RX Family

R01AN4664EJ0113
Rev.1.13
Nov.5.21

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

RX65N Group

RX72N Group

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.

1.2.1 Connection with SX-ULPGN

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.

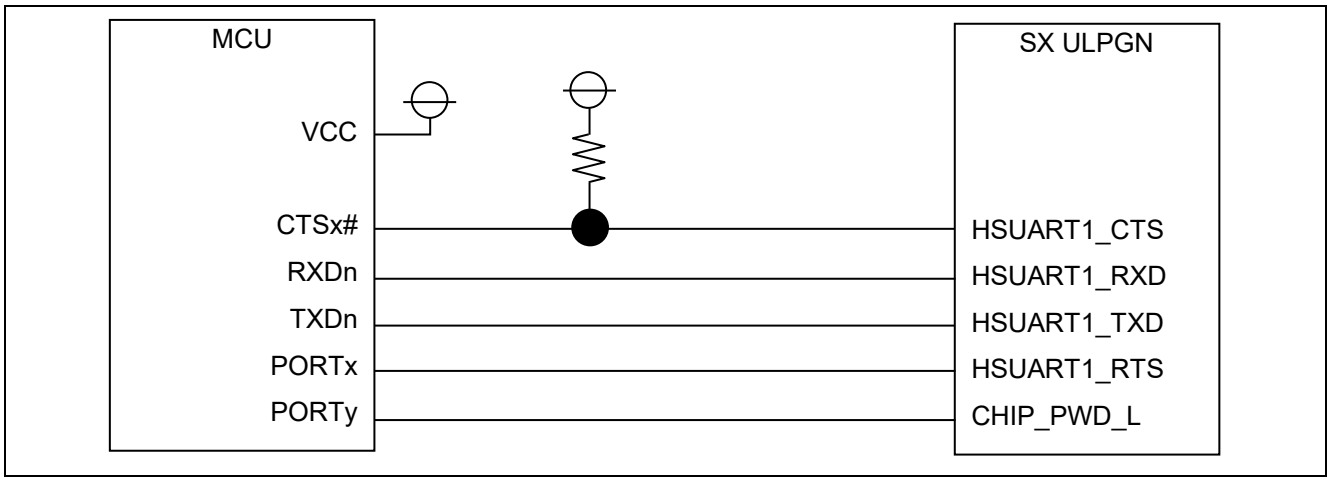

Figure 1.1 Example Connections for Single-Channel Communication Mode

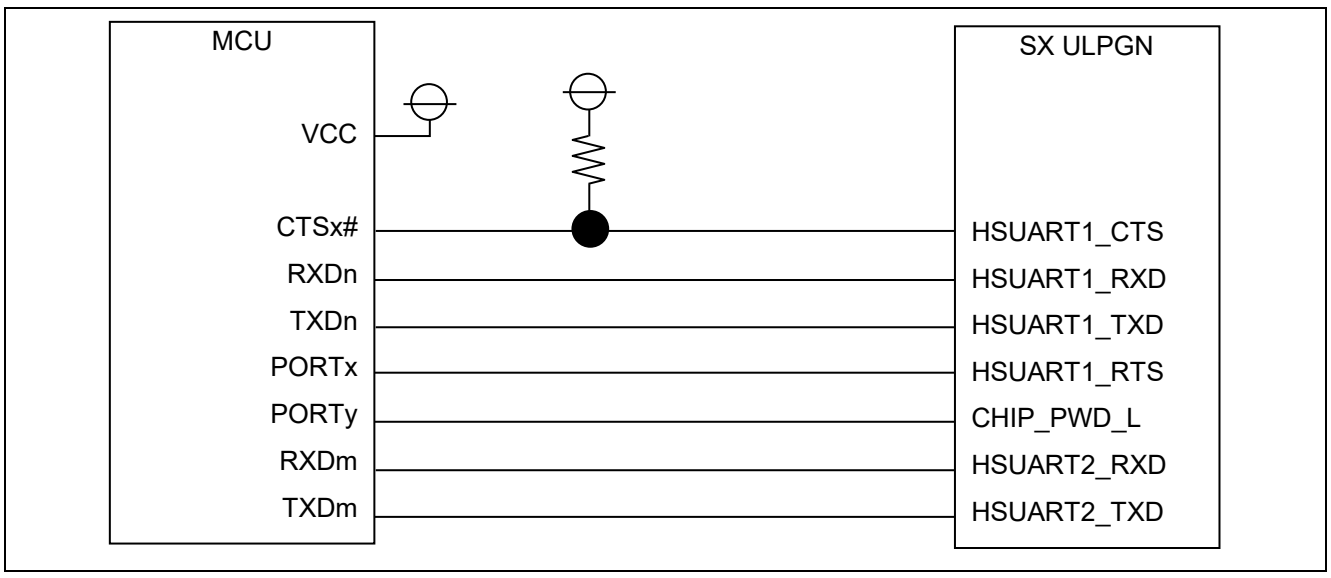

Figure 1.2 Example Connections for Two-Channel Communication Mode

1.2.2 Software configuration

Figure 1.3 shows the software configuration.

