RX Family

SX-ULPGN-2000 Wi-Fi Module Control Module Using Firmware Integration Technology

Introduction

This application note describes the usage of the SX-ULPGN-2000 Wi-Fi module control module, which conforms to the Firmware Integration Technology (FIT) standard.

In the following pages, the SX-ULPGN-2000 Wi-Fi module control module software is referred to collectively as “the SX-ULPGN Wi-Fi FIT module” or “the FIT module.”

The FIT module supports the following Wi-Fi module.

Silex ULPGN (SX-ULPGN-2000)

In the following pages, the Silex ULPGN (SX-ULPGN-2000) is referred to as “the Wi-Fi module.”

The FIT module makes use of the functionality of an RTOS. It is intended to be used in conjunction with an RTOS. In addition, the FIT module does not include a device driver to control the serial communication functionality of the MCU, so you will need to obtain the following application note separately.

RX Family SCI Module Using Firmware Integration Technology (R01AN1815)

Target Device

RX Family

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Related Documents

- Firmware Integration Technology User’s Manual (R01AN1833)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)
- RX Smart Configurator User’s Guide: e² studio (R20AN0451)
- RX Family SCI Module Using Firmware Integration Technology (R01AN1815)
- RX Family BYTEQ Module Using Firmware Integration Technology (R01AN1683)

Contents

1. Overview

1.1 SX-ULPGN Wi-Fi FIT Module

The FIT module is designed to be added to user projects as an API. For instructions on adding the FIT module, refer to 2.10, Adding the FIT Module to Your Project.

1.2 Overview of SX-ULPGN Wi-Fi FIT Module

The FIT module supports both the transparent mode (single-channel communication mode) and separate port mode (two-channel communication mode) of the SX-ULPGN.

1.3 Overview of API

Table 1.1 lists the API functions included in the FIT module. The required memory sizes are listed in 2.8, Code Size.

Table 1.1 API Functions

Function	Function Description
R_WIFI_SX_ULPGN_Open()	Initializes the Wi-Fi module.
R_WIFI_SX_ULPGN_Close()	Closes the Wi-Fi module.
R_WIFI_SX_ULPGN_SetDnsServerAddress()	Sets the DNS server addresses.
R_WIFI_SX_ULPGN_Scan()	Obtains a list of access points.
R_WIFI_SX_ULPGN_Connect()	Connects to an access point.
R_WIFI_SX_ULPGN_Disconnect()	Disconnects from an access point.
R_WIFI_SX_ULPGN_IsConnected()	Obtains the status of a connection to an access point.
R_WIFI_SX_ULPGN_GetMACaddress()	Obtains the MAC address of the Wi-Fi module.
R_WIFI_SX_ULPGN_GetIPaddress()	Obtains the IP address of the Wi-Fi module.
R_WIFI_SX_ULPGN_CreateSocket()	Creates a socket.
R_WIFI_SX_ULPGN_ConnectSocketct()	Starts socket communication.
R_WIFI_SX_ULPGN_SendSocket()	Transmits socket data.
R_WIFI_SX_ULPGN_ReceiveSocket()	Receives socket data.
R_WIFI_SX_ULPGN_ShutdownSocket()	Ends socket communication.
R_WIFI_SX_ULPGN_CloseSocket()	Closes a socket.
R_WIFI_SX_ULPGN_DnsQuery()	Performs a DNS query.
R_WIFI_SX_ULPGN_Ping()	Pings a specified IP address.
R_WIFI_SX_ULPGN_GetVersion()	Returns version information for the module.
R_WIFI_SX_ULPGN_GetTcpSocketStatus()	Obtains the connection status with the WiFi module.
Function related to use of Wi-Fi module SSL functionality	
R_WIFI_SX_ULPGN_RequestTlsSocket ()	Requests use of SSL for socket communication.
Functions related to certificate storage	
R_WIFI_SX_ULPGN_WriteServerCertificate ()	Writes a certificate to the Wi-Fi module.
R_WIFI_SX_ULPGN_EraseServerCertificate ()	Erases a certificate stored in the Wi-Fi module.
R_WIFI_SX_ULPGN_GetServerCertificate()	Obtains certificate information stored in the Wi-Fi module.
R_WIFI_SX_ULPGN_EraseAllCertificate()	Erases all certificates stored in the Wi-Fi module.
R_WIFI_SX_ULPGN_SetCertificateProfile()	Links server information to certificates stored in the Wi-Fi module.

1.4 Processing Example

1.4.1 Hardware

Examples of connections to the SX-ULPGN are shown below.

Figure 1.1 shows connections for single-channel communication mode and Figure 1.2 for two-channel communication mode.

