RL78/G22

Wi-Fi Communication (Soft AP mode) with DA16200/DA16600

Introduction

The station (STA) mode of the Wi-Fi device is used in connection with the access point (AP) mode. The station (STA) mode of the Wi-Fi device is used in connection with the access point (AP) mode. Therefore, devices that operate mainly in station (STA) mode, such as smartphones, are generally used by connecting to the access point (AP) of a Wi-Fi router. By operating a Wi-Fi device in access point (AP) mode, user can direct connect without using a Wi-Fi router. As a result, it is possible to communicate one-on-one with TCP clients such as smartphones without the need for a Wi-Fi router, and it can be used to control devices from smartphones and notify to smartphones of device status. With this function, it can be applied to " Wi-Fi Garage Door Control ".

This application note describes how to use the RL78/G22 and DA16200 / DA16600 for Wi-Fi communication in AP mode. DA16200 / DA16600 for STA mode and AP mode. It is a module capable of Wi-Fi communication, connected to the host MCU via UART communication, and controlled by AT commands. The RL78/G22 acts as a host MCU for DA16200 / DA16600. In the sample program, the host MCU RL78/G22 controls the DA16200/DA16600 and perform TCP communication. In addition, a program is implemented to send AT commands from the RL78/G22 to the DA16200 / DA16600 using the AT command management framework. By using the AT command management framework, it is possible to implement applications that take advantage of various communication protocols supported by DA16200 / DA16600 Wi-Fi communication functions. This application note provides a detailed explanation of the Wi-Fi communication application and the AT command management framework implemented in this sample program.

Target Device

RL78/G22

DA16200 / DA16600

Related Documents

- RL78/G22 User's Manual: Hardware (R01UH0978)
- RL78/G22 Fast Prototyping Board User's Manual (R20UT5121)
- RL78/G22 Fast Prototyping Board Quick Start Guide (R20UT5123)
- DA16200 / DA16600 Host Interface and AT Command User Manual (UW-WI-0003)
- DA16200 / DA16600 SDK Update Guide(R12AN0129)

Pmod™ is registered to Digilent Inc.

Contents

1. Overview

1.1 Operation overview

The DA16200 / DA16600 is a module with Wi-Fi communication function support. Wi-Fi communication function can be controlled by AT commands via the UART from RL78/G22.

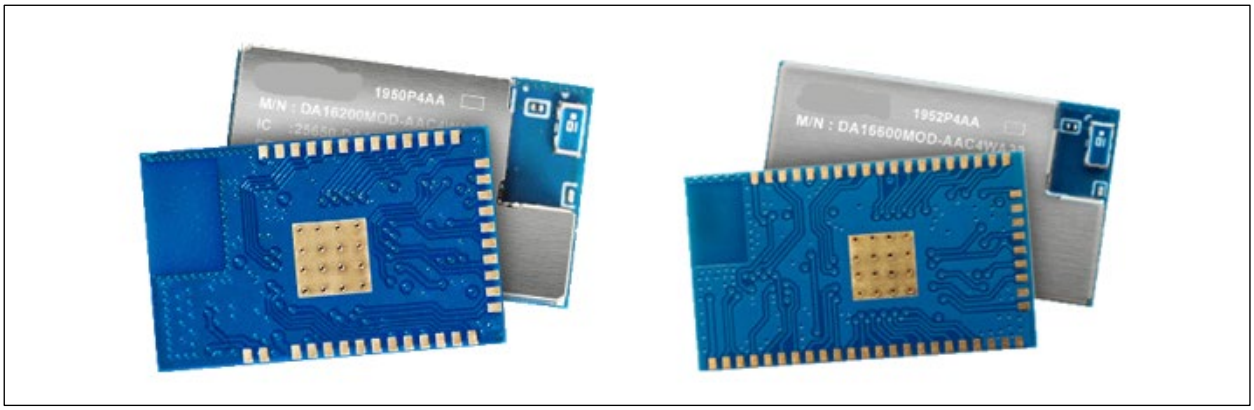

Figure 1-1 DA16200 / DA16600

In this sample program, AT command framework software, which controls Wi-Fi communication of DA16200 / DA16600 using the RL78/G22 as the host MCU. The RL78/G22 sends AT command as string data to DA16200 / DA16600 via UART communication. The response string data for the AT command is also received by UART communication. Through these exchanges, the RL78/G22 utilizes the Wi-Fi communication function of the DA16200 / DA16600.

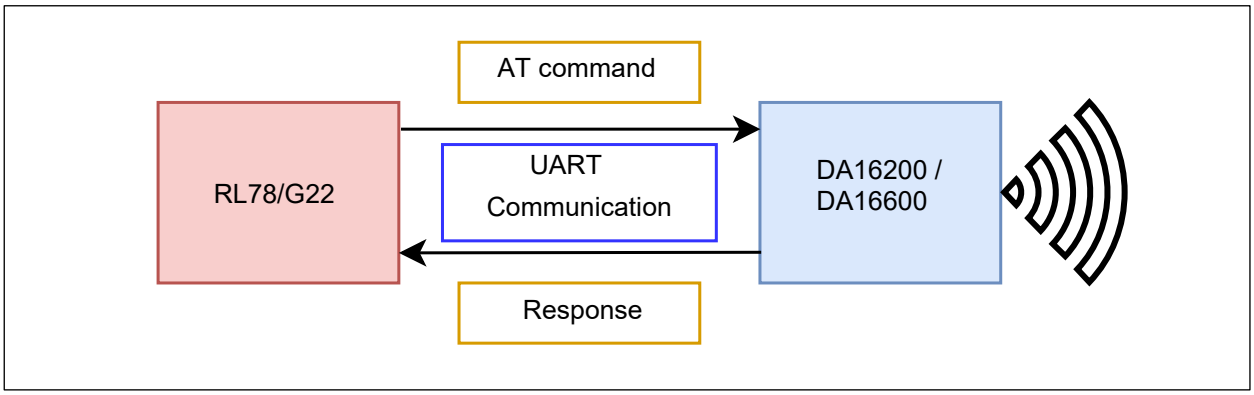

Figure 1-2 Communication between RL78/G22 and DA16200 / DA16600

1.2 Description of the Software

This sample program initializes it to operate as a TCP Server in Wi-Fi Soft AP mode. After connecting in STA mode of Wi-Fi such as a smartphone and connecting from TCP Client, if TCP client send “on” with ASCII data, the LED of RL78/G22 Fast Prototyping Board (RL78/G22 FPB) will turn on, and then with ASCII data Sending “off” turns off the RL78/G22 FPB LED. Send the string to the TCP Client by pressing the switch (SW) on the RL78/G22 FPB.

1.2.1 Sample program Configuration

[Table 1-1 Contents of this Application Note] shows the configuration of this sample program.

Table 1-1 Contents of this Application Note

File Name	Description
r01an7287xxrrrr--wifi-sample.pdf xx: Language classification, Creation country rrrr: Revision number	This Document
sample_rl78g22_DA16200 / DA16600	Sample Program for Soft AP Mode Operation

1.2.2 Hierarchy of sample programs

Software configuration of this sample framework is shown below.

Smart Configurator is an automatic code generation tool, and the three green functions use a program generated by Smart Configurator.

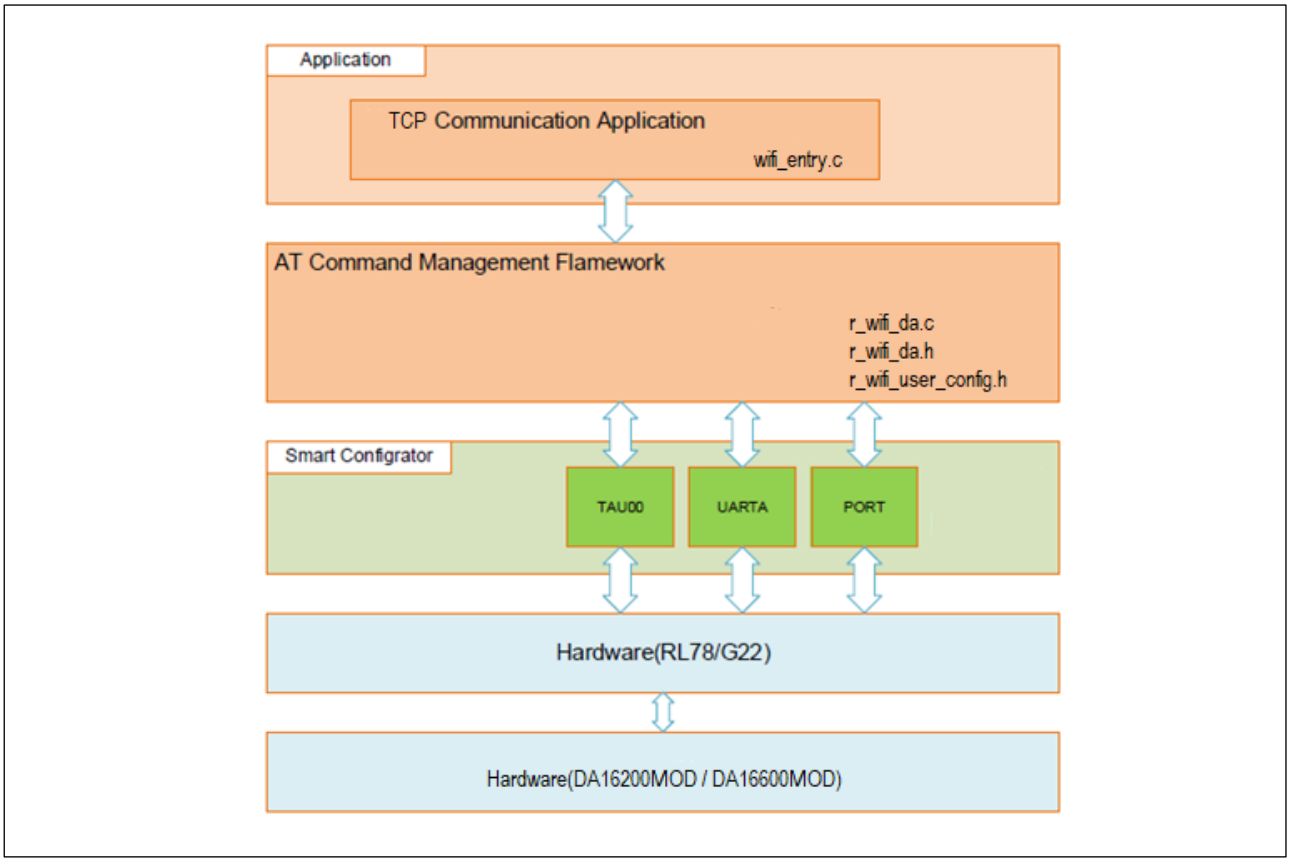

Figure 1-3 Software Configuration

The sample program is a program that performs TCP communication using the DA16200 / DA16600 module. The application is implemented on the host MCU side. This program consists of two features, TCP communication application, which calls APIs for the TCP communication and the AT command framework, which sends the AT command to the DA16200 / DA16600.

The TCP communication application is a program that turns LED on and off after receiving data from a TCP client. TCP communication applications are created using the AT command management framework's URC reception. For a detailed description about the TCP communication application, refer to [2 TCP Communication Application].

The AT Command Management Framework is a framework for implementing the transmission of AT commands to the DA16200 / DA16600 and the processing of responses received from the DA16200 / DA16600. By calling the API function implemented on the AT Command Management Framework, multiple AT commands are sent to the DA16200 / DA16600, and application is notified by a callback function.

In this sample program, a framework-based program is implemented using a framework so that the RL78/G22 can perform TCP communication through the DA16200 / DA16600. The AT Command Management Framework is intended to be used as a base for application development when using functions of the DA16200 / DA16600 other than TCP communication. For a detailed description about AT Command Management Framework, refer to [3 AT Command Management Framework].

1.2.3 Used peripheral function

[Table 1-2 Peripheral Functions to be used and their purpose] shows the peripheral functions used in this sample program.

Table 1-2 Peripheral Functions to be used and their purpose

Peripheral Function	Purpose
TAU00	AT command communication timeout
UARTA(P72/TxDA0)	UART TX
UARTA(P71/RxDA0)	UART RX
PORT(P70)	RTS signal for UART (Unused)
PORT(P50)	CTS signal for UART (Unused)
PORT(P17)	DA16200 / DA16600 RESET control (Unused)
SW(P137/INTP0)	User switch external pin interrupt
INTP(P51/INTP2)	DA16200 / DA16600 INT signal external pin interrupt (Unused)
UART(P12/TxD0)	UART TX for monitor log output

1.2.4 List of Option Byte Settings

[Table 1-3 Option Byte Settings] shows the option byte settings.

Table 1-3 Option Byte Settings

Address	Setting Value	Contents
000C0H/020C0H	11101111B	Watchdog time counter operation disable (Counting stopped after reset)
000C1H/020C1H	11111100B	LVD0 detection voltage: Reset mode At rising edge TYP. 2.67 V (2.59 V to 2.75 V) At falling edge TYP. 2.62 V (2.54 V to 2.70 V)
000C2H/020C2H	11101000B	HS mode, High-speed on-chip oscillator clock (f_{IH}): 32 MHz
000C3H/020C3H	10000100B	Enables on-chip debugging

1.2.5 Folder/File structure

[Table 1-4 Sample Program file structure] shows the file structure of the sample program provided in this application note.

Table 1-4 Sample Program file structure

Folder name, File name	Description
r01an7287_rl78g22_wifi	Program storage folder
├ src	Source files
│ ├ da16	DA16200 / DA16600 related source files
│ ├ smc_gen	Smart configurator generation
│ │ ├ Config_INTC	
│ │ ├ Config_PORT	
│ │ ├ Config_TAU0_1	
│ │ ├ Config_UART0	
│ │ ├ Config_UARTA0	
│ │ ├ general	
│ │ ├ r_bsp	
│ │ ├ r_config	
│ │ ├ r_pincfg	
│ ├ wifi_entry.c	Main program of the sample program
│ ├ wifi_entry.h	
│ ├ main.c	
│ ├ main.h	

1.2.6 Code size

[Table 1-4 ROM/RAM Sizes] shows ROM/RAM Sizes of the RL78/G22's sample program provided in this application note.