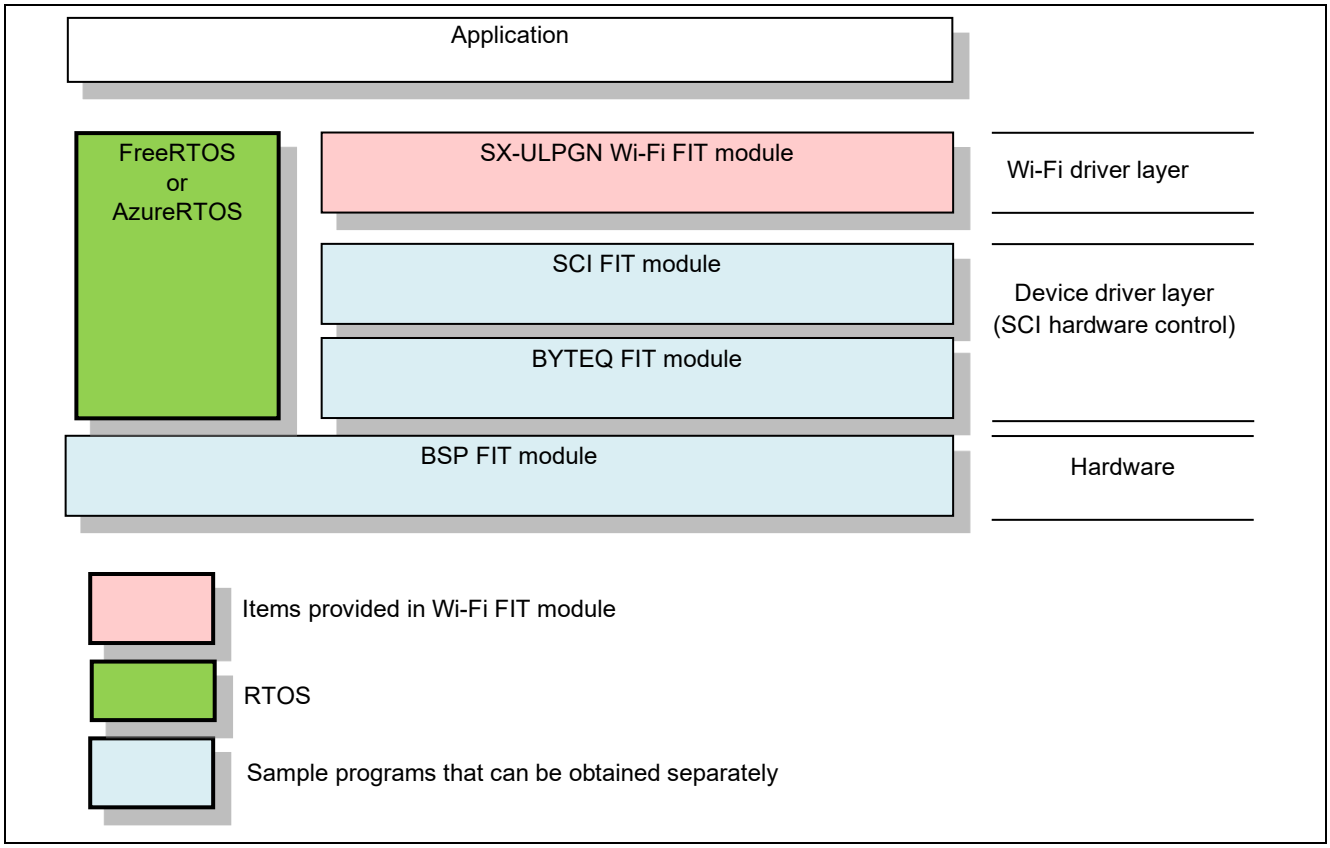

Figure 1.3 Software Configuration Diagram

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 FreeRTOS and AzureRTOS.

1.2.3 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 data.
R_WIFI_SX_ULPGN_ReceiveSocket()	Receives data.
R_WIFI_SX_ULPGN_ShutdownSocket()	Ends socket communication.
R_WIFI_SX_ULPGN_CloseSocket()	Closes a socket.
R_WIFI_SX_ULPGN_GetTcpSocketStatus()	Obtains a socket status.
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.
Function related to use of Wi-Fi module SSL functionality	
R_WIFI_SX_ULPGN_RequestTlsSocket ()	Allocate the created TCP socket for SSL 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.

