AN1282: RS9116W Guide for SAPI Application Examples

Version 2.1

10/21/2020

Table of Contents

About this Document

This document provides a list of various example applications, along with the steps required for configuring and executing these applications. Important feature mechanisms and IoT Cloud (AWS and Alibaba) usage mechanisms are also described in detail.

1 Introduction to Wireless SAPI Examples

Application examples enables users or developers to get a clear understanding on how to use SAPIs for implementing end user applications.

Examples for following categories are provided in this document,

- BT Classic
- BLE
- BT BLE
- WLAN
- WLAN BT
- WLAN BLE
- WLAN BT BLE
- Crypto
- Debug Utility
- PER Test

2 SAPI Examples Directory Structure

This section shows the complete list of SAPI Examples as per the directory structure. It shows different Example groups and how these are structured in SAPI library.

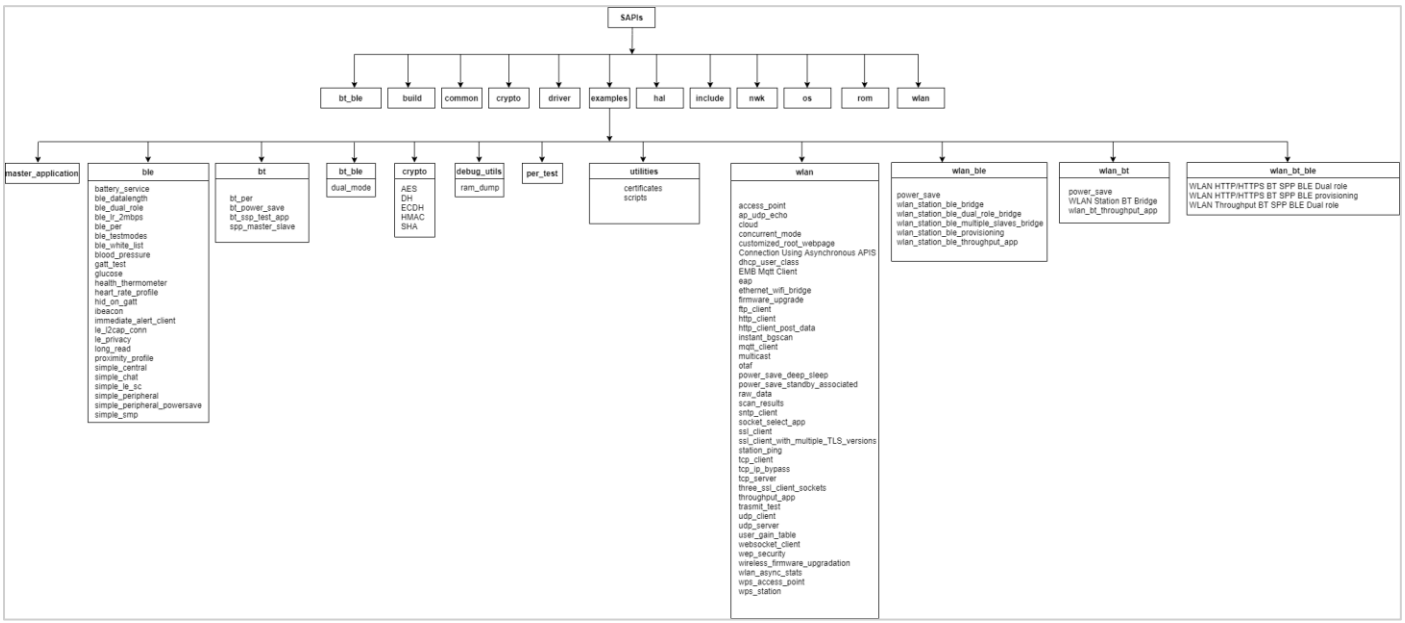


Figure 1: SAPI Examples Directory Structure

SAPIs:

- **bt_ble**: This folder contains simplified APIs to use Bluetooth Classic and Bluetooth low energy wireless protocols

- **build**: This folder contains common Makefile to build all the applications present in "sapis" folder.

- **common**: This folder contains source files for common APIs like device init, driver init, firmware query etc.

- **crypto**: This folder contains the APIs related to cryptographic functions.

- **driver**: This folder contains driver source files for different host interfaces like SPI, SDIO and UART.

- **examples**: This folder contains reference examples for each wireless protocol.

- **hal**: This folder contains hardware abstraction layer for different host interfaces like UART, SPI and SDIO etc. for MCUs.

- **include**: This folder contains all the dependent header files for the APIs/applications.

- **nwk**: This folder contains network related applications (MQTT, HTTP, DNS etc.)

- **os**: This folder contains wrapper files if user wants to use Embedded OS.

- **rom**: This folder contains the rom related APIs for host interfaces, network and MCU relates APIs.

- **wlan**: This folder contains simplified APIs related to WLAN wireless protocol like scan, join, ipconfig etc.

Example Categories:

- **master_application**: This folder contains a dummy example which can be replaced by any of the examples listed in below folders. The replaced example can be compiled and executed on STM32.

- **ble**: This folder contains examples for Bluetooth low energy wireless protocol.

- **bt** : This folder contains examples for Bluetooth Classic protocol.

- **bt_ble**: This folder contains examples for Bluetooth Classic and Bluetooth low energy wireless protocols in dual mode.

- **crypto**: This folder contains examples related to cryptographic functions.

- **debug_utils**: This folder contains ram dump example.

- **per_test**: This folder contains per test example.

- **wlan**: This folder contains examples for WLAN with Bluetooth low energy protocols.

- **wlan_bt**: This folder contains examples for WLAN with Bluetooth Classic protocols.

- **wlan_bt_ble**: This folder contains examples for WLAN with Bluetooth Classic and Bluetooth low energy protocols.

Most of the projects provided in RS9116.NB0.WC.GENR.OSI.X.X.X release package can be compiled and executed on Linux (>= Fedora 16) platform. And for some examples, Keil projects for STM32 are provided which can be compiled in Keil (Windows) and executed on STM32 MCU. STM32 can be interfaced to RS9116W via SPI or UART. List of examples are listed in the next sections.

STM32 based sample projects are provided for the following example categories,

- master_application
- wlan_bt_ble
- wlan_ble (only few)
- wlan (only few)

Note:

Refer to **UG454: RS9116W with STM32 User's Guide.pdf** at https://docs.silabs.com/rs9116 for list of examples projects on STM32 and steps to compile to execute.

Note:

An example 'Master_application' is provided in the RS9116.NB0.WC.GENR.OSI.X.X.X release package, which can be used compile and execute any of the above listed example on STM32 with host interface as SPI. Refer to **UG454: RS9116W with STM32 User's Guide.pdf**

Refer to **Getting Started with Keil IDE** in **UG454: RS9116W with STM32 User's Guide.pdf** for steps to compile are execute above applications on STM32 platform.

Note:

All the example applications work for both Chip and Module.

Set up diagrams shown in the following sections use Single Band EVK image, this is for reference purpose. All examples can be used on other EVKs as well.

Note:

- When updating 'RSI_CONFIG_FEATURE_BITMAP' for any example application, use 'rsi_wlan_common_config.h' at 'RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\include' instead of 'rsi_wlan_config.h'.
- USB and SDIO interfaces are currently not supported.

3 BT Classic

Following is the list of examples described in this section

S.No	Example	Description	Example Source Path
1	BT Per Example	This application demonstrates how to configure the necessary parameters to start transmitting or receiving BT PER packets.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\bt\bt_per
2	BT SPP Master Slave Example	This application demonstrates how to configure the device in Master mode and establish SPP profile connection with remote slave device and data exchange between two devices using SPP profile.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\bt\spp_master_slave
3	BT SSP Test Example	This application demonstrates how to configure the device in Slave mode and establish SPP profile connection with remote Master device using secure simple paring (SSP) and data exchange between two devices using SPP profile.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\bt\bt_ssp_test_app
4	BT Power Save	This application demonstrates that how to configure the device in power save in bt connected mode in bt_power_save example.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\bt\bt_powersave_app

3.1 BT PER

Overview

This application demonstrates how to configure the necessary parameters to start transmitting or receiving BT PER packets.

Sequence of Events

This Application explains user how to:
Configure the BT PER TX or RX mode.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Two Silicon Labs modules

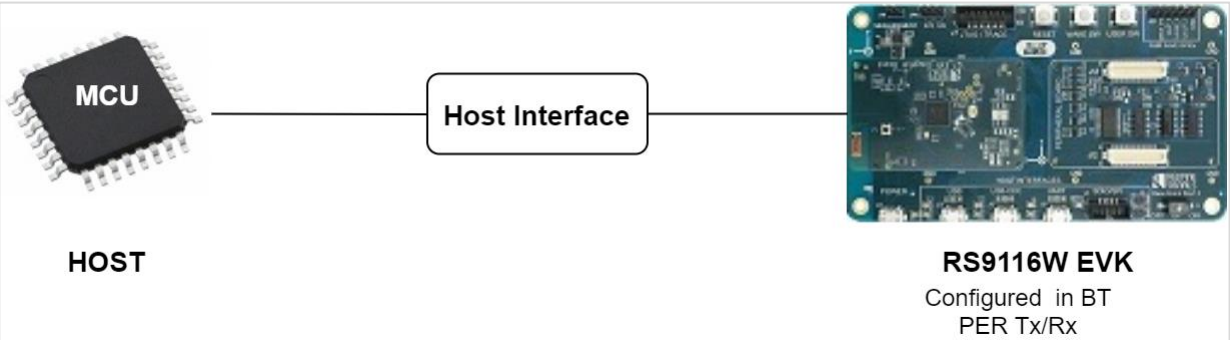


Figure 2: Setup Diagram for BT PER Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_bt_per.c*** file and update/modify following macros,

   RSI_BT_LOCAL_NAME refer to name of Silicon Labs device.

```
#define RSI_BT_LOCAL_NAME                    "PER"
```

   RSI_CONFIG_PER_MODE refer to configuration mode BT PER TX or RX

```
#define RSI_CONFIG_PER_MODE                  RSI_BT_PER_RECEIVE_MODE
```

   OR

```
#define RSI_CONFIG_PER_MODE                  RSI_BT_PER_TRANSMIT_MODE
```

   CMD_ID refer to command id to transmit or receive

```
#define BT_TRANSMIT_CMD_ID                   0x15
#define BT_RECEIVE_CMD_ID                    0x16
```

   PAYLOAD_TYPE refers to type of payload to be transmitted
   '0' – Payload consists of all zeros
   '1' – Payload consists of all 0xFF's
   '2' – Payload consists of all 0x55's
   '3' – Payload consists of all 0xF0's
   '4' – Payload consists of PN9 sequence.

```
#define SEQUENCE_0                           0
#define SEQUENCE_1                           1
#define SEQUENCE_2                           2
#define SEQUENCE_F0                          3
#define SEQUENCE_PRBS                        4
```

```
#define PAYLOAD_TYPE                         SEQUENCE_F0
```

   PACKET_TYPE: Type of the packet to be transmitted, as per the Bluetooth standard. Refer Bluetooth Core 5.0 spec.

```
#define PACKET_TYPE                          15
```

   PACKET_LEN: Length of the packet, in bytes to be transmitted. Refer Bluetooth Core 5.0 spec.

```
#define PACKET_LEN                           339
```

   BT_RX_CHNL_NUM- Receive channel index, as per the Bluetooth standard. i.e., 0 to 78
   BT_TX_CHNL_NUM - Transmit channel index, as per the Bluetooth standard. i.e., 0 to 78

```
#define BT_RX_CHNL_NUM                       10
#define BT_TX_CHNL_NUM                       10
```

   SCRAMBLER_SEED: Initial seed to be used for whitening. It should be set to '0' in order to disable whitening.

```
#define SCRAMBLER_SEED                          0
```

LINK_TYPE: **ACL_LINK**

```
#define ACL_LINK                                1
```

TX_MODE: Burst mode - 0 Continuous mode - 1

```
#define BURST_MODE                              0
#define CONTINUOUS_MODE                         1
```

HOPPING TYPE : no hopping -0 fixed hopping - 1 random hopping - 2

```
#define NO_HOPPING                              0
#define FIXED_HOPPING                           1
#define RANDOM_HOPPING                          2
```

ANT_SEL: on chip antenna - 2 u.f.l - 3

```
#define ONBOARD_ANT_SEL                         2
#define EXT_ANT_SEL                             3
```

RF_TYPE: External RF – 0 Internal RF – 1

```
#define BT_EXTERNAL_RF                          0
#define BT_INTERNAL_RF                          1
```

RF CHAIN: WLAN_HP_CHAIN 0
BT_HP_CHAIN 2

```
#define WLAN_HP_CHAIN_BIT                       0
#define BT_HP_CHAIN_BIT                         2
```

PLL_MODE: PLL_MODE0 – 0 PLL_MODE1 – 1

```
#define PLL_MODE_0                              0
#define PLL_MODE_1                              1
```

LOOP_BACK_MODE: enable 1 or disable 0

```
#define LOOP_BACK_MODE_DISABLE                  0
```

```
#define LOOP_BACK_MODE_ENABLE                    1
```

Following are the **non-configurable** macros in the application.
BT_GLOBAL_BUFF_LEN refer to number of bytes required by the application and the driver

#define BT_GLOBAL_BUFF_LEN 15000

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                          RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                      FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                        RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP               TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP               FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP           EXT_FEAT_384K_MODE
#define RSI_BAND                                 RSI_BAND_2P4GHZ
```

Executing the Application

1. Power on the WiSeConnect module and run the sapis rsi_bt_per application.
2. After the program gets executed, Silicon Labs module starts BT PER transmit or BT PER receive.
3. For receiving purpose use other WiSeConnect module and keep it in BT PER RX mode.
4. Check for BT PER stats whether configured values are affecting or not.

3.2 BT SSP Test App

Overview

This application demonstrates how to configure the device in Slave mode and establish SPP profile connection with remote Master device using secure simple paring (SSP) and data exchange between two devices using SPP profile. In this Application, Silicon Labs module configures in Slave mode and waits to accept SPP profile level connection using secure simple pairing (SSP) from remote device. After successful SPP connection, Application will wait for data to receive from connected remote device. If remote device sends data to Silicon Labs module, Silicon Labs module receives the data and sends back the same data to remote device using SPP profile.

Sequence of Events

This Application explains user how to:

- Configure Silicon Labs module to act as Slave
- Configure device to secure simple pairing (SSP)
- Configure device in discoverable and connectable mode
- Accept SPP level connection from the Smartphone
- Loop back the received messaged

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Mobile with SPP application

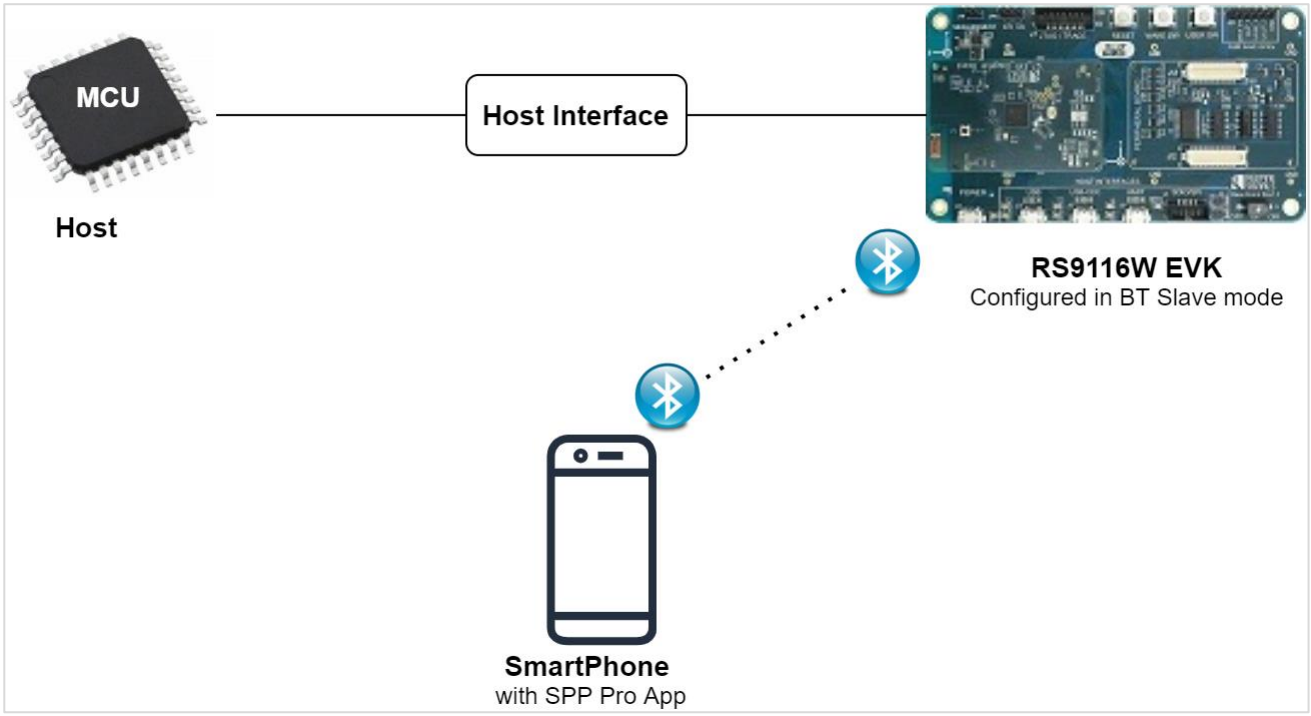


Figure 3: Setup Diagram for SSP Test Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ssp_test_app.c*** file and update/modify following macros:

   RSI_BT_LOCAL_ANME refer to name of Silicon Labs module to appear during scanning by remote devices.

```
#define RSI_BT_LOCAL_NAME                                "BT_SSP_SNIFF"
```

   PIN_CODE refer to four bytes string required for pairing process.

```
#define PIN_CODE                                         "1234"
```

   Following are the **non-configurable** macros in the application:
   BT_GLOBAL_BUFF_LEN refer to number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                               15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

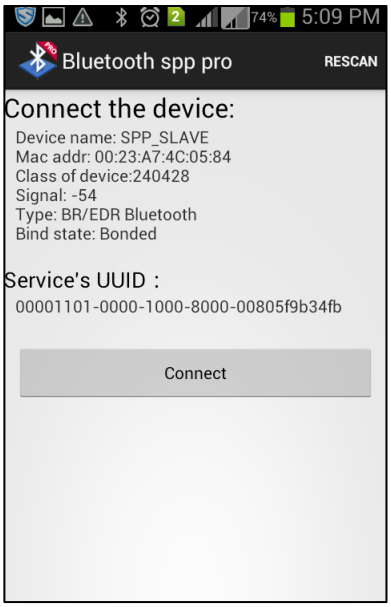
```
#define CONCURRENT_MODE                                  RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                              FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                                RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                       TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                       FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                   EXT_FEAT_384K_MODE
#define RSI_BAND                                         RSI_BAND_2P4GHZ
```

Executing the Application

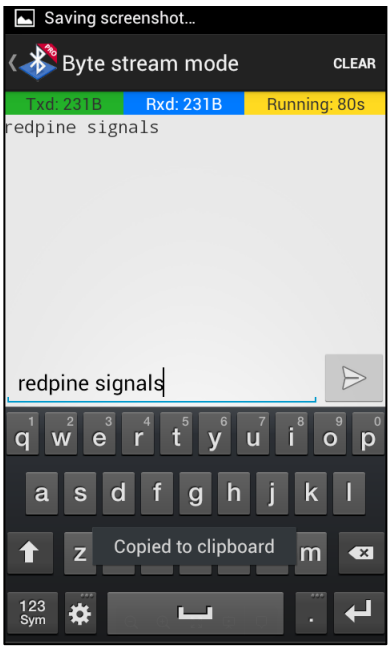
1. After the program gets executed, Silicon Labs module initializes the SPP profile and waits for the incoming connection.

2. Open Bluetooth SPP pro app on mobile and do the scan until Silicon Labs device (Ex: "SPP_SLAVE") gets present in the scan list.
3. After successful scan, select the device and initiate pairing to Silicon Labs device.

4. After initiating paring, Pairing request will pop-up at smart phone side and accept the pairing request.



5. After successful SPP connection, select "Byte stream mode" to send and receive the data.
6. Send some data (Ex: "Silicon Labs signals") from remote device to Silicon Labs device and same data will send back from Silicon Labs device to remote device. Please find below image for sending and receiving data from remote device.



3.3 BT Power Save

Overview

This application demonstrates the process of configuring the device in power save in bt connected mode in bt_power_save example.

Sequence of Events

This Application explains user how to:

- Configure Silicon Labs module to act as Slave
- Configure device in discoverable and connectable mode
- Accept SPP level connection from the Smartphone
- Configure the module in power save mode
- Loop back the received messaged
- Analyze power save functionality when the WiSeConnect device in the bt connected state using an Agilent power analyzer

Application Setup

The WiSeConnect in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

SPI based Setup Requirements

- Windows PC with KEIL or IAR IDE
- Silicon Labs module
- Smartphone with spp pro app or spp manager app
- Agilent power analyzer

UART/USB-CDC based Setup Requirements

- Windows PC with Dev-C++ IDE
- WiSeConnect device
- Smartphone with spp pro app or spp manager app
- Agilent power analyzer

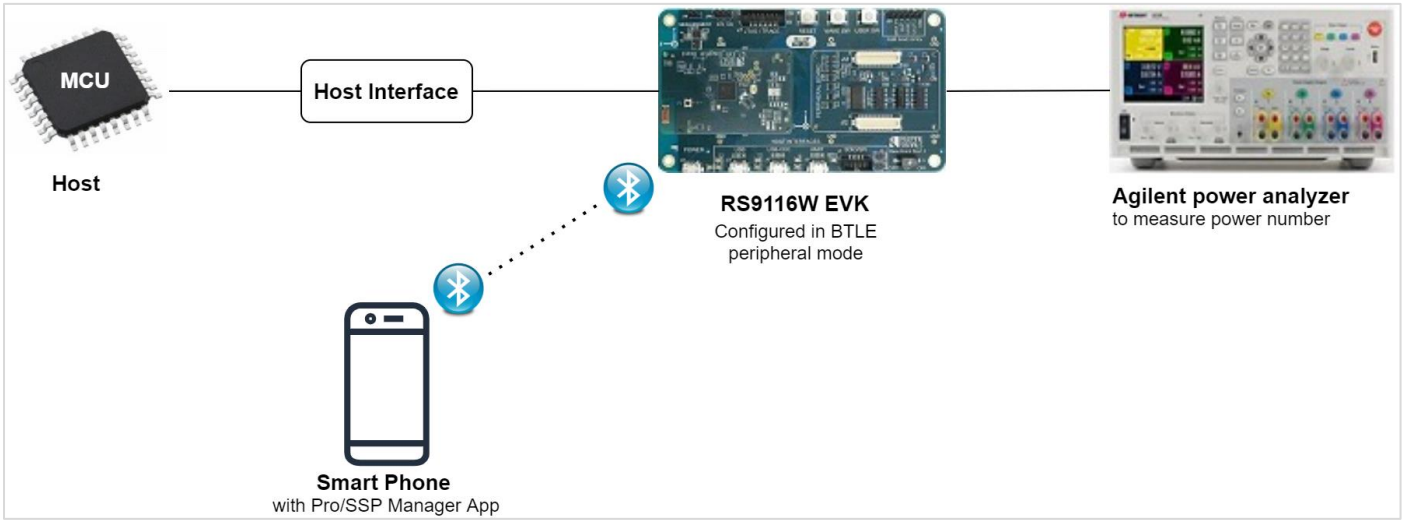


Figure 4: Setup Diagram for BT PowerSave Example

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_bt_power_save_profile.c** file and update/modify following macros.
    a. RSI_BT_LOCAL_NAME **-** Name of the Wyzbee (Master) device
    b. PIN_CODE - Four-byte string required for pairing process.
    c. PSP_TYPE – Power save profile type.

> **Note:**
> 1. PSP_TYPE is only valid RSI_SLEEP_MODE_2.
> 2. RSI_MAXRSI_MAX_PSP is only valid in case of BT._PSP is only valid in case of BT.

    d. SNIFF_MAX_INTERVAL - Sniff Maximum interval value
    e. SNIFF_MIN_INTERVAL – Sniff Minimum interval value
    f. SNIFF_ATTEMPT - Sniff Attempt Value
    g. SNIFF_TIME_OUT – Sniff Timeout Value

2. **To Enable Power Save**
   PSP_MODE refers power save profile mode. The WiSeConnect device supports following power modes in BT

   RSI_ACTIVE (0): In this mode, the module is active and power save is disabled.

   RSI_SLEEP_MODE_2 (1): This mode is applicable when the module is connected state. In this sleep mode, SoC will go to sleep based on GPIO handshake or Message exchange, therefore handshake is required before sending data to the module.

   RSI_SLEEP_MODE_8 (8): In this power mode, the module goes to power save when it is in the unassociated state with the remote device. In this sleep mode, SoC will go to sleep based on GPIO handshake or Message exchange, therefore handshake is required before sending the command to the module.

   #define PSP_MODE RSI_SLEEP_MODE_2

> **Note:**
> For ***RSISLEEP_MODE_2*** *and* ***RSI_SLEEP_MODE_8*** *modes, GPIO or Message based handshake can be selected using* ***RSI_HAND_SHAKE_TYPE*** *macro which is defined in* ***rsi_wlan_config.h***

Note:

In this example, user can verify RSI_SLEEP_MODE_2 with Message based handshake. If the user wants to verify other power modes, the user has to change the application as well as GPIO handshake signals

PSP_TYPE refers power save profile type. The WiSeConnect device supports following power save profile types in BT mode,
RSI_MAX_PSP (0): In this mode, the WiSeConnect device will be in Maximum power save mode. i.e. Device wakes up for every DTIM beacon and does data Tx and Rx.

```
#define PSP_TYPE                                    RSI_MAX_PSP
```

3. Open **rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP
(EXT_FEAT_LOW_POWER_MODE|EXT_FEAT_XTAL_CLK_ENABLE|EXT_FEAT_384K_MODE)
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

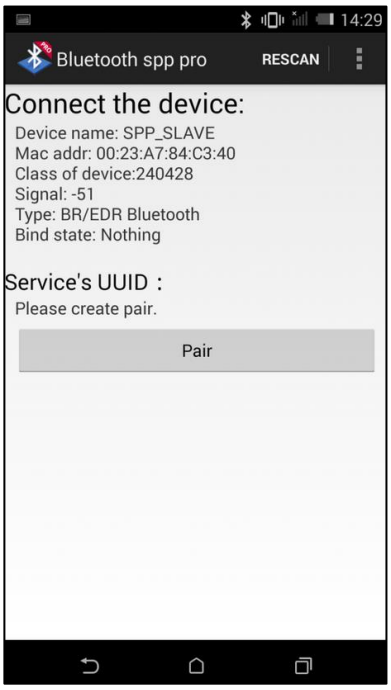
RSI_HAND_SHAKE_TYPE is used to select GPIO or Message based handshake in **RSI_SLEEP_MODE_2** and **RSI_SLEEP_MODE_8** modes**.**

```
#define RSI_HAND_SHAKE_TYPE                         GPIO_BASED
```

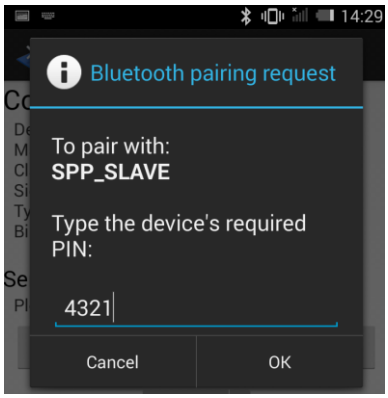
Executing the Application

1. After the program gets executed, Silicon Labs module initializes the SPP profile and waits for the incoming connection.

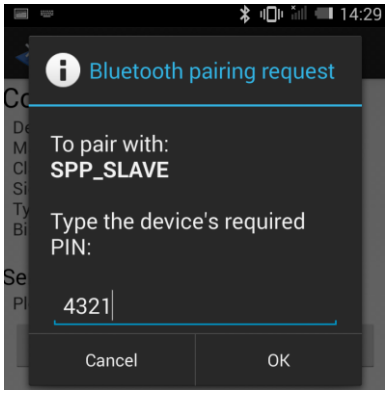
2. Open Bluetooth SPP pro app on mobile and do the scan until Silicon Labs module (Ex: "SPP_SLAVE") gets present in the scan list



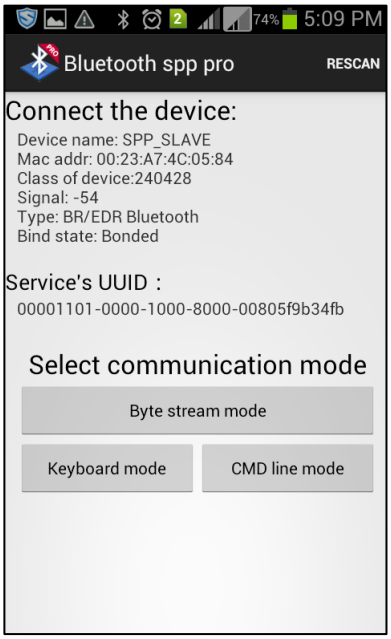
3. After the successful scan, select the device and initiate pairing to Silicon Labs module.



4. After initiating paring, Pairing request will pop-up at smartphone side and issue secret key which is given at Silicon Labs module (PIN_CODE) side.



5. After successful pair, initiate SPP connection to Silicon Labs module and give the secret key for receiving pairing request at remote device side.



6. After successful SPP Connection, Module go to sleep depending on the selected type of PSP TYPE.

7. select "Byte stream mode" to send and receive the data.

8. Send some data (Ex: "Silicon Labs signals") from the remote device to Silicon Labs device and same data will send back from Silicon Labs device to remote device. Please refer the given image for sending and receiving data from the remote device.
9. Note down power measurement by connecting the module to Agilent Power Meter.

3.4 BT SPP Master Slave

Overview

This application demonstrates how to configure the device in Master mode and establish SPP profile connection with remote slave device and data exchange between two devices using SPP profile.
In this Application, Silicon Labs module configures in Master mode and initiates basic connection with remote slave device. After successful basic connection, Application waits to accept SPP profile level connection from remote device. Once SPP connection success, Application will wait for data to receive from connected remote device. If remote device sends data to Silicon Labs module, module receives the data and send back the same data to remote device using SPP profile.

Sequence of Events

This Application explains user how to:

- Configure Silicon Labs module to act as Master
- Connect the Silicon Labs module with the Slave
- Accept SPP level connection from the Smartphone
- Loop back the received messaged

Application Setup

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module

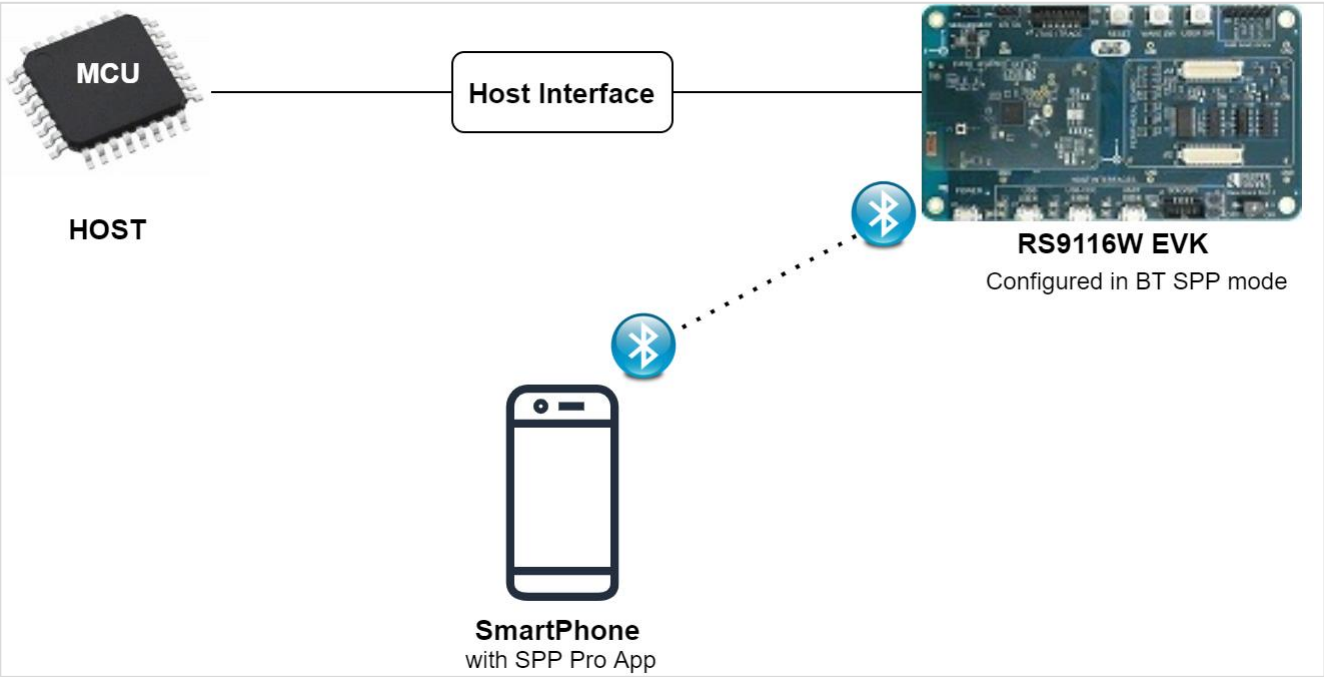


Figure 5: Setup Diagram for BT SPP Master Slave Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_spp_master_slave.c*** file and update/modify following macros,

   SPP_MODE refers to type of module mode, whether it's master/slave.

```
#define SPP_SLAVE    0
#define SPP_MASTER   1
#define SPP_MODE                                    SPP_MASTER
#if (SPP_MODE == SPP_MASTER)
#define RSI_BT_LOCAL_NAME             "SPP_MASTER"
#else
#define RSI_BT_LOCAL_NAME             "SPP_SLAVE"
#endif
```

   PIN_CODE refers four bytes string required for pairing process.

```
#define PIN_CODE                                    "4321"
```

   REMOTE_BD_ADDR refers Remote device BD address to connect.
   Provide the Smart phone BD address,

```
#define REMOTE_BD_ADDR                              "00:1B:DC:07:2C:F0"
```

> **Note:**
> In the smartphone, User Can check the BD address of Bluetooth device in the following location:
> Settings/About phone/status/Bluetooth Address

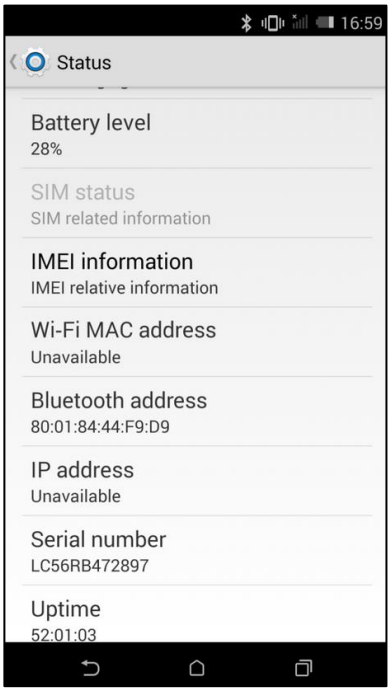


Figure 6: Bluetooth Address

Following are the **non-configurable** macros in the application.

BT_GLOBAL_BUFF_LEN refers the number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                          15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP              EXT_FEAT_384K_MODE
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

Role Switch Configuration

Following 3 API's used to get the role, to set the role and to know the status of the role switch

```
//To know the role of the device
rsi_bt_get_local_device_role((int8_t *)str_conn_bd_addr, &device_state);

//To set the device role to either Master or Slave.(set_role = 0 -->Master,  set_role = 1 -->Slave)
rsi_bt_set_local_device_role((int8_t *)str_conn_bd_addr, set_role, &device_state);

//To know the status of the Switch Role
role_change (uint16_t resp_status, rsi_bt_event_role_change_t *role_change_status1);
```

The status of the role_change function should return success for successful role switch, otherwise fail.

Executing the Application

1. Power on Bluetooth in smart phone and put it in visible mode to all Bluetooth devices.

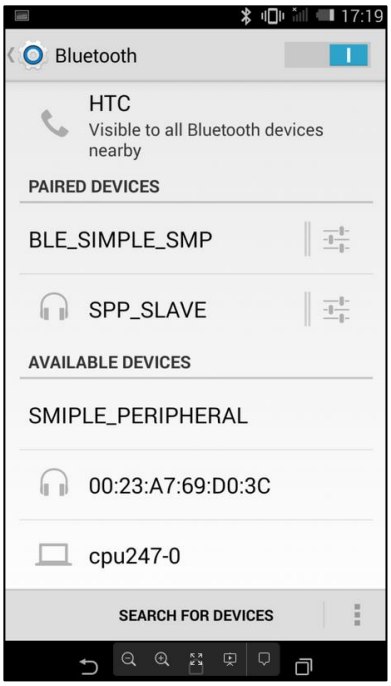


Figure 7: Bluetooth Power On - Visible Mode

2. After the program gets executed, Silicon Labs module initiates basic connection with the remote device (Smart phone). User has to provide **PIN_CODE** at remote device for successful connectivity. Please find below images for connection at remote device.

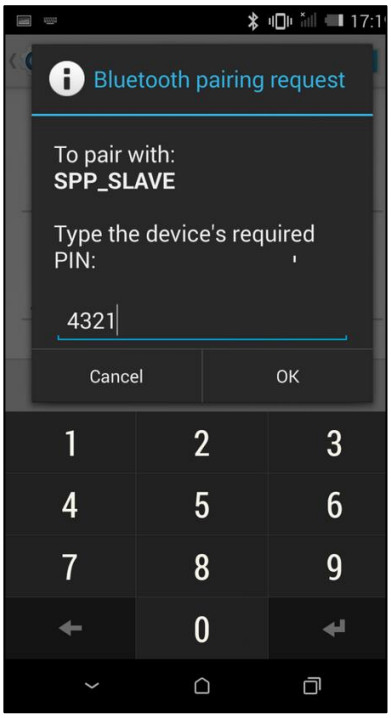


Figure 8: Bluetooth Pairing Request - Enter PIN

3. After successful connection, in smart phone Silicon Labs module lists under Paired devices.
4. After successful connection, Open Sena BT term Bluetooth serial app on mobile which will be in discoverable mode and do the scan from Silicon Labs module. After successful scan, Silicon Labs module can initiate connection to already bonded device as we have already completed basic pairing from Silicon Labs module.
5. At remote device, Bluetooth pairing request will pop-up for SPP connection success. providing secret key (PIN_CODE) for SPP connection success.
6. Send some data (Ex: "Silicon Labs signals") from remote device to Silicon Labs module and some data from Silicon Labs device to remote device.

4 BLE

Following is the list of examples described in this section.

S.No	Example	Description	Example Source Path
1	heart_rate_example	This application demonstrates how to configure Heart rate as GATT server in BLE peripheral mode and explains how to do indicate operation with GATT server from connected remote device using GATT client.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\heart_rate_profile
2	Simple_central_example	This application demonstrates how to connect with remote BLE device in BLE central mode.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\simple_central
3	Simple_peripheral_example	This application demonstrates how to configure the device in simple peripheral mode and how to get connected from the remote Central device.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\simple_peripheral
4	Simple_chat_example	This application demonstrates how to configure GATT server in BLE peripheral mode and explains how to do read&write operations with GATT server from connected remote device using GATT client.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\simple_chat
5	Simple_SMP_example	This application demonstrates how to configure the Silicon labs device in Central mode and connects with remote slave device and how to enable SMP (Security Manager Protocol) pairing.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\simple_smp
6	Simple_peripheral_power save _example	This application demonstrates that how to configure the device in power save in Advertising mode and in connected mode in simple BLE peripheral mode.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\simple_peripheral_powersave
7	BLE_Immediate_alert_client_example	This application demonstrates how a GATT client device accesses a GATT server device.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\immediate_alert_client
8	iBeacon_example	This application demonstrates how to set the iBeacon data format in advertising parameters in simple BLE peripheral mode.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\ibeacon
9	Privacy_example	Bluetooth LE supports a feature that reduces the ability to track an LE device over a period of time by changing the Bluetooth device address on a frequent basis, called Privacy of that particular device.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\le_privacy
10	Proximity_profile	This application demonstrates how to configure Proximity as GATT server in BLE peripheral mode and explains how to do indicate operation with GATT server from connected remote device using GATT client.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\proximity_profile
11	BLE_Long_Read_Example	This application demonstrates how a GATT client device accesses a GATT	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\long_read

S.No	Example	Description	Example Source Path
		server device for long read, means when user wants to read more than MTU (minimum of local and remote devices MTU's) size of data. Silicon labs module acts as a GATT client/server (based on user configuration) and explains reads/writes. Client role is initialized with Battery Service. Server role is initialized with a custom service.	
12	BLE_Long_range and 2Mbps Example	This application connects as a Central and can be used to update the phy rate. The PHY update Procedure is used to change the Transmit or receive PHYs, or both. The procedure can be initiated either on a request by the Host or autonomously by the Link Layer. Either the master or the slave may initiate this procedure at any time after entering the Connection State. The procedure can be initiated either on a request by the Host or autonomously by the Link Layer. Either the master or the slave may initiate this procedure at any time after entering the Connection State.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\ble_lr_2mbps
13	Blood_pressure_profile	The Blood Pressure Service exposes blood pressure and other data from a blood pressure monitor intended for health care applications.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\blood_pressure
14	BLE_data_length_extension_example	This application acts as a Central role and can be used to set data length with connected remote device. Ble Data Packet Length Extension refers to increase in the Packet Data Unit (PDU) size from 27 to 251 bytes. This is the amount of data sent during connection events. Both master and slave can initiate this procedure at any time after entering the Connection.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\ble_datalength
15	BLE_secure connections example	This application demonstrates how to configure the Silicon labs device in peripheral role and connect with remote device. By default, module enables the SMP secure connection.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\simple_le_sc
16	BLE_L2cap_based flow control example	This application demonstrates the l2cap connection-oriented channel connection.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\le_l2cap_conn
17	Battery service example	This application demonstrates how to configure GATT server in BLE peripheral mode, how to configure GATT client in BLE central mode, explains how to do read & notify operations with GATT server from connected remote device using GATT client and explains how to get GATT information from remote GATT server in case of our module as client.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\battery_service
18	Health_thermometer	This application demonstrates how to configure Health thermometer as GATT server in BLE peripheral mode and	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\health_thermometer

S.No	Example	Description	Example Source Path
		explains how to do indicate operation with GATT server from connected remote device using GATT client.	
19	BLE_Per example	This application demonstrates how to configure the necessary parameters to start transmitting or receive BLE PER packets.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\ble_per
20	Glucose service example	This application demonstrates how to configure GATT server in BLE peripheral mode, how to configure GATT client in BLE central mode, explains how to do read, notify and indicate operations with GATT server from connected remote device using GATT client and explains how to get GATT information from remote GATT server in case of our module as client.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\glucose
21	Human interface device service	This application demonstrates how to configure GATT server in BLE peripheral mode, how to configure GATT client in BLE central mode, explains how to do read, notify and indicate operations with GATT server from connected remote device using GATT client and explains how to get GATT information from remote GATT server in case of our module as client.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\hid_on_gatt
22	White list example	This application is used to add a BD-Address to the White List. The device to connect is saved on the white list located in the LL block of the controller. This enumerates the remote devices that are allowed to communicate with the local device. The White List can restrict which device are allowed to connect to another device. If is not, is not going to connect. Once the address was saved, the connection with that device is going to be an auto connection establishment procedure. This means that the Controller autonomously establishes a connection with the device address that matches the address stored in the While List.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\ble_white_list
23	BLE dual role example	This application demonstrates how to connect with multiple (6) slaves as Silicon labs module in central mode and connect with multiple (2) masters as Silicon labs module in peripheral mode.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\ble_dual_role
24	BLE_Testmodes example	This application demonstrates how Silicon labs module will Transmit the packets with desired length.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\ble\ble_testmodes

4.1 Heart Rate Profile

Overview

This application demonstrates how to configure Heart rate as GATT server in BLE peripheral mode and explains how to do indicate operation with GATT server from connected remote device using GATT client.
In this Application, Heart rate GATT server configures with heart rate service with indicate characteristic UUID. When connected remote device writes data to writable characteristic UUID, WiseConnect device receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID and sends indications to the connected device (or) remote device can read the same data using read characteristic UUID if indication enabled on client side.

Sequence of Events

This Application explains user how to:

- Create Heart rate service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smartphone
- Give the indications to connected device

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE Smart Phone with GATT client

Note:

Install Light blue App for tablet for ipad mini and BLE scanner app for android smart phone.

User can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

User can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

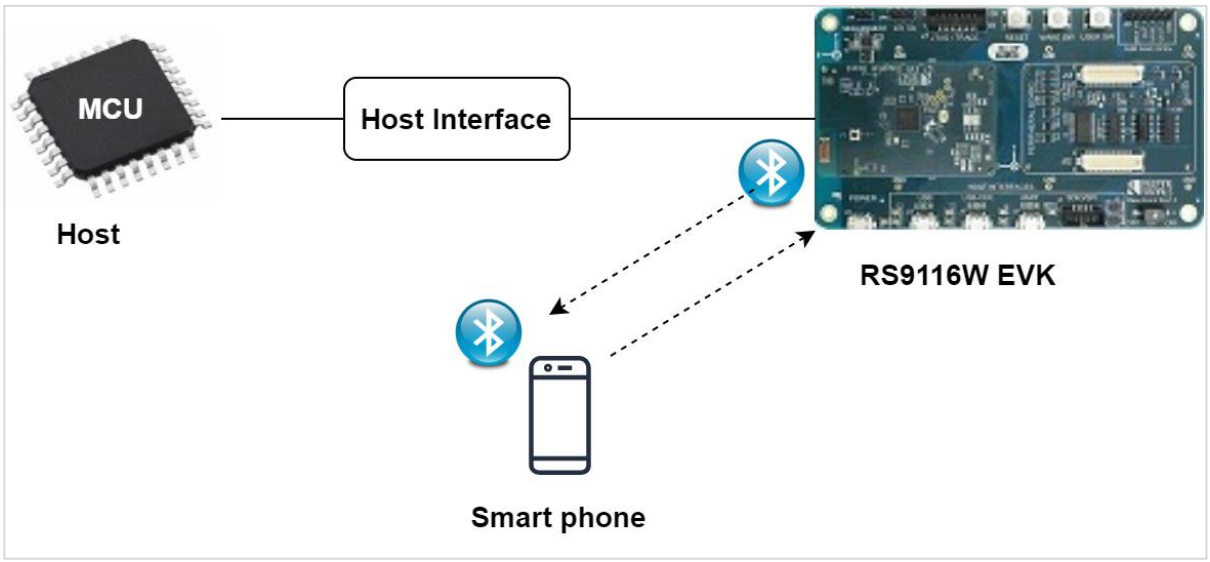


Figure 9: Setup Diagram for Heart Rate Example

Configuration and Steps for Execution

Configuring the Application

Open ***rsi_ble_heart_rate.c*** file and update/modify following macros,

RSI_BLE_HEART_RATE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_HEART_RATE_SERVICE_UUID                0x180D
```

RSI_BLE_HEART_RATE_MEASUREMENT_UUID refers to the attribute type of the first attribute under this service (**RSI_BLE_HEART_RATE_SERVICE_UUID**).

```
#define RSI_BLE_HEART_RATE_MEASUREMENT_UUID            0x2A37
```

RSI_BLE_SENSOR_LOCATION_UUID refers to the attribute type of the second attribute under this service (**RSI_BLE_HEART_RATE_SERVICE_UUID**).

```
#define RSI_BLE_SENSOR_LOCATION_UUID                   0x2A38
```

RSI_BLE_HEART_RATE_CONTROL_POINT_UUID refers to the attribute type of the second attribute under this service (**RSI_BLE_HEART_RATE_SERVICE_UUID**).

```
#define RSI_BLE_HEART_RATE_CONTROL_POINT_UUID          0x2A39
```

RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#define RSI_BLE_MAX_DATA_LEN                           20
```

BLE_HEART_RATE_PROFILE refers name of the Repine device to appear during scanning by remote devices.

```
#define RSI_BLE_HEART_RATE_PROFILE                     "BLE_HEART_RATE_PROFILE"
```

Following are the **non-configurable** macros in the application.

RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                         0x2803
```

RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CLIENT_CHAR_UUID                       0x2902
```

RSI_BLE_ATT_PROPERTY_READ is used to set the read property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_READ                      0x02
```

RSI_BLE_ATT_PROPERTY_WRITE is used to set the WRITE property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_WRITE                    0x08
```

RSI_BLE_ATT_PROPERTY_NOTIFY is used to set the NOTIFY property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_NOTIFY                   0x10
```

RSI_BLE_ATT_PROPERTY_INDICATE is used to set the INDICATE property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_INDICATE                 0x20
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver.

```
#define BT_GLOBAL_BUFF_LEN                            15000
```

GATT_ROLE refers the role of the Silicon Labs module to be selected.
If user configure **SERVER**, Silicon Labs module will act as GATT SERVER, means will add heart rate profile.
If user configure **CLIENT**, Silicon Labs module will act as GATT CLIENT, means will connect to remote GATT server and get services.

```
#define GATT_ROLE                                     1 //Client
```

RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                         LE_PUBLIC_ADDRESS
```

Valid configurations based on address type of the remote device are
LE_RANDOM_ADDRESS

LE_PUBLIC_ADDRESS

Note:
Depends on the remote device, address type will be changed.

RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                              "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect.

```
#define RSI_REMOTE_DEVICE_NAME                        "REDPINE_DEV"
```

Note:
Silicon Labs module can connect to remote device by referring either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

1. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_384K_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

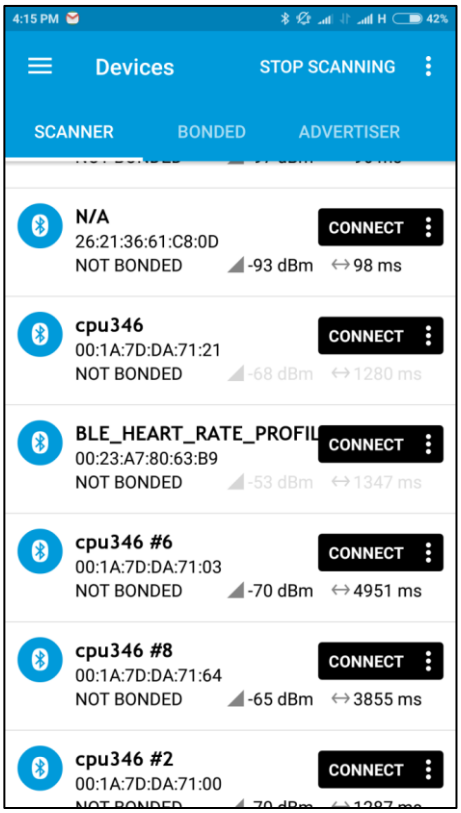
2. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                         30
#define RSI_BLE_PWR_SAVE_OPTIONS                0
```

Note:
rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

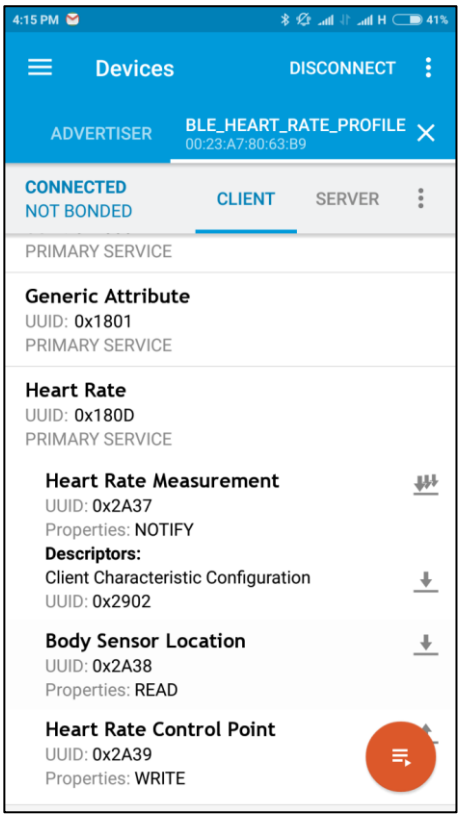
Executing the Application

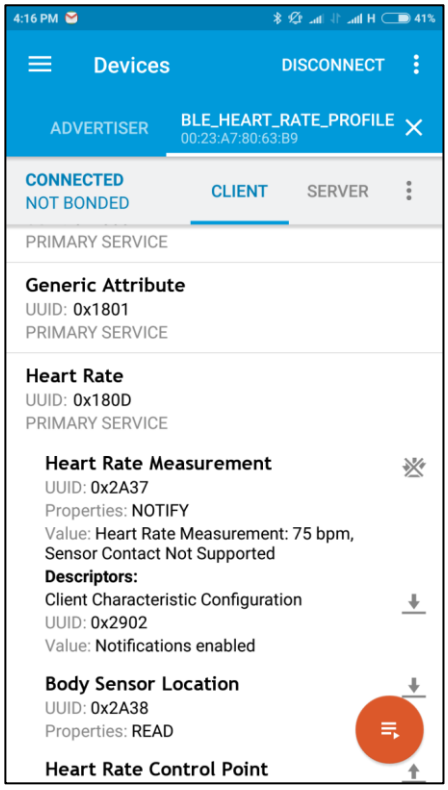
1. After the program gets executed, Silicon Labs will be in Advertising state.
2. Open a LEApp in the Smartphone and do the scan.
3. In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_HEART_RATE_PROFILE (Ex: "BLE_HEART_RATE_PROFILE")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".



4. Initiate connection from the App.

5. After successful connection, LE scanner displays the supported services of Silicon Labs module.
6. Select the attribute service which is added **RSI_BLE_HEART_RATE_PROFILE_UUID**
7. Enable notify for the characteristic **RSI_BLE_HEART_RATE_MEASUREMENT_UUID**
   So that GATT server indicates when value updated in that particular attribute.
8. Whenever the value is updated at server it will be notified to the client which can be read at Heart_Rate_Measurement attribute.
9. Please refer the below images for notify operation from remote device GATT client.





4.2 Simple Central

Overview

This application demonstrates how to connect with remote BLE device in BLE central mode.

Sequence of Events

This Application explains user how to:

- Connect with remote BTLE peripheral device.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device

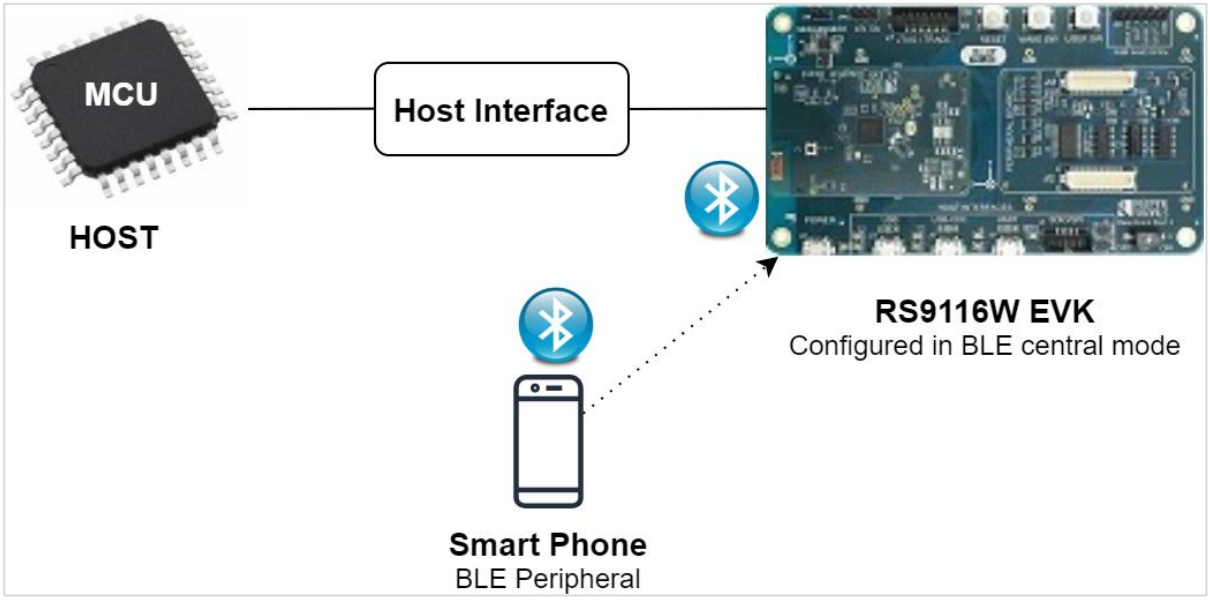


Figure 10: Setup Diagram for Simple Central Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_central.c*** file and update/modify following macros,
   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                    LE_PUBLIC_ADDRESS
```

   Based on address type of remote device, valid configurations are
   LE_RANDOM_ADDRESS

   LE_PUBLIC_ADDRESS

   RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                         "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect.

```
#define RSI_REMOTE_DEVICE_NAME                          "REDPINE_DEV"
```

Note:

user can configure either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

Following are the event numbers for advertising, connection and disconnection events

```
#define RSI_APP_EVENT_ADV_REPORT                        0
#define RSI_APP_EVENT_CONNECTED                         1
#define RSI_APP_EVENT_DISCONNECTED                      2
```

Following are the non-configurable macros in the application.
BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                                 30
#define RSI_BLE_PWR_SAVE_OPTIONS                        0
```

Note:

rsi_wlan_config.h and **rsi_ble_config.h** files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

1. Configure the remote ble device in peripheral mode and put it in advertising mode.
2. After the program gets executed, Silicon Labs device tries to connect with the remote device specified in **RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME** macro.
3. Observe that the connection is established between the desired device and Silicon Labs device.

Note:

Examples for BLE peripherals: Blue tooth Dongle, mobile application, TA sensor tag.

4.3 Simple Peripheral

Overview

This application demonstrates how to configure the device in simple peripheral mode and how to get connected from the remote Central device.

Sequence of Events

This Application explains user how to:

- Set a local name for the device
- Configure the device to advertise
- Start advertising
- Continue advertising even after disconnection with the peer

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable the variety of host processors.

Note:
Install Light blue App on the tablet for iPad mini and BLE scanner app for Android smartphone.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE central device

Note:

Install Light blue App on the tablet for iPad mini and BLE scanner app for Android smartphone

user can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

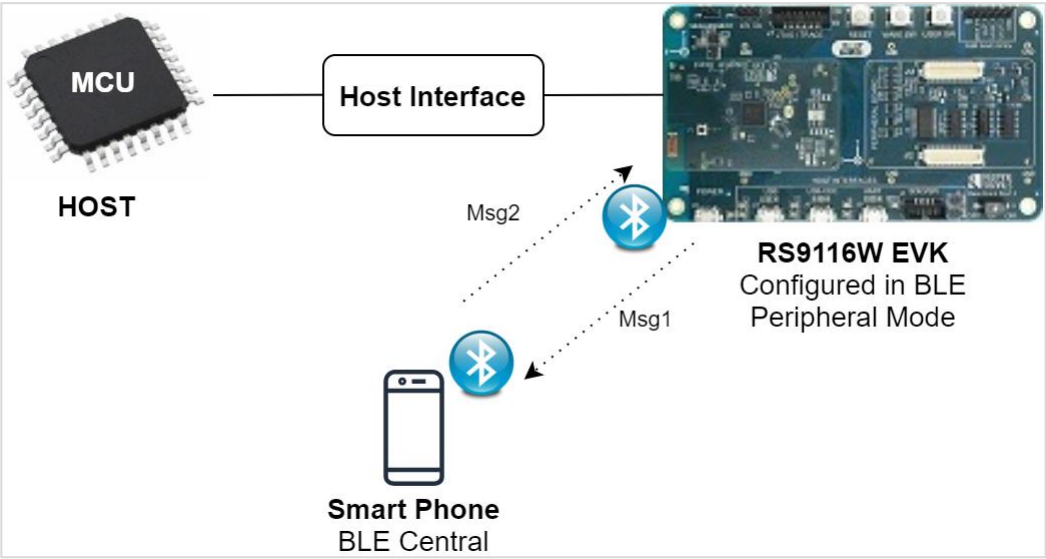


Figure 11: Setup Diagram for Simple Peripheral Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_peripheral.c*** file and update/modify following macros:

   RSI_BLE_LOCAL_NAME refers the name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_LOCAL_NAME                                  "WYZBEE_PERIPHERAL"
```

   RSI_SEL_ANTENNA refers to the antenna which is to be used by Silicon Labs module.
   If the user using internal antenna then set,

```
#define RSI_SEL_ANTENNA                                     RSI_SEL_INTERNAL_ANTENNA
```

   If the user using an external antenna (U.FL connector) then set, **RSI_SEL_EXTERNAL_ANTENNA**

   Following are the **non-configurable** macros in the application.

   Following are the event numbers for connection and Disconnection events.

```
#define RSI_APP_EVENT_CONNECTED                             1
#define RSI_APP_EVENT_DISCONNECTED                          2
```

   BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                                  15000
```

2. *Open **rsi_wlan_config.h** file and update/modify following macros,*

```
#define CONCURRENT_MODE                                     RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                                 FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                                   RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                          TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                          FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                      EXT_FEAT_384K_MODE
#define RSI_BAND                                            RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                                     30
#define RSI_BLE_PWR_SAVE_OPTIONS                            0
```

> **Note:**
> rsi_wlan_config.h and rsi_ble_config.h files are already set with the desired configuration in respective example folders user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs module will be in Advertising state.
2. Open an LE App in the Smartphone and do the scan.

3. In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_LOCAL_NAME (Ex: "WYZBEE_PERIPHERAL")** or sometimes observed as the Silicon Labs device as the internal name "**SimpleBLEPeripheral**".
4. Initiate connection from the mobile App.
5. Observe that the connection is established between Smartphone and Silicon Labs module.

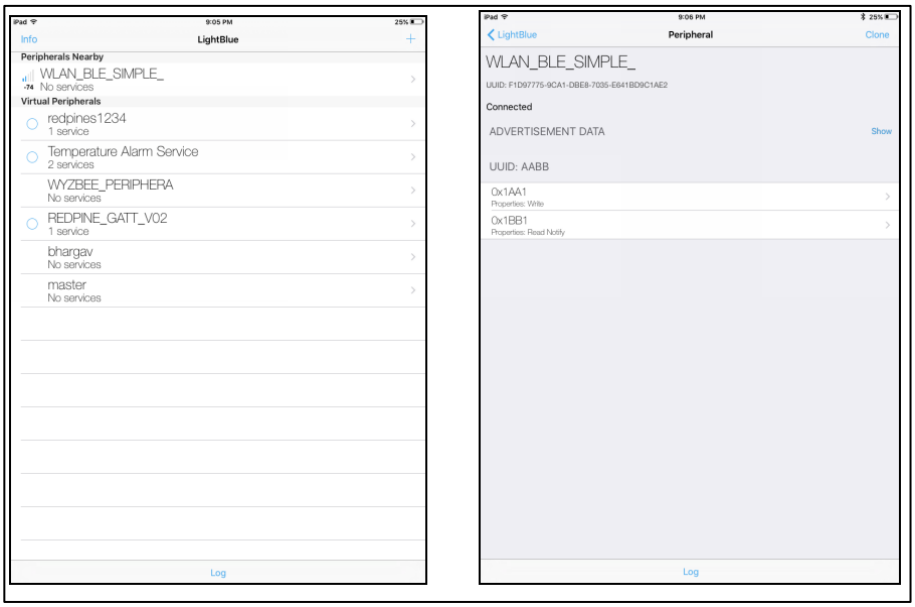


Figure 12: Scanning for BLE Devices and Connecting to WLAN_BLE_SIMPLE Device

4.4 Simple Chat

Overview

This application demonstrates how to configure GATT server in BLE peripheral mode and explains how to do read & write operations with GATT server from connected remote device using GATT client.
In this Application, GATT server configures with Custom service with write and readable characteristic UUIDs. When connected remote device writes data to writable characteristic UUID, Silicon Labs device receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID and sends notifications to the connected device (or) remote device can read the same data using read characteristic UUID.

Sequence of Events

This Application explains user how to:

- Create Simple chat service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smartphone
- Loop back the received message

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE central device

Note:

Install Light blue App for tablet for ipad mini and BLE scanner app for android smart phone.

user can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

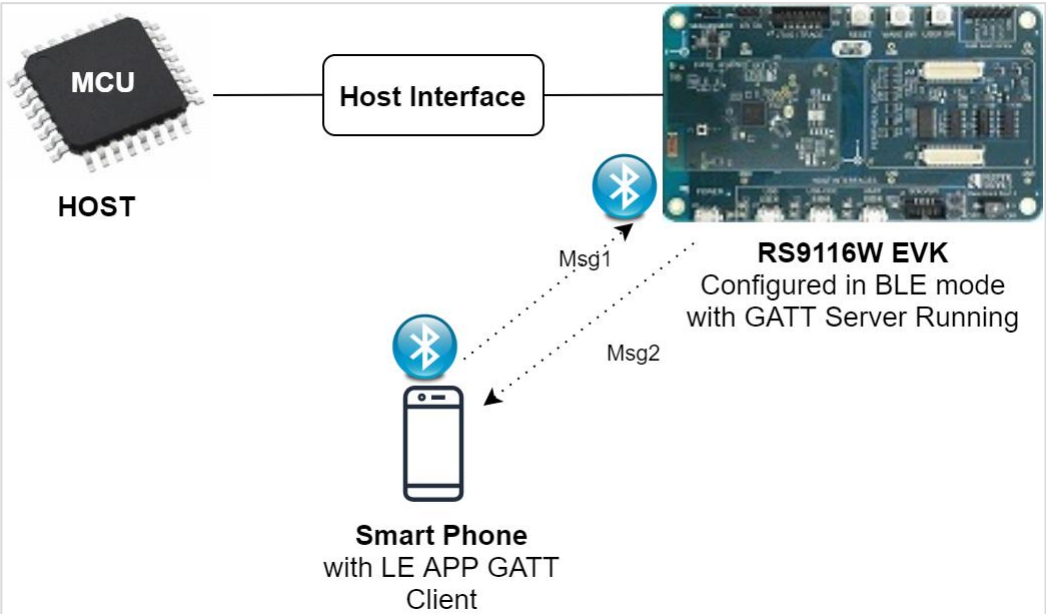


Figure 13: Setup Diagram for Simple Chat Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_simple_chat.c*** file and update/modify following macros,

   RSI_BLE_NEW_SERVICE_UUID refers to the attribute value of the newly created service.
   RSI_BLE_ATTRIBUTE_1_UUID refers to the attribute type of the first attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#define RSI_BLE_NEW_SERVICE_UUID                    0xAABB
#define RSI_BLE_ATTRIBUTE_1_UUID                    0x1AA1
```

   RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#define RSI_BLE_MAX_DATA_LEN                        20
```

   RSI_BLE_APP_SIMPLE_CHAT refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_ APP_SIMPLE_CHAT                    "BLE_SIMPLE_CHAT"
```

   Following are the **non-configurable** macros in the application.

RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.
RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                          0x2803
#define RSI_BLE_CLIENT_CHAR_UUID                        0x2902
```

Following are the properties

RSI_BLE_ATT_PROPERTY_READ is used to set the READ property to an attribute value.
RSI_BLE_ATT_PROPERTY_WRITE is used to set the WRITE property to an attribute value.
RSI_BLE_ATT_PROPERTY_NOTIFY is used to set the NOTIFY property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_READ                       0x02
#define RSI_BLE_ATT_PROPERTY_WRITE                      0x08
#define RSI_BLE_ATT_PROPERTY_NOTIFY                     0x10
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

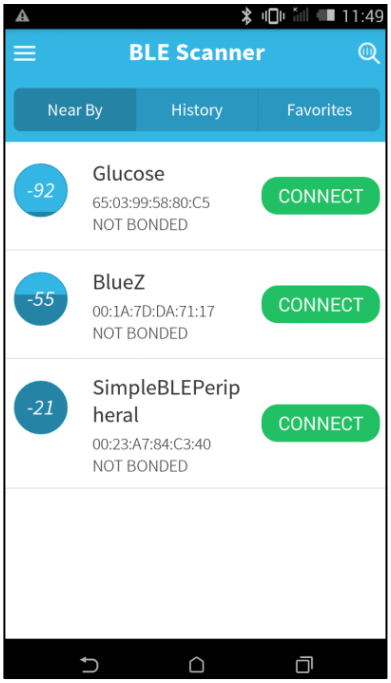
```
#define RSI_BLE_PWR_INX                                 30
#define RSI_BLE_PWR_SAVE_OPTIONS                        0
```

Note:

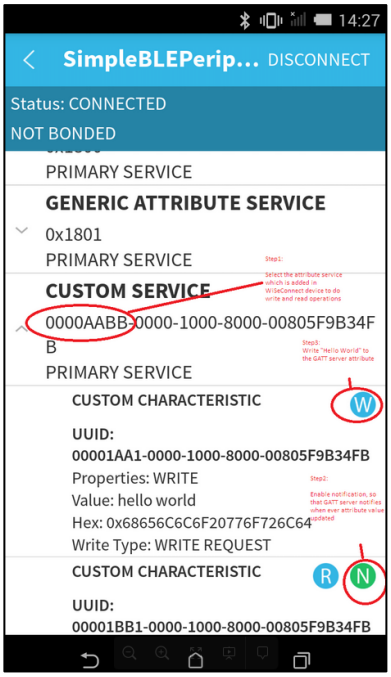
rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

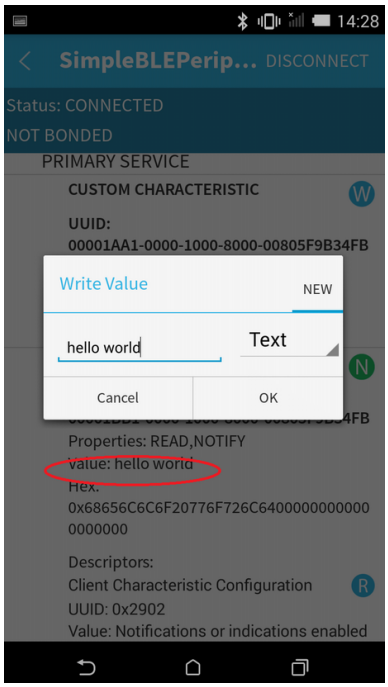
Executing the Application

- After the program gets executed, Silicon Labs module will be in Advertising state.
- Open a LE App in the Smartphone and do the scan.
- In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT (Ex: "BLE_SIMPLE_CHAT")**or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".



- Initiate connection from the App.
- After successful connection, LE scanner displays the supported services of Silicon Labs module
- Select the attribute service which is added **RSI_BLE_NEW_SERVICE_UUID (Ex: 0xAABB)**
- After selecting the service do **Write and Read** operations with GATT server.
- Enable notifications for the read attribute **RSI_BLE_ATTRIBUTE_2_UUID (Example: 0x1BB1) So that GATT server notifies when value updated in that particular attribute.**
- Write data (Ex: "Hello World") to attribute **RSI_BLE_ATTRIBUTE_1_UUID**(Ex: 0x1AA1). So that GATT server notifies when value updated in that particular attribute.
- Silicon Labs module receives the data sent by remote device and same data writes into the attribute **RSI_BLE_ATTRIBUTE_2_UUID** (Ex: 0x1BB1) and will notifies the GATT client (remote device).
- Please refer the given below images for write and read operations from remote device GATT client.





4.5 Simple SMP

Overview

This application demonstrates how to configure the Silicon Labs device in Central mode and connects with remote slave device and how to enable SMP(Security Manager Protocol) pairing.
In this application, Silicon Labs module connects with peripheral or central device and initiates SMP pairing process in case of master role. After successful SMP pairing, SMP encryption will be enabled in both Central and Peripheral device.

> **Note:**
>
> This application is applicable for device which supports Bluetooth 4.0 and 4.1. For devices which supports Bluetooth 4.2 and above version can run LE Secure Connections application to test LE pairing.

Sequence of Events

With salve role, This Application explains user how to:

- Configure device in peripheral/Central mode based on user configuration
- Connect with remote BTLE central/peripheral device.
- Initiate SMP paring from central device
- Initiate SMP pair response.
- Send SMP passkey for the received SMP passkey request.
- Encryption is enabled.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module

- BTLE Peripheral device in case of Silicon Labs module as Master(BLE central)
- BTLE Central device in case of Silicon Labs module as slave(BLE peripheral)

Note:

Install Light blue App for tablet for ipad mini and BLE scanner app for android smart phone.

user can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

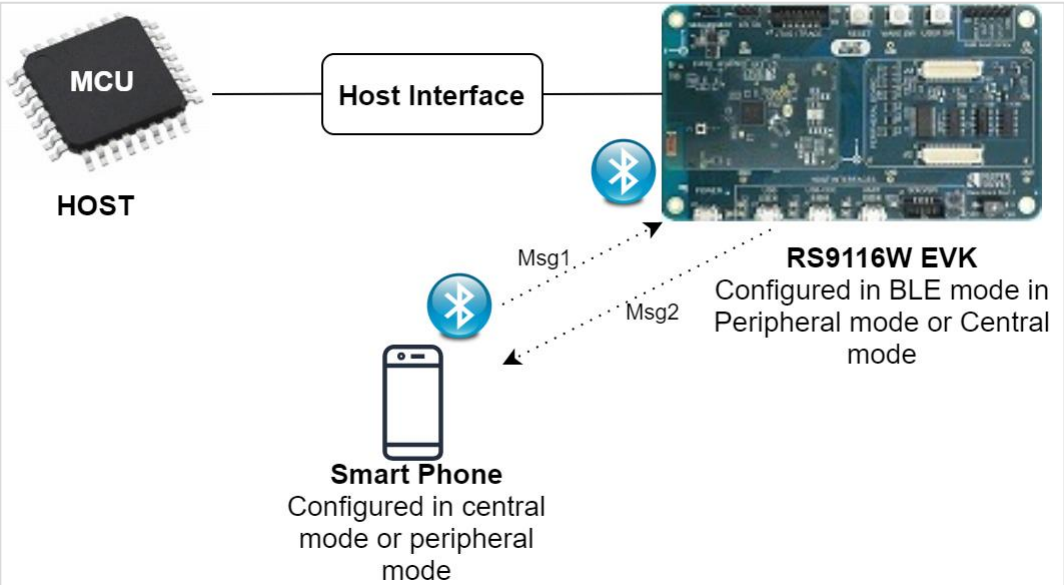


Figure 14: Setup Diagram for SMP Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_smp.c*** file and update/modify following macros,
   ROLE refers the role of Silicon Labs module

```
#define MASTER                                  1
#define SLAVE                                   0
#define ROLE                                    SLAVE
```

RSI_BLE_LOCAL_NAME refers the name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_DEVICE_NAME                     "BLE_SIMPLE_SMP"
```

RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                   LE_PUBLIC_ADDRESS
```

> **Note:**
> Depending on the remote device, address type will be changed.

Valid configurations are
LE_RANDOM_ADDRESS
LE_PUBLIC_ADDRESS
RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                                "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                          "REDPINE_DEV"
```

> **Note:**
> Silicon Labs module can connect to remote device by referring either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

RSI_BLE_SMP_IO_CAPABILITY refers IO capability.
RSI_BLE_SMP_PASSKEY refers address type of the remote device to connect.

```
#define RSI_BLE_SMP_IO_CAPABILITY                       0x03
#define RSI_BLE_SMP_PASSKEY                             0
```

Following are the non-configurable macros in the application.

```
#define RSI_BLE_CONN_EVENT                              0x01
#define RSI_BLE_DISCONN_EVENT                           0x02
#define RSI_BLE_SMP_REQ_EVENT                           0x03
#define RSI_BLE_SMP_RESP_EVENT                          0x04
#define RSI_BLE_SMP_PASSKEY_EVENT                       0x05
#define RSI_BLE_SMP_FAILED_EVENT                        0x06
#define RSI_BLE_ENCRYPT_STARTED_EVENT                   0x07
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                          30
#define RSI_BLE_PWR_SAVE_OPTIONS                 0
```

Note:

rsi_wlan_config.h and **rsi_ble_config.h** files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

Following the execution process in case of Silicon Labs device as master

1. If user select the **MASTER** role, After the program gets executed, Silicon Labs device will be trying to connect to remote BT device, advertise the third party device.
   If user select the **SLAVE** role, After the program gets executed, Silicon Labs device will be advertising, then connect from remote device.
2. After successful connection, flow of commands is as below:
   Master device will initiate SMP pairing
   Slave device give SMP response.
   Both devices will exchange SMP passkey as zero
3. If SMP succeed, host receives SMP encrypt enabled event. If not success, Device sends SMP failure event to host.
4. In encryption enabled event LocalEDIV, Local Rand, LocalLTK parameters will be indicated.
5. Again, after disconnection, if Master want to connect, master ask for LE LTK Request event to slave by giving LocalEDIV and LocalRand, and if same, this example give LocalLTK with positive reply using ltk request reply command.

Note:

We can also send negative reply but remote device may or may not initiate pairing again.
Currently, in encryption enabled event EDIV, RAND, LTK are of local device so that if master initiate connection he will ask for LTK request by giving slave's (in this example) EDIV and RAND.

4.6 Simple Peripheral PowerSave

Overview

This application demonstrates that how to configure the device in power save in Advertising mode and in connected mode in simple BLE peripheral mode.

Sequence of Events

This Application explains user how to:

- Set a local name for the device
- Configure the module in power save mode
- Configure the device to advertise
- Connect from the remote Master device
- Analyze power save functionality when the WiSeConnect device in Advertise mode and in the connected state using an Agilent power analyzer

Application Setup

The WiSeConnect in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

SPI based Setup Requirements

- Windows PC with KEIL or IAR IDE
- Silicon Labs module
- Smartphone (Android)/tablet with LE application.
  Example: Install Light blue App on the tablet for iPad mini and BLE scanner app for the Android smartphone.
- Agilent power analyzer

UART/USB-CDC based Setup Requirements

- Windows PC with Dev-C++ IDE
- WiSeConnect device
- Smartphone (Android)/tablet with LE application.
  Example: Install Light blue App on the tablet for iPad mini and BLE scanner app for the Android smartphone.
- Agilent power analyzer

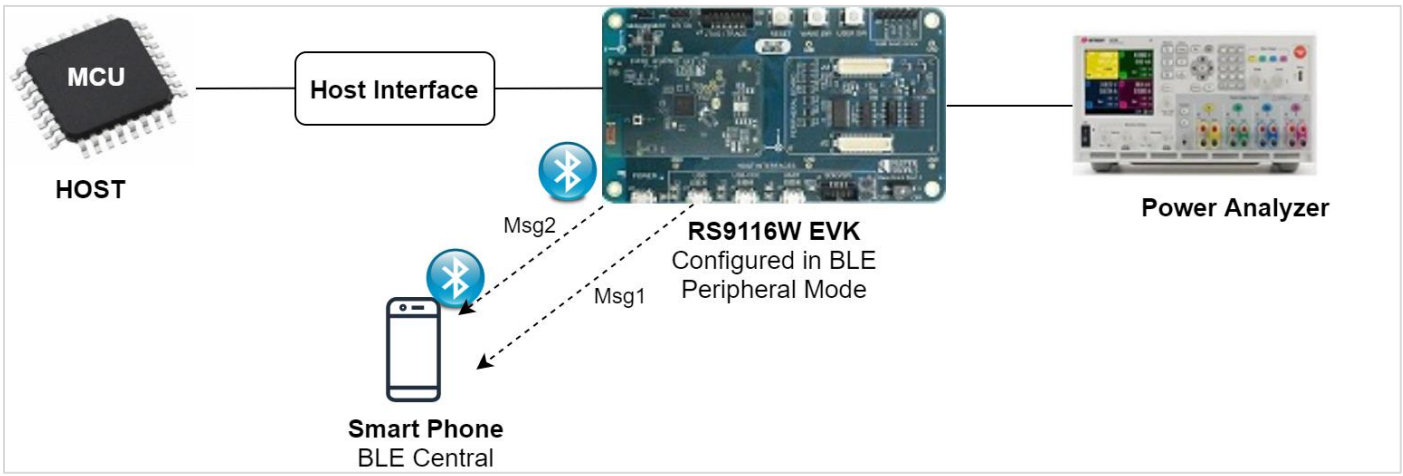


Figure 15: Setup Diagram for Simple Peripheral Power Save Example

Configuration and Steps for Execution
Configuring the Application

1. Open rsi_ble_powersave_peripheral.c file and update/modify following macros

   RSI_BLE_LOCAL_ANME refers the name of the WiSeConnect device to appear during scanning by remote devices.

```
#define RSI_BLE_LOCAL_NAME                    "WLAN_BLE_SIMPLE"
```

   RSI_SEL_ANTENNA refers antenna to be used by WiSeConnect module.
   If the user using internal antenna then set,

```
#define RSI_SEL_ANTENNA                       RSI_SEL_INTERNAL_ANTENNA
```

   If the user using an external antenna (U.FL connector) then set,

```
#define RSI_SEL_ANTENNA                       RSI_SEL_EXTERNAL_ANTENNA
```

   To Enable Power Save
   PSP_MODE refers power save profile mode. The WiSeConnect device supports following power modes in BTLE,

   RSI_ACTIVE (0): In this mode, the module is active and power save is disabled.

   RSI_SLEEP_MODE_2 (1): This mode is applicable when the module is in Advertising state as well as in connected state. In this sleep mode, SoC will go to sleep based on GPIO handshake or Message exchange, therefore handshake is required before sending data to the module.

RSI_SLEEP_MODE_8 (8): In this power mode, the module goes to power save when it is in the unassociated state with the remote device. In this sleep mode, SoC will go to sleep based on GPIO handshake or Message exchange, therefore handshake is required before sending the command to the module.

#define PSP_MODE RSI_SLEEP_MODE_2

Note:

For **RSI*SLEEP_MODE_2*** *and* ***RSI_SLEEP_MODE_8*** *modes****,*** *GPIO or Message based handshake can be selected using* ***RSI_HAND_SHAKE_TYPE*** *macro which is defined in* ***rsi_wlan_config.h***

Note:

In this example, user can verify RSI_SLEEP_MODE_2 with Message based handshake. If the user wants to verify other power modes, the user has to change the application as well as GPIO handshake signals

PSP_TYPE refers power save profile type. The WiSeConnect device supports following power save profile types in BTLE mode,
RSI_MAX_PSP (0): In this mode, the WiSeConnect device will be in Maximum power save mode. i.e. Device will wake up for every DTIM beacon and do data Tx and Rx.

```
#define PSP_TYPE                                RSI_MAX_PSP
```

Following are the non-configurable macros in the application.
Following are the event numbers for advertising, connection and Disconnection events:

```
#define RSI_APP_EVENT_CONNECTED                 1
#define RSI_APP_EVENT_DISCONNECTED              2
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                      15000
```

2. Open **rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          (EXT_FEAT_384K_MODE |
EXT_FEAT_LOW_POWER_MODE)
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

RSI_HAND_SHAKE_TYPE is used to select GPIO or Message based handshake in **RSI_SLEEP_MODE_2** and **RSI_SLEEP_MODE_8** modes**.**

```
#define RSI_HAND_SHAKE_TYPE                     MSG_BASED
```

RSI_SELECT_LP_OR_ULP_MODE is used to select low power mode or ultra low power mode. Valid configurations are**,** **RSI_LP_MODE** or **RSI_ULP_WITH_RAM_RET** or **RSI_ULP_WITHOUT_RAM_RET**

RSI_LP_MODE :
In this, the module will be in Low power mode.

RSI_ULP_WITH_RAM_RET :
In this, the module will be in Ultra low power mode and it will remember the previous state after issuing power save mode command.

RSI_ULP_WITHOUT_RAM_RET :
In this, the module will be in Ultra low power mode and it will not remember the previous state after issuing power save mode command. After wakeup, the module will give CARD READY indication and the user has to issue commands from wireless initialization.

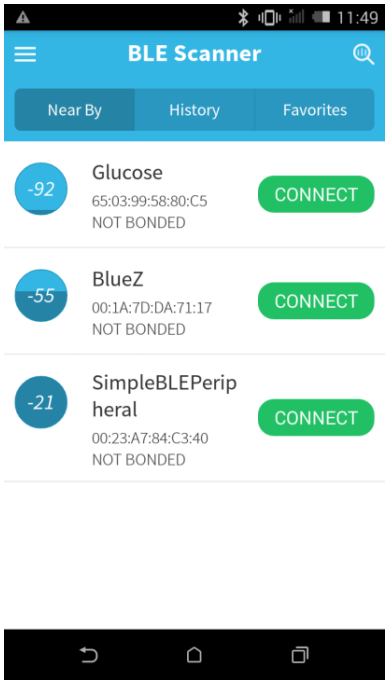
```
#define RSI_SELECT_LP_OR_ULP_MODE                    RSI_ULP_WITH_RAM_RET
```

Executing the Application

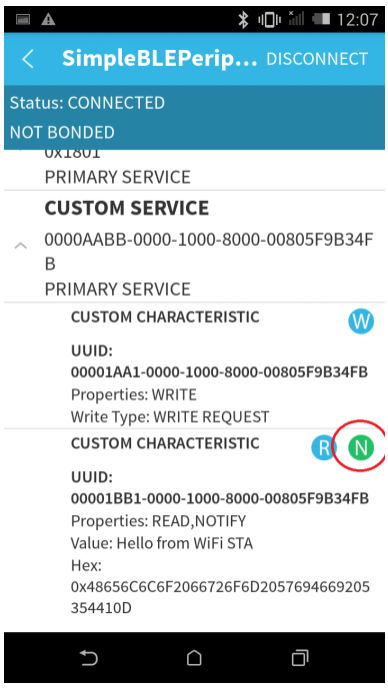
1. After the program gets executed, the WiSeConnect module would be in Advertising state with configured power save the profile.
2. The WiSeConnect device will go to sleep and wakes up for every advertising interval and goes back to sleep after advertising. Please refer the given below image for power save cycle in advertising mode.

Figure 16: Power Profile in Advertising Mode

3. Open an LE App in the Smartphone and do Scan
4. In the App, WiSeConnect module device will appear with the name configured in the macro **RSI_BLE_LOCAL_NAME (Ex: "WLAN_BLE_SIMPLE")** or sometimes observed as the WiSeConnect device as the internal name "**SimpleBLEPeripheral**".



5. Initiate connection from the mobile App.
6. After successful connection, User can see the connected state in BLE scanner app and also check the supported services by the WiSeConnect device.



7. After successful connection, Module goes to sleep and wakes up for every connection interval. Please check the below image for power save cycle after connection.

Figure 17: Power Profile in the Connected State

Note:

Default configuration of connection interval of Master device (smartphone) is 18ms. So, the WiSeConnect device will wake up for every 18ms sec and goes back to sleep after advertising.
Above power save profile image is captured when it is in the idle state, after successful connection. So, the user may not get same profile as shown in the above image. It varies based on the traffic.

4.7 Immediate Alert Client

Overview

This application demonstrates how a GATT client device accesses a GATT server device.

Silicon Labs module acts as a GATT client device in Central mode. Smartphone acts as a GATT server device in peripheral mode which is having immediate alert service. Set up Silicon Labs module to act as GATT client by running the GATT client application. After connecting with the GATT Server device, actual immediate alert notification functionality will be active. In this application, when a peer device moves beyond certain range (can be configured by using the RSSI threshold) of RSSI values, it gives an alert notification to the user and is demonstrated by glowing LED with Yellow (Moving far) and Green (Moving near) colors.

Sequence of Events

This Application explains user how to:

- Connect to BTLE device
- Bring profiles and services from the remote device
- Get rssi value every time

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device

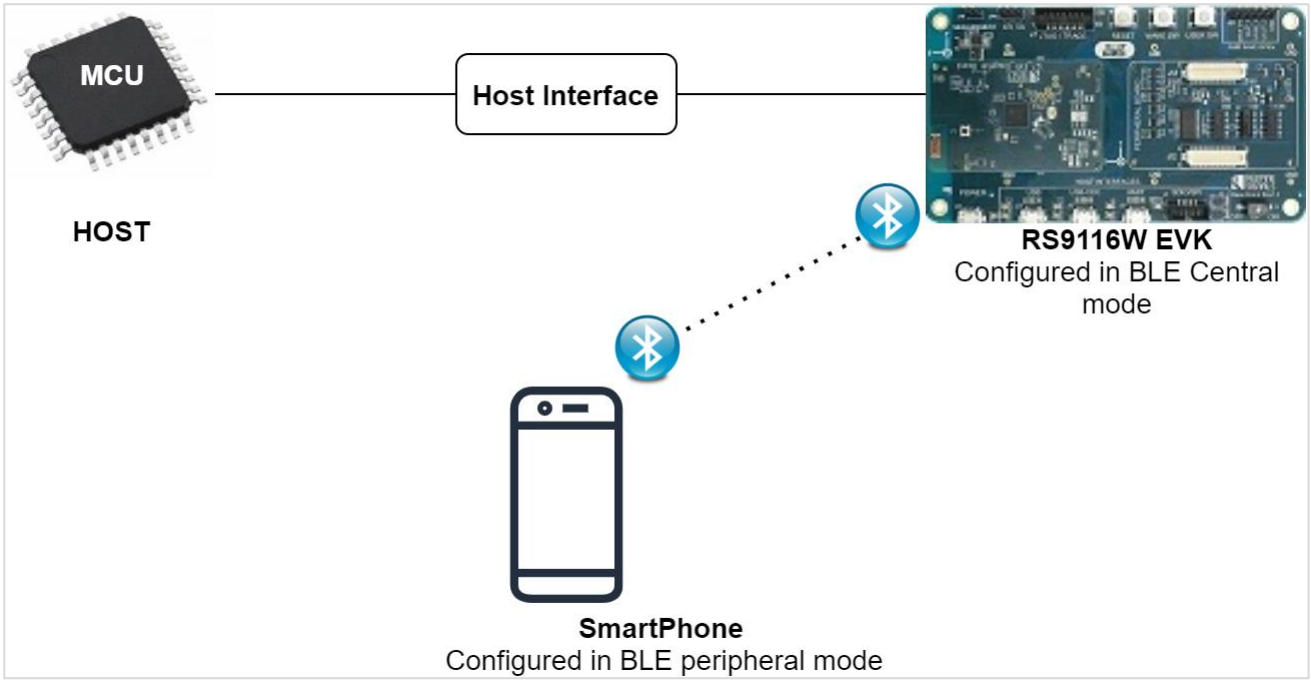


Figure 18: Setup Diagram for BLE Immediate Alert Example

Details of the Application

The application (running in Silicon Labs module) includes following steps.

1. Make Silicon Labs module to act as GATT client device.
2. Connect the Silicon Labs module with the remote device.

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_immediate_alert_client.c*** file and update/modify following macros,
   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                    LE_PUBLIC_ADDRESS
```

Valid configurations are
LE_RANDOM_ADDRESS

LE_PUBLIC_ADDRESS

> **Note:**
>
> Depends on the remote device, address type will be changed.

RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                         "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                   "REDPINE_DEV"
```

Note:

Silicon Labs module can connect to remote device by referring either **RSI_BLE_DEV_ADDR** or **RSI_REMOTE_DEVICE_NAME** of the remote device.

RSI_BLE_RSSI_THRESHOLD_VALUE- To set the RSSI threshold for immediate alert service.

```
#define RSI_BLE_RSSI_THRESHOLD_VALUE                    40
```

RSI_BLE_NEW_SERVICE_UUID refers to the attribute value of the newly created service.

RSI_BLE_ATTRIBUTE_1_UUID refers to the attribute type of the first attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#define RSI_BLE_NEW_SERVICE_UUID                        0x1802
#define RSI_BLE_ATTRIBUTE_1_UUID                        0x2A06
```

RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#define RSI_BLE_MAX_DATA_LEN                             20
```

RSI_BLE_APP_SIMPLE_CHAT refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_NAME                                 "IMMEDIATE_ALERT"
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.

RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                          0x2803
#define RSI_BLE_CLIENT_CHAR _UUID                       0x2902
```

RSI_BLE_ATT_PROPERTY_READ is used to set the READ property to an attribute value.

RSI_BLE_ATT_PROPERTY_WRITE is used to set the WRITE property to an attribute value.

RSI_BLE_ATT_PROPERTY_NOTIFY is used to set the NOTIFY property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_READ                       0x02
#define RSI_BLE_ATT_PROPERTY_WRITE                      0x08
#define RSI_BLE_ATT_PROPERTY_NOTIFY                     0x10
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_384K_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                     30
#define RSI_BLE_PWR_SAVE_OPTIONS               0
```

Note:

rsi_wlan_config.h and **rsi_ble_config.h** files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs module would be trying to connect to remote device
2. Advertise GATT server LE device, which has the support for immediate alert service.
3. Silicon Labs module can connect to it.
4. After connecting, application use immediate alert service in server and depending on rssi value, a notification will be sent to user.

4.8 IBeacon

Overview

This application demonstrates how to set the iBeacon data format in advertising parameters in simple BLE peripheral mode.

Sequence of Events

This Application explains user how to:

- Set a local name to the device
- Set the iBeacon data to be advertised in Silicon Labs module
- Configure the device in Advertise mode
- Scan from remote Master device

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone with ibeacon detector application

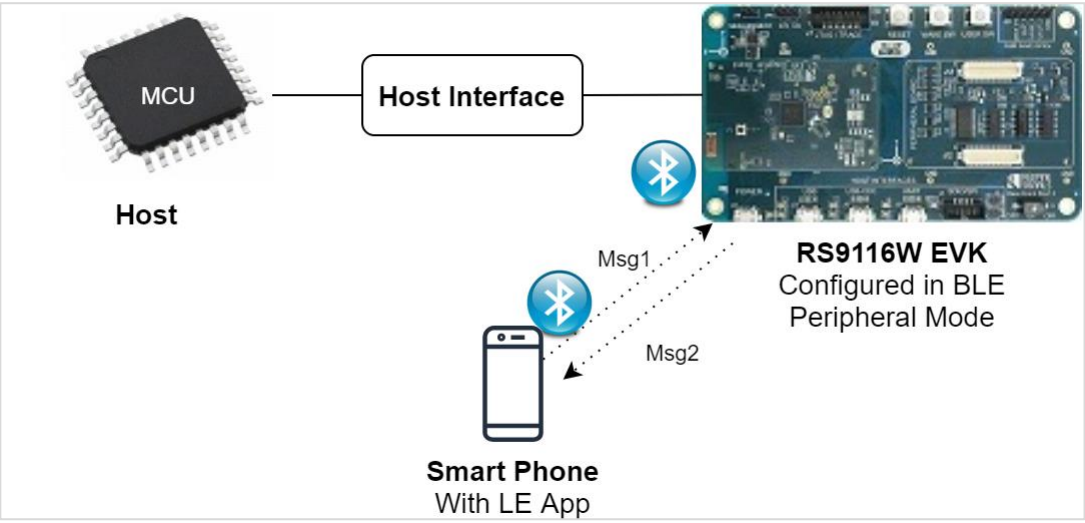


Figure 19: Setup Diagram for iBeacon Example

Note:
Install iBeaconDetector app for android smart phone.
https://play.google.com/stor/apps/details?id=youten.redo.ble.ibeacondetector

iBeacon Advertise data format

iBeacon prefix 9Bytes	UUID 16bytes`	Major Number 2Bytes	Minor Number 2 Bytes	Tx Power 1bytes

Adv Flags 3 Bytes	Adv Header 2 Bytes	Company ID 2 Bytes	iBeacon Type 1Byte	iBeacon Length 1Bytes

iBeacon Prefix:
Vendor specific fixed value.
Default iBeacon prefix values setting by application is,
Prefix = {0x02, 0x01, 0x02, 0x1A, 0xFF, 0x4C, 0x00, 0x02, 0x15}

UUID:
User generated proximity UUID.
Remote devices recognize which beacon they approach on the basis of UUID, major and minor numbers.
Default UUID, Major and Minor values setting by application is,
UUID = {0xFB , 0x0B , 0x57 , 0xA2 , 0x82 , 0x28 , 0x44 , 0xCD , 0x91 ,
0x3A , 0x94 , 0xA1 , 0x22 , 0xBA , 0x12 , 0x06}
major_num = {0x11, 0x22}
minor_num = {0x33, 0x44}

Tx power is used to calculate distance from iBeacon.
Default Tx power value setting by application is,
Tx Power = 0x33

Note:
If the user wants to change the prefix, UUID, Major number, Minor number and Tx Power values, Please change the following values in **rsi_ble_ibeacon.c_** file.

For Prefix:
uint8_t adv[31] = {0x02, 0x01, 0x02, 0x1A, 0xFF, 0x4C, 0x00, 0x02, 0x15};
//prefix(9bytes)

For UUID:

uint8_t uuid[16] = {0xFB , 0x0B , 0x57 , 0xA2 , 0x82 , 0x28 , 0x44 , 0xCD , 0x91 , 0x3A , 0x94 , 0xA1 , 0x22 , 0xBA , 0x12 , 0x06};
For Major Number:
uint8_t major_num[2] = {0x11, 0x22};
For Major Number:
uint8_t minor_num[2] = {0x33, 0x44};
For Tx Power:
uint8_t tx_power = 0x33;

```
86  * This function is used to test the BLE peripheral role and simple GAP API's.
87  */
88 int32_t rsi_ble_ibeacon(void)
89 {
90     int32_t status = 0;
91     int32_t temp_event_map = 0;
92     uint8_t remote_dev_addr[18] = {0};
93     uint8_t adv[31] =  {0x02, 0x01, 0x02, 0x1A, 0xFF, 0x4C, 0x00, 0x02, 0x15}; //prefix(9
94     uint8_t uuid[16] = {0xFB , 0x0B , 0x57 , 0xA2 , 0x82 , 0x28 , 0x44 , 0xCD , 0x91 , 0x
95   uint8_t major_num[2] = {0x11, 0x22};
96   uint8_t minor_num[2] = {0x33, 0x44};
97   uint8_t tx_power = 0x33;
98
```

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_ibeacon.c*** file and update/modify following macros:

   RSI_BLE_LOCAL_ANME refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_LOCAL_NAME                    "ibeacon"
```

   Following are the event numbers for connection and Disconnection events,

```
#define RSI_APP_EVENT_CONNECTED               1
#define RSI_APP_EVENT_DISCONNECTED            2
```

   Following are the **non-configurable** macros in the application.
   BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                    15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                       RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                   FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                     RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP            TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP            FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP        EXT_FEAT_384K_MODE
#define RSI_BAND                              RSI_BAND_2P4GHZ
```

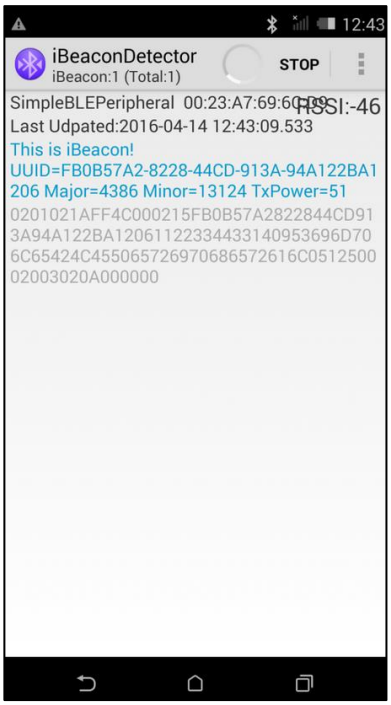
3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                       30
#define RSI_BLE_PWR_SAVE_OPTIONS              0
```

Note:
rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs module would be in Advertising state.
2. Open iBeaconDetector app in the Smartphone and do Scan.
3. In the App, Silicon Labs module device would appear with the name configured in the macro **RSI_BLE_LOCAL_NAME (Ex: "ibeacon")** or sometimes observed as "**SimpleBLEPeripheral**".
4. After successful scan, User can see the Silicon Labs device advertised data i.e. UUID, Maximum Number, Minimum Number and Tx Power in iBeaconDetector application.



4.9 LE Privacy

Feature Overview

Bluetooth LE supports a feature that reduces the ability to track an LE device over a period of time by changing the Bluetooth device address on a frequent basis, called Privacy of that particular device.

The device address of the remote device referred to as the private address will be resolved by local device in order to connect to that device. The private address is generated by using Identity Resolving Key (IRK) exchange in between devices during SMP bonding procedure. Our local device will add the remote devices in one Resolving list(to maintain remote device identity addresses) along with that IRK's and enable the Resolution, sets privacy mode and connect to the remote device with remote identity address.

Overview

This application demonstrates how to configure device with privacy feature by organizing resolving list and resolution process and how to connect to remote Peripheral device.

Sequence of Events

This Application explains user how to:

- Set a local name to the device
- Scan devices
- Connection to remote device

- SMP level connection
- Exchange of IRK's and store them
- Disconnect with remote device
- Add remote device to resolve list with identity address
- Get resolve list size
- Set resolution enable and time out
- Set privacy mode
- Connect remote device with identity address
- Start Encryption instead of SMP repairing

> **Note:**
>
> If both devices having resolution enable, enhanced connection event will come for any privacy mode. if remote device is without resolution, privacy mode should be device privacy mode.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device which supports privacy feature(Generally phones with the nRF Connect application)

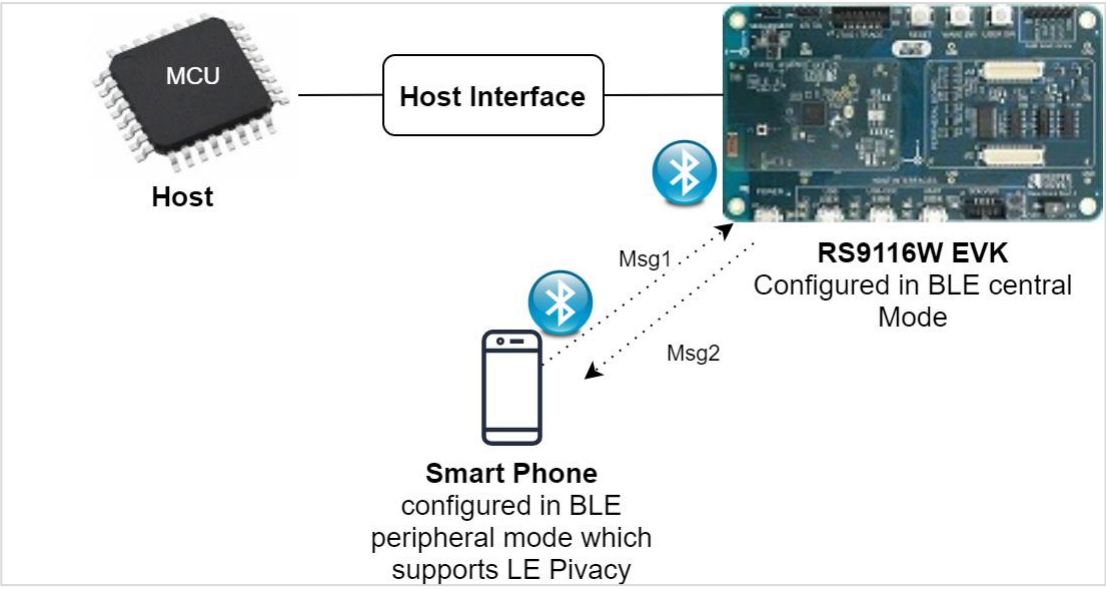


Figure 20: Setup Diagram for Privacy Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_privacy.c*** file and update/modify following macros:

   RSI_BLE_DEVICE_NAME refers the name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_DEVICE_NAME                    "BLE_PRIVACY"
```

RSI_DEVICE_ROLE refers the role of the Silicon Labs device.

```
#define RSI_DEVICE_ROLE                        "RSI_SLAVE"
```

Note:

RSI_DEVICE_ROLE should be RSI_MASTER

RSI_BLE_SMP_IO_CAPABILITY refers the IO capability of Silicon Labs device for SMP,
RSI_BLE_SMP_PASSKEY is smp passkey key from Silicon Labs device

```
#define RSI_BLE_SMP_IO_CAPABILITY              0x00
#define RSI_BLE_SMP_PASSKEY                    0
```

Following are the **non-configurable** macros in the application.
Following are the event numbers for connection, Disconnection, and enhanced connection events.

```
#define RSI_APP_EVENT_ADV_REPORT               0x00
#define  RSI_BLE_CONN_EVENT                    0x01
#define  RSI_BLE_DISCONN_EVENT                 0x02
#define  RSI_BLE_SMP_REQ_EVENT                 0x03
#define  RSI_BLE_SMP_RESP_EVENT                0x04
#define  RSI_BLE_SMP_PASSKEY_EVENT             0x05
#define  RSI_BLE_SMP_FAILED_EVENT              0x06
#define  RSI_BLE_ENCRYPT_STARTED_EVENT         0x07
#define  RSI_BLE_SMP_PASSKEY_DISPLAY_EVENT     0x08
#define  RSI_BLE_SC_PASSKEY_EVENT              0x09
#define  RSI_BLE_LTK_REQ_EVENT                 0x0A
#define  RSI_BLE_SECURITY_KEYS_EVENT           0x0B
#define  RSI_BLE_ENHANCE_CONNECTED_EVENT       0x0C
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver.

```
#define BT_GLOBAL_BUFF_LEN                     15000
```

RSI_BLE_DEV_ADDR_TYPE refers the address type of the remote device.

```
#define RSI_BLE_REMOTE_ADDR_TYPE               LE_PUBLIC_ADDRESS
```

RSI_BLE_DEV_ADDR_1 refers remote device address which has to connect.

```
#define RSI_BLE_REMOTE_ADDR                    "00:15:83:6A:64:17"
```

RSI_REMOTE_DEVICE_NAME refers the name of the Remote device to which Silicon Labs module initiate connection.

```
#define RSI_REMOTE_DEVICE_NAME                 "BLE_SIMPLE_PRIVACY"
```

RSI_BLE_SET_RESOLVABLE_PRIV_ADDR_TOUT refers resolution timeout , that is the length of time the Controller uses a Resolvable Private Address before a new resolvable private address is generated and starts being used.

```
#define RSI_BLE_SET_RESOLVABLE_PRIV_ADDR_TOUT          120
```

Process type refers the operation to be performed on the resolving list. valid configurations for the process type are

```
#define RSI_BLE_ADD_TO_RESOLVE_LIST                    1
#define RSI_BLE_REMOVE_FROM_RESOLVE_LIST               2
#define RSI_BLE_CLEAR_RESOLVE_LIST                     3
```

RSI_BLE_PRIVACY_MODE refers the privacy mode of local device

```
#define RSI_BLE_PRIVACY_MODE                           RSI_BLE_DEVICE_PRIVACY_MODE
```

RSI_BLE_RESOLVING_LIST_SIZE refers the resolving list size of Silicon Labs device.

```
#define RSI_BLE_RESOLVING_LIST_SIZE                    5
```

2. Open **rsi_ble_config.h** file and update/modify following macros

RSI_BLE_DEV_ADDR_RESOLUTION_ENABLE refers address resolution is enable or not. It should be 1 to enable privacy feature.

```
#define RSI_BLE_DEV_ADDR_RESOLUTION_ENABLE             1
```

RSI_BLE_ADV_DIR_ADDR_TYPE refers the address type of remote device which use while advertising.

```
#define RSI_BLE_ADV_DIR_ADDR_TYPE                      LE_PUBLIC_ADDRESS
```

RSI_BLE_ADV_DIR_ADDR refers to which device the local device will advertise with private address, it should be one of the device in resolve list.

```
#define RSI_BLE_ADV_DIR_ADDR                           "00:15:83:6A:64:17"
```

```
#define RSI_BLE_PWR_INX                                30
#define RSI_BLE_PWR_SAVE_OPTIONS                       0
```

3. Open **rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                                RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                            FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                              RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                     TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                     FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                 EXT_FEAT_384K_MODE
#define RSI_BAND                                       RSI_BAND_2P4GHZ
```

Note:

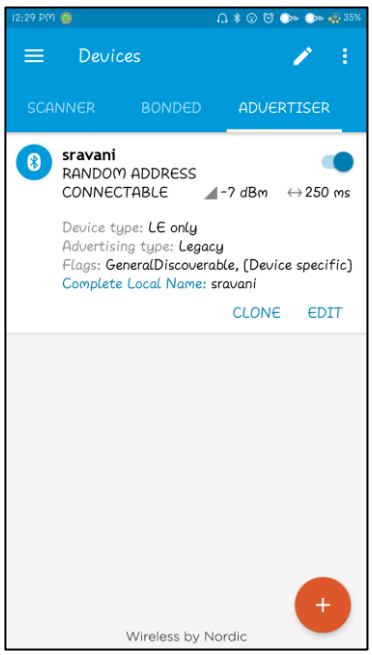
rsi_wlan_config.h and **rsi_ble_config.h** files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

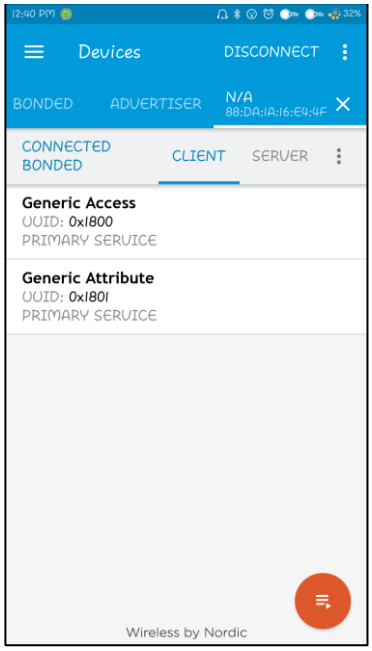
1. After the program gets executed, Silicon Labs module will be in Scanning state.
2. Advertise remote device,
3. If Silicon Labs module get device with name configured RSI_BLE_LOCAL_NAME or bd address with address configured in RSI_BLE_REMOTE_ADDR in results ,local device will try to connect with remote device.
4. After connection Silicon Labs device which is in master mode will initiate SMP request
5. Give response from Remote device and passkey
6. After successful SMP connection security keys will exchanged between Remote device and Silicon Labs device.
7. Silicon Labs device will add remote device's IRK's and local IRK's in to resolve list and enable resolution
8. Give disconnect from remote device and keep in advertise mode.
9. Now Silicon Labs module will try to connect to remote device with identity address.
10. After successful connection, Silicon Labs module will give start encryption instead of SMP repairing.
11. Encryption will be enabled on both sides.

Please find following screen shots for reference.

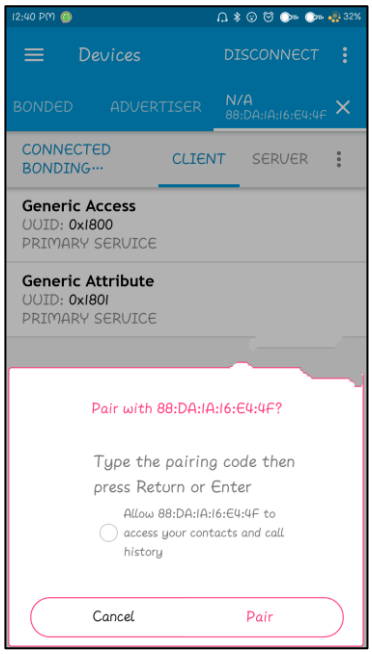
1. Advertise remote device



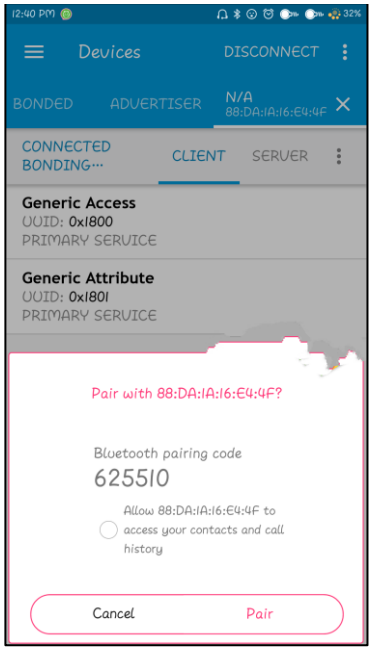
2. Connection in between Silicon Labs module and remote device



3. Pairing confirmation



4. Passkey confirmation



4.10 Proximity Profile

Overview

This application demonstrates how to configure Proximity as GATT server in BLE peripheral mode and explains how to do indicate operation with GATT server from connected remote device using GATT client.
In this Application, Proximity GATT server configures with Proximity service with indicate characteristic UUID. When connected remote device writes data to writable characteristic UUID, WiseConnect device receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID.

Sequence of Events

This Application explains user how to:

- Create Proximity service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smartphone
- Give the information by writing alert values to connected device write handle

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- RS9116W EVK
- BTLE Central device

Note:

Install Light blue App for tablet for ipad mini and BLE scanner or nRF connect app for android smart phone.

user can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

user can download the nRF connect App from the following link
https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

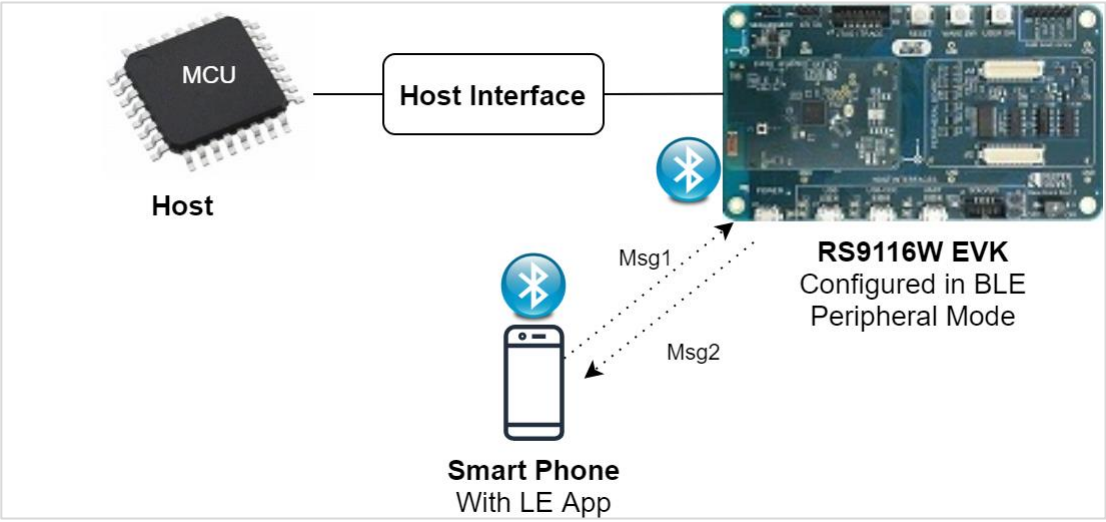


Figure 21: Setup Diagram for Proximity Profile Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_proximity.c*** file and update/modify following macros,

   RSI_BLE_DEVICE_NAME refers the name of the RS9116W device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_PROXIMITY_REPORTER                  "BLE_PROXIMITY_REPORTER"
```

   Following are the event numbers for advertising, connection and Disconnection events,

```
#define RSI_BLE_CONN_EVENT                              0x01
#define RSI_BLE_DISCONN_EVENT                           0x02
#define RSI_LINK_LOSS_WRITE_EVENT                       0x03
#define RSI_BLE_IMME_ALT_WRITE_EVENT                    0x04
```

   Following are the non-configurable macros in the application.
   BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

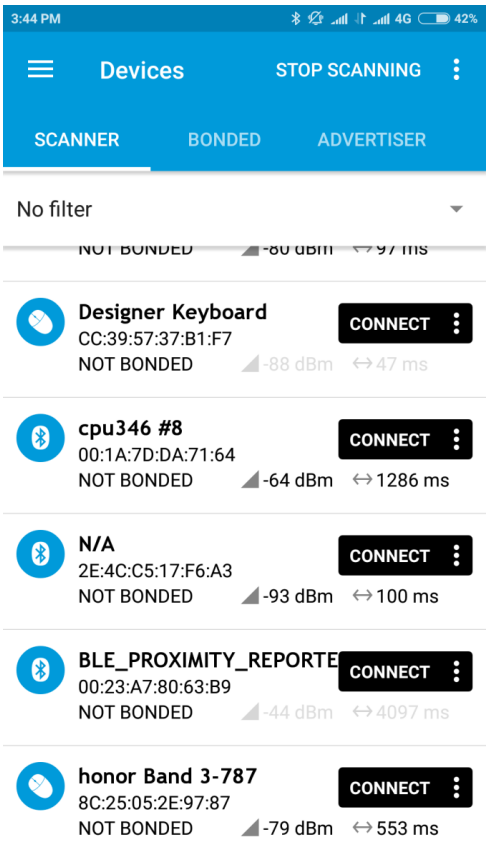
```
#define RSI_BLE_PWR_INX                          30
#define RSI_BLE_PWR_SAVE_OPTIONS                 0
```

Note:

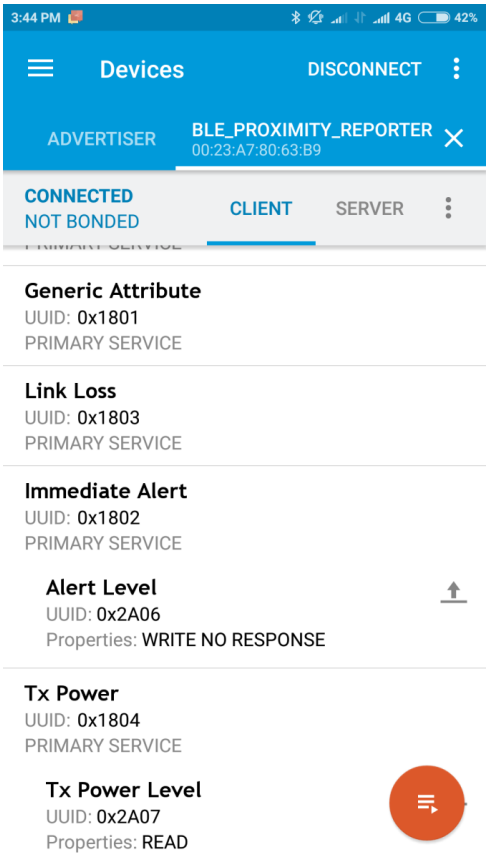
rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

- After the program gets executed, RS9116W EVK will be in Advertising state.
- Open a LE SCANNER App in the Smartphone and do the scan.
- In the App, RS9116W EVK device will appear with the name configured in the macro **RSI_BLE_APP_PROXIMITY_REPORTER(Ex:"BLE_PROXIMITY_REPORTER").**
- Please refer the given below images for write operation from remote device GATT client and check in our application at write handle.



- Initiate connection from the App.
- Please refer the given below images for write operation from remote device GATT client and check in our application at write handle.



4.11 Long Read

Overview

This application demonstrates how a GATT client device accesses a GATT server device for long read, means when user wants to read more than MTU(minimum of local and remote devices MTU's) size of data.Silicon Labs module acts as a GATT client/server(based on user configuration) and explains reads/writes .Client role is initialized with Battery Service. Server role is initialized with a custom service.

Sequence of Events

This Application explains user how to:

- Advertising in SLAVE role
- Connects with remote device in MASTER role.
- Loop back the data came from the remote device
- Read request to the remote device

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device in case of Silicon Labs module as master
- BTLE central device in case of Silicon Labs module as slave

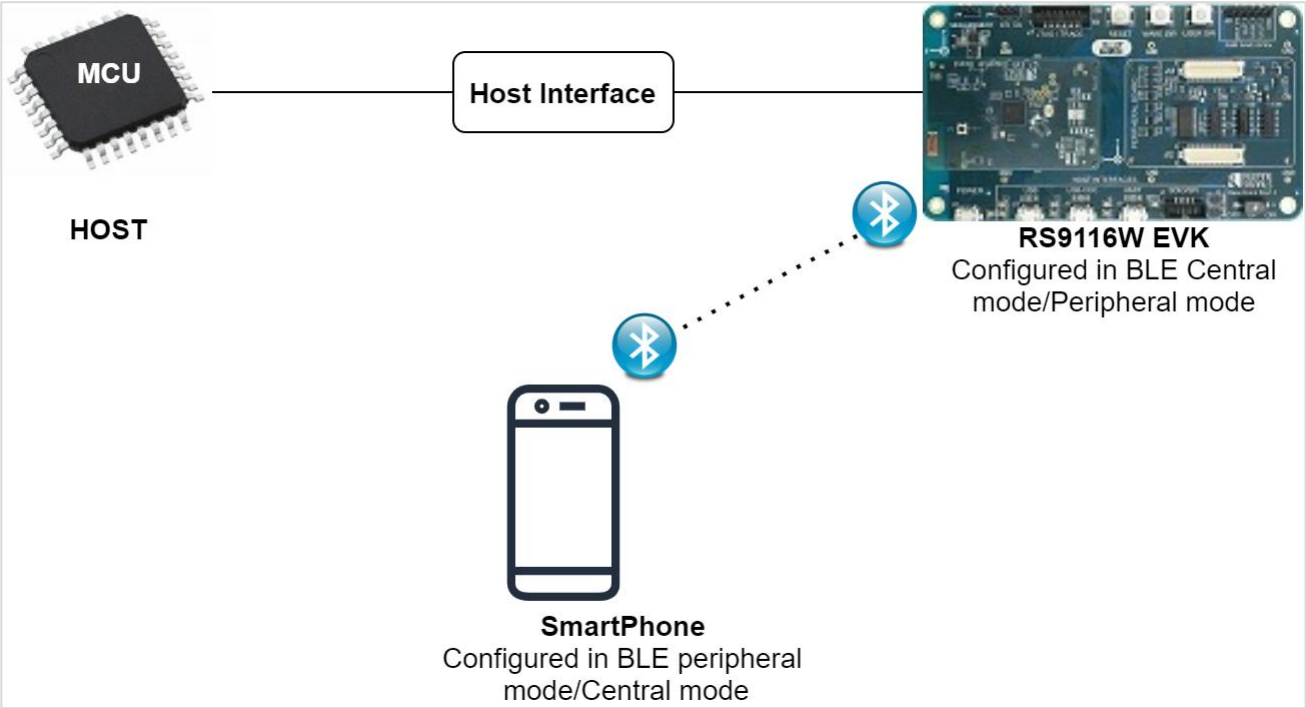


Figure 22: Setup Diagram for BLE Long Read Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_long_read.c*** file and configure the below macros.

   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                    LE_PUBLIC_ADDRESS
```

   Valid configurations are

   LE_RANDOM_ADDRESS

   LE_PUBLIC_ADDRESS

> **Note:**
> Depends on the remote device, address type will be changed.

   RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                         "00:1A:7D:DA:71:13"
```

   RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                   "REDPINE_DEV"
```

Note:
Silicon Labs module can connect to remote device by referring either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

GATT_ROLE refers the GATT role of the Silicon Labs device

```
#define SERVER                                  0
#define CLIENT                                  1
#define GATT_ROLE                               SERVER
```

Valid configurations of GATT_ROLE are:

SERVER

CLIENT

BT_GLOBAL_BUFF_LEN refers the Number of bytes required for the Application and the Driver.

```
#define BT_GLOBAL_BUFF_LEN                      15000
```

RSI_BLE_CHAR_SERV_UUID refers standard attribute type of characteristic service

RSI_BLE_CLIENT_CHAR_UUID refers standard attribute type of client characteristic configuration descriptor.

#define RSI_BLE_CHAR_SERV_UUID 0x2803

```
#define RSI_BLE_CLIENT_CHAR_UUID                0x2902
```

RSI_BLE_NEW_SERVICE_UUID refers service uuid when module acts as server

RSI_BLE_ATTRIBUTE_1_UUID refers characteristic uuid when module acts as server

```
#define RSI_BLE_NEW_SERVICE_UUID                0xAABB
#define RSI_BLE_ATTRIBUTE_1_UUID                0x1AA1
```

RSI_BLE_NEW_CLIENT_SERVICE_UUID refers service present in GATT server LE device.

RSI_BLE_CLIENT_ATTRIBUTE_1_UUID refers characteristic present under above service in GATT server LE device.

```
#define RSI_BLE_NEW_CLIENT_SERVICE_UUID         0x180F
#define RSI_BLE_CLIENT_ATTRIBUTE_1_UUID         0x2A19
```

RSI_BLE_MAX_DATA_LEN refers the maximum attribute value length.

```
#define RSI_BLE_MAX_DATA_LEN                    20
```

Following are event numbers for specific events

```
#define RSI_BLE_CONNN_EVENT                    1
#define RSI_BLE_DISCONN_EVENT                  2
#define RSI_BLE_GATT_WRITE_EVENT               3
#define RSI_BLE_READ_REQ_EVENT                 4
#define RSI_BLE_MTU_EVENT                      5
#define RSI_BLE_GATT_PROFILE_RESP_EVENT        6
#define RSI_BLE_GATT_CHAR_SERVICES_RESP_EVENT  7
```

Following are the **non-configurable** macros in the application.

```
#define RSI_BLE_ATT_PROPERTY_READ              0x02
#define RSI_BLE_ATT_PROPERTY_WRITE             0x08
#define RSI_BLE_ATT_PROPERTY_NOTIFY            0x10
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                        RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                    FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                      RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP             TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP             FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP         EXT_FEAT_384K_MODE
#define RSI_BAND                               RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                        30
#define RSI_BLE_PWR_SAVE_OPTIONS               0
```

Executing the Application

1. After the program gets executed,
2. In Client mode,
   Silicon Labs module tries to connect with remote device as specified by RSI_BLE_REMOTE_BD_ADDRESS or RSI_REMOTE_DEVICE_NAME.
3. After connecting, mtu size will be updated. As per mtu (maximum transmit unit) size, read requests will be happen from Silicon Labs device
4. In Server mode,
   Silicon Labs module will advertise,
5. Initiate connection from master.
6. After connecting, mtu size will be updated. As per mtu size, write will be happen from Silicon Labs device
7. In either role: If mtu size is of 100 bytes, module can read upto 98 bytes, write upto 97 bytes
8. For the data more than 20 bytes, application has to store value and send using gatt_read_response function whenever remote device reads some handle's data.

Note:

For read request event to be raised auth_read flag in rsi_ble_add_char_val_att function need to be set.
Based on GATT_ROLE configurable macro, this application will be act as a GATT server or GATT client device.

4.12 BLE Lr 2Mbps

Overview

This application connects as a Central and can be used to update the phy rates .The PHY update Procedure is used to change the Transmit or receive PHYs, or both. The procedure can be initiated either on a request by the Host or autonomously by the Link Layer. Either the master or the slave may initiate this procedure at any time after entering the Connection State. The procedure can be initiated either on a request by the Host or autonomously by the Link Layer. Either the master or the slave may initiate this procedure at any time after entering the Connection State.

Sequence of Events

This Application explains how to use below commands:

- Connect with remote BTLE peripheral device.
- Read PHY rate.
- Set PHY rate
- PHY update complete event will appear.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device which supports BLE long range and 2Mbps feature.

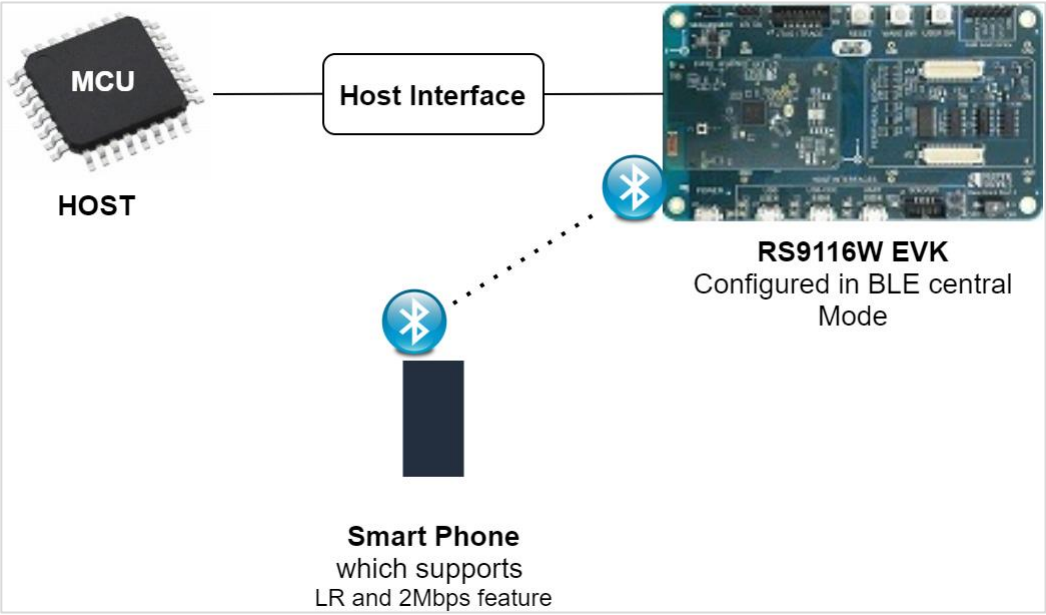


Figure 23: Setup Diagram for Long Range and 2Mbps Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_lr_2mbps.c*** file and update/modify following macros,
   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                    LE_PUBLIC_ADDRESS
```

   Valid configurations are
   LE_RANDOM_ADDRESS

   LE_PUBLIC_ADDRESS

Note:

Depends on the remote device, address type will be changed.

RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                                "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labsdevice has to connect

```
#define RSI_REMOTE_DEVICE_NAME                          "REDPINE_DEV"
```

Note:

Silicon Labs module can connect to remote device by referring either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

Following are the event numbers for advertising, connection and Disconnection events,

```
#define RSI_APP_EVENT_ADV_REPORT                        0
#define RSI_APP_EVENT_CONNECTED                         1
#define RSI_APP_EVENT_DISCONNECTED                      2
#define RSI_APP_EVENT_PHY_UPDATE_COMPLETE               3
```

Following are the non-configurable macros in the application.
BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                                 30
#define RSI_BLE_PWR_SAVE_OPTIONS                        0
```

Note:

rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

1. Configure the remote BLE device in advertising mode
2. after the program gets executed, Silicon Labs device tries to connect with the remote device specified in **RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME** macro.
3. Observe that the connection is established between the desired device and Silicon Labs device.
4. After connection, Silicon Labs device will read PHY rate of the remote device and set PHY rate of the remote device.
5. Observe PHY update complete event after setting PHY rate.

4.13 BLE DataLength

Overview

This application acts as a Central role and can be used to set data length with connected remote device. Ble Data Packet Length Extension refers to increase in the Packet Data Unit (PDU) size from 27 to 251 bytes. This is the amount of data sent during connection events. Both the master and slave can initiate this procedure at any time after entering the Connection.

Sequence of Events

This Application explains sequence of below commands:

- Connect with remote BTLE peripheral device.
- Set data length.
- Data length change event will appear.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device which supports data length extension feature.

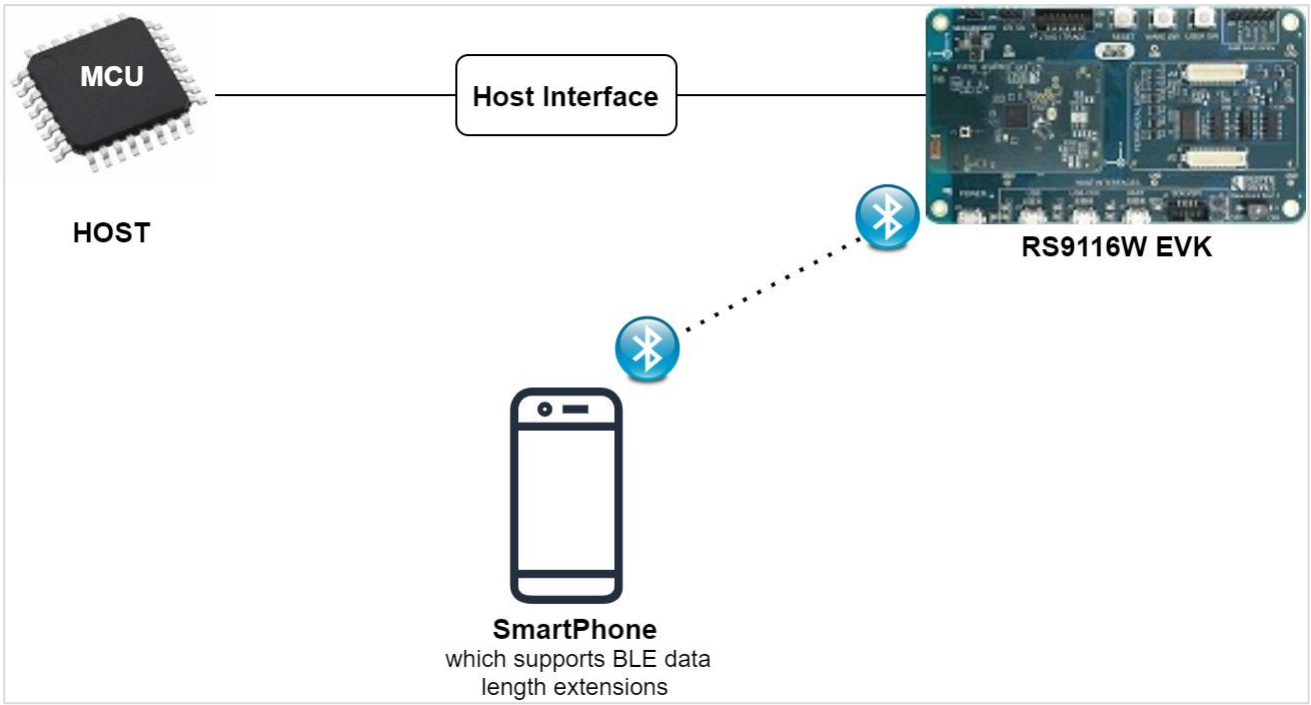


Figure 24: Setup Diagram for BLE Data Length Extensions

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_datalength.c*** file and update/modify following macros,
   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                          LE_PUBLIC_ADDRESS
```

Based on the address of the advertising device, Valid configurations are

LE_RANDOM_ADDRESS

LE_PUBLIC_ADDRESS

> **Note:**
> Depends on the remote device, address type will be changed.

RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                               "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                        "REDPINE_DEV"
```

> **Note:**
> Silicon Labs module can connect to remote device by referring either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

Following are the event numbers for advertising, connection and Disconnection events,

```
#define RSI_APP_EVENT_ADV_REPORT                       0
#define RSI_APP_EVENT_CONNECTED                        1
#define RSI_APP_EVENT_DISCONNECTED                     2
#define RSI_APP_EVENT_DATA_LENGTH_CHANGE               3
#define RSI_BLE_MTU_EVENT                          4
```

Following are the macros for setting data length(TX length and TX time)

```
#define TX_LEN                                         0x001e
#define TX_TIME                                        0x01f4
```

Following are the non-configurable macros in the application.
BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                             15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_384K_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                         30
#define RSI_BLE_PWR_SAVE_OPTIONS                0
```

> **Note:**
>
> rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

1. Configure the remote device in peripheral mode and put it in advertising mode.
2. After the program gets executed, Silicon Labs device tries to connect with the remote device specified in **RSI_BLE_DEV_ADDR** or **RSI_REMOTE_DEVICE_NAME** macro.
3. Observe that the connection is established between the desired device and Silicon Labs device.
4. After connection Silicon Labs device will set data length of the remote device.
5. Observe data length change event after setting data length.

4.14 LE-L2CAP Conn

Overview

This application demonstrates the l2cap connection oriented channel connection.

Sequence of Events

- Configure the device in Master mode
- Scan from remote slave device
- Connect with the remote device.
- Configure required psm value and connect with the remote device.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device which supports L2CAP connection based flow control feature

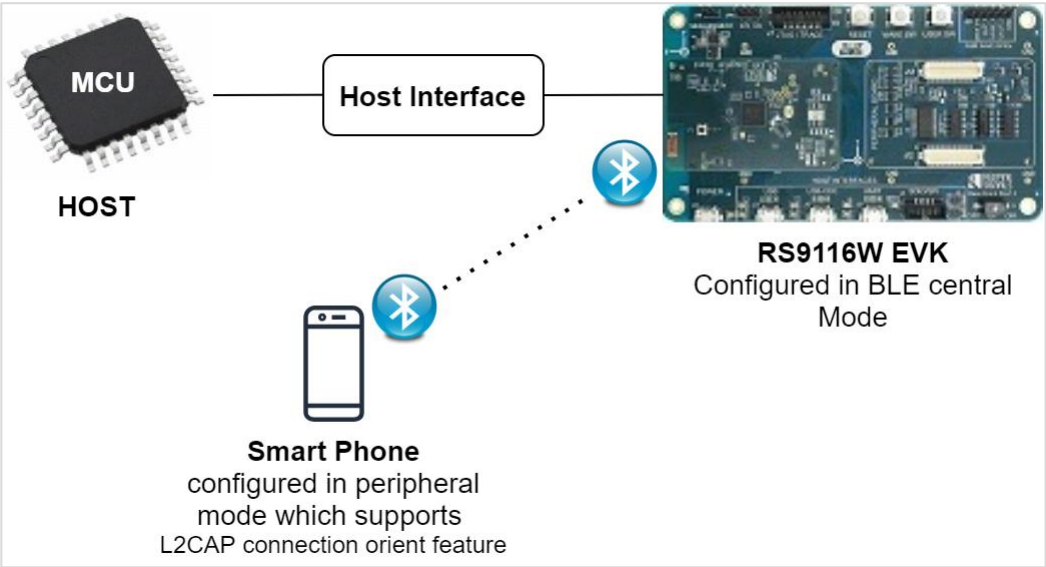


Figure 25: Setup Diagram For BLE-L2CAP Connection Based Flow Control Example

Configuration and Steps for Execution

Configuring the Application

1. **Open *rsi_ble_cbfc.c*** file and update/modify following macros:

   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                    LE_PUBLIC_ADDRESS
```

   Valid configurations are

   LE_RANDOM_ADDRESS

   LE_PUBLIC_ADDRESS

> **Note:**
>
> Depends on the remote device, address type will be changed.

   RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                         "00:1A:7D:DA:71:13"
```

   RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                   "REDPINE_DEV"
```

> **Note:**
>
> Silicon Labs module can connect to remote device by referring either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

   Following are the **non-configurable** macros in the application.

Following are the event numbers for events.

```
#define RSI_APP_EVENT_ADV_REPORT                    0
#define RSI_APP_EVENT_CONNECTED                     1
#define RSI_APP_EVENT_DISCONNECTED                  2
#define RSI_APP_EVENT_CBFC_CONN_REQ                 3
#define RSI_APP_EVENT_CBFC_CONN_CMPL                4
#define RSI_APP_EVENT_CBFC_RX_DATA                  5
#define RSI_APP_EVENT_CBFC_DISCONN                  6
```

RSI_BLE_PSM_VALUE to set PSM value

```
#define RSI_BLE_PSM_VALUE                           0x23
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                      15000
```

2. Open **sapis/include/rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP              EXT_FEAT_384K_MODE
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                             30
#define RSI_BLE_PWR_SAVE_OPTIONS                    0
```

Note:

rsi_wlan_config.h and rsi_ble_config.h files are already set with the desired configuration in respective example folders user need not change for each example

Executing the Application

1. Configure Remote device in advertising mode.
2. Compile and launch the application.
3. After the program gets executed, Silicon Labs module would try to connect with the device with the address specified in the macro **RSI_BLE_DEV_ADDR** or **RSI_REMOTE_DEVICE_NAME**
4. Observe that the connection is established between the desired device and Silicon Labs module.
5. Now l2cap channel connection can be initiated with desired psm value.
6. After the connection got established data can be exchanged between them.
7. If required l2cap disconnection command can be sent to disconnect the connection with required psm value.

4.15 Battery Service

Overview

This application demonstrates how to configure GATT server in BLE peripheral mode, how to configure GATT client in BLE central mode, explains how to do read & notify operations with GATT server from connected remote device using GATT client and explains how to get GATT information from remote GATT server in case of our module as client.
In this Application, Battery Service GATT server configures with Battery service with notification characteristic UUID. When connected remote device writes data to writable characteristic UUID, Silicon Labs device receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID and sends notifications to the connected device (or) remote device can read the same data using read characteristic UUID if notification enabled on client side.
Battery Service GATT client will get Battery service (primary service) , Battery level service (characteristic service), and descriptors(client characteristic configuration and characteristic presentation format) information from the remote GATT server. If remote device supports notify, our module will enable notify property and will be notified by the remote GATT server when value changed.

Sequence of Events

Server

This mode explains user how to:

- Create Battery service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smartphone
- Give the notifications to connected device

Client

This mode explains user how to:

- Connect to remote device based on BD address given
- Getting primary service information
- Getting characteristic services information
- Getting descriptors information
- Enable notify based on client characteristic configuration(if remote GATT server supports notify property)
- Receive notifications from remote GATT server when value changed

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE supported Smart phone with GATT client in case of Silicon Labs module as GATT server
- BTLE supported Smart phone with GATT Battery server in case of Silicon Labs as GATT client

Note:

Install Light blue App for tablet for ipad mini and BLE scanner or nRF connect app or BLE Peripheral Simulator(act like GATT server)
for android smart phone.
user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

user can download the nRF connect App from the following link
https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

user can download the BLE Peripheral Simulator App from the following link

https://play.google.com/store/apps/details?id=io.github.webbluetoothcg.bletestperipheral

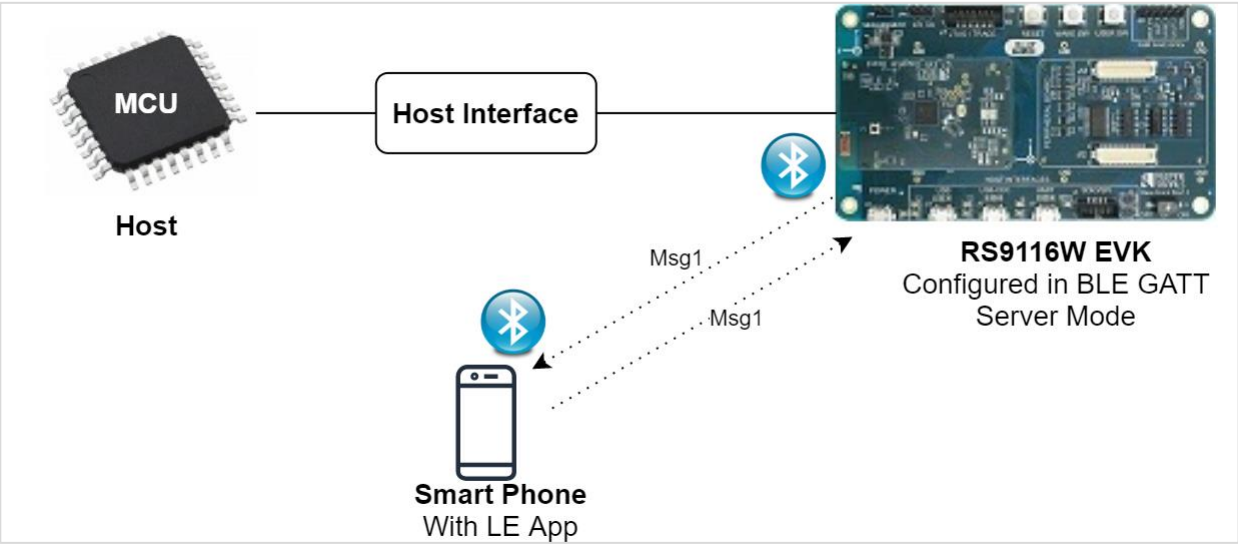


Figure 26: Setup Diagram For Battery Service Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_battery_service.c*** file and update/modify following macros,

   RSI_BLE_BATTERY_SERVICE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_BATTERY_SERVICE_UUID                    0x180F
```

   RSI RSI_BLE_BATTERY_LEVEL_UUID refers to the attribute type of the first attribute under this above primary service.

```
#define RSI RSI_BLE_BATTERY_LEVEL_UUID                  0x2A19
```

   RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#define RSI_BLE_MAX_DATA_LEN                            20
```

   BLE_BATTERY_SERVICE refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_LOCAL_DEVICE_NAME                           "BLE_BATTERY_SERVICE"
```

   GATT_ROLE refers the role of the Silicon Labs module to be selected.
   If user configure **SERVER**, Silicon Labs module will act as GATT SERVER, means will add battery service profile.
   If user configure **CLIENT**, Silicon Labs module will act as GATT CLIENT, means will connect to remote GATT server and get services.

```
#define GATT_ROLE                                   "SERVER"
```

If user configure **CLIENT** role following macros should be configured.
RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                       LE_PUBLIC_ADDRESS
```

Valid configurations are

LE_RANDOM_ADDRESS

LE_PUBLIC_ADDRESS

> **Note:**
> Depends on the remote device, address type will be changed.

RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                            "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                      "REDPINE_DEV"
```

> **Note:**
> Silicon Labs module can connect to remote device by referring either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

Following Characteristic Presentation Format fields

```
#define RSI_BLE_UINT8_FORMAT                        0x04
#define RSI_BLE_EXPONENT                            0x00
#define RSI_BLE_PERCENTAGE_UNITS_UUID               0x27AD
#define RSI_BLE_NAME_SPACE                          0x01
#define RSI_BLE_DESCRIPTION                         0x010B
```

Following are the non configurable macros related to client characteristic configuration.

```
#define RSI_BLE_NOTIFY_VALUE                        0x01
#define RSI_BLE_INDICATE_VALUE                      0x02
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                      0x2803
```

RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CLIENT_CHAR_UUID                    0x2902
```

RSI_BLE_CHAR_PRESENTATION_FORMATE_UUID refers to the attribute type of the characteristic presentation format descriptor to be added in a service.

```
#define RSI_BLE_CHAR_PRESENTATION_FORMATE_UUID      0x2904
```

Following are the Macros for the GATT properties

```
#define  RSI_BLE_ATT_PROPERTY_READ                  0x02
#define  RSI_BLE_ATT_PROPERTY_WRITE                 0x08
#define  RSI_BLE_ATT_PROPERTY_NOTIFY                0x10
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                          15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP              EXT_FEAT_384K_MODE
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                             30
#define RSI_BLE_PWR_SAVE_OPTIONS                    0
```

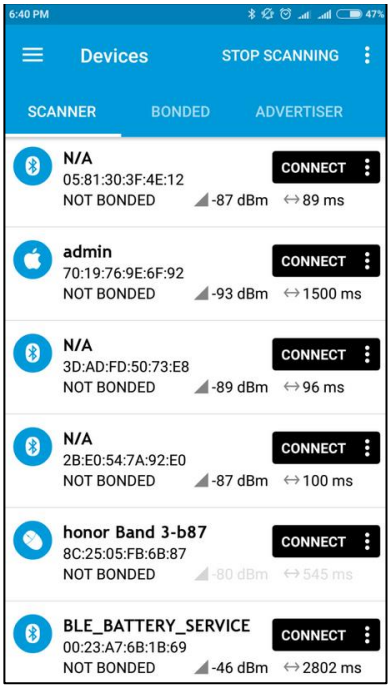
Note:

rsi_wlan_config.h and **rsi_ble_config.h** files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

Server role

1. After the program gets executed, Silicon Labs module will be in Advertising state.
2. Open a nRFConnect App and do the scan.
3. In the App, Silicon Labs module will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT (Ex: "BLE_BATTERY_SERVICE")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".



4. Initiate connection from the App.
5. After successful connection, nRFConnect displays the supported services of Silicon Labs module.
6. Select the attribute service which is added **RSI_BLE_BATTERY_SERVICE_UUID**

   (Ex: 0x180F).
7. Enable Notify for the characteristic **RSI_BLE_BATTERY_LEVEL_UUID**

   (Ex: 0x2A19). So that GATT server Notifies when value updated in that particular attribute.
8. Silicon Labs module send the Battery Service battery level data to the attribute **RSI_BLE_BATTERY_LEVEL_UUID** (Ex: 0x2A19) of the remote device and will Notifies the GATT client (remote device).
9. **RSI_BLE_CHAR_PRESENTATION_FORMATE_UUID** will describe the value by its fields as shown in fig.
10. Please refer the given below images for Notify operation from remote device GATT client.

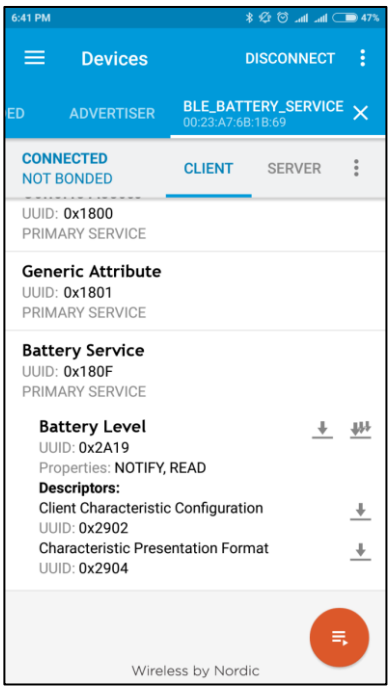


Figure 27: Battery Service and its Characteristic Service (Battery Level)

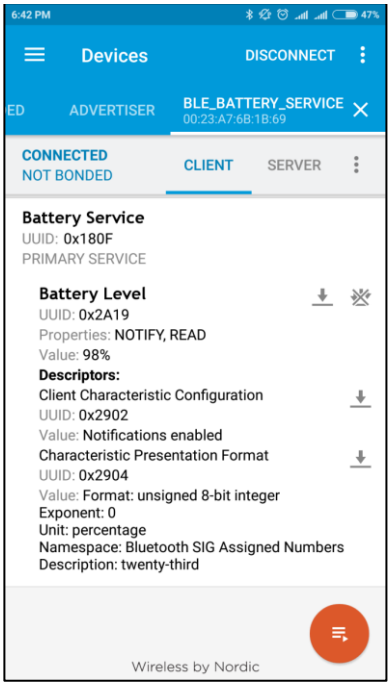
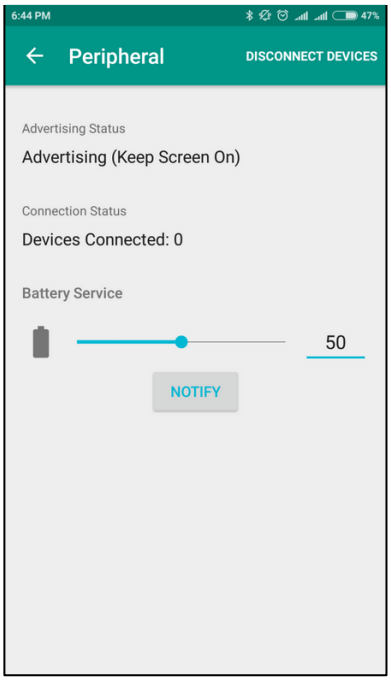


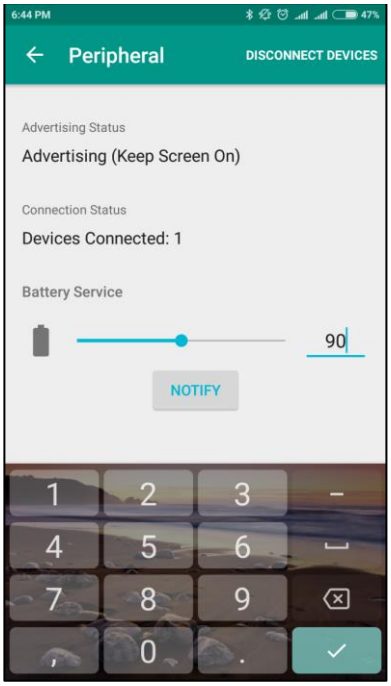
Figure 28: Client Characteristic Configuration and Characteristic Presentation Format

Client role



1. Advertise a LE device which supports Battery Service.

2. After the program gets executed, Silicon Labs module will connect to that remote device based on given BD address or name
3. After successful connection Silicon Labs module will read the services from the remote GATT server.
4. If remote device support notify property Silicon Labs module will enable notify, and ready to receive notifications from remote device.
5. Whenever GATT server changes value and notifies that Silicon Labs module will receive that value.



4.16 Health Thermometer

Overview

This application demonstrates how to configure Health thermometer as GATT server in BLE peripheral mode and explains how to do indicate operation with GATT server from connected remote device using GATT client.
In this Application, Health thermometer GATT server configures with health thermometer service with indicate characteristic UUID. When connected remote device writes data to writable characteristic UUID, Silicon Labs device receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID and sends indications to the connected device (or) remote device can read the same data using read characteristic UUID if indication enabled on client side.

Sequence of Events

This Application explains user how to:

- Create Health thermometer service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smartphone
- Give the indications to connected device

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE supported Smart phone with GATT client

Note:

Install Light blue App for tablet for ipad mini and BLE scanner or nRF connect app for android smart phone.

user can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

user can download the nRF connect App from the following link
https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

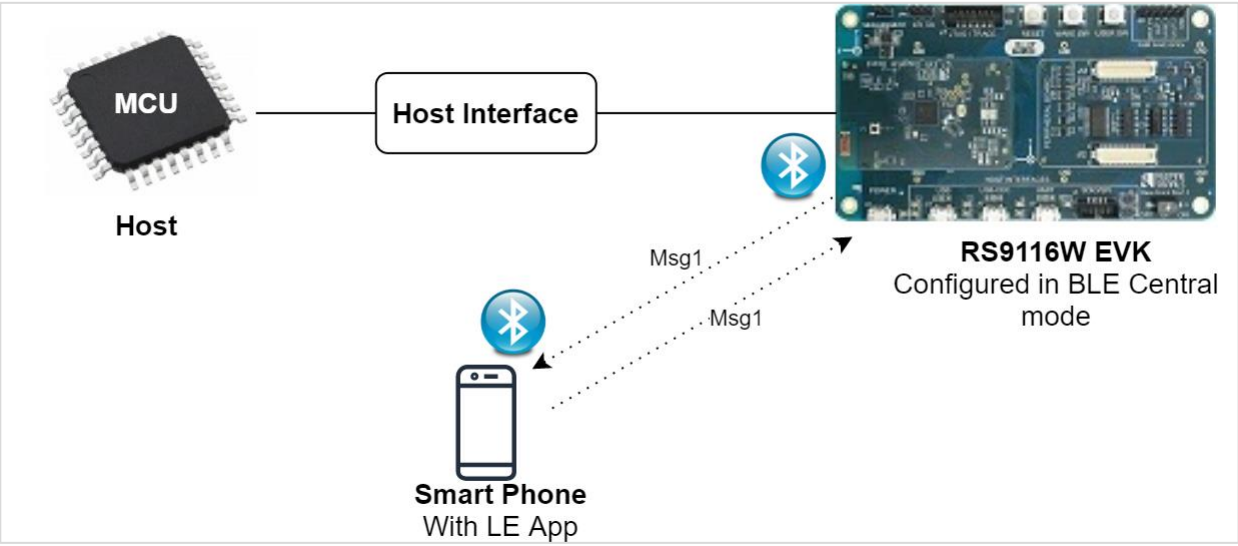


Figure 29: Setup Diagram For Health Thermometer Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_health_thermometer.c*** file and update/modify following macros,

   RSI_BLE_HEALTH_THERMOMETER_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_HEALTH_THERMOMETER_UUID                 0x1809
```

   RSI_BLE_TEMPERATURE_MEASUREMENT_UUID refers to the attribute type of the first attribute under this service (**RSI_BLE_HEALTH_THERMOMETER_UUID**).

   RSI_BLE_TEMPERATURE_TYPE_UUID refers to the attribute type of the second attribute under this service (**RSI_BLE_HEALTH_THERMOMETER_UUID**).

   RSI_BLE_INTERMEDIATE_TEMPERATURE_UUID refers to the attribute type of the second attribute under this service (**RSI_BLE_HEALTH_THERMOMETER_UUID**).

```
#define RSI_BLE_TEMPERATURE_MEASUREMENT_UUID            0x2A1C
#define RSI_BLE_TEMPERATURE_TYPE_UUID                   0x2A1D
#define RSI_BLE_INTERMEDIATE_TEMPERATURE_UUID           0x2A1E
```

   RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#define RSI_BLE_MAX_DATA_LEN                            20
```

   BLE_HEALTH_THERMOMETER refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_SIMPLE_CHAT                         "BLE_HEALTH_THERMOMETER"
```

Following are the **non-configurable** macros in the application.

RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.

RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                          0x2803
#define RSI_BLE_CLIENT_CHAR_UUID                        0x2902
```

Following are the GATT properties

```
#define RSI_BLE_ATT_PROPERTY_READ                       0x02
#define RSI_BLE_ATT_PROPERTY_WRITE                      0x08
#define RSI_BLE_ATT_PROPERTY_NOTIFY                     0x10
#define RSI_BLE_ATT_PROPERTY_INDICATE                   0x20
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

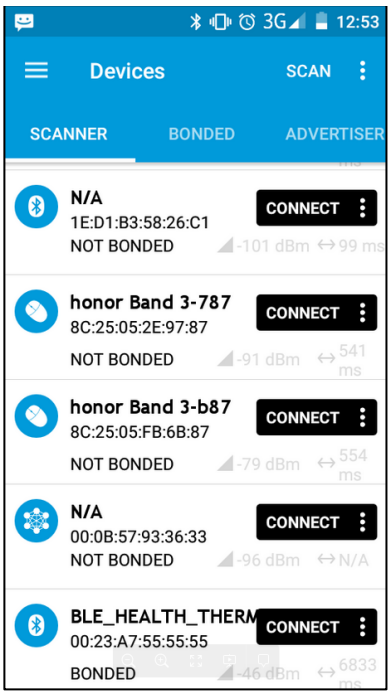
3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                                 30
#define RSI_BLE_PWR_SAVE_OPTIONS                        0
```

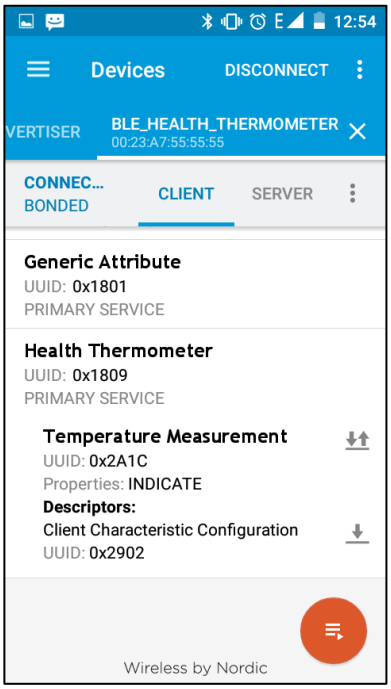
> **Note:**
>
> rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

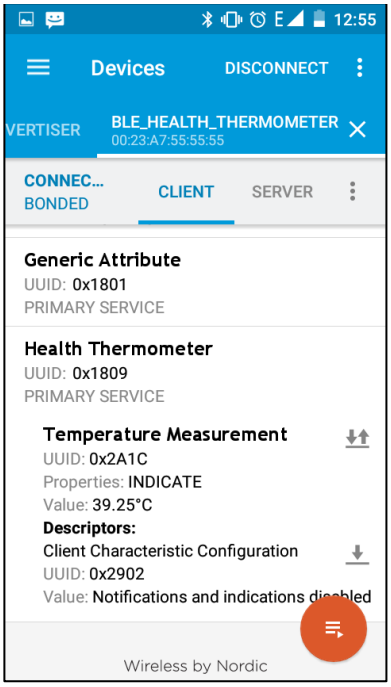
Executing the Application

1. After the program gets executed, Silicon Labs module will be in Advertising state.
2. Open a LE App in the Smartphone and do the scan.
3. In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT (Ex: "BLE_HEALTH_THERMOMETER")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".



4. Initiate connection from the App.
5. After successful connection, LE scanner displays the supported services of Silicon Labs module.
6. Select the attribute service which is added **RSI_BLE_HEALTH_THERMOMETER_UUID** (Ex: 0x1809).
7. Enable Indicate for the characteristic **RSI_BLE_TEMPERATURE_MEASUREMENT_UUID** (Ex: 0x2A1C). So that GATT server indicates when value updated in that particular attribute.
8. Silicon Labs module sends the health thermometer temperature measurement data to the attribute **RSI_BLE_TEMPERATURE_MEASUREMENT_UUID** (Ex: 0x2A1C) of the remote device and indicates the GATT client (remote device).
9. Please refer the given below images for indicate operation from remote device GATT client.





4.17 BLE PER

Overview

This application demonstrates how to configure the necessary parameters to start transmitting or receiving BLE PER packets.

Sequence of Events

This Application explains user how to:
Configure the BLE PER TX or RX mode.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Remote device for the BLE TX or RX mode on the other side

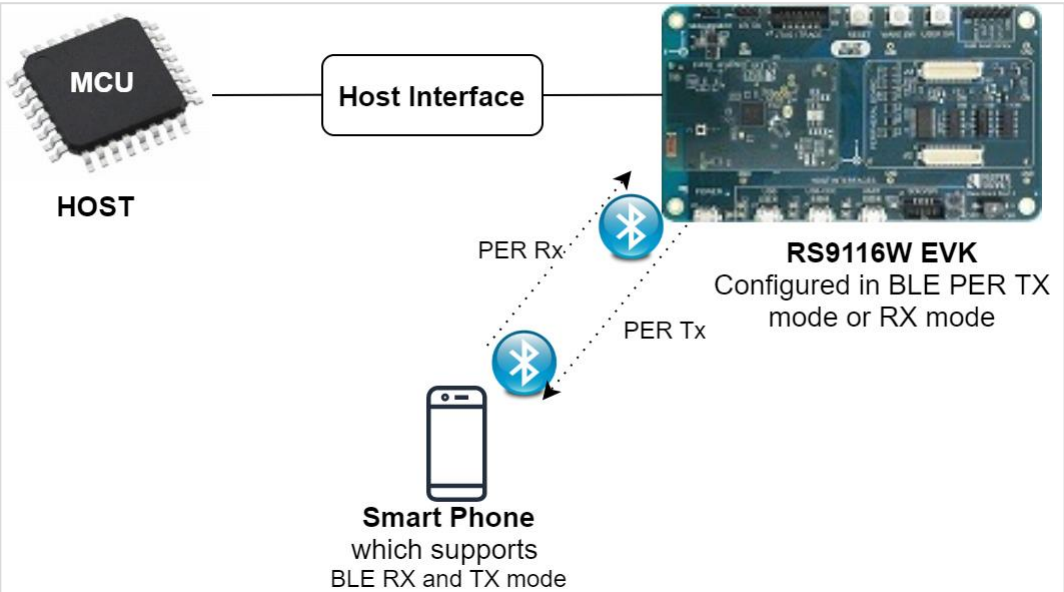


Figure 30: Setup Diagram For BLE PER Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_per.c*** file and update/modify following macros,
   RSI_CONFIG_PER_MODE refers configuration mode BT PER TX or RX

```
#define RSI_CONFIG_PER_MODE                 RSI_BLE_PER_TRANSMIT_MODE
```

OR

```
#define RSI_CONFIG_PER_MODE                 RSI_BLE_PER_RECEIVE_MODE
```

CMD_ID refers the command id for transmit or receive

```
#define BLE_TRANSMIT_CMD_ID                 0x13
#define BLE_RECEIVE_CMD_ID                  0x14
```

PAYLOAD_TYPE refers type of payload to be transmitted

```
#define DATA_PRBS9                          0x00
#define DATA_FOUR_ONES_FOUR_ZEROES          0x01
#define DATA_ALT_ONES_AND_ZEROES            0x02
#define DATA_PRSB15                         0x03
#define DATA_ALL_ONES                       0x04
#define DATA_ALL_ZEROES                     0x05
#define DATA_FOUR_ZEROES_FOUR_ONES          0x06
#define DATA_ALT_ZEROES_AND_ONES            0x07
```

LE_CHNL_TYPE: advertising channel - 0 data channel - 1

```
#define LE_ADV_CHNL_TYPE                    0
#define LE_DATA_CHNL_TYPE                   1
```

PACKET_LEN: Length of the packet, in bytes, to be transmitted. Packet length range 0 to 255.

```
#define BLE_TX_PKT_LEN                  32
```

BLE_RX_CHNL_NUM- Receive channel index, as per the Bluetooth standard.i.e, 0 to 39
BLE_TX_CHNL_NUM - Transmit channel index, as per the Bluetooth standard. i.e, 0 to 39

```
#define BLE_RX_CHNL_NUM                 10
#define BLE_TX_CHNL_NUM                 10
```

BLE_PHY_RATE: ,2Mbps - 2 , 125Kbps - 4, 500Kbps - 8

```
#define LE_ONE_MBPS                     1
#define LE_TWO_MBPS                     2
#define LE_125_KBPS_CODED               4
#define LE_500_KBPS_CODED               8
#define BLE_PHY_RATE                    LE_ONE_MBPS
```

SCRAMBLER_SEED: Initial seed to be used for whitening. It should be set to '0' in order to disable whitening.

```
#define SCRAMBLER_SEED                  0
```

TX_MODE: Burst mode - 0 Continuous mode - 1

```
#define BURST_MODE                      0
#define CONTINUOUS_MODE                 1
```

HOPPING TYPE : no hopping -0 fixed hopping - 1 random hopping - 2

```
#define NO_HOPPING                      0
#define FIXED_HOPPING                   1
#define RANDOM_HOPPING                  2
```

ANT_SEL : onchip antenna - 2 u.f.l - 3

```
#define ONBOARD_ANT_SEL                 2
#define EXT_ANT_SEL                     3
```

RF_TYPE : External RF – 0 Internal RF – 1

```
#define BLE_EXTERNAL_RF                 0
#define BLE_INTERNAL_RF                 1
```

RF CHAIN: Select the required RF chain

```
#define NO_CHAIN_SEL                    0
#define WLAN_HP_CHAIN_BIT               0
#define WLAN_LP_CHAIN_BIT               1
#define BT_HP_CHAIN_BIT                 2
#define BT_LP_CHAIN_BIT                 3
```

pll_mode : PLL_MODE0 – 0 PLL_MODE1 – 1

```
#define PLL_MODE_0                      0
#define PLL_MODE_1                      1
```

LOOP_BACK_MODE : enable 1 or disable 0 #define LOOP_BACK_MODE_DISABLE 0

```
#define LOOP_BACK_MODE_ENABLE                1
```

Following are the non-configurable macros in the application.
BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                   15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                      RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                  FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                    RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP           TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP           FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP       EXT_FEAT_384K_MODE
#define RSI_BAND                             RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                      30
#define RSI_BLE_PWR_SAVE_OPTIONS             0
```

> **Note:**
>
> rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example

Executing the Application

1. After the program gets executed, Silicon Labs module starts BLE PER transmit or BLE PER receive.
2. For receiving purpose use BT dongle and keep it in BLE PER RX mode.
3. Check for BLE PER stats whatever configured values are affecting or not.

4.18 Blood Pressure

Definitions and Acronyms

- BLS - Blood Pressure Service
- GATT - Generic Attribute Profile
- UUID - Universal unique identifier
- BLE - Bluetooth low energy
- BD - Bluetooth Device

Overview

This application demonstrates how to configure GATT server in BLE peripheral mode, how to configure GATT client in BLE central mode, explains how to do read, notify and indicate operations with GATT server from connected remote device using GATT client and explains how to get GATT information from remote GATT server in case of our module as client.
In this Application, Blood Pressure Service GATT server configures with Blood Pressure service with notification characteristic UUID. When connected remote device writes data to writable characteristic UUID, Silicon Labs device

receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID and sends notifications to the connected device (or) remote device can read the same data using read characteristic UUID if notification enabled on client side.
Blood Pressure Service GATT client will get Blood Pressure service (primary service) , Blood Pressure Measurement service (characteristic service), and descriptors(client characteristic configuration and characteristic presentation format) information from the remote GATT server. If remote device supports notify, our module will enable notify property and will be notified by the remote GATT server when value changed.

Sequence of Events

Server Role

This mode explains user how to:

- Create Blood Pressure service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smart phone
- Give the notifications to connected device.

Client Role

This mode explains user how to:

- Connect to remote device based on BD address given
- Getting primary service information
- Getting characteristic services information
- Getting descriptors information
- Enable notify based on client characteristic configuration(if remote GATT server supports notify property)
- Receive notifications from remote GATT server when value changed.

Application Setup

The Silicon Labs module WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE supported Smart phone with GATT client in case of our module as GATT server
- BTLE supported Smart phone with GATT Blood Pressure server in case of our module as GATT client

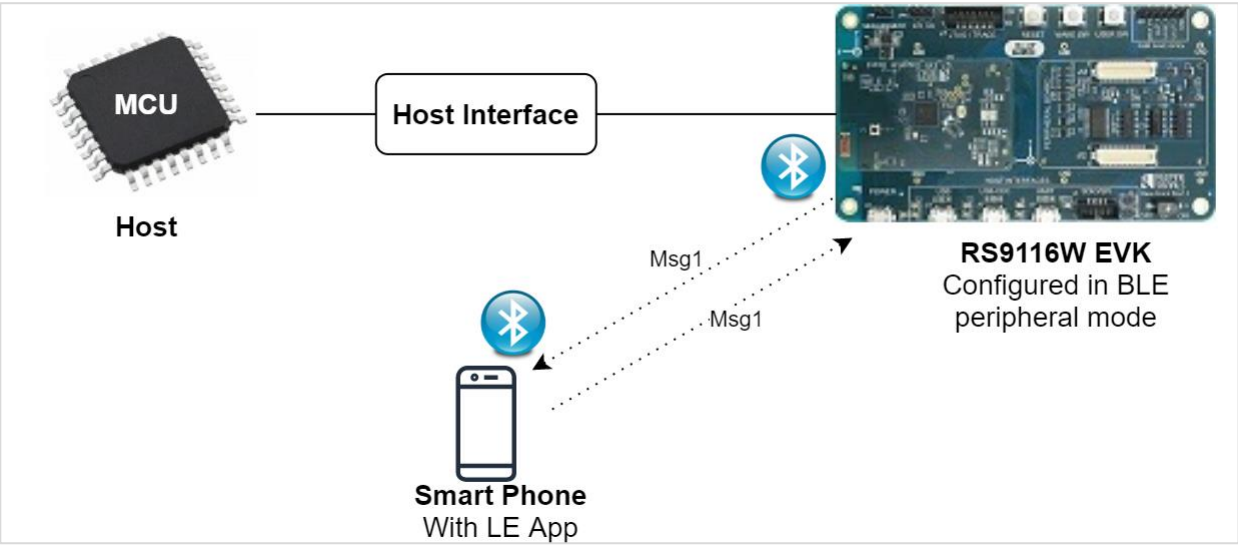


Figure 31: Setup Diagram For Boold Pressure Service Example

Note:

Install Light blue App for tablet for ipad mini and BLE scanner or nRF connect app

user can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

user can download the nRF connect App from the following link
https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_blood_pressure.c*** file and update/modify following macros.
   RSI_BLE_BLOOD_PRESSURE_SERVICE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_BLOOD_PRESSURE_SERVICE_UUID              0x1810
```

RSI_BLE_BLOOD_PRESSURE_MEASUREMENT_UUID refers to the attribute type of the first attribute under this above primary service.
RSI_BLE_INTERMEDIATE_CUFF_PRESSURE_UUID refers to the attribute type of the second attribute under this above primary service.
RSI_BLE_BLOOD_PRESSURE_FEATURE_UUID refers to the attribute type of the third attribute under this above primary service.

```
#define  RSI_BLE_BLOOD_PRESSURE_MEASUREMENT_UUID         0x2A35
#define  RSI_BLE_INTERMEDIATE_CUFF_PRESSURE_UUID         0x2A36
#define  RSI_BLE_BLOOD_PRESSURE_FEATURE_UUID             0x2A49
```

RSI_BLE_APP_BLOOD_PRESSURE refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_BLOOD_PRESSURE                       "BLS"
```

GATT_ROLE refers the role of the Silicon Labs module to be selected.
If user configure **SERVER**, Silicon Labs module will act as GATT SERVER, means will add blood pressure service profile.
If user configure **CLIENT**, Silicon Labs module will act as GATT CLIENT, means will connect to remote GATT server and get services.

```
#define GATT_ROLE                                   SERVER
```

If user configure **CLIENT** role following macros should be configured.
RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                       LE_PUBLIC_ADDRESS
```

Valid configurations are
LE_RANDOM_ADDRESS

LE_PUBLIC_ADDRESS

RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                            "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                      "REDPINE_DEV"
```

Note:

User can configure either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

Following are the non configurable macros related to attribute properties.

```
#define  RSI_BLE_ATT_PROPERTY_READ                  0x02
#define  RSI_BLE_ATT_PROPERTY_WRITE                 0x08
#define  RSI_BLE_ATT_PROPERTY_WRITE_WITHOUT_RESP    0x04
#define  RSI_BLE_ATT_PROPERTY_NOTIFY                0x10
#define  RSI_BLE_ATT_PROPERTY_INDICATE              0x20
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.
RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                      0x2803
#define RSI_BLE_CLIENT_CHAR_UUID                    0x2902
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                          15000
```

Following are the event numbers for advertising, connection and Disconnection events,

```
#define  RSI_APP_EVENT_ADV_REPORT                0x00
#define  RSI_BLE_CONN_EVENT                      0x01
#define  RSI_BLE_DISCONN_EVENT                   0x02
#define  RSI_BLE_GATT_WRITE_EVENT                0x03
#define  RSI_BLE_GATT_PROFILE_RESP_EVENT         0x04
#define  RSI_BLE_GATT_CHAR_SERVICES_RESP_EVENT   0x05
#define  RSI_BLE_GATT_CHAR_DESC_RESP_EVENT       0x06
#define  RSI_BLE_GATT_PROFILE                    0x07
#define  RSI_BLE_GATT_CHAR_SERVICES              0x08
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                          RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                      FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                        RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP               TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP               FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP           EXT_FEAT_384K_MODE
#define RSI_BAND                                 RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros, #define RSI_BLE_PWR_INX 8

```
#define RSI_BLE_PWR_INX                          30
#define RSI_BLE_PWR_SAVE_OPTIONS                 0
```

> **Note:** rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

Server role

1. After the program gets executed, Silicon Labs module will be in Advertising state.
2. Open a nRFConnect App and do the scan. (see Figure 2)
3. In the App, Silicon Labs module will appear with the name configured in the macro **RSI_BLE_APP_BLOOD_PRESSURE (Ex: "BLS")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".
4. Initiate connection from the App.
5. After successful connection, nRFConnect displays the supported services of Silicon Labs module.
6. Select the attribute service which is added **RSI_BLE_BLOOD_PRESSURE_SERVICE_UUID** (Ex: 0x1810). (see Figure 3 and 4)
7. Enable Notify for the characteristic **RSI_BLE_BLOOD_PRESSURE_MEASUREMENT_UUID** (Ex: 0x2A35). So that GATT server Notifies when value updated in that particular attribute.(see Figure 5)
8. Silicon Labs module send the Blood pressure measurement value to the attribute **RSI_BLE_BLOOD_PRESSURE_MEASUREMENT_UUID** (Ex: 0x2A35) of the remote device and will indicates the GATT client (remote device).
9. **RSI_BLE_BLOOD_PRESSURE_FEATURE_UUID** will describe the value by its fields as shown in fig.

Following are the snapshots of smart phone nRFConnert App act as a client and Silicon Labs device as a server.

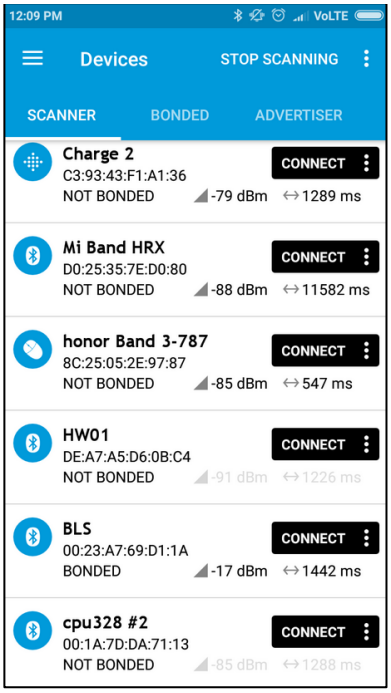


Figure 32: Scanning for BLS Device and Connecting to it

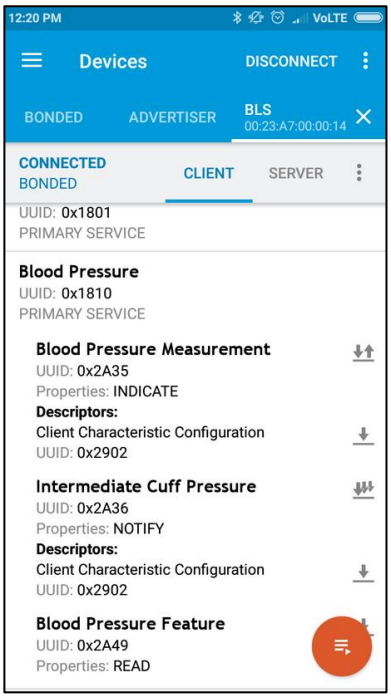


Figure 33: BLS and its Characteristic Discovery

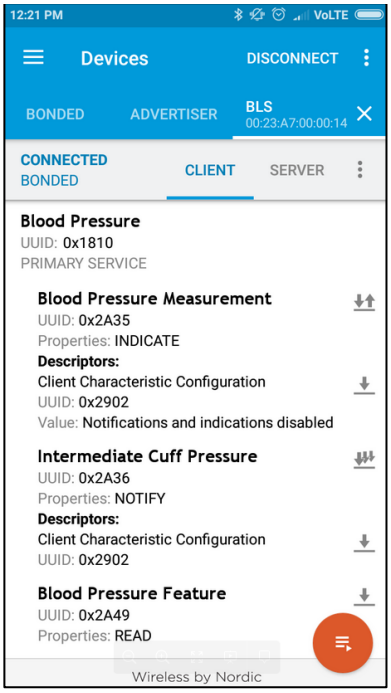


Figure 34: Client Characteristic Configuration (Indication Disable)

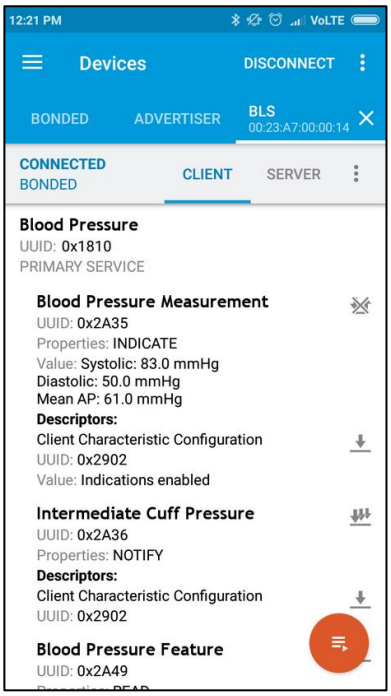


Figure 35: Blood Pressure Measurement Value (Indication Enable)

Client role

1. Advertise a LE device which supports Blood Pressure Service.
2. After the program gets executed, Silicon Labs module will connect to that remote device based on given BD address.
3. After successful connection Silicon Labs module will read the services from the remote GATT server.

4. If remote device support notify property Silicon Labs module will enable notify, and ready to receive notifications from remote device.
5. Whenever GATT server changes value and notifies that Silicon Labs module will receive that value.

4.19 Glucose

Definitions and Acronyms

- GLS - Glucose Service
- GATT - Generic Attribute Profile
- UUID - Universal unique identifier
- BLE - Bluetooth low energy
- BD - Bluetooth Device

Overview

This application demonstrates how to configure GATT server in BLE peripheral mode, how to configure GATT client in BLE central mode, explains how to do read, notify and indicate operations with GATT server from connected remote device using GATT client and explains how to get GATT information from remote GATT server in case of our module as client.
In this Application, Glucose Service GATT server configures with Glucose service with notification characteristic UUID. When connected remote device writes data to writable characteristic UUID, Silicon Labs device receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID and sends notifications to the connected device (or) remote device can read the same data using read characteristic UUID if notification enabled on client side.
Glucose Service GATT client will get Glucose service (primary service) , Glucose Measurement service (characteristic service), and descriptors(client characteristic configuration and characteristic presentation format) information from the remote GATT server. If remote device supports notify, our module will enable notify property and will be notified by the remote GATT server when value changed.

Sequence of Events

Server Role

This mode explains user how to:

- Create Glucose service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smart phone
- Give the notifications to connected device.

Client Role

This mode explains user how to:

- Connect to remote device based on BD address given
- Getting primary service information
- Getting characteristic services information
- Getting descriptors information
- Enable notify based on client characteristic configuration(if remote GATT server supports notify property)
- Receive notifications from remote GATT server when value changed.

Application Setup

The Silicon Labs module WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE supported Smart phone with GATT client in case of our module as GATT server
- BTLE supported Smart phone with GATT Glucose server in case of our module as GATT client

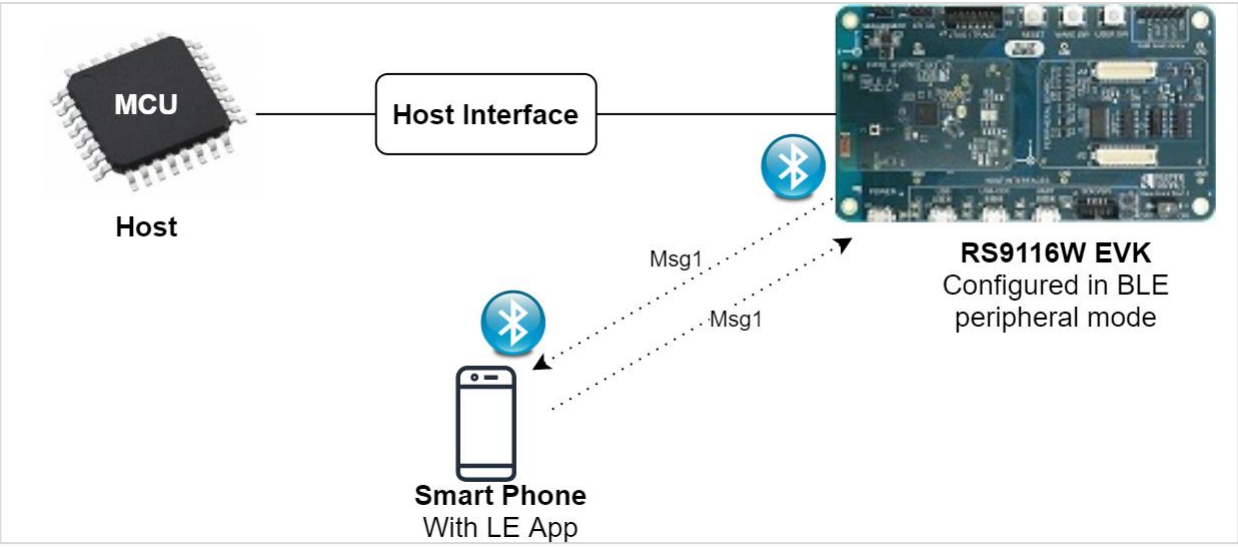


Figure 36: Setup Diagram For Glucose Service Example

Note:

Install Light blue App for tablet for ipad mini and BLE scanner or nRF connect app

user can download the BLE scanner App from the following link
https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=en

user can download the Light blue App from the following link
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en

user can download the nRF connect App from the following link
https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_glucose.c*** file and update/modify following macros,
   RSI_BLE_GLUCOSE_SERVICE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_GLUCOSE_SERVICE_UUID                    0x1808
```

RSI_BLE_GLUCOSE_MEASUREMENT_UUID refers to the attribute type of the first attribute under this above primary service.
RSI_BLE_GLUCOSE_MEASUREMENT_CONTEXT_UUID refers to the attribute type of the second attribute under this above primary service.
RSI_BLE_GLUCOSE_FEATURE_UUID refers to the attribute type of the third attribute under this above primary service.
RSI_BLE_RECORD_ACCESS_CONTROL_POINT_UUID refers to the attribute type of the fourth attribute under this above primary service.

```
#define  RSI_BLE_GLUCOSE_MEASUREMENT_UUID               0x2A18
#define  RSI_BLE_GLUCOSE_MEASUREMENT_CONTEXT_UUID       0x2A34
```

```
#define  RSI_BLE_GLUCOSE_FEATURE_UUID                    0x2A51
#define  RSI_BLE_RECORD_ACCESS_CONTROL_POINT_UUID        0x2A52
```

RSI_BLE_APP_GLUCOSE refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_GLUCOSE                              "GLS"
```

GATT_ROLE refers the role of the Silicon Labs module to be selected.
If user configure **SERVER**, Silicon Labs module will act as GATT SERVER, means will add Glucose service profile.
If user configure **CLIENT**, Silicon Labs module will act as GATT CLIENT, means will connect to remote GATT server and get services.

```
#define GATT_ROLE                                        SERVER
```

If user configure **CLIENT** role following macros should be configured.
RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                            LE_PUBLIC_ADDRESS
```

Valid configurations are

LE_RANDOM_ADDRESS

LE_PUBLIC_ADDRESS

RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                                 "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                           "REDPINE_DEV"
```

Note:

User can configure either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

Following are the non configurable macros related to attribute properties.

```
#define  RSI_BLE_ATT_PROPERTY_READ                       0x02
#define  RSI_BLE_ATT_PROPERTY_WRITE                      0x08
#define  RSI_BLE_ATT_PROPERTY_WRITE_WITHOUT_RESP         0x04
#define  RSI_BLE_ATT_PROPERTY_NOTIFY                     0x10
#define  RSI_BLE_ATT_PROPERTY_INDICATE                   0x20
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.
RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                           0x2803
#define RSI_BLE_CLIENT_CHAR_UUID                         0x2902
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                              15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                                 30
#define RSI_BLE_PWR_SAVE_OPTIONS                        0
```

Note:

rsi_wlan_config.h and **rsi_ble_config.h** files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

Server role

1. After the program gets executed, Silicon Labs module will be in Advertising state.
2. Open a nRFConnect App and do the scan.
3. In the App, Silicon Labs module will appear with the name configured in the macro **RSI_BLE_APP_GLUCOSE (Ex: "GLS")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**" (See Figure 2)
4. Initiate connection from the App.
5. After successful connection, nRFConnect displays the supported services of Silicon Labs module.
6. Select the attribute service which is added **RSI_BLE_GLUCOSE_SERVICE_UUID** (Ex: 0x1808).
7. Enable Notify for the characteristic **RSI_BLE_GLUCOSE_MEASUREMENT_UUID** (Ex: 0x2A18). So that GATT server Notifies when value updated in that particular attribute.
8. Silicon Labs module send the Battery Service battery level data to the attribute **RSI_BLE_GLUCOSE_MEASUREMENT_UUID** (Ex: 0x2A18) of the remote device and will Notifies the GATT client (remote device).
9. **RSI_BLE_GLUCOSE_FEATURE_UUID** will describe the value by its fields as shown in fig.
10. Figure 6 showing the use of **RSI_BLE_RECORD_ACCESS_CONTROL_POINT_UUID**.

Following are the snapshots of smart phone act as a client and Silicon Labs device as a server.

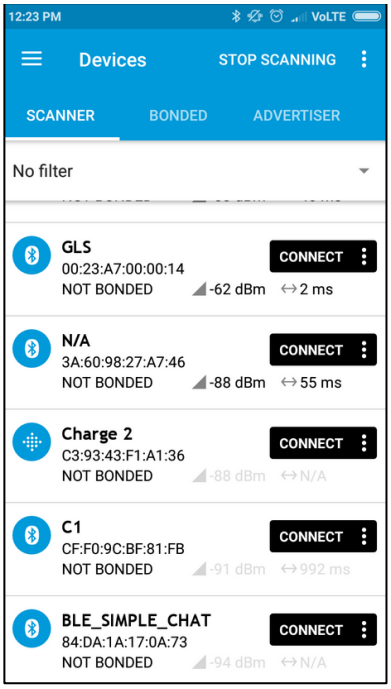


Figure 37: Scanning for GLS Device and Connecting to it

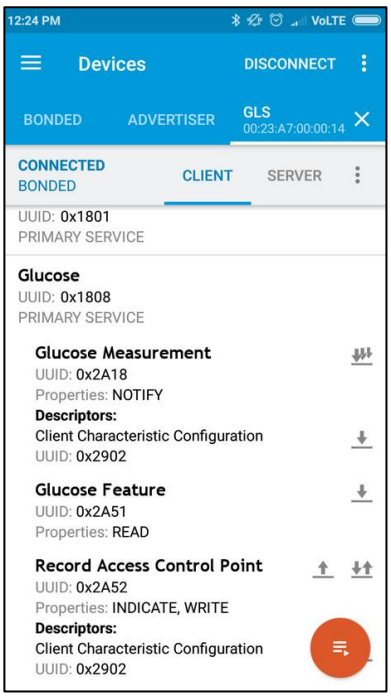


Figure 38: GLS and its Characteristic dDiscovery

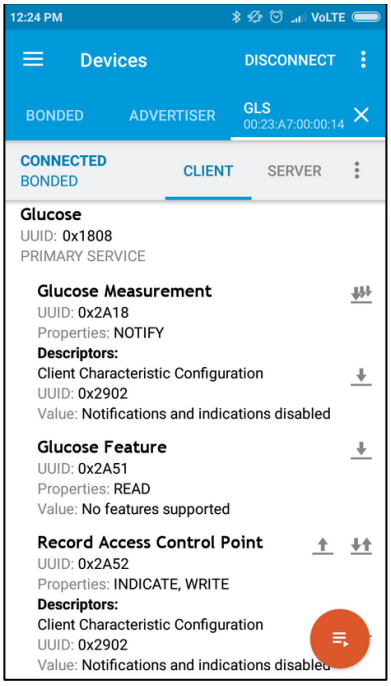


Figure 39: Client Characteristic Configuration (Indication Disable)

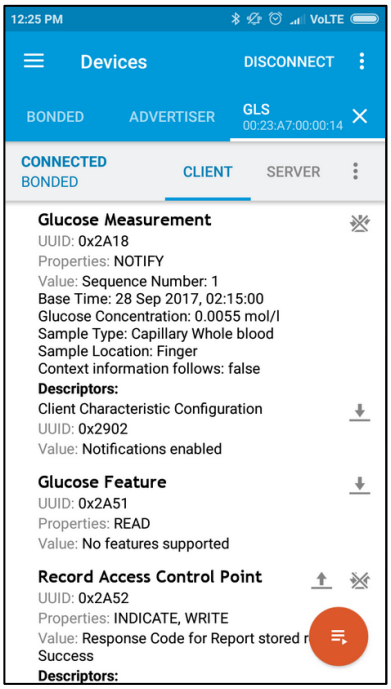


Figure 40: Glucose Measurement Value (Indication Enable)

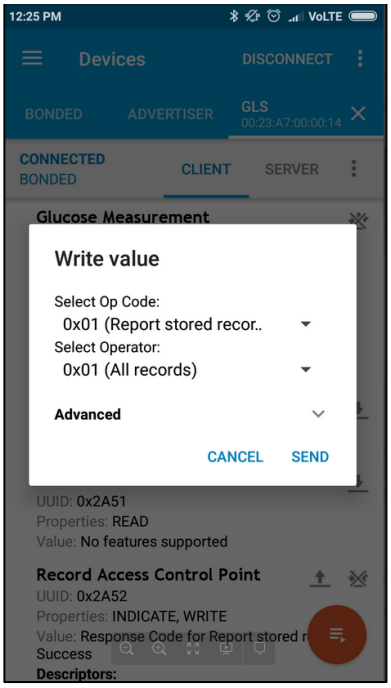


Figure 41: Get Records using Control Point Characteristic

Client role

1. Advertise a LE device which supports Glucose Service.
2. After the program gets executed, Silicon Labs module will connect to that remote device based on given BD address.
3. After successful connection Silicon Labs module will read the services from the remote GATT server.
4. If remote device support notify property Silicon Labs module will enable notify, and ready to receive notifications from remote device.
5. Whenever GATT server changes value and notifies that Silicon Labs module will receive that value.

4.20 HID On Gatt

Definitions and Acronyms

- HIDS - Human Interface Device Service
- GATT - Generic Attribute Profile
- UUID - Universal unique identifier
- BLE - Bluetooth low energy
- BD - Bluetooth Device

Overview

This application demonstrates how to configure GATT server in BLE peripheral mode, how to configure GATT client in BLE central mode, explains how to do read, notify and indicate operations with GATT server from connected remote device using GATT client and explains how to get GATT information from remote GATT server in case of our module as client.
In this Application, Human Interface Device Service GATT server configures with Human Interface Device service with notification characteristic UUID. When connected remote device writes data to writable characteristic UUID, Silicon Labs device receives the data which is received on writable characteristic UUID and writes the same data to readable characteristic UUID and sends notifications to the connected device (or) remote device can read the same data using read characteristic UUID if notification enabled on client side.
Human Interface Device Service GATT client will get Human Interface Device service (primary service) , Report Map (characteristic service), and descriptors(client characteristic configuration and report reference) information from the remote GATT server. If remote device supports notify, our module will enable notify property and will be notified by the remote GATT server when value changed.

Sequence of Events

Server Role

This mode explains user how to:

- Create Human Interface Device service
- Make the device to advertise
- Connect from remote BTLE device
- Receive the message from the connected peer/Smart phone
- Give the notifications to connected device.

Client Role

This mode explains user how to:

- Connect to remote device based on BD address given
- Getting primary service information
- Getting characteristic services information
- Getting descriptors information
- Enable notify based on client characteristic configuration(if remote GATT server supports notify property)
- Receive notifications from remote GATT server when value changed.

Application Setup

The Silicon Labs module WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE supported Smart phone with GATT client in case of our module as GATT server
- BTLE supported Smart phone with GATT Human Interface Device server in case of our module as GATT client

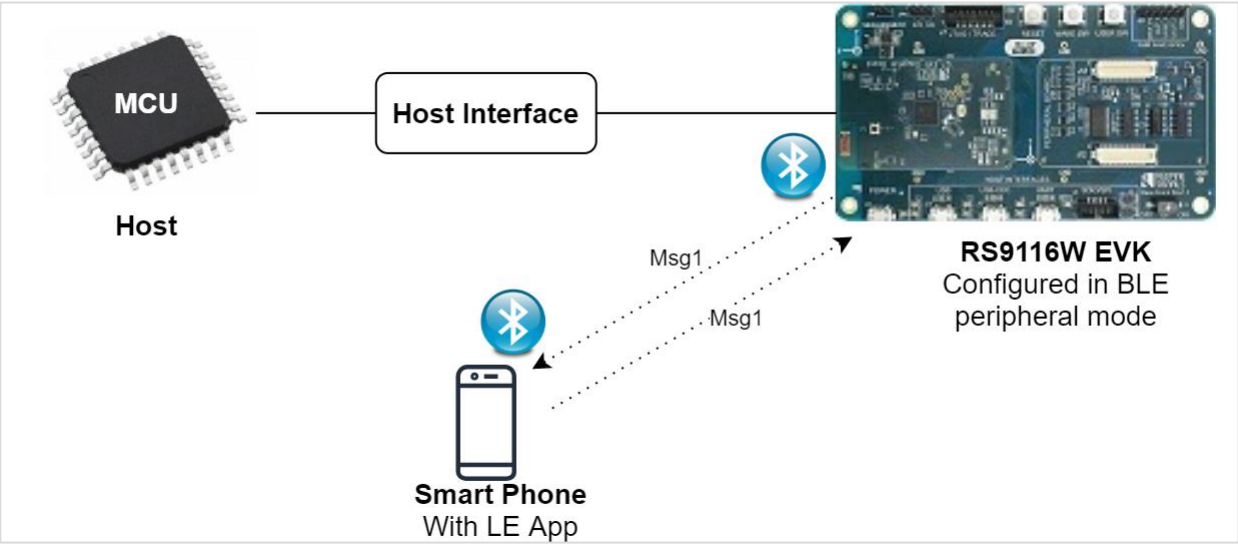


Figure 42: Setup Diagram For Human Interface Device Service Example

Note:

Use default Bluetooth application in smart phones which has BLE support.

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_hid.c*** file and update/modify following macros.
   RSI_BLE_HID_SERVICE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_HID_SERVICE_UUID                    0x1812
```

RSI_BLE_HID_PROTOCOL_MODE_UUID refers to the attribute type of the first attribute under this above primary service.
RSI_BLE_HID_REPORT_UUID refers to the attribute type of the second attribute under this above primary service.
RSI_BLE_HID_REPORT_MAP_UUID refers to the attribute type of the third attribute under this above primary service.
RSI_BLE_HID_INFO_UUID refers to the attribute type of the fourth attribute under this above primary service.
RSI_BLE_HID_CONTROL_POINT_UUID refers to the attribute type of the fifth attribute under this above primary service.

```
#define RSI_BLE_HID_PROTOCOL_MODE_UUID              0x2A4E
#define RSI_BLE_HID_REPORT_UUID                     0x2A4D
#define RSI_BLE_HID_REPORT_MAP_UUID                 0x2A4B
#define RSI_BLE_HID_INFO_UUID                       0x2A4A
#define RSI_BLE_HID_CONTROL_POINT_UUID              0x2A4C
```

RSI_BLE_APP_HIDS refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_HIDS                            "HID_OVER_GATT"
```

GATT_ROLE refers the role of the Silicon Labs module to be selected.
If user configure **SERVER**, Silicon Labs module will act as GATT SERVER, means will add Human Interface Device service profile.
If user configure **CLIENT**, Silicon Labs module will act as GATT CLIENT, means will connect to remote GATT server and get services.

```
#define GATT_ROLE                                   SERVER
```

Valid configurations are SERVER and CLIENT.
If user configure **CLIENT** role following macros should be configured.
RSI_BLE_REMOTE_BD_ADDRESS_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_REMOTE_BD_ADDRESS_TYPE              RANDOM_ADDRESS
```

Valid configurations are RANDOM_ADDRESS and PUBLIC_ADDRESS.

RSI_BLE_REMOTE_BD_ADDRESS refers address of the remote device to connect. Replace this with valid BD address.

```
#define RSI_BLE_REMOTE_BD_ADDRESS                   "00:1A:7D:DA:71:13"
```

RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                      "Designer Keyboard"
```

Note:

User can configure either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

Following are the non configurable macros related to attribute properties.

```
#define RSI_BLE_ATT_PROP_RD                    0x02
#define RSI_BLE_ATT_PROP_WR_NO_RESP            0x04
#define RSI_BLE_ATT_PROP_WR                    0x08
#define RSI_BLE_ATT_PROP_NOTIFY                0x10
#define RSI_BLE_ATT_PROP_INDICATE              0x20
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.
RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.
RSI_BLE_REPORT_REFERENCE_UUID refers to the attribute type of the report reference descriptor to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                 0x2803
#define RSI_BLE_CLIENT_CHAR_UUID               0x2902
#define RSI_BLE_REPORT_REFERENCE_UUID          0x2908
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                     15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                        RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                    FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                      RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP             TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP             FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP         EXT_FEAT_384K_MODE
#define RSI_BAND                               RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros, #define RSI_BLE_PWR_INX 8

```
#define RSI_BLE_PWR_INX                        30
#define RSI_BLE_PWR_SAVE_OPTIONS               0
```

Note:

rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

Server role

1. After the program gets executed, Silicon Labs module will be in Advertising state.
2. Open a default Bluetooth App and do the scan.

3. In the App, Silicon Labs module will appear with the name configured in the macro **RSI_BLE_APP_HIDS (Ex: "HIDS")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".
4. Initiate connection from the App and complete the paining process.
5. After successful connection, open note pad or any text editor in phone, you can see some text printing.
6. By default, application is sending some text (i.e. "hog ") in regular interval, which will come as a notification to smart phone.
7. While connection, smart phone will do service discovery and it will find the HID service with UUID **RSI_BLE_HID_SERVICE_UUID.** After that it will read report map and enables the notification.
8. Following are the screen shots of smart phone to test HID over GATT application.

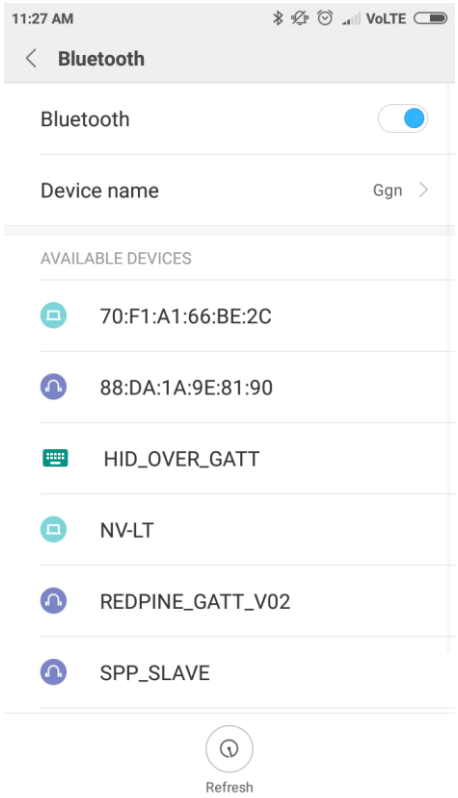


Figure 43: Scanning for HID_OVER_GATT Device

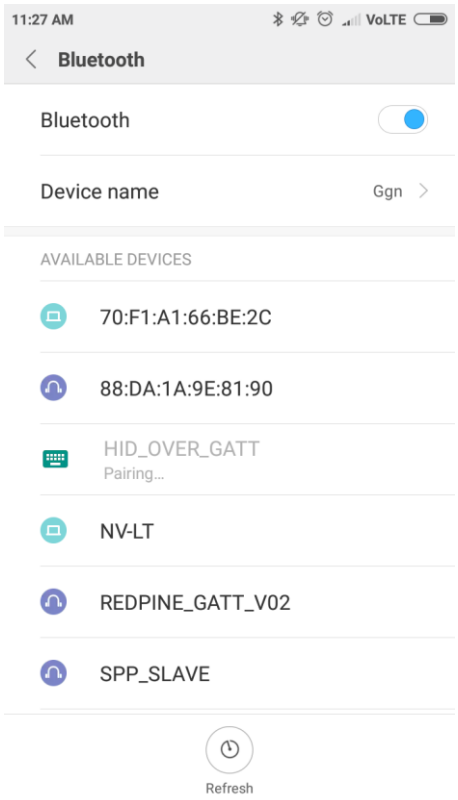


Figure 44: Connect to HID_OVER_GATT Device

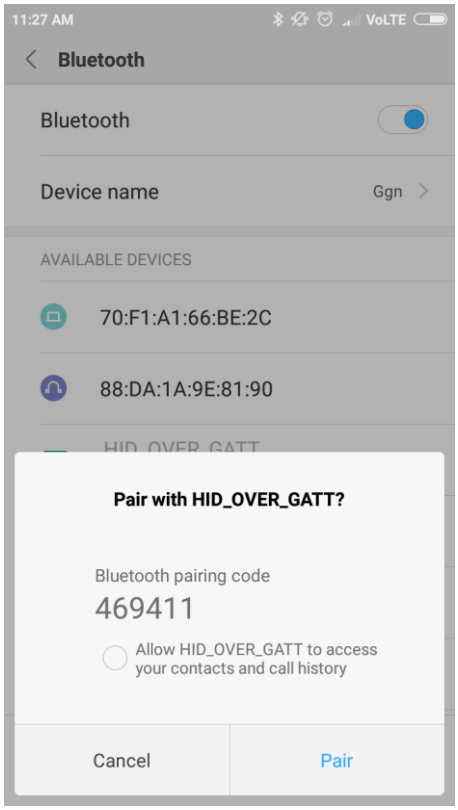


Figure 45: Pair with HID_OVER_GATT Device

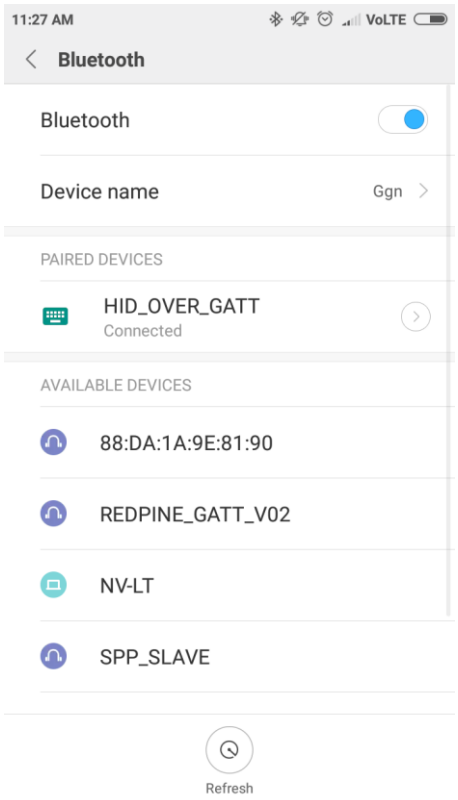


Figure 46: HID_OVER_GATT Device Connected

Figure 47: Receiving data from HID_OVER_GATT Device

Client role

1. Advertise a LE device which supports Human Interface Device Service.
2. After the program gets executed, Silicon Labs module will connect to that remote device based on given BD address.
3. After successful connection Silicon Labs module will read the services from the remote GATT server.
4. If remote device support notify property Silicon Labs module will enable notify, and ready to receive notifications from remote device.
5. Whenever GATT server changes value and notifies that Silicon Labs module will receive that value.

4.21 BLE White List

Overview

This application is used to add a particular BD-Address to the White List. The device to connect is saved on the white list located in the LL block of the controller. This enumerates the remote devices that are allowed to communicate with the local device. The White List can restrict which device are allowed to connect to other device. If is not, is not going to connect. Once the address was saved, the connection with that device is going to be an auto connection establishment procedure.This means that the Controller autonomously establishes a connection with the device address that matches the address stored in the While List.

Sequence of Events

This Application explains user how to:

- Adding BD Address to the whitelist
- Scan remote devices
- Connect to the Whitelisted remote device

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device

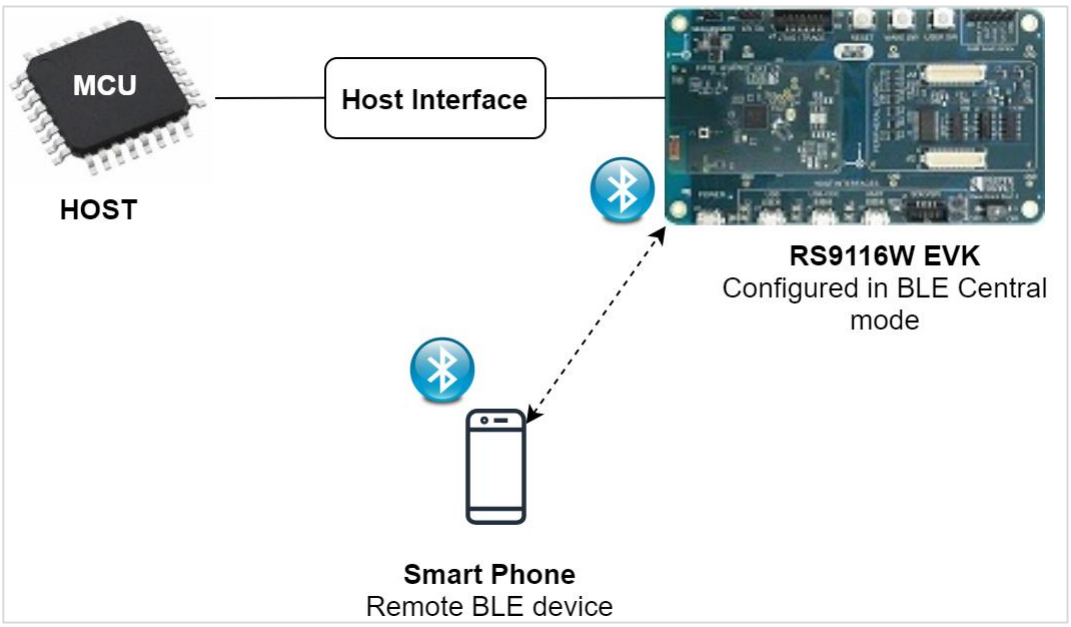


Figure 48: Setup Diagram For BLE Whitelist Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_whitelist.c*** file and update/modify following macros,
   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                          LE_PUBLIC_ADDRESS
```

   Based on address type of remote device, valid configurations are
   LE_RANDOM_ADDRESS

   LE_PUBLIC_ADDRESS

   RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR                               "00:1A:7D:DA:71:48"
```

   RSI_BLE_WHITELIST_DEV_ADDR1_TYPE,**RSI_BLE_WHITELIST_DEV_ADDR1_TYPE** refers address of the remote devices to be whitelisted

```
#define RSI_BLE_WHITELIST_DEV_ADDR1_TYPE                 LE_PUBLIC_ADDRESS
#define RSI_BLE_WHITELIST_DEV_ADDR2_TYPE                 LE_PUBLIC_ADDRESS
```

   RSI_BLE_WHITELIST_DEV_ADDR1, **RSI_BLE_WHITELIST_DEV_ADDR2** refers address of the whitelisted remote devices to connect.

```
#define RSI_BLE_WHITELIST_DEV_ADDR1                      "00:1A:7D:DA:71:48"
#define RSI_BLE_WHITELIST_DEV_ADDR2                      "00:23:A7:80:70:B9"
```

   RSI_REMOTE_DEVICE_NAME refers the name of remote device to which Silicon Labs device has to connect

```
#define RSI_REMOTE_DEVICE_NAME                           "REDPINE_DEV"
```

> **Note:**
> user can configure either RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME of the remote device.

   Following are the event numbers for advertising, connection and Disconnection events,

```
#define RSI_APP_EVENT_ADV_REPORT                         0
#define RSI_APP_EVENT_CONNECTED                          1
#define RSI_APP_EVENT_DISCONNECTED                       2
```

   Following are the non-configurable macros in the application.
   BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                               15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                  RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                              FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                                RSI_DISABLE
```

```
#define RSI_TCP_IP_FEATURE_BIT_MAP                    TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                    FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                EXT_FEAT_384K_MODE
#define RSI_BAND                                      RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                               30
#define RSI_BLE_PWR_SAVE_OPTIONS                      0
#define RSI_BLE_SCAN_FILTER_TYPE                      SCAN_FILTER_TYPE_ONLY_WHITE_LIST
```

Note:

rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

1. Configure the remote ble device in peripheral mode and put it in advertising mode.
2. After the program gets executed, it adds the configured remote device addresses to the whitelist, and Silicon Labs device tries to connect only with the whitelisted remote device specified in **RSI_BLE_DEV_ADDR or RSI_REMOTE_DEVICE_NAME** macro &
3. Observe that the connection is established between the desired device and Silicon Labs device.

Note:

Examples for ble peripherals: Blue tooth Dongle, mobile application, TA sensor tag

4.22 BLE Dual Role

Overview

This application demonstrates how to connect with multiple(6) slaves as Silicon Labs module in central mode and connect with multiple(2) masters as Silicon Labs module in peripheral mode.

Sequence of Events

This Application explains user how to:

- Connect with remote BTLE peripheral devices.
- Connect with remote BTLE central devices.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral devices
- BTLE central devices.

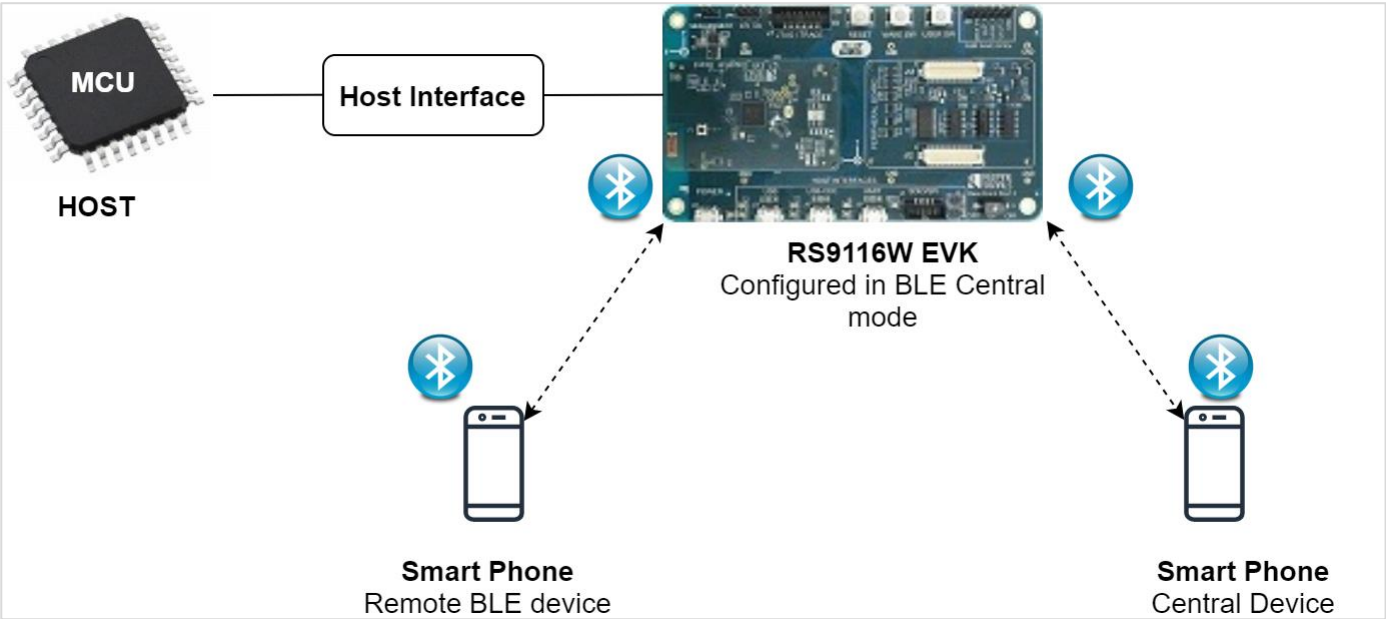


Figure 49: Setup Diagram of Dual Role Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ble_dualrole.c*** file and update/modify following macros,

   RSI_BLE_LOCAL_NAME refers the name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_LOCAL_NAME                              "WYZBEE_PERIPHERAL
```

   RSI_BLE_DEV_ADDR_TYPE refers address type of the remote device to connect.

```
#define RSI_BLE_DEV_ADDR_TYPE                       LE_PUBLIC_ADDRESS
```

   Based on address type of remote device, valid configurations are
   LE_RANDOM_ADDRESS

   LE_PUBLIC_ADDRESS

   RSI_BLE_DEV_ADDR refers address of the remote device to connect.

```
#define RSI_BLE_DEV_1_ADDR                          "00:1B:DC:07:2C:F0"
#define RSI_BLE_DEV_2_ADDR                          "00:1A:7D:DA:71:73"
#define RSI_BLE_DEV_3_ADDR                          "00:1A:7D:DA:71:44"
#define RSI_BLE_DEV_4_ADDR                          "00:1A:7D:34:54:66"
#define RSI_BLE_DEV_5_ADDR                          "00:1A:7D:DA:71:48"
#define RSI_BLE_DEV_6_ADDR                          "00:1A:7D:DA:72:13"
```

   Following are the event numbers for advertising,connection, Disconnection events and scan restart events.

```
#define RSI_APP_EVENT_ADV_REPORT                    0
#define RSI_APP_EVENT_CONNECTED                     1
#define RSI_APP_EVENT_DISCONNECTED                  2
#define RSI_BLE_SCAN_RESTART_EVENT                  3
```

Following are the non-configurable macros in the application.
BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                          15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP              EXT_FEAT_384K_MODE
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                             30
#define RSI_BLE_PWR_SAVE_OPTIONS                    0
```

To configure the Nbr of master and No of slaves to be connected

```
#define RSI_BLE_MAX_NBR_SLAVES                      8
#define RSI_BLE_MAX_NBR_MASTERS                     2
```

Note:

rsi_wlan_config.h and rsi_ble_config.h files are already set with desired configuration in respective example folders user need not change for each example.

Executing the Application

1. Configure the remote ble device in peripheral mode and put it in advertising mode.
2. After the program gets executed, Silicon Labs device tries to connect with the remote device address specified in the Macros
    a. **example: RSI_BLE_DEV_1_ADDR**
3. Silicon Labs device also in advertising mode, connect from the remote BLE Central device.
4. Observe that the connection is established between the desired device and Silicon Labs device.

Note:

Maximum we can connect with 2 Remote BLE Centrals.

Note:

Examples for ble peripherals: Blue tooth Dongle, mobile application, TA sensor tag

4.23 BLE TestModes

Setup

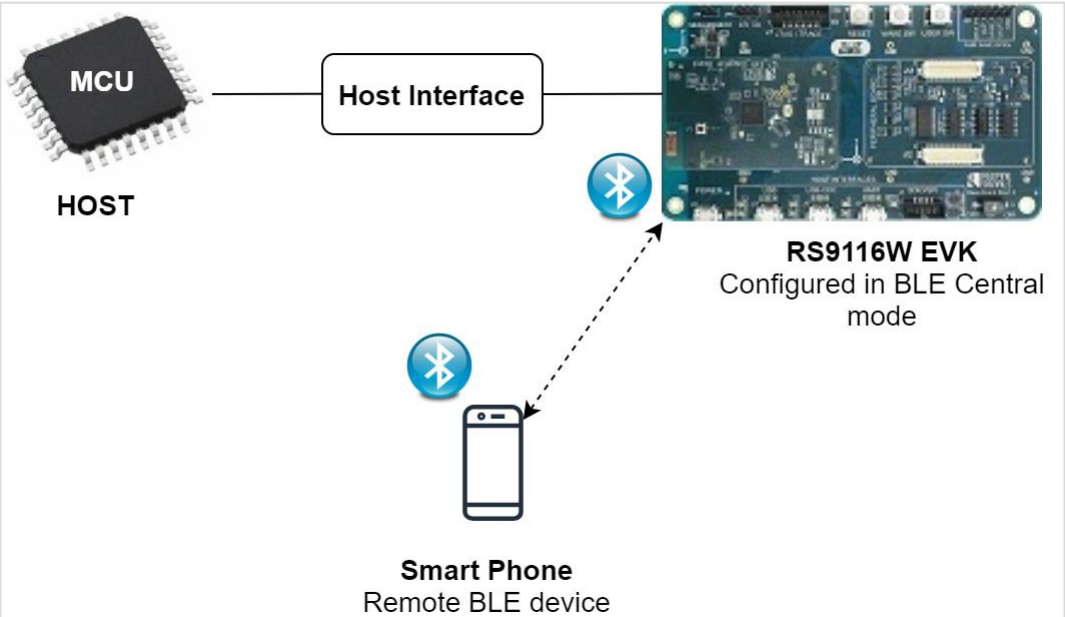


Figure 50: Setup Diagram for TestModes Example

Configuration and Steps for Execution

Configuring the Application for Tx:

1. Open ***rsi_ble_testmodes.c*** file and update/modify following macros:

```
#define RSI_CONFIG_TEST_MODE                    RSI_BLE_TESTMODE_TRANSMIT
#define RSI_BLE_TX_PAYLOAD_LEN                  0x20
#define RSI_BLE_TX_PAYLOAD_TYPE                 PRBS9_SEQ
```

RSI_SEL_ANTENNA refers to the antenna which is to be used by Silicon Labs module.
If the user using internal antenna then set,

```
#define RSI_SEL_ANTENNA                         RSI_SEL_INTERNAL_ANTENNA
```

If the user using an external antenna (U.FL connector) then set, **RSI_SEL_EXTERNAL_ANTENNA**

Following are the **non-configurable** macros in the application.

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                      15000
```

2. *Open* ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_384K_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

3. Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                          30
#define RSI_BLE_PWR_SAVE_OPTIONS                 0
```

Note:

rsi_wlan_config.h and **rsi_ble_config.h** files are already set with the desired configuration in respective example folders user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs module will Transmit the packets with desired length.
2. Run the below command using Third party dongle to verify whether the packets are transmitted or not from the Silicon Labs module.
3. hcitool -i hcix cmd 0x08 0x001D 0x10 → (hcix – Interface of third party dongle, 0x10 – Received channel)
4. Received channel of third party dongle should be same as transmit channel of Silicon Labs module.
5. Run the below command using Third party dongle to stop receiving.
6. hcitool -i hcix cmd 0x08 0x001F (hcix – Interface of third party dongle)
7. Verify the status parameters whether the packets are received or not after receiving stop command.

Configuring the Application for Rx:

1. Open ***rsi_ble_testmodes.c*** file and update/modify following macros:

```
#define RSI_CONFIG_TEST_MODE                     RSI_BLE_TESTMODE_RECEIVE
```

RSI_SEL_ANTENNA refers to the antenna which is to be used by Silicon Labs module.
If the user using internal antenna then set,

```
#define RSI_SEL_ANTENNA                          RSI_SEL_INTERNAL_ANTENNA
```

If the user using an external antenna (U.FL connector) then set, **RSI_SEL_EXTERNAL_ANTENNA**

Executing the Application

1. After the program gets executed, Silicon Labs module will Receive the packets.
2. Run the below command using Third party dongle to transmit the packets to Silicon Labs module.
3. hcitool -i hcix cmd 0x08 0x001E 0x10 0x20 0x01 → (hcix – Interface of third party dongle, 0x10 – Received channel , 0x20 – Payload length, 0x01 – Payload Type)
4. Received channel of Silicon Labs module should be same as transmit channel of Third party dongle.

4.24 Simple LE Se

Overview

This application demonstrates how to configure the Silicon Labs device in peripheral role and connects with remote device. By default, our module has enable the SMP secure connection is enabled.
In this application, Silicon Labs module connects with remote device and initiates SMP pairing process. After successful SMP pairing, SMP encryption will be enabled in both Central and Peripheral device.

Sequence of Events

This Application explains user how to:

- Configure device as peripheral/Central mode
- Connect with remote device.
- Initiate SMP paring with connected remote device.
- Initiate SMP pair response for the received SMP response event.
- Received SMP passkey events on both devices
- Responding the with SMP passkey command on both sides.
- Encryption event will be received on both sides.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- BTLE peripheral device which supports SMP pairing(This Application uses TI sensor tag for a remote device)

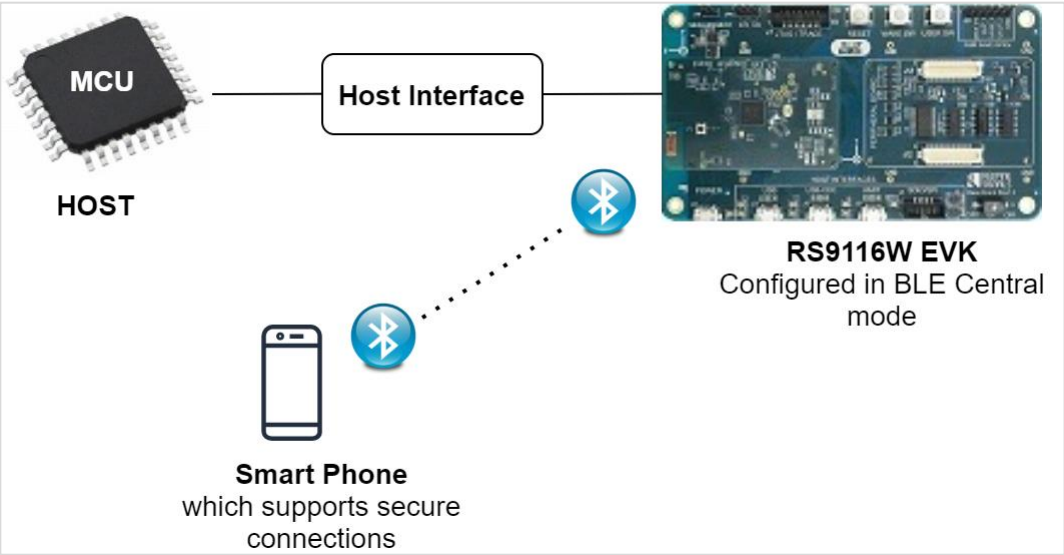


Figure 51: Setup Diagram For LE Secure Connections Example

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_ble_sc.c** file and update/modify following macros,

   RSI_BLE_DEVICE_NAME refers the name of the WiSeConnect device to appear during scanning by remote devices.

```
#define RSI_BLE_DEVICE_NAME                              "BLE_SMP_SC"
```

RSI_BLE_SMP_IO_CAPABILITY refers IO capability.

```
#define RSI_BLE_SMP_IO_CAPABILITY                        0x00
```

RSI_BLE_SMP_PASSKEY refers address type of the remote device to connect.

```
#define RSI_BLE_SMP_PASSKEY                              0
```

Following are the non-configurable macros in the application.

```
#define RSI_BLE_CONN_EVENT                               0x01
#define RSI_BLE_DISCONN_EVENT                            0x02
#define RSI_BLE_SMP_REQ_EVENT                            0x03
#define RSI_BLE_SMP_RESP_EVENT                           0x04
#define RSI_BLE_SMP_PASSKEY_EVENT                        0x05
#define RSI_BLE_SMP_FAILED_EVENT                         0x06
#define RSI_BLE_ENCRYPT_STARTED_EVENT                    0x07
#define RSI_BLE_SMP_PASSKEY_DISPLAY_EVENT                0x08
#define RSI_BLE_SC_PASSKEY_EVENT                         0X09
#define RSI_BLE_LTK_REQ_EVENT                       0x0A
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver

```
#define BT_GLOBAL_BUFF_LEN                               15000
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                  RSI_DISABLE
 #define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
 #define RSI_TCP_IP_BYPASS                               RSI_DISABLE
 #define RSI_TCP_IP_FEATURE_BIT_MAP                      TCP_IP_FEAT_DHCPV4_CLIENT
 #define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                   EXT_FEAT_384K_MODE
 #define RSI_BAND                                        RSI_BAND_2P4GHZ
```

Executing the Application

1. After the program gets executed, WSC device will be in advertising state.
2. Open a LEApp in the Smartphone and do the scan.
3. In the App, Silicon Labs module device will appear with the name configured in the macro **"BLE_SMP_SC"** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".
4. Initiate connection from the App.
5. Observe that the connection is established between the desired device and Silicon Labs device.
6. After successful connection, application will initiate SMP paring and wait for SMP response event and SMP passkey request event. After receiving SMP response and SMP SC passkey events, application sends SMP response and stores passkey in numeric value and sets SMP Sc Passkey responses event. If SMP success, Device sends SMP encrypt started event to host. If not success, Device sends SMP failure event to host.

5 BT BLE

Following example is described in this section.

S.No	Example	Description	Example Source Path
1	dual_mode	This application demonstrates how information can be exchanged seamlessly using wireless protocols (BT and BLE) running in the same device.	RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\bt_ble\dual_mode

5.1 Dual Mode

Overview

The bt ble dual mode application demonstrates how information can be exchanged seamlessly using wireless protocols (BT and BLE) running in the same device.

Description

The coex application has BLE and BT tasks act as an interface between Smartphones.
Smartphone1 interacts with BT task, Smartphone2 interacts with BLE task.
When Smartphone2 connects and sends message to Silicon Labs device, BLE task accepts.
When Smartphone1 connects and sends message to Silicon Labs device, BT task accepts. Details of the Application

Silicon LabsBT acts as a slave device with SPP profile running in it, while Smart phone acts as a Master device with SPP profile running in it.
Silicon Labs BLE acts as a Peripheral (Slave) device with GATT Server running in it, while Smart phone acts as a Central (Master) device with GATT Client running in it.
Initially, proprietary Simple chat service is created with SPP profile (Silicon Labs device) to facilitate message exchanges.

- The BT task (running in Silicon Labs device) mainly includes following steps.

1. Creates chat service
2. Configures the device in Discoverable mode and Connectable mode.

- The BLE task (running in Silicon Labs device) mainly includes following steps.

1. Creates chat service
2. Configures the device to Advertise

BLE and BT tasks forever run in the application to serve the asynchronous events.

Sequence of Events

BT Task

This Application explains user how to:

- Configure Silicon Labs device to SPP profile mode
- Configure device in discoverable and connectable mode
- Establish SPP profile level connection with remote smart phone
- Receive data sent by Smart phone
- Send data to Smart phone

LE Task

This Application explains user how to:

- Create chat service
- Configure device in advertise mode
- Connect from Smart phone
- Receive data sent by Smart phone
- Send data to Smart phone

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BT Application (Ex: Bluetooth SPP Pro)

- Smart phone/tablet with BLE Application (Ex: Light Blue APP for iPhone/nRF Connect APP for android)

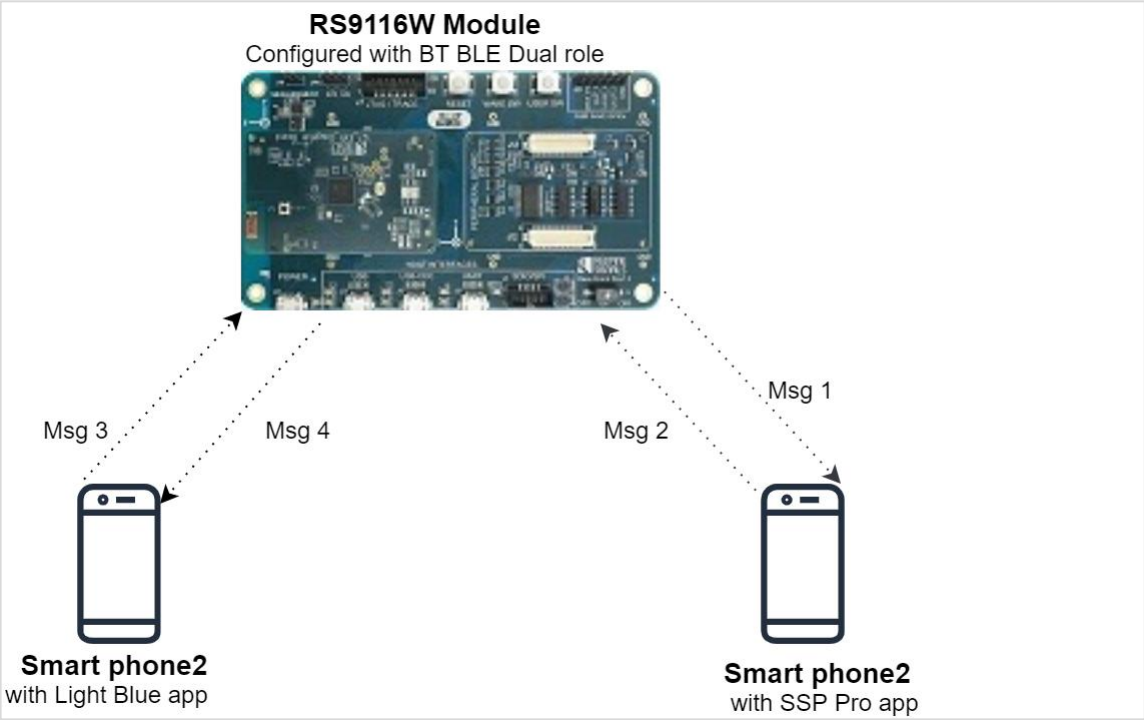


Figure 52: Setup Diagram for BT BLE Dual Mode

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_384K_MODE
```

Configuring the BT task

1. Open ***rsi_bt_app.c*** file and update/modify following macros:

- RSI_BT_LOCAL_NAME **-** Name of the BT device
- PIN_CODE - Four byte string required for pairing process.

Following are the **non-configurable** Macros in the Application file.

- RSI_APP_EVENT_CONNECTED - Event number to be set on connection establishment.
- RSI_APP_EVENT_DISCONNECTED - Event number to be set on disconnection.
- RSI_APP_EVENT_PINCODE_REQ - Event number to be set on Pincode request for pairing.
- RSI_APP_EVENT_LINKKEY_SAVE - Event number to be set on link key save.
- RSI_APP_EVENT_AUTH_COMPLT - Event number to be set on authentication complete.
- RSI_APP_EVENT_LINKKEY_REQ - Event number to be set on link key request for connection.
- RSI_APP_EVENT_SPP_CONN - Event number to be set on SPP connection.
- RSI_APP_EVENT_SPP_DISCONN - Event number to be set on SPP disconnection.

- RSI_APP_EVENT_SPP_RX - Event number to be set on SPP data received from Master.

Configuring the BLE Application

1. Open ***rsi_ble_app.c*** file and update/modify following macros,

- RSI_BLE_NEW_SERVICE_UUID - The attribute value of the newly created service. Ex: 0xAABB
- RSI_BLE_ATTRIBUTE_1_UUID - The attribute type of the first attribute under this Service. Ex: 0x1AA1
- RSI_BLE_ATTRIBUTE_2_UUID - The attribute type of the second attribute under this Service. Ex: 0x1BB1
- RSI_BLE_MAX_DATA_LEN - Maximum length of the attribute data(limited to max of 20 bytes)
- RSI_BLE_APP_DEVICE_NAME - Name of the Silicon Labs device to appear during Scanning by peer devices.

Following are the **non-configurable** macros in the application.

- RSI_BLE_ATT_PROPERTY_READ – Used to set read property to an attribute value.
- RSI_BLE_ATT_PROPERTY_WRITE - Used to set write property to an attribute value.
- RSI_BLE_ATT_PROPERTY_NOTIFY - Used to set notify property to an attribute value.
- RSI_BLE_CHAR_SERV_UUID - The attribute type of the characteristics to be added in a service. Ex: 0x2803
- RSI_BLE_CLIENT_CHAR_UUID - The attribute type of the client characteristics descriptor to be added in a service characteristic. Ex: 0x2902
- BT_GLOBAL_BUFF_LEN – Number of bytes required for the Application and the Driver.

Executing the coex Application

1. Connect Silicon Labs device to the Windows PC running KEIL or IAR IDE
2. Build and launch the application.
3. After the program gets executed, Silicon Labs BT is in Discoverable state and BLE is in Advertising state.
4. Now initiate connection from the SPP App running in the Smartphone1.
5. In the App, Silicon Labs BT would appear with the name configured in the macro RSI_BT_LOCAL_NAME.
6. After BT SPP connection is established, send a message from the App to Silicon Labs BT.
7. Open a LE App in the Smartphone and do Scan.
8. In the App, Silicon Labs BLE would appear with the name configured in the macro RSI_BLE_APP_DEVICE_NAME.
9. Initiate BLE connection from the App.
10. After BLE connection, User can write or read messages through BLE.

6 WLAN

Following is the list of examples described in this section.

S.No	Example	Description	Example Source Path	STM32 Project Path
1	Access point start example	The AP start example demonstrates how to configure the Silicon Labs device as a soft Access point and allows stations to connect to it. The example also enables TCP data transmission from the connected Wi-Fi station to Silicon Labs Access Point.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\access_point	NA
2	AP UDP Echo example	This Application demonstrates how to configure UDP socket for Echo service in AP TCP/IP bypass mode.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\ap_udp_echo	NA
3	Cloud example	1.AWS IOT SDK 2.MQTT Client Example 3.SSL Client Example	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\cloud	NA
4	Concurrent mode example	This application demonstrates how to configure the device in both Wi-Fi Station mode and Access point mode and how to transfer data on both modes. In this Application, Silicon Labs device starts as access point and connects with an access point in station mode. After successful creation of access point and successful connection with the same, the application opens TCP socket and transfers TCP data in station mode and device responds for the Ping request sent by connected station with Ping Reply in Access Point mode.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\concurrent_mode	NA
5	Connection using Asynchronous APIS	Asynchronous APIs instruct the module and return the status. The actual response by the module is indicated to the application in the registered call backs. So the driver need not indefinitely wait for the response from the module. It can schedule its tasks.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\connection_using_asynchronous_apis_app	NA
6	Customized root Webpage example	The Custom root webpage application demonstrate how to customize the root webpage. In this application, Silicon Labs device starts as an Access point. After successful creation of AP, User can open the root webpage by connecting to HTTP server running in device.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\customized_root_webpage	NA
7	DHCP User class example	This application demonstrates how DHCP USER CLASS option can be used. In this application, the device connects to the Access Point.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\dhcp_user_class	NA
8	EMB MQTT Client	This application demonstrates how to configure Silicon Labs device as MQTT client and how to establish connection with MQTT broker and how to subscribe,	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\emb_mqtt	

S.No	Example	Description	Example Source Path	STM32 Project Path
		publish and receive the MQTT messages from MQTT broker.		
9	Ethernet WiFi bridge example	The ethernet_wifi_bridge example demonstrates how to configure the Silicon Labs device as a soft Access Point and allows stations to connect to it. The example also enables the M4 Ethernet connectivity with TA WiFi and TCP data transmission from the connected ethernet station to Wi-Fi station through Silicon Labs Access Point.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\ethernet_wifi_bridge	
10	Enterprise Example	This Application demonstrates how to configure device in Enterprise client and connects with Enterprise secured AP and data traffic in Enterprise security mode.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\eap	RS9116.NB0.WC.GENR.OSI.x.x.x\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\eap (Free RTOS)
11	Firmware up gradation from server example	This application demonstrates how to upgrade new firmware to Silicon Labs device using remote TCP server. In this application, the device connects to access point and establishes TCP client connection with TCP server opened on remote peer. After successful TCP connection, application sends the firmware file request to remote TCP server and server responds with Firmware file and waits for the next firmware file request. Once firmware file receives from the TCP server, application loads the firmware file into device using firmware upgrade API and gets next firmware file from TCP server. After successful firmware upgrade, firmware upgrade API returns 0x03 response	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\firmware_upgrade	RS9116.NB0.WC.GENR.OSI.xxxx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\Firmware_upgrade (Free RTOS project)
12	FTP Client Example	This application demonstrates how to connect to FTP server opened on remote peer using FTP client and how to read file from FTP server and how to write file on to the FTP server.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\ftp_client	NA
13	HTTP/HTTPS Client Example	This application demonstrates how to create Silicon Labs device as HTTP/HTTPs client and do HTTP PUT, GET and POST operations with the HTTP/HTTPs server opened on remote peer.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\http_client	NA
14	HTTP/HTTPS Client Post Data Example	This application demonstrates how to create Silicon Labs device as HTTP/HTTPs client and do GET and POST operations with the HTTP/HTTPs server opened on remote peer. In this application, the device configures as Wi-Fi station and connects to Access	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\http_client_post_data	NA

S.No	Example	Description	Example Source Path	STM32 Project Path
		point and do HTTP/HTTPs Post and Get operation with HTTP/HTTPs server opened on remote peer.		
15	Instant Background scan Example	This application demonstrates how to enable Background scan and get results of available access points after successful connection with the Access Point in station mode.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\instant_bgscan	NA
16	MQTT Client Example	This application demonstrates how to configure Silicon Labs device as MQTT client and how to establish connection with MQTT broker and how to subscribe, publish and receive the MQTT messages from MQTT broker.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\mqtt_client	RS9116.NB0.WC.GENR.OSI.x.x.x\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI\mqtt_client
17	Multicast Example	This application demonstrates how to add Silicon Labs device to a multicast group and how to send and receive multicast data on a UDP socket. In this application, the Silicon Labs device connects to Wi-Fi access point and opens UDP socket and joins to a Multicast group ID. After successful join, application sends data to multicast group ID and receives data from Multicast group ID using opened UDP socket.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\multicast	NA
18	Over the Air Firmware up gradation	This application demonstrates how to upgrade new firmware to Silicon Labs device using remote TCP server.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\otaf	NA
19	power save deep sleep	This is a sample application demonstrating how to enable power save deep sleep profile with WiseConnectTM module. This application enables power mode 8 and then wait in a scheduler for some time. Once it will come out of delay, it will connect to configured AP and then open udp client socket. It then sends some packet to the udp server and then disconnect from AP and goes back to deep sleep.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\power_save_deep_sleep	RS9116.NB0.WC.GENR.OSI.xxxx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\Power_save (Free RTOS)
20	power_save_standby_associated	The application demonstrates how to configure device in power save profile mode 2 after successful connection with Access point in station mode and how to send UDP data from WiSeConnect device to remote peer in configured power save mode. In this application, Silicon Labs device connects to Access Point and configures to Power save profile mode2 and does UDP data transfer.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\power_save_standby_associated	NA
21	Raw data example	The raw data application demonstrates how the Silicon Labs device receives the raw data packets (packets of other IP network) and sends them to host, and	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\raw_data	NA

S.No	Example	Description	Example Source Path	STM32 Project Path
		also how it receives raw data packets from host and sends on air. In this Application, Silicon Labs device will be created as Access point, allow Wi-Fi stations to connect to it. It processes the ARP request packet (raw data) and sends ARP response (raw data). It also processes ping request (raw data) of other IP network, and sends ping response (raw data) to it.		
22	Scan Results Example	The scan results application demonstrates how to get configured number of scan results to host and how to access the scan results obtained.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\scan_results	NA
23	SNTP Client Example	This application demonstrates how Silicon Labs device gets info from SNTP server. In this application, Silicon Labs device connects to Access Point in client mode and connects to SNTP server. After successful connection with SNTP server, application gets time and date info from SNTP server.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\sntp_client	NA
24	Socket select Example	The socket select application demonstrates how to monitor multiple sockets for a certain activity to occur. If there is some data to be read on one of the sockets, select will provide that information.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\socket_select_app	NA
25	ssl_client_with_multiple_TLS_versions	This application demonstrates how to open and use a standard TCP client socket with secure connection using SSL and sends data on socket.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\ssl_client_with_multiple_TLS_versions	NA
26	Station ping Example	The application demonstrates how to configure Silicon Labs device in client mode to send ping request to target IP address.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\station_ping	NA
27	TCP IP Bypass Example	The TCP IP Bypass application demonstrates how to open and use of a standard TCP/UDP client/server socket. It bypasses the embedded TCP/IP stack for sending or receiving the data over socket.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\tcp_ip_bypass	NA
28	TCP Client Socket Example	The TCP client application demonstrates how to open and use a standard TCP client socket and sends data to TCP server socket.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\tcp_client	NA
29	TCP Server socket	The TCP server application demonstrates how to open and use a standard TCP server socket and receives data from TCP client socket.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\tcp_server	NA
30	Three_SSL_Client_sockets	This application demonstrates how to connect to three different SSL servers with three different set of SSL	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\three_ssl_client_sockets	NA

S.No	Example	Description	Example Source Path	STM32 Project Path
		certificates, using the loading certificates into FLASH.		
31	Throughput Example	Throughput is the rate of production or the rate at which something can be processed. When used in the context of communication networks, such as Ethernet or packet radio, throughput or network throughput is the rate of successful message delivery over a communication channel. This application will demonstrate the throughput measurement.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\throughput_app	NA
32	Transmit Test Example	While measuring the performance of 802.11 Wireless devices, packet error test has become today's choice for FCC certification. The Transmit test application demonstrates how Silicon Labs device starts transmitting test in Burst mode which is used for FCC certification	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\trasmit_test	NA
33	UDP Client Socket	The UDP client application demonstrates how to open and use a standard UDP client socket and sends data to UDP server socket. Once it is configured as UDP client, it can establish and maintain a network conversation by means of application program for exchanging of data.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\udp_client	NA
34	UDP Server Socket	The UDP server application demonstrates how to open and use a standard UDP server socket and receives data on socket sent by remote peer. Once it is configured as UDP server, it can establish and maintain a network conversation by means of application program for exchanging of data.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\udp_server	NA
35	User config gain table	While measuring the performance of 802.11 Wireless devices, packet error test has become today's choice for FCC certification. The user config gain table application demonstrates how Silicon Labs device starts transmitting test in Burst mode with the user configured gain values which is used for FCC certification.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\user_gain_table	NA
36	Web Socket example	This application demonstrates how to configure device in client mode to open Web socket to transmit data over Web socket to Web Server.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\websocket_client	NA
37	WEP Security Example	The WEP security application demonstrates how to connect to a WEP secured Access point and open a standard TCP client socket and sends data on server socket with the Silicon Labs device.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\wep_security	NA

S.No	Example	Description	Example Source Path	STM32 Project Path
38	Wireless Firmware upgrade Example	The Wireless Firmware upgrade application demonstrates how WiSeConnect device would be created as Access point and allow stations to connect to update the firmware through webpage.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\wireless_firmware_upgradation	NA
39	Wlan Asynchronous statistics example	This example demonstrates how to get asynchronous messages to host to indicate the module state.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\wlan_async_stats	NA
40	SSL Client	This application demonstrates how to open and use a standard TCP client socket with secure connection using SSL and sends data on socket.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\ssl_client	NA
41	WPS Access point example	Silicon Labs device supports both Push button method and WPS pin method. The WPS Access point application demonstrates how to configure the Silicon Labs device as an access point and allow client device to connect to it using WPS PUSH method. The application also enables TCP data transmission from connected Wi-Fi Station to device Access Point.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\wps_access_point	NA
42	WPS Station example	Silicon Labs Module supports both Push button method and WPS pin method to connect to an Access point. The WPS station application demonstrates how Silicon Labs module would be connected to secured (WPA2) Access point using WPS PUSH method. In this application, after successful connection with the access point using WPS PUSH button method, Silicon Labs device opens TCP socket and sends TCP data to the configured remote peer.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan\wps_station	NA

6.1 Access Point

Overview

This example demonstrates how to configure the Silicon Labs device as a soft Access point and allows stations to connect to it. This example also enables TCP data transmission from the connected Wi-Fi station to Silicon Labs Access Point.

Sequence of Events

This example explains users how to:

- Create device as 'Soft Access Point'
- Open TCP server socket on configured port number on the device
- Connect Wi-Fi Station to device Access Point
- Establish TCP connection from connected Wi-Fi Station to TCP server which is opened on device Access Point
- To send TCP data from Connected station to device Access point
- Read configured number of TCP data packets sent by connected WiFi station.

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB)
- RS9116W module
- A Mobile device as a Wi-Fi station (This example uses a windows Laptop)
- A TCP client application running on the Wi-Fi station (This example uses iperf for windows)

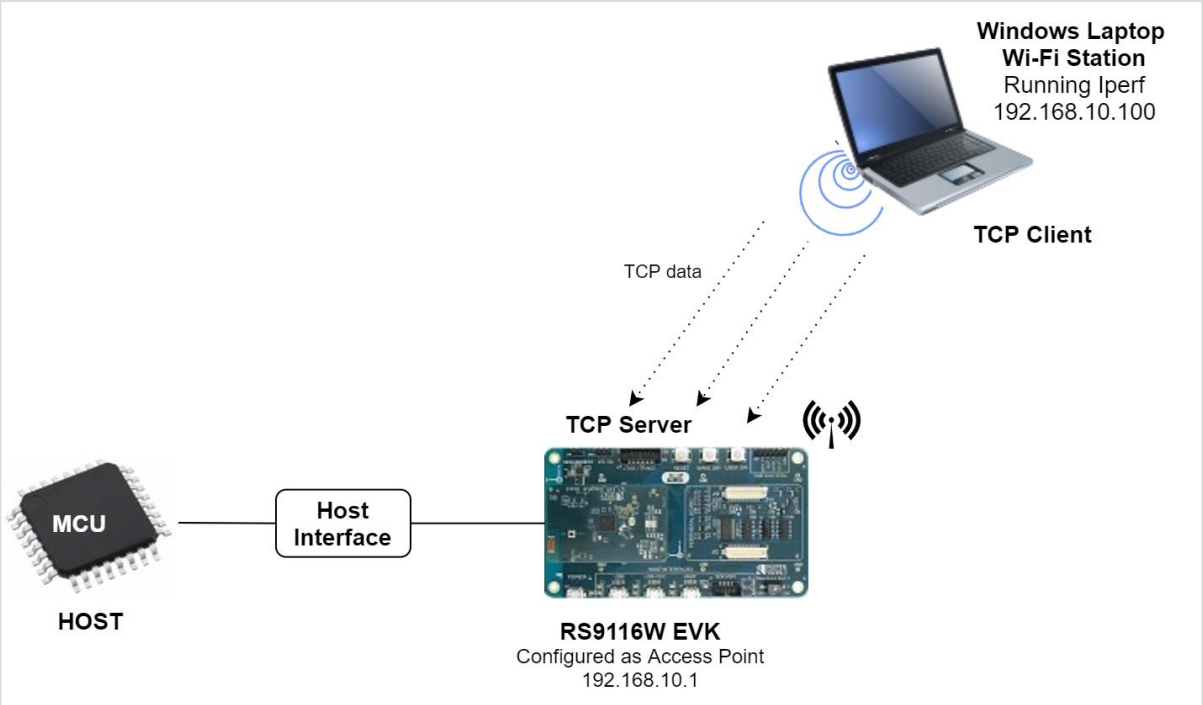


Figure 53: Setup Diagram for Access Point Start Example

Configuration and steps for Execution

Configuring the Application

- Open ***rsi_ap_start.c*** file and update / modify the following macros.

SSID refers to the name of the Access point to be created.

```
#define SSID                                "REDPINE_AP"
```

CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                          11
```

Note:
Valid values for CHANNEL_NO in 2.4GHz band are 1 to 11 5GHZ band are 36 to 48 and 149 to 165. In this example default band configured is 2.4GHz.
To use 5GHz band, set RSI_BAND macro to 5GHz band in (Example folder $) rsi_wlan_config.h file.

SECURITY_TYPE refers type of security. Access point supports Open, WPA, WPA2 securities.

Valid configurations are:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

#define SECURITY_TYPE RSI_WPA2

ENCRYPTION_TYPE refers to the type of Encryption method. Access point supports OPEN, TKIP, CCMP encryption methods.

Valid configurations are:
RSI_CCMP - For CCMP encryption
RSI_TKIP - For TKIP encryption
RSI_NONE - For open encryption

```
#define ENCRYPTION_TYPE                    RSI_CCMP
```

PSK refers to the secret key if the Access point to be configured in WPA/WPA2 security modes.

```
#define PSK                                "12345678"
```

BEACON_INTERVAL refers to the time delay between two consecutive beacons in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

```
#define BEACON_INTERVAL                    100
```

DTIM_INTERVAL refers DTIM interval of the Access Point. Allowed values are from 1 to 255.

```
#define DTIM_INTERVAL                      4
```

DEVICE_PORT port refers TCP server port number

```
#define DEVICE_PORT                        5001
```

NUMEBR_OF_PACKETS refers how many packets to receive from remote TCP client.

```
#define NUMBER_OF_PACKETS                  1000
```

```
#define GLOBAL_BUFF_LEN   15000
```

GLOBAL_BUFF_LEN refers the memory length for driver

```
#define GLOBAL_BUFF_LEN   15000
```

RECV_BUFFER_SIZE refers receive data length

```
#defineRECV_BUFFER_SIZE                    1000
```

To configure IP address
IP address to be configured to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.1" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0**.

```
#define DEVICE_IP                       0X010AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro **GATEWAY** as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

Note:
In AP mode, configure same IP address for both DEVICE_IP and GATEWAY macros

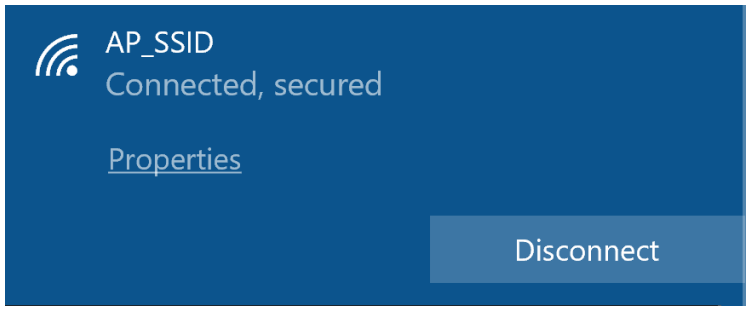
- Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_PSK
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_SERVER
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

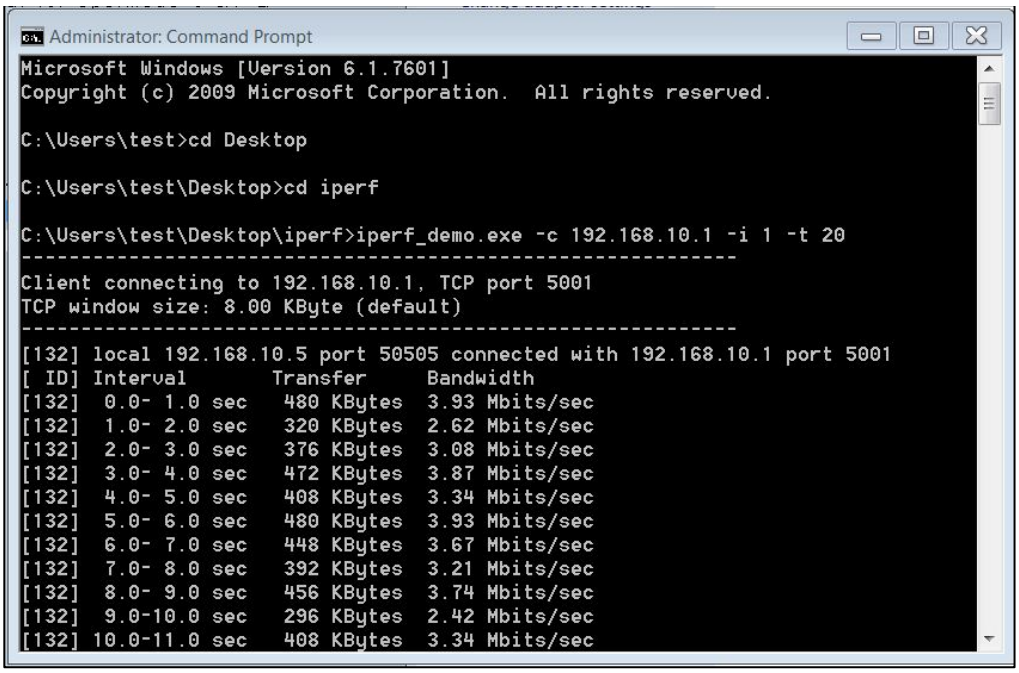
Note: rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

- After program gets executed, device will be created as an Access Point with configured **SSID** (Ex: "**AP_SSID")** and opens TCP server socket on **DEVICE_PORT** and waits for TCP connection request from TCP client. Now scan and connect to Device Access Point (Ex: "AP_SSID" is the AP name) from Laptop.



- After successful connection, open iperf client from laptop. Users can download application from https://iperf.fr/iperf-download.php#windows link.
- Connect to TCP Server running on AP using command : iperf.exe -c <DEVICE_IP> -p <DEVICE_PORT> -i 1 -t 100



- The device accepts connection request and receives data on the TCP server port and exit after receiving configured NUMBER_OF_PACKETS

6.2 AP UDP Echo

Overview

This Application demonstrates how to configure UDP socket for Echo service in AP TCP/IP bypass mode.

Sequence of Events

This Application explains user how to:

- Create RS9116W device as Soft Access point in TCP/IP bypass mode
- Assign static IP to device soft Access point
- Open UDP socket for Echo service
- Connect Wi-Fi Station to device Access Point
- Send UDP datagram from Connected station to device Access Point
- Send UDP echo by transmitting same received data from Silicon Labs device to connected station

Example Setup

Host processor should be connected to the WiSeConnect device using SPI, UART or USB host interface. The host processor firmware needs to be initialized host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- A Mobile device as a Wi-Fi station (This example uses a windows Laptop)
- A UDP application running on the Wi-Fi station (This example uses Socket Test application for windows)

Note:

Download UDP Socket Application from link**:**
http://sourceforge.net/projects/sockettest/files/latest/download

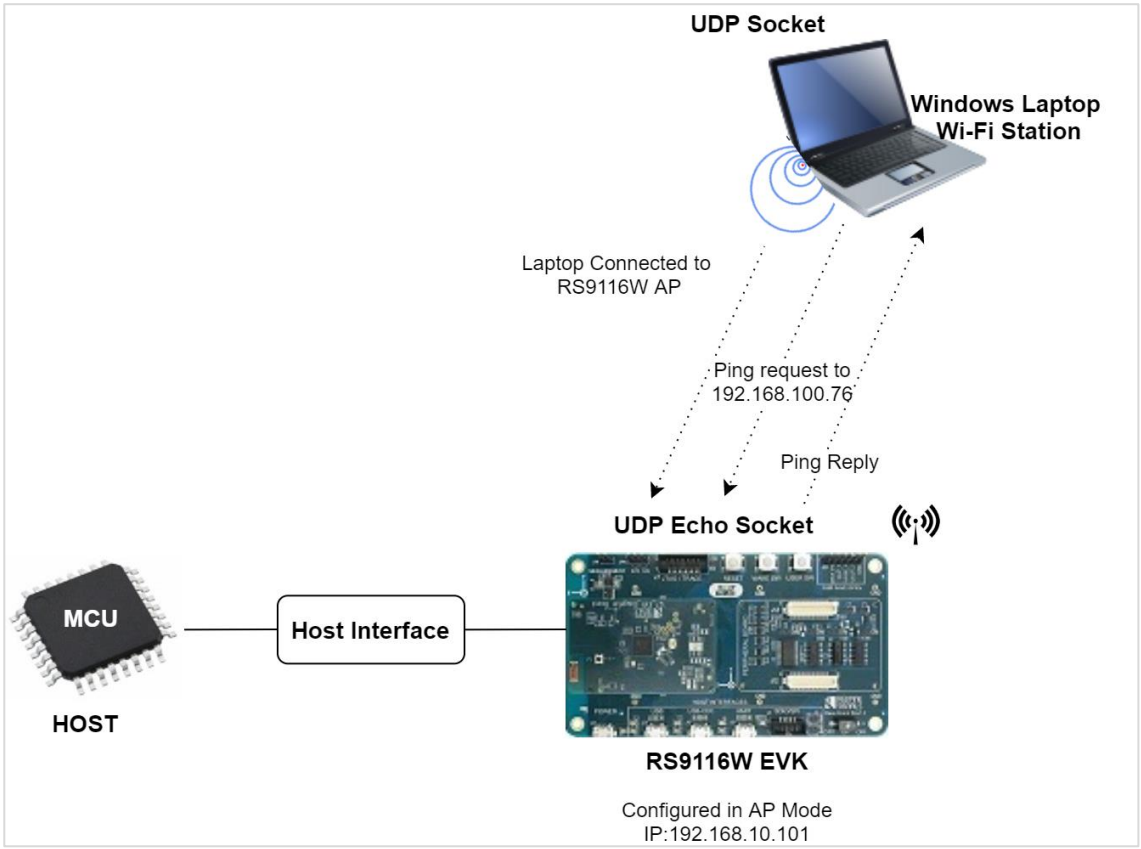


Figure 54: Setup Diagram for AP UDP Echo Example

Configuration and steps for Execution

Configuring the Application

Open ***rsi_ap_udp_echo_tcpipbypass.c*** file and update/modify following macros

SSID refers to the name of the Access point to be created.

```
#define SSID                                    "REDPINE_AP"
```

CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                              11
```

Note:

Valid values for CHANNEL_NO are 1 to 11 in 2.4GHz band and 36 to 48 & 149 to 165 in 5GHz band. In this example default configured band is 2.4GHz. If user wants to use 5GHz band then user has to set RSI_BAND macro to 5GHz band in ***rsi_wlan_config.***h file.

SECURITY_TYPE Refers to the type of security .Access Point supports Open, WPA and WPA2 securities.
Valid configurations are:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#defineSECURITY_TYPE                            RSI_WPA2
```

ENCRYPTION_TYPE refers to the type of encryption method. Access point supports OPEN, TKIP and CCMP methods.

