Dear customer

ROHM Co., Ltd. ("ROHM"), on the 1st day of April, 2024,
has absorbed into merger with 100%-owned subsidiary of LAPIS Technology Co., Ltd.

Therefore, all references to "LAPIS Technology Co., Ltd.", "LAPIS Technology"
and/or "LAPIS" in this document shall be replaced with "ROHM Co., Ltd."
Furthermore, there are no changes to the documents relating to our products other than
the company name, the company trademark, logo, etc.

Thank you for your understanding.

ROHM Co., Ltd.
April 1, 2024

MK715x1
AT Command Application
User's Manual

Issue date: Nov. 25, 2020

Notes

1) The information contained herein is subject to change without notice.

2) When using LAPIS Technology Products, refer to the latest product information (data sheets, user's manuals, application notes, etc.), and ensure that usage conditions (absolute maximum ratings, recommended operating conditions, etc.) are within the ranges specified. LAPIS Technology disclaims any and all liability for any malfunctions, failure or accident arising out of or in connection with the use of LAPIS Technology Products outside of such usage conditions specified ranges, or without observing precautions. Even if it is used within such usage conditions specified ranges, semiconductors can break down and malfunction due to various factors. Therefore, in order to prevent personal injury, fire or the other damage from break down or malfunction of LAPIS Technology Products, please take safety at your own risk measures such as complying with the derating characteristics, implementing redundant and fire prevention designs, and utilizing backups and fail-safe procedures. You are responsible for evaluating the safety of the final products or systems manufactured by you.

3) Descriptions of circuits, software and other related information in this document are provided only to illustrate the standard 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. And the peripheral conditions must be taken into account when designing circuits for mass production. LAPIS Technology disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, and other related information.

4) No license, expressly or implied, is granted hereby under any intellectual property rights or other rights of LAPIS Technology or any third party with respect to LAPIS Technology Products or the information contained in this document (including but not limited to, the Product data, drawings, charts, programs, algorithms, and application examples、etc.). Therefore LAPIS Technology shall have no responsibility whatsoever for any dispute, concerning such rights owned by third parties, arising out of the use of such technical information.

5) The Products are intended for use in general electronic equipment (AV/OA devices, communication, consumer systems, gaming/entertainment sets, etc.) as well as the applications indicated in this document. For use of our Products in applications requiring a high degree of reliability (as exemplified below), please be sure to contact a LAPIS Technology representative and must obtain written agreement: transportation equipment (cars, ships, trains, etc.), primary communication equipment, traffic lights, fire/crime prevention, safety equipment, medical systems, servers, solar cells, and power transmission systems, etc. LAPIS Technology disclaims any and all liability for any losses and damages incurred by you or third parties arising by using the Product for purposes not intended by us. Do not use our Products in applications requiring extremely high reliability, such as aerospace equipment, nuclear power control systems, and submarine repeaters, etc.

6) The Products specified in this document are not designed to be radiation tolerant.

7) LAPIS Technology has used reasonable care to ensure the accuracy of the information contained in this document. However, LAPIS Technology does not warrant that such information is error-free and LAPIS Technology shall have no responsibility for any damages arising from any inaccuracy or misprint of such information.

8) Please use the Products in accordance with any applicable environmental laws and regulations, such as the RoHS Directive. LAPIS Technology shall have no responsibility for any damages or losses resulting non-compliance with any applicable laws or regulations.

9) When providing our Products and technologies contained in this document to other countries, you must abide by the procedures and provisions stipulated in all applicable export laws and regulations, including without limitation the US Export Administration Regulations and the Foreign Exchange and Foreign Trade Act..

10) Please contact a ROHM sales office if you have any questions regarding the information contained in this document or LAPIS Technology's Products.

11) This document, in part or in whole, may not be reprinted or reproduced without prior consent of LAPIS Technology.

(Note) "LAPIS Technology" as used in this document means LAPIS Technology Co., Ltd.

LAPIS Technology Co.,Ltd.
2-4-8 Shinyokohama, Kouhoku-ku, Yokohama 222-8575, Japan
https://www.lapis-tech.com/en/

Introduction

This document is the user's manual of the AT command application designed for LAPIS **Bluetooth®** low energy modules: MK71511 and MK71521 that support Bluetooth 5.
As well as this document, read the following provided documents as needed.

- MK71511 Data Sheet
- MK71521 Data Sheet
- MK715x1EK1 Hardware Manual
- MK715x1EK1A/MK715x1EK1AP Hardware Manual
- MK715x1 Software Development Startup Guide
- BLE Tool User's Manual

Note: In this document, MK715x1 is used to indicate both MK71511 and MK71521.

Notation

Category	Notation	Description
● Value	0x*nn*	Represents a hexadecimal number.
	0b*nnnn*	Represents a binary number.
● Address	0x*nnnn_nnnn*	Represents a hexadecimal number. (indicates 0xnnnnnnnn)
● Unit	Word, WORD	1 word = 32 bits
	Byte, BYTE	1 byte = 8 bits
	Mega, M	10^6
	Kilo, K	$2^{10} = 1024$
	Kilo, k	$10^3 = 1000$
	Milli, m	10^{-3}
	Micro,μ	10^{-6}
	Nano, n	10^{-9}
	Second, s (lowercase)	Second
● Term	"H" level	Indicates high voltage signal levels V_{IH} and V_{OH} as specified by the electrical characteristics.
	"L" level	Indicates low voltage signal levels V_{IL} and V_{OL} as specified by the electrical characteristics.

● Register Description
Read/write attribute: R indicates read-enabled; W indicates write-enabled.
MSB: Most significant bit in an 8-bit register (memory)
LSB: Least significant bit in an 8-bit register (memory)

Table of Contents

1. Overview

The AT command application can realize Bluetooth low energy communication easily using the simple command interface called AT command via UART. After a Bluetooth low energy connection is established, the data transmitted to UART can be received by a counter device. Therefore, the communication path can be made wireless easily with Bluetooth low energy.
This document describes how to use the MK715x1 AT command application using the MK715x1 evaluation kit and also the procedure of the function. For details about the MK715x1 evaluation kit, refer to "MK715x1EK1 Hardware Manual" or "MK715x1EK1A/MK715x1EK1AP Hardware Manual".

1.1. System Configuration

The following shows the configuration diagram of the AT command application. The AT command application is an application that operates on MK715x1 and is controlled from the host system using AT commands. The host system can control MK715x1 using AT commands and result codes via the UART interface.
The following figure shows the configuration that MK715x1 is connected to a central device such as smartphone as a peripheral device.

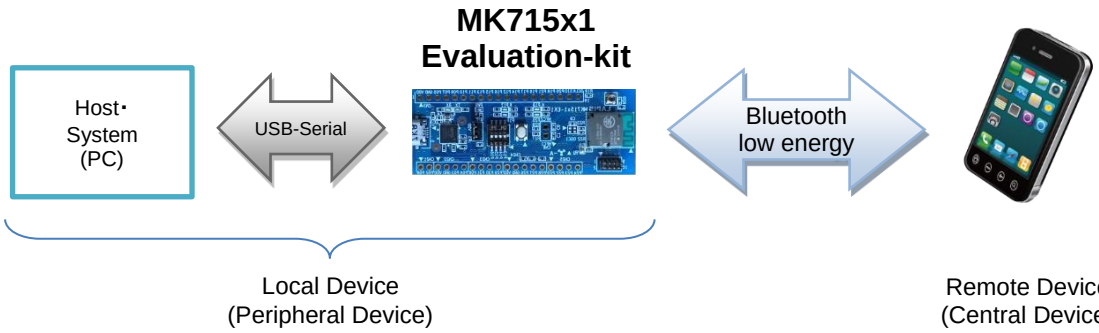


Figure 1-1 System Configuration (Connecting to Smartphone, etc.)

Since MK715x1 is equipped with the central function, data communication between MK715x1 modules is possible as shown in the following figure.

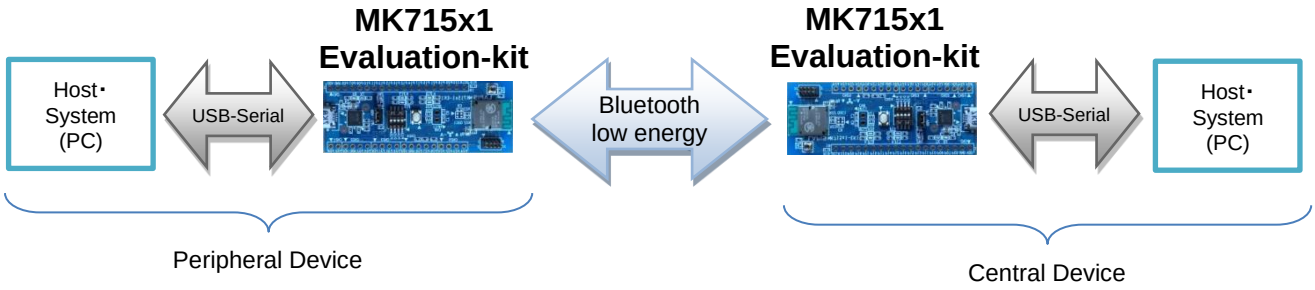


Figure 1-2 System Configuration (Connecting between MK715x1 Modules)

1.2. Function Overview

The AT command application provides the following functions.

- Connection control on the peripheral side (advertisement/pairing/connection/disconnection)
- Connection control on the central side (scanning/pairing/connection/disconnection)
- Pairing (Just Works/Passkey Entry) supported: Pairing with up to five terminals is possible.
- Terminal unique information setting (BD address, etc.)
- Parameter setting related to Bluetooth low energy communication
- Operation with low current consumption by dynamic change of connection parameter
- Mounted profile/service:
  LAPIS original VSSPP (Vendor Specific Serial Port Profile)
  Bluetooth SIG standard BAS (Battery Service)
  Bluetooth SIG standard DIS (Device Information Service)

[Note]
The central side of the AT command application provides the functions to perform data communication between MK715x1 modules mainly.

2. Let's Start Using

This chapter explains how to use the MK715x1 AT command application simply. Prepare the MK715x1 evaluation kit and the smartphone application "BLE Tool" at hand. BLE Tool can be downloaded for free from Google Play or App Store.

Google Play	https://play.google.com/store/apps/details?id=com.lapis_semi.bleapp
App Store	https://itunes.apple.com/jp/app/ble-tool/id915714158?mt=8&ign-mpt=uo%3D4

2.1. Preparation on MK715x1 Side

2.1.1. Installing Driver for USB Serial Conversion IC

To start the MK715x1 side, connect the MK715x1 evaluation kit to the USB port of a PC. When connecting for the first time, the driver for USB serial conversion IC needs to be installed. Download the latest version of the USB serial conversion IC driver from the following site as needed.

https://www.ftdichip.com/Drivers/VCP.htm

2.1.2. Updating MK715x1 Software

Update the MK715x1 software according to the following procedure.

2.1.2.1. Installing PC Tool

For the PC tool installation procedure, refer to "2.6 Installing nRF Connect for Desktop" in "MK715x1 Software Development Startup Guide".

2.1.2.2. Downloading Application Program

Download the "MK715x1 Software Development Kit" file (ZIP file) from the Bluetooth low energy related section in the following LAPIS support site and unzip the file to obtain the following Hex file (*.hex) of the AT command application program.

LAPIS support site:	https://www.lapis-semi.com/cgi-bin/MyLAPIS/regi/login.cgi	(English)
	https://www.lapis-semi.com/cgi-bin/MyLAPIS/regi/login_J.cgi	(Japanese)

ZIP file of MK715x1 software development kit: mk715x1_sdk_verXXX.zip * XXX indicates a version number.

Hex file storage folder: ".¥mk715x1_sdk_verXXX¥software¥examples¥lapis¥mk715x1_at_cmd¥hex"

AT command application code:	mk71511ek1_at_cmd_rXXX.hex	(MK71511EK1)
	mk71511ek1a_at_cmd_rXXX.hex	(MK71511EK1A/ MK71511EK1AP)
	mk71521ek1_at_cmd_rXXX.hex	(MK71521EK1)
	mk71521ek1a_at_cmd_rXXX.hex	(MK71521EK1A/MK71521EK1AP)

* XXX indicates a revision number of the code.

2.1.2.3. Writing Application Program

For how to write an application program, refer to " 3.4. Writing Built Program " in "MK715x1 Software Development Startup Guide".

2.1.2.4. Executing Application

Connect the MK715x1 evaluation kit to the USB (power feed included) of a PC. For the MK715x1-EKA evaluation kit, turn the power switch (POWER_SW) ON. Then, set all the DIP SW on the MK715x1 evaluation kit to OFF and press the button to execute the AT command application.

Start the terminal software such as TeraTerm and set the serial port as follows:

Port:	COM port number used
Baud rate:	57600 bps
Data:	8 bit
Parity:	None
Stop: 1 bit	
Flow control:	Hardware

Input "at<CR>", which is the AT command for command reception confirmation, from the terminal. When the result code string is output as shown below, it indicates that UART communication is performed normally between the PC and MK715x1 evaluation kit. The input of the "at" command is not output because echoing back from MK715x1 is disabled.

Figure 2-1 Screen Outputting Result Code String for Command Reception Confirmation

After that, input "atd <CR>" to start the peripheral operation. The MK715x1 evaluation kit will start transmission of advertisement. To start the central operation, input "ata <CR>". The MK715x1 evaluation kit will start scanning to search for peripheral devices. The preparation on the MK715x1 side is now completed. For the overview of AT commands, refer to "4.5 AT Command Specifications".

2.2. Preparation on Central Side (Smartphone)

2.2.1. Starting Application

Tap the following "BLE Tool" icon on the smartphone to start the application.
The left figure shows the icon for Android terminals, and the right figure shows the icon for iOS terminals.

Figure 2-2 BLETool Icons (Left: Android, Right: iOS)

2.2.2. Data Communication

Bluetooth low energy communication can be performed according to the following steps. For details about the usage of BLE Tool, refer to the related document "BLE Tool User's Manual".

A) When BLE Tool is started, the following screen (A) is displayed. This screen displays Bluetooth low energy devices from the advertisement packet detected through scanning by the central side. The MK715x1 AT command application is displayed as the device name "LapisDev" by default. Tap this device.

B) The Bluetooth low energy connection procedure is executed, and the following service search screen (B) is displayed. Also, at this time, "CONNECT" is output to the terminal screen on the peripheral side. The following screen (B) displays the services detected with a service search by the central side. For the MK715x1 AT command application, the two services: "Device Information" and "LAPIS Serial Port Profile" are displayed. The latter is the service used by the AT command application for data communication. Tap the [VSSPP] icon.

C) The VSSPP service becomes enabled, and the following screen (C) is displayed. Data transmission/reception is performed on this screen. Tapping the text box at the bottom of the screen displays a software keyboard. When a character string is input using the software keyboard and the [Send] button is tapped, the input character string will be transmitted to the peripheral side. Likewise, characters input from the terminal screen on the peripheral side are transmitted to the central side.

(A) (B) (C)

Figure 2-3 Examples of BLETool Operation Screen

The following figure shows an example of data communication performed according to the above steps. The character string input from BLE Tool is output in black as shown by (a) in the following figure, and the same character string is output to the terminal screen on the peripheral side ((b) in the following figure). The character string input from the terminal on the peripheral side ((c) in the following figure) is output to the BLE Tool screen in red ((d) in the following figure).

Central
（Smartphone screen）

Peripheral
（MK715x1 terminal software screen）

Figure 2-4 Examples of Data Communication Screen

2.2.3. Reading Device Information

The MK715x1 AT command application also provides the Bluetooth SIG standard DIS (Device Information Service). As shown in the following figure, the device information held by the peripheral can be read by tapping the [DIS] icon on the service search screen. The following figure shows the default settings of the MK715x1 AT command application. The device information needs to be changed according to the system used. For correction of the device information, refer to "4.2.5.2 Device Information Service".

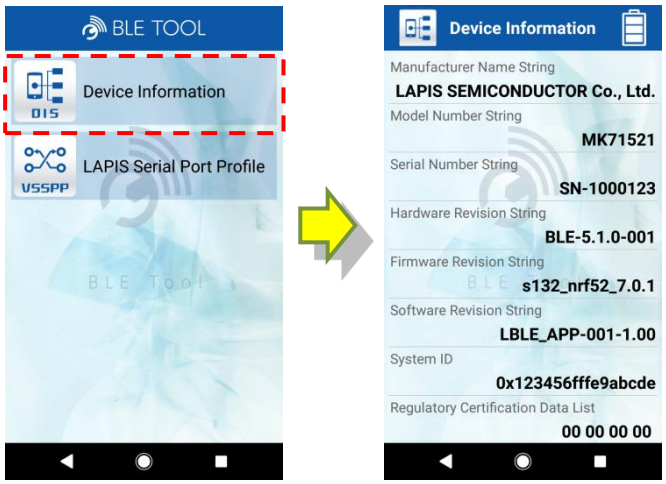


Figure 2-5 Examples of Device Information Screen

This is the end of the explanation about the usage of the MK715x1 AT command application.
To develop a smartphone application on the central side, refer to "4.2 Bluetooth Low Energy Communication".
To control MK715x1 through connection with the host MCU instead of controlling it from a PC, refer to the next chapter "3 Incorporating into Customer System".

3. Incorporating into Customer System

This chapter describes the case where the system to control MK715x1 from the host MCU is configured using the MK715x1 AT command application.

3.1. Connecting with Host MCU

The following shows an example of connection configuration between the host MCU and MK715x1. The host MCU executes transmission/reception of AT commands via UART and also transmission/reception of data in Bluetooth low energy communication. Also, the host MCU controls MK715x1 using GPIO as well as UART.

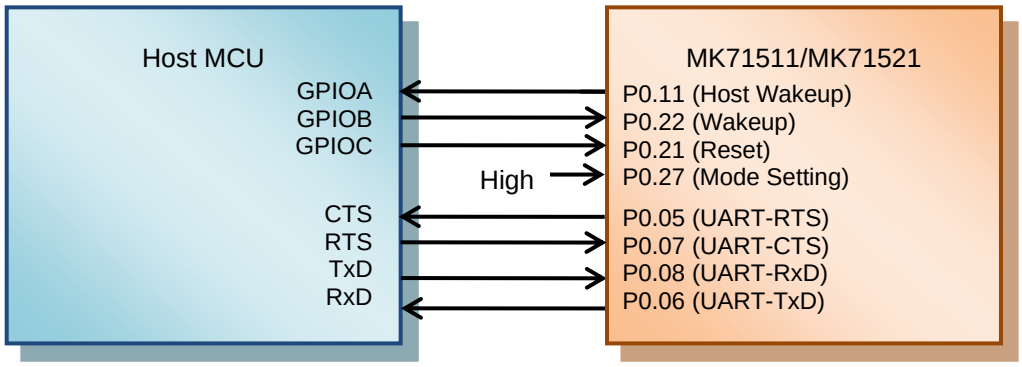


Figure 3-1 Example of Connection Configuration with Host MCU

The following table shows the functions of the connection interface between MK715x1 and host MCU.

