Redpine SAPI Porting Guide

Version 1.5

May 2020

Redpine Signals, Inc.

2107 North First Street, Suite #540, San Jose, California 95131,

United States of America.

Phone: +1-408-748-3385, Fax: +1-408-705-2019

Email: sales@redpinesignals.com

Website: www.redpinesignals.com

Table of Contents

About this Document

This document explains the procedure to port Redpine Wireless SAPIs to a host platform. This guide is applicable to the RS9116W WiSeConnect and the RS14100 WiSeMCU product line

1 Overview of Wireless SAPIs

Redpine software release package contain wireless library/APIs to facilitate user application development. This document contains description about wireless API's which are needed to be ported on host platform to use WiSeConnect™ / nLink™ / WiseMCU™ Wireless libraries.Hardware abstraction layer contains the platform specific functions which are supposed to be ported. OS abstraction layer contains the OS specific functions which are supposed to be ported if OS is required.

2 Creation of a project for porting

This section explains how to create a new project for porting the APIs on any host platform. This project uses a binary format to communicate with the module.
The user can approach porting in many ways but before porting, it is recommended to follow the below general guidelines.

1. Decide on the interface and modes for your use case.
2. Import the necessary project files to your development environment. It is recommended to create a blank project to simplify porting.
3. Verify that the drivers can compile and can be built on their own. They should build even if the HAL is not implemented. If this does not work, configure including headers, compiler macros, and other settings as necessary until the drivers are successfully built.
4. Implement the HAL and verify that whether it is working by running some of the examples provided in host/ sapis/examples/.
5. Integrate the implemented and tested HAL to your application.

2.1 Interfaces supported

The module supports SPI, UART, USB, and USB-CDC interfaces. The same APIs can be used. This document explains the SPI and UART but for USB and USB-CDC, the steps will be almost similar.
For communicating with a host application CPU, RSI_SPI_INTERFACE macro should be defined for SPI interface. Search RSI_SPI_INTERFACE macro in the driver source codes to acquire more information on the SPI uses. This macro can be defined in the compiler symbols.
Similarly, to enable the UART interface, it defines RSI_UART_INTERFACE.

2.2 Supported modes

The supported modes are outlined below:

- WLAN Only
- BLE Only
- BT Only
- WLAN Station + BLE
- WLAN Station (TCP/IP bypass) +BT

2.3 Compile Time Macros

The RS9116 drivers require some macros to be defined during compilation. The macros needed are dependent on the desired features or interfaces. The macros and their descriptions are listed below:

Parameter	Description
RSI_UART_INTERFACE	This enables the UART interface. It cannot be defined at the same time as RSI_SPI_INTERFACE
RSI_SPI_INTERFACE	This enables SPI Interface
RSI_WLAN_ENABLE	This enables WLAN support
RSI_BT_ENABLE	This enables Bluetooth Classic support
RSI_BLE_ENABLE	This enables Bluetooth Low Energy Support. Also, it requires RSI_BT_ENABLE
RSI_SAMPLE_HAL	This uses the sample HAL definitions defined in include/ rsi_hal.h. If not defined, then use the definitions present in the user-created hal/rsi_hal.h
LINUX_PLATFORM	This is used for porting to Linux platform.

Parameter	Description
WINDOWS	This is used for porting to Windows. It enables Windows HAL porting changes
RSI_VCC_1P8_V	This enables sdio pads operating voltage and sel-18 NPSS gpio bit for 1.8v supply for SDIO interface.
RSI_M4_INTERFACE	This is used only for WISE_MCU products.
RSI_ENABLE_DEBUG_PRINT	This is used to enable debug prints in case of linux interface.
RSI_SDIO_INTERFACE	This is used to enable SDIO support.
RSI_WAC_MFI_ENABLE	This macro is used by RSI_HOMEKIT_APP.
RSI_IPV6_ENABLE	This is used to enable IPV6 type.
ASYNC_MQTT	This Macro is used to run Async sapis of MQTT.
RSI_HOMEKIT_APP	This Macro is used to enable Homekit applications.
RSI_PUF_ENABLE	If this Macro is enabled,then all the Puf related API's are enabled.
RSI_CRYPTO_ENABLE	Enabling this Macro,will include the crypto library.
ROM_WIRELESS	This macro is needed in case of M4 interface.
APIS_ROM	This macro is used in case of M4 interface.

