RS9116W SAPI Porting Guide

Version 2.4

June 28, 2021

Table of Contents

About this Document

This document explains the steps and procedures to port Simple APIs (SAPIs) library to a host platform.

1 Overview of Wireless SAPIs

The RS9116W software release package provides a SAPI library to facilitate user application development. This document describes the APIs that are needed to be ported on a host platform to use RS9116W.

The Hardware Abstraction Layer (HAL) contains platform-specific functions like drivers/timers etc. which are supposed to be ported. The OS abstraction layer contains the OS-specific functions which are supposed to be ported as per the RTOS used for user applications. HAL and OS Layers should be modified according to the Host Platform to be used with RS9116W. This is not required when using bare metal applications on the host.

The figure below shows the high-level architecture of SAPIs with 'OS Abstractions Layer' and 'Hardware Abstraction Layer' and how a host can be interfaced with RS9116W.

Note:

The USB interface is currently not supported.

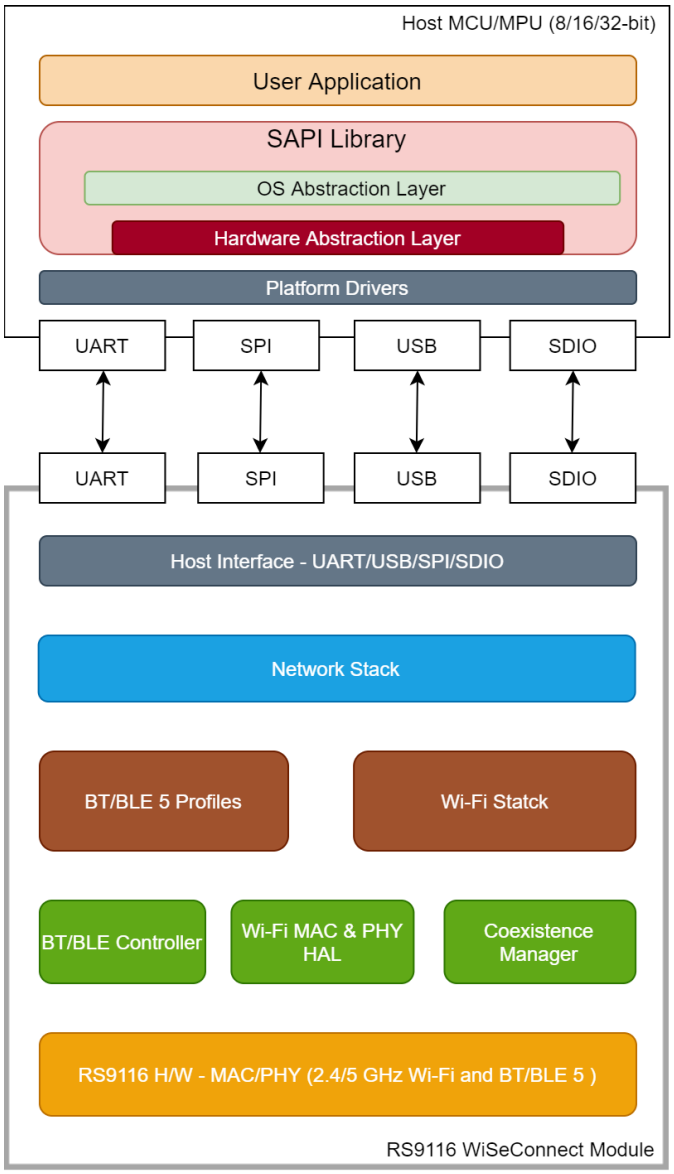

Figure 1: RS9116W Architecture with SAPI

2 SAPI Directory Structure

This section shows the files that must be modified for porting the SAPI Library.

The figure below highlights the HAL and OS directories, which contain files that are required for porting. APIs in these files are described in the next sections of this document.

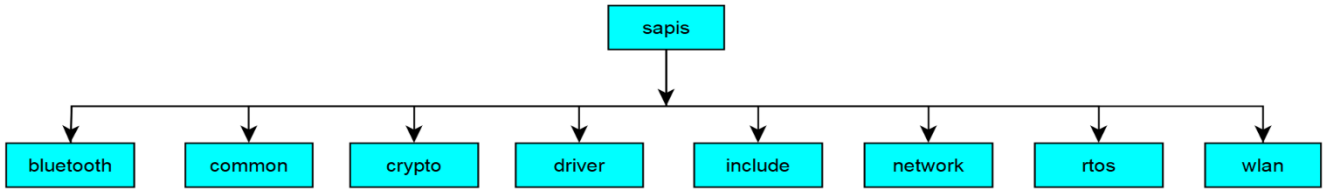

Figure 2: SAPI Directory Structure

SAPIs

Directory Name	Description
	Contains simplified APIs to use Bluetooth Classic and Bluetooth low energy wireless protocols
common	Contains source files for common APIs like device init, driver init, firmware query, etc.
crypto	Contains the APIs related to cryptographic functions
driver	Contains source files for SAPI driver and different host interfaces like SPI, SDIO, and UART
include	Contains all the dependent header files for the APIs/applications
network	Contains source files for network-related applications (MQTT, HTTP, DNS, etc.)
	Contains wrapper files for RTOS (if any) used on the host platform
wlan	Contains simplified APIs related to WLAN wireless protocol like scan, join, ipconfig etc.