Table 3-1 External Interface Functions of MK715x1

MK715x1 pin	Function description
P0.11 (Host Wakeup)	An output pin used to request the host MCU to recover from the low current consumption state when data is output from UART. Using a system parameter, this pin can be enabled/disabled and also the delay time before data is output from UART can be set.
P0.22 (Wakeup)	An input pin used to request MK715x1 to enter the low current consumption state. With High set on the host MCU, MK715x1 is requested to enter the low current consumption state. Set to Low when inputting an AT command.
P0.21 (Reset)	An input pin used to reset MK715x1. Set to Low to reset, or set to High to release reset.
P0.27 （Mode Setting)	A pin used to set the MK71521 operation mode through the input at the time of startup. This is not supported by MK71511. Set to High when starting the AT command application.
P0.05 (UART-RTS)	An output pin used to control the UART reception flow. It notifies of Low when MK715x1 is in the UART receivable state.
P0.07 (UART-CTS)	An input pin used to control the UART transmission flow. Set to Low when the host MCU is in the UART receivable state.
P0.08 （UART-RxD）	An input pin used to receive UART data.
P0.06 (UART-TxD)	An output pin used to transmit UART data.

3.2. GPIO Control

3.2.1. From Startup to AT Command Input

The following chart shows the relation between the GPIO pins and AT command after the startup of MK715x1.

(1) Set Reset (P0.21) to High and release the reset of MK715x1.
(2) Set Wakeup (P0.22) to Low to input an AT command.
(3) Set UART-RTS (P0.05) to Low output when the input of an AT command is available.
(4) Input an AT command to UART.
(5) After a result code is output, set Wakeup (P0.22) to High.
(6) Set UART-RTS (P0.05) to High output. This causes MK715x1 to enter the low current consumption state.

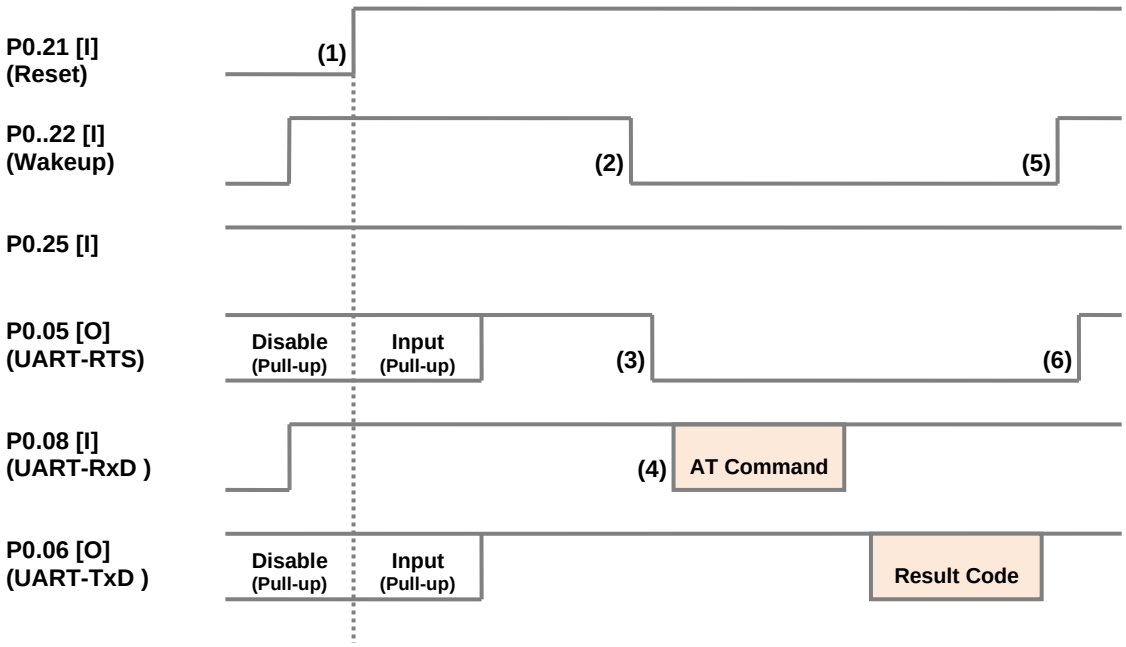


Figure 3-2 Example of GPIO Control Sequence (from Startup to AT Command Input)

3.2.2. After Establishing Connection

The following chart shows the relation between the GPIO pins and AT command/transmission and reception data via radio communication after connecting MK715x1.

(1) Set Wakeup (P0.22) to Low to input the transmission data via radio communication.
(2) Set UART-RTS (P0.05) to Low output when the input of the transmission data via radio communication is available.
(3) Input the transmission data via radio communication to UART.
(4) Set Wakeup (P0.22) to High.
(5) UART-RTS (P0.05) becomes High output. This causes MK715x1 to enter the low current consumption state.
(6) Because UART-CTS (P0.07) is Low, the reception data via radio communication is output to UART.
(7) Set UART-CTS (P0.07) to High when UART reception is unavailable on the host MCU side.
The output to UART is suspended.
(8) Set UART-CTS (P0.07) to Low when UART reception is available on the host MCU side.
The output to UART is resumed.
(9) Set Wakeup (P0.22) to Low to input an AT command.
(10) Set UART-RTS (P0.05) to Low output when the input of an AT command is available.
(11) Input an AT command to UART.
(12) After a result code is output, set Wakeup (P0.22) to High.
(13) Set UART-RTS (P0.05) to High output. This causes MK715x1 to enter the low current consumption state.

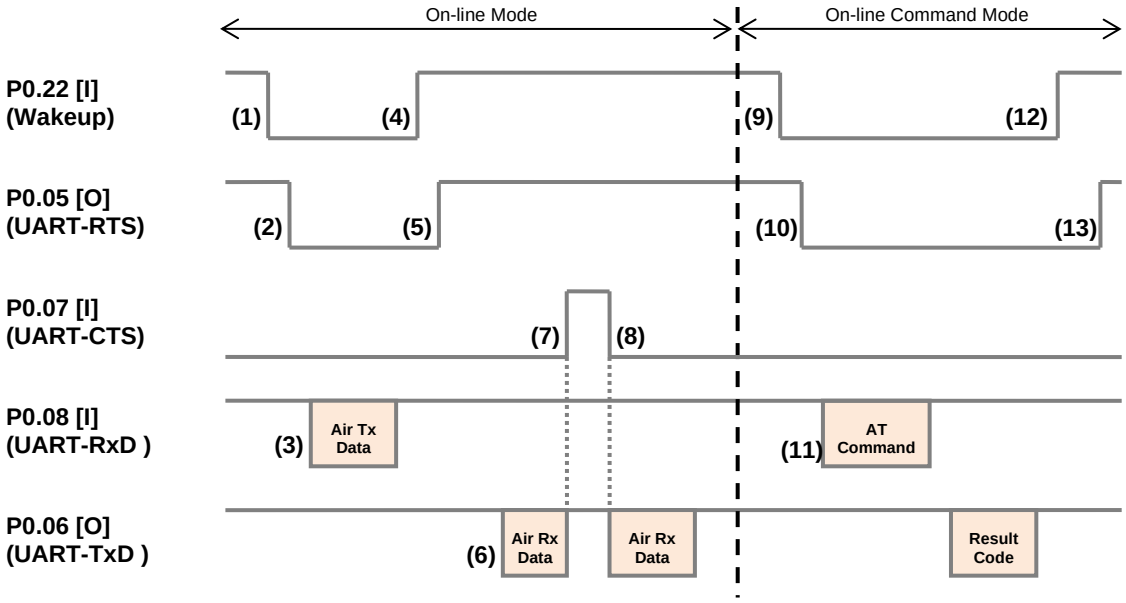


Figure 3-3 Example of GPIO Control Sequence (after Establishing Connection)

3.2.3. Host Wakeup

Host Wakeup is an output signal used to request the host MCU to recover from the low current consumption state when data is output from UART. This function is assigned to the Host Wakeup pin (P0.11). The function can be disabled (Low Active)/enabled (High Active) using the setting of a system parameter. When Host Wakeup is enabled, the delay time before data is output from UART after the Host Wakeup state is changed to Active can be adjusted.
The following chart shows the relation between Host Wakeup of MK715x1 and AT command/reception data via radio communication.

(1) Set Host Wakeup (P0.11) to Low output to output a result code.
(2) Output a result code.
    * While Wakeup (P0.22) is set to Low input, the delay time is disabled.
(3) Set Host Wakeup (P0.11) to High output.
(4) Set Host Wakeup (P0.11) to Low output to output a result code or reception data via radio communication.
(5) Output a result code or reception data via radio communication after the delay time.
(6) Set Host Wakeup (P0.11) to High output.

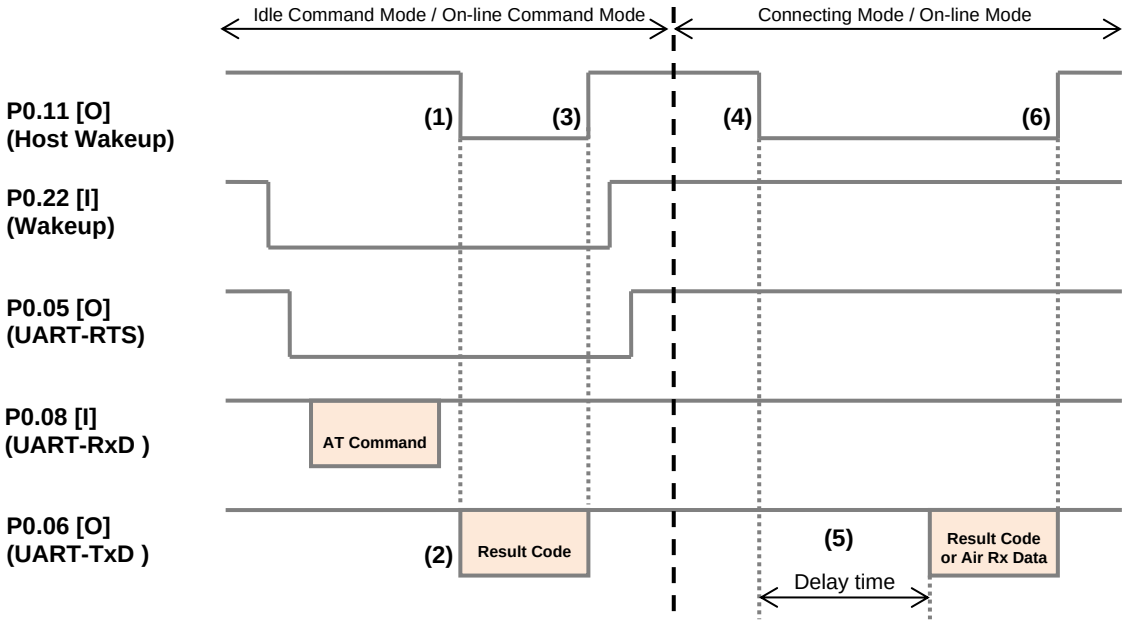


Figure 3-4 Example of GPIO Control Sequence (Host Wakeup [Low Active])

(1) Set Host Wakeup (P0.11) to High output to output a result code.
(2) Output a result code.
* While Wakeup (P0.22) is set to Low input, the delay time is disabled. Set Host Wakeup (P0.11) to Low output.
(3) Set Host Wakeup (P0.11) to High output to output a result code or reception data via radio communication.
(4) Output a result code or reception data via radio communication after the delay time.
(5) Set Host Wakeup (P0.11) to Low output.

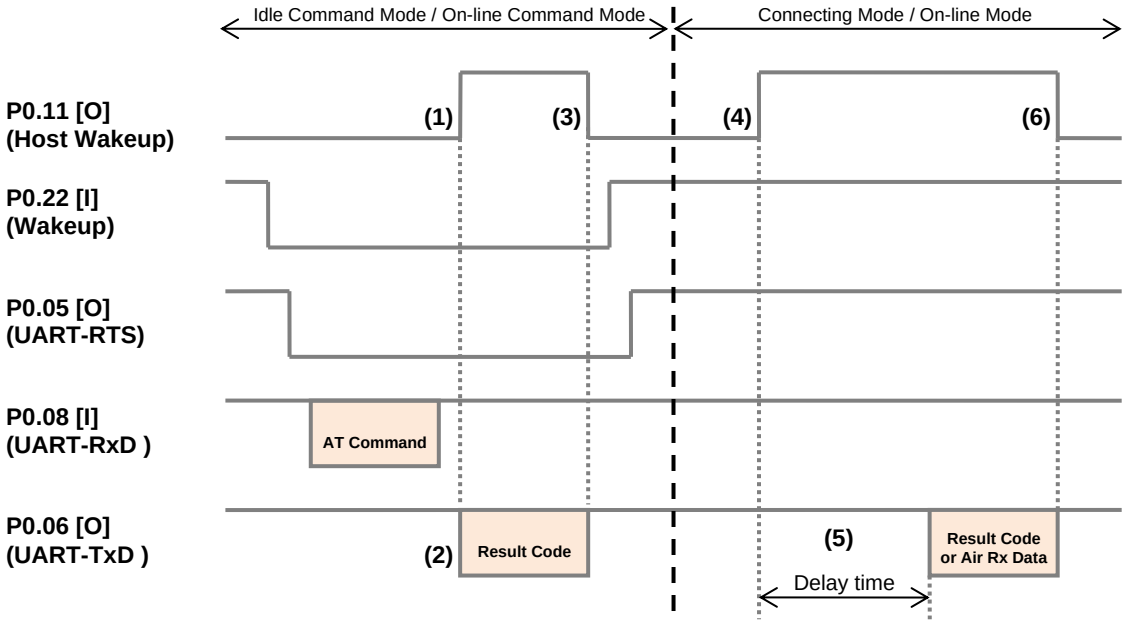


Figure 3-5 Example of GPIO Control Sequence (Host Wakeup [High Active])

4. AT Command Application Specifications

This chapter describes the detailed specifications related to the MK715x1 AT command application. This chapter mainly describes the following items.

- Operation Mode
- Bluetooth Low Energy Communication
- UART Flow Control
- Low Current Consumption Control
- AT Command Specifications

4.1. Operation Mode

4.1.1. MK71521 Operation Mode [TBD]

MK71521 has two operation modes. One is called "User Application Mode" in which the AT command application can operate, and the other is called "DFU Mode" which is used to update the firmware. Switching between the user application mode and DFU mode can be made by the following operation. This is not supported by MK71511.

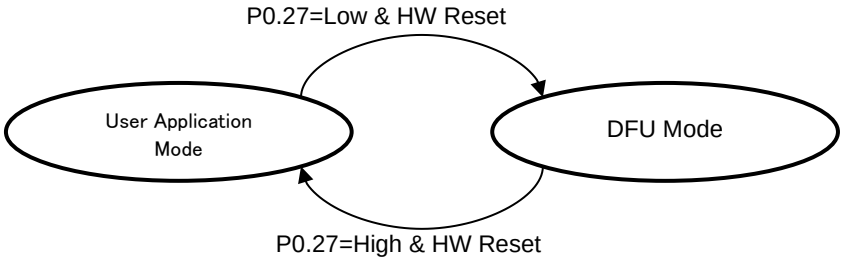


Figure 4-1 Switching between User Application Mode and DFU Mode

4.1.2. Application Operation Mode

The AT command application operates in some operation modes. The following shows the state transition diagram. Result codes from MK715x1 are shown in red and bold in the diagram.

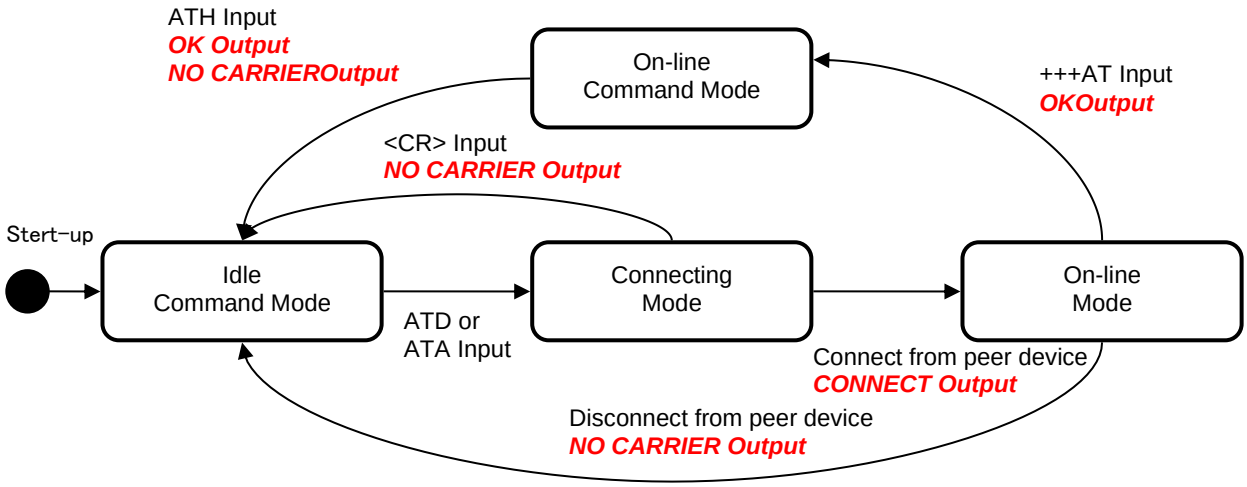


Figure 4-2 Application Operation Mode Transition Diagram

(1) Idle Command Mode

After the MK715x1 power is turned on, the AT command application starts operation in the idle command mode. In this mode, the connection related to Bluetooth low energy communication can be controlled with AT commands.

(2) Connecting Mode

・ Peripheral device

When the ATD command is input in the idle command mode, the connecting mode is entered to start transmission of advertisement. When <CR> (carriage return) is input during transmission of advertisement, the advertising operation will be stopped, and the idle command mode will be entered again. When connection or pairing is successful, the CONNECT result code is output and the on-line mode is entered.

・ Central device

When the ATA command is input in the idle command mode, the connecting mode is entered to start scanning. When <CR> (carriage return) is input during scanning, the scanning operation will be stopped, and the idle command mode will be entered again. When connection or pairing is successful, the CONNECT result code is output and the on-line mode is entered.

(3) On-line Mode

All the data input from UART is transmitted to the counter terminal being connected, and all the data received from the counter terminal is output from UART. The on-line command mode is entered by the input of the escape code "+++AT<CR>".

(4) On-line Command Mode

When the ATH command is input, the disconnection process is executed, and the idle command mode is entered. The data received from the counter terminal in this mode is not output from UART.

4.2. Bluetooth Low Energy Communication

4.2.1. Service and Characteristic

The AT command application employs the following service configuration. The MK715x1 AT command application performs data communication using LAPIS original VSA (Vendor Specific Access) characteristic held by LAPIS original VSSPP (Vendor Specific Serial Port Profile) on the peripheral device side.
The VSA characteristic properties: "Notify", "Indicate", "Write without Response" and "Write" can be set using the system parameter "VSSPP: VSA property (LS_VSSPP_PROPERTY)".