Table 1-5 ROM/RAM Sizes

Resource	Size
ROM	17,898 Byte
RAM	1,390 Byte

2. TCP Communication Application

2.1 Application environment

This section describes the environment to operate the TCP communication application.
This application operates in the following hardware environment.

Table 2-1 Hardware environment

Hardware	Description
RL78/G22 Fast Prototyping Board (RL78/G22 FPB)	Evaluation board with RL78/G22 (RTK7RLG220C00000BJ)
US159-DA16200EVZ DA16200 Wi-Fi Pmod™ board *	PMOD board with DA16200 module (US159-DA16200MEVZ)
US159-DA16600EVZ Wi-Fi + Bluetooth® Low Energy Combo Pmod™ Board *	PMOD board with DA16600 module (US159-DA16600EVZ)
Windows PC	RL78/G22's application development environment and debug console for operation conformation

* DA16200 / DA16600 software uses DA16200 DA16600 FreeRTOS SDK Image v3.2.8.1.

This sample program has been developed and checked with the following software environment.

Table 2-2 Software environment of sample program

Item	Description
Integrated development environment (e2 studio)	Made by Renesas Electronics Corporation e2 studio V2024-01 (24.1.0)
C compiler (e2 studio)	Made by Renesas Electronics Corporation CC-RL V1.13.00
Integrated development environment (CS+)	Made by Renesas Electronics Corporation CS+ for CC V8.11
C compiler (CS+)	Made by Renesas Electronics Corporation CC-RL V1.13.00
Smart Configurator	Made by Renesas Electronics Corporation V1.9.0
Board support package (BSP)	Made by Renesas Electronics Corporation V1.62
Renesas Flash Programmer (RFP)	Made by Renesas Electronics Corporation V3.14.00

For application operation, follow these steps:

1. Update the firmware on the US159-DA16200EVZ / US159-DA16600EVZ to the latest version just to be sure.
   For the update method, refer to DA16200/DA16600 SDK Update Guide (renesas.com).
   The operation is checked with DA16200 DA16600 FreeRTOS SDK Image v3.2.8.1.

2. Connect RL78/G22 FPB and US159-DA16200EVZ / US159-DA16600EVZ with PMOD connector.
   Please use PMOD2 connector for RL78/G22 FPB.

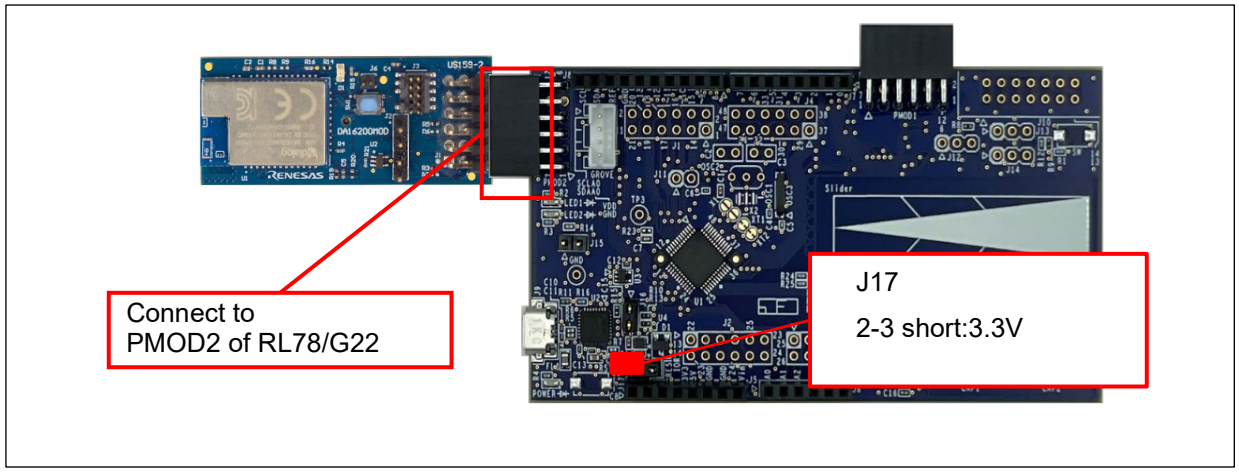

Figure 2-1 Connect RL78/G22 FPB and DA16200 / DA16600

3. Connect USB cables to RL78/G22 FPB.

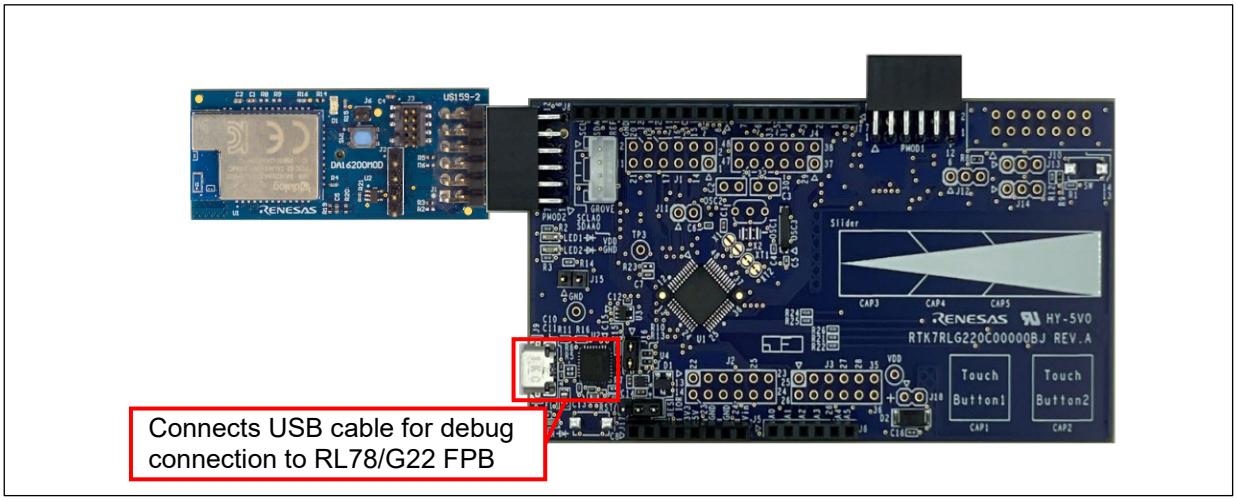

Figure 2-2 Connect USB cable

4. Import sample project

The sample program is provided in the project format of e² studio. This chapter shows how to import the project to e² studio and CS+.

- Importing a Project into e² studio

  To use sample programs in e² studio, follow the steps below to import them into e² studio. In projects managed by e² studio, do not use space codes, multibyte characters, and symbols such as "$", "#", "%" in folder names or paths to them.

  (Note that depending on the version of e² studio you are using, the interface may appear somewhat different from the screenshots below.)

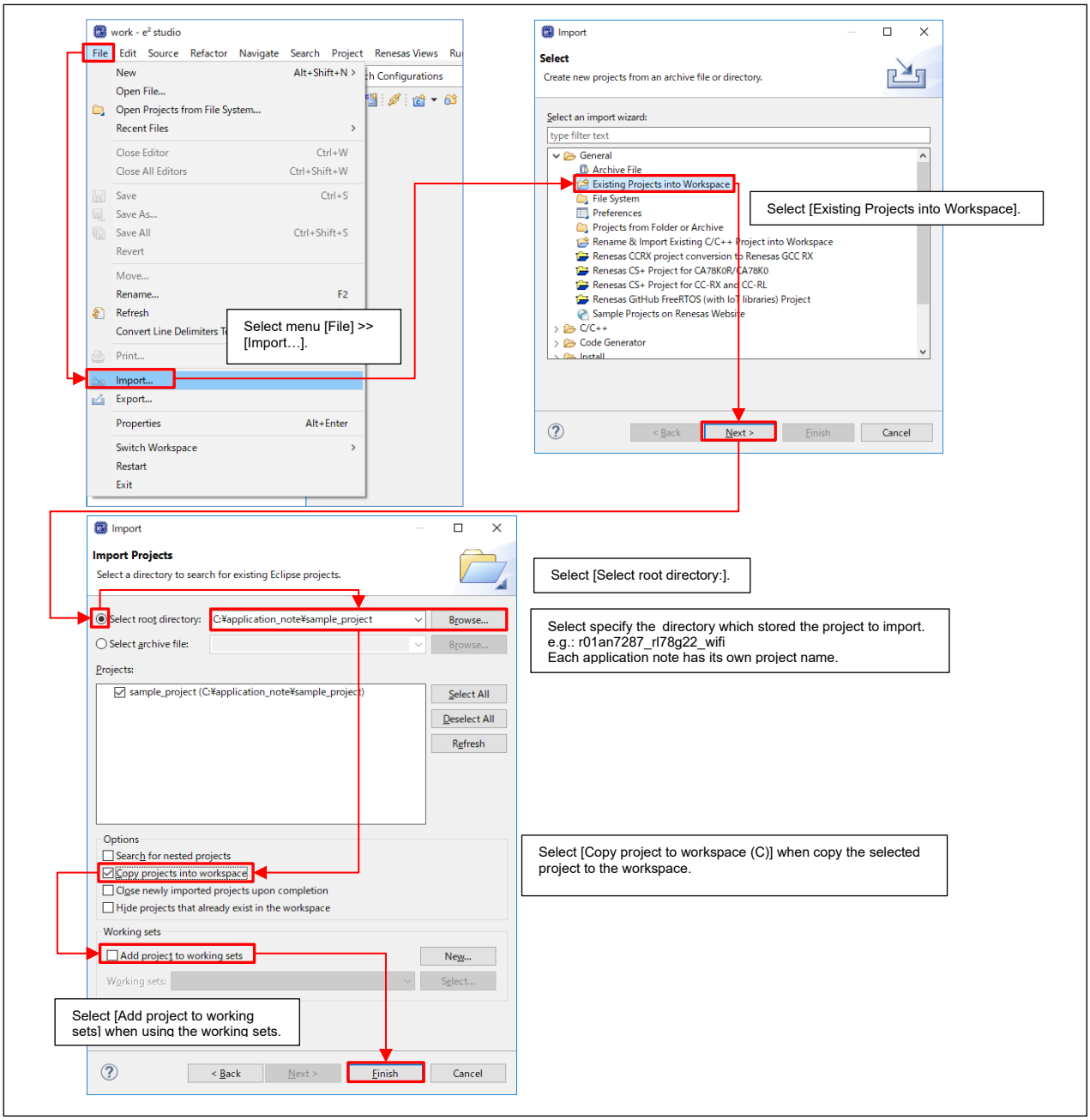

Figure 2-3 **Importing a Project into e² studio**

- Importing a Project into CS+

  To use sample programs in CS+, follow the steps below to import them into CS+. In projects managed by CS +, do not use space codes, multibyte characters, and symbols such as "$", "#", "%" in folder names or paths to them.

  (Note that depending on the version of CS+ you are using, the interface may appear somewhat different from the screenshots below.)

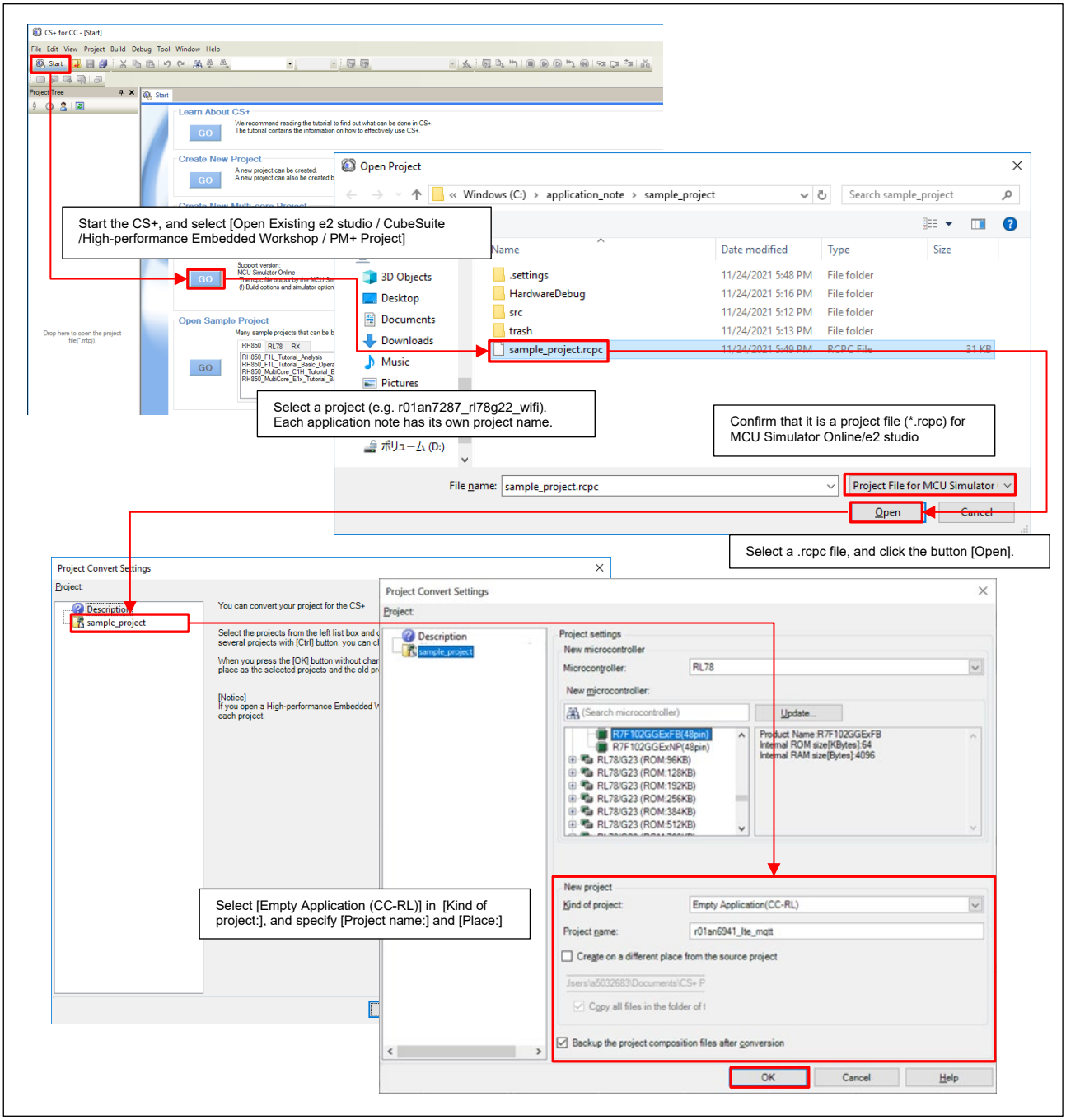

Figure 2-4 Importing a Project into CS+

5. Change the SSID, password, country name, security protocol, and cipher type. The IP address and subnet mask used to connect to the Wi-Fi network. Start IP and end IP of the DHCP server configuration range. And the local port of the TCP server is specified. Change these values to match your application. The files to be modified are as follows.