Valid configurations are:
RSI_CCMP - For CCMP encryption
RSI_TKIP - For TKIP encryption
RSI_NONE - For open encryption

```
#define ENCRYPTION_TYPE                                 RSI_CCMP
```

PSK refers to the secret key if the Access point to be configured in WPA/WPA2 security modes.

```
#define PSK                                             "1234567890"
```

BEACON_INTERVAL refers to the time delay between two consecutive beacons in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

```
#define BEACON_INTERVAL                                 100
```

DTIM_INTERVAL refers DTIM interval of the Access Point. Allowed values are from 1 to 255.

```
#define DTIM_INTERVAL                                   4
```

DEVICE_PORT port refers internal UDP server port number

```
#define DEVICE_PORT                                     5001
```

REMOTE_PORT port refers remote UDP server port number

```
#define REMOTE_PORT                                     5001
```

GLOBAL_BUFF_LEN refers application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                                 15000
```

To configure IP address
IP address to be configured to device should be in long format and in little endian byte order.
Example: To configure "192.168.10.1" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0**.

```
#define DEVICE_IP                                       0X010AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order.
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order.
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                         0x00FFFFFF
```

Note: In AP mode, configure the same IP address for both DEVICE_IP and GATEWAY macros.

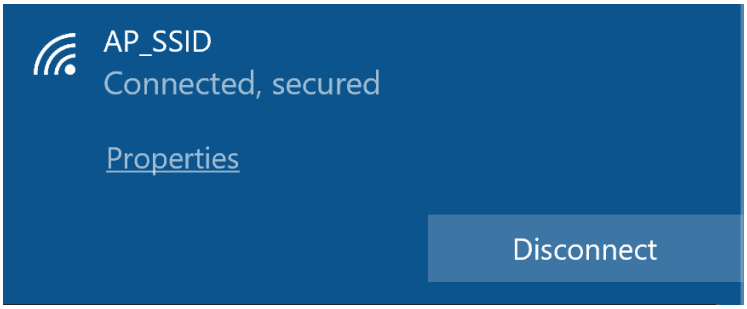
Open ***rsi_wlan_config.h*** file and update/modify following macros,

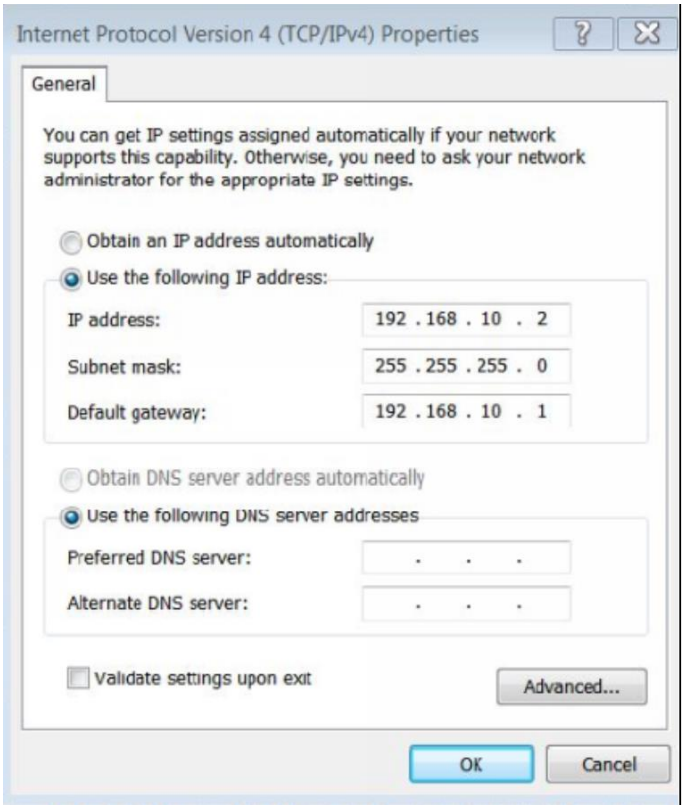
```
#define CONCURRENT_MODE                    RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                FEAT_SECURITY_PSK
#define RSI_TCP_IP_BYPASS                  RSI_ENABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP         TCP_IP_FEAT_BYPASS
#define RSI_CUSTOM_FEATURE_BIT_MAP         FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                           RSI_BAND_2P4GHZ
```

Note: rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

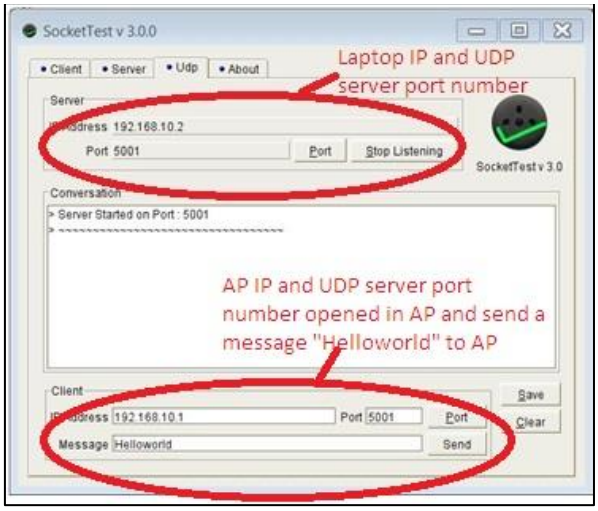
Executing the Application

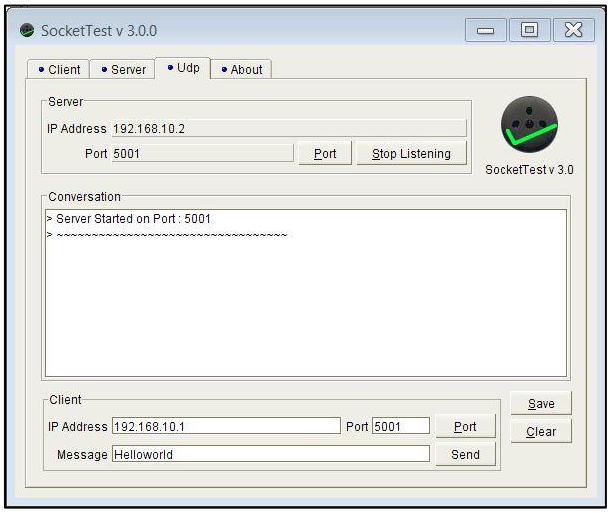
1. After the program gets executed, Silicon Labs device will be configured as an Access Point.
2. Connect a Wi-Fi station (Windows Laptop) to device AP (Ex: "Silabs_AP" is the AP name) and assign a static IP in the same Network of AP.



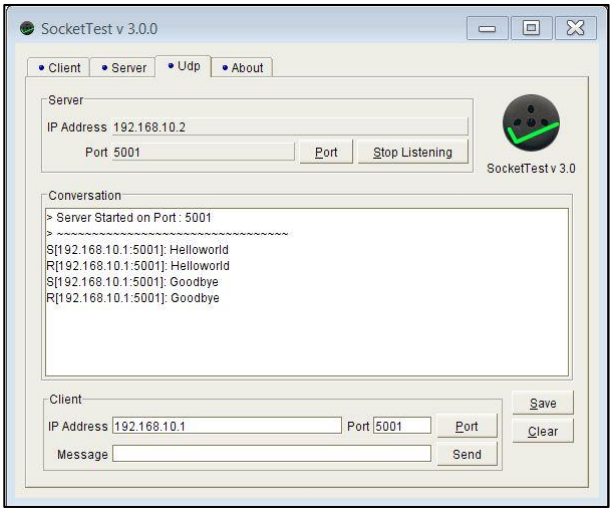


3. At remote side device (Wi-Fi Station), open SocketTest application to open UDP server socket and client socket. As per the below image, open UDP server socket on port number **REMOTE_PORT** to receive data sent by AP and open UDP client socket with port number **DEVICE_PORT** to send UDP data to AP.





4. Send "Helloworld" and "Goodbye" messages from UDP client to UDP server opened in AP and same messages will send back by AP to the UDP server opened on Wi-Fi Station. The below image depicts the messages sent by Wi-Fi Station and AP.



6.3 Cloud

6.3.1 AWS IoT SDK

6.3.1.1 Device Shadow

Protocol Overview

MQTT is a publish-subscribe based "light weight" messaging protocol for using on top of the TCP/IP protocol. The MQTT connection itself is always between one client and the broker, no client is connected to another client directly.

MQTT client
A MQTT client is any device from a micro controller to a full-fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. MQTT Clients can share the information on a particular topic using MQTT protocol. MQTT clients connect to the MQTT broker using TCP connection and can subscribe and publish on any desired topic. The other clients which are subscribed for that topic will receive the published messages.

MQTT Broker
The publish-subscribe messaging pattern requires a message broker. The broker is primarily responsible for receiving all messages, filtering them, deciding like who is interested in it and then sending the message to all subscribed clients.
It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients.

Overview

User has to create an IoT Thing in AWS cloud before running this application and this procedure is described at section "Appendix A: Cloud User Manual".

In this application, RS9116W device configured as Wi-Fi station and connects to an Access Point. After successful WiFi connection, application connects to MQTT broker and subscribes to a topic.

Device thing shadow is used to store and retrieve current state information for a device. The Device Shadow service maintains a shadow for each device you connect to AWS IoT. To get and set the state of a device over MQTT can

be observed in shadow.

This example will update the device thing shadow document. After successful execution updates can be observed in thing shadow present in AWS cloud.

Setup Requirements

- Windows PC1 with Keil or IAR IDE
- RS9116W Module
- Access point with an Internet connection
- AWS account

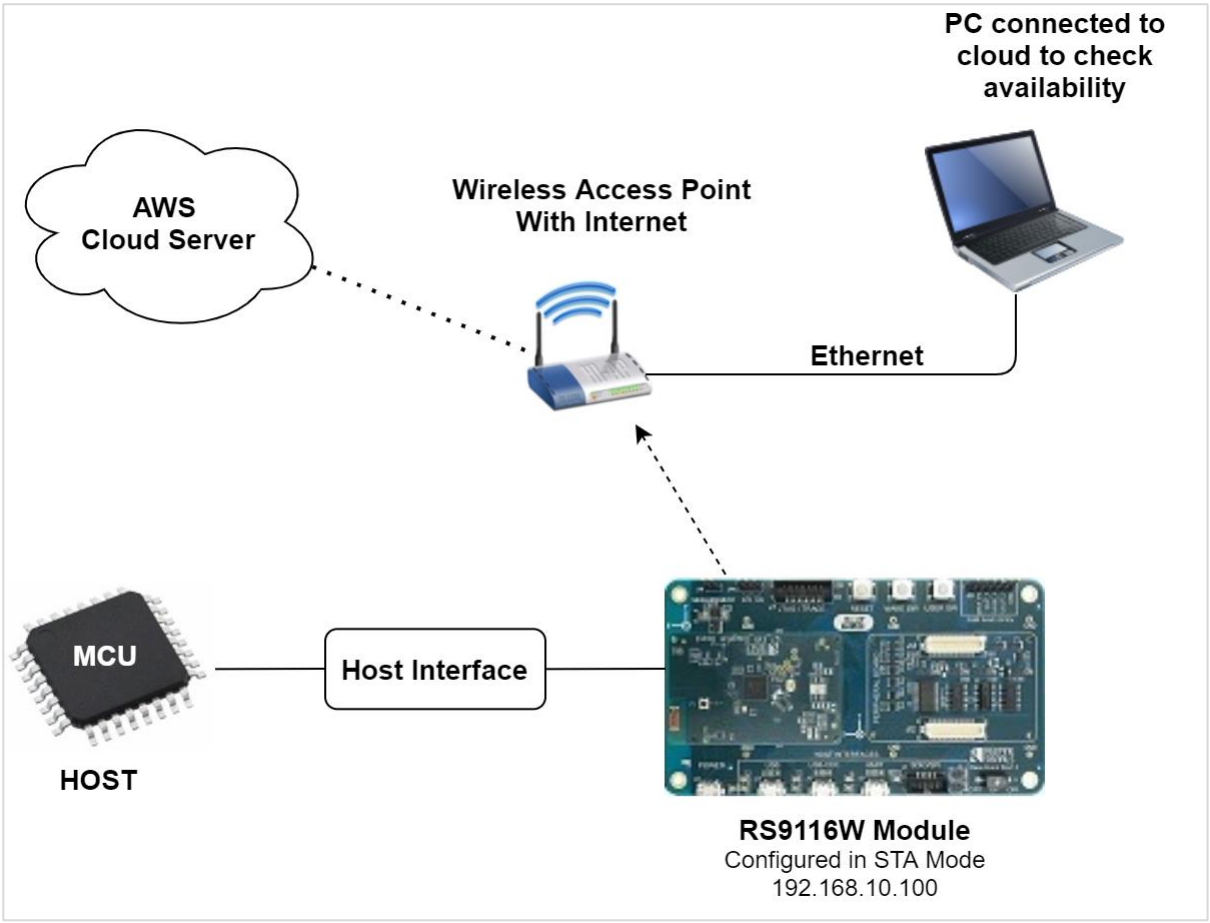


Figure 55: Setup Diagram for Device Shadow Example

Configuring the Application

Open Project from host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\AWS_IoT

Open **rsi_shadow_sample.c** file and update/modify following macros:

SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     "<psk>"
```

To Load certificate

rsi_wlan_set_certificate API expects the certificate in the form of linear array. Convert the pem certificate into linear array form using python script provided in the release package "***host/sapis/examples/utilities/certificates/certificate_script.py***".

Example:

If the certificate is wifi-user.pem, enter the command in the following way:
python certificate_script.py ca-cert.pem
The script will generate wifiuser.pem in which one linear array named **cacert** contains the certificate.

- After the conversion of certificate, update *rsi_mqtt.c* source file by including the certificate file and also by providing the required parameters to **rsi_wlan_set_certificate** API.

Once the certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.So, define LOAD_CERTIFICATE as 0, if certificate is already present in the device.

CLIENT_PORT port refers device MQTT client port number

```
#define CLIENT_PORT                              5001
```

SERVER_PORT port refers remote MQTT broker/server port number

```
#define SERVER_PORT                              8883
```

SERVER_IP_ADDRESS refers remote peer IP address (Windows PC2) to connect with MQTT broker/server socket. IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the
macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS                        0x6400A8C0
```

Memory to initialize MQTT client Info structure

```
#define MQTT_CLIENT_INIT_BUFF_LEN                3500
```

Global buffer or memory which is used for MQTT client initialization. This buffer is used for the MQTT client information storage.
uint8_t mqqt_client_buffer[MQTT_CLIENT_INIT_BUFF_LEN];
QOS indicates the level of assurance for delivery of an Application Message.
QoS levels are:
0 - At most once delivery
1 - At least once delivery
2 - Exactly once delivery

RSI_MQTT_TOPIC refers to which topic WiSeConnect MQTT client is supposed to subscribe.

```
#define RSI_MQTT_TOPIC                           "" (Update with a Topic name of IoT Thing in AWS cloud)
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                                1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through **STATIC** then set **DHCP_MODE** macro to "0" and configure **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                              0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                0x00FFFFFF
```

The following parameters are configured if OS is used.
WLAN task priority is given and this should be of low priority

```
#define RSI_WLAN_TASK_PRIORITY                 1
```

Driver task priority is given and this should be of highest priority

```
#define RSI_DRIVER_TASK_PRIORITY               1
```

WLAN Task stack size is configured by this macro

```
#define RSI_WLAN_TASK_STACK_SIZE               500
```

Driver Task stack size is configured by this macro

```
#define RSI_DRIVER_TASK_STACK_SIZE             500
```

Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                        RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                    FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                      RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP             (TCP_IP_FEAT_DHCPV4_CLIENT|TCP_IP_FEAT_SSL
|TCP_IP_FEAT_DNS_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP             FEAT_CUSTOM_FEAT_EXTENTION_VALID#define
RSI_EXT_CUSTOM_FEATURE_BIT_MAP                 EXT_FEAT_256k_MODE

#define RSI_BAND                               RSI_BAND_2P4GHZ
```

Note: rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

->Configure below parameter in aws_iot_config.h file

```
  #define AWS_IOT_MQTT_HOST                          " "  ///< Customer specific MQTT HOST. The same
will be used for Thing Shadow

  #define AWS_IOT_MQTT_PORT                              ///< default port for MQTT/S

  #define AWS_IOT_MQTT_CLIENT_ID                    " "  ///< MQTT client ID should be unique for every
device

  #define AWS_IOT_MY_THING_NAME                     " "  ///< Thing Name of the Shadow this device is
associated with

  #define AWS_IOT_ROOT_CA_FILENAME                " "  ///< Root CA file name

  #define AWS_IOT_CERTIFICATE_FILENAME     " "  ///< device signed certificate file name

  #define AWS_IOT_PRIVATE_KEY_FILENAME    " "  ///< Device private key filename
```

Executing the Application

1. Configure the Access point with Internet connection in OPEN/WPA-PSK/WPA2-PSK mode to connect RS9116W device in STA mode.
2. Log into AWS account and open IoT Thing created.

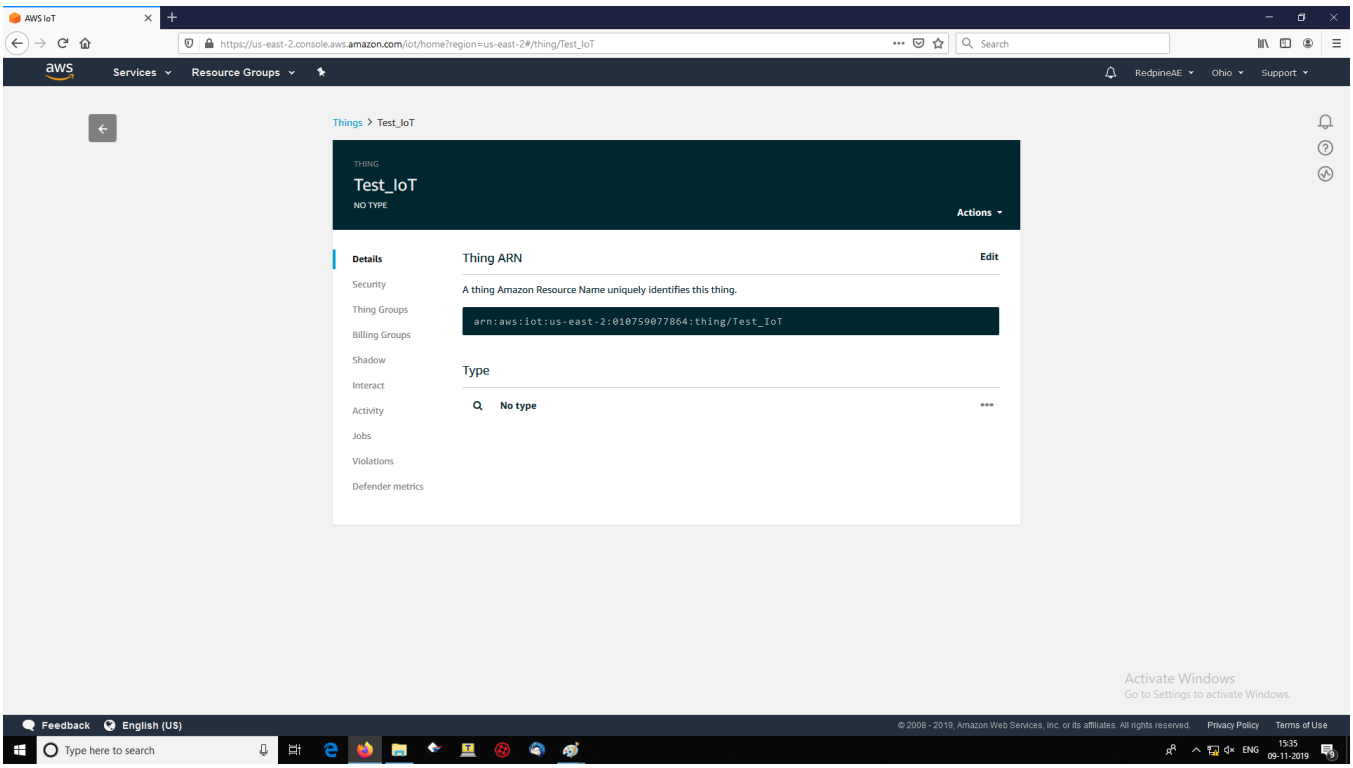


Figure 56: Test_IoT Thing

3. Update Domain name and topics name to connect to AWS mqtt cloud. Application next Subscribes to a topic to get shadow update.

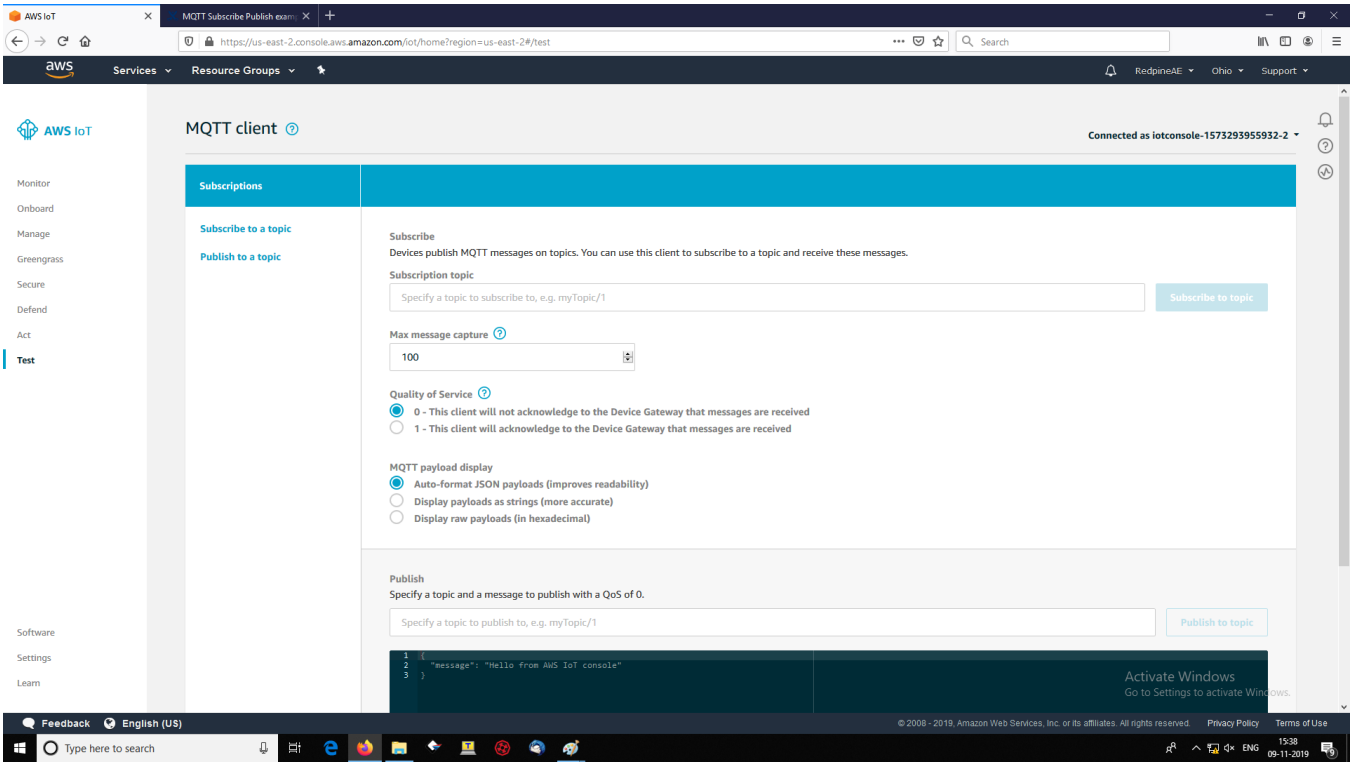


Figure 57: Subscribe to a Topic

4. Update the device thing shadow document. After successful execution, updates can be observed in thing shadow present in AWS cloud. updated messages can be observed in cloud shadow under the Thing.

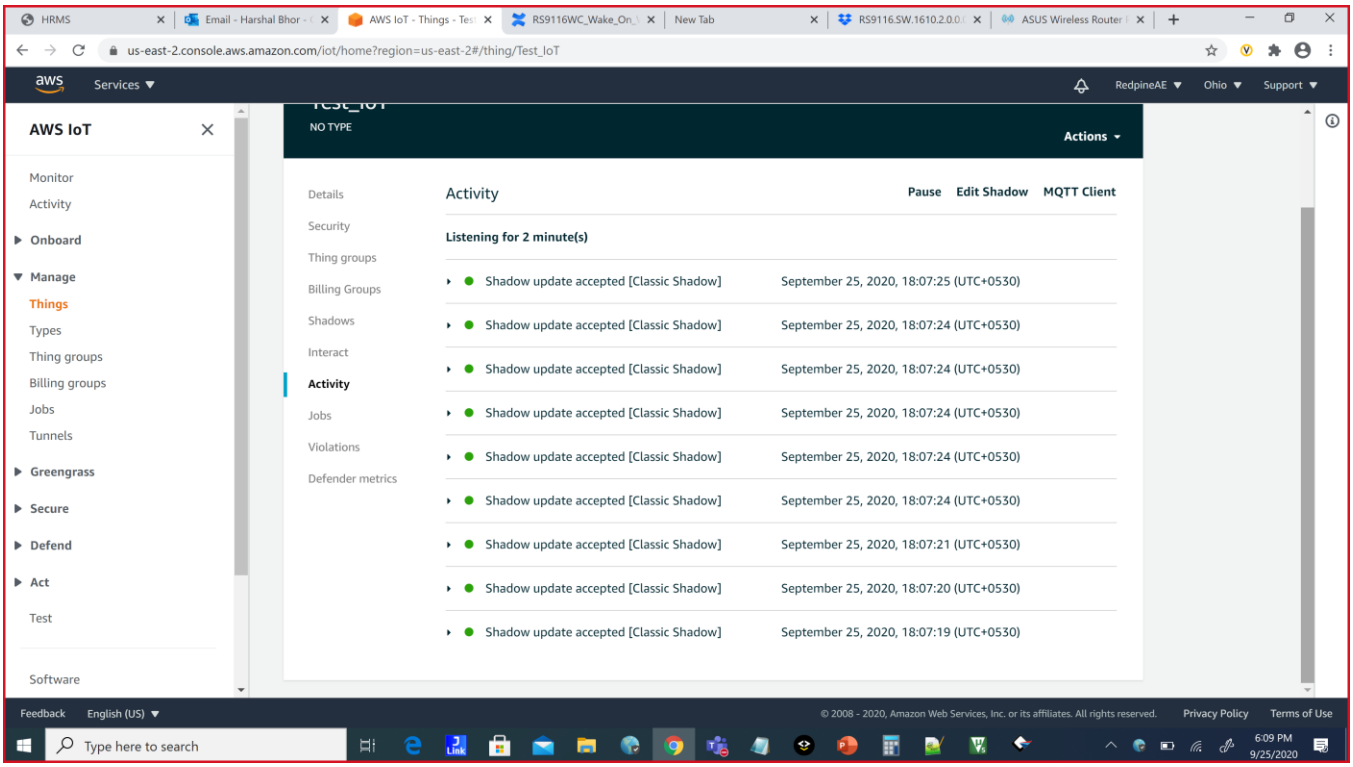


Figure 58: Shadow Update Activity

5. Following debug prints will be displayed for successful execution.

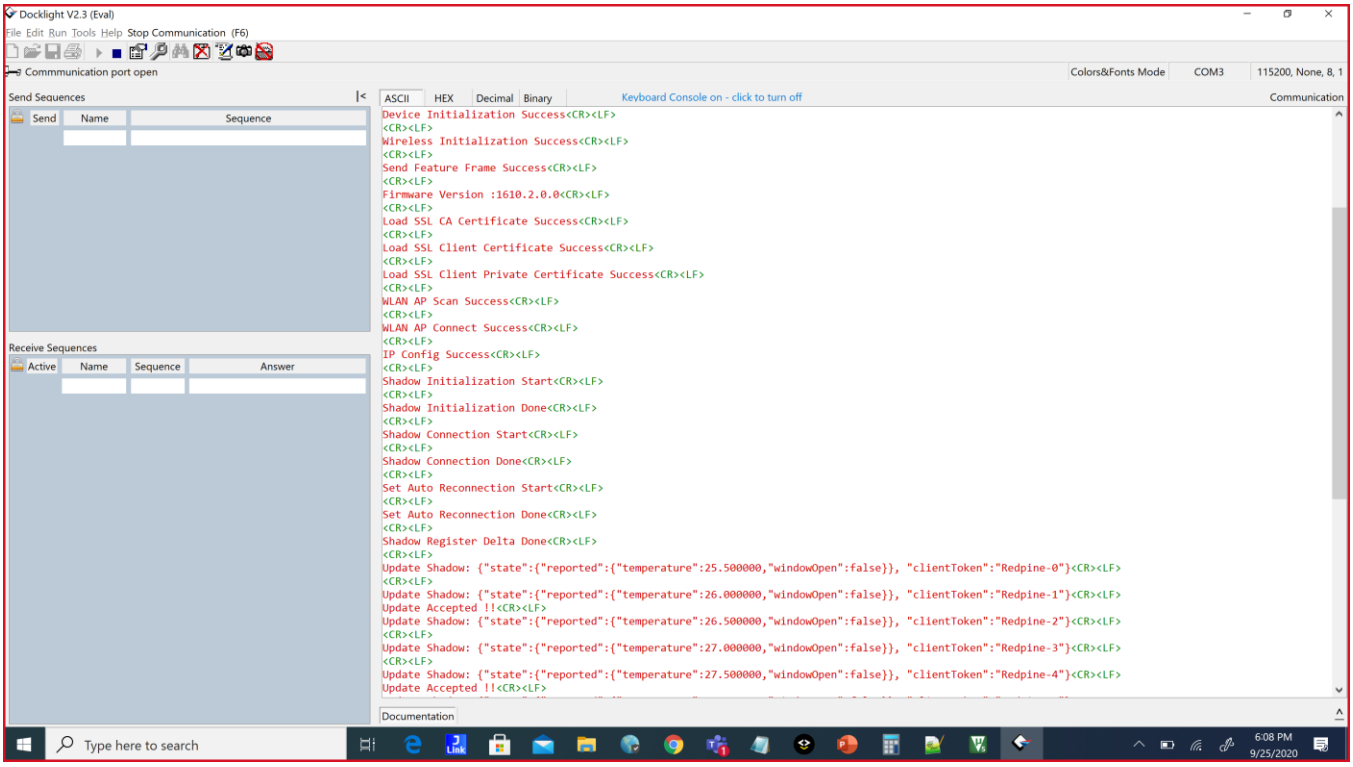


Figure 59: Debug Prints

->Follow this below link for API description.
http://aws-iot-device-sdk-embedded-c-docs.s3-website-us-east-1.amazonaws.com/index.html

6.3.1.2 MQTT (Subscribe Publish)

Protocol Overview

MQTT is a publish-subscribe based "light weight" messaging protocol for using on top of the TCP/IP protocol. The MQTT connection itself is always between one client and the broker, no client is connected to another client directly.

MQTT client

A MQTT client is any device from a micro controller to a full-fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. MQTT Clients can share the information on a topic using MQTT protocol. MQTT clients connect to the MQTT broker using TCP connection and can subscribe and publish on any desired topic. The other clients which are subscribed for that topic will receive the published messages.

MQTT Broker

The publish-subscribe messaging pattern requires a message broker. The broker is primarily responsible for receiving all messages, filtering them, deciding like who is interested in it and then sending the message to all subscribed clients.
It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients.

Overview

User has to create an IoT Thing in AWS cloud before running this application and this procedure is described at section "Appendix A: Cloud User Manual".

This application demonstrates how to configure RS9116W module as an MQTT client and how to establish connection with MQTT broker present in AWS cloud and how to subscribe, publish and receive the MQTT messages from MQTT broker.

In this application,

- RS9116W EVK is configured as Wi-Fi station and connects to an Access Point which has an internet access.
- After successful Wi-Fi connection, application connects to MQTT broker and subscribes to a topic.
- Publishes a message on subscribed topic and application waits to receive the data published on subscribed topic from the cloud.

Sequence of Events

This Application explains user how to:

- Connect to an Access Point.
- Establish MQTT client connection with MQTT broker in AWS cloud.
- Subscribe to a topic.
- Publish message on subscribed topic.
- Receive data published by other clients on a subscribed topic.

Setup Requirements

- Windows PC1 with Keil or IAR IDE
- RS9116W EVK
- Access Point with an Internet connection
- AWS account

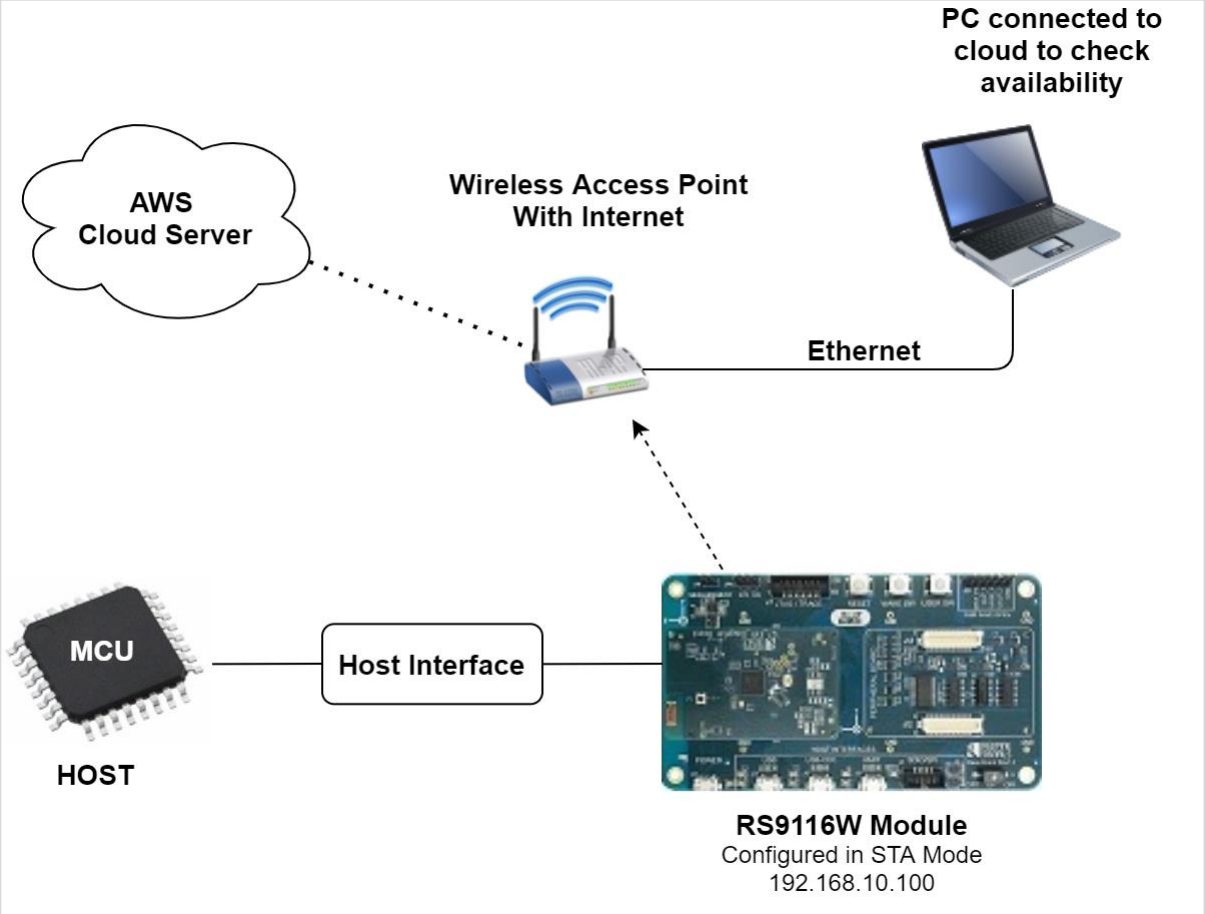


Figure 60: Setup Diagram for Subscribe Publish Example

Import and Compile Project

Refer to '**AWS SDK Porting**' section in RS9116W_SAPI_Porting_Guide_vXX.pdf for importing and compiling the example project.

Configuration and Steps for Execution

Configuring the Application

Open **rsi_subscribe_publish_sample.c** file and update/modify following macros:

SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     "<psk>"
```

To Load certificate

rsi_wlan_set_certificate API expects the certificate in the form of linear array. Convert the pem certificate into linear array form using python script provided in the release package "***host/sapis/examples/utilities/certificates/certificate_script.py***".

Example:

If the certificate is wifi-user.pem, enter the command in the following way:
python certificate_script.py ca-cert.pem
The script will generate wifiuser.pem in which one linear array named **ca-cert** contains the certificate.

- After the conversion of certificate, update *rsi_mqtt.c* source file by including the certificate file and also by providing the required parameters to **rsi_wlan_set_certificate** API.

Once the certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change. So, define LOAD_CERTIFICATE as 0, if certificate is already present in the device.

RSI_MQTT_TOPIC refers to which topic WiSeConnect MQTT client is supposed to subscribe.

```
#define RSI_MQTT_TOPIC                          "" (Update with a Topic name of IoT Thing in AWS cloud)
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                               1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through **STATIC** then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

The following parameters are configured if OS is used.
WLAN task priority is given and this should be of low priority

```
#define RSI_WLAN_TASK_PRIORITY                  1
```

Driver task priority is given and this should be of highest priority

```
#define RSI_DRIVER_TASK_PRIORITY                1
```

WLAN Task stack size is configured by this macro

```
#define RSI_WLAN_TASK_STACK_SIZE                500
```

Driver Task stack size is configured by this macro

```
#define RSI_DRIVER_TASK_STACK_SIZE              500
```

Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT|TCP_IP_FEAT_SSL
|TCP_IP_FEAT_DNS_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID#define
RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_256k_MODE

#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Note: rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

->Configure below parameter in aws_iot_config.h file

```
 #define AWS_IOT_MQTT_HOST            " " //< Customer specific MQTT HOST. The same will be used for
Thing Shadow
```

```
 #define AWS_IOT_MQTT_PORT              //< default port for MQTT/S
```

```
 #define AWS_IOT_MQTT_CLIENT_ID       " " //< MQTT client ID should be unique for every device
```

```
 #define AWS_IOT_MY_THING_NAME        " " //< Thing Name of the Shadow this device is associated with
```

```
 #define AWS_IOT_ROOT_CA_FILENAME     " " //< Root CA file name
```

```
 #define AWS_IOT_CERTIFICATE_FILENAME  " " //< device signed certificate file name
```

```
 #define AWS_IOT_PRIVATE_KEY_FILENAME  " " //< Device private key filename
```

Executing the Application

- Configure the Access point with Internet connection in OPEN/WPA-PSK/WPA2-PSK mode to connect RS9116W EVK STA mode.
- Login into AWS account and open IoT Thing created.
- Update Domain name and topics name to connect to AWS mqtt cloud and publish message to IoT Thing.
- Connect any serial console for prints.

Steps for Execution

1. Compile and run the example demo, following debug prints will be displayed.

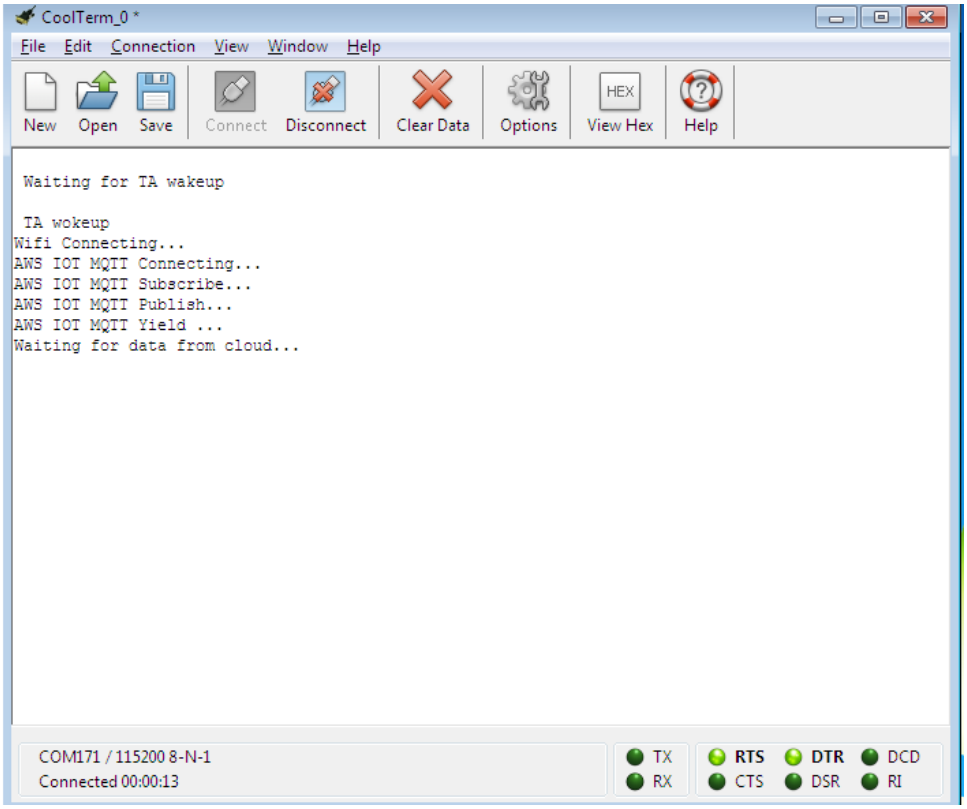


Figure 61: Debug Prints

2. Application will wait for "**toggleled**" message from cloud. Subscribe to a topic (configured in 'RSI_MQTT_TOPIC' in rsi_subscribe_publish_sample.c)

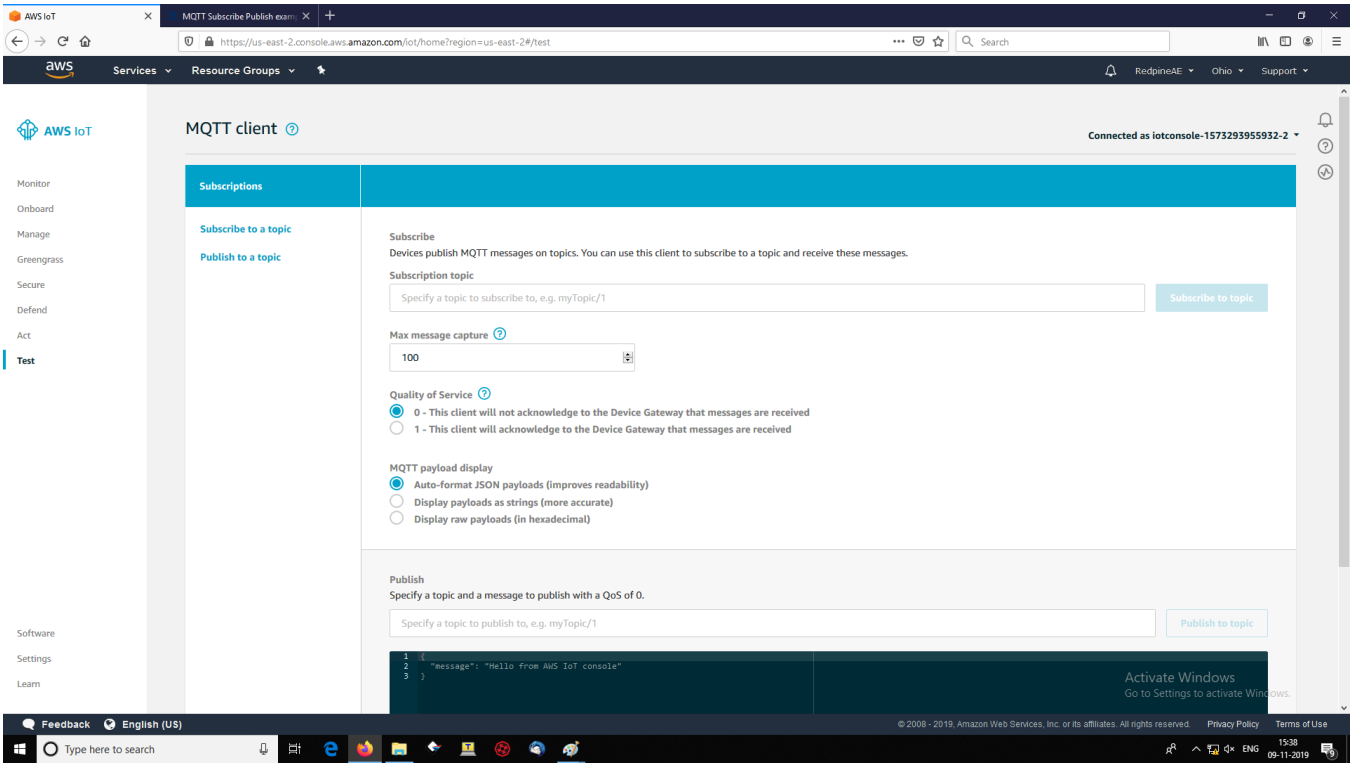


Figure 62: Subscribe to a Topic

3. Publish message on that topic from cloud like below. Message should be '**toggleled**'.

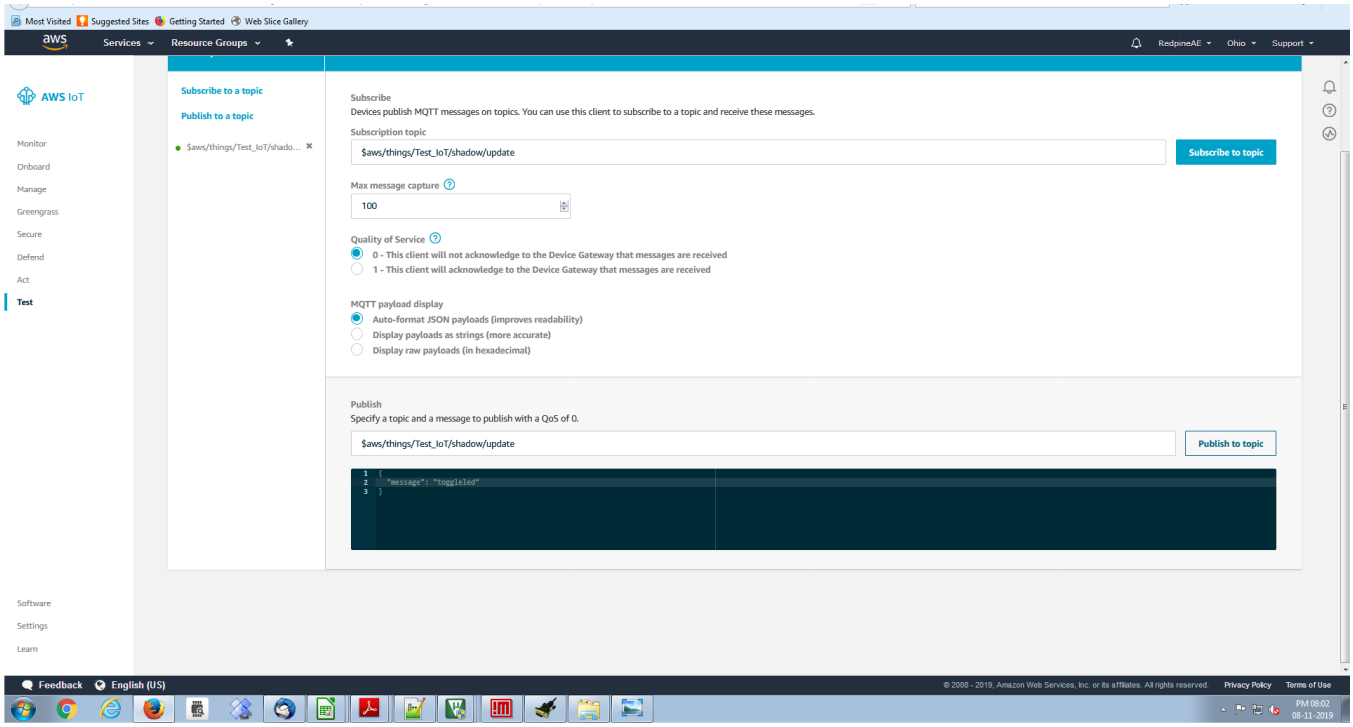


Figure 63: Publish Message

4. After this, module receives published message from the cloud, LED will be toggled.

Steps to Check Published Data on Cloud

1. Open AWS cloud page.

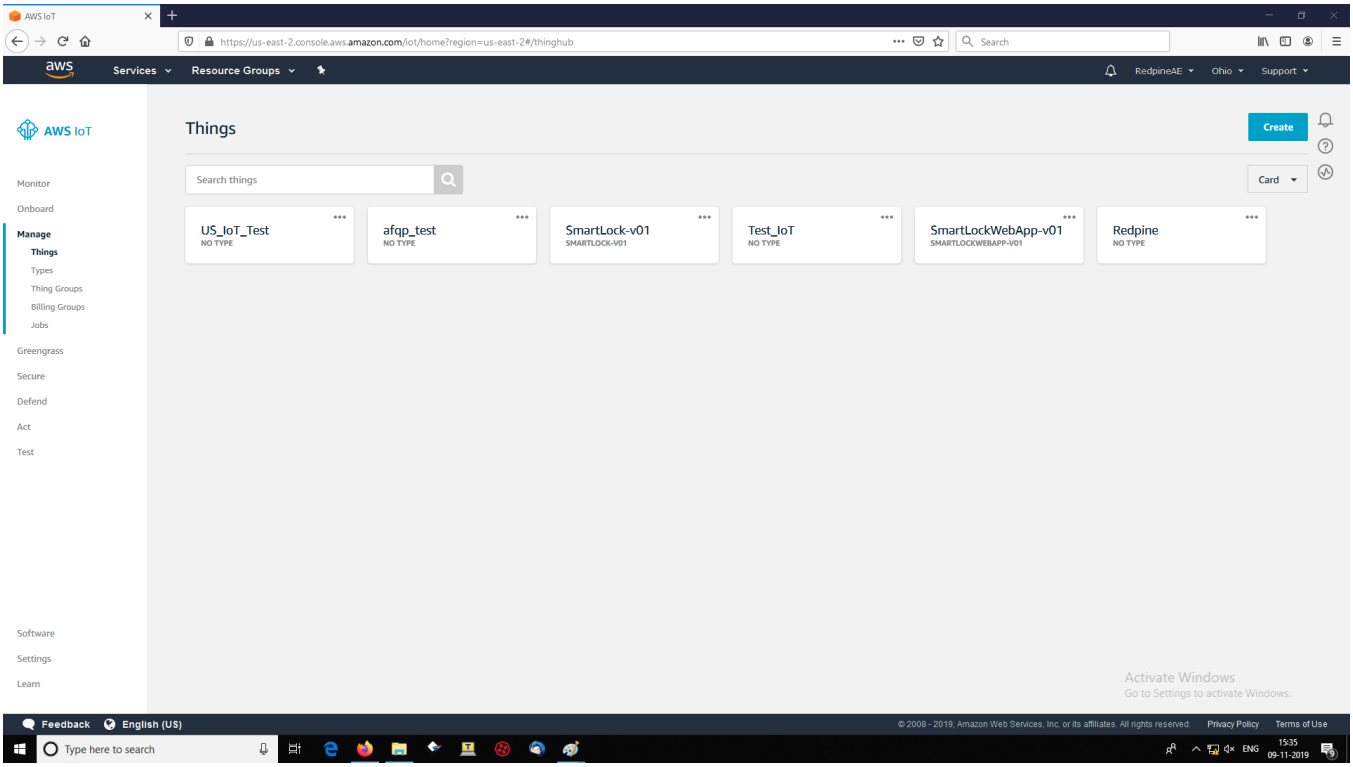


Figure 64: AWS Cloud

2. Click on IoT Thing.

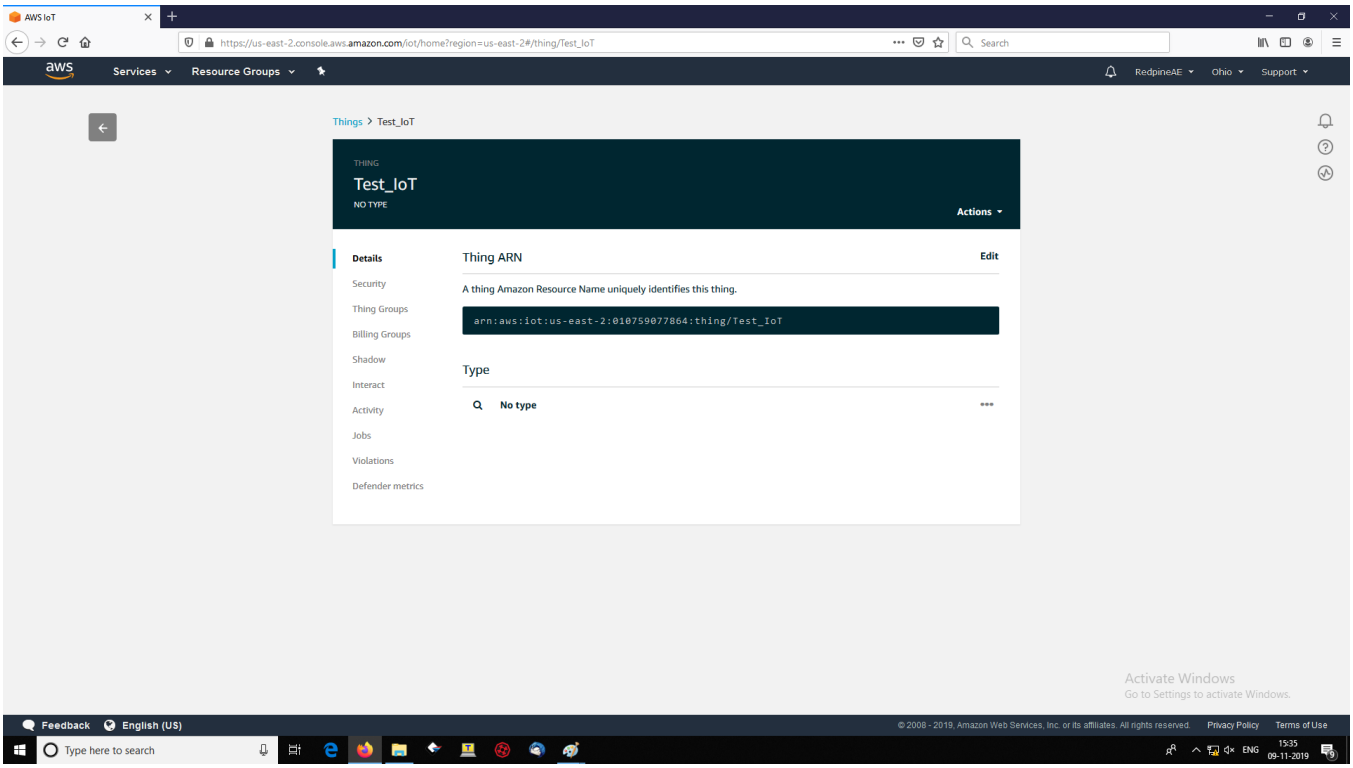


Figure 65: Test_IoT Thing

3. Click on activity window.

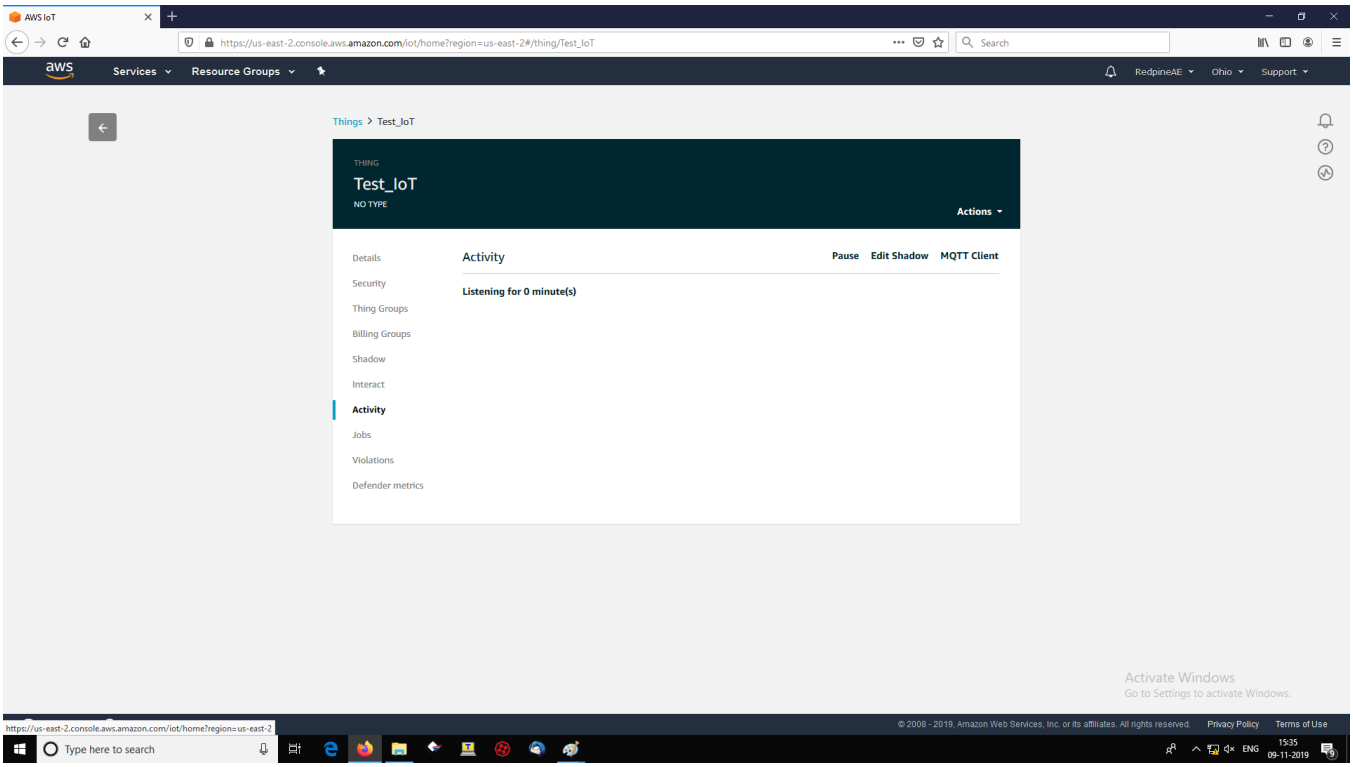


Figure 66: Activity Window

4. Received messages will be like shown below.

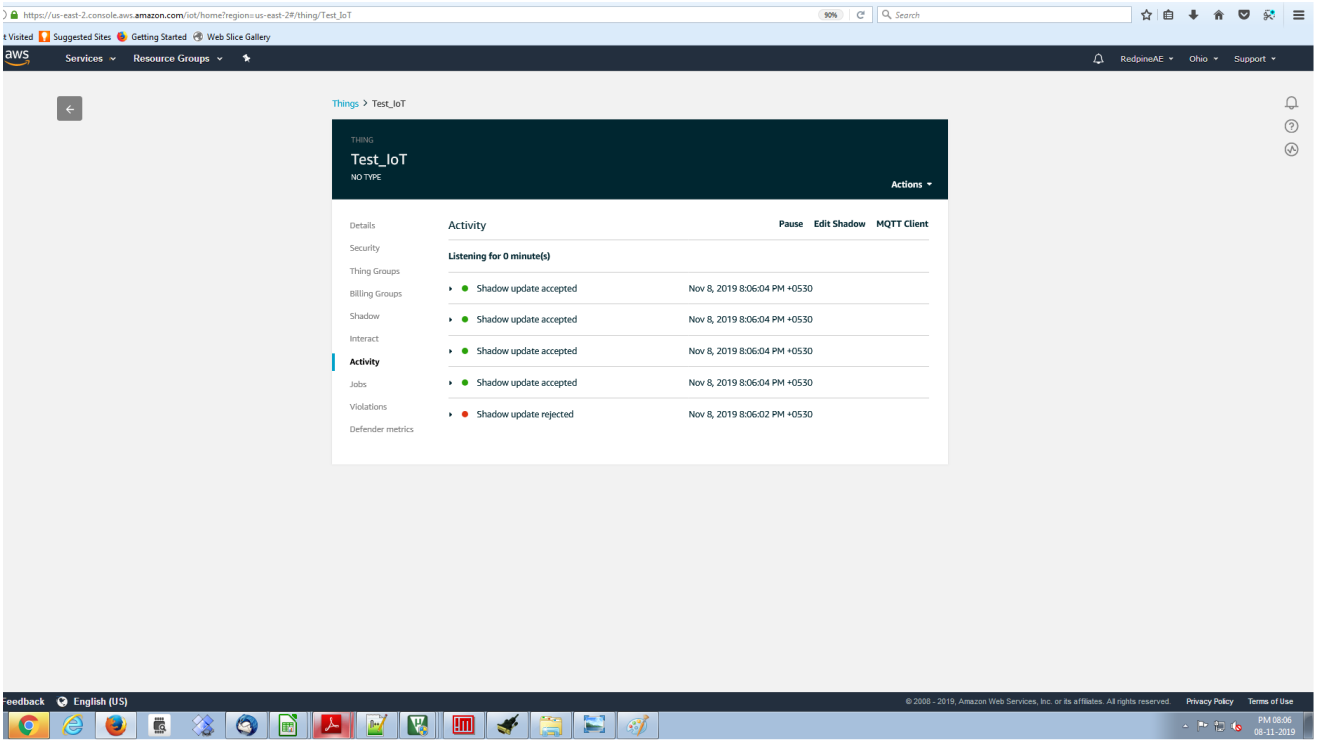


Figure 67: Message Updates

- Follow this below link for API description

http://aws-iot-device-sdk-embedded-c-docs.s3-website-us-east-1.amazonaws.com/index.html

6.3.2 MQTT

Note:

For more information about AWS cloud and Alibaba cloud creation and topics please refer to section "Appendix A: Cloud User Manual".

Protocol Overview

MQTT is a publish-subscribe based "light weight" messaging protocol for using on top of the TCP/IP protocol. The MQTT connection itself is always between one client and the broker, no client is connected to another client directly.

MQTT client
A MQTT client is any device from a micro controller to a full-fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. MQTT Clients can share the information on a particular topic using MQTT protocol. MQTT clients connect to the MQTT broker using TCP connection and can subscribe and publish on any desired topic. The other clients which are subscribed for that topic will receive the published messages.

MQTT Broker
The publish-subscribe messaging pattern requires a message broker. The broker is primarily responsible for receiving all messages, filtering them, deciding like who is interested in it and then sending the message to all subscribed clients.
It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. A simple demonstration of subscribing and publishing of temperature is shown below: Here MQTT broker is present in AWS cloud.

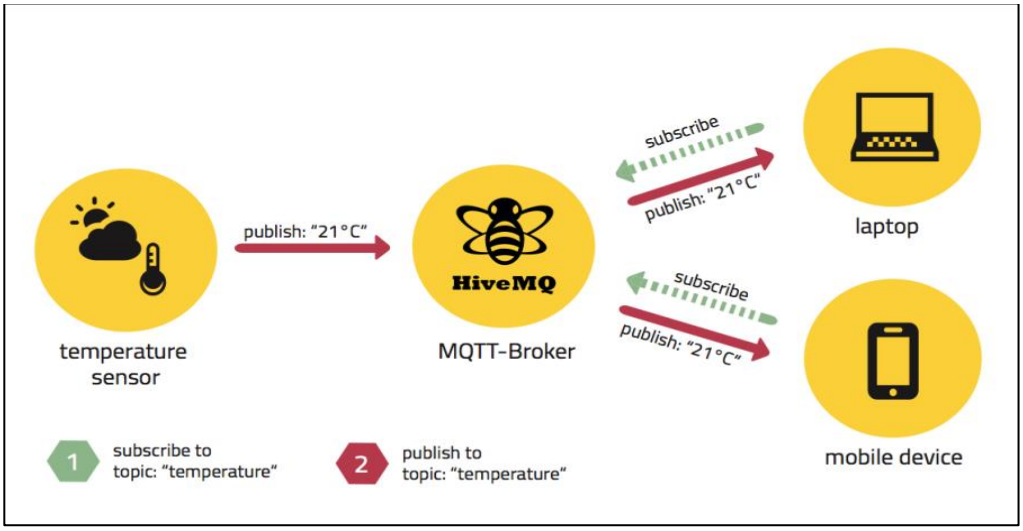


Figure 68: Demonstration of MQTT Protocol

Overview

User has to create a IoT Thing in AWS cloud before running this application and this procedure is described in AWS_User_Guide_v1.0 / Alibaba_User_Guide_v1.0 .And user has to follow the procedure described in the same document to update Domain name and Topics.

This application demonstrates how to configure RS9116W EVK as MQTT client and how to establish connection with MQTT broker present in AWS cloud and how to subscribe, publish and receive the MQTT messages from MQTT broker.
In this application, RS9116W EVK configured as WiFi station and connects to Access Point. After successful WiFi connection, application connects to MQTT broker and subscribe to a topic as discussed in the section 3 of AWS_User_Guide_v1.0 / Alibaba_User_Guide_v1.0 and publishes a message on that subscribed topic and application waits to receive the data published on subscribed topic by other clients.

Sequence of Events

This Application explains user how to:

- Connect to Access Point
- Establish MQTT client connection with MQTT broker in AWS cloud
- Subscribe to a topic
- Publish message on the subscribed topic
- Receive data published by other clients on a subscribed topic

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC1 with Keil or IAR IDE
- RS9116W EVK
- Access point with Internet connection
- AWS account

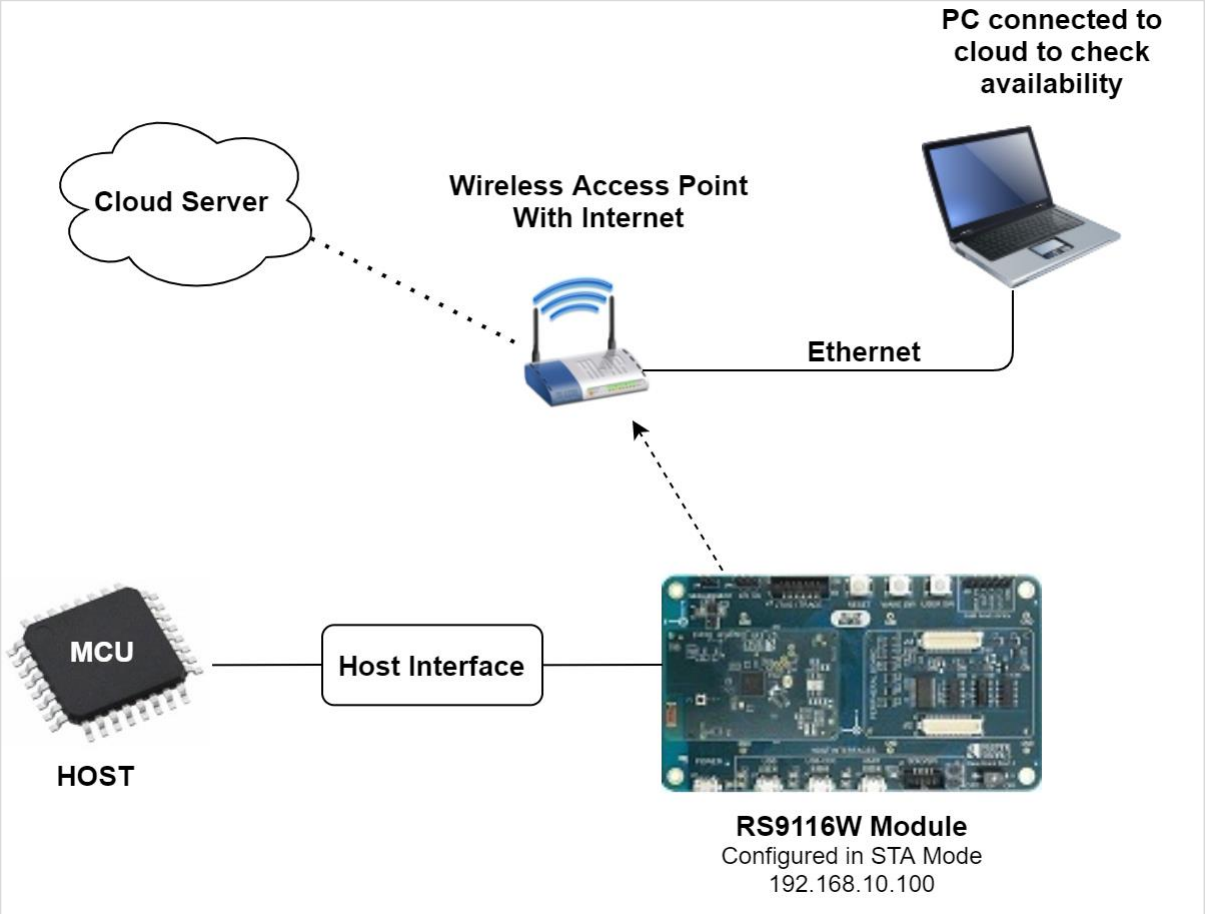


Figure 69: Setup Diagram for MQTT Client Demo

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_mqtt.c** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                  "<REDPINE_AP>"
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                         RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                   "<psk>"
```

To Load certificate

For the information about to get certificates from " AWS cloud please refer sections 2.2 & 3 of AWS_User_Guide_v1.0." / Alibaba_User_Guide_v1.0 .
By default, application loads certificates.

rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pemcertificate into linear array form using python script provided in the release package "***sapis/examples/utilities/certificates/certificate_script.py***".

Example:

If the certificate is wifi-user.pem, enter the command in the following way:
python certificate_script.py ca-cert.pem
The script will generate wifiuser.pem in which one linear array named **cacert** contains the certificate.

- After the conversion of certificate, update *rsi_mqtt.c* source file by including the certificate file and also by providing the required parameters to **rsi_wlan_set_certificate** API.

Once the certificate loads into the device, it will write into the device flash. So, userneed not load certificate for every boot up unless certificate change. So, define LOAD_CERTIFICATE as 0, if certificate is already present in the device.

CLIENT_PORT port refers device MQTT client port number

```
#define CLIENT_PORT                          5001
```

SERVER_PORT port refers remote MQTT broker/server port number

```
#define SERVER_PORT                          8883
```

SERVER_IP_ADDRESS refers remote peer IP address (Windows PC2) to connect with MQTT broker/server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS                    0x6400A8C0
```

MQTT client keep alive period

```
#define RSI_KEEP_ALIVE_PERIOD                100
```

Memory to initialize MQTT client Info structure

```
#define MQTT_CLIENT_INIT_BUFF_LEN            3500
```

Global buffer or memory which is used for MQTT client initialization. This buffer is used for the MQTT client information storage. **uint8_t mqqt_client_buffer[MQTT_CLIENT_INIT_BUFF_LEN];**
QOS indicates the level of assurance for delivery of an Application Message.
QoS levels are:
0 - At most once delivery
1 - At least once delivery
2 - Exactly once delivery

```
#define QOS                                  0
```

RSI_MQTT_TOPIC refers to which topic WiSeConnect MQTT client is supposed to subscribe.

```
#define RSI_MQTT_TOPIC                       "" (Update with a Topic name of IoT Thing in AWS
cloud)
```

MQTT Client ID with which MQTT client connects to MQTT broker
uint8_t clientID[] = "Redpine" //Can be changed by user
User name for login credentials
int8_t username[] = "username"
Password for login credentials
int8_t password[] = "password"

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                 15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                       1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through **STATIC** then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order. Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order. Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order. Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

The following parameters are configured if OS is used. WLAN task priority is given and this should be of low priority

```
#define RSI_WLAN_TASK_PRIORITY          1
```

Driver task priority is given and this should be of highest priority

```
#define RSI_DRIVER_TASK_PRIORITY        1
```

WLAN Task stack size is configured by this macro

```
#define RSI_WLAN_TASK_STACK_SIZE        500
```

Driver Task stack size is configured by this macro

```
#define RSI_DRIVER_TASK_STACK_SIZE      500
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT|TCP_IP_FEAT_SSL
|TCP_IP_FEAT_DNS_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID#define
RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_256k_MODE

#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point with Internet connection in OPEN/WPA-PSK/WPA2-PSK mode to connect RS9116W EVK in STA mode.

2.Log into AWS account and open IoT Thing created.

3. Update Domain name and topics name to connect to AWS mqtt cloud and publish message to IoT Thing.

4. Receive published data from cloud by subscribing to a topic. Publish to same topic from IoT Thing in AWS cloud.

5. User can see the published message in cloud by following the procedure given in the "section 3 of AWS_User_Guide_v1.0." / Alibaba_User_Guide_v1.0

6.3.3 SSL Client

Instance in AWS Cloud

User must create instance in AWS cloud to open a TCP server in the cloud. For this please refer to section "Appendix A: Cloud User Manual".

SSL Overview

SSL stands for Secure Sockets Layer. SSL is the standard security technology for establishing an Encrypted link between a web server and a browser. This link ensures that all data passed between the web servers and the browsers remain Private & Integral.
Data encryption, Server authentication, Message integrity, Optional client authentication for a TCP / IP connection are the main objectives of SSL protocol**.**

Overview

User has to create instance in AWS cloud to open a TCP server in the cloud. This application demonstrates how to open and use a standard TCP client socket with secure connection using SSL and sends data to a remote server in AWS cloud.

Sequence of Events

This Application explains user how to:

- Connect the device to an Access point with internet connection and get IP address through DHCP
- Open TCP Server socket over SSL at the Instance created in AWS cloud as described in the section "Executing the Application" given below.
- Establish TCP connection over SSL
- Send TCP data from client to remote device

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon lab Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC1 with KEIL or IAR IDE
- RS9116W EVK
- WiFi Access point with internet connection
- Linux PC to log into an instance running in AWS cloud.
- AWS account

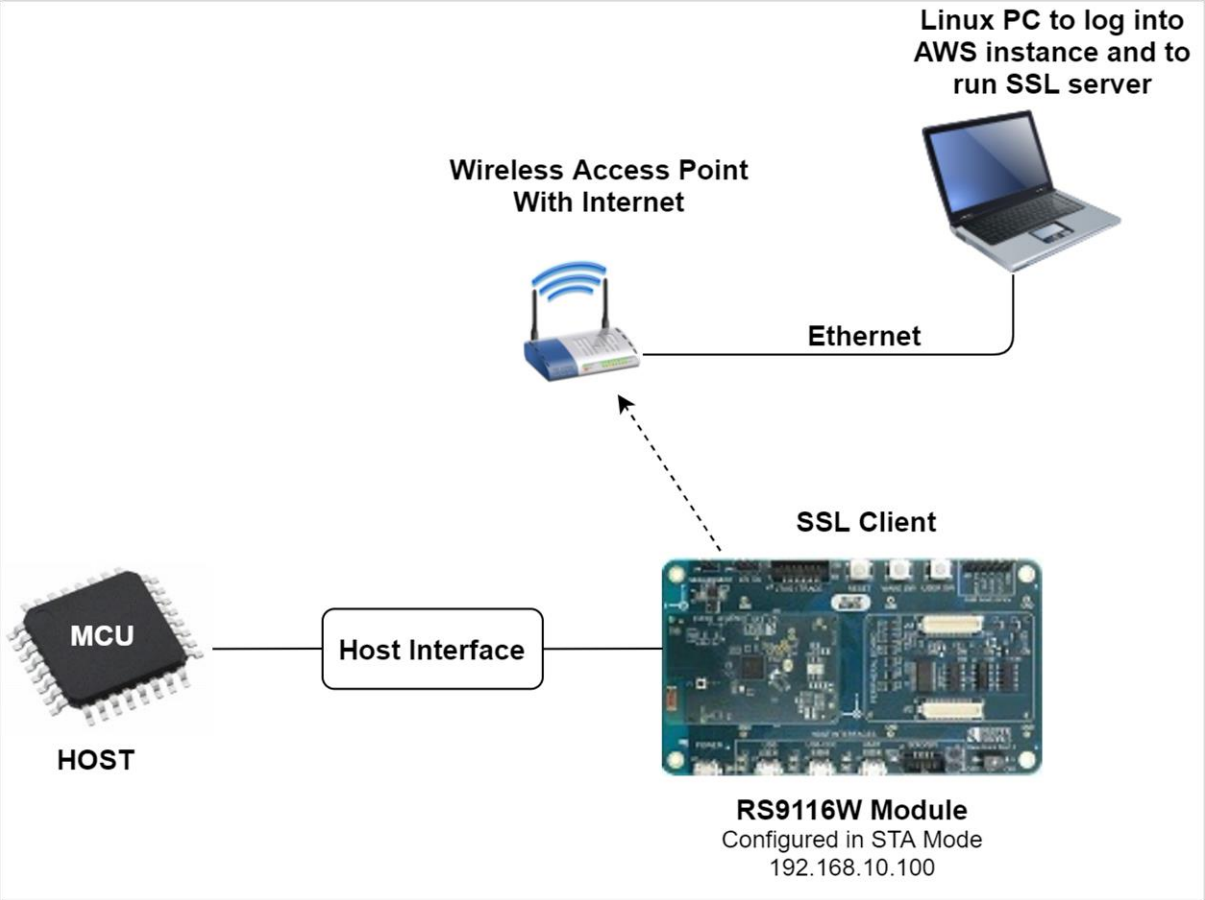


Figure 70: Setup Diagram for SSL Client Demo

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ssl_client.c*** file and update / modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                          "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels

```
#define CHANNEL_NO                    0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                   RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                             "<psk>"
```

To Load certificate

```
#define LOAD_CERTIFICATE                1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.
By default, application loading **"cacert.pem"** certificate if **LOAD_CERTIFICATE** enabled. In order to load different certificate, user has to follow the following steps:

- rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pemcertificate into linear array form using python script provided in the release package "***sapis/examples/utilities/certificates/certificate_script.py***".

Example:

If the certificate is wifi-user.pem, enter the command in the following way:
python certificate_script.py ca-cert.pem
The script will generate wifiuser.pem in which one linear array named **cacert** contains the certificate.

- After the conversion of certificate, update *rsi_ssl_client.c* source file by including the certificate file and also by providing the required parameters to **rsi_wlan_set_certificate** API.

Once the certificate loads into the device, it will write into the device flash. So, userneed not load certificate for every boot up unless certificate change. So, define LOAD_CERTIFICATE as 0, if certificate is already present in the device.

> **Note:**
>
> All the certificates are given in the release package. Path: sapis/examples/utilities/certificates
> Enable **SSL** macro to open SSL socket over TCP.

```
#define SSL                             1
```

DEVICE_PORT port refers SSL client port number

```
#define DEVICE_PORT                     5001
```

SERVER_PORTportrefers remote SSL server port number which is opened in Windows PC2

```
#define SERVER_PORT                     5001
```

SERVER_IP_ADDRESS refers remote peer IP address of instance created in AWS cloud to connect with SSL server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS               0x6400A8C0
```

NUMEBR_OF_PACKETS refers how many packets to send from TCP client

```
#define NUMBER_OF_PACKETS                10000
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                  15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC.

```
#define DHCP_MODE                        1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                        0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                          0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                          0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                  RSI_DISABLE
#define RSI_FEATURE_BIT_MAP              FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_CLIENT|TCP_IP_FEAT_SSL)
#define RSI_CUSTOM_FEATURE_BIT_MAP       FEAT_CUSTOM_FEAT_EXTENTION_VALID

#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP EXT_FEAT_256k_MODE

#define RSI_BAND RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point with Internet connection in OPEN/WPA-PSK/WPA2-PSK mode to connect RS9116W EVK in STA mode.

2. To create an instance in AWS cloud refer section 4 of AWS_User_Guide_v1.0.

3. A pem file will be generated while creating the instance. User has to download this pem file and has to change owner permissions of the file to test instead of root using below command.

chown test *.pem

This file should be given as an input to access instance as shown in step4.

4.Using a linux pc connect to internet and log on to instance which we have created in AWS cloud using the below command.

$ ssh -i "*.pem" ubuntu@ip_addr of instance

$ sudo su

5.Copy SSL_Server_throughput_d.py in host/sapis/examples/utilities/scripts to the instance with below commad.

$scp -i "*.pem" SSL_Server_throughput_d.py ubuntu@ip_addr:/home/test/

6. Copy server-cert.pem ,server-key.pem in sapis/examples/utilities/certificates to the instance using below command.

$scp -i "*.pem" server-cert.pem ubuntu@ip_addr:/home/test/

$scp -i "*.pem" server-key.pem ubuntu@ip_addr:/home/test/

7. Run ssl server python script in linux pc using below command.

python SSL_Server_throughput_d.py

8.Update SERVER_IP_ADDRESS in the application with IP address of instance created in AWS cloud.

9.Compile and run the example application in the client then tcp server in AWS cloud will receive data from client.

6.4 Concurrent Mode

Overview

This application demonstrates how to configure the device in both Wi-Fi Station mode and Access Point mode and how to transfer data on both modes.
In this Application, device starts as an Access Point and connects with an Access Point in station mode. After successful creation of access point and successful connection with the same, application opens TCP socket and transfers TCP data in station mode and device responds for Ping request sent by connected station with Ping Reply in Access Point mode.

Sequence of Events

This Application explains users how to:

- Create RS9116W device as Soft Access point
- Connect with 3rd party Access Point in Station mode
- Open TCP server socket on configured port number on the device.
- Send TCP data to remote peer in station mode

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- Access point
- Windows PC2

- A TCP server application running on the Wi-Fi station (This example uses iperf for windows)

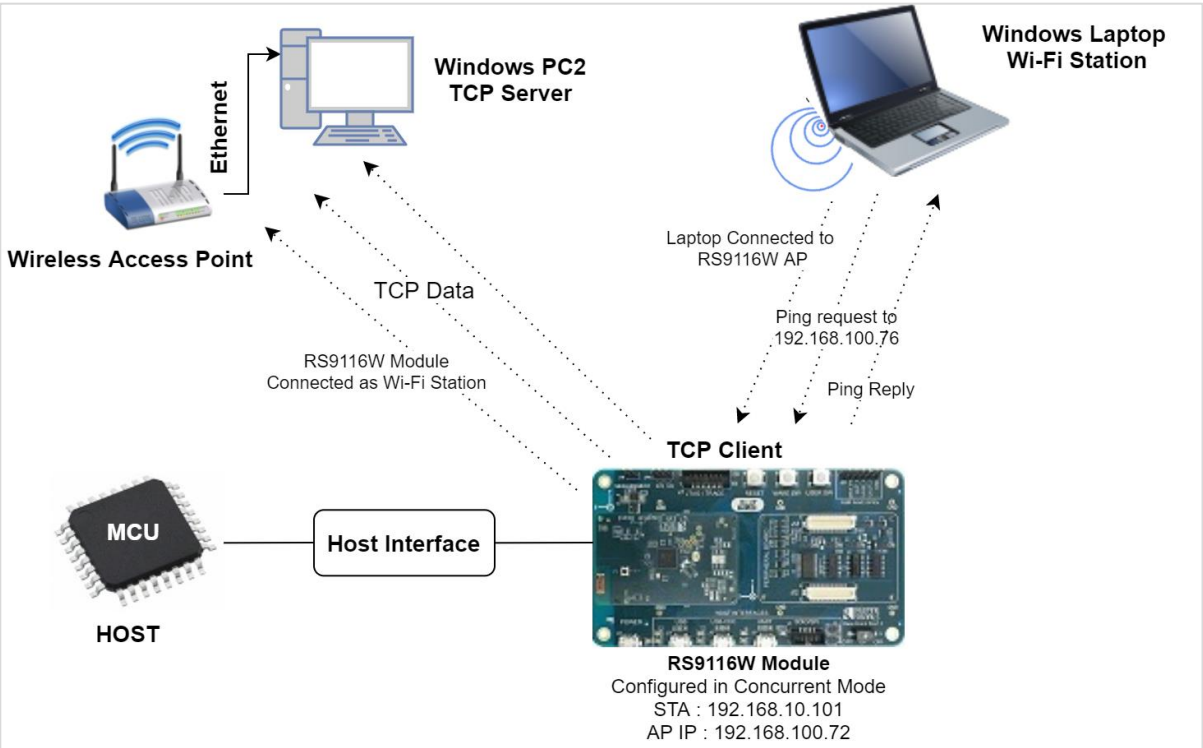


Figure 71: Setup Diagram for Concurrent Mode Example

Configuration and steps for Execution

Configuring the Application

- Open **rsi_concurrent_mode.c** file and update/modify following macros:
- **SSID** refers to the name of the Access point.

```
#define SSID                          "REDPINE_AP"
```

STA_SECURITY_TYPE refers to the type of security. In concurrent mode STA supports Open, WPA and WPA2 securities.

Valid configurations are:

- **RSI_OPEN** - For OPEN security mode
- **RSI_WPA** - For WPA security mode
- **RSI_WPA2** - For WPA2 security mode

```
#define STA_SECURITY_TYPE             RSI_OPEN
```

STA_PSK refers to the STA secret key to connect with the secured Access Point.

```
#define STA_PSK                       NULL
```

AP_SSID refers to the name of the WiSeConnect Access point would be created.

```
#define AP_SSID                       "REDPINE_AP"
```

AP_CHANNEL_NO refers to the channel in which AP would be started

```
#define AP_CHANNEL_NO                11
```

6.5 Connection Using Asynchronous APIs App

Asynchronous APIs Overview

Asynchronous APIs instructs the module and returns the status. The actual response by the module is indicated to the application from the registered call backs. So the driver need not indefinitely wait for the response from the module, it can schedule its tasks.

Overview

The application demonstrates how the RS9116W will be connected to an access point using Asynchronous APIs. After successful connection, Silicon Labs device opens TCP socket with remote peer and sends TCP data on opened socket.

Sequence of Events

This Application explains user how to:

- Scan for Access Point using Asynchronous API
- Handle scan responses
- Connect to Access Point using Asynchronous API
- Handle Join response
- Open TCP client socket on configured port number on the device.
- Send TCP data from device to remote peer.

Example Setup

The WiSeConnect parts requires that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Access Point
- Windows PC2
- A TCP Server application running on the Windows PC2 (This example uses iperf for windows)

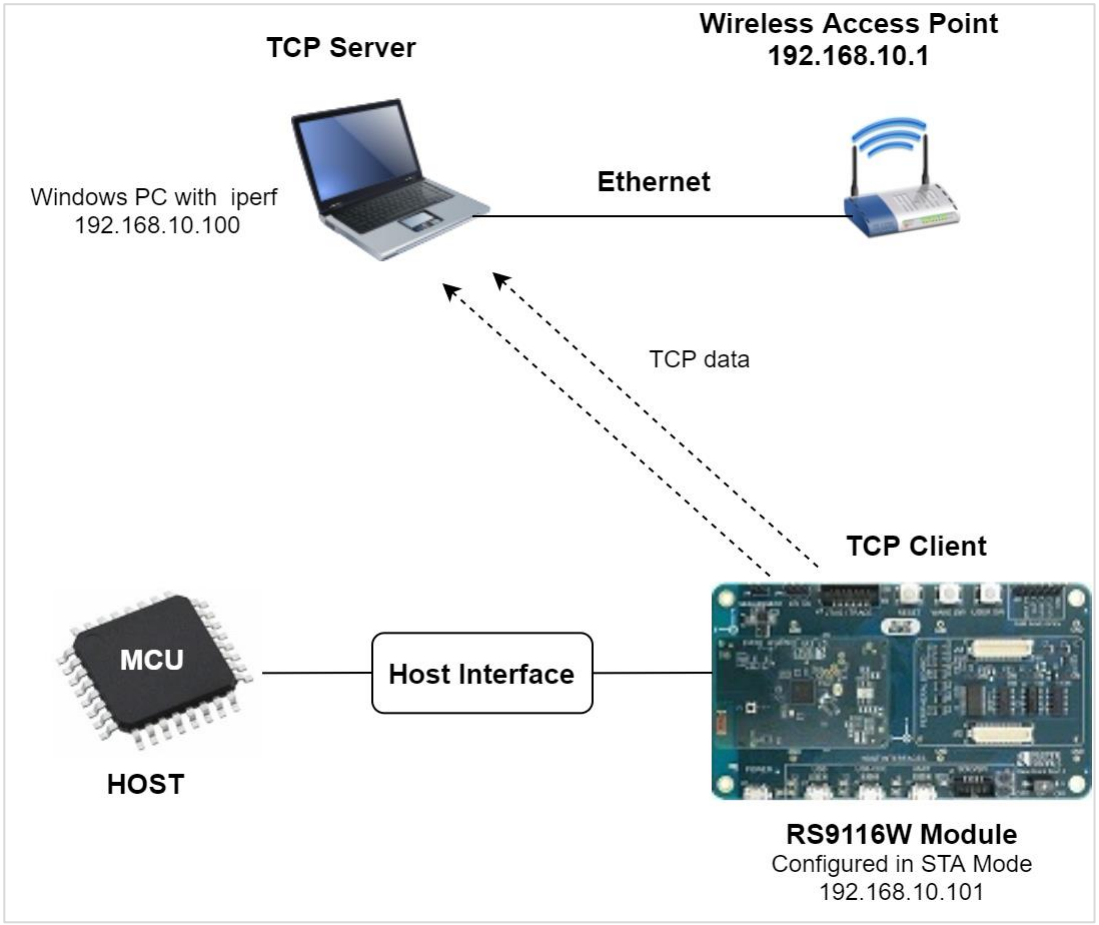


Figure 72: Setup dDiagram for Connection Using Asynchronous APIs Example

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_connection_using_asynchronous_apis_app.c** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                          "AP_SSID"
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK and WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                 RSI_OPEN
```

   PSK refers to the secret key if the Access Point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                           NULL
```

   DEVICE_PORT port refers module TCP client port number

```
#define DEVICE_PORT                   5001
```

SERVER_PORT port refers remote TCP server port number which is opened in Windows PC2.

```
#define SERVER_PORT                     5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS               0x640AA8C0
```

NUMEBR_OF_PACKETS refers how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS               1000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                       1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through **STATIC** then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

Open **rsi_wlan_config.h** file and update/modify following macros:

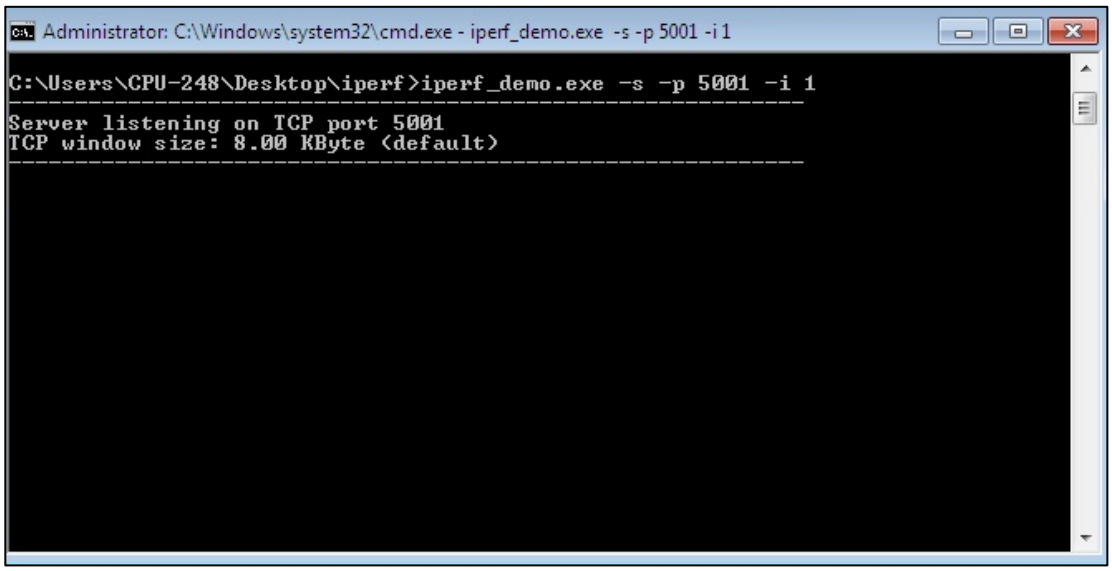
```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_CLIENT
```

```
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

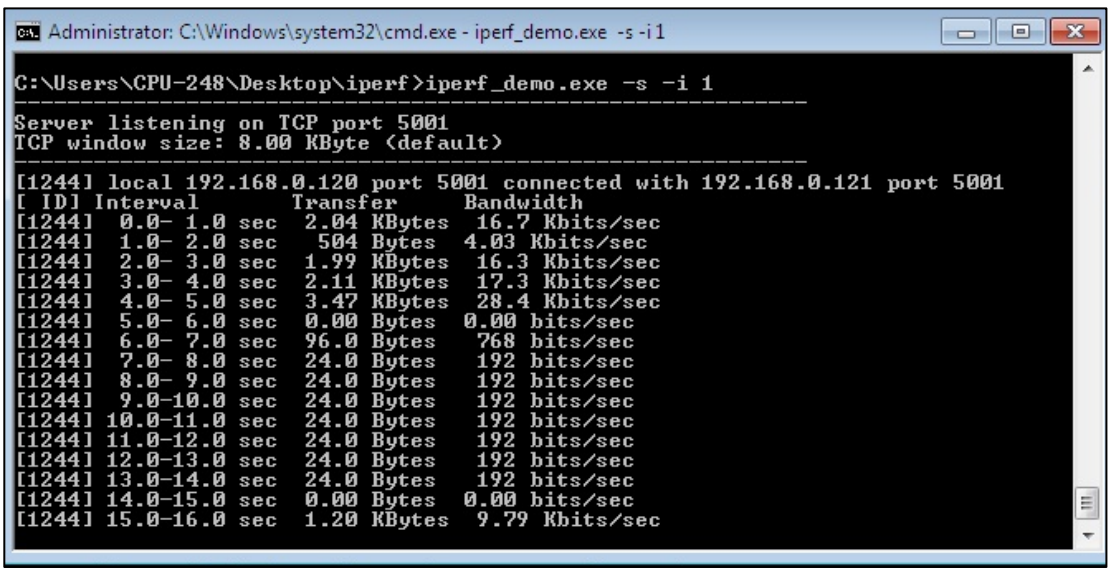
Note:
The **rsi_wlan_config.h** file is already set with the desired configuration in respective example folders, user need not to change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Open TCP server application using iperf application in Windows PC2 which is connected to Access point through LAN.
   iperf_demo.exe -s -p **<SERVER_PORT> -i 1**



3. After the program gets executed, RS9116W device scans and will get connected to the access point and get IP.
4. After successful connection, device STA connects to TCP server socket opened on Windows PC2 using TCP client socket and sends configured **NUMBER_OF_PACKETS** to remote TCP server. Refer the below image for reception of TCP data on TCP server



6.6 Customized Root Webpage

Overview

The Custom root webpage application demonstrates how to customize the root webpage.
In this application, RS9116W device starts as an Access point. After successful creation of Access Point, user can open root webpage by connecting to HTTP server running in device.

Sequence of Events

This Application explains user how to:

- Start Silicon Labs device as Access Point and load a custom webpage to the root webpage.
- Connect a station to the device and get IP address through DHCP.
- Open root webpage of the Device from the browser of the connected station (STA).
- The root webpage now has the desired page instead of the default page.

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Wireless Access Point
- Smart Phone
- Silicon Labs module

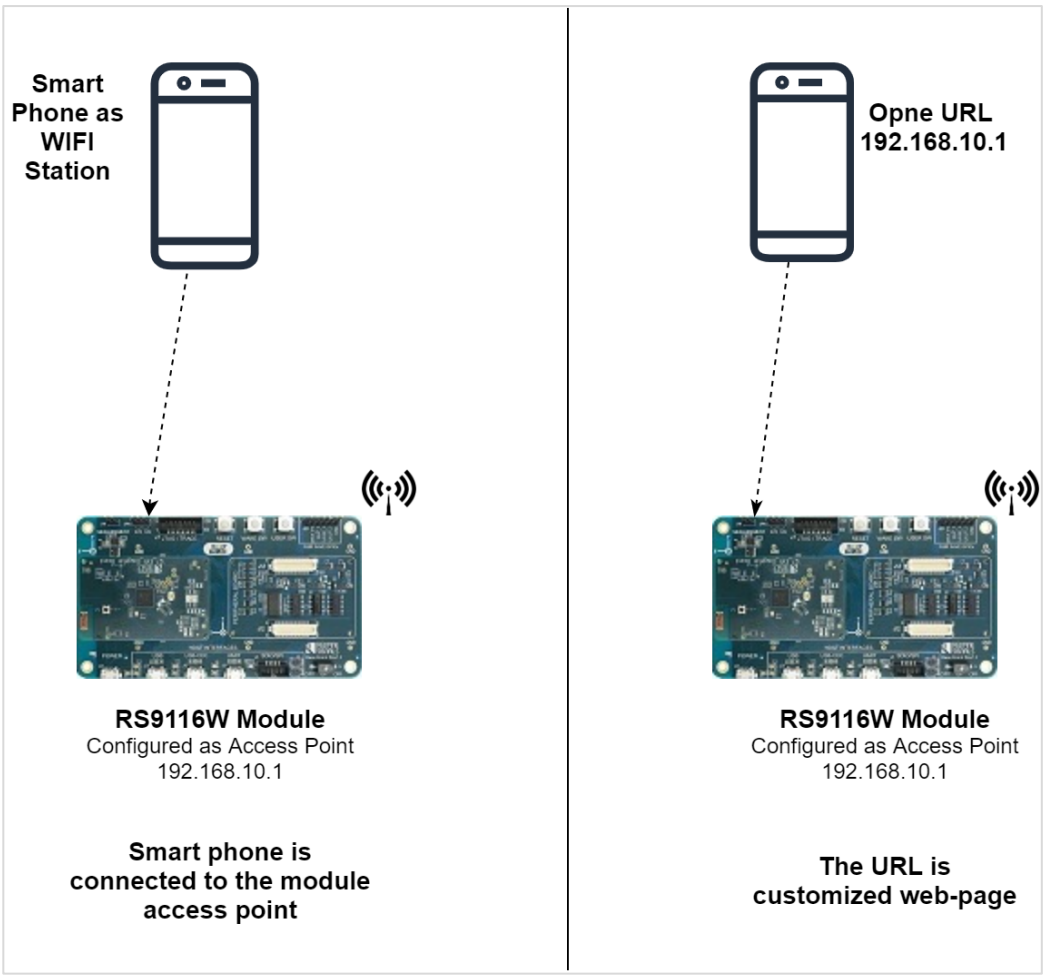


Figure 73: Setup Diagram for Customized Web Page

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi*_customized_root_webpage.c_file** and update/modify the following macros.

   SSID refers to the name of the Access point.

```
#define SSID                              "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                        11
```

> **Note:**
>
> lid values for **CHANNEL_NO** are 1 to 11 in 2.4GHz band and 36 to 48 & 149 to 165 in 5GHz band. In this example default configured band is 2.4GHz. So, if user wants to use 5GHz band then the user has to set **RSI_BAND** macro to 5GHz band in **rsi_wlan_config.h** file.

   SECURITY_TYPE refers to the type of security. Access point supports Open, WPA, WPA2 securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                     RSI_WPA2
```

   ENCRYPTION_TYPE refers to the type of Encryption method .Access point supports OPEN, TKIP, CCMP methods.
   Valid configuration is:
   RSI_CCMP - For CCMP encryption
   RSI_TKIP - For TKIP encryption
   RSI_NONE - For open encryption

```
#define ENCRYPTION_TYPE                  <RSI_CCMP>
```

   PSK refers to the secret key if the Access point to be configured in WPA/WPA2 security modes.

```
#define PSK                               "1234567890"
```

   BEACON_INTERVAL refers to the time delay between two consecutive beacons in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

```
#define BEACON_INTERVAL                   100
```

   DTIM_INTERVAL refers DTIM interval of the Access Point. Allowed values are from 1 to 255.

   #define DTIM_INTERVAL 4

   To configure IP address
   IP address to be configured in the device should be in long format and in little endian byte order.

Example: To configure "192.168.10.1" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0.**

```
#define DEVICE_IP                       0X010AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0X00FFFFFF
```

Note:

In AP mode, configure same IP address for both **DEVICE_IP** and **GATEWAY** macros

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                 DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_PSK
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_SERVER|TCP_IP_FEAT_HTTP_SERVER
|TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_EXT_TCPIP_FEATURE_BITMAP    EXT_TCP_IP_HTTP_SERVER_BYPASS
```

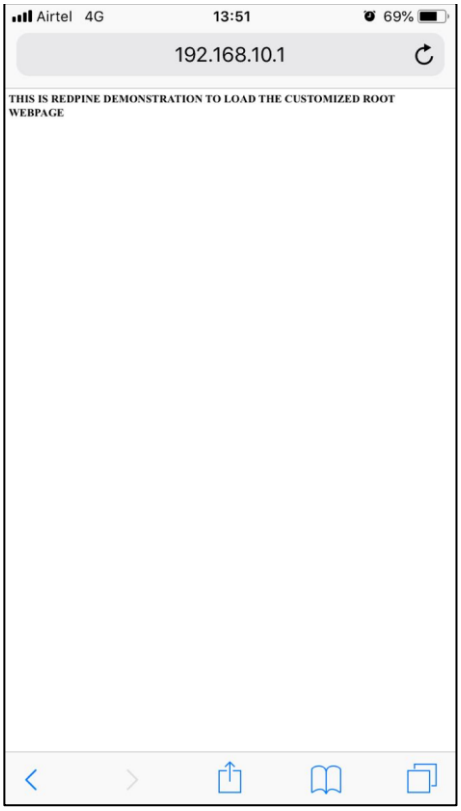
```
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP             EXT_FEAT_256K_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs Device will be started as an Access point having the configuration same as that of in the application.
2. Now connect a Smart phone(STA) to Device and get IP address.
3. After successful connection open the provisioning page from STA browser by giving the following URL:
   URI: DEVICE_IP
   When the webpage is loaded, user can see the desired webpage instead of the default webpage.



6.7 DHCP User Class

Protocol Overview

This Option is used by a DHCP client to optionally identify the type or category of user or application it represents. A DHCP server uses the user class option to choose the address pool, it allocates an address from and to select any other configuration options.

Overview

This application demonstrates how DHCP USER CLASS option can be used. In this application, the device connects to the Access Point.

Sequence of Events

This Application explains user how to:

- Connect to Access Point
- Send DHCP User class option

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- LINUX PC
- Silicon Labs Module
- Wi-Fi Access point

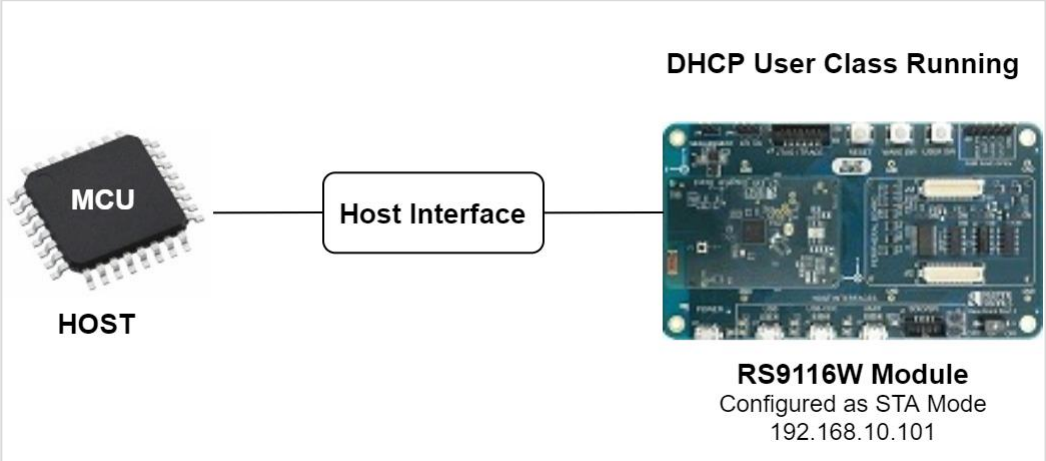


Figure 74: Setup Diagram for DHCP User Class Example

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_dhcp_user_class_app.c** file and update/modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                          "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels.

```
#define CHANNEL_NO                    0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                 RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                           "<psk>"
```

   To configure DHCP user class Parameters
   //MAX count is 4

```
#define RSI_DHCP_USER_CLASS_COUNT     4
```

   //MAX LENGTH 33 including NULL

```
#define RSI_DHCP_USER_CLASS_DATA_1    "Redpine-class1"
#define RSI_DHCP_USER_CLASS_DATA_2    "Redpine-class2"
#define RSI_DHCP_USER_CLASS_DATA_3    "Redpine-class3"
#define RSI_DHCP_USER_CLASS_DATA_4    "Redpine-class4"
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                    1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through **STATIC** then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                    0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                      0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                       0x00FFFFFF
```

2. Open **rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                RSI_DISABLE
#define RSI_FEATURE_BIT_MAP            FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS              RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP     (TCP_IP_FEAT_DHCPV4_CLIENT|TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP     FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP    EXT_FEAT_256k_MODE
#define RSI_BAND                       RSI_BAND_2P4GHZ
#define RSI_EXT_TCP_IP_FEATURE_BIT_MAP
EXT_TCP_FEAT_DHCP_OPT                  77
```

Note:

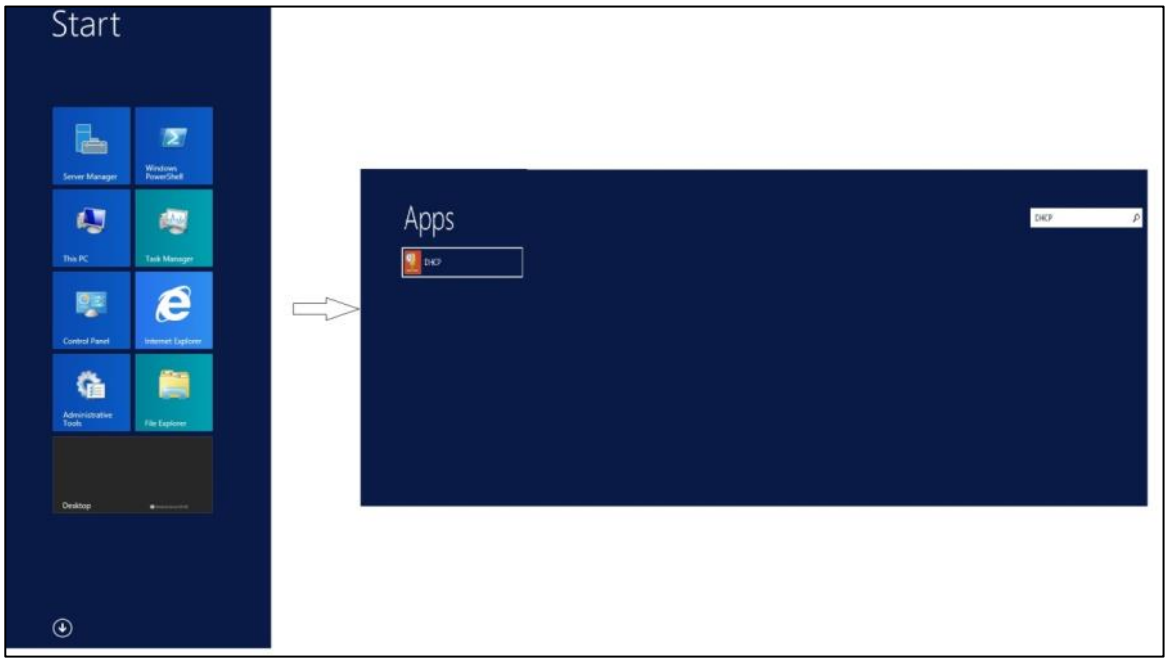
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not to change for each example.

Executing the Application

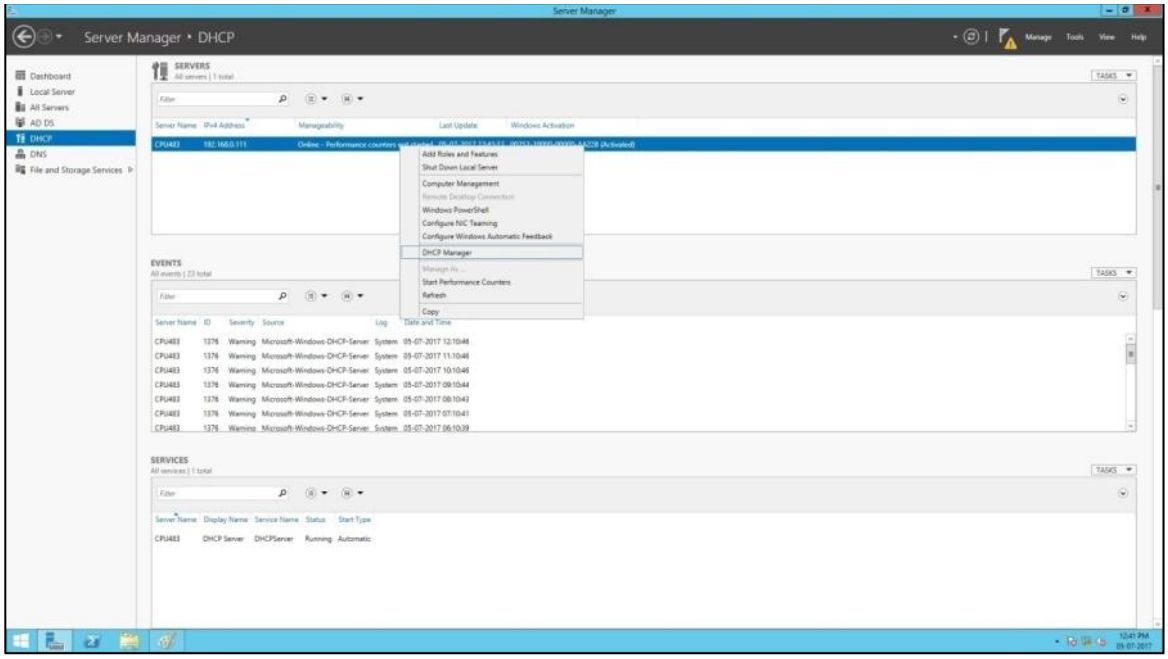
1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. After the program gets executed, the device would be connected to access point .
3. After successful connection with Access Point, Send DHCP command after getting join response.

WINDOWS

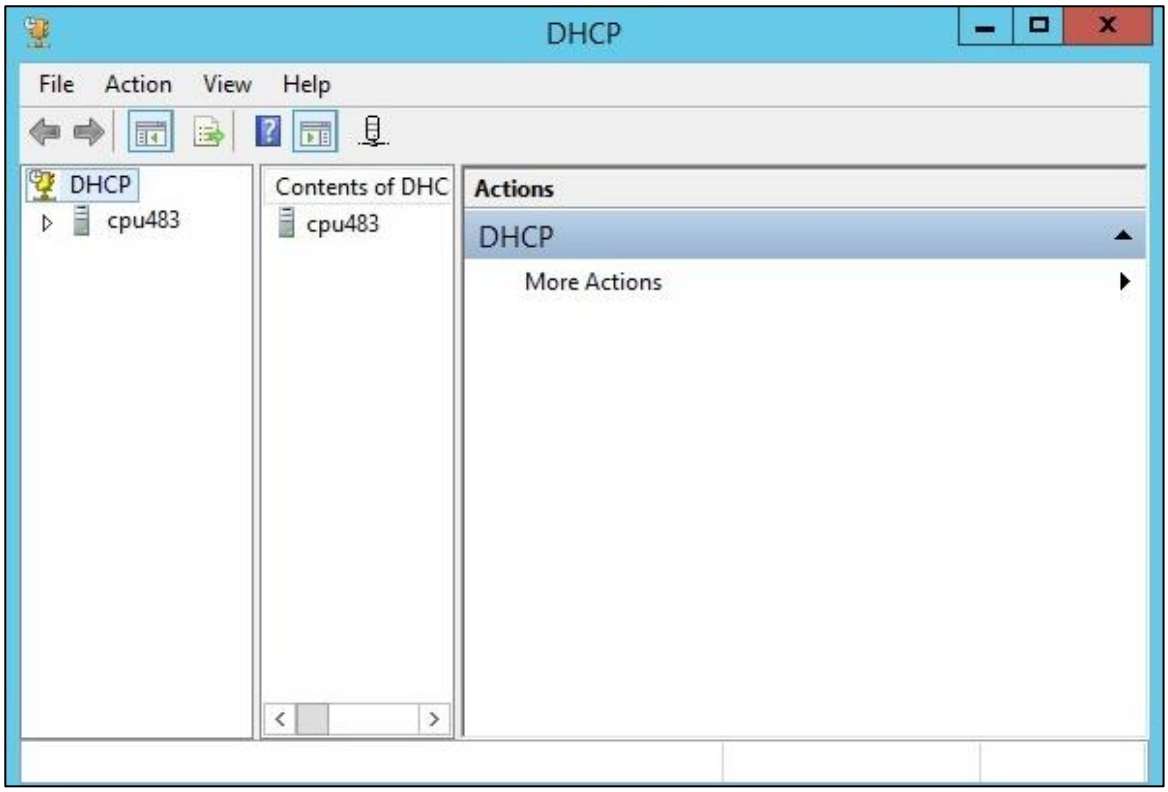
1. Bring up Windows Server 2012.
2. Click on Windows button and select "All apps", search for "DHCP". It opens Server manager.



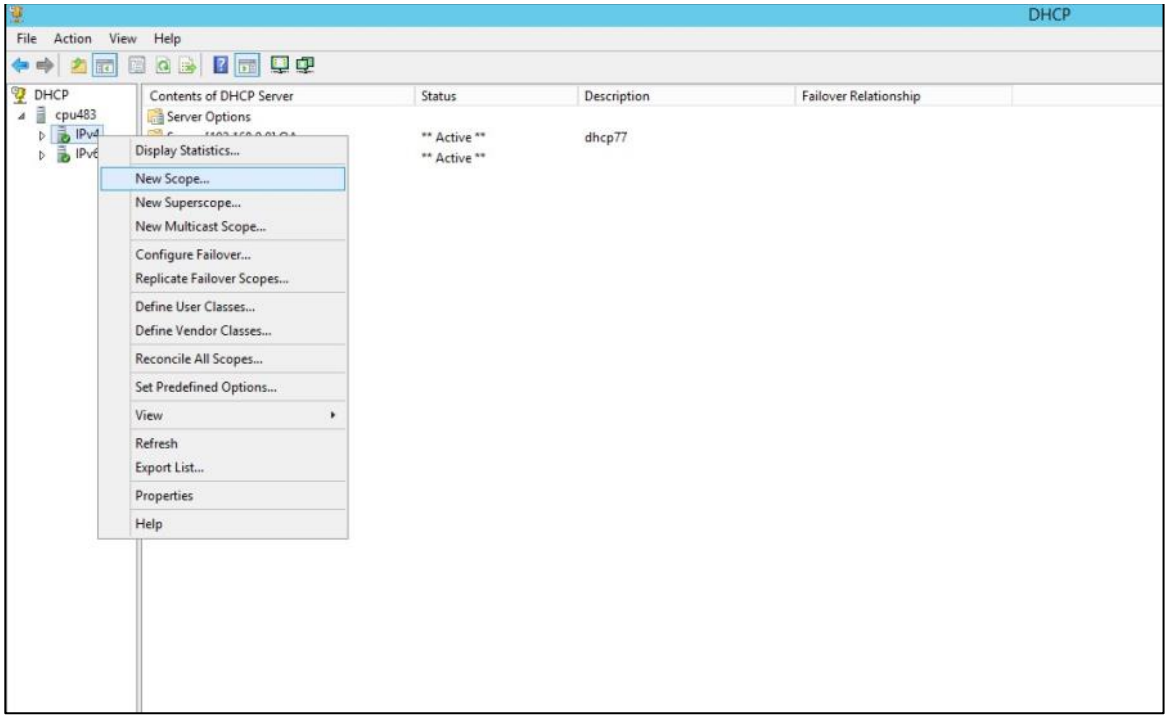
3. Click on DHCP under the Dashboard on the left side of the panel. In the right side of the panel select Server name, right click and select DHCP manager as shown in the below image.



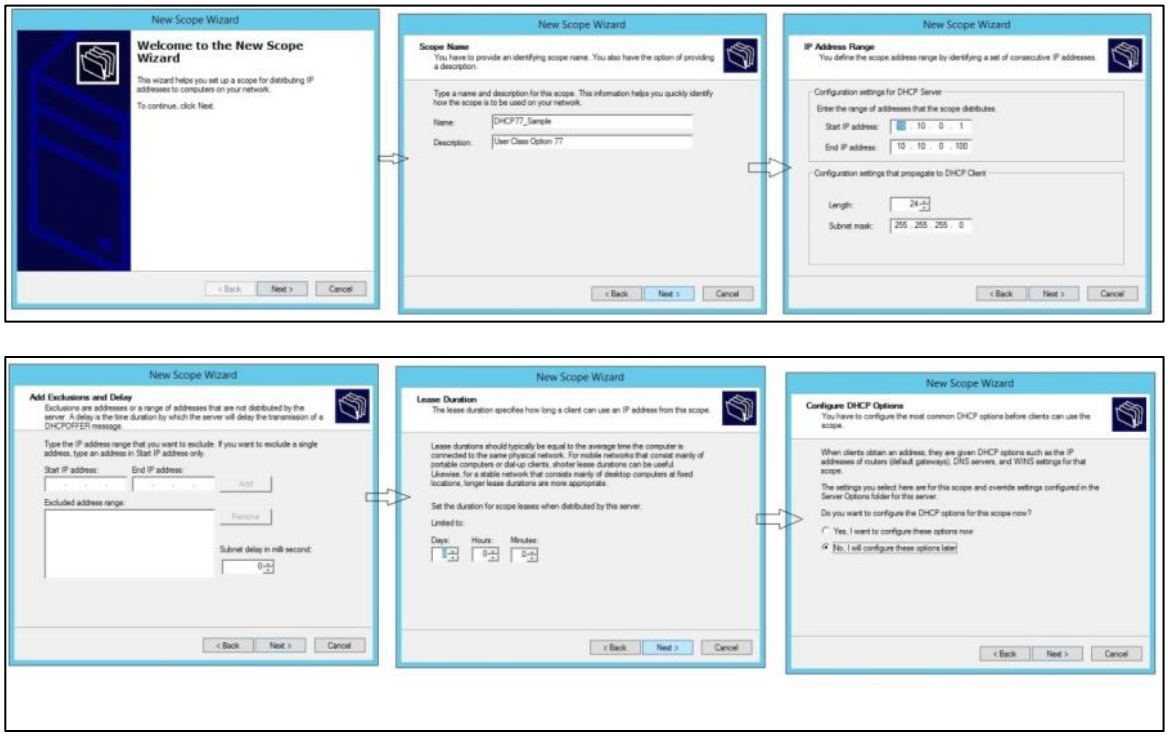
4. When DHCP manager is selected it opens a DHCP window as shown below.



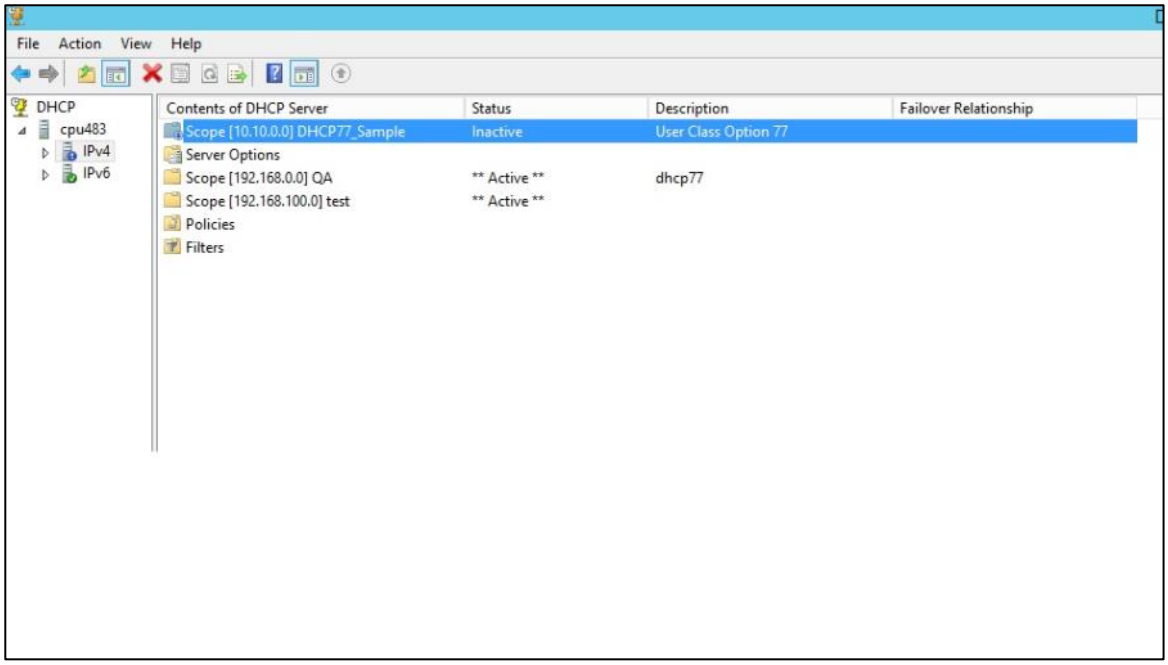
5. In the left panel select "ipv4" under DHCP and right click to select "New Scope".

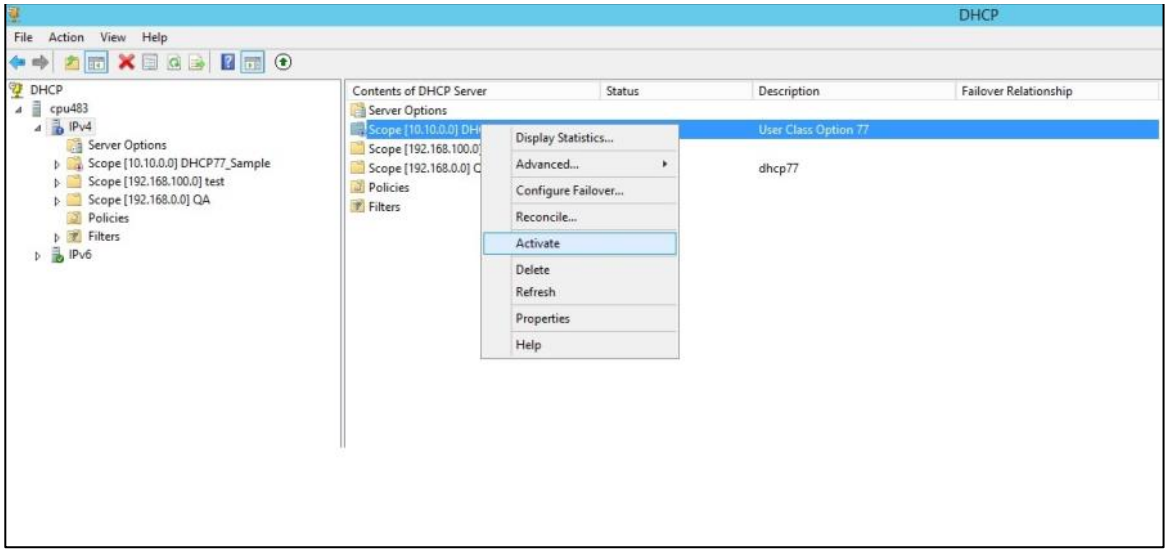


6. In the New scope wizard enter the scope name, IP address pool as required, exclusion IP address range and Lease time (usually these couple of options are not needed). Select the option "No" when dialogue box asks to set an option for settings default gateways, DNS servers and WINS settings for the current New scope created. Refer the below set of pictures.

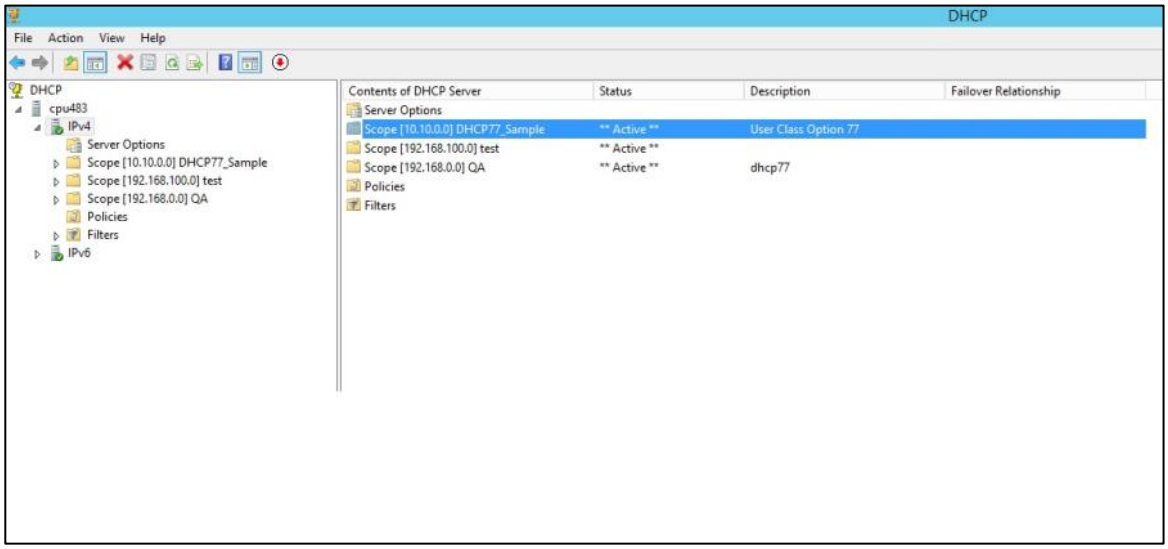


7. This creates the New scope (DHCP77_Sample is shown as reference for New scope in the pics). The New scope created is inactive by default. Right click and Activate as shown below:

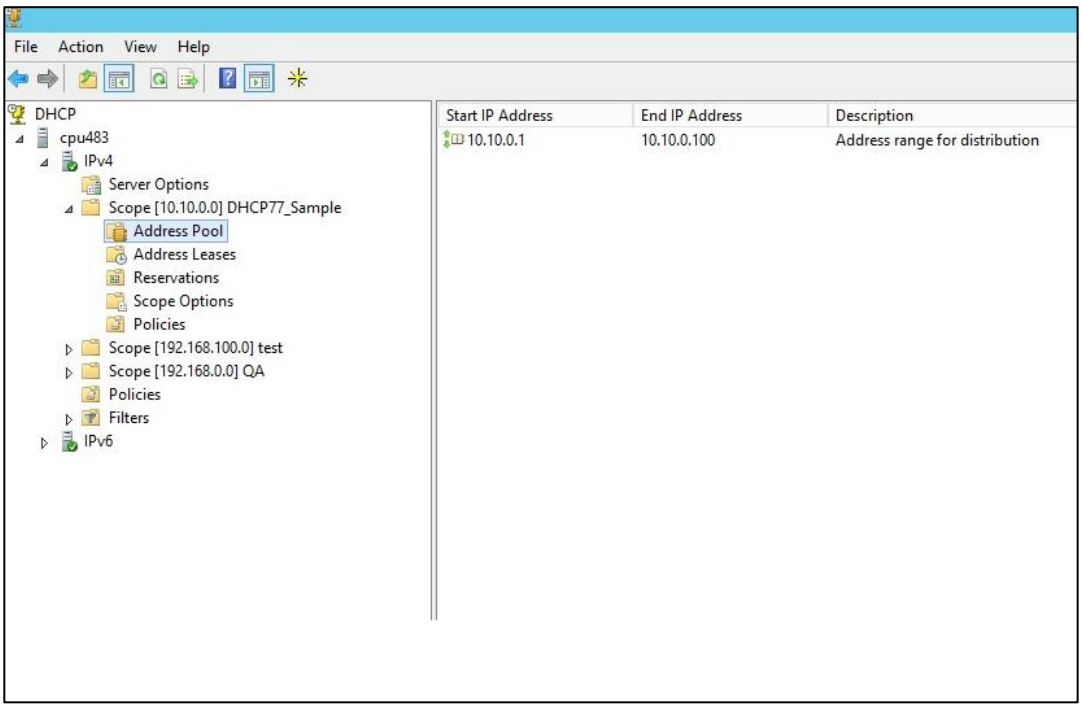




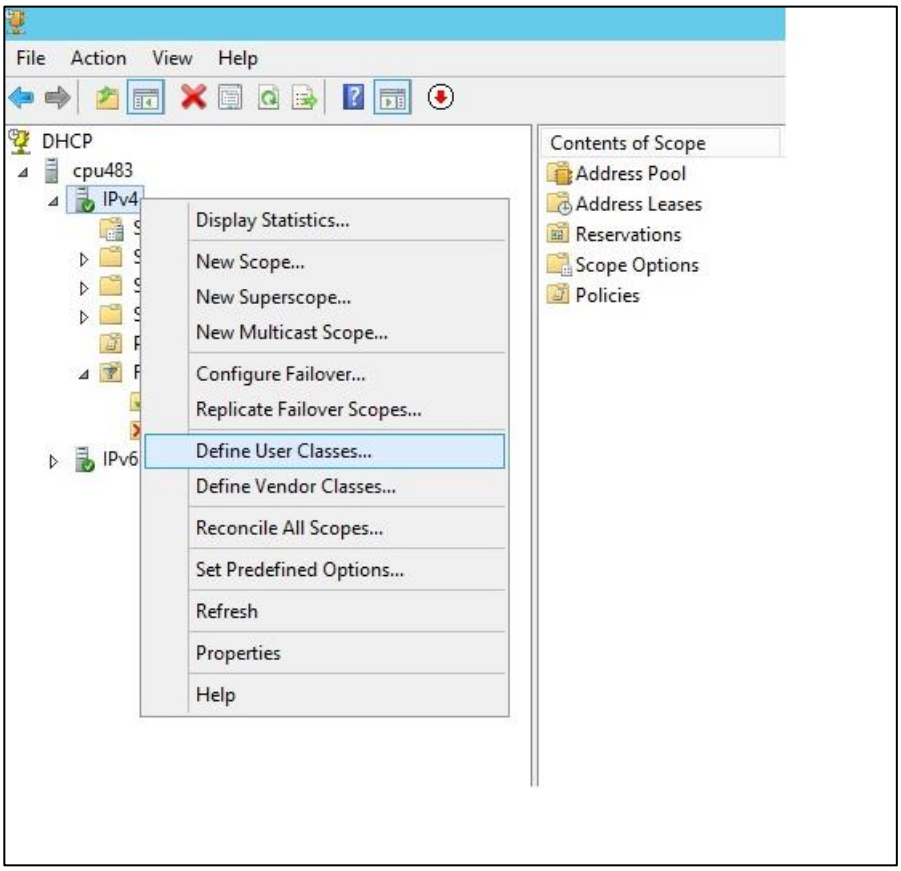
The "status" will show as "Active" once the Scope is activated.



