Table of Contents

List of Figures

1 Introduction

1.1 Overview

This document describes the UART API Library for the RS9110-N-11-22/24/26/28 Wi-Fi modules of the Connect-io-n family from Redpine Signals. The library is delivered with the intent of providing an easy and quick way to program and configure the modules. This library has to be compiled along with the application(s) and HAL on the MCU to build a wireless communication system with the Wi-Fi module.

The library is based on the Software Programming Reference Manual (PRM) for these modules and forms a subset of the features mentioned in the Software PRM – please refer to the Release Notes of the library to know which commands are not supported. The APIs provide a simple way to setup a wireless connection and transmit/receive data from remote clients.

The library is platform independent and is written in simple C language. The document also discusses the requirements of the MCU"s HAL APIs and its memory.

1.2 Folder Structure

The folder structure and contents of this library are as follows:

1. `API_Lib`: The source code of the API Library for Wi-Fi over UART interface.
2. `Applications`: Contains sample applications for MCU and PC
    a. `MCU`: Contains dummy main.c and main.h files along with the rsi_global.h file which contains the global macros for the API Library.
    b. `PC (Linux/Windows)`: Contains example applications which can be executed on a PC connected to the Access Point and exchange data with the MCU+Wi-Fi Module system.
3. `Documentation\html`: Contains the documentation for the API Library"s source code in HTML form.

Each of these contents is explained in more detail in the subsequent sections.

2 API Library for Wi-Fi over UART Interface

This section discusses the API Library and the requirements of the HAL of the MCU.

2.1 API Library

The API Library provides APIs which are called by the Application of the MCU in order to configure the Wi-Fi module and also exchange data over the network. The `API_Lib\rsi_lib_api.c` file contains these APIs. The rest of the files with „rsi_lib_" prefix are related to the implementation of these APIs.

The APIs included in the library are listed below. Please refer to the HTML documentation for more details on each API like the parameters, return values, etc.

All the parameters to the APIs are INPUT parameters unless otherwise mentioned explicitly as OUTPUT parameter. All the APIs are non-blocking – the application has to call APIs to issue commands to the Wi-Fi module and then call the rsi_read_cmd_rsp API to read the response for the command. Application should retry giving the same command, only if the return value of the rsi_read_cmd_rsp is not RSI_ERROR_NO_RX_PENDING. In other words retry should be done only if the rsi_read_cmd_rsp returns error code or RSI_ERROR_INVALID_PKT_RECVD. All error codes are explained in section 2.1.28.

2.1.1 Set Band

```
API Prototype:
```

```
int16 rsi_band (uint8 band)
```

Description: Configures the band for the Wi-Fi module.

Parameters:

Band: 0 – 2.4GHz, 1 – 5GHz

Return value: 0 for success, 0xff for failure

Prerequisites: This is the first API to be called, to configure the band.

2.1.2 Init

```
API Prototype:
```

```
int16 rsi_init_baseband (void)
```

Description: Initializes the baseband and RF components of the Wi-Fi module.

Prerequisites: rsi_band should have been executed before calling this API.

2.1.3 Set number of scan results

```
API Prototype:
```

```
int16 rsi_scan_num (uint8 num)
```

Description: Configures the number of scan results returned by the module each time the rsi_scan_next API is called. This function is useful if the MCU has limited memory and cannot accommodate the complete list of scanned Access Points at one go.

This API can be skipped if the host has enough memory to store the scan results. Scan response structure is explained later in this document.

Parameters:

num: This parameter configures the number of scan results the module returns for the Scan and NextScan commands.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband should have been executed

2.1.4 Passive scan

```
API Prototype:
```

```
int16 rsi_scan_passive (uint16 *bitmap)
```

Description: Scans for Wi-Fi networks in Passive mode.

Paremeters:

bit_map: Parameter to configure for which channels passive scan is to be done.

For example, if only channel 1 and channel 4 are required to be scanned passively, then the value for bit_map is calculated as

Channel[n] …….	Channel[4]	Channel[3]	Channel[2]	Channel[1]
0	1	0	0	1

Decimal for <000..001001> is 9. Hence, bitmap value is 9.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband should have been executed.

2.1.5 Scan

```
API Prototype:
```

```
int16 rsi_scan (struct rsi_scanFrameSnd_s *scan_frame)
```

Description: Scans for Wi-Fi networks in Active mode.

Parameters: scan_frame : A pointer to the rsi_scanFrameSnd_s structure, which is explained below

```
typedef struct rsi_scanFrameSnd_s
{
  uint16 channel;
  uint8  ssid[SSID_LEN]; /* SSID_LEN is 32 bytes */
} rsi_scanFrameSnd_t;
```

Structure description

channel: 0- All channel scan, or a particular channel

Channel Number on which the scan has to be done.

