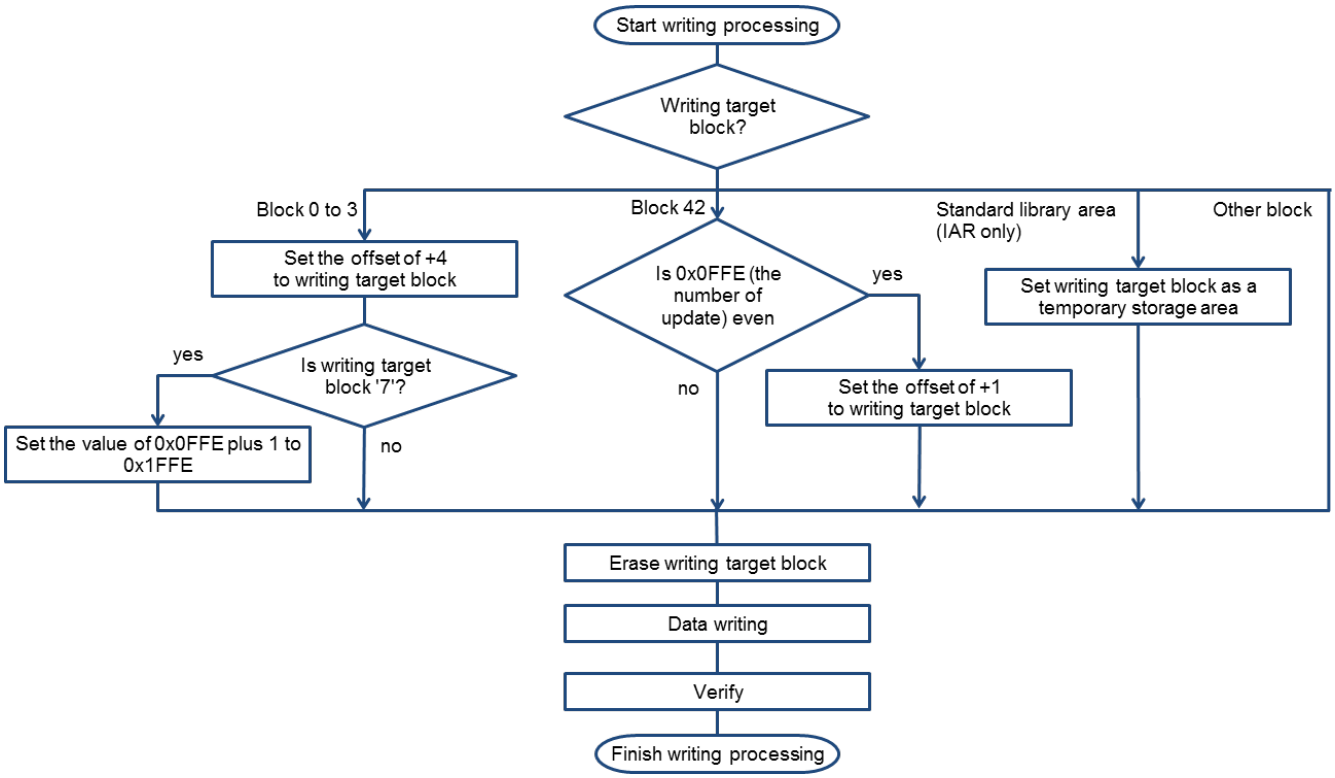


Figure 11-3 Example of the data writing flow

11.2.4 Application for update control (for Sender device)

This function is required on the Sender device. This controls the transmitting update data.

To fit into the system of Sender device, please implement the creation of the update data, the controlling the data transmission and reception profile and the management of the update data.

11.3 Limitation and Special Processing

The limitation and special processing to implement the FW Update feature are described in this section.

11.3.1 Area switching control

Part of the memory area is used while switching by FW update feature.

There is the target memory area and the non-target memory area for updating.