   - wifi_entry.c

   Refer to the following documentation for detail on the settings.

   UM-WI-003 DA16200 DA16600 Host Interface and AT Command User Manual (renesas.com)

```
50
51          /* Application specific string data for AP */
52          static uint8_t s_str_ap_ssid[]     = "test1234";
53          static uint8_t s_str_ap_password[] = "12345678";
54
55          /* Country code (optional). If exists, code is essential */
56          static uint8_t s_str_ap_country[]  = "JP";
57
58          /* Operating channel (optional). Default is 1 or uses the current channel if Soft AP is operating */
59          static uint8_t s_str_ap_ch[]       = "0";
60
61          /* Security protocol. 0 (OPEN), 2 (WPA), 3 (WPA2), 4 (WPA+WPA2) ),
62             5 (WPA3 OWE), 6 (WPA3 SAE), 7 (WPA2 RSN & WPA3 SAE) */
63          static uint8_t s_str_ap_sec[]      = "3";
64
65          /* Encryption. 0 (TKIP), 1 (AES), 2 (TKIP+AES) */
66          static uint8_t s_str_ap_enc[]      = "1";
67
68          /* IP Address Setting */
69          static uint8_t s_str_ip_addr[]     = "10.0.0.1";
70          static uint8_t s_str_ip_netmask[]  = "255.255.255.0";
71
72          /*Network DHCP Server setting */
73          static uint8_t s_str_dhcp_staip[]   = "10.0.0.2";
74          static uint8_t s_str_dhcp_endip[]   = "10.0.0.11";
75
76          /* TCP server socket setting */
77          static uint8_t s_str_local_port[]   = "10194";
78
```

Figure 2-5 Setting of SSID, password, etc. (wifi_entry.c)

Based on this setting, the following AT command is executed.

```
ATF
+INIT:DONE,0
AT+TMRFNOINIT=0
OK
AT+WFMODE=1
OK
AT+WFSAP=test1234,3,1,12345678,0,JP
+WFSAP:test1234
OK
AT+RESTART
OK
+INIT:DONE,1
(The received URC is different between DA16200 and DA16600.)
AT+NWIP=1,10.0.0.1,255.255.255.0,10.0.0.1
OK
AT+NWDHS=1,10.0.0.2,10.0.0.11,1800
OK
AT+TRTS=10194
+TRTS:0
OK
AT+TRTRM=0
OK
AT+TRTS=10194
+TRTS:0
OK
```

Figure 2-6 Command Execution Content (Soft AP Mode Setting)

6. The DA16200 and DA16600 of the Wi-Fi module have different initialization times after reset or restart, and different wait times can be set. If you set the DA16600 to DA16600 due to the long initialization time, the DA16200 and DA16600 can operate with the same program. By default, the sample program is set to DA16600.The files to be modified are as follows.

   ・da16\r_wifi_da.c

```
46  /*****************************************************************************
47   * Macro definition
48   *****************************************************************************/
49  /* 1: DA16600 setting, 0: DA16200 (Adjust period)  */
50  /* Setting due to different initialization times of DA16200 and DA16600 */
51  #define DA16600        (1)
52
```

Figure 2-7 Select of DA16200 / DA16600

7. Set Motorola S-record file output.

   (1): Right mouse click on the project.

   (2): Select property.

   (3): Select Settings → Converter Output, check "Output hex file",

   confirm that Output Motorola S-record file is selected, click Apply and Close.

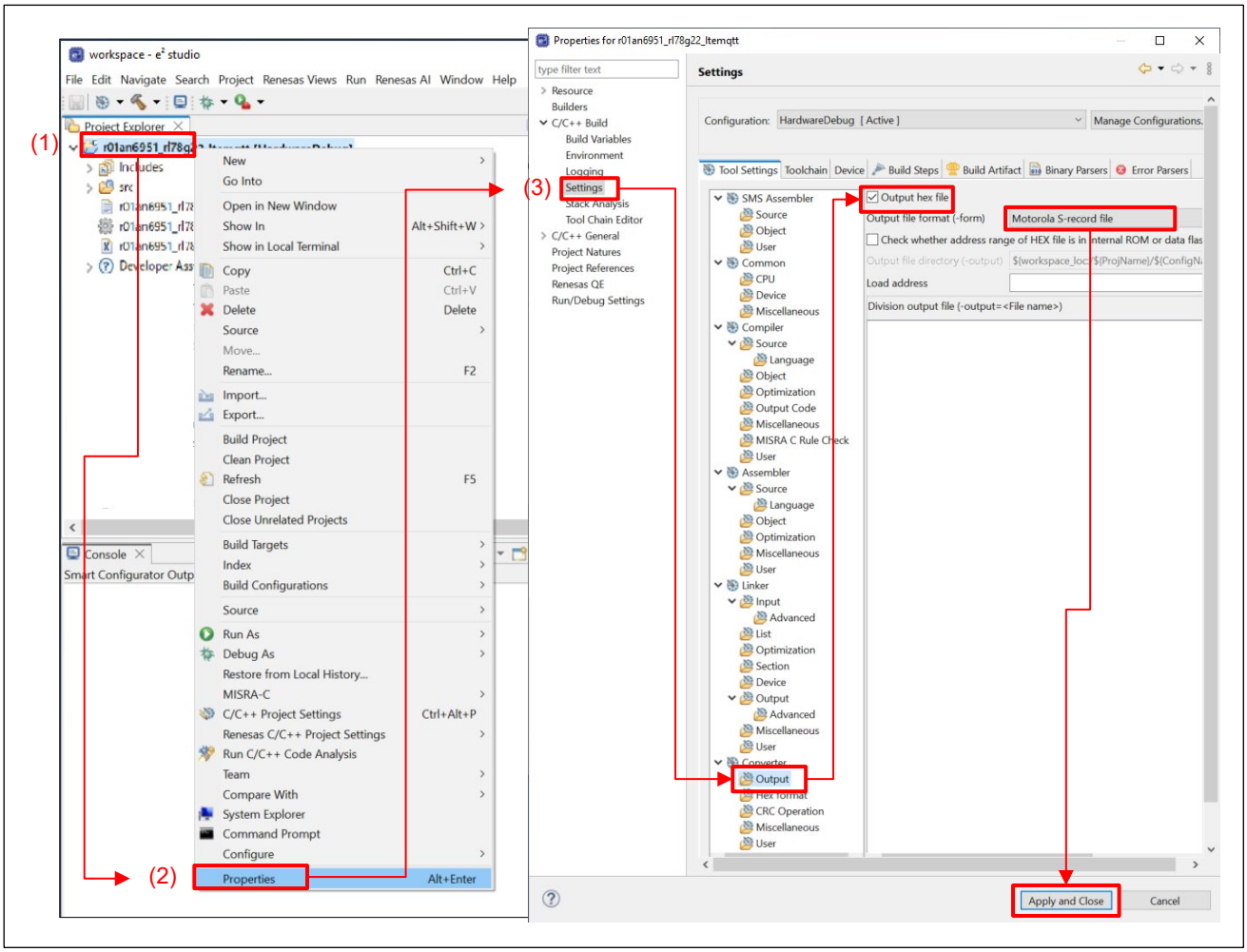

Figure 2-8 Motorola S-record file output setting

8. Build the sample project. When you build, Motorola S-record file (.mot) will be output.

9. Program the Motorola S-record file to the RL78/G22 FPB using the Renesas Flash Programmer.

   (1): Select File → New Project…

   (2): Select RL78/G2x from the pull-down menu.

   (3): Specify an arbitrary name and arbitrary folder for the Project Name and Project folder.

   (4): Select COM port and 2 wire UART from the pull-down menu.

   (5): Click Tool Details…

   (6): Specify COM port of RL78/G22 FPB

   (7): Click OK

   (8): Click Connect (If the connection is successful, proceed to (9). In case of error, please check the settings before (6).

   (9): Specify Motorola S-record file generated by build.

   (10): Click Start to start programing.

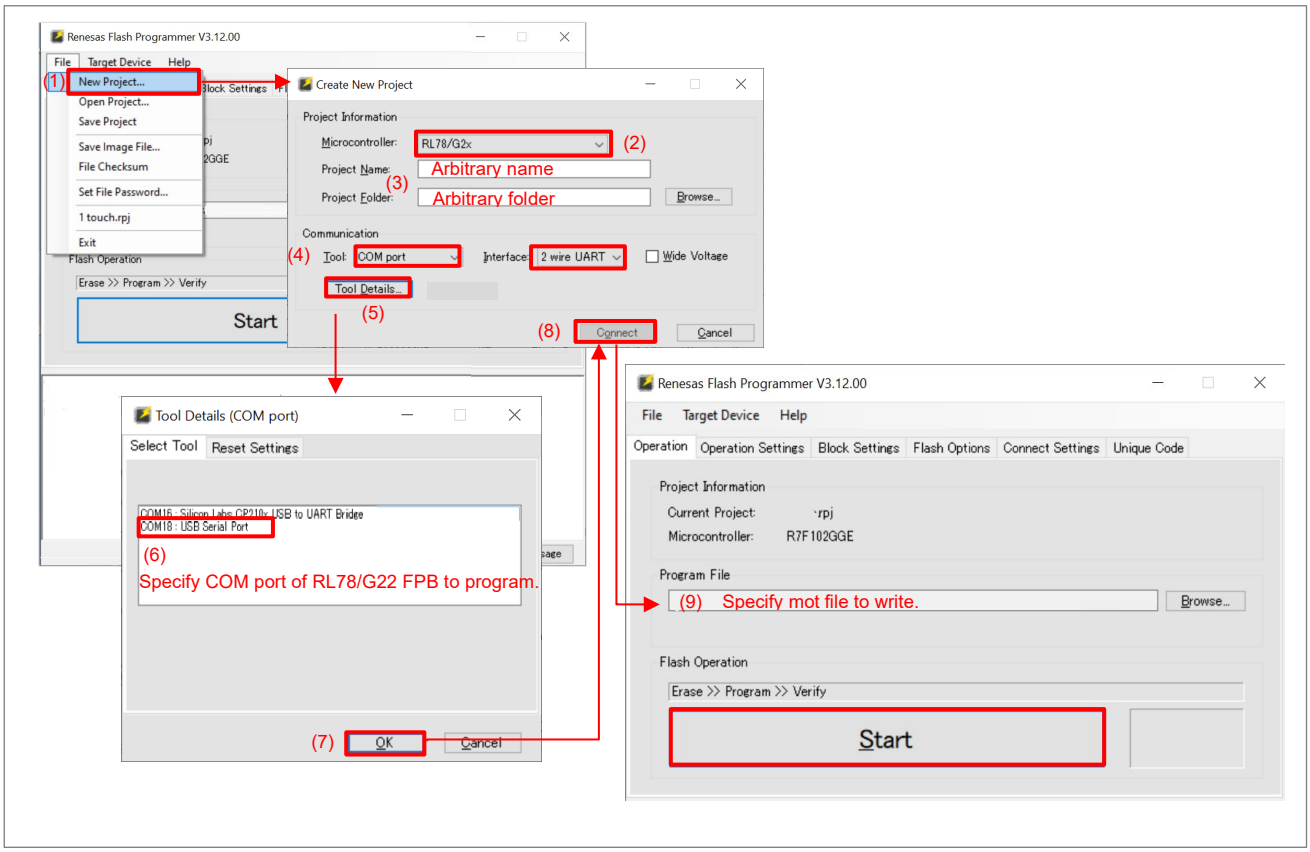

Figure 2-9 Programing file

10. Prepare for monitoring.

    The execution log of the sample program is output from USB of RL78/G22 FPB. Specify the COM port of RL78/G22 FPB with terminal software such as TeraTerm.

    Baud rate is 115200bps, data is 8bit, parity is none, stop bit is 1bit. The line feed code reception setting is “LF”.

2.2 Application operation

This sample program is initialized to operate as a TCP Server in Soft AP mode of Wi-Fi, and after connecting from a STA mode of Wi-Fi connect such as a smartphone and a TCP Client, it sends "on" in ASCII data to turn on the LED of the RL78/G22 FPB. Sending "off" in ASCII data turns off the LED on the RL78/G22 FPB. Also, by pressing the switch(SW) on the RL78/G22 FPB, a string of characters is sent to the smartphone.

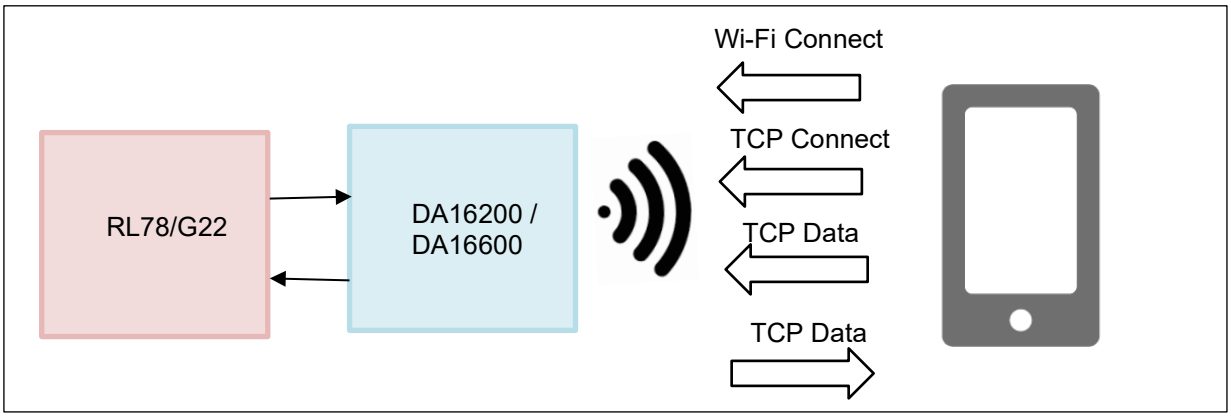

Figure 2-10 sample program system configuration.