1.2.4 Status Transitions

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

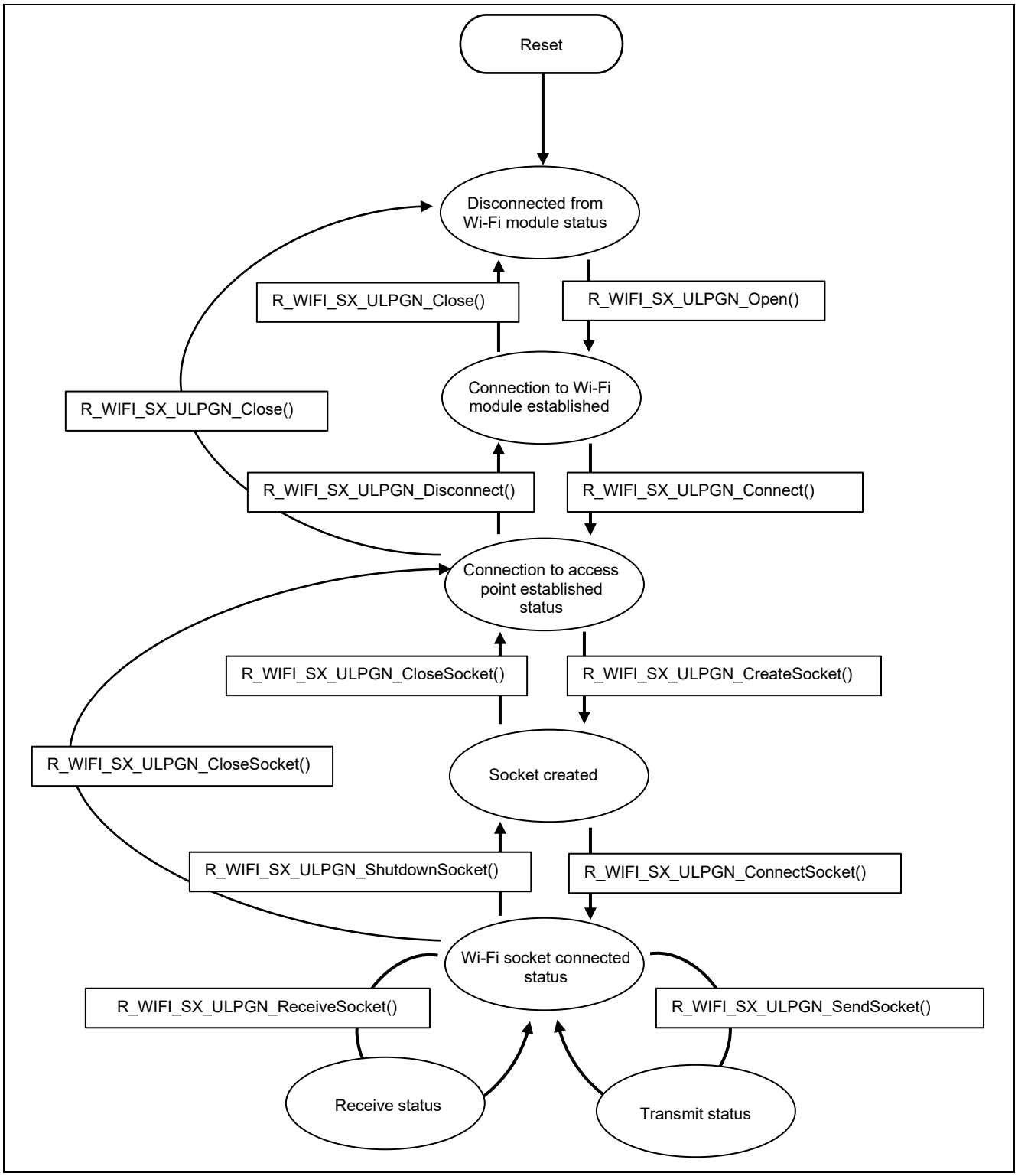

Figure 1.4 Status Transitions

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
- AzureRTOS

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 channel number assigned to HSUART1.
WIFI_CFG_SCI_SECOND_CHANNEL Note: The default is 1.	Specifies the SCI channel number assigned to HSUART2. If you specify the same number as WIFI_CFG_SCI_CHANNEL, it will operate in 1ch mode.
WIFI_CFG_SCI_INTERRUPT_LEVEL Note: The default is 4.	Sets the interrupt priority for the SCI module. Set according to the system.
WIFI_CFG_SCI_PCLK_HZ Note: The default is 60000000.	Specifies the SCI PLCK clock in Hz. Set according to the system. * Valid only in 1ch mode.
WIFI_CFG_SCI_USE_FLOW_CONTROL Note: The default is 1.	Enables or disables the hardware flow control of HSUART1.
WIFI_CFG_RESET_PORT Note: The default is D.	Set GPIO for PWD_L pin. Specify the port number with WIFI_CFG_RESET_PORT and WIFI_CFG_RESET_PIN. ex) PD0 #define WIFI_CFG_RESET_PORT D #define WIFI_CFG_RESET_PIN 0
WIFI_CFG_RESET_PIN Note: The default is 0.	
WIFI_CFG_RTS_PORT Note: The default is 2.	Set GPIO for RTS pin. Specify the port number with WIFI_CFG_RTS_PORT and WIFI_CFG_ RTS_PIN. ex) P22 #define WIFI_CFG_RTS_PORT 2 #define WIFI_CFG_ RTS_PIN 2
WIFI_CFG_ RTS_PIN Note: The default is 2.	
WIFI_CFG_CREATABLE_SOCKETS Note: The default is 4.	Set the number of sockets that can be created. The maximum number is 4. Set according to the system.
WIFI_CFG_SOCKETS_RECEIVE_BUFFER_SIZE Note: The default is 8192.	Sets the receive buffer size for the socket. Set according to the memory usage and data reception.
WIFI_CFG_USE_CALLBACK_FUNCTION Note: The default is 0.	Enables or disables the user callback function. 1 = enabled, 0 = disabled
WIFI_CFG_CALLBACK_FUNCTION_NAME Note: The default is NULL.	Register the user callback function name. See Chapter 4 for how to implement the callback function. This item is invalid when WIFI_CFG_USE_CALLBACK_FUNCTION is 0.