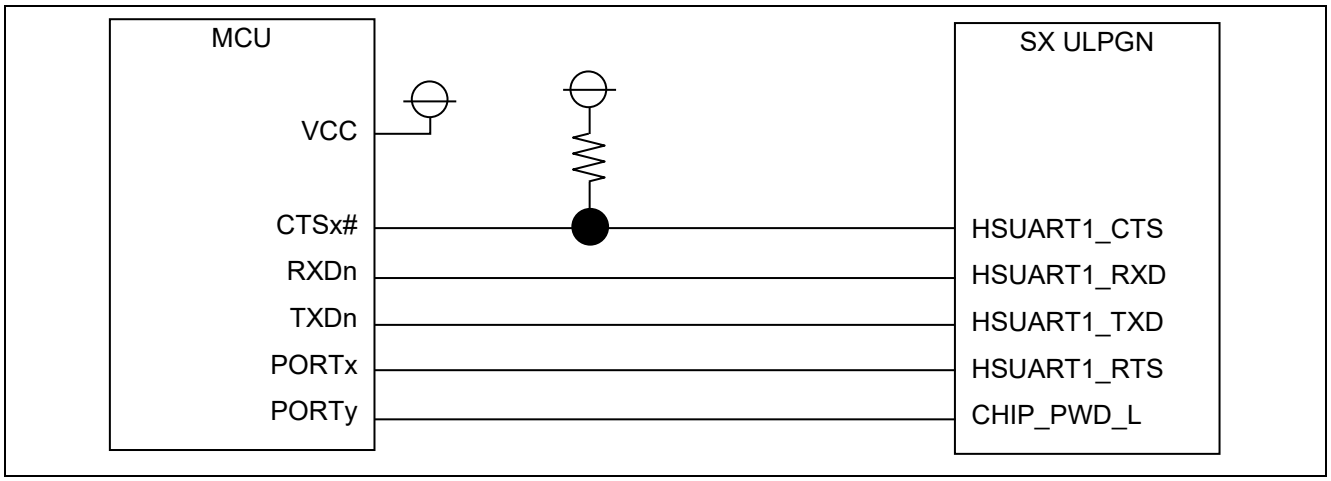

Figure 1.1 Example Connections for Single-Channel Communication Mode

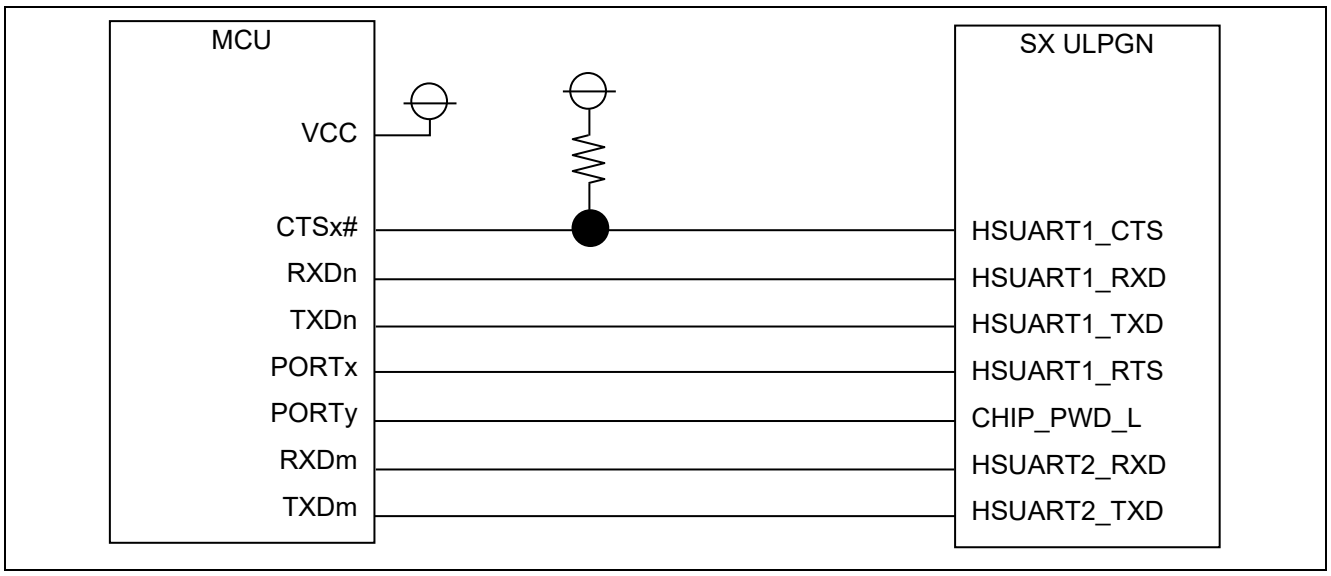

Figure 1.2 Example Connections for Two-Channel Communication Mode

1.4.2 Software

Figure 1.3 shows the software configuration.

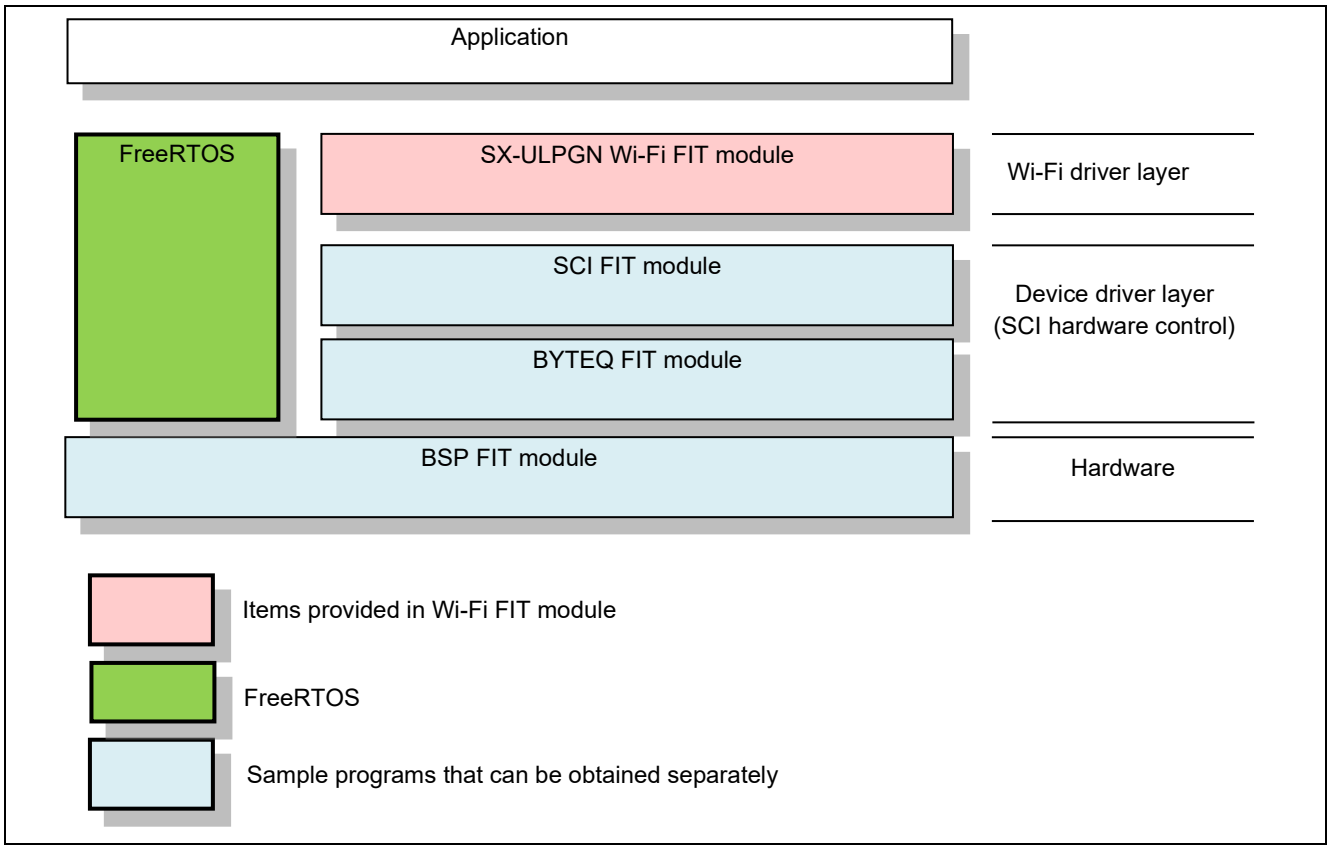

Figure 1.3 Software Configuration Diagram

1. SX-ULPGN Wi-Fi FIT module
   The FIT module. This software is used to control the Wi-Fi module.

2. SCI FIT module
   Implements communication between the Wi-Fi module and the MCU. A sample program is available. Refer to “Related Documents” on page 1 and obtain the software.

3. Peripheral function modules
   This software implements timer control and buffer management. Sample programs are available. Refer to “Related Documents” on page 1 and obtain the software.

4. RTOS
   The RTOS manages the system overall. Operation of the FIT module has been verified using Amazon FreeRTOS.

1.5 Status Transitions

Figure 1.4 shows the status transitions of the FIT module up to communication status.