When the sample program is executed, the RL78/G22 resets the DA16200 / DA16600 module. After resetting the DA16200 / DA16600, go to the Operation Modo settings and set the Soft-AP mode. Then restart the module. In the Network settings, configure the IP settings, DHCP settings, TCP server operation settings, and local port settings. The execution log of the terminal software is shown below.

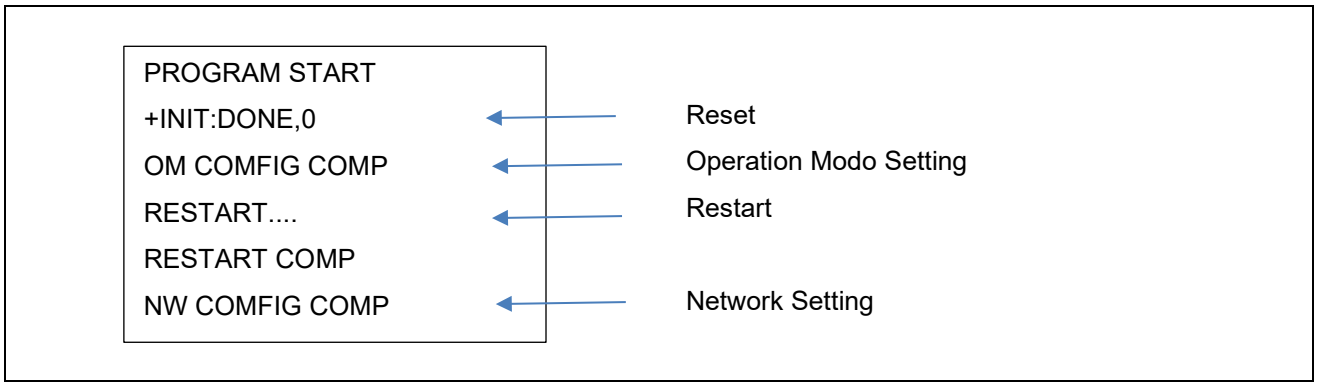

Figure 2-11 RL78/G22 FPB log display (terminal software) settings.

The following is an example of operation after setting the operation.

TCP Client uses TCP Client software such as TCP Client - Apps on Google Play. A smartphone example is explained inTCP Client - Apps on Google Play.

(1) Make a Wi-Fi connection from your smartphone. In the terminal software connected to the RL78/G22 FPB, "URC: +WFCST:XX:XX:XX:XX:XX" is displayed.

(2) Set the TCP Address to "10.0.0.1" and the Port to "10194" on the smartphone and make a TCP connection. In the terminal software connected to the RL78/G22 FPB, "TCP connected" is displayed. In addition, LED1 of the RL78/G22 FPB turns on during the TCP connection.

(3) When "on\n" is sent from the smartphone, "LED on" is displayed in the terminal software connected to the RL78/G22 FPB. It also turns on LED2 on the RL78/G22 FPB.

(4) When "off\n" is sent from the smartphone, "LED off" is displayed in the terminal software connected to the RL78/G22 FPB. It also turns off LED2 on the RL78/G22 FPB.

(5) When you press the switch (SW) of the RL78/G22 FPB, "Switch: X" (where X is increment) is sent from the RL78/G22 FPB. The smartphone is displayed the data.

(6) Make a TCP disconnect on your smartphone. When TCP is disconnected, "TCP disconnected" is displayed in the terminal software connected to the RL78/G22 FPB. In addition, LED1 of the RL78/G22 FPB is turned off during the TCP connection.

(7) Disconnect the Wi-Fi from smartphone. In the terminal software connected to the RL78/G22 FPB, "URC: +WFDST:XX:XX:XX:XX" is displayed.

The execution log of the terminal software of the operation example is shown below by the item number of the operation example.

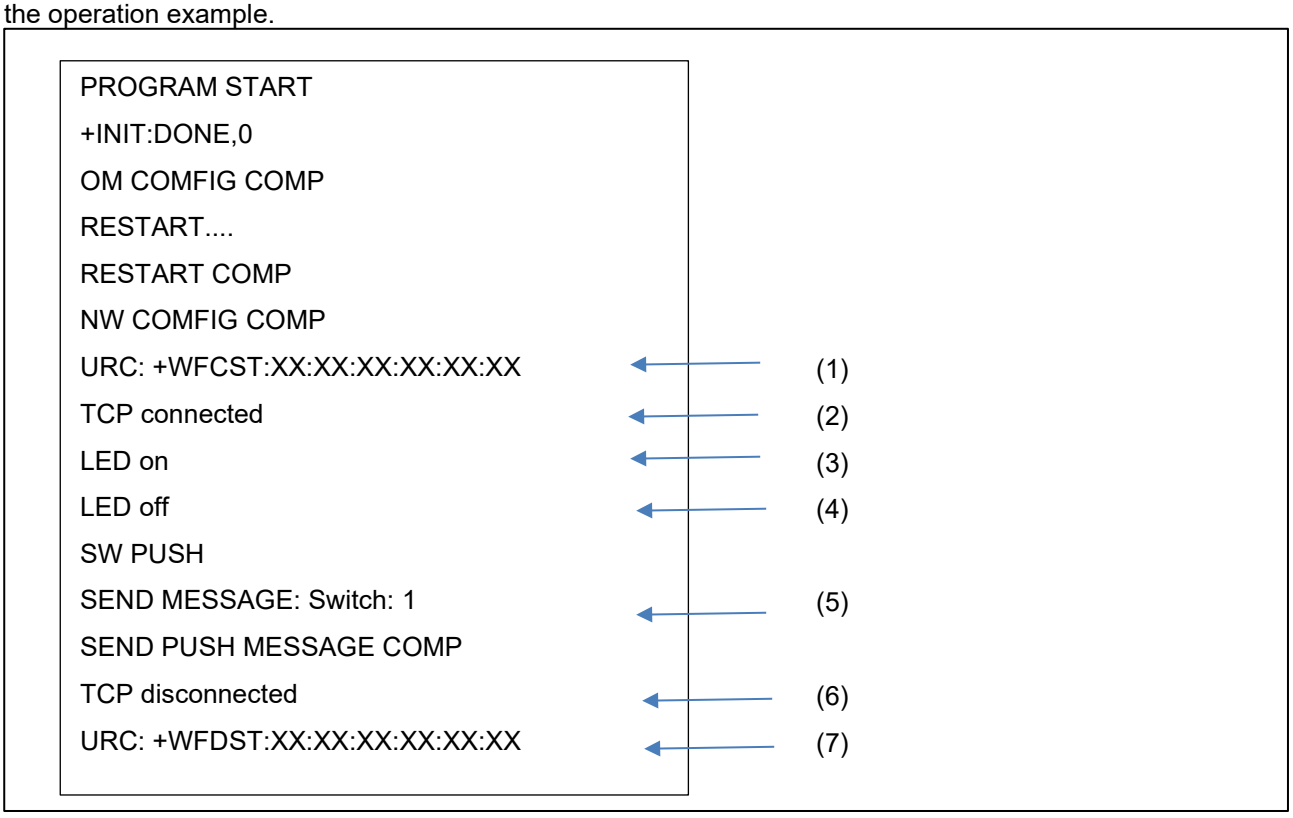

Figure 2-12 RL78/G22 FPB Log Display (Terminal Software) TCP operation.

The operation of the smartphone in the operation example is shown below by the item number of the operation example.

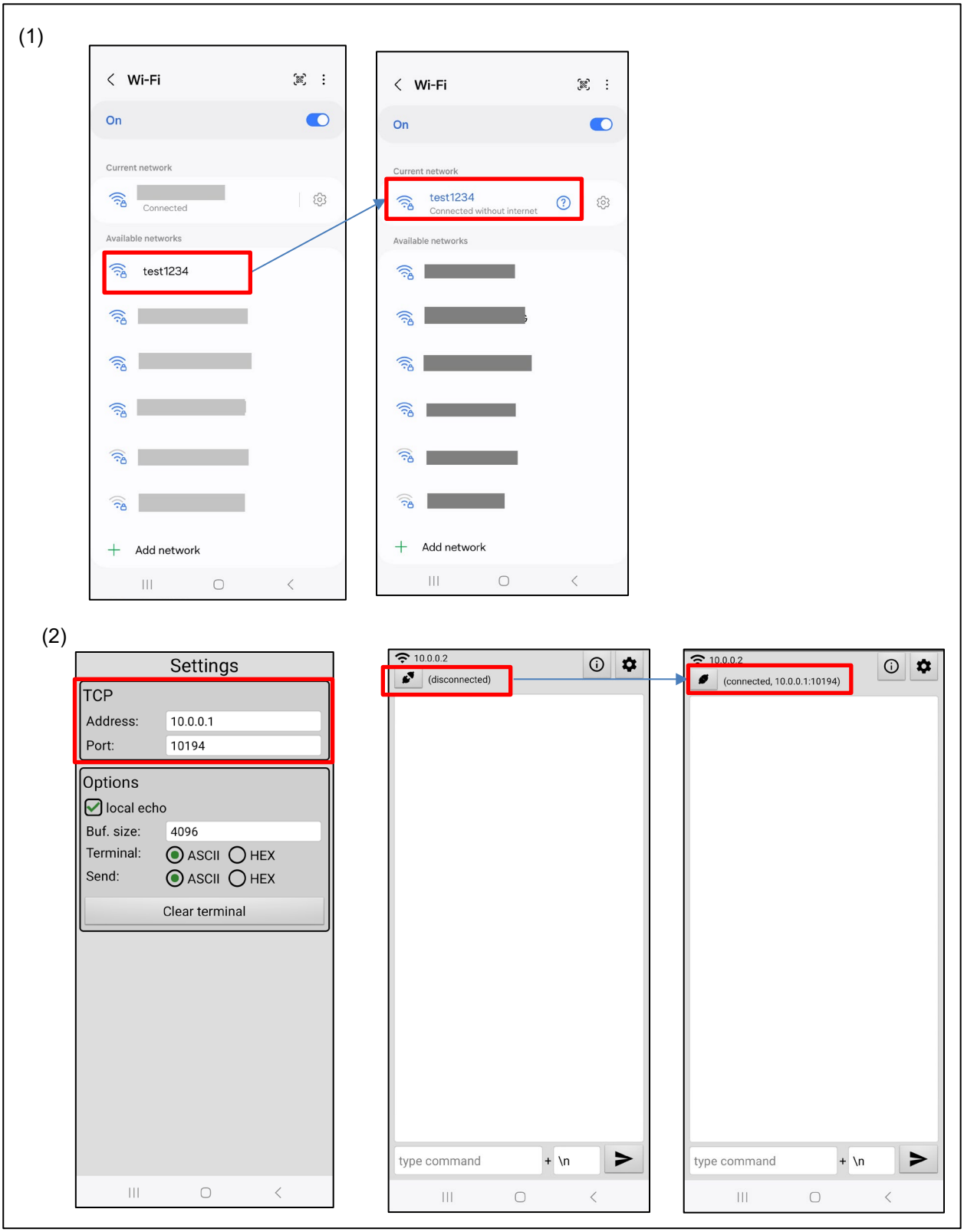

Figure 2-13 Smartphone Display (1)

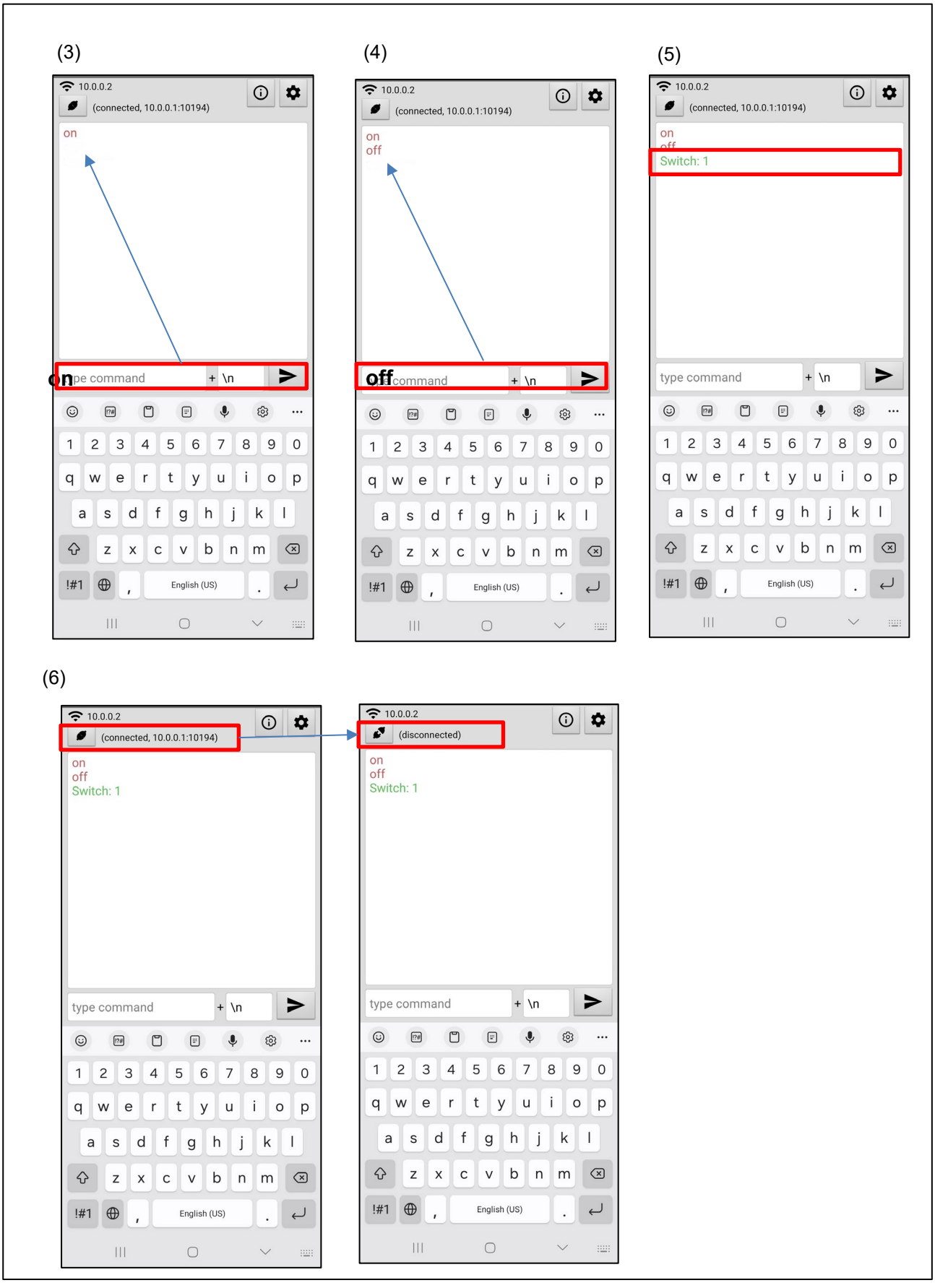

Figure 2-14 Smartphone Display (2)

The position of the RL78/G22 FPB LEDs and switch (SW) in the operation example is shown.