3 Hardware Abstraction Layer

This section describes the APIs in SAPI Hardware Abstraction Layer which should be ported on to host platform.

This document focuses on the SPI and UART interfaces. RS9116W WiSeConnect acts as the SPI slave for control and data transfers.

The HAL related files for STM32 and EFR platform are available in the following paths:

<Release_Package>/platforms/stm32/hal

Or

<Release_Package>/platforms/efx32/hal

3.1 SPI Interface APIs

This section describes the APIs used by the SAPI Library to perform SPI transfers to/from the RS9116W module.

The RS9116W operates in SPI mode 0 (clock polarity = 0 and phase = 0) and Chip select is ACTIVE LOW. The SAPI driver uses **rsi_spi_transfer** to send and receive data on SPI. In this function, the actual platform specific SPI transfer function should be to be called.

Source File: rsi_hal_mcu_spi.c

Path: <Release_Package>/platforms/stm32/hal

3.1.1 rsi_spi_transfer

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 the SAPI Library to perform SPI transfer from/to the RS9116W module.

Parameters

Parameter	Description
tx_buff	Pointer to the buffer containing the data to be sent to the module Can be null, if it is a receive-only operation
rx_buff	Pointer to the buffer holding the data received from the module Can be null, if it is a transfer only operation
transfer_length	Length of the RX and TX transfer
mode	Mode of the transfer 0 – 8-bit mode

Example

Default Wrapper	Example - STM32

int16_t rsi_spi_transfer(uint8_t *tx_buff, uint8_t *rx_buff, uint16_t transfer_length,uint8_t mode)
{
uint16_t i;
return 0;
}
	int16_t rsi_spi_transfer(uint8_t *tx_buff, uint8_t *rx_buff, uint16_t transfer_length,uint8_t mode)
	{
	uint16_t i;
	if(tx_buff == NULL) {
	tx_buff = (uint8_t *)&dummy; }
	else if(rx_buff == NULL) {
	rx_buff = (uint8_t *)&dummy; }
	//! enable CS PIN
	cs_enable();
	HAL_SPI_TransmitReceive(&hspi1,tx_buff,rx_buff,transfer_length,20);
	//! disable CS PIN
	cs_disable();
	return 0;
	}

Return Values

Parameter	Description
0	Success
<0	Failure

3.2 UART Interface APIs

This section explains the APIs used by the SAPI library to perform the UART interface with the module. The following list of UART macros must be set for interfacing with the RS9116W module.

Parameter	Description
RSI_UART_DEVICE	Sets UART device port
BAUDRATE	Sets UART Baud rate
RSI_PRE_DESC_LEN	Puts pre-descriptor length
UART_HW_FLOW_CONTROL	Enables UART hardware flow control. 0 - disable 1- Enable
RSI_FRAME_DESC_LEN	Frame descriptor length
RSI_SKIP_CARD_READY	Skip card ready if it is in UART mode
RSI_USB_CDC_DEVICE	UART device or USB-CDC device 0-UART 1-USB-CDC

Source File: rsi_hal_mcu_uart.c

Path: <Release_Package>/platforms/stm32/hal

3.2.1 rsi_uart_send

Prototype

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

Description
This API is used by SAPI Library to perform UART send from/to RS9116W module.

Parameters

Parameter	Description
ptrBuf	Pointer to the buffer with the data to be sent/received
bufLen	Number of bytes to be sent

Return Values

Parameter	Description
0	Success
<0	Failure

Example

Default Wrapper	Example – STM32

int16_t rsi_uart_send(uint8_t *ptrBuf, uint16_t bufLen)
{
return 0;
}
	int16_t rsi_uart_send(uint8_t *ptrBuf, uint16_t bufLen)
	{
	HAL_UART_Transmit_IT(&huart1,(uint8_t *)ptrBuf,bufLen);
	return 0;
	}

3.2.2 rsi_uart_recv

Prototype

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

Description
This API is used by SAPI Library to perform UART receive from/to RS9116W module.

Parameters

Parameter	Description
ptrBuf	Pointer to the buffer with the data to be sent/received
bufLen	Number of bytes to be received

Return Values

Parameter	Description
0	Success
<0	Failure

Example

None

3.2.3 rsi_uart_byte_read

Prototype

```
uint8_t rsi_uart_byte_read(void)
```

Description
This API is used to read the byte data from the RS9116W module through the UART interface.