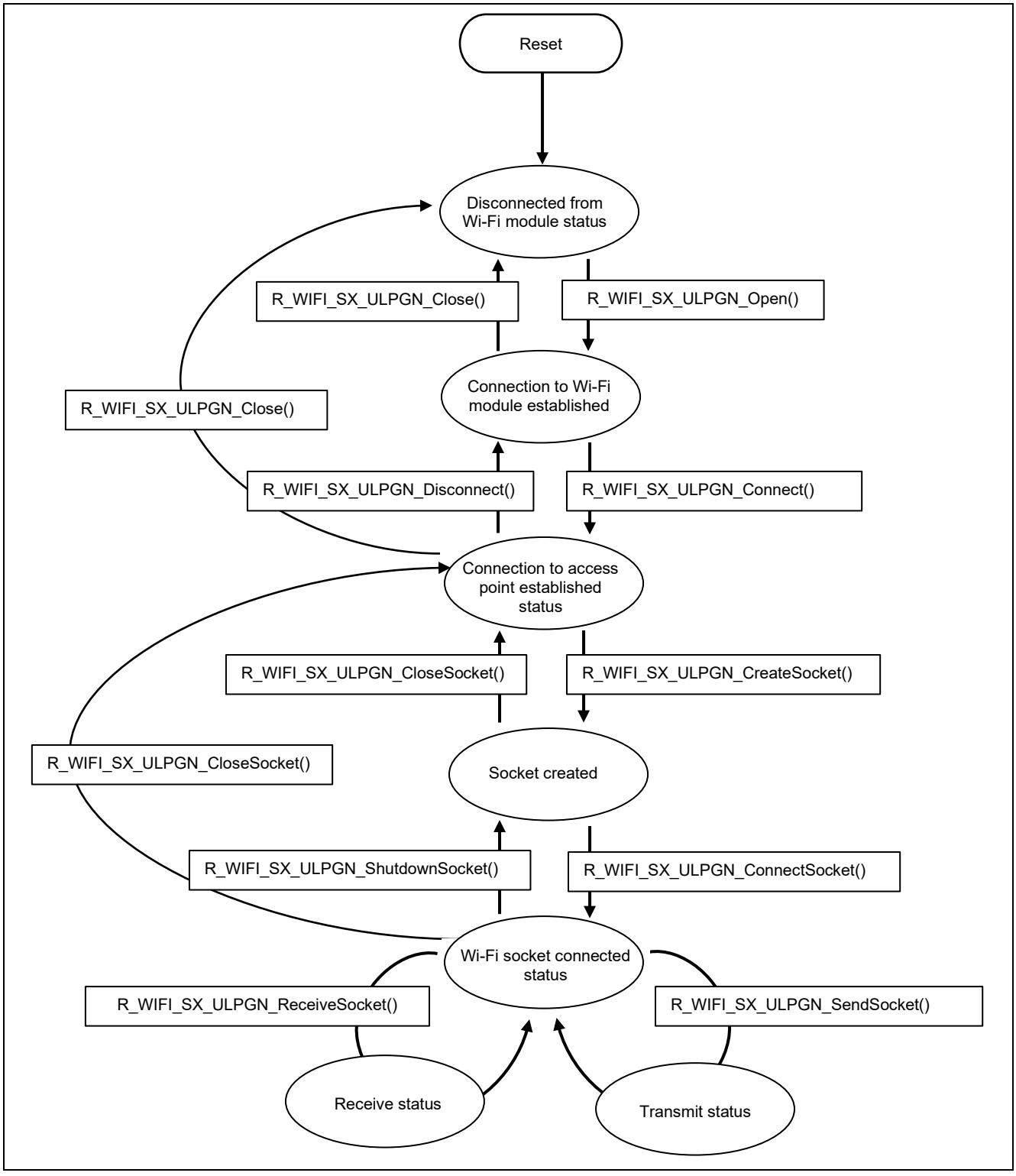

Figure 1.4 Status Transitions

2. API Information

The FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- Serial communication
- I/O ports

2.2 Software Requirements

The driver is dependent upon the following FIT module:

- r_bsp
- r_sci_rx
- r_byteq_rx
- FreeRTOS

2.3 Supported Toolchain

The FIT module has been confirmed to work with the toolchain listed in 5.1, Confirmed Operation Environment.

2.4 Interrupt Vector

None

2.5 Header Files

All API calls and their supporting interface definitions are located in r_wifi_sx_ulpgn_if.h.

2.6 Integer Types

The Wi-Fi FIT module uses ANSI C99. These types are defined in stdint.h.

2.7 Compile Settings

The configuration option settings of the FIT module are contained in r_wifi_sx_ulpgn_config.h and r_sci_rx_config.h.

The names of the options and their setting values are listed in the table below.

Table 2.1 Configuration Options (r_wifi_sx_ulpgn_config.h)

Configuration Options in r_wifi_sx_ulpgn_config.h	
WIFI_CFG_SCI_CHANNEL Note: The default is 0.	Specifies the SCI port number used for communication with HSUART1 on the SX-ULPGN. The default value specifies that SCI port number 0 is used. Enter a setting that matches the SCI port to be controlled.
WIFI_CFG_SCI_SECOND_CHANNEL Note: The default is 1.	Specifies the SCI port number used for communication with HSUART2 on the SX-ULPGN. The default value specifies that SCI port number 1 is used. Enter a setting that matches the SCI port to be controlled.
WIFI_CFG_RTS_PORT Note: The default is 2.	Sets the PDR (port direction register) of the general port that controls the RTS pin of the SX-ULPGN. The default value specifies that port 22 is used. Enter a setting that matches the controlling port.
WIFI_CFG_ RTS_PIN Note: The default is 2.	Sets the PODR (port output data register) of the general port that controls the RTS pin of the SX-ULPGN. The default value specifies that port 22 is used. Enter a setting that matches the controlling port.
WIFI_CFG_RESET_PORT Note: The default is D.	Sets the PDR (port direction register) of the general port that controls the PWD_L pin of the SX-ULPGN. The default value specifies that port D0 is used. Enter a setting that matches the controlling port.
WIFI_CFG_RESET_PIN Note: The default is 0.	Sets the PODR (port output data register) of the general port that controls the PWD_L pin of the SX-ULPGN. The default value specifies that port D0 is used. Enter a setting that matches the controlling port.
WIFI_CFG_CREATABLE_SOCKETS Note: The default is 4.	Sets the number of sockets that the SX-ULPGN can create. The default value is 4. Enter a setting that matches the firmware specifications of the SX-ULPGN.
WIFI_CFG_SOCKETS_RECEIVE_BUFFER_SIZE Note: The default is 8192.	Sets the receive buffer size of the sockets. The default value is 8192. Enter a setting that is appropriate for the memory usage and amount of data to be received.
WIFI_CFG_SCI_INTERRUPT_LEVEL Note: The default is 14.	Sets the interrupt priority of the serial module used for communication with the SX-ULPGN. The default value is 14. Enter a setting that matches the system priority.
WIFI_CFG_SCI_BAUDRATE Note: The default is 460800.	Sets the baud rate (bps) of the serial port used for communication with the SX-ULPGN. The default value is 460800. Enter a setting that is appropriate for the system.
WIFI_CFG_USE_CALLBACK_FUNCTION Note: The default is 0.	Sets whether or not a callback function is registered. 1 = enabled, 0 = disabled
WIFI_CFG_CALLBACK_FUNCTION_NAME Note: The default is NULL.	(This setting is not needed when WIFI_CFG_USE_ERROR_REPORT_FUNCTION = 0.) Registers the name of the callback function. The user must create the callback function. For details, refer to section 4.

Table 2.2 Configuration Options (r_sci_rx_config.h)

Configuration Options in r_sci_rx_config.h	
define SCI_CFG_CHx_INCLUDED Notes: 1. CHx = CH0 to CH12 2. The default values are as follows: CH0 and CH2 to CH12: 0, CH1: 1	Each channel has resources such as transmit and receive buffers, counters, interrupts, other programs, and RAM. Setting this option to 1 assigns related resources to the specified channel.
#define SCI_CFG_CHx_TX_BUFSIZ Notes: 1. CHx = CH0 to CH12 2. The default value is 80 for all channels.	Specifies the transmit buffer size of an individual channel. The buffer size of the channel specified by WIFI_CFG_SCI_CHANNEL should be set to 2048.
#define SCI_CFG_CHx_RX_BUFSIZ Notes: 1. CHx = CH0 to CH12 2. The default value is 80 for all channels.	Specifies the receive buffer size of an individual channel. The buffer size of the channel specified by WIFI_CFG_SCI_CHANNEL should be set to 2048.
#define SCI_CFG_TEI_INCLUDED Note: The default is 0.	Enables the transmit end interrupt for serial transmissions. This option should be set to 1.

Table 2.3 Configuration Options (r_byteq_config.h)

Configuration Options in r_byteq_config.h	
#define BYTEQ_CFG_MAX_CTRL_BLKS	Add the value specified by WIFI_CFG_CREATABLE_SOCKETS.