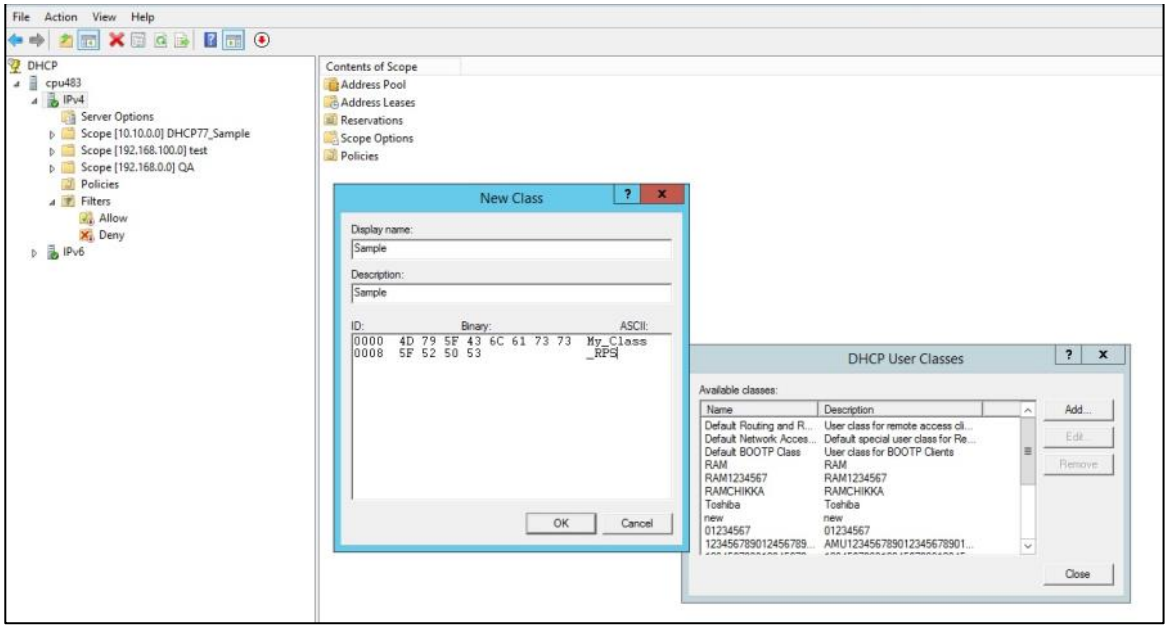
8. On the Left panel of DHCP server manager click on New scope created. It lists out Address pool, Address leases, reservations, Scope options and Policies. Click on Address Pool and observe whether the IP pool displayed in the right side of the panel is same as the IP pool assigned by the user in Step-5.



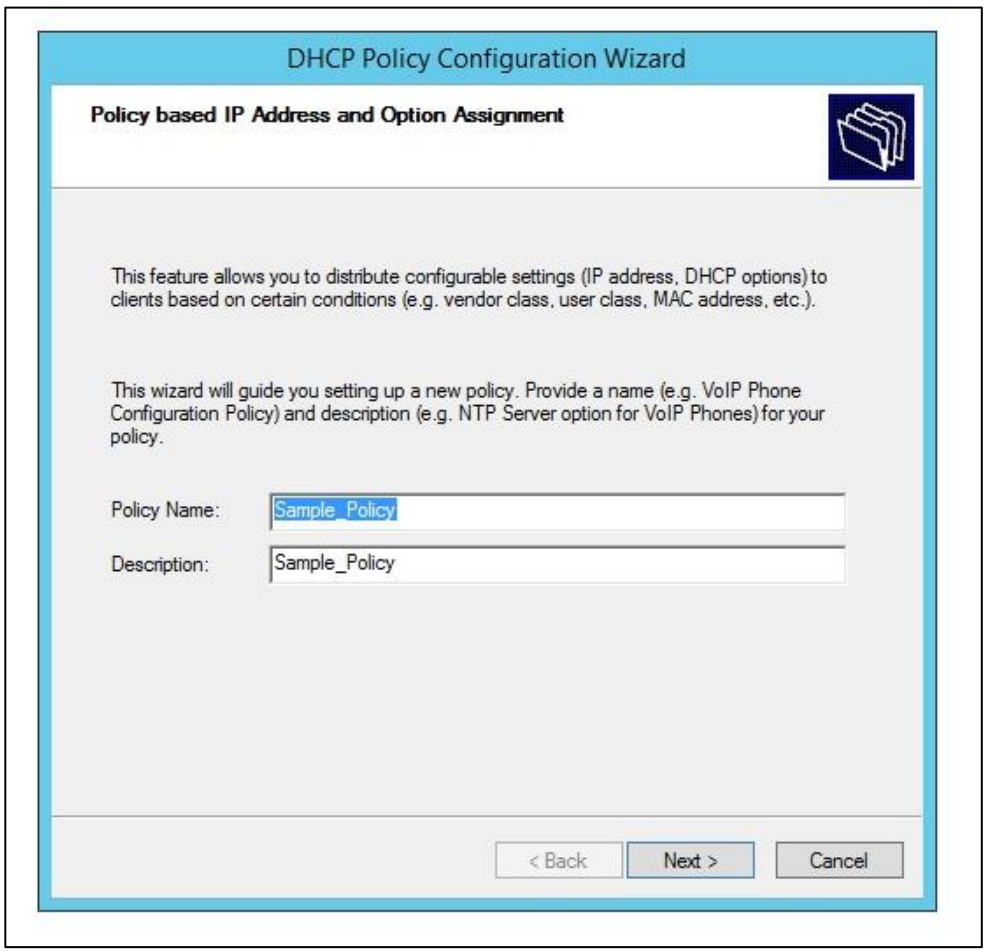
9. Click on IPv4 and right click, select "Define User classes".

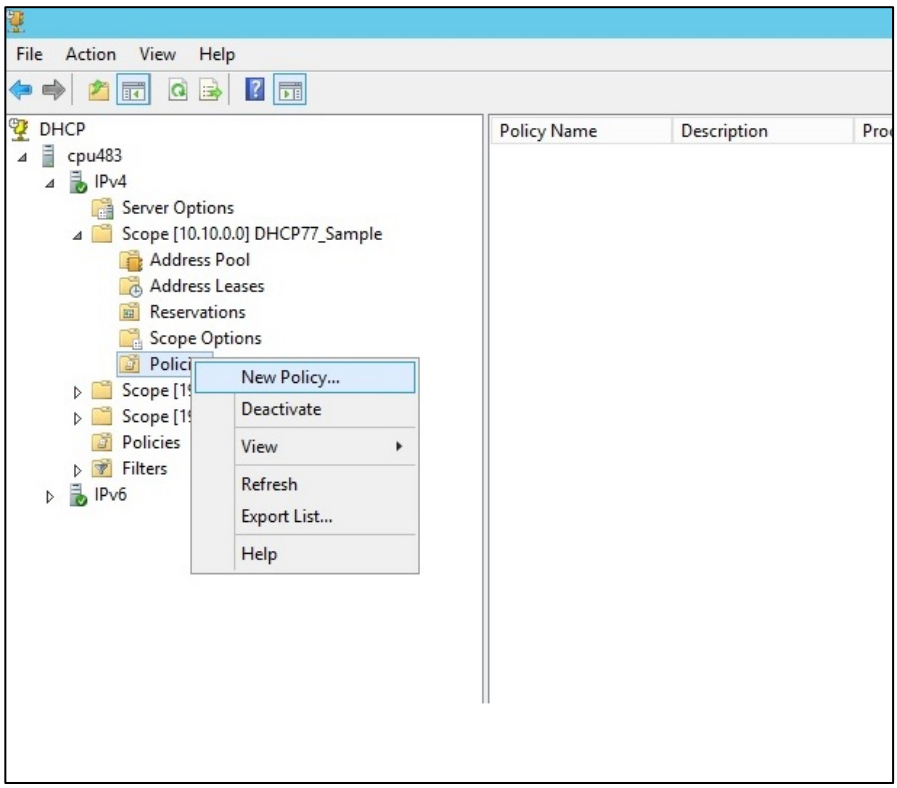


10. This opens a dialog box with name "DHCP User Classes". Click on "Add". Enter the Display name and Description as required in the new dialog box opened and click under space for ASCII. Name the user class as required (Pic below shows user class with name "My_Class_RPS").

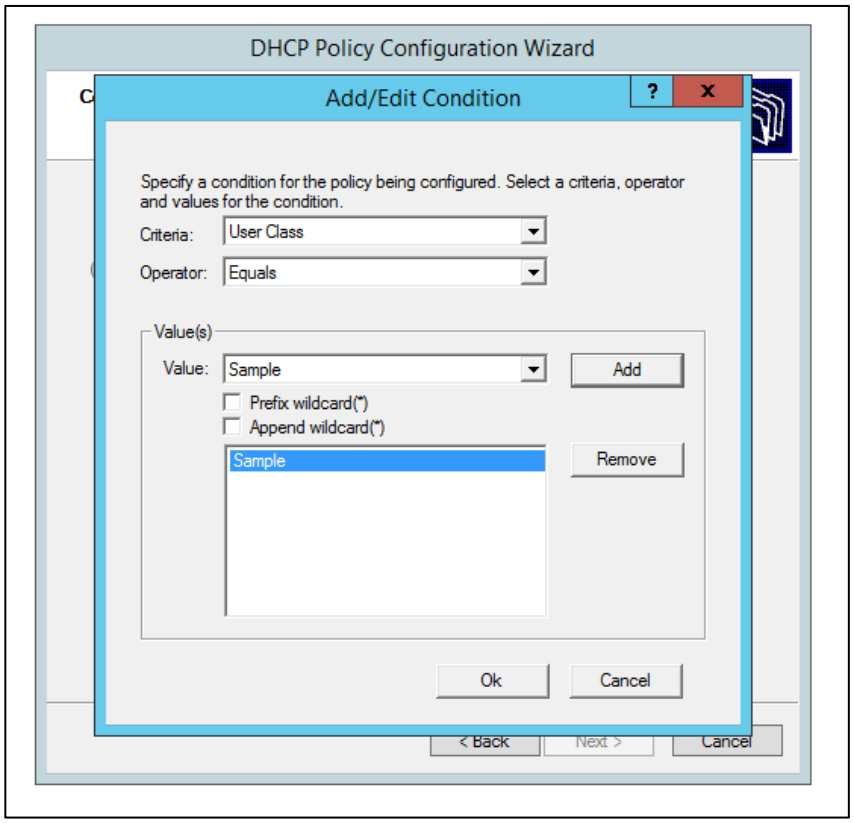


11. Click on "ok". This creates the DHCP class for User Option 77. The same name has to be given in the STA (client) in order to avail DHCP user class option 77.
12. Now click on the "New scope created" and right click on the "Policies", select "New Policy".
13. Name the policy as required. (Ex: Policy name in pic is "Sample_Policy")

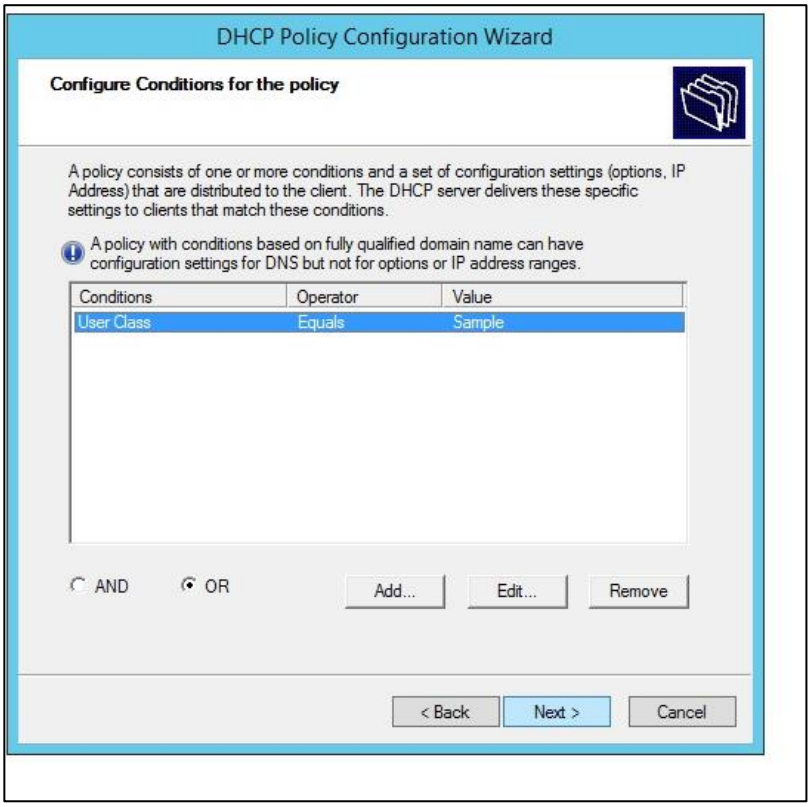




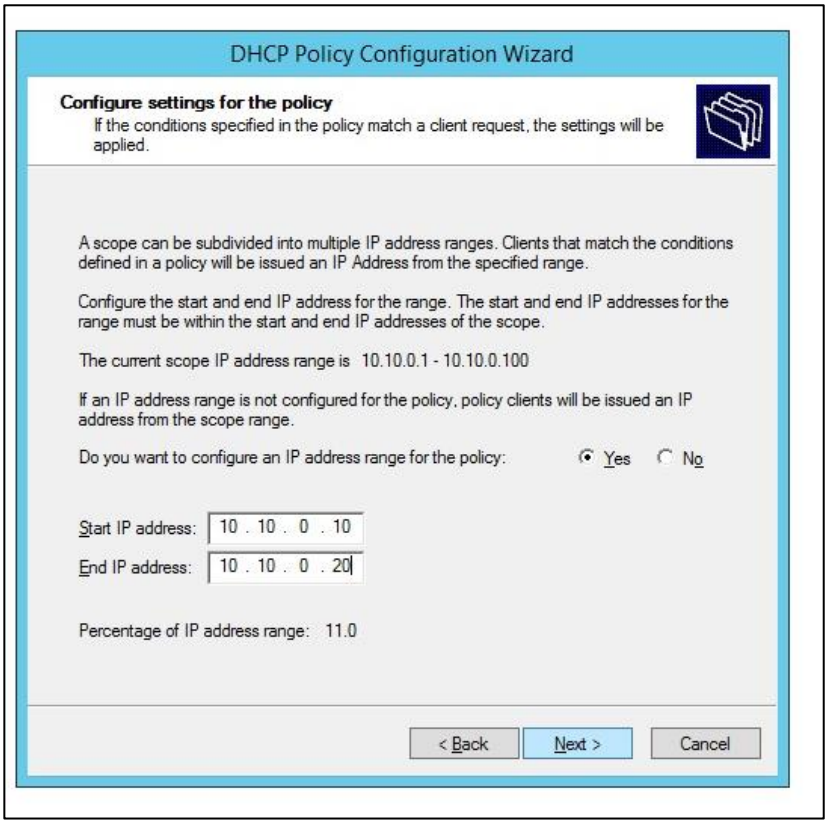
14. This opens dialog box. DHCP Policy Configuration Wizard. Click on "Add" in the dialog box opened. Select the "Criteria" as "User Class" and "Operator" as "Equals". Select value as the User-class created at step-8. (Ex: Sample is the User class created in the step-8 in the Pic)



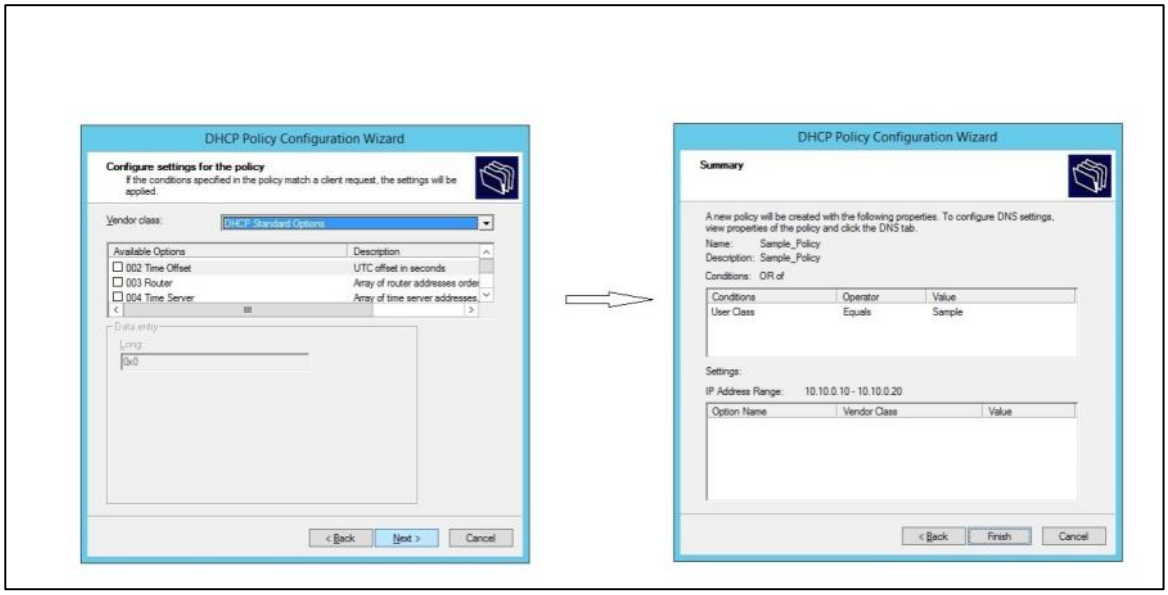
15. Click Next.



16. Select a range of IP addresses for the User class 77 and this pool must be a subset of Ip address range declared by user in Step-5.



17. Click Next to continue and finally click "Finish"



18. Finally, the policies are shown in the right panel with the User class option 77 pool declared by the user. (Ex: Pic shows a policy by name "Sample_policy" with user address pool 10.10.0.10-10.10.0.20)

Note:

The User policy created can be verified using a Windows STA. Independent of Server (whether Linux or Windows) the usage of DHCP 77 option command is same in the Windows STA.

Special Note:
Windows Server 2012 supports only Single user class in the DHCP request sent from a Client. It does not support "Multiple user Class".

6.8 EAP

Overview

This Application demonstrates how to configure device in Enterprise client and connects with Enterprise secured AP and data traffic in Enterprise security mode.
In this application, the device connects to Enterprise secured AP using EAP-TLS/TTLS/PEAP/FAST method. After successful connection, application establishes TCP client connection with TCP server opened on remote peer and sends TCP data on opened socket.

EAP overview

In wireless communications using EAP, a user requests connection to a WLAN through an AP, then requests the identity of the user and transmits that identity to an authentication server such as RADIUS. The server asks the AP for proof of identity, which AP gets from the user and then sends back to the server to complete the authentication.

PING overview

Ping is used diagnostically to ensure that a host computer that the user is trying to reach is actually operating. Ping works by sending an Internet Control Message Protocol (ICMP) Echo Request to a specific interface on the network and waiting for a reply. Ping can be used for troubleshooting, and to test connectivity and determine response time.

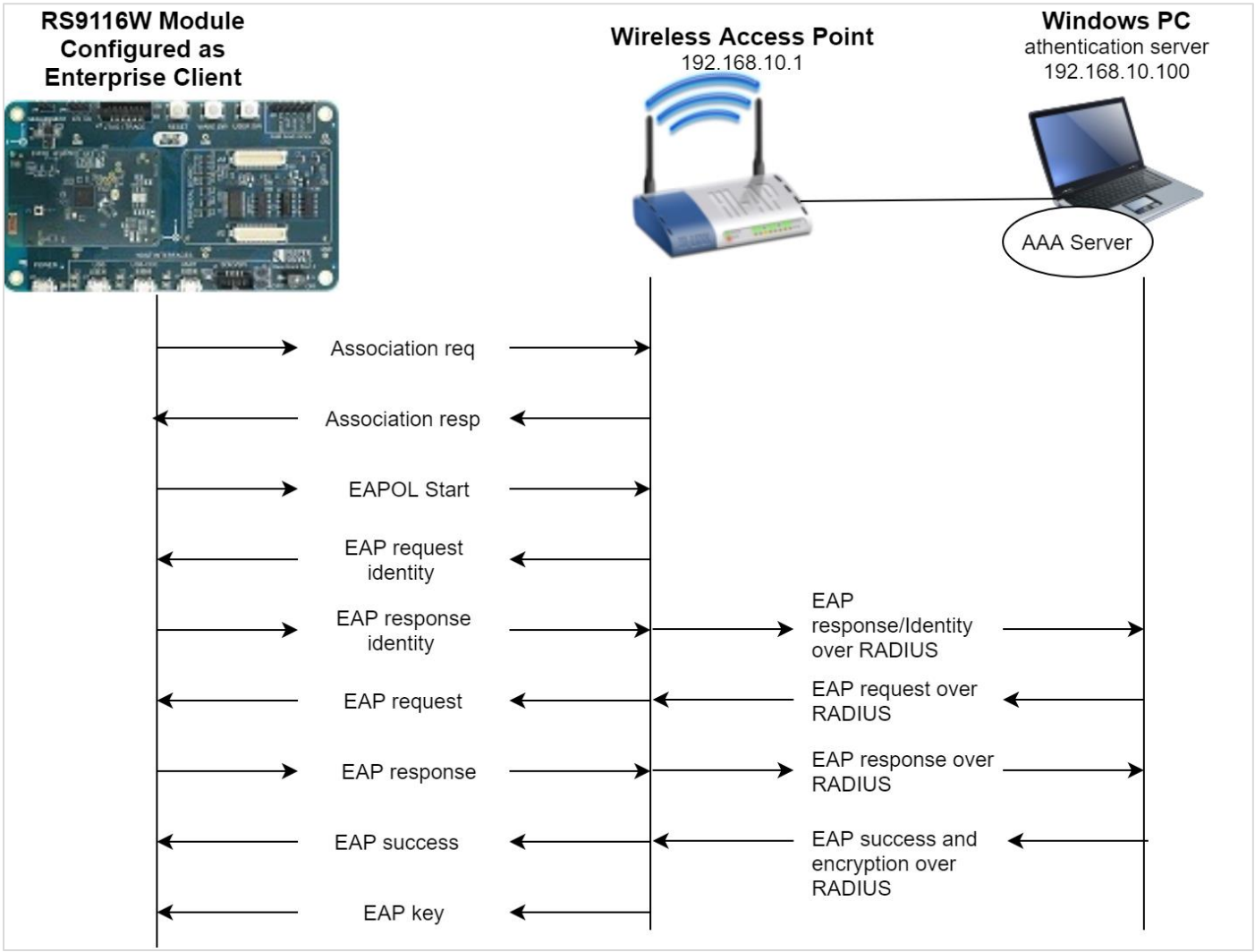


Figure 75: EAPOL-Keys Exchange

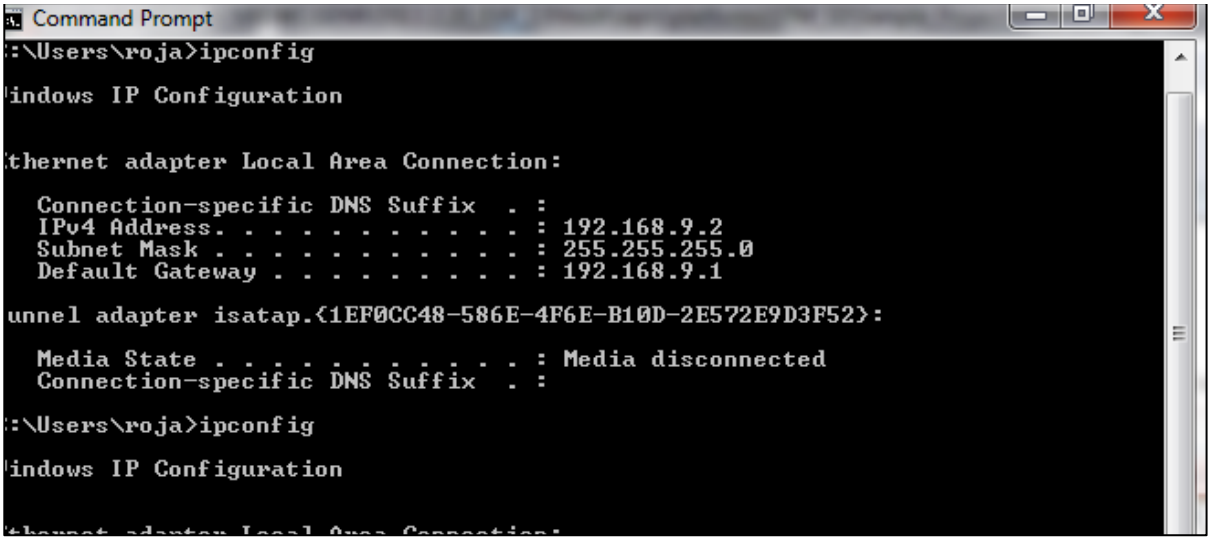
Note:

Ensure the LAN connection is removed from your PC. Remove any proxy server settings as well.

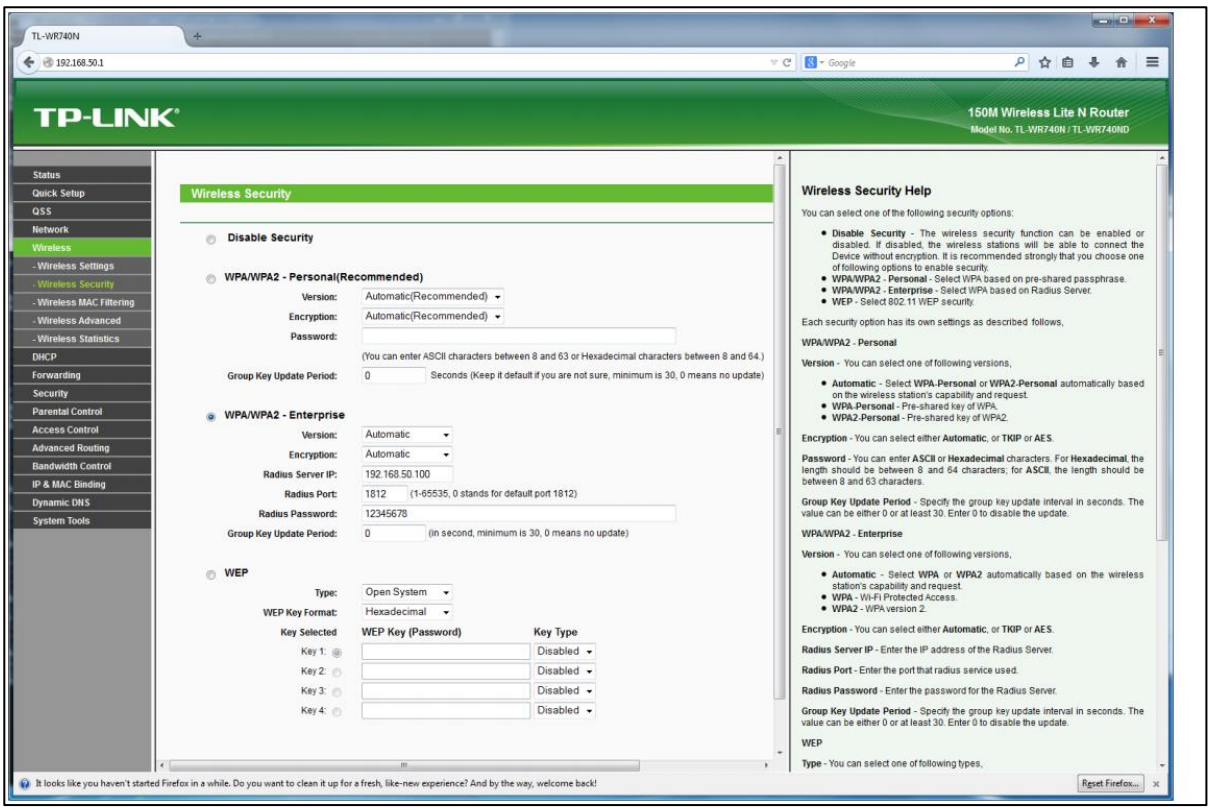
TP-link setup:

When working with the EAP-Ping example, LAN cable is connected between the TP-LINK modem and CPU.

1. After the connection, using the command prompt give "ipconfig" command to know the IP and gateway address of the Radius server. The below image is for reference purpose.



2. Connect the Access Point to PC over Ethernet and open the Access Point page in browser by typing the IP address of the AP's Default Gateway address and configure it.
3. Navigate to the Wireless Security section and enable the “WPA/WPA2 – Enterprise” option, as shown in the figure below. The image below is for a TP-Link Access Point.



4. Enter the IP address of the Radius Server in the field labeled, “Radius Server IP”. In the above figure, it is 192.168.50.100.
5. Enter the Radius Password as “12345678”. This is the same as that entered in the 'clients.conf' file of the Radius Server.

Radius server setup:

Description:

The figure below shows the setup for Wi-Fi Client in Enterprise Security Mode.

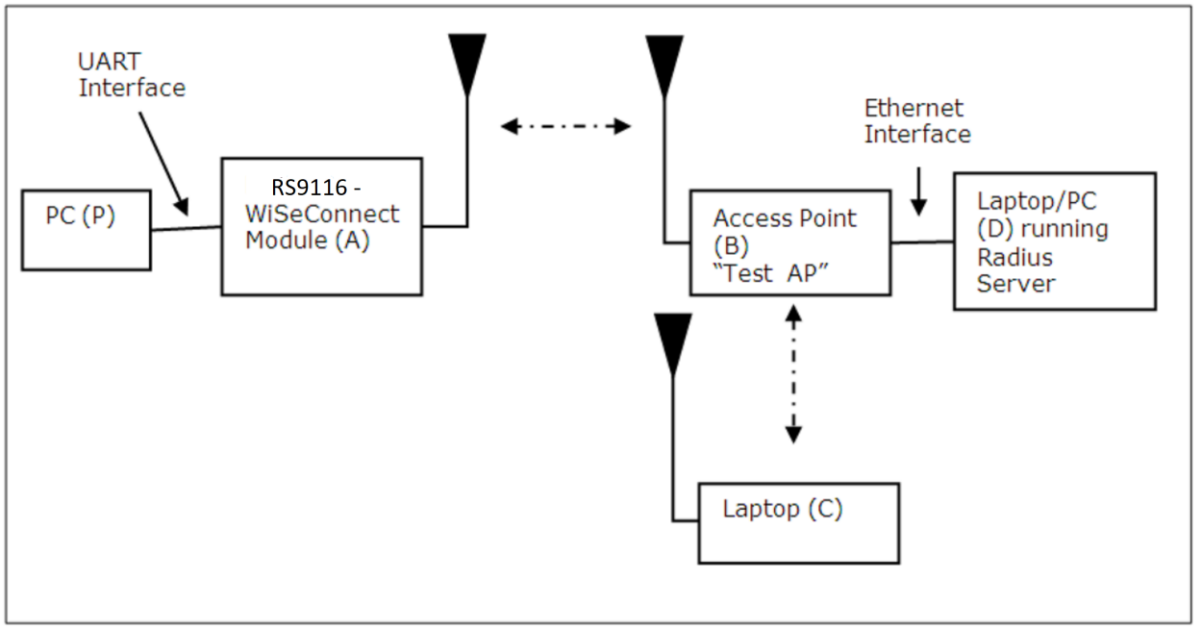


Figure 76: Setup for Wi-Fi Client in Enterprise Security Mode

Radius server Set-up guide:
The WiSeConnect module supports four Enterprise Security modes:

1. EAP-TLS
2. EAP-TTLS
3. EAP-PEAP
4. EAP-FAST

Radius Server Configuration

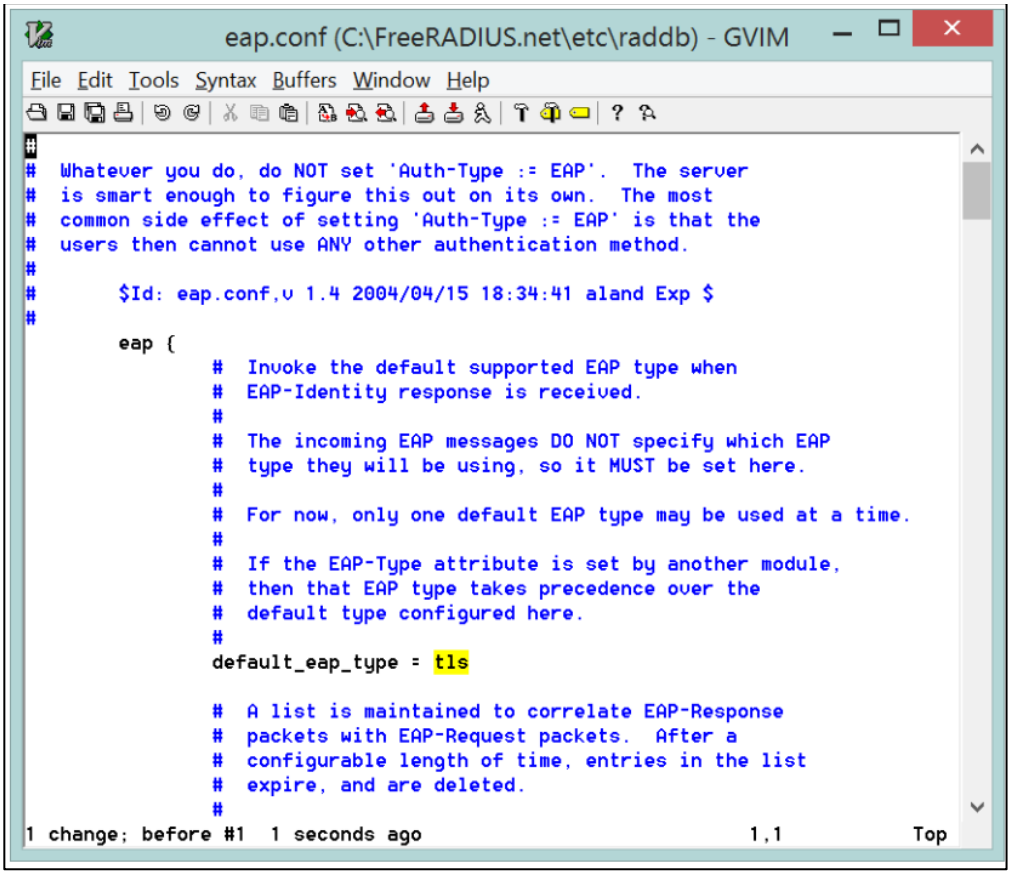
The configuration explained below is for Windows OS, similar process may be followed for other OS.

1. Free Radius Server installation link:
   https://freeradius.org/
   http://xperiencetech.com/download/radius-free-download.asp

> **Note:**
> Application was tested in FreeRADIUS-server-2.2.3-x86.

2. Once installed, go to the **C:\FreeRADIUS\etc\raddb** folder and make the following modifications.
3. Open the 'clients.conf' file and add the following lines at the end of the file.

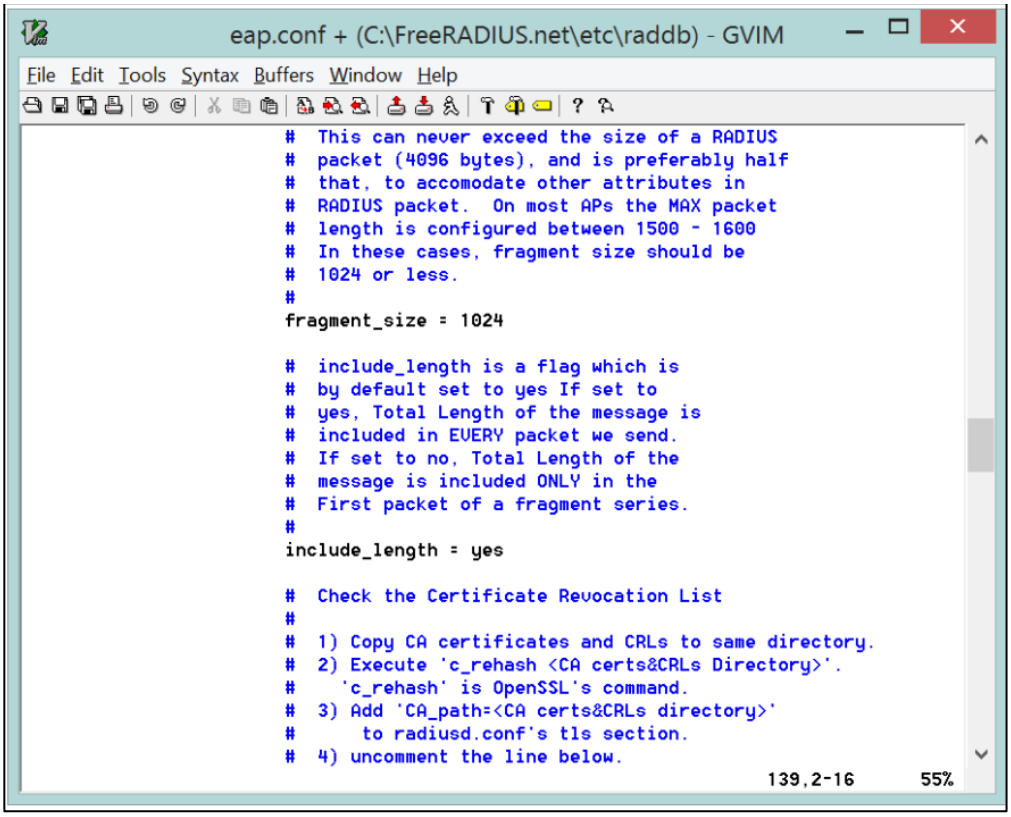
   client 192.168.50.1/24 {
   secret = 12345678
   shortname = private-network-1
   }
4. The IP address in the above lines (192.168.50.1) is the IP address of the Access Point in this example setup. The "12345678" input is the key to be entered in the Access Point's radius server configuration page to authenticate it with the Radius Server.
5. Open the 'eap.conf' file and make the following changes:
   a. Change the input for the "default_eap_type" field under the "eap" section to "tls", as shown in the figure below.



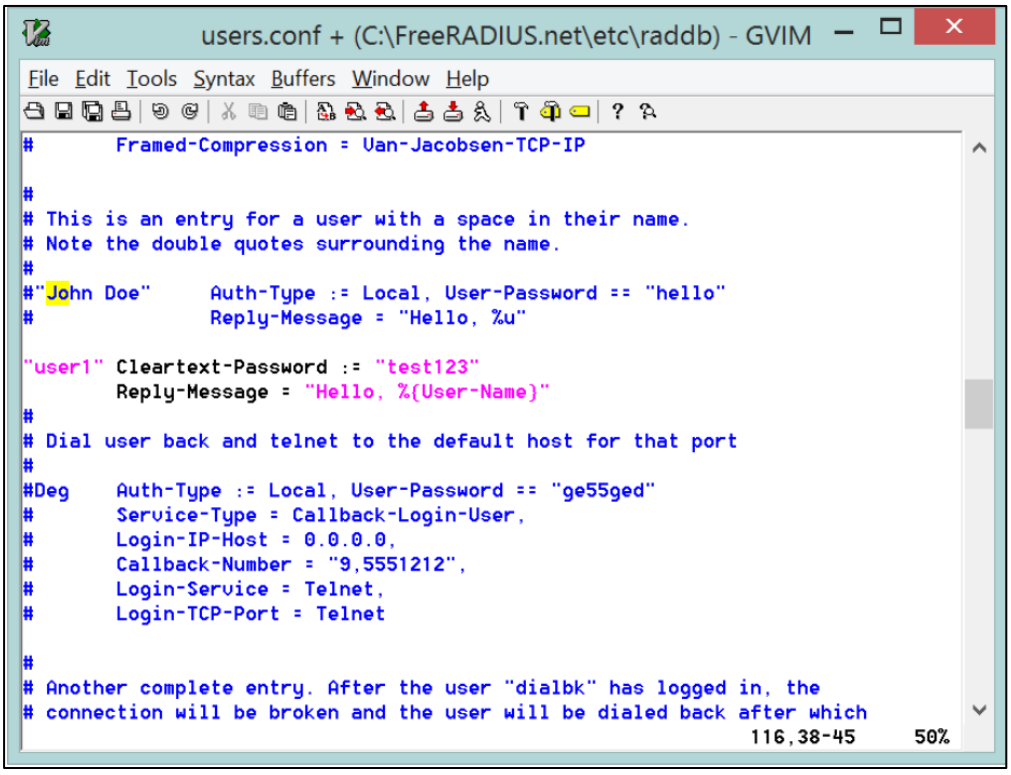
b. Change the inputs for “private_key_file”, “certificate_file” and “CA_file” fields under the “tls” section to “${certdir}/wifi-user.pem”, as shown in the figure below.

```
#  ANYONE who has a certificate signed by them can
#  authenticate via EAP-TLS!  This is likely not what you want.
tls {
        #
        #  These is used to simplify later configurations.
        #
        certdir = ${db_dir}/certs
        cadir = ${db_dir}/certs

        private_key_password = wifi
        private_key_file = ${certdir}/wifi-user.pem

        #private_key_file = ${certdir}/server-key.pem

        #  If Private key & Certificate are located in
        #  the same file, then private_key_file &
        #  certificate_file must contain the same file
        #  name.
        #
        #  If CA_file (below) is not used, then the
        #  certificate_file below MUST include not
        #  only the server certificate, but ALSO all
        #  of the CA certificates used to sign the
        #  server certificate.
        certificate_file = ${certdir}/wifi-user.pem

        #certificate_file = ${certdir}/server.pem

        #  Trusted Root CA list
        #
        #  ALL of the CA's in this list will be trusted
        #  to issue client certificates for authentication.
        #
        #  In general, you should use self-signed
        #  certificates for 802.1x (EAP) authentication.
        #  In that case, this CA file should contain
        #  *one* CA certificate.
        #
        #  This parameter is used only for EAP-TLS,
        #  when you issue client certificates.  If you do
        #  not use client certificates, and you do not want
        #  to permit EAP-TLS authentication, then delete
        #  this configuration item.
        #CA_file = ${cadir}/RootCA.pem
        CA_file = ${cadir}/wifi-user.pem

        #
        #  For DH cipher suites to work, you have to
        #  run OpenSSL to create the DH file first:
        #
```

c. Uncomment the “fragment_size” and “include_length” lines under the “tls” section, as shown in the figure below.



6. Open the users file and add the lines shown in the figure below starting with “user1”. This adds a user with username “user1” and password “test123”.



7. Copy the 'wifi-user.pem’ file from RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\example\utilities\certificates folder to C:\FreeRADIUS\etc\raddb\certs folder.
8. Click on the windows key and just search for Start RADIUS Server and click on it.

9. Then Radius server has started successfully you will see a print at the end which says, “Ready to process requests".

Note:

The radius server has to run before the application is executed. You will observe some transactions when the module is trying to connect to the radius server. Restart the Radius server when you execute the application every time.

Sequence of Events

This Application explains user how to:

1. Configure device as an Enterprise client
2. Connect with Enterprise secured AP using EAP-TLS/TTLS/PEAP/FAST method
3. Run Radius server in Windows/Linux PC2 which is connected to AP through LAN by providing required certificate and credentials.
4. After the program gets executed, Silicon Labs device will get connected to access point which is in enterprise security having the configuration same as that of in the application and gets IP.
5. After a successful connection with the Access Point, the device starts sending ping requests to the given REMOTE_IP with configured PING_SIZE to check the availability of the target device.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

1. Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
2. Silicon Labs Module
3. Windows/Linux PC2 with AAA Radius Server or Free Radius server
4. TP-link router connected to Windows PC through a LAN cable

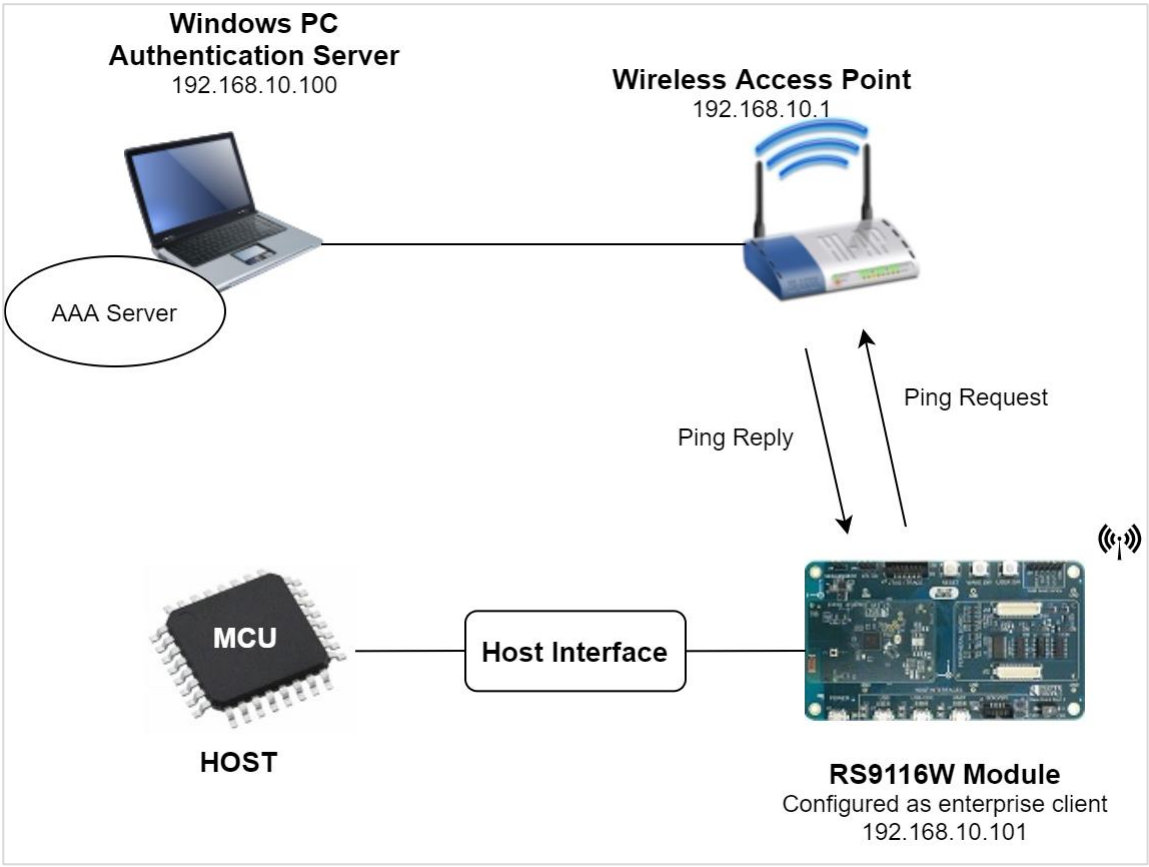


Figure 77: Setup Diagram for Enterprise Client Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_eap_connectivity.c*** file and update/modify following macros

   SSID refers to the name of the Access point.

```
#define SSID                                    "REDPINE_AP"
```

   SECURITY_TYPE refers to the type of security. In In this application STA supports WPA-EAP, WPA2-EAP securities.
   Valid configuration is:
   RSI_WPA_EAP - For WPA-EAP security mode
   RSI_WPA2_EAP - For WPA2-EAP security mode

```
#define SECURITY_TYPE                           RSI_WPA2_EAP
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     "123456789"
```

   To Load certificate
   LOAD_CERTIFICATE refers whether certificate to load into module or not.

```
#define LOAD_CERTIFICATE                        1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.
By default, application is loading "wifiuser.pem" certificate when **LOAD_CERTIFICATE** enabled. In order to load different certificate, user has to do the following steps:

a. **rsi_wlan_set_certificate** API expects the certificate in the form of linear array. So, convert the pem certificate into linear array form using python script provided in the release package **"utilities/certificates/certificate_script.py"**

Ex: If the certificate is wifi-user.pem. Give the command like the following way:
python certificate_script.py wifi-user.pem
Script will generate wifiuser.pem in which one linear array named wifiuser contains the certificate.

a. After conversion of certificate, update ***rsi_eap_connectivity.c*** source file by including the certificate file and by providing *the* required parameters to **rsi_wlan_set_certificate** API.

b. Once certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.

So, define **LOAD_CERTIFICATE** as 0, if certificate is already present in the device.
USER_IDENTITY refers to user ID which is configured in the user configuration file of the radius server. In this example, user identity is "user1".

```
#define USER_IDENTITY                                   "\"user1\""
```

PASSWORD refers to the password which is configured in the user configuration file of the Radius Server for that User Identity.
In this example, password is "test123"

```
#define PASSWORD                                        "\"test123\""
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                                     5001
```

SERVER_PORT port refers remote TCP server port number which is opened in Windows PC2.

```
#define SERVER_PORT                                     5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS                               0x640AA8C0
```

NUMEBR_OF_PACKETS refer to how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS                               1000
```

To configure IP address in STA mode
DHCP_MODE refers whether IP address configured through DHCP or STATIC in STA mode

```
#define DHCP_MODE                                   1
```

Note:

If the user wants to configure STA IP address through DHCP then skip configuring the **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If the user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.0.10" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0**.

```
#define DEVICE_IP                                   0X0A00A8C0
```

IP address of the gateway should also be in long format and in little endian byte order.
Example: To configure "192.168.0.1" as Gateway, update the macro GATEWAY as **0x0100A8C0**

```
#define GATEWAY                                     0x0100A8C0
```

IP address of the network mask should also be in long format and in little endian byte order.
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                     0x00FFFFFF
```

Configure the following macro to initiate ping with the remote peer IP address (AP IP address).
Example: To configure "192.168.10.1" as remote IP, update the macro **REMOTE_IP** as 0x6B01A8C0

```
#define REMOTE_IP                                   0x010AA8C0
```

Ping size refers to the size of the ping packet

```
#define PING_SIZE                                   100
```

2. Open **rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_PSK
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP                 EXT_FEAT_256k_MODE
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

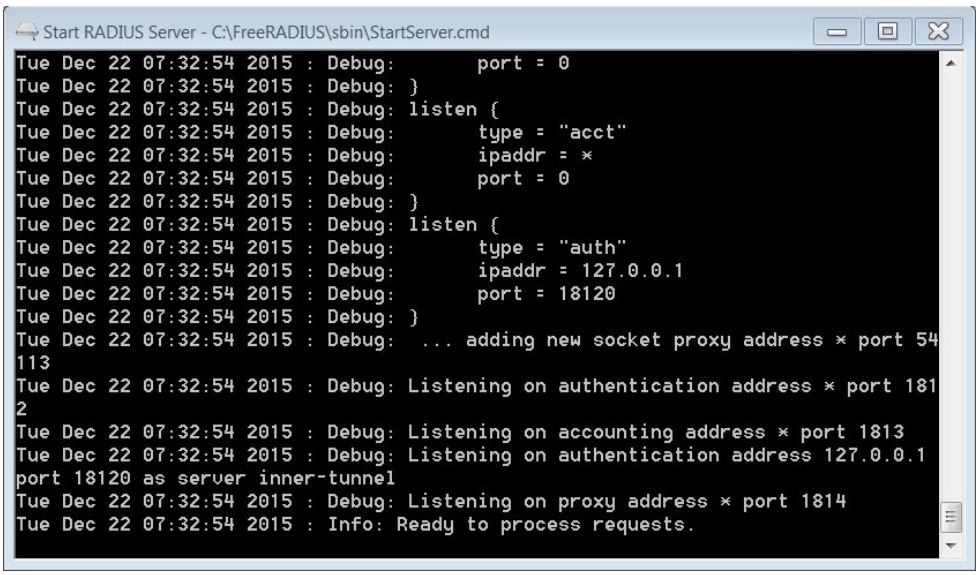
Note:

For TLS version selection, use '**rsi_wlan_common_config.h**' at 'RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\include' instead of '**rsi_wlan_config.h**' and enable respective bits as shown below.

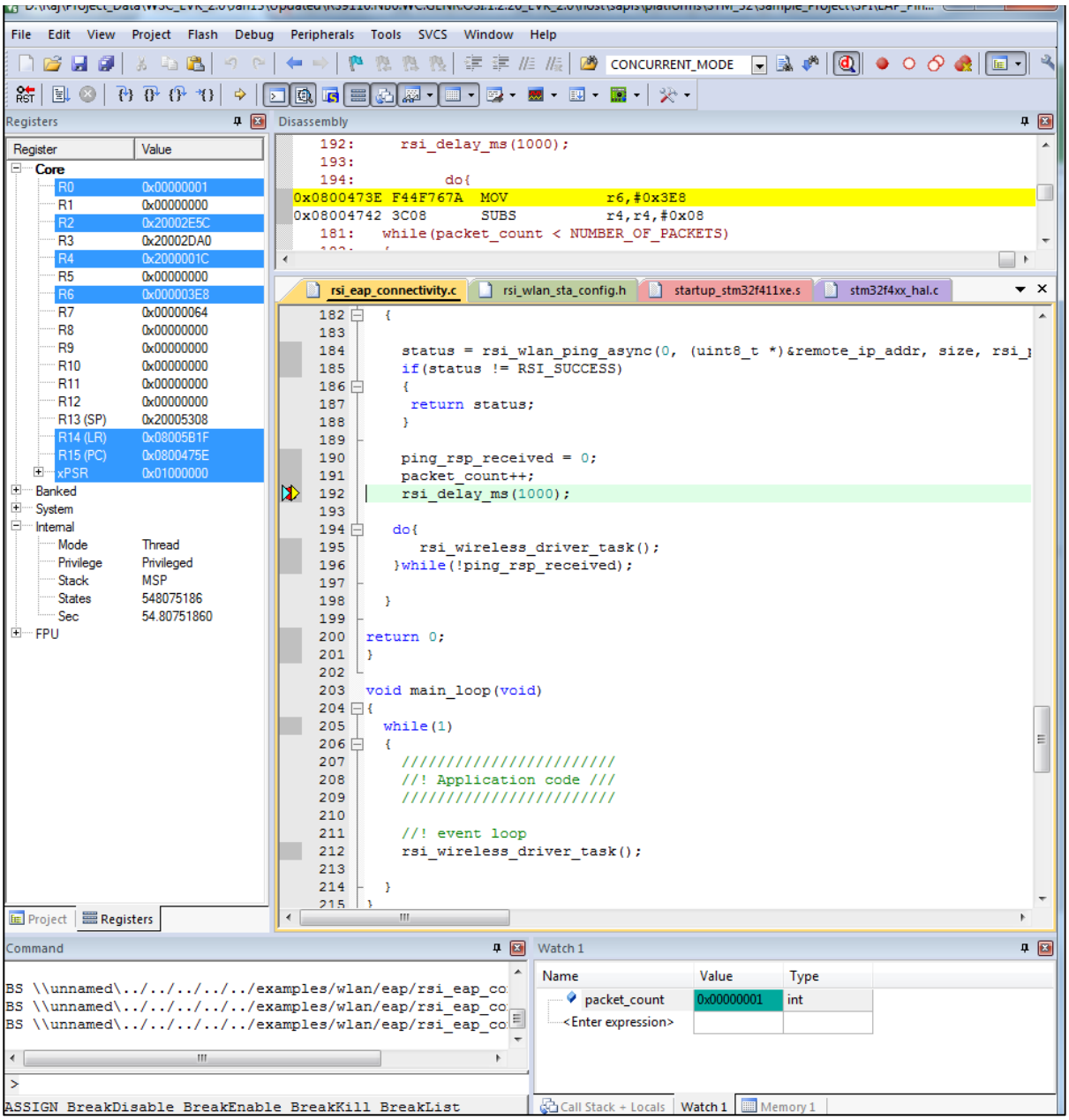
- To select TLS 1.0 version, enable **RSI_FEAT_EAP_TLS_V1P0** BIT(14) in **RSI_CONFIG_FEATURE_BITMAP**
- To select TLS 1.0 version, enable **RSI_FEAT_EAP_TLS_V1P2** BIT(15) in **RSI_CONFIG_FEATURE_BITMAP**

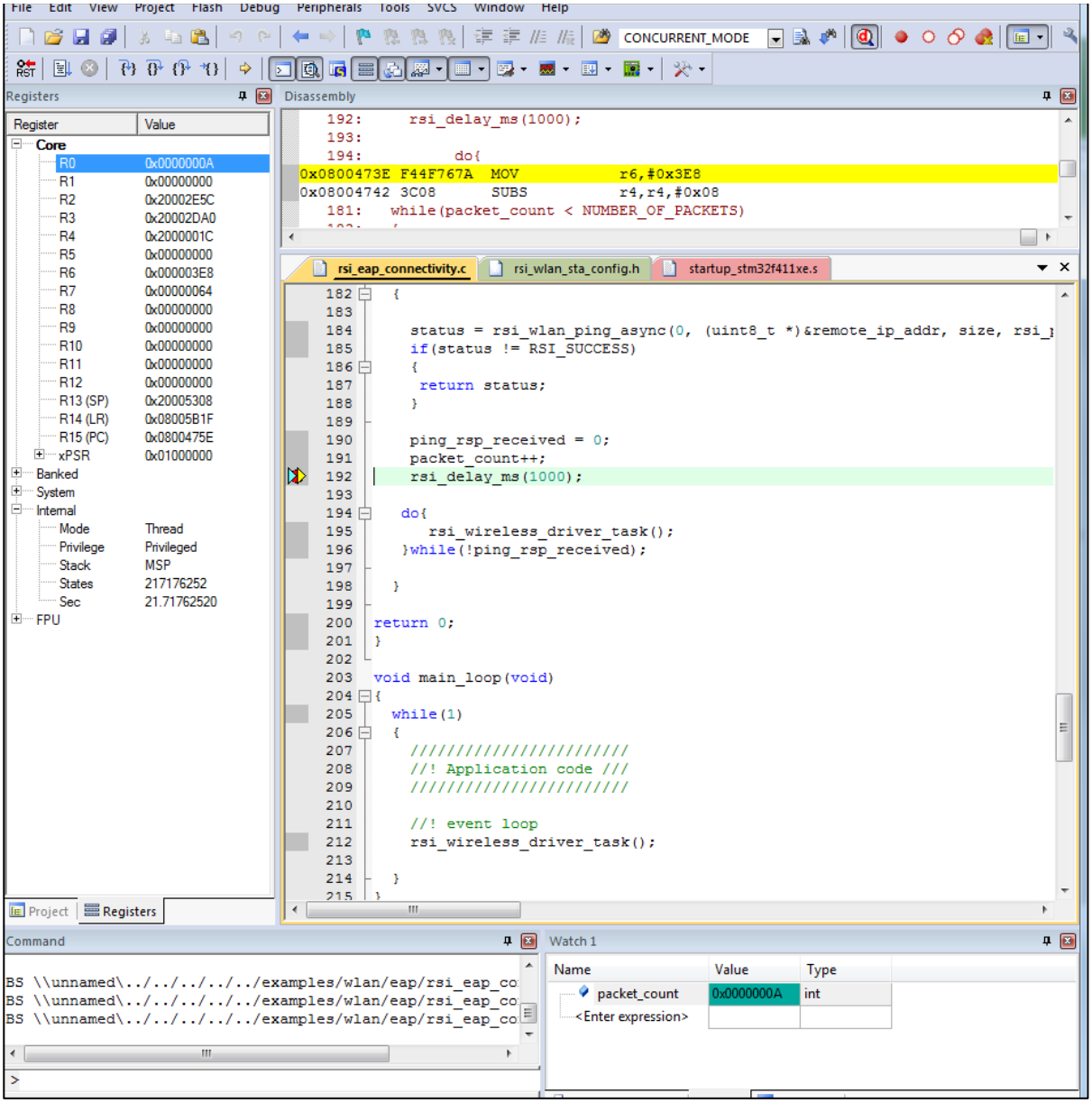
Executing the Application

1. Connect the WiSeConnect device (Silicon Labs module) to the Windows PC running Keil IDE.
2. Configure the Access point in WPA-EAP/WPA2-EAP mode to connect the Silicon Labs device in enterprise secured mode.
3. Run Radius server in Windows/Linux PC2 which is connected to AP through LAN by providing required certificate and credentials.



4. After the program gets executed, Silicon Labs device will get connected to access point which is in enterprise security having the configuration same as that of in the application and gets IP.
5. After a successful connection with the Access Point, the device starts sending ping requests to the given REMOTE_IP with configured PING_SIZE to check the availability of the target device.
6. The device sends the number of ping packets configured in NUMBER_OF_PACKETS.
7. In the rsi_eap_connectivity.c file, rsi_wlan_ping_async API returns success status, which means that the ping request packet is successfully sent into the medium. When the actual ping response comes from the remote node, it is known from the status parameter of the callback function (rsi_ping_response_handler) registered in the Ping API.
8. The following figures shows the Packet_count is continuously incremented, which means the ping request packet is successfully sent into the medium. Place a breakpoint at rsi_delay_ms (1000) and add the packet_count variable to watch the window and monitor the packet count.


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
CONCURRENT_MODE
Registers
Register Value
Core
R0 0x00000001
R1 0x00000000
R2 0x20002E5C
R3 0x20002DA0
R4 0x2000001C
R5 0x00000000
R6 0x000003E8
R7 0x00000064
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x20005308
R14 (LR) 0x08005B1F
R15 (PC) 0x0800475E
xPSR 0x01000000
Banked
System
Internal
Mode Thread
Privilege Privileged
Stack MSP
States 548075186
Sec 54.80751860
FPU
Disassembly
192: rsi_delay_ms(1000);
193:
194: do{
0x0800473E F44F767A MOV r6,#0x3E8
0x08004742 3C08 SUBS r4,r4,#0x08
181: while(packet_count < NUMBER_OF_PACKETS)
rsi_eap_connectivity.c
rsi_wlan_sta_config.h
startup_stm32f411xe.s
stm32f4xx_hal.c
182 {
183
184 status = rsi_wlan_ping_async(0, (uint8_t *)&remote_ip_addr, size, rsi_
185 if(status != RSI_SUCCESS)
186 {
187 return status;
188 }
189
190 ping_rsp_received = 0;
191 packet_count++;
192 rsi_delay_ms(1000);
193
194 do{
195 rsi_wireless_driver_task();
196 }while(!ping_rsp_received);
197
198 }
199
200 return 0;
201 }
202
203 void main_loop(void)
204 {
205 while(1)
206 {
207 ////////////////////////
208 //! Application code ///
209 ////////////////////////
210
211 //! event loop
212 rsi_wireless_driver_task();
213
214 }
215 }
Project Registers
Command
BS \\unnamed\../../../../../examples/wlan/eap/rsi_eap_co
BS \\unnamed\../../../../../examples/wlan/eap/rsi_eap_co
BS \\unnamed\../../../../../examples/wlan/eap/rsi_eap_co
ASSIGN BreakDisable BreakEnable BreakKill BreakList
Watch 1
Name Value Type
packet_count 0x00000001 int
<Enter expression>
Call Stack + Locals Watch 1 Memory 1



6.9 EMB MQTT

Protocol Overview

MQTT is a publish-subscribe based "light weight" messaging protocol for using on top of the TCP/IP protocol. The MQTT connection itself is always between one client and the broker, no client is connected to another client directly.

MQTT client

A MQTT client is any device from a micro controller to a full-fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. MQTT Clients can share the information on a particular topic using MQTT protocol. MQTT clients connect to the MQTT broker using TCP connection and can subscribe and publish on any desired topic. The other clients which are subscribed for that topic will receive the published messages.

MQTT Broker

The publish-subscribe messaging pattern requires a message broker. The broker is primarily responsible for receiving all messages, filtering them, deciding like who is interested in it and then sending the message to all subscribed clients.

It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. A simple demonstration of subscribing and publishing of temperature is shown below:

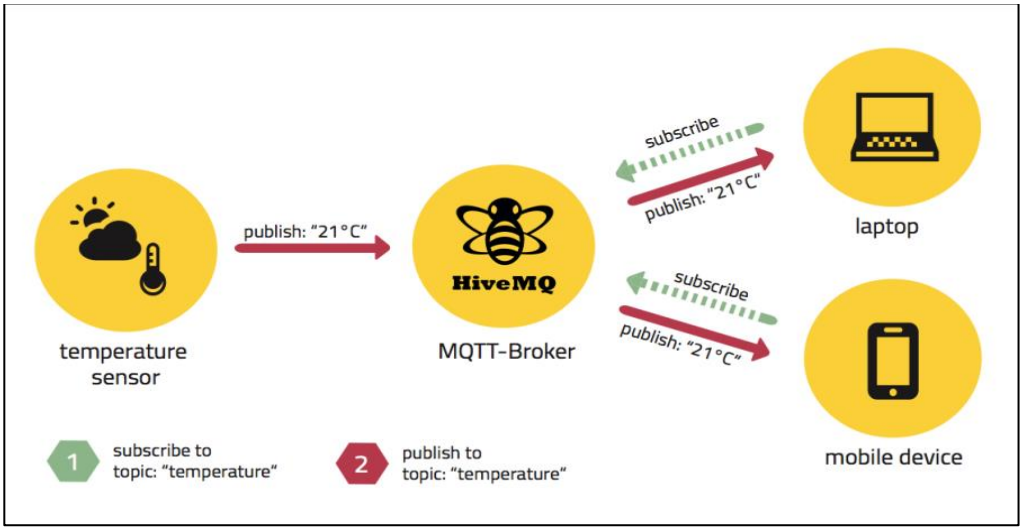


Figure 78: Demonstration of MQTT Protocol

Overview

This application demonstrates how to configure RS9116W device as MQTT client and to establish connection with MQTT broker and how to subscribe, publish and receive the MQTT messages from MQTT broker.
In this application, RS9116W device configured as WiFi station and connects to Access Point. After successful WiFi connection, application connects to MQTT broker and subscribes to the topic **"REDPINE_TEST"** and publishes a message **"THIS IS MQTT CLIENT DEMO FROM REDPINE"** on that subscribed topic. After publishing the message on the subscribed topic, the MQTT client unsubscribes and disconnects with the MQTT broker.

Sequence of Events

This Application explains user how to:

- Connect to Access Point
- Establish MQTT client connection with MQTT broker
- Subscribe the topic **"REDPINE_TEST"**
- Publish message **"THIS IS MQTT CLIENT DEMO FROM REDPINE"** on the subscribed topic **"REDPINE_TEST"**
- Receive data published on the same subscribed topic **"REDPINE_TEST"**.
- Unsubscribe to the MQTT broker
- Disconnect with the MQTT broker

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- RS9116W Module
- Windows PC2 with Keil and MQTT broker installed in it
- Windows PC3 with MQTT client utility installed in it

Note:

MQTT broker for different OS platforms can be downloaded from the below link
http://mosquitto.org/download/
Ex: Install "mosquitto-1.4.8-install-win32.exe"
MQTT Utility which has to be installed in Windows PC 3 can be downloaded from the below given link
https://www.eclipse.org/downloads/download.php?file=/paho/1.0/org.eclipse.paho.mqtt.utility-1.0.0.jar

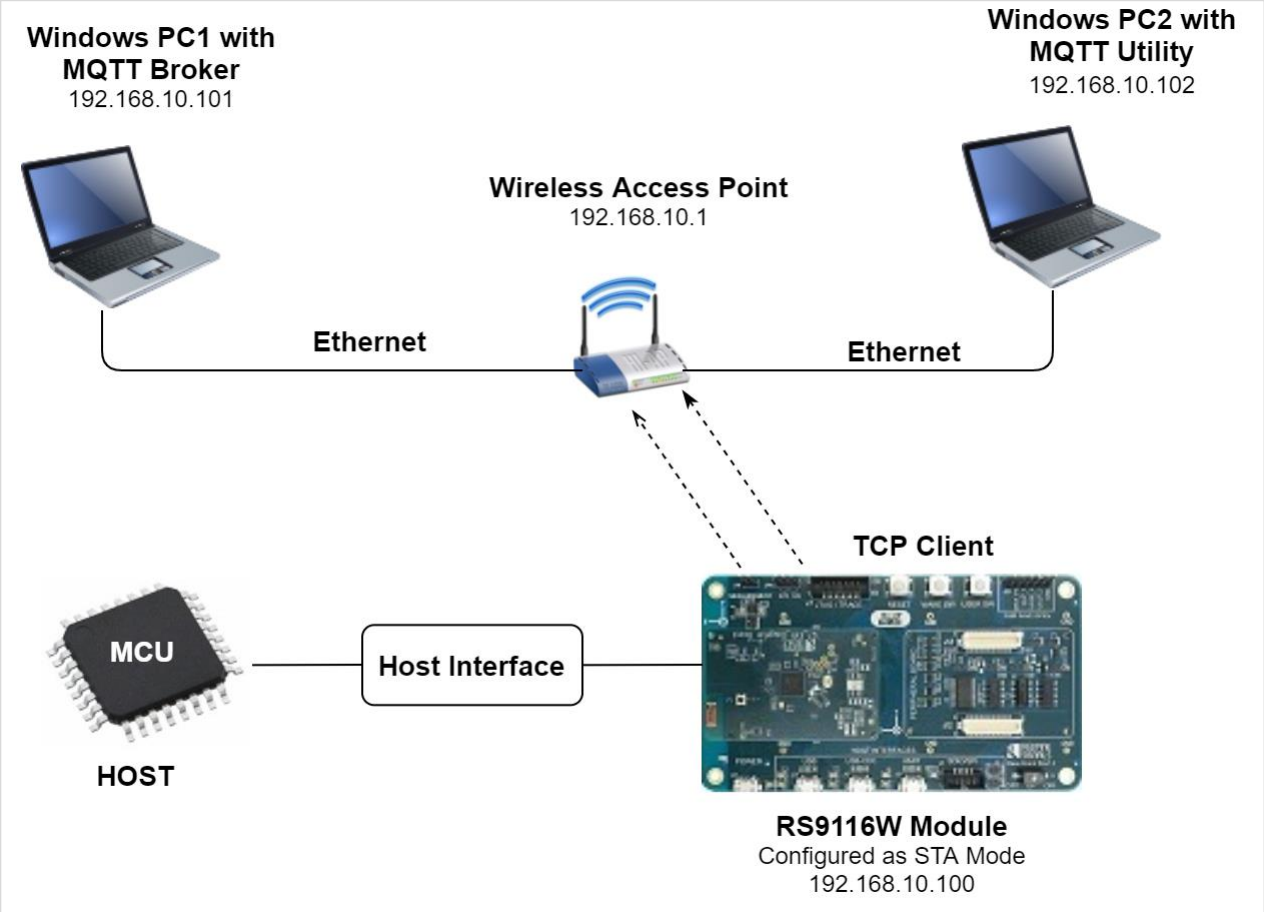


Figure 79: Setup Diagram for Emb_MQTT Client Example

Configuration and Steps for Execution

Configuring the Application

1. If user wants to use asynchronous MQTT, Open **rsi_emb_mqtt.c** file and update/modify following macros:

   SSID refers to the name of an Access point.

```
#define SSID                                  "<REDPINE_AP>"
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK and WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                         RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                   "<psk>"
```

   CLIENT_PORT port refers device MQTT client port number

```
#define CLIENT_PORT                           5001
```

SERVER_PORT port refers remote MQTT broker/server port number

```
#define SERVER_PORT                          1234
```

SERVER_IP_ADDRESS refers remote peer IP address (Windows PC2) to connect with MQTT broker/server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS                    0x6400A8C0
```

MQTT client keep alive period

```
#define RSI_KEEP_ALIVE_PERIOD                0
```

QOS indicates the level of assurance for delivery of an Application Message.
QoS levels are :
0 - At most once delivery
1 - At least once delivery
2 - Exactly once delivery

```
#define QOS                                  0
```

RSI_MQTT_TOPIC refers to which topic WiSeConnect MQTT client is supposed to subscribe.

```
#define RSI_MQTT_TOPIC                       "REDPINE_TEST"
```

MQTT Message to publish on the topic subscribed

uint8_t publish_message[] ="THIS IS MQTT CLIENT DEMO FROM REDPINE"
MQTT Client ID with which MQTT client connects to MQTT broker/server
uint8_t clientID[] = "MQTTCLIENT"
User name for login credentials
int8_t username[] = "username"
Password for login credentials
int8_t password[] = "password"

NUMEBR_OF_PACKETS refers how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS                    <no of packets>
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                      15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                            1
```

Note:

If the user wants to configure STA IP address through DHCP then skip configuring the **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If the user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order. Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                               0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order. Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order. Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

The following parameters are configured if OS is used.WLAN task priority is given and this should be of low priority

```
#define RSI_WLAN_TASK_PRIORITY                  1
```

Driver task priority is given and this should be of highest priority

```
#define RSI_DRIVER_TASK_PRIORITY                1
```

WLAN Task stack size is configured by this macro

```
#define RSI_WLAN_TASK_STACK_SIZE                500
```

Driver Task stack size is configured by this macro

```
#define RSI_DRIVER_TASK_STACK_SIZE              500
```

Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP
(TCP_IP_FEAT_DHCPV4_CLIENT|TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP              EXT_FEAT_CUSTOM_FEAT_EXTENTION_VALID
```

```
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_256k_MODE
#define RSI_EXT_TCPIP_FEATURE_BITMAP            EXT_EMB_MQTT_ENABLE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

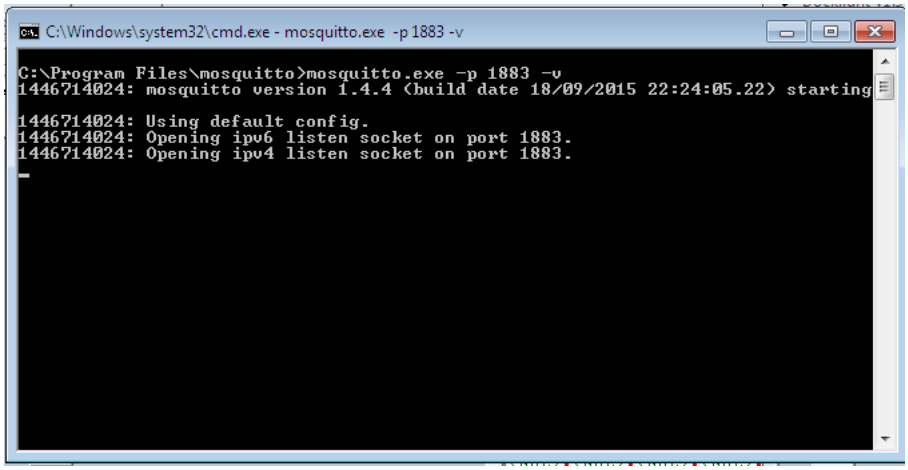
Note:

- **rsi_wlan_config.h** file is already set with desired configuration in respective example folders, user need not change for each example.
- In **rsi_mqtt_client.h** change **MQTT_VERSION** macro to either 3 or 4 based on the MQTT broker support version. (Supported versions 3 and 4).

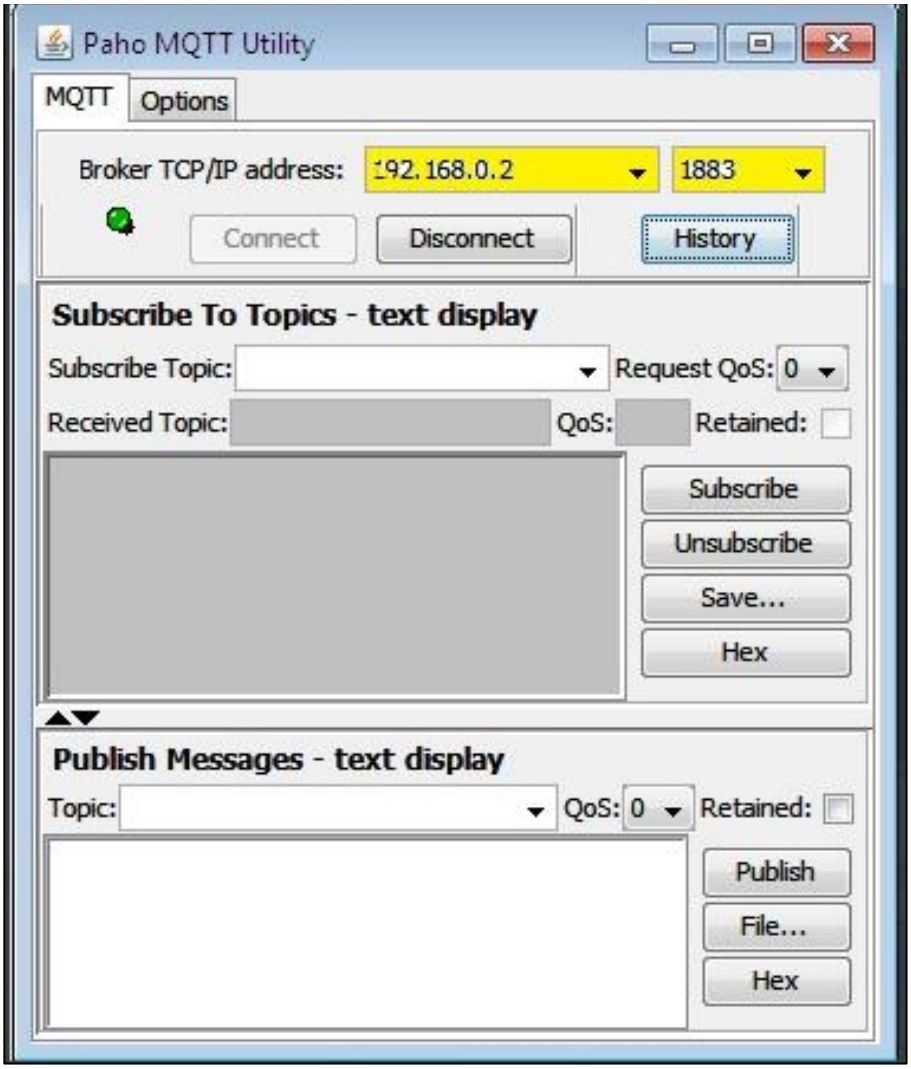
Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect RS9116W device in STA mode.
2. Install MQTT broker in Windows PC2 which is connected to Access Point through LAN.
3. Run MQTT broker in Windows PC2 using following command. Open Command prompt and go to MQTT installed folder (Ex: C:\Program Files\mosquitto) and run the following command**:**

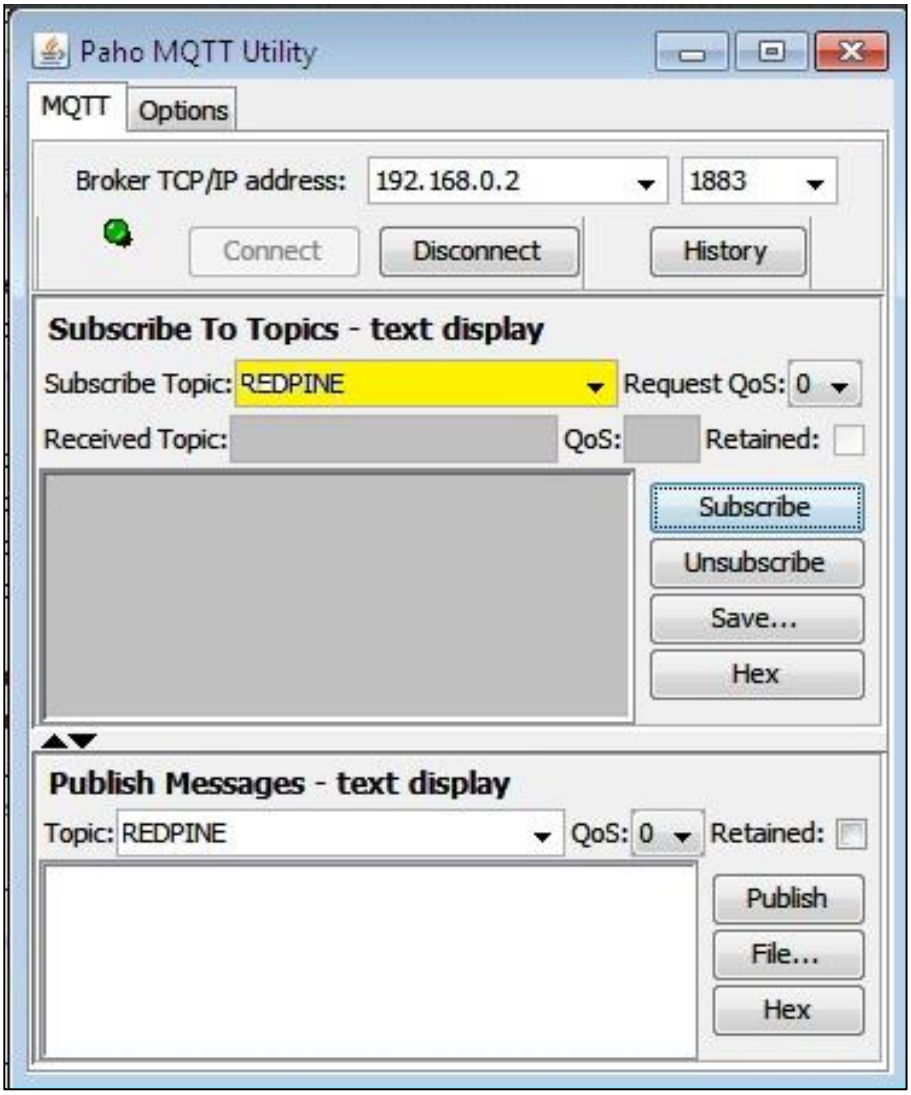
mosquito.exe –p 1883 –v



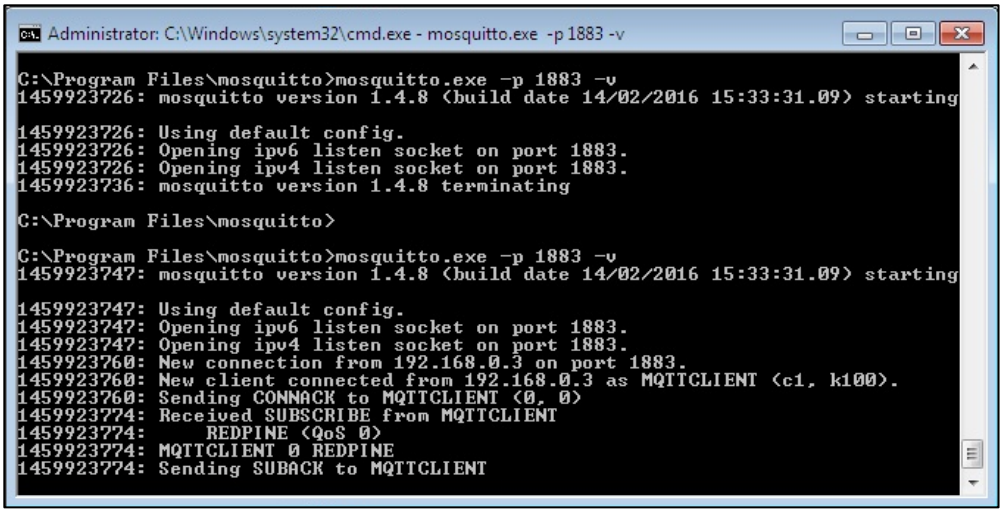
4. Open MQTT client utility in Windows PC3 and connect to MQTT broker by giving Windows PC2 IP address and MQTT broker port number in Broker TCP/IP address field.



5. After successful connection, subscribe to the topic from MQTT client utility.

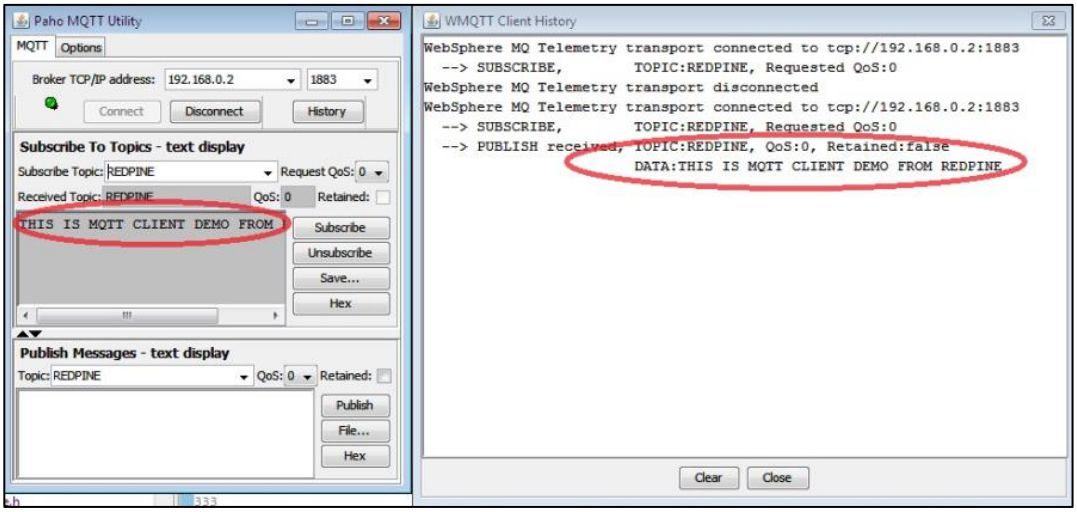


6. After the program gets executed, device will get connected to the same access point having the configuration same as that of in the application and get IP.
7. Once the device gets connected to the MQTT broker, it will subscribe to the topic **RSI_MQTT_TOPIC (Ex: "REDPINE_TEST").** The user can see the client connected and subscribe information in the MQTT broker.

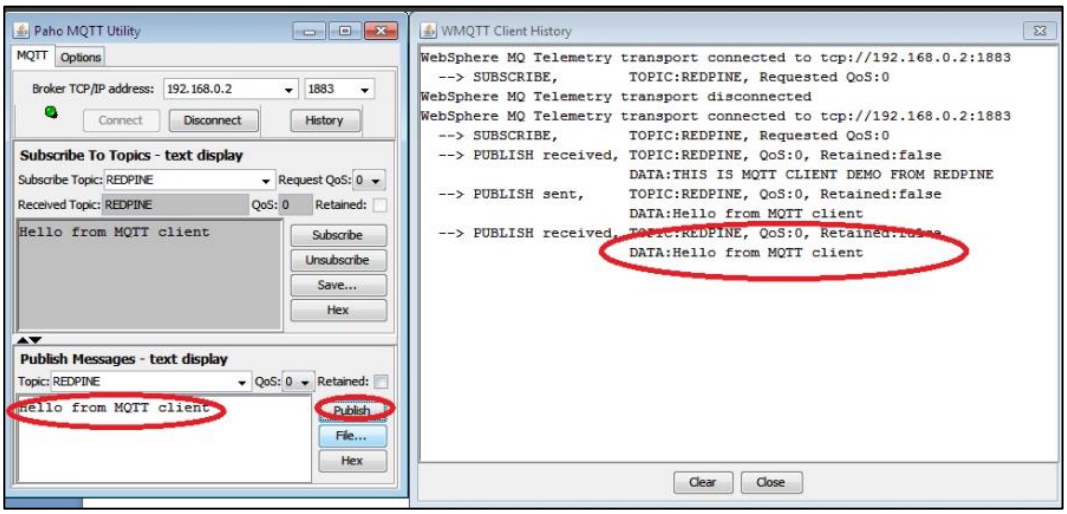


8. After successful subscription to the topic **RSI_MQTT_TOPIC (Ex: "REDPINE"),** the device publishes a message which is given in **publish_message** array (Ex: "THIS IS MQTT CLIENT DEMO FROM REDPINE") on the subscribed topic.
9. MQTT client utility which is running on Windows PC3 will receive the message published by the device as it

subscribes to the same topic.
Please refer to the below image for MQTT client utility and message history.



10. Now publish a message using MQTT Utility on the same topic. Now this message is the message received by the device.



Note:
Multiple MQTT client instances can be created

6.10 Ethernet WIFI Bridge

Overview

The ethernet_wifi_bridge example demonstrates how to configure the Silicon Labs device as a soft Access Point and allows stations to connect to it. The example also enables the M4 Ethernet connectivity with TA WiFi and TCP data transmission from the connected ethernet station to Wi-Fi station through Silicon Labs Access Point.

Sequence of Events

This example explains user how to:

- Create device as 'Soft Access point'
- Open TCP server socket on configured port number on the device
- Connect Wi-Fi Station to device Access point
- Connect ethernet station to device access point

- Establish TCP connection from connected ethernet station to Wi-Fi Station.
- Send TCP data from Connected ethernet station to Wi-Fi Station.

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. The Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors. The WiSeconnect parts offer integrated wireless connectivity and does not require host interface initialization.

WiSeConnect based Setup Requirements

Windows PC with KEIL IDE in case of WiSeConnect

- RS9116W module with ethernet peripheral card.
- Ethernet peripheral card uses the GPIO_6,GPIO_7,GPIO_8,GPIO_9,GPIO_10,GPIO_11,GPIO_12,GPIO_15 and GPIO_16 for functionality, use the proper Silicon Labs module package which is supporting all this GPIO's and confirm the same are enabled in RTE_DEVICE.h.
- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- A Mobile device as a Wi-Fi station (This example uses a windows Laptop)

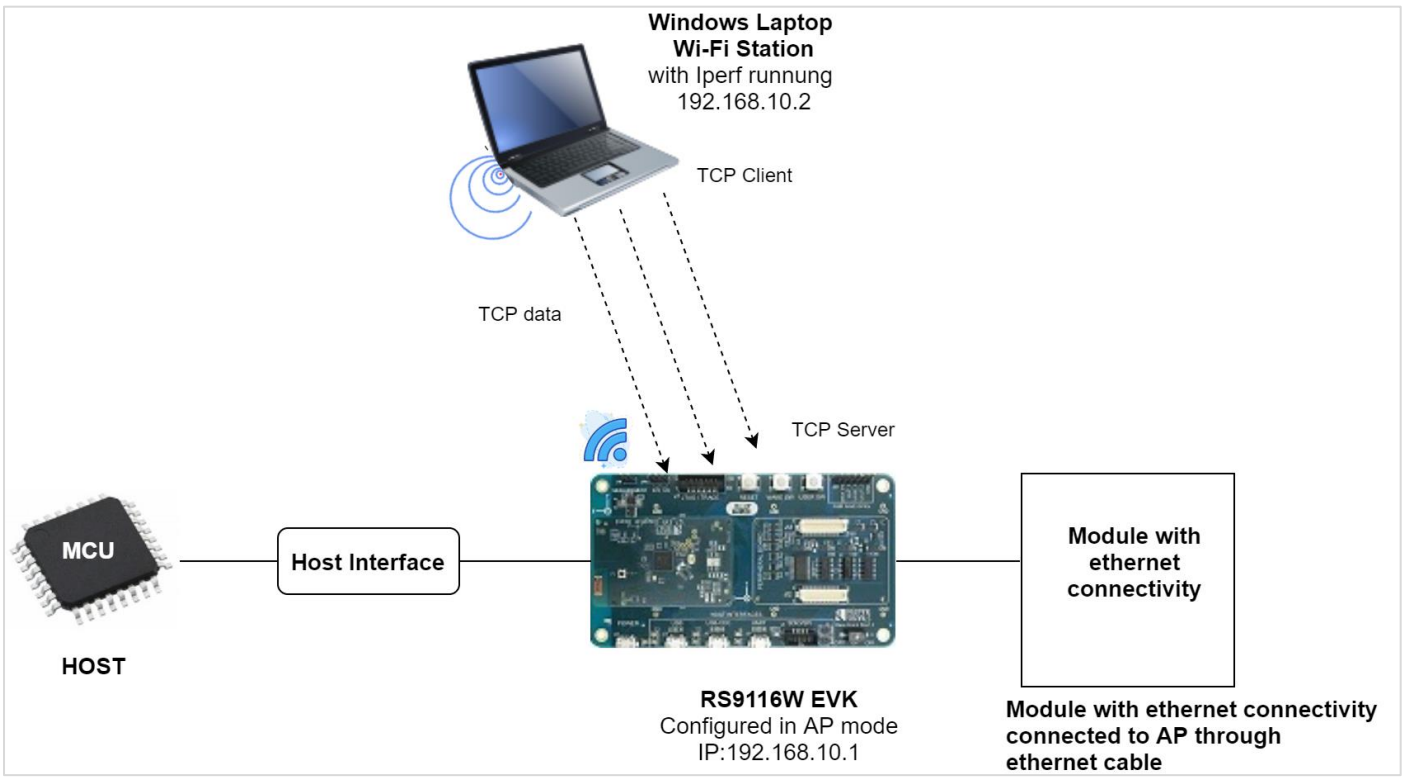


Figure 80: Setup Diagram for Ethernet Wi-Fi Bridge

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ethernet_wifi_bridge.c*** file and update / modify the following macros.

   SSID refers to the name of the Access point to be created.

```
#define SSID                                "REDPINE_AP"
```

   CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                      11
```

Note:

Valid values for CHANNEL_NO in 2.4GHz band are 1 to 11 and 5GHZ band are 36 to 48 and 149 to 165.
In this example default configured band is 2.4GHz.
So, if user wants to use 5GHz band then user has to set RSI_BAND macro to 5GHz band in (Example folder $) rsi_wlan_config.h file.

SECURITY_TYPE refers to the type of security. Access point supports Open, WPA, WPA2 securities.

Valid configurations are:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                   RSI_WPA2
```

ENCRYPTION_TYPE refers to the type of Encryption method .Access point supports OPEN, TKIP, CCMP encryption methods.

Valid configurations are:
RSI_CCMP - For CCMP encryption
RSI_TKIP - For TKIP encryption
RSI_NONE - For open encryption

```
#define ENCRYPTION_TYPE                 RSI_CCMP
```

PSK refers to the secret key if the Access point to be configured in WPA/WPA2 security modes.

```
#define PSK                             "12345678"
```

BEACON_INTERVAL refers to the time delay between two consecutive beacons in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

```
#define BEACON_INTERVAL                 100
```

DTIM_INTERVAL refers DTIM interval of the Access Point. Allowed values are from 1 to 255.

```
#define DTIM_INTERVAL                   4
```

DEVICE_PORT port refers TCP server port number

```
#define DEVICE_PORT                     5001
```

NUMEBR_OF_PACKETS refers how many packets to receive from remote TCP client.

```
#define NUMBER_OF_PACKETS               1000
```

RECV_BUFFER_SIZE refers receive data length

```
#defineRECV_BUFFER_SIZE                    1000
```

To configure IP address
IP address to be configured to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.1" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0**.

```
#define DEVICE_IP                          0X010AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro **GATEWAY** as **0x010AA8C0**

```
#define GATEWAY                            0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                            0x00FFFFFF
```

Note:
In AP mode, configure same IP address for both DEVICE_IP and GATEWAY macros

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                    RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                FEAT_SECURITY_PSK
#define RSI_TCP_IP_BYPASS                  RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP         (TCP_IP_FEAT_DHCPV4_SERVER | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_EXT_TCPIP_FEATURE_BITMAP       EXT_TCP_IP_ETH_WIFI_BRIDGE
#define RSI_CUSTOM_FEATURE_BIT_MAP         FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                           RSI_BAND_2P4GHZ
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders. User need not change for each example.

Note:
Make sure onboard UART port is not connected when executing this application.

Executing the Application

1. After the program gets executed, device will be created as an Access point with configured **SSID** (Ex: "**AP_SSID")** and acts as a bridge between the connected station and Ethernet

2. After successful connection, Ethernet sends the data over WIFI to the connected station

6.11 Firmware Upgrade

Overview

This application demonstrates how to upgrade new firmware to Silicon Labs device using remote TCP server. In this application, the device connects to an access point and establishes TCP client connection with TCP server opened on remote peer. After successful TCP connection, application sends the firmware file request to remote TCP server and server responds with firmware file and waits for the next firmware file request. Once firmware file receives from the TCP server, application loads the firmware file into device using firmware upgrade API and gets next firmware file from TCP server. After successful firmware upgrade, firmware upgrade API returns 0x03 response.

Sequence of Events

This Application explains user how to:

- Configure the device in station mode
- Open TCP server socket at Access Point
- Connect to Access Point and open TCP client socket
- Request Firmware file from remote server
- Send firmware file from remote server
- Upgrade the received Firmware into the device.

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- Wireless Access point
- Linux PC with TCP server application (TCP server application providing as part of release package)

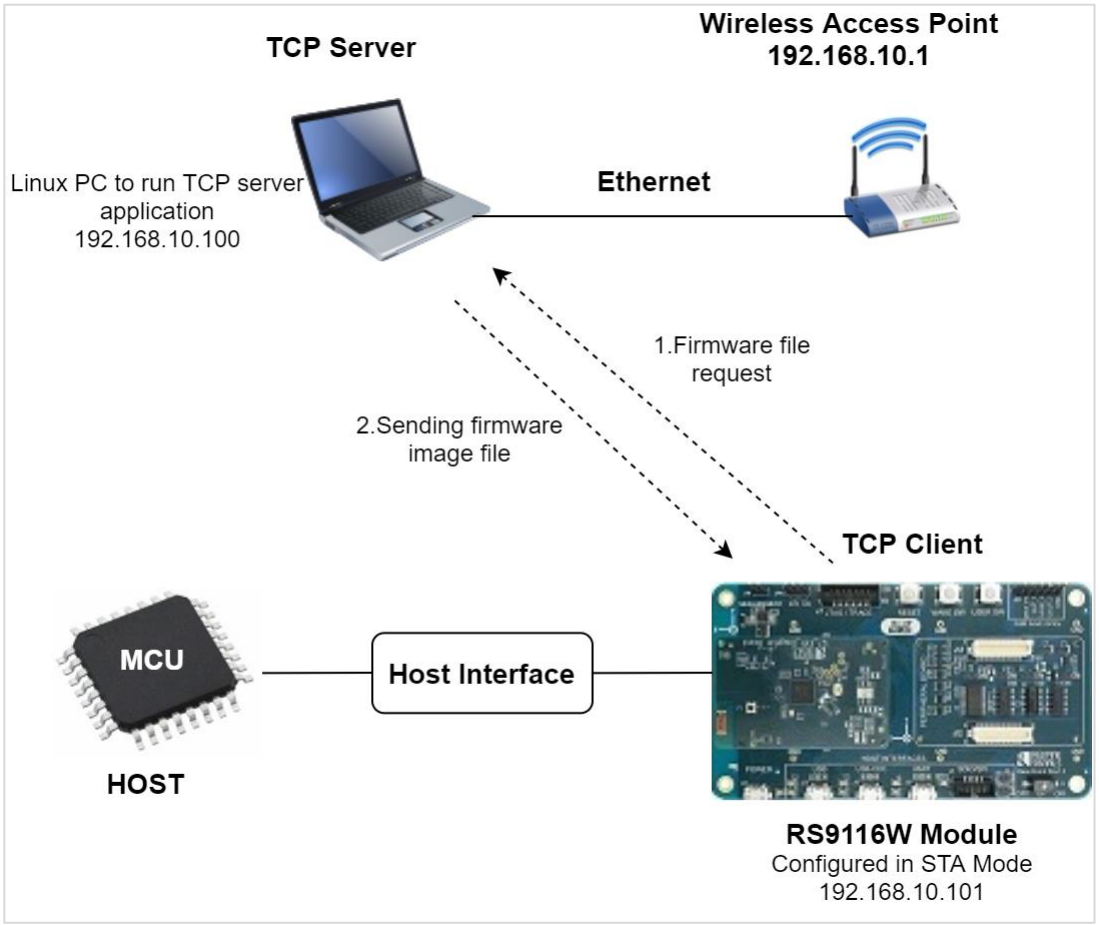


Figure 81; Setup Diagram for Firmware Upgradation from Server Example

Configuration and Steps for Execution

Configuring the Application

- Open ***rsi_firmware_upgradation_app.c*** file and update/modify following macros
  SSID refers to the name of the Access point.

```
#define SSID                          "<REDPINE_AP>"
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                 RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                             "<psk>"
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                     5001
```

SERVER_PORT port refers remote TCP server port number which is opened in Linux PC.

```
#define SERVER_PORT                     5001
```

SERVER_IP_ADDRESS refers remote peer IP (Linux PC) address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS               0x6400A8C0
```

RECV_BUFFER_SIZE refers Memory for receive data

```
#define RECV_BUFFER_SIZE                1027
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                       1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

The IP address needs to be configuring to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order.
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

- Open **rsi_wlan_config.h** file and update/modify following macros :

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Note: rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Copy TCP server application present in release package "***firmware_upgradation/firmware_upgarde_tcp_server.c "*** *into* Linux PC which is connected to access point through LAN.
   Compile and run by providing port number and Firmware file path
3. compile by giving **gcc firmware_upgarde_tcp_server.c** command
4. Run the application **./a.out 5001 RS9116.NBZ.WC.GEN.OSI.x.x.x.rps**
5. After the program gets executed, device connects to AP and open TCP client socket.
6. After TCP connection established with remote server, application sends firmware file request to the server.
7. Server receive request and sends firmware file in chunks.
8. After receiving chunk from remote server, application again sends firmware request to server. Server will wait for the firmware request from the device before sending next chunk.
9. Packet is sent to the device in chunks as shown in the given below figure. After successful up-gradation in TCP server terminal shows "reach end of file".

10. In Application ***rsi_firmware_upgradation_app.c,rsi_fwup_load*** API returns 0x03 response after successful firmware up gradation and closes TCP client socket.

Note:
After Firmware up-gradation, Device needs to be rebooted to get effective of new firmware file. After reboot, Device will take few minutes to give CARD READY indication after first reboot.
Wait for few minutes after power up.

6.12 FTP Client

Protocol Overview

File Transfer Protocol (FTP) is a protocol through which internet users can upload files from their computers to a website or download files from a website to their PCs.
FTP is a client-server protocol that relies on two TCP communications channels between client and server and a command channel for controlling the conversation (command port) and a data channel for transmitting file content(data port). The standard port number used by FTP servers is 21 and is used only for sending commands. Clients initiate conversations with servers by requesting to download a file. Using FTP, a client can upload, download, delete, rename, move and copy files on a server. A user typically needs to log on to the FTP server, although some servers make some or all of their content available without login, which is also known as anonymous FTP.
FTP sessions work in passive or active modes. In active mode, after a client initiates a session via a command channel request, the server initiates a data connection back to the client and begins transferring data. In passive mode, the server instead uses the command channel to send the client the information it needs to open a data channel.

Note:
Silicon Labs devices support only Active mode of FTP sessions.

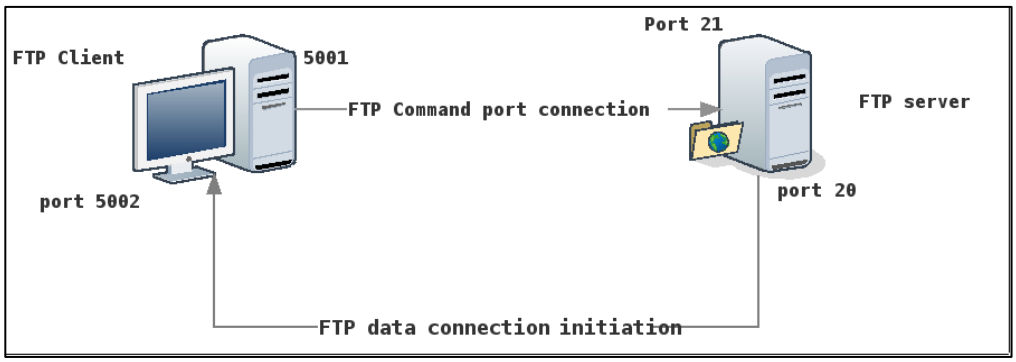


Figure 82: Simple FTP Connection between FTP Server and FTP Client

Overview

This application demonstrates how to connect to FTP server opened on remote peer using FTP client and how to read file from FTP server and how to write file on to the FTP server.
In this application, the Silicon Labs device connects to Access Point and establishes FTP client connection with FTP server opened on remote peer. After successful connection, the application reads the data from **"read.txt"** file present in FTP server and writes back same data read from the file **"read.txt"** by replacing first few bytes with the string "**REDPINE FTP CLIENT DEMO**" to the FTP server by creating file **"write.txt"**.

Sequence of Events

This Application explains user how to:

- Connect to Access Point
- Establish FTP connection with FTP server opened on remote peer
- Read data from "read.txt" file present in FTP server
- Write the data to "write.txt" file which is read from the "read.txt" file by replacing first few bytes with the string "**REDPINE FTP CLIENT DEMO"**

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- Windows PC2 with FTP server installed in it

Note:

- FTP Server demo application can be downloaded from the given below link: https://filezilla-project.org/download.php?type=server
- FTP client functionality verified with FileZilla server version 0.9.51. So, recommended version of FileZilla software is "FileZilla_Server-0_9_51.exe"

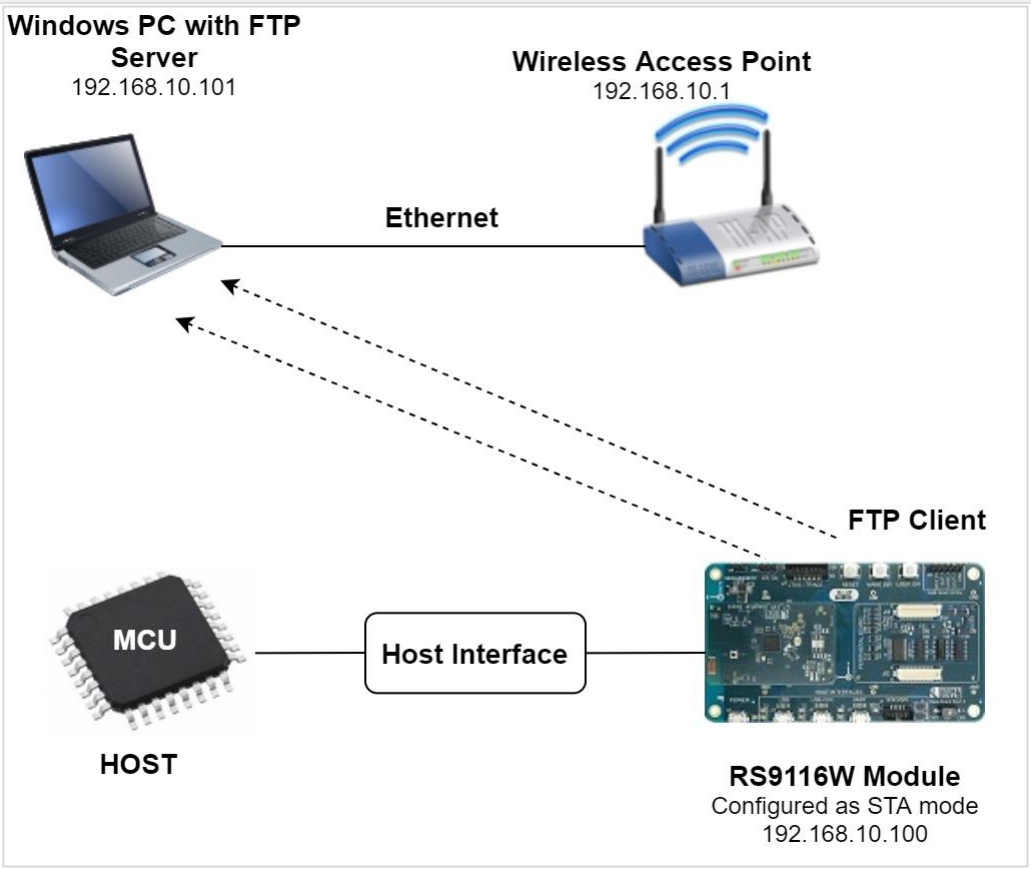
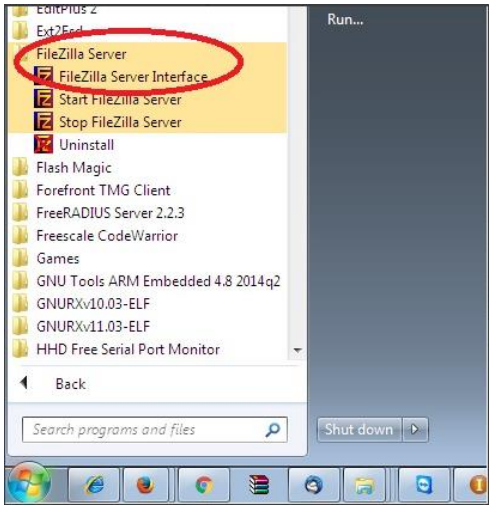


Figure 83: Setup Diagram for FTP Client Demo Example

Configuration and Steps for Execution

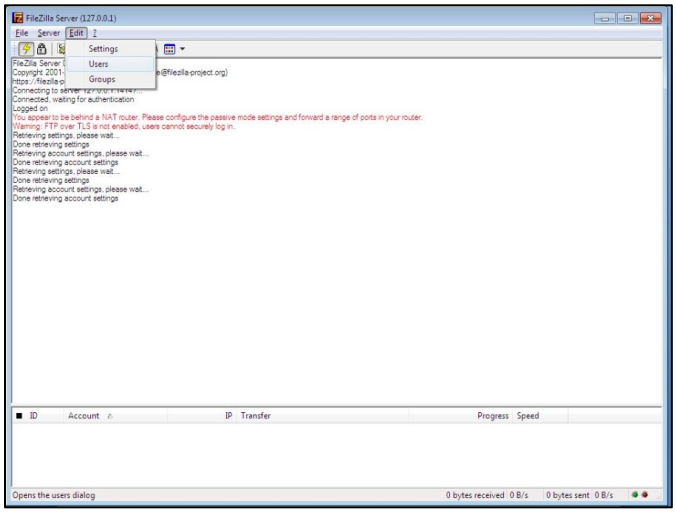
Installation of FTP server

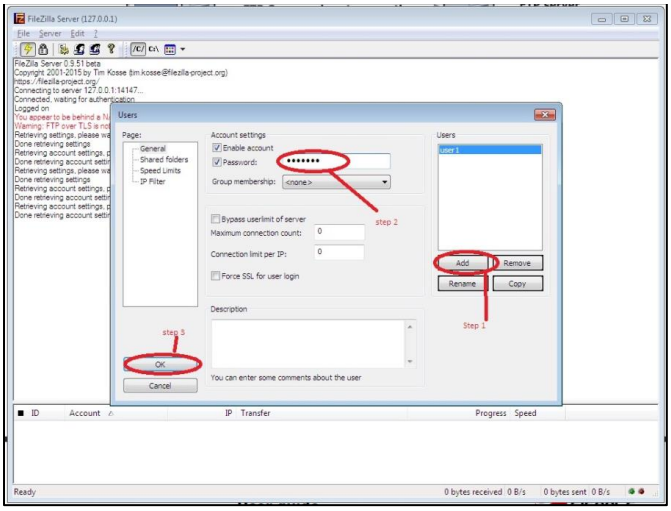
1. Download the FileZilla FTP server from below link - https://filezilla-project.org/download.php?type=server
2. Install downloaded FileZilla FTP server in Windows PC2 which is connected to AP through LAN.
3. Configure and run FTP server. Please refer the below images for configuring and running FileZilla FTP server.
4. After installation, open FileZilla Server interface.

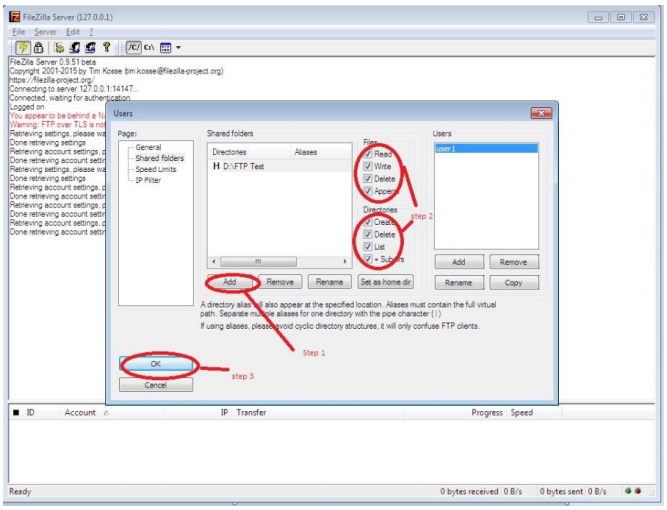


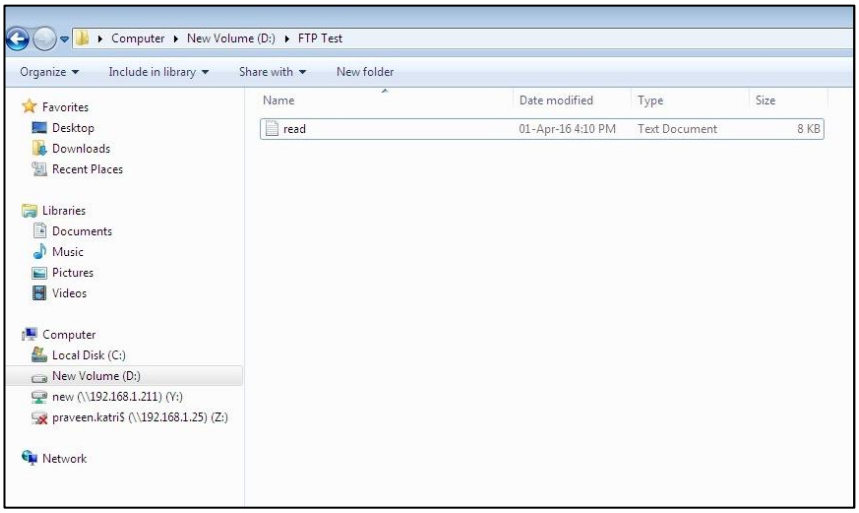
5. After opening FileZilla server interface, connect to server admin interface.

6. After connection with server, configure the user settings.









7. Place "read.txt" file in FTP directory (Ex: "D:\FTP Test\read.txt")

Configuring the Application

- Open ***rsi_ftp_client.c*** file and update/modify the following macros,

  SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

  SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
  Valid configuration is:

RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                          RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                    "<psk>"
```

FTP_SERVER_PORT port refers remote FTP server port number.

By default, FileZilla Server runs on port number 21.

```
#define FTP_SERVER_PORT                        21
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS                      0x6400A8C0
```

FTP Server login username

```
#define FTP_SERVER_LOGIN_USERNAME              "username"
```

FTP Server login password

```
#define FTP_SERVER_LOGIN_PASSWORD              "password"
```

File to read which is on FTP server
Create a file on FTP server with the file name along with the path with data given below

```
#define FTP_FILE_TO_READ                       "read.txt"
```

(Or)

```
#define FTP_FILE_TO_READ                       "D:\FTP Test\read.txt"
```

FILE_CONTENT_LENGTH refers content length of the read file from FTP server (Ex: configure **FILE_CONTENT_LENGTH** >= Sizeof ("read.txt"))

```
#define FILE_CONTENT_LENGTH                    10000
```

File name to create on FTP server and write the same content which is read from "read.txt"

```
#define FTP_FILE_TO_WRITE                     "write.txt"
```

(Or)

```
#define FTP_FILE_TO_WRITE                    "D:\FTP Test\write.txt"
```

To rename a file on FTP server

```
#define FTP_FILE_TO_RENAME                   "example.txt"
```

To set the directory on FTP server

```
#define FTP_DIRECTORY_SET                    "/work/FTP_EXAMPLE/FTP"
```

To create directory on FTP server

```
#define FTP_DIRECTORY_CREATE                 "FTP"
```

To list the directories on FTP server

```
#define FTP_DIRECTORY_LIST                   "/work/FTP_EXAMPLE"
```

To configure IP address

DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                             1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode, it should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                            0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                              0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                              0x00FFFFFF
```

The following parameters need to be configured if OS is used.
WLAN task priority is given, and this should be of low priority

```
#define RSI_WLAN_TASK_PRIORITY              1
```

Driver task priority is given, and this should be of highest priority

```
#define RSI_DRIVER_TASK_PRIORITY            1
```

WLAN Task stack size is configured by this macro

```
#define RSI_WLAN_TASK_STACK_SIZE            500
```

Driver Task stack size is configured by this macro

```
#define RSI_DRIVER_TASK_STACK_SIZE          500
```

1. pen **rsi_wlan_config.h** file and update/modify following macros :

```
#define CONCURRENT_MODE                     RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                 FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                   RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP          TCP_IP_FEAT_DHCPV4_CLIENT| TCP_IP_FEAT_FTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP          FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP         EXT_FEAT_256k_MODE
#define RSI_BAND                            RSI_BAND_2P4GHZ
```

> **Note**:
> **rsi_wlan_config.h** file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect the Silicon Labs device in STA mode.
2. Run FTP server on Windows PC2 and place "read.txt" file in the FTP directory.
3. After the program gets executed, the Silicon Labs device will be connected to same access point having the configuration same as that of in the application and get IP.
4. After successful connection with the Access Point, the device connects to FTP server and reads the file content of given file ("**read.txt**") and creates a file name "**write.txt**" in FTP directory and writes the same content which is read from "**read.txt**" by replacing first few bytes with "**FTP CLIENT DEMO**". After successful file write WiSeConnect device disconnects from FTP server.
5. Refer below images for message exchanges with FTP server and for read and write files,

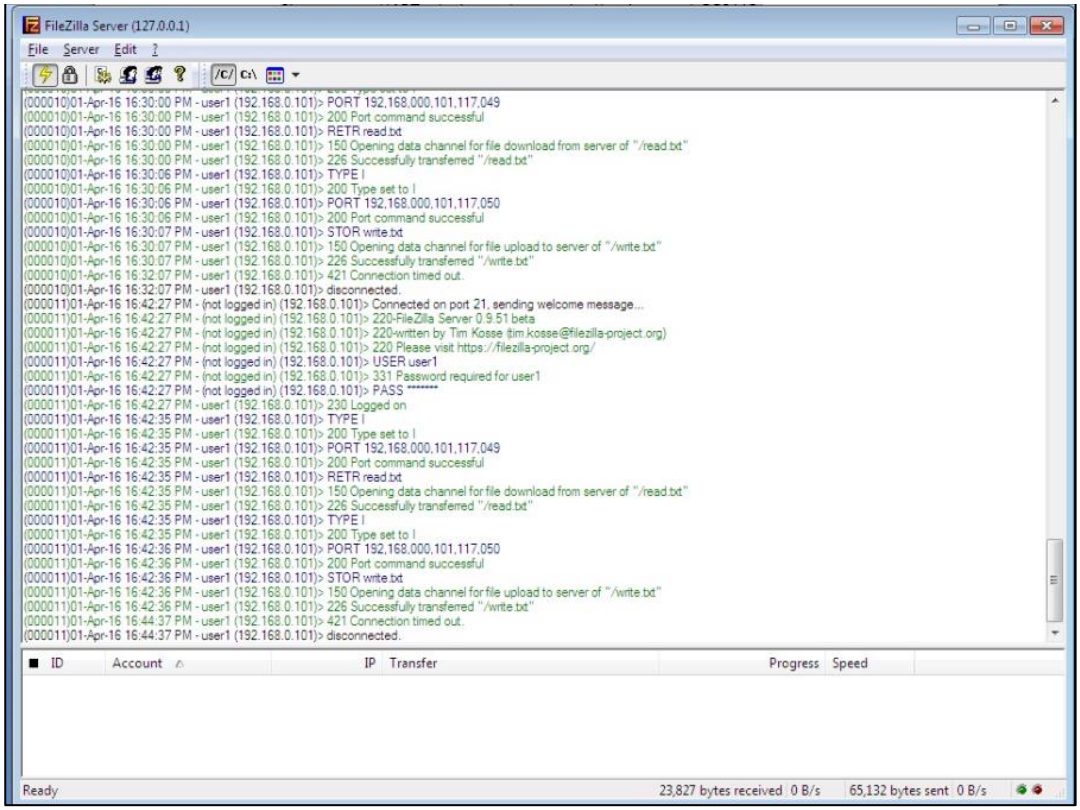
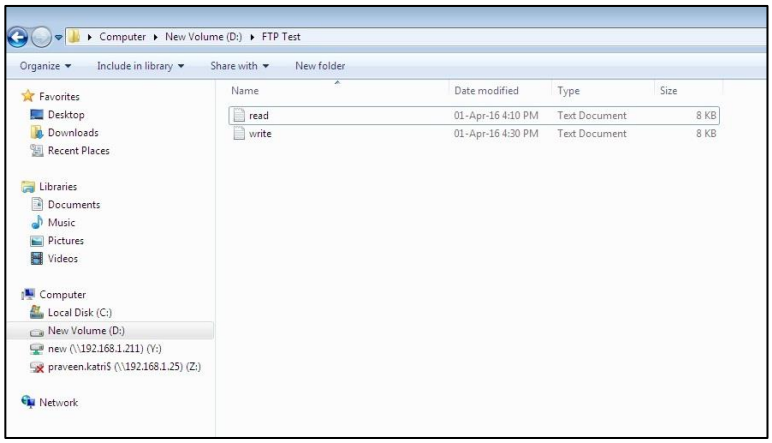


Figure 84: FTP Message Exchange



6.13 HTTP Client

Protocol Overview

HTTP client is a client-side HTTP transport library. HTTP client's purpose is to transmit and receive HTTP messages. HTTP client will not attempt to process content, execute javascript embedded in HTML pages, try to guess content type, or other functionality unrelated to the HTTP transport.

Overview

This application demonstrates how to create Silicon Labs device as HTTP/HTTPs client and do HTTP PUT, GET and POST operations with the HTTP/HTTPs server opened on remote peer.
In this application, the device configures as Wi-Fi station and connects to Access point and do HTTP/HTTPs PUT, GET and post operation with HTTP/HTTPs server opened on remote peer.

Sequence of Events

This Application explains user how to:

- Load appropriate CA certificate to the Device to interact with HTTPS Server.

- Connect to Access Point
- Run HTTP/HTTPS Server Remote side.
- Request for HTTP/HTTPs PUT, GET and POST.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- Wi-Fi Access point
- Windows PC2 with openssl support and python installed

> **Note**:
> Installed python should support the following modules: Thread, HTTPServer, BaseHTTPRequestHandler, cgi, curdir, sep, sys

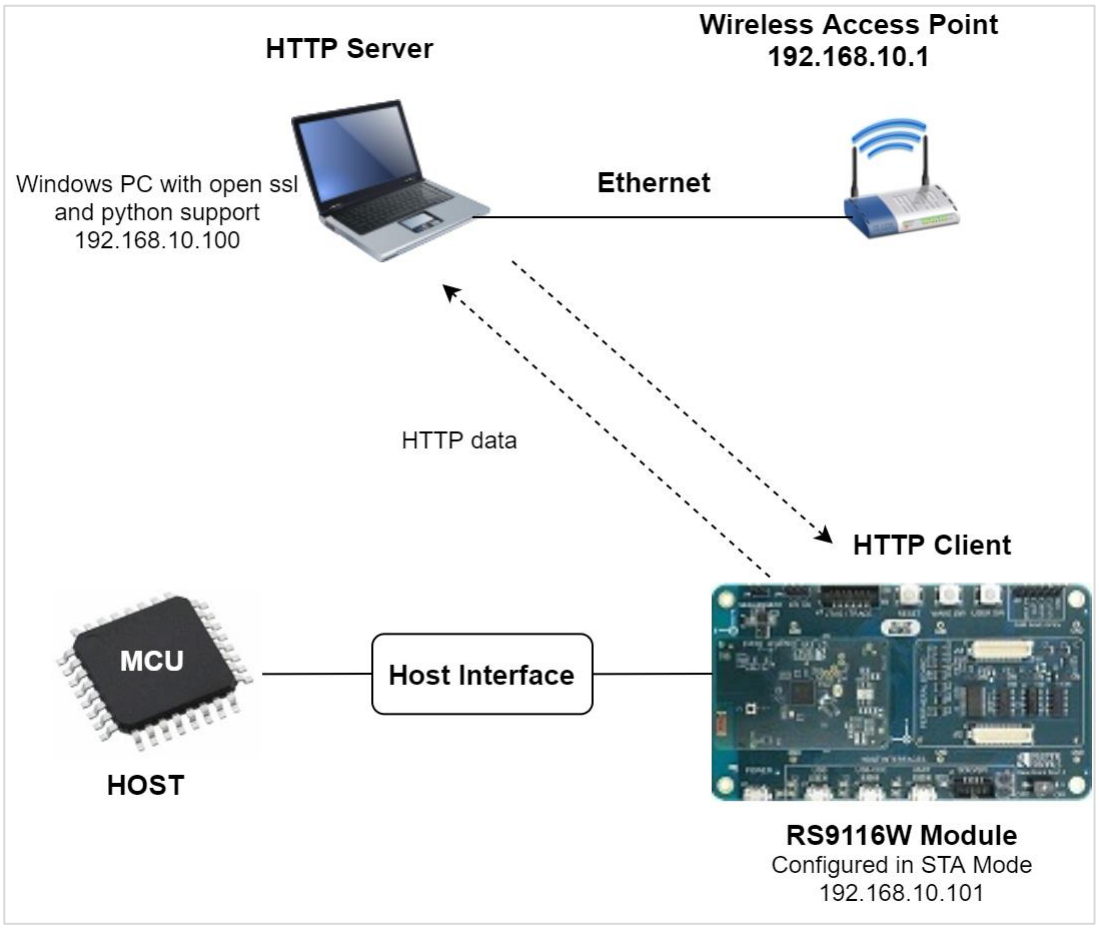


Figure 85: Setup Diagram for HTTP/HTTPS Client Example

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_http_client_app.c** file and update/modify following macros,

SSID refers to the name of the Access point.

```
#define SSID                          "<REDPINE_AP>"
```

CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

#define CHANNEL_NO 0

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
Valid configuration is
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                 RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                           "<psk>"
```

To Load certificate

```
#define LOAD_CERTIFICATE              1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API. By default, application loading "cacert.pem" certificate **LOAD_CERTIFICATE** enable.