Parameters
None

Return Values
Read character

Example

None

3.2.4 rsi_uart_init

Source File: rsi_uart.c
Path: <Release_Package>\sapi\driver\device_interface\uart\

Prototype

```
int32_t rsi_uart_init(void)
```

Description
This API is used by SAPI Library to perform initialization of the UART interface with the RS9116W module.
Parameters
None

Return Values

Parameter	Description
0	Success
!=0	Failure

Example

None

3.2.5 rsi_uart_deinit

Prototype

```
int32_t rsi_uart_deinit(void)
```

Description

This API is used by SAPI Library to perform de-initialization of the UART interface with the RS9116W module.

Parameters
None

Return Values

Parameter	Description
0	Success
!=0	Failure

Example

None

3.3 Interrupt APIs

This section describes APIs related to interrupts that should be ported onto the host platform.

Source File: rsi_hal_mcu_interrupt.c
Path: <Release_Package>/platforms/stm32/hal

3.3.1 rsi_hal_intr_config

Prototype

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

Description
This API is used by SAPI Library to configure and to receive packet pending interrupt from the RS9116W module. By default, the RS9116W interrupt line is active high.

Parameters

Parameter	Description
rsi_interrupt_handler	Pointer to a function that should be called in an interrupt handler. In this function, the SAPI library will clear the interrupt and set RX_EVENT.

Return Values

None

Example

Default wrapper	Example – STM32

`void rsi_hal_intr_config(void (* rsi_interrupt_handler)())`
`{`
`//! Configure interrupt pin/register in input mode and register the interrupt handler`
`return;`
`}`
	`void rsi_hal_intr_config(void (* rsi_interrupt_handle)()) {`
	`call_back = rsi_interrupt_handler;`
	`return;`
	`}`

3.3.2 rsi_hal_intr_mask

Prototype

```
void rsi_hal_intr_mask(void)
```

Description
This API is used by SAPI Library to mask/disable the receive packet pending interrupt from the RS9116W module.

Parameters
None

Return Values
None

Example

Default Wrapper	Example – STM32

void rsi_hal_intr_mask(void) {
//! Mask/Disable the interrupt
}
	void rsi_hal_intr_mask(void) {
	HAL_NVIC_DisableIRQ(EXTI9_5_IRQn);
	}

3.3.3 rsi_hal_intr_unmask

Prototype

```
void rsi_hal_intr_unmask(void)
```

Description

This API is used by SAPI Library to unmask the receive packet pending interrupt from the RS9116W module.

Parameters
None

Return Values

None

Example

Default Wrapper	Example – STM32

void rsi_hal_intr_unmask(void) {
//! Unmask/Enable the interrupt
return;
}
	void rsi_hal_intr_unmask(void) {
	HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
	return;
	}

3.3.4 rsi_hal_intr_clear

Prototype

```
void rsi_hal_intr_clear(void)
```

Description
This API is used by SAPI Library to clear the receive packet pending interrupt from the RS9116W module.

Parameters
None

Return Values
None

Example

None

3.3.5 rsi_hal_intr_pin_status

Prototype

```
uint8_t rsi_hal_intr_pin_status(void)
```

Description
This API is used by SAPI Library to check the status of the interrupt pin, to check whether the packet pending from the RS9116W module or not.

Parameters
None

Return Values
Return interrupt pin status (high (1) /low (0)).

Example

Default Wrapper	Example – STM32

uint8_t rsi_hal_intr_pin_status(void) {
volatile uint8_t status = 0;
//! Return interrupt pin status (high (1) /low (0))
return status;
}
	uint8_t rsi_hal_intr_pin_status(void) {
	volatile uint8_t status = 0;
	status = rsi_hal_get_gpio(RSI_HAL_MODULE_INTERRUPT_PIN);
	return status;
	}

Note:

For SDIO Interface:

1. SAPI driver uses Data pin 1 or SMIH_D1 or GPIO_28 as SDIO interrupt line. The user should configure this API to read the status of the GPIO in Hardware Abstraction Porting (HAP) layer.
2. This API should return **"0"** for card interrupts.
3. Users can read the GPIO status, once after SDIO enumeration is completed.

3.4 GPIO APIs

This section describes APIs related to GPIO configurations and usage, which should be ported on the host platform. These GPIOs are used to reset the module, used for data indication and power save handshakes.
These are provided in **rsi_hal_mcu_ioports.c** file in the HAL folder.

Source File: rsi_hal_mcu_ioports.c
Path: <Release_Package>/platforms/stm32/hal

3.4.1 rsi_hal_config_gpio

Prototype

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

Description
This API is used by SAPI Library to configure GPIOs on the host platform which is connected to the RS9116W module. Following is the list of GPIOs used by the SAPI Library.