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 this FIT module, set the following. FreeRTOS:1 AzureRTOS:5

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	8,729 bytes	
	RAM	4,759 bytes	The size excluding the socket buffer (8192 * number of sockets).
	Max. stack size used	214 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.

```
/*  WiFi API error code */
typedef enum
{
    WIFI_SUCCESS              = 0,  // success
    WIFI_ERR_PARAMETER        = -1, // invalid parameter
    WIFI_ERR_ALREADY_OPEN     = -2, // already WIFI module opened
    WIFI_ERR_NOT_OPEN         = -3, // WIFI module is not opened
    WIFI_ERR_SERIAL_OPEN            = -4, // serial open failed
    WIFI_ERR_MODULE_COM       = -5, // cannot communicate WiFi module
    WIFI_ERR_NOT_CONNECT            = -6, // not connect to access point
    WIFI_ERR_SOCKET_NUM       = -7, // no available sockets
    WIFI_ERR_SOCKET_CREATE    = -8, // create socket failed
    WIFI_ERR_CHANGE_SOCKET    = -9, // cannot change socket
    WIFI_ERR_SOCKET_CONNECT   = -10,// cannot connect socket
    WIFI_ERR_BYTEQ_OPEN       = -11,// cannot assigned BYTEQ
    WIFI_ERR_SOCKET_TIMEOUT   = -12,// socket timeout
    WIFI_ERR_TAKE_MUTEX       = -13 // cannot take mutex
} wifi_err_t;

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

/* Query current socket status */
typedef enum
{
    ULPGN_SOCKET_STATUS_CLOSED = 0, // "CLOSED"
    ULPGN_SOCKET_STATUS_SOCKET,     // "SOCKET"
    ULPGN_SOCKET_STATUS_BOUND,            // "BOUND"
    ULPGN_SOCKET_STATUS_LISTEN,     // "LISTEN"
    ULPGN_SOCKET_STATUS_CONNECTED,  // "CONNECTED"
    ULPGN_SOCKET_STATUS_BROKEN,     // "BROKEN"
    ULPGN_SOCKET_STATUS_MAX         // Stopper
} sx_ulpgn_socket_status_t;

/* Error event for user callback */
typedef enum
{
    WIFI_EVENT_WIFI_REBOOT = 0,     // reboot WIFI
    WIFI_EVENT_WIFI_DISCONNECT,     // disconnected WIFI
    WIFI_EVENT_SERIAL_OVF_ERR,            // serial : overflow error
    WIFI_EVENT_SERIAL_FLM_ERR,            // serial : flaming error
    WIFI_EVENT_SERIAL_RXQ_OVF_ERR,  // serial : receiving queue overflow
    WIFI_EVENT_RCV_TASK_RXB_OVF_ERR,      // receiving task : receive buffer
overflow
    WIFI_EVENT_SOCKET_CLOSED,       // socket is closed
    WIFI_EVENT_SOCKET_RXQ_OVF_ERR   // socket : receiving queue overflow
} wifi_err_event_enum_t;
```

```
typedef struct
{
    wifi_err_event_enum_t event;    // Error event
    uint8_t socket_number;          // Socket number
} wifi_err_event_t;

/* AP scan result */
typedef struct
{
    uint8_t  ssid[33];              // SSID
    uint8_t  bssid[6];              // BSSID
    wifi_security_t security;       // kinds of security
    int8_t   channel;               // Channel
    int8_t   rssi;                      // RSSI
    uint8_t  hidden;                // Hidden channel
} wifi_scan_result_t;

/* IP configurations */
typedef struct
{
    uint32_t ipaddress;             // IP address
    uint32_t subnetmask;                // subnet mask
    uint32_t gateway;               // gateway
} wifi_ip_configuration_t;

/* Certificate information */
typedef struct {
    uint8_t  num_of_files;          // certificate number
    struct {
        uint8_t  file_name[20];         // certificate file name
    } cert[10];
} wifi_certificate_infomation_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 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 mutex*
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

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

Reentrant

No

Example