> **Note**:
> All the certificates are given in the release package **(Path: RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\utilities\certificates)**

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC.

```
#define DHCP_MODE                     1
```

> **Note**:
> If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
>
> (Or)
>
> If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order. Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                     0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

To establish connection and request for HTTP PUT or HTTP GET or HTTP POST to the HTTP/HTTPS Server configure the below macros.

FLAGS refers to open normal HTTP client socket or HTTP client socket over SSL with IPv4 or IPv6
Default configuration of application is normal HTTP client socket with IPv4.

```
#define  FLAGS                      0
```

(Or)
If user wants to open HTTP client socket over SSL with IPv4 then set FLAGS to 2 **(HTTPS_SUPPORT)**.

```
#define FLAGS                    HTTPS_SUPPORT
```

(Or)
If user wants to use HTTP client post large data then set FLAGS to 32 **(HTTP_POST_DATA)**.

```
#define FLAGS                         HTTP_POST_DATA
```

(Or)
If user wants to open HTTP client with version 1.1 then set FLAGS to 64 **(HTTP_V_1_1)**.

```
#define FLAGS                         HTTP_V_1_1
```

(Or)
If user wants to open normal HTTP client socket with IPv6 then set FLAGS macro to 1 **(HTTPV6)**.

```
#define FLAGS                         HTTPV6
```

(Or)
If user wants to open HTTP client socket over SSL with IPv6 then set FLAGS macro to 3 **(HHTPV6 |HTTPS_SUPPORT)**

```
#define FLAGS                          (HTTPV6 | HTTPS_SUPPORT)
```

HTTP_PORT refers Port number of the remote HTTP server which is opened in Windows PC2.

```
#define HTTP_PORT                    80
```

(Or)

HTTP_PORT refers Port number of the remote HTTPS server which is opened in Windows PC2.

```
#define HTTP_PORT                    443
```

HTTP_SERVER_IP_ADDRESS refers IP address of the HTTP/HTTPS server

Note:
HTTP_SERVER_IP_ADDRESS should be as the below mentioned format as it is a string.

```
#define HTTP_SERVER_IP_ADDRESS       "192.168.10.1"
```

HTTP_URL refers HTTP resource name

```
#define HTTP_URL                     "/index.html"
#define HTTP_HOSTNAME                "192.168.10.1"
```

HTTP_HOSTNAME refers host name

```
#define HTTP_HOSTNAME                "192.168.10.1"
```

HTTP extended header

```
#define HTTP_EXTENDED_HEADER         NULL
```

HTTP/HTTPS user name

```
#define USERNAME                    "admin"
```

Password for server

```
#define PASSWORD                    "admin"
```

HTTP/HTTPS post data

```
#define HTTP_DATA
"employee_name=xxx&employee_id=RSXYZ123&designation=Engineer&company=Silicon Labs&location=xxxx"
```

Max HTTP PUT buffer length

```
#define MAX_HTTP_CLIENT_PUT_BUFFER_LENGTH 900
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN             15000
```

Application buffer length

```
#define APP_BUFF_LEN                2000
```

Enable this macro if Web-page content is returned as HTTP response from server

```
#define WEBPAGE_AS_HTTP_PUT_RESPONSE    0
```

2. Open **rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE            RSI_DISABLE
#define RSI_FEATURE_BIT_MAP        FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS          RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP (TCP_IP_FEAT_DHCPV4_CLIENT| TCP_IP_FEAT_HTTP_CLIENT)
```

If user wants to connect with HTTPs server set RSI_TCP_IP_FEATURE_BIT_MAP as follows,

```
#define RSI_TCP_IP_FEATURE_BIT_MAP   (TCP_IP_FEAT_DHCPV4_CLIENT| TCP_IP_FEAT_SSL |
TCP_IP_FEAT_HTTP_CLIENT
```

```
#define RSI_CUSTOM_FEATURE_BIT_MAP   FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP  EXT_FEAT_256k_MODE
#define RSI_BAND                     RSI_BAND_2P4GHZ
```

If user want to get the HTTP response code as returned by server, this flag should be enabled.

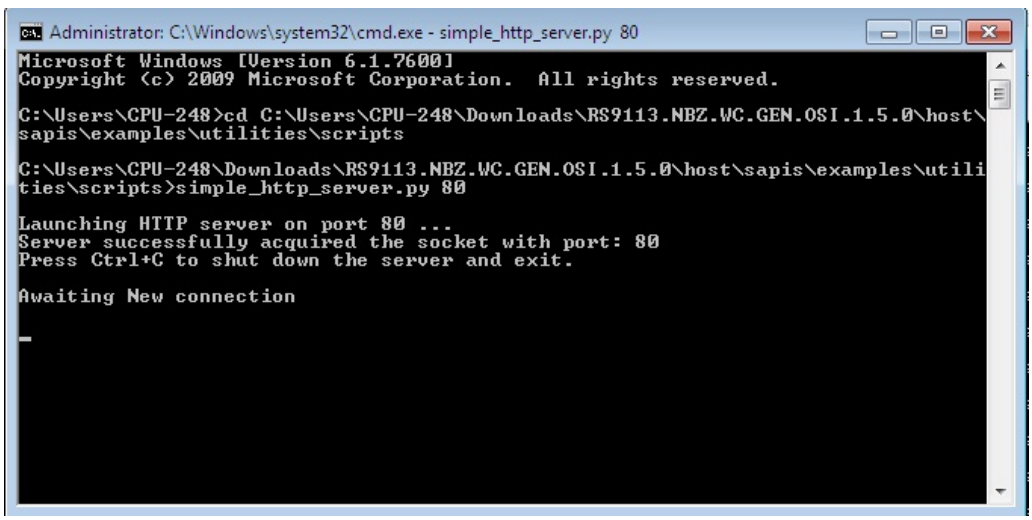
/! Provide HTTP/HTTPS response status code indication to application E.g. 200, 404 etc
/*==*/
//! Enable or Disable feature

```
#define RSI_HTTP_STATUS_INDICATION_EN     RSI_DISABLE
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application (Python Script as HTTP Server)

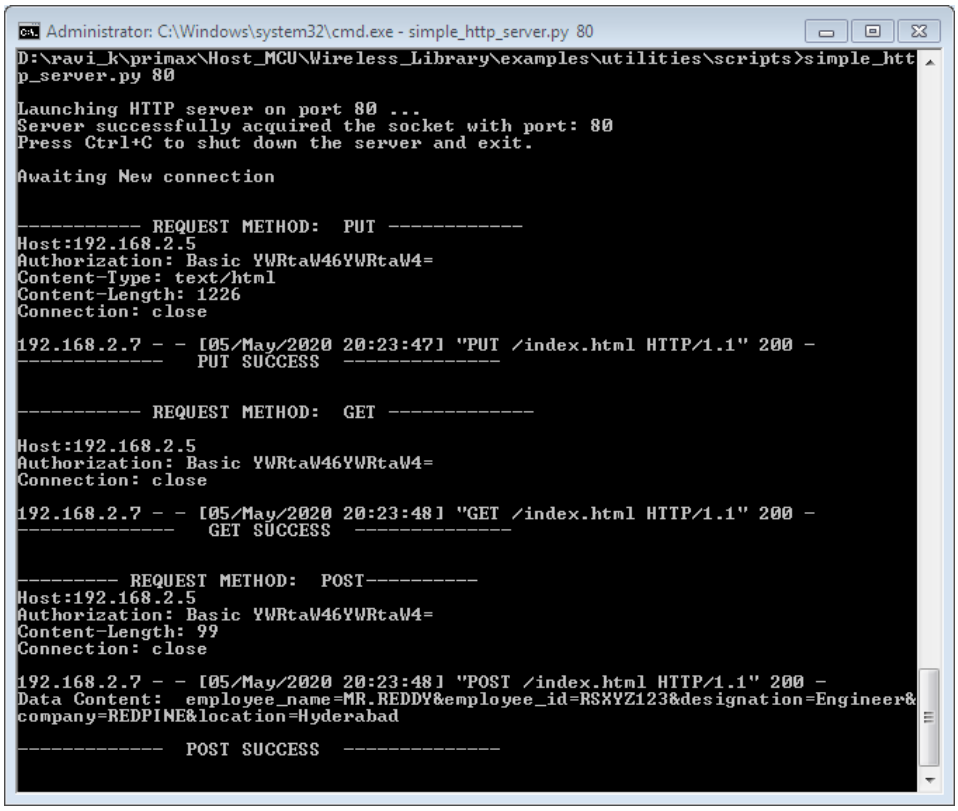
- In Windows PC2, install python and run HTTP server.
- In release package python scripts are provided to open HTTP server in the path: ***utilities/script***
- Run ***simple_http_server.py*** by port number 80 as argument to open HTTP server.



Note:

Release package includes only HTTP server script. If user wants to test HTTPs client, then user has to run HTTPs server which supports HTTPs PUT, GET and POST.

- After the program gets executed, the device connects to AP and get IP.
- After successful connection with Access Point, the Silicon Labs device request for HTTP PUT to PUT/Create the file on to the server, which is given in index.txt file and wait until put file complete.
- Remote web server accepts a PUT request and writes the received data to a file. User can find the created new file "index.html" on Windows PC2 in the following path, ***utilities/scripts***
- After successful creation of file using HTTP PUT, Silicon Labs device request for the file "index.html" using HTTP GET method and wait until complete response receive from Server.
- After receiving complete response for the given HTTP GET, the device posts the given data in HTTP_DATA macro to HTTP server using HTTP POST meth
- User can see the log messages at HTTP server. Please find the below image for success responses for HTTP PUT, HTTP GET and HTTP POST.



Executing the Application (Apache as HTTP/s Server)

- In Windows PC2, install Apache and run HTTP/s server.
- Install and configure Apache using below steps

To install an Apache HTTP Server:

Step 1:

1. Navigate to Apache Website - (httpd.apache.org)
2. Click on "*Download*" link for the latest stable version
3. After being redirect to the download page, Select: "*Files for Microsoft Windows*"
4. Select one of the websites that provide binary distribution (for example: ***Apache Lounge***)
5. After being redirect to "Apache Lounge" website (https://www.apachelounge.com/download/), Select: Apache x.x.xx Win64 link

6. After downloaded, unzip the file httpd-x.x.xx-Win64-VC15.zip into C:/

Step 2:

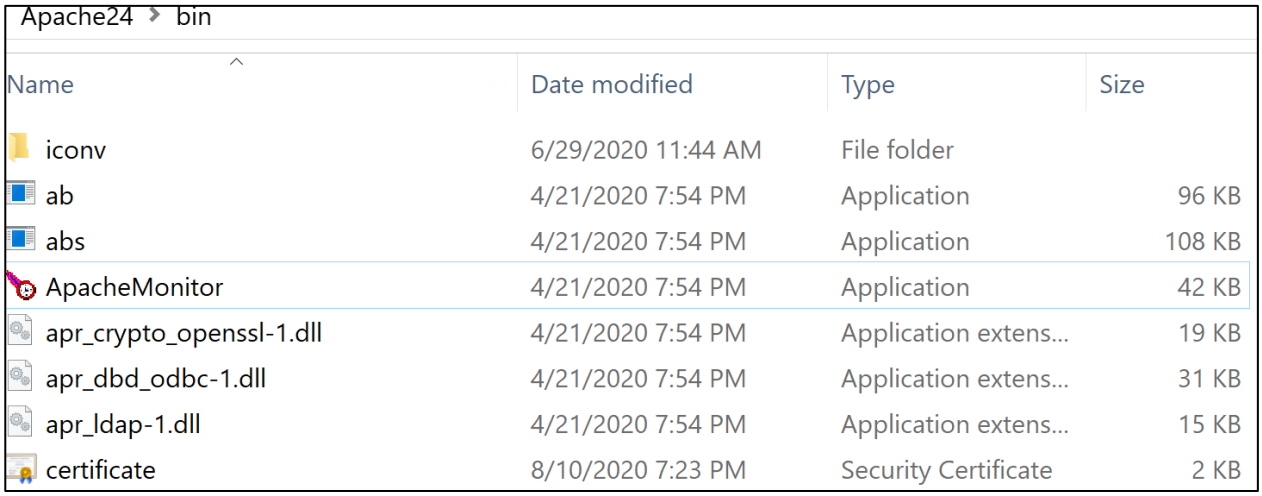
1. Open a command prompt: *Run as Administrator*
2. Navigate to directory c:/Apache24/bin
3. Add Apache as a Windows Service: **httpd.exe -k install**

Step 4:

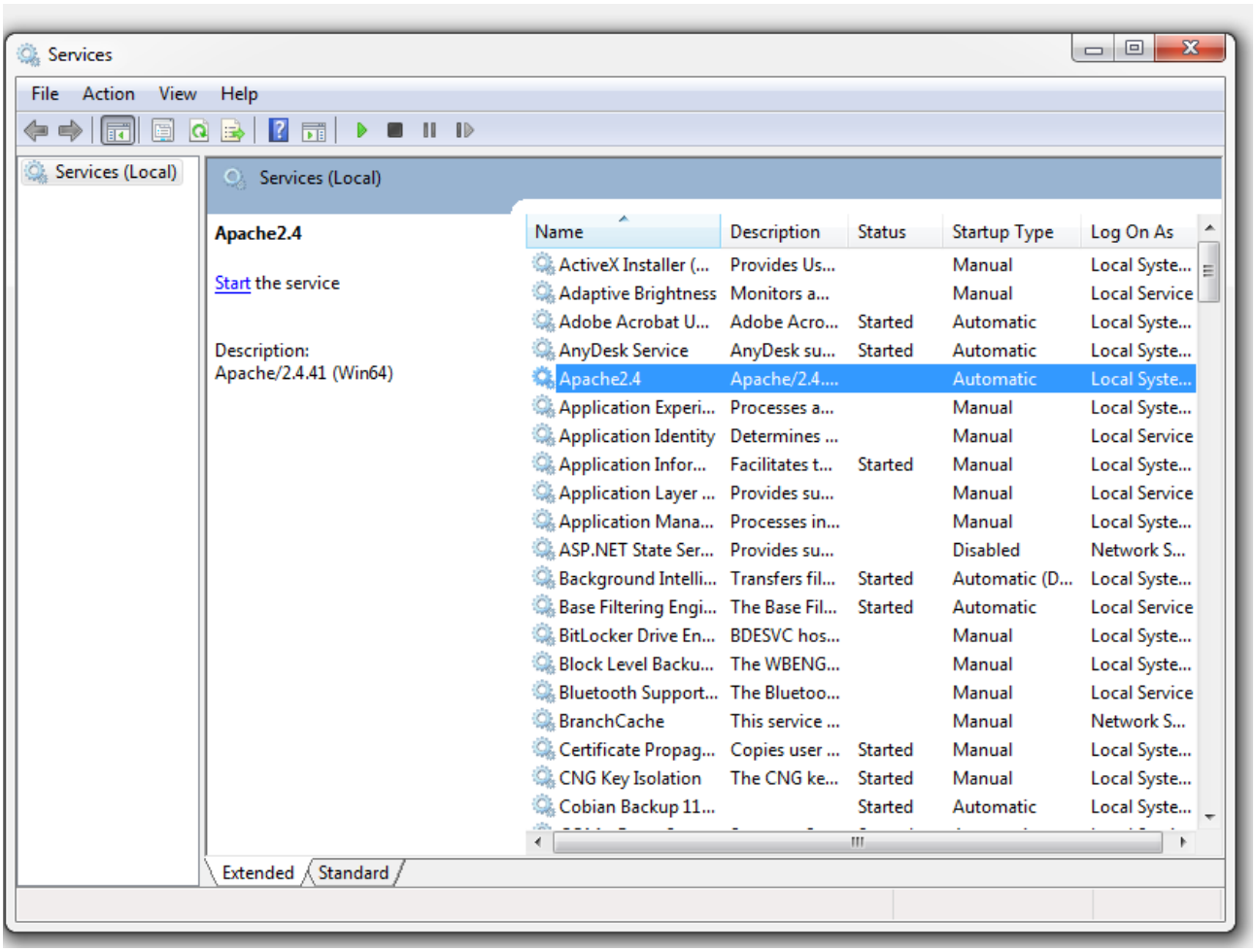
1. Open Windows Services and start Apache HTTP Server
2. Open a Web browser and type the machine IP in the address bar and hit Enter
   The message "*It works!*" should be seen.

To Configure an Apache HTTP Server:

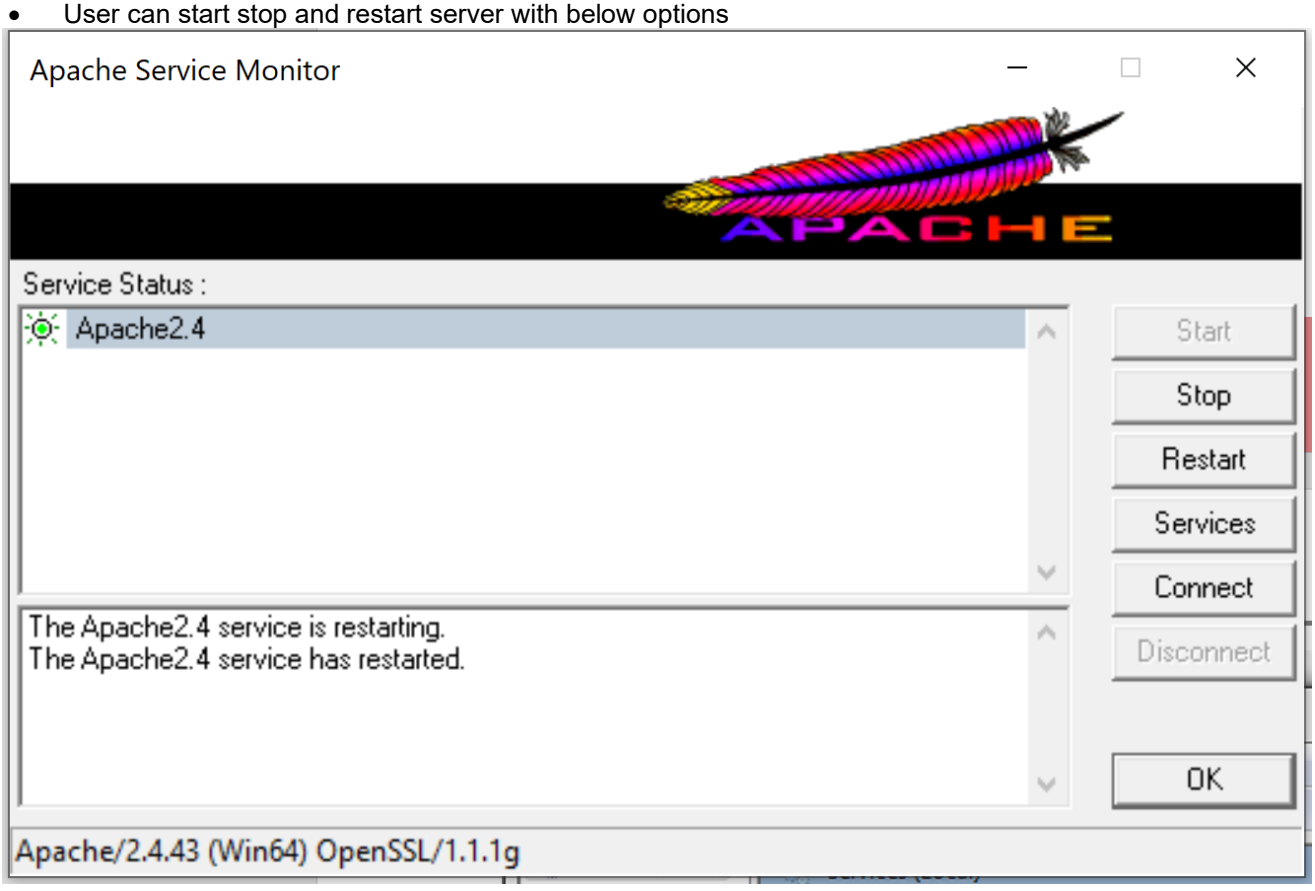
- Navigate to the below path:
  - C:\Apache24\conf
- Now edit the file httpd.conf
  - vim httpd.conf
- Change the below lines with your local host IP.
  - Listen {localhost IP}:80
  - ServerName {localhost IP}:80
- Save and exit the file.
- Navigate to the below path:
  - C:\Apache24\bin
  - Right click on ApacheMonitor and Run as Administrator



- Apache server get started as shown below



- User can start stop and restart server with below options



To Configure an Apache HTTPS Server:

- Path for openssl.exe and openssl.cnf files in Apache:
  - C:\Apache24\conf\openssl.cnf
  - Copy server-cert.pem -key server-key.pem from RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\examples\utilities\certificates to C:\Apache\bin
- Edit the httpd_ssl.conf present in path C:\Apache24\conf\extra
  - Edit the following lines:
    - ServerName 192.168.0.103:443
    - SSLCertificateFile "${SRVROOT}/conf/ server-key.pem "
    - SSLCertificateKeyFile "${SRVROOT}/conf/ server-cert.pem "
- Connect remote server AP , get IP and start the server
- Start the module to connect to AP , get IP and connect to Server
- Remote web server accepts a PUT request and writes the received data to a file. User can find the created new file "index.html" on Windows PC2 in the following path, Apache24/htdocs
- After successful creation of file using HTTP PUT, Silicon Labs device request for the file "index.html" using HTTP GET method and wait until complete response receive from Server.
- After receiving complete response for the given HTTP GET, the device post the given data in HTTP_DATA macro to HTTP server using HTTP POST method

6.14 HTTP Client Post Data

Protocol Overview

HTTP client is a client-side HTTP transport library. HTTP client's purpose is to transmit and receive HTTP messages. HTTP client will not attempt to process content, execute javascript embedded in HTML pages, try to guess content type, or other functionality unrelated to the HTTP transport.

Overview

This application demonstrates how to create Silicon Labs device as HTTP/HTTPs client and do GET and POST operations with the HTTP/HTTPs server opened on remote peer.
In this application, the device configures as Wi-Fi station and connects to Access point and do HTTP/HTTPs Post and Get operation with HTTP/HTTPs server opened on remote peer.

Sequence of Events

This Application explains user how to:

- Load appropriate CA certificate to the Device to interact with HTTPS Server.
- Connect to Access Point
- Run HTTP/HTTPS Server Remote side.
- Request for HTTP/HTTPs POST and GET.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors. The following sections are applicable to WiSeConnect parts only.

WiSeConnect based Setup Requirements

- Windows PC1 with Keil or IAR IDE
- Silicon Labs Module
- WiFi Access point
- Windows PC2 with openssl support and python installed

Note: Installed python should support the following modules:
Thread, HTTPServer, BaseHTTPRequestHandler, cgi, curdir, sep, sys

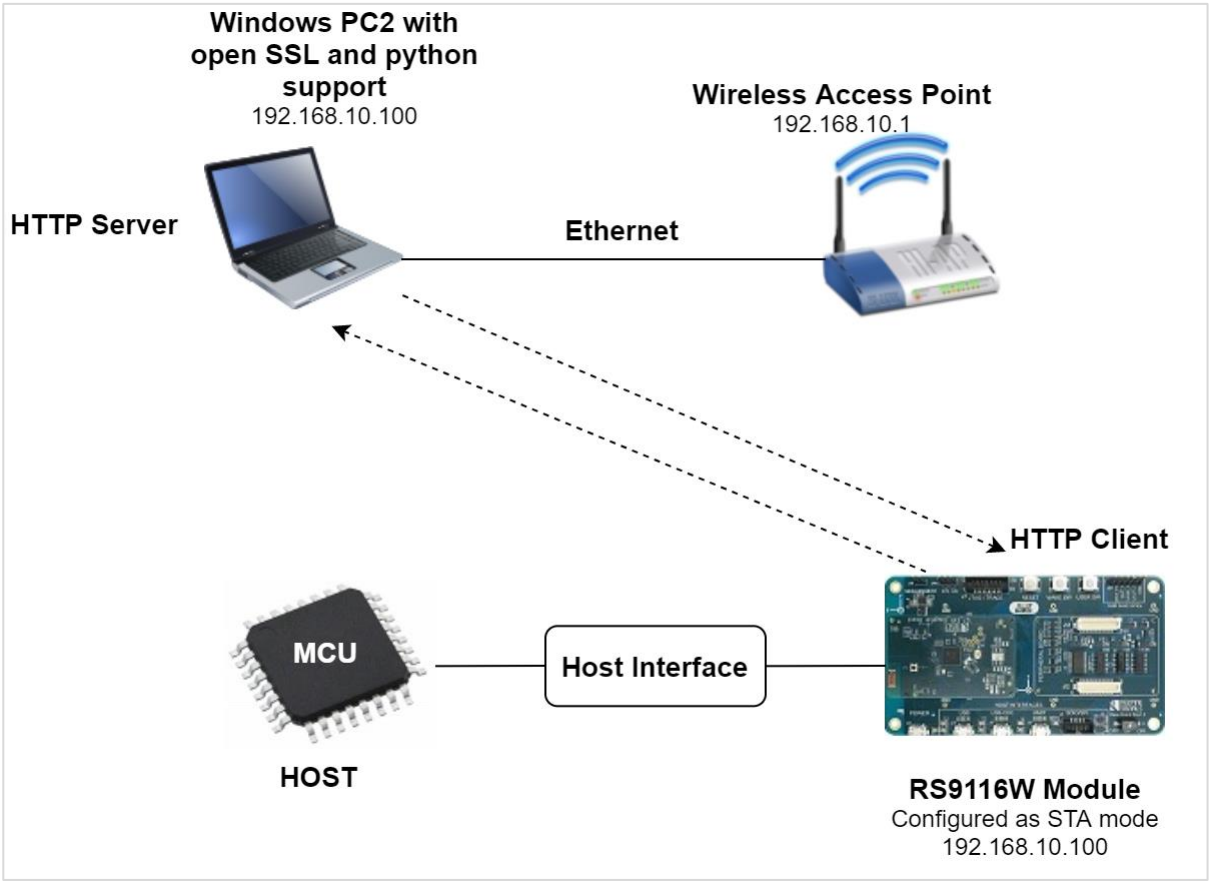


Figure 86: Setup Diagrams

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_http_client_post_data_app.c** file and update/modify following macros,

   SSID refers to the name of the Access point.

```
#define SSID                          "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

```
#define CHANNEL_NO                    0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                 RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                           "<psk>"
```

To abort the application after the mentioned packet count

```
#define RSI_HTTP_MAX_PKT_COUNT_ABORT     10
```

To enable RSI_ENABLE_HTTP_ABORT functionality, set the define ABORT to 1.

```
#define RSI_ENABLE_HTTP_ABORT            0
```

To Load certificate

```
#define LOAD_CERTIFICATE                 1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API. By default, application loading "cacert.pem" certificate LOAD_**CERTIFICATE** enable.

> **Note**:
> All the certificates are given in the release package Path:
> **RS9116.NB0.WC.GENR.OSI.xxx\host\sapis\examples\utilities\certificates**

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                        1
```

> **Note**:
> If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
>
> (Or)
>
> If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                        0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                          0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                          0x00FFFFFF
```

To establish connection and request for HTTP PUT or HTTP GET or HTTP POST to the HTTP/HTTPS Server configure the below macros.
DEVICE_PORT refers internal socket port number.

```
#define DEVICE_PORT                     5001
```

FLAGS refer to open normal HTTP client socket or HTTP client socket over SSL with IPv4 or IPv6
Default configuration of application is normal HTTP client socket with IPv4.

```
#define FLAGS 0
```

(Or)
If user wants to open HTTP client socket over SSL with IPv4 then set FLAGS to 2 **(HTTPS_SUPPORT)**.

```
#define FLAGS HTTPS_SUPPORT
```

(Or)
If user wants to open normal HTTP client socket with IPv6 then set FLAGS macro to 1 **(HTTPV6)**.

```
#define FLAGS                           HTTPV6
```

(Or)
If user wants to open HTTP client socket over SSL with IPv6 then set FLAGS macro to 3 **(HHTPV6 |HTTPS_SUPPORT)**

```
#define FLAGS                          (HTTPV6 | HTTPS_SUPPORT)
```

If user wants to use HTTP client to post large HTTP data set FLAGS macro to 32

```
#define FLAGS                           (HTTP_POST_DATA)
```

If user wants to use HTTP client version 1.1 set FLAGS macro to 64

```
#define FLAGS                           (HTTP_V_1_1)
```

HTTP_PORT refers Port number of the remote HTTP/HTTPS server which is opened in Windows PC2.

```
#define HTTP_PORT                       80
```

(Or)

HTTP_PORT refers Port number of the remote HTTPS server which is opened in Windows PC2.

```
#define HTTP_PORT                       443
```

HTTP_SERVER_IP_ADDRESS refers IP address of the HTTP/HTTPS server

> **Note**:
> HTTP_SERVER_IP_ADDRESS should be as the below mentioned format as it is a string.

```
#define HTTP_SERVER_IP_ADDRESS          "192.168.10.1"
```

HTTP_URL refers HTTP resource name

```
#define HTTP_URL                          "/index.html"
#define HTTP_HOSTNAME                     "192.168.10.1"
```

HTTP_HOSTNAME refers host name

```
#define HTTP_HOSTNAME                    "192.168.10.1"
```

HTTP extended header

```
#define HTTP_EXTENDED_HEADER              NULL
```

HTTP/HTTPS user name

```
#define USERNAME                          "admin"
```

Password for server

```
#define PASSWORD                         "admin"
```

Max HTTP POST DATA buffer length

```
#define MAX_HTTP_CLIENT_POST_DATA_BUFFER_LENGTH 900
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN               15000
```

Application buffer length

```
#define APP_BUFF_LEN                   2000
```

2. Open **rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                RSI_DISABLE
#define RSI_FEATURE_BIT_MAP            FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS              RSI_DISABLE
```

If user wants to connect with HTTP server set RSI_TCP_IP_FEATURE_BIT_MAP as follows,

```
#define RSI_TCP_IP_FEATURE_BIT_MAP    (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_HTTP_CLIENT)
```

If user wants to connect with HTTPs server set RSI_TCP_IP_FEATURE_BIT_MAP as follows,

```
#define RSI_TCP_IP_FEATURE_BIT_MAP  (TCP_IP_FEAT_DHCPV4_CLIENT| TCP_IP_FEAT_SSL |
TCP_IP_FEAT_HTTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP   FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP  EXT_FEAT_256k_MODE
#define RSI_BAND                     RSI_BAND_2P4GHZ
```

If user want to get the HTTP response code as returned by server, this flag should be enabled.

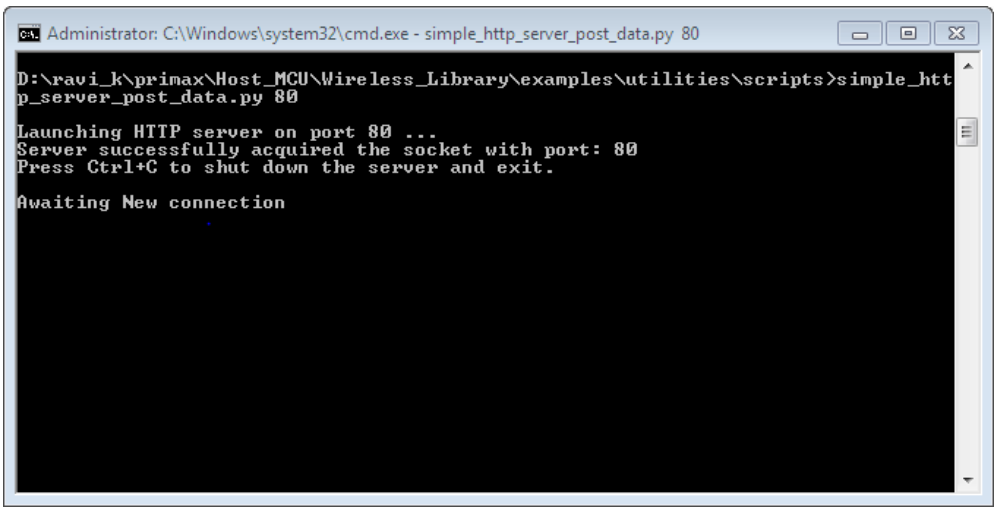
/! Provide HTTP/HTTPS response status code indication to application e.g 200, 404 etc
/*==*/
//! Enable or Disable feature

```
#define RSI_HTTP_STATUS_INDICATION_EN     RSI_DISABLE
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application (Python script as HTTP Server)

- In Windows PC2, install python and run HTTP server.
- In release package python scripts are provided to open HTTP server in the following path:
  - ***sapis/examples/utilities/scripts***
- Run ***simple_http_server_post_data.py*** by giving port number 80 as argument to open HTTP server.



Note:
Release package includes only HTTP server script. If user wants to test HTTPs client, then user has to run HTTPs server which supports HTTPs PUT, GET and POST.

- After the program gets executed, the Silicon Labs device connects to AP and get IP.
- After successful connection with Access Point, the Silicon Labs device request for HTTP POST to send the user given file to the server, which is given in index.txt file and wait until post file complete.
- Remote web server accepts a POST request and give response.
- After successful sending of file using HTTP POST, the device request for the file "index.html" using HTTP GET method and wait until complete response receive from Server.

Executing the Application (Apache as HTTP/s Server)

- In Windows PC2, install Apache and run HTTP/s server.
- Install and configure Apache using below steps

To install an Apache HTTP/s Server:

Step 1:

- Navigate to Apache Website - (httpd.apache.org)
- Click on "*Download*" link for the latest stable version
- After being redirect to the download page, Select: "*Files for Microsoft Windows*"
- Select one of the websites that provide binary distribution (for example: ***Apache Lounge***)
- After being redirect to "Apache Lounge" website (https://www.apachelounge.com/download/), Select: Apache x.x.xx Win64 link
- After downloaded, unzip the file httpd-x.x.xx-Win64-VC15.zip into C:/

Step 2:

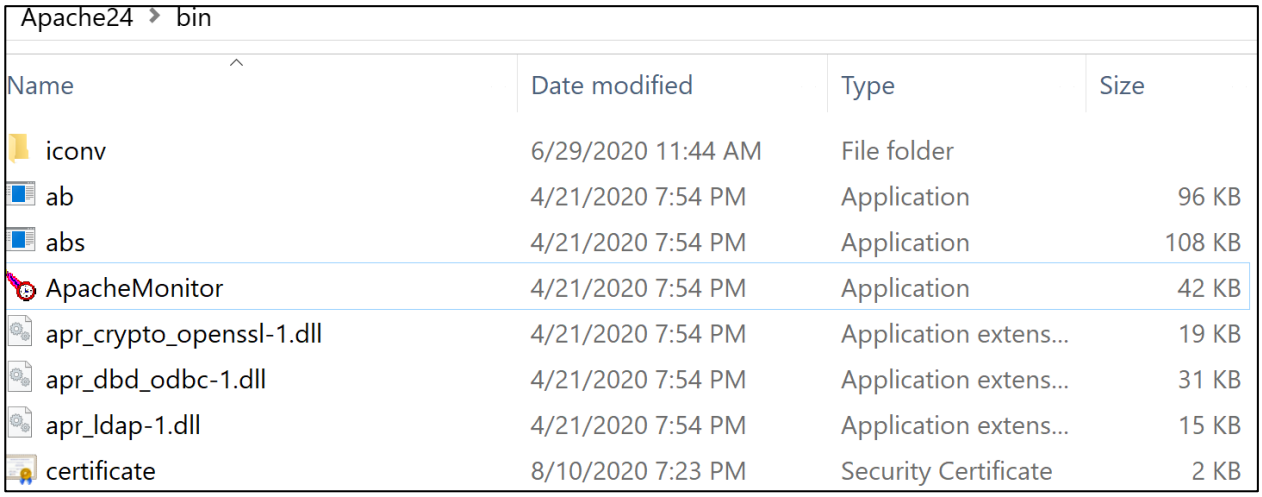
- Open a command prompt: *Run as Administrator*
- Navigate to directory c:/Apache24/bin
- Add Apache as a Windows Service: **httpd.exe -k install**

Step 4:

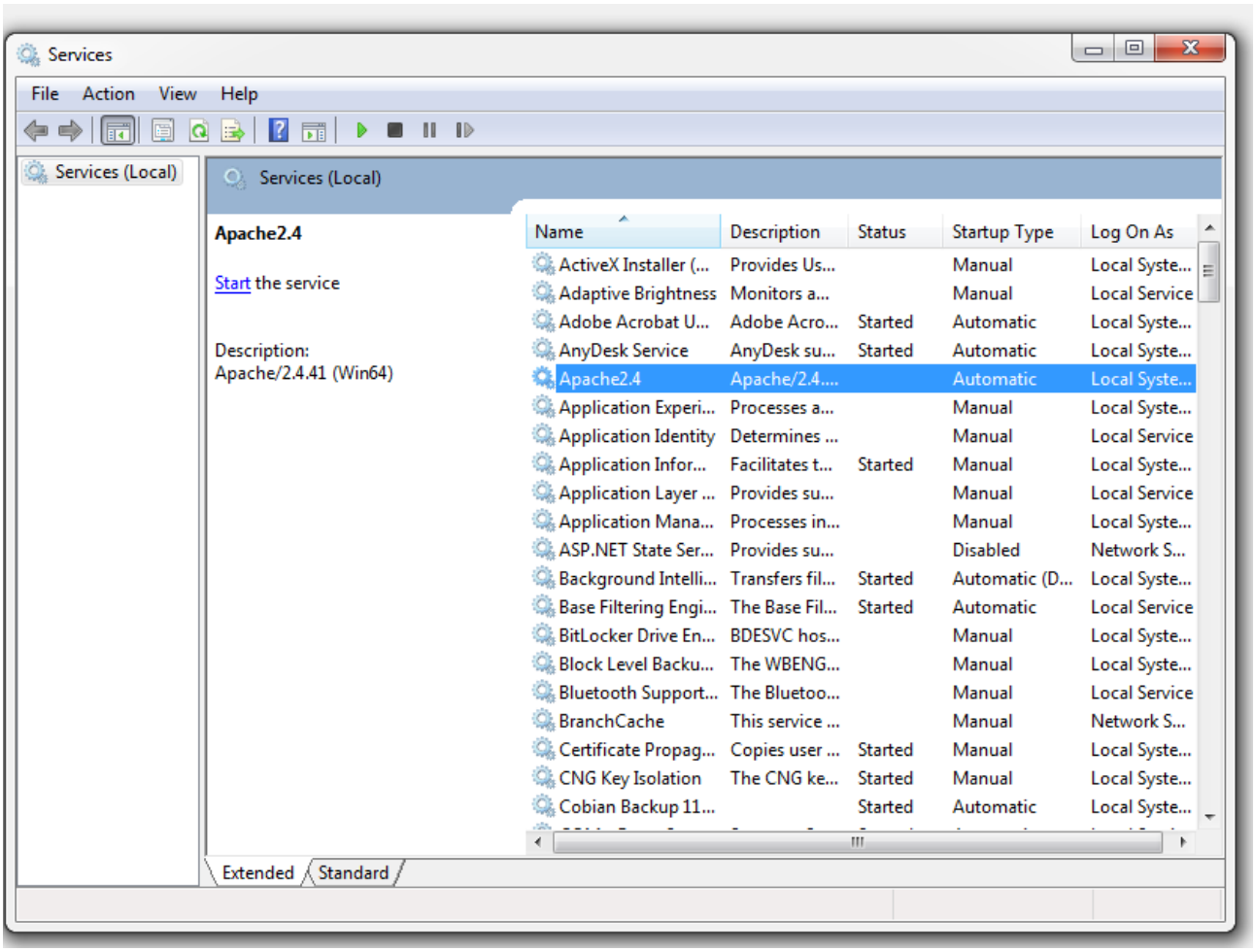
- Open Windows Services and start Apache HTTP Server
- Open a Web browser and type the machine IP in the address bar and hit Enter
- The message "*It works!*" should be seen.

To Configure an Apache HTTP Server:

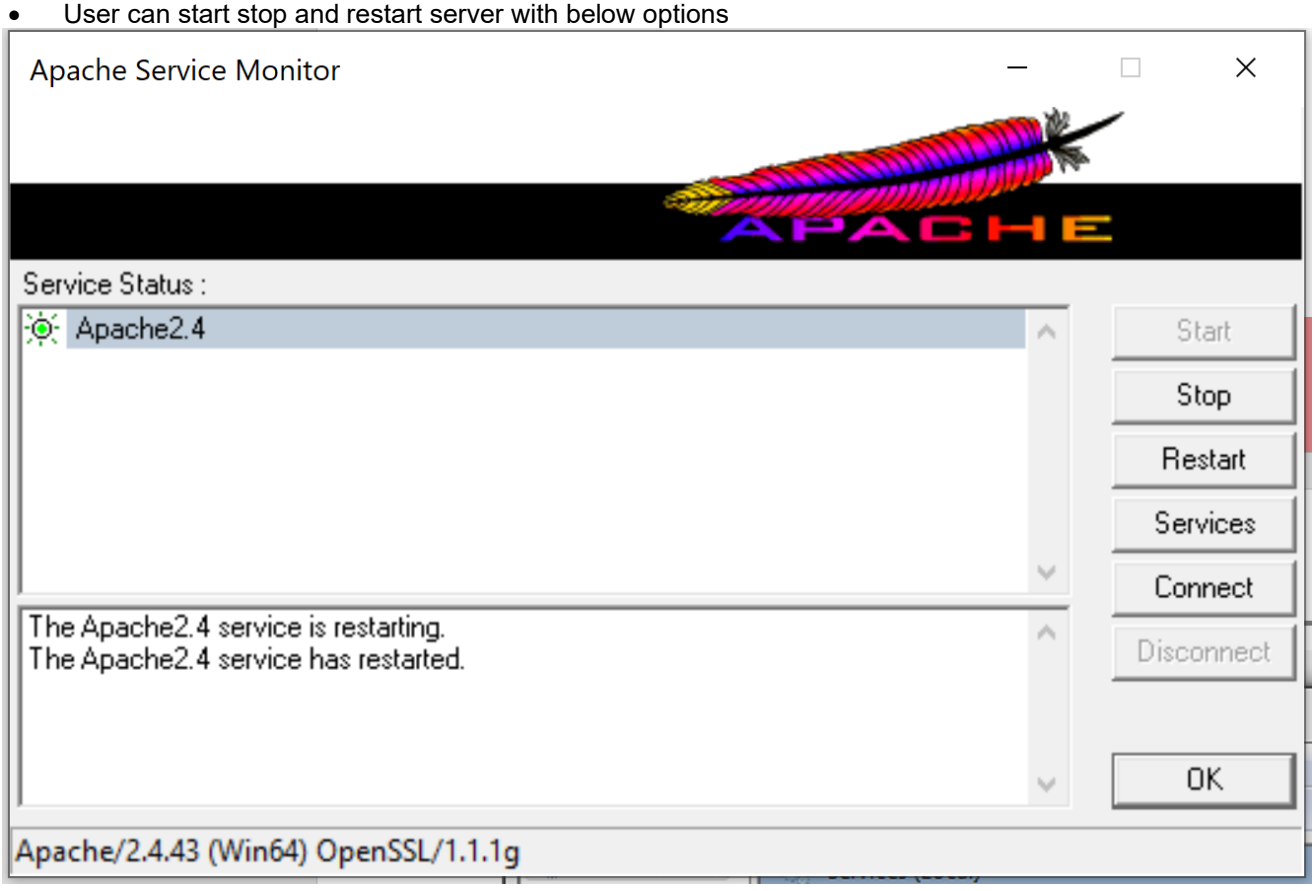
- Navigate to the below path:
  - C:\Apache24\conf
- Now edit the file httpd.conf
  - vim httpd.conf
- Change the below lines with your local host IP.
  - Listen {localhost IP}:80
  - ServerName {localhost IP}:80
- Save and exit the file.
- Navigate to below path:
  - C:\Apache24\bin
  - Right click on ApacheMonitor and Run as Administrator



- Apache server get started as shown below



- User can start stop and restart server with below options



To Configure an Apache HTTPS Server:

- Path for openssl.exe and openssl.cnf files in Apache:
  - C:\Apache24\conf\openssl.cnf
  - Copy server-cert.pem -key server-key.pem from RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\examples\utilities\certificates to C:\Apache\bin
- Edit the httpd_ssl.conf present in path C:\Apache24\conf\extra
  - Edit the following lines:
    - ServerName 192.168.0.103:443
    - SSLCertificateFile "${SRVROOT}/conf/ server-key.pem "
    - SSLCertificateKeyFile "${SRVROOT}/conf/ server-cert.pem "

- After the program gets executed, the Silicon Labs device connects to AP and get IP.
- After successful connection with Access Point, the Silicon Labs device request for HTTP POST to send the user given file to the server, which is given in index.txt file and wait until post file complete.
- Remote web server accepts a POST request and give response.
- After successful sending of file using HTTP POST, the device request for the file "index.html" using HTTP GET method and wait until complete response receive from Server.

6.15 Instant BgScan

Overview

This application demonstrates how to enable Background scan and get results of available access points after successful connection with the Access Point in station mode.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Initiate Instant Background scan.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Silicon Labs Module
- Wi-Fi Access point

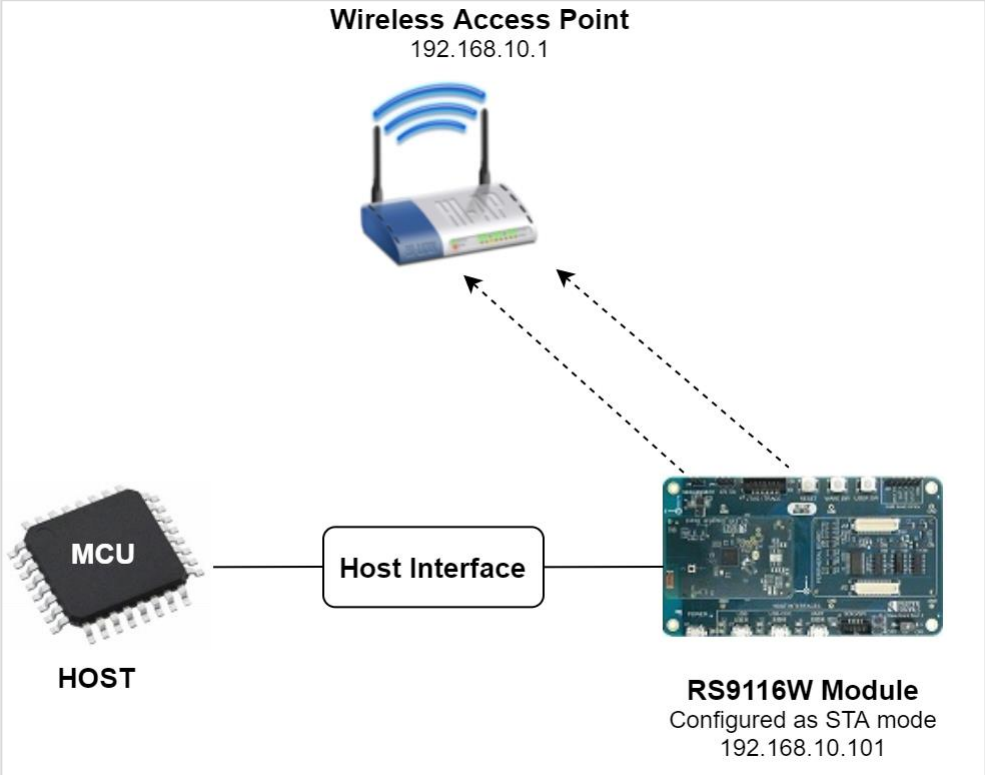


Figure 87: Setup Diagram for Instant Background Scan

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_instant_bgscan.c** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels.

```
#define CHANNEL_NO                              0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     "<psk>"
```

   To configure IP address
   DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                              1
```

> **Note:** If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
> (Or)
> If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                              0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                0x00FFFFFF
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                        15000
```

APP_BUFF_LEN refers buffer length to read back ground scan results.

```
#define APP_BUFF_LEN                           200
```

2. Open **rsi_wlan_config.h** file and update/modify following macros :

```
#define CONCURRENT_MODE                        RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                    FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                      RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP             TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP             FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP            EXT_FEAT_256k_MODE
#define RSI_BAND                               RSI_BAND_2P4GHZ
#define RSI_BG_SCAN_SUPPORT                    RSI_ENABLE
#define RSI_BG_SCAN_ENABLE                     RSI_ENABLE
#define RSI_INSTANT_BG                         RSI_ENABLE
#define RSI_MULTIPROBE                         RSI_ENABLE
```

> **Note: rsi_wlan_config.h** file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. After program gets executed, the device would scan and connect to the access point and get IP.
2. After successful connection, the application initiates Instant Background scan. The Silicon Labs device scans for Access Points and gives scanned Access Points information in "**rsi_wlan_bgscan_profile"** API response. User can parse the response buffer **"bgscan_results"** for Access Points details.

6.16 MQTT Client

Protocol Overview

MQTT is a publish-subscribe based "light weight" messaging protocol for using on top of the TCP/IP protocol. The MQTT connection itself is always between one client and the broker, no client is connected to another client directly.

MQTT client
A MQTT client is any device from a micro controller to a full-fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. MQTT Clients can share the information on a particular topic using MQTT protocol. MQTT clients connect to the MQTT broker using TCP connection and can subscribe and publish on any desired topic. The other clients which are subscribed for that topic will receive the published messages.

MQTT Broker
The publish-subscribe messaging pattern requires a message broker. The broker is primarily responsible for receiving all messages, filtering them, deciding like who is interested in it and then sending the message to all subscribed clients.
It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. A simple demonstration of subscribing and publishing of temperature is shown below:

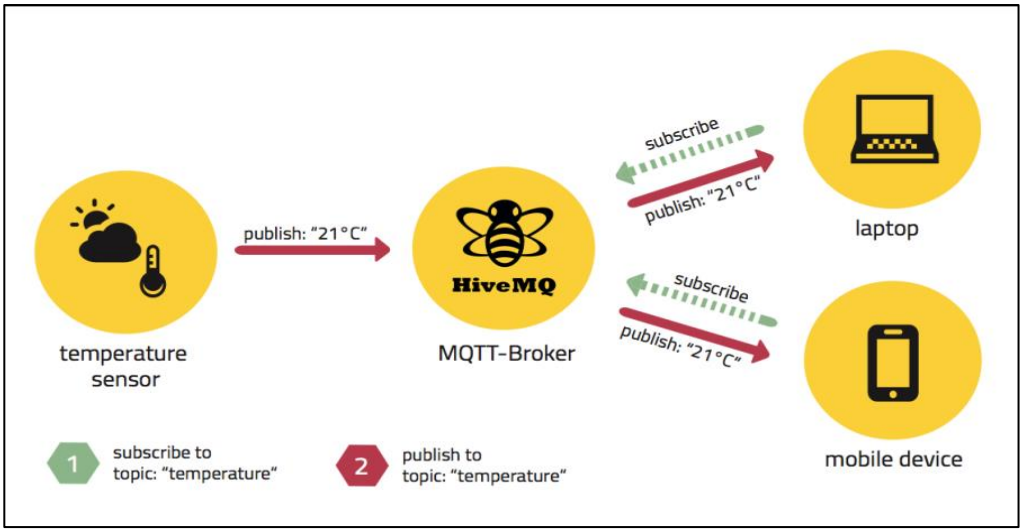


Figure 88: Demonstration of MQTT Protocol

Overview

This application demonstrates how to configure Silicon Labs device as MQTT client and how to establish connection with MQTT broker and how to subscribe, publish and receive the MQTT messages from MQTT broker.
In this application, Silicon Labs device configured as WiFi station and connects to Access Point. After successful WiFi connection, application connects to MQTT broker and subscribes to the topic **"REDPINE"** and publishes a message **"THIS IS MQTT CLIENT DEMO FROM REDPINE"** on that subscribed topic. And application waits to receive the data published on subscribed topic by other clients.

Sequence of Events

This Application explains user how to:

- Connect to Access Point

- Establish MQTT client connection with MQTT broker
- Subscribe the topic **"REDPINE"**
- Publish message **"THIS IS MQTT CLIENT DEMO FROM REDPINE"** on the subscribed topic **"REDPINE"**
- Receive data published by other clients on the same subscribed topic **"REDPINE"**.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Silicon Labs Module
- Windows PC2 with with MQTT broker installed in it
- Windows PC3 with with MQTT client utility installed in it

Note:

MQTT broker for different OS platforms can be downloaded from the below link
http://mosquitto.org/download/
Ex: Install "mosquitto-1.4.8-install-win32.exe"
MQTT Utility which has to be installed in Windows PC 3 can be downloaded from the below given link
https://www.eclipse.org/downloads/download.php?file=/paho/1.0/org.eclipse.paho.mqtt.utility-1.0.0.jar

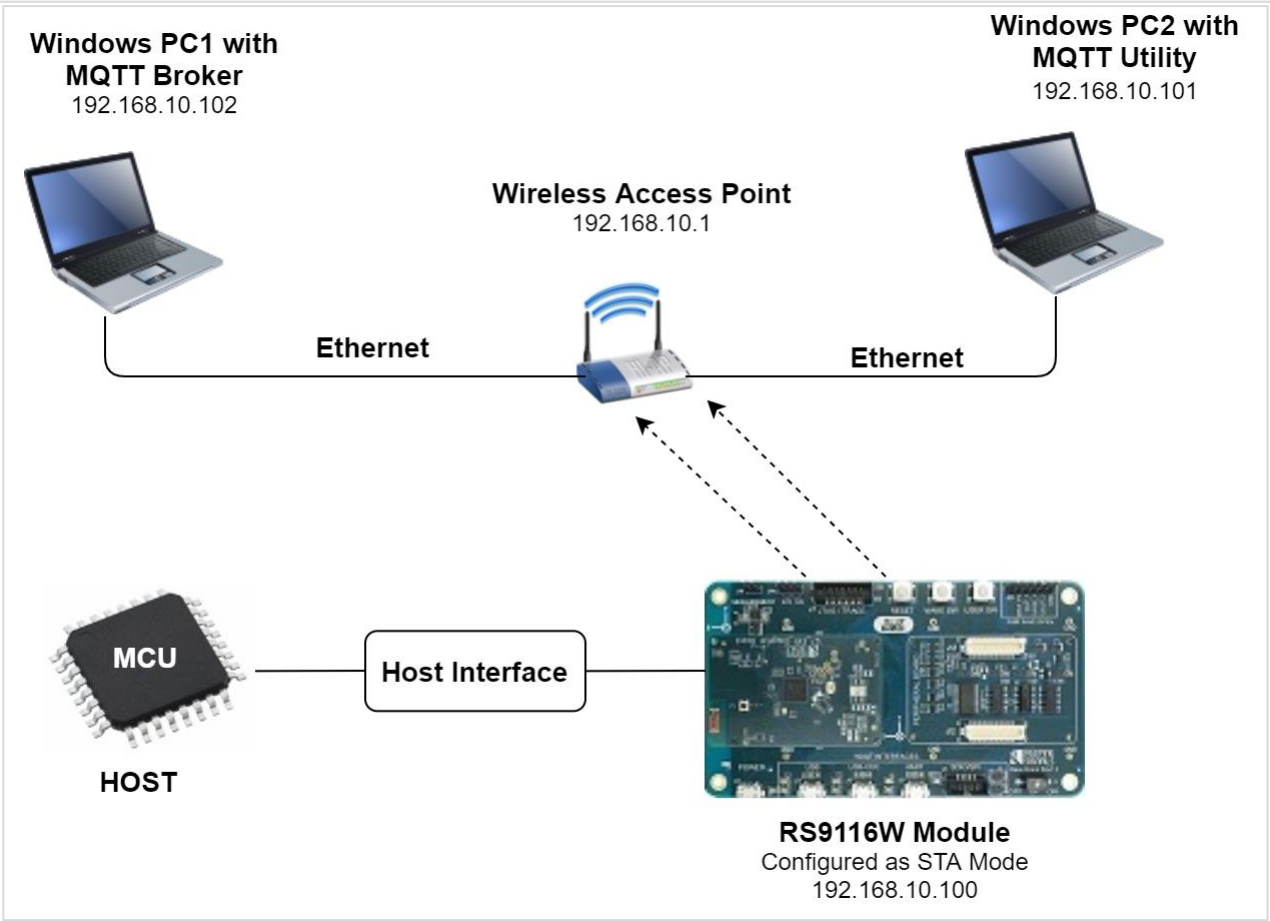


Figure 89: Setup Diagram for MQTT Client Example

Configuration and Steps for Execution

Configuring the Application

1. Open **RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\examples\wlan\mqtt_client\rsi_mqtt.c** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                  "<REDPINE_AP>"
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                         RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                   "<psk>"
```

   CLIENT_PORT port refers device MQTT client port number

```
#define CLIENT_PORT                           5001
```

   SERVER_PORT port refers remote MQTT broker/server port number

```
#define SERVER_PORT                           1883
```

   SERVER_IP_ADDRESS refers remote peer IP address (Windows PC2) to connect with MQTT broker/server socket.
   IP address should be in long format and in little endian byte order.
   Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS                     0x6400A8C0
```

   MQTT client keep alive period

```
#define RSI_KEEP_ALIVE_PERIOD                 100
```

   Memory to initialize MQTT client Info structure

```
#define MQTT_CLIENT_INIT_BUFF_LEN             3500
```

   Global buffer or memory which is used for MQTT client initialization. This buffer is used for the MQTT client information storage.
   uint8_t mqqt_client_buffer[MQTT_CLIENT_INIT_BUFF_LEN];
   QOS indicates the level of assurance for delivery of an Application Message.
   QoS levels are:
   0 - At most once delivery
   1 - At least once delivery
   2 - Exactly once delivery

```
#define QOS                                  0
```

RSI_MQTT_TOPIC refers to which topic WiSeConnect MQTT client is supposed to subscribe.

#define RSI_MQTT_TOPIC "REDPINE"

MQTT Message to publish on the topic subscribed

uint8_t publish_message[] ="THIS IS MQTT CLIENT DEMO FROM SILABS"
MQTT Client ID with which MQTT client connects to MQTT broker/server
uint8_t clientID[] = "MQTTCLIENT"
User name for login credentials
int8_t username[] = "username"
Password for login credentials
int8_t password[] = "password"

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                      15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                            1
```

> **Note:**
> If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
> (Or)
> If user wants to configure STA IP address through **STATIC** then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                            0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                              0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                              0x00FFFFFF
```

The following parameters are configured if OS is used.
WLAN task priority is given and this should be of low priority

```
#define RSI_WLAN_TASK_PRIORITY                  1
```

Driver task priority is given and this should be of highest priority

```
#define RSI_DRIVER_TASK_PRIORITY                1
```

WLAN Task stack size is configured by this macro

```
#define RSI_WLAN_TASK_STACK_SIZE                500
```

Driver Task stack size is configured by this macro

```
#define RSI_DRIVER_TASK_STACK_SIZE              500
```

Open **RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\examples\wlan\mqtt_client\rsi_wlan_config.h** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP             EXT_FEAT_256k_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

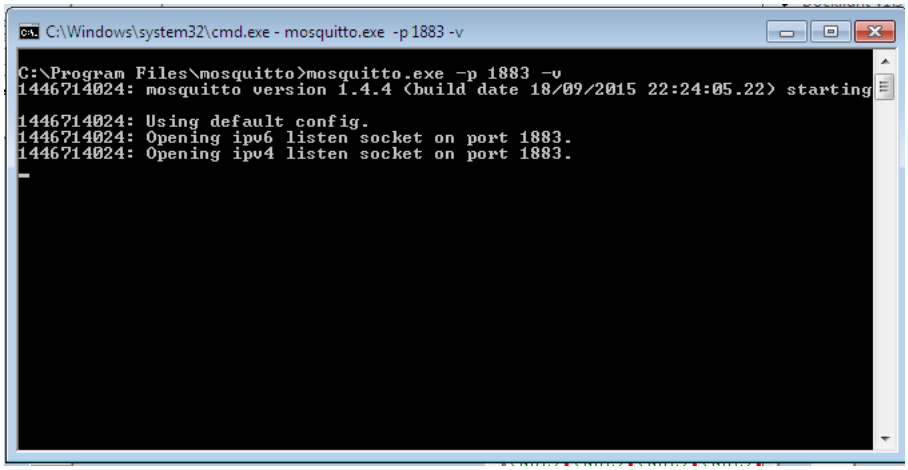
Note:

- **rsi_wlan_config.h** file is already set with desired configuration in respective example folders, user need not change for each example.
- In **rsi_mqtt_client.h** change **MQTT_VERSION** macro to either 3 or 4 based on the MQTT broker support version. (Supported versions 3 and 4).

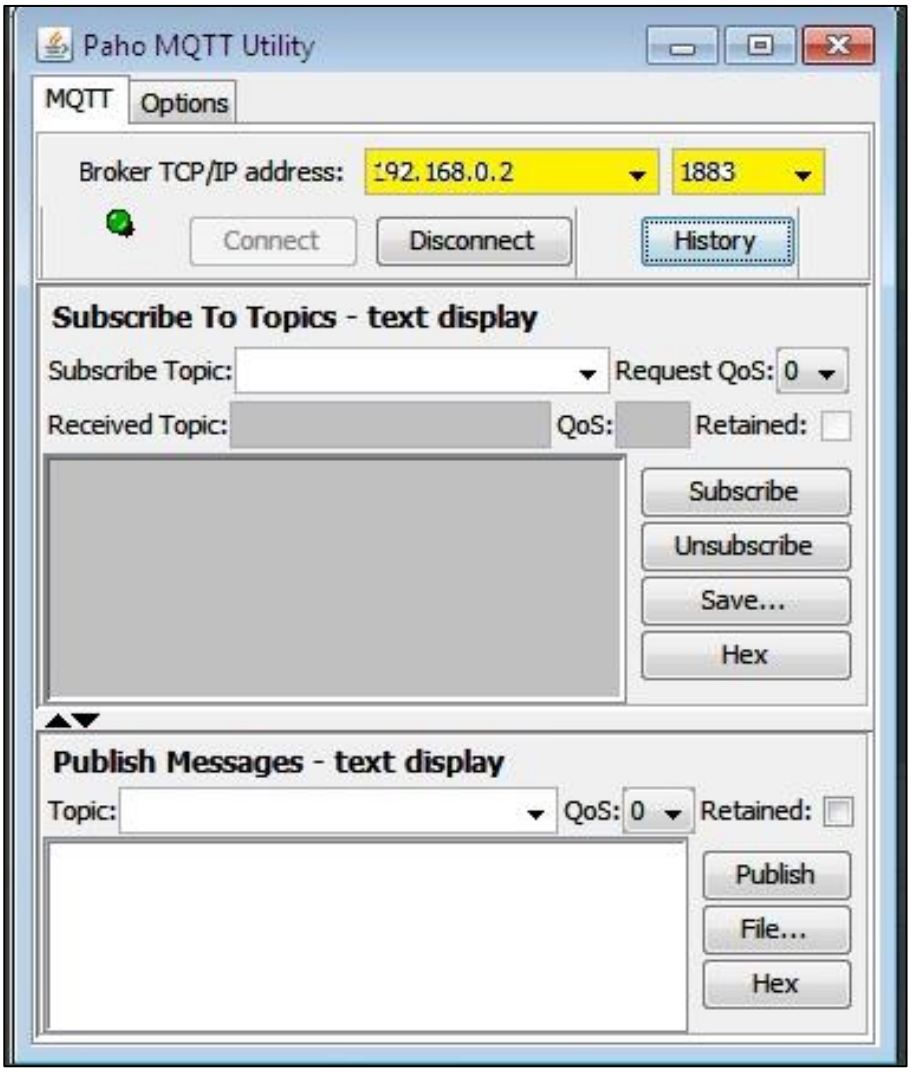
Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Install MQTT broker in Windows PC2 which is connected to Access Point through LAN.
3. Run MQTT broker in Windows PC2 using following command. Open Command prompt and go to MQTT installed folder (Ex: C:\Program Files\mosquitto) and run the following command**:**

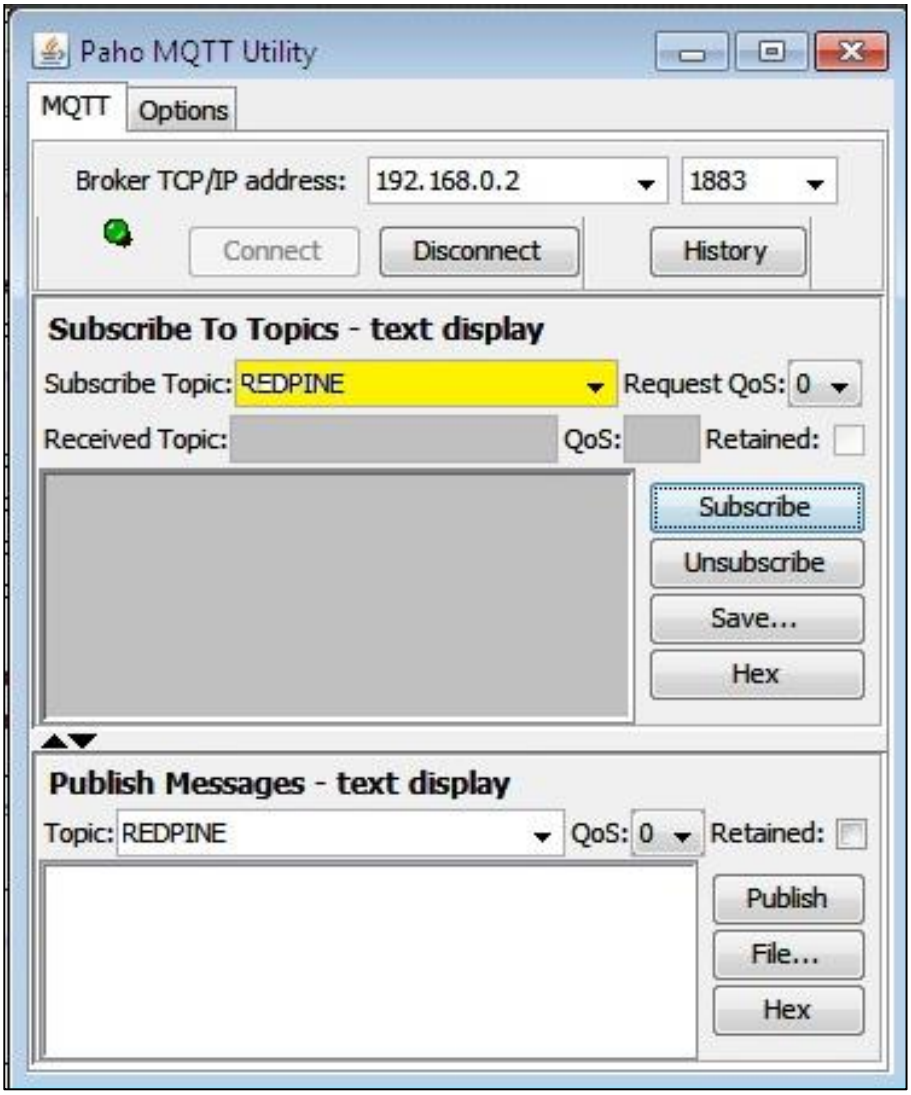
mosquito.exe –p 1883 –v



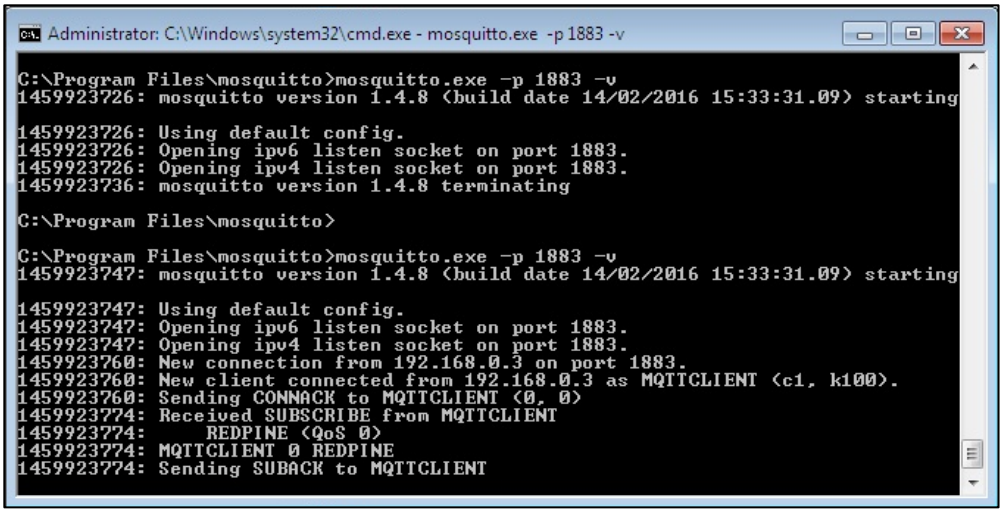
4. Open MQTT client utility in Windows PC3 and connect to MQTT broker by giving Windows PC2 IP address and MQTT broker port number in Broker TCP/IP address field.



5. After successful connection, subscribe to the topic from MQTT client utility.



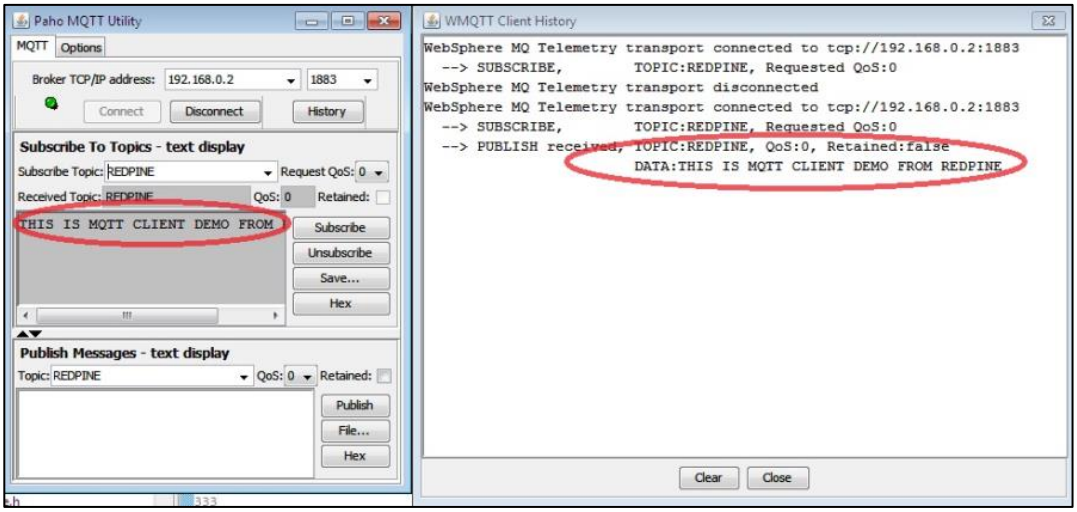
6. After the program gets executed, the Silicon Labs device will be connected to the same access point having the configuration same as that of in the application and get IP.
7. Once the device gets connected to the MQTT broker, it will subscribe to the topic **RSI_MQTT_TOPIC (Ex: "REDPINE").** The user can see the client connected and subscribe information in the MQTT broker.



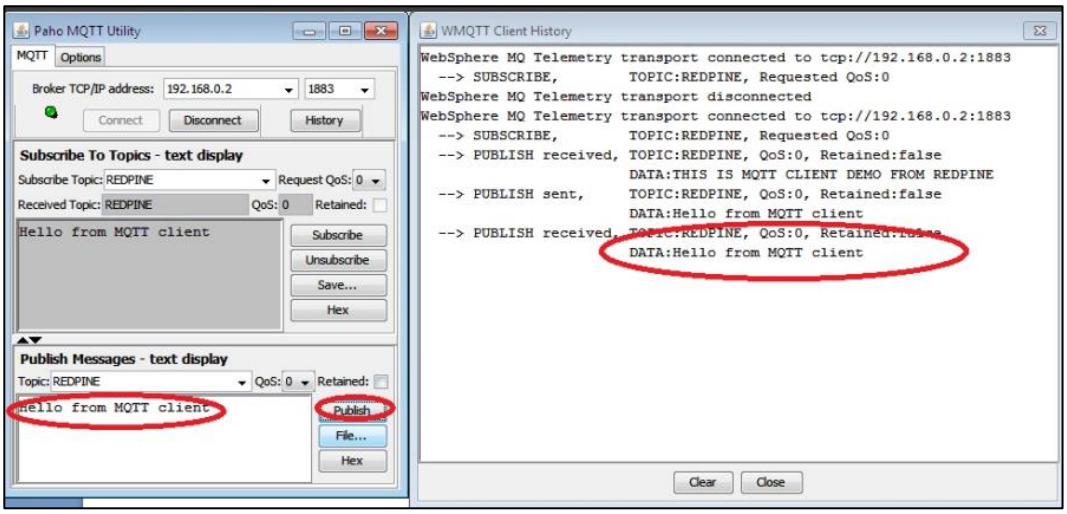
8. After successful subscription to the topic **RSI_MQTT_TOPIC (Ex: "REDPINE"),** the device publishes a message which is given in **publish_message** array (Ex: "THIS IS MQTT CLIENT DEMO FROM REDPINE") on the subscribed topic.
9. MQTT client utility which is running on Windows PC3 will receive the message published by the Silicon Labs device

as it subscribes to the same topic.

Please refer to the below image for MQTT client utility and message history.



10. Now publish a message using MQTT Utility on the same topic. Now this message is the message received by the Silicon Labs device.



Note:
Multiple MQTT client instances can be created

Limitations

MQTT client application keeps on polling for the data to receive on the subscribed topic irrespective of receive timeout mentioned in the rsi_mqtt_poll_for_recv_data API.

6.17 Multicast

Protocol Overview

In Networking, Multicast IP Routing protocols are used to distribute data (for example, audio/video streaming broadcasts) to multiple recipients. Using multicast, a source can send a single copy of data to a single multicast address, which is then distributed to an entire group of recipients.
A multicast group identifies a set of recipients that are interested in a particular data stream, and is represented by an IP address from a well-defined range. Data sent to this IP address is forwarded to all members of the multicast group. Routers between the source and recipients duplicate data packets and forward multiple copies wherever the path to

recipients diverges. Group membership information is used to calculate the best routers at which to duplicate the packets in the data stream to optimize the use of the network.

Overview

This application demonstrates how to add Silicon Labs device to a multicast group and how to send and receive multicast data on a UDP socket.
In this application, the Silicon Labs device connects to Wi-Fi access point and opens UDP socket and joins to a Multicast group ID. After successful join, application sends data to multicast group ID and receives data from Multicast group ID using opened UDP socket.

Sequence of Events

This Application explains user how to:

- Configure Silicon Labs device as a Wi-Fi station
- Connect to Wi-Fi Access Point
- Open UDP socket
- Join multicast group ID
- Send UDP data to multicast group ID
- Receive UDP data coming from Multicast group

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect using either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Silicon Labs Module
- WLAN Access point
- Windows PC2 with iperf to send and receive Multicast data

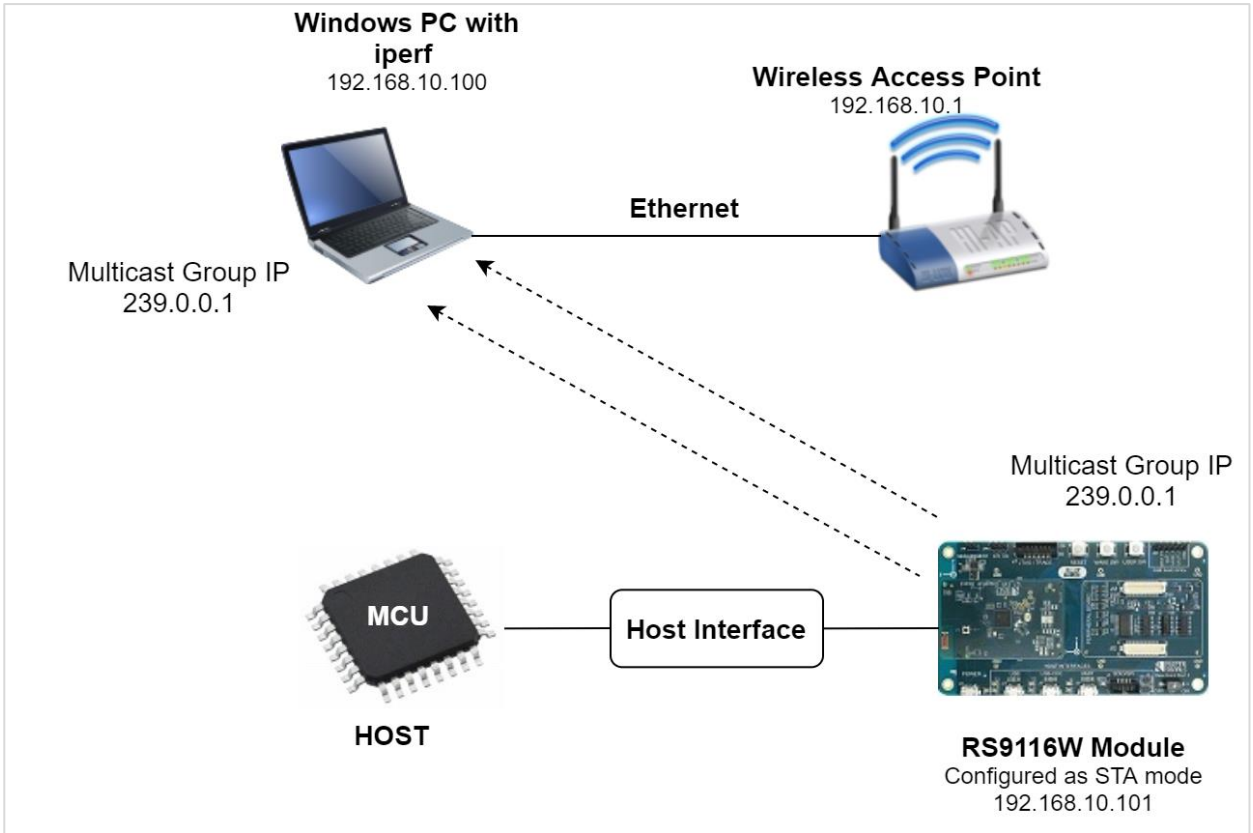


Figure 90: Setup Diagram for Multicast Example

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_multicast_app.c** file and update/modify following macros

   SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel to be scanned, If it is 0,device will scan all the channels

```
#define CHANNEL_NO                          0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                       RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                 "<psk>"
```

   DEVICE_PORT port refers internal UDP port number

```
#define DEVICE_PORT                          5001
```

   SERVER_PORT port refers remote UDP server port number

```
#define SERVER_PORT                         5002
```

   MULTICAST_GROUP_ADDRESS refers the device to which multicast group address has to join.
   MULTICAST_GROUP_ADDRESS address should be configured in long format and in little endian byte order.
   Example: To configure "239.0.0.1" as multicast group IP address, update the macro **MULTICAST_GROUP_ADRESS** as **0x010000EF.**

```
#define MULTICAST_GROUP_ADDRESS             0x010000EF
```

   NUMEBR_OF_PACKETS refer to how many packets to send/receive to/from multicast group before leaving multicast group.

```
#define NUMBER_OF_PACKETS                   <no of packets>
```

   RECV_BUFFER_SIZE is expected size of data in each packet. If packet is half the size of receive buffer, then Device will read for the data again.

```
#define RECV_BUFFER_SIZE                   <receive buf size>
```

To configure IP address

DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                       1
```

> **Note:**
>
> If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
> (Or)
> If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

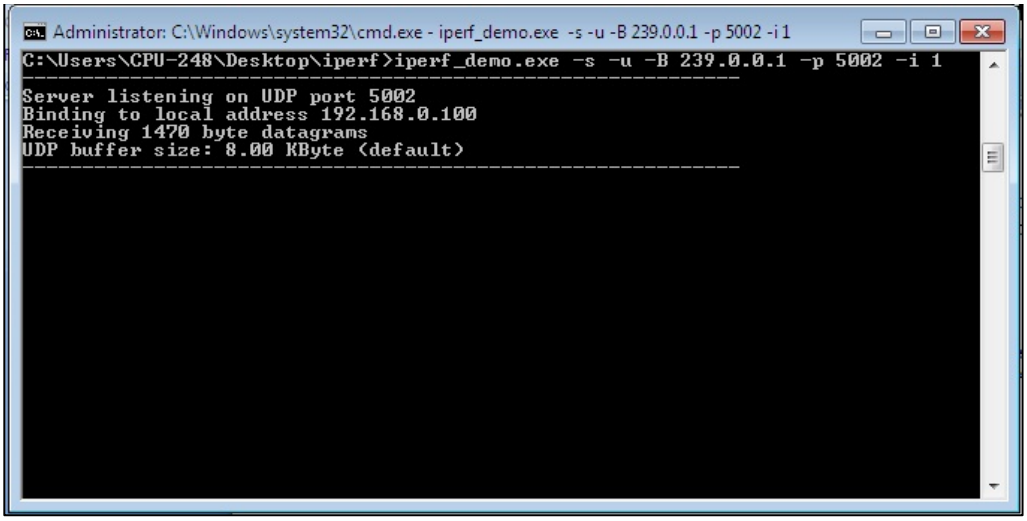
```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

> **Note:**
> **rsi_wlan_config.h** file is already set with desired configuration in respective example folders, user need not change for each example.

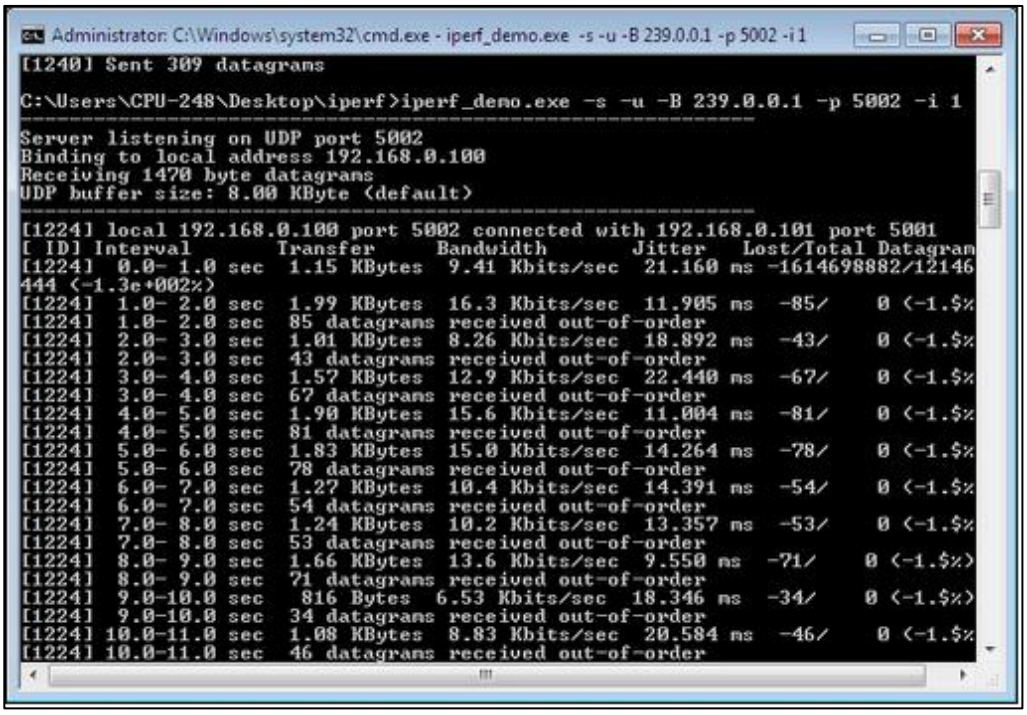
Executing the Application

1. Configure the access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Open multicast UDP server socket on port number **SERVER_PORT(Ex:5002)** by binding to **MULTICAST_GROUP_ADRESS(Ex:239.0.0.1)** using iperf application in Windows PC2 which is connected to access point through LAN.

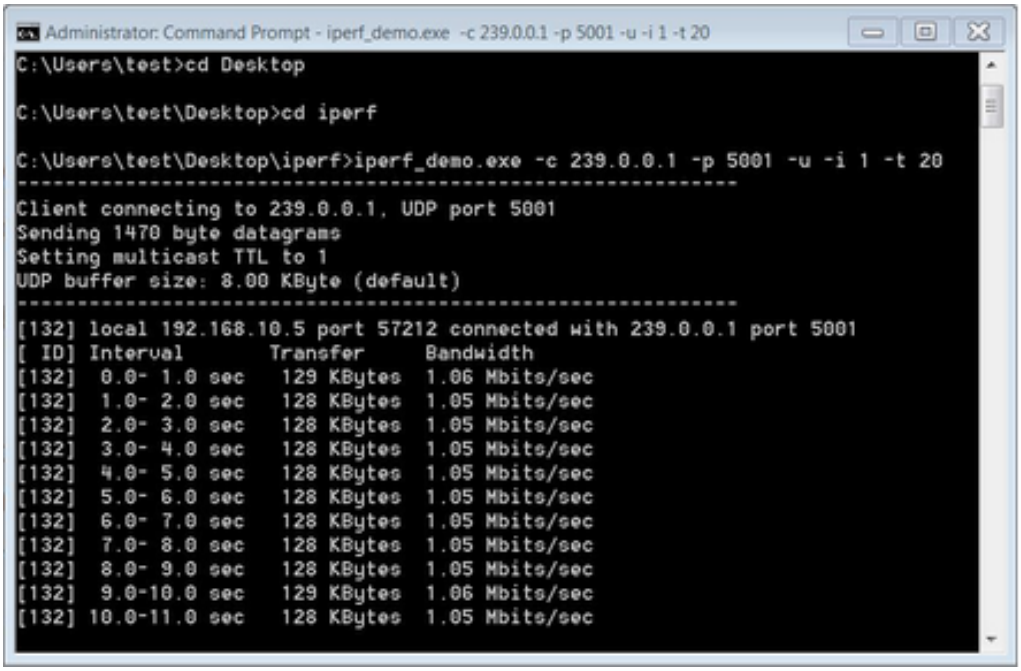
iperf_demo.exe -s –u –B 239.0.0.1 -p 5002 -i 1



3. After the program gets executed, the Silicon Labs device will be connected to the access point and get IP.
4. After successful connection with access point, the device will join to the multicast group and sends configured number of UDP packets to multicast group address on port number **SERVER_PORT.** After the device starts sending multicast data, user can see UDP receiving data on opened UDP socket on port number **SERVER_PORT**.



5. After sending configured NUMBER of UDP packets from device, remote Windows PC2 stops receiving data on SERVER_PORT and the device waits for receiving multicast data on UDP port number **DEVICE_PORT.**
6. From Windows PC2, after UDP data reception stops, open UDP client socket and send UDP data to multicast IP address with port number **DEVICE_PORT** by giving following command in **iperf, iperf_demo.exe -c239.0.0.1 -p <DEVICE_PORT> -u -i 1–t 100 –T32.**



7. The device will read configured number of packets which are coming from joined multicast group address and leave from that joined multicast group.

6.18 OTAF

Overview

This application demonstrates how to upgrade new firmware to Silicon Labs device using remote TCP server.

In this application Silicon Labs device connects to Access Point and using OTAF command establishes TCP client connection with TCP server opened on remote peer. After successful TCP connection, module sends the firmware file request to remote TCP server and server responds with Firmware file and waits for the next firmware file request. Once firmware file receives from the TCP server, Module loads the firmware file into on to the modules flash. After successful firmware upgrade, OTAF API returns success response.

Sequence of Events

This Application explains user how to:

- Configure as station mode
- Open TCP server socket at Access Point
- Connect to Access Point
- Call OTA firmware upgrade api to request Firmware file from remote server
- Send firmware file from remote server.

Example Setup

The Silicon Labs device parts require that the host processor should be connected to the device either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Wireless Access point
- Silicon Labs module
- Linux PC with TCP server application (TCP server application providing as part of release package)

Note:

TCP server application providing in release package in the following path:
sapis/examples/wlan/otaf/firmware_upgarde_ota_server.c

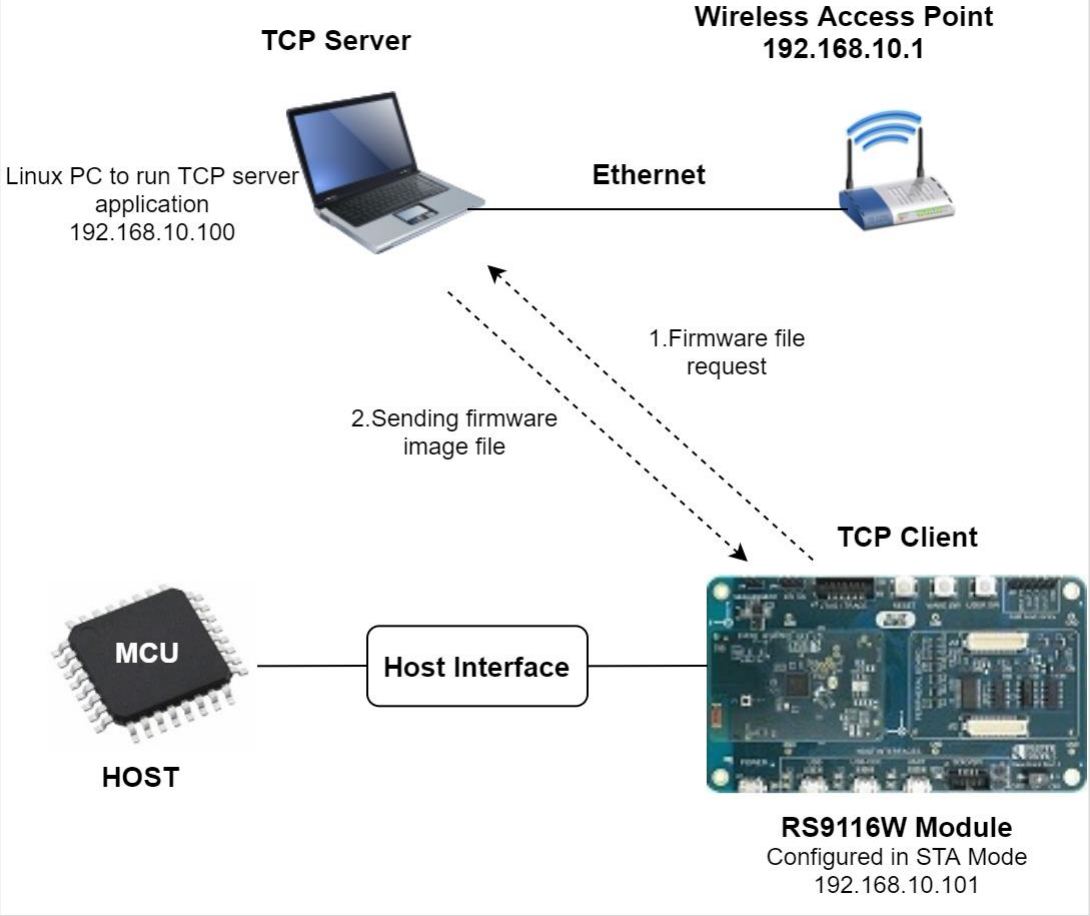


Figure 91: Setup Diagram for Over The Air Firmware Upgradation from Server

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ota_firmware_upgradation_app.c*** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     "<psk>"
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                         5001
```

SERVER_PORT port refers remote TCP server port number which is opened in Linux PC.

```
#define SERVER_PORT                         5001
```

SERVER_IP_ADDRESS refers remote peer IP (Linux PC) address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS                   0x6400A8C0
```

RECV_BUFFER_SIZE refers Memory for receive data

```
#define RECV_BUFFER_SIZE                    1027
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                           1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

The IP address needs to be configuring to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                           0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                             0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order.
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                             0x00FFFFFF
```

OTAF_SERVER_PORT refers remote TCP server port number which is opened in Linux PC.

```
#define OTAF_SERVER_PORT                    5001
```

OTAF_RX_TIMEOUT refers remote TCP RX packet receive timeout .

```
#define OTAF_RX_TIMEOUT                          200
```

OTAF_TCP_RETRY_COUNT refers to TCP maximum retransmissions count.

```
#define OTAF_TCP_RETRY_COUNT                     20
```

OTAF_RETRY_COUNT refers to OTAF upgradation retry count.

```
#define OTAF_RETRY_COUNT                         10
```

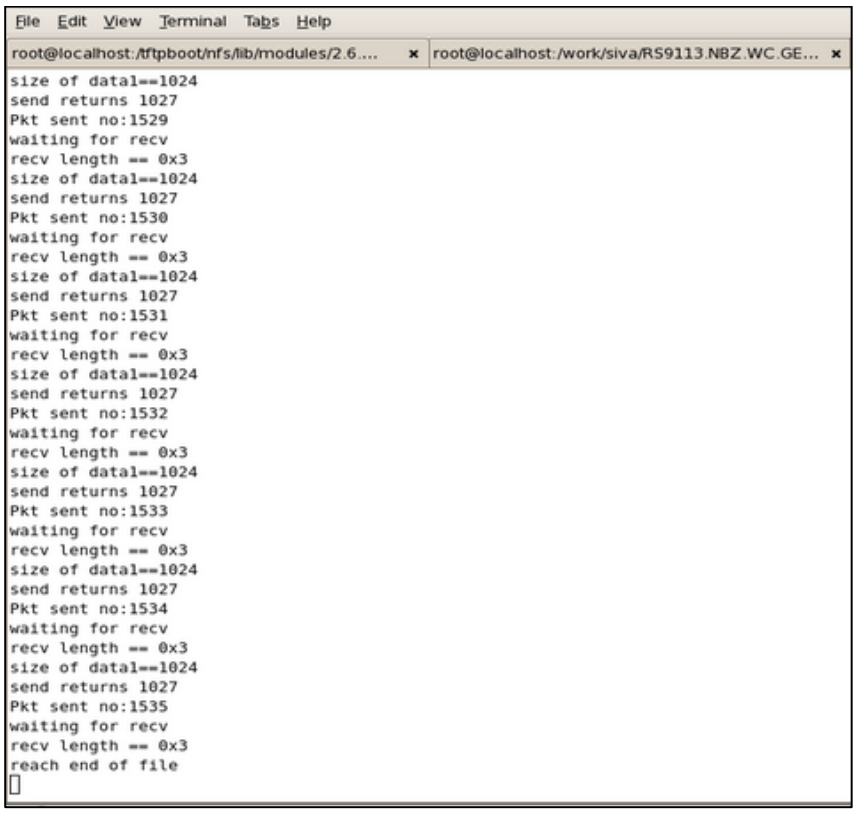
2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                          DISABLE
#define RSI_FEATURE_BIT_MAP                      FEAT_SECURITY_PSK
#define RSI_TCP_IP_BYPASS                        RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP               (TCP_IP_FEAT_OTAF | TCP_IP_FEAT_DHCPV4_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP               FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP           EXT_FEAT_256k_MODE
#define RSI_BAND                                 RSI_BAND_2P4GHZ
```

> **Note: rsi_wlan_config.h** file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Compile and run by providing port number and Firmware file path from the server application from the path: "***/otaf/firmware_upgarde_ota_server.c "***.
   gcc firmware_upgarde_ota_server.c
   ./a.out 5001 RS9116.NBZ.WC.GEN.OSI.x.x.x.rps
3. After the program gets executed, device connects to AP and open TCP client socket.
4. After TCP connection established with remote server, application sends firmware file request to the server.
5. Server receive request and sends Firmware file in chunks.
6. After receiving chunk from remote server, application again sends firmware request to server. Server will wait for the firmware request from WiSeConnect device before sending next chunk.
7. Packet is sent to the device in chunks as shown in the given below figure. After successful upgradation in TCP server terminal shows "reach end of file".



Note:

After Firmware upgradation, Device needs to be reboot to get effective of new firmware file. After reboot, device will take few minutes to give CARD READY indication after first reboot.

6.19 Power Save Deep Sleep

Overview

This is a sample application demonstrating how to enable power save deep sleep profile with WiseConnectTM module. This application enables power mode 8 and then wait in a scheduler for some time. Once it will come out of delay, it will connect to configured AP and then open UDP client socket. It then sends some packet to the UDP server and then disconnect from AP and goes back to deep sleep.

Sequence of Events

This Application explains user how to:

• Create device as a station

• Enable power mode 8 and then wait in a scheduler for some time.

• Once it will come out of delay, connect to configured AP

• Open UDP client socket.

• Sends some packet to the UDP server and then disconnect from AP and goes back to deep sleep.

Application Setup

The WiSeConnect in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

SPI based Setup Requirements

• Windows PC with KEIL IDE

• STM32 micro controller

Note:

If user does not have STM32 host platform, please go through the RS9116W SAPI Porting Guide at https://docs.silabs.com/rs9116 guide for SAPIs porting to that particular platform.

- WiSeConnect device
- WiFi Access point
- Windows PC2 with UDP server application (iperf)
- Agilent power analyzer

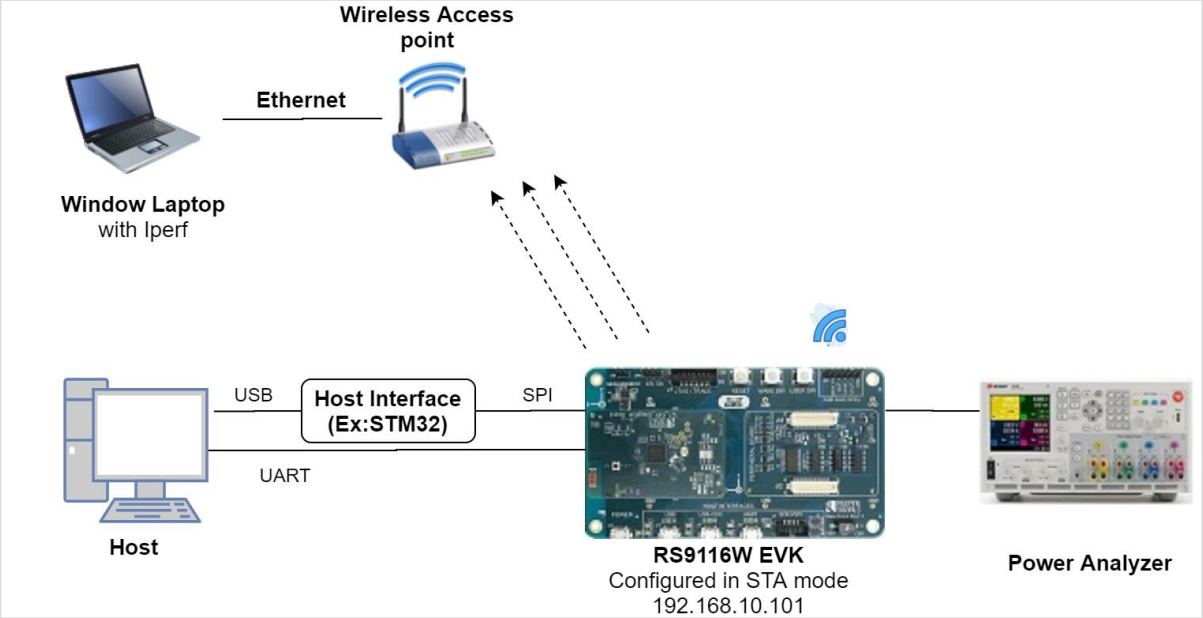


Figure 92: Setup Diagram for WLAN Power Save

Configuration and Steps for Execution

The example application is available in the Release at {Release $}/host/sapis/examples. These examples will have to be initialized, configured and executed to test the application.

The initialization varies based on the interface but configuration and execution are the common.

Initializing the Application

SPI Interface

If User is using SPI interface, please refer to the document /host/platforms/STM32/Readme_STM32_Nucleo_F411RE for opening the power_save example in KEIL IDE.

UART Interface

If User using UART interface, please refer to the document /host/platforms/STM32/Readme_STM32_Nucleo_F411RE for opening the power_save example in KEIL IDE.

Configuring the Application

1.Open **sapis/examples/wlan/power_save/rsi_wlan_power_save_profile.c** file and update/modify following macros

SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities. Valid configuration is RSI_OPEN - For OPEN security mode, RSI_WPA - For WPA security mode and RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                          RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define  PSK                                   "<psk>"
```

SERVER_PORT port refers remote UDP server port number which is opened in Windows PC2.

```
#define  SERVER_PORT                           5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket. IP address should be in long format and in little endian byte order.

Example: To configure “192.168.10.100” as IP address, update the macro DEVICE_IP as 0x640AA8C0.

```
#define  SERVER_IP_ADDRESS                     0x640AA8C0
```

NUMEBR_OF_PACKETS refers how many packets to send from device to remote UDP server.

```
#define   NUMBER_OF_PACKETS                     <no of packets>
```

Application memory length which is required by the driver.

```
#define   GLOBAL_BUFF_LEN                       10000
```

To configure IP address DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define   DHCP_MODE                             1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure “192.168.10.10” as IP address, update the macro DEVICE_IP as 0x0A0AA8C0.

```
#define   DEVICE_IP                             0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order

Example: To configure “192.168.10.1” as Gateway, update the macro GATEWAY as 0x010AA8C0

```
#define   GATEWAY                              0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order.

Example: To configure “255.255.255.0” as network mask, update the macro NETMASK as 0x00FFFFFF

```
#define   NETMASK                              0x00FFFFFF
```

In this application, default power save mode configuration is set to low power mode 8 (RSI_SLEEP_MODE_8) with maximum power save (RSI_MAX_PSP) with message based handshake.

```
#define   PSP_MODE                            RSI_SLEEP_MODE_8
#define   PSP_TYPE                             RSI_MAX_PSP
```

2. Open sapis/include/rsi_wlan_config.h file and update/modify following macros,

```
#define   CONCURRENT_MODE                RSI_DISABLE
#define   RSI_FEATURE_BIT_MAP           FEAT_SECURITY_OPEN
#define   RSI_TCP_IP_BYPASS            RSI_DISABLE
#define    RSI_TCP_IP_FEATURE_BIT_MAP     TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP       FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP   EXT_FEAT_256K_MODE
#define   RSI_BAND                   RSI_BAND_2P4GHZ
```

Default configuration of low power save mode 2

```
#define   RSI_HAND_SHAKE_TYPE             MSG_BASED
#define   RSI_SELECT_LP_OR_ULP_MODE       RSI_LP_MODE
#define   RSI_DTIM_ALIGNED_TYPE           0
#define    RSI_MONITOR_INTERVAL            50
#define    RSI_WMM_PS_ENABLE               RSI_DISABLE
#define   RSI_WMM_PS_TYPE                 0
#define   RSI_WMM_PS_WAKE_INTERVAL        20
#define   RSI_WMM_PS_UAPSD_BITMAP         15
```

3. If user wants to select different power save mode profiles, please go through the step #4 and #5 otherwise skip step #4 and #5

4. Open sapis/examples/wlan/power_save/rsi_wlan_power_save_profile.c file and update/modify following macros, PSP_MODE refers power save profile mode. WiSeConnect device supports following power modes:

RSI_ACTIVE (0): In this mode, module is active and power save is disabled.

RSI_SLEEP_MODE_1 (1): In this power mode, module goes to power save after association with the Access Point. In this sleep mode, SoC will never turn off, therefore no handshake is required before sending data to the module.

RSI_SLEEP_MODE_2 (1): In this power mode, module goes to power save after association with the Access Point. In this sleep mode, SoC will go to sleep based on GPIO handshake or Message exchange, therefore handshake is required before sending data to

the module.

RSI_SLEEP_MODE_8 (8): In this power mode, module goes to power save when it is in unassociated state with the Access Point. In this sleep mode, SoC will go to sleep based on GPIO handshake or Message exchange, therefore handshake is required before sending

the command to the module.

```
#define   PSP_MODE            RSI_SLEEP_MODE_8
```

WiSeConnect device supports following power save modes:

RSI_CONNECTED_GPIO_BASED_PS = 2

RSI_CONNECTED_MSG_BASED_PS = 3

RSI_GPIO_BASED_DEEP_SLEEP = 8

RSI_MSG_BASED_DEEP_SLEEP = 9

Among the above mentioned four power save modes, which power save mode must be selected will depend upon the two macros i.e. PSP_MODE and RSI_HAND_SHAKE_TYPE selection.

RSI_HAND_SHAKE_TYPE	**PSP_MODE**	
	RSI_SLEEP_MODE_2(Connected_SLEEP)	RSI_SLEEP_MODE_8(Deep_SLEEP)
GPIO_BASED	RSI_CONNECTED_GPIO_BASED_PS = 2	RSI_GPIO_BASED_DEEP_SLEEP = 8
MSG_BASED	RSI_CONNECTED_MSG_BASED_PS = 3	RSI_MSG_BASED_DEEP_SLEEP = 9

Note1: For RSI_SLEEP_MODE_2 and RSI_SLEEP_MODE_8 modes, GPIO or Message based handshake can be selected using RSI_HAND_SHAKE_TYPE macro which is define in

sapis/examples/wlan/power_save/rsi_wlan_config.h

Note2: In this example user can verify RSI_SLEEP_MODE_2 with Message based handshake. If user wants to verify other power modes, user has to change the application as well as GPIO handshake signals.

PSP_TYPE refers power save profile type. WiSeConnect device supports following power save profile types:

RSI_MAX_PSP (0): In this mode, WiSeConnect device will be in Maximum power save mode. i.e Device will wake up for every DTIM beacon and do data Tx and Rx. RSI_FAST_PSP (1): In this mode, WiSeConnect device will disable power save for any Tx/Rx packet for monitor interval of time (monitor interval can be set through macro in sapis/examples/wlan/power_save/rsi_wlan_config.h file, default value is 50 ms).If there is no data for monitor interval of time then module will again enable power save. RSI_UAPSD (2): This PSP_TYPE is used to enable WMM power save.

```
#define   PSP_TYPE          RSI_MAX_PSP
```

Note1: PSP_TYPE is valid only when PSP_MODE set to RSI_SLEEP_MODE_1 or RSI_SLEEP_MODE_2 mode.

Note2: RSI_UAPSD power profile type in PSP_TYPE is valid only when RSI_WMM_PS_ENABLE is enabled in sapis/include/rsi_wlan_config.h file.

5. Open sapis/examples/wlan/power_save/rsi_wlan_config.h file and update/modify following macros, RSI_HAND_SHAKE_TYPE is used to select GPIO or Message based hand shake in RSI_SLEEP_MODE_2 and RSI_SLEEP_MODE_8 modes.

```
#define   RSI_HAND_SHAKE_TYPE      MSG_BASED
```

RSI_SELECT_LP_OR_ULP_MODE is used to select low power mode or ultra-low power mode. Valid configurations are, RSI_LP_MODE or RSI_ULP_WITH_RAM_RET or RSI_ULP_WITHOUT_RAM_RET

RSI_LP_MODE: In this module will be in Low power mode.

RSI_ULP_WITH_RAM_RET: In this module will be in Ultra low power mode and it will remember the previous state after issuing power save mode command.

RSI_ULP_WITHOUT_RAM_RET: In this module will be in Ultra low power mode and it will not remember the previous state after issuing power save mode command. After wakeup, module will give CARD READY indication and user has to issue commands from wireless initialization.

```
#define   RSI_SELECT_LP_OR_ULP_MODE        RSI_LP_MODE
```

RSI_DTIM_ALIGNED_TYPE refers whether module has to wake up at normal beacon or DTIM beacon which is just before listen interval. If RSI_DTIM_ALIGNED_TYPE is set to 0(Zero) i.e module will wake up at normal beacon which is just before listen interval

If RSI_DTIM_ALIGNED_TYPE is set to 1(Zero) i.e. module will wake up at DTIM beacon which is just before listen interval

```
#define    RSI_DTIM_ALIGNED_TYPE            0
```

RSI_MONITOR_INTERVAL refers amount of time (in ms) to wait for Tx or Rx before giving power save indication to connected Access Point.

```
#define    RSI_MONITOR_INTERVAL             50
```

Note:

RSI_MONITOR_INTERVAL is applicable only when PSP_TYPE selected as RSI_FAST_PSP

RSI_WMM_PS_ENABLE is used to enable or disable WMM power save.

```
#define     RSI_WMM_PS_ENABLE           0
```

RSI_WMM_PS_TYPE is used to set Tx based or Periodic based WMM power save. Update RSI_WMM_PS_TYPE macro with 0 for Tx Based or 1 for periodic based WMM power save.

```
#define     RSI_WMM_PS_TYPE          0
```

RSI_WMM_PS_WAKE_INTERVAL refers at periodic time (in ms) module has to wake up module when RSI_WMM_PS_TYPE selected as Periodic.

```
#define     RSI_WMM_PS_WAKE_INTERVAL             20
```

RSI_WMM_PS_UAPSD_BITMAP refers UAPSD bitmap

```
#define     RSI_WMM_PS_UAPSD_BITMAP            15
```

Note:

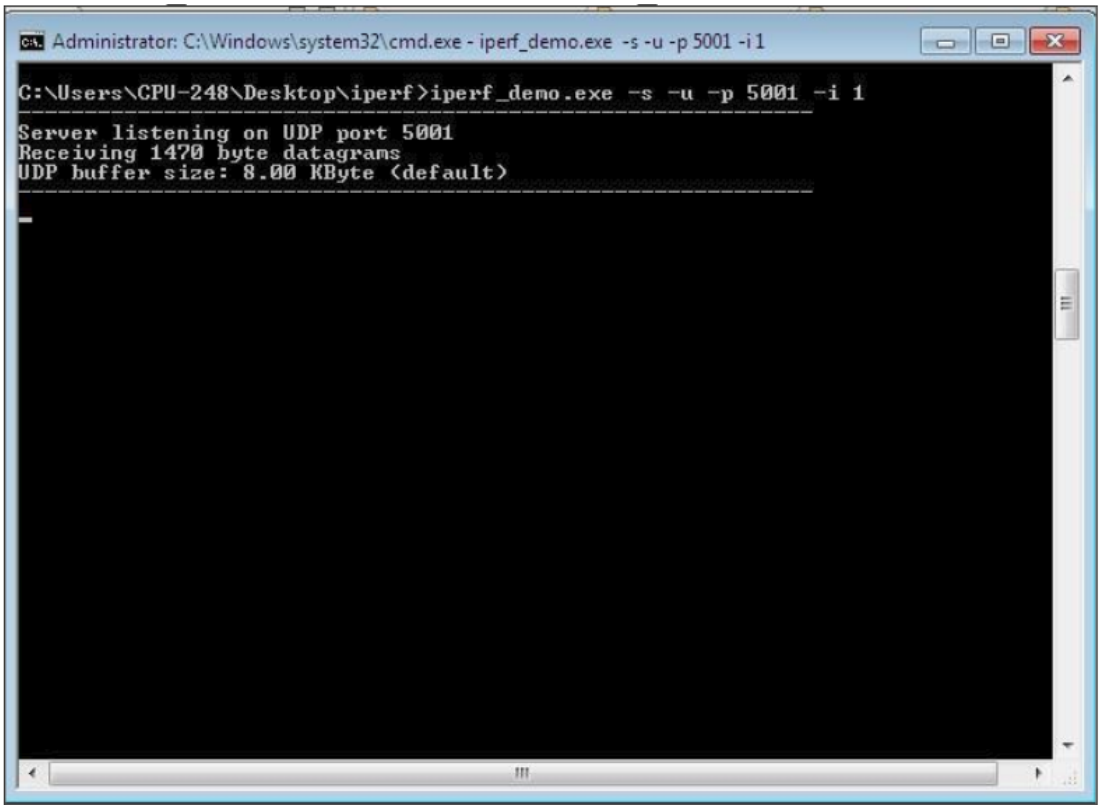
If RSI_WMM_PS_ENABLE is enabled, then user has to set PSP_TYPE to RSI_UAPSD in order to work WMM power save

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect WiSeConnect device in STA mode.

2. Open UDP server application using iperf application in Windows PC2 which is connected to Access point through LAN.

iperf_demo.exe –s -u -p <SERVER_PORT> -i 1



3. SPI Interface

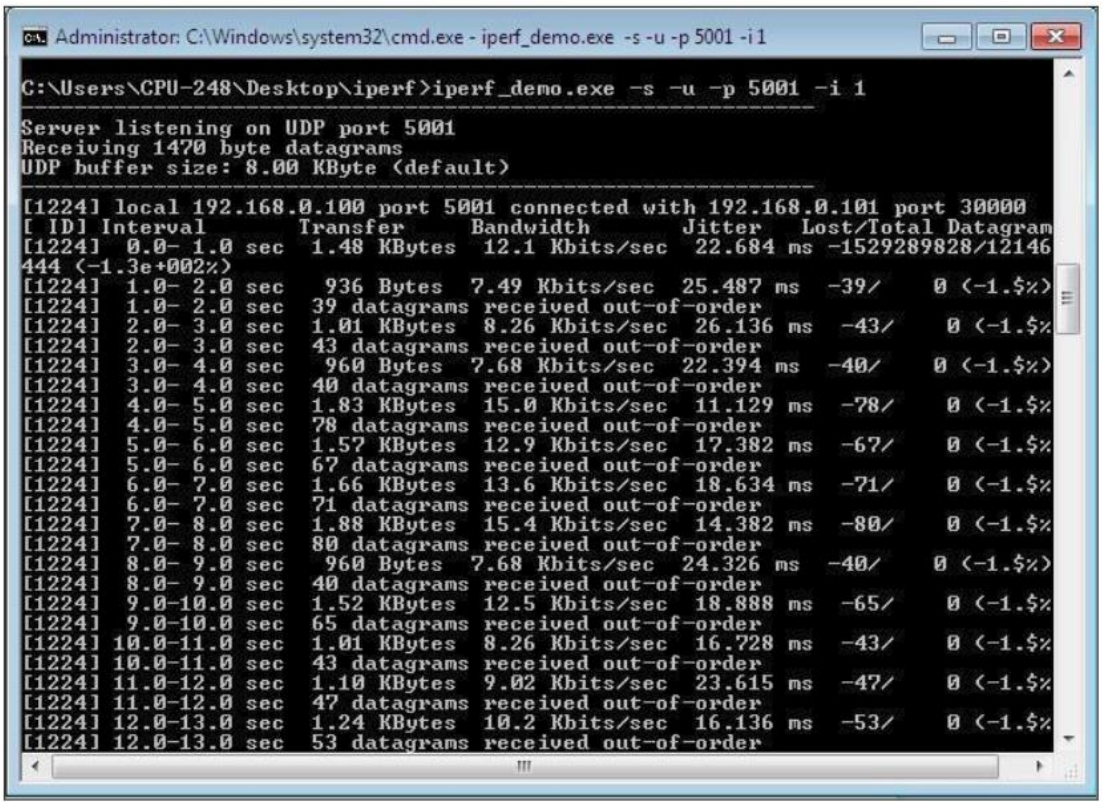
If User is using SPI interface, please refer to the document /host/platforms/STM32/Readme_STM32_Nucleo_F411RE for executing the power_save example in KEIL IDE.

4.UART Interface

If User is using UART interface, please refer to the document /host/platforms/STM32/Readme_STM32_Nucleo_F411RE for executing the power_save example in KEIL IDE

5. After program gets executed, WiSeConnect Device will go to sleep based on the selected power mode and wakes up after deep sleep timeout (Default deep sleep time is 3sec in RSI_SLEEP_MODE_8 with message based handshake). Please refer the given below image for power save cycle for default deep sleep time.

6.After successful wake up from deep sleep, WiSeConenct device connects to AP and sends configured number of (NUMBER_OF_PACKETS) UDP packets to remote peer which is connected to Access point through LAN. Please refer the given below image for reception of UDP data on UDP server.



7.After sending configured number of packets, WiSeConnect device disconnects from connected AP and again repeat the steps from #10 to #11 (Again it will go to sleep and wakes up after time out and connects to AP and sends configured number of packets). Please find below image for power save profile cycle.

Figure 93 Deep sleep and wakeup power save profile

6.20 Power Save Standby Associated

Overview

The application demonstrates the process of configuring the device in power save profile mode 2 after successful connection with Access point in station mode and provides the steps to send UDP data from RS9116W device to remote peer in the configured power save mode.
In this application, RS911W EVK connects to Access Point, configures to Power save profile mode2 and transfers data using UDP.

Sequence of Events

This Application explains user how to:

- Create RS911W EVK as a WLAN station.
- Connect RS911W EVK to Access point.
- Configure device in Power Save profile mode 2.
- Open UDP client socket in device.
- Send UDP data from RS911W EVK to remote peer.
- Analyze power save profile while it is in Associated state and while data transfer.

Example Setup

The RS9116W device requires the host processor to be connected to it using either SPI or UART or USB interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- RS9116W EVK
- Wi-Fi Access point
- Windows PC2 with UDP server application (iperf)
- Agilent power analyzer

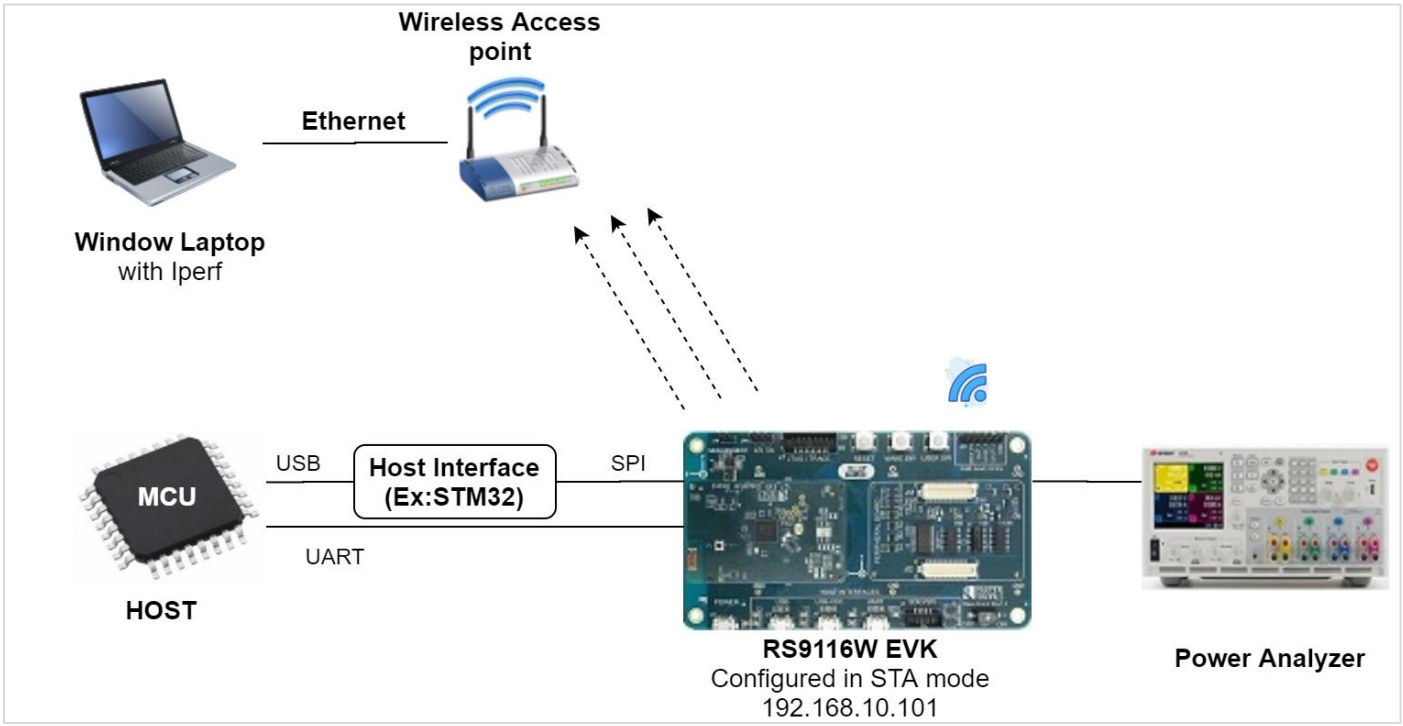


Figure 94: Setup Diagram for Power Save Standby Example

Configuration and Steps for Execution

Configuring the Application

> **Note**:
>
> If user wants to transfer data, **ENABLE_DATA_TRANSFER_DEMO** macro should be enabled in compiler options.

1. Open **rsi_wlan_connected_sleep_app.c** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                "<ap name>"
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configurations are:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                       RSI_OPEN
```

PSK refers to the secret key if the Access point is configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                             "<psk>"
```

SERVER_PORT refers to the remote UDP server port number which is opened in Windows PC2.

```
#define SERVER_PORT                     <remote port>
```

SERVER_IP_ADDRESS refers to the remote peer IP address to connect with UDP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as IP address, update the macro **DEVICE_IP** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS               0x640AA8C0
```

NUMEBR_OF_PACKETS refers to the number of packets to be sent from device to remote UDP server.

```
#define NUMBER_OF_PACKETS               <no of packets>
```

GLOBAL_BUFF_LEN refers to the application memory length which is required by the driver.

```
#define GLOBAL_BUFF_LEN                 15000
```

DHCP_MODE refers to the mode of configuring the IP address, that is whether through DHCP or STATIC.

```
#define DHCP_MODE                       1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address which is to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

In this application, default power save mode configuration is set to low power mode 2 (RSI_SLEEP_MODE_2) with maximum power save (RSI_MAX_PSP) and with message based handshake.

```
#define PSP_MODE                          RSI_SLEEP_MODE_2
#define PSP_TYPE                          RSI_MAX_PSP
```

2. Open **rsi_wlan_config.h** file and update/modify following macros:

```
#define CONCURRENT_MODE                   RSI_DISABLE
#define RSI_FEATURE_BIT_MAP               FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                 RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP        TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP        0
#define RSI_BAND                          RSI_BAND_2P4GHZ
```

Default configuration of low power save mode 2

```
#define RSI_HAND_SHAKE_TYPE               MSG_BASED
#define RSI_SELECT_LP_OR_ULP_MODE         RSI_ULP_MODE
#define RSI_DTIM_ALIGNED_TYPE             0
#define RSI_MONITOR_INTERVAL              50
#define RSI_WMM_PS_ENABLE                 RSI_DISABLE
#define RSI_WMM_PS_TYPE                   0
#define RSI_WMM_PS_WAKE_INTERVAL          20
#define RSI_WMM_PS_UAPSD_BITMAP           15
```

3. If user wants to select different power save mode profiles, please go through the step #4 and #5, otherwise skip step #4 and #5.

4. Open ***rsi_wlan_connected_sleep_app.c*** file and update / modify the following macros.

PSP_MODE refers to the power save profile mode. RS916W EVK supports the following power modes:
RSI_ACTIVE (0): In this mode, module is active and power save is disabled.
RSI_SLEEP_MODE_1 (1): In this mode, module goes to power save after association with the Access Point. In this sleep mode, SoC will never turn off, therefore no handshake is required before sending data to the module.
RSI_SLEEP_MODE_2 (2): In this mode, module goes to power save after association with the Access Point. In this sleep mode, SoC will go to sleep based on GPIO hand shake or Message exchange, therefore handshake is required before sending data to the module.
RSI_SLEEP_MODE_8 (8): In this mode, module goes to power save when it is not in associated state with the Access Point. In this sleep mode, SoC will go to sleep based on GPIO handshake or Message exchange, therefore handshake is required before sending the command to the module.

```
#define PSP_MODE                          RSI_SLEEP_MODE_2
```

Note:

For RSI_SLEEP_MODE_2 and RSI_SLEEP_MODE_8 modes, GPIO or Message based handshake can be selected using RSI_HAND_SHAKE_TYPE macro which is defined in rsi_wlan_config.h.

Note:

In this example user can verify RSI_SLEEP_MODE_2 with Message based handshake. If user wants to verify other power modes, change the application as well as GPIO handshake signals.

PSP_TYPE refers to power save profile type. RS9116W EVK supports following power save profile types:

RSI_MAX_PSP (0): In this mode, RS9116W EVK will be in Maximum power save mode. i.e device will wake up for every DTIM beacon and do data Tx and Rx.
RSI_FAST_PSP (1): In this mode, RS9116W EVK will disable power save for any Tx/Rx packet for monitor interval of time (monitor interval can be set through macro in ***rsi_wlan_config.h*** file, default value is 50 ms). If there is no data for monitor interval of time, then module will again enable power save.
RSI_UAPSD (2): This **PSP_TYPE** is used to enable WMM power save.

```
#define PSP_TYPE                        RSI_MAX_PSP
```

Note1:
PSP_TYPE is valid only when PSP_MODE is set to RSI_SLEEP_MODE_1 or RSI_SLEEP_MODE_2 mode.

Note2:
RSI_UAPSD power profile type in PSP_TYPE is valid only when RSI_WMM_PS_ENABLE is enabled in rsi_wlan_config.h file.

5. Open **rsi_wlan_config.h** file and update/modify following macros,

   data transfer happens only if ENABLE_DATA_TRANSFER MODE

```
#define RSI_HAND_SHAKE_TYPE             MSG_BASED
```

RSI_HAND_SHAKE_TYPE is used to select the hand shake type (**GPIO or Message)** in **RSI_SLEEP_MODE_2** and **RSI_SLEEP_MODE_8** modes**.**

```
#define RSI_HAND_SHAKE_TYPE             MSG_BASED
```

RSI_SELECT_LP_OR_ULP_MODE is used to select low power mode or ultra-low power mode. Valid configurations are **RSI_LP_MODE** or **RSI_ULP_WITH_RAM_RET** or **RSI_ULP_WITHOUT_RAM_RET.**
RSI_LP_MODE: In this, module will be in Low power mode.
RSI_ULP_WITH_RAM_RET: In this, module will be in Ultra low power mode and it will remember the previous state after issuing power save mode command.
RSI_ULP_WITHOUT_RAM_RET: In this, module will be in Ultra low power mode and it will not remember the previous state after issuing power save mode command. After wakeup, module will give CARD READY indication and user has to issue commands from wireless initialization.

```
#defineRSI_SELECT_LP_OR_ULP_MODE        RSI_LP_MODE
```

RSI_DTIM_ALIGNED_TYPE is used to decide whether module has to wake up at normal beacon or DTIM beacon which is just before listen interval.
If **RSI_DTIM_ALIGNED_TYPE** is set to 0(Zero), module will wake up at normal beacon which is just before listen interval.
If **RSI_DTIM_ALIGNED_TYPE** is set to 1(Zero), module will wake up at DTIM beacon which is just before listen interval.

```
#defineRSI_DTIM_ALIGNED_TYPE            0
```

RSI_MONITOR_INTERVAL refers to the amount of time (in ms) to wait for Tx or Rx before giving power save indication to the connected Access Point.

```
#defineRSI_MONITOR_INTERVAL             50
```

Note:

RSI_MONITOR_INTERVAL is applicable only when PSP_TYPE selected as RSI_FAST_PSP.

RSI_WMM_PS_ENABLE is used to enable or disable WMM power save.

```
#define   RSI_WMM_PS_ENABLE             0
```

RSI_WMM_PS_TYPE is used to set Tx based or Periodic based WMM power save.
Update RSI_WMM_PS_TYPE macro with 0 for Tx based or 1 for periodic based WMM power save.

```
#define  RSI_WMM_PS_TYPE                  0
```

RSI_WMM_PS_WAKE_INTERVAL refers to the periodic time (in ms) in which the module has to wake up when RSI_WMM_PS_TYPE is selected as Periodic.