Though the area that is stored the code to run during update is the non-target memory area basically, there are target areas to be updated despite running during update. In this target area, it is necessary to reserve the memory space for updating aside from the area running for the FW Update feature.

Switching of these areas are managed by using the information of the number of updates.

The overview of area switching control is shown in Figure 11-4.

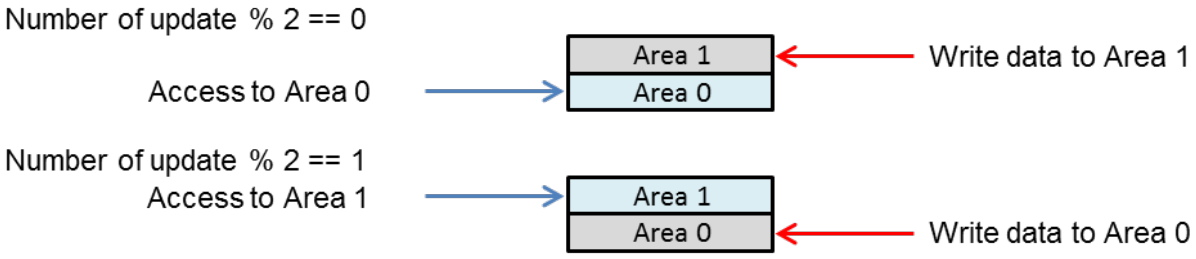


Figure 11-4 Overview of area switching control

The separately reserved area is updated during the FW Update feature running. Then the program of the updated area will be enabled after the update is complete.

11.3.2 Limitation for FW Update feature implementation

Limit of when implementing the FW Update feature to the Receiver device is as follows.

(1). Do not run during updating except the following function.
- Data transmission and reception profile
- Application for update control
- BLE protocol stack (except GATT based profile)
- Code flash library

(2). Do not change the runtime library and standard library that are forced to link. For measures of forcibly link the runtime library and the standard library, please refer to the Sample program application note (R01AN1375) ‘6.8.3. Notes of making FW Update Environment’.

(3). Do not use the initialized variable in the application for update control and the data transmission and reception profile.

11.3.3 Update target area and User RAM area

A list of update target area and user RAM area is shown in Table 11-1.

Table 11-1 Update target area and user RAM area

Environment	Configuration	Area	Usage
CS+ for CA,CX (CA78K0R)	Embedded	0x00000 - 0x01FFF	Boot cluster 0/1
		0x04400 - 0x0A7FF	Area for variables allocated in ROM
		0x0B000 - 0x0DFFF	Functions to be allocated in the NEAR area, Interrupt function, Area for storing the initial value of initialized variable
		0x31400 - 0x3FBFF	Area for the program code
		0xFBD10 - 0xFFE1F	Area for variables allocated in RAM Including a stack memory
	Modem	0x00000 - 0x01FFF	Boot cluster 0/1
		0x04400 - 0x0A7FF	Area for variables allocated in ROM
		0x0B000 - 0x0DFFF	Functions to be allocated in the NEAR area, Interrupt function, Area for storing the initial value of initialized variable
		0x33C00 - 0x3FBFF	Area for the program code
		0xFC210 - 0xFFE1F	Area for variables allocated in RAM Including a stack memory
e^2 studio / CS+ for CC (CC-RL)	Embedded	0x00000 - 0x01FFF	Boot cluster 0/1
		0x04400 - 0x0A7FF	Area for variables allocated in ROM
		0x0B000 - 0x0DFFF	Functions to be allocated in the NEAR area, Interrupt function, Area for storing the initial value of initialized variable
		0x30000 - 0x3FBFF	Area for the program code
		0xFBD20 - 0xFFE1F	Area for variables allocated in RAM Including a stack memory
	Modem	0x00000 - 0x01FFF	Boot cluster 0/1
		0x04400 - 0x0A7FF	Area for variables allocated in ROM
		0x0B000 - 0x0DFFF	Functions to be allocated in the NEAR area, Interrupt function, Area for storing the initial value of initialized variable
		0x30000 - 0x3FBFF	Area for the program code
		0xFC220 - 0xFFE1F	Area for variables allocated in RAM Including a stack memory

Note: The above values is the default setting when memory allocation is not changed.