1. The above mentioned macros should define in the following project space.
   Project → Options for target → C/C++ → Preprocessor symbols → Define
2. These macros are only needed to be defined. Assigning a value to it is not mandatory

2.4 Directory structure

This package contains the following folders:

- **Build:** This folder contains Makefile to compile all the APIs for x86 platforms.
- **Docs:** This folder contains API guide document which describes the usage of APIs.
- **Driver:** This folder contains core driver APIs.

Examples
This folder contains different example applications along with App notes for each example. A user can use these applications and can develop the required applications accordingly.

Hal: This folder contains HAL APIs which the user needs to port on to the host Platform
Include: This folder contains all the header files
Nwk: This folder contains networking related APIs
Wlan: This folder contains WLAN related APIs
bt_ble: This folder contains BT Classic and BLE related APIs
common: This folder contains common files for BLE, BT, WLAN.

2.5 Required Include Directories

In order to build the RS9116 drivers, your build system may require specifying the path to RS9116 header files. The following include paths must be specified as follows:
All Interfaces:

```
include/
rom/
examples/utilities
```

TCP Applications:

```
nwk/applications/
nwk/applications/http_server/
nwk/applications/mqtt_client/src/
nwk/applications/mqtt_client/src/MQTTPacket/src/
```

AWS_IOT Applications:

```
host/sapis/nwk/applications/aws_sdk/platforms/inc
host/sapis/nwk/applications/aws_sdk/include
host/sapis/examples/utilities/certificates
```

ⓘ Include the above header file directories in the following project space.
Project → Options for target → C/C++ → Include Paths

2.6 Steps to create a project

For creating a simple project, follow the below steps:

2.6.1 WLAN only project

1. Copy/ Add all the files present in the following sapis folder -

```
driver/
hal/
include/
nwk/
wlan/
common/
rom/
```

2. Port the HAL APIs present in the HAL folder based upon the platform. Please refer to the section **3 Hardware Abstraction Layer**.
3. Copy **tcp_client** folder present in the path - **sapis/examples/wlan/.**
4. Configure the parameters in **rsi_tcp_client.c** file as explained in **RS9116_RS14100_Wireless_SAPI_Examples_vx.x.pdf** present in the Docs folder**.**
5. Include the respective example path in the following project space.
Project → Options for target → C/C++ → Include Paths
6. Define **RSI_WLAN_ENABLE macro.**
7. Build and run the application.

2.6.2 For aws_iot Applications :

1.For compiling aws_iot applications,include the files,present in below path to group,examples.

host/sapis/nwk/applications/aws_sdk/src

host/sapis/nwk/application.

2.Also,include the respective example path in the following project space.

Project → Options for target → C/C++ → Include paths

3.In Define,include **RSI_WLAN_ENABLE,RSI_UART_INTERFACE,RSI_SAMPLE_HAL WINDOWS RSI_STM32(**For STM32 Platform**)**

4.Build and run the Application.

2.6.3 BT only project

1. Copy/Add all the files present in the following sapis folder -

```
driver/
hal/
include/
nwk/
bt_ble/
common/
rom/
```

2. Port the HAL APIs present in the HAL folder based upon the platform. Please refer to the section **3 Hardware Abstraction Layer**.
3. Copy **bt_ssp_test_app** folder present in the following path - **sapis/examples/bt/.**
4. Configure the parameters in**rsi_bt_ssp_test_app.c** file as explained in the **RS9116_RS14100_Wireless_SAPI_Examples_vx.x.pdf** present in the Docs folder.
5. Include the respective example path in the following project space.
   Project → Options for target → C/C++ → Include Paths
6. Define **RSI_BT_ENABLE** and **RSI_BLE_ENABLE** macros.
7. Build and run the application.