Service: LAPIS original service (VSSPP)

Type UUID = 0x2800 (GATT Primary Service)
Service UUID = 0x0179BBD0-5351-48B5-BF6D-2167639BC867
Property = Read

Characteristics: LAPIS original characteristics (VSA)

TypeUUID = 0x2803 (GATT Characteristic)
ServiceUUID = 0x0179BBD1-5351-48B5-BF6D-2167639BC867
Property = Notify / Indicate / Write without Response / Write

Figure 4-3 Service Configuration

4.2.2. Communication Procedure

4.2.2.1. Peripheral Device Side

An upstream data communication from the MK715x1 peripheral device to the central device is enabled by writing Notify attribute enable or Indicate attribute enable to CCCD (Client Characteristic Configuration Descriptor) held by the LAPIS VSSPP characteristics ((1) in the following figure). To transmit data, set the Wakeup pin (P0.22) to Low input. Then, MK715x1 sets the UART-RTS pin (P0.05) to Low output ((2) in the following figure) to accept transmission data. When the host MCU inputs data to MK715x1 via UART, the data is transmitted to the central device in radio communication by Notification or Indication according to the VSA characteristic property ((3) in the following figure).

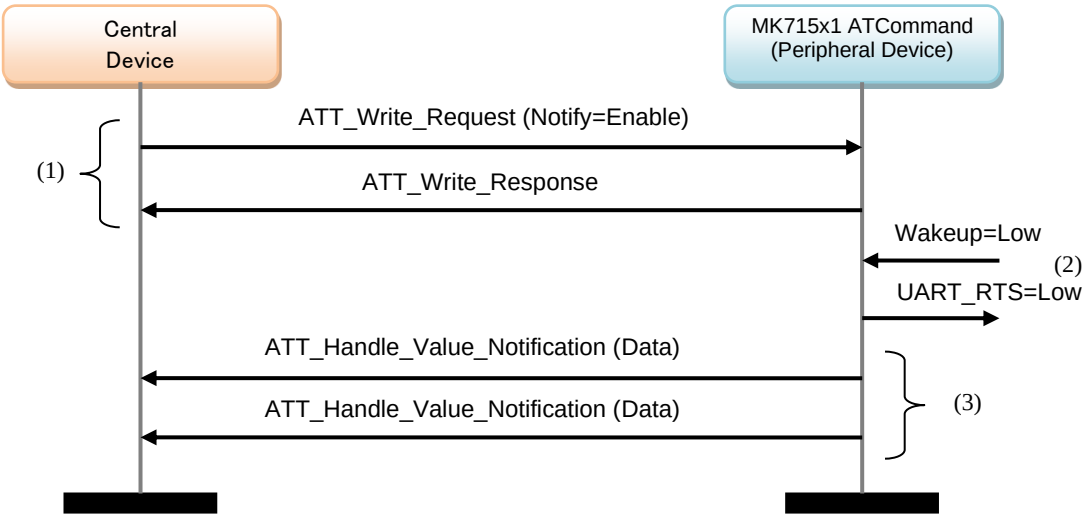


Figure 4-4 Peripheral Upstream Data Communication (Notify)

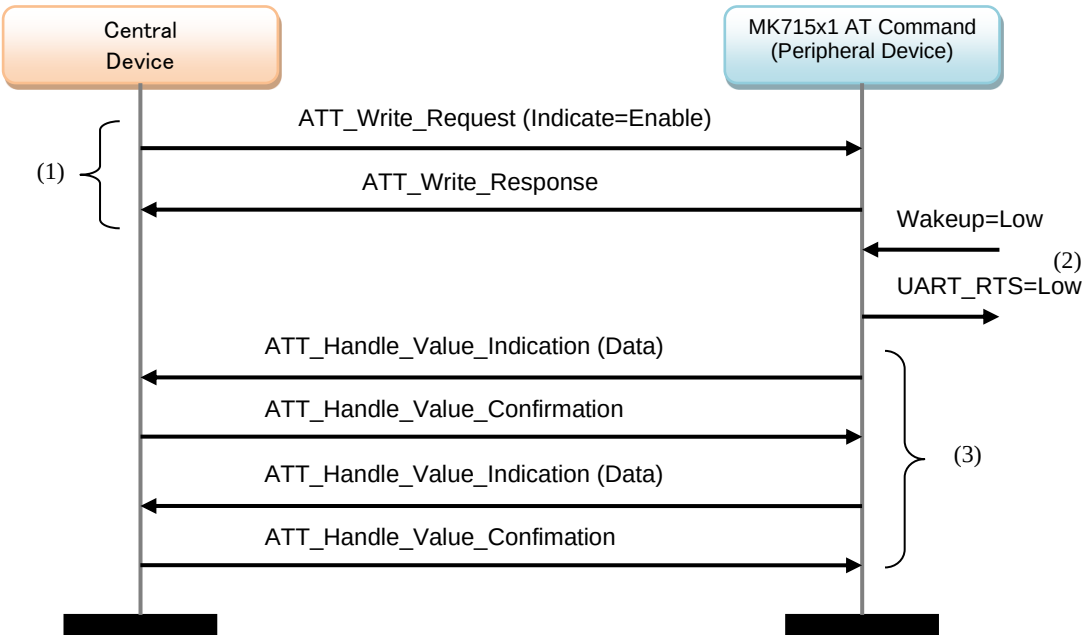


Figure 4-5 Peripheral Upstream Data Communication (Indicate)

A downstream data transmission from the central device to the MK715x1 peripheral device is performed via radio communication according to the VSA characteristic property on receipt of Write command or Write request ((4) in the following figure).

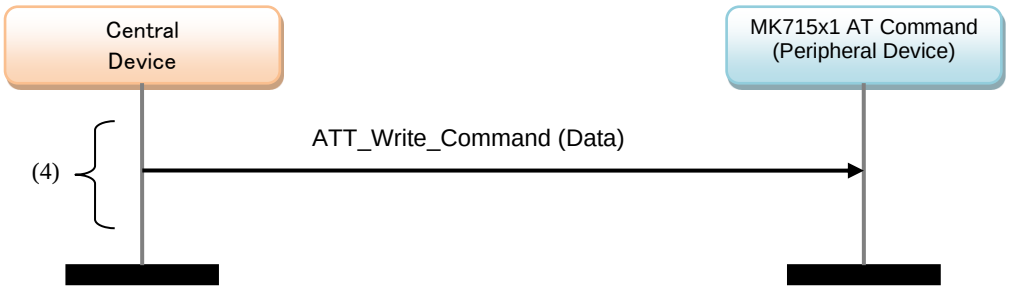


Figure 4-6 Peripheral Downstream Data Communication (Write without Response)

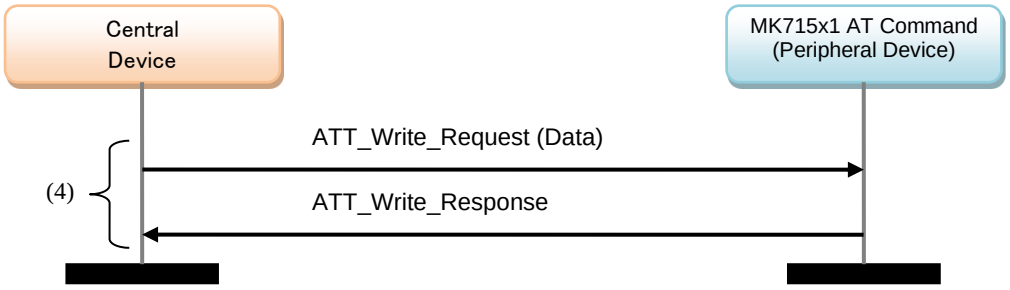


Figure 4-7 Peripheral Downstream Data Communication (Write)

4.2.2.2. Central Device Side

An upstream data communication from the peripheral device to the MK715x1 central device is enabled when MK715x1 writes Notify attribute enable or Indicate attribute enable to CCCD (Client Characteristic Configuration Descriptor) held by the LAPIS VSSPP characteristics automatically at the time of connection ((1) in the following figure). Data transmission from the peripheral device to the central device is performed via radio communication by Notification or Indication according to the VSA characteristic property ((2) in the following figure).

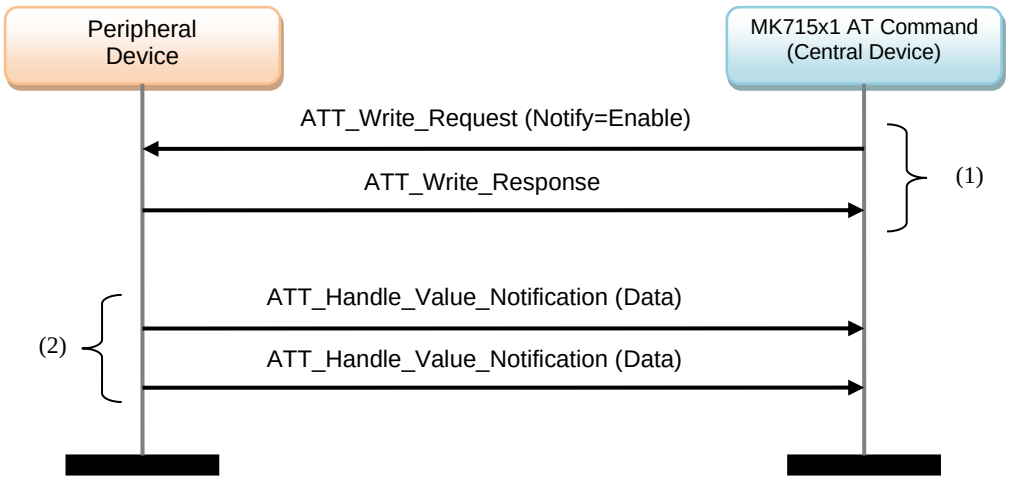


Figure 4-8 Central Upstream Data Communication (Notify)

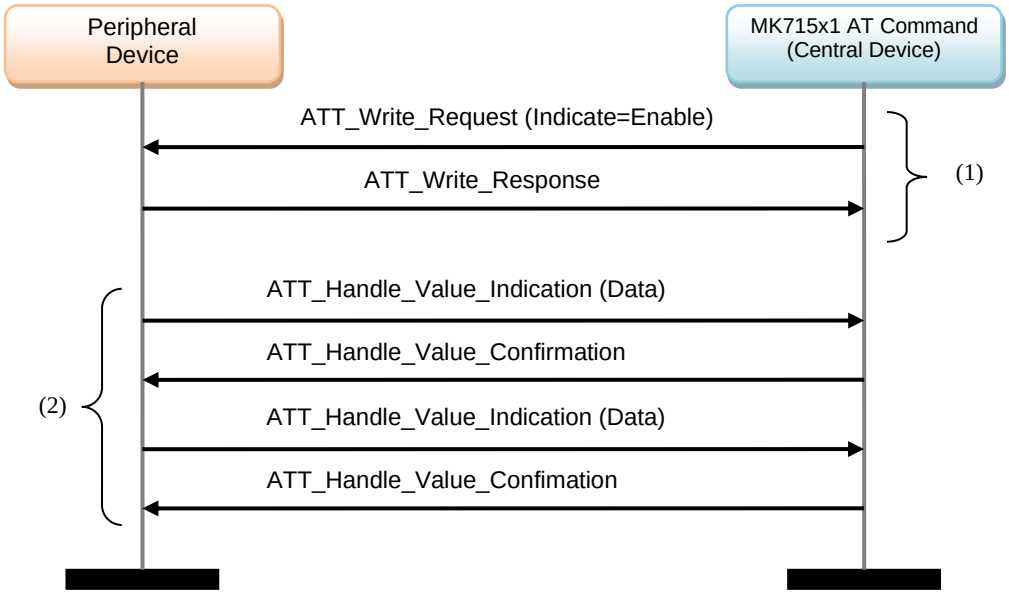


Figure 4-9 Central Upstream Data Communication (Indicate)

To perform a downstream data communication from the MK715x1 central device to the peripheral device, the Wakeup pin (P0.22) is set to Low input. Then, MK715x1 sets the UART-RTS pin (P0.05) to Low output ((3) in the following figure) to accept transmission data. When the host MCU inputs data to MK715x1 via UART, the data is transmitted to the peripheral device in radio communication by Write command or Write request according to the VSA characteristic property ((4) in the following figure).

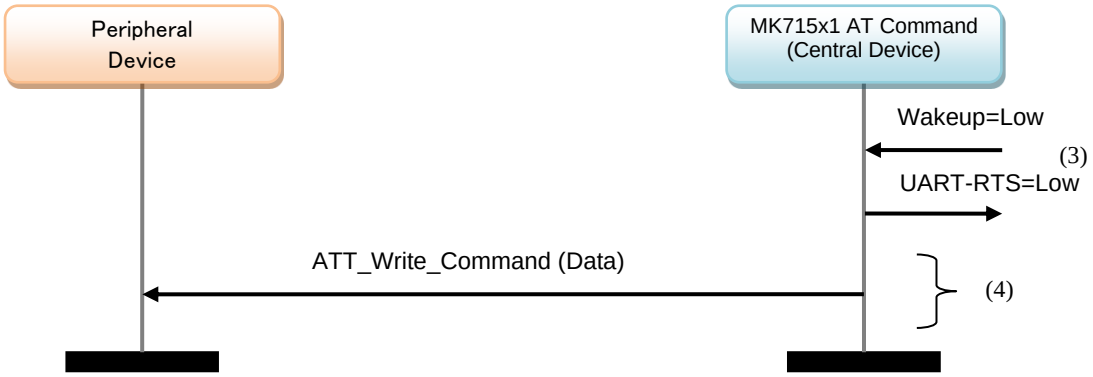


Figure 4-10 Central Downstream Data (Write without Response)

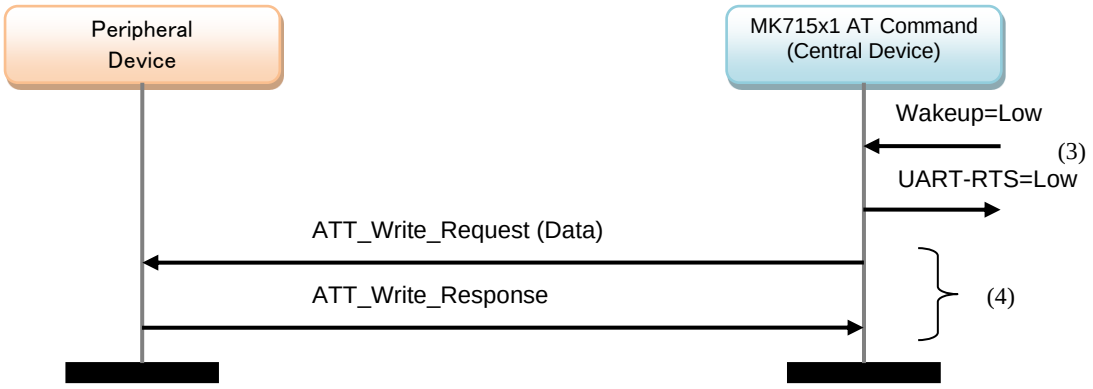


Figure 4-11 Central Downstream Data Communication (Write)

4.2.3. CCCD Control

As described previously, an upstream data communication is enabled by writing Notify attribute enable or Indicate attribute enable to the CCCD held by the LAPIS VSSPP characteristic by the central device. If a Bluetooth low energy connection is disconnected without writing Notify attribute enable or Indicate attribute enable to this CCCD by the central device, the CCCD enters the Notify attribute disable and Indicate attribute disable state regardless of the presence of pairing at the time of reconnection attempted later. Therefore, after a Bluetooth low energy connection is established for the MK715x1 AT command application, the central device must write Notify attribute enable or Indicate attribute enable to the CCCD held by the VSSPP characteristic.
Do not attempt writing to the CCCD other than 0x0000 (Notify attribute enable), 0x0001 (Notify attribute enable) and 0x0002 (Indicate attribute enable).

4.2.4. Pairing

The following pairing modes can be selected using the setting of a system parameter. LE Secure Connection pairing modes are supported only by the AT command application for MK71521.

Pairing mode = 0: No pairing (encrypted communication unavailable)
Pairing mode = 1: Legacy Just Works (encrypted communication without passkey entry)
Pairing mode = 2: Legacy Passkey Entry (encrypted communication with passkey entry)
Pairing mode = 3: LE Secure Connection Just Works (encrypted communication without passkey entry)
Pairing mode = 4: LE Secure Connection Passkey Entry (encrypted communication with passkey entry)
Pairing mode = 5: LE Secure Connection Numeric Comparison (encrypted communication with passkey comparison)

The following figure shows the pairing sequence. When the pairing mode is set to 1 to 5, the MK715x1 AT command application on the peripheral device side issues the "Security Request" packet autonomously to execute the pairing sequence.

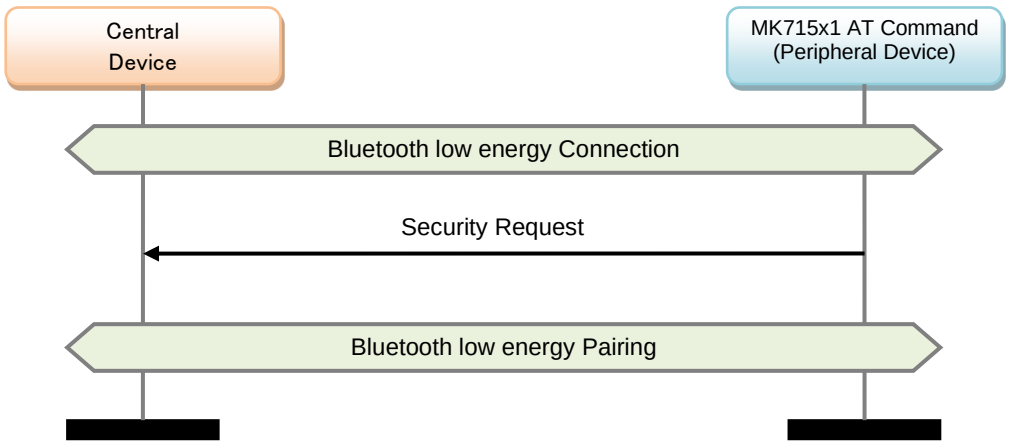


Figure 4-12 Pairing Sequence

When the pairing sequence is completed, the MK715x1 AT command application saves the counter device information in the Flash ROM. This is called bonding information. When the pairing sequence ends, encrypted communication will be started. When a reconnection with the device having the bonding information is attempted, the pairing sequence will be omitted, and encrypted communication will be started. The bonding information can be saved for up to five units. When pairing with the sixth or subsequent unit is performed, the oldest bonding information will be overwritten. It is also possible to delete the specified bonding information using an AT command. In this case, delete also the bonding information saved in the counter terminal and then execute the pairing sequence.
For a passkey used at Passkey Entry, a random number value or fixed value can be used using the setting of a system parameter on the peripheral device side of the MK715x1 AT command application.