12. HCI Packet Monitoring Feature

HCI packet monitoring feature is used to display the internal information of BLE protocol stack as the data of HCI packet format in virtually. By using this feature, you can easily confirm the Advertising parameters, connection parameters, and sending and receiving data in real time.

12.1 Functional Composition of the HCI Packet Monitoring

Figure 12-1 shows functional composition of the HCI packet monitoring. In this feature, outputs the internal information of BLE protocol stack from UART1. In the PC, internal information of received BLE protocol stack is converted to HCI packet format and displayed on Wireshark.

Note: Wireshark is a network protocol analyzer for Unix/Linux, Mac OS X and Windows. It supports Bluetooth packet analysis, this feature uses it.

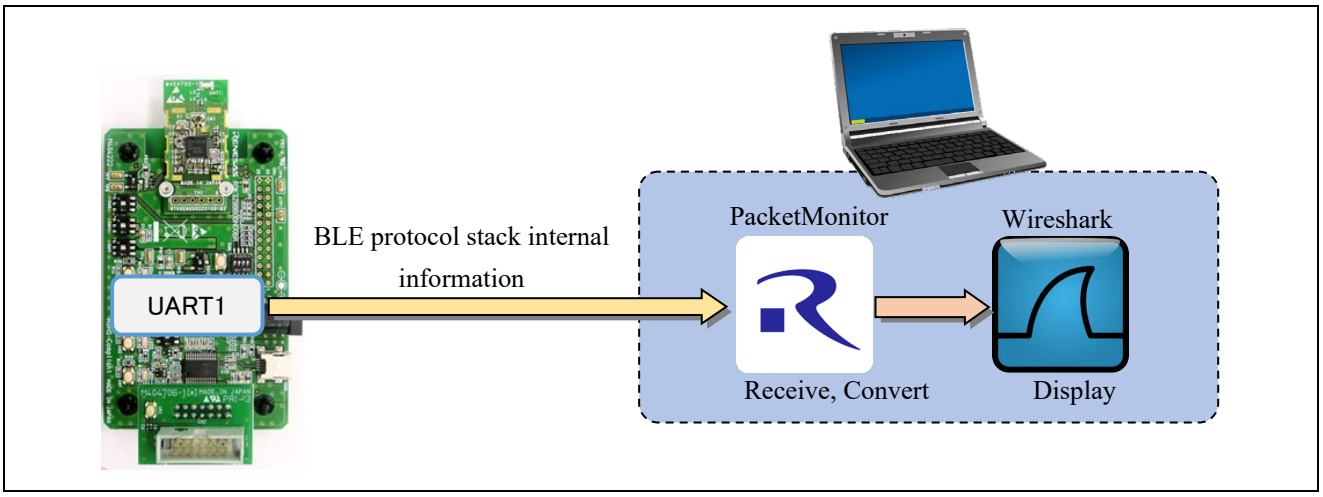


Figure 12-1 HCI packet monitoring functional composition

The following hardware software is needed to use this function.

- Hardware
  - UART←→RS-232C(USB) converter cable
    - baud rate : At least 1Mbps
    - TTL level : 3.3V
- Software
  - Wireshark (available from https://www.wireshark.org/)

Furthermore, the operation of this function was confirmed in the following environment.

- Hardware
  - UART←→RS-232C(USB) converter cable
    TTL-232RG-VREG3V3-WE (http://www.ftdichip.com/Products/Cables/USBTTLSerial.htm)
- Software
  - Windows 7 Enterprise Service Pack1 (32-bit)
  - Wireshark Version 1.12.7 (32-bit)

12.2 Enabling the HCI Packet Monitoring Feature

The HCI packet monitoring feature can be enabled by defining the following macro in the project compile options.