2.6.4 BLE only project

1. Copy/Add all the files present in the following sapis folder -

```
driver/
hal/
include/
nwk/
bt_ble/
common/
rom/
```

2. Port the HAL APIs present in the HAL folder based upon the platform. Please refer to the section **3 Hardware Abstraction Layer**.
3. Copy **simple_chat** folder present in the following path - **sapis/examples/ble/.**
4. Configure the parameters in **rsi_ble_simple_chat.c** file as explained in the **RS9116_RS14100_Wireless_SAPI_Examples_vx.x.pdf** present in the Docs folder.
5. Include the respective example path in the following project space.
   Project → Options for target → C/C++ → Include Paths
6. Define **RSI_BT_ENABLE** and **RSI_BLE_ENABLE** macros.
7. Build and run the application.

2.6.5 WLAN+BT project

1. Copy/ Add all the files present in the following sapis folder -

```
driver/
hal/
include/
nwk/
wlan/
bt_ble/
common/
rom/
```

2. Port the HAL APIs present in the HAL folder based upon the platform. Please refer to the section **3 Hardware Abstraction Layer**.
3. Copy **wlan_bt_bridge** folder present in the following path - **sapis/examples/wlan_bt/.**
4. Configure the parameters in **rsi_bt_app.c** and **rsi_wlan_app.c** file as explained in the **RS9116_RS14100_Wireless_SAPI_Examples_vx.x.pdf** present in the Docs folder.
5. Include the respective example path in the following project space.
   Project → Options for target → C/C++ → Include Paths
6. Define **RSI_WLAN_ENABLE** and **RSI_BT_ENABLE** macros.
7. Build and run the application.

2.6.6 WLAN+BLE project

1. Copy/ Add all the files present in the following sapis folder -

```
driver/
hal/
include/
nwk/
wlan/
bt_ble/
common/
rom/
```

2. Port the HAL APIs present in the HAL folder based upon the platform. Please refer to the section **3 Hardware Abstraction Layer**.
3. Copy **wlan_ap_ble_bridge** folder present in the following path **sapis/examples/wlan_ble/.**
4. Configure the parameters in**rsi_ble_app.c** and **rsi_wlan_ap_app.c** file as explained in the **RS9116_RS14100_Wireless_SAPI_Examples_vx.x.pdf** present in the Docs folder**.**
5. Include the respective example path in the following project space.
   Project → Options for target → C/C++ → Include Paths
6. Define **RSI_WLAN_ENABLE**, **RSI_BT_ENABLE** and **RSI_BLE_ENABLE** macros.
7. Build and run the application.

2.6.7 **Procedure for host with linux platform** :

1.Port the sapis into the PC.

2.In Makefile,change the tool chain path according to the architecture of host.

3.Build the application by using the command : "make clean ; make"

4.Run the application by using, . ". /rsi_wc_app"

3 Hardware Abstraction Layer

This Section explains the HAL APIs which are expected to be ported on host platform to use Wireless Library.

This document focuses on the SPI,SDIO and UART interfaces. RS9116W WiSeConnect acts as an SPI/SDIO slave for control and data transfer. The Appendix A shows how to get the simple example to study porting of the SPI/SDIO slave interface of RS9116W WiSeConnect with Spansion /RS12100 board and UART interface on Windows.For more information on RS9116 WiSeConnect and Spansion MB9BF568NBGL, refer to the RS9116 WiSeConnect and Spansion MB9BF568NBGL datasheets respectively.

The HAL related files are available in the following path:

RS9116.NBZ.WC.GEN.OSI.x.x.x\host\sapis\platforms\<target>\<interface>\hal

example :target is RS12100,STM32

interface is SDIO,SPI,UART

3.1 SPI interface handling API

This section contains API's used by Wireless library to perform SPI transfer to/from the module.
The RS9116 operates in SPI mode 0. That is, clock polarity and phase should both be 0. The Chip select is active low.
The driver uses **rsi_spi_transfer** function to send and receive data on SPI. In this function, platform specific SPI transfer function is supposed to be called.