Table 2.4 Configuration Options (r_bsp_config.h)

Configuration Options in r_byteq_config.h	
#define BSP_CFG_RTOS_USED Note: The default is 0.	Specifies the type of realtime OS. When using the FIT module, set this option to 1.

2.8 Code Size

The code sizes associated with the FIT module are listed in the table below.

Table 2.5 Code Sizes

ROM, RAM and Stack Code Sizes			
Device	**Category**	**Memory Used**	**Remarks**
RX65N	ROM	14,199 bytes	
	RAM	4,826 bytes	
	Max. stack size used	256 bytes	Since use of interrupt interrupts is prohibited, the maximum value when using one channel is shown.

2.9 Return Values

The error codes returned by API functions are listed below. The enumerated types of return values and API function declarations are contained in r_wifi_sx_ulpgn_if.h.

```
typedef enum                                // Wi-Fi API error code
{
    WIFI_SUCCESS                = 0,        // Success
    WIFI_ERR_PARAMETER          = -1,       // Invalid argument.
    WIFI_ERR_ALREADY_OPEN       = -2,       // Already initialized.
    WIFI_ERR_NOT_OPEN           = -3,       // Not initialized.
    WIFI_ERR_SERIAL_OPEN        = -4,       // Serial cannot be initialized.
    WIFI_ERR_MODULE_COM         = -5,       // Communication with Wi-Fi module failed.
    WIFI_ERR_NOT_CONNECT        = -6,       // Access point not connected.
    WIFI_ERR_SOCKET_NUM         = -7,       // Socket is not usable.
    WIFI_ERR_SOCKET_CREATE      = -8,       // Cannot create socket.
    WIFI_ERR_CHANGE_SOCKET      = -9,       // Cannot change socket.
    WIFI_ERR_SOCKET_CONNECT     = -10,      // Cannot connect to socket.
    WIFI_ERR_BYTEQ_OPEN         = -11,      // BYTEQ assignment failure.
    WIFI_ERR_SOCKET_TIMEOUT     = -12,      // Socket transmission timed out.
    WIFI_ERR_TAKE_MUTEX         = -13,      // Mutex lock failure.
} wifi_err_t;
```

The “security” structures are as follows.

```
typedef enum
{
    WIFI_SECURITY_OPEN=0,       // Security type: OPEN
    WIFI_SECURITY_WEP,          // Security type: WEP
    WIFI_SECURITY_WPA,          // Security type: WPA
    WIFI_SECURITY_WPA2,         // Security type: WPA2
    WIFI_SECURITY_UNDEFINED,    // Undefined
} wifi_security_t;

typedef enum
{
    WIFI_SOCKET_STATUS_CLOSED=0,        // Closed status
    WIFI_SOCKET_STATUS_SOCKET,          // Creation successful status
    WIFI_SOCKET_STATUS_BOUND,           // BOUND status
    WIFI_SOCKET_STATUS_LISTEN,          // LISTEN status
    WIFI_SOCKET_STATUS_CONNECTED,       // Connect status
    WIFI_SOCKET_STATUS_MAX,             // Max. number of socket statuses
} wifi_socket_status_t;

typedef struct
{
    uint8_t ssid[ 33 ];             // SSID storage area
    uint8_t bssid[ 6 ];             // BSSID storage area
    wifi_security_t security;       // Security type storage area
    int8_t rssi;                    // Signal strength storage area
    int8_t channel;                 // Channel number storage area
    uint8_t hidden;                 // Hidden channel storage area
} wifi_scan_result_t;
```

```
typedef struct
{
   uint32_t ipadress;       // IP address
   uint32_t subnetmask;     // Subnet mask
   uint32_t gateway;        // Gateway
} wifi_ip_configuration_t;

typedef enum
{
   WIFI_EVENT_WIFI_REBOOT = 0,
   WIFI_EVENT_WIFI_DISCONNECT,
   WIFI_EVENT_SERIAL_OVF_ERR,
   WIFI_EVENT_SERIAL_FLM_ERR,
   WIFI_EVENT_SERIAL_RXQ_OVF_ERR,
   WIFI_EVENT_RCV_TASK_RXB_OVF_ERR,
   WIFI_EVENT_SOCKET_CLOSED,
   WIFI_EVENT_SOCKET_RXQ_OVF_ERR,
} wifi_err_event_enum_t;

typedef struct
{
  wifi_err_event_enum_t event,
  uint32_t socket_number
} wifi_err_event_t;

typedef enum
{
   ULPGN_SOCKET_STATUS_CLOSED            = 0,
   ULPGN_SOCKET_STATUS_SOCKET,
   ULPGN_SOCKET_STATUS_BOUND,
   ULPGN_SOCKET_STATUS_LISTEN,
   ULPGN_SOCKET_STATUS_CONNECTED,
   ULPGN_SOCKET_STATUS_MAX,
} sx_ulpgn_socket_status_t;;

typedef struct
{
   uint8_t   certificate_file[20],
   uint8_t   certificate_number,
   wifi_err_t error_flag,
   void     *nextr_certificate_name
} wifi_err_event_t;
```

2.10 Adding the FIT Module to Your Project

The FIT module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e^2 studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

2.11 RTOS Usage Requirement

The FIT module utilizes RTOS functionality.

2.12 Restrictions

The FIT module is subject to the following restrictions.

- If WIFI_ERR_SERIAL_OPEN occurs, use R_WIFI_SX_ULPGN_Close() to close the Wi-Fi FIT module.
- If R_WIFI_SX_ULPGN_WriteServerCertificate() generates an error, use R_WIFI_SX_ULPGN_EraseAllCertificate() to erase all the certificates stored in the Wi-Fi module, then use R_WIFI_SX_ULPGN_WriteServerCertificate() to write in the certificates again.

3. API Functions

3.1 R_WIFI_SX_ULPGN_Open()

This function initializes the SX-ULPGN Wi-Fi FIT module and Wi-Fi module.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_Open (
        void
)
```

Parameters

None.

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_SERIAL_OPEN	*/* Failed to initialize serial */*
WIFI_ERR_SOCKET_BYTEQ	*/* BYTEQ allocation failure */*
WIFI_ERR_ALREADY_OPEN	*/* Already open */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Initializes the SX-ULPGN Wi-Fi FIT module and also initializes the connected Wi-Fi module.

Determines whether the SX-ULPGN operates in single-channel communication mode or two-channel communication mode.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.2 R_WIFI_SX_ULPGN_Close()

This function disconnects from the access point and disconnects from the Wi-Fi module.

Format
wifi_err_t R_WIFI_SX_ULPGN_Close (

void

)

Parameters
None.

Return Values
WIFI_SUCCESS */* Normal end */*

Properties
Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description
This function disconnects from the access point and terminates communication with the Wi-Fi module.

Reentrant
No

Example
```
Source code
```

Special Notes:
None.

3.3 R_WIFI_SX_ULPGN_SetDnsServerAddress()

This function sets the DNS server IP addresses.

Format

wifi_err_t R_WIFI_SX_ULPGN_SetDnsServerAddress (

uint32_t dns_address1,

uint32_t dns_address2,

)

Parameters

dns_address1
First DNS server IP address

dns_address2
Second DNS server IP address

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not initialized */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Description

When **dns_address1** is other than 0 and **dns_address2** is specified as 0

The address specified by **dns_address1** is used as the DNS server address.

When **dns_address1** is other than 0 and **dns_address2** is other than 0

The addresses specified by **dns_address1** and **dns_address2** are used as the DNS server addresses.

In addition, if R_WIFI_SX_ULPGN_Connect() is run to connect to an access point without first running this function, the DNS server address used is as follows:

When auto_ip_assign = 0 for R_WIFI_SX_ULPGN_Connect(), the specified gateway address is used.

When auto_ip_assign = 1 for R_WIFI_SX_ULPGN_Connect(), a DNS server address assigned by the DHCP server is used.

Call this function after calling R_WIFI_SX_ULPGN_Open() and before calling R_WIFI_SX_ULPGN_Connect().

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

3.4 R_WIFI_SX_ULPGN_Scan()

This function scans for access points.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_Scan (
        wifi_scan_result_t *ap_results,
        uint32_t max_networks,
        uint32_t *exist_ap_count
)
```

Parameters

**ap_results*
Pointer to start address of **wifi_scan_result_t** array for storing scan results

max_networks
Number of **ap_results** values that can be stored

exist_ap_count
Number of access points that exist

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not initialized */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function scans for access points in the periphery of the Wi-Fi module.