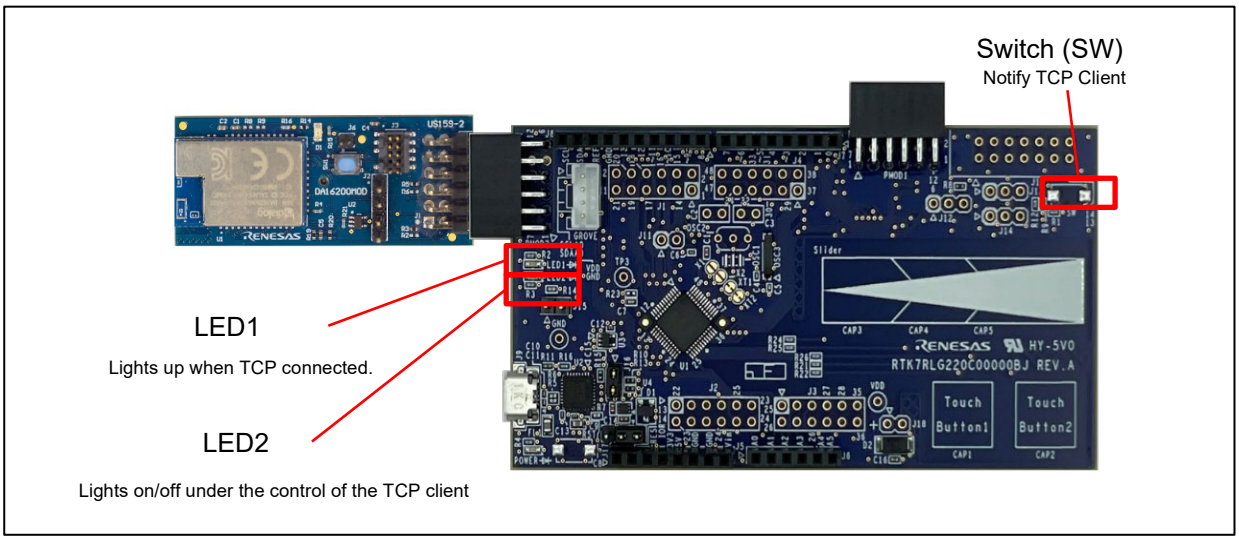

Figure 2-15 LED and switch of RL78/G22 FPB

3. AT Command Management Framework

3.1 Framework Overview

The DA16200 / DA16600 is operated from the RL78/G22 through the transmission of AT commands and reception of responses using serial communication through the UART. The AT Command Management Framework is a framework for efficiently implementing the transmission and reception of AT commands and responses. This sample program implements a framework-based program for TCP communication using the AT Command Management Framework.

The API implemented in the framework-based program of this sample program is classified into two types: management API and AT command API. The management API is an API for initializing framework-based programs and sending a series of AT commands in response to a response message. The AT Command API is an API for sending AT commands. The execution result of the AT command sent by the AT Command API is notified to the application as a callback function.

AT Command Management Framework is created using TAU, UARTA, PORT and interrupt controller generated by Smart Configurator.

- The UARTA is used to send AT commands to the DA16200 / DA16600 and receive responses from the DA16200 / DA16600.
- The TAU is used to measure timeout condition after AT command is executed.
- The interrupt controller is not used for DA16200 / DA16600 interrupt. However, it is used in an interrupt to signal that a switch (SW) has been pressed.

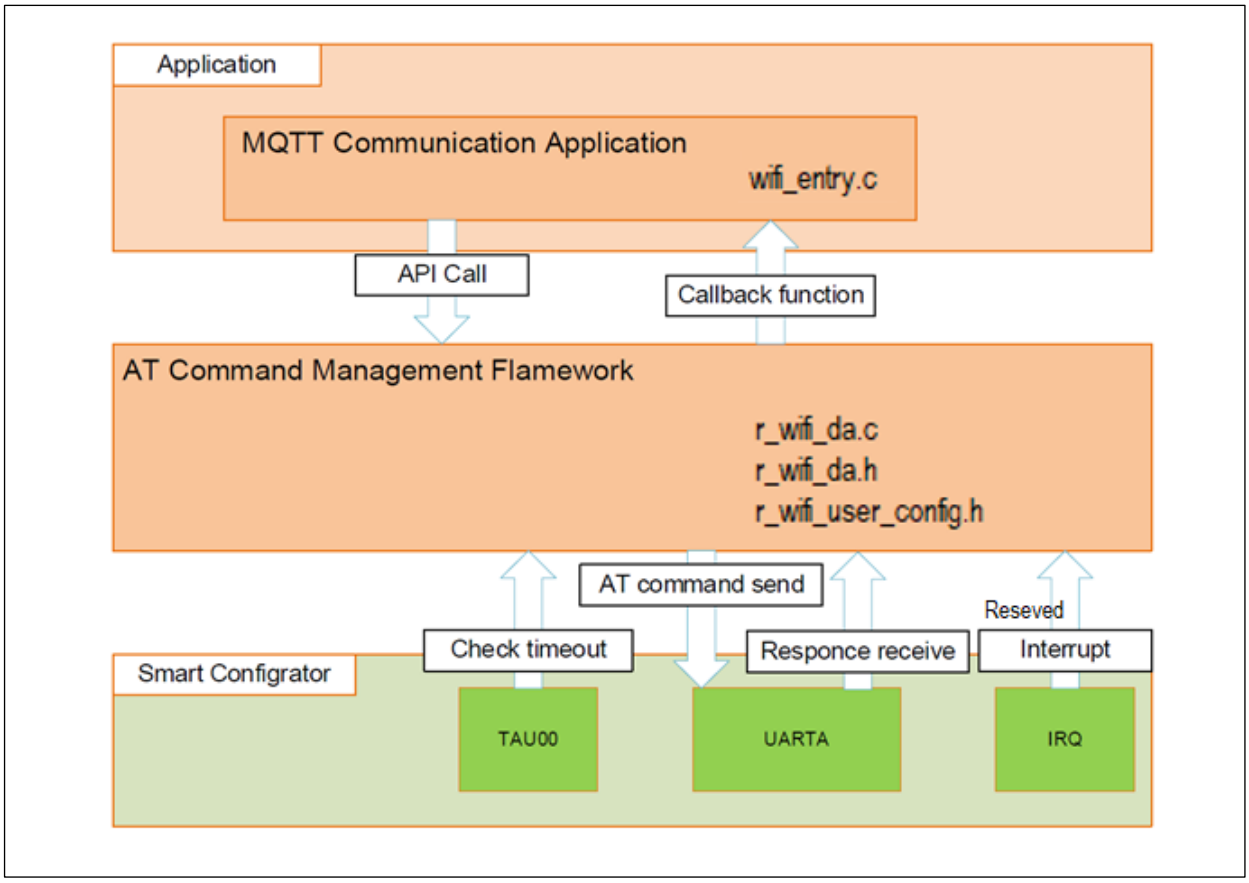

Figure 3-1 Overview of sample framework

The AT command framework implements an AT command API that sends AT commands to use the Wi-FI communication function of the DA16200 / DA16600. The series of AT commands required to perform the desired action by calling the AT Command API in the application are added to the transmit waiting list. AT commands added to the transmit waiting list are sent to DA16200 / DA16600 in order.

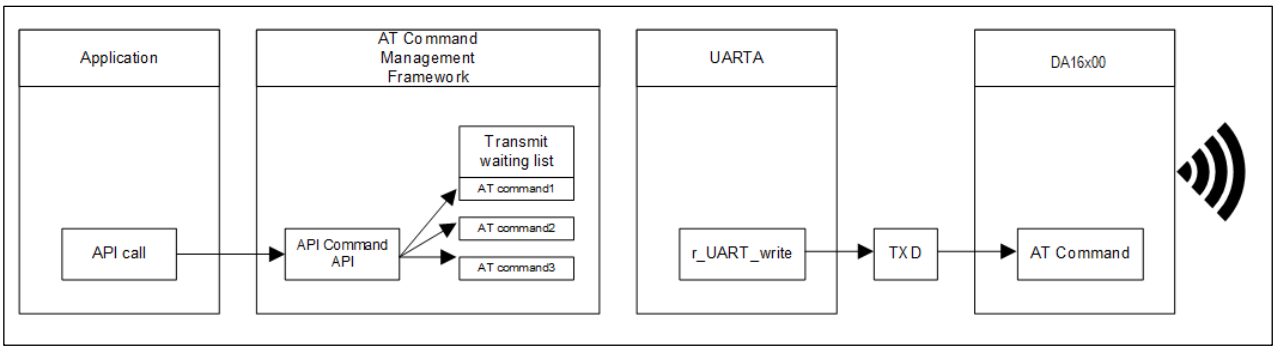

Figure 3-2 AT command API call

The result of executing the AT command on DA16200 / DA16600 is sent as a response. This response data is received by the UART module's callback function and analyzed by the R_WIFI_Execute function. If the execution result is correct, the R_WIFI_Execute function sends the next AT command that is added to the transmit waiting list. This procedure is repeated until all AT commands added to the transmit waiting list have been sent.

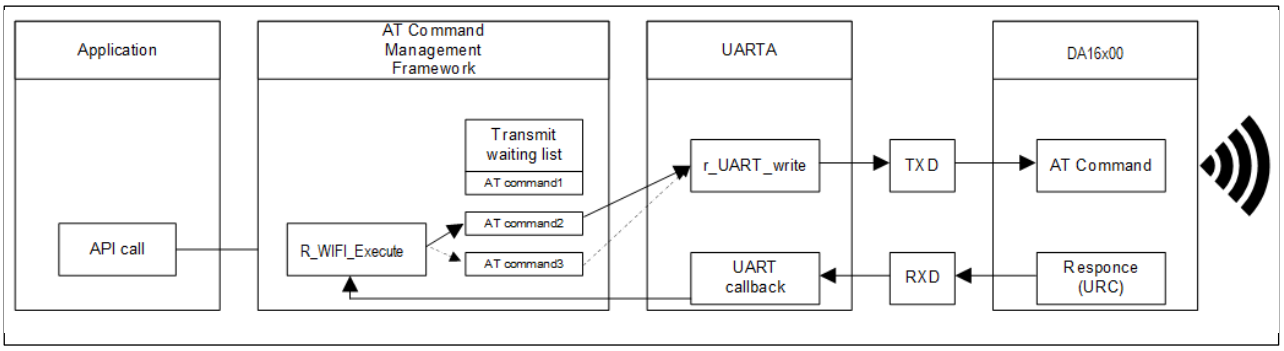

Figure 3-3 Receive response and AT command send

If all AT commands added to the send waiting list have been sent, the response from DA16200 / DA16600 is an error, or other data unrelated to the AT command is received, the R_WIFI_Execute function notifies the application as a callback function.

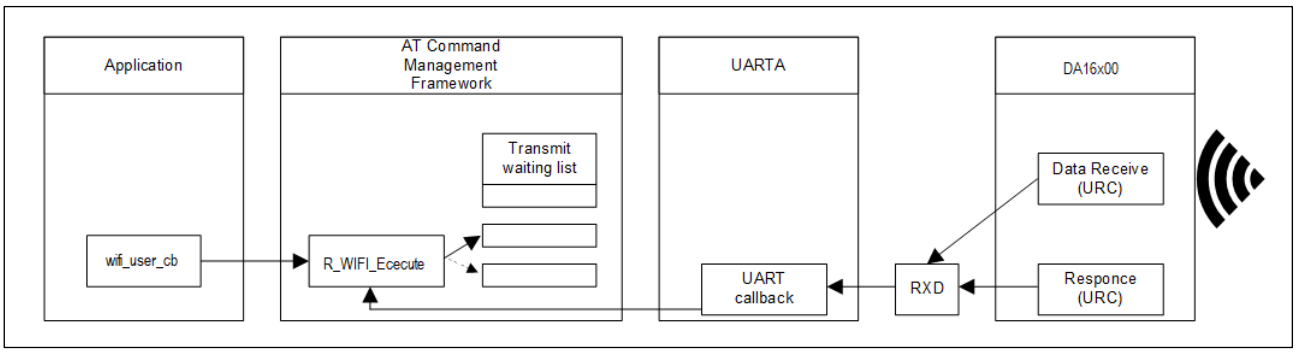

Figure 3-4 Notification in callback function

Executable API function provided from the framework-based program of this sample program is described in [3.2 API function].

The result of the API execution is notified to application through callback function. Details about callback function are described in [3.3 Callback function].

If user want to use the sample framework with other RL78 MCUs, the user only needs to change the “r_wifi_user_config.h” file. Configurable values are described in [3.4 User Specific Configuration].