3.1.1 rsi_spi_transfer

Source File: rsi_hal_mcu_spi.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int16_t rsi_spi_transfer
(
        uint8_t *tx_buff,
        uint8_t *rx_buff,
        uint16_t transfer_length,
        uint8_t mode
)
```

Description
This API is used by Wireless Library to perform SPI transfer from/to the module.

Parameters

Parameter	Description
tx_buff	This is the pointer to buffer containing data to be sent to the module. This buffer can be null, if it is receive only operation.
rx_buff	This is the pointer to buffer holding data received from the module. This buffer can be null, if it is a transfer only operation.
transfer_length	This is the length of the transfer
mode	This is the mode of the transfer 0 – 8 bit mode

Return Values
0 – on Success

3.2 SDIO interface handling API

This section contains API's used by Wireless library to perform SDIO transfer to/from the module.
The driver uses **RSI_SDIOH_WriteCommandCmd52** function to send data in byte by byte format and **RSI_SDIOH_ReadCommandCmd52** receive data in byte by byte format on SDIO.

and **RSI_SDIOH_ByteBlockWriteCmd53** function to send data in block format and **RSI_SDIOH_ByteBlockReadCmd53** receive data in block format on SDIO

Source File: rsi_sdioh.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\platforms\RS12100\source\Peripheral_Library\driver\src

Prototype

```
error_t RSI_SDIOH_WriteCommandCmd52(SMIH_CARD_CONFIG_T *pSmihConfig,uint32_t Argument)
```

Description
This API is used by Wireless Library to perform SDIO transfer to the module.

Parameters .

Parameter	Description
pSmihConfig	Pointer to the card information structure
Argument	Argument to the command

Return Values
0 – on Success

Prototype

```
error_t RSI_SDIOH_ReadCommandCmd52(SMIH_CARD_CONFIG_T *pSmihConfig,uint32_t Argument)
```

Description
This API is used by Wireless Library to perform SDIO receive from the module .

Parameters

Parameter	Description
pSmihConfig	Pointer to the card information structure
Argument	Argument to the command

Return Values
0 – on Success

Prototype

```
error_t RSI_SDIOH_ByteBlockWriteCmd53(SMIH_CARD_CONFIG_T *pSmihConfig,uint8_t *pData ,uint32_t Addr)
```

Description
This API is used by Wireless Library to perform SDIO transfer to the module .

Parameters

Parameter	Description
pSmihConfig	Pointer to the smih config struct
pData	Pointer to the buffer data to write
Addr	Address to write the data

Return Values
0 – on Success

Prototype

```
error_t RSI_SDIOH_ByteBlockReadCmd53(SMIH_CARD_CONFIG_T *pSmihConfig,uint8_t *pData  ,uint32_t Addr)
```

Description
This API is used by Wireless Library to perform SDIO receive from the module.

Parameters

Parameter	Description
pSmihConfig	Pointer to the smih config struct
pData	Pointer to the buffer data to write
Addr	Address to write the data

Return Values
0 – on Success

3.3 UART interface handling API

This section explains API's used by Wireless library to perform UART interface with module. Following is the list of UART macros to be set for interface with module.

Parameter	Description
RSI_UART_DEVICE	This is to set UART device port
BAUDRATE	This is to set UART Baud rate
RSI_PRE_DESC_LEN	This is to put Pre descriptor length

Parameter	Description
UART_HW_FLOW_CONTROL	This enables UART hardware flow control. 0 - disable 1- Enable
RSI_FRAME_DESC_LEN	This is the frame descriptor length
RSI_SKIP_CARD_READY	This is the skip card ready if it is in UART mode
RSI_USB_CDC_DEVICE	This is the UART device or USB-CDC device 0-UART 1-USB-CDC

3.3.1 rsi_uart_send

Source File: rsi_hal_mcu_uart.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int16_t rsi_uart_send(uint8_t *ptrBuf, uint16_t bufLen)
```

Description
This API is used by Wireless Library to perform UART send from/to the module.

Parameters

Parameter	Description
ptrBuf	This is the pointer to the buffer with the data to be sent / received.
bufLen	This is the number of bytes to be sent

Return Values
0 – on Success

3.3.2 rsi_uart_recv

Source File: rsi_hal_mcu_uart.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int16_t rsi_uart_recv(uint8_t *ptrBuf, uint16_t bufLen)
```

Description
This API is used by Wireless Library to perform UART receive from/to the module.

Parameters

Parameter	Description
ptrBuf	This is the pointer to the buffer with the data to be sent / received
bufLen	This is the number of bytes to be received

Return Values
0 – on Success

3.3.3 rsi_uart_byte_read

Source File: rsi_hal_mcu_uart.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint8_t rsi_uart_byte_read(void)
```

Description
This API is used to read the byte data from the module through UART interface.