GPIO PIN Mapping

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

RSI_HAL_SLEEP_CONFIRM_PIN and RSI_HAL_MODULE_WAKEUP_PIN are the pins used in low power modes and are not required to bring up the module.

Users can change the GPIO macro definition according to the host GPIO port numbers.

Parameters

Parameter	Description
gpio_number	GPIO number (constant) used to differentiate GPIO's used by Wireless Library. HAL layer can map these GPIO numbers to the actual GPIO number/ports used in the host platform.
mode	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	Default value to drive on GPIO, if GPIO configured in output mode. 1. Low 2. High

Return Values

None

Example

Default Wrapper

```
void rsi_hal_config_gpio(uint8_t
gpio_number,uint8_t mode,uint8_t value)
{
//! Initialize the gpio pins in input/output mode
//! Drive a default value on gpio if gpio is
configured in output mode
return;

}
```

Example – STM32

```
void rsi_hal_config_gpio(uint8_t gpio_number,uint8_t
mode,uint8_t value) {
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_InitStruct.Pin = GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
GPIO_InitStruct.Pin = GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
GPIO_InitStruct.Pin = GPIO_PIN_6;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

Default Wrapper	Example – STM32
	GPIO_InitStruct.Pull = GPIO_PULLDOWN; GPIO_InitStruct.Speed = GPIO_SPEED_LOW; HAL_GPIO_Init(GPIOC, &GPIO_InitStruct); return; }

3.4.2 rsi_hal_set_gpio

Prototype

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

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

Parameters

Parameter	Description
gpio_number	GPIO number (constant) used to differentiate GPIOs used by SAPI Library. HAL layer can map these GPIO number to the actual GPIO number/ports used in the host platform.

Return Values

None

Example

Default Wrapper	Example – STM32

void rsi_hal_set_gpio(uint8_t gpio_number) {
//! drives a high value on GPIO
return; }
	void rsi_hal_set_gpio(uint8_t gpio_number) {
	if(gpio_number == RSI_HAL_SLEEP_CONFIRM_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_8,GPIO_PIN_SET); }
	if(gpio_number == RSI_HAL_WAKEUP_INDICATION_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_6,GPIO_PIN_SET); }
	if(gpio_number == RSI_HAL_LP_SLEEP_CONFIRM_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_8,GPIO_PIN_SET); }
	if(gpio_number == RSI_HAL_RESET_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_5,GPIO_PIN_SET); }
	return; }

3.4.3 rsi_hal_get_gpio

Prototype

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

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

Parameters

Parameter	Description
gpio_number	GPIO number (constant) used to differentiate GPIOs used by SAPI Library. HAL layer can map these GPIO number to the actual GPIO number/ports used in the host platform.

Return Values

Parameter	Description
0	Low
1	High

Example

Default Wrapper	Example – STM32

uint8_t rsi_hal_get_gpio(uint8_t gpio_number) {
uint8_t gpio_value = 0;
//! Get the gpio value
return gpio_value;
}
	uint8_t rsi_hal_get_gpio(uint8_t gpio_number){
	volatile uint8_t gpio_value = 0; //! Get the gpio value
	if(gpio_number == RSI_HAL_SLEEP_CONFIRM_PIN) {
	gpio_value = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_8); }
	if(gpio_number == RSI_HAL_WAKEUP_INDICATION_PIN) {
	gpio_value = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_6); }
	if(gpio_number == RSI_HAL_LP_SLEEP_CONFIRM_PIN) {
	gpio_value = HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_8); }
	if(gpio_number == RSI_HAL_MODULE_INTERRUPT_PIN) {
	gpio_value = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_9); }
	return gpio_value;
	}

Note:

For SDIO Interface:

1. SAPI driver uses Data pin 1 or SMIH_D1 or GPIO_28 as SDIO interrupt line. The user should configure this API to read the status of the GPIO in Hardware Abstraction Porting (HAP) layer.
2. This API should return **"0"** for card interrupts.
3. User can read the GPIO status, once after SDIO enumeration is completed.

3.4.4 rsi_hal_clear_gpio

Prototype

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

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

Parameters

Parameter	Description
gpio_number	GPIO number (constant) used to differentiate GPIOs used by SAPI Library. HAL layer can map these GPIO number to the actual GPIO number/ports used in the host platform.

Return Values

None

Example

Default Wrapper	Example – STM32

void rsi_hal_clear_gpio(uint8_t gpio_number){
//! drives a low value on GPIO
return;
}
	void rsi_hal_clear_gpio(uint8_t gpio_number) {
	//! drives a low value on GPIO
	if(gpio_number == RSI_HAL_SLEEP_CONFIRM_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_8,GPIO_PIN_RESET) }
	if(gpio_number == RSI_HAL_WAKEUP_INDICATION_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_6,GPIO_PIN_RESET); }
	if(gpio_number == RSI_HAL_LP_SLEEP_CONFIRM_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_8,GPIO_PIN_RESET); }
	if(gpio_number == RSI_HAL_RESET_PIN) {
	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_5,GPIO_PIN_RESET); }
	return;
	}

3.5 Random Number Generation API

This section explains the API used by SAPI Library to get a random number.

Source File: rsi_hal_mcu_random.c
Path: <Release_Package>/platforms/stm32/hal

3.5.1 rsi_get_random_number

Prototype