3.2 API functions

The API functions implemented in the framework-based program of this sample program are classified into two types: Management API and AT command API. The management API is an API for initializing the framework-based program and sending a series of AT commands in response to the response message. The AT command API is an API to send AT commands. The management API is described in [3.2.1 Management API] and AT command API is described in [3.2.2 AT command API].

3.2.1 Management API

The management API is an API for initializing the framework-based program and sending a series of AT commands in response to the response message. It must be implemented in the main routine of the application. Even when adding functions based on the AT Command Management Framework, basically there is no need to change the management API program (r_wifi_da.c, r_wifi_da.h, r_wifi_user_config.c).

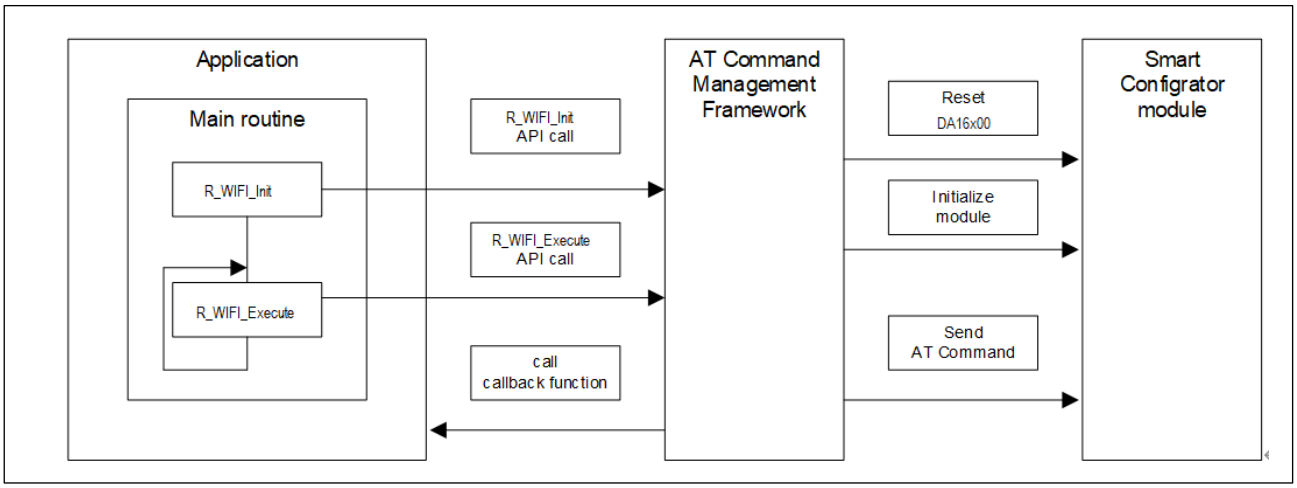

Figure 3-5 Management API

3.2.1.1 R_WIFI_Init

Function name	R_WIFI_Init	
Functional overview	Initialize framework-based program	
Argument	wifi_cb_t * p_callback_fun (IN)	Callback function to register For information about type “wifi_cb_t”, refer [3.3 Callback function]
Return value	WIFI_SUCCESS (0x0000)	API call success
	WIFI_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
Advanced description	Initialize DA16200 / DA16600 sample framework. As part of initialization, following operation are performed: • Initialization of Smart Configurator modules used in the framework. • Execute hardware reset of DA16200 / DA16600. • Registration of callback function to notify result of API to application. After this function is executed, AT command to reset DA16200 / DA16600 will be sent. Since the initialization time is different between DA16200 and DA16600, different weight times are set. The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. WIFI_API_INIT (0xFF) Please call this function before the main loop of your application.	

3.2.1.2 R_WIFI_Execute

Function name	R_WIFI_Execute	
Functional overview	Perform processing of the framework-based program.	
Argument	void	None
Return value	void	None
Advanced description	Execute various operations to be performed by the framework. The following operation are performed: • Send AT command specified by other API • Parse string data received from DA16200 / DA16600 • Notifies the application of completion of API operation or receipt of errors or others by calling callback function Please call this function repeatedly in the main loop of your application.	

3.2.2 AT command API

The AT command API is an API for sending AT commands. The AT commands are added to the transmit waiting list by calling the AT command API from the application. AT commands added to the transmit waiting lists are sent sequentially to DA16200 / DA16600 in response. When all AT commands specified in the AT command API have been sent, the AT command transmission result is notified to the application by callback function.

After calling the AT command API, the next AT command API cannot be called before the result is notified by the callback function. Also, the AT command API cannot be called from an interrupt handler. Call the AT command API only from main routine (including callback function of AT Command Management Framework).

The framework-based program of this sample program implements the API necessary for TCP communication with the DA16200 / DA16600. If the user wants to implement a function that uses AT commands that are not used in TCP communication applications, it is assumed that users will add a new AT Command API using this framework and develop an application.

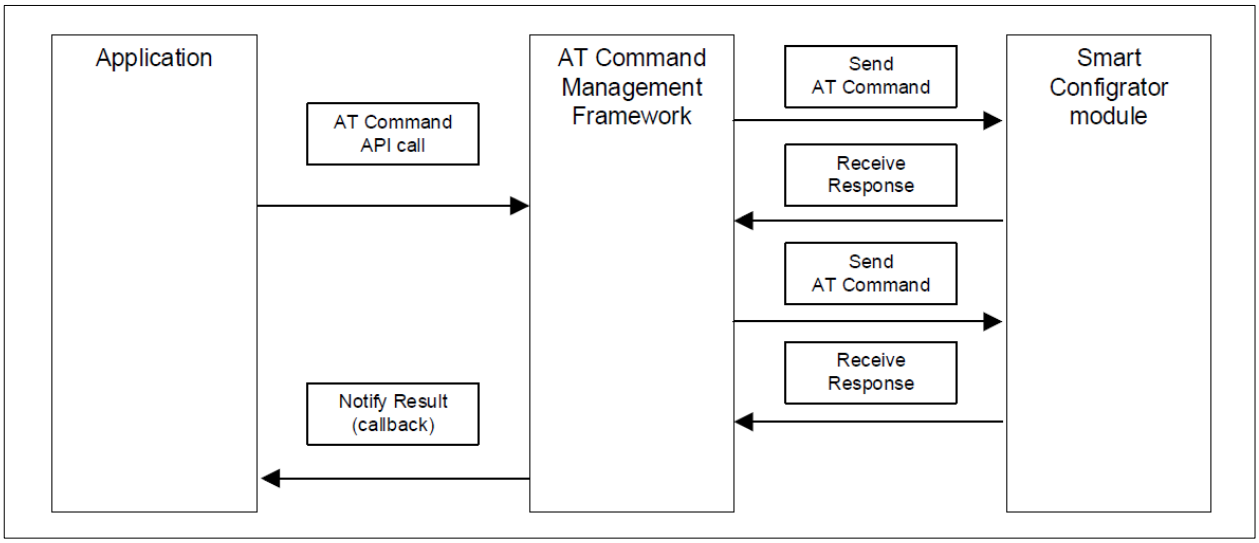

Figure 3-6 AT command API

3.2.2.1 R_WIFI_OM_Config

Function Name	R_WIFI_OM_Config	
Functional Overview	Configure operator mode	
Argument	uint8_t * p_ap_ssid (IN)	SSID of AP mode Eample: “test1234”
	uint8_t * p_ap_password (IN)	Password of AP mode Eample: “12345678”
	uint8_t * p_ap_country (IN)	Country name of AP mode Eample: “JP”
	uint8_t * p_ap_ch (IN)	Operating channel Eample: “0”
	uint8_t * p_ap_sec (IN)	Security protocol Eample: “3” (WPA2)
	uint8_t * p_ap_enc (IN)	Encryption mode Eample: “1” (AES)
Return value	WIFI_SUCCESS (0x0000)	API call success
	WIFI_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	WIFI_ERR_IN_PROCESS (0x0002)	Other API is in process
Advanced description	Sends the following AT commands in order: AT+TMRFNOINIT=0 AT+WFMODE=1 AT+WFSAP=[p_ap_ssid], [p_ap_sec], [p_ap_enc], [p_ap_password], [p_ap_ch], [p_ap_country] The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. WIFI_API_OM_CONFIG (0x01)	

3.2.2.2 R_WIFI_Restart

Function Name	R_WIFI_Restart	
Functional Overview	Restart the module	
Argument	void	none
Return value	WIFI_SUCCESS (0x0000)	API call success
Advanced description	Sends the following AT commands in order: AT+RESTART The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. WIFI_API_RESTART (0x02)	

3.2.2.3 R_WIFI_RestartWait

Function Name	R_WIFI_RestartWait	
Functional Overview	Wait for the module to resume after restarting	
Argument	void	none
Return value	WIFI_SUCCESS (0x0000)	API call success
Advanced description	Weighting the recovery from the AT+RESTART command Since the initialization time is different between DA16200 and DA16600, different weight times are set. The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. WIFI_API_RESTART_WAIT (0x03)	

3.2.2.4 R_WIFI_NW_Config

Function Name	R_WIFI_NW_Config	
Functional Overview	Configuring Network Settings	
Argument	uint8_t * p_ip_addr (IN)	IP Address Example: “10.0.0.1”
	uint8_t * p_ip_netmask (IN)	IP sab net mask Example: “255.255.255.0”
	uint8_t * p_dhcp_staip (IN)	DHCP start IP Example: “10.0.0.2”
	uint8_t * p_dhcp_endip (IN)	DHCP end IP Example: ”10.0.0.11”
	uint8_t * p_local_port (IN)	Local port Example: ”10194”
Return value	WIFI_SUCCESS (0x0000)	WIFI_SUCCESS (0x0000)
	WIFI_ERR_IN_PROCESS (0x0002)	WIFI_FAIL_IN_PROCESS (0x0002)
Advanced description	Sends the following AT commands in order: AT+NWIP=1, [p_ip_addr], [p_ip_netmask], [p_ip_addr] AT+NWDHS=1, [p_dhcp_staip], [p_dhcp_endip],1800 AT+TRTS=[p_local_port] AT+TRTRM=0 AT+TRTS=[p_local_port] The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. WIFI_API_NW_CONFIG (0x04)	

3.2.2.5 R_WIFI_PUSH_Message

Function Name	R_WIFI_PUSH_Message	
Functional Overview	Sending data to a TCP client	
Argument	uint8_t* p_tcp_socket	TCP socket to send data
	uint16_t length	Length of data
	uint8_t* p_message	Data
Return value	WIFI_SUCCESS (0x0000)	API call success
	WIFI_ERR_POINTER_NULL (0x0001)	Pointer of argument is NULL
	WIFI_ERR_IN_PROCESS (0x0002)	Other API is in process
	WIFI_ERR_DATASIZE_OVERFLOW (0x0003)	The data size of the argument exceeds the size that can be registered in transmit waiting list
Advanced description	Sends the following data in order: <ESC>S0[ASCII(length)],[p_tcp_socket],[p_message] The result of the AT command sent by this API is notified by the callback function. Following API_ID is used in callback function. WIFI_API_PUSH_MESSAGE (0x05)	

3.3 Callback function

When an AT command is sent to the DA16200 / DA16600, string data is received as a response. Also, DA16200 / DA16600 sends an Unsolicited Response Code (URC) that is not a result of an AT command. The framework-based program of this sample program receives string data from DA16200 / DA16600, and then parses the string data within the function R_WIFI_Execute. If information is needed to be notified to the user application, the function R_WIFI_Execute calls a callback function to notify the user application. This allows the application to check the execution result of the AT Command API and to check the URC of the DA16200 / DA16600. This section describes the structure of the callback function and the events and data that are signaled by the callback function.

The callback function has the following structure.

Type name	void * wifi_cb_t	
Argument	uint16_t event_type (In)	Notified event ID Refer IDs in Table 3-1.
	uint16_t api_id (In)	ID identifying API which framework-based program is processing. Refer IDs in Table 3-2.
	uint16_t data_len (in)	Data size of “p_data”
	void * p_data (out)	Notified event data. Value changes depending on notified event type

The event_type and api_id values use values defined in macro formats within framework-based programs. The values for each are shown below.

Table 3-1 Event Type IDs (event_type) and value

Macro	Value	Description
WIFI_EVENT_API_COMPLETE	0x0000	An event that notifies application that the operation specified in the API function has completed successfully. ”p_data” is set according to the called API.
WIFI_EVENT_ERROR	0x0001	An event that notifies application that an error has occurred in the behavior specified in the API function. Numeric data of error is set to “p_data”.
WIFI_EVENT_RCVURC	0x0002	An event that notifies application that a URC has been received. String data of URC is set to “p_data”.
WIFI_EVENT_TIMEOUT_ERROR	0x0003	An event that notifies application that timeout error has occurred for sending AT command and receiving a response. Timeout occurs when 60s has passed after sending an AT command.
WIFI_EVENT_FATAL_ERROR	0x0004	An event that is notified when a fatal error occurs. (Unused)

Table 3-2 API IDs (api_id) and value

Macro	Value	Corresponding API
WIFI_API_NO_CURRENT_API	0x0000	None
WIFI_API_OM_CONFIG	0x0001	R_WIFI_OM_Config
WIFI_API_RESTART	0x0002	R_WIFI_Restart
WIFI_API_RESTART_WAIT	0x0003	R_WIFI_RestartWait
WIFI_API_NW_CONFIG	0x0004	R_WIFI_NW_Config
WIFI_API_PUSH_MESSAGE	0x0005	R_WIFI_PUSH_Message
WIFI_API_INIT	0x00FF	R_WIFI_Init

The callback function is called from R_WIFI_Execute function in certain situations. The following is a list of when the callback function is called and the data to be set.

The source code containing the callback function is shown below.