```
R_WIFI_SX_ULPGN_Open();
```

Special Notes:

If WIFI_ERR_SERIAL_OPEN occurs, execute R_WIFI_SX_ULPGN_Close().

3.2 R_WIFI_SX_ULPGN_Close()

This function closes 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 closes the Wi-Fi module.
If this function is executed while the access point is connected, the access point will be disconnected and the Wi-Fi module will be closed.

Reentrant

No

Example

```
R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_Close();
```

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	*DNS server IP address1 (0: Parameter invalid)*
dns_address2	*DNS server IP address2 (0: Parameter invalid)*

Return Values

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

Description

Sets the address specified by dns_address1 / dns_address2 to the DNS server address.

When R_WIFI_SX_ULPGN_Connect () is executed with DHCP disabled setting, the DNS server address set by this function is applied.
Call this function before executing R_WIFI_SX_ULPGN_Connect ()..

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Example

```
R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_SetDnsServerAddress(0xc0a80105, 0xc0a80106);
```

Special Notes:

None.

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 the structure that stores the scan results*
max_networks	*Maximum number of access points to store in ap_results*
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

```
wifi_scan_result_t scan_rslt[5];
uint32_t max_networks = 5;
uint32_t exist_ap_count;
uint32_t max_ap;

R_WIFI_SX_ULPGN_Scan(scan_rslt, max_networks, &exist_ap_count);
printf("Found access point(s) : %d\n", exist_ap_count);
if (exist_ap_count >= max_networks)
{
    max_ap = max_networks;
}
else
{
    max_ap = exist_ap_count;
}
for (int i = 0; i < max_ap; i++ )
{
    printf("  ---------------------\n");
    printf("  ssid     : %s\n", p[i].ssid);
    printf("  channel  : %d\n", p[i].channel);
    printf("  rssi     : %d\n", p[i].rssi);
    printf("  security : %d\n", p[i].security);
}
```

Special Notes:

None.

3.5 R_WIFI_SX_ULPGN_Connect()

This function connects to the specified access point.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_Connect (
        const uint8_t *ssid,
        const 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,WIFI_SECURITY_WPA2)*
dhcp_enable	*Automatic IP address assignment (0: Disabled, 1: Enabled)*
ip_config	*IP configuration structure pointer*

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 mutex*
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.

If dhcp_enable = 0, set the IP address in ip_config.

If dhcp_enable = 1, the IP address assigned by DHCP is stored in ip_config.

Reentrant

No

Example

```
int32_t sock;
uint32_t ipadr = 0xc0a8010a; /* 192.168.1.10 */
uint16_t port = 80;          /* Port 80 */
wifi_ip_configuration_t ip_cfg;


R_WIFI_SX_ULPGN_Open();

/* DHCP 有効の場合 */
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 1, &ip_cfg);

/* DHCP 無効の場合 */
ip_cfg.ipaddr = 0xc0a80003;  //192.168.0.3
ip_cfg.subnetmask = 0xffffff00;   //255.255.255.0
ip_cfg.gateway = 0xc0a80001;   //192.168.0.1
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 0, &ip_cfg);

sock = R_WIFI_SX_ULPGN_CreateSocket(WIFI_SOCKET_IP_PROTOCOL_TCP,
WIFI_SOCKET_IP_VERSION_4);
R_WIFI_SX_ULPGN_RequestTlsSocket(sock);
R_WIFI_SX_ULPGN_ConnectSocket(sock, ipadr, port, NULL);
```

Special Notes:

None.

3.6 R_WIFI_SX_ULPGN_Disconnect ()

This function disconnects the connecting 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 mutex*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function disconnects the connecting access point.

Reentrant

No

Example

```
int32_t sock;
wifi_ip_configuration_t ip_cfg;

R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 1, &ip_cfg);
R_WIFI_SX_ULPGN_Disconnect();
```

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	*Connecting to the access point*
-1	*Not connected to access point*

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

```
if (0 == R_WIFI_SX_ULPGN_IsConnected())
{
    printf("connected \n");
}
else
{
    printf("not connect \n");
}
```

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 (6 bytes)*

Return Values

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

```
uint8_t mac[6];

R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_GetMacAddress(mac);
printf("— MAC adr : %lx:%lx:%lx:%lx:%lx:%lx\r\n",
            mac[0],mac[1],mac[2],mac[3],mac[4],mac[5]);
```

Special Notes:

None.

3.9 R_WIFI_SX_ULPGN_GetIpAddress()

This function obtains the IP address assigned to the Wi-Fi module.

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 mutex*
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

This function obtains the IP address, subnet mask and gateway assigned to the Wi-Fi module and stores them in ip_config.

Reentrant

No

Example

```
wifi_ip_configuration_t ip_cfg;
R_WIFI_SX_ULPGN_GetIpAddress(&ip_cfg);
```

Special Notes:

None.

3.10 R_WIFI_SX_ULPGN_CreateSocket ()

This function creates a socket by specifying the socket type and IP type.

Format

int32_t R_WIFI_SX_ULPGN_CreateSocket (

uint32_t type,

uint32_t ip_version

)

Parameters

type *Socket type*
WIFI_SOCKET_IP_PROTOCOL : TCP
WIFI_SOCKET_IP_PROTOCOL_UDP : UDP

ip_version *IP version (WIFI_SOCKET_IP_VERSION_4)*

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

This function returns the number of the created socket as an integer value.

Reentrant

No

Example