The results of the scan are stored in the area specified by the **ap_results** argument, up to the maximum number of values specified by the **max_networks** argument.

In addition, the number of access points detected is reported in **exist_ap_count**.

Example

```
#define AP_LIST_BUFFER_COUNT 10
wifi_scan_result_t ap_result_buffer[AP_LIST_BUFFER_COUNT];
uint32_t exist_ap;
wifi_err_t err;

err = R_WIFI_SX_ULPGN_Scan (
       ap_result_buffer, AP_LIST_BUFFER_COUNT, &exist_ap);
```

3.5 R_WIFI_SX_ULPGN_Connect()

This function connects the Wi-Fi module to an access point.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_Connect (
        uint8_t  *ssid,
        uint8_t *pass,
        uint32_t security,
        uint8_t dhcp_enable
        wifi_ip_configuration_t *ip_config

)
```

Parameters

**ssid*
Pointer to SSID of access point

**pass*
Pointer to password of access point

security
Security type information

```
WIFI_SECURITY_WPA       /* WPA type */
WIFI_SECURITY_WPA2      /* WPA2 type */
```

dhcp_enable
Automatic IP address assignment

0: Disabled (Sets a static IP address.)

1: Enabled (Automatically assigns an IP address from the access point.)

ip_config
When the value of **auto_ip_assign** is 0, the IP address information specified in **ip_config** is set in the Wi-Fi module.

When the value of **auto_ip_assign** is 1, automatically assigned IP address information is stored in **ip_config**.

Return Values

```
WIFI_SUCCESS            /* Normal end */
WIFI_ERR_NOT_OPEN       /* Wi-Fi module not initialized */
WIFI_ERR_PARAMETER      /* Invalid argument */
WIFI_ERR_TAKE_MUTEX     /* Failed to obtain semaphore */
WIFI_ERR_MODULE_COM     /* Failed to communicate with Wi-Fi module */
```

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description
Connects to the access point specified by **pssid**.

When DHCP is enabled, the function waits for an IP address to be assigned after connection to the access point succeeds.

Reentrant
No

Example
DHCP enabled:

```
wifi_err_t err;
wifi_ip_configuration_t ipconfig;

R_WIFI_SX_ULPGN_Connect (
       "test_SSID", "test_password", WIFI_SECURITY_WPA2, 1, &ip_config);
```

DHCP disabled:

```
wifi_err_t err;
wifi_ip_configuration_t ip_config;

ip_config.ipaddr = 0xc0a80003;  //192.168.0.3
ip_config.subnetmask = 0xffffff00;   //255.255.255.0
ip_config.gateway = 0xc0a80001;   //192.168.0.1

R_WIFI_SX_ULPGN_Connect (
       "test_SSID", "test_password", WIFI_SECURITY_WPA2, 0, &ip_config);
```

Special Notes:
None.

3.6 R_WIFI_SX_ULPGN_Disconnect ()

This function disconnects the Wi-Fi module from the access point.

Format

wifi_err_t R_WIFI_SX_ULPGN_Disconnect (

void

)

Parameters

None.

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not initialized */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Disconnects the Wi-Fi module from the access point.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.7 R_WIFI_SX_ULPGN_IsConnected()

This function obtains the connection status of the Wi-Fi module and access point.

Format
int32_t R_WIFI_SX_ULPGN_IsConnected (

void

)

Parameters
None.

Return Values
0 /* Connected */
-1 /* Not connected */

Properties
Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description
Returns the connection status of the Wi-Fi module and access point.

Reentrant
No

Example
```
Source code
```

Special Notes:
None.

3.8 R_WIFI_SX_ULPGN_GetMacAddress ()

This function obtains the MAC address value of the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_GetMacAddress (
        uint8_t *mac_address
)
```

Parameters

**mac_address*
Pointer to storage area for MAC address of Wi-Fi module (6 bytes)

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not initialized */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Obtains the MAC address value of the Wi-Fi module. The MAC address is stored as binary data in **mac_address**.

Example

MAC address 11:22:33:44:55:66

mac_address[0] = 0x11, mac_address [1] = 0x22, mac_address [3] = 0x33, …, mac_address [5] = 0x66

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.9 R_WIFI_SX_ULPGN_GetIpAddress()

This function obtains the IP address of the Wi-Fi module from the internet server.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_GetIpAddress (
        wifi_ip_configuration_t *ip_config
)
```

Parameters

** ip_config*
Pointer to IP address storage area

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not initialized */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

The obtained IP address, subnet mask, and gateway address are stored in **ip_config**.

Example: When the IP address is 192.168.0.3, the subnet mask is 255.255.255.0, and the gateway is 192.168.0.1

ip_config ->ipaddr = 0xc0a80003

ip_config ->subnetmask = 0xffffff00

ip_config ->gateway = 0xc0a80001

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.10 R_WIFI_SX_ULPGN_CreateSocket ()

This function sets the socket type and IP type of a socket available for use.

Format

```
int32_t R_WIFI_SX_ULPGN_CreateSocket (
        uint32_t type,
        uint32_t ip_version,
)
```

Parameters

type
Socket type

```
        WIFI_SOCKET_IP_PROTOCOL          /* TCP */
```

ip_version
IP version

```
        WIFI_SOCKET_IP_VERSION_4         /* IPv4 */
```

Return Values

Positive value	*/* Normal end (number of socket that was created) */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_NOT_CONNECT,	*/* Not connected to access point */*
WIFI_ERR_SOCKET_CREATE,	*/* Failed to create socket */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Creates a socket and reports the socket number as a return value. The socket number setting is an integer value between 0 and 3.

Reentrant

No

Example

```
Source code
```

3.11 R_WIFI_SX_ULPGN_ConnectSocket ()

This function connects to the specified address.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_ConnectSocket (
        int32_t socket_number,
        uint32_t ip_address,
        uint16_t port,
        char     *destination,
)
```

Parameters

socket_number
Socket number

ip_address
IP address of communications partner

port
Port number of communications partner

destination
Server name of communications partner

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_SOCKET_NUM,	*/* No socket available for connection socket */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*
WIFI_ERR_NOT_CONNECT,	*/* Not connected to access point */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Connects to the specified IP address and port number using the socket specified by **socket_number**. Before calling this API function, call R_WIFI_SX_SocketCreate() to create the socket to be used.

If R_WIFI_SX_SocketCreate() has not been run, WIFI_ERR_SOCKET_NUM is returned.

When SSL communication is enabled, the function makes SSL-related settings. To use SSL, call R_WIFI_SX_ULPGN_SocketOptRequireTls() to enable SSL communication before calling this API function and set the certificate for the socket to be used. If R_WIFI_SX_ULPGN_SocketOptRequireTls() has not been run, non-SSL communication takes place.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.12 R_WIFI_SX_ULPGN_SendSocket ()

This function transmits data using the specified socket.