Macro name: `CFG_PKTMON (default: noCFG_PKTMON)`

Note1: This feature is intended to be used for debugging. This feature is not applicable to any other purposes.
Note2: When using UART1 as a communication interface with the APP MCU in the Modem configuration, or if the user application uses UART1, it isn't possible to use this feature.

12.3 How to Use the HCI Packet Monitoring Feature

This section describes about the preparations for using and how to use the HCI packet monitoring feature.

12.3.1 Preparations

To use the HCI packet monitoring feature, it requires following preparations.

- RL78/G1D

  Create a HEX file by enabling the HCI packet monitoring feature, and write it into RL78/G1D.

  About enabling HCI packet monitoring feature, refer to *Section 12.2*.

- Evaluation board

  Connect the UART←→RS-232C(USB) converter cable to evaluation board. The connection signals of the evaluation board and the converter cable are shown in Table 12-1.

Table 12-1 Evaluation board pins and Converter cable signals

Evaluation board			Converter cable signal
connector	pin	signal	
CN4	4	GND	GND
CN4	12	TXD1	RXD

- PC

  Obtain the Wireshark from the following, and install.

  https://www.wireshark.org/

12.3.2 How to Use

By performing the procedures described below, you will be able to monitor the HCI packets.

(1) Launch the PaketMonitor

Launch the PacketMonitor of the executable file that is appropriate for your development environment.

CS+ for CA, CX (CA78K0R) : `PacketMonitor_for_CA.exe`

Other than the above : `PacketMonitor.exe`

Executable file of PacketMonitor has been stored in the following folder.

\Renesas\BLE_Software_Ver_X_XX\PacketMonitor

(2) Select the Wireshark.exe

Once open the PacketMonitor application, the dialog window pop up as shown in Figure 12-2. Select the Wireshark.exe that installed in advance. Then click [Run] button.

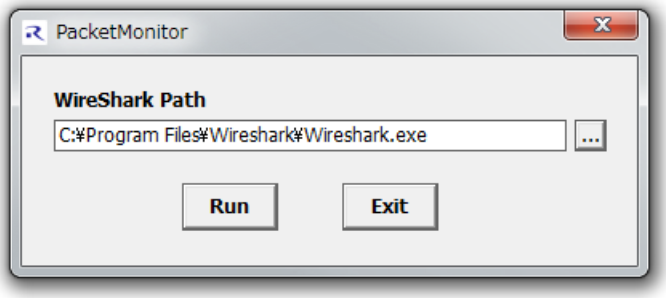


Figure 12-2 Select the Wireshark.exe

(3) Setting of serial port

When click the [Run] button, the Serial port settings dialog window pop up as shown in Figure 12-3.

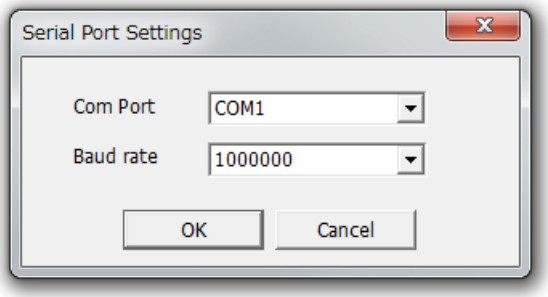


Figure 12-3 Serial port settings dialog

Select the serial port which connected UART←→RS-232C(USB) converter cable. Then, select one of the following baud rate which according to operating frequency of the RL78/G1D. Next click [OK] button.

4MHz : `500,000bps`

8, 16, 32MHz : `1,000,000bps`

Note: Baud rate setting in the RL78/G1D is fixed internally. It can't be changed.

(4) Launch the Wireshark

When setting of the serial port is completed by pressing the [OK] button, automatically Wireshark is launched. Please wait until the Wireshark to fully boot.

(5) Power on of RL78/G1D

After booting the Wireshark, capture of a HCI packet starts by supplying power to the RL78/G1D. After supplying power to the RL78/G1D, behavior of application is possible as usual.