```
#define   RSI_WMM_PS_WAKE_INTERVAL          20
```

RSI_WMM_PS_UAPSD_BITMAP refers to the UAPSD bitmap

```
#define  RSI_WMM_PS_UAPSD_BITMAP           15
```

Note:

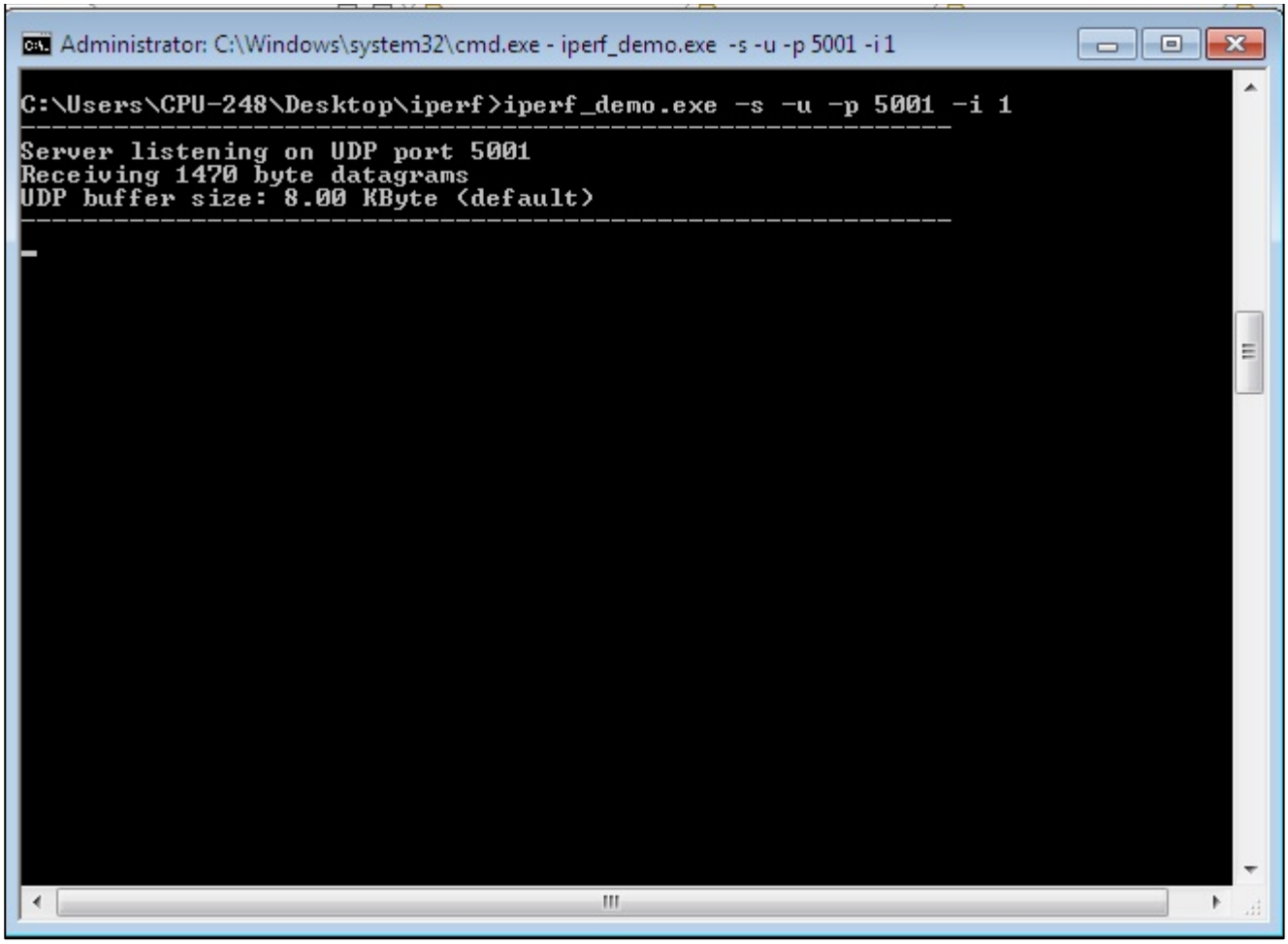
If RSI_WMM_PS_ENABLE is enabled, then user has to set PSP_TYPE to RSI_UAPSD in order to work WMM power save.

Note:

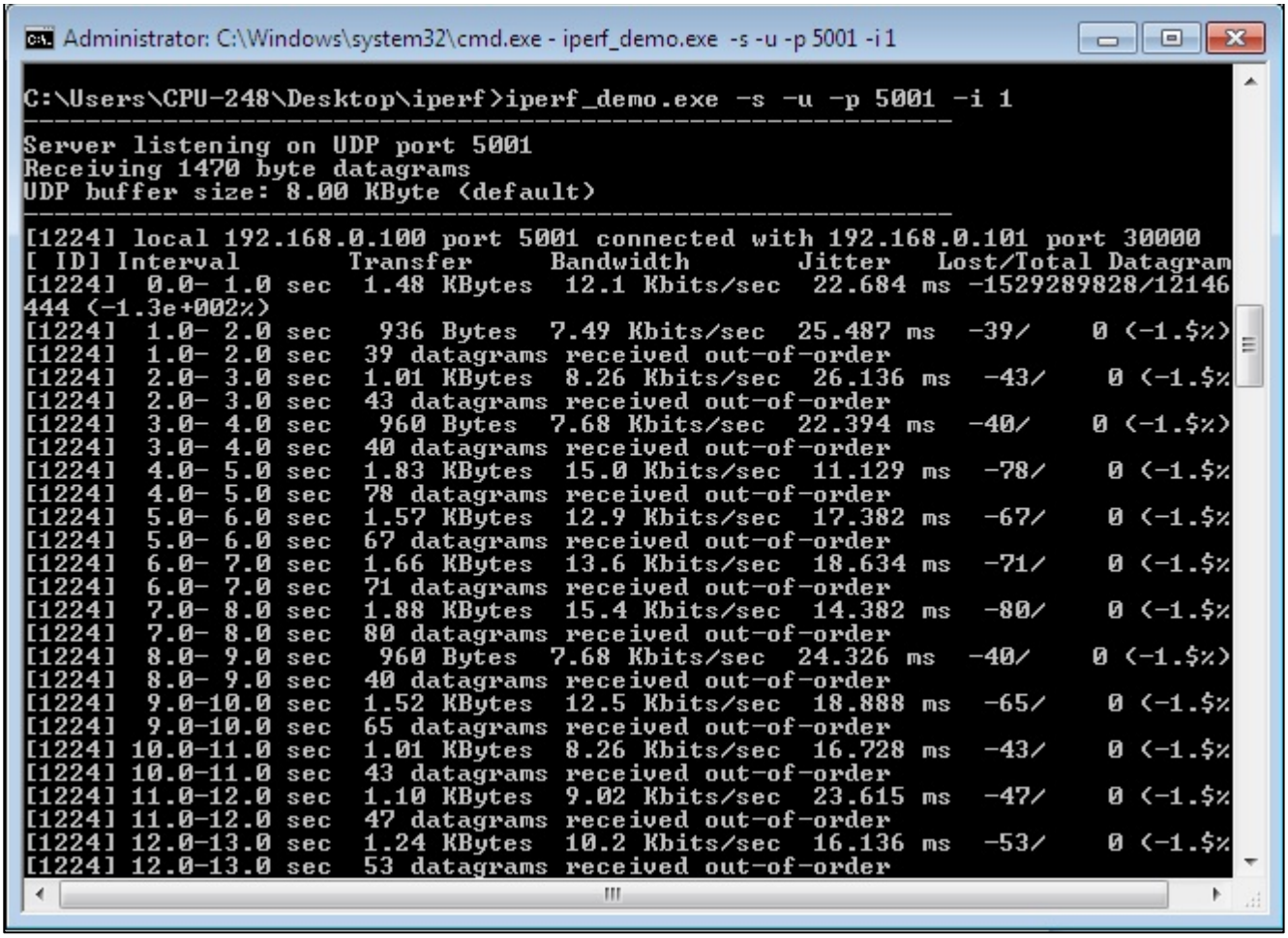
rsi_wlan_config.h file is already set with desired configuration in respective example folders user, need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect RS9116W EVK in STA mode.
2. Open UDP server application using iperf application in Windows PC2 which is connected to the access point through LAN.
3. Users can download the application from the link below:
   https://iperf.fr/iperf-download.php#windows
   iperf_demo.exe –s -u -p <SERVER_PORT> -i 1



4. After program gets executed, RS9116W EVK Device will scan and connect to access point and get IP.
5. After successful connection, the device goes into configured power save and sends configured number of (**NUMBER_OF_PACKETS)** UDP packets to remote peer which is connected to access point through LAN. Please refer the below image for reception of UDP data on UDP server.



6.21 Raw Data

Overview

The raw data application demonstrates how the Silicon Labs device receives the raw data packets (packets of other IP network) and sends them to host, and also how it receives raw data packets from host and sends on air. In this Application, Silicon Labs device will be created as Access point, allow Wi-Fi stations to connect to it. It processes the ARP request packet (raw data) and sends ARP response (raw data). It also processes ping request (raw data) of other IP network, and sends ping response (raw data) to it.

Sequence of Events

This Application explains user how to:

- Silicon Labs Device starts as an access point
- Allow stations to connect
- Reply for ping request and ARP request of other networks also

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Windows Laptop for Wi-Fi Station

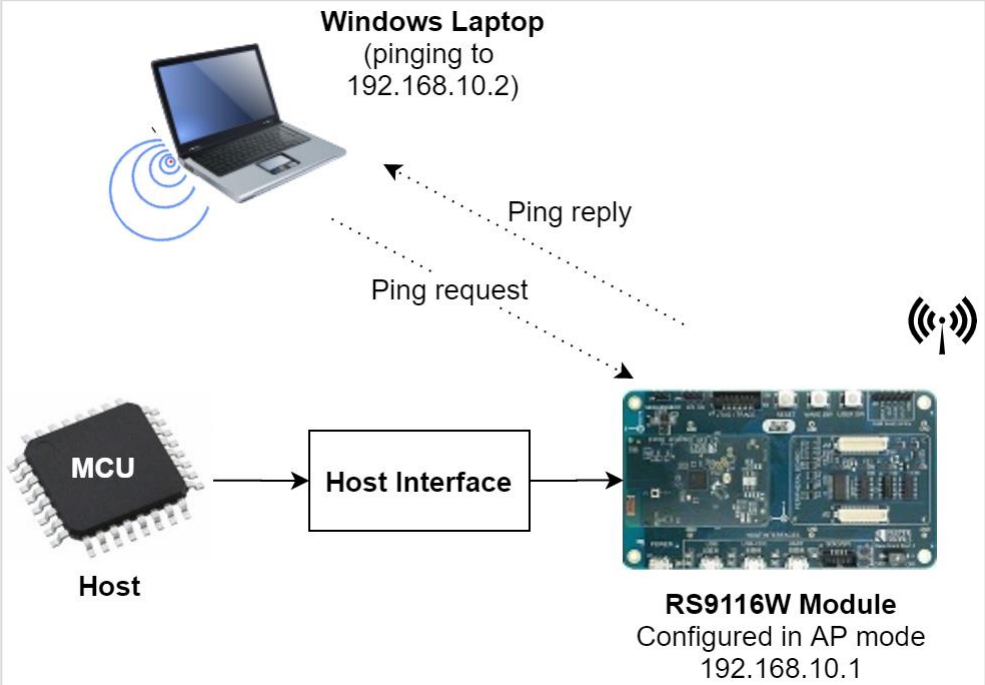


Figure 95: Setup Diagram for Raw Data Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_raw_data_app.c*** file and update/modify following macros,

   SSID refers to the name of the Access point to be created.

```
#define SSID                                    "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                              11
```

> **Note:**
> Valid values for **CHANNEL_NO** is 1 to 11 in 2.4GHz and 36 to 48 & 149 to 165 in 2.4GHz. In this example default configured band is 2.4GHz. So, if user wants to use 5GHz band then user has to set **RSI_BAND** macro to 5GHz band in ***rsi_wlan_config.h*** file.

   SECURITY_TYPE refers to the type of security .Access point supports Open, WPA, WPA2 securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   ENCRYPTION_TYPE refers to the type of Encryption method. Access point supports OPEN, TKIP, CCMP methods.
   Valid configuration is:
   RSI_CCMP - For CCMP encryption
   RSI_TKIP - For TKIP encryption
   RSI_NONE - For open encryption

```
#define ENCRYPTION_TYPE                          RSI_NONE
```

PSK refers to the secret key if the Access point is to be configured in WPA/WPA2 security modes.

```
#define PSK                                      "<psk>"
```

BEACON_INTERVAL refers to the time delay between two consecutive beacons in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

```
#define BEACON_INTERVAL                          100
```

DTIM_INTERVAL refers DTIM interval of the Access Point. Allowed values are from 1 to 255.

```
#define DTIM_INTERVAL                            4
```

To configure IP address

IP address to be configured to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.1" as IP address, update the macro DEVICE_IP as 0x010AA8C0.

```
#define DEVICE_IP                                0X010AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                          0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                          0x00FFFFFF
```

> **Note:**
> In AP mode, configure same IP address for both DEVICE_IP and GATEWAY macro

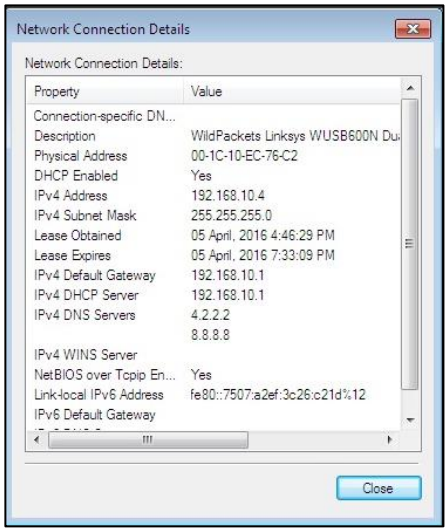
2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
3.  #define CONCURRENT_MODE                          RSI_DISABLE
    #define RSI_FEATURE_BIT_MAP                      FEAT_SECURITY_OPEN
    #define RSI_TCP_IP_BYPASS                        RSI_DISABLE
    #define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT |TCP_IP_FEAT_RAW_DATA)
    #define RSI_CUSTOM_FEATURE_BIT_MAP               FEAT_CUSTOM_FEAT_EXTENTION_VALID
    #define RSI_EXT_CUSTOM_FEAT_BIT_MAP              EXT_FEAT_256k_MODE
    #define RSI_BAND                                 RSI_BAND_2P4GHZ
```

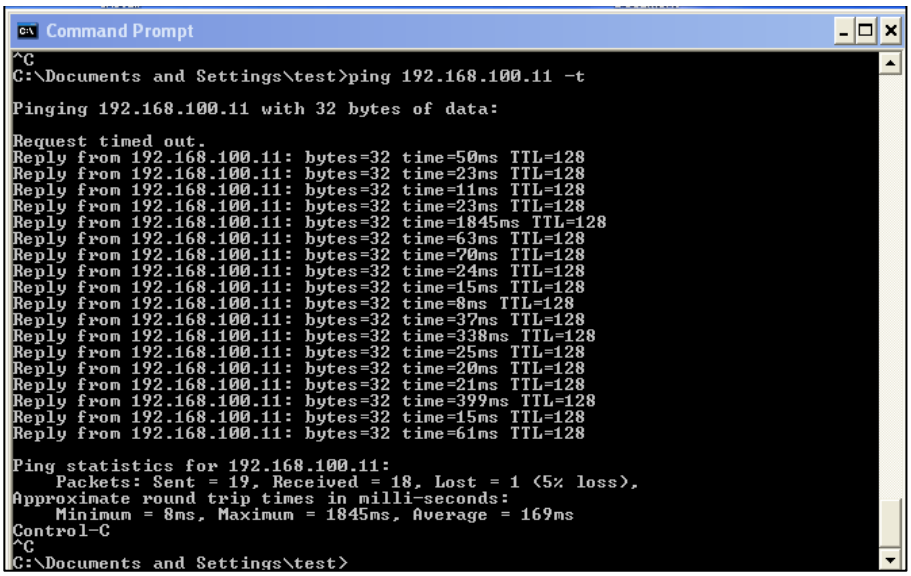
> **Note:**
> rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs Device will be created as Access point and starts beaconing.
2. Now connect Wi-Fi STA (Laptop) to Silicon lab AP (Ex: AP SSID is "REDPINE_AP"). After successful connection, Wi-Fi STA gets IP in the configured IP network (Ex: 192.168.10.4)



3. Initiate ping to an IP of other network (Ex: 192.168.100.11) from Wi-Fi STA (laptop)**.**
   Ping 192.168.100.11 –t
4. Module will reply with ARP response, if connected stations try to ping other IP (which is not in a connected network) and also responds with ping reply for the prior resolved ARP.



6.22 Scan Results

Overview

The scan results application demonstrates how to get configured number of scan results to host and how to access the scan results obtained.

Sequence of Events

This Application explains user how to:

- Issues scan with option to provide scan results to host.
- Selects the AP to connect in the scan results.
- Issues scan with particular ssid.
- Connects to the access point and obtains IP address.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- WiFi Access points

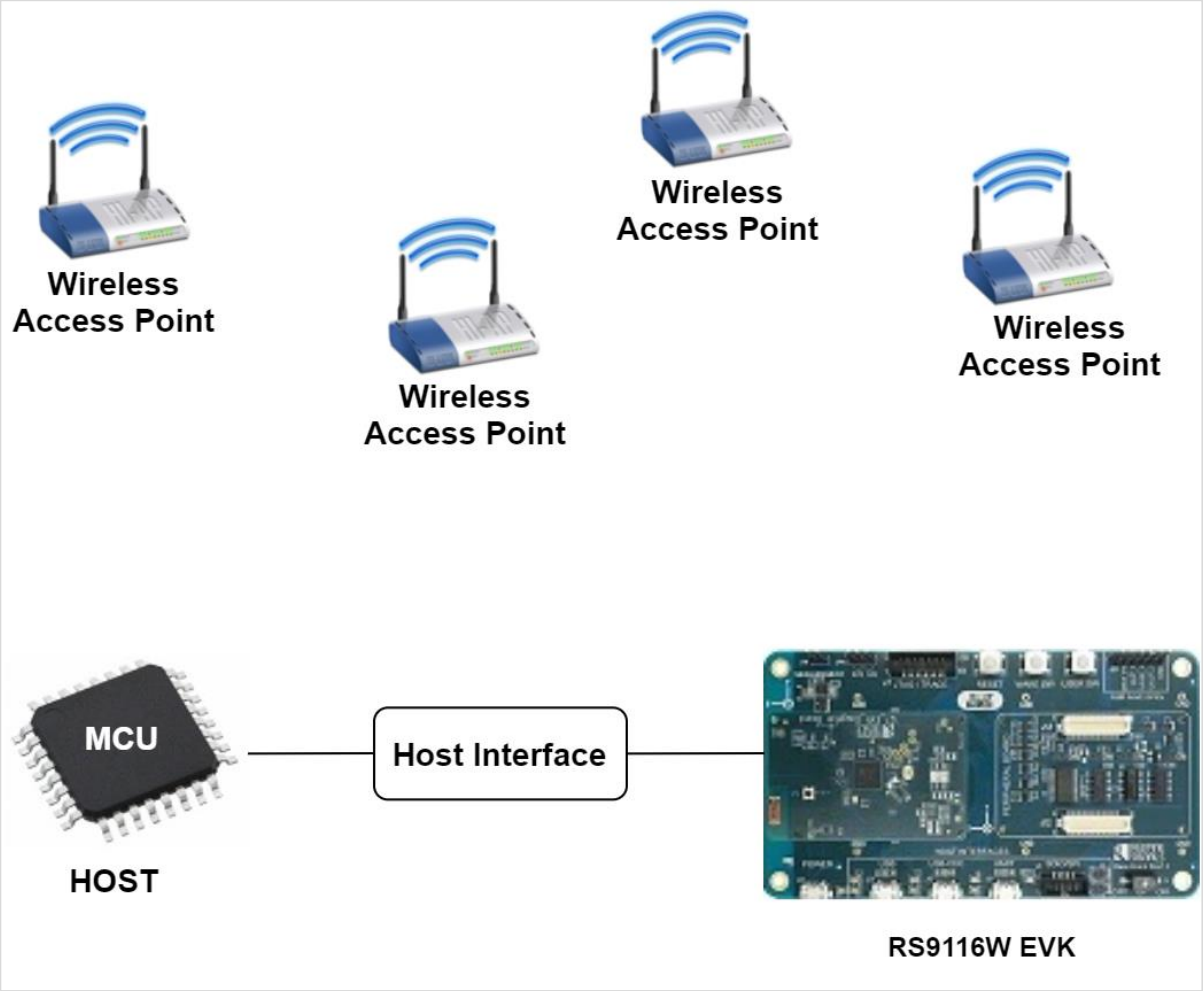


Figure 96: Setup Diagram for Scan Results

Configuration and Steps for Execution

Configuring the Application

1. Open rsi_scan_results_example.c file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels

```
#define CHANNEL_NO                          0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration are :
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                       RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                 "<psk>"
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                     15000
```

To configure IP address
DHCP_MODErefers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                           0
```

> **Note:** Configure STATIC IP to WiSeConnect device. So that user knows the IP address of WiSeConnect device to establish TCP connection from remote peer. In case of DHCP, User has to know the assigned IP by parsing IPCONF response.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.0.101" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                           0X6500A8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                             0x0100A8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                             0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                     RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                 FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                   RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP          (TCP_IP_FEAT_DHCPV4_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP          FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP      EXT_FEAT_256k_MODE
#define RSI_EXT_TCPIP_FEATURE_BITMAP        0
#define RSI_BAND                            RSI_BAND_2P4GHZ
#define PROCESS_SCAN_RESULTS_AT_HOST        1
```

> **Note: rsi_wlan_config.h** file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs device will scan for access points.
2. If Debug prints are enabled, it will print scan results.
3. It will connect to the access point having the configuration same as that of in the application and get IP.

6.23 SNTP Client

Protocol Overview

Simple Network Time Protocol (SNTP) is a simplified version of Network Time Protocol (NTP) that is used to synchronize computer clocks on a network. This simplified version of NTP is generally used when full implementation of NTP is not needed.
SNTP is a simplified access strategy for servers and clients using NTP. SNTP synchronizes a computer's system time with a server that has already been synchronized by a source such as a radio, satellite receiver or modem.
SNTP supports unicast, multicast and anycast operating modes. In unicast mode, the client sends a request to a dedicated server by referencing its unicast address. Once a reply is received from the server, the client determines the time, roundtrip delay and local clock offset in reference to the server. In multicast mode, the server sends an unsolicited message to a dedicated IPv4 or IPv6 local broadcast address. Generally, a multicast client does not send any requests to the service because of the service disruption caused by unknown and untrusted multicast servers. The disruption can be avoided through an access control mechanism that allows a client to select a designated server he or she knows and trusts.

Overview

This application demonstrates how Silicon Labs device gets info from SNTP server. In this application, Silicon Labs device connects to Access Point in client mode and connects to SNTP server. After successful connection with SNTP server, application gets time and date info from SNTP server.

Sequence of Events

This Application explains user how to:

- Silicon Labs device in a client mode
- Connect with SNTP server
- Get time and date info from the SNTP server

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- WiFi Access point with internet
- Silicon Labs module

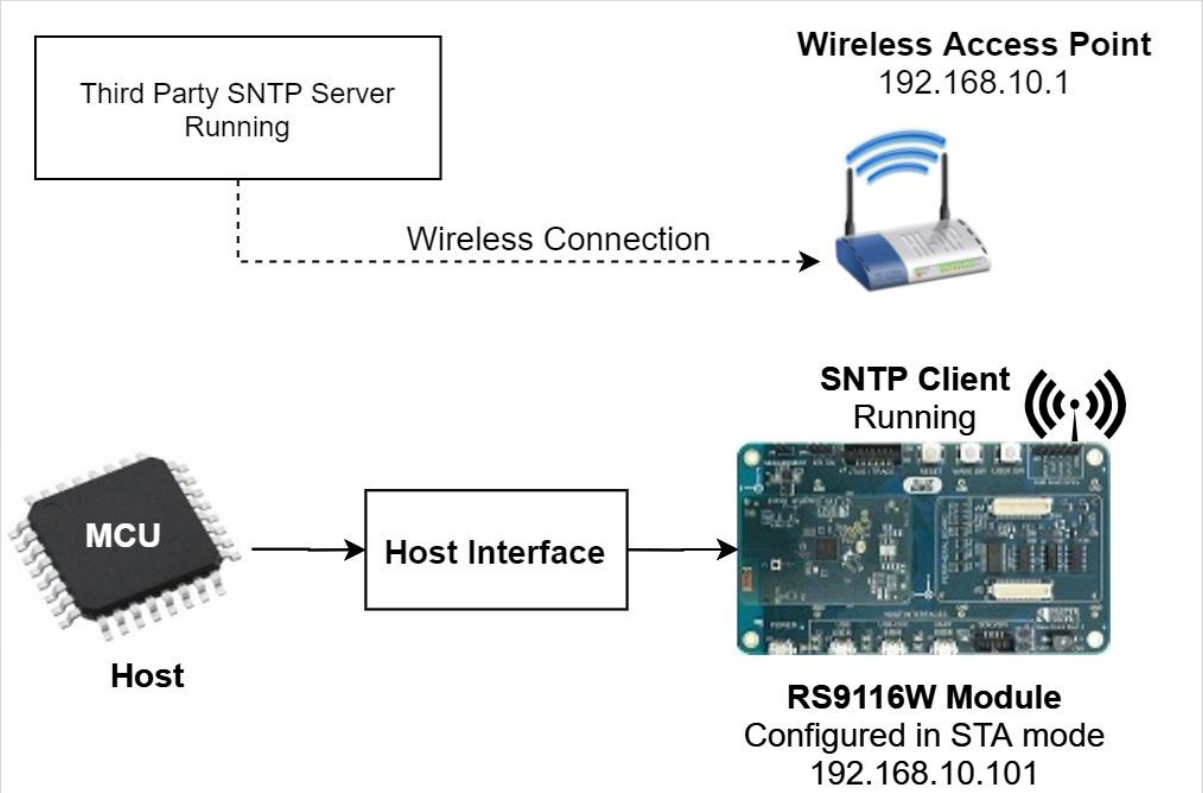


Figure 97: Setup diagram for SNTP Client

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_sntp_client_app.c*** file and update / modify the following macros.

   SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels.

```
#define CHANNEL_NO                                  0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     NULL
```

To configure SNTP client Parameters:
To select IPv6, FLAGS should be set to 1, by default it supports IPv4

```
#define FLAGS                                   0
```

SERVER_IP_ADDRESS refers remote SNTP Server IP address to connect.
IP address should be in long format and in little endian byte order.

```
#define SERVER_IP                               0x640AA8C0
```

Configure the SNTP method to use the server

```
#define SNTP_METHOD                             RSI_SNTP_UNICAST_MODE
```

SNTP time out value to use

```
#define SNTP_TIMEOUT                            50
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                               1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                               0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order.

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order.

Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT |
TCP_IP_FEAT_SNTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP             EXT_FEAT_256k_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN / WPA-PSK / WPA2-PSK mode in order to connect Silicon Labs device in STA mode.
2. Connect to SNTP server and request server for information.
   Eg: SNTP server ip address 128.138.141.172
3. After the program gets executed, Silicon Labs Device would be connected to Access point and gets IP.
4. After successful connection with access Point, Device starts connection with the SNTP server.
5. After successful connection, module will send request to the server for time,date and Server Details.
6. Application gets all the information requested.

6.24 Socket Select App

SOCKET SELECT Overview

Select concept helps to handle multiple clients.

- Select command allows to monitor multiple file descriptors, waiting until one of the file descriptors become active.
- For example, if there is some data to be read on one of the sockets select will provide that information.
- **Select** works like an interrupt handler, which gets activated as soon as any file descriptor sends any data.

Overview

The socket select application demonstrates how to monitor multiple sockets for a certain activity to occur. If there is some data to be read on one of the sockets, select will provide that information.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Open TCP server socket in device
- Select the socket on which data to be read.
- Connect to TCP server socket opened in device from remote peer using TCP client socket.
- Parse the result if the data is to be read from the TCP client socket to receive the data.

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- WiFi Access point
- Windows PC2
- TCP client application running in Windows PC2 (This application uses iperf application to open TCP client socket)

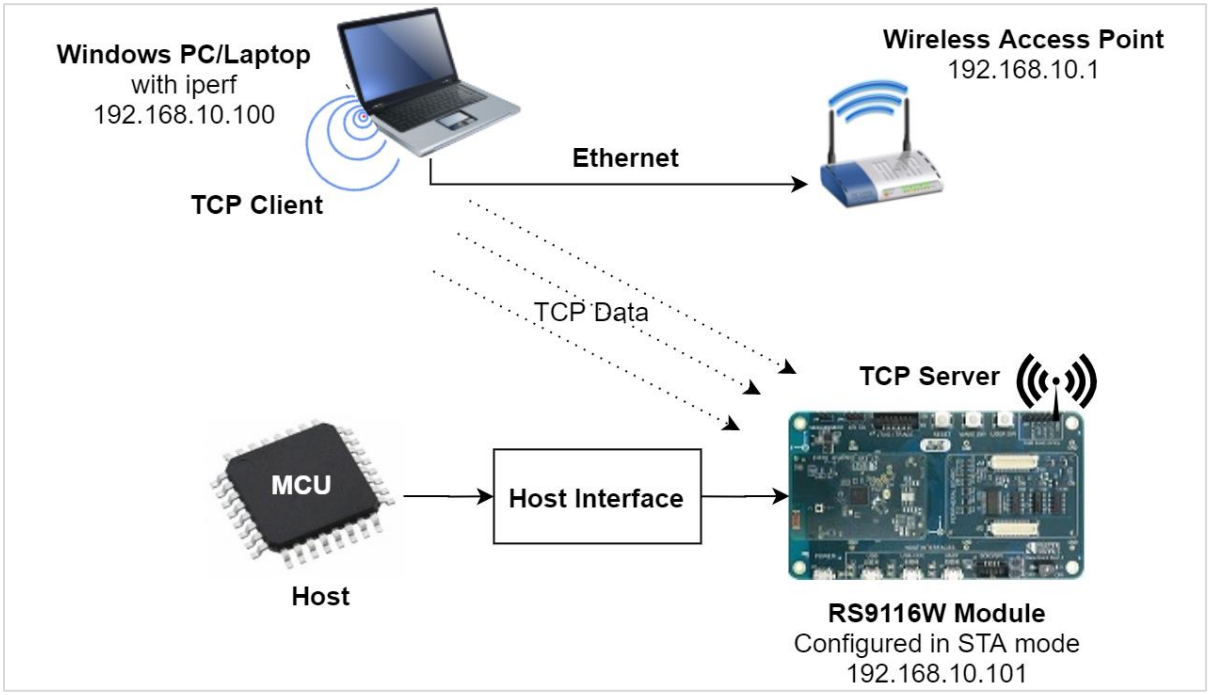


Figure 98: Setup Diagram for Socket Select Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_socket_select.c*** file and update/modify following macros,

   SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels.

```
#define CHANNEL_NO                          0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                       RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK "<psk>"
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                              1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                              0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                0x00FFFFFF
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                        15000
```

2. User can connect to access point through PMK
   To Enable keep 1 else 0

```
#define CONNECT_WITH_PMK                        0
```

Note:
If CONNECT_WITH_PMK is enabled, SECURITY_TYPE is set to RSI_WPA2_PMK

3. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                        RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                    FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                      RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP            (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_EXT_TCPIP_FEATURE_BITMAP           EXT_TCP_IP_TOTAL_SELECTS_2
#define RSI_CUSTOM_FEATURE_BIT_MAP             FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP            EXT_FEAT_256k_MODE
#define RSI_BAND                               RSI_BAND_2P4GHZ
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. After the program gets executed, Silicon Labs module configured as client and connects to AP and gets IP.
3. After successful connection with the Access Point, the socket select is issued for the desired socket.
4. Open TCP client fromWindowsPC2 and connect to TCP server opened on the device on port number **DEVICE_PORT**.
5. Select provides the response about the socket whether the data is to be read on the socket or not
6. If data is to be received on the socket, then the receive function is called on the socket.

6.25 SSL_Client

SSL Overview

SSL stands for Secure Sockets Layer. SSL is the standard security technology for establishing an Encrypted link between a web server and a browser. This link ensures that all data passed between the web servers and the browsers remain Private & Integral.
Data encryption, Server authentication, Message integrity, Optional client authentication for a TCP / IP connection are the main objectives of SSL protocol**.**

Overview

This application demonstrates how to open and use a standard TCP client socket with secure connection using SSL and sends data on socket.

Sequence of Events

This Application explains user how to:

- Connect the device to an Access point and get IP address through DHCP
- Open TCP Server socket over SSL at Access point using OpenSSL
- Establish TCP connection over SSL with TCP server opened on remote peer
- Send TCP data from WiSeConnect device to remote device

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- WiFi Access point
- Windows PC2
- TCP server over SSL running in Windows PC2 (This application uses OpenSSL to create TCP server over SSL)

Note:

Please download OpenSSL for windows from the below link:
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

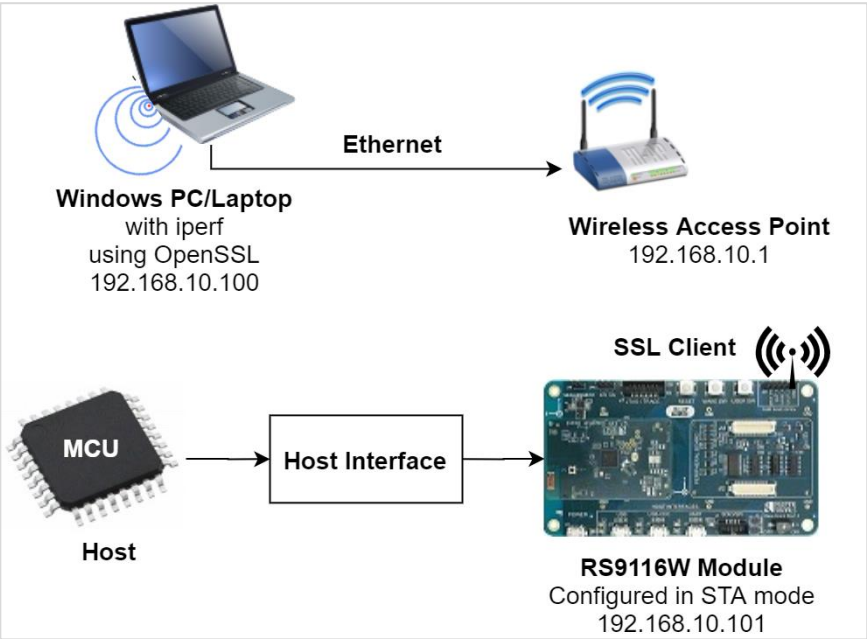


Figure 99: Setup Diagram for TCP Client Socket over SSL

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ssl_client.c*** file and update / modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                          "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels

```
#define CHANNEL_NO                    0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                 RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                           "<psk>"
```

   To Load certificate

```
#define LOAD_CERTIFICATE              1
```

   If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.
   By default, application loading **"cacert.pem"** certificate if **LOAD_CERTIFICATE** enable. In order to load different certificate,user has to follow the following steps:

- rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pemcertificate into linear array form using python script provided in the release package "***sapis/examples/utilities/certificates/certificate_script.py***".

Example:

If the certificate is wifi-user.pem, enter the command in the following way:
python certificate_script.py ca-cert.pem
The script will generate wifiuser.pem in which one linear array named **cacert** contains the certificate.

- After the conversion of certificate, update *rsi_ssl_client.c* source file by including the certificate file and also by providing the required parameters to **rsi_wlan_set_certificate** API.

Once the certificate loads into the device, it will write into the device flash. So,user need not load certificate for every boot up unless certificate change. So, define LOAD_CERTIFICATE as 0, if certificate is already present in the device.

Note:

All the certificates are given in the release package.
Path: **sapis/examples/utilities/certificates**
Enable **SSL** macro to open SSL socket over TCP.

```
#define SSL                             1
```

DEVICE_PORT port refers SSL client port number

```
#define DEVICE_PORT                     5001
```

SERVER_PORT refers remote SSL server port number which is opened in Windows PC2

```
#define SERVER_PORT                     5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with SSL server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.0.100" as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS               0x6400A8C0
```

NUMEBR_OF_PACKETS refers how many packets to send from TCP client

```
#define NUMBER_OF_PACKETS               <no of packets>
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                 15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC.

```
#define DHCP_MODE                       1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP     (TCP_IP_FEAT_DHCPV4_CLIENT|TCP_IP_FEAT_SSL)
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

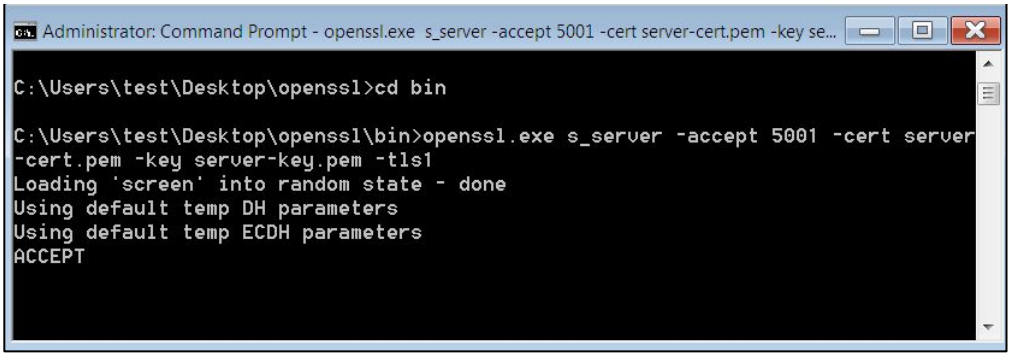
Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. In Windows PC2 which is connected to AP through LAN, Download the Openssl package from above mentioned link and run SSL server by giving following command,
   Open ssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
   Example: open ssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1.

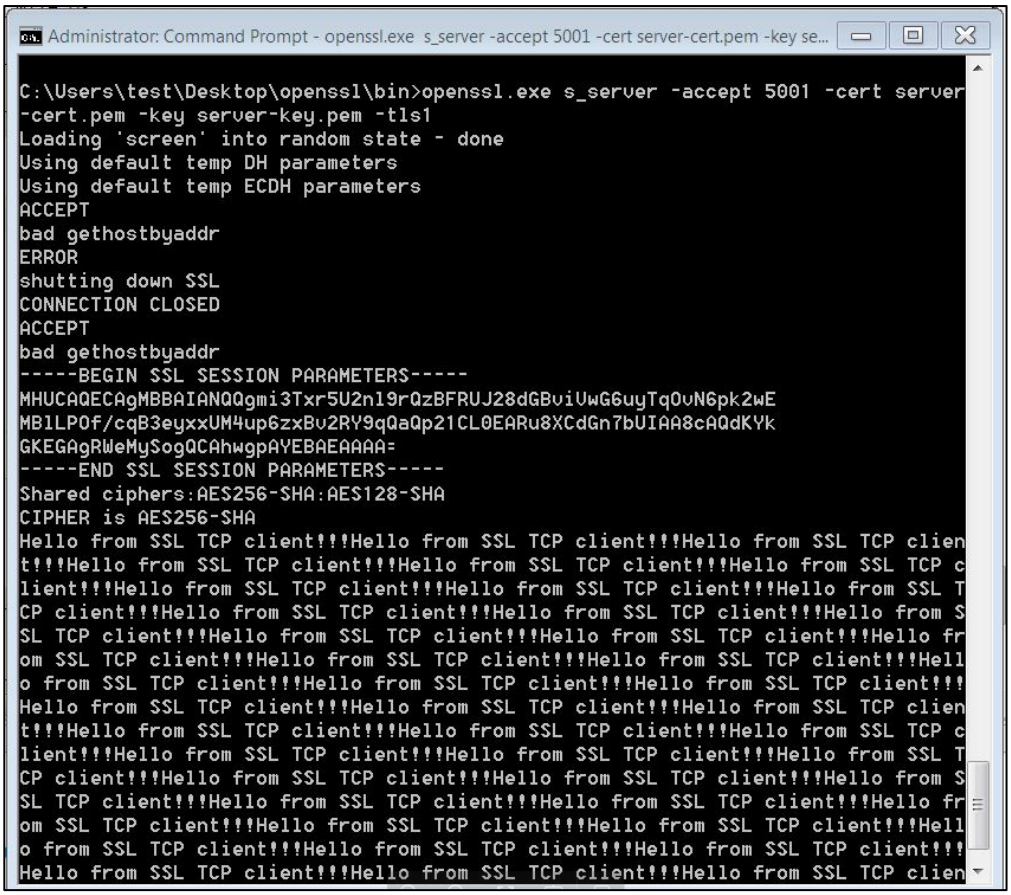
Note:

All the certificates are given in the release package.
Path: sapis/examples/utilities/certificates



3. After the program gets executed, the Silicon Labs device would be connected to the access point having the configuration same as that of in the application and get IP.

4. The device which is configured as SSL client will connect to remote SSL server and sends number of packets configured in **NUMBER_OF_PACKETS.**



6.26 SSL Client with multiple TLS versions

SSL Overview

SSL stands for Secure Sockets Layer. SSL is the standard security technology for establishing an Encrypted link between a web server and a browser. This link ensures that all data passed between the web servers and the browsers remain Private & Integral. Data encryption, Server authentication, Message integrity, Optional client authentication for a TCP/IP connection are the main objectives of SSL protocol. In this user can open multiple sockets with Different TLS versions.

Overview

This application demonstrates how to open and use a standard TCP client socket with secure connection using SSL and sends data on socket.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Open TCP Server socket over SSL at Access point using openssl.
- Establish TCP connection over SSL with TCP server opened on remote peert.
- Send TCP data from Silicon Labs device to remote device

Application Setup

The Silicon Labs device in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Windows PC1 with CooCox IDE Spansion (MB9BF568NBGL) micro controller
- Windows PC2
- Silicon Labs module
- TCP server over SSL running in Windows PC2 (This application uses OpenSSL to create TCP server over SSL)

Note: Please download openssl for windows from below link,
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

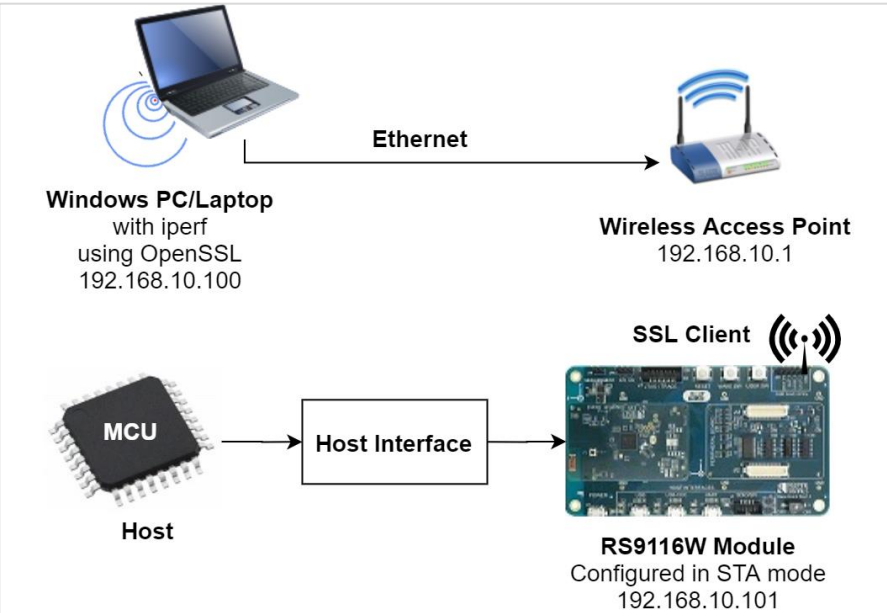


Figure 100: Setup Diagram for TCP Client Socket over SSL Application with Multiple TLS Versions

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_ssl_client_tls_versions.c*** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                    "REDPINE_AP"
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.

Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                                   RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                             NULL
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT1                                     <local port>
```

SERVER_PORT port refers remote TCP server port number which is opened in Windows PC2.

```
#define SERVER_PORT1                                    <remote port>
```

DEVICE_PORT2 refers another TCP client port number

```
#define DEVICE_PORT2                                   <local port>
```

SERVER_PORT2 refers another remote TCP server port number which is opened in Windows PC2.

```
#define SERVER_PORT2                                   <remote port>
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as IP address, update the macro **DEVICE_IP** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS                               0x640AA8C0
```

NUMEBR_OF_PACKETS refers how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS                               <no of packets>
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                                 15000
```

LOAD_CERTIFICATE refers to load certificates into flash

0-Already certificates are there in flash so no need to load

1-Certicates will load into flash

```
#define LOAD_CERTIFICATE                               0
```

Note: If certificates are not there in flash, then SSL handshake fails.

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                                   1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                                   0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order.

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                     0x010AA8C0
```

```
#define NETMASK                                     0x00FFFFFF
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                     0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_SSL)
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP              (EXT_FEAT_SSL_VERSIONS_SUPPORT |
EXT_FEAT_256k_MODE)
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

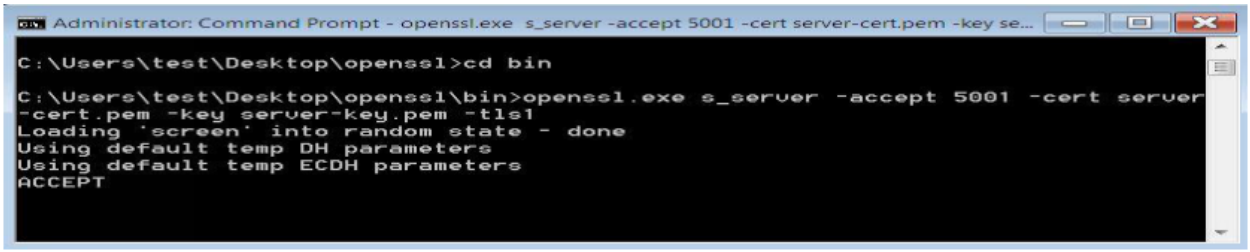
Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. In Windows PC2 which is connected to AP through LAN, Download the Open SSL package from the above mentioned link and run Two SSL servers by giving the following command,
   Openssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
   Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem

Example: openssl.exe s_server –accept 5002 –cert server-cert.pem -key server-key.pem

Note:
All the certificates are given in the release package. Path: **sapis/examples/utilities/certificate**



3. After the program gets executed, Silicon Labs Device would be connected to Access point having the configuration same that of the application and gets IP.
4. The Device which is configured as SSL client will connect to remote SSL server and sends number of packets configured in **NUMBER_OF_PACKETS.**

Note:
Please download openssl for windows from below link,
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

6.27 Station Ping

PING Overview

Ping is used diagnostically to ensure that a host computer the user is trying to reach is actually operating. Ping works by sending an Internet Control Message Protocol (ICMP) Echo Request to a specified interface on the network and waiting for a reply. Ping can be used for troubleshooting to test connectivity and determine response time.

Overview

The application demonstrates how to configure Silicon Labs device in client mode to send ping request to target IP address .

Sequence of Events

This Application explains user how to:

- Connect to Access Point in station mode
- Send Ping requests to configured target IP address

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Wireless Access Point
- Silicon Labs module
- A TCP server application running on the Windows PC2 (This example uses iperf for windows)

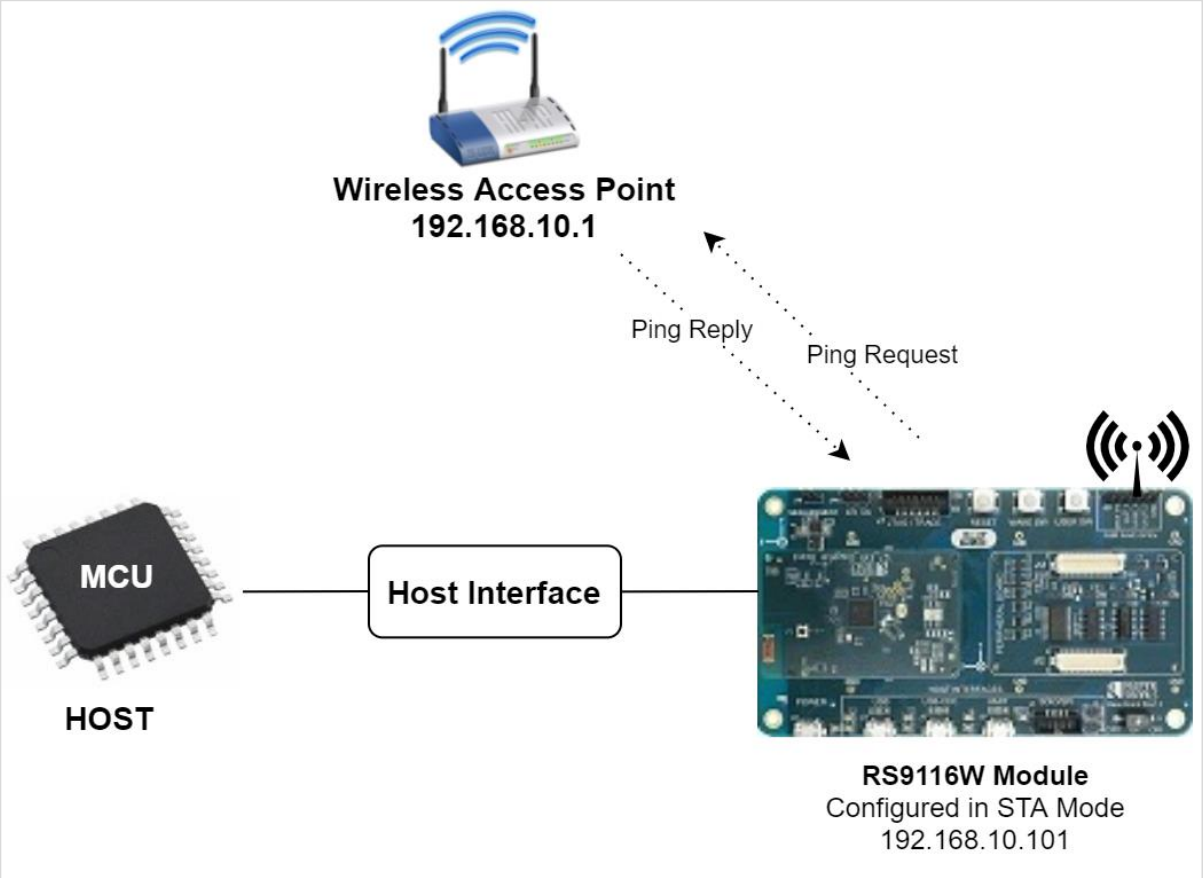


Figure 101: Setup Diagram for Station ping Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_station_ping.c*** file and update/modify following macros,

   SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels.

```
#define CHANNEL_NO                          0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                       RSI_OPEN
```

   AP_BSSID refer to BSSID of AP, join based up on BSSID (Example: If two Access points had same SSID then at the time based on this BSSID, module will join to particular AP). This feature is valid only if RSI_JOIN_FEAT_BIT_MAP set to RSI_JOIN_FEAT_BSSID_BASED in the rsi_wlan_config.h file.

```
#define AP_BSSID                            { }
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK "<psk>"
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                       1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

Configure following macro stoping initiate ping with the remote peer
IP address of the remote peer (AP IP address).
Example: To configure "192.168.10.1" as **REMOTE_IP**, update the macro **REMOTE_IP** as **0x0A0AA8C0**.

```
#define REMOTE_IP                       0x010AA8C0
```

PING_SIZE refers the size of ping packet.

```
#define PING_SIZE                       100
```

NUMBER_OF_PACKETS refers how many number of pings to send from device.

```
#define NUMBER_OF_PACKETS               1000
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                 15000
```

2. User can connect to access point through PMK
   To Enable keep 1 else 0

```
#define CONNECT_WITH_PMK                        0
```

Note:

If CONNECT_WITH_PMK is enabled, SECURITY_TYPE is set to RSI_WPA2_PMK

3. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP             (TCP_IP_FEAT_DHCPV4_CLIENT| TCP_IP_FEAT_ICMP)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP             EXT_FEAT_256k_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
/* ping response timeout in seconds */
#define RSI_PING_REQ_TIMEOUT                    1
/* If want to join with BSSID of AP, then enable this feature and add BSSID of AP in the
rsi_station_ping.c file*/
#define  RSI_JOIN_FEAT_BIT_MAP                  RSI_JOIN_FEAT_BSSID_BASED
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. After the program gets executed, Silicon Labs module configured as client and connects to AP and gets IP.
3. After successful connection with the Access Point, the device starts sending ping requests to the given **REMOTE_IP** with configured **PING_SIZE** to check availability of target Device.
4. Device sends the number of ping packets configured in **NUMBER_OF_PACKETS**.
5. In rsi_station_ping.c file, **rsi_wlan_ping_async** API returns success status, which means that the ping request packet is successfully sent in to the medium. When actual ping response comes from the remote node, it is known from the status parameter of the callback function (**rsi_ping_response_handler**) registered in the Ping API.

6.28 TCP Client

TCP Protocol Overview

TCP (Transmission control protocol) is a connection-oriented protocol for transferring data reliably in either direction between a pair of users. When TCP client sends data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically re-transmit the data and waits for a longer period of time till timeout. After time out socket would be closed. To open a connection, a message is sent with SYN(synchronize) flag. To close a connection, a message is sent with FIN(finish) flag. Urgent messages may also be sent by selecting the PSH(push) flag as a protocol parameter.

Overview

The TCP client application demonstrates how to open and use a standard TCP client socket and sends data to TCP server socket.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP

- Open TCP Server socket at Access point using iperf application.
- Open TCP client socket in device and establish TCP connection with TCP server opened in remote peer.
- Send data from Silicon Labs device to remote peer using opened TCP socket.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- WPS supported Access Point
- Windows PC2
- Silicon Labs module
- A TCP server application running on the Windows PC2 (This example uses iperf for windows)

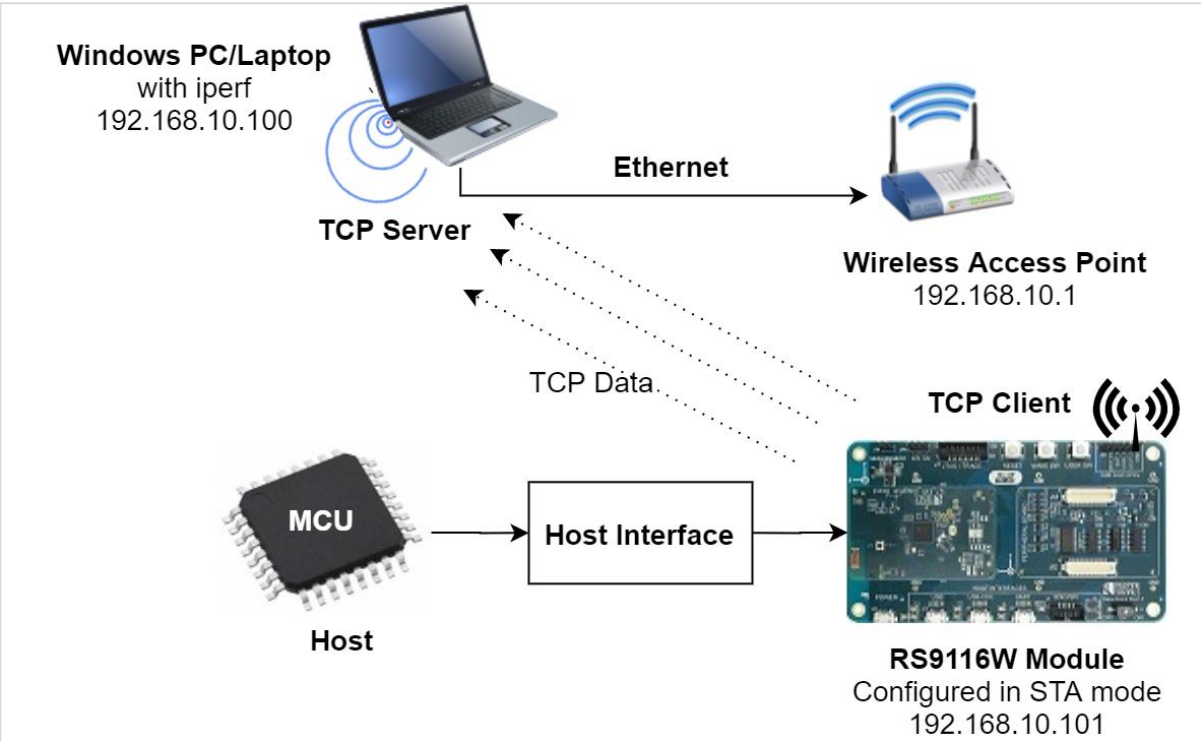


Figure 102: Setup Diagram for TCP Client Socket

Configuration and Steps for Execution

Configuring the Application

1. Open rsi_tcp_client.c file and update / modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                  "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels.

```
#define CHANNEL_NO                            0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK

securities.
Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                          RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                    "<psk>"
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                            5001
```

SERVER_PORT port refers remote TCP server port number which is opened in windows PC2.

```
#define SERVER_PORT                            5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as IP address, update the macro **DEVICE_IP** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS                      0x640AA8C0
```

NUMEBR_OF_PACKETS refers how many packets to send from device to TCP server

```
#define NUMBER_OF_PACKETS                      1000
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                        15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                              1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                              0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

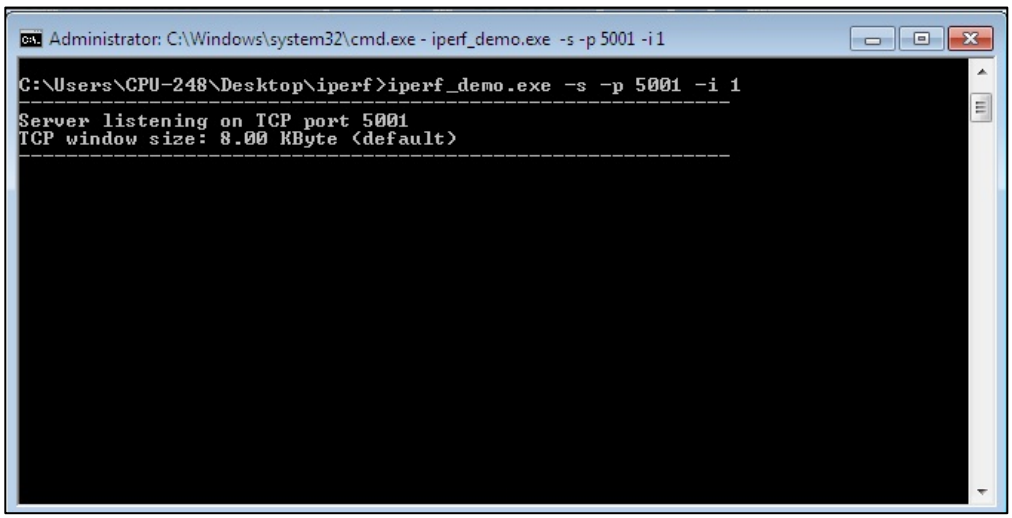
```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP             EXT_FEAT_256k_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Note:

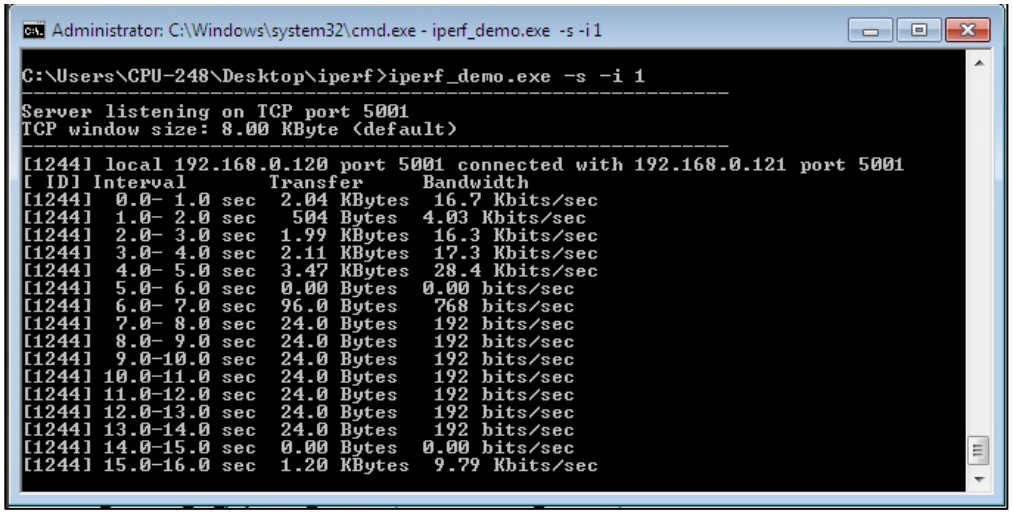
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Open TCP server application using iperf application in Windows PC2 which is connected to Access point through LAN.
3. Users can download application from the link below,
https://iperf.fr/iperf-download.php#windows
iperf_demo.exe -s -p <SERVER_PORT> -i 1



4. After program gets executed, Silicon Labs Device would scan and connect to Access point and get IP.
5. After successful connection, device STA connects to TCP server socket opened on Windows PC2 using TCP client socket and sends configured **NUMBER_OF_PACKETS** to remote TCP server. Please find below image for reception of TCP data on TCP server.



6.29 TCP IP Bypass

TCP Protocol Overview

TCP (Transmission control protocol) is a connection-oriented protocol for transferring data reliably in either direction between a pair of users. When TCP client sends data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically re-transmit the data and waits for a longer period of time till timeout. After time out socket would be closed. To open a connection, a message is sent with SYN(synchronize) flag. To close a connection, a message is sent with FIN(finish) flag. Urgent messages may also be sent by selecting the PSH(push) flag as a protocol parameter.

UDP Protocol Overview

UDP(User Datagram protocol) is a connection less and non-stream oriented protocol for transferring data in either direction between a pair of users. In UDP there is no guarantee that the messages or packets sent would reach at all. No handshake in UDP Protocol because it is connection less protocol.

Overview

The TCP IP Bypass application demonstrates how to open and use of a standard TCP/UDP client/server socket. It bypasses the embedded TCP/IP stack for sending or receiving the data over socket.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point.
- Assign the IP using DHCP or Statically to the device interface (ex: rsi_wlan0)
- Open TCP/UDP Server/Client socket at an Access point using iperf application.
- Open TCP/UDP Client/Server socket in the x86 platform and establish TCP/UDP connection with TCP/UDP server/client opened in remote peer.
- Send data from RS9116 device to remote peer using opened TCP/UDP socket or vice versa.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable a variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(USB) in case of WiSeConnect
- Windows/Linux PC2 for connecting to Access Point.
- Silicon Labs RS9116 module
- A TCP server/client application running on the Windows/Linux PC2 (This example uses iperf for windows)

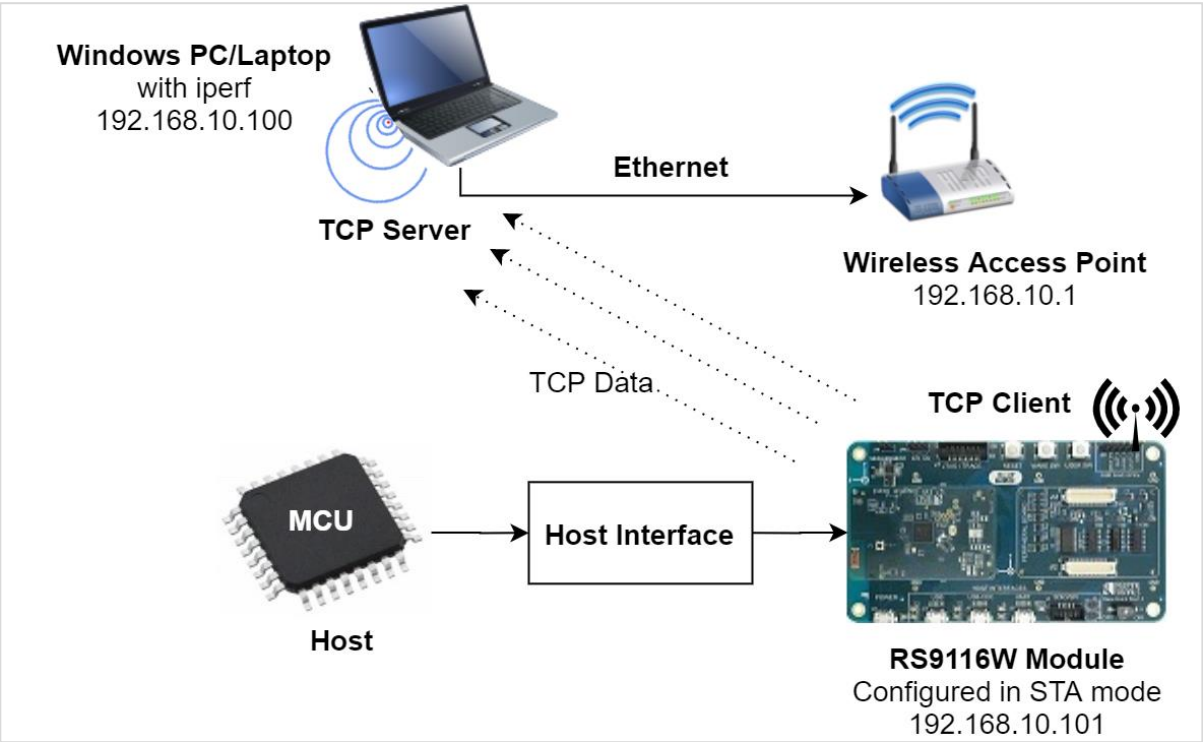


Figure 103: Setup Diagram for TCP IP Bypass

Configuration and Steps for Execution

Configuring the Application

1. Open rsi_tcp_ip_bypass.c file and update / modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, the device will scan all channels.

```
#define CHANNEL_NO                              0
```

   SECURITY_TYPE refers to the type of security. In this application, STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                  "<psk>"
```

RSI_MODULE_IP_ADDRESS refers RS9116 module static IP address, by default it is 192.168.100.76

```
#define  RSI_MODULE_IP_ADDRESS                          "<Static IP for RS9116 device>"
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                        15000
```

Note:
If the user wants to configure STA IP address through DHCP, Follow the below steps on x86 machine.

1) After connection with remote peer

2) Open new tab and give the 3,4 steps

3) dhclient -r

4) dhclient <interface> -v

5) Run iperf application for client or server according to the requirement.

Open ***rsi_wlan_config.h*** file and update/modify the following macros:

```
#define CONCURRENT_MODE                        RSI_DISABLE#define
RSI_FEATURE_BIT_MAP                            FEAT_SECURITY_OPEN#define
RSI_TCP_IP_BYPASS                              RSI_ENABLE#define
RSI_TCP_IP_FEATURE_BIT_MAP                     TCP_IP_FEAT_BYPASS

#define RSI_CUSTOM_FEATURE_BIT_MAP             FEAT_CUSTOM_FEAT_EXTENTION_VALID

#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP         EXT_FEAT_256k_MODE
```

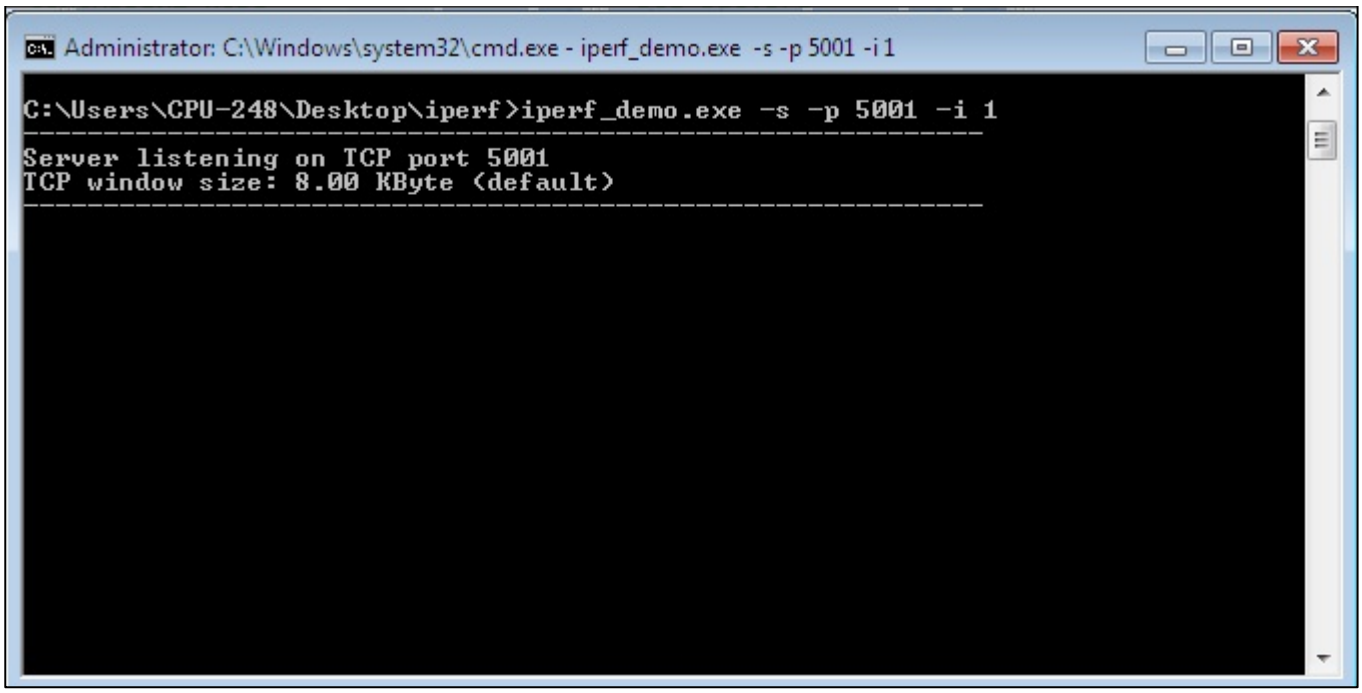
Note:

1. rsi_wlan_config.h file is already set with the desired configuration in the respective example folder.
2. RSI_TCP_IP_BYPASS macro must be enabled for this example in rsi_wlan_config.h as shown in the above configurations.

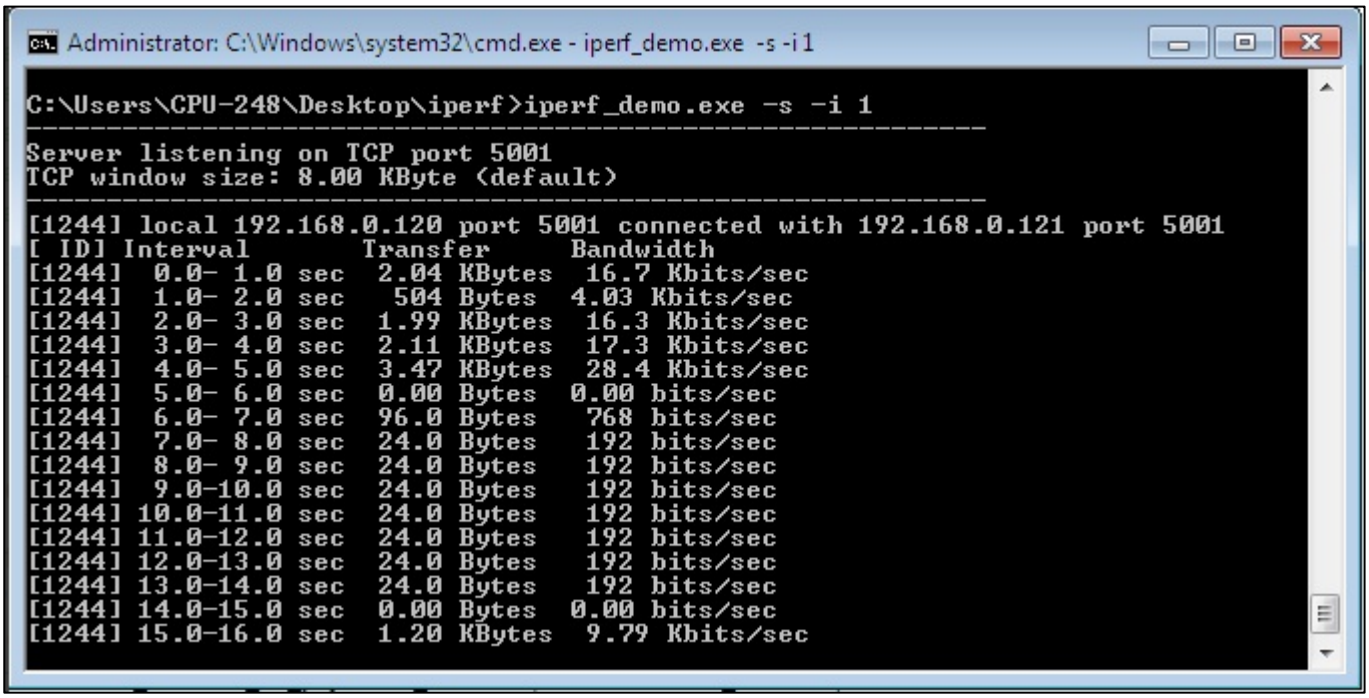
Executing the Application

TCP:

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect the Silicon Labs device in STA mode.
2. Open TCP server/client application using the iperf in Windows PC2 which is connected to the Access point through LAN.
3. Users can download application from the link https://iperf.fr/iperf-download.php#windows
   iperf_demo.exe -s -p <SERVER_PORT> -i 1



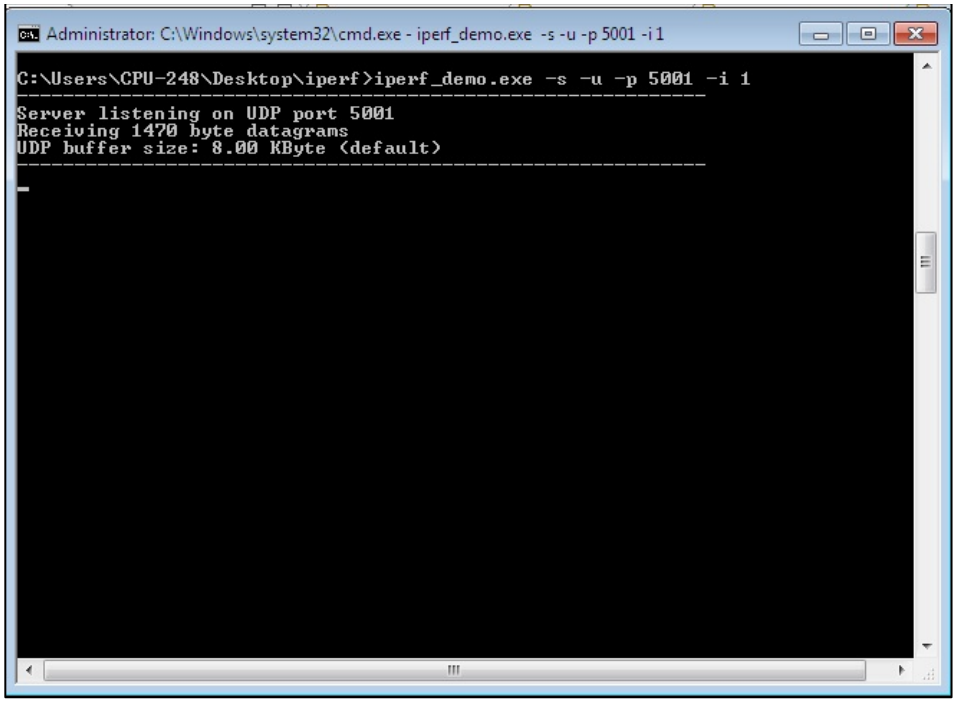
4. After program gets executed, Silicon Labs Device would scan and connect to the Access point and get IP.
5. After successful connection, device STA connects to TCP server/client socket opened on Windows PC2 using TCP client socket and sends data to a TCP server. Please find below image for reception of TCP data on TCP server.



UDP:

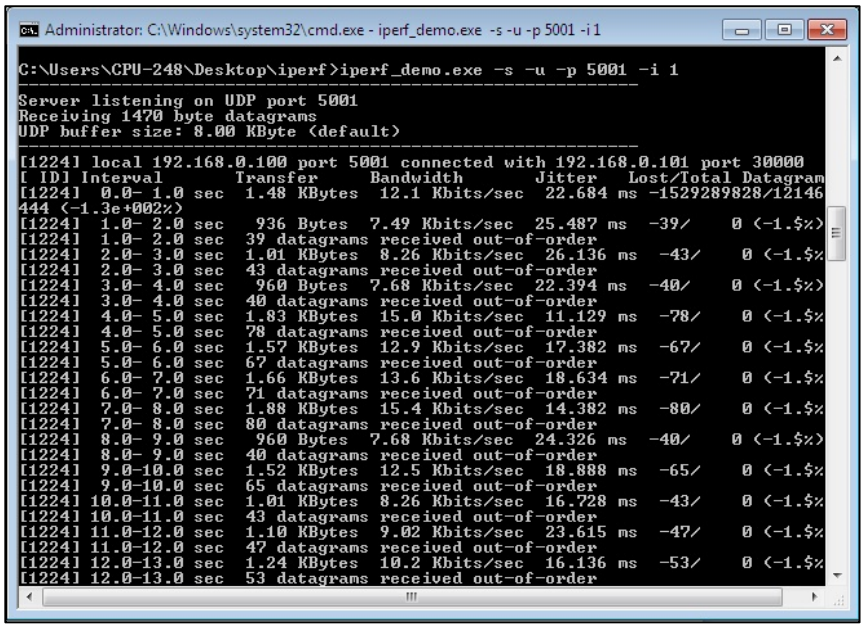
1. Configure the Access point in OPEN / WPA-PSK/WPA2-PSK mode in order to connect the Silicon Labs device in STA mode.
2. Open UDP server/client application using the iperf in Windows PC2 which is connected to the Access point through LAN.

iperf_demo.exe –s -u -p <SERVER_PORT> -i 1



3. After the program gets executed, The Silicon Labs device would scan and connect to the Access point and get IP.

4. After a successful connection, the device STA connects to UDP server/client socket opened on Windows PC2 using UDP client/server socket and sends data to UDP server. Please refer the below image for reception of UDP data on UDP server.



Note:
In the similar way for Linux machine we want to install iperf using dnf install or yum install iperf

To run:

1. TCP client: iperf -c <server ip> -i <interval> -p <server port> -t <time in sec>
2. UDP client: iperf -c <server ip> -i <interval> -p <server port> -t <time in sec> -u -b<bandwidth>M
3. TCP server: iperf -s -i <interval> -p <server port>

UDP server: iperf -s -i <interval> -p <server port> -u

6.30 TCP Server

TCP Protocol Overview

TCP(Transmission control protocol) is a connection-oriented protocol for transferring data reliably in either direction between a pair of users.
TCP server waits for the connections from TCP clients and accepts Incoming TCP connections.

Overview

The TCP server application demonstrates how to open and use a standard TCP server socket and receives data from TCP client socket.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Open TCP server socket in device
- Connect to TCP server socket opened in device from remote peer using TCP client socket.
- Receive data from TCP client socket.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- WiFi Access point
- Windows PC2
- TCP client application running in Windows PC2 (This application uses iperf application to open TCP client socket)

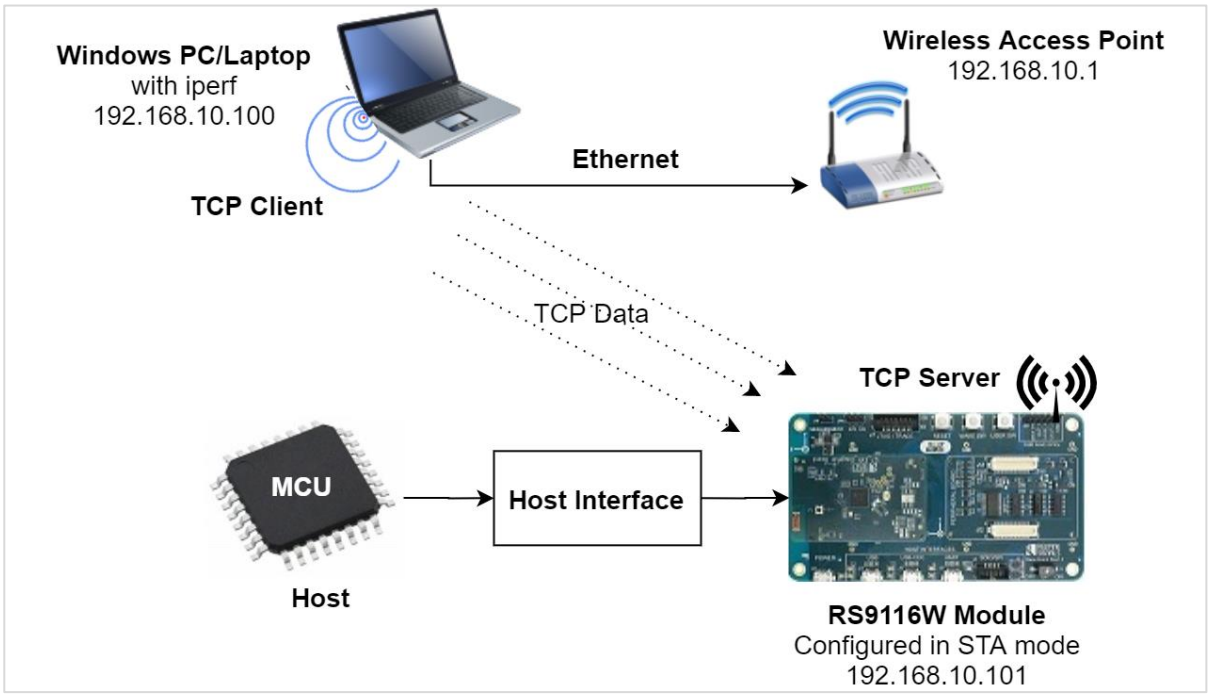


Figure 104: Setup Diagram for TCP Server Socket

Configuration and Steps for Execution

Configuring the Application

1. Open **rsi_tcp_server.c** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                              "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

```
#define CHANNEL_NO                        0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configurations are:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                     RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                               "<psk>"
```

   DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                       5001
```

   Receive data length

```
#define RECV_BUFFER_SIZE                  <recv_buf_size>
```

   NUMEBR_OF_PACKETS refer to how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS                 <no of packets>
```

   Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                   15000
```

   To configure IP address
   DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                         0
```

> **Note:**
> Configure STATIC IP to WiSeConnect device. So that user knows the IP address of WiSeConnect device to establish TCP connection from remote peer. In case of DHCP, User has to know the assigned IP by parsing IPCONF response.

   IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
   Example: To configure "192.168.0.101" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                         0X6500A8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                          0x0100A8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                          0x00FFFFFF
```

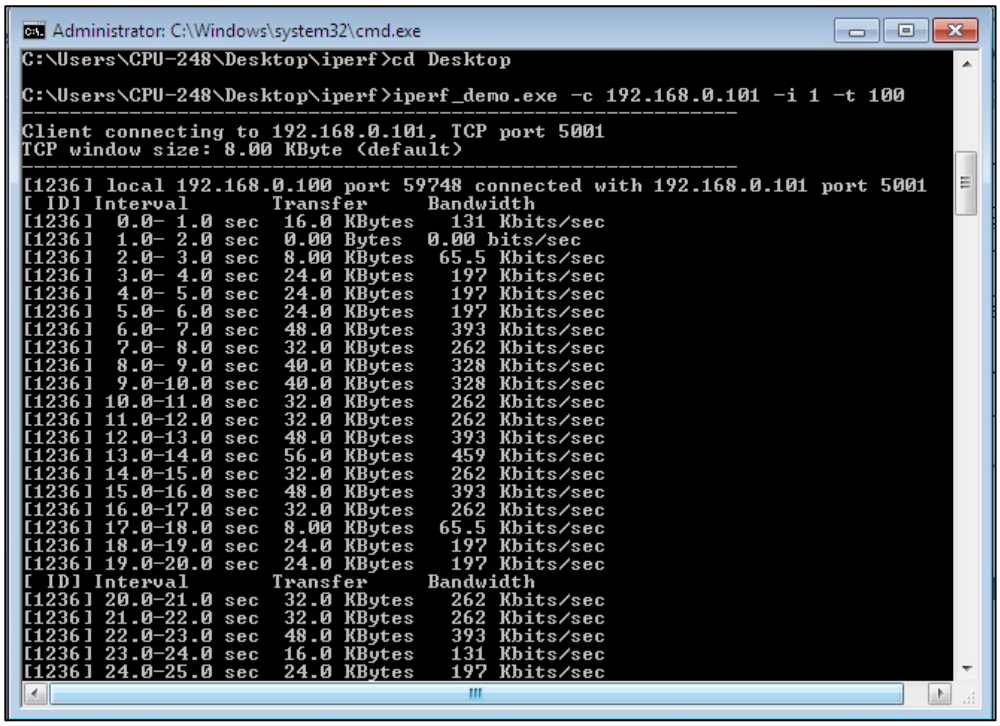
2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                  RSI_DISABLE
#define RSI_FEATURE_BIT_MAP              FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP       TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP       FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP      EXT_FEAT_256k_MODE
#define RSI_BAND                         RSI_BAND_2P4GHZ
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. After the program gets executed, Silicon Labs device will be connected to the access point having the configuration same as that of in the application and get IP.
2. Open TCP client using iperf from Windows PC2 and connect to TCP server opened on the device on port number **DEVICE_PORT** using the following command:
Iperf_demo.exe –c <DEVICE_IP> -p <DEVICE_PORT> -i 1 –t 1000



3. The Silicon Labs device will receive the number of packets configured in **NUMBER_OF_PACKETS** from iperf TCP client and closes the socket.

6.31 Three_SSL_Client_Sockets

SSL Overview

SSL stands for Secure Sockets Layer. SSL is the standard security technology for establishing an Encrypted link between a web server and a browser. This link ensures that all data passed between the web servers and the browsers remain Private & Integral. Data encryption, Server authentication, Message integrity, Optional client authentication for a TCP/IP connection are the main objectives of SSL protocol. In this application user can connect to three different SSL servers having three different set of certificates using certificates loading into FLASH.

Overview

This application demonstrates how to connect to three different SSL servers with three different set of SSL certificates, using the loading certificates into FLASH.

Sequence of Events

This Application explains user how to:

- Loading three different set of SSL certificates into FLASH.
- Connect the Device to an Access point with internet connection and get IP address through DHCP
- Open TCP Server socket over SSL at Access point using openssl.
- Establish first TCP connection over SSL with TCP server opened on remote peer (Ex: Openssl) / SSL server running on cloud (Ex: AWS cloud).
- Establish second TCP connection over SSL with TCP server opened on remote peer (Ex: Openssl) / SSL server running on cloud (Ex: AWS cloud).
- Establish third TCP connection over SSL with TCP server opened on remote peer (Ex: Openssl) / SSL server running on cloud (Ex: AWS cloud).

Application Setup

The RS9116W EVK in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect.
- Windows PC with Keil IDE for STM32 (F411RE) micro controller.
- Windows PC with CooCox IDE Spansion (MB9BF568NBGL) micro controller.
- AWS server information like domain name running in the cloud which supports SSL connection.
- RS9116W EVK
- TCP server over SSL running in Windows PC (This application uses OpenSSL to create TCP server over SSL)

Note:
Please download openssl for windows from below link,
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

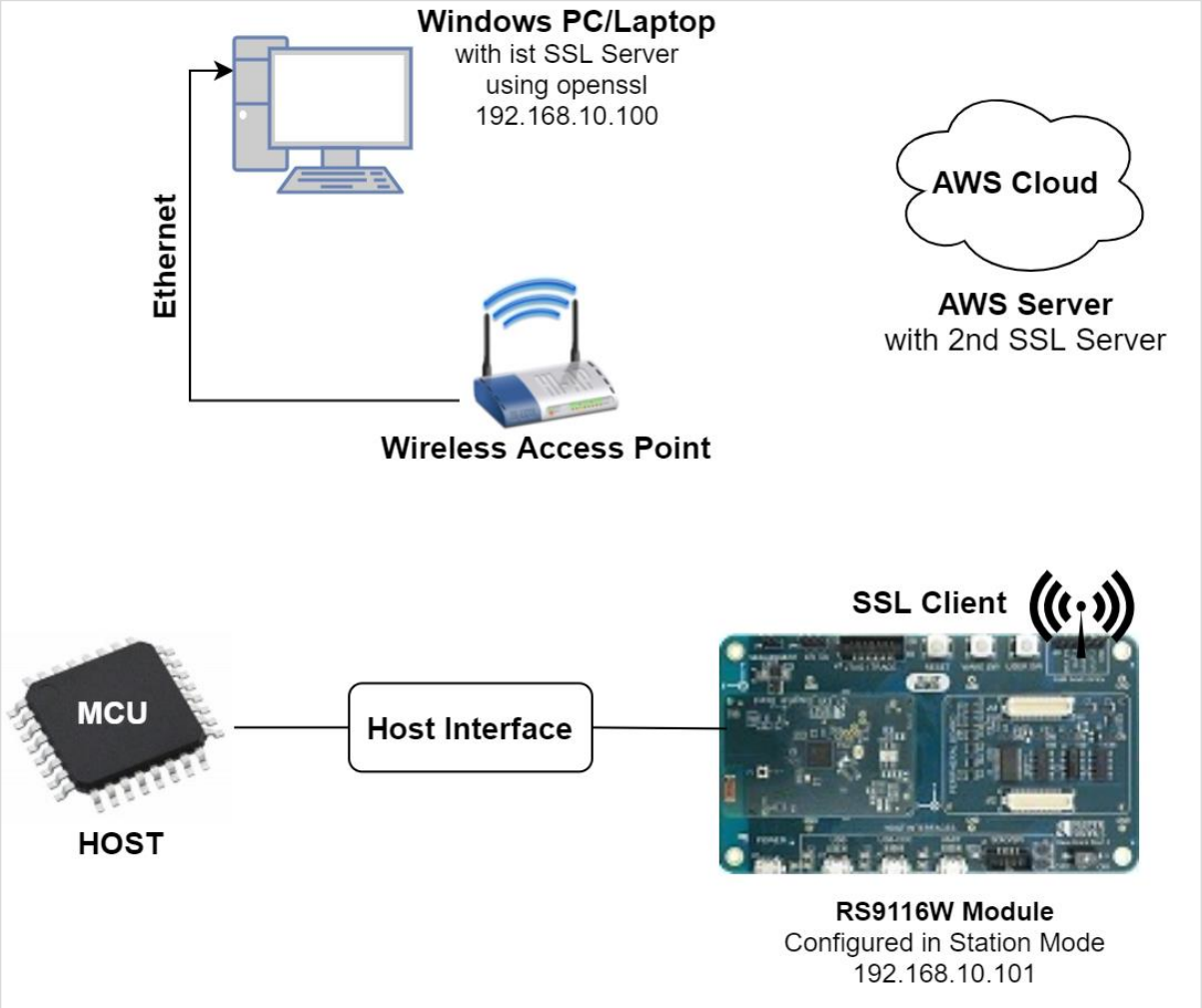


Figure 105: Setup Diagram fro Three SSL Client Sockets Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_three_ssl_client_sockets.c*** file and update/modify following macros:
   SSID refers to the name of the Access point.

```
#define SSID                                    "REDPINE_AP"
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     NULL
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT1                            <local port>
```

SERVER_PORT port refers remote TCP server port number which is opened in remote peer/ which is running on cloud.

```
#define SERVER_PORT1                                    <remote port>
```

DEVICE_PORT2 refers another TCP client port number

```
#define DEVICE_PORT2                                   <local port>
```

SERVER_PORT2 port refers another remote TCP server port number which is opened in remote peer/ which is running on cloud.

```
#define SERVER_PORT2                                   <remote port>
```

DEVICE_PORT3 refers another TCP client port number

```
#define DEVICE_PORT3                                   <local port>
```

SERVER_PORT3 port refers another remote TCP server port number which is opened in remote peer/ which is running on cloud.

```
#define SERVER_PORT3                                   <remote port>
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket over SSL running on the Windows PC.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as IP address, update the macro **DEVICE_IP** as **0x640AA8C0**.

Note:
For Servers running on cloud, get the IP using DNS server.

```
#define SERVER_ADDR                                  0x640AA8C0
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                                15000
```

RSI_SSL_BIT_ENABLE

0- Disable SSL bitmap.

1- Enable SSL bitmap.

This bit should be enabled for SSL connection

```
#define RSI_SSL_BIT_ENABLE                              1
```

Note: If certificates are not there in flash, then SSL handshake fails.

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                                      1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                                   0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order.

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                     0x010AA8C0
```

```
#define NETMASK                                     0x00FFFFFF
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                     0x00FFFFFF
```

AWS_DOMAIN_NAME refers to domain name of the AWS server

```
#define AWS_DOMAIN_NAME                            "a25jwtlmds8eip-ats.iot.us-east-2.amazonaws.com"
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_SSL |
TCP_IP_FEAT_DNS_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP              (EXT_FEAT_256k_MODE |
EXT_FEAT_RSA_KEY_WITH_4096_SUPPORT |
EXT_FEAT_SSL_CERT_WITH_4096_KEY_SUPPORT)

#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

Note:
rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

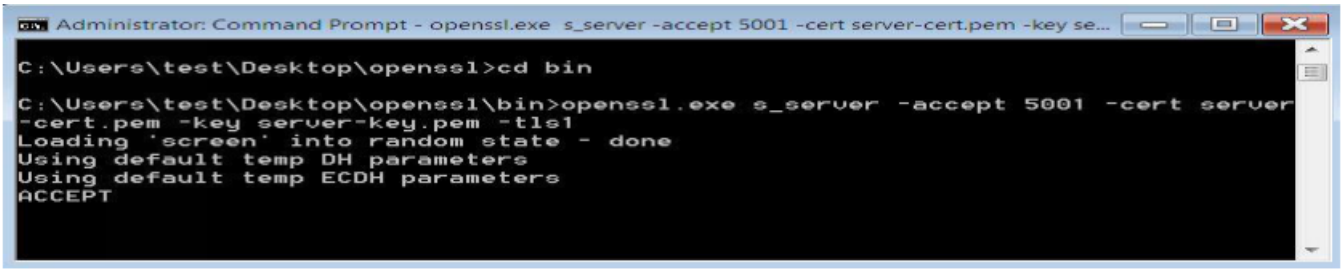
Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect RS9116W EVK in STA mode.

In Windows PC which is connected to AP through LAN, Download the Openssl package from the above mentioned link and run SSL server by giving the following command,
Openssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>

Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem

Note:
All the certificates are given in the release package. Path: sapis/examples/utilities/certificate.



2. Make sure the SSL server is running in the cloud (check with the domain name)
3. After the program gets executed, RS9116W EVK would be connected to Access point having the configuration same that of in the application and get IP.

The Device which is configured as SSL client will connect to three different remote SSL servers.

Note:
Please download openssl for windows from below link,
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

6.32 Throughput App

Overview

Throughput is the rate of production or the rate at which something can be processed. When used in the context of communication networks, such as Ethernet or packet radio, throughput or network throughput is the rate of successful message delivery over a communication channel. This application will demonstrate the throughput measurement.

Example setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- Windows Laptop with application program like iperf
- Access Point

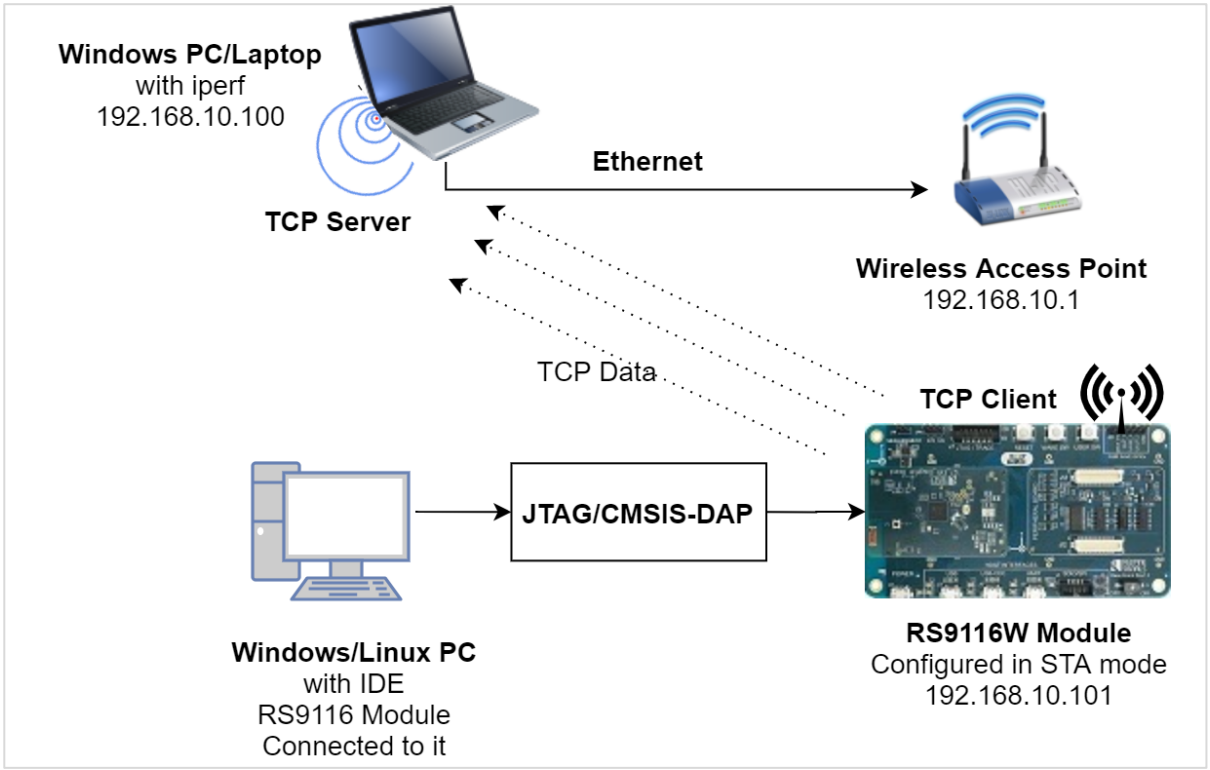


Figure 106: Module Configured in TCP Client Mode

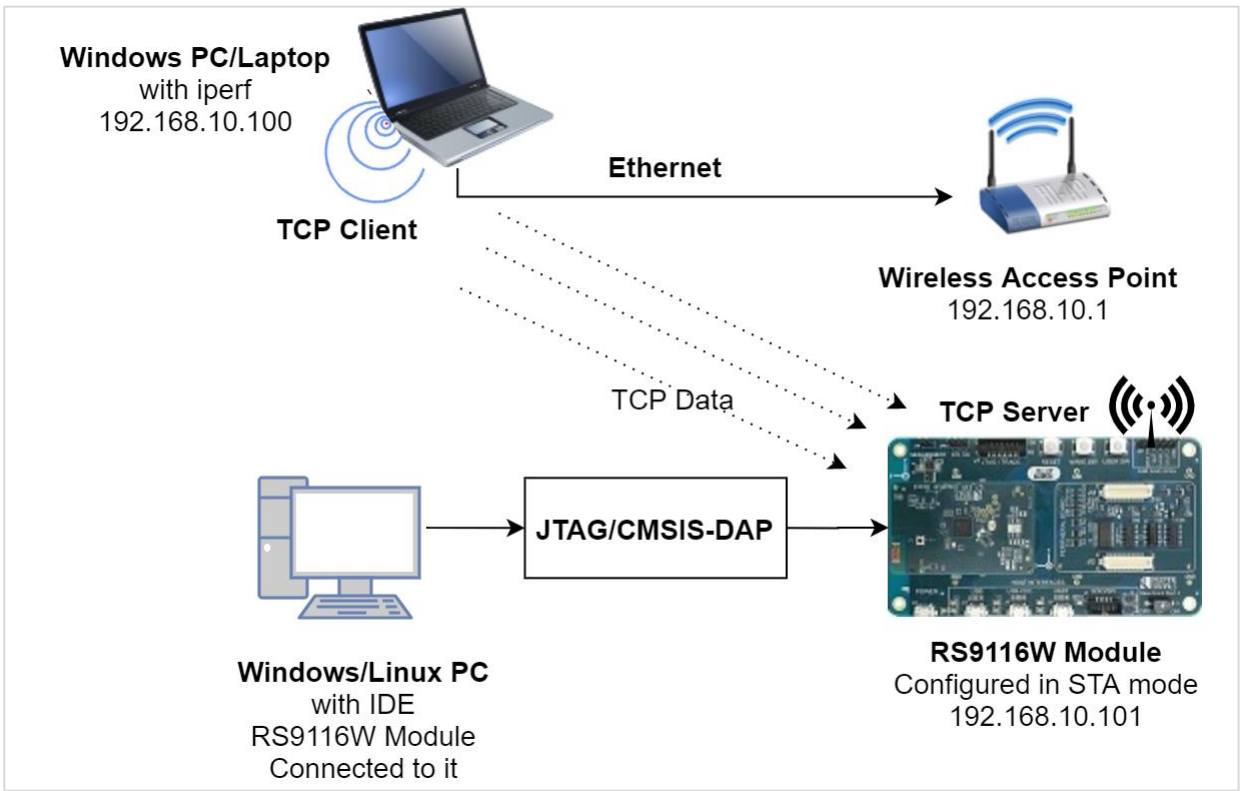


Figure 107: Module Configured in TCP Server Mode

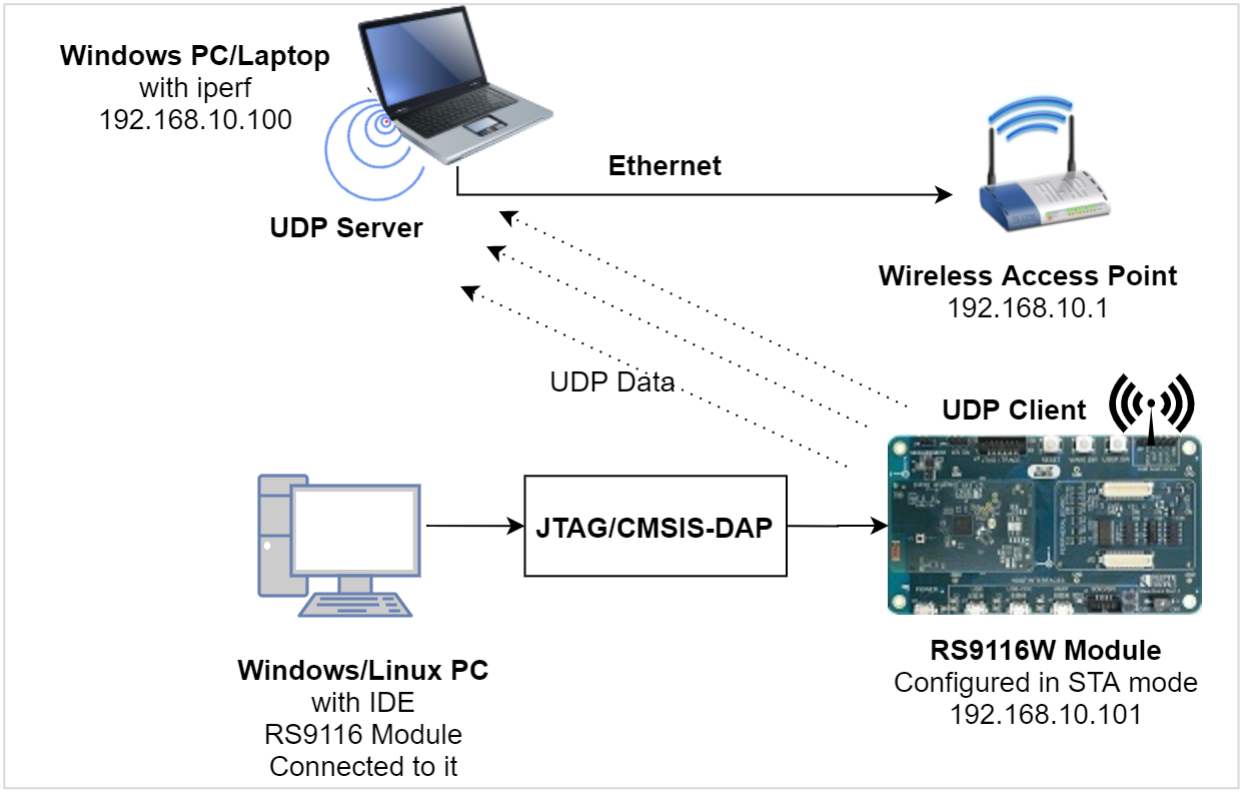


Figure 108: Module Configured in UDP Client Mode

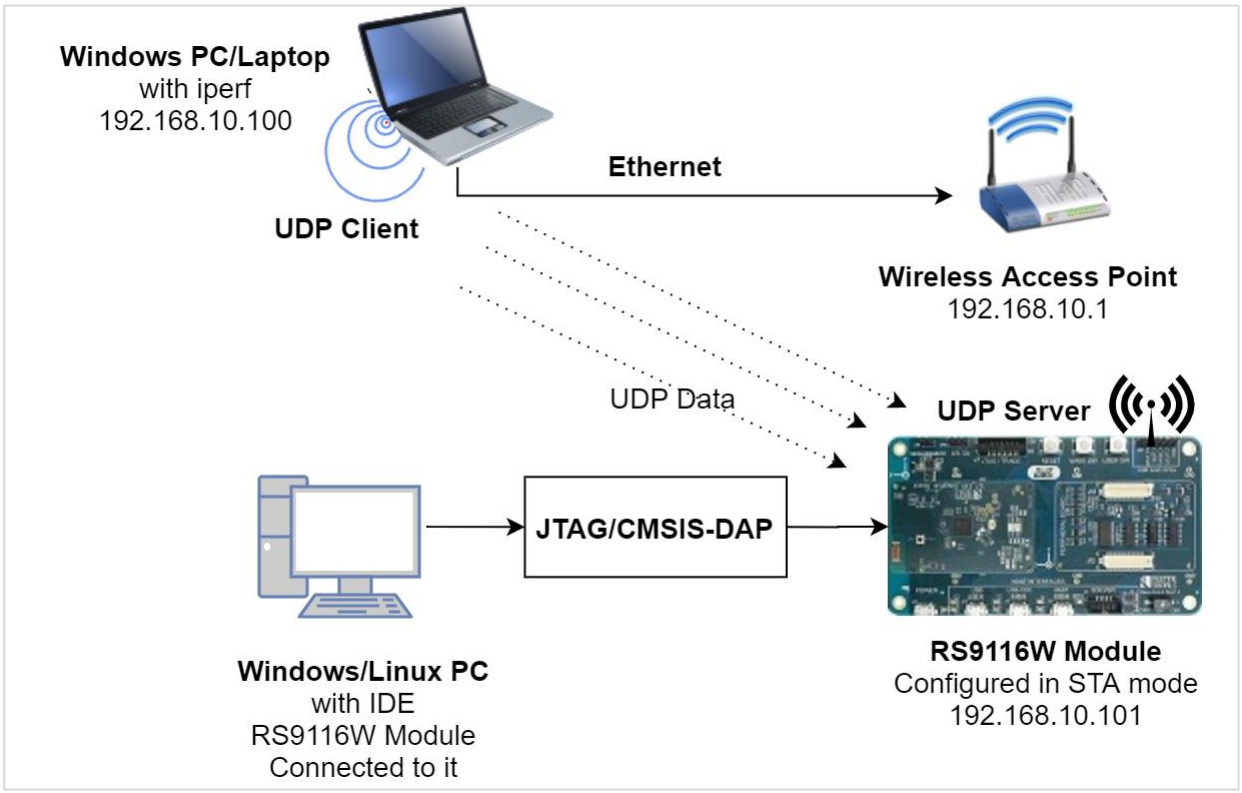


Figure 109; Module Configured in UDP Server Mode

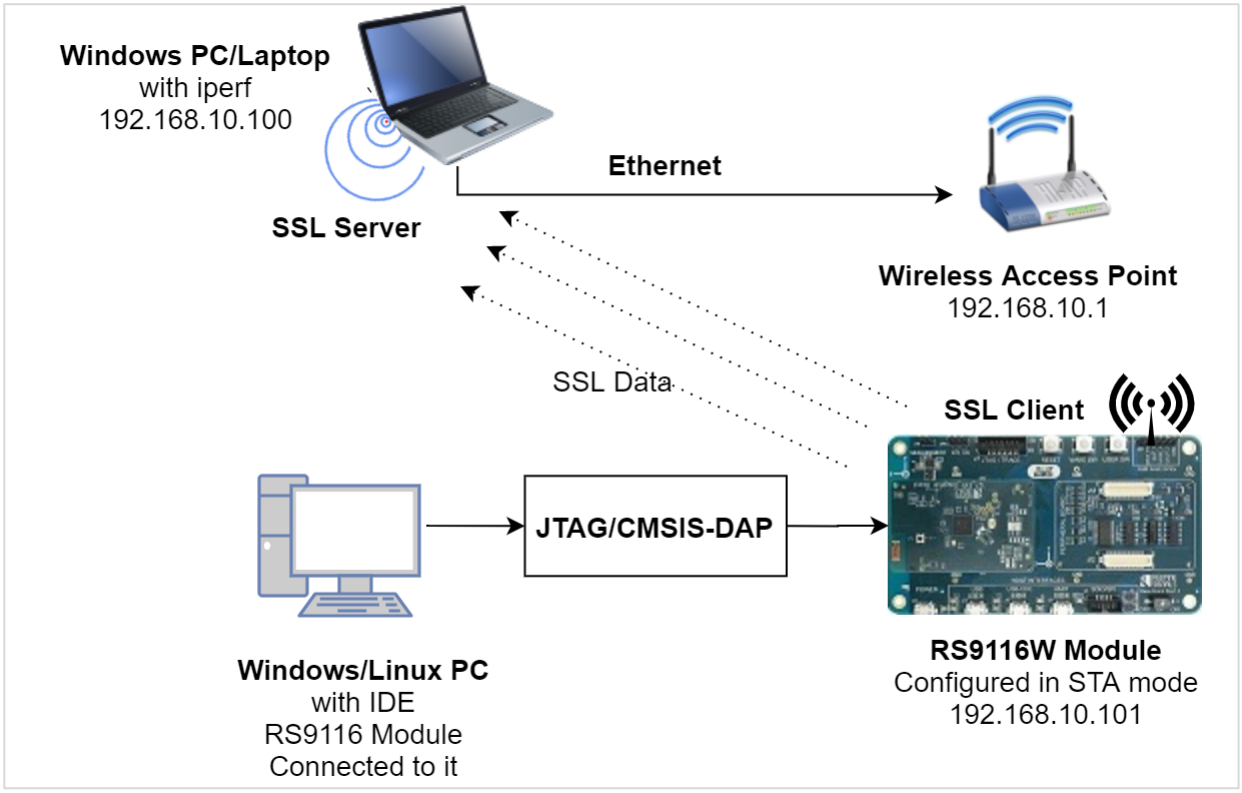


Figure 110: Module Configured in SSL Client Mode in TX

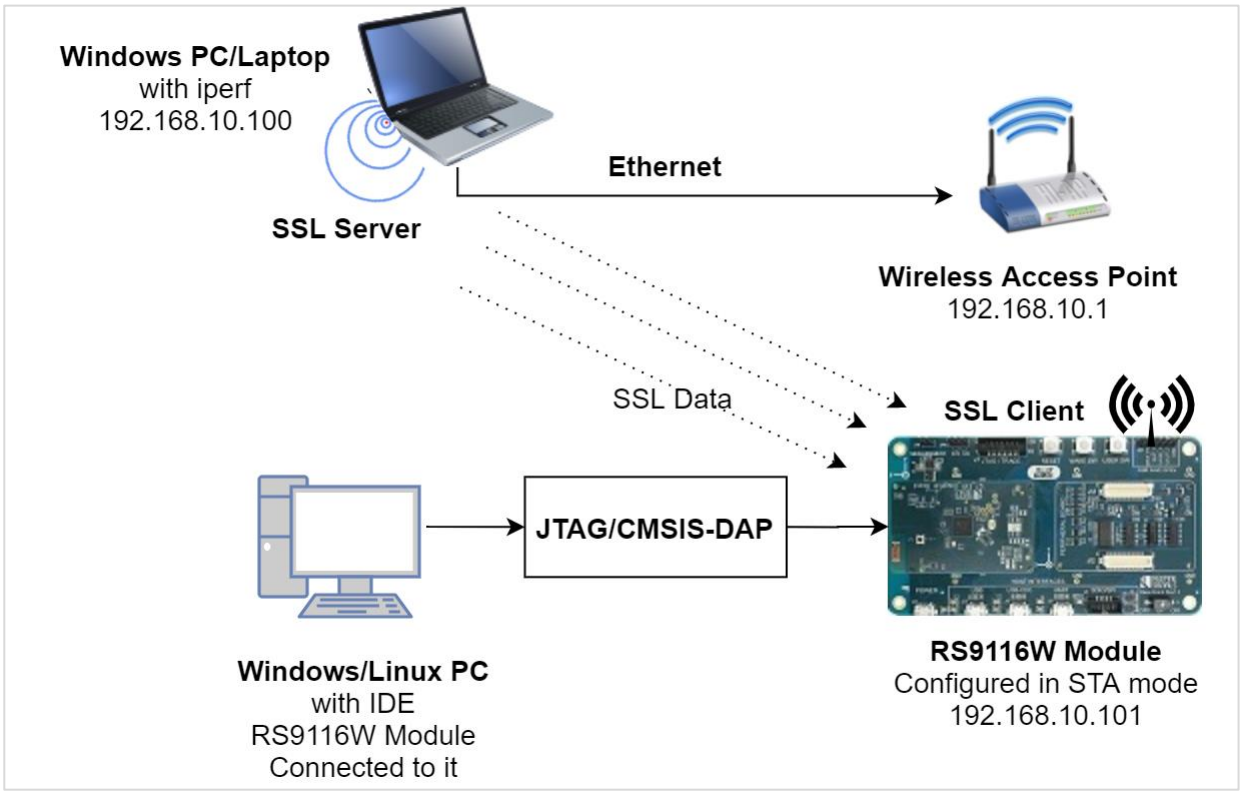


Figure 111: Module Configured in SSL Client Mode in RX

Description:

This application can be used to configure Silicon Labs module in UDP client / server or TCP client / server SSL client / server. To measure throughput, following configuration can be applied.

1. To measure TCP Tx throughput, module should be configured as TCP client.

2. To measure TCP Rx throughput, module should be configured as TCP server.
3. To measure UDP Tx throughput, module should be configured as UDP client.
4. To measure UDP Rx throughput, module should be configured as UDP server.
5. To measure SSL Tx throughput, module should be configured as SSL client.
6. To measure SSL Rx throughput, module should be configured as SSL server.

Configuration and Steps for Execution

Configuring the Application:

1. Open ***rsi_throughput_app.c*** file and update / modify the following macros,

   SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                              11
```

   SECURITY_TYPE refers to the type of security. Access point supports Open, WPA, WPA2 securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point is to be configured in WPA/WPA2 security modes.

```
#define PSK                                     "<psk>"
```

   Enable / Disable DHCP mode
   1-Enables DHCP mode (gets the IP from DHCP server)
   0-Disables DHCP mode

```
#define DHCP_MODE                               1
```

   To configure static IP address
   IP address to be configured to the device should be in long format and in little endian byte order.
   Example: To configure "192.168.10.1" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0**.

```
#define DEVICE_IP                               0X010AA8C0
```

   IP address of the gateway should also be in long format and in little endian byte order
   Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

   IP address of the network mask should also be in long format and in little endian byte order
   Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

2. To establish UDP/TCP connection and transfer/receive data to the remote socket configure the below macros

   Internal device port number

```
#define PORT_NUM                                <Local_port>
```

Port number of the remote server

```
#define SERVER_PORT                             <Remote_port_num>
```

IP address of the remote server

```
#define SERVER_IP_ADDRESS                       0x640AA8C0
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         15000
```

Application can use receive buffer size of 1400

```
#define BUFF_SIZE                               1400
```

Note: To measure SSL Tx/Rx throughput, BUFF_SIZE should be configured as 1370 bytes.

In rsi_throughput.c file, this macro is used to configure the Tx, Rx & Global buffers ratio for better throughputs.

```
#define TX_RX_RATIO_ENABLE          1
```

In rsi_throughput.c file, this macro will allow sockets with more buffers based on buffers availability. This option is valid for TCP data receive sockets.

```
#define RSI_HIGH_PERFORMANCE_SOCKET         1
```

Application can select throughput type as UDP Tx, UDP Rx, TCP Tx, TCP Rx, SSL Tx, SSL Rx. Following is macro need to use.

```
#define THROUGHPUT_TYPE                         UDP_TX
```

Following is macro used for throughput type selection

```
#define TCP_TX                                  0
#define TCP_RX                                  1
#define UDP_TX                                  2
#define UDP_RX                                  3
#define SSL_TX                                  4
#define SSL_RX                                  5
```

Note:

In AP mode, configure same IP address for both DEVICE_IP and GATEWAY macros.

3. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         DISABLE
#define RSI_FEATURE_BIT_MAP                     (FEAT_SECURITY_OPEN | FEAT_AGGREGATION )
#define RSI_TCP_IP_BYPASS                       DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP             EXT_FEAT_256k_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Connect Silicon Labs device to the Windows PC running Cocoox IDE.
2. Configure the macros in the files **rsi_throughput_app.c rsi_wlan_config.h**
3. To measure throughput, following configuration can be applied.

a) To measure UDP Tx throughput, module should be configured as UDP client. Open UDP server at remote port

iperf.exe -s -u -p <SERVER_PORT> -i 1

b) To measure UDP Rx throughput, module should be configured as UDP server. Open UDP client at remote port

iperf.exe -c <Module_IP> -u -p <Module_Port> -i 1 -b <Bandwidth>

c) To measure TCP Tx throughput,module should be configured as TCP client. Open TCP server at remote port.

iperf.exe -s -p <SERVER_PORT> -i 1

d) To measure TCP Rx throughput, module should be configured as TCP server. Open TCP client at remote port.

iperf.exe -c <Module_IP> -p <module_PORT> -i 1

e) To measure SSL Tx throughput, module should be configured as SSL client. Open SSL server at remote port.

Open ssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>

f) To measure SSL Rx throughput, module should be configured as SSL server. Open SSL client at remote port.

For running SSL Rx throughput, User has to run the below mentioned script, which is present in the Path: /release/host/sapis/examples/utilities/scripts

python SSL_Server_throughput_d.py <module_PORT>

4. To measure throughput, following configuration can be applied.
5. Build and launch the application.
6. After the program gets executed, the device would be connected to Access point having the configuration same as that of in the application and get.
7. The device which is configured as UDP/TCP/SSL server / client will connect to iperf server / client and sends / receives data continuously. It will print the throughput per second.

Note: If M4 frequency need to Switch higher clock then follow below steps.

step1: Call **switch_m4_frequency()** api after device initialization.

step2: Update systics to higher clock as **SysTick_Config(SystemCoreClock /1000).**

6.33 Transmit Test

Overview

While measuring the performance of 802.11 Wireless devices, packet error test has become today's choice for FCC certification.
The Transmit test application demonstrates how Silicon Labs device starts transmitting test in Burst mode which is used for FCC certification.

Sequence of Events

This Application explains user how to:

- Start transmission in Burst mode with different data rates, transmit power and lengths.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to initialize the selected host interface. The Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- RS9116W EVK
- Spectrum Analyzer

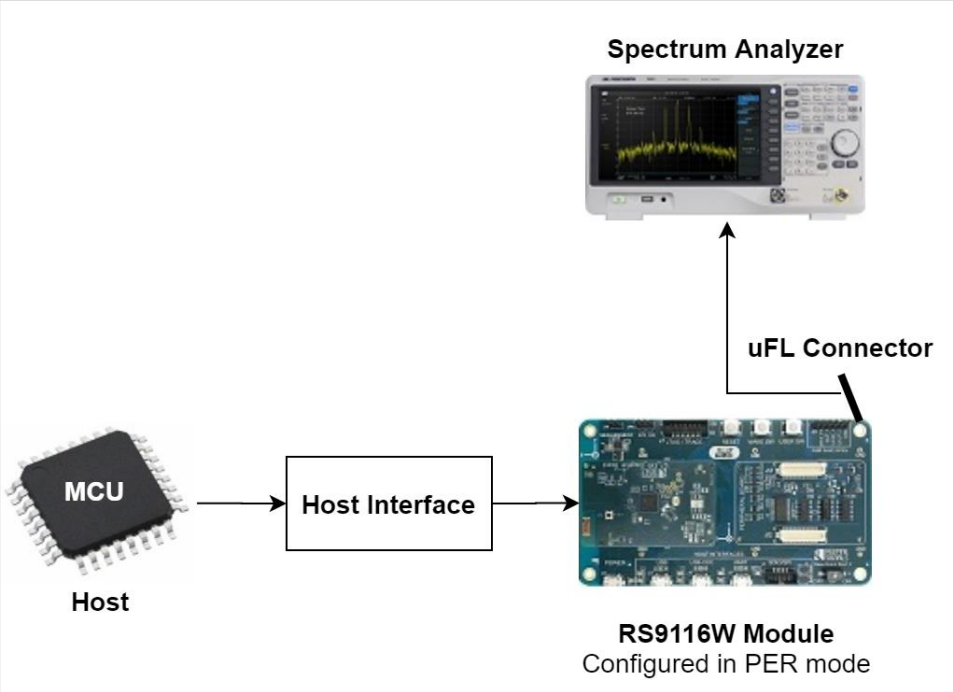


Figure 112: Setup Diagram for Transmit Test Example

Configuration and Steps for Execution

1. Open ***rsi* trasmit_test_app.c**_ file and update / modify following macros:

   To set TX power in dbm. The valid values are from 2dbm to 18dbm for WiSeConnectTM module.

```
#define RSI_TX_TEST_POWER                       4
```

   To set transmit data rate.

```
#define RSI_TX_TEST_RATE                        RSI_RATE_1
```

To configure length of the TX packet. Valid values are in the range of 24 to 1500 bytes in the burst mode and range of 24 to 260 bytes in the continuous mode.

```
#define RSI_TX_TEST_LENGTH                      30
```

To configure Burst mode or Continuous mode

```
#defineRSI_TX_TEST_MODE                         RSI_BURST_MODE
```

To configure the channel number in 2.4 GHz/5GHz.

```
#define RSI_TX_TEST_CHANNEL                     1
```

To select internal antenna or UFL connector,
0 - to select internal antenna or RF_OUT2
1 - to select UFL connector or RF_OUT1

```
#define RSI_ANTENNA                             1
```

To select antenna gain in db for 2.4GHz band. Valid values are from 0 to 10.

```
#define RSI_ANTENNA_GAIN_2G                      0
```

To select antenna gain in db for 5GHz band. Valid values are from 0 to 10.

```
#define RSI_ANTENNA_GAIN_5G                      0
```

2. Open **rsi_wlan_config.h** file and update/modify following macros:

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_256k_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

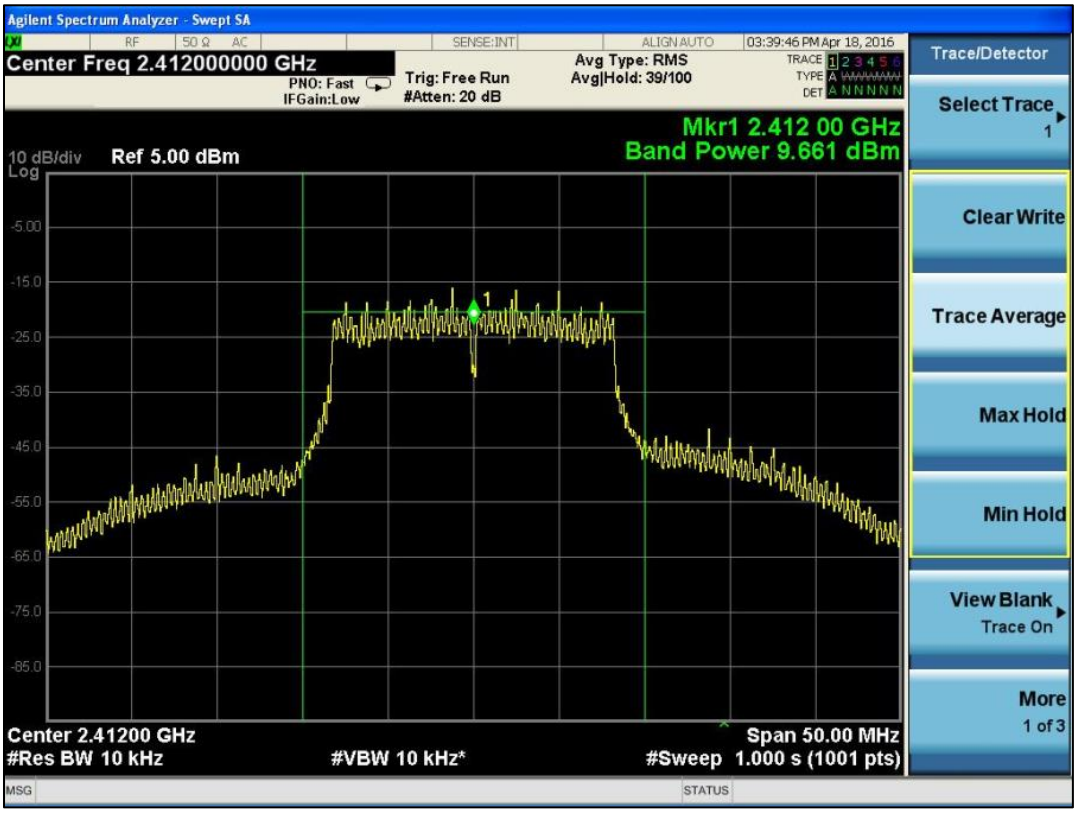
Executing the Application

1. After the program gets executed, the RS9116W device will start the transmit test with the given configuration.
2. Analyzer can be used to monitor the device behavior with certificate constraints.

   Refer the below image which shows when RS9116W device transmits packets in Burst mode with different Tx power and different transmission rates in channel 1 with length 30bytes.

   RSI_TX_TEST_POWER - 4dbm
   RSI_TX_TEST_RATE - 1Mbps
   RSI_TX_TEST_LENGTH - 30
   RSI_TX_TEST_MODE - BURST mode
   RSI_TX_TEST_CHANNEL - 1

RSI_TX_TEST_POWER - 12dbm
RSI_TX_TEST_RATE - 6Mbps
RSI_TX_TEST_LENGTH - 30
RSI_TX_TEST_MODE - BURST mode
RSI_TX_TEST_CHANNEL - 1



6.34 UDP Client

UDP Protocol Overview

UDP (USER Datagram protocol) is a connection less and non-stream oriented protocol for transferring data in either direction between a pair of users. In UDP there is no guarantee that the messages or packets sent would reach at all. No handshake in UDP Protocol because it is connection less protocol.

Overview

The UDP client application demonstrates how to open and use a standard UDP client socket and sends data to UDP server socket. Once it is configured as UDP client, it can establish and maintain a network conversation by means of application program for exchanging of data.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Open UDP Server socket at Access point using iperf application.
- Open UDP client socket in device
- Send data from Silicon Labs device to remote peer using opened UDP socket.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with Dev-C++ IDE / SPI/ USB) in case of WiSeConnect
- Silicon Labs Module
- Wi-Fi Access point
- Windows PC2
- UDP server application running in Windows PC2 (This application uses iperf application to open UDP server socket)

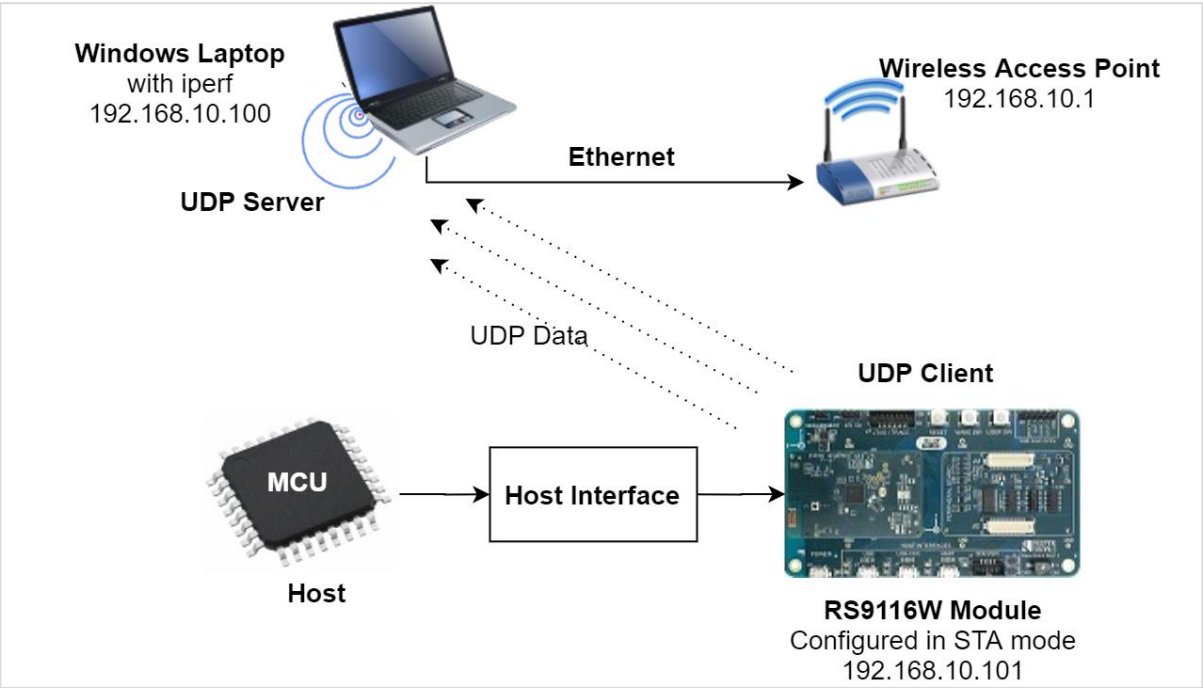


Figure 113: Setup Diagram for UDP Client Socket

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_udp_client.c*** file and update/modify following macros,

   SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels

```
#define CHANNEL_NO                          0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK and WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                       RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                 "<psk>"
```

   DEVICE_PORT port refers UDP client port number

```
#define DEVICE_PORT                         5001
```

   SERVER_PORT port refers remote UDP server which is opened in windows PC2.

```
#define SERVER_PORT                         5001
```

   SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
   IP address should be in long format and in little endian byte order.
   Example: To configure "192.168.10.100" as IP address, update the macro **DEVICE_IP** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS                   0x640AA8C0
```

   NUMEBR_OF_PACKETS refer how many packets to send from device to UDP server.