4.2.5. Other Services

The MK715x1 AT command application provides the battery level notification and device information services as well as LAPIS VSSPP. This section describes the functions of the services other than LAPIS VSSPP service.

4.2.5.1. Battery Level Notification Service

The battery level is updated automatically by the updating interval (LC_BAS_UPDATE_INTERVAL) of the system parameter. After a Bluetooth low energy connection is established, make the central device issue a read request to the battery level characteristic to obtain the battery level ((1) in the following figure). Then, the battery level applies the Notification. The AT command application for MK715x1 notifies the battery level by the notification interval (LC_BAS_NTF_INTERVAL) of the system parameter if the CCCD is updated to the Notify Property Enabled from the central device ((2) in the following figure).

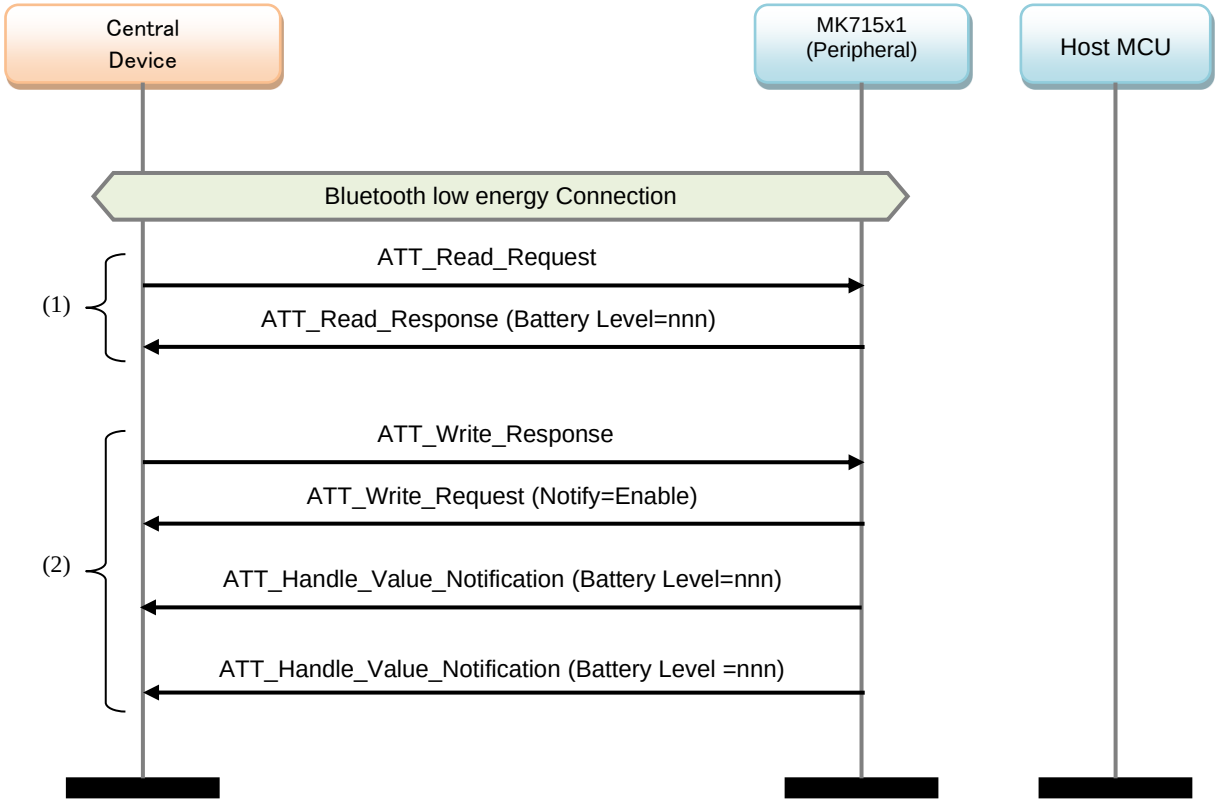


Figure 4-13 Battery Level Notification Service Sequence

4.2.5.2. Device Information Service

The AT command application supports the device information service (DIS：Device Information Service) which is a Bluetooth Core Spec standard service. The following table shows default settings. Delete a characteristic and change the value according to the customer system used. Each piece of device information can be obtained by making the central device issue a read request to a characteristic of DIS as with the battery level notification service.

Table 4-1 Device Information Default Settings

DIS characteristic name	Value
Manufacture Name String	"LAPIS SEMICONDUCTOR Co., Ltd."
Model Number String	MK71511: "MK71511" MK71521: "MK71521"
Serial Number String	"SN-1000123"
Firmware Revision String	MK71511: "s140_nrf52_7.0.1" MK71521: "s132_nrf52_7.0.1"
Hardware Revision String	"BLE-5.1.0-001"
Software Revision String	"LBLE_APP-001-1-00"
System ID	0x123456FFFE9ABCDE
Regulatory Certification Data List	0x00000000
PnP ID	0x01790101000001

4.3. UART Flow Control

The AT command application performs UART flow control using the UART-RTS output pin (P0.05) of MK715x1. When UART-RTS (P0.05) makes High output, it indicates that the UART receive buffer in MK715x1 is full. Therefore, input an AT command or transmit data from the host MCU after checking that UART-RTS (P0.05) makes Low output.

4.4. Low Current Consumption Control

The AT command application provides two types of low current consumption control. One method makes MK715x1 enter the low current consumption state using the MK715x1 Wakeup (P0.22) pin, and the other method changes the connection interval dynamically to reduce the average current consumption.

4.4.1. Low Current Consumption Control through GPIO0 Control

The low current consumption control can be performed using the MK715x1 Wakeup (P0.22) pin. The host MCU sets Wakeup (P0.22) to High to request the transition to the low current consumption state from MK715x1. At this time, MK715x1 enters the low current consumption state except when a Bluetooth low energy communication is performed, and thus AT commands or transmission data cannot be input. Set the Wakeup (P0.22) pin to Low and then input an AT command or transmission data.
The following figure simply shows changes in current consumption. Setting the Wakeup (P0.22) pin to High helps reduce the current consumption of the shaded areas in the figure where a Bluetooth low energy communication is not performed.

（Rx/Tx: Bluetooth low energy data communication intervals）

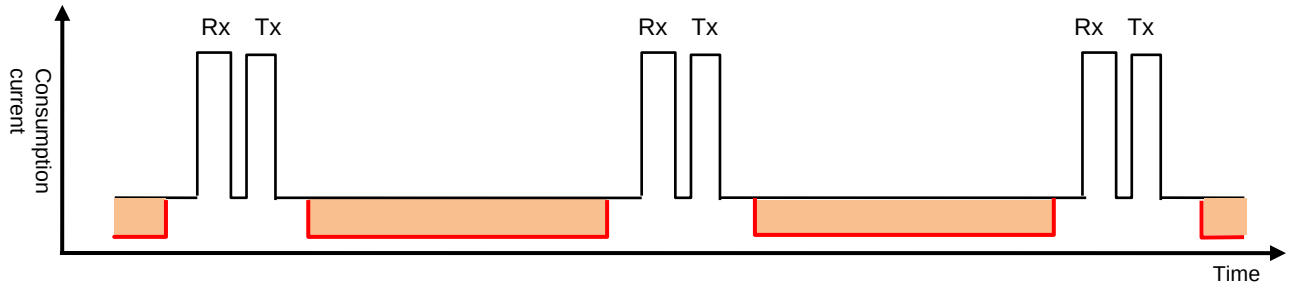


Figure 4-14 Changes in Current Consumption in Low Current Consumption State

4.4.2. Low Current Consumption Control through Connection Interval Control

The AT command application provides the function to change the connection interval dynamically. When this function is enabled, the connection parameters are updated using the settings of low power mode parameters normally. If a valid data communication does not take place, the parameters are updated instead of using the settings of normal mode parameters. This enables reduction of the average current consumption.
The following figure simply shows changes in current consumption.

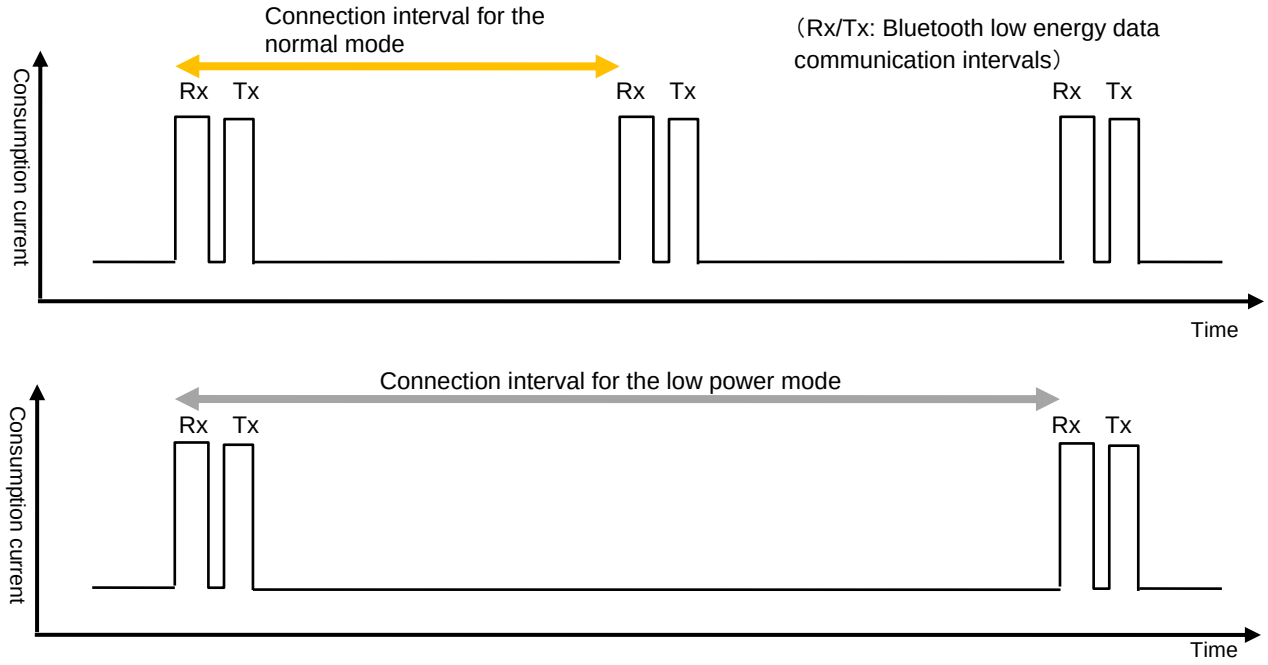


Figure 4-15 Changes in Current Consumption for Different Connection Intervals

The following figure shows the update sequence of connection interval.

- Immediately after a Bluetooth low energy connection is established, the connection interval specified by the central device is used ((1) in the following figure).
- After that, the peripheral device for MK715x1 issues a connection parameter update request for the connection interval for the normal mode. When the central device accepts this request and completes the update procedure, the connection interval for the normal mode is applied ((2) in the following figure).
- If data communication does not take place for a certain period, the peripheral device for MK715x1 issues a connection parameter update request for the connection interval for the low power mode. When the central device accepts this request and completes the update procedure, the connection interval for the low power mode is applied ((3) in the following figure).
- When data communication takes place, the peripheral device for MK715x1 issues a connection parameter update request for the connection interval for the normal mode. When the connection parameter update procedure is completed, the connection interval for the normal mode is applied ((4) in the following figure).

* There are some other cases where a connection parameter update request is issued from the central side.

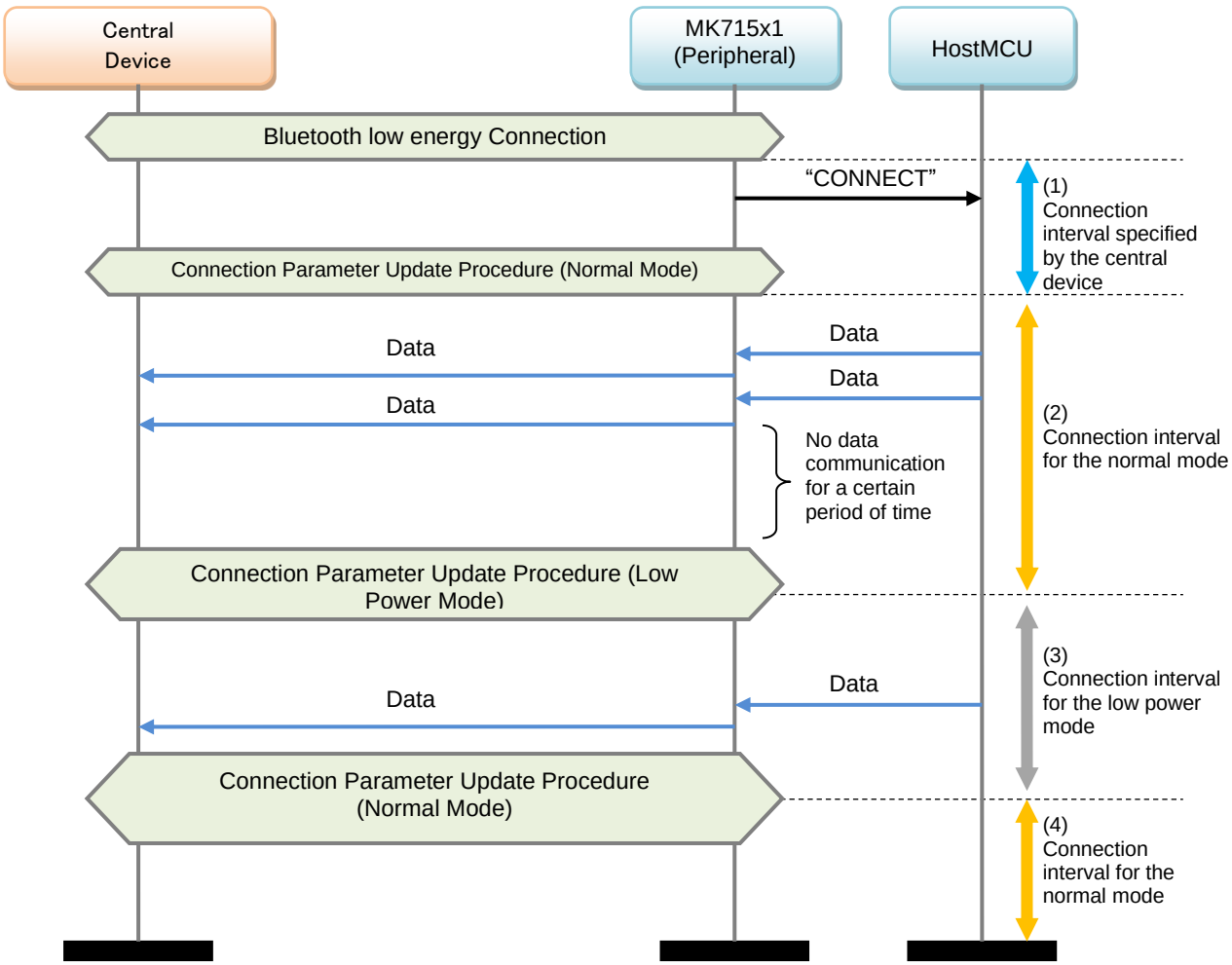


Figure 4-16 Connection Interval Update Sequence

4.5. AT Command Specifications

The host MCU performs various controls by sending an AT command to the MK715x1AT command application. As a response to the AT command, the AT command application returns result code. Also, the parameters related to Bluetooth low energy or UART communication can be modified by the system parameters.

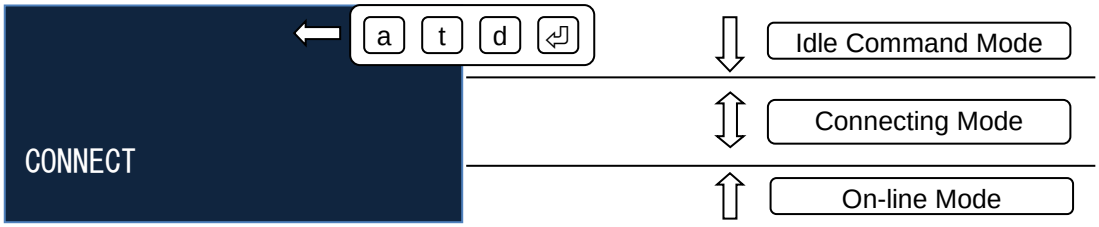
4.5.1. AT Command Operation

4.5.1.1. Connection in Peripheral Operation

The following describes the AT command operation related to the connection in peripheral operation.

(1) Peripheral connection (no pairing, encrypted communication without passkey change, reconnection)

For Peripheral connection (no pairing, encrypted communication without passkey change, reconnection), when "atd<CR>" is input, the connecting mode is entered to start transmission of advertisement. When the connection is established, the CONNECT result code is displayed and the on-line mode is entered. The same operation is performed to reconnect to the device with succeeded pairing.

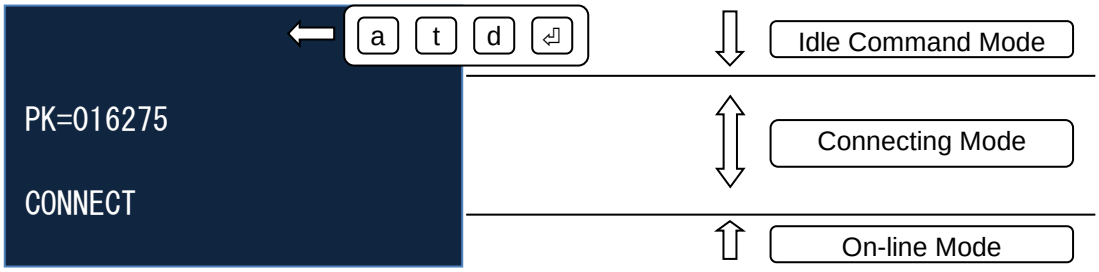


•"atd" has not output because the echo-back function is disabled on MK715x1.

Figure 4-17 Peripheral Connection (1)

(2) Peripheral connection (encrypted communication with passkey change)

For Peripheral connection (encrypted communication with passkey change), when "atd<CR>" is input, the connecting mode is entered to start transmission of advertisement. When the passkey change succeeded after connection, the CONNECT result code is displayed and the on-line mode is entered.

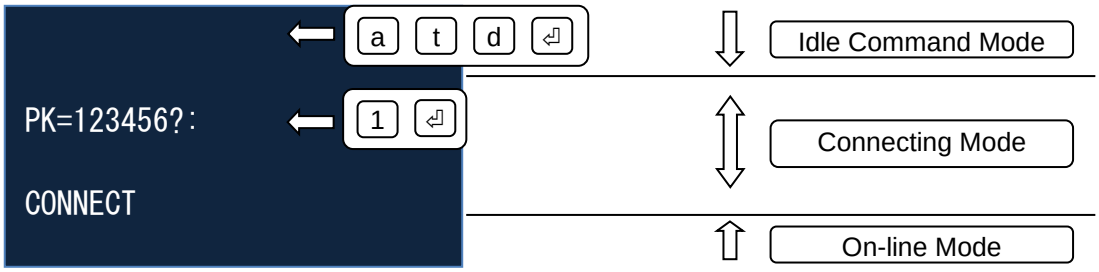


•"atd" has not output because the echo-back function is disabled on MK715x1.