Parameters for 2.4 GHz

Channel Number	chan_num parameter
All channels	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14

Table 1: Channel Parameters for 2.4 Ghz

Parameters for 5 GHz

Channel Number	chan_num parameter
All channels	0
36	1
40	2
44	3
48	4
52	5
56	6
60	7
64	8
100	9
104	10
108	11
112	12
116	13
120	14
124	15

128	16
132	17
136	18
140	19
149	20
153	21
157	22
161	23
165	24

Table 2: Channel Parameters for 5 Ghz

ssid : SSID of the AP to be scanned if it is in hidden mode.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband should have been executed.

2.1.6 Query number of scan results

API Prototype:

```
int16 rsi_query_scan_num (void)
```

Description: Requests the number of Access Points scanned by the module.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband and rsi_scan should have been executed.

2.1.7 Scan next WiFi networks

API Prototype:

```
int16 rsi_scan_next (void)
```

Description: Requests the next set of scanned Access Points. This API is valid only if the rsi_scan_num API has been used to configure the number of scan results to be returned by the module each time this API is called.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband, rsi_scan_num and rsi_scan should have been executed.

2.1.8 Query BSSIDs of scanned WiFi networks

API Prototype:

```
int16 rsi_query_bssid (void)
```

Description: Requests for the BSSIDs of the Access Points scanned by the module.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband and rsi_scan should have been executed.

2.1.9 Set Network type

API Prototype:

```
int16 rsi_setNetworkType (uint8 nwType, uint16 ibss_type,
uint16 channel_num)
```

Description: Sets the network type to Infrastructure, IBSS or IBSS Security

Paremeters:

nwType: 0-IBSS (Ah-hoc), 1- Infrastructure, 2- IBSS Security (WEP)

ibss_type: If nwType is „0", then it is applicable. 0 indicates IBSS Joiner and 1 indicates IBSS Creator

channel_num: This is valid only in Creator Mode and indicates the channel in which the IBSS has to be created. For Joiner mode, this parameter should be set to 0.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband and rsi_scan should have been executed.

2.1.10 Set Pre Shared key

API Prototype:

```
int16 rsi_set_psk (uint8 *psk)
```

Description: Sets the pre-shared key which is used while connecting to a secure Access Point.

Paremeters:

psk: Passphrase string (ASCII) .The maximum length of the PSK is 32 characters

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband and rsi_scan should have been executed

2.1.11Set Authentication Mode

API Prototype:

```
int16 rsi_set_auth_mode (uint8 auth_mode)
```

Description: Sets the authentication mode to Open or Shared Key Authentication. This command is required if the Access Point supports only Shared Key mode of authentication. This command should be issued before the Join command. This command is relevant only when the mode of encryption is WEP.

Paremeters:

auth_mode: 0 – Open, 1 – Shared Key

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband and rsi_scan should have been executed

2.1.12 Join

```
API Prototype:

int16 rsi_join (struct rsi_joinFrameSnd_s *jf)
```

Description: Connects to an Access Point.

Paremeters:

Jf: A pointer to the join frame structure, which is explained below

```
typedef struct rsi_joinFrameSnd_s

{

  uint8  txdataRate;

  uint8  txpowerLevel;

  uint8  ssid[SSID_LEN];

} rsi_joinFrameSnd_t;

Structure description:
```

ssid: The SSID name of the network. The maximum length of the SSID name is 32 characters. This can be the SSID of the Access Point or Client to which the module has to connect to in Infrastructure or IBSS Joiner modes respectively. It can also be the SSID of the IBSS that is to be created by the module, according to the inputs to the `rsi_setNetworkType` command.

txpowerLevel: This fixes the Transmit Power level of the module. This value can be set as follows:

0 – Low power (7dBm)

1 – Medium power (10dBm)

2 – High power (16 to 17dBm)

TxdataRate: Rate at which the data has to be transmitted. Refer to the table below for the various data rates and the corresponding values. For Channel 14, only 11b rates (1, 2, 5.5 and 11 Mbps) are allowed.

Data Rate (Mbps)	**Value of uTxDataRate**
Auto-rate	0
1	1
2	2
5.5	3
11	4
6	5

Data Rate (Mbps)	Value of uTxDataRate
9	6
12	7
18	8
24	9
36	10
48	11
54	12
MCS0	13
MCS1	14
MCS2	15
MCS3	16
MCS4	17
MCS5	18
MCS6	19
MCS7	20

Table 3: Data Rate Parameter

This structure in the code may have other fields related to join, but these are the parameters that are used for rsi_join command.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband and rsi_scan should have been executed successfully.

2.1.13 Disassociate

```
API Prototype:
```

```
int16 rsi_disconnect (void)
```

Description: Disconnects the Wi-Fi connection.

Return value: 0 for success 0xff for failure

Prerequisites: rsi_band, rsi_init_baseband, rsi_scan and rsi_join should have been executed successfully.

2.1.14Power Modes and commands

The RS9110-N-11-2X module supports three power modes with the UART interface. The Host can switch among the power modes using the rsi_ `powermode` commands depending on the Wi-Fi connection status as defined in this section.

The power modes supported by the RS9110-N-11-2X module for UART interface are classified based on the Host"s capability to negotiate with RS9110-N-11-2X and the Wi-Fi connection status.