```
uint32_t rsi_get_random_number(void)
```

Description
This API is used by SAPI Library to get the random number.

Parameters
None

Return Values
32-bit random number

Example

None

3.6 Timer APIs

This section describes timer APIs used by SAPI Library to perform timer handling.

Source File: rsi_hal_mcu_timer.c
Path: <Release_Package>/platforms/stm32/hal

3.6.1 rsi_timer_start

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 SAPI Library to start the host timer.

Parameters

Parameter	Description
timer_node	Timer node to be configured
mode	millisecond timer/ microsecond timer
type	shot timer/ period timer
duration	Time out value
rsi_timer_expiry_handler	Handler which should be called on timeout

Return Values

Parameter	Description
0	Success
!=0	Failure

Example

None

3.6.2 rsi_timer_stop

Prototype

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

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

Parameters

Parameter	Description
timer_node	Unique timer node number

Return Values

Parameter	Description
0	Success
!=0	Failure

Example

None

3.6.3 rsi_timer_read

Prototype

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

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

Parameters

Parameter	Description
timer_node	Unique timer node number

Return Values

Current timer value

Example

None

3.6.4 rsi_delay_ms

Prototype

```
void rsi_delay_ms(uint32_t delay_ms )
```

Description
This API is used by SAPI Library to set delay in milliseconds.

Parameters

Parameter	Description
delay_ms	Delay in milliseconds

Return Values

None

Example

Default Wrapper	Example – STM32

void rsi_delay_ms(uint32_t delay_ms) {
uint32_t start;
if (delay_ms == 0)
return;
start = rsi_hal_gettickcount();
do {
} while (rsi_hal_gettickcount() - start < delay_ms);
return;
}
	void rsi_delay_ms(uint32_t delay_ms) {
	uint32_t start;
	if (delay_ms == 0)
	return;
	start = rsi_hal_gettickcount();
	do {
	} while (rsi_hal_gettickcount() - start < delay_ms);
	return;
	}

3.6.5 rsi_delay_us

Prototype

```
void rsi_delay_us(uint32_t delay_us)
```

Description
This API is used by SAPI Library to set delays in microseconds.

Parameters

Parameter	Description
delay_us	This is the delay in microseconds

Return Values

None

Example

None

3.6.6 rsi_hal_gettickcount

Prototype

```
uint32_t rsi_hal_gettickcount(void)
```

Description
This API is used by SAPI Library to get the current tick count from the SYSTICK timer.

Parameters
None

Return Values
32-bit tick count value

Example

Default Wrapper	Example – STM32

uint32_t rsi_hal_gettickcount() {
//! Define plaftform API to get the tick count delay in milli seconds from systick ISR and return the value
#ifdef LINUX_PLATFORM
struct rsi_timeval tv1;
gettimeofday(&tv1, NULL);
return (tv1.tv_sec * 1000 + tv1.tv_usec/1000);
#endif
}
	uint32_t rsi_hal_gettickcount() {
	return HAL_GetTick();
	}

4 OS Interface Layer

WiSeConnect SAPI library supports both OS and Non-OS platforms. This section describes the OS wrappers required to port to the platform-specific OS with SAPI Library.

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

> Enable RSI_WITH_OS macro in SAPIs, if OS is used. Enable FREERTOS Macro if the FreeRTOS library is used.
>
> Also need to make "driver_task_handle" variable global in every example in case of OS.

Below functions can be found in

Source File: rsi_os_wrapper.c

Path: <Release_Package>/sapi/rtos/freertos_wrapper

4.1 rsi_critical_section_entry

Prototype

```
rsi_reg_flags_t rsi_critical_section_entry();
```

Description
This API is used to enter the critical section.

Precondition
NA

Parameters

None

Return Values

Interrupt status before entering the critical section.

Example

```
rsi_critical_section_entry();
```

4.2 rsi_critical_section_exit

Prototype

```
void  rsi_critical_section_exit(rsi_reg_flags_t xflags);
```

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

Precondition
NA

Parameters

Parameter	Description
xflags	Interrupt status to restore interrupt on exit from the critical section

Return Values

None

Example

```
rsi_critical_section_exit();
```

4.3 rsi_mutex_create

Prototype

```
rsi_error_t rsi_mutex_create(rsi_mutex_handle_t *mutex);
```

Description
This API is used to create and initialize mutex instances.

Precondition
NA

Parameters

Parameter	Description
mutex	mutex handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_mutex_handle_t mutex;
rsi_mutex_create(&mutex);
```

4.4 rsi_mutex_lock

Prototype

```
rsi_error_t rsi_mutex_lock(volatile rsi_mutex_handle_t *mutex);
```

Description
This API is used to acquire the lock on mutex.