Format

```
int32_t R_WIFI_SX_ULPGN_SendSocket (
        uint8_t socket_number,
        const uint8_t *data,
        uint32_t length,
        uint32_t timeout_ms,
)
```

Parameters

socket_number
Socket number

**data*
Pointer to transmit data storage area

length
Number of bytes of data to be transmitted

timeout_ms (not used)
Transmission timeout duration [ms]

Return Values

Positive value	*/* Normal end (number of bytes that have been transmitted) */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_SOCKET_NUM,	*/* No socket available for connection socket */*
WIFI_ERR_NOT_CONNECT,	*/* Not connected to access point */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_CHANGE_SOCKET	*/* Failed to change socket */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Transmits from the specified socket the number of bytes specified by **length** of the data stored in **data**.

R_WIFI_SX_ULPGN_SocketConnect() must be called before calling this API function.

If R_WIFI_SX_ULPGN_SocketConnect() has not been run and the socket is not connected, WIFI_ERR_SOCKET_CONNECT is returned.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.13 R_WIFI_SX_ULPGN_ReceiveSocket ()

This function receives data from the specified socket.

Format

```
int32_t R_WIFI_SX_ULPGN_ReceiveSocket (
        int32_t socket_number,
        uint8_t *data,
        int32_t length,
        uint32_t timeout_ms
)
```

Parameters

socket_number
Socket number

**data*
Pointer to receive data storage area

data_length
Number of bytes of data to be received

timeout_ms
Reception timeout duration [ms]

No timeout when set to 0

Return Values

Positive value	*/* Normal end (number of bytes that have been received) */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_NOT_CONNECT,	*/* Not connected to access point */*
WIFI_ERR_SOCKET_NUM	*/* No socket available for connection socket */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_CHANGE_SOCKET	*/* Failed to change socket */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Receives from the specified socket the number of bytes specified by **length** of the data stored in **data**. If the amount of data specified by **length** is not obtained during the duration specified by **timeout_ms**, the amount of data that has been received is returned.

Reentrant

No

Example

```
Source code
```

Special Notes:
None.

3.14 R_WIFI_SX_ULPGN_ShutdownSocket ()

This function ends communication using the specified socket.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_ShutdownSocket (
        int32_t socket_number
)
```

Parameters

socket_number
Socket number

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_NOT_CONNECT,	*/* Not connected to access point */*
WIFI_ERR_SOCKET_NUM	*/* No socket available for connection socket */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_CHANGE_SOCKET	*/* Failed to change socket */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Ends communication using the socket specified by **socket _number**.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.15 R_WIFI_SX_ULPGN_CloseSocket ()

This function disconnects the specified socket from the network.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_CloseSocket (
        uint8_t socket_number
)
```

Parameters

socket_number
Socket number

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_NOT_CONNECT,	*/* Not connected to access point */*
WIFI_ERR_SOCKET_NUM	*/* No socket available for connection socket */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_CHANGE_SOCKET	*/* Failed to change socket */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Closes the socket specified by **socket _number**.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.16 R_WIFI_SX_ULPGN_DnsQuery()

This function performs a DNS query.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_DnsQuery (
        const uint8_t *domain_name,
        uint32_t *ip_address
)
```

Parameters

** domain_name*
Domain name

**ip_address*
IP address storage area

Return Values

```
WIFI_SUCCESS          /* Normal end */
WIFI_ERR_NOT_CONNECT, /* Not connected to access point */
WIFI_ERR_PARAMETER    /* Invalid argument */
WIFI_ERR_MODULE_COM   /* Failed to communicate with Wi-Fi module or domain does not exist */
```

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Performs a DNS query to obtain the IP address of the specified domain.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.17 R_WIFI_SX_ULPGN_Ping()

This function pings the specified IP address.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_Ping (
        uint32_t ip_address,
        uint16_t count,
        uint32_t interval_ms
)
```

Parameters

ip_address
IP address

count
Number of ping transmissions

interval_ms
Wait time between ping transmissions [ms]

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_PARAMETER	*/* Invalid argument */*
WIFI_ERR_NOT_CONNECT	*/* Not connected to access point */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module or no response */*
WIFI_ERR_TAKE_MUTEX	*/* Mutex lock failure */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

Pings the IP address specified by **ip_address**. The IP address is stored in **ip_address** in the following format.

(Example) IP address: 11.22.33.44

ip_address = 0x0b16212c;

R_WIFI_SX_ULPGN_Ping (ip_address, 1, 1000);

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.18 R_WIFI_SX_ULPGN_GetVersion()

This function obtains version information for the FIT module.

Format
uint32_t R_WIFI_SX_ULPGN_GetVersion(

void

)

Parameters
None.

Return Values
Version number

Properties
Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description
This function returns the version number of the FIT module. The version number is encoded, with the upper two bytes designating the major version number, and the lower two bytes designating the minor version number.

Reentrant
No

Example
```
Source code
```

Special Notes:
None.

3.19 R_WIFI_SX_ULPGN_GetTcpSocketStatus()

This function obtains the connection status with the WiFi module.

Format

```
int32_t R_WIFI_SX_ULPGN_GetTcpSocketStatus(
      uint8_t socket_number
)
```

Parameters

socket_number

ソケット番号

Return Values

WIFI_SOCKET_STATUS_CLOSED=0,	*// close status*
WIFI_SOCKET_STATUS_SOCKET,	*// socket status*
WIFI_SOCKET_STATUS_BOUND,	*// bound status*
WIFI_SOCKET_STATUS_LISTEN,	*// listen status*
WIFI_SOCKET_STATUS_CONNECTED,	*// connected status*
-1	*// error status*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function obtains the connection status with the WiFi module.

Reentrant

No

Example

```
Source code
```

Special Notes:

None.

3.20 R_WIFI_SX_ULPGN_RequestTlsSocket ()

This function performs TLS communication requests by means of socket communication.

Format

wifi_err_t R_WIFI_SX_ULPGN_RequestTlsSocket

int32_t socket_number,

)

Parameters

socket_number
Socket number

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_SOCKET_NUM	*/* No socket available for connection socket */*
WIFI_ERR_NOT_CONNECT,	*/* Not connected to access point */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function performs TLS communication requests by means of socket communication.

Call this function after calling R_WIFI_SX_ULPGN_CreateSocket() and before calling RX_WIFI_SX_ULPGN_ConnectSocket().

Reentrant

No

Example

Issues a TLS communication request on socket number 0 and assigns a certificate with ID code 0.

```
Source code

R_WIFI_SX_ULPGN_RequestTlsSocket (0);
```

Special Notes:

None.

3.21 R_WIFI_SX_ULPGN_WriteServerCertificate()

This function stores a certificate in the Wi-Fi module.

Format

wifi_err_t R_WIFI_SX_ULPGN_WriteServerCertificate

uint32_t data_id,

uint32_t data_type,

uint8_t *certificate,

uint32_t certificate_length

)

Parameters

data_id

Certificate ID code (0 to 4)

data_type

Certificate type

0: Certificate

1: CA list

certificate

Pointer to certificate data

Specifies variable where certificate is stored

certificate_length

Certificate size

Specifies the size of the certificate

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIRI_ERR_PARAMETER	*/* Certificate data not set correctly */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not open */*
WIFI_ERR_TAKE_MUTEX	*/* Mutex lock failure */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function writes a certificate to the Wi-Fi module.

R_WIFI_SX_ULPGN_Open() must be called before calling this API function.

The file name of the certificate is set as follows, based on the certificate ID code and certificate type.