Program: wifi_entry.c

- When all AT commands specified by the AT Command API are sent and responses are received without error:
  - Value “WIFI_EVENT_API_COMPLETE” is set to “event_type”.
  - In “p_data”, the data is set according to the AT command to be executed.
    - When URC is received as a response to AT command, string data of received URC is registered. The size of the string data to be notified is set to "data_len"
    - When calling the AT command API that starts data receive operation such, the received string data is registered. If the received data size exceeds "WIFI_DATA_STR_SIZE", the excess data is discarded and the data of the first half is registered. The size of the string data to be notified is set to "data_len".
    - Otherwise, no data is set in “p_data”. "data_len" is set to 0.

```
void wifi_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    if(WIFI_EVENT_API_COMPLETE == event_type)
    {
        switch (api_id)
        {
            case WIFI_API_INIT:
            {
                /* Configure operation mode afte
                if (0 != data_len)
                {
                    /* URC after ATF */
                    sprintf((char *)s_sbuf, (char *)p_data);
                }
                else
                {
/* Omission */
                }
                debug_printf(s_sbuf);
                R_WIFI_OM_Config(s_str_ap_ssid, s_str_ap_password,
                     s_str_ap_country, s_str_ap_ch, s_str_ap_sec, s_str_ap_enc);
                break;
            }

            case WIFI_API_OM_CONFIG:
            {
                /* Restart after configuration of operation mode complete */
                sprintf((char *)s_sbuf, "OM COMFIG COMP\n");
                debug_printf(s_sbuf);
                R_WIFI_Restart();
                break;
            }
/* Omission */
        }
```

WIFI_EVENT_API_COMPLETE event notification

Define which AT command API result with api_id

Received data is registered in p_data

Figure 3-7 WIFI_EVENT_API_COMPLETE event notification

- When the response to the AT command sent to the DA16200 / DA16600 has an error in the expected response:
  - Value “WIFI_EVENT_ERROR” is set to “event_type”.

```
void wifi_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{

/* Omission */
    if(WIFI_EVENT_ERROR == event_type)
    {
        sprintf(sbuf,"ERROR RESPONSE\n");
        debug_printf(sbuf);
        r_wifi_delay_devspe(5);
    }
/* Omission */
```

WIFI_EVENT_ERROR event notification

Figure 3-8 WIFI_EVENT_ERROR event notification

- When the AT command sent to the DA16200 / DA16600 times out:
  - Value “WIFI_EVENT_TIMEOUT_ERROR” is set to “event_type”.
  - No data is set to “p_data”.
  - When a timeout occurs, it is often assumed that the behavior of the DA16200 / DA16600 is abnormal. Therefore, it is recommended to perform initialization.

```
void wifi_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
/* Omission */
    if (WIFI_EVENT_TIMEOUT_ERROR == event_type)
    {
        /* Set flag to initialize in main loop */
        s_reinitialize_flag = 1;
    }
}
/* Omission */

void wifi_entry(void)
{
/* Omission */
       if (1 == s_reinitialize_flag)
       {
           /* Initialize when timeout occur */
           R_WIFI_Init(wifi_user_cb);
           s_reinitialize_flag = 0;
       }
```

WIFI_EVENT_TIMEOUT_ERROR event notification

Initialization of framework base program

Figure 3-9 WIFI_EVENT_TIMEOUT_ERROR event notification

- When URC is sent from DA16200 / DA16600:
  - Value “WIFI_EVENT_RCVURC” is set to “event_type”.
  - Received URC string data is registered to “p_data”. Execute user process according to the URC. Please execute the process according to the URC. If the data size of the received URC exceeds "WIFI_DATA_STR_SIZE", the excess data is discarded and the data of the first half is registered. The size of the string data to be notified is set to "data_len"

```
void wifi_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{

/* Omission *
   if (WIFI_EVENT_RCVURC == event_type)
    {
        const uint8_t p_str_onconnect[] = "+TRCTS";
        const uint8_t p_str_onmessage[] = "+TRDTS";
        const uint8_t p_str_ondisconn[] = "+TRXTS";
        uint8_t i;

        /* Check TCP data  */
        if (0 == memcmp(p_data, p_str_onmessage, ((sizeof(p_str_onmessage)) - 1)))
        {
            if(0 == memcmp(p_data + 26, "on", 2))
            {
                sprintf((char *)s_sbuf, "LED on\n");
                debug_printf(s_sbuf);
                PIN_WRITE(LED2) = LED_ON;
            }

            if(0 == memcmp(p_data + 26, "off", 3))
            {
                sprintf((char *)s_sbuf, "LED off\n");
                debug_printf(s_sbuf);
                PIN_WRITE(LED2) = LED_OFF;
            }
        }
    }
/* Omission */
```

WIFI_EVENT_RCVURC event notification

Check received URC
Execute process if received data is “+TRDTS”

Analyze parameter of URC

Figure 3-10 WIFI_EVENT_RCVURC event notification

3.4 User Specific Configuration

When users are developing applications based on this sample program, they need to change some settings depending on the RL78 MCU used. In the AT Command Management Framework, a program for setting these user-specific setting values is defined in "r_wifi_da.c". Users can modify this file to use the AT Command Management Framework in the configuration that suits their environment. This section describes the values that can be set.

[Table 3-3 Pin function setting for DA16200 / DA16600] shows the setting items for the RL78/G22 pins connected to each pin of the DA16200 / DA16600. The sample program does not use RTS/CTS or INT pins. Therefore, the functions associated with the RTS/CTS and INT pins are Unused.
Please confirm when changing the board to be used as the host MCU.

Table 3-3 Pin function setting for DA16200 / DA16600

Function name	Description	Used pin
void r_wifi_reset_low_devspe(void)	The RESET pin of DA16200 / DA16600 to low	P17
void r_wifi_reset_high_devspe(void)	The RESET pin of DA16200 / DA16600 to high	P17
void r_wifi_rts_low_devspe(void) (Unused)	The RTS pin of DA16200 / DA16600 to low	P70
void r_wifi_rts_high_devspe(void) (Unused)	The RTS pin of DA16200 / DA16600 to high	P70
uint8_t r_wifi_cts_read_devspe(void) (Unused)	The CTS pin of DA16200 / DA16600 to read	P50
uint8_t r_wifi_int_read_devspe(void) (Unused)	The INT pin of DA16200 / DA16600 to read	P51

[Table 3-4 Smart Configurator of RL78/G22] shows the setting items for using the Smart Configurator within the AT command management framework. Please check if you want to edit the Smart Configurator, change the RL78 MCU to use, etc.

Table 3-4 Smart Configurator of RL78/G22

Tag name	Component	Description
Clock	-	Operation mode: High-speed main mode 1.8 (V) to 5.5 (V) High-speed on-chip oscillator: 32MHz f_{OCO} start setting: Normal f_{IHP}: 32MHz f_{MAIN}: 32MHz f_{CLK}: 32000kHz
System	-	On-chip debug operation setting: Use emulator Emulator setting：E2 Lite Pseudo-RRM/DMM function setting: Used Start/Stop function setting: Unused Security ID setting: Use security ID Security ID: 0x00000000000000000000 Security ID authentication failure setting: Erase flash memory data

Tag name	Component	Description
Component	r_bsp	Start up select: Enable (use BSP startup) Control of invalid memory access detection: Disable RAM guard space (GRAM0-1): Disabled Guard of control registers of port function (GPORT): Disabled Guard of registers of interrupt function (GINT): Disabled Guard of control registers of clock control function, voltage detector, and RAM parity error detection function (GCSC): Disabled Data flash access control (DFLEN): Disables Initialization of peripheral functions by Code Generator/Smart Configurator: Enable API functions disable: Enable Parameter check enable: Enable Setting for starting the high-speed on-chip oscillator at the times of release from STOP mode and of transitions to SNOOZE mode: High-speed Enable user warm start callback (PRE): Unused Enable user warm start callback (POST): Unused Watchdog Timer refresh enable: Unused
	Config_TAU0_1	Component: Interval timer Operation mode: 16 bit count mode Resource: TAU0_1 Operation clock: CK00 Clock source: f_{CLK}/2 Interval value: 1ms Interval setting: Used Priority: Level 3 (low)
	Config_UART0	Component: UART Communication Operation: Transmission/reception Resource：UART0 Operation clock:CK00 Clock source: f_{CLK}/2 Transfer mode setting: Single transfer mode Data length setting: 8 bits Transfer direction setting: LSB Parity setting: None Stop bit length setting: 1bit Transfer data level setting: Non-reverse Transfer rate setting:115200bps Transmit end interrupt priority (INTST0): Level 3(low) Callback function setting: Transmission end

Tag name	Component	Description
Component	Config_UARTA0	Component: UART Communication Operation: Transmission/reception Resource: UARTA0 Operation clock: f_{SEL} Clock source: f_{SEL} clock select f_{IHP} Data length setting: 8 bits Transfer direction setting: LSB Parity setting: None Stop bit length setting: 1bit Transfer data level setting: Non-reverse Transmit mode setting: Continuous transmit by polling Receive error occurs setting: INTUR interrupt occurs Transfer rate setting:115200bps Reception end interrupt priority (INTUR0): Level3(low) Callback function: Reception end, Reception error
	Config_INTC	INTP0 Valid edge: Falling edge, Priority: Level3(low) INTP2 Valid edge: Falling edge, Priority: Level3(low)
	Config_PORT	Port selection: PORT1, PORT5, PORT7 Port mode setting: Read Pmn register values P17: Out P50: In P70: Out

[Table 3-5 Size setting of AT command transmission waiting list] shows the size setting items for various data used within the AT command management framework. AT command management framework defines these setting values in "r_wifi_user_config.h". Please change it according to the data size of the AT command and string used in the application and the stack size of the MCU to be used.

Table 3-5 Size setting of AT command transmission waiting list

Name	Default value	Description
WIFI_ATC_STR_SIZE	60	Maximum length of the AT command string.
WIFI_DATA_STR_SIZE	60	The maximum length of data to receive from the DA16200 / DA16600. If the data to be received exceeds this size, the excess data is discarded.
WIFI_ATC_LIST_SIZE	6	The number of AT commands that can be added to the send waiting list. Define "maximum number of AT commands to be registered + 1".

3.5 Smart Configurator module used in the framework

The AT Command Management Framework uses Smart Configurator modules to implement its functionality. The Smart Configurator module is configured not only in code but also in the Smart Configurator. This section describes how to use and configure the Smart Configurator module used in the AT Command Management Framework.

3.5.1 UARTA module

The AT Command Management Framework uses the UARTA module to implement UART communication between the RL78/G22 and the DA16200 / DA16600.

When sending AT commands from the RL78/G22 to the DA16200 / DA16600, the write function (R_Config_UARTA0_Send) of the UARTA module is used. After calling the AT Command API from your application, a series of AT commands are registered in the Transmit waiting list in the framework. Transmission of AT commands from the waiting list are sequentially processed from the beginning of the list using the write function.

When sending a response from the DA16200 / DA16600 to the RL78/G22, the data is received using the callback function of the UARTA module. This callback function receives a character data one by one and stores it in a ring buffer in the framework. Character data stored in the ring buffer is processed one character at a time R_WIFI_Execute each function call.

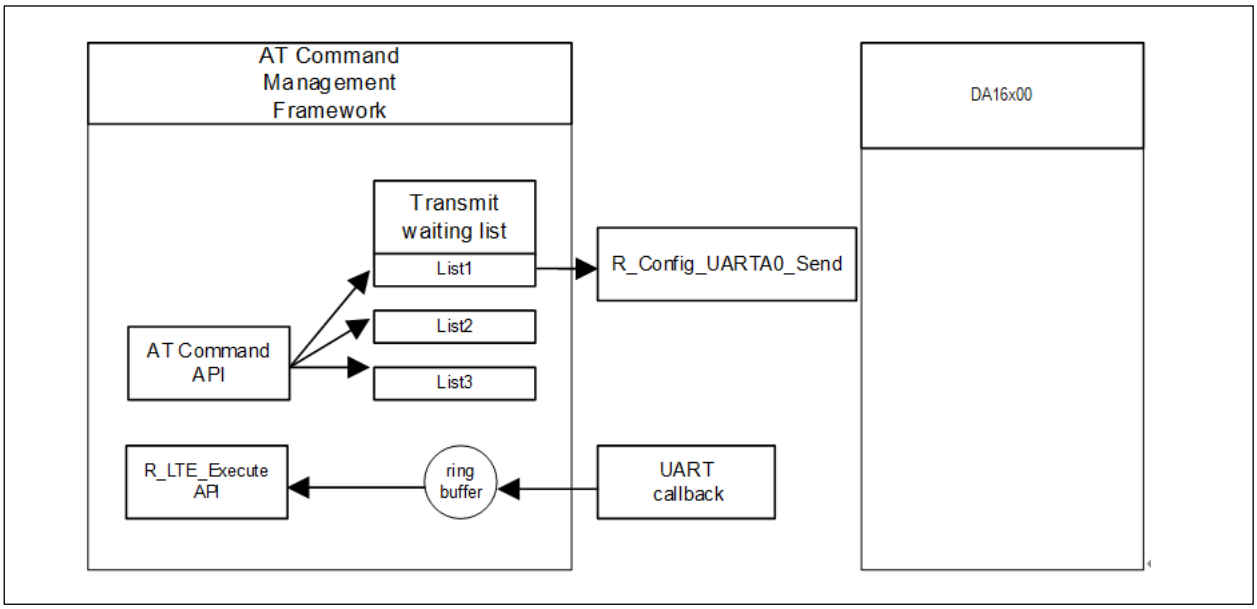

Figure 3-11 Using UARTA module

3.5.2 TAU module

The AT Command Management Framework uses the TAU module to implement the timeout function. After sending an AT command, a timeout occurs when 60 seconds elapse before receiving a response. [Table 3-6 AT command timeout setting (r_wifi_da.c)] shows the definition that sets the timeout period.

Table 3-6 AT command timeout setting (r_wifi_da.c)

Name	Default value	Description
AT_COMMAND_TIMETOUT	30	Timeout count of AT command. unit: 2sec e.g.) 2sec * 30 = 60sec

Framework starts the timer at the timing of sending the AT command. This timer stops when the response specified in the comp_msg is received or when an error response is received. If a response is not received for a certain period after sending an AT command, the timer callback function is called in the framework to signal a timeout has occured. After the callback function is called, framework calls the user's callback function in the R_WIFI_Execute function to notify the application that a timeout has occurred.

The timer count time is set in the Smart Configurator. To change the timeout period, use the Smart Configurator to change the timer count time and [Table 3-6 AT command timeout setting (r_wifi_da.c)].