```
int32_t sock_tcp;
int32_t sock_udp;
uint32_t ipadr = 0xc0a8010a; /* 192.168.1.10 */
uint16_t port = 80;          /* Port 80 */
wifi_ip_configuration_t ip_cfg;

R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 1, &ip_cfg);
Sock_tcp = R_WIFI_SX_ULPGN_CreateSocket(WIFI_SOCKET_IP_PROTOCOL_TCP,
WIFI_SOCKET_IP_VERSION_4);
Sock_udp = R_WIFI_SX_ULPGN_CreateSocket(WIFI_SOCKET_IP_PROTOCOL_UDP,
WIFI_SOCKET_IP_VERSION_4);
```

Special Notes:

None.

3.11 R_WIFI_SX_ULPGN_ConnectSocket ()

This function connects to the created socket.

Format

```
wifi_err_t R_WIFI_SX_ULPGN_ConnectSocket (
        int8_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 mutex*
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

Connect to the socket created by R_WIFI_SX_ULPGN_CreateSocket().
When R_WIFI_SX_ULPGN_RequestTlsSocket() is executed, it connects with SSL.
If you specify a socket that does not exist, WIFI_ERR_SOCKET_NUM is returned.

Reentrant

No

Example

```
int32_t sock;
uint32_t ipadr = 0xc0a8010a; /* 192.168.1.10 */
uint16_t port = 80;          /* Port 80 */
wifi_ip_configuration_t ip_cfg;


R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 1, &ip_cfg);
sock = R_WIFI_SX_ULPGN_CreateSocket(WIFI_SOCKET_IP_PROTOCOL_TCP,
WIFI_SOCKET_IP_VERSION_4);
R_WIFI_SX_ULPGN_RequestTlsSocket(sock);
R_WIFI_SX_ULPGN_ConnectSocket(sock, ipadr, port, NULL);
```

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,
        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	*Transmission timeout duration [ms] (not used)*

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 mutex*
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

This function sends the data stored in the *data* from the specified socket the number of bytes specified by *length*.

Reentrant

No

Example

```
int32_t xReturned;
xReturned = R_WIFI_SX_ULPGN_SendSocket(sock, buffer, sizeof(buffer), 1000);
```

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 (
        uint8_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]*

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 mutex*
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

This function receives *data_length* worth of data from the specified socket.
When a timeout occurs, the received data size at the time of timeout is returned.

Reentrant

No

Example

```
int32_t xReturned;
xReturned = R_WIFI_SX_ULPGN_ReceiveSocket(sock, buffer, sizeof(buffer), 1000);
```

Special Notes:

None.

3.14 R_WIFI_SX_ULPGN_ShutdownSocket ()

This function disconnects communication with the specified socket.

Format

wifi_err_t R_WIFI_SX_ULPGN_ShutdownSocket (

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 mutex*
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

This function disconnects communication with the specified socket.
The socket itself cannot be deleted. You can reconnect with R_WIFI_SX_ULPGN_ConnectSocket().

Reentrant

No

Example

```
int32_t sock;
wifi_ip_configuration_t ip_cfg;

R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 1, &ip_cfg);
sock = R_WIFI_SX_ULPGN_CreateSocket(WIFI_SOCKET_IP_PROTOCOL_TCP,
WIFI_SOCKET_IP_VERSION_4);
R_WIFI_SX_ULPGN_ConnectSocket(sock, ipadr, port, NULL);
R_WIFI_SX_ULPGN_ShutdownSocket(sock);
R_WIFI_SX_ULPGN_ConnectSocket(sock, ipadr, port, NULL);
```

Special Notes:

None.

3.15 R_WIFI_SX_ULPGN_CloseSocket ()

This function disconnects communication with the specified socket and deletes the socket.

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 mutex*
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

This function disconnects communication with the specified socket and deletes the socket.
If R_WIFI_SX_ULPGN_ShutdownSocket() has already been executed, the socket will be deleted.

Reentrant

No

Example

```
/* Socket disconnected */
R_WIFI_SX_ULPGN_ConnectSocket(sock, ipadr, port, NULL);
R_WIFI_SX_ULPGN_ShutdownSocket(sock);
R_WIFI_SX_ULPGN_CloseSocket(sock);

/* Socket undisconnected */
R_WIFI_SX_ULPGN_ConnectSocket(sock, ipadr, port, NULL);
R_WIFI_SX_ULPGN_CloseSocket(sock);
```

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 (

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

This function performs a DNS query to obtains the IP address of the specified domain.

Reentrant

No

Example

```
uint32_t ipadr;
R_WIFI_SX_ULPGN_DnsQuery("hostname", &ipadr);
```

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	*Failed to obtain mutex*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function pings the IP address specified by *ip_address*.
The parameters (*count*, *interval_ms*) specify the number of transmissions and the transmission interval.