```
#define NUMBER_OF_PACKETS                   1000
```

   Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                     15000
```

   To configure IP address:
   DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                           1
```

Note:

If the user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.
(Or)
If the user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address which is to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                              0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order.
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                        RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                    FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                      RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP             TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP             FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP            EXT_FEAT_256k_MODE
```

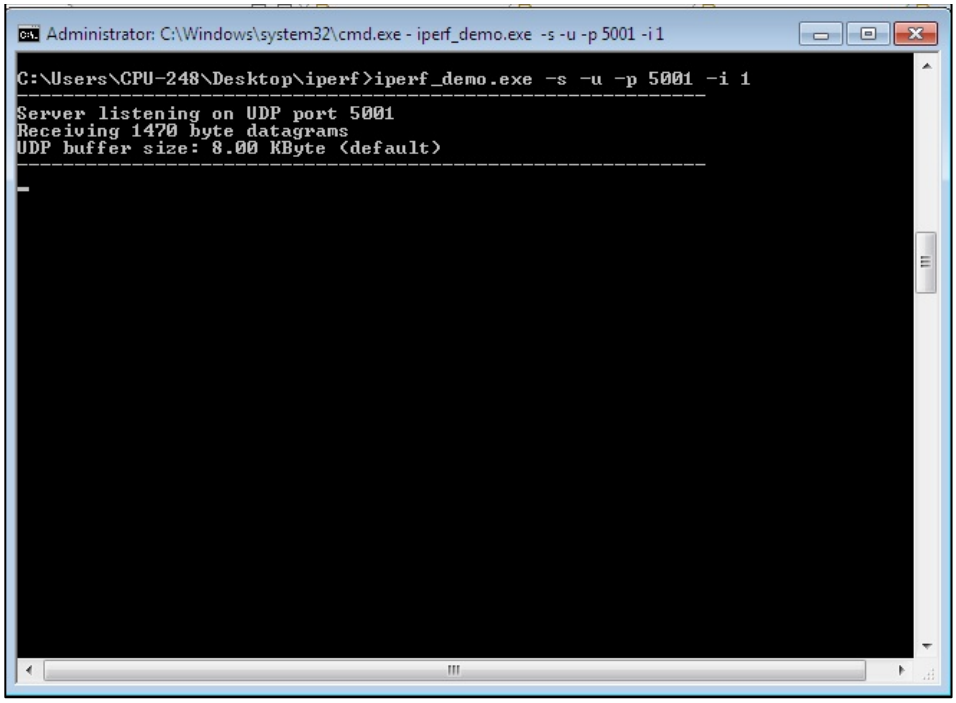
Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

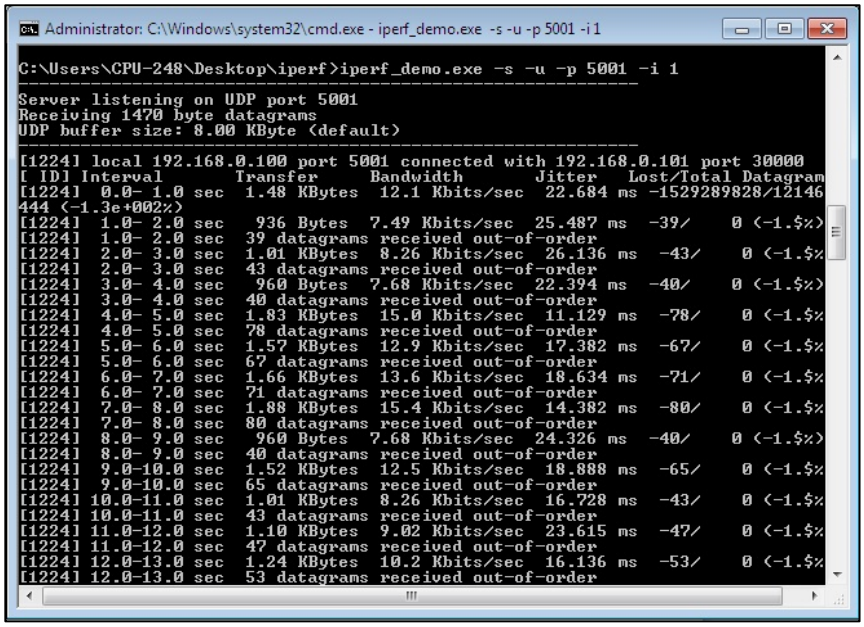
1. Configure the Access point in OPEN / WPA-PSK/WPA2-PSK mode in order to connect Silicon Labs device in STA mode.
2. Open UDP server application using iperf application in Windows PC2 which is connected to Access point through LAN.

iperf_demo.exe –s -u -p <SERVER_PORT> -i 1



3. After program gets executed, The Silicon Labs device would scan and connect to Access point and get IP.

4. After successful connection, the device STA connects to UDP server socket opened on Windows PC2 using UDP client socket and sends configured **NUMBER_OF_PACKETS** to remote UDP server. Please refer the below image for reception of UDP data on UDP server.



6.35 UDP Sever

UDP Protocol Overview

UDP (USER Datagram protocol) is a connection less and non-stream oriented protocol for transferring data in either direction between a pair of users.

Overview

The UDP server application demonstrates how to open and use a standard UDP server socket and receives data on socket sent by remote peer. Once it is configured as UDP server, it can establish and maintain a network conversation by means of application program for exchanging of data.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Open UDP server socket in device
- Send UDP data from remote peer to Silicon Labs device on opened port using UDP client socket
- Receive data from UDP client socket.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- WiFi Access point
- Windows PC2
- UDP client application running in Windows PC2 (This application uses iperf application to open UDP client socket)

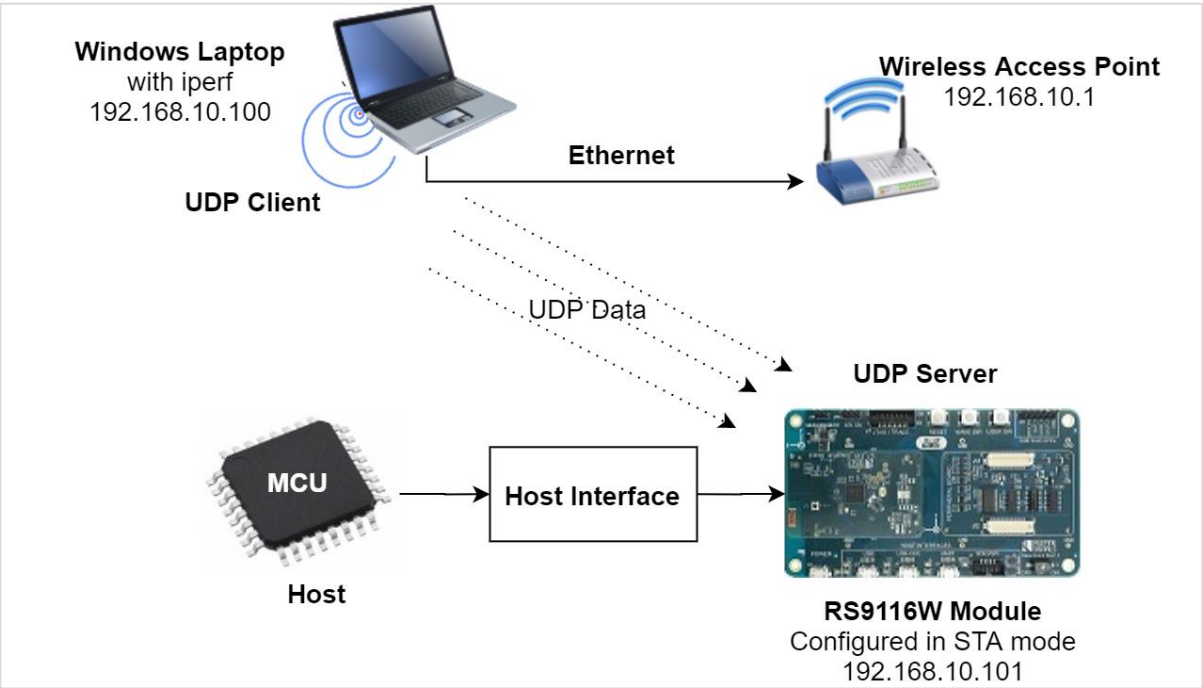


Figure 114: Setup Diagram for UDP Server Socket Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_udp_server.c*** file and update/modify following macros,

   SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

```
#define CHANNEL_NO                      0
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                   RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                             "<psk>"
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                     5001
```

Receive data length

```
#define RECV_BUFFER_SIZE                <recv_buf_size>
```

NUMEBR_OF_PACKETS refer to how many packets to receive from UDP client

```
#define NUMBER_OF_PACKETS               <no of packets>
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                 15000
```

To configure IP address:
DHCP_MODE refer to whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                       0
```

Note:

Configure STATIC IP to WiSeConnect device. So that user knows the IP address of WiSeConnect device to establish UDP connection from remote peer. In case of DHCP, User has to know the assigned IP by parsing IPCONF response.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.0.101" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X6500A8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x0100A8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

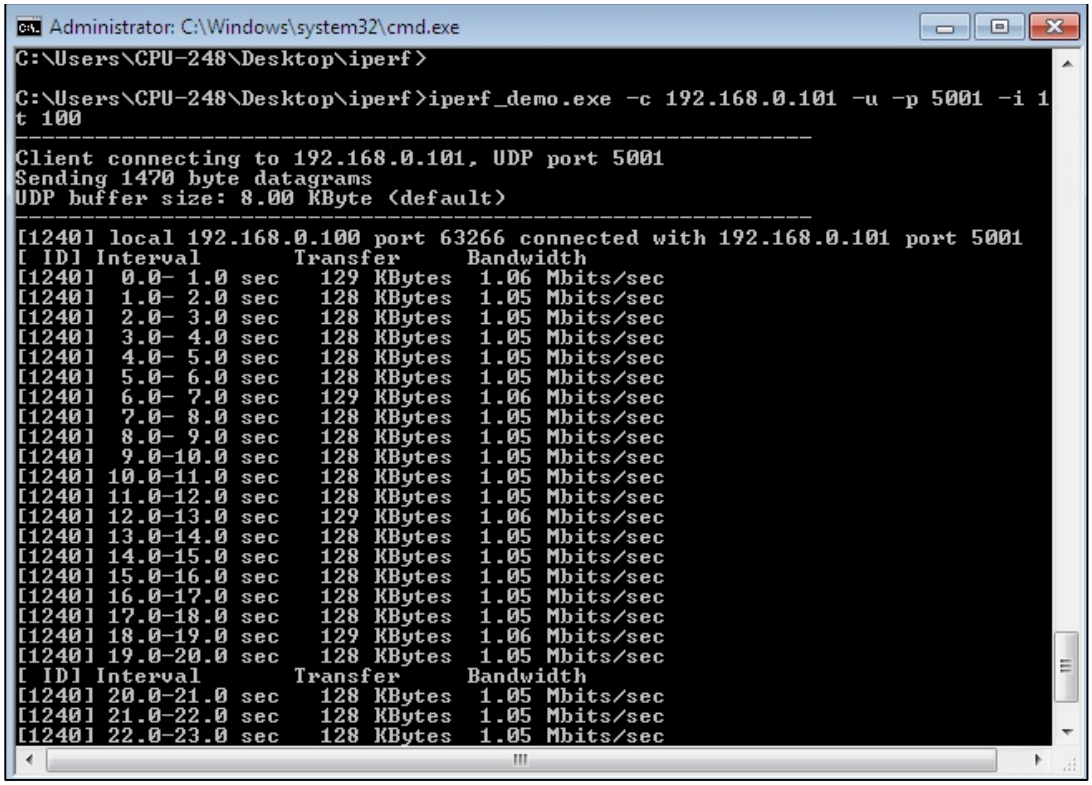
Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. After the program gets executed, the Silicon Labs device will be connected to Access point having the configuration as same as that of in the application and get IP.
2. Open UDP client using iperf from windows PC2
   Users can download application from the link below:
   https://iperf.fr/iperf-download.php#windows
3. Connect to UDP server opened on Silicon Labs device on port number **DEVICE_PORT** using the following command:
   Iperf_demo.exe –c <DEVICE_IP> -u -p <DEVICE_PORT> -i 1 –t 100

4. Silicon Labs Device will receive the number of packets configured in **NUMBER_OF_PACKETS** from iperf UDP client.



6.36 User Gain Table

Overview

While measuring the performance of 802.11 Wireless devices, packet error test has become today's choice for FCC certification.
The user config gain table application demonstrates how Silicon device starts transmitting test in Burst mode with the user configured gain values which is used for FCC certification.

Sequence of Events

This Application explains user how to:

- Start transmission in Burst mode with different data rates, transmit power and lengths with user gain tables values.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ SPI/ USB) in case of WiSeConnect
- RS9116W EVK
- Spectrum Analyzer

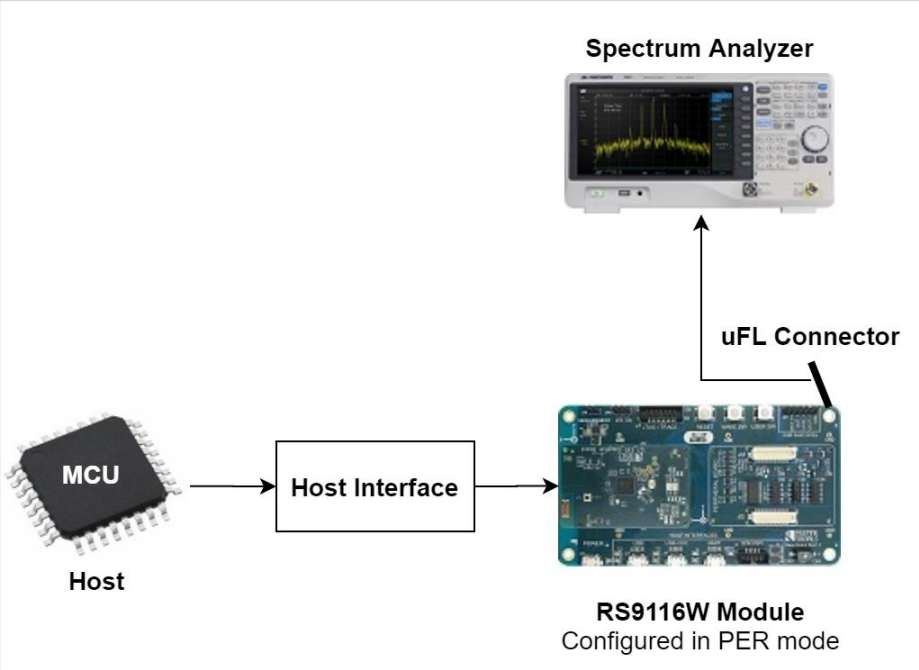


Figure 115: Setup Diagram for User Config Gain Table

Configuration and Steps for Execution

1. Open ***rsi*_user_config_gain_table.c_** file and update / modify following macros:

   To set TX power in dbm.

```
#define RSI_TX_TEST_POWER                       127
```

   To set transmit data rate.

```
#define RSI_TX_TEST_RATE                        RSI_RATE_1
```

   To configure length of the TX packet. Valid values are in the range of 24 to 1500 bytes in the burst mode and range of 24 to 260 bytes in the continuous mode.

```
#define RSI_TX_TEST_LENGTH                      1000
```

   To configure Burst mode or Continuous mode

```
#define RSI_TX_TEST_MODE                         RSI_BURST_MODE
```

   To configure the channel number in 2.4 GHz/5GHz.

```
#define RSI_TX_TEST_CHANNEL                     1
```

   To select internal antenna or UFL connector,
   0 - to select internal antenna or RF_OUT2
   1 - to select UFL connector or RF_OUT1

```
#define RSI_ANTENNA                             1
```

   To select antenna gain in db for 2.4GHz band. Valid values are from 0 to 10.

```
#defineRSI_ANTENNA_GAIN_2G                      0
```

   To select antenna gain in db for 5GHz band. Valid values are from 0 to 10.

```
#defineRSI_ANTENNA_GAIN_5G                      0
```

```
//! BAND : 1 --> 2.4Ghz, 2 --> 5.0Ghz
#define BAND                            1
```

```
//! RSI_BANDWIDTH: RSI_BW_20MHZ --> 20Mhz, RSI_BW_40MHZ --> 40Mhz
#define RSI_BANDWIDTH             RSI_BW_20MHZ
```

```
//! Transmission packets duration on AIR
#define TRANSMIT_TIME                   10000  //! Time in milliseconds
```

```
uint8_t gain_table_payload[] = {}; //! Fill the user gain table values in the below mentioned way.

<TABLE NAME>[] = { //<COMMENTS if any>    <NO.of Regions>,
        <REGION NAME 1>, <NO.OF CHANNELS IN THIS REGION>,
            <CHANNEL NUMBER 1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
            <CHANNEL NUMBER 2>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
            .
            .
            <CHANNEL NUMBER m-1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n
RATE>,
            <CHANNEL NUMBER m>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
        <REGION NAME 2>, <NO.OF CHANNELS IN THIS REGION>,
            <CHANNEL NUMBER 1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
            <CHANNEL NUMBER 2>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
            .
            .
            <CHANNEL NUMBER m-1>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n
RATE>,
            <CHANNEL NUMBER m>, <MAX POWER FOR b RATE>, <MAX POWER FOR g RATE>, <MAX POWER FOR n RATE>,
        <REGION NAME 3>, <NO.OF CHANNELS IN THIS REGION>,
        .
        .
};
```

2. Open **rsi_wlan_config.h** file and update/modify following macros:

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_256k_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
#define  RSI_SET_REGION_SUPPORT                 RSI_ENABLE
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

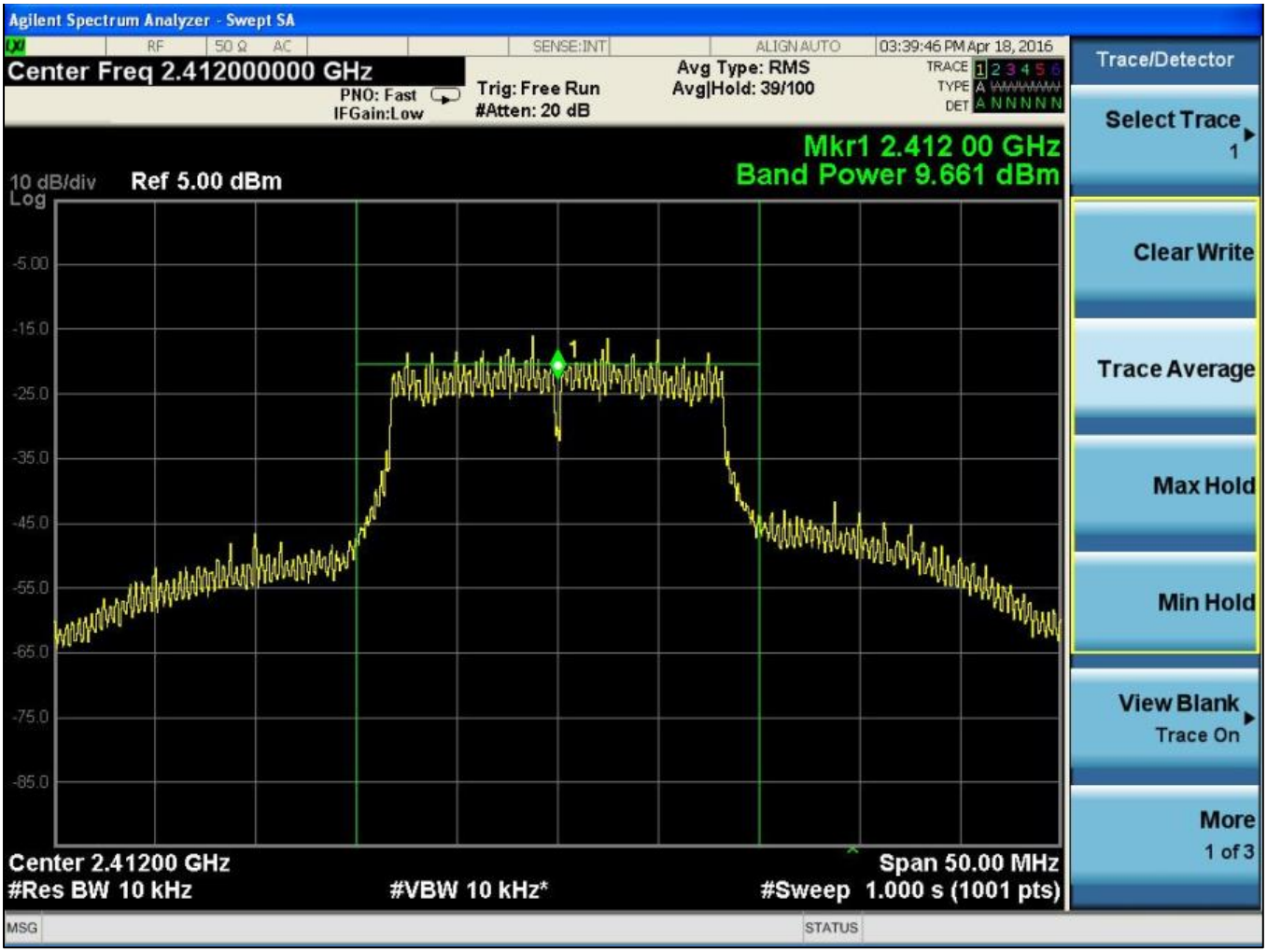
Executing the Application

1. After the program gets executed, the Silicon device will start the transmit test with the given configuration.
2. Analyzer can be used to monitor the device behavior with certificate constraints.

   Refer the below image which shows when silicon device transmits packets in Burst mode with different Tx power and different transmission rates in channel 1 with length 30bytes.

   RSI_TX_TEST_POWER - 4dbm
   RSI_TX_TEST_RATE - 1Mbps
   RSI_TX_TEST_LENGTH - 30
   RSI_TX_TEST_MODE - BURST mode
   RSI_TX_TEST_CHANNEL - 1

RSI_TX_TEST_POWER - 12dbm
RSI_TX_TEST_RATE - 6Mbps
RSI_TX_TEST_LENGTH - 30
RSI_TX_TEST_MODE - BURST mode
RSI_TX_TEST_CHANNEL - 1



6.37 Web Socket Client

WebSocket Overview

WebSocket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. WebSocket enables streams of messages on top of TCP.

Overview

This application demonstrates how to configure device in client mode to open Web socket to transmit data over Websocket to Web Server.

Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Connect to Web server opened on remote peer using web socket client
- Send data to web socket server

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with Dev-C++ IDE / SPI/ USB) in case of WiSeConnect

- Silicon Labs module
- Wi-Fi Access point
- Linux/Window PC with Web socket server and open SSL support (This application using no-poll server for web server)

Note:

Download **No-Poll server** from the below link:
http://www.aspl.es/nopoll/downloads/nopoll-0.3.2.b232.tar.gz

Note:

Installation of open SSL is needed also, SSL lib files if not installed in prior.

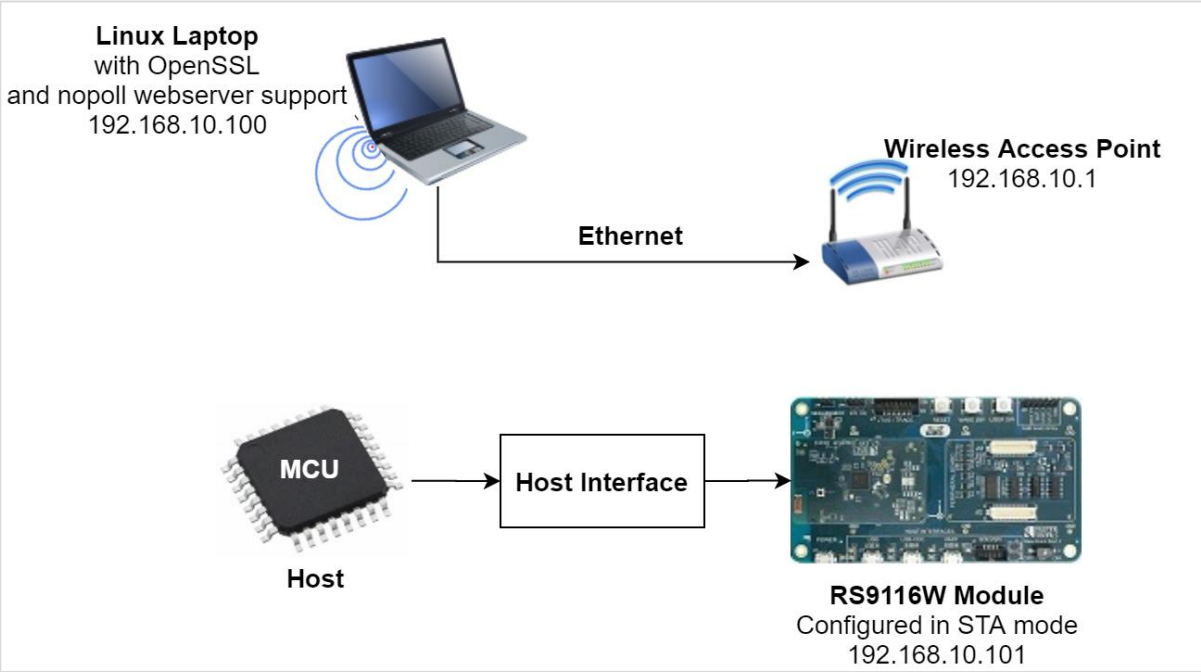


Figure 116: Setup Diagram for Web Socket Example

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_websocket_client_app.c*** file and update/modify following macros:

   SSID refers to the name of the Access point.

```
#define SSID                              "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0,device will scan all channels.

```
#define CHANNEL_NO                        0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.

   Valid configuration is:

   RSI_OPEN - For OPEN security mode

RSI_WPA - For WPA security mode

RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                    RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                              "<psk>"
```

CONNECTION_CLOSE_OPCODE refers web socket close frame, to test opcode 8 added test case 2 in ***rsi_websocket_client_app.c***

```
#define CONNECTION_CLOSE_OPCODE                8
```

To Load certificate

```
#define LOAD_CERTIFICATE                 1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.
By default, application loading **"ca-cert.pem"** certificate if **LOAD_CERTIFICATE** is enable. In order to load different certificate, user has to follow the following steps:

- rsi_wlan_set_certificate API expects the certificate in the form offline array. So, convert the pem certificate into linear array form using python script provided in the release package "**sapis/examples/utilities/certificates/certificate_script.py**"

Example: If the certificate is WiFi-user.pem, Give the command in the following way **python certificate_script.py ca-cert.pem**
Script will generate WiFi user.pem in which one linear array name dca cert contains the certificate.

- After conversion of certificate, update ***rsi_ssl_client.c*** source file by including the certificate file and by providing *the* required parameters to rsi_wlan_set_certificate API.

Note:

Once certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.
So define LOAD_CERTIFICATE as 0, if certificate is already present in the device.

Note:

All the certificates are given in the release package.

Path:*sapis/examples/utilities/certificates*

To Open Websocket client:

FLAGS refer to open normal web socket or web socket over SSL with IPv4 or IPv6

If User wants to open normal web socket client set FLAGS to 0 or user wants to open Web socket over SSL then set FLAGS to 2 (WEB SOCKET_SSL). Default configuration is Normal Web socket client

```
#define FLAGS                            0
```

Port number of there mote web socket server. Default configuration of server port number is Normal Web socket server.

```
#define SERVER_PORT                     1234
```

Note:

If user wants to open Web socket over SSL then update SERVER_PORT macro with 1235 or 1236 as in no poll SSL web socket server is running on port numbers 1235 and 1236.

IP address of the remote web socket server
IP address should be configured in long format and in little endian byte order.
Example: To configure **"192.168.10.101"** as IP address, update the macro **SERVER_IP** as **0x650AA8C0**.

```
#define SERVER_IP                       0x650AA8C0
```

Web socket resource name, maximum 50 characters

```
#define WEB_SOCKET_RESOURCE_NAME        "<NULL>"
```

Websocket host name, maximum 50 characters

```
#define WEB_SOCKET_HOST_NAME            "<NULL>"
```

Message to send remote server

```
#define MESSAGE                         "<this is web socket>"
```

FIN_BIT is used to indicate whether it is the last packet or not.
In this example, **FIN_BIT** is setting in last packet of configured number of packets (**NUMBER_OF_PACKETS)**.
After receiving packet with FIN BIT set, web socket server sends back the received data to WiSeConnect device.

```
#define FIN_BIT                         128
```

Number of packets to send

```
#define NUMBER_OF_PACKETS               1000
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                 15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                       1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                       0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                         0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                         0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_SSL)
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Install no-poll server in Linux/Windows PC and run Websocket server. Please follow the below steps to run no-poll server in LINUX PC,
    - Download No-Poll server from below link,
      http://www.aspl.es/nopoll/downloads/nopoll-0.3.2.b232.tar.gz
    - Unzip the package and do ". /configure", "make" and "make install"**.**
    - After successful installation, go to test folder to run sample websocket server.
    - If user wants to test websocket over SSL, copy server-cert.pem and server-key certificates from release package (sapis/examples/utilities/certificates) to test folder.
    - Open nopoll-regression-listener.c file and change certificates to server-cert.pem and server-key.pem in nopoll_listener_set_certificate(listener2, "server-cert.pem","server-key.pem",NULL) and nopoll_ctx_set_certificate(ctx,NULL,"server-cert.pem","server-key.pem",NULL) functions.
    - Default nopoll server can receive maximum data length (allowed by client firmware) but can print only 99 bytes but if you want to print whole data make change in respective place in nopoll-regression-listener.c.

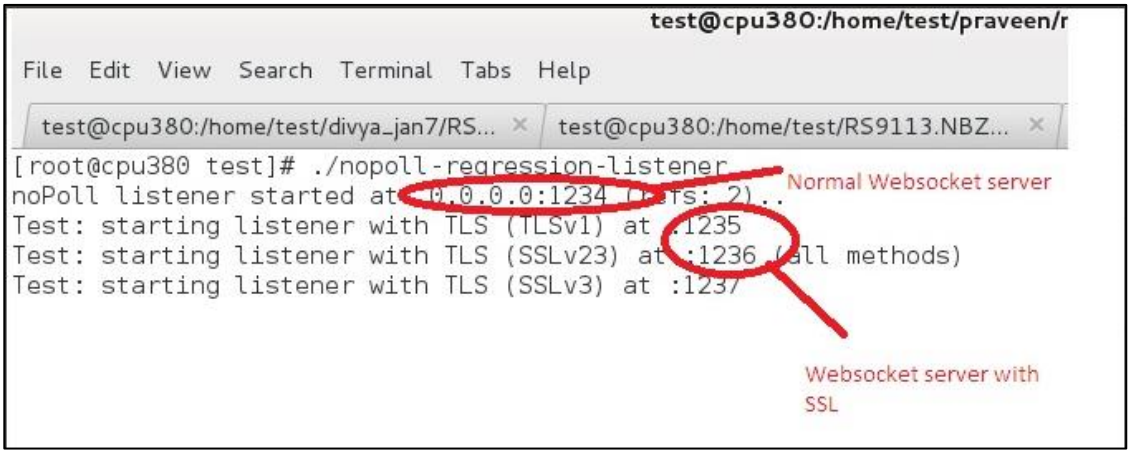
- Compile the source code by giving "make" command in test folder.
- Run server by giving "./nopoll-regression-listener" command.
- After running the nopoll server, it will open Four sockets on port numbers 1234, 1235, 1236 and 1237.

  1234 – For Normal Websocket server

  1235 – For TLSv1 websocket server

  1236 – For SSLv23 websocket server

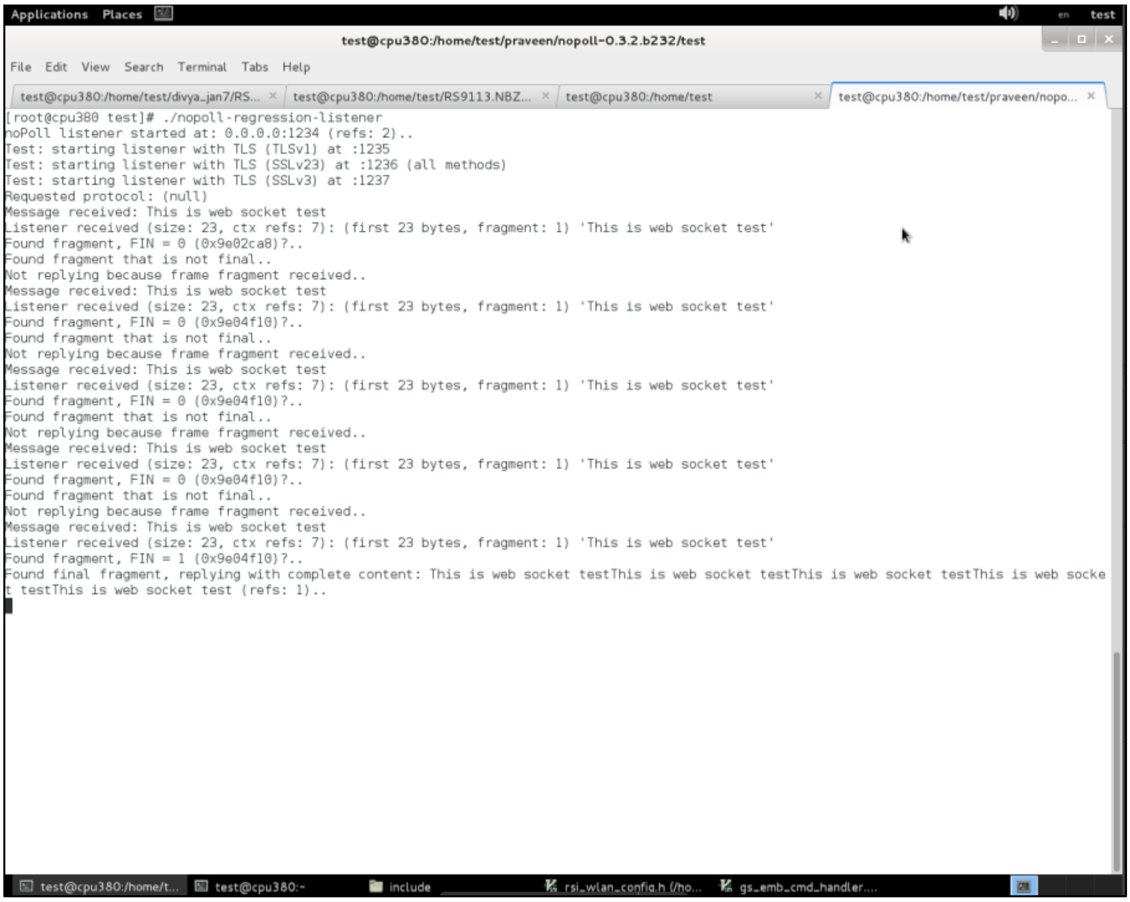
  12367– For SSLv3 websocket server



3. After the program gets executed, the device would be connected to access point having the configuration same as that of in the application and get IP.

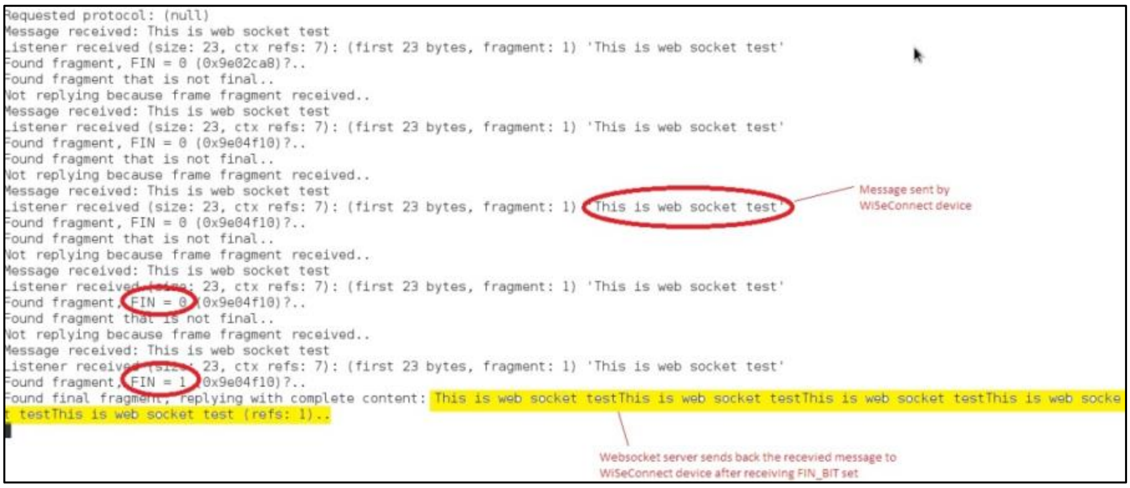
4. The Device which is configured as Websocket client will connect to remote SSL server and sends number of packets configured in **NUMBER_OF_PACKETS**.

Please refer the below image for Data Rx on Websocket server.



5. From application, set FIN_BIT in last data packet. After receiving data packet with FIN_BIT set, WebSocket server sends back the data received until FIN_BIT set to Silicon Labs device.

Please find the below image for webSocket server sending back data to the Silicon Labs device after receiving packet with FIN_BIT set,



6. The device will receive the message sent by Websocket server and initiate connection close to websocket server. Refer the below image for connection close at websocket server.

```
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 1 (0x9e04f10)?..
Found final fragment, replying with complete content: This is web socket testThis is web socket test
t testThis is web socket test (refs: 1)..
Reg test: called connection close (TLS: 1)..
```

6.38 WEP Security

WEP protocol Overview

Wired Equivalent Privacy (WEP) is a security protocol for wireless networks that encrypts transmitted data . The disadvantage is that without any security, your data can be intercepted without difficulty. WEP has three settings: Off (no security), 64-bit (weak security), 128-bit (a bit better security). WEP is not difficult to crack, and using it reduces performance slightly. However, WEP was an early attempt to secure wireless networks, and better security is now available such as DES, VPN, and WPA.

Overview

The WEP security application demonstrates how to connect to a WEP secured Access point and open a standard TCP client socket and sends data on server socket with the Silicon Labs device.

Sequence of events

- Configure an Access point in WEP secured mode
- Connect the Silicon Labs device to the WEP secured AP and get IP through DHCP
- Server socket at Access point has to be opened by means of application program like iperf
- TCP client socket would be connected to the server socket that is opened
- Once the connection is established, Data is sent to the TCP server

Example setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- WiFi Access point
- Access Point with WEP security
- Application program like iperf
- Windows PC2
- TCP server application running in Windows PC2 (This application uses iperf application to open TCP server socket)

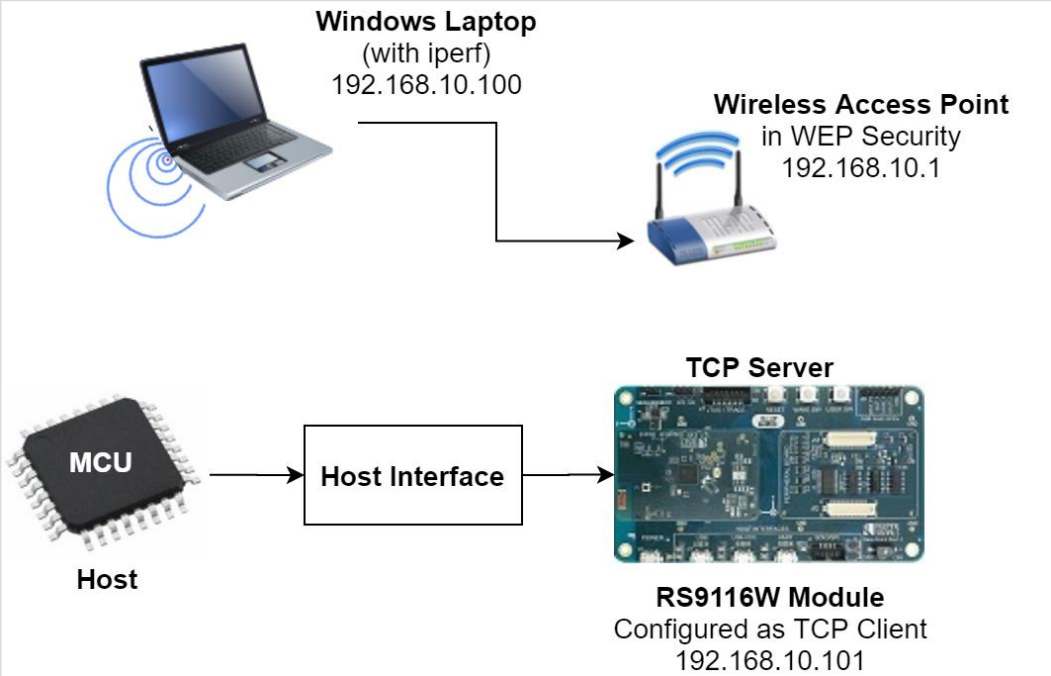


Figure 117: Setup Diagram for WEP Security Example

Configuration and Steps for Execution

Configuring the Application:

1. Open **rsi_tcp_client.c** files and update/modify following macros:

   SSID refers to the name of the Access point to be created.

```
#define SSID                          "<REDPINE_AP>"
```

   CHANNEL_NO refers to particular channel used to scan by the device. If channel is 0 then it will scan all channels.

```
#define CHANNEL_NO                    <channel_num>
```

   SECURITY_TYPE refers to type of security WEP (RSI_WEP)**WEP_INDEX** refers to the one of the four keys to be used for connection **WEP_KEY0, WEP_KEY1, WEP_KEY2, WEP_KEY3** are the keys , they can be 10 bytes or 26 bytes.

```
#define SECURITY_TYPE                 RSI_WEP
#define WEP_INDEX                     "<index>"
#define WEPKEY0                       "<wep key 0>"
#define WEPKEY1                       "<wep key 1>"
#define WEPKEY2                       "<wep key 2>"
#define WEPKEY3                       "<wep key 3>"
```

   DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                   5001
```

   SERVER_PORT port refers remote TCP server port number which is opened in windows PC2.

```
#define SERVER_PORT                   5001
```

   SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
   IP address should be in long format and in little endian byte order.

   Example: To configure "192.168.10.100" as IP address, update the macro **DEVICE_IP** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS             0x640AA8C0
```

NUMEBR_OF_PACKETS refer to how many packets to send from device to TCP server

```
#define NUMBER_OF_PACKETS              1000
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                      1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                      0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                        0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                        0x00FFFFFF
```

2. Open **rsi_wlan_config.h** file and update/modify following macros:

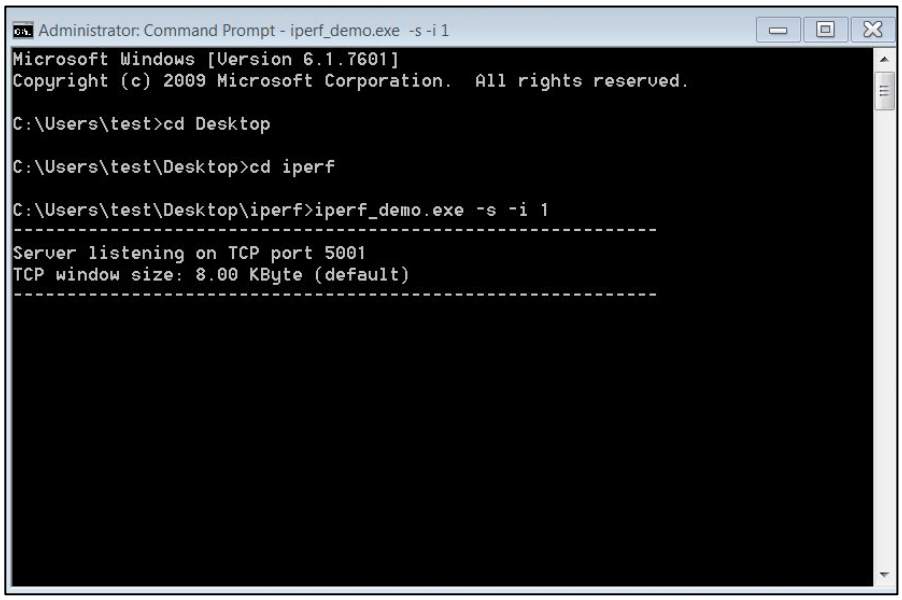
```
#define CONCURRENT_MODE                RSI_DISABLE
#define RSI_FEATURE_BIT_MAP            FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS              RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP     EXT_FEAT_256k_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in security WEP (RSI_WEP)**WEP_INDEX** refers to the one of the four keys to be used for connection **WEP_KEY0, WEP_KEY1, WEP_KEY2, WEP_KEY3** are the keys , they can be 10 bytes or 26 bytes.
2. Open an iperf, Users can download application from the link below, https://iperf.fr/iperf-download.php#windowsTCP server listening on port SERVER_PORT on remote machine in the following format.
   iperf.exe -s -p <SERVER_PORT> -i 1



3. After program gets executed, Silicon Labs Device would scan and connect to Access point and get IP. The Device which is configured as TCP client will connect to iperf server and sends number of packets configured in NUMBER_OF_PACKETS.

6.39 Wireless Firmware Upgradation

Overview

The Wireless Firmware upgrade application demonstrates how WiSeConnect device would be created as Access point and allow stations to connect to update the firmware through webpage.

Sequence of Events

This Application explains user how to:

- Silicon Labs Device starts as an Access point
- Allows stations to connect to update firmware through webpage.
- A Laptop having WiFi card can be used as Wireless station.
- Silicon Labs device creates a network and acts as router between the connected stations
- Upgrade the received Firmware into the device.

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Silicon Labs Module
- Wireless Access point

- Linux PC with TCP server application (TCP server application providing as part of release package)

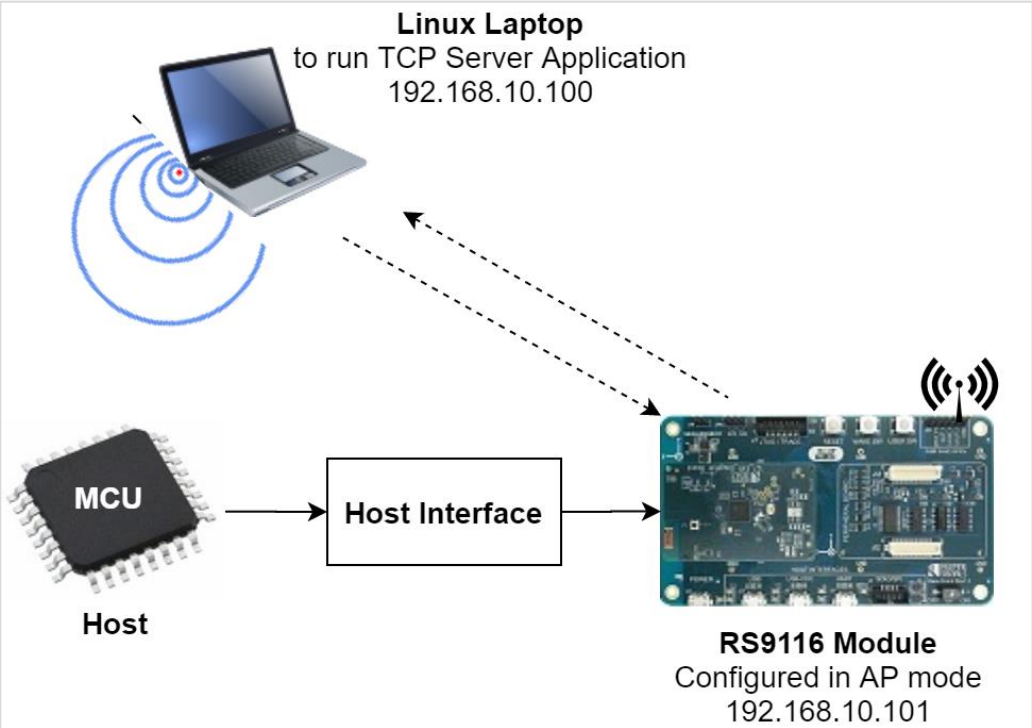


Figure 118: Setup Diagram for Wireless Firmware Upgrade

Configuration and Steps for Execution

Configuring the Application

1. Open rsi_wfup_app.c file and update/modify following macros: From given configuration,

   SSID refers to the name of the Access point

```
#define SSID                                    "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                              <0>
```

   SECURITY_TYPE refers to the type of security .Access point supports Open, WPA, WPA2 securities

```
#define SECURITY_TYPE                           <RSI_OPEN>
```

   ENCRYPTION_TYPE refers to the type of Encryption method .Access point supports Open,TKIP, CCMP methods

```
#define ENCRYPTION_TYPE                         <RSI_NONE>
```

   PSK refers to the secret key if the Access point to be configured in WPA/WPA2 security modes.

```
#define PSK                                     "<psk>"
```

   BEACON_INTERVAL refers to the time delay between two consecutive beacons

```
#define BEACON_INTERVAL                                 <beacon_interval>
```

To configure DTIM interval of the Access Point

```
#define DTIM_INTERVAL                                   <dtim_interval>
```

FLAGS:

```
#define FLAGS                                           WEB_PAGE_ASSOCIATED_TO_JSON
```

2. To configure IP address
   IP address to be configured to the device should be in long format and in little endian byte order.
   Example: To configure “192.168.10.1” as IP address, update the macro DEVICE_IP as 0x010AA8C0.

```
#define DEVICE_IP                                       0X010AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure “192.168.10.1” as Gateway, update the macro GATEWAY as 0x010AA8C0

```
#define  GATEWAY                                        0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure “255.255.255.0” as network mask, update the macro NETMASK as 0x00FFFFFF

```
#define   NETMASK                                       0x00FFFFFF
```

3. Open **rsi_wlan_config.h** file and update/modify following macros:

```
#define CONCURRENT_MODE                                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                      (TCP_IP_FEAT_DHCPV4_SERVER
|TCP_IP_FEAT_HTTP_SERVER)
#define RSI_CUSTOM_FEATURE_BIT_MAP                      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_256K_MODE
#define RSI_BAND                                        RSI_BAND_2P4GHZ
```

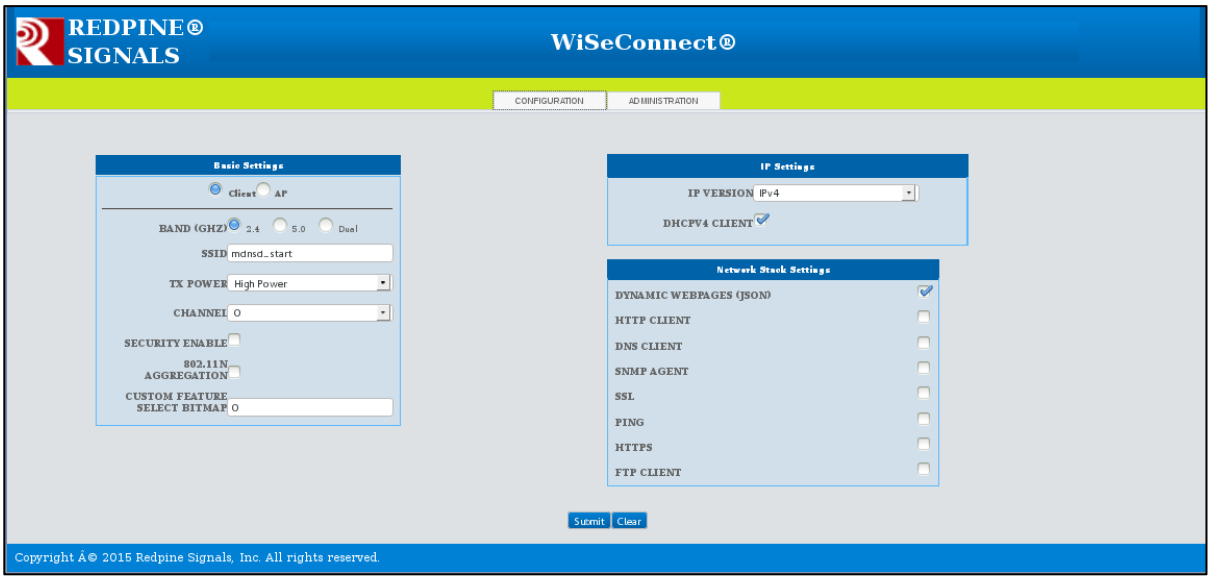
Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Connect WiSeConnect device to the Windows PC running Cocoox IDE.
2. Configure the macros in the files rsi_wfup_app.c & rsi_wlan_config.h
3. Build and launch the application.
4. After the program gets executed, Silicon Labs Device would be created as Access point and starts Beaconing.
5. TCP server socket is created and wait for accepting any TCP connection request on that port.
6. Now connect Laptop as WiFi station and open Webpage by typing the IP address of the module.


Currently connected to:
REDPINE_AP 5
No Internet access
Wireless Network Connection 33
REDPINE_AP
Connected
I2test_ulp
UDP_BULK
?ADO27wT^b6=YXBEEg{haw\
BIN2
MODULE
TPLINK_WIN7
Open Network and Sharing Center


REDPINE®
SIGNALS
WiSeConnect®
CONFIGURATION
ADMINISTRATION
Basic Settings
Client
AP
BAND (GHZ)
2.4
5.0
Dual
SSID
mdnsd_start
TX POWER
High Power
CHANNEL
0
SECURITY ENABLE
802.11N AGGREGATION
CUSTOM FEATURE SELECT BITMAP
0
IP Settings
IP VERSION
IPv4
DHCPV4 CLIENT
Network Stack Settings
DYNAMIC WEBPAGES (JSON)
HTTP CLIENT
DNS CLIENT
SNMP AGENT
SSL
PING
HTTPS
FTP CLIENT
Submit
Clear
Copyright Â© 2015 Redpine Signals, Inc. All rights reserved.

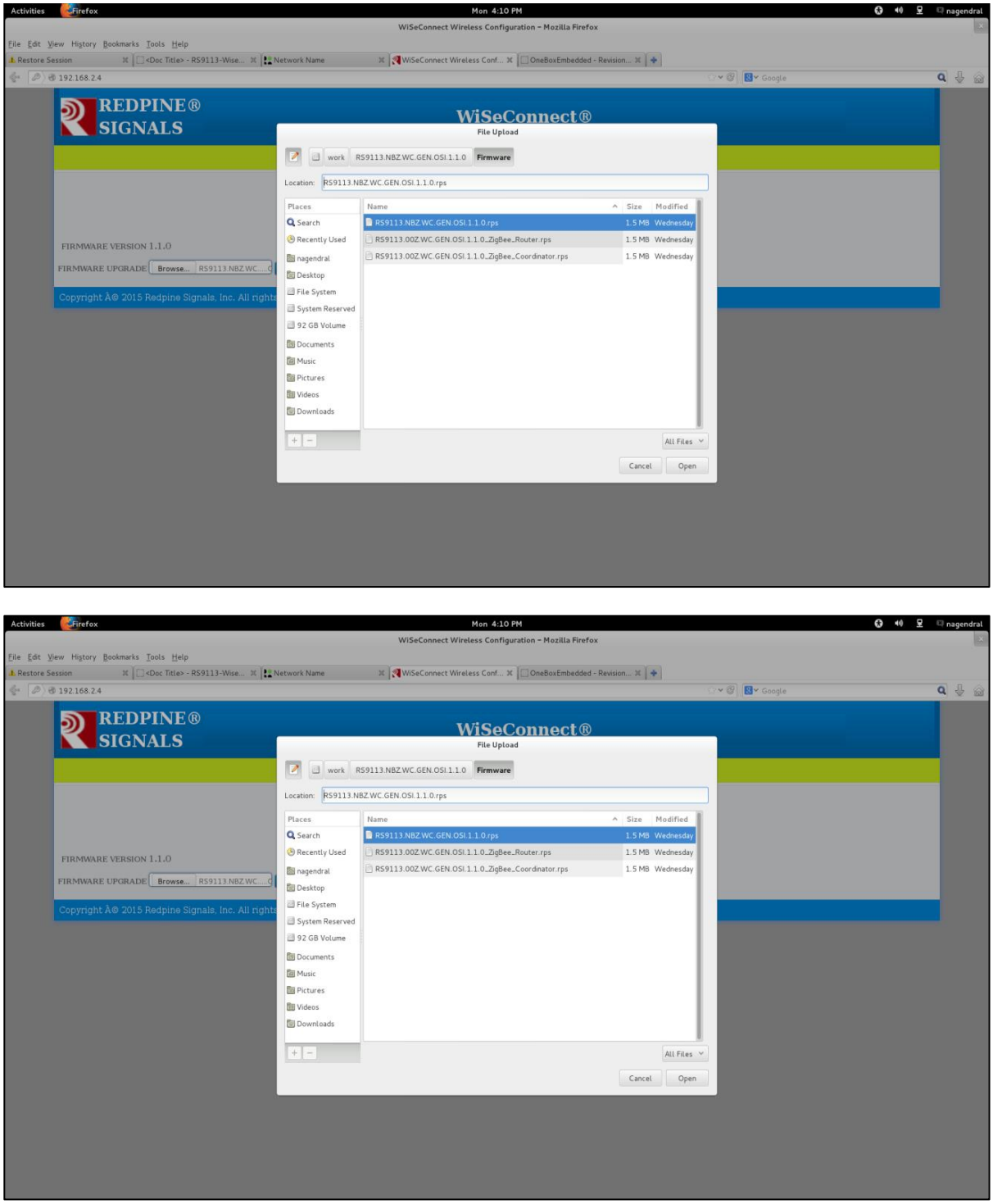


Figure 119: Firmware Upgrade Successful

6.40 WLAN Asyc Stats

Overview

This example demonstrates how to get asynchronous messages to host to indicate the module state.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- WPS supported Access Point
- Windows PC2
- Silicon Labs module

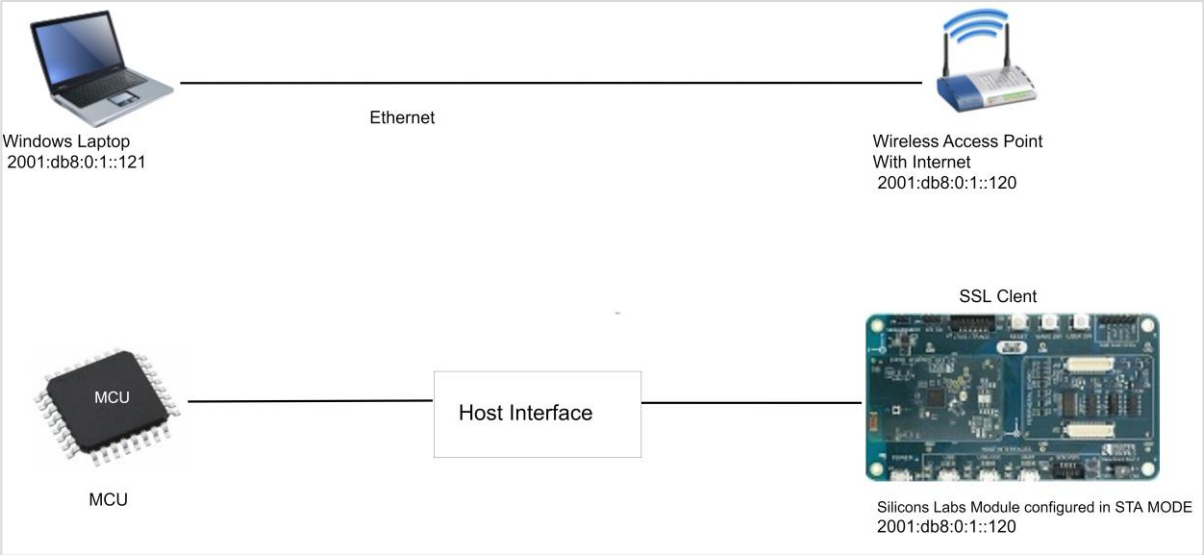


Figure 120: Setup Diagram for WLAN Asynchronous Statistics

Configuration and Execution of the Application

Configuring the Application

1. Open rsi_wlan_async_stats.c file and update / modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels.

```
#define CHANNEL_NO 0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                     "<psk>
```

   Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN      15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE 1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP 0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY 0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK 0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP      (CUSTOM_FEAT_ASYNC_CONNECTION_STATUS |
FEAT_CUSTOM_FEAT_EXTENTION_VALID )
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_256K_MODE
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. Host will get asynchronous messages.
   i) Module state : Indicates the connection state change. (for example : Associated).
   ii) reason : Indicates reason for state change. (for example : No reason specified).
   iii) Reason_code : This is used to get the reason code. (for example : De-authentication).
   iv) Channel : AP channel at the given stage. (for example : channel of association).
   v) rssi : Rssi information. (for example : RSSI of AP).

vi) rsi_bssid : Represents the meaning of AP MAC at the given stage. (for example : MAC of AP).

6.41 WPS Access Point

WPS Overview

WPS (WiFi Protected setup) is a network standard to create a secure wireless home network using PUSH button and PIN button method.
In Push button method, in which the user has to push a button, either an actual or virtual one, on both the access point and the new wireless client device. On most devices, this discovery mode turns itself off as soon as a connection is established or after a delay (typically 2 minutes or less).
In PIN method, in which the user has to enter secret PIN on both the access point and the new wireless client device. On most devices, this discovery mode turns itself off as soon as a connection is established or after a delay (typically 2 minutes or less).

Overview

Silicon Labs device supports both Push button method and WPS pin method.
The WPS Access point application demonstrates how to configure the Silicon Labs device as an access point and allow client device to connect to it using WPS PUSH method. The application also enables TCP data transmission from connected Wi-Fi Station to device Access Point.

Sequence of Events

This Application explains user how to:

- Create WiSeConnect device as an Access point
- Enable WPS PUSH button method in Access Point
- Connect to client device to the access point using WPS PUSH method
- Open TCP client socket after successful connection
- Establish TCP connection with the TCP server opened in remote peer
- Send TCP data from remote peer to WiSeConnect device

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module as Access Point.
- WPS supported Mobile device as Wi-Fi station.
- A TCP server application running on the Windows PC2 (This example uses iperf for windows)

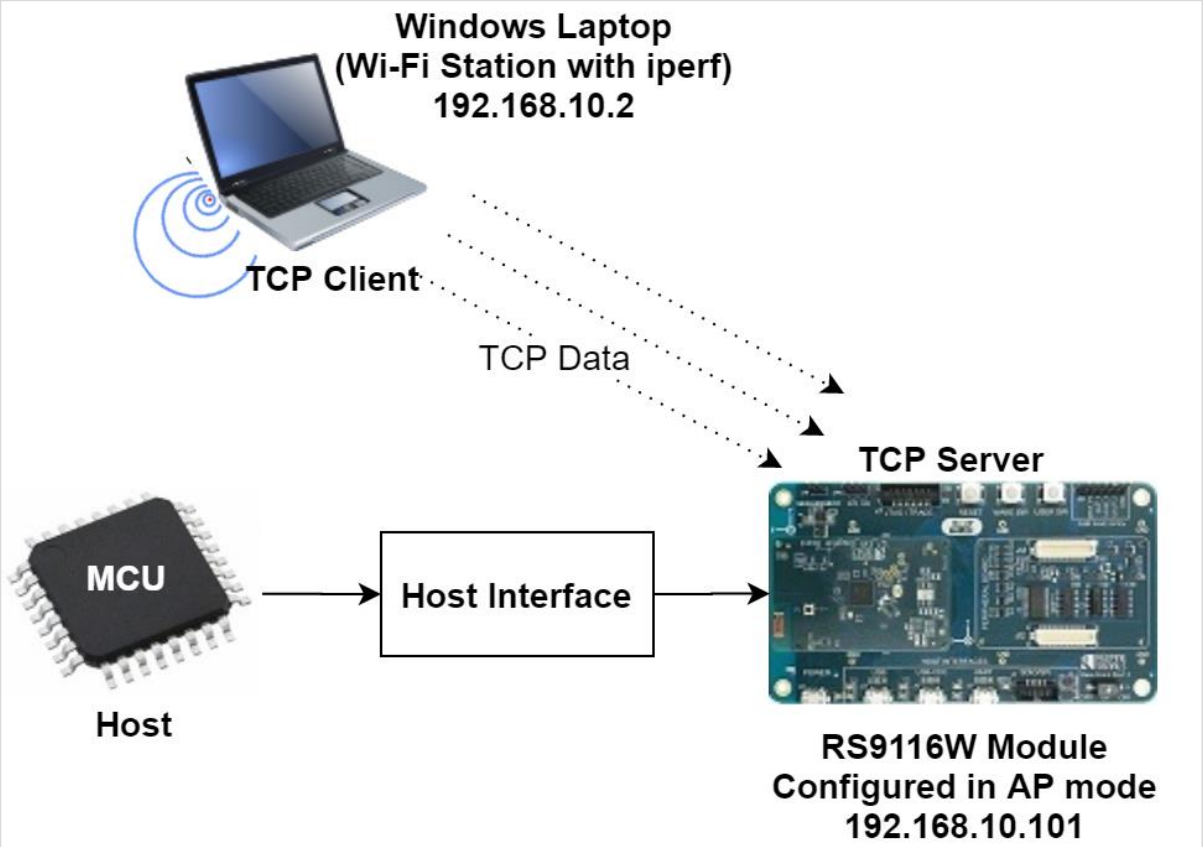


Figure 121: Setup Diagram for WPS Access Point

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_wps_ap_start.c*** file and update / modify the following macros:
   SSID refers to the name of the Access point to be created. In WPS method it should NULL.

```
#define SSID                                    ""
```

   SECURITY_TYPE refers to the type of security. In WPS method WiSeConnect module supports WPS push button method and WPS PIN method.
   In this examples, WiSeConnect module connects to AP using WPS push button method. So, set **SECURITY_TYPE** macro to **RSI_WPS_PUSH_BUTTON.**

```
#define SECURITY_TYPE                           RSI_WPA2
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes. In WPS method PSK is not needed.

```
#define PSK                                     NULL
```

   DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                             <local port>
```

   NUMEBR_OF_PACKETS refer to how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS                       <no of packets>
```

   Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         15000
```

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                                   0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order.
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                     0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                     0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update /modify the following macros:

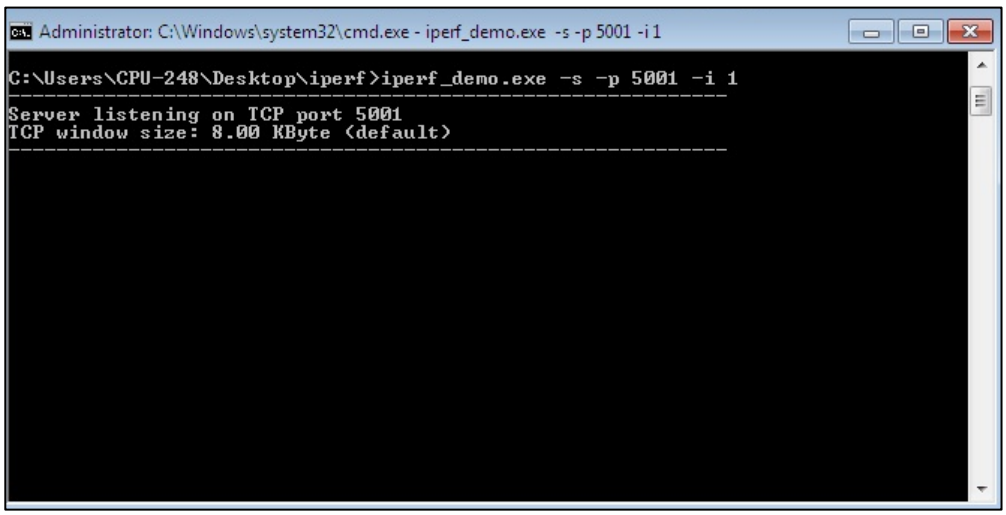
```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_SERVER
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP                 EXT_FEAT_256K_MODE
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

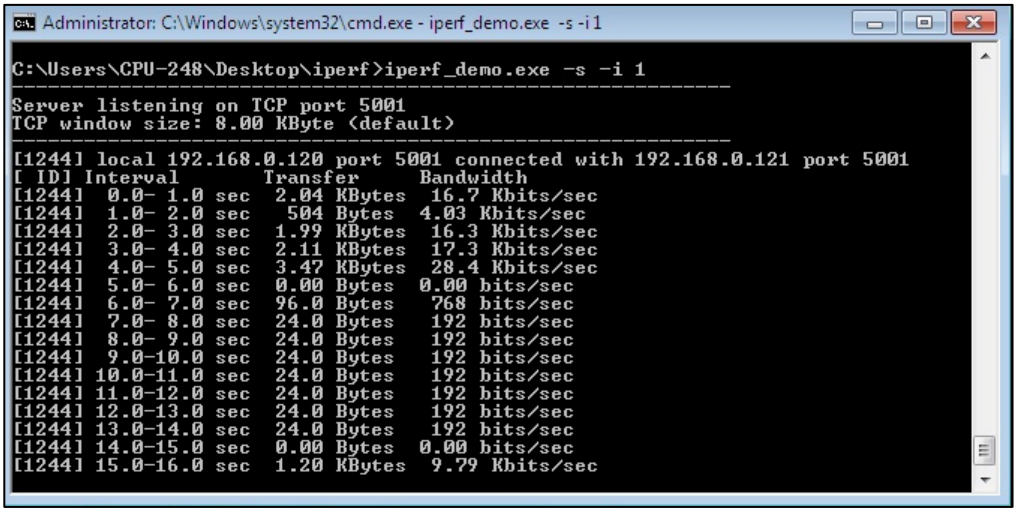
Executing the Application

1. Configure the Access point in WPA2-PSK mode in order to connect Silicon Labs device in STA mode.
2. Open TCP server application using iperf application in Windows PC1 which is connected to Access point through LAN.
3. Users can download application from the link below:
   https://iperf.fr/iperf-download.php#windows
   iperf_demo.exe -s -p <SERVER_PORT> -i 1



4. Press the WPS Push button on Access point which lasts for 2 minutes.
5. After program gets executed, the Silicon Labs device would scan and connect to the access point using WPS push button method and get IP.
6. After successful connection, the device STA connects to TCP server socket opened on Windows PC1 using

TCP client socket and sends configured **NUMBER_OF_PACKETS** to remote TCP server.
Please refer the below image for reception of TCP data on TCP server.



6.42 WPS Station

WPS Overview

WPS (Wi-Fi Protected setup) is a network standard to create a secure wireless home network using PUSH button and PIN button method.
In Push button method, in which the user has to push a button, either an actual or virtual one, on both the access point and the new wireless client device. On most devices, this discovery mode turns itself off as soon as a connection is established or after a delay (typically 2 minutes or less).
In PIN method, in which the user has to enter secret PIN on both the access point and the new wireless client device. On most devices, this discovery mode turns itself off as soon as a connection is established or after a delay (typically 2 minutes or less).

Overview

Silicon Labs Module supports both Push button method and WPS pin method to connect to an Access point. The WPS station application demonstrates how Silicon Labs module would be connected to secured (WPA2) Access point using WPS PUSH method.
In this application, after successful connection with the access point using WPS PUSH button method, Silicon Labs device opens TCP socket and sends TCP data to the configured remote peer.

Sequence of Events

This Application explains user how to:

- Configure device as WPS enable station
- Enable WPS PUSH button method in Access Point
- Connect to WPS enabled Access Point using PUSH button method
- Open TCP client socket after successful connection
- Establish TCP connection with the TCP server opened in remote peer
- Send TCP data from WiSeConnect device to remote peer

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- WPS supported Access Point
- Windows PC2

- Silicon Labs module
- A TCP server application running on the Windows PC2 (This example uses iperf for windows)

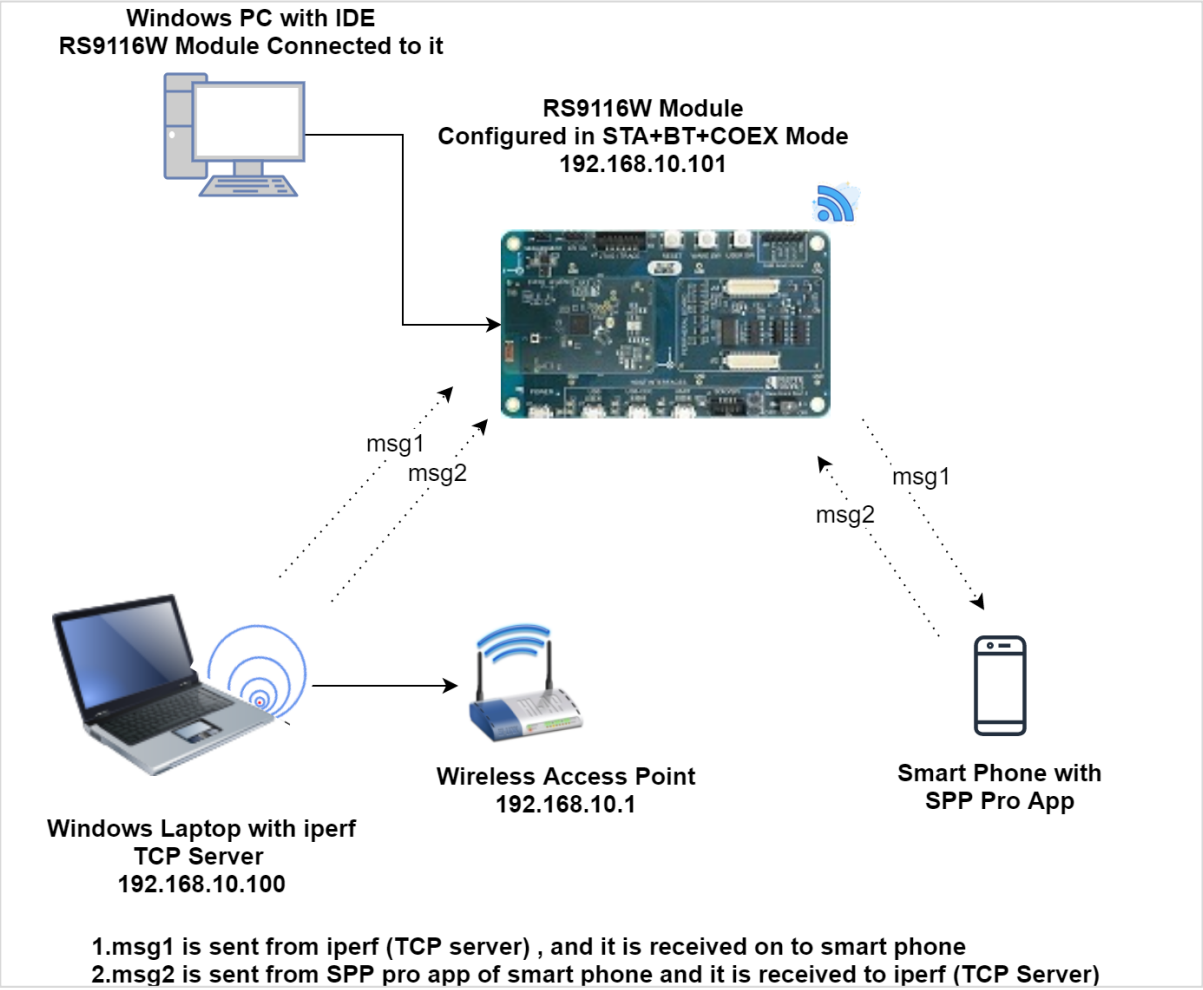


Figure 122: Setup Diagram for WPS Station

Configuration and Steps for Execution

Configuring the Application

1. Open ***rsi_wps_station.c*** file and update/modify following macros:

   SSID refers to the name of the Access point. In WPS method it should NULL.

```
#define SSID                                    REDPINE_AP
```

   SECURITY_TYPE refers to the type of security. In WPS method WiSeConnect module supports WPS push button method and WPS PIN method.
   In this examples, WiSeConnect module connecting to AP using WPS push button method. So, set **SECURITY_TYPE** macro to **RSI_WPS_PUSH_BUTTON.**

```
#define SECURITY_TYPE                           RSI_WPS_PUSH_BUTTON
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes. In WPS method PSK is not needed.

```
#define PSK                                     NULL
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                             <local port>
```

SERVER_PORT port refers remote TCP server port number which is opened in Windows PC2.

```
#define SERVER_PORT                             <remote port>
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with TCP server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as IP address, update the macro **DEVICE_IP** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS                       0x640AA8C0
```

NUMEBR_OF_PACKETS refers how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS                       <no of packets>
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         15000
```

To configure IP address
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                               1
```

Note:

If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                               0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order.

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEAT_BIT_MAP             EXT_FEAT_256K_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

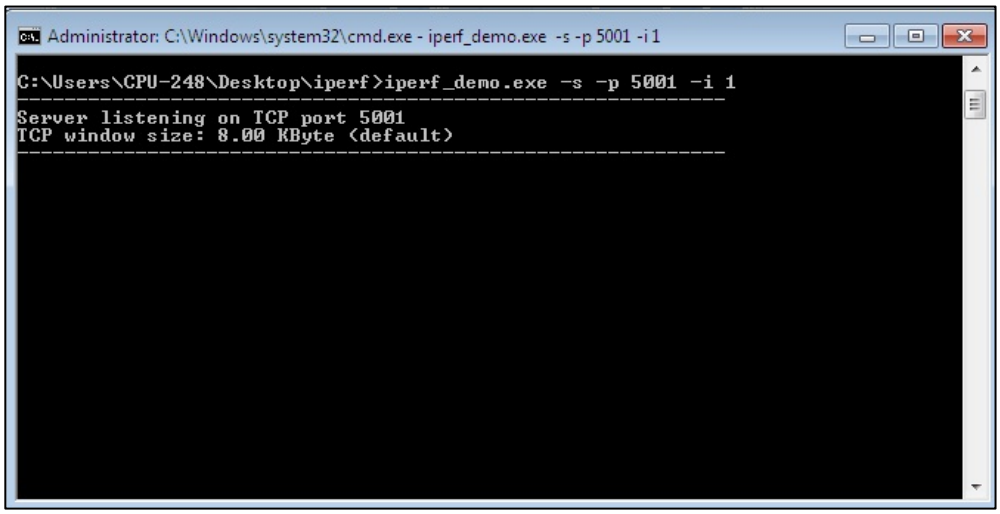
Note:

rsi_wlan_config.h file is already set with desired configuration in respective example folders, user need not change for each example.

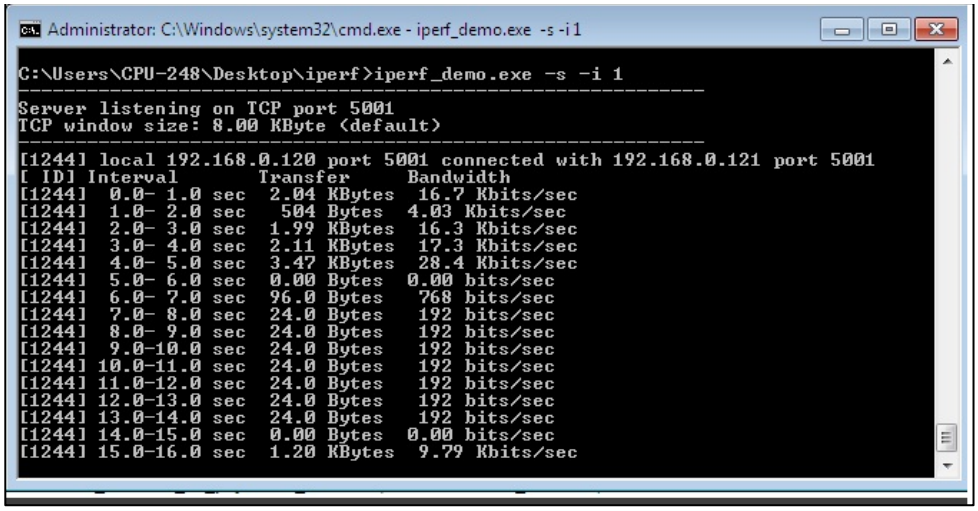
Executing the Application

1. Configure the Access point in WPA2-PSK mode in order to connect Silicon Labs device in STA mode.
2. Open TCP server application using iperf application in Windows PC1 which is connected to Access point through LAN.
3. Users can download application from the link below,
   https://iperf.fr/iperf-download.php#windows

   iperf_demo.exe -s -p <SERVER_PORT> -i 1



4. Press the WPS Push button on Access point and it lasts for 2 minutes.
5. After program gets executed, Silicon Labs Device would scan and connect to access point using WPS push button method and get IP.
6. After successful connection, Silicon Labs device STA connects to TCP server socket opened on Windows PC1 using TCP client socket and sends configured **NUMBER_OF_PACKETS** to remote TCP server.
   Please refer the below image for reception of TCP data on TCP server,


Administrator: C:\Windows\system32\cmd.exe - iperf_demo.exe -s -i 1
C:\Users\CPU-248\Desktop\iperf>iperf_demo.exe -s -i 1
Server listening on TCP port 5001
TCP window size: 8.00 KByte (default)
[1244] local 192.168.0.120 port 5001 connected with 192.168.0.121 port 5001
[ID] Interval Transfer Bandwidth
[1244] 0.0- 1.0 sec 2.04 KBytes 16.7 Kbits/sec
[1244] 1.0- 2.0 sec 504 Bytes 4.03 Kbits/sec
[1244] 2.0- 3.0 sec 1.99 KBytes 16.3 Kbits/sec
[1244] 3.0- 4.0 sec 2.11 KBytes 17.3 Kbits/sec
[1244] 4.0- 5.0 sec 3.47 KBytes 28.4 Kbits/sec
[1244] 5.0- 6.0 sec 0.00 Bytes 0.00 bits/sec
[1244] 6.0- 7.0 sec 96.0 Bytes 768 bits/sec
[1244] 7.0- 8.0 sec 24.0 Bytes 192 bits/sec
[1244] 8.0- 9.0 sec 24.0 Bytes 192 bits/sec
[1244] 9.0-10.0 sec 24.0 Bytes 192 bits/sec
[1244] 10.0-11.0 sec 24.0 Bytes 192 bits/sec
[1244] 11.0-12.0 sec 24.0 Bytes 192 bits/sec
[1244] 12.0-13.0 sec 24.0 Bytes 192 bits/sec
[1244] 13.0-14.0 sec 24.0 Bytes 192 bits/sec
[1244] 14.0-15.0 sec 0.00 Bytes 0.00 bits/sec
[1244] 15.0-16.0 sec 1.20 KBytes 9.79 Kbits/sec

7 WLAN BT

Following is the list of examples described in this section.

Table 1 Examples List for WLAN BT

S.No	Example	Description	Example Source Path
1	WLAN_STATION_BT_BRIDGE	The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BT) running in the same device	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan_bt\wlan_bt_bridge
2	Wlan BT Power save	The coex application demonstrates the procedure about how to configure the device in WisConnect coex mode with wlan standby and BT connected power save. In this coex application, Silicon Labs BT device connects with remote BT device (ex:Smart phone with SPP pro application) and issue connected power save command to module. In parallel Silicon Labs WiFi interface connects with an Access Point in station mode and issue connected power save command.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\wlan_bt\power_save

7.1 WLAN BT Bridge

Overview

The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BT) running in the same device.

Sequence of Events

WLAN Task

This Application explains user how to:

- Create Silicon Labs device as Station
- Connect Silicon Labs station to remote Access point
- Receive TCP data sent by connected station and forward to BT task
- Send data received by BT task to connected station using TCP protocol

BT Task

This Application explains user how to:

- Configure Silicon Labs device to SPP profile mode
- Configure device in discoverable and connectable mode
- Establish SPP profile level connection with remote smart phone
- Receive data sent by Smart phone and forward to WLAN task
- Send data received by WLAN task and send to Smart phone

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BT Application (Ex: Bluetooth SPP Pro from Google play store)
- WLAN Access Point and a Windows PC with iperf application

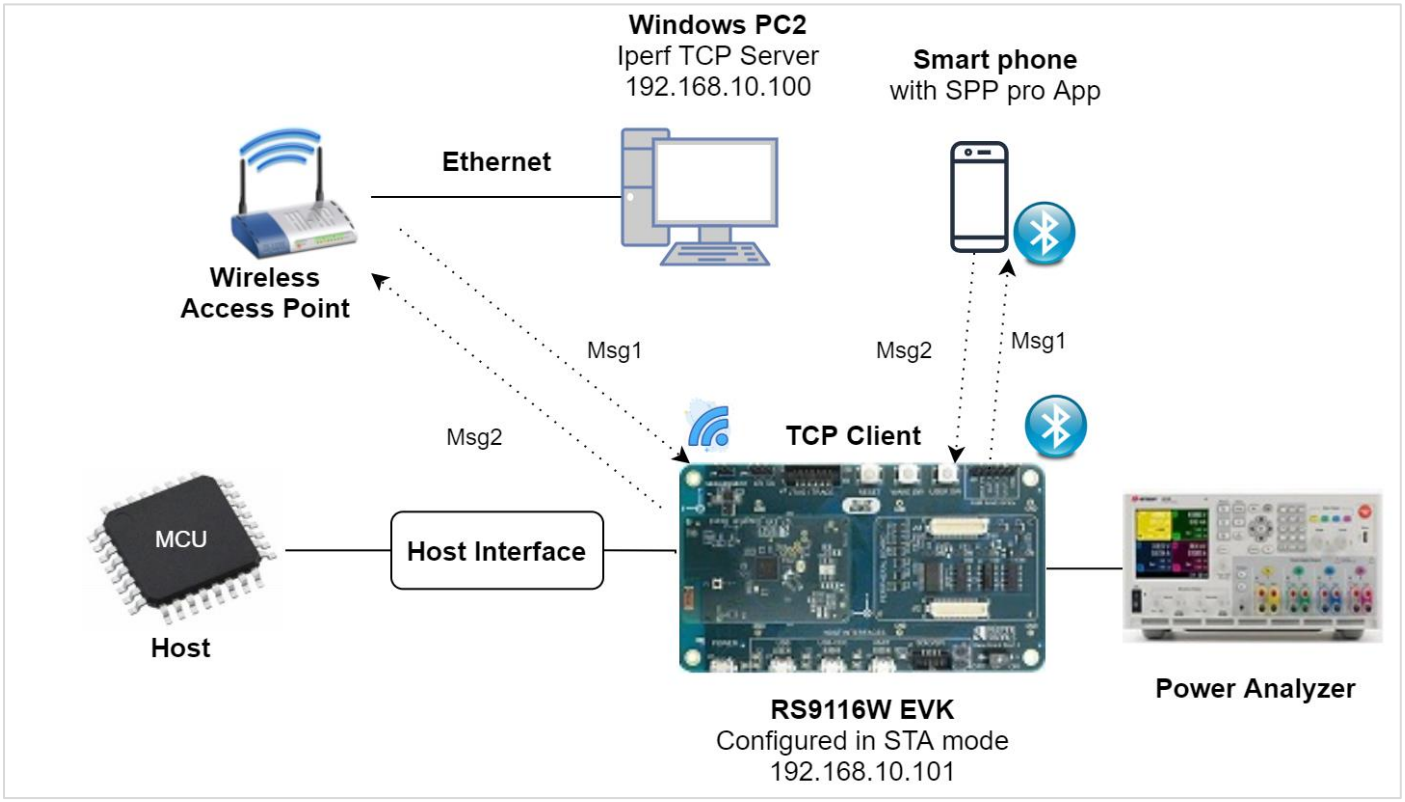


Figure 123: Setup Diagram for WLAN Station BT Bridge Application

Description

The coex application has WLAN and BT tasks and acts as an interface between Smartphone and PC. Smartphone interacts with BT task, while PC Both PC and Silicon Labs WLAN would be connected to a Wireless Access Point, thus both are connected together wirelessly interacts with WLAN task. When Smartphone connects and sends message to Silicon Labs device, BT task accepts and sends to WLAN task, which in turn sends to Access Point connected PC.
Similarly, when PC sends message to Silicon Labs device, the message will be sent to Smartphone via BT task. Thus messages can be seamlessly transferred between Windows PC and Smartphone.

Details of the Application

Silicon Labs WLAN acts as a Station and connects to an Access Point
Silicon Labs BT acts as a Slave device with SPP profile running in it, while Smart phone acts as Master device with SPP profile running in it.
Initially, proprietary Simple chat service is created with SPP profile (Silicon Labs device) to facilitate message exchanges.

- The WLAN task (running in Silicon Labs device) mainly includes following steps.

1. Connects to a Access Point
2. Exchanges data over TCP Server socket with the peer(Windows PC)

- The BT task (running in Silicon Labs device) mainly includes following steps.

1. Creates chat service
2. Configures the device in Discoverable mode and connectable mode.

WLAN and BT tasks forever run in the application to serve the asynchronous events

Configuration and Steps for Execution

Configuring the Application

Configuring the WLAN task

1. Open **rsi_wlan_app.c** file and update/modify following macros in order to establish connection with the Access-point.

   SSID refers to the Access point to which user wants to connect.
   SECURITY_TYPE is the security type of the Access point.
   PSK refers to the secret key if the Access point is configured in WPA/WPA2 security modes.

```
#define SSID                                    "<REDPINE_AP>"
#define SECURITY_TYPE                           RSI_OPEN
#define PSK                                     " "
```

2. Enable/Disable DHCP mode

   1 – Enables DHCP mode (gets the IP from DHCP server)
   0 – Disables DHCP mode

```
#define DHCP_MODE                               1
```

3. If DHCP mode is disabled, then change the following macros to configure static IP address

   IP address to be configured to the device should be in long format and in little endian byte order.
   Example: To configure "192.168.10.101" as IP address, update the macro **DEVICE_IP** as **0x650AA8C0**.

```
#define DEVICE_IP                               0x650AA8C0
```

   IP address of the gateway should also be in long format and in little endian byte order
   Example: To configure "192.168.10.1" as Gateway, update the macro **GATEWAY** as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

   IP address of the network mask should also be in long format and in little endian byte order
   Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

   To establish TCP connection and transfer data to the remote socket configure the below macros. Internal socket port number.

```
#define DEVICE_PORT                             5001
```

   Port number of the remote server

```
#define REMOTE_PORT                             5001
```

   IP address of the remote server

```
#define REMOTE_IP_ADDRESS                       0x650AA8C0
```

   Open ***rsi_bt_app.c*** file and update/modify following macro. Number of packets to send

```
#define NUMBER_OF_PACKETS                                   1000
```

Open ***main.c*** file and update/modify following macro. Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                                     10000
```

Include r**si_wlan_app.c** , **rsi_bt_app.c** and **main.c** files in the project, build and launch the application
Open server socket on remote machine, For example, to open TCP server socket with port number 5001 on remote side, use the command as given below

4. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                     RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                                 FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                                   RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                          (TCP_IP_FEAT_DHCPV4_CLIENT |
TCP_IP_TOTAL_SOCKETS_1 | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP                          FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                      EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BIT_MAP                       EXT_DYNAMIC_COEX_MEMORY
#define RSI_BAND                                            RSI_BAND_2P4GHZ
```

Configuring the BT task

1. Open ***rsi_bt_app.c*** file and update/modify following macros:

- RSI_BT_LOCAL_NAME **-** Name of the Silicon Labs device
- PIN_CODE - Four byte string required for pairing process.

Following are the **non-configurable** Macros in the Application file.

1. BT_GLOBAL_BUFF_LEN – Number of bytes required for the Application and the Driver.
2. RSI_APP_EVENT_CONNECTED - Event number to be set on connection establishment.
3. RSI_APP_EVENT_DISCONNECTED - Event number to be set on disconnection.
4. RSI_APP_EVENT_PINCODE_REQ - Event number to be set on Pincode request for pairing.
5. RSI_APP_EVENT_LINKKEY_SAVE - Event number to be set on link key save.
6. RSI_APP_EVENT_AUTH_COMPLT - Event number to be set on authentication complete.
7. RSI_APP_EVENT_LINKKEY_REQ - Event number to be set on link key request for connection.
8. RSI_APP_EVENT_SPP_CONN - Event number to be set on SPP connection.
9. RSI_APP_EVENT_SPP_DISCONN - Event number to be set on SPP disconnection.
10. RSI_APP_EVENT_SPP_RX - Event number to be set on SPP data received from Master.

Executing the coex Application

1. Connect WiSeConnect device to the Windows PC running KEIL or IAR IDE
2. Build and launch the application.
3. After the program gets executed, Silicon Labs BT is in Discoverable state and WLAN has established TCP server socket with peer (PC).
4. Open a BT SPP Pro App in the Smartphone and do scan.

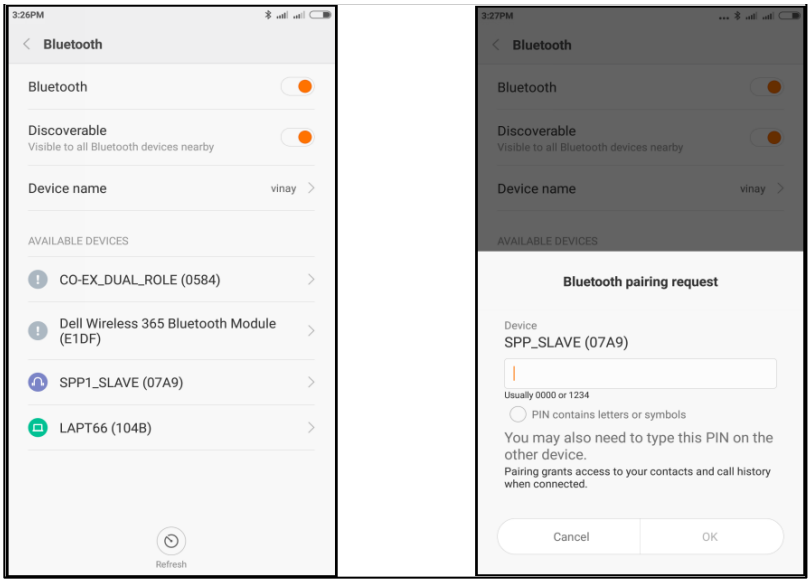


Figure 124: Turn ON Bluetooth, Scan for BT Devices and Pair with the Silicon Labs BT Device

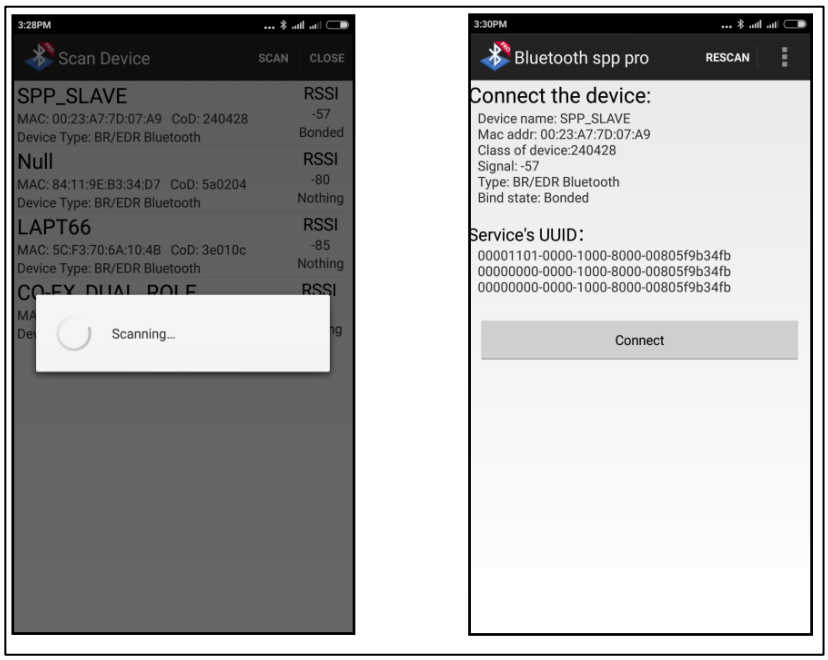


Figure 125: Scan Using SPP Pro App and Get Connected to the Peripheral

5. In the App, Silicon Labs BT would appear with the name configured in the macro **RSI_BT_LOCAL_NAME**.

6. Now initiate connection from the SPP App running in the Smartphone.

7. After BT connection is established, send a message from the App to Silicon Labs BT. Observe this message in the PC connected via TCP server socket with WiSeConnect WLAN.

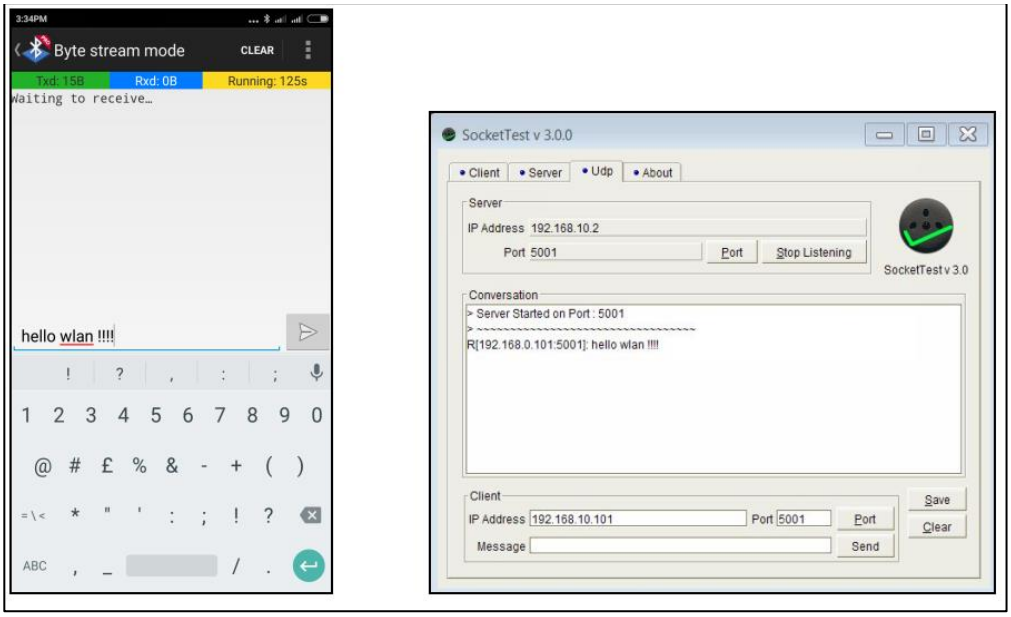


Figure 126: Sending Data from BT SPP Pro APP and Received on Wifi AP Socket App

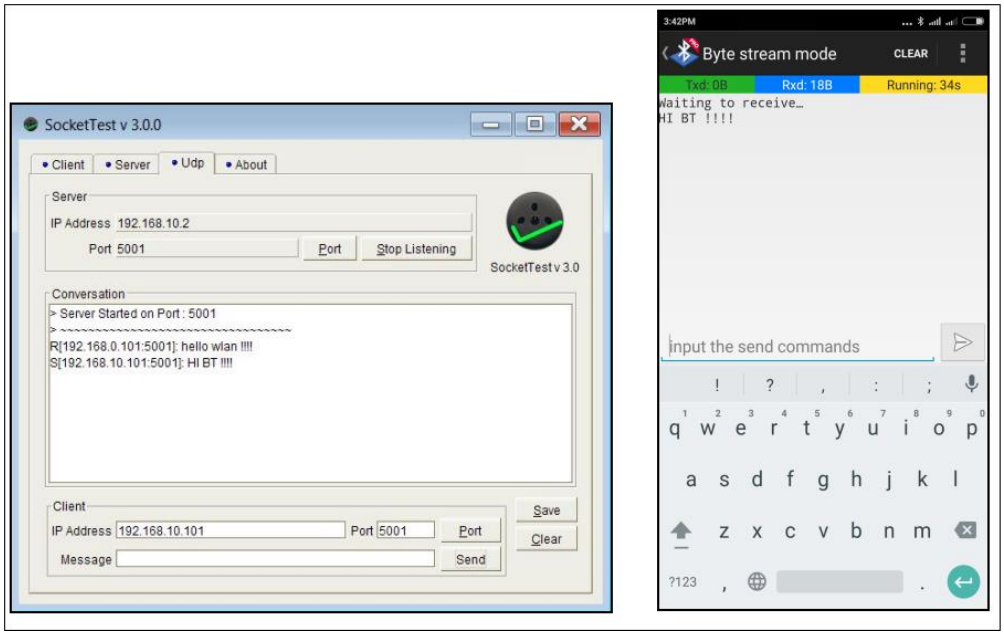


Figure 127: Sending Data from Wifi AP Socket App and Received on BT SPP Pro App

8. Now, send a message from PC to Silicon Labs WLAN via TCP server socket and observe the same in the Smartphone

9. rsi_bt_app_send_to_wlan() function defined in rsi_wlan_app.c to send message from BT task to WLAN task.

10. With the help of wlan task, message is transferred to PC.

11. Message from PC to WLAN application via socket and rsi_wlan_app_send_to_bt() function defined in rsi_bt_app.c called asynchronously to send message from WLAN task to BT task. From BT task message transferred to client with the event.

7.2 WLAN BT Power Save

Overview

The coex application demonstrates the procedure about how to configure the device in WisConnect coex mode with wlan standby and bt connected power save.
In this coex application, Silicon Labs BT device connects with remote BT device (ex:Smart phone with spp pro

application) and issue connected power save command to module. In parallel Silicon Labs WiFi interface connects with an Access Point in station mode and issue connected power save command.

WLAN Task:

Sequence of Events

This Application explains user how to:

- Create Silicon Labs device as Station
- Connect Silicon Labs station to remote Access point and get IP
- Enable appropriate power save mode and then wait in a scheduler for some time
- Receive TCP data sent by connected station and forward to BT task
- Send data received by BT task to connected station using TCP protocol

BT Task:

Sequence of Events

This Application explains user how to:

- Configure Silicon Labs device to SPP profile mode
- Configure device in discoverable and connectable mode
- Configure device in power save profile mode 2
- Establish SPP profile level connection with remote smart phone
- Receive data sent by Smart phone and forward to WLAN task
- Send data received by WLAN task and send to Smart phone

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BT Application (Ex: Bluetooth SPP Pro from Google play store)
- WLAN Access Point and a Windows PC with iperf application

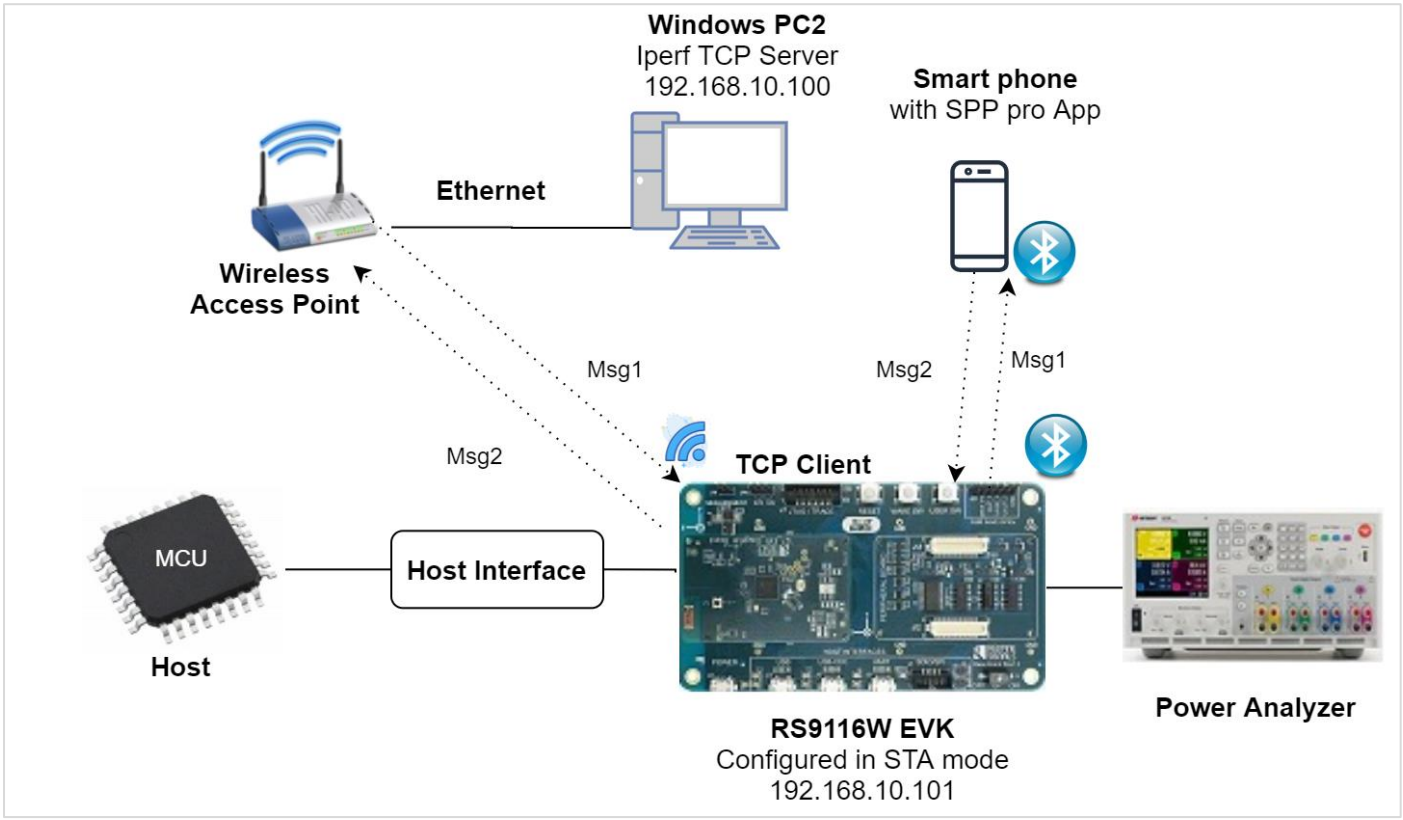


Figure 128: Setup Diagram for WLAN BT Power Save

Details of the Application

Silicon Labs WLAN acts as a Station and connects to an Access Point and will go to power_save depending on the selected mode
Silicon Labs BT acts as a Slave device with SPP profile running in it, while Smart phone acts as Master device with SPP profile running in it.
Initially, proprietary Simple chat service is created with SPP profile (Silicon Labs device) to facilitate message exchanges.

- The WLAN task (running in Silicon Labs device) mainly includes following steps.

1. Connects to a Access Point and goto power_save
2. Once wake-up after certain time, exchanges data over TCP Server socket with the peer(Windows PC)

- The BT task (running in Silicon Labs device) mainly includes following steps.

1. Creates chat service
2. Configures the device in Discoverable mode and connectable mode.

WLAN and BT tasks forever run in the application to serve the asynchronous events

Configuration and Steps for Execution

Configuring the Application Configuring the WLAN task

1. Open **rsi_wlan_app.c** file and update/modify following macros in order to establish connection with the Access-point.

   SSID refers to the Access point to which user wants to connect.
   SECURITY_TYPE is the security type of the Access point.
   PSK refers to the secret key if the Access point is configured in WPA/WPA2 security modes.

```
#define SSID "<REDPINE_AP>" #define SECURITY_TYPE 0 #define PSK " "
```

   Enable/Disable DHCP mode

   1 – Enables DHCP mode (gets the IP from DHCP server)
   0 – Disables DHCP mode

```
#define DHCP_MODE 1
```

If DHCP mode is disabled, then change the following macros to configure static IP address

IP address to be configured to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.101" as IP address, update the macro **DEVICE_IP** as **0x650AA8C0**.

```
#define DEVICE_IP 0x650AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro **GATEWAY** as **0x010AA8C0**

```
#define GATEWAY 0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK 0x00FFFFFF
```

To establish TCP connection and transfer data to the remote socket configure the below macros. Internal socket port number.

```
#define DEVICE_PORT 5001
```

Port number of the remote server

```
#define SERVER_PORT 5001
```

IP address of the remote server

```
#define SERVER_IP_ADDRESS 0x650AA8C0
```

Open ***rsi_bt_app.c*** file and update/modify following macro. Number of packets to send

```
#define NUMBER_OF_PACKETS 1000
```

Open ***main.c*** file and update/modify following macro. Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN    10000
```

Include r**si_wlan_app.c** , **rsi_bt_app.c** and **main.c** files in the project, build and launch the application
Open server socket on remote machine, For example, to open TCP server socket with port number 5001 on remote side, use the command as given below

In this application, default power save mode configuration is set to low power mode 2 (RSI_SLEEP_MODE_2) with maximum power save (RSI_MAX_PSP) with GPIO based handshake.

```
#define   PSP_TYPE                      RSI_MAX_PSP
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                                  RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                              FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                                RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                       (TCP_IP_FEAT_DHCPV4_CLIENT |
TCP_IP_TOTAL_SOCKETS_1 | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP                       FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP                   EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BIT_MAP                    EXT_DYNAMIC_COEX_MEMORY
#define RSI_BAND                                         RSI_BAND_2P4GHZ
```

Default configuration of low power save mode 2

```
#define   RSI_HAND_SHAKE_TYPE            GPIO_BASED
#define   RSI_SELECT_LP_OR_ULP_MODE      RSI_ULP_WITH_RAM_RET
#define   RSI_DTIM_ALIGNED_TYPE          0
#define   RSI_MONITOR_INTERVAL           50
#define   RSI_WMM_PS_ENABLE              RSI_DISABLE
#define   RSI_WMM_PS_TYPE                0
#define   RSI_WMM_PS_WAKE_INTERVAL       20
#define   RSI_WMM_PS_UAPSD_BITMAP        15
```

Configuring the BT task:

1. Open ***rsi_bt_app.c*** file and update/modify following macros:

- RSI_BT_LOCAL_NAME **-** Name of the Silicon Labs device
- PIN_CODE - Four byte string required for pairing process.

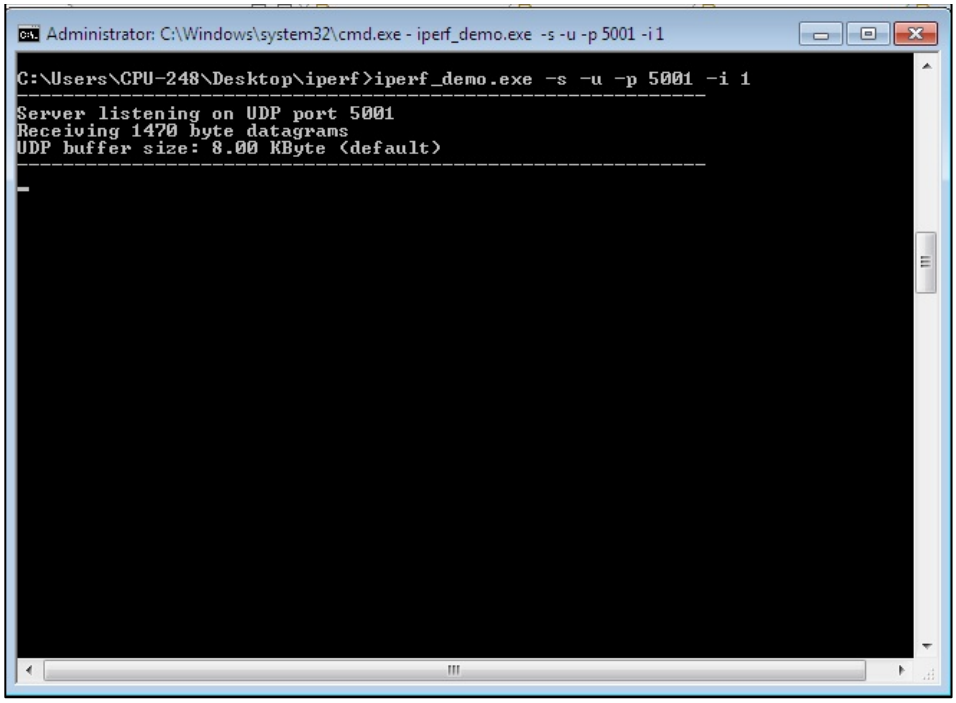
Following are the **non-configurable** Macros in the Application file.

1. BT_GLOBAL_BUFF_LEN – Number of bytes required for the Application and the Driver.
2. RSI_APP_EVENT_CONNECTED - Event number to be set on connection establishment.
3. RSI_APP_EVENT_DISCONNECTED - Event number to be set on disconnection.
4. RSI_APP_EVENT_PINCODE_REQ - Event number to be set on Pincode request for pairing.
5. RSI_APP_EVENT_LINKKEY_SAVE - Event number to be set on link key save.
6. RSI_APP_EVENT_AUTH_COMPLT - Event number to be set on authentication complete.
7. RSI_APP_EVENT_LINKKEY_REQ - Event number to be set on link key request for connection.
8. RSI_APP_EVENT_SPP_CONN - Event number to be set on SPP connection.
9. RSI_APP_EVENT_SPP_DISCONN - Event number to be set on SPP disconnection.
10. RSI_APP_EVENT_SPP_RX - Event number to be set on SPP data received from Master.

Executing the Application

1. Configure the Access point in OPEN / WPA-PSK/WPA2-PSK mode in order to connect Silicon Labs device in STA mode.
2. Open TCP server application using iperf application in Windows PC2 which is connected to Access point through LAN.

iperf_demo.exe –s -u -p <SERVER_PORT> -i 1



3.After the program gets executed, Silicon Labs BT is in Discoverable state and WLAN has established TCP server socket with peer (PC).

4.Open a BT SPP Pro App in the Smartphone and do scan.

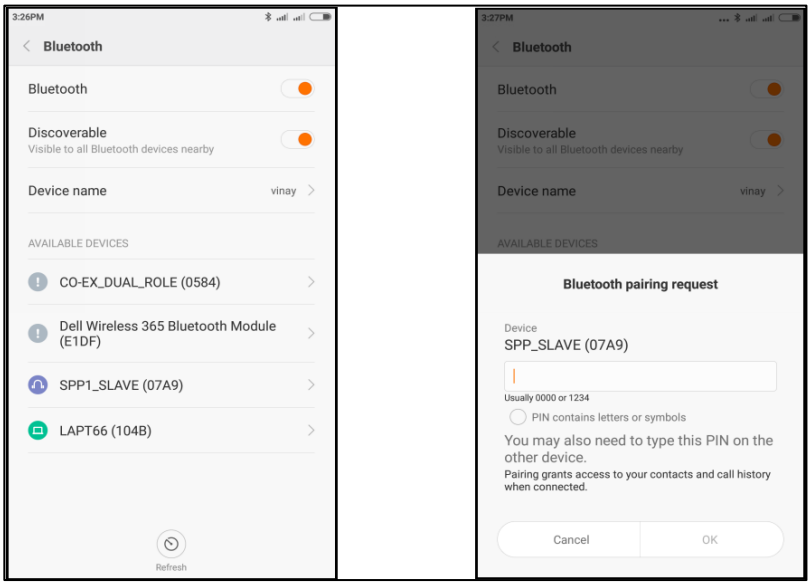


Figure 129: Turn ON Bluetooth, Scan for BT Devices and Pair with the Silicon Labs BT Device

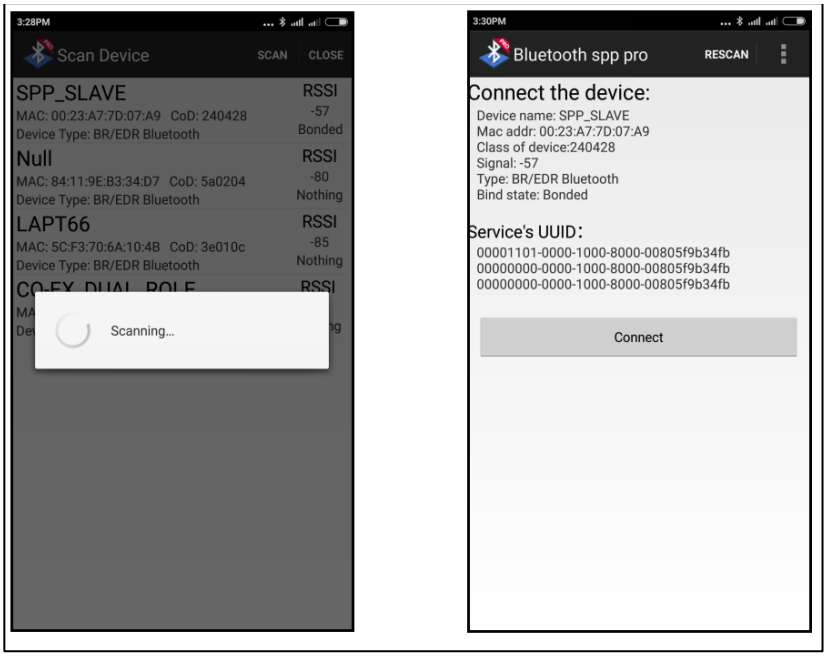


Figure 130: Scan using SPP Pro App and Get Connected to the Peripheral

5. In the App, Silicon Labs BT would appear with the name configured in the macro **RSI_BT_LOCAL_NAME**.

6. Now initiate connection from the SPP App running in the Smartphone.

7. After BT connection is established, send a message from the App to Silicon Labs BT. Observe this message in the PC connected via TCP server socket with WiSeConnect WLAN.

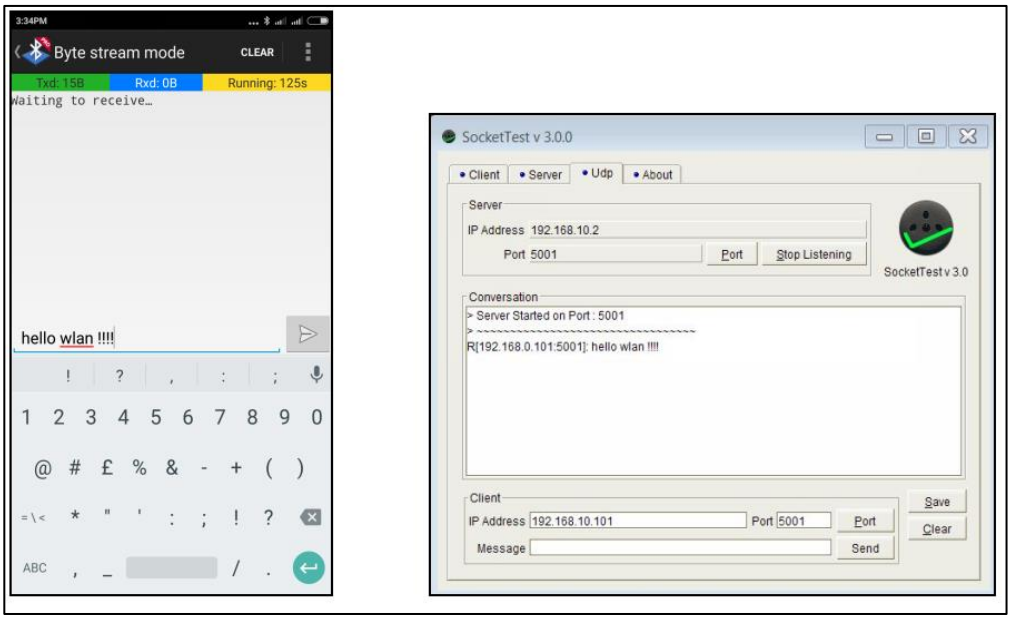


Figure 131: Sending Data from BT SPP Pro APP and Received on Wifi AP Socket App

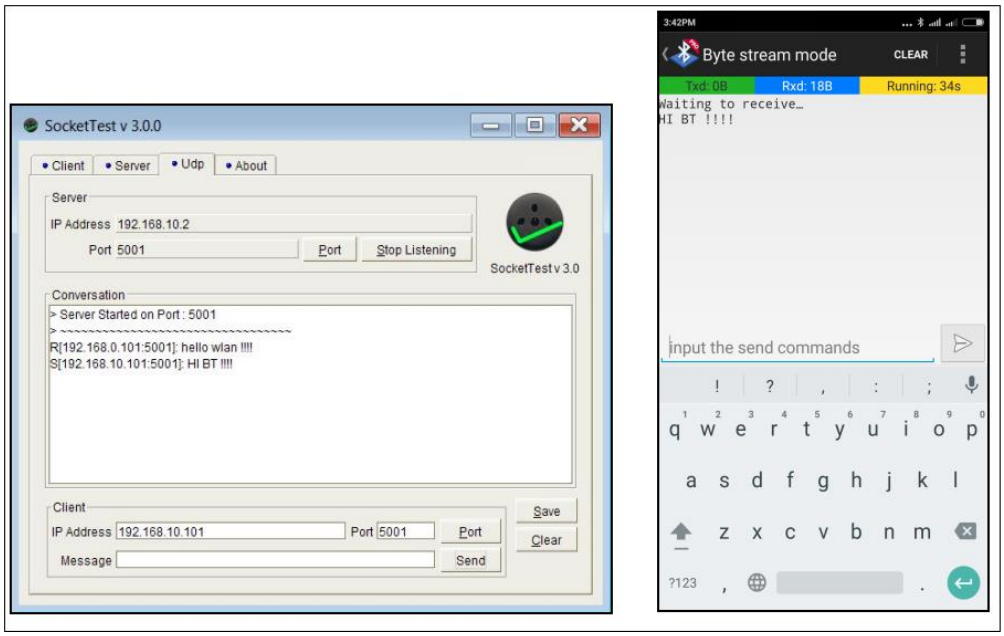


Figure 132: Sending Data from Wifi AP Socket App and Received on BT SPP Pro App

8. Now, send a message from PC to Silicon Labs WLAN via TCP server socket and observe the same in the Smartphone

9. rsi_bt_app_send_to_wlan() function defined in rsi_wlan_app.c to send message from BT task to WLAN task.

10. With the help of wlan task, message is transferred to PC.

11. Message from PC to WLAN application via socket and rsi_wlan_app_send_to_bt() function defined in rsi_bt_app.c called asynchronously to send message from WLAN task to BT task. From BT task message transferred to client with the event.

7.3 WLAN BT Throughput App

Introduction

This example is applicable to WiSeConnect™. The feature(s) used in this example may or may not be available in your part. Refer to the product datasheet to verify the features available in your part.

Overview

The coex application demonstrates throughput measurement of wifi while BT is in connection.

Sequence of Events

WLAN Task

This application can be used to configure Silicon Labs module in UDP client / server or TCP client / server. To measure throughput, following configuration can be applied.

- To measure UDP Tx throughput, module should be configured as UDP client.
- To measure UDP Rx throughput, module should be configured as UDP server.
- To measure TCP Tx throughput, module should be configured as TCP client.
- To measure TCP Rx throughput, module should be configured as TCP server.

BT Task

This Application explains user how to:

- Configure Silicon Labs device to SPP profile mode
- Configure device in discoverable and connectable mode
- Establish SPP profile level connection with remote smart phone
- Receive data sent by Smart phone.

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BT Application (Ex: Bluetooth SPP Pro from Google play store)
- WLAN Access Point and a Windows PC with iperf application

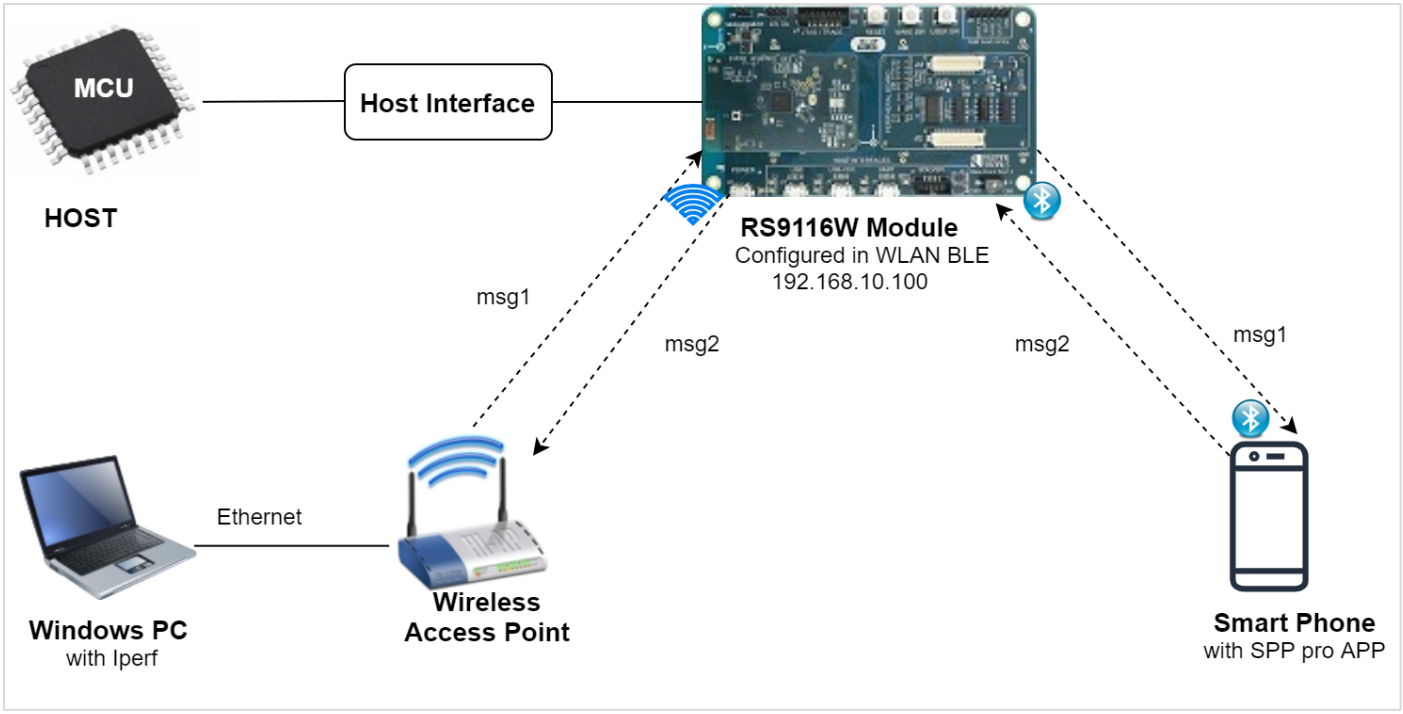


Figure 133: Setup Diagram for WLAN Station BT Bridge Application

Description

The coex application has WLAN and BT tasks and acts as an interface between Smartphone and PC. Smartphone interacts with BT task, while Both PC and Silicon Labs WLAN would be connected to a Wireless Access Point, thus both are connected together wirelessly interacts with WLAN task. When Smartphone connects and sends message to Silicon Labs device, BT task accepts.
Similarly, data transfer will happen for Station and AP.

Details of the Application

Silicon Labs WLAN acts as a Station and connects to an Access Point
Silicon Labs BT acts as a Slave device with SPP profile running in it, while Smart phone acts as Master device with SPP profile running in it.
Initially, proprietary Simple chat service is created with SPP profile (Silicon Labs device) to facilitate message exchanges.

- The WLAN task (running in Silicon Labs device) mainly includes following steps.

1. Connects to a Access Point
2. Exchanges data over socket with the peer(Windows PC)

- The BT task (running in Silicon Labs device) mainly includes following steps.

1. Creates chat service

2. Configures the device in Discoverable mode and connectable mode.

WLAN and BT tasks forever run in the application to serve the asynchronous events

Configuration and Steps for Execution

Configuring the Application

Configuring the WLAN task

1. Open ***rsi_wlan_app.c*** file and update / modify the following macros,

   SSID refers to the name of the Access point.

```
#define SSID                                    "<REDPINE_AP>"
```

   CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                              11
```

   SECURITY_TYPE refers to the type of security .Access point supports Open, WPA, WPA2 securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point is to be configured in WPA/WPA2 security modes.

```
#define PSK                                     "<psk>"
```

   Enable / Disable DHCP mode
   1-Enables DHCP mode (gets the IP from DHCP server)
   0-Disables DHCP mode

```
#define DHCP_MODE                               1
```

   To configure static IP address
   IP address to be configured to the device should be in long format and in little endian byte order.
   Example: To configure "192.168.10.1" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0**.

```
#define DEVICE_IP                               0X010AA8C0
```

   IP address of the gateway should also be in long format and in little endian byte order
   Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

   IP address of the network mask should also be in long format and in little endian byte order
   Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

2. To establish UDP/TCP connection and transfer/receive data to the remote socket configure the below macros

   Internal device port number

```
#define PORT_NUM                                <5001>
```

IP address of the remote server

```
#define SERVER_IP_ADDRESS                       0x640AA8C0
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         8000
```

Application can use receive buffer size of 1400

```
#define BUF_SIZE                                1400
```

Application can select throughput type as UDP Tx, UDP Rx, TCP Tx or TCP Rx. Following is macro need to use.

```
#define THROUGHPUT_TYPE                         UDP_TX
```

Following is macro used for throughput type selection

```
#define UDP_TX                                  0
#define UDP_RX                                  1
#define TCP_TX                                  2
#define TCP_RX                                  3
```

Note:

In AP mode, configure same IP address for both DEVICE_IP and GATEWAY macros.

3. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     (FEAT_SECURITY_OPEN |FEAT_AGGREGATION)
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT |
TCP_IP_TOTAL_SOCKETS_1)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_384K_MODE
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Configuring the BT task

1. Open ***rsi_bt_app.c*** file and update/modify following macros:

- RSI_BT_LOCAL_NAME **-** Name of the Silicon Labs device
- PIN_CODE - Four byte string required for pairing process.