Figure 4-18 Peripheral Connection (2)

(3) Peripheral connection (encrypted communication with passkey comparison)

For Peripheral connection (encrypted communication with passkey comparison), when "atd<CR>" is input, the connecting mode is entered to start transmission of advertisement. When the passkey comparison succeeded after connection, the CONNECT result code is displayed and the on-line mode is entered.

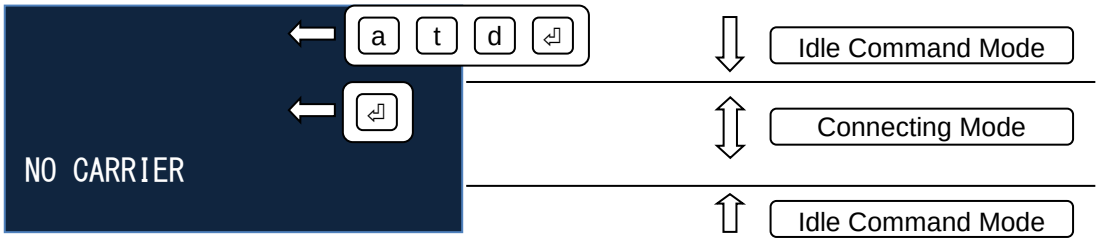


•"atd" has not output because the echo-back function is disabled on MK715x1.

Figure 4-19 Peripheral Connection (3)

(4) Peripheral Connection Canceling

When "<CR>" is input during the time from the input of "ATD<CR>" to the display of CONNECT result code, the AT command application transitions from the connecting mode to the idle command mode to cancel the connection operation. The NO CARRIER result code is returned when the connection is canceled.



•"atd" has not output because the echo-back function is disabled on MK715x1.

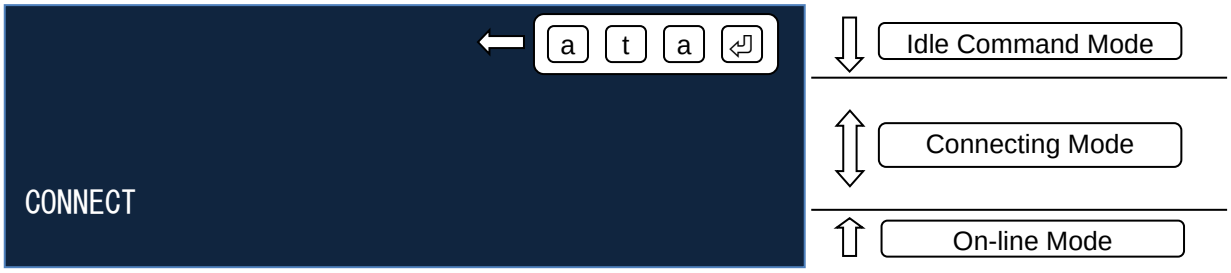
Figure 4-20 Peripheral Connection Canceling

4.5.1.2. Connection in Central Operation

The following describes the AT command operation related to the connection in central operation.

(1) Central connection (no pairing, encrypted communication without passkey change, reconnection)

For the central connection (no pairing, encrypted communication without passkey change, reconnection), when "ATA<CR>" is input, the connecting mode is entered and the scanning is started, and the advertising is received from the peripheral devices. Next, the peripheral device for connection is decided by the device name of the received advertising, and the connection is requested. When the connection is success, "CONNECT" result code is output and "On-line Mode" is entered. The advertising device name (LS_ADV_DEVICE_NAME) of the system parameter is set to decide the peripheral device for the connection.

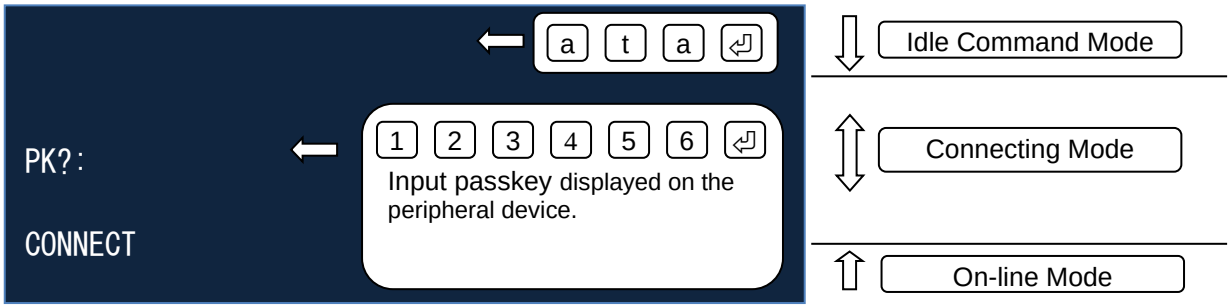


•"ata" has not output because the echo-back function is disabled on MK715x1.

Figure 4-21 Central Connection (1)

(2) Central connection (encrypted communication with passkey change)

The central connection (encrypted communication with passkey change) starts the scanning operation by "ATA<CR>", and the advertising from the peripheral device is received. Next, the connection device is decided from the received advertising, the connection is requested. When the passkey change succeeded after connection, the CONNECT result code is displayed and the on-line mode is entered. The device name for the connected peripheral device is set to the advertising device name (LS_ADV_DEVICE_NAME) of the system parameter.

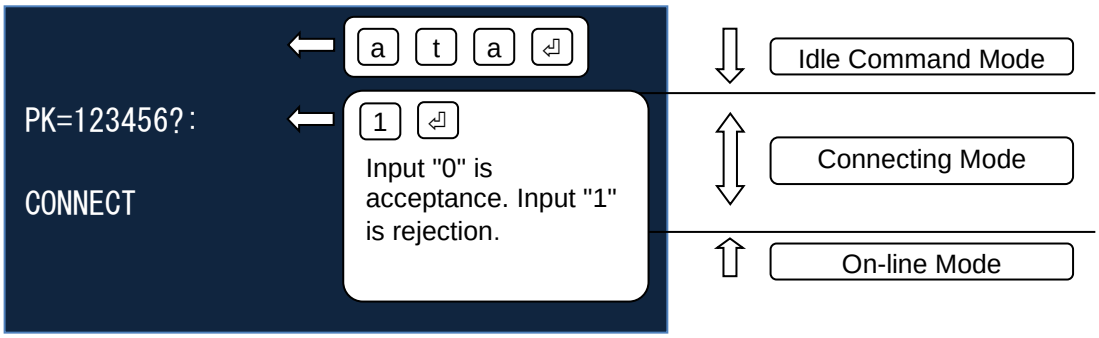


•"ata" has not output because the echo-back function is disabled on MK715x1.

Figure 4-22 Central Connection (2)

(3) Central connection (encrypted communication with passkey comparison)

The central connection (encrypted communication with passkey comparison) starts the scanning operation by "ATA<CR>", and the advertising from the peripheral device is received. Next, the connection device is decided from the received advertising, the connection is requested. When the passkey change succeeded after connection, the CONNECT result code is displayed and the on-line mode is entered. The device name for the connected peripheral device is set to the advertising device name (LS_ADV_DEVICE_NAME) of the system parameter.

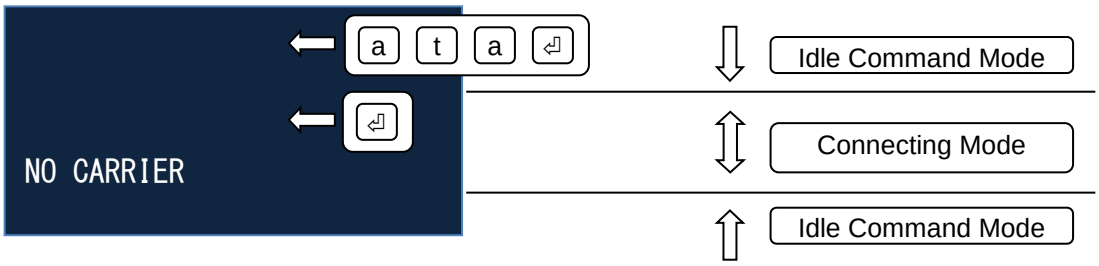


•"ata" and passkey comparison entering has not output because the echo-back function is disabled on MK715x1.

Figure 4-23 Central Connection (3)

(4) Central connection canceling

When "<CR>" is input during the time from the input of "ATA<CR>" to the display of CONNECT result code, the AT command application transitions from the connecting mode to the idle command mode to interrupt the connection operation. The NO CARRIER result code is returned when the connection is canceled.



•"ata" has not output because the echo-back function is disabled on MK715x1.

Figure 4-24 Central Connection Canceling

4.5.1.3. Disconnect

The following describes the AT command operation related to the disconnection.

(1) Disconnection Preparing

To execute disconnection, the on-line mode needs to be changed to the on-line command mode in advance. Input the escape code "+++AT<CR>" to change the mode. Data cannot be transmitted and received in the on-line command mode.

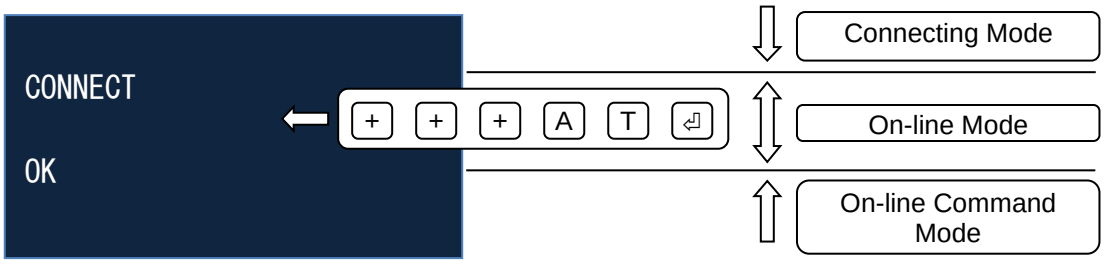
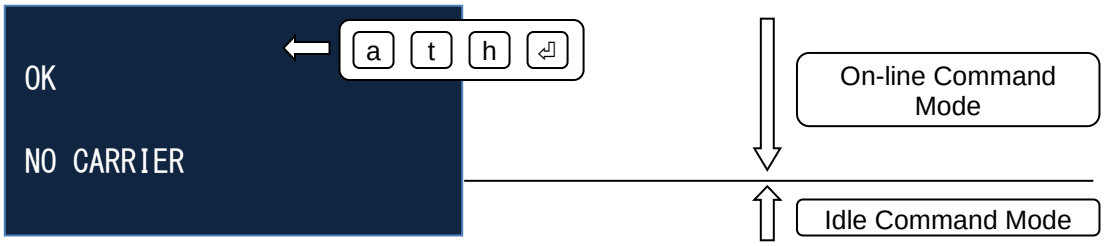


Figure 4-25 Disconnection Preparing

(2) Disconnection Executing

Disconnection is executed by "ATH<CR>" input in the on-line command mode. After the disconnection is completed, the AT command application enters the idle command mode with NO CARRIER result code displayed.



•"ath" has not output because the echo-back function is disabled on MK715x1.

Figure 4-26 Disconnection Executing

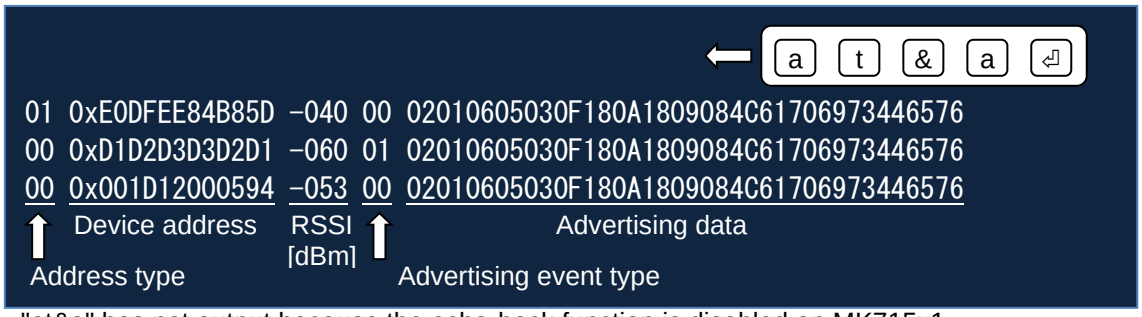
4.5.1.4. Continuous Scanning

The following describes the AT command operation related to the continuous scanning.

(1) Start Continuous Scanning

Continuous scanning is started by “AT&A<CR>”, and it outputs an advertising report of detected peripheral devices. The output format of advertising report can be changed to the following by the system parameter.

・Character Output Format



•"at&a" has not output because the echo-back function is disabled on MK715x1.

Figure 4-27 Start Continuous Scanning (Character Output Format)

・Binary Output Format

•"at&a" has not output because the echo-back function is disabled on MK715x1.

Figure 4-28 Starting Continuous Scanning (Binary Output Format)

Table 4-2 Continuous Scanning Binary Output Format

Parameter	Size [byte]	Description	Value [Hex]
Event_code	1	LE event code	3E
Data_Length	1	Advertising report total length	XX
Sub_Event_Code	1	Advertising report sub-event code	02
Num_Reports	1	Number of advertising reports Valid range: 1 only	01
Evet_Type	1	Advertising PDU type 0x00: ADV_IND 0x01: ADV_DIRECT_IND 0x02: ADV_SCAN_IND 0x03: ADV_NONCONN_IND 0x04: SCAN_RSP	XX
Address_Type	1	Address type	XX
Address	6	Device address	XX XX XX XX XX XX
Length_Data	1	Advertising data length	XX
Data	Length_Data	Advertising data	XX XX・・・・・・ XX
RSSI	1	RSSI (Signed Integer) Valid range: -127[dBm] ~ 20[dBm], 127 Invalid)	XX

(1) Stop Continuous Scanning

Continuous scanning is stopped by “<CR>”.

・Character output format

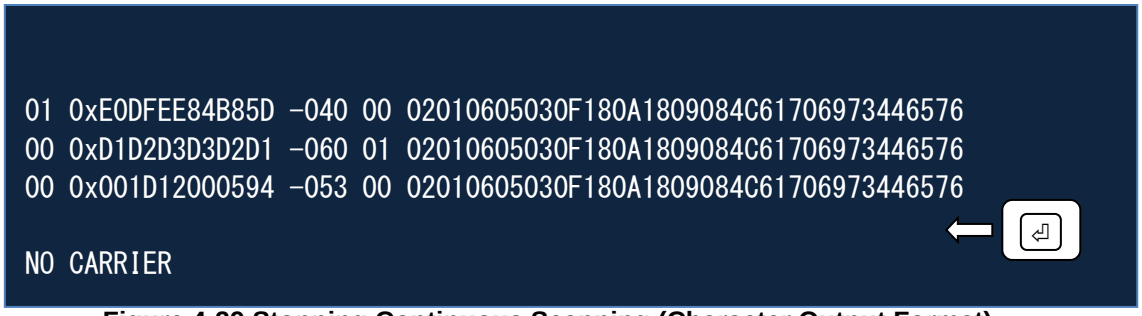


Figure 4-29 Stopping Continuous Scanning (Character Output Format)

・Binary output format

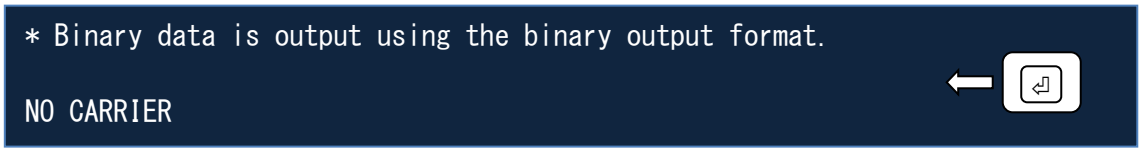


Figure 4-30 Stopping Continuous Scanning (Binary Output Format)

4.5.2. AT Command Structure

4.5.2.5. Command Format

Figure 4-31 AT Command Format

AT commands start with <AT> or <at>. The <string> before the carriage return code is parsed as a command. Commands are sequentially executed. After completion of the parsing, result code is returned. Limitations are imposed on AT commands and S registers that can be continuously input. Wait for the result code, and then input the next command.
The A and D commands have to be placed at the end of the series of commands. Any subsequence commands are not executed. The command string can contain up to 16 characters. (spaces <20h> are ignored).
Input characters can be deleted only with the backspace function. <CR> is the carriage return code.
Command characters are not case-sensitive. However, At and aT are not recognized.
For these AT commands, the speed and format are not automatically detected. They are based on the system parameter settings for UART.

For the support commands of AT commands, refer to "4.5.2.7 AT Command List". For the input of unlisted command characters and parameters, the subsequent operations are not guaranteed.

4.5.2.6. Communication Speed and Character Format

Communication speed: 1000000bps/921600bps/460800bps/250000bps/230400bps/115200bps/76800bps/57600bps/
56000bps/38400bps/31250bps/28800bps/19200bps/14400bps/9600bps/4800bps/2400bps/1200bps
(Set in the UART communication speed for the system parameter)

Character format: Refer to the following table.

Table 4-3 Character Format

Start Bit	Data Bit	Parity Bit	Stop Bit	Bit Length
1	8	None	1	10

[Notices]
If parity, flaming, and overrun errors occur, error processing (such as data discarded) is not performed.

4.5.2.7. AT Command List

Table 4-4 AT Command List

Command	Format	Function	Factory default
A	ATA	Starts connection as the central device.	
D	ATD	Starts connection as the peripheral device.	
H	ATH	Disconnects connection in the on-line command mode.	
***C**	AT*Cn	Erases the address of the paired peer terminal. n=0: Erase all areas n=1 to 5: Erase a specified area The "AT*D" command can be used to show the number "n" where the address of the peer terminal is saved.	
***D**	AT*D	Displays the address of the paired peer terminal. Display example) AT*D Dev01:P:0x00121D000001 Dev02:P:0x00121D000002 Dev03:P:0x00121D000003 Dev04:P:0x00121D000004 Dev05:P:0x00121D000005 OK "P" (public address) becomes "R" when a random address is used.	

4.5.3. Escape Code

Table 4-5 Escape Code

Format	Function
+++AT<CR>	An escape code used to change the mode from the on-line to on-line command mode. Avoid this character sequence from being included in data.

4.5.4. Result Code

4.5.4.1. Result Code Format

Result code is returned for an executed AT command.

The two result code formats are as follows.