12.4 HCI Packet Monitoring Screen

Figure 12-4 shows Wireshark's window during HCI packet monitoring (default setting). In Wireshark, HCI packets are color-coded for each type. It also supports the protocol analysis of ATT and SMP.

In upper pane, the packet exchanged between the Host-Controller is displayed in chronological order. In middle and lower pane, details of the packet selected at the upper pane is displayed.

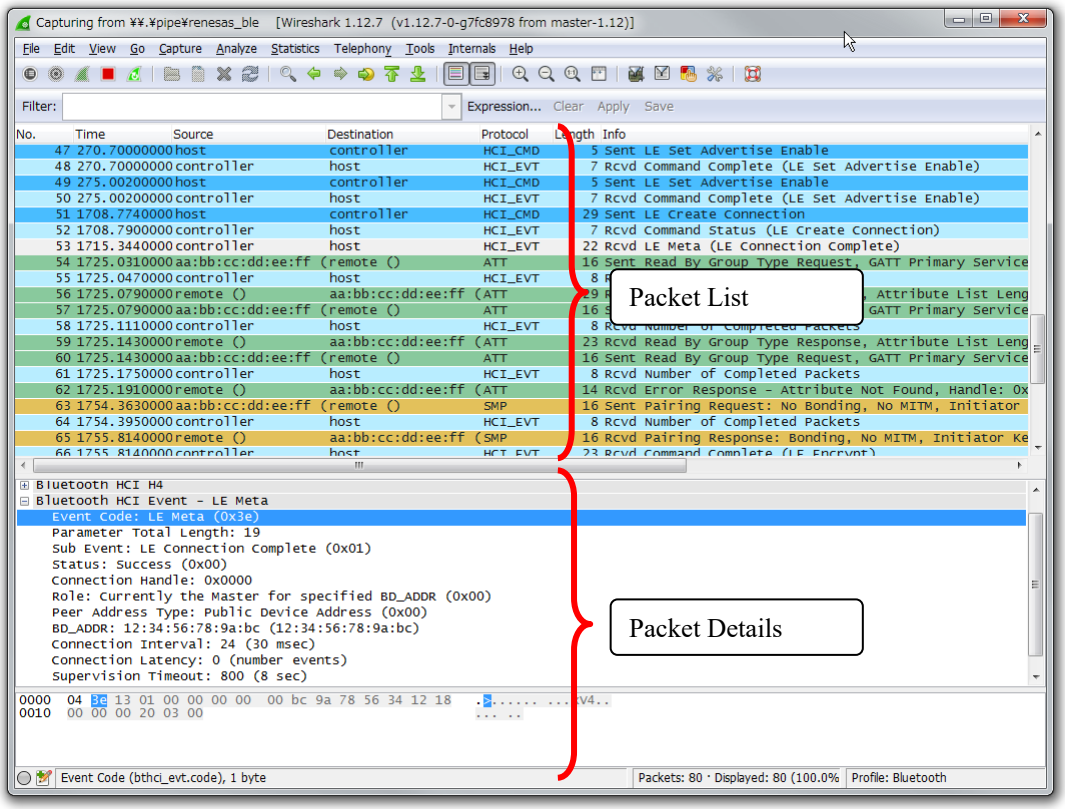


Figure 12-4 Wireshark's window during HCI packet monitoring

For details about HCI Commands and Events, refer to *Bluetooth Core Specification v4.2[Vol. 2], Part E Section 7. HCI Commands and Events*.

For details about packet format of ATT, refer to *Bluetooth Core Specification v4.2[Vol. 3], Part F Section 3.3 Attribute PDU*.

For details about packet format of SM, refer to *Bluetooth Core Specification v4.2[Vol. 3], Part H Section 3 Security Manager Protocol*.