Precondition
NA

Parameters

Parameter	Description
mutex	mutex handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_mutex_handle_t mutex;
rsi_mutex_lock(&mutex);
```

4.5 rsi_mutex_unlock

Prototype

```
rsi_error_t rsi_mutex_unlock(volatile rsi_mutex_handle_t *mutex);
```

Description
This API is used to release the lock on mutex.

Precondition
NA

Parameters

Parameter	Description
mutex	mutex handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_mutex_handle_t mutex;
rsi_mutex_unlock(&mutex);
```

4.6 rsi_mutex_destroy

Prototype

```
rsi_error_t rsi_mutex_destroy(rsi_mutex_handle_t *mutex);
```

Description
This API is used to destroy mutex instances.

Precondition
NA

Parameters

Parameter	Description
mutex	mutex handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_mutex_handle_t mutex;
rsi_mutex_destroy(&mutex);
```

4.7 rsi_semaphore_create

Prototype

```
rsi_error_t rsi_semaphore_create(rsi_semaphore_handle_t *semaphore,uint32_t count);
```

Description
This API is used to create and initialize semaphore instances.

Precondition
NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer
count	resource count

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_semaphore_handle_t sem;
rsi_semaphore_create(&sem,0);
```

4.8 rsi_semaphore_destroy

Prototype

```
rsi_error_t rsi_semaphore_destroy(rsi_semaphore_handle_t *semaphore);
```

Description
This API is used to destroy the semaphore instance.

Precondition
NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_semaphore_handle_t sem;
rsi_semaphore_destroy(&sem);
```

4.9 rsi_semaphore_check_and_destroy

Prototype

```
rsi_error_t rsi_semaphore_check_and_destroy(rsi_semaphore_handle_t *semaphore);
```

Description

This API checks whether a semaphore is created or not. It destroys the semaphore instance if it is created, otherwise, it returns success.

Precondition
NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_semaphore_handle_t sem;
rsi_semaphore_check_and_destroy(&sem);
```

4.10 rsi_semaphore_wait

Prototype

```
rsi_error_t rsi_semaphore_wait(rsi_semaphore_handle_t *semaphore, uint32_t timeout_ms );
```

Description

This API is used by the SAPI library to acquire or wait for the semaphore.

Precondition
NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer
timeout_ms	Maximum time to wait to acquire the semaphore. If timeout_ms is 0 then wait till semaphore is acquired.

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_semaphore_handle_t sem;
rsi_semaphore_wait(&sem,0);
```

4.11 rsi_semaphore_post

Prototype

```
rsi_error_t rsi_semaphore_post(rsi_semaphore_handle_t *semaphore);
```

Description

This API is used by the SAPI library to release semaphore, which was acquired.

Precondition
NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_semaphore_handle_t sem;
rsi_semaphore_post(&sem);
```

4.12 rsi_semaphore_post_from_isr

Prototype

```
rsi_error_t rsi_semaphore_post_from_isr(rsi_semaphore_handle_t *semaphore);
```

Description

This API is used by the wireless library to release semaphore, which was acquired.

Precondition
NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_semaphore_handle_t sem;
rsi_semaphore_post_from_isr(&sem);
```

4.13 rsi_semaphore_reset

Prototype

```
rsi_error_t rsi_semaphore_reset(rsi_semaphore_handle_t *semaphore);
```

Description

This API is used by the SAPI library to reset the semaphore.

Precondition
NA

Parameters

Parameter	Description
semaphore	Semaphore handle pointer

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_semaphore_handle_t sem;
rsi_semaphore_reset(&sem);
```

4.14 rsi_task_create

Prototype

```
rsi_error_t rsi_task_create( rsi_task_function_t task_function,uint8_t *task_name,
    uint32_t stack_size, void *parameters,
    uint32_t task_priority, rsi_task_handle_t  *task_handle);
```

Description

This API is used to create platform-specific OS task/thread.

Precondition
NA

Parameters

Parameter	Description
task_function	Semaphore handle pointer
task_name	Name of the created task
stack_size	Stack size given to the created task
parameters	Pointer to the parameters to be passed to task function
task_priority	task priority
task_handle	task handle/instance created

Return Values

Value	Description
0	Success
<0	Failure

Example

```
#define STACK_SIZE     512
#define TASK_PRIORITY  0
rsi_task_handle_t task_handle;
int32_t function()
{
       //
```

```
}
rsi_task_create(function, "wlan_task", STACK_SIZE, NULL, TASK_PRIORITY, &task_handle);
```

4.15 rsi_task_destroy

Prototype

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

Description

This function is used to destroy a task/thread, which was already created using rsi_task_create() API.