Reentrant

No

Example

```
uint32_t ip_addr = 0xc8a8010a; /* 192.168.1.10 */

R_WIFI_SX_ULPGN_Ping(ip_addr, 1, 1000);
```

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 upper 2 bytes indicate the major version and the lower 2 bytes indicate the minor version.

Reentrant
No

Example
```
uint32_t ver;
ver = R_WIFI_SX_ULPGN_GetVersion();
printf("Version V%d.%2d\n", ((ver >> 16) & 0x0000FFFF), (ver & 0x0000FFFF));
```

Special Notes:
None.

3.19 R_WIFI_SX_ULPGN_RequestTlsSocket ()

This function allocates the created TCP socket for TLS communication.

Format

wifi_err_t R_WIFI_SX_ULPGN_RequestTlsSocket

uint8_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 allocates the TCP socket created by R_WIFI_SX_ULPGN_CreateSocket() for TLS communication.
Call R_WIFI_SX_ULPGN_ConnectSocket() before executing.

Reentrant

No

Example

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

```
int32_t sock;
uint32_t ipadr = 0xc0a8010a; /* 192.168.1.10 */
uint16_t port = 80;            /* Port 80 */
wifi_ip_configuration_t ip_cfg;

R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 1, &ip_cfg);
sock=R_WIFI_SX_ULPGN_CreateSocket(WIFI_SOCKET_IP_PROTOCOL_TCP,WIFI_SOCKET_IP_VERSION_4);
R_WIFI_SX_ULPGN_RequestTlsSocket(sock);
R_WIFI_SX_ULPGN_ConnectSocket(sock, ipadr, port, NULL);
```

Special Notes:

None.

3.20 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,
        const 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*
certificate_length	*Certificate size*

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	*Failed to obtain mutex*
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 certificate file name is set as follows depending 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.
The certificate data must be converted to binary format. See Chapter 5.3 for details.

Reentrant

不可

Example

```
const uint8_t cert_data[1053] =
{
    ...
};

const uint8_t ca_data[1053] =
{
    ...
};

cert_info_t cert_info;

/* Write certificate0 : cert0.crt */
R_WIFI_SX_ULPGN_WriteServerCertificate(0, 0, cert_data, sizeof(cert_data));

/* Write CA list0 : calist0.crt */
R_WIFI_SX_ULPGN_WriteServerCertificate(0, 1, ca_data, sizeof(ca_data));

/* Write certificate1 : cert1.crt */
R_WIFI_SX_ULPGN_WriteServerCertificate(1, 0, cert_data, sizeof(cert_data));

/* Write CA list1 : calist1.crt */
R_WIFI_SX_ULPGN_WriteServerCertificate(1, 1, ca_data, sizeof(ca_data));

/* Get certificate list */
/* cert_info.cert[0].file_name = cert0.crt */
/* cert_info.cert[1].file_name = calist0.crt */
/* cert_info.cert[2].file_name = cert1.crt */
/* cert_info.cert[3].file_name = calist1.crt */
R_WIFI_SX_ULPGN_GetServerCertificate(&cert_info);

/* Erase cert0.crt */
R_WIFI_SX_ULPGN_EraseServerCertificate(cert_info.cert[0].file_name);

/* Erase all */
R_WIFI_SX_ULPGN_EraseAllServerCertificate();
```

Special Notes:

None.

3.21 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 mutex*
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

```
See
R_WIFI_SX_ULPGN_WriteServerCertificate()
```

Special Notes:

None.

3.22 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	*Failed to obtain mutex*
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

```
See
R_WIFI_SX_ULPGN_WriteServerCertificate()
```

Special Notes:

None.

3.23 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	*Failed to obtain mutex*
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

```
See
R_WIFI_SX_ULPGN_WriteServerCertificate()
```

Special Notes:

None.

3.24 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	*Normal end*

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

```
uint32_t ip_addr = 0xc0a80105; /* 192.168.1.5 */

/* Link IP address to certificate ID0 */
R_WIFI_SX_ULPGN_SetCertificateProfile(0, ip_addr, NULL);

/* Link the server name to certificate ID1 */
R_WIFI_SX_ULPGN_SetCertificateProfile(1, 0, "ServerName");
```

Special Notes:

None.

3.25 R_WIFI_SX_ULPGN_GetTcpSocketStatus()

This function returns the status of the specified socket.

Format

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

Parameters

socket_number Socket number

Return Values

0	*close*
1	*socket*
2	*bound*
3	*listen*
4	*connected*
-1	*Specified socket does not exist*

Properties

Prototype declarations are contained in r_wifi_sx_ulpgn_if.h.

Description

This function returns the status of the socket specified by the parameter.