Parameters
None

Return Values
Read character

3.3.4 rsi_uart_init

Source File: rsi_hal_mcu_uart.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int32_t rsi_uart_init(void)
```

Description
This API is used by Wireless Library to perform initialization of UART interface with the module.

Parameters
None

Return Values
0 – on Success
!=0 – on Failure

3.3.5 rsi_uart_deinit

Source File: rsi_hal_mcu_uart.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int32_t rsi_uart_deinit(void)
```

Description
This API is used by Wireless Library to perform de initialization of UART interface with the module.

Parameters
None

Return Values
0 – on Success
!=0 – on Failure

3.4 Interrupt handling API

This section contain descriptions about APIs related to interrupts need to be ported on host platform.

3.4.1 rsi_hal_intr_config

Source File: rsi_hal_mcu_interrupt.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_config(
void (*rsi_interrupt_handler)())
```

Description
This API is used by Wireless Library to configure and to receive packet pending interrupt from the module. By default, the RS9116 interrupt line is active high.**Parameters**

Parameter	Description
rsi_interrupt_handler	This is a pointer to a function that should be called in interrupt handler. In this function, the wireless library will perform specific handling operation.

Return Values
None

3.4.2 rsi_hal_intr_mask

Source File: rsi_hal_mcu_interrupt.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_mask(void)
```

Description
This API is used by Wireless Library to mask / disable received packet pending interrupt from the module.

Parameters
None

Return Values
None

3.4.3 rsi_hal_intr_unmask

Source File: rsi_hal_mcu_interrupt.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_unmask(void)
```

Description
This API is used by Wireless Library to unmask received packet pending interrupt from the module.

Parameters
None

Return Values
None

3.4.4 rsi_hal_intr_clear

Source File: rsi_hal_mcu_interrupt.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_intr_clear(void)
```

Description
This API is used by Wireless Library to clear receive packet pending interrupt from the module, after receiving pending packet from the module.

Parameters
None

Return Values
None

3.4.5 rsi_hal_intr_pin_status

Source File: rsi_hal_mcu_interrupt.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint8_t rsi_hal_intr_pin_status(void)
```

Description
This API is used by Wireless Library to check the status of interrupt pin, to check whether packet pending from module or not.

Parameters
None

Return Values
None

3.5 GPIO Port handling API

This section explains descriptions about APIs related to GPIO which are needed to be ported on the host platform used by Wireless Library.
These GPIOs are used to reset the module, power save the mode and for other handshakes.
These are provided in **rsi_hal_mcu_ioports.c** file in HAL folder.

3.5.1 rsi_hal_config_gpio

Source File: rsi_hal_mcu_ioports.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_config_gpio(
uint8_t gpio_number,
uint8_t mode,
uint8_t value)
```

Description
This API is used by Wireless Library to configure GPIOs on host platform which are connected to the module.
Following is the list of GPIOs used by Wireless Library.

GPIO Numbers	Description
RSI_HAL_RESET_PIN	This is the GPIO to reset WiSeConnect Module
RSI_HAL_MODULE_INTERRUPT_PIN	This is the GPIO to receive packet pending interrupt
RSI_HAL_WAKEUP_INDICATION_PIN	This is the GPIO to receive module wakeup from power save indication
RSI_HAL_SLEEP_CONFIRM_PIN	This is the GPIO to give sleep confirmation to the module to go to sleep in power save

Table 1: GPIO PIN Mapping

RSI_HAL_SLEEP_CONFIRM_PIN and RSI_HAL_MODULE_WAKEUP_PIN are the only pins used in low power modes and are not required to bring up the module
User can change the GPIO macro definition according to the Host GPIO port numbers.