Precondition
NA

Parameters

Parameter	Description
*task_handle	Task handle/instance to be deleted

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_task_handle_t task_handle;
rsi_task_destroy(&rsi_task_destroy);
```

4.16 rsi_os_task_delay

Prototype

```
void rsi_os_task_delay(uint32_t timeout_ms);
```

Description

This function induces the required delay in milliseconds.

Precondition
NA

Parameters

Parameter	Description
timeout_ms	

Return Values

Value	Description
0	Success
<0	Failure

Example

```
rsi_os_task_delay(1000);
```

4.17 rsi_start_os_scheduler

Prototype

```
void rsi_start_os_scheduler();
```

Description

This function schedules the tasks created.

Precondition

NA

Parameters

None

Return Values

None

Example

```
rsi_start_os_scheduler();
```

4.18 rsi_task_notify

Prototype

```
rsi_base_type_t rsi_task_notify(rsi_task_handle_t xTaskToNotify, uint32_t ulValue);
```

Description

This function is used to send a notification directly to a task.

Precondition
NA

Parameters

Parameter	Description
xTaskToNotify	The handle of the task to which the notification is being sent
ulValue	How ulValue is used is dependent on the eNotifyAction value

Return Values

None

Example

```
rsi_task_handle_t task_handle;
rsi_task_notify(task_handle,1);
```

4.19 rsi_task_notify_wait

Prototype

```
rsi_base_type_t rsi_task_notify_wait( uint32_t ulBitsToClearOnEntry, uint32_t ulBitsToClearOnExit,
uint32_t *pulNotificationValue, uint32_t timeout);
```

Description

This function allows a task to wait with an optional timeout.

Precondition
NA

Parameters

Parameter	Description
ulBitsToClearOnEntry	bits set here will be cleared in the task's notification value on entry to the function
ulBitsToClearOnExit	bits set here will be cleared in the task's notification value on exit from the function
*pulNotificationValue	used to pass out the task notification value and an optional parameter
timeout	the maximum amount of time the calling task should remain in a blocked state

Return Values

None

Example

```
uint32_t rsi_driver_event;
rsi_task_notify_wait(0,0xFFFFFFFF,rsi_driver_event,0xFFFFFFFF);
```

4.20 rsi_task_notify_from_isr

Prototype

```
rsi_base_type_t rsi_task_notify_from_isr(rsi_task_handle_t xTaskToNotify, uint32_t ulValue,
rsi_base_type_t *pxHigherPriorityTaskWoken );
```

Description

This function is used to send a notification directly to a task and should be used in ISR only.

Precondition
NA

Parameters

Parameter	Description
xTaskToNotify	The handle of the task to which the notification is being sent
ulValue	How ulValue is used is dependent on the eNotifyAction value
*pxHigherPriorityTaskWoken	*pxHigherPriorityTaskWoken must be initialized to 0. Sets the *pxHigherPriorityTaskWoken value to pdTRUE if sending the notification caused a task to unblock, and the unblocked task has a priority higher than the currently running task.

Return Values

None

Example

```
rsi_task_handle_t task_handle;
xHigherPriorityTaskWoken = pdFALSE;
rsi_task_notify_from_isr(task_handle,BIT(event),&xHigherPriorityTaskWoken);
```

4.21 rsi_os_task_notify_take

Prototype

```
uint32_t rsi_os_task_notify_take( BaseType_t xClearCountOnExit, TickType_t xTicksToWait );
```

Description

This function allows a task to wait in a blocked state for its notification value >0.

Precondition
NA

Parameters

Parameter	Description
xClearCountOnExit	The handle of the task to which the notification is being sent
xTicksToWait	How ulValue is used is dependent on the eNotifyAction value

Return Values

None

Example

```
uint32_t MAX_Delay;
rsi_os_task_notify_take(1,MAX_Delay);
```

4.22 rsi_os_task_notify_give

Prototype

```
BaseType_t rsi_os_task_notify_give( rsi_task_handle_t xTaskToNotify );
```

Description

This function is used to notify directly to a task.

Precondition
NA

Parameters

Parameter	Description
xTaskToNotify	Task handle to notify

Return Values

None

Example