For numeric result code:

Figure 4-32 Result Format

4.5.4.2. Result Code List

Table 4-6 Result Code List

Character code	Numeric code	Meaning
OK	Commands executed successfully	
CONNECT	Connected	
ERROR	An error in commands	
NO CARRIER	Connection interrupted or communication disconnected	
PK?:	Input the 6-digit passkey displayed on the peer device.	
PK = xxxxxx	Display a passkey (xxxxxx).	
PK = xxxxxx?:	Display a passkey (xxxxxx) for the 6-digit passkey comparison of the local device and peer device. And, the confirmation number is input. "0" is acceptance. "1" is rejection.	

4.5.5. System Parameters

The AT command application for MK715x1 prepares the system parameter for the variable of Bluetooth wireless technology. When the system parameter is modified, the following file of "MK715x1 Software Evaluation Kit" is modified and it is necessary to create code for the AT command application.

Folder : <nRF SDK Folder> ¥examples¥lapis¥mk715x1_at_cmd¥src
File : at_cmd_cfg.h

For the system parameter in the list table, the out value of range doesn't guarantee operation.

4.5.5.1. System Parameters List

For the communication parameters of the system parameter, see and understand the Bluetooth Core Specification.

Table 4-7 System Parameters List

Parameter (Definition)	Functional description	Factory default
UART communication speed (LS_UART_BR)	0 : 1,200 bps 1 : 2,400 bps 2 : 4,800 bps 3 : 9,600 bps 4 : 14,400 bps 5 : 19,200 bps 6 : 28,800 bps 7 : 31,250 bps 8 : 38,400 bps 9 : 56,000 bps 10 : 57,600 bps 11 : 76,800 bps 12 : 115,200 bps 13 : 230,400 bps 14 : 250,000 bps 15 : 460,800 bps 16 : 921,600 bps 17 : 1000,000 bps	10 (57,600bps)
Transmission power for advertising (LC_RF_TX_POWER_ADV)	MK71511 -40, -20, -16, -12, -8, -4, 0, +2, +3, +4 [Unit: dBm] MK71521 -40, -20, -16, -12, -8, -4, 0, +3, +4 [Unit: dBm]	0dBm
Transmission power setting for connection (LC_RF_TX_POWER_CONN)	MK71511 -40, -20, -16, -12, -8, -4, 0, +2, +3, +4 [Unit: dBm] MK71521 -40, -20, -16, -12, -8, -4, 0, +3, +4 [Unit: dBm]	0dBm
Inactivity monitoring time (LS_CONN_INACT_TIMER)	If there is no data communication over the period set in this register, the communication parameter is switched to the low power mode setting. Setting to 0 disables the switching function to the low power mode. 0: Invalid 500 ~ 18000 [Unit: 10ms]: 5sec ~ 180sec	3000 (30sec)
Connection interval in the normal mode (LS_NORMAL_CI)	8 ~ 4000 [Unit: ms] (Note-1) (Note-3)	40ms
Slave latency in the normal mode (LS_NORMAL_SL)	Number of connection intervals skipped by the peripheral device 0 ~ 499	0
Connection monitoring timeout in the normal mode (LS_NORMAL_SVTO)	100 ~ 3200 [Unit: 10ms]: 1000ms ~ 32sec	256 (2560ms)
Connection interval in the low power mode (LS_LP_CI)	10 ~ 4000 [Unit: ms] (Note-1) (Note-3)	100ms

Parameter (Definition)	Functional description	Factory default
Slave latency in the low power mode (LS_LP_SL)	Number of connection intervals skipped by the peripheral device 0 ~ 499	0
Connection monitoring timeout in the low power mode (LS_LP_SVTO)	100 ~ 3200 [Unit: 10ms]: 1000ms ~ 32se	512 (5120ms)
Maximum data length to transmit (LS_DATA_TX_LEN)	27 ~ 251 [Unit: byte]	27byte
Maximum data transmission time (LS_DATA_TX_TIME)	328 ~ 2120 [Unit: µs]	328µs
PHY Updating (LS_PHY_UPDATE)	bit 0: LE 1M PHY Request bit 1: LE 2M PHY Request bit 2: LE Coded PHY Request S=8 (125 kbit/s) bit3-7: Reserved When all bit is set to 0, it is auto-setting. "bit 2: LE Coded PHY Request is MK71511 only.	0x00
FAST advertising interval (LS_FAST_ADV_INTERVAL)	20 ~ 1000 [Unit: ms] (Note-2)	20ms
SLOW advertising interval (LS_FAST_ADV_INTERVAL)	1000 ~ 10240 [Unit: ms] (Note-2)	1000ms
Mode switching time from FAST to SLOW advertising (LS_ADV_INACT_TIMER)	0: Invalid 1 ~ 18000 [Unit: 10ms]: 10ms ~ 180sec	3000 (30sec)
Advertising packet type (LS_ADV_TYPE)	0: ADV_IND 1: ADV_DIRECT_IND (high duty cycle) 2: ADV_SCAN_IND 3: ADV_NONCONN_IND 4: ADV_DIRECT_IND (low duty cycle) (Note-5)	0
Peer address in direct advertising (LS_DIRECT_ADDR)	000000000000-FFFFFFFFFFFF	0x000000000000
Peer address type in direct advertising (LS_DIRECT_ADDR_TYPE)	0: Public address 1: Static random address	0
Advertising channel map (LS_ADV_CH_MAP)	1: CH37 Enable 2: CH38 Enable 3: CH37/38 Enable 4: CH39 Enable 5: CH37/39 Enable 6: CH38/39 Enable 7: CH37/38/39 Enable	7
Advertising data (LS_ADV_DAT)	Use a hexadecimal number and a format conforming to Bluetooth Core Specification to change data.	02010605030F 180A18
Scanning response data (LS_SCAN_RSP_DAT)	Use a hexadecimal number and a format conforming to Bluetooth Core Specification to change data.	09FF79010A0B0C0D0E0F
Advertising device name (LS_ADV_DEVICE_NAME)	Advertising data /Scanning response data The device name is added to the advertising data /scanning response data. When the central device, it is used for the connection.	"LapisDev"
Advertising device name type (LS_ADV_DEVICE_NAME_TYPE)	0: not added to advertising data, not added to scanning response data 1: added to advertising data, not added to scanning response data (Default) 2: not added to advertising data, added to scanning response data 3: added to advertising data, added to scanning response data	1

Parameter (Definition)	Functional description	Factory default
Scanning time (LS_SCAN_TIME)	50 ~ 4000 [Unit: 10ms]: 500ms ~ 40sec	200 (2sec)
Scanning device max. number (LS_SCAN_NUM_MAX)	When scanning, it is the number of maximum devices that can be detected during the scanning time.	8
Scanning type (LS_SCAN_TYPE)	0: Passive scanning 1: Active scanning	0
Scanning interval (LS_SCAN_INTERVAL)	3 ~ 10240 [Unit: ms]	60ms
Scanning window (LS_SCAN_WINDOW)	3 ~ 10240 [Unit: ms]	60ms
RSSI filter for scanning (LS_SCAN_RSSI_FILTER)	-128 ~ 127 [Unit: dBm]	-128dBm
Continuous Scanning Enabled (LS_CONT_SCAN_ENA)	0: Enabled 1: Disabled	0 (MK71511) 1（MK71521)
Advertising report output format (LS_SCAN_ADV_REPORT_FMT)	0: Character output 1: Binary output	0
Pairing mode (LC_SM_PAIR_MODE)	0: No pairing (no encryption during communication) 1: Legacy Just Works (passkey not exchanged) 2: Legacy Passkey Entry (passkey exchanged) 3: LESC Just Works (passkey not exchanged) 4: LESC Passkey Entry (passkey exchanged) 5: LESC Numeric Comparison（passkey compared） (Note-4)	0
Fixed passkey enabled (LC_SM_FIXED_PASSKEY_ENA)	0: Fixed passkey disabled (generation of random numbers) 1: Fixed passkey enabled	0
Fixed passkey (LC_SM_FIXED_PASSKEY)	000000-999999	123456
MTU size of the attribute layer (LS_ATT_MTU)	23 ~ 247	23
Device address mode (LC_GAP_ADDR_MODE)	0: FICR device address 1: Public address (Fixed) 2: Random address (Fixed) 3: Random address (Random numbers generation)	0
Fixed device address (LC_GAP_FIXED_ADDR)	000000000000-FFFFFFFFFFFF	0xD1D2D3D4D5D6
GAP device name (LC_GAP_DEVICE_NAME)	GAP device name setting	MK71511, "LAPIS MK71511 Application" MK71521, "LAPIS MK71521 Application"
BAS: Updating interval (LC_BAS_UPDATE_INTERVAL)	1 ～ 65535 [Unit: sec]	60sec
BAS: Notification interval (LC_BAS_NTF_INTERVAL)	0: Invalid 1 ～ 65535 [Unit: sec]	10sec
DIS: Manufacturer Name (LC_DIS_MANUFACTURER_NAME)	Manufacturer Name setting for device information.	"LAPIS SEMICONDUCTOR Co., Ltd."
DIS: Model Number (LC_DIS_MODEL_NUM)	Model Number setting for device information.	MK71511, "MK71511" MK71521, "MK71521"
DIS: Manufacturer-defined Identifier (LC_DIS_MANUFACTURER_ID)	Manufacturer-defined Identifier setting for device information.	0xFEFF563412
DIS: OUI (LC_DIS_ORG_UNIQUE_ID)	System ID/Organizationally Unique Identifier (OUI) setting for device information.	0xDEBC9A
DIS: Serial Number (LC_DIS_SERIAL_NUM)	Serial Number setting for device information.	"SN-1000123"
DIS: Hardware Revision (LC_DIS_HW_REV)	Hardware Revision setting for device information.	"BLE-5.1.0-001"
DIS: Firmware Revision (LC_DIS_FW_REV)	Firmware Revision setting for device information.	MK71511, "s140_nrf52_7.0.1" MK71521, "s132_nrf52_7.0.1"
DIS: Software Revision	Software Revision setting for device information.	"LBLE_APP-001-1.00"

Parameter (Definition)	Functional description	Factory default
(LC_DIS_SW_REV)		
DIS: Regulatory Certification Data List (LC_DIS_REG_CERT_DATA_LIST)	IEEE 11073-20601 Regulatory Certification Data List setting for device information.	0x00000000
DIS: Vendor ID source (LC_DIS_VENDOR_ID_SOURCE)	PnP ID/Vendor ID source setting for device information.	0x01
DIS: Vendor ID (LC_DIS_VENDOR_ID)	PnP ID/Vendor ID setting for device information.	0x0179
DIS: Product ID (LC_DIS_PRODUCT_ID)	PnP ID/Product ID setting for device information.	0x001
DIS: Product Version (LC_DIS_PRODUCT_VERSION)	PnP ID/Product Version setting for device information.	0x0100
VSSPP: VSA property (LS_VSSPP_PROPERTY)	0: Write w/o Response & Notify 1: Write & Notify 2: Write w/o Response & Indicate 3: Write & Indicate	0
Host Wakeup enable (LS_HOST_WAKEUP_ENA)	0: Host Wakeup output pin disabled 1: Host Wakeup output pin enabled (Low Active) 2: Host Wakeup output pin enabled (High Active)	0
Host Wakeup time (LS_HOST_WAKEUP_TIME)	Output delay time of UART transmission data is set when the Host Wakeup output pin is enabled. 0: Invalid 10～ 10000 [Unit: ms]	10

(Note-1) N * 1.25 ms should be used to specify a connection interval in Bluetooth Core Specification. Thus, the value set in the S register is converted to a value close to and not in excess of the setting value. For example, when 8 is set to the S register, 7.5 ms (N=6) is used as an actual value. The low power mode and normal mode may not be switched as expected because of a difference caused by this conversion. Therefore, the setting of a value divided evenly by 5 ms is recommended for the system parameter.

(Note-2) N * 0.625 ms should be used to specify an advertising interval in Bluetooth Core Specification v4.2. Thus, the value set in the S register is converted to a value close to and not in excess of the setting value. For example, when 21 is set to the S register, 20.625 ms (N=33) is used as an actual value.

(Note-3) This is used as a Min value for a change request of a connection parameter. A Max value is a setting value plus 20 (the maximum value is 4000). Therefore, satisfy the following condition to make the setting:

LS_LP_CI > LS_NORMAL_CI + 20

(Note-4) When the setting of the paring mode is changed, delete the existing paired peer information and pairing information about the peer device.

(Note-5) When ADV_SCAN_IND(0x02) or ADV_NONCONN_IND(0x03) is set and the advertising interval for FAST Advertising is less than 100ms, the advertising packet is transmitted at 100ms intervals.

5. Software Development

This chapter describes the construction procedure of development environment of sample software that LAPIS provides, and the software processing of various functions in AT command application.

5.1. LAPIS Development Environment Construction

The following procedure constructs the development environment for LAPIS's sample software.

5.1.1. Nordic nRF5 SDK Downloading & Installing

Refer to the "2.1. Downloading and Installing Nordic nRF5 SDK" of " MK715x1 Software Development Start-up Guide" for the downloading and installing of Nordic SDK.

5.1.2. Segger Embedded Studio Downloading & Installing

The building environment for the AT command application uses the Segger Embedded Studio. Refer to the "2.4. Installing Segger Embedded Studio" of "MK715x1 Software Development Start-up Guide" for the Segger Embedded Studio downloading and installing.

5.1.3. MK715x1 Software Development Kit Downloading & Installing

The file (ZIP file) of "MK715x1 Software Development Kit" is downloaded from the following Bluetooth low energy of LAPIS support site and unzips it. Next, the following folder and file of copy source of "MK715x1 Software Development Kit" is copied (overwrite) to the following copy destination of Nordic SDK.

LAPIS Support Site: https://www.lapis-semi.com/cgi-bin/MyLAPIS/regi/login.cgi (English)
https://www.lapis-semi.com/cgi-bin/MyLAPIS/regi/login_J.cgi (Japanese)

MK715x1 Software Development Kit ZIP File: mk715x1_sdk_verXXX.zip *XXX is version number.

Copy Source Folder & File: All folder and file under the following path for MK715x1 software development kit.
".¥mk715x1_sdk_verXXX¥software¥"

Copy Destination: The following folder for Nordic SDK.
".¥<nRF5 SDK>¥"

5.1.4. Folder Structure for LAPIS Development Environment

The following figure shows the folder structure for the LAPIS development environment.

```
.¥<nRF5 SDK>¥
   ├──components                                          // Nordic社 Component Files
   │   └──ble
   │       ├──ble_services                                // LAPIS Additional Service Module
   │       │   └──ble_vsspp                              // VSSPP
   │       │          ble_nus.c
   │       │          ble_nus.h
   │       │          ble_nus_c.c
   │       │          ble_nus_c.h
   │       └── etc.
   └──examples                                            // Nordic社 Sample Softwares
       └──lapis                                           // LAPIS Additional Sample Softwares
           └──mk715x1_at_cmd                              // AT Command Application Files
               ├──hex                                     // AT Command Application Hex File
               │      mk71511ek1_at_cmd_rXXX.hex
               │      mk71511ek1a_at_cmd_rXXX.hex
               │      mk71521ek1_at_cmd_rXXX.hex
               │      mk71521ek1a_at_cmd_rXXX.hex
               │
               ├──mk71511ek1                              // Build Environment for MK71511EK1
               │   └──s140
               │       ├──config
               │       │      sdk_config.h                // Nordic社 SDK Configuration File
               │       └──ses
               │              ble_mk71511_dk_s140.emProject  // Segger Embedded Studio Project File
               │
               ├──mk71511ek1a                             // Build Environment for MK71511EK1A/ MK71511EK1AP
               │   └──s140
               │       ├──config
               │       │      sdk_config.h                // Nordic社 SDK Configuration File
               │       └──ses
               │              ble_mk71511_dk_s140.emProject  // Segger Embedded Studio Project File
               │
               ├──mk71521ek1                              // Build Environment for MK71521EK1
               │   └──s132
               │       ├──config
               │       │      sdk_config.h                // Nordic社 SDK Configuration File
               │       └──ses
               │              ble_mk71521_dk_s132.emProject  // Segger Embedded Studio Project File
               │
               ├──mk71521ek1a                             // Build Environment for MK71521EK1A/ MK71521EK1AP
               │   └──s132
               │       ├──config
               │       │      sdk_config.h                // Nordic社 SDK Configuration File
               │       └──ses
               │              ble_mk71521_dk_s132.emProject  // Segger Embedded Studio Project File
               │
               └──src                                     // AT Command Application Source code
                      app_manager.c
                      app_manager.h
                      at_cmd_cfg.h                        // System Parameters
                      at_cmd_handler.c
                      at_cmd_handler.h
                      ble_handler.c
                      ble_handler.h
                      l_ble_fw.h
                      main.c
```

Figure 5-1 Folder Structure for LAPIS Development Environment

5.2. Software Processing

The following describes the software processing of the AT command application according to the functions.

- Initial Setting
- GPIO
- UART
- BLE Service Discovery
- Log
- Application Timer
- Power Management
- BLE Stack (SoftDevice)
- Peer Manager
- Generic Access Profile (GAP)
- Generic Attribute Profile (GATT)
- Device Information Service
- Battery Service
- LAPIS Vendor Specific Service

5.2.1. Initial Setting

The following describes the software processing for the initial setting of the AT Command Application
Refer to the descriptions of each function for details.

Source Code: main.c

```
/**@brief Function for application main entry.
 */
int main(void)
{
    uint32_t err_code;

    // Initialize.
    gpio_init();                                // GPIO Initialization
    app_boot_chk();                             // Application Processing
    db_discovery_init();                        // BLE Service Discovery Module Initialization
    log_init();                                 // Log Module Initialization (Debug)
    timers_init();                              // Application Timer Initialization
    power_management_init();                    // Power Management Initialization
    ble_stack_init();                           // BLE Stack Initialization (SoftDevice Handler Initialization)
    peer_manager_init();                        // Peer Manager Initialization
    gap_params_init();                          // GAP Initialization
    gatt_init();                                // GATT Initialization
    services_init();                            // BLE Service Initialization

    // Start execution.
    NRF_LOG_INFO("LAPIS applicaton started");   // Log Output (Debug)
    at_init_req();                              // AT Command Analysis Initialization
    app_init();                                 // Application Manager Initialization
    lble_init_req();                            // BLE Handler Initialization

    app_low_power();                            // Application Processing *include UART Initialization

    // Enter main loop.
    for (;;)
    {
        idle_state_handle();
    }
}
```

5.2.2. GPIO

The following describes the software processing for the AT Command Application (for example, Wakeup port, GPIO).