When certificate type: 0 (certificate)

cert<certificate ID>.crt

When certificate type: 1 (CA list)

calist<certificate ID>.crt

Up to five certificate file sets can be stored in the Wi-Fi module.

Certificate data must be converted to SharkSSLParseCert binary format, and CA lists must be converted to SharkSSLPerseCAList binary format.

For instructions on creating certificates, converting them to SharkSSLParseCert binary format, and importing them into projects, refer to 5.3, Appendix (Procedure for Importing Certificate Data).

Reentrant

No

Example

Source code

```
void prvWifiSetCertification(void)
{
     /* Get Initial Server Certificate Information */
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
     R_WIFI_SX_ULPGN_EraseAllServerCertificate();
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
     R_WIFI_SX_ULPGN_EraseAllServerCertificate();
     R_WIFI_SX_ULPGN_WriteServerCertificate (0,1,(uint8_t*)&sharkSslRSACert_PC,
(uint32_t)sharkSslRSACert_PCLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (0,0,(uint8_t*)&sharkSslCAList_PC,
(uint32_t)sharkSslCAList_PCLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (1,1,(uint8_t*)&sharkSslRSACert,
(uint32_t)sharkSslRSACertLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (1,0,(uint8_t*)&sharkSslCAList,
(uint32_t)sharkSslCAListLength);
     /* Get Updated Server Certificate Information */
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
}
```

Special Notes:

None.

3.22 R_WIFI_SX_ULPGN_EraseServerCertificate()

This function deletes a certificate stored in the Wi-Fi module.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_EraseServerCertificate
        uint8_t *certificate_name
)
```

Parameters

certificate_name
Pointer to certificate file name

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIRI_ERR_PARAMETER	*/* Certificate file name not set correctly */*
WIFI_ERR_TAKE_MUTEX	*/* Failed to obtain semaphore */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not open */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function deletes the certificate with the specified file name from the Wi-Fi module.

R_WIFI_SX_ULPGN_Open() must be called before calling this API function.

The certificates stored in the Wi-Fi module can be checked by calling R_WIFI_SX_ULPGN_GetServerCertificate().

Reentrant

No

Example

```
  Source code

  void prvWifiSetCertification(void)
  {
       /* Get Initial Server Certificate Information */
       R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
       R_WIFI_SX_ULPGN_EraseAllServerCertificate();
       R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
       R_WIFI_SX_ULPGN_EraseAllServerCertificate();
       R_WIFI_SX_ULPGN_WriteServerCertificate (0,1,(uint8_t*)&sharkSslRSACert_PC,
(uint32_t)sharkSslRSACert_PCLength);
       R_WIFI_SX_ULPGN_WriteServerCertificate (0,0,(uint8_t*)&sharkSslCAList_PC,
(uint32_t)sharkSslCAList_PCLength);
       R_WIFI_SX_ULPGN_WriteServerCertificate (1,1,(uint8_t*)&sharkSslRSACert,
(uint32_t)sharkSslRSACertLength);
       R_WIFI_SX_ULPGN_WriteServerCertificate (1,0,(uint8_t*)&sharkSslCAList,
(uint32_t)sharkSslCAListLength);
       /* Get Updated Server Certificate Information */
       R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
  }
```

Special Notes:

None.

3.23 R_WIFI_SX_ULPGN_GetServerCertificate()

This function obtains the file names of the certificates stored in the Wi-Fi module.

Format

wifi_err_t R_WIFI_SX_ULPGN_GetServerCertificate

wifi_certificate_infomation_t *wifi_certificate_information

)

Parameters

wifi_certificate_information

Pointer to certificate information storage area

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIRI_ERR_PARAMETER	*/* Certificate file name not set correctly */*
WIFI_ERR_TAKE_MUTEX	*/* Mutex lock failure */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function obtains certificate information stored in the Wi-Fi module and returns the certificate information start address in **wifi_certificate_information**.

R_WIFI_SX_ULPGN_Open() must be called before calling this API function.

Reentrant

No

Example

```
Source code

void prvWifiSetCertification(void)
{
     /* Get Initial Server Certificate Information */
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
     R_WIFI_SX_ULPGN_EraseAllServerCertificate();
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
     R_WIFI_SX_ULPGN_EraseAllServerCertificate();
     R_WIFI_SX_ULPGN_WriteServerCertificate (0,1,(uint8_t*)&sharkSslRSACert_PC,
(uint32_t)sharkSslRSACert_PCLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (0,0,(uint8_t*)&sharkSslCAList_PC,
(uint32_t)sharkSslCAList_PCLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (1,1,(uint8_t*)&sharkSslRSACert,
(uint32_t)sharkSslRSACertLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (1,0,(uint8_t*)&sharkSslCAList,
(uint32_t)sharkSslCAListLength);
     /* Get Updated Server Certificate Information */
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
}
```

Special Notes:

None.

3.24 R_WIFI_SX_ULPGN_EraseAllServerCertificate()

This function erases all the certificates stored in the Wi-Fi module.

Format

wifi_err_t R_WIFI_SX_ULPGN_EraseAllServerCertificate

    void

)

Parameters

None.

Return Values

WIFI_SUCCESS	*/* Normal end */*
WIFI_ERR_NOT_OPEN	*/* Wi-Fi module not open */*
WIFI_ERR_TAKE_MUTEX	*/* Mutex lock failure */*
WIFI_ERR_MODULE_COM	*/* Failed to communicate with Wi-Fi module */*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function erases all the certificates stored in the Wi-Fi module.
R_WIFI_SX_ULPGN_Open() must be called before calling this API function.

Reentrant

No

Example

Source code

```
void prvWifiSetCertification(void)
{
     /* Get Initial Server Certificate Information */
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
     R_WIFI_SX_ULPGN_EraseAllServerCertificate();
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
     R_WIFI_SX_ULPGN_EraseAllServerCertificate();
     R_WIFI_SX_ULPGN_WriteServerCertificate (0,1,(uint8_t*)&sharkSslRSACert_PC,
(uint32_t)sharkSslRSACert_PCLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (0,0,(uint8_t*)&sharkSslCAList_PC,
(uint32_t)sharkSslCAList_PCLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (1,1,(uint8_t*)&sharkSslRSACert,
(uint32_t)sharkSslRSACertLength);
     R_WIFI_SX_ULPGN_WriteServerCertificate (1,0,(uint8_t*)&sharkSslCAList,
(uint32_t)sharkSslCAListLength);
     /* Get Updated Server Certificate Information */
     R_WIFI_SX_ULPGN_GetServerCertificate(wifi_certificate_information);
}
```

Special Notes:

None.

3.25 R_WIFI_SX_ULPGN_SetCertificateProfile()

This function links server information to certificates stored in the Wi-Fi module.

Format

void R_WIFI_SX_ULPGN_SetCertificateProfile

uint8_t certificate_id,

uint32_t ip_address,

char *sever_name

)

Parameters

certificate_id
Certificate ID number

ip_address
Server IP address

server_name
Pointer to server name

Return Values

WIFI_SUCCESS

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function links server information to certificates stored in the Wi-Fi module.

The certificate ID is a required item that must be specified. Either the server IP address or the server name may be specified. If both are specified, the server IP address takes precedence.

Reentrant

No

Example

Links an IP address to certificate ID0 and a server name to certificate 1.

Source code

```
void prvSetCertificateProfile(void)
{
    uint32_t ipaddress;

    ipaddress =
SOCKETS_inet_addr_quick(tcptestECHO_SERVER_TLS_ADDR3,tcptestECHO_SERVER_TLS_ADDR
2,tcptestECHO_SERVER_TLS_ADDR1,tcptestECHO_SERVER_TLS_ADDR0);
    R_WIFI_SX_ULPGN_SetCertificateProfile(0,ipaddress,"");
    R_WIFI_SX_ULPGN_SetCertificateProfile(1,0,(char*)clientcredentialMQTT_BROK
ER_ENDPOINT);
}
```

Special Notes:

None.

4. Callback Function

4.1 callback()

This function reports an event from the Wi-Fi module.

Format