Parameters

Parameter	Description
gpio_number	This is the unique GPIO number (constant) used to differentiate GPIO's used by Wireless Library. HAL layer can map these GPIO number to the actual GPIO number/ports used in the host platform. Refer **Table 1: GPIO PIN Mapping** for GPIO number
mode	This is the bit map used to configure GPIO. BIT(0): 0 – Configure GPIO in input mode 1 – Configure GPIO in output mode BIT(1-7): Reserved
value	This is the default value to drive on GPIO, if GPIO configured in output mode. 1. Low 2. High

Return Values
None

3.5.2 rsi_hal_set_gpio

Source File: rsi_hal_mcu_ioports.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_set_gpio(uint8_t gpio_number)
```

Description
This API is used by Wireless Library to get the driver value (1) on a specified GPIO configured in output mode.

Parameters

Parameter	Description
gpio_number	This is a unique GPIO number (constant) used to differentiate GPIOs used by Wireless Library. HAL layer can map these GPIO number to the actual GPIO number/ports used in the host platform. Refer **Table 1: GPIO PIN Mapping** for GPIO number

Return Values
None

3.5.3 rsi_hal_get_gpio

Source File: rsi_hal_mcu_ioports.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint8_t rsi_hal_get_gpio(uint8_t gpio_number)
```

Description
This API is used by Wireless Library to get the value driven on a specified GPIO configured in input mode.

Parameters

Parameter	Description
gpio_number	This is the unique GPIO number (constant) used to differentiate GPIOs used by Wireless Library. HAL layer can map these GPIO number to the actual GPIO number/ports used in the host platform. Refer **Table 1: GPIO PIN Mapping** for GPIO number.

Return Values
0 – Low
1 – High

3.5.4 rsi_hal_clear_gpio

Source File: rsi_hal_mcu_ioports.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_hal_clear_gpio(uint8_t gpio_number)
```

Description
This API is used by Wireless Library to get the driver value (0) on a specified GPIO configured in output mode.

Parameters

Parameter	Description
gpio_number	This is the unique GPIO number (constant) used to differentiate GPIOs used by Wireless Library. HAL layer can map these GPIO number to the actual GPIO number/ports used in the host platform. Refer **Table 1: GPIO PIN Mapping** for GPIO number.

Return Values
None

3.6 Random number generation API

This section explains the API used by wireless Library to read random numbers.

3.6.1 rsi_get_random_number

Source File: rsi_hal_mcu_random.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint32_t rsi_get_random_number(void)
```

Description
This API is used by Wireless Library to get the random numbers.

Parameters
None

Return Values
32 bit random number

3.7 Timer handling API

This section explains APIs used by wireless Library to perform timer handling.

3.7.1 rsi_timer_start

Source File: rsi_hal_mcu_timer.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int32_t rsi_timer_start(
uint8_t timer_node,
uint8_t mode,
uint8_t type,
uint32_t duration,
void (*rsi_timer_expiry_handler)(void))
```

Description
This API is used by Wireless Library to start the timer.

Parameters

Parameter	Description
timer_node	This is the unique timer number
mode	This is the millisecond timer/ microsecond timer
type	This is the single shot timer/ period timer
duration	This is the time out value
rsi_timer_expiry_handler	This is the handler which should be called on timeout

Return Values
0 – on Success
!=0 – on Failure

3.7.2 rsi_timer_stop

Source File: rsi_hal_mcu_timer.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
int32_t rsi_timer_stop(
uint8_t timer_node)
```

Description
This API is used by Wireless Library to stop the timer.

Parameters

Parameter	Description
timer_node	This is the unique timer node number

Return Values

0 – on Success
!=0 – on Failure

3.7.3 rsi_timer_read

Source File: rsi_hal_mcu_timer.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint32_t rsi_timer_read(
uint8_t timer_node)
```

Description
This API is used by Wireless Library to read the timer count.

Parameters

Parameter	Description
timer_node	This is the unique timer node number

Return Values
Read timer value

3.7.4 rsi_delay_ms

Source File: rsi_hal_mcu_timer.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_delay_ms(uint32_t delay_ms
```

Description
This API is used by Wireless Library to have delay in milliseconds.