```
rsi_task_handle_t task_handle;
rsi_os_task_notify_give(task_handle);
```

5 RTOS Porting

This section describes the procedure to port FreeRTOS to the RS9116W SAPI library.

Introduction

A Real-Time Operating System (RTOS) is software that supplements computer hardware complexities. Almost all real-time operating systems provide threading, the illusion that processors can execute many tasks at the same time. System software that offers primitive I/O and a little more than threading is referred to as a tiny kernel.

Porting RTOS to a new platform requires,

- Determining the system's architecture.
- Figuring out what files to be changed based on the platform's architecture.
- Making necessary changes to the files. This includes implementing the code, creating test code, and exercising the board to ensure that the **RTOS** is up and working as expected.

Block Diagram

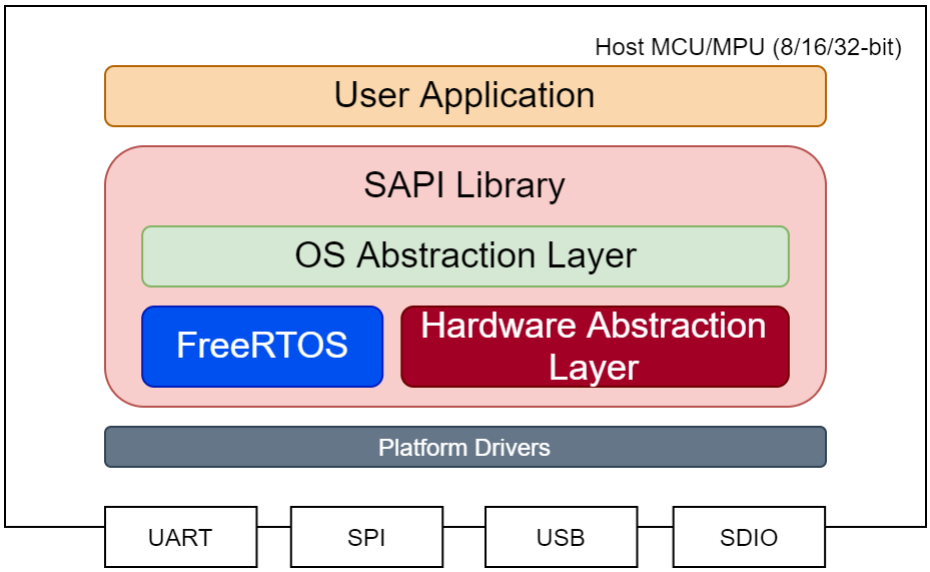

Figure 3: SAPI Block Diagram - OS Abstraction Layer

Steps for Porting FreeRTOS:

1. FreeRTOS is already ported and integrated as part of the release package.

   Path of FreeRTOS kernel: "<Release_Package>/third_party/freertos/"

   For further reference, use the below link for downloading FreeRTOS from GitHub.

   link: "https://github.com/FreeRTOS/FreeRTOS-Kernel/releases/tag/V10.4.3/"
2. To test the FreeRTOS with an example, refer to any of the FreeRTOS projects located in the below path

   Path: "<Release_Package>/examples/featured/"
3. RSI_DRIVER_TASK_PRIORITY should be high in all FreeRTOS applications

6 Revision History

Revision No.	Version No.	Date	Changes
1	1.0	Nov 2017	Preliminary version
2	1.1	Jun 2018	Removed references to WiSeMCU as it is meant for WiSeConnect users only
3	1.2	Jun 2018	Added Broadcast filter command
4	1.3	Jun 2018	Structural changes have been carried out
5	1.4	Jul 2018	Deleted the examples from the chapter hardware abstraction layer
6	1.5	May 2020	Deleted zigbee and wlan+zigbee projects info
7	2.0	Sep 2020	1. Added the 'SAPI Directory Structure' section. 2. Updated the paths and removed SDIO APIs in 'Hardware Abstraction Layer'. 3. Added 'SDK Porting' section with subsection 'AWS SDK Porting' 4. Added ' RTOS Porting' section with subsections 'FreeRTOS Porting for STM32' and 'FreeRTOS Porting-Generic'. 5. Added the below APIs in 'OS Interface Layer' 1. rsi_base_type_t rsi_task_notify_wait(...); 2. rsi_base_type_t rsi_task_notify(...); 3. rsi_base_type_t rsi_task_notify_from_isr(...); 4. rsi_error_t rsi_semaphore_check_and_destroy(..); 5. void rsi_os_task_delay(..); 6. void rsi_start_os_scheduler(); 7. uint32_t rsi_os_task_notify_take(..); 8. BaseType_t rsi_os_task_notify_give(..);
8	2.1	Feb 2021	Added a note under section 6.1, regarding when to call device_init() in FreeRTOS projects.
11	2.4	Jun 2021	1. Added a note under 6.2, regarding RSI_DRIVER_TASK_PRIORITY 2. Updated all sections with changes to match with IoT SDK v2.4 release 3. Removed section on AWS SDK Porting 4. Added a Note for SDIO interface under section 3.3.5 rsi_hal_intr_pin_status.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com