Appendix A Referenced Documents

1. Bluetooth Core Specification v4.2, Bluetooth SIG
2. Find Me Profile Specification v1.0, Bluetooth SIG
3. Immediate Alert Service Specification v1.0, Bluetooth SIG
4. Proximity Profile Specification v1.0, Bluetooth SIG
5. Link Loss Service Specification v1.0, Bluetooth SIG
6. Tx Power Service Specification v1.0, Bluetooth SIG
7. Health Thermometer Profile Specification v1.0, Bluetooth SIG
8. Health Thermometer Service Specification v1.0, Bluetooth SIG
9. Device Information Service Specification v1.1, Bluetooth SIG
10. Blood Pressure Profile Specification v1.0, Bluetooth SIG
11. Blood Pressure Service Specification v1.0, Bluetooth SIG
12. HID over GATT Profile Specification v1.0, Bluetooth SIG
13. HID Service Specification v1.0, Bluetooth SIG
14. Battery Service Specification v1.0, Bluetooth SIG
15. Scan Parameters Profile Specification v1.0, Bluetooth SIG
16. Scan Parameters Service Specification v1.0, Bluetooth SIG
17. Heart Rate Profile Specification v1.0, Bluetooth SIG
18. Heart Rate Service Specification v1.0, Bluetooth SIG
19. Cycling Speed and Cadence Profile Specification v1.0, Bluetooth SIG
20. Cycling Speed and Cadence Service Specification v1.0, Bluetooth SIG
21. Cycling Power Profile Specification v1.0, Bluetooth SIG
22. Cycling Power Service Specification v1.0, Bluetooth SIG
23. Glucose Profile Specification v1.0, Bluetooth SIG
24. Glucose Service Specification v1.0, Bluetooth SIG
25. Time Profile Specification v1.0, Bluetooth SIG
26. Current Time Service Specification v1.0, Bluetooth SIG
27. Next DST Change Service Specification v1.0, Bluetooth SIG
28. Reference Time Update Service Specification v1.0, Bluetooth SIG
29. Alert Notification Service Specification v1.0, Bluetooth SIG
30. Alert Notification Profile Specification v1.0, Bluetooth SIG
31. Location and Navigation Service Specification v1.0, Bluetooth SIG
32. Location and Navigation Profile Specification v1.0, Bluetooth SIG
33. Phone Alert Status Service Specification v1.0, Bluetooth SIG
34. Phone Alert Status Profile Specification v1.0, Bluetooth SIG
35. Bluetooth SIG Assigned Numbers https://www.bluetooth.com/specifications/assigned-numbers
36. Services & Characteristics UUID http://developer.bluetooth.org/gatt/Pages/default.aspx
37. Personal Health Devices Transcoding White Paper v1.2, Bluetooth SIG

Appendix B Terminology

Term	Description
Service	A service is provided from a GATT server to a GATT client. The GATT server exposes some characteristics as the interface. The service prescribes how to access the exposed characteristics.
Profile	A profile enables implementation of a use case by using one or more services. The services used are defined in the specifications of each profile.
Characteristic	A characteristic is a value used to identify services. The characteristics to be exposed and their formats are defined by each service.
Role	Each device takes the role prescribed by the profile or service in order to implement the specified use case.
Client Characteristic Configuration Descriptor	A descriptor is used to control notifications or indications of characteristic values that include the client characteristic configuration descriptor sent from the GATT server.
Connection Handle	This is the handle determined by the controller stack and is used to identify connection with a remote device. The valid handle range is between 0x0000 and 0x0EFF.
Universally Unique Identifier (UUID)	This is an identifier for uniquely identifying an item. In the BLE standard, a 16-bit UUID is defined for identifying services and their characteristics.
Bluetooth Device Address (BD Address)	This is a 48-bit address for identifying a Bluetooth device. The BLE standard defines both public and random addresses, and at least one or the other must be supported.
Public Address	This is an address that includes an allocated 24-bit OUI (Organizationally Unique Identifier) registered with the IEEE.
Random Address	This is an address that contains a random number and belongs to one of the following three categories : • Static Address • Non-Resolvable Private Address • Resolvable Private Address
Static Address	This is an address whose 2 most significant bits are both 1, and whose remaining 46 bits form a random number other than all 1’s or all 0’s. This static address cannot be changed until the power is switched off.
Non-Resolvable Private Address	This is an address whose 2 most significant bits are both 0, and whose remaining 46 bits form a random number other than all 1’s or all 0’s. Static addresses and public addresses must not be equal. This type of address is used to make tracking by an attacker difficult by changing the address frequently.
Resolvable Private Address	This is an address generated from an IRK and a 24-bit random number. Its 2 most significant bits are 0 and 1, and the remaining higher 22 bits form a random number other than all 1’s or all 0’s. The lower 24 bits are calculated based on an IRK and the higher random number. This type of address is used to make tracking by an attacker difficult by changing the address frequently. By allocating an IRK to the peer device, the peer device can identify the communicating device by using that IRK.
Broadcaster	This is one of the roles of GAP. It is used to transmit advertising data.