Parameters

Parameter	Description
delay_ms	This is the delay in milliseconds

Return Values
None

3.7.5 rsi_delay_us

Source File: rsi_hal_mcu_timer.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
void rsi_delay_us(uint32_t delay_us
```

Description
This API is used by Wireless Library to have delay in micro seconds.

Parameters

Parameter	Description
delay_ms	This is the delay in micro seconds

Return Values
None

3.7.6 rsi_hal_gettickcount

Source File: rsi_hal_mcu_timer.c
Path: RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\hal

Prototype

```
uint32_t rsi_hal_gettickcount(void)
```

Description
This API is used by Wireless Library to get the tick count delay in milli seconds from systic ISR

Parameters
None

Return Values
32-bit tick count valu

4 OS Interface Layer

WiSeConnect wireless library supports both OS and NonOS platforms. This section describes the OS wrappers required to port to the platform specific OS with Wireless Library.

For Example: Driver uses **rsi_mutex_create** function to create the semaphore.In this function, OS specific mutex create functions which are supposed to be called.

> Enable RSI_WITH_OS macro in SAPIS if OS is used,Need to enable FREERTOS Macro if freertos library is used.
> Also need to make "driver_task_handle" variable global in every example in case of OS.

Below are the functions which can be found in **rsi_os_wrapper.c**inthe following path:
RS9116.NBZ.WC.GEN.OSI.1.x.x\host\sapis\os

4.1 rsi_critical_section_enrty

Prototype

```
rsi_reg_flags_trsi_critical_section_entry()
```

Description
This API is used by Wireless Library to protect the critical section by disabling interrupts. This API implementation should contain code to disable interrupts and return the interrupt status before disabling interrupt (used to the restore interrupt status in exit critical section).

Parameters
None

Return Values
rsi_reg_flags_t
Platform specific data structure to hold the interrupt status before entering critical section.

4.2 rsi_critical_section_exit

Prototype

```
void rsi_critical_section_exit(rsi_reg_flags_txflags
```

Description
This API is used by Wireless Library to exit critical section by restoring interrupt status which was stored while entering critical section. This API implementation should contain code to restore interrupt status based on xflags.

Parameters

Parameter	Description
xflags	This holds interrupt status to restore on exit critical section

Return Values
None

4.3 rsi_mutex_create

Prototype

```
rsi_error_trsi_mutex_create(rsi_mutex_handle_t *mutex)
```

Description
This API is used by Wireless Library to create and initialize mutex instance.

Parameters

Parameter	Description
mutex	This is an instance of rsi_mutex_handle_t type, where definition of rsi_mutex_handle_t structure should map to platform specific OS mutex structure.

Return Values
=0 - On success
!=0 – On Failure

4.4 rsi_mutex_lock

Prototype

```
rsi_error_trsi_mutex_lock(
        volatile rsi_mutex_handle_t *mutex,
        uint32_t timeout_ms)
```

Description
This API is used by Wireless Library to acquire lock on mutex.

Parameters

Parameter	Description
mutex	This is an instance of rsi_mutex_handle_t type, where definition of rsi_mutex_handle_t structure should map to the platform specific OS mutex structure.
timeout_ms	This is the maximum time in milliseconds to wait to acquire mutex lock. If timeout_ms is 0 then wait till mutex lock is acquired.

Return Values
=0 - On success
!=0 – On Failure

4.5 rsi_mutex_unlock

Prototype

```
rsi_error_trsi_mutex_unlock(
        volatile rsi_mutex_handle_t *mutex)
```

Description
This API is used by Wireless Library to release lock on mutex.

Parameters

Parameter	Description
mutex	This is an instance of rsi_mutex_handle_t type, where definition of rsi_mutex_handle_t structure should map to platform specific OS mutex structure.

Return Values
=0 - On success
!=0 – On Failure

4.6 rsi_mutex_destory

Prototype

```
rsi_error_trsi_mutex_destory(rsi_mutex_handle_t *mutex)
```

Description
This API is used by Wireless Library to destroy mutex instance.

Parameters

Parameter	Description
mutex	This is an instance of rsi_mutex_handle_t type, where definition of rsi_mutex_handle_t structure should map to platform specific OS mutex structure.

Return Values
=0 - On success
!=0 – On Failure

4.7 rsi_semaphore_create

Prototype