Reentrant

No

Example

```
int32_t sock_status;
int32_t sock;
wifi_ip_configuration_t ip_cfg;

R_WIFI_SX_ULPGN_Open();
R_WIFI_SX_ULPGN_Connect("ssid", "passwd", WIFI_SECURITY_WPA2, 1, &ip_cfg);
sock = R_WIFI_SX_ULPGN_CreateSocket(WIFI_SOCKET_IP_PROTOCOL_TCP,
WIFI_SOCKET_IP_VERSION_4);
sock_status = R_WIFI_SX_ULPGN_GetTcpSocketStatus(sock);
```

Special Notes:

None.

4. Callback Function

4.1 callback()

This function notifies the user application of a Wi-Fi module error.

Format

void * callback(

void * pevent

)

Parameters

pevent *Pointer to error information area*

Return Values

None.

Properties

This function is implemented by the user.

Description

Enable this API with the following configuration. The function name does not have to be "callback".

```
#define WIFI_CFG_USE_CALLBACK_FUNCTION  (1)
#define WIFI_CFG_CALLBACK_FUNCTION_NAME (wifi_callback)
```

Since the event is notified as a void pointer type, cast it to wifi_err_event_t type before referencing it.

```
void wifi_callback(void *p_args)
{
   wifi_err_event_t *pevent;
   pevent = (wifi_err_event_t *)p_args;

   switch(pevent->event)
   {
      case WIFI_EVENT_SERIAL_OVF_ERR:
         break;
. . .
   }
}
```

The notification events are as follows.

- WIFI_EVENT_SERIAL_OVF_ERR
  Reports that the SCI module has detected a receive overflow error.
- WIFI_EVENT_SERIAL_FLM_ERR
  Reports that the SCI module has detected a receive framing error.
- WIFI_EVENT_SERIAL_RXQ_OVF_ERR
  Reports that the SCI module has detected a receive queue (BYTEQ) overflow.
- WIFI_EVENT_RCV_TASK_RXB_OVF_ERR
  Reports that the FIT module has detected the overflow of the AT command receive buffer.
- WIFI_EVENT_SOCKET_RXQ_OVF_ERR
  Reports that the socket has detected a receive queue (BYTEQ) overflow.

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_SERIAL_OVF_ERR:
         break;
      case WIFI_EVENT_SERIAL_FLM_ERR:
         break;
      case WIFI_EVENT_SERIAL_RXQ_OVF_ERR:
         break;
      case WIFI_EVENT_RCV_TASK_OVF_ERR:
         break;
      case WIFI_EVENT_SOCKET_RXQ_OVF_ERR:
         switch(pevent->socket_number)
         {
             case 0:
                 break;
             case 1:
                 break;
             case 2:
                 break;
             case 3:
                 break;
         }
         break;
       default:
         break;
   }
}
```

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

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 2021.07
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.03.00
	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.1.13
Board used	Renesas RX65N Cloud Kit (product No.: RTK5RX65N0SxxxxxBE) RX72N Envision Kit (product No.: RTK5RX72N0C00000BJ)

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.

(4) Q: When SX-ULPGN is used in Transparent mode (1CH communication mode), an error log is output to the terminal software.

A: Please set the contents of r_wifi_sx_ulpgn_config.h as follows.
Ex) For RX72N Enviosion Kit

```
#define WIFI_CFG_SCI_CHANNEL (7)           // Use SCI7
#define WIFI_CFG_SCI_SECOND_CHANNEL (7)    // 2ndCh unused (use the same Ch as above)
#define WIFI_CFG_SCI_PCLK_HZ (120000000)   // SCI0-6,12 are PCLKB speeds,
                                           // SCI7-11 are PCLKA speeds
```

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 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.

The method of converting the format of certificates is described below.

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
1.00	Jul. 21, 2021	-	First edition issued
1.13	Nov. 5. 2021	-	Added Azure RTOS to the supported OS. 1.3.2 Software Description 2.2 Software requirements Figure 1.3 Application configuration diagram Table 2.4 Configuration options (r_bsp_config.h) 2.9. Corrected the return value style Update the table according to the latest config Table 2.1 Configuration options (r_wifi_sx_ulpgn_config.h) Add new API 3.25 R_WIFI_SX_ULPGN_GetTcpSocketStatus() Fixed API argument type 3.5 R_WIFI_SX_ULPGN_Connect() 3.11 R_WIFI_SX_ULPGN_ConnectSocket() 3.12 R_WIFI_SX_ULPGN_SendSocket() 3.13 R_WIFI_SX_ULPGN_ReceiveSocket() 3.14 R_WIFI_SX_ULPGN_ShutdownSocket() 3.16 R_WIFI_SX_ULPGN_DnsQuery() 3.19 R_WIFI_SX_ULPGN_RequestTlsSocket() 3.20 R_WIFI_SX_ULPGN_WriteServerCertificate() Added UDP related description 3.10 R_WIFI_SX_ULPGN_CreateSocket() 3.19 R_WIFI_SX_ULPGN_RequestTlsSocket() 3.1-3.25 Review Description and Example of all APIs 4.1 Review the description of the callback function Table 2.5 Updated code size table

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/.