(1) Initialization (Wakeup)

Source Code: main.c

```
void gpio_init(void)
{
    ret_code_t                  err_code;
    nrf_drv_gpiote_in_config_t  in_config;

    err_code = nrf_drv_gpiote_init();                           // GPIO Driver Initialization
    APP_ERROR_CHECK(err_code);

    nrf_gpio_cfg(   (uint32_t)WAKEUP_PIN_NUM,                   // GPIO Setting:        GPIO Port Number
                    NRF_GPIO_PIN_DIR_INPUT,                     //                      Input/Output
                    NRF_GPIO_PIN_INPUT_CONNECT,                 //                      Input Buffer Connection
                    NRF_GPIO_PIN_PULLUP,                        //                      Pull-up/Pull-down
                    NRF_GPIO_PIN_S0S1,                          //                      Drive Capacity
                    NRF_GPIO_PIN_SENSE_HIGH );                  //                      Wakeup Level

    in_config.sense             = NRF_GPIOTE_POLARITY_TOGGLE;   // GPIO Input Setting:  Interruput Configration
    in_config.pull              = NRF_GPIO_PIN_PULLUP;          //                      Pull-up/Pull-down
    in_config.is_watcher        = false;                        //                      Outpu Port Watcher
    in_config.hi_accuracy       = false;                        //                      Input High Accuracy
    in_config.skip_gpio_setup   = false;                        //                      GPIO Setting Skip
                                                                //                      Event Handler
    err_code = nrf_drv_gpiote_in_init(WAKEUP_PIN_NUM, &in_config, app_gpio_handler);
    APP_ERROR_CHECK(err_code);

    nrf_drv_gpiote_in_event_enable(WAKEUP_PIN_NUM, true);       // GPIO Interruput Enabled

}
```

(2) Event Handler (Wakeup)

Source Code: main.c

```
static void app_gpio_handler(nrfx_gpiote_pin_t pin, nrf_gpiote_polarity_t action)
{

    app_low_power();                                            // Application Low Power Control
}

static void app_low_power(void)
{
    ret_code_t err_code;

    (省略)

        if(nrf_gpio_pin_read(WAKEUP_PIN_NUM) == 0x00)           // Wakeup Port Input State Branch (Low)
        {
            nrf_serial_uninit(&serial0_uarte);                  // UART Disabled

            if(app_get_sts() == APP_MODE_IDLE_CMD)              // Application Operation Mode Branch (Idel Command Mode)
            {
                // Go to system-off mode (this function will not return; wakeup will cause a reset).
                err_code = sd_power_system_off();               // Low Power Mode Setting
            }
        }
        else                                                    // Wakeup Port  Input State Branch (High)
        {
            uart_init();                                        // UART Enabled
        }

}
```

5.2.3. UART

The following describes the software processing for the UART. It is controlled by API and Event Handler of the data transmission/Reception.

(1) Initialization

Source Code: main.c

```
                                                        // UART Instance
NRF_SERIAL_DRV_UART_CONFIG_DEF( m_uarte0_drv_config,                        /**< UART configuration. */
                                RX_PIN_NUMBER,
                                TX_PIN_NUMBER,
                                RTS_PIN_NUMBER,
                                CTS_PIN_NUMBER,
                                NRF_UART_HWFC_ENABLED,
                                NRF_UART_PARITY_EXCLUDED,
                                NRF_UART_BAUDRATE_57600,
                                UART_DEFAULT_CONFIG_IRQ_PRIORITY );

NRF_SERIAL_CONFIG_DEF(serial0_config, NRF_SERIAL_MODE_DMA,       // Event Handler Registration
                      &serial0_queues, &serial0_buffs, serial_event_handle, sleep_handler);

static void uart_init(void)
{
    uint32_t err_code;

                                                        // UART Initialization
    err_code = nrf_serial_init(&serial0_uarte, &m_uarte0_drv_config, &serial0_config);
    if(err_code != NRF_ERROR_MODULE_ALREADY_INITIALIZED)
    {
        APP_ERROR_CHECK(err_code);
```

(2) UART Data Reception

Source Code: main.c

```
static void uart_tx(void)
{
    ret_code_t  err_code;
    size_t      w_len;


    if(g_uart_tx_num > 0)
    {
                                                        // UART Data Transmission Request
        err_code = nrf_drv_uart_tx( &serial0_uarte.instance,
                                    &g_uart_buf[g_uart_block_r][0],
                                    (size_t)g_uart_buf_len[g_uart_block_r]);
        APP_ERROR_CHECK(err_code);

        g_uart_tx_num--;
        g_uart_buf_len[g_uart_block_r] = 0;
        g_uart_block_r++;
        if(g_uart_block_r == UART_TX_BUF_BLOCK)
        {
            g_uart_block_r = 0;
        }

        uart_ctrl_tx_type = 1;
    }
    else
    {
        uart_ctrl_tx_type = 0;

        if(nrf_gpio_pin_read(WAKEUP_PIN_NUM) == 0x00)
        {
            application_timers_ctrl(T_CTRL_ID_START_UART_CTRL_POST2);
        }
    }
}
```

(3) UART Data Transmission

Source Code: main.c

```
void serial_event_handle(nrf_serial_t const * p_serial, nrf_serial_event_t event)
{
    ret_code_t  err_code;
    uint16_t    ret_sts;
    size_t      read_cnt;


    switch (event) {

    case NRF_SERIAL_EVENT_TX_DONE:                              // UART Data Transmission Done Event

        NRF_LOG_INFO("NRF_SERIAL_EVENT_TX_DONE");

        uart_tx();

        break;

    case NRF_SERIAL_EVENT_RX_DATA:                              // UART Data Reception Event

                                                                // UART Data Reception
        err_code = nrf_serial_read( &serial0_uarte,
                                    &data_array[uart_rx_idx],
                                    m_ble_nus_max_data_len - uart_rx_idx,
                                    &read_cnt, 0 );

        if( (read_cnt > 0) && ((err_code == NRF_SUCCESS) || (err_code == NRF_ERROR_TIMEOUT)) )
        {
            ret_sts = app_dat_in(&data_array[0], read_cnt);
            if( (ret_sts == AT_TX_VALID) &&
                ((g_lble_info.state == LBLE_STS_CONNCTED_C) || (g_lble_info.state == LBLE_STS_CONNCTED_P)) )
            {
                uart_rx_idx += read_cnt;

                if(uart_rx_idx == m_ble_nus_max_data_len)
                {
                    uart_ctrl_rx_timer_handler_sub();
                }
            }
        }

        break;
    }
}
```

5.2.4. BLE Service Discovery

The following describes the software processing for the BLE service discovery.

(1) Initialization

Source Code: main.c

```
/** @brief Function for initializing the database discovery module. */
static void db_discovery_init(void)
{
    ble_db_discovery_init_t db_init;

    memset(&db_init, 0, sizeof(ble_db_discovery_init_t));

    db_init.evt_handler  = db_disc_handler;                     // BLE Service Discovery Event Handler
    db_init.p_gatt_queue = &m_ble_gatt_queue;                   // GATT Queue Management Area Setting

    ret_code_t err_code = ble_db_discovery_init(&db_init);      // BLE Service Discovery Module Initialization
    APP_ERROR_CHECK(err_code);
}
```

(2) Start BLE Service Discovery

Source Code: main.c

```
/**@brief Function for handling BLE events.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 * @param[in]   p_context   Unused.
 */
                                                                // BLE Stack Event Handler
static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
{
    ret_code_t err_code = NRF_SUCCESS;


    switch (p_ble_evt->header.evt_id)                           // Event Branch
    {
    (omit)
        case BLE_GAP_EVT_CONNECTED:                             // Event Branch (Connection Success)

            NRF_LOG_INFO("BLE_GAP_EVT_CONNECTED");

             (omit)
            if(g_lble_info.state == LBLE_STS_CONNCTED_C)        // BLE State Branch (Connection for Central)
            {
                                                                // BLE Seervice Discovery Request
                // start discovery of services. The NUS Client waits for a discovery result
                err_code = ble_db_discovery_start(&m_db_disc, p_ble_evt->evt.gap_evt.conn_handle);
                APP_ERROR_CHECK(err_code);
            }

            application_timers_ctrl(T_CTRL_ID_START_UART_CTRL_RX);
```

(3) Event Handler

Source Code: main.c

```
/**@brief Function for handling database discovery events.
 *
 * @details This function is a callback function to handle events from the database discovery module.
 *          Depending on the UUIDs that are discovered, this function forwards the events
 *          to their respective services.
 *
 * @param[in] p_event  Pointer to the database discovery event.
 */
static void db_disc_handler(ble_db_discovery_evt_t * p_evt)
{
    ble_nus_c_on_db_disc_evt(&m_ble_nus_c, p_evt);              // VSSPP Discovery
}
```

5.2.5. Log

The following describes the software processing for the Log. The AT command application outputs the log to the Debug Terminal on the Segger Embedded Studio for Debug.

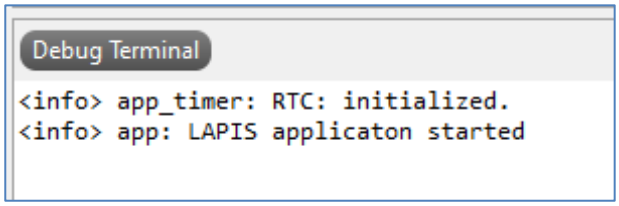


Figure 5-2 Log Output to Debug Terminal on Segger Embedded Studio

When the log output is enabled, the Nordic SDK configuration file is modified to the following definition.

Source Code: sdk_config.h

```
#define NRF_LOG_ENABLED 1

#define NRF_LOG_BACKEND_RTT_ENABLED 1

#define NRF_FPRINTF_FLAG_AUTOMATIC_CR_ON_LF_ENABLED 0
```

(1) Initialization

Source Code: main.c

```
static void log_init(void)
{
    ret_code_t err_code = NRF_LOG_INIT(NULL);                   // Log Initialization
    APP_ERROR_CHECK(err_code);

    NRF_LOG_DEFAULT_BACKENDS_INIT();
}
```

(2) Log Outout

Source Code: main.c

```
    // Start execution.
    NRF_LOG_INFO("LAPIS applicaton started");                   // Log Output
```

5.2.6. Application Timer

The following describes the software processing for the application timer as an example of the updating interval of the battery service.

(1) Initialization & Start Timer (Interval Timer for Battery Service)

Source Code: main.c

```
                                                        // Application Timer Instance Definition
APP_TIMER_DEF(m_battery_id);                                            /**< Battery timer.             */

/**@brief Function for the Timer initialization.
 *
 * @details Initializes the timer module. This creates and starts application timers.
 */
static void timers_init(void)
{
    ret_code_t err_code;

    // Initialize timer module.
    err_code = app_timer_init();                        // Application Timer Initialization
    APP_ERROR_CHECK(err_code);

    // Create timers.
    err_code = app_timer_create(    &m_battery_id,              // Timer Setting:   Updating Interval for Battery Service
                                    APP_TIMER_MODE_REPEATED,    //                  Timer Type(Repetition /One-Shot)
                                    battery_level_update_timer_handler );          Timer Handler
    APP_ERROR_CHECK(err_code);

    (省略)
                                                        // Start Timer:     Updating Interval for Battery Service
                                                        //                  Timeout Time
    err_code = app_timer_start(m_battery_id, APP_TIMER_TICKS(LC_BAS_UPDATE_INTERVAL * 1000), NULL);
    APP_ERROR_CHECK(err_code);

}
```

(2) Stop Timer (Interval Timer for Battery Service)

Source Code: main.c

```
        err_code = app_timer_stop(m_battery_id);            // Stop Timer:      Updating Interval for Battery Service
        APP_ERROR_CHECK(err_code);
```

(3) Event Handler (Interval Timer for Battery Service)

Source Code: main.c

```
/**@brief Function for handling the Battery measurement timer timeout.
 *
 * @details This function will be called each time the battery level measurement timer expires.
 *
 * @param[in] p_context  Pointer used for passing some arbitrary information (context) from the
 *                       app_start_timer() call to the timeout handler.
 */
                                                                    // Event Handler for Timeout
static void battery_level_update_timer_handler(void * p_context)
{

    UNUSED_PARAMETER(p_context);

    battery_level_update();                                         // Battery Level Updating
}

/**@brief Function for performing battery measurement and updating the Battery Level characteristic
 *        in Battery Service.
 */
static void battery_level_update(void)                              // Battery Level Updating
{
    ret_code_t err_code;
    uint8_t  battery_level;

    battery_level = app_batt_lvl_get();                             // Battery Levvel Acquisition

                                                                    // Battery Level Updating
     err_code = ble_bas_battery_level_update(&m_bas, battery_level, g_lble_info.conn_handle);
    if ((err_code != NRF_SUCCESS) &&
        (err_code != NRF_ERROR_INVALID_STATE) &&
        (err_code != NRF_ERROR_RESOURCES) &&
        (err_code != NRF_ERROR_BUSY) &&
        (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
       )
    {
        APP_ERROR_HANDLER(err_code);
    }
```

5.2.7. Power Management

The following describes the software processing for the power management. The AT command application transits to low power consumption in the idle state.

(1) Initialization

Source Code: main.c

```
/**@brief Function for initializing power management.
 */
static void power_management_init(void)
{
    ret_code_t err_code;
    err_code = nrf_pwr_mgmt_init();                                 // Power Management Initialization
    APP_ERROR_CHECK(err_code);
}
```

(2) Idle State Handler

Source Code: main.c

```
/**@brief Function for handling the idle state (main loop).
 *
 * @details If there is no pending log operation, then sleep until next the next event occurs.
 */
static void idle_state_handle(void)                             // Executed in loop of main function
{
    if (NRF_LOG_PROCESS() == false)
    {
        nrf_pwr_mgmt_run();                                     // Power Management

    }
}
```

5.2.8. BLE Stack (SoftDevice)

The following describes the software processing for the BLE stack (SoftDevice). The AT command application is implemented by API and event handler of the BLE stack.

(1) Initialization

Source Code: main.c

```
/**@brief Function for initializing the BLE stack.
 *
 * @details Initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
    ret_code_t err_code;

    err_code = nrf_sdh_enable_request();                        // SoftDevice Handler Enabled
    APP_ERROR_CHECK(err_code);

    // Configure the BLE stack using the default settings.      // BLE Stack Setting
    // Fetch the start address of the application RAM.
    uint32_t ram_start = 0;
    err_code = nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG, &ram_start);
    APP_ERROR_CHECK(err_code);

    // Enable BLE stack.
    err_code = nrf_sdh_ble_enable(&ram_start);                  // BLE Stack Enabled
    APP_ERROR_CHECK(err_code);

    // Register a handler for BLE events.                       // BLE Stack Event Handler Registration

    NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO, ble_evt_handler, NULL);
}
```

(2) Connection Request for Peripheral (Start Advertising)

Source Code: ble_handler.c

```
void lble_adv_req(uint8_t adv_mode)
{
    ret_code_t err_code;

    lble_adv_init(adv_mode);                                    // Advertising Setting

    err_code = sd_ble_gap_adv_start(m_adv_handle, 1);           // Start Advertising
    APP_ERROR_CHECK(err_code);

    g_lble_info.state = LBLE_STS_ADV;
}

static void lble_adv_init(uint8_t adv_mode)                     // Advertising Setting
{
    uint32_t err_code;
    uint16_t adv_dat_len;

    ble_gap_adv_params_t    m_adv_params;
    ble_gap_adv_data_t      *m_adv_data_p;

    ble_gap_adv_data_t      m_adv_data;

    static uint8_t  m_adv_dat_work[BLE_GAP_ADV_SET_DATA_SIZE_MAX];
    static uint8_t  m_scan_rsp_work[BLE_GAP_ADV_SET_DATA_SIZE_MAX];

    const uint8_t   adv_dat[]           = LS_ADV_DAT;
    const uint8_t   scan_rsp_dat[]      = LS_SCAN_RSP_DAT;
    const uint8_t   adv_device_name[]   = LS_ADV_DEVICE_NAME;

    // Initialize advertising parameters (used when starting advertising).
    memset(&m_adv_params, 0, sizeof(m_adv_params));
                                                                // Advertising Type Setting
    m_adv_params.properties.type    = BLE_GAP_ADV_TYPE_CONNECTABLE_SCANNABLE_UNDIRECTED;

    m_adv_params.interval       = LS_FAST_ADV_INTERVAL;         // Advertising Interval Setting
    m_adv_params.duration       = (uint32_t)LS_ADV_INACT_TIMER; // Advertising Time Setting

    m_adv_params.filter_policy      = BLE_GAP_ADV_FP_ANY;       // Advertising Filter Policy Setting
                                                                // Advertising Channel Map Setting
    m_adv_params.channel_mask[4]    = ((uint8_t)LS_ADV_CH_MAP ^ 0x07) << 5;

    adv_dat_len = sizeof(adv_dat);                              // Advertising Data Setting
    memcpy(&m_adv_dat_work[0], adv_dat, adv_dat_len);
    m_adv_data.adv_data.len = adv_dat_len;

    m_adv_data.adv_data.p_data = &m_adv_dat_work[0];

    adv_dat_len = sizeof(scan_rsp_dat);                         // Scanning Response Data Setting
    memcpy(&m_scan_rsp_work[0], scan_rsp_dat, adv_dat_len);
    m_adv_data.scan_rsp_data.len = adv_dat_len;

    m_adv_data.scan_rsp_data.p_data = &m_scan_rsp_work[0];

    m_adv_data_p = &m_adv_data;
                                                                // Advertising Setting Request
    err_code = sd_ble_gap_adv_set_configure(&m_adv_handle, m_adv_data_p, &m_adv_params);
    APP_ERROR_CHECK(err_code);
                                                                // Advertising TX Power Setting
    err_code = sd_ble_gap_tx_power_set( BLE_GAP_TX_POWER_ROLE_ADV,
                                        m_adv_handle,
                                        (int8_t)LC_RF_TX_POWER_ADV);
    APP_ERROR_CHECK(err_code);
}
```

(3) Disconnection

Source Code: ble_handler.c

```
uint16_t lble_disc_req(void)
{
    uint16_t    ret = (uint16_t)LBLE_SUCCESS;


    if( (g_lble_info.state == LBLE_STS_CONNCTED_P)
        || (g_lble_info.state == LBLE_STS_CONNCTED_C) )
    {
        g_lble_info.state = LBLE_STS_DISC;
                                                        // Disconnection Request
        (void)sd_ble_gap_disconnect(    g_lble_info.conn_handle,
                                        BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
    }
    else
    {
        ret = (uint16_t)LBLE_ERR_STS;
    }


    return ret;
}
```

(4) Event Handler

Source Code: main.c

```
/**@brief Function for handling BLE events.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 * @param[in]   p_context   Unused.
 */
static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
{
    ret_code_t err_code = NRF_SUCCESS;


    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_ADV_SET_TERMINATED:                      // Advertising Stoping Event

            NRF_LOG_INFO("BLE_GAP_EVT_ADV_SET_TERMINATED");

            lble_adv_dis_handle();

            break;

        case BLE_GAP_EVT_DISCONNECTED:                            // Disconnection Event

            NRF_LOG_INFO("BLE_GAP_EVT_DISCONNECTED");

            lble_disc_handle();

            break;

        case BLE_GAP_EVT_CONNECTED:                               // Connection Completion Event

            NRF_LOG_INFO("BLE_GAP_EVT_CONNECTED");

            err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, g_lble_info.conn_handle);
            APP_ERROR_CHECK(err_code);

            g_lble_info.conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
            lble_conn_handle(p_ble_evt->evt.gap_evt.params.connected.role);

            if(g_lble_info.state == LBLE_STS_CONNCTED_C)
            {
                // start discovery of services. The NUS Client waits for a discovery result
                err_code = ble_db_discovery_start(&m_db_disc, p_ble_evt->evt.gap_evt.conn_handle);
                APP_ERROR_CHECK(err_code);
            }

            application_timers_ctrl(T_CTRL_ID_START_UART_CTRL_RX);

            break;

        case BLE_GAP_EVT_CONN_PARAM_UPDATE:                       // Connection Update Completion Event

            NRF_LOG_INFO("BLE_GAP_EVT_CONN_PARAM_UPDATE");

            lble_conn_update_comp(0x00);

            break;


        case BLE_GAP_EVT_CONN_PARAM_UPDATE_REQUEST:               // Connection Update Request Evevnt

            NRF_LOG_INFO("BLE_GAP_EVT_CONN_PARAM_UPDATE_REQUEST");

            lble_conn_update_req(
             (ble_gap_evt_conn_param_update_request_t *)&(p_ble_evt->evt.gap_evt.params.conn_param_update_request),
                                0x00 );

            break;
    (Omit)
```

5.2.9. Peer Manager

The following describes the software processing for the peer manager. The AT command application manages the pairing sequence and the bonding information by API and event handler of the peer manager. The Pairing sequence notifies the related event to BLE stack (SoftDevice) event handler.