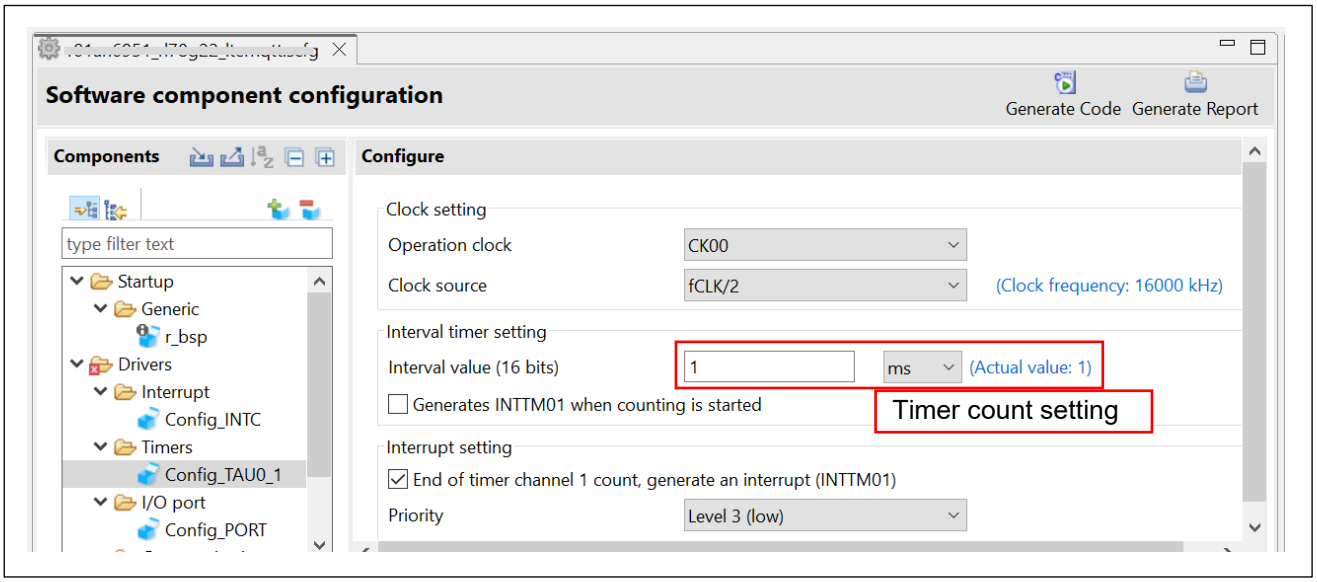

Figure 3-12 TAU module setting

3.5.3 Interrupt function

Using the interrupt function, the interrupt of the INT signal from the DA16200 / DA16600 is connected to INTP2. This INT interrupt is not used in the sample program.

Also, INTP0 is used to detect that SW is pressed.

3.5.4 UART0 module

The UART0 module is used to output the execution results of the sample program. The execution results of the sample program are output via the RL78/G22 FPB's USB (COM port).

4. Application development using AT Command Management Framework

The AT Command Management Framework is intended to be used as a base for user application development. By using the AT Command Management Framework, communication between the RL78/G22 and the DA16200 / DA16600 can be efficiently implemented. In this section, we will describe how to develop user applications using this sample program as an example.

4.1 Overview of application development

The AT Command Management Framework is a specification that allows you to efficiently implement additional APIs within the framework. The API implemented in the framework is called in the application program to realize the operation desired by the user. In this sample program, operation is realized with the following file.

- Framework base program:
  - r_wifi_da.c
  - r_wifi_da.h
  - r_wifi_user_config.h
- Bare metal Application program:
  - wifi_entry.c

The APIs implemented in framework-based programs are classified into two types: management API and AT command API.

Management API

The Management API is the API for managing interactions with the DA16200 / DA16600. It must be implemented in the proper place in the application program. In addition, users do not need to change it during application development.

The following two APIs are implemented in the management API:

- R_WIFI_Init
  This is a function for initializing framework-based programs. This function performs hardware reset of the DA16200 / DA16600. DA16200 / DA16600 will be able to accept AT commands after initialization is completed. After that, it sends an ATF and performs a factory reset. Wait for +INIT:DONE,0 to return, and the callback function specified in the argument will be notified of the event with API_ID = "WIFI_API_INIT". This function should be executed first in all API implemented in the framework.
- R_WIFI_Execute
  This is a function that holds and parses the data received from DA16200 / DA16600, calls the callback function according to the data, and sends AT commands. Since this function processes each character stored in the ring buffer each time it is called, it is necessary to call it repeatedly in the main loop.

```
void wifi_entry(void)
{
    sprintf(sbuf,"PROGRAM START\n");
    debug_printf(sbuf);
    wifi_user_sw_enable_devspe();
    PIN_WRITE(LED1) = LED_OFF;
    PIN_WRITE(LED2) = LED_OFF;

    /* SW  INT interrupt start */
    R_Config_INTC_INTP0_Start();
    /* Initialize framework-based program and register callback function */
    R_WIFI_Init(wifi_user_cb);

    while(1)
   {
        /* Execute variable process in framework-based program */
       R_WIFI_Execute();

/* Omission */
   }
}
```

Call R_WIFI_Init before calling any other APIs in framework

Call R_WIFI_Execute repeatedly in the main loop.

Figure 4-1 Implement management API (wifi_entry.c)

AT command API

A set of AT commands necessary for the operation you want to perform is added to the transmit waiting list by calling the AT command API. The registered AT commands are sent sequentially in response to the response from the DA16200 / DA16600. The execution result of a series of AT commands is notified to the application by a callback function. Users develop applications by calling the AT command API in the order they want and implementing processing corresponding to callback functions. In addition, users can add a new AT command API by themselves and use AT commands not used in this sample program.

```
v void wifi_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len,
uint8_t * p_data)
{
    if(WIFI_EVENT_API_COMPLETE == event_type)
    {
        switch (api_id)
        {
            case WIFI_API_INIT:
            {
                /* Configure operation mode after initiation complete */
                if (0 != data_len)
                {
                    /* URC after ATF */
                    sprintf((char *)s_sbuf, (char *)p_data);
                }
                else
                {
                    /* If URC is not received, this messa
                    sprintf((char *)s_sbuf, "INIT COMP\r
                }
                debug_printf(s_sbuf);
                R_WIFI_OM_Config(s_str_ap_ssid, s_str_ap_password,
                       s_str_ap_country, s_str_ap_ch, s_str_ap_sec, s_str_ap_enc);
                break;
            }

            case WIFI_API_OM_CONFIG:
            {
                /* Restart after configuration of operation mode complete */
                sprintf((char *)s_sbuf, "OM COMFIG COMP\n");
                debug_printf(s_sbuf);
                R_WIFI_Restart();
                break;
            }

/* Omission */
```

1. Call AT command API

2. Receive result with callback function

3. Call next AT command API

Figure 4-2 Implement AT command API (wifi_entry.c)

4.2 Adding an AT command API

This framework assumes that the AT command API is added according to the user's application. This section explains how the AT command API implemented in this sample program and explains how to implement the new AT command API.

To add the AT command API, follow these steps:

1. Adding API IDs and Function Prototype Declarations

   Add the API ID so that the added AT command API can be identified in the callback function. User also adds prototype declarations to the header file (r_wifi_da.h) so that the AT Command API can be executed from the application program.

```
typedef enum
{
    WIFI_API_NO_CURRENT_API = 0,
    WIFI_API_OM_CONFIG,
    WIFI_API_RESTART,
    WIFI_API_RESTART_WAIT,
    WIFI_API_NW_CONFIG,
    WIFI_API_PUSH_MESSAGE,
    WIFI_API_INIT = 0xFF,
} e_wifi_api_id_t;
```

Figure 4-3 API IDs of this sample program (r_wifi_da.h)

2. Implementing the AT command API

   Implement the actual state of the AT command API in the source file (r_wifi_da.c). The AT command API of this sample program is implemented with the following configuration.

Checking arguments and checking the running AT command API

If the argument has a pointer, make sure you do not specify NULL. Also check "s_process_api" to make sure that no other AT command API is running. If it is running, the AT command API cannot operate properly if you change the AT command transmit waiting list, so the error "WIFI_ERR_IN_PROCESS" will be returned without executing any process. After that, to indicate that this AT command API is executing, register the API_ID in "s_process_api".

```
e_wifi_err_t R_WIFI_OM_Config(uint8_t* p_pdp_type, uint8_t* p_pdp_apn, uint8_t*
p_bandlist)
{
    /* Check Argument and current state */
    if((NULL == p_pdp_type) || (NULL == p_pdp_apn) || (NULL == p_bandlist))
    {
        return WIFI_ERR_POINTER_NULL;
    }

    if(WIFI_API_NO_CURRENT_API != gs_process_api)
    {
        return WIFI_ERR_IN_PROCESS;
    }

    /* Clear ATC list and set processing API ID */
    r_wifi_clear_atc_list();
    s_process_api = WIFI_API_OM_CONFIG;

/* Omission */
```

Check argument

Check running AT command API

Register running AT command API

Figure 4-4 Checking the arguments and running AT Command API of R_WIFI_OM_Config (r_wifi_da.c)

Registering AT commands in the Transmission Waiting List

Register the AT command as string data in the transmit waiting list "s_atc_list". The following must be registered in the transmission waiting list "s_atc_list" for one AT command.

- atcommand:
  This is the string data of the AT command you want to execute. The length of the string should be registered in "atcommand_size". The maximum length of a string data that can be registered is 256 characters. If you want to use a larger AT command string data, change the "WIFI_ATC_STR_SIZE" in the user configuration file (r_wifi_user_config.h).
- comp_msg:
  This is a response message that can be considered as the completion of the AT command you want to execute. Specify "OK" or URC. The length of the string should be registered in "comp_msg_size". The following AT command is sent immediately after receiving the string specified in the comp_msg. If "OK" and URC are sent consecutively, register the response to be sent last. In addition, the last comp_msg of a series of AT commands to be added to the send waiting list changes the data notified in the callback function. For details, see [3.3 Callback function].
- data_exist_flag:
  This flag indicates that the AT command to be sent is set. R_WIFI_Execute function checks this value to confirm that the AT command is registered. If you want to register the AT command, set it to "1".

The transmit waiting list "s_atc_list" holds string data by fixed-length arrays. Therefore, if the string data to be registered exceeds the maximum length that can be registered in the transmit waiting list, an error due to a buffer overflow may occur. If the user expects the data size of string data that is being registered can exceed the maximum length, add processing to check the data size.

```
e_wifi_err_t R_WIFI_PUSH_Message(uint8_t* p_tcp_socket, uint16_t length, uint8_t*
p_message)
{
/* Omission */

    /* Set AT command to ATC list */
    s_atc_list[0].atcommand_size = (uint16_t)snprintf((char *)s_atc_list[0].atcommand,
                        WIFI_ATC_STR_SIZE, "%s%s%s,%s,%s\r",
                        "\x1b","S0", s_length, p_tcp_socket, p_message);

    s_atc_list[0].comp_msg_size = (uint16_t)snprintf(
                                                (char *)s_atc_list[0].comp_msg,
                                                WIFI_ATC_STR_SIZE, "%s", "OK");  //
    s_atc_list[0].data_exist_flag = 1;
    if(s_atc_list[0].atcommand_size > WIFI_ATC_STR_SIZE)
    {
        r_wifi_clear_atc_list();
        return WIFI_ERR_DATASIZE_OVERFLOW;
    }

/* Omission */
```

Add following to first of Transmit waiting list:

atcommand = <ESC>S0[ASCII(length)],[p_tcp_socket],[p_message]
comp_msg = "OK"
data_exist_flag = 1

Check the Data Size to register the argument in the transmit waiting list

Figure 4-5 Register AT command of R_WIFI_PUSH_Message (r_wifi_da.c)

Sending the first AT command

Send the AT command from the beginning of the registered transmit waiting list. Subsequent transmission of AT commands is done in R_WIFI_Execute function corresponding to the response.

```
e_wifi_err_t R_WIFI_OM_Config(uint8_t* p_pdp_type, uint8_t* p_pdp_apn, uint8_t*
p_bandlist)
{

/* Omission */

    /* Send first AT command from ATC list */
    r_wifi_transmit_atc_list(WIFI_TRANSMIT_ATCOMMAND);

    return WIFI_SUCCESS;
```

Send first AT command registered in transmit waiting list

Figure 4-6 Sending the first AT command of R_WIFI_OM_Config (r_wifi_da.c)

4.3 Guideline of error handling

In a communication control system, it is necessary to develop an application assuming that various errors occur in the control of the communication controller and network operation. The following is a guideline for application development using this AT command Management Framework for detection and processing. In practice, the processing will vary depending on the requirements for the application product, so please handle it as reference information.

See also the UM-WI-003 DA16200 DA16600 Host Interface and AT Command User Manual (renesas.com).

UART communication and DA16200 / DA16600 behavior error

Defect status	Framework behavior	Application response
UART communication from the DA16200 / DA16600 to the host MCU results in bit errors or character reception errors	If the character string does not match the string specified in the comp_msg, or if the string does not end with "\n", a timeout occurs and calls callback function to notify application.	The callback function is called in event "WIFI_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_WIFI_Init function for this event.
	If the received string matches the URC specified in the comp_msg in front, the application is notified by the callback function.	The callback function is called in event "WIFI_EVENT_RCVURC ". check the data registered in p_data because received string data is registered.
UART communication from the host MCU to the DA16200 / DA16600 results in bit errors or character reception errors	If there is no response to the sent string, a timeout occurs and calls callback function to notify application.	The callback function is called in event "WIFI_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_WIFI_Init function for this event.
The MCU transmission and the transmission timing of the DA16200 / DA16600 overlap, and the DA16200 / DA16600 does not perform the expected operation	A timeout occurs when the operation stops. Framework calls callback function to notify application.	The callback function is called in event "WIFI_EVENT_TIMEOUT_ERROR". It is recommended to initialize using R_WIFI_Init function for this event.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Apr. 8.24	-	1st edition

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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

© 2024 Renesas Electronics Corporation. All rights reserved.