```
rsi_error_trsi_semaphore_create(rsi_semaphore_handle_t *semaphore,uint32_t count)
```

Description
This API is used by Wireless Library to create and initialize semaphore instance.

Parameters

Parameter	Description
semaphore	This is an instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to the platform specific OS semaphore structure
count	This is the resource count

Return Values
=0 - On success
!=0 – On Failure

4.8 rsi_semaphore_destroy

Prototype

```
rsi_error_trsi_semaphore_destroy(rsi_semaphore_handle_t *semaphore)
```

Description
This API is used by Wireless Library to destroy semaphore instance.

Parameters

Parameter	Description
semaphore	This is an instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to the platform specific OS semaphore structure.

Return Values
=0 - On success
!=0 – On Failure

4.9 rsi_semaphore_wait

Prototype

```
rsi_error_trsi_semaphore_wait(rsi_semaphore_handle_t *semaphore,uint32_t timeout_ms )
```

Description
This API is used by Wireless Library to acquire or wait for semaphore.

Parameters

Parameter	Description
semaphore	This is an instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to the platform specific semaphore structure
timeout_ms	This is the maximum time to wait to acquire semaphore. If timeout_msis 0 then wait till it acquires semaphore.

Return Values
=0 - On success
!=0 – On Failure

4.10 rsi_semaphore_post

Prototype

```
rsi_error_trsi_semaphore_post(rsi_semaphore_handle_t *semaphore)
```

Description
This API is used by Wireless Library to release semaphore which was acquired.

Parameters

Parameter	Description
semaphore	This is an instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to the platform specific OS semaphore structure.

Return Values
=0 - On success
!=0 – On Failure

4.11 rsi_semaphore_reset

Prototype

```
rsi_error_trsi_semaphore_reset(rsi_semaphore_handle_t *semaphore)
```

Description
This API is used by Wireless Library to the semaphore to the initial state.

Parameters

Parameter	Description
semaphore	This is an instance of rsi_semaphore_handle_t type, where definition of rsi_semaphore_handle_t structure should map to the platform specific OS semaphore structure.

Return Values
=0 - On success
!=0 – On Failure

4.12 rsi_task_create

Prototype

```
rsi_error_trsi_task_create(
rsi_task_function_t*task_function,
uint8_t *stack_buffer,
uint32_t stack_size,
void *parameters,
uint32_t task_priority,
rsi_task_handle_t *task_handle)
```

Description
This API is used by Wireless Library to create platform specific OS task/thread.

Parameters

Parameter	Description
task_function	This is the pointer to a function to be executed by created thread. The prototype of the function is as follows: void (*task_function) (void *paramters)
stack_buffer	This is the pointer to a buffer to hold task stack
stack_size	This is the size of the stack buffer
parameter	This is the pointerto the parameters to be passed to task_function
task_priority	This is the task priority (0 – highest priority task)

Parameter	Description
task_handle	This is an instance of rsi_task_handle_t type (task control block), where definition of rsi_task_handle_tstructure should map to platform specific OS task control block structure.

Return Values
=0 - On success
!=0 – On Failure

4.13 rsi_task_destroy

Prototype

```
void rsi_task_destroy(rsi_task_handle_t *task_handle)
```

Description
This API is used by Wireless Library to destroy task/thread, which was already created using rsi_task_createAPI.

Parameters

Parameter	Description
task_handle	This is an instance of rsi_task_handle_t type (task control block), where definition of rsi_task_handle_t structure should map to platform specific task OS control block structure.

Return Values
=0 - On success
!=0 – On Failure

5 Revision History

Revision No.	Version No.	Date	Changes
1	v1.0	November 2017	Preliminary version
2	v1.1	June 2018	Removed references to WiSeMCU as it is meant for WiSeConnect users only
3	v1.2	June 2018	Added Broadcast filter command
4	v1.3	June 2018	Structural changes has been carried out
5	v1.4	July 2018	Deleted the examples from chapter hardware abstraction layer
6	v1.5	May 2020	Deleted zigbee and wlan+zigbee projects info

Disclaimer

The information in this document pertains to information related to Redpine Signals, Inc. products. This information is provided as a service to our customers, and may be used for information purposes only.

Redpine assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at Redpine's sole discretion without any prior notice to anyone. Redpine is not committed to updating this document in the future.