(1) Initialization

Source Code: main.c

```
/**@brief Function for the Peer Manager initialization.
 */
static void peer_manager_init(void)
{
    ble_gap_sec_params_t sec_param;
    ret_code_t           err_code;

    err_code = pm_init();                                       // Peer Manager Initialization
    APP_ERROR_CHECK(err_code);

    // Security parameters to be used for all security procedures.
    lble_pair_para_set(&sec_param);

    err_code = pm_sec_params_set(&sec_param);                   // Pairing Parameter Setting
    APP_ERROR_CHECK(err_code);

    err_code = pm_register(pm_evt_handler);                     // Event Handler Registration
    APP_ERROR_CHECK(err_code);
}
```

(2) Pairing Request

Source Code: ble_handler.c

```
void lble_pair_req(void)
{
    ret_code_t err_code;
    ble_gap_sec_params_t sec_para;

    lble_pair_para_set(&sec_para);                              // Pairing Parameter Setting
                                                                // Pairing Request
    err_code = sd_ble_gap_authenticate( g_lble_info.conn_handle,
                                        &sec_para );
    if(err_code != NRF_ERROR_INVALID_STATE)
    {
        APP_ERROR_CHECK(err_code);
    }
}

void lble_pair_para_set(ble_gap_sec_params_t *sec_para)         // Pairing Parameter Setting for Passkey Entry
{
    memset( sec_para, 0x00, sizeof(ble_gap_sec_params_t) );

    sec_para -> bond        = 1;
    sec_para -> mitm        = 1;

    if(g_lble_info.state == LBLE_STS_CONNCTED_C)
    {
        sec_para -> io_caps = BLE_GAP_IO_CAPS_KEYBOARD_ONLY;
    }
    else
    {
        sec_para -> io_caps = BLE_GAP_IO_CAPS_DISPLAY_ONLY;
    }
    sec_para -> min_key_size    = 16;
    sec_para -> max_key_size    = 16;

    sec_para -> kdist_own.enc   = 1;
    sec_para -> kdist_peer.enc  = 1;
    sec_para -> kdist_peer.id   = 1;
}
```

(3) Event Handler

Source Code: main.c

```
                                                    // BLE Stack (SoftDevice) Event Handler
static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
{
    ret_code_t err_code = NRF_SUCCESS;

    switch (p_ble_evt->header.evt_id)
    {

    (省略)

        case BLE_GAP_EVT_AUTH_STATUS:                   // Pairing Completion Event

            NRF_LOG_INFO("BLE_GAP_EVT_AUTH_STATUS");

                                                        // Pairing Result Branch (Success)
            if((p_ble_evt->evt.gap_evt.params.auth_status.auth_status) == BLE_GAP_SEC_STATUS_SUCCESS)
            {
                lble_sec_success();
            }
            else                                        // Pairing Result Branch (Failure)
            {
                lble_sec_failure();
            }

            break;

        case BLE_GAP_EVT_PASSKEY_DISPLAY:               // Passkey Display Indication  Event

            NRF_LOG_INFO("BLE_GAP_EVT_PASSKEY_DISPLAY");

            app_pk_disp((ble_gap_evt_passkey_display_t *)&(p_ble_evt->evt.gap_evt.params.passkey_display));

            break;

        case BLE_GAP_EVT_AUTH_KEY_REQUEST:              // Passkey Entry Indication Event

            NRF_LOG_INFO("BLE_GAP_EVT_AUTH_KEY_REQUEST");

            app_pe_ind();

            break;

    }
}
```

Source Code: ble_handler.c

```
static void pm_evt_handler(pm_evt_t const * p_evt)              // Peer Manager Event Handler
{
    ret_code_t err_code;

    pm_handler_on_pm_evt(p_evt);
    pm_handler_flash_clean(p_evt);

    switch (p_evt->evt_id)
    {

    (省略)

        case PM_EVT_CONN_SEC_SUCCEEDED:                         // Re-connection Success Event after Pairing

            NRF_LOG_INFO("PM_EVT_CONN_SEC_SUCCEEDED");

            if(p_evt -> params.conn_sec_succeeded.procedure == PM_CONN_SEC_PROCEDURE_ENCRYPTION)
            {
                lble_sec_success();
            }

            break;

        case PM_EVT_CONN_SEC_FAILED:                            // Pairing Failed Event

            NRF_LOG_INFO("PM_EVT_CONN_SEC_FAILED");

            lble_sec_failure();

            break;

        default:

            NRF_LOG_INFO("PM_EVT_ETC");

            break;
    }
}
```

5.2.10. Generic Access Profile (GAP)

The following describes the software processing for GAP. The AT command application manages the device address and the device name by API and event handler of GAP.

(1) Initialization

Source Code: main.c

```
static void gap_params_init(void)
{
    ret_code_t              err_code;
    uint8_t     *dev_name_p;
    ble_gap_conn_sec_mode_t sec_mode;

    fds_find_token_t    tok  = {0};

    ble_gap_addr_t          p_addr;

     (省略)

#if (LC_GAP_ADDR_MODE == 2)
    p_addr.addr_type = BLE_GAP_ADDR_TYPE_RANDOM_STATIC;          // Address Type Setting

    memcpy(&p_addr.addr[0], &bd_addr[0], 6);                     // Static Random Addrss Setting

    err_code = sd_ble_gap_addr_set(&p_addr);                     // Devicde Address Settign
    APP_ERROR_CHECK(err_code);
#endif

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
                                                                 // Read Device Name from Flash ROM
    err_code = fds_record_find(CONFIG_FILE, CONFIG_REC_KEY_DEV_NAME, &desc, &tok);
    if(err_code == NRF_SUCCESS)                                  // Valid Device Name in Flash Flash Branch (Valid)
    {
        fds_record_t fds_record_dev_name_work = {0};

        err_code = fds_record_open(&desc, &fds_record_dev_name_work);
        APP_ERROR_CHECK(err_code);

        dev_name_p = (uint8_t *)fds_record_dev_name_work.data.p_data;
                                                                 // Device Name Setting from Flash ROM
        err_code = sd_ble_gap_device_name_set(  &sec_mode,
                                                fds_record_dev_name_work.data.p_data + 1,
                                                (uint16_t)*dev_name_p );

        err_code = fds_record_close(&desc);
        APP_ERROR_CHECK(err_code);
    }
    else                                                         // Valid Device Name in Flash Flash Branch (Invalid)
    {
                                                                 // Default Devicce Name Setting
        err_code = sd_ble_gap_device_name_set(  &sec_mode,
                                                (const uint8_t *)LC_GAP_DEVICE_NAME,
                                                strlen(LC_GAP_DEVICE_NAME) );
    }
    APP_ERROR_CHECK(err_code);
}
```

5.2.11. Generic Attribute Profile (GATT)

The following describes the software processing for GATT.

(1) Initialization

Source Code: main.c

```
static void gatt_init(void)
{
    ret_code_t err_code;


    err_code = nrf_ble_gatt_init(&m_gatt, gatt_evt_handler);     // GATT Initialization & Event Handler Registration
    APP_ERROR_CHECK(err_code);
                                                                 // ATT MTU Size Setting
    err_code = nrf_ble_gatt_att_mtu_central_set(&m_gatt, NRF_SDH_BLE_GATT_MAX_MTU_SIZE);
    APP_ERROR_CHECK(err_code);

}
```

(2) Event Handler

Source Code: main.c

```
void gatt_evt_handler(nrf_ble_gatt_t * p_gatt, nrf_ble_gatt_evt_t const * p_evt)
{
    if (p_evt->evt_id == NRF_BLE_GATT_EVT_ATT_MTU_UPDATED)       // ATT MTU Size Updating Event
    {
        NRF_LOG_INFO("ATT MTU exchange completed.");
                                                                 // VSSPP Max. Data length Updating
        m_ble_nus_max_data_len = p_evt->params.att_mtu_effective - OPCODE_LENGTH - HANDLE_LENGTH;
        NRF_LOG_INFO("Ble NUS max data length set to 0x%X(%d)", m_ble_nus_max_data_len, m_ble_nus_max_data_len);
    }
}
```

5.2.12. Device Information Service

The following describes the software processing for the device information service. The AT command application is set to be able to read device composition information from the remote device.

(1) Initialization

Source Code: main.c

```
static void services_init(void)
{
    ret_code_t         err_code;
    nrf_ble_qwr_init_t qwr_init = {0};

    ble_dis_init_t                  dis_init;
    ble_dis_pnp_id_t                pnp_id;
    ble_dis_reg_cert_data_list_t    reg_cert_data_list;
    ble_dis_sys_id_t                sys_id;
    uint32_t                        reg_cert_data;

    (省略)

    /* --- Device Information Service --- */
    memset(&dis_init, 0, sizeof(dis_init));
                                                            // Manufacture Name Setting
    ble_srv_ascii_to_utf8(&dis_init.manufact_name_str, LC_DIS_MANUFACTURER_NAME);
                                                            // Model Number Setting
    ble_srv_ascii_to_utf8(&dis_init.model_num_str, LC_DIS_MODEL_NUM);
                                                            // Serial Number Setting
    ble_srv_ascii_to_utf8(&dis_init.serial_num_str, LC_DIS_SERIAL_NUM);

                                                            // System ID Setting
    sys_id.manufacturer_id            = LC_DIS_MANUFACTURER_ID;
    sys_id.organizationally_unique_id = LC_DIS_ORG_UNIQUE_ID;
    dis_init.p_sys_id                 = &sys_id;

                                                            // Hardware Revision Setting
    ble_srv_ascii_to_utf8(&dis_init.hw_rev_str, LC_DIS_HW_REV);
                                                            // Firmware Revision Setting
    ble_srv_ascii_to_utf8(&dis_init.fw_rev_str, LC_DIS_FW_REV);
                                                            // Software Revision Setting
    ble_srv_ascii_to_utf8(&dis_init.sw_rev_str, LC_DIS_SW_REV);

                                                            // Regulatory Certification Data List Setting
    reg_cert_data                   = LC_DIS_REG_CERT_DATA_LIST;
    reg_cert_data_list.p_list       = (uint8_t *)&reg_cert_data;
    reg_cert_data_list.list_len     = sizeof(LC_DIS_REG_CERT_DATA_LIST);
    dis_init.p_reg_cert_data_list   = &reg_cert_data_list;

                                                            // PnP ID Setting
    pnp_id.vendor_id_source         = LC_DIS_VENDOR_ID_SOURCE;
    pnp_id.vendor_id                = LC_DIS_VENDOR_ID;
    pnp_id.product_id               = LC_DIS_PRODUCT_ID;
    pnp_id.product_version          = LC_DIS_PRODUCT_VERSION;
    dis_init.p_pnp_id               = &pnp_id;

    dis_init.dis_char_rd_sec = SEC_OPEN;

    err_code = ble_dis_init(&dis_init);                     // Device Infromation Setting
    APP_ERROR_CHECK(err_code);

}
```

5.2.13. Battery Service

The following describes the software processing for the battery service. The AT command application notifies the battery level by the request from the remote device. The battery level is updated by the application timer. Refer to "5.2.6 Application Timer" for the timer processing of the battery level.

(1) Initialization

Source Code: main.c

```
static void services_init(void)
{
    ret_code_t         err_code;
    nrf_ble_qwr_init_t qwr_init = {0};

    ble_bas_init_t                  bas_init;

    (省略)

    /* --- Battery Service --- */
    memset(&bas_init, 0, sizeof(bas_init));

                                                            // Security Setting
                                                            bas_init.bl_rd_sec        = SEC_OPEN;
    bas_init.bl_cccd_wr_sec   = SEC_OPEN;
    bas_init.bl_report_rd_sec = SEC_OPEN;

#if (LC_BAS_NTF_INTERVAL != 0)
                                                            // Event Handler Registration for Notification Enabled
    bas_init.evt_handler          = bas_evt_handler;
                                                            // Notification Enabled Setting
    bas_init.support_notification = true;
#endif /* (LC_BAS_NTF_INTERVAL != 0) */
                                                            // Battry Level Setting
    bas_init.initial_batt_level   = 100;

                                                            // Battry Service Initialization
    err_code = ble_bas_init(&m_bas, &bas_init);
    APP_ERROR_CHECK(err_code);
}
```

(2) CCCD Management *Notification Enabled

Source Code: main.c

```
#if (LC_BAS_NTF_INTERVAL != 0)
                                                            // Battry Service Event Handler
static void bas_evt_handler(ble_bas_t * p_bas, ble_bas_evt_t *evt)
{
    uint8_t time_type;

                                                            // Event Branch (CCCD:Notify Enavled)
    if(evt -> evt_type == BLE_BAS_EVT_NOTIFICATION_ENABLED)
    {
        battery_level_update();

        time_type = (uint8_t)T_CTRL_ID_START_BAS_NTF;
    }
    else                                                    // Event Branch (CCCD:Notify Disabled)
    {
        time_type = (uint8_t)T_CTRL_ID_START_BAS_UPDATE;
    }

    application_timers_ctrl(time_type);                     // Timer Control for Battry Service

}
#endif /* (LC_BAS_NTF_INTERVAL != 0) */
```

5.2.14. LAPIS Vendor Specific Service

The following describes the software processing for the LAPIS vendor specific service. The AT command application provides the data communication function by the Vendor Specific Serial Port Profile (VSSPP) of the LAPIS vendor specific service.

(1) Initialization

Source Code: main.c

```
static void services_init(void)
{
    ret_code_t         err_code;
    nrf_ble_qwr_init_t qwr_init = {0};

    ble_nus_init_t                  vsspp_init;
    ble_nus_c_init_t                init;

    (省略)

    /* --- Vendor Specific Serial Port Profile --- */
    memset(&vsspp_init, 0, sizeof(vsspp_init));

    vsspp_init.data_handler = nus_data_handler;                 // Event Handler Registration (Peripheral)

    err_code = ble_nus_init(&m_vsspp_p, &vsspp_init);           // VSSPP Initialization (Peripheral)
    APP_ERROR_CHECK(err_code);

    init.evt_handler   = ble_nus_c_evt_handler;                 // Event Handler Registration (Central)
    init.error_handler = nus_error_handler;                     // Error Handler Registration (Central)
    init.p_gatt_queue  = &m_ble_gatt_queue;                     // GATT Queue Area Assignment (Central)

    err_code = ble_nus_c_init(&m_ble_nus_c, &init);             // VSSPP Initialization (Central)
    APP_ERROR_CHECK(err_code);
}
```

(2) TX Data

Source Code: main.c

```
        do
        {
            if(g_lble_info.state == LBLE_STS_CONNCTED_C)        // Communication Stat Branch (Connection for Central)
            {
                                                                // Data Tx Request for Central
                err_code = ble_nus_c_string_send(&m_ble_nus_c, data_array, uart_rx_idx);
            }
            else                                                // Communication Stat Branch (Connection for Peripheral)
            {
                                                                // Data TX Request for Peripheral
                err_code = ble_nus_data_send(&m_vsspp_p, data_array, &uart_rx_idx, g_lble_info.conn_handle);
            }

            if ((err_code != NRF_ERROR_INVALID_STATE) &&
                (err_code != NRF_ERROR_RESOURCES) &&
                (err_code != NRF_ERROR_NOT_FOUND) &&
                (err_code != NRF_ERROR_BUSY) )
            {
              APP_ERROR_CHECK(err_code);
            }
        } while (err_code == NRF_ERROR_RESOURCES);
```

(3) RX Data

Source Code: main.c

```
                                                        // VSSPP Event Handler (Central)
static void ble_nus_c_evt_handler(ble_nus_c_t * p_ble_nus_c, ble_nus_c_evt_t const * p_ble_nus_evt)
{
    ret_code_t err_code;

    switch (p_ble_nus_evt->evt_type)                    // Event Branch
    {
        case BLE_NUS_C_EVT_DISCOVERY_COMPLETE:
            NRF_LOG_INFO("Discovery complete.");
            err_code = ble_nus_c_handles_assign(p_ble_nus_c, p_ble_nus_evt->conn_handle, &p_ble_nus_evt->handles);
            APP_ERROR_CHECK(err_code);

            err_code = ble_nus_c_tx_notif_enable(p_ble_nus_c);
            APP_ERROR_CHECK(err_code);
            NRF_LOG_INFO("Connected to device with Nordic UART Service.");
            break;

        case BLE_NUS_C_EVT_NUS_TX_EVT:                  // Event Branch (RX Data Event)
                                                        // UART Output
            uart_tx_req(0, p_ble_nus_evt->p_data, p_ble_nus_evt->data_len);

            break;

        case BLE_NUS_C_EVT_DISCONNECTED:
            NRF_LOG_INFO("Disconnected.");
            break;
    }
}

                                                        // VSSPP Event Handler (Peripheral)
static void nus_data_handler(ble_nus_evt_t * p_evt)
{

    if (p_evt->type == BLE_NUS_EVT_RX_DATA)             // Evant Branch (RX Data Event)
    {
        uint32_t err_code;

        NRF_LOG_DEBUG("Received data from BLE NUS. Writing data on UART.");
        NRF_LOG_HEXDUMP_DEBUG(p_evt->params.rx_data.p_data, p_evt->params.rx_data.length);

        !ble_conn_update_req(NULL, 0x00);

                                                        // UART Output
        uart_tx_req(0, (uint8_t *)&(p_evt->params.rx_data.p_data[0]), (uint16_t)(p_evt->params.rx_data.length) );
    }
}
```

Revision History

Document No.	Issue date	Page		Remarks
		Before revision	After revision	
FEXK715x1_AT_UsersManual-01	Nov. 25, 2020	-	-	Final edition 1st