Following are the **non-configurable** Macros in the Application file.

1. BT_GLOBAL_BUFF_LEN – Number of bytes required for the Application and the Driver.
2. RSI_APP_EVENT_CONNECTED - Event number to be set on connection establishment.
3. RSI_APP_EVENT_DISCONNECTED - Event number to be set on disconnection.
4. RSI_APP_EVENT_PINCODE_REQ - Event number to be set on Pincode request for pairing.
5. RSI_APP_EVENT_LINKKEY_SAVE - Event number to be set on link key save.
6. RSI_APP_EVENT_AUTH_COMPLT - Event number to be set on authentication complete.
7. RSI_APP_EVENT_LINKKEY_REQ - Event number to be set on link key request for connection.
8. RSI_APP_EVENT_SPP_CONN - Event number to be set on SPP connection.
9. RSI_APP_EVENT_SPP_DISCONN - Event number to be set on SPP disconnection.
10. RSI_APP_EVENT_SPP_RX - Event number to be set on SPP data received from Master.

Executing the coex Application

1. Connect WiSeConnect device to the Windows PC running KEIL or IAR IDE
2. Build and launch the application.
3. After the program gets executed, Silicon Labs BT is in Discoverable state and WLAN has established TCP server socket with peer (PC).
4. Open a BT SPP Pro App in the Smartphone and do scan.

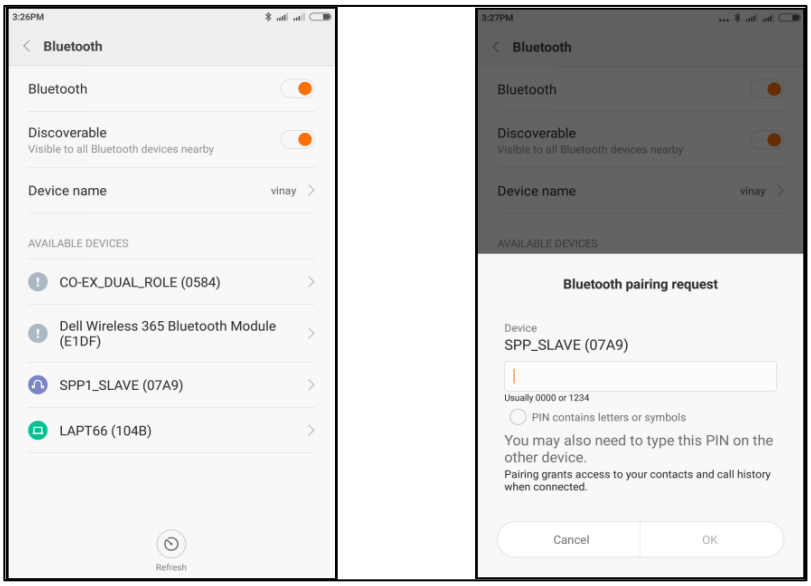


Figure 134: Turn ON Bluetooth, Scan for BT Devices and Pair with the Silicon Labs BT Device

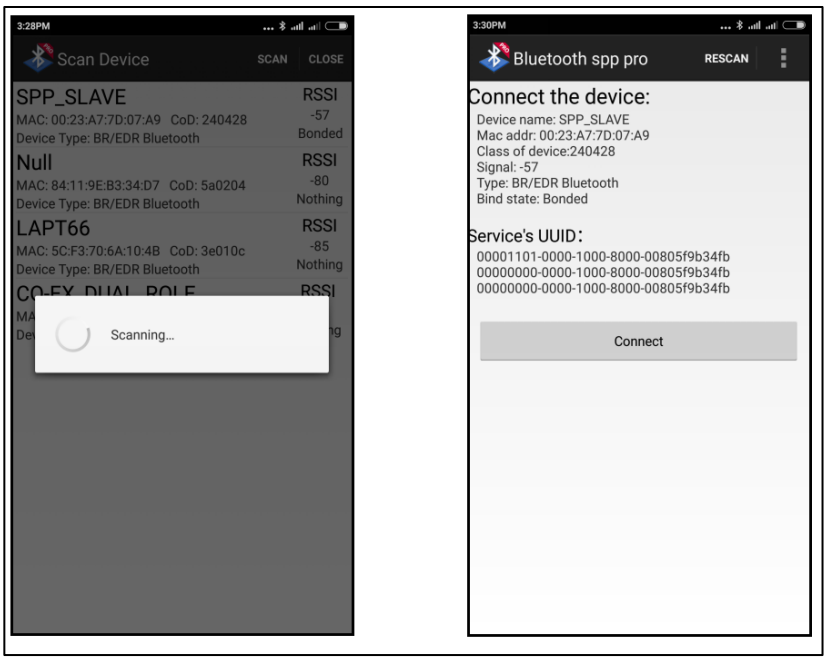


Figure 135: Scan using SPP Pro App and Get Connected to the Peripheral

5. In the App, Silicon Labs BT would appear with the name configured in the macro **RSI_BT_LOCAL_NAME**.
6. Now initiate connection from the SPP App running in the Smartphone.
7. After BT connection is established, send a message from the App to Silicon Labs BT.

8. To measure throughput, following configuration can be applied.
    a. To measure UDP Tx throughput,module should configured as UDP client. Open UDP server at remote port

       iperf.exe -s -u -p <SERVER_PORT> -i 1

    b. To measure UDP Rx throughput,module should configured as UDP server. Open UDP client at remote port

       iperf.exe -c <Module_IP> -u -p <Module_Port> -i 1 -b <Bandwidth>

    c. To measure TCP Tx throughput,module should configured as TCP client. Open TCP server at remote port.

       iperf.exe -s -p <SERVER_PORT> -i 1

    d. To measure TCP Rx throughput,module should configured as TCP server. Open TCP client at remote port.

       iperf.exe -c <Module_IP> -p <module_PORT> -i 1

9. To measure throughput, following configuration can be applied.
10. Build and launch the application.
11. After the program gets executed, the device would be connected to Access point having the configuration same as that of in the application and get.
12. The Device which is configured as UDP / TCP server / client will connect to iperf server / client and sends / receives data continuously. It will print the throughput per second.

Note: If M4 frequency need to Switch higher clock then follow below steps.

step1: Call **switch_m4_frequency()** api after device initialization.

step2: Update systics to higher clock as **SysTick_Config(SystemCoreClock /1000).**

8 WLAN BLE

Following is the list of examples described in this section.

S.No	Example	Description	Example Source Path	STM32 Project Path
1	Wlan_station_BLE_bridge	The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BLE) running in the same. In this coex application, Silicon Labs BTLE device connects with remote BTLE device (Smart Phone) and Silicon Labs WiFi interface connects with an Access Point in station mode and do data transfer in BTLE and WiFi interfaces.	RS9116.NB0.WC.GENR.OSI.xxxxx\host\sapis\examples\wlan_ble\wlan_station_ble_bridge	RS9116.NB0.WC.GENR.OSI.xxxxx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\wlan_sta_ble_bridge\ (Free RTOS Project)
2	Wlan_station_ble_dual_role_bridge	The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BLE) running in the same device as a LE master as well as slave.	RS9116.NB0.WC.GENR.OSI.xxxxx\host\sapis\examples\wlan_ble\wlan_station_ble_dual_role_bridge	NA
3	Wlan_station_ble_multiple_slaves	The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BLE) running in the same device.	RS9116.NB0.WC.GENR.OSI.xxxxx\host\sapis\examples\wlan_ble\wlan_station_ble_multiple_slaves_bridge	NA
4	Wlan_station_ble_throughput_example	This example is applicable to WiSeConnect™ For simplicity, this document refers to WiSeConnect, but all discussion applies to both WiSeConnect . The feature(s) used in this example may or may not be available in your part. Refer to the product datasheet to verify the features available in your part.	RS9116.NB0.WC.GENR.OSI.xxxxx\host\sapis\examples\wlan_ble\wlan_station_ble_throughput_app	NA
5	Wlan Ble Power save example	The coex application demonstrates the procedure about how to configure the device in WiseConnect coex mode with wlan standby and ble connected power save. In this coex application, Silicon Labs BLE device connects with remote BLE device (ex:Smart phone with spp pro application) and issue connected power save command to module.In parallel Silicon Labs WiFi interface connects with an Access Point in station mode and issue connected power save command.	RS9116.NB0.WC.GENR.OSI.xxxxx\host\sapis\examples\wlan_ble\power_save	NA
6	Wlan Ble Provisioning Examle	This application explains how to get the WLAN connection functionality using BLE provisioning. In this application,	RS9116.NB0.WC.GENR.OSI.xxxxx\host\sapis\examples\wlan_ble\wlan_station_ble_provisioning	RS9116.NB0.WC.GENR.OSI.x.x.x\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\wlan_sta_ble_provisioning\

S.No	Example	Description	Example Source Path	STM32 Project Path
		• Silicon Labs Module starts advertising and with BLE Provisioning the Access Point details are fetched • Silicon Labs device is configured as a WiFi station and connects to an Access Point.		(Free RTOS Project)

8.1 WLAN BLE Power Save

Overview

The coex application demonstrates the procedure about how to configure the device in WisConnect coex mode with wlan standby and ble connected power save.
In this coex application, Silicon Labs BLE device connects with remote BLE device (ex:Smart phone with spp pro application) and issue connected power save command to module.In parallel Silicon Labs WiFi interface connects with an Access Point in station mode and issue connected power save command.

WLAN Task:

Sequence of Events

This Application explains user how to:

- Create Silicon Labs device as Station
- Connect Silicon Labs station to remote Access point and get IP
- Enable appropriate power save mode and then wait in a scheduler for some time
- Receive TCP data sent by connected station and forward to BLE task
- Send data received by BLE task to connected station using TCP over SSL client protocol.

BLE Task:

This Application explains user how to:

- Configure device in advertise mode
- Connect from Smart phone/Dongle
- Configure device in power save profile mode 2 .

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Access point
- Windows PC2 with SSL server application (openssl)

> **Note:**
> Download Open SSL for windows from below link,
> http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

Note:
Install BLE scanner for GATT client application.

BTLE supported Smart phone with GATT client application

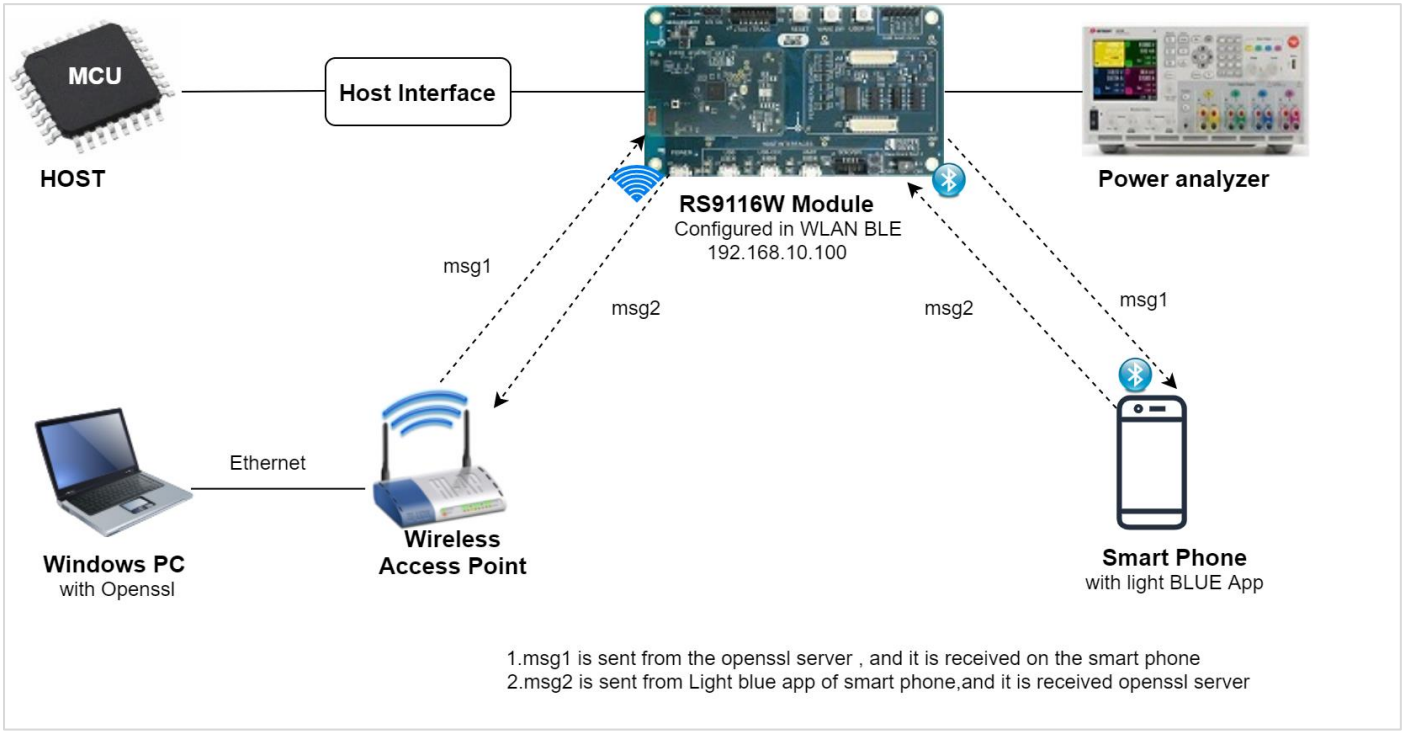


Figure 136: Setup Diagram for WLAN BLE Power Save

Configuration and Steps for Execution

Configuring the WLAN task

1. Open **rsi_wlan_app.c** file and update/modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                    "REDPINE_AP"
```

   CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

```
#define CHANNEL_NO                               0
```

   SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                     "<psk>"
```

To Load certificate:

```
#define LOAD_CERTIFICATE                        1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.
By default, application loading "cacert.pem" certificate if LOAD_**CERTIFICATE** enable. In order to load different certificate, user has to follow the following steps:

- rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pem certificate into linear array form using python script provided in the release package "certificate_script.py"

Ex: If the certificate is wifi-user.pem. Give the command in the following way
python certificate_script.py ca-cert.pem
Script will generate wifiuser.pem in which one linear array named cacert contains the certificate.

- After conversion of certificate, update *rsi_ssl_client.c* source file by including the certificate file and by providing *the* required parameters to rsi_wlan_set_certificate API.

Note:

Once certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.

So, define LOAD_CERTIFICATE as 0, if certificate is already present in the Device.

Note:
All the certificates are given in the release package *certificates*

DEVICE_PORT port refers SSL client port number

```
#define DEVICE_PORT                             5001
```

SERVER_PORT port refers remote SSL server port number

```
#define SERVER_PORT                             5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with SSL server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as IP address, update the macro **SERVER_IP_ADDRESS** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS                       0x640AA8C0
```

To configure IP address:
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                               1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)

If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                                   0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                     0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                     0x00FFFFFF
```

In **rsi_ble_app.c** file, default power save mode configuration is set to low power mode 2 (RSI_SLEEP_MODE_2) with maximum power save (RSI_MAX_PSP) with message based handshake.

```
#define   PSP_MODE                      RSI_SLEEP_MODE_2
#define   PSP_TYPE                      RSI_MAX_PSP
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                             RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                         (FEAT_ULP_GPIO_BASED_HANDSHAKE |
FEAT_DEV_TO_HOST_ULP_GPIO_1)
#define RSI_TCP_IP_BYPASS                           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP                  TCP_IP_FEAT_DHCPV4_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP                  FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP              (EXT_FEAT_LOW_POWER_MODE |
EXT_FEAT_XTAL_CLK_ENABLE | EXT_FEAT_384K_MODE)
#define RSI_BAND                                    RSI_BAND_2P4GHZ
```

Default configuration of low power save mode 2

```
#define   RSI_HAND_SHAKE_TYPE           GPIO_BASED
#define   RSI_SELECT_LP_OR_ULP_MODE     RSI_ULP_WITH_RAM_RET
#define   RSI_DTIM_ALIGNED_TYPE         0
#define   RSI_MONITOR_INTERVAL          50
#define   RSI_WMM_PS_ENABLE             RSI_DISABLE
#define   RSI_WMM_PS_TYPE               0
#define   RSI_WMM_PS_WAKE_INTERVAL      20
#define   RSI_WMM_PS_UAPSD_BITMAP       15
```

Configuring the BLE task:

Configuring the BLE Application

1. **Open *rsi_ble_app.c* file and update/modify following macros,**
   RSI_BLE_NEW_SERVICE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_NEW_SERVICE_UUID                    0xAABB
```

RSI_BLE_ATTRIBUTE_1_UUID refers to the attribute type of the first attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#define RSI_BLE_ATTRIBUTE_1_UUID                    0x1AA1
```

RSI_BLE_ATTRIBUTE_2_UUID refers to the attribute type of the second attribute under this service (**RSI_BLE_NEW_SERVICE_UUID).**

```
#define RSI_BLE_ATTRIBUTE_2_UUID                    0x1BB1
```

RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#define RSI_BLE_MAX_ DATA_LEN                       20
```

RSI_BLE_APP_DEVICE_NAME refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_DEVICE_NAME                     "WLAN_BLE_SIMPLE_CHAT"
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                      0x2803
```

RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CLIENT_CHAR _UUID                   0x2902
```

RSI_BLE_ATT_PROPERTY_READ is used to set the READ property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_READ                   0x02
```

RSI_BLE_ATT_PROPERTY_WRITE is used to set the WRITE property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_WRITE                  0x08
```

RSI_BLE_ATT_PROPERTY_NOTIFY is used to set the NOTIFY property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_NOTIFY                 0x10
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver.

```
#define BT_GLOBAL_BUFF_LEN                          15000
```

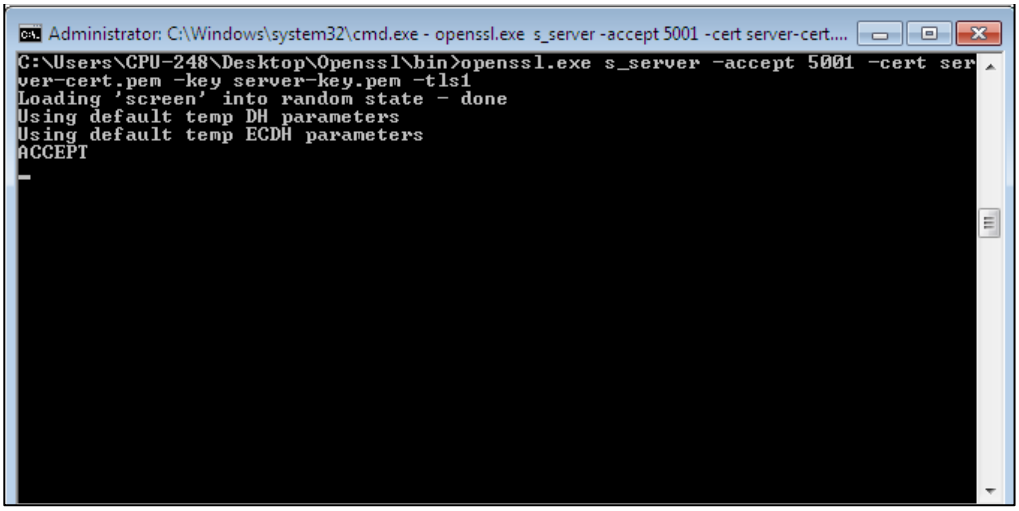
2.Open ***rsi_ble_config.h*** file and update/modify following macros,

```
#define RSI_BLE_PWR_INX                             30
#define RSI_BLE_PWR_SAVE_OPTIONS                    0
```

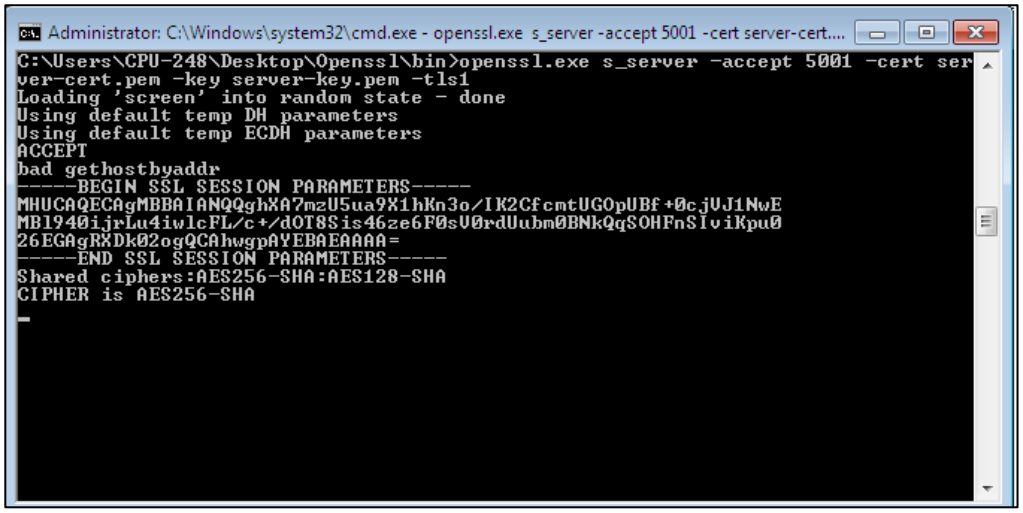
Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. In Windows PC2 which is connected to AP through LAN, Download the Openssl package from above mentioned link and run SSL server by giving following command:

Openssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1

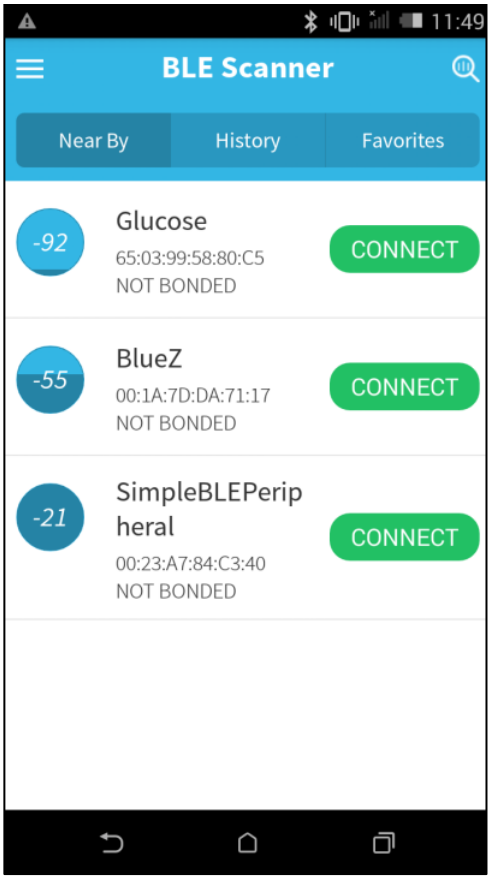


3. After the program gets executed, Silicon Labs BLE is in Advertising state and WLAN connects to Access Point and establishes SSL connectivity with SSL server opened on Windows PC1. Please refer the given below image for connection establishment at windows PC1,



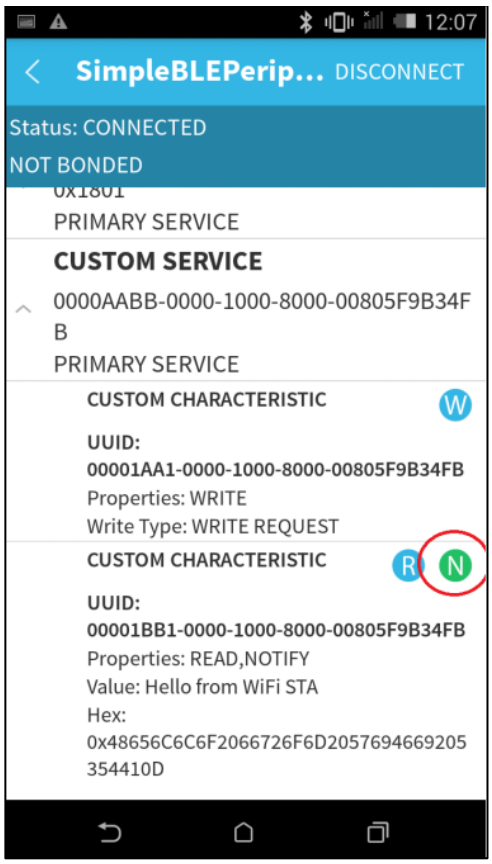
4. Open a BLE scanner App in the Smartphone and do the Scan.

5. In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT (Ex: "WLAN_BLE_SIMPLE_CHAT")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".

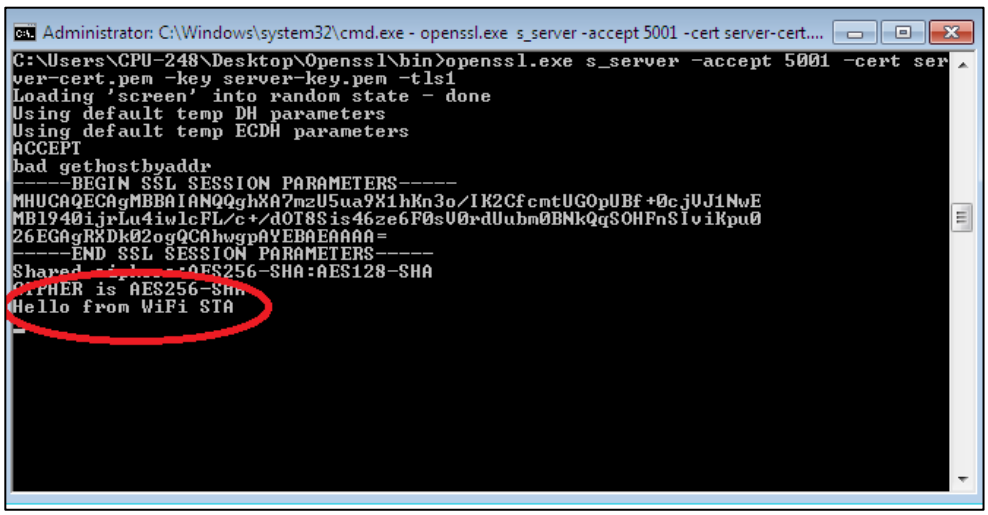


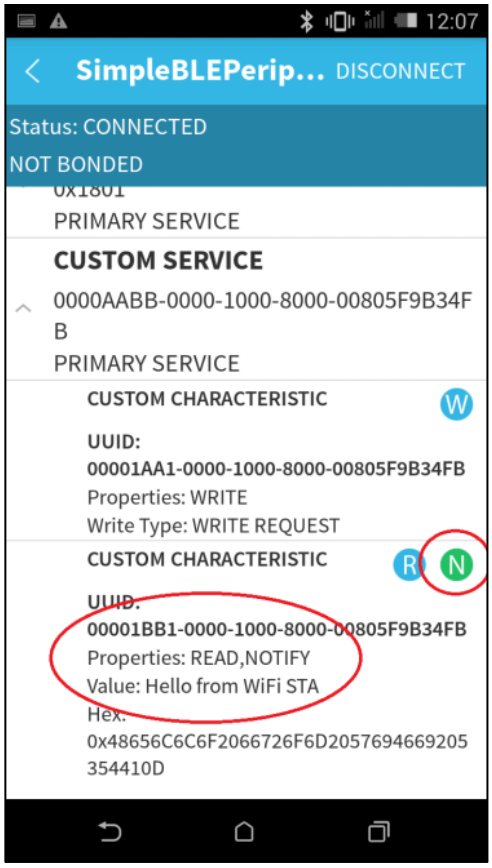
6. Initiate BLE connection from the App.
7. After successful connection, LE scanner displays the supported services of Silicon Labs module.
8. Select the attribute service which is added **RSI_BLE_NEW_SERVICE_UUID**

(Ex: 0xAABB) and enable Notification for attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID (Ex: 0x1BB1)** to receive data sent by Wi-Fi STA.



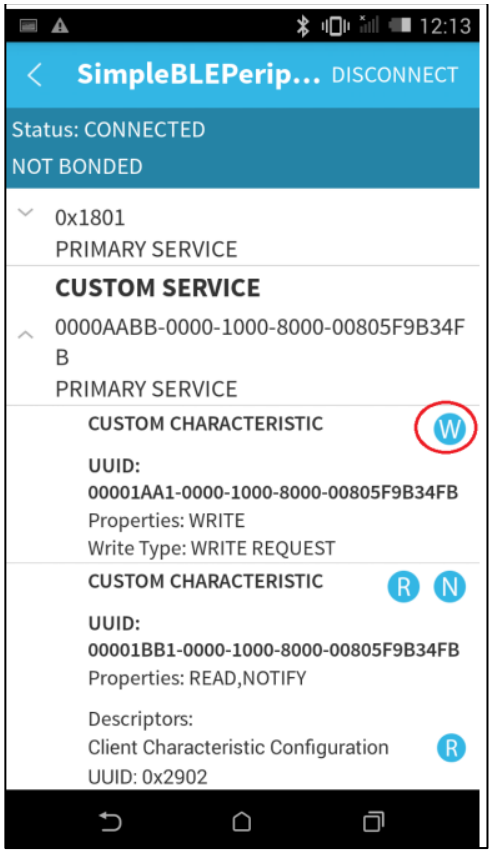
9. Now from SSL server (windows PC1), send a message (Ex: "Hello from WiFi STA") to Silicon Labs device. Silicon Labs device forwards the received message from SSL server to remote BTLE device which is connected to Silicon Labs BTLE device over BTLE protocol. User can observe the message notification on attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID (Ex: 0x1BB1)** in BTLE scanner app.

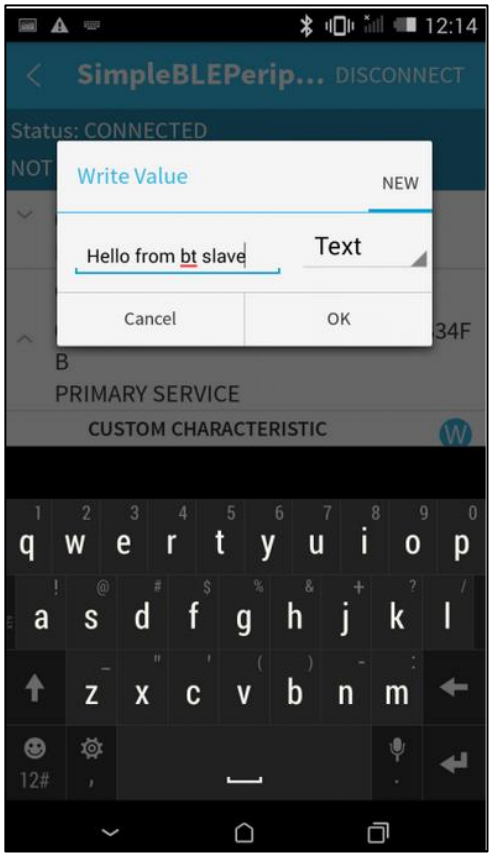


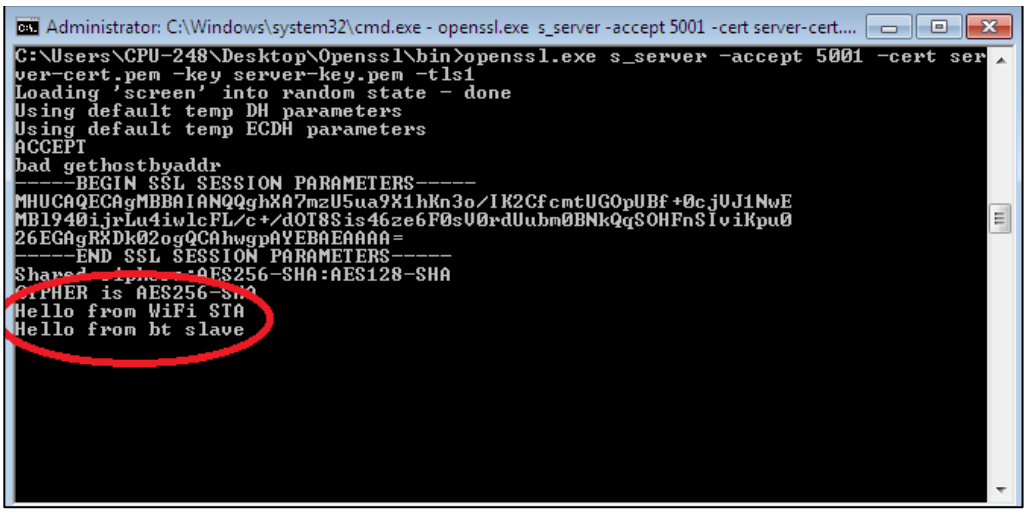


Note:
rsi_wlan_app_send_to_btle() function defined in rsi_ble_app.c to send message from WLAN task to BTLE task

10. Now send a message (Ex: "Hello from BT slave") from GATT client (from smart phone BLE scanner app) using attribute **RSI_BLE_ATTRIBUTE_1_UUID** (Ex: 0x1AA1) to Silicon Labs device. Silicon Labs device forwards the received message from BTLE remote device to SSL server over WiFi protocol. User can observe the message on UDP socket application.


12:13
SimpleBLEPerip... DISCONNECT
Status: CONNECTED
NOT BONDED
0x1801
PRIMARY SERVICE
CUSTOM SERVICE
0000AABB-0000-1000-8000-00805F9B34F
B
PRIMARY SERVICE
CUSTOM CHARACTERISTIC
W
UUID:
00001AA1-0000-1000-8000-00805F9B34FB
Properties: WRITE
Write Type: WRITE REQUEST
CUSTOM CHARACTERISTIC
R
N
UUID:
00001BB1-0000-1000-8000-00805F9B34FB
Properties: READ,NOTIFY
Descriptors:
Client Characteristic Configuration
R
UUID: 0x2902


12:14
SimpleBLEPerip... DISCONNECT
Status: CONNECTED
Write Value
NEW
Hello from bt slave
Text
Cancel
OK
34F
B
PRIMARY SERVICE
CUSTOM CHARACTERISTIC
W



11. After proper coex power save mode selection then power save command go to the module.
12. Note down power measurement by connecting Module to Agilent Power meter setup.

8.2 WLAN Station BLE Bridge

Overview

The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BLE) running in the same.
In this coex application, Silicon Labs BTLE device connects with remote BTLE device (Smart Phone) and Silicon Labs WiFi interface connects with an Access Point in station mode and do data transfer in BTLE and WiFi interfaces.
The coex application has WLAN and BLE tasks and acts as an interface between remote Smartphone BTLE device and remote PC which is connected to Access point. Smartphone interacts with BLE task, while remote PC interacts with WLAN task. When Smartphone connects and sends message to Silicon Labs device, BLE task accepts and sends to WLAN task, which in turn sends to remote PC which is connected to Access Point. Similarly, when remote PC sends message to Silicon Labs device, the message will be sent to Smartphone via BLE task.
Thus messages can be seamlessly transferred between PC and Smartphone.

Sequence of Events

WLAN Task

This Application explains user how to:

- Create Silicon Labs device as Station
- Connect Silicon Labs station to remote Access point
- Receive TCP data sent by connected station and forward to BT task
- Send data received by BT task to connected station using TCP over SSL client protocol.

BLE Task

This Application explains user how to:

- Create chat service
- Configure device in advertise mode
- Connect from Smart phone
- Configure device in power save profile mode 2
- Receive data sent by Smart phone and forward to WLAN task
- Send data received by WLAN task and send to Smart phone

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Access point
- Windows PC2 with SSL server application (openssl)

Note:
Download Open SSL for windows from below link,
http://ufpr.dl.sourceforge.net/project/gnuwin32/openssl/0.9.8h-1/openssl-0.9.8h-1-bin.zip

- BTLE supported Smart phone with GATT client application.

Note:
Install BLE scanner for GATT client application.

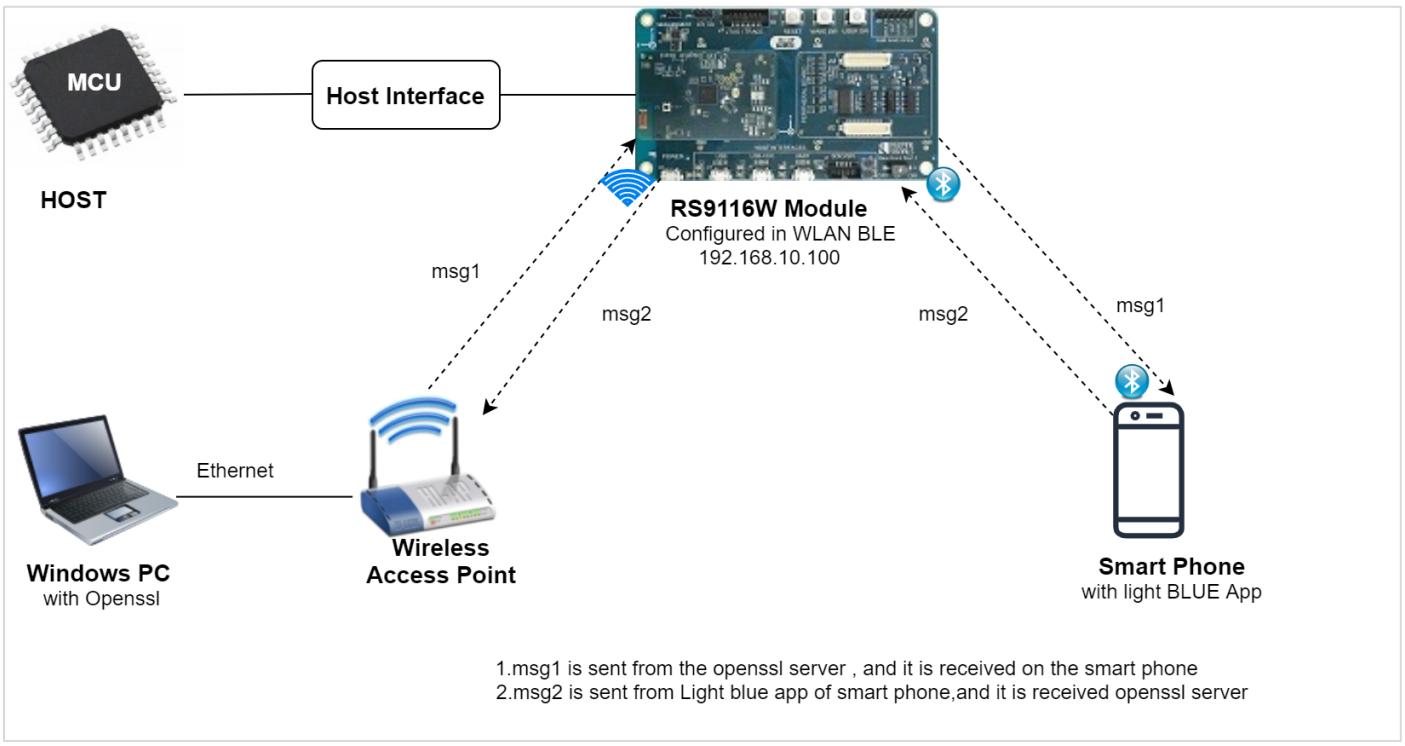


Figure 137: Setup Diagram for WLAN Station BLE Bridge Example

Configuration and Steps for Execution

Configuring the Application

Configuring the WLAN task

1. Open **rsi_wlan_app.c** file and update/modify the following macros:

   SSID refers to the name of the Access point.

```
#define SSID                                    "REDPINE_AP"
```

CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels

```
#define CHANNEL_NO                                  0
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.
Valid configuration is:
RSI_OPEN - For OPEN security mode
RSI_WPA - For WPA security mode
RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                               RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                                         "<psk>"
```

To Load certificate:

#define LOAD_CERTIFICATE 1

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.
By default, application loading "cacert.pem" certificate if LOAD_**CERTIFICATE** enable. In order to load different certificate, user has to follow the following steps:

- rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pem certificate into linear array form using python script provided in the release package "certificate_script.py"

Ex: If the certificate is wifi-user.pem. Give the command in the following way
python certificate_script.py ca-cert.pem
Script will generate wifiuser.pem in which one linear array named cacert contains the certificate.

- After conversion of certificate, update *rsi_ssl_client.c* source file by including the certificate file and by providing *the* required parameters to rsi_wlan_set_certificate API.

Note:

Once certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.

So, define LOAD_CERTIFICATE as 0, if certificate is already present in the Device.

Note:
All the certificates are given in the release package certificates.

DEVICE_PORT port refers SSL client port number

```
#define DEVICE_PORT                                 5001
```

SERVER_PORT port refers remote SSL server port number

```
#define SERVER_PORT                                 5001
```

SERVER_IP_ADDRESS refers remote peer IP address to connect with SSL server socket.
IP address should be in long format and in little endian byte order.
Example: To configure "192.168.10.100" as IP address, update the macro **SERVER_IP_ADDRESS** as **0x640AA8C0**.

```
#define SERVER_IP_ADDRESS                    0x640AA8C0
```

To configure IP address:
DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE                            1
```

Note:
If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.
(Or)
If user wants to configure STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.
Example: To configure "192.168.10.10" as IP address, update the macro **DEVICE_IP** as **0x0A0AA8C0**.

```
#define DEVICE_IP                            0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                              0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                              0x00FFFFFF
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                      RSI_DISABLE

#define RSI_FEATURE_BIT_MAP                  FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                    RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP           (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_SSL |
TCP_IP_TOTAL_SOCKETS_1 | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP           FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP       EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BIT_MAP        EXT_DYNAMIC_COEX_MEMORY
#define RSI_BAND                             RSI_BAND_2P4GHZ
```

Configuring the BLE Application

1. Open ***rsi_ble_app.c*** file and update/modify following macros,

   RSI_BLE_NEW_SERVICE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_NEW_SERVICE_UUID             0xAABB
```

   RSI_BLE_ATTRIBUTE_1_UUID refers to the attribute type of the first attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#defineRSI_BLE_ATTRIBUTE_1_UUID              0x1AA1
```

RSI_BLE_ATTRIBUTE_2_UUID refers to the attribute type of the second attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#defineRSI_BLE_ATTRIBUTE_2_UUID                    0x1BB1
```

RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#defineRSI_BLE_MAX_DATA_LEN                         20
```

RSI_BLE_APP_DEVICE_NAME refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#defineRSI_BLE_APP_DEVICE_NAME                     "WLAN_BLE_SIMPLE_CHAT"
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.

```
#defineRSI_BLE_CHAR_SERV_UUID                       0x2803
```

RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#defineRSI_BLE_CLIENT_CHAR _UUID                    0x2902
```

RSI_BLE_ATT_PROPERTY_READ is used to set the READ property to an attribute value.

```
#defineRSI_BLE_ATT_PROPERTY_READ                    0x02
```

RSI_BLE_ATT_PROPERTY_WRITE is used to set the WRITE property to an attribute value.

```
#defineRSI_BLE_ATT_PROPERTY_WRITE                   0x08
```

RSI_BLE_ATT_PROPERTY_NOTIFY is used to set the NOTIFY property to an attribute value.

```
#defineRSI_BLE_ATT_PROPERTY_NOTIFY                  0x10
```

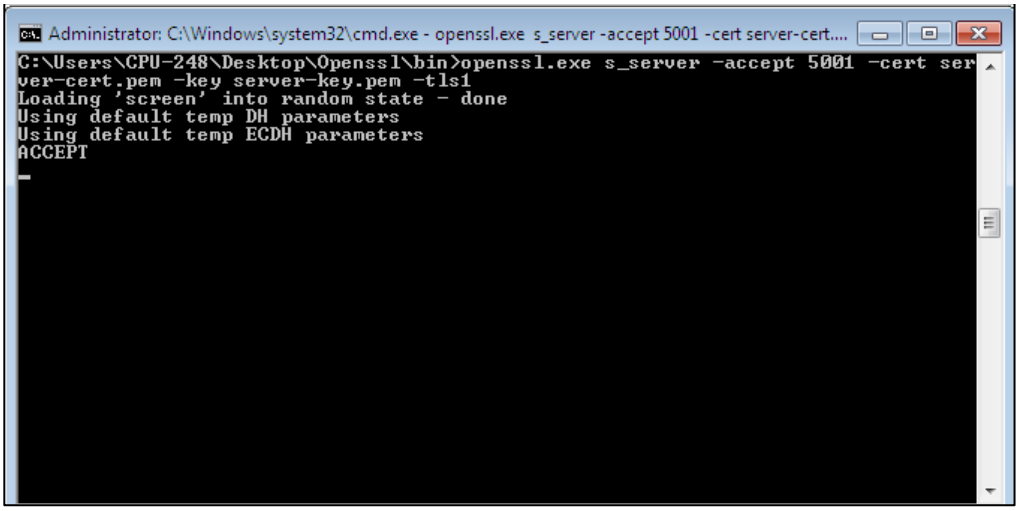
BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver.

```
#defineBT_GLOBAL_BUFF_LEN                          15000
```

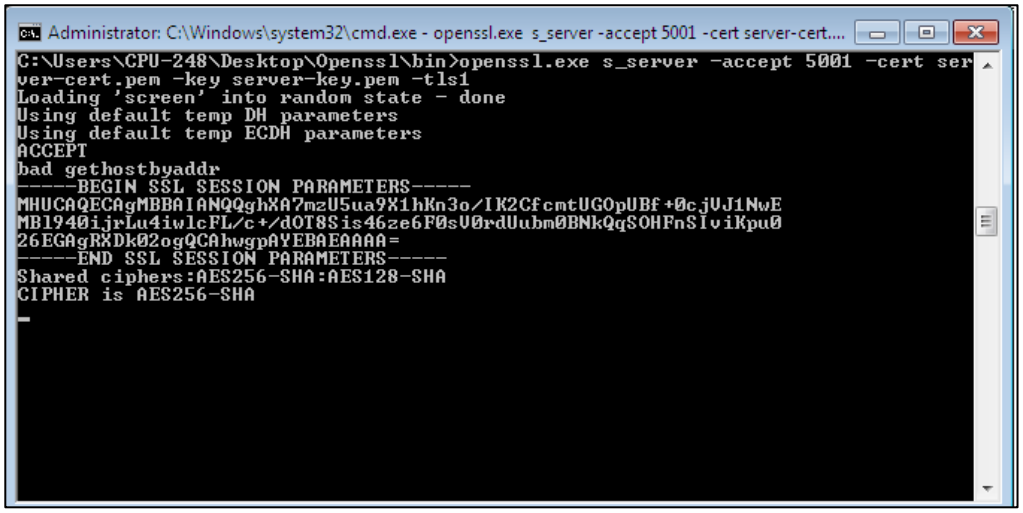
Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
2. In Windows PC2 which is connected to AP through LAN, Download the Openssl package from above mentioned link and run SSL server by giving following command:

Openssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1

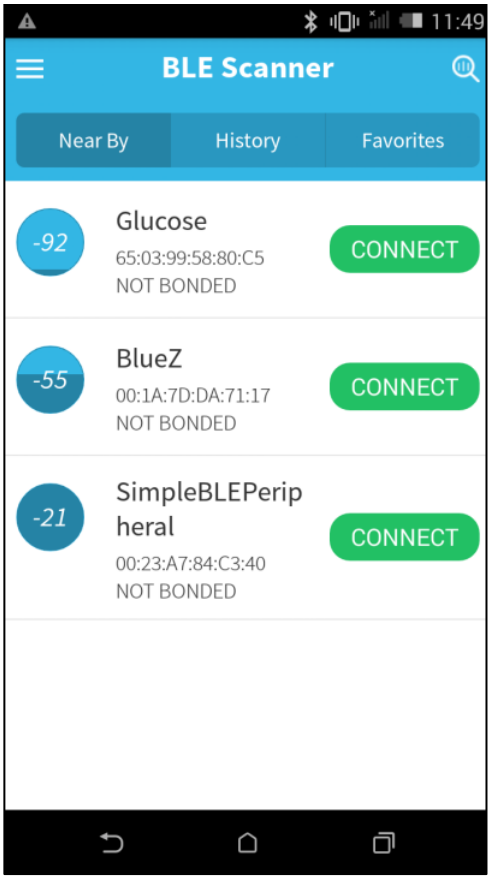


3. After the program gets executed, Silicon Labs BLE is in Advertising state and WLAN connects to Access Point and establishes SSL connectivity with SSL server opened on Windows PC1. Please refer the given below image for connection establishment at windows PC1,



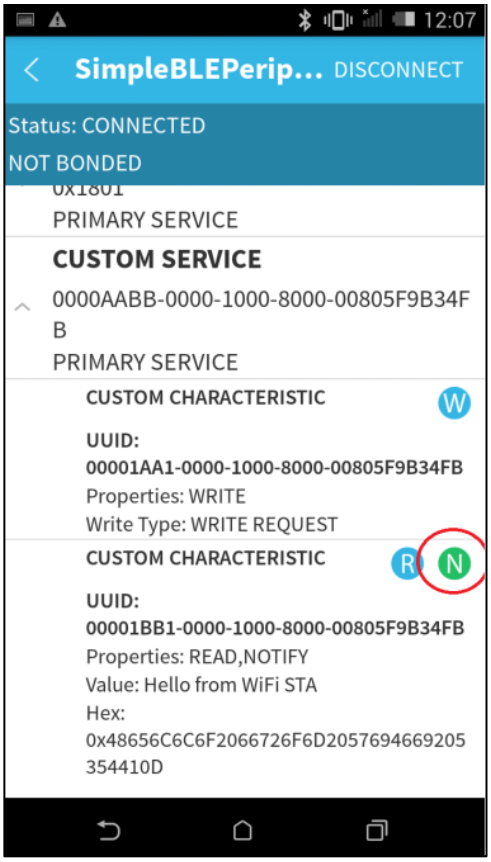
4. Open a BLE scanner App in the Smartphone and do the Scan.

5. In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT (Ex: "WLAN_BLE_SIMPLE_CHAT")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".

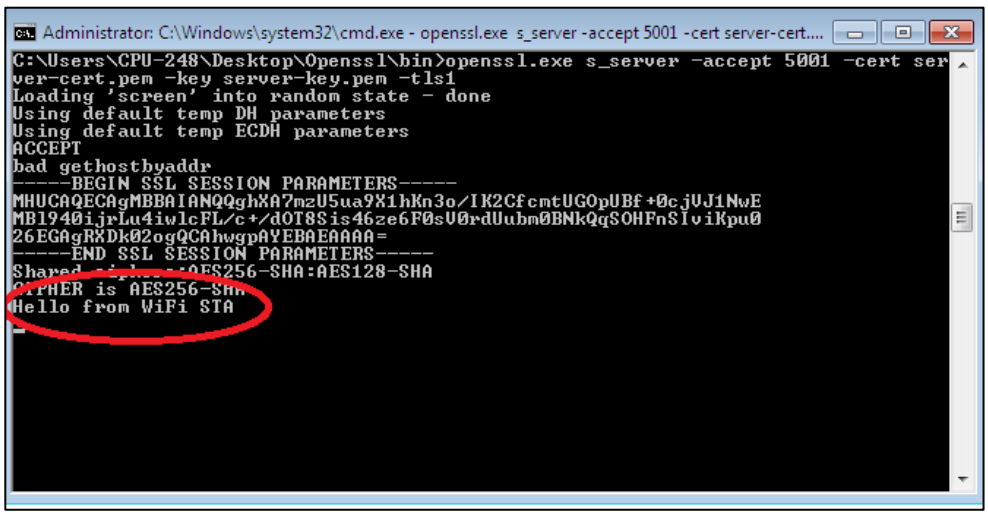


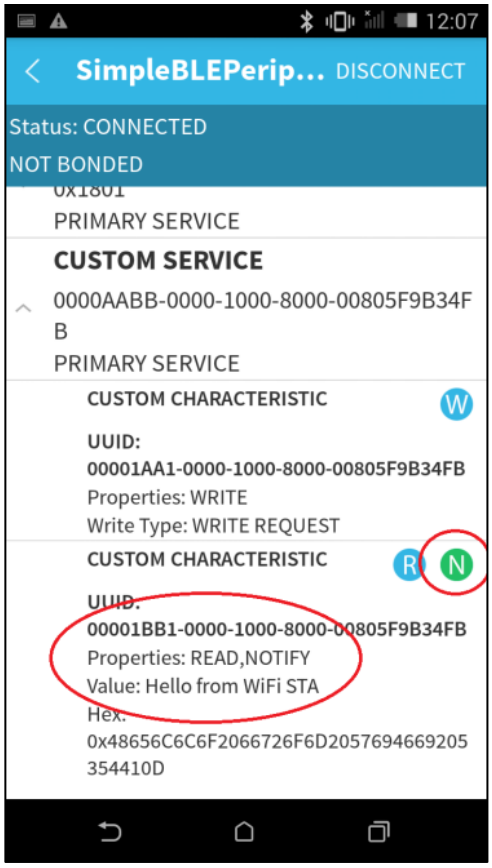
6. Initiate BLE connection from the App.
7. After successful connection, LE scanner displays the supported services of Silicon Labs module.
8. Select the attribute service which is added **RSI_BLE_NEW_SERVICE_UUID**

(Ex: 0xAABB) and enable Notification for attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID (Ex: 0x1BB1)** to receive data sent by Wi-Fi STA.



9. Now from SSL server (windows PC1), send a message (Ex: "Hello from WiFi STA") to Silicon Labs device. Silicon Labs device forwards the received message from SSL server to remote BTLE device which is connected to Silicon Labs BTLE device over BTLE protocol. User can observe the message notification on attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID (Ex: 0x1BB1)** in BTLE scanner app.

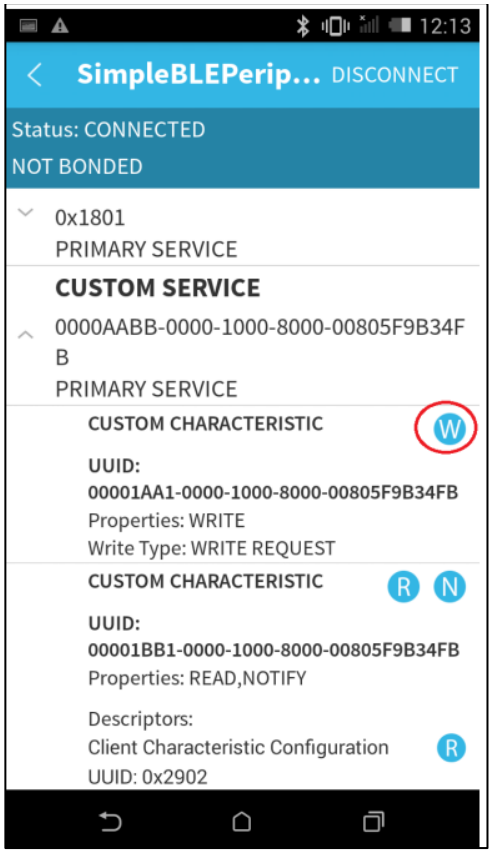


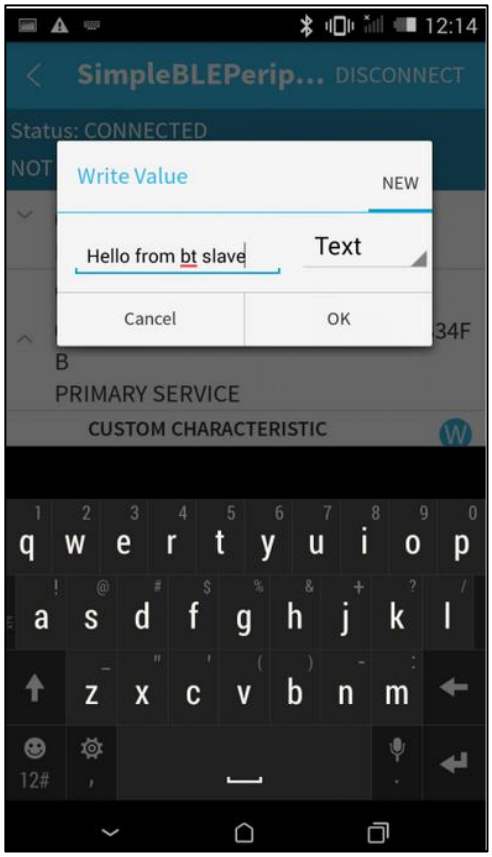


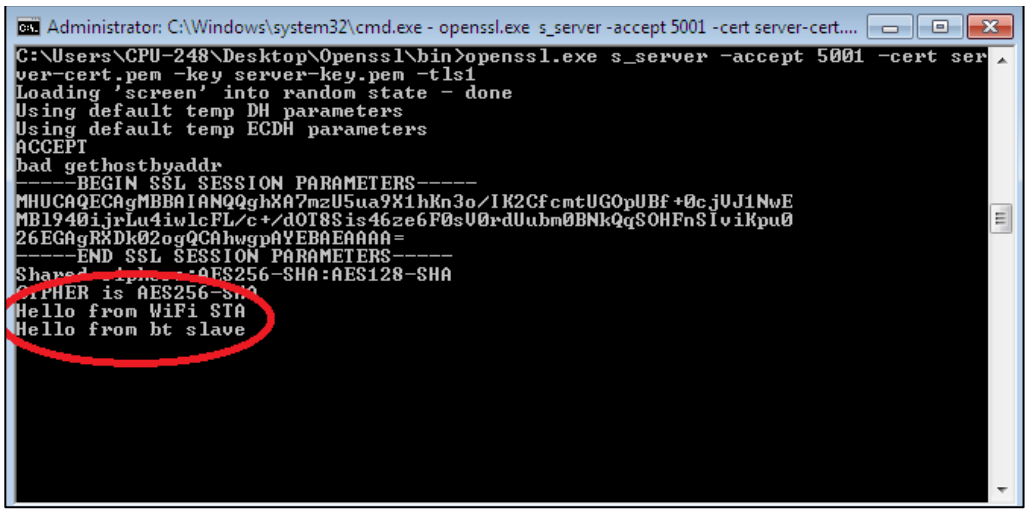
Note:

rsi_wlan_app_send_to_btle() function defined in rsi_ble_app.c to send message from WLAN task to BTLE task.

10. Now send a message (Ex: "Hello from BT slave") from GATT client (from smart phone BLE scanner app) using attribute **RSI_BLE_ATTRIBUTE_1_UUID** (Ex: 0x1AA1) to Silicon Labs device. Silicon Labs device forwards the received message from BTLE remote device to SSL server over WiFi protocol. User can observe the message on UDP socket application.


12:13
SimpleBLEPerip... DISCONNECT
Status: CONNECTED
NOT BONDED
0x1801
PRIMARY SERVICE
CUSTOM SERVICE
0000AABB-0000-1000-8000-00805F9B34F
B
PRIMARY SERVICE
CUSTOM CHARACTERISTIC
W
UUID:
00001AA1-0000-1000-8000-00805F9B34FB
Properties: WRITE
Write Type: WRITE REQUEST
CUSTOM CHARACTERISTIC
R
N
UUID:
00001BB1-0000-1000-8000-00805F9B34FB
Properties: READ,NOTIFY
Descriptors:
Client Characteristic Configuration
R
UUID: 0x2902


12:14
SimpleBLEPerip... DISCONNECT
Status: CONNECTED
Write Value
NEW
Hello from bt slave
Text
Cancel
OK
34F
B
PRIMARY SERVICE
CUSTOM CHARACTERISTIC
W



Note: rsi_bt_app_send_to_wlan() function defined in rsi_wlan_app.c to send message from BTLE task to WLAN task.

8.3 WLAN Station BLE Dual Role Bridge

Overview

The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BLE) running in the same device as a LE master as well as slave.

Description

The coex application has WLAN and BLE tasks and acts as an interface between Smartphone and sensor tag and PC.
Smartphone and sensor tag interacts with BLE task, while PC Both PC and Silicon Labs WLAN would be connected to a Wireless Access Point, thus both are connected together wirelessly interacts with WLAN task.
When Smartphone and sensor tag connects and sends messages/notifications to Silicon Labs, BLE task accepts and sends to WLAN task, which in turn sends to Access Point connected PC.
Thus messages can be seamlessly transferred from Smartphone and sensor tag to Windows PC.

Details of the Application

Silicon Labs WLAN acts as a Station and connects to an Access Point
Silicon Labs BLE acts as a Peripheral (Slave) device and Central (Master), while Smart phone acts as a Central (Master) device and Sensor tag acts as a Peripheral (Slave) device.
Initially, proprietary Simple chat service is created at GATT Server (Silicon Labs device) to facilitate message exchanges.

- The WLAN task (running in Silicon Labs device) mainly includes following steps.

1. Connects to a Access Point
2. Exchanges data over SSL socket with the peer(Windows PC)

- The BLE task (running in Silicon Labs device) mainly includes following steps.

1. Creates chat service
2. Configures the device to scanning.
3. After connection of slave configures the device to Advertise

WLAN and BLE tasks forever run in the application to serve the asynchronous events.

Sequence of Events

WLAN Task

This Application explains user how to:

- Create Silicon Labs device as Station

- Connect Silicon Labs station to remote Access point
- Receive TCP data sent by connected station and forward to BT task
- Send data received by BT task to connected station using TCP over SSL client protocol.

BLE Task

This Application explains user how to:

- Create chat service
- Configure device in advertise mode
- Connect from Smart phone
- Configure device in master mode
- Connect to remote device
- Receive data sent by Smart phone and forward to WLAN task
- Send data received by WLAN task and send to Smart phone

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with LE Application (Ex: Light blue App in iPad mini)
- WLAN Access Point and a Windows PC
- TI simple sensor tag or 3rd party BLE dongle as a slave.

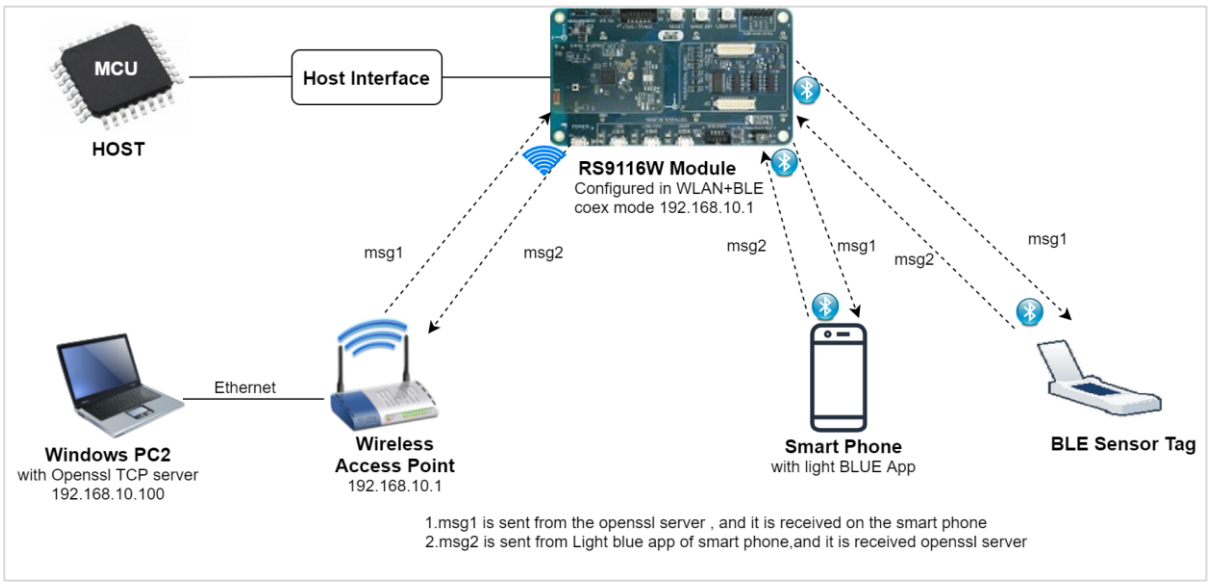


Figure 138: Setup Diagram for WLAN Station BLE Dual Role Bridge Example

Configuration and Steps for Execution

Configuring the Application

Configuring the WLAN task

1. Open **rsi_wlan_app.c** file and update/modify following macros,

SSID refers to the Access point to which user wants to connect.
SECURITY_TYPE is the security type of the Access point.
PSK refers to the secret key if the Access point is configured in WPA/WPA2 security modes.

```
#define SSID                                "REDPINE_AP"
#define SECURITY_TYPE                       <security-type>
#define PSK                                 ""
```

Load the SSL CA- certificate using rsi_wlan_set_certificate API after wireless initialization.

> **Note:**
> rsi_wlan_set_certificate expects the certificate in the form of linear array. Python script is provided in the release package named **"certificate_script.py"** in the following path "certificates" to convert the pem certificate into linear array.

Example: If the certificate is ca-cert.pem, give the command as
python certificate_script.py ca-cert.pem
The script will generate cacert.pem, in which one linear array named *cacert* contains the certificate

Enable/Disable DHCP mode

1 – Enables DHCP mode (gets the IP from DHCP server)
0 – Disables DHCP mode

```
#define DHCP_MODE                           1
```

If DHCP mode is disabled, then change the following macros to configure static IP address

IP address to be configured to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.101" as IP address, update the macro **DEVICE_IP** as **0x650AA8C0**.

```
#define DEVICE_IP                           0x650AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro **GATEWAY** as **0x010AA8C0**

```
#define GATEWAY                             0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                             0x00FFFFFF
```

To establish TCP connection and transfer data to the remote socket configure the below macros. If SSL is enabled, open the socket with protocol type as 1.

Internal socket port number.

```
#define DEVICE_PORT                         5001
```

Port number of the remote server

```
#define SERVER_PORT                         5001
```

IP address of the remote server

```
#define SERVER_IP_ADDRESS                   0x650AA8C0
```

Include rsi_wlan_app.c , rsi_ble_app.c and main.c files in the project, build and launch the application

Open SSL server socket on remote machine

For example, to open SSL socket with port number 5001 on remote side, use the command as given below
openssl s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
Example: openssl s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1_2

> **Note:**
> All the certificates are given in the release package.

2. Enable/Disable power save

   1 – Enables Power save mode
   0 – Disables Power save mode

```
#define WLAN_POWER_SAVE                    0
```

   By default, Power save is disabled.

3. Open ***rsi_wlan_config.h*** file and update/modify following macros:

```
#define CONCURRENT_MODE                    RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                  RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP         (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_SSL |
TCP_IP_TOTAL_SOCKETS_1 | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP         FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP     EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BIT_MAP      EXT_DYNAMIC_COEX_MEMORY
#define RSI_BAND                           RSI_BAND_2P4GHZ
```

Configuring the BLE Application

1. Open ***rsi_ble_app.c*** file and update/modify following macros,

- RSI_BLE_NEW_SERVICE_UUID - The attribute value of the newly created service. Ex: 0xAABB
- RSI_BLE_ATTRIBUTE_1_UUID - The attribute type of the first attribute under this Service. Ex: 0x1AA1
- RSI_BLE_ATTRIBUTE_2_UUID - The attribute type of the second attribute under this Service. Ex: 0x1BB1
- RSI_BLE_MAX_DATA_LEN - Maximum length of the attribute data(limited to max of 20 bytes)
- RSI_BLE_APP_DEVICE_NAME - Name of the Silicon Labs device to appear during Scanning by peer devices.
- BLE_PS_ENABLE – To Enable/Disable power save.
- RSI_BLE_DEV_ADDR – Address of the peer device to connect.
- BLE_DUAL_ROLE_FIRST_MASTER – To create local device as a master first.

Following are the **non-configurable** macros in the application.

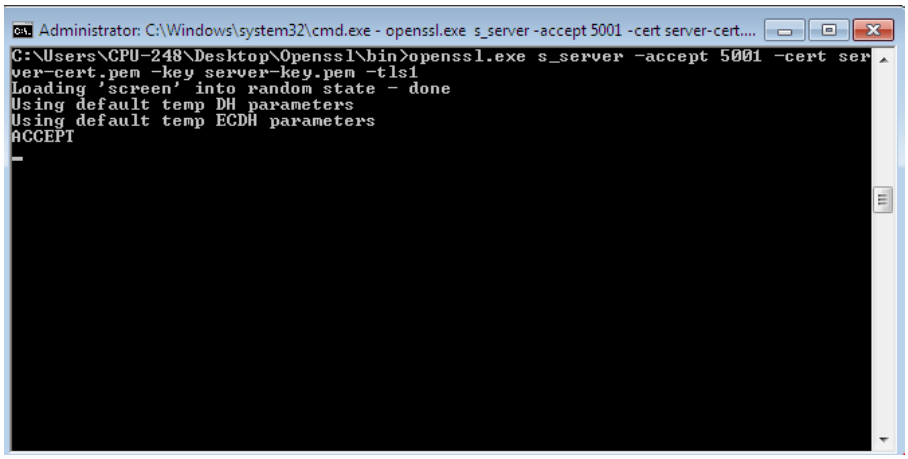
- RSI_BLE_ATT_PROPERTY_READ – Used to set read property to an attribute value.
- RSI_BLE_ATT_PROPERTY_WRITE - Used to set write property to an attribute value.
- RSI_BLE_ATT_PROPERTY_NOTIFY - Used to set notify property to an attribute value.
- RSI_BLE_CHAR_SERV_UUID - The attribute type of the characteristics to be added in a service. Ex: 0x2803
- RSI_BLE_CLIENT_CHAR_UUID - The attribute type of the client characteristics descriptor to be added in a service characteristic. Ex: 0x2902
- BT_GLOBAL_BUFF_LEN – Number of bytes required for the Application and the Driver.

Executing the coex Application

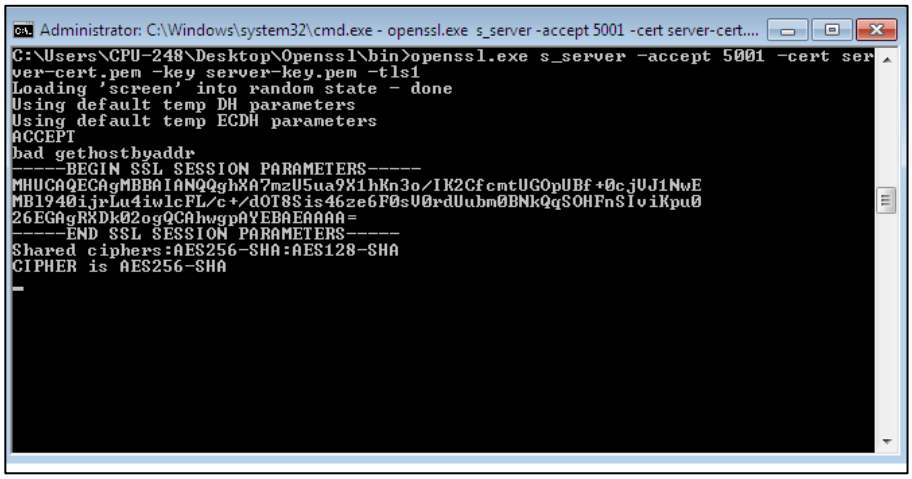
1. Connect Silicon Labs device to the Windows PC running KEIL or IAR IDE
2. Build and launch the application.
3. Advertise 3rd party TI sensor tag.

4. After the program gets executed, If BLE_DUAL_ROLE_FIRST_MASTER macro as 1 then Silicon Labs BLE is in scanning state and it creates a connection with TI sensor tag. If BLE_DUAL_ROLE_FIRST_MASTER macro as 0 then Silicon Labs BLE is in Advertising state and WLAN has established SSL socket with peer(PC).
5. Open a LE App in the Smartphone and do scan.
6. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.

   Openssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
   Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls



WLAN connects to Access Point and establishes SSL connectivity with SSL server opened on Windows PC1. Please refer the given below image for connection establishment at windows PC1,

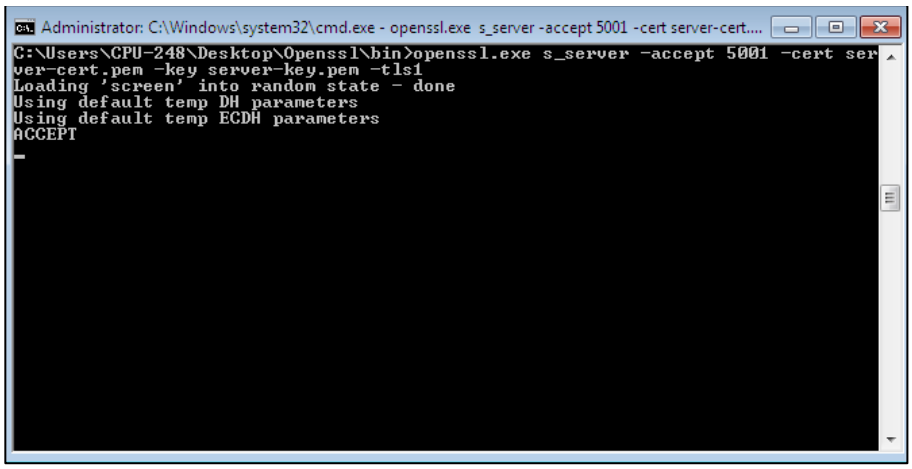


send a message or notification from the App to Silicon Labs BLE. Observe this message in the PC connected via SSL socket with Silicon Labs WLAN.

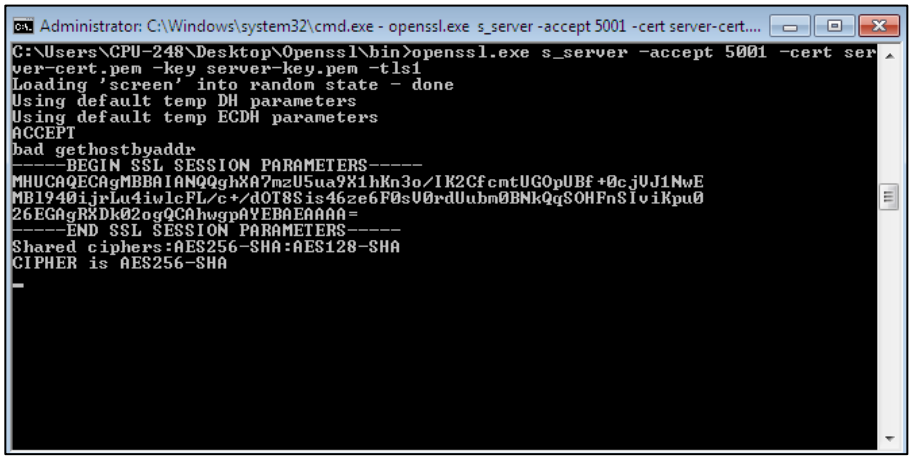
- rsi_ble_app_send_to_wlan() function defined in rsi_wlan_app.c to send message from BLE task to WLAN task.

In Windows PC2 which is connected to AP through LAN, Download the Openssl package from above mentioned link and run SSL server by giving following command:

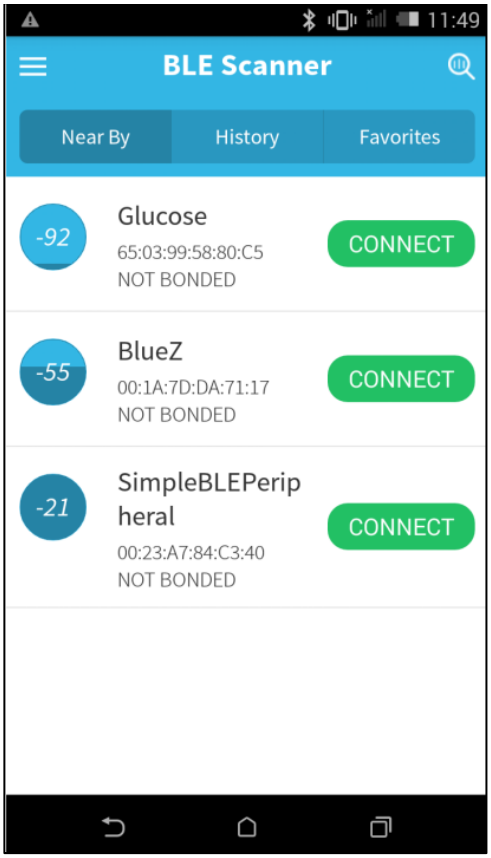
Openssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1



7. After the program gets executed, Silicon Labs BLE is in Advertising state and WLAN connects to Access Point and establishes SSL connectivity with SSL server opened on Windows PC1. Please refer the given below image for connection establishment at windows PC1,

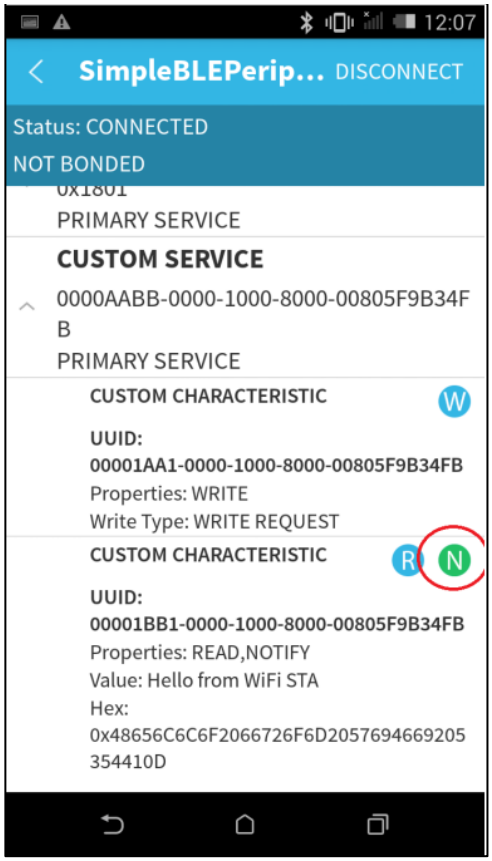


8. Open a BLE scanner App in the Smartphone and do the Scan.
9. In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT (Ex: "WLAN_BLE_SIMPLE_CHAT")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".

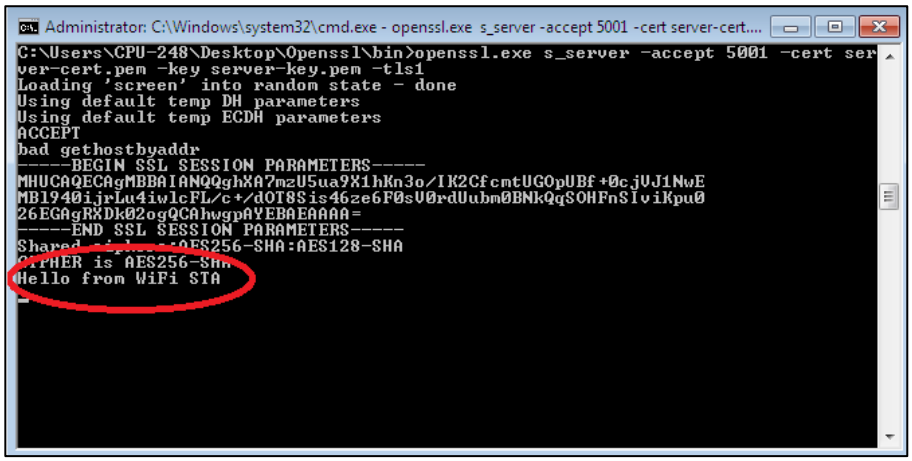


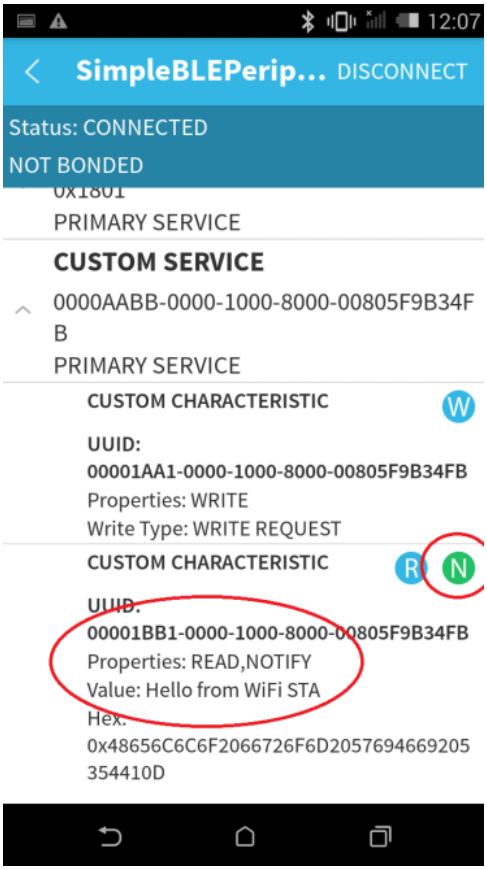
10. Initiate BLE connection from the App.
    - After successful connection, LE scanner displays the supported services of Silicon Labs module.
    - Select the attribute service which is added **RSI_BLE_NEW_SERVICE_UUID**

    (Ex: 0xAABB) and enable Notification for attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID(Ex: 0x1BB1)** to receive data sent by WiFi STA.



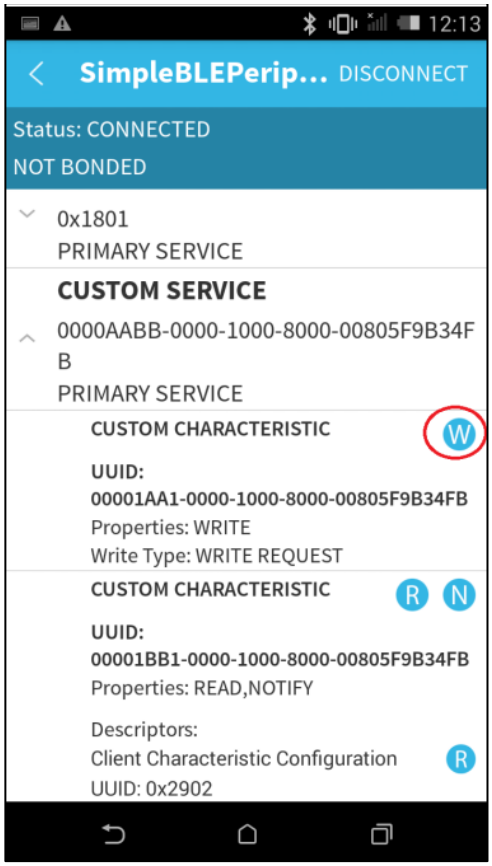
11. Now from SSL server (windows PC1), send a message (Ex: "Hello from WiFi STA") to Silicon Labs device. Silicon Labs device forwards the received message from SSL server to remote BTLE device which is connected to Silicon Labs BTLE device over BTLE protocol. User can observe the message notification on attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID(Ex: 0x1BB1)** in BTLE scanner app.

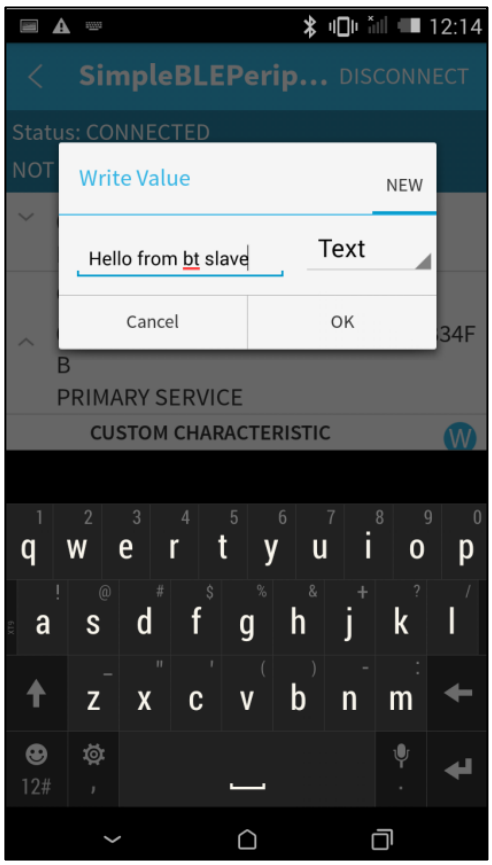


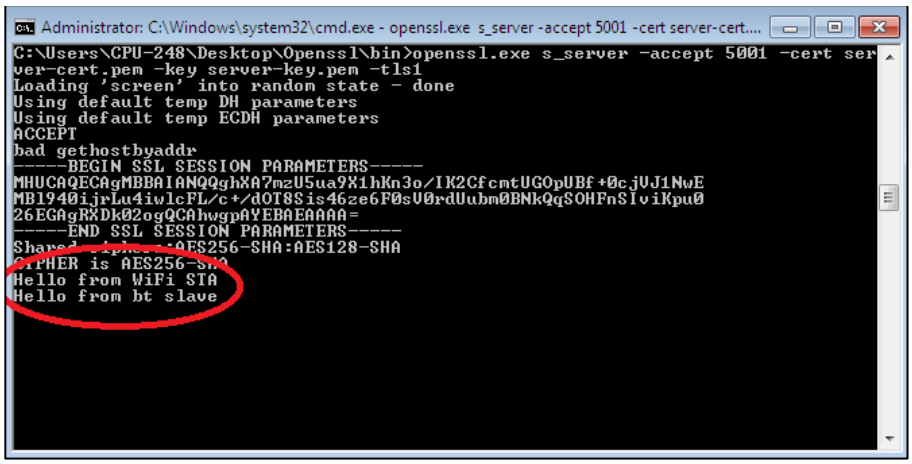


Note:
rsi_wlan_app_send_to_btle() function defined in **rsi_ble_app.c** to send message from WLAN task to BTLE task.

12. Now send a message (Ex: "Hello from bt slave") from GATT client (from smart phone BLE scanner app) using attribute **RSI_BLE_ATTRIBUTE_1_UUID** (Ex: 0x1AA1) to Silicon Labs device. Silicon Labs device forwards the received message from BTLE remote device to SSL server over WiFi protocol. User can observe the message on UDP socket application


12:13
SimpleBLEPerip... DISCONNECT
Status: CONNECTED
NOT BONDED
0x1801
PRIMARY SERVICE
CUSTOM SERVICE
0000AABB-0000-1000-8000-00805F9B34F
B
PRIMARY SERVICE
CUSTOM CHARACTERISTIC
W
UUID:
00001AA1-0000-1000-8000-00805F9B34FB
Properties: WRITE
Write Type: WRITE REQUEST
CUSTOM CHARACTERISTIC
R
N
UUID:
00001BB1-0000-1000-8000-00805F9B34FB
Properties: READ,NOTIFY
Descriptors:
Client Characteristic Configuration
R
UUID: 0x2902


12:14
SimpleBLEPerip... DISCONNECT
Status: CONNECTED
Write Value
NEW
Hello from bt slave
Text
Cancel
OK
B
PRIMARY SERVICE
CUSTOM CHARACTERISTIC
W



Note:
rsi_bt_app_send_to_wlan() function defined in rsi_wlan_app.c to send message from BTLE task to WLAN task.

8.4 WLAN Station BLE Multiple Slaves Bridge

Overview

The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BLE) running in the same device.

Description

The coex application has WLAN and BLE tasks and acts as an interface between TI sensor tag and PC.
TI Sensor tag interacts with BLE task, while PC Both PC and Silicon Labs WLAN would be connected to a Wireless Access Point, thus both are connected together wirelessly interacts with WLAN task.
When TI sensor tag connects and sends notifications to Silicon Labs, BLE task receive and sends to WLAN task, which in turn sends to Access Point connected PC.
Thus messages can be seamlessly receives from TI sensor tag to Windows PC.

Details of the Application

Silicon Labs WLAN acts as a Station and connects to an Access Point
Silicon Labs BLE acts as a Central (Master) with GATT Server running in it, while TI Sensor tag acts as a Peripheral (Slave) device.
Initially, proprietary simple chat service is created at GATT Server (Silicon Labs device) to facilitate message/notification exchanges.

1. The WLAN task (running in Silicon Labs device) mainly includes following steps.

- Connects to a Access Point
- Exchanges data over SSL socket with the peer(Windows PC)

1. The BLE task (running in Silicon Labs device) mainly includes following steps.

- Configures the device to scanning.

WLAN and BLE tasks forever run in the application to serve the asynchronous events

Sequence of Events

WLAN Task

This Application explains user how to:

- Create Silicon Labs device as Station
- Connect Silicon Labs station to remote Access point
- Receive TCP data sent by connected station and forward to BT task

- Send data received by BT task to connected station using TCP over SSL client

BLE Task

This Application explains user how to:

- Create chat service
- Configure device in Central mode
- Connect to multiple slaves one by one
- Receive data sent by Smart phone and forward to WLAN task
- Send data received by WLAN task and send to Smart phone

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- WLAN Access Point and a Windows PC with openssl support
- TI simple sensor tag or 3rd party BLE dongles

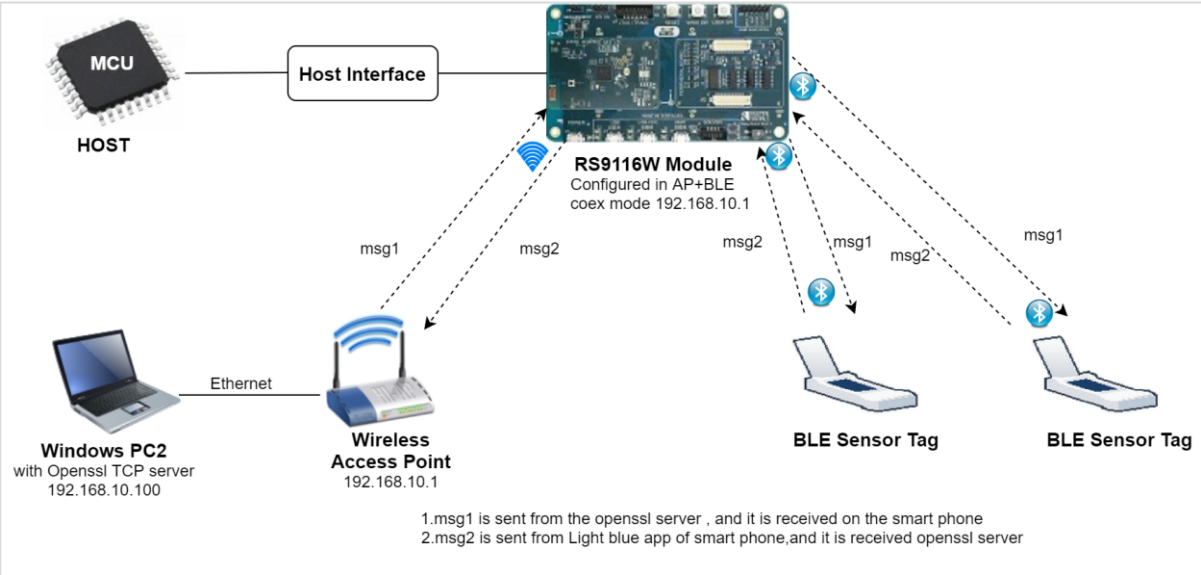


Figure 139: Setup Diagram for Master WLAN Station BLE Multiple Slaves Bridge Example

Configuration and Steps for Execution

Configuring the Application

Configuring the WLAN task

1. Open **rsi_wlan_app.c** file and update/modify following macros,

   SSID refers to the Access point to which user wants to connect.
   SECURITY_TYPE is the security type of the Access point.
   PSK refers to the secret key if the Access point is configured in WPA/WPA2 security modes.

```
#define SSID                                    "REDPINE_AP"
#define SECURITY_TYPE                           RSI_OPEN
#define PSK                                     ""
```

   Load the SSL CA- certificate using rsi_wlan_set_certificate API after wireless initialization.

Note:

rsi_wlan_set_certificate expects the certificate in the form of linear array. Python script is provided in the release package named **"certificate_script.py"** in the following path "certificates" to convert the pem certificate into linear array.

Example: If the certificate is ca-cert.pem, give the command as
python certificate_script.py ca-cert.pem
The script will generate cacert.pem, in which one linear array named *cacert* contains the certificate

Enable/Disable DHCP mode

1 – Enables DHCP mode (gets the IP from DHCP server)
0 – Disables DHCP mode

```
#define DHCP_MODE                                1
```

If DHCP mode is disabled, then change the following macros to configure static IP address

IP address to be configured to the device should be in long format and in little endian byte order.
Example: To configure "192.168.10.101" as IP address, update the macro **DEVICE_IP** as **0x650AA8C0**.

```
#define DEVICE_IP                                0x650AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order
Example: To configure "192.168.10.1" as Gateway, update the macro **GATEWAY** as **0x010AA8C0**

```
#define GATEWAY                                  0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                  0x00FFFFFF
```

To establish TCP connection and transfer data to the remote socket configure the below macros. If SSL is enabled, open the socket with protocol type as 1.

Internal socket port number.

```
#define DEVICE_PORT                              5001
```

Port number of the remote server

```
#define SERVER_PORT                              5001
```

IP address of the remote server

```
#define SERVER_IP_ADDRESS                        0x650AA8C0
```

Include rsi_wlan_app.c , rsi_ble_app.c and main.c files in the project, build and launch the application

Open SSL server socket on remote machine

For example, to open SSL socket with port number 5001 on remote side, use the command as given below
openssl s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>
Example: openssl s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1_2

Note:

All the certificates are given in the release package.

Enable/Disable power save

1 – Enables Power save mode
0 – Disables Power save mode

```
#define WLAN_POWER_SAVE                         1
```

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT| TCP_IP_FEAT_SSL |
TCP_IP_TOTAL_SOCKETS_1 | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BIT_MAP           EXT_DYNAMIC_COEX_MEMORY
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Configuring the BLE Application

1. Open ***rsi_ble_app.c*** file and update/modify following macros,

- RSI_BLE_NEW_SERVICE_UUID - The attribute value of the newly created service. Ex: 0xAABB
- RSI_BLE_ATTRIBUTE_1_UUID - The attribute type of the first attribute under this Service. Ex: 0x1AA1
- RSI_BLE_ATTRIBUTE_2_UUID - The attribute type of the second attribute under this Service. Ex: 0x1BB1
- RSI_BLE_MAX_DATA_LEN - Maximum length of the attribute data(limited to max of 20 bytes)
- RSI_BLE_APP_DEVICE_NAME - Name of the Silicon Labs device to appear during Scanning by peer devices.
- BLE_PS_ENABLE – To Enable/Disable power save.
- MAX_NUM_OF_SLAVES – Maximum number of slaves.
- RSI_BLE_DEV_1_ADDR – Address of the 1st peer device to connect.
- RSI_BLE_DEV_2_ADDR – Address of the 2nd peer device to connect.
- RSI_BLE_DEV_3_ADDR – Address of the 3rd peer device to connect.

Update the BLE configuration file to enable Multiple slaves and to configure number of slaves:

2. Open ***rsi_ble_config.h*** file and update/modify following macros

a. RSI_BLE_MAX_NBR_SLAVES – Maximum number of slaves supported by firmware which is used in Opermode ble feature bitmap

Note:

- The current document explains in refer to 3 slaves, but the application has max of 8 slaves.
- The current application consists 3 slaves, 20 attributes, 5 services in BLE and open SSL feature in WiFi. If incase to increase slaves, services or attributes please refer WiSeConnect_TCPIP_Feature_Selection_v1.x.x document for memory limitations.

b. RSI_BLE_MAX_NBR_ATT_REC – Maximum number of attribute records.

c. RSI_BLE_MAX_NBR_ATT_SERV – Maximum number of services.

Following are the **non-configurable** macros in the application.

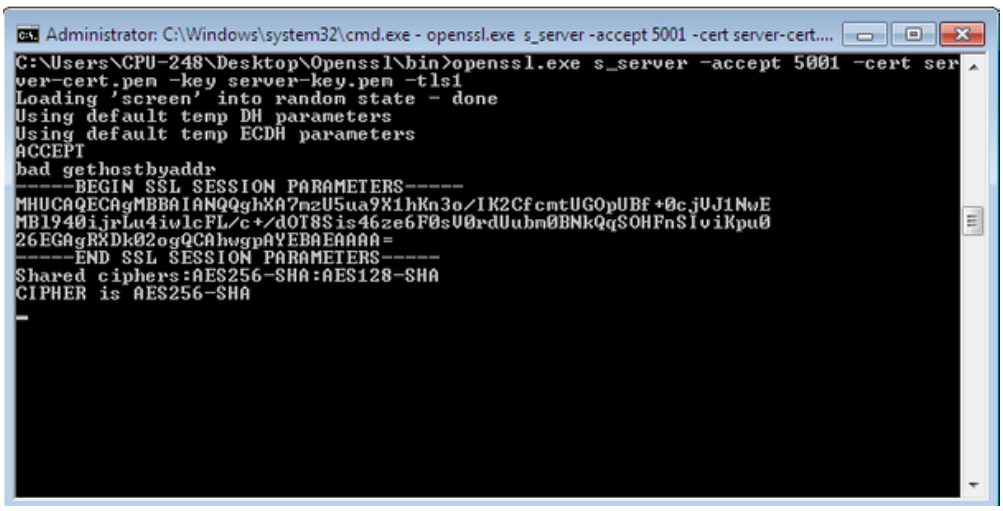
- RSI_BLE_ATT_PROPERTY_READ – Used to set read property to an attribute value.
- RSI_BLE_ATT_PROPERTY_WRITE - Used to set write property to an attribute value.
- RSI_BLE_ATT_PROPERTY_NOTIFY - Used to set notify property to an attribute value.

- RSI_BLE_CHAR_SERV_UUID - The attribute type of the characteristics to be added in a service. Ex: 0x2803
- RSI_BLE_CLIENT_CHAR_UUID - The attribute type of the client characteristics descriptor to be added in a service characteristic. Ex: 0x2902
- BT_GLOBAL_BUFF_LEN – Number of bytes required for the Application and the Driver

Executing the Application

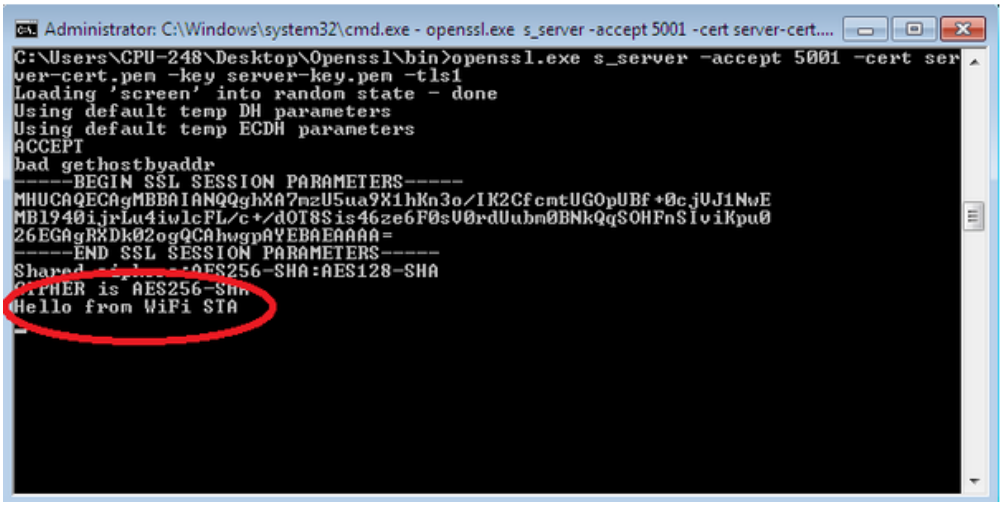
1. Connect Silicon Labs device to the Windows PC running KEIL or IAR IDE
2. Build and launch the application.
3. Advertise 3rd party TI sensor tags.
4. After the program gets executed, then Silicon Labs BLE is in scanning state and it creates a connection with TI sensor tag. Based on MAX_NUM_OF_SLAVES Silicon Labs device goes to scanning state.
5. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect Silicon Labs device in STA mode.
6. In Windows PC2 which is connected to AP through LAN, Download the Openssl package from above mentioned link and run SSL server by giving following command:
7. **Openssl.exe s_server -accept<SERVER_PORT> –cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>**
   Example: openssl.exe s_server –accept 5001 –cert server-cert.pem -key server-key.pem –tls1

8. After the program gets executed, Silicon Labs BLE is in Scanning state and WLAN connects to Access Point and establishes SSL connectivity with SSL server opened on Windows PC1. Please refer the given below image for connection establishment at windows PC1,



9. After BLE connection is established, Sending start notification command to TI sensor tag. And start sending a notification from the TI sensor tag to Silicon Labs BLE. Observe this notification in the PC connected via SSL socket with Silicon Labs WLAN.

10. Now from SSL server (windows PC1), send a message (Ex: "Hello from WiFi STA") to Silicon Labs device. Silicon Labs device forwards the received message from SSL server to remote BTLE device which is connected to Silicon Labs BTLE device over BTLE protocol. User can observe the message notification on attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID(Ex: 0x1BB1)** in BTLE scanner app.



rsi_ble_app_send_to_wlan() function defined in rsi_wlan_app.c to send message from BLE task to WLAN task.

With the help of wlan task, message is transferred to PC.

8.5 WLAN Station BLE Provisioning

Overview

This application explains how to get the WLAN connection functionality using BLE provisioning.

In this application,

- Silicon Labs Module starts advertising and with BLE Provisioning the Access Point details are fetched
- Silicon Labs device is configured as a WiFi station and connects to an Access Point.

Sequence of Events

WLAN Task

This Application explains user how to:

- Create Silabs device in Station mode
- Connect Silabs station to the remote Access point

BLE Task

This Application explains user how to:

- Configure Silabs device in advertise mode

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART, or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable a variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with Host interface(UART/ USB-CDC/ SPI/ USB)
- RS9116W module
- Wireless Access point
- PC connected to cloud

Block Diagram

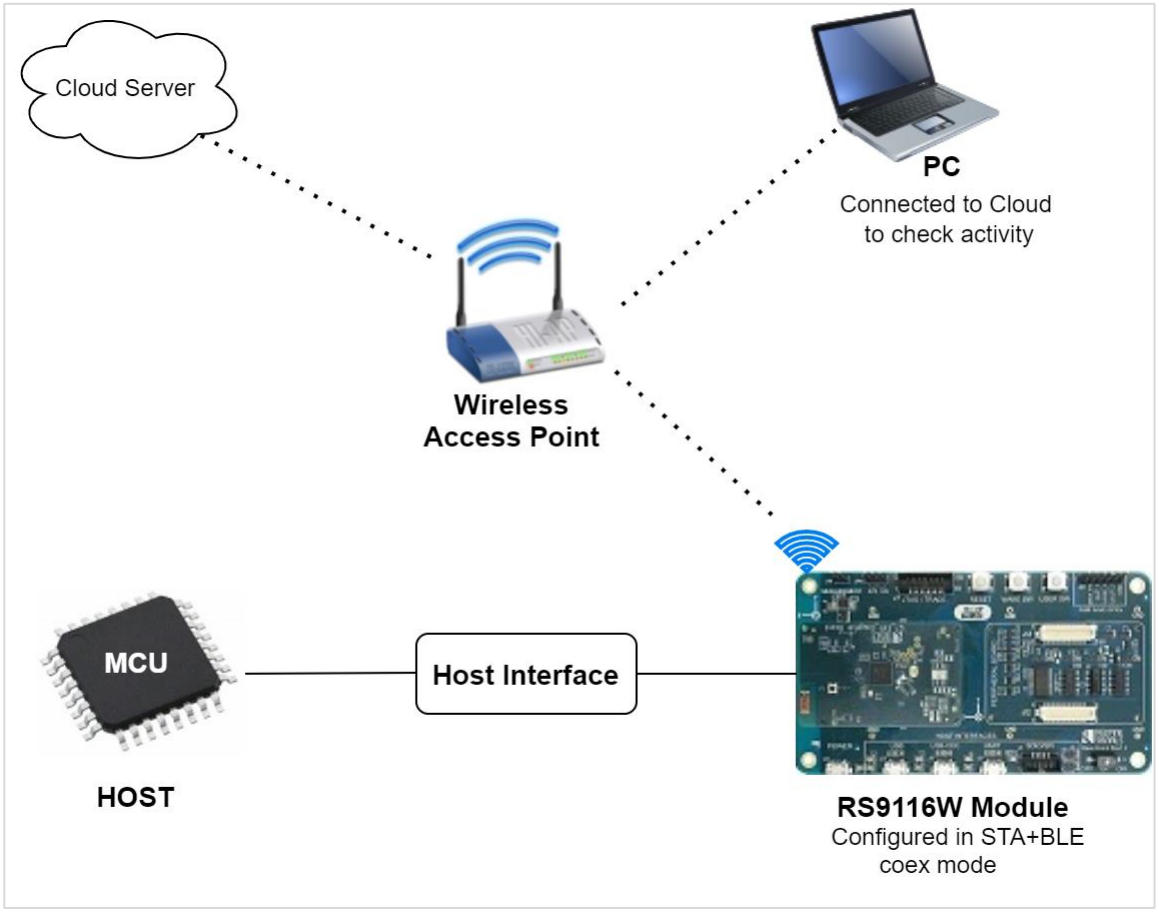


Figure 140: Setup Diagram for WLAN Station BLE Provisioning Application

Configuration and Steps for Execution

Configuring the Application

Open ***rsi_wlan_config.h*** file and update/modify following macros,

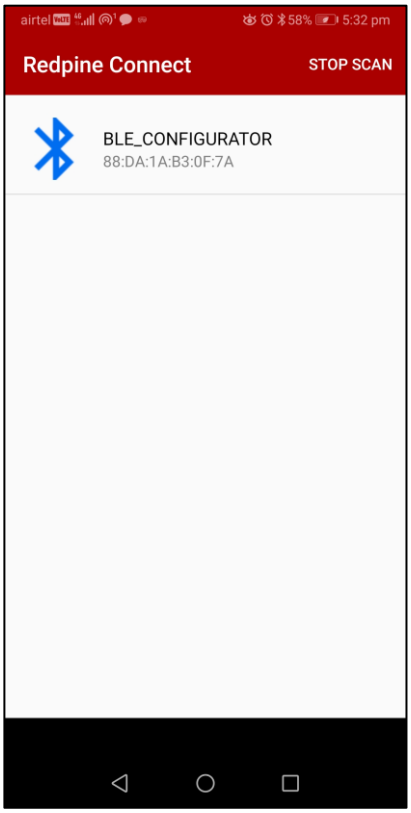
```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     (FEAT_SECURITY_OPEN  | FEAT_AGGREGATION)
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_TOTAL_SOCKETS_1 |
TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID#define
RSI_EXT_CUSTOM_FEATURE_BIT_MAP                  EXT_FEAT_384K_MODE

#define RSI_EXT_TCPIP_FEATURE_BIT_MAP           EXT_DYNAMIC_COEX_MEMORY

#define RSI_BAND                                RSI_BAND_2P4GHZ
```

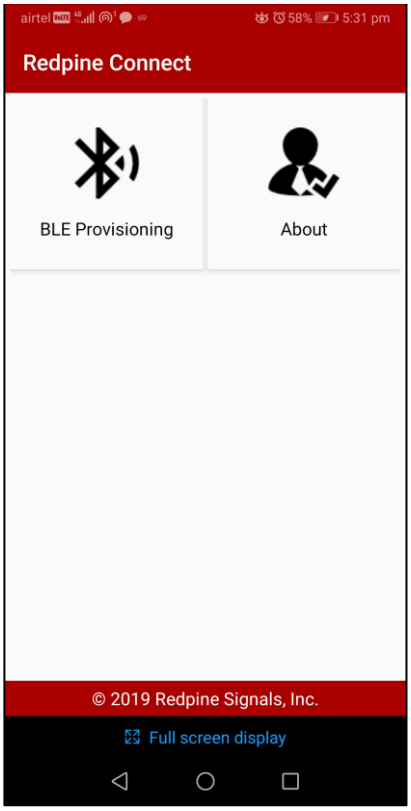
> **Note**:
>
> **rsi_wlan_config.h** file is already set with the desired configuration in respective example folders user need not change for each example.

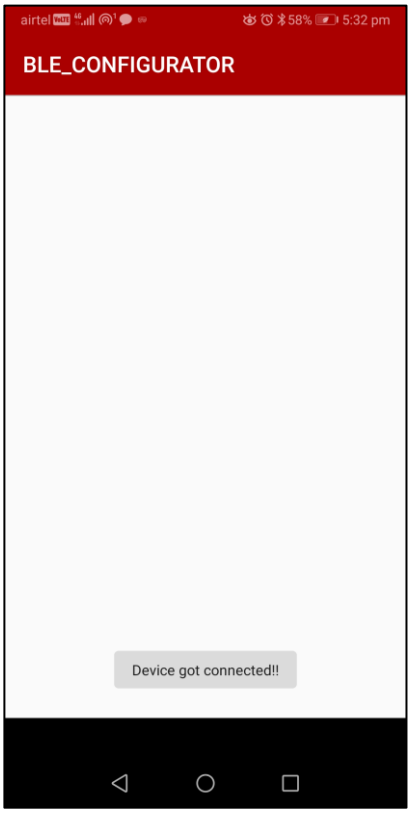
Executing the Application

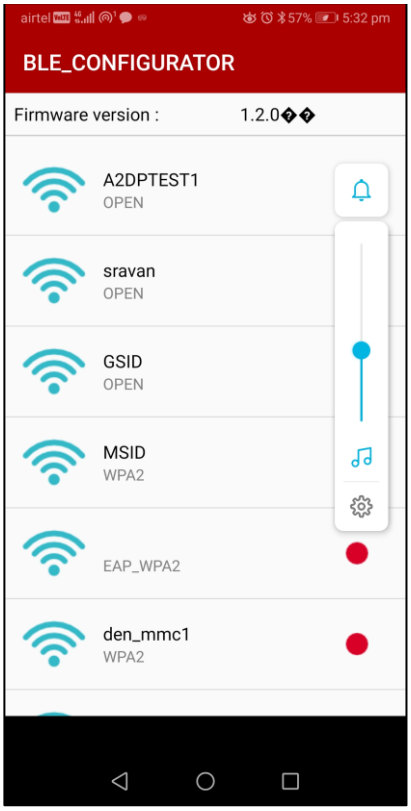
- Configure the Access point with Internet connection in OPEN/WPA-PSK/WPA2-PSK mode to connect the Silicon Labs device in STA mode.
- Connect any serial console for prints.

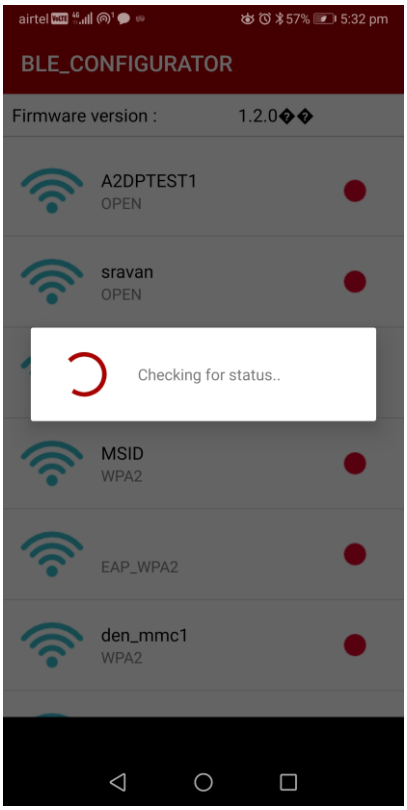
- Redpine Connect app is available in the release utils folder. Currently, this app will be available for Android devices only.
  path for the application: RS9116.NB0.WC.GENR.OSI.x.x.x/utils
- Launch the App Redpine Connect.
- Click on BLE Provisioning.
- Click on BLE_CONFIGURATOR.
- Once the BLE gets the connected, list of available Access Points get displayed on the screen
- Connect to the Access Point, once the device gets connected to AP STM32L4R9 screen show download screen

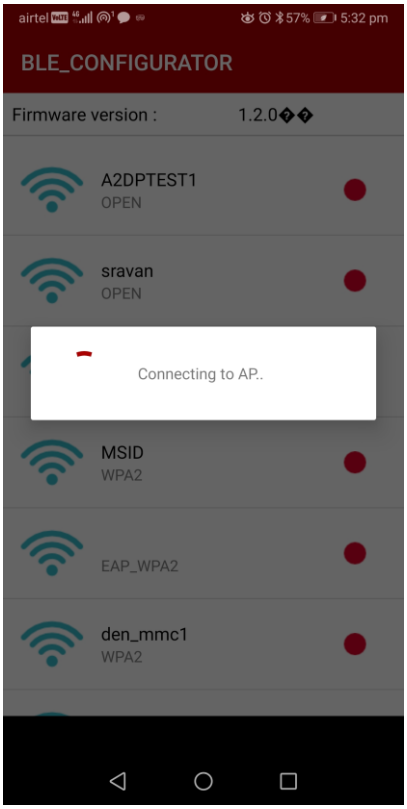


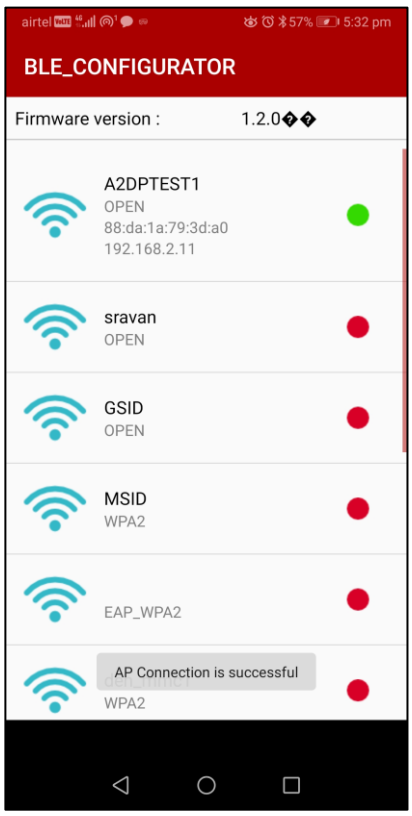

airtel
58%
5:31 pm
Redpine Connect
BLE Provisioning
About
© 2019 Redpine Signals, Inc.
Full screen display


airtel
58%
5:32 pm
BLE_CONFIGURATOR
Device got connected!!


BLE_CONFIGURATOR
Firmware version :
1.2.0
A2DPTEST1
OPEN
sravan
OPEN
GSID
OPEN
MSID
WPA2
EAP_WPA2
den_mmc1
WPA2


BLE_CONFIGURATOR
Firmware version :
1.2.0
A2DPTEST1
OPEN
sravan
OPEN
Checking for status..
MSID
WPA2
EAP_WPA2
den_mmc1
WPA2





8.6 WLAN Station BLE Throughput

Introduction

This example is applicable to WiSeConnect™. The feature(s) used in this example may or may not be available in your part. Refer to the product datasheet to verify the features available in your part.

Overview

The coex application demonstrates throughput measurement of wifi while BLE is in connection.
Sequence of Events

WLAN Task

This application can be used to configure Silicon Labs module in UDP client / server or TCP client / server. To measure throughput, following configuration can be applied.

- To measure UDP Tx throughput, module should be configured as UDP client.
- To measure UDP Rx throughput, module should be configured as UDP server.
- To measure TCP Tx throughput, module should be configured as TCP client.
- To measure TCP Rx throughput, module should be configured as TCP server.

BLE Task

This Application explains user how to:

- Create chat service
- Configure device in advertise mode
- Connect from Smart phone
- Receive data sent by Smart phone

Application Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors. The WiSeMC parts offer integrated wireless connectivity and does not require host interface initialization.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BLE Application (Ex: GATT client application)
- WLAN Access Point and a Windows PC with iperf application

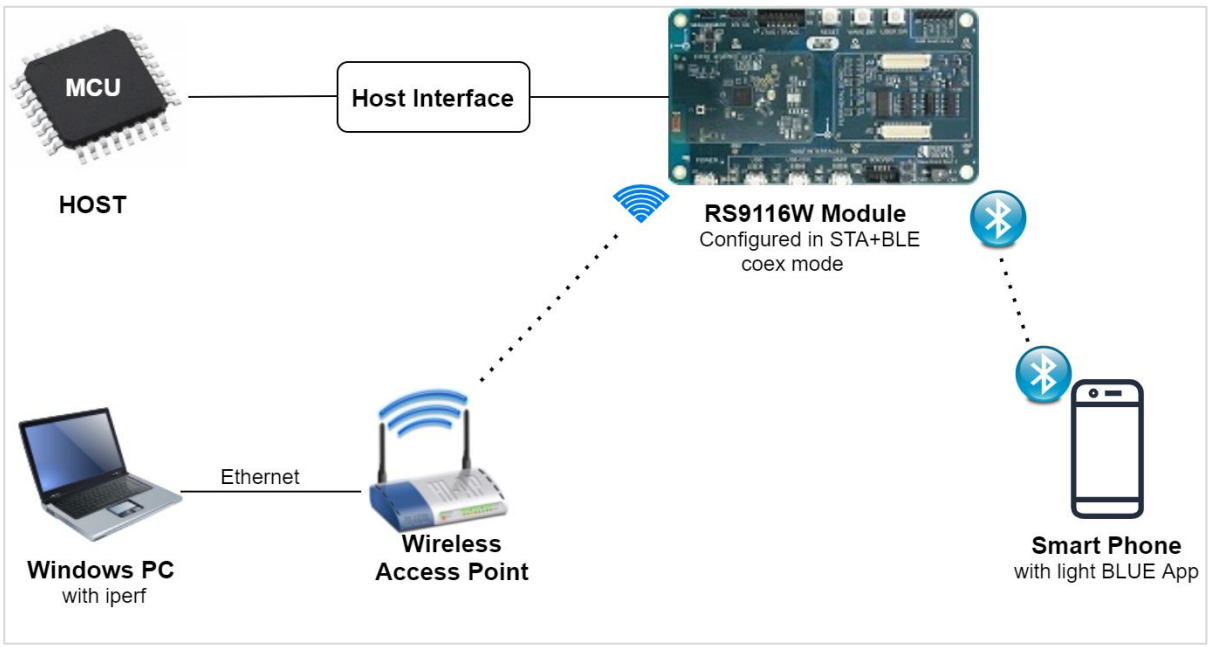


Figure 141: Setup Diagram for WLAN Station BLE Throughput Example

Description

The coex application has WLAN and BLE tasks and acts as an interface between Smartphone and PC. Smartphone interacts with BLE task, while Both PC and Silicon Labs WLAN would be connected to a Wireless Access Point, thus

both are connected together wirelessly interacts with WLAN task. When Smartphone connects and sends message to Silicon Labs device, BT task accepts. Similarly, data transfer will happen for Station between AP.

Details of the Application

Silicon Labs WLAN acts as a Station and connects to an Access Point
Silicon Labs BLE acts as a Slave device.

- The WLAN task (running in Silicon Labs device) mainly includes following steps.

1. Connects to a Access Point
2. Exchanges data over socket with the peer(Windows PC)

- The BLE task (running in Silicon Labs device) mainly includes following steps.

1. Creates chat service
2. Configures the device in advertise mode and connectable mode.

WLAN and BLE tasks forever run in the application to serve the asynchronous events

Configuration and Steps for Execution

Configuring the Application

Configuring the WLAN task

1. Open ***rsi_wlan_app.c*** file and update / modify the following macros,

   SSID refers to the name of the Access point.

```
#define SSID                                    "REDPINE_AP"
```

   CHANNEL_NO refers to the channel in which AP would be started

```
#define CHANNEL_NO                              0
```

   SECURITY_TYPE refers to the type of security .Access point supports Open, WPA, WPA2 securities.
   Valid configuration is:
   RSI_OPEN - For OPEN security mode
   RSI_WPA - For WPA security mode
   RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE                           RSI_OPEN
```

   PSK refers to the secret key if the Access point is to be configured in WPA/WPA2 security modes.

```
#define PSK                                     NULL
```

   Enable / Disable DHCP mode
   1-Enables DHCP mode (gets the IP from DHCP server)
   0-Disables DHCP mode

```
#define DHCP_MODE                               1
```

   To configure static IP address
   IP address to be configured to the device should be in long format and in little endian byte order.
   Example: To configure "192.168.10.1" as IP address, update the macro **DEVICE_IP** as **0x010AA8C0**.

```
#define DEVICE_IP                               0X010AA8C0
```

   IP address of the gateway should also be in long format and in little endian byte order
   Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **0x010AA8C0**

```
#define GATEWAY                                 0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order
Example: To configure "255.255.255.0" as network mask, update the macro **NETMASK** as **0x00FFFFFF**

```
#define NETMASK                                 0x00FFFFFF
```

2. To establish UDP/TCP connection and transfer/receive data to the remote socket configure the below macros

   Port number of the remote server

```
#define PORT_NUM                                5001
```

IP address of the remote server

```
#define SERVER_IP_ADDRESS                       0x640AA8C0
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN                         15000
```

Application can use receive buffer size of 1400

```
#define BUFF_SIZE                               1400
```

Application can select throughput type as UDP Tx, UDP Rx, TCP Tx or TCP Rx. Following is macro need to use.

```
#define THROUGHPUT_TYPE                         UDP_TX
```

Following is macro used for throughput type selection

```
#define UDP_TX                                  0
#define UDP_RX                                  1
#define TCP_TX                                  2
#define TCP_RX                                  3
```

> **Note**:
> In AP mode, configure same IP address for both DEVICE_IP and GATEWAY macros.

3. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                         RSI_DISABLE
#define RSI_FEATURE_BIT_MAP                     (FEAT_SECURITY_OPEN  | FEAT_AGGREGATION)
#define RSI_TCP_IP_BYPASS                       RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP              (TCP_IP_FEAT_DHCPV4_CLIENT |
TCP_IP_TOTAL_SOCKETS_1 | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP              FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP          EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BIT_MAP           EXT_DYNAMIC_COEX_MEMORY
#define RSI_BAND                                RSI_BAND_2P4GHZ
```

Configuring the BLE Application

1. Open ***rsi_ble_app.c*** file and update/modify following macros,

   RSI_BLE_NEW_SERVICE_UUID refers to the attribute value of the newly created service.

```
#define RSI_BLE_NEW_SERVICE_UUID                0xAABB
```

RSI_BLE_ATTRIBUTE_1_UUID refers to the attribute type of the first attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#define RSI_BLE_ATTRIBUTE_1_UUID                    0x1AA1
```

RSI_BLE_ATTRIBUTE_2_UUID refers to the attribute type of the second attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#define RSI_BLE_ATTRIBUTE_2_UUID                    0x1BB1
```

RSI_BLE_MAX_DATA_LEN refers to the Maximum length of the attribute data.

```
#define RSI_BLE_MAX_DATA_LEN                         20
```

RSI_BLE_APP_DEVICE_NAME refers name of the Silicon Labs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_DEVICE_NAME                     "WLAN_BLE_SIMPLE_CHAT"
```

Following are the **non-configurable** macros in the application.
RSI_BLE_CHAR_SERV_UUID refers to the attribute type of the characteristics to be added in a service.

```
#define RSI_BLE_CHAR_SERV_UUID                       0x2803
```

RSI_BLE_CLIENT_CHAR_UUID refers to the attribute type of the client characteristics descriptor to be added in a service.

```
#define RSI_BLE_CLIENT_CHAR _UUID                   0x2902
```

RSI_BLE_ATT_PROPERTY_READ is used to set the READ property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_READ                   0x02
```

RSI_BLE_ATT_PROPERTY_WRITE is used to set the WRITE property to an attribute value.

```
#define RSI_BLE_ATT_PROPERTY_WRITE                  0x08
```

RSI_BLE_ATT_PROPERTY_NOTIFY is used to set the NOTIFY property to an attribute value.

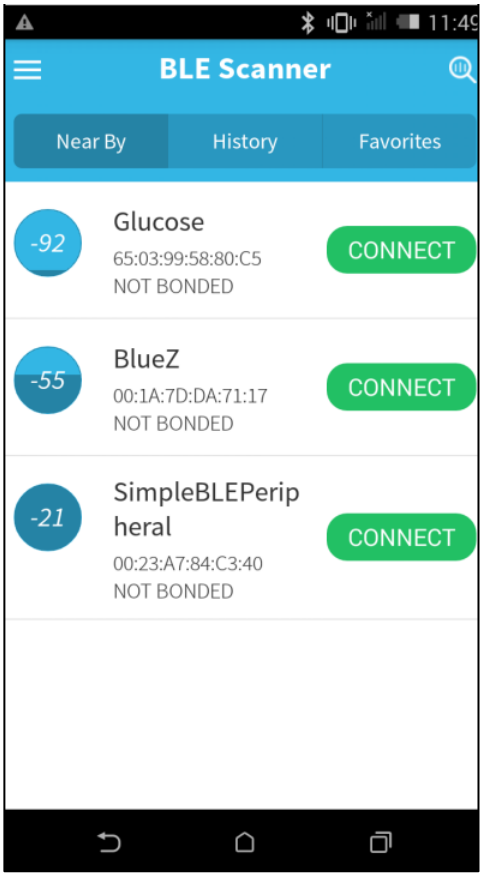
```
#define RSI_BLE_ATT_PROPERTY_NOTIFY                 0x10
```

BT_GLOBAL_BUFF_LEN refers Number of bytes required by the application and the driver.

```
#define BT_GLOBAL_BUFF_LEN                         15000
```

Executing the coex Application

1. Connect WiSeConnect device to the Windows PC running KEIL or IAR IDE
2. Build and launch the application.
3. After the program gets executed, Silicon Labs BLE is in Advertising state and WLAN has established has the configuration given
4. Open a BLE scanner App in the Smartphone and do the Scan.



5. In the App, Silicon Labs module device will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT (Ex: "WLAN_BLE_SIMPLE_CHAT")** or sometimes observed as Silicon Labs device as internal name "**SimpleBLEPeripheral**".
6. Initiate BLE connection from the App.
7. After BT connection is established, send a message from the App to Silicon Labs BLE
8. To measure throughput, following configuration can be applied.
    a. To measure UDP Tx throughput, module should be configured as UDP client. Open UDP server at remote port

       iperf.exe -s -u -p <SERVER_PORT> -i 1

    b. To measure UDP Rx throughput, module should be configured as UDP server. Open UDP client at remote port

       iperf.exe -c <Module_IP> -u -p <Module_Port> -i 1 -b <Bandwidth>

    c. To measure TCP Tx throughput,module should be configured as TCP client. Open TCP server at remote port.

       iperf.exe -s -p <SERVER_PORT> -i 1

    d. To measure TCP Rx throughput, module should be configured as TCP server. Open TCP client at remote port

       iperf.exe -c <Module_IP> -p <module_PORT> -i 1
9. To measure throughput, following configuration can be applied.
10. Build and launch the application.
11. After the program gets executed, the device would be connected to Access point having the configuration same as that of in the application and get.
12. The Device which is configured as UDP / TCP server / client will connect to iperf server / client and sends / receives data continuously. It will print the throughput per second.

Note: If M4 frequency need to Switch higher clock then follow below steps.

step1: Call **switch_m4_frequency()** API after device initialization.

step2: Update systics to higher clock as **SysTick_Config(SystemCoreClock /1000).**

9 WLAN BT BLE

Following is the list of examples described in this section.

Note: Support for compilation and execution in Linux is not supported for above examples. STM32 based Keil projects are provided for these examples with FreeRTOS support.

Note:
FreeRTOS should be downloaded and copied to project folder. Please refer the FreeRTOS porting Guide.

Table 2 Examples List for WLAN BT BLE

S.No	Example	Description	Example Source Path	STM32 Project Path (with FreeRTOS support)
1	WLAN HTTP/HTTPS BT SPP BLE dual role example	The purpose of this example is to demonstrate the ability of RS9116 simultaneous data transfer from all the radios (BT/BLE/WIFI). This application provides user to configure the individual/combined protocols.	RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\examples\wlan_bt_ble\wlan_http_s_bt_spp_ble_dual_role\	RS9116.NB0.WC.GENR.OSI.x.x.x\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI\wlan_https_bt_spp_ble_dual_role\
2	WLAN HTTP/HTTPS BT SPP BLE provisioning example	This example demonstrates WLAN connection using Access point details provided from Redpine BLE provisioning app, along with BT SPP data transfer and BLE data transfer.	RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\examples\wlan_bt_ble\wlan_http_s_bt_spp_ble_provisioning\	RS9116.NB0.WC.GENR.OSI.x.x.x\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI\wlan_https_bt_spp_ble_provisioning\
3	WLAN Throughput BT SPP BLE dual role example	This example demonstrates the throughput measurements of WLAN, while BLE and BT data transfer is in progress. Provides user option to choose individual / combined protocols.	RS9116.NB0.WC.GENR.OSI.x.x.x\host\sapis\examples\wlan_bt_ble\wlan_throughput_bt_spp_ble_dual_role\	RS9116.NB0.WC.GENR.OSI.x.x.x\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI\wlan_throughput_bt_spp_ble_dual_role\

9.1 WLAN HTTP/HTTPS BT SPP BLE Dual Role

Overview

The purpose of this example is to demonstrate the ability of RS9116 simultaneous data transfer from all the radios (BT/BLE/WIFI).

Module will connect to AP and then download the fixed file from PC acting as Server. In parallel to WLAN download, BT/BLE connection/data transfers are supported. Two connections (Master and Slave) are supported with BLE.

This application provides an user to configure the individual/combined protocols.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BT Application (Ex: Bluetooth SPP Pro)
- Smart phone/tablet with BLE Application (Ex: Light Blue APP)
- WiFi client device (PC) with HTTP/HTTPS server running.