```
void * callback(
        void * pevent
)
```

Parameters

pevent
Pointer to error information area

Return Values

None.

Properties

These are declared by the user.

Description

The FIT module calls a callback function set by the user when an error report is received from the SCI FIT module.

The callback function is specified by storing the address of a user function in configuration item WIFI_CFG_ERROR_REPORT_FUNCTION_NAME, listed in 2.7, Compile Settings. It is not necessary to use “callback” as the name of the function.

When the callback function is called, the start address of the notification details indicated by the wifi_err_event_t type is passed to it as an argument.

The argument is passed as a void pointer type, so it is necessary to convert the argument of the callback function to a void type pointer variable as shown in the example below.

To use the argument internally in the callback function, cast it to the wifi_err_event_t type.

The set values of the **event** members of the **wifi_err_event_t** type and their descriptions are listed below.

- WIFI_EVENT_SERIAL_OVF_ERR
  Reports that the SCI module has detected a receive overflow error. This indicates a status in which UART transmission/reception control cannot be performed correctly. Restart the Wi-Fi module.
- WIFI_EVENT_SERIAL_FLM_ERR
  Reports that the SCI module has detected a receive framing error. This indicates a status in which UART transmission/reception control cannot be performed correctly. Restart the Wi-Fi module.
- WIFI_EVENT_SERIAL_RXQ_OVF_ERR
  Reports that the SCI module has detected an error indicating that receive data cannot be set in the receive queue area. This indicates a status in which UART transmission/reception control cannot be performed correctly. Restart the Wi-Fi module.
- WIFI_EVENT_RCV_TASK_RXB_OVF_ERR
  Reports that an error indicating that receive data cannot be set in the receive buffer of the Wi-Fi module has been detected.

- WIFI_EVENT_SOCKET_RXQ_OVF_ERR
  Reports that an error indicating that receive data cannot be set in the socket receive queue has been detected. The socket number is indicated by **pevent -> socket_number**.

Reentrant
No

Example

```
[r_wifi_sx_ulpgn_config.h]
#define WIFI_CFG_USE_CALLBACK_FUNCTION   1
#define WIFI_CFG_CALLBACK_FUNCTION_NAME   wifi_callback

[xxx.c]
void wifi_callback(void *p_args)
{
     wifi_err_event_t *pevent;
     pevent = (wifi_err_event_t *)p_args;

     switch(pevent->event)
     {
           case WIFI_EVENT_WIFI_REBOOT:
                 break;
           case WIFI_EVENT_WIFI_DISCONNECT:
                 break;
           case WIFI_EVENT_SERIAL_OVF_ERR:
                 break;
           case WIFI_EVENT_SERIAL_FLM_ERR:
                 break;
           case WIFI_EVENT_SERIAL_RXQ_OVF_ERR:
                 break;
           case WIFI_EVENT_RCV_TASK_RXB_OVF_ERR:
                 break;
           case WIFI_EVENT_SOCKET_CLOSED:
                 switch(pevent->socket_number)
                 {
                       case 0:
                             break;
                       case 1:
                             break;
                 /*  To omit  */
                       case 3:
                             break;
                 }
                 break;
           case WIFI_EVENT_SOCKET_RXQ_OVF_ERR:
                 switch(pevent->socket_number)
                 {
                       case 0:
                             break;
                       case 1:
                             break;
                 /*  To omit  */
                       case 3:
                             break;
                 }
                 break;
     }
}
```

Special Notes:

Do not call any of the functions listed in section 3, API Functions, from the callback function.

Example pin settings when using a Renesas Target Board are shown below.

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the FIT module.

Table 5.1 Confirmed Operation Environment (Ver. 1.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.08.00
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.02.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big endian/little endian
Revision of the module	Rev.1.00
Board used	Renesas RX65N Cloud Kit (product No.: RTK5RX65N0SxxxxxBE)

5.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:
  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”
- Using e^2 studio:
  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file “r_wifi_sx_ulpgn_config.h” may be wrong. Check the file “r_wifi_sx_ulpgn_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.7 Compile Settings for details.

(3) Q: The pin setting is supposed to be done, but this does not look like it.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 4. Pin Setting for details.

5.3 Appendix (Procedure for Importing Certificate Data)

The procedure for creating a certificate to be written to the Wi-Fi module to enable TLS communication is described below.

5.3.1 Creating a Certificate

Use OpenSSL to create a certificate. Install OpenSSL on the PC you wish to use. The steps for creating a certificate are as follows.

- openssl genrsa –out certs/client.key 2048
- openssl req –new –key certs/client.key –out certs/client.csr ¥
  –subj “/C=JP/L=<States>/O=<Company>/OU=<Department>/CN=<Object>/email=<EmailAddress>”
- openssl x509 –req –in certs/client.csr –CA certs.server.pem –CAkey certs/server.key ¥
  –CAcreateserial –out certs/client.pem –days 365 –sha256”

5.3.2 Converting the Format

In order to be written to the Wi-Fi module, certificate data must be converted to SharkSSLParseCert binary format and CA lists to SharkSSLPerseCAList binary format.

The following freeware application can be used for format conversion.

SharkSSL <https://realtimelogic.com/downloads/sharkssl/>

Follow the software instructions to download and install the application. The method of converting the format of certificates is described below.

The format conversion can produce two types of output file. One is used when importing the converted certificate into a program, and the other is used when writing the converted certificate directly from a PC.

1. Obtain a root certificate (Class 2 Root CA).

2. Convert the root certificate to SharkSSL binary format.

   (For importing into a program)
   > SharkSSLParseCAList.exe xxxx.cer > starfield.c
   (For direct writing from a PC)
   > SharkSSLParseCAList.exe xxxx.cer –b xxxx.bin

3. Convert the client certificate and private key to SharkSSL binary format.

   (For importing into a program)
   > SharkSSLParseCert XXXX-certificate.pem.crt XXXX-private.pem.key > mycert.c
   (For direct writing from a PC)
   > SharkSSLParseCert XXXX-certificate.pem.crt XXXX-private.pem.key –b XXXX-certificate.bin

5.3.3 Registering the Certificate in the Wi-Fi Driver

To use the API to write the certificate to the Wi-Fi module, import the converted file into your project. For information on writing the certificate to the Wi-Fi module, refer to section 3, API Functions.

To write the converted certificate (binary file) to the Wi-Fi module directly from your PC, connect the PC to pins TX0 and RX0 of the Wi-Fi module via a USB-serial converter, then use AT commands to write the data. Set the baud rate to 115,200 bps.

The example below shows the AT command used to write the certificate to the Wi-Fi module.

(AT command example)

ATNSSLCERT=<certificate file name>,<certificate size>

Transmit the binary file within 30 seconds after issuing the above AT command.

Certificate file name: This is the certificate file name recorded in the Wi-Fi module. Set a name no more than 20 characters long.

Use “calist<number>.crt” for a CA list.

Use “cert<number>.crt” for a client certificate.

Certificate size: Set the binary data size (byte count).

6. Reference Documents

User's Manual: Hardware
(The latest versions can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News
(The latest information can be downloaded from the Renesas Electronics website.)

User's Manual: Development Tools
RX Family CC-RX Compiler User's Manual (R20UT3248)
(The latest versions can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.02	Mar. 15, 2021	—	First edition issued

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

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

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

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

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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