Term	Description
Observer	This is one of the roles of GAP. It is used to receive advertising data.
Central	This is one of the roles of GAP. It is used to establish a physical link. In the link layer, it is called Master.
Peripheral	This is one of the roles of GAP. It is used to accept the establishment of a physical link. In the link layer, it is called Slave.
Advertising	Advertising is used to transmit data on a specific channel for the purpose of establishing a connection or performing data transmission.
Scan	Scans are used to receive advertising data. There are two types of scans : Passive scan, in which data is simply received, and active scan, in which additional information is requested by sending SCAN_REQ.
White List	By registering known devices that are connected or bonded to a White List, it is possible to filter devices that can accept advertising data or connection requests.
Device Name	This is a user-friendly name freely assigned to a Bluetooth device to identify it. In the BLE standard, the device name is exposed to the peer device by the GATT server as a GAP characteristic.
Reconnection Address	If a non-resolvable private address is used and the address is changed frequently, not only attackers but also the peer device will have difficulty identifying the device. Therefore, the address to be used at reconnection is reported by setting a new reconnection address as the exposed reconnection address characteristic.
Scan Interval	This is the interval for receiving advertising data.
Scan Window	This is the period of time during which advertising data is received at the scan interval.
Connection Interval	This is the interval for transmitting and receiving data periodically following connection establishment.
Connection Event	This is the period of time during which data is transmitted and received at the connection interval.
Slave Latency	This is the period of time during which data is transmitted and received at the connection interval.
Supervision Timeout	This is the timeout interval after which the link is considered to have been lost when no response is received from the peer device.
Passkey Entry	This is a pairing method whereby a six-digit number is input by each device to the other, or a six-digit number is displayed by one of the devices and that number is input to the other device.
Just Works	This is a pairing method that does not require user action.
OOB	This is a pairing method whereby pairing is performed by using data obtained by a communication method other than Bluetooth.
Identity Resolving Key (IRK)	This is a 128-bit key used to generate and resolve resolvable private addresses.
Connection Signature Resolving Key (CSRK)	This is a 128-bit key used to create data signatures and verify the signature of incoming data.
Long Term Key (LTK)	This is a 128-bit key used for encryption. The key size to be used is the size agreed on during pairing.
Short Term Key (STK)	This is a 128-bit key used for encryption during key exchange. It is generated using TK.

Term	Description
Temporary Key (TK)	This is a 128-bit key required for STK generation. In the case of Just Works, the TK value is 0. In the case of Passkey Entry, it is the 6-digit number that was input, and in the case of OOB, it is the OOB data.

Bluetooth Low Energy Protocol Stack
User's Manual

Publication Date : Rev.1.19 Jan 31, 2022

Published by : Renesas Electronics Corporation

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

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

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

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

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

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

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

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

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

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

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 5.1

Bluetooth Low Energy Protocol Stack

Renesas Electronics Corporation

R01UW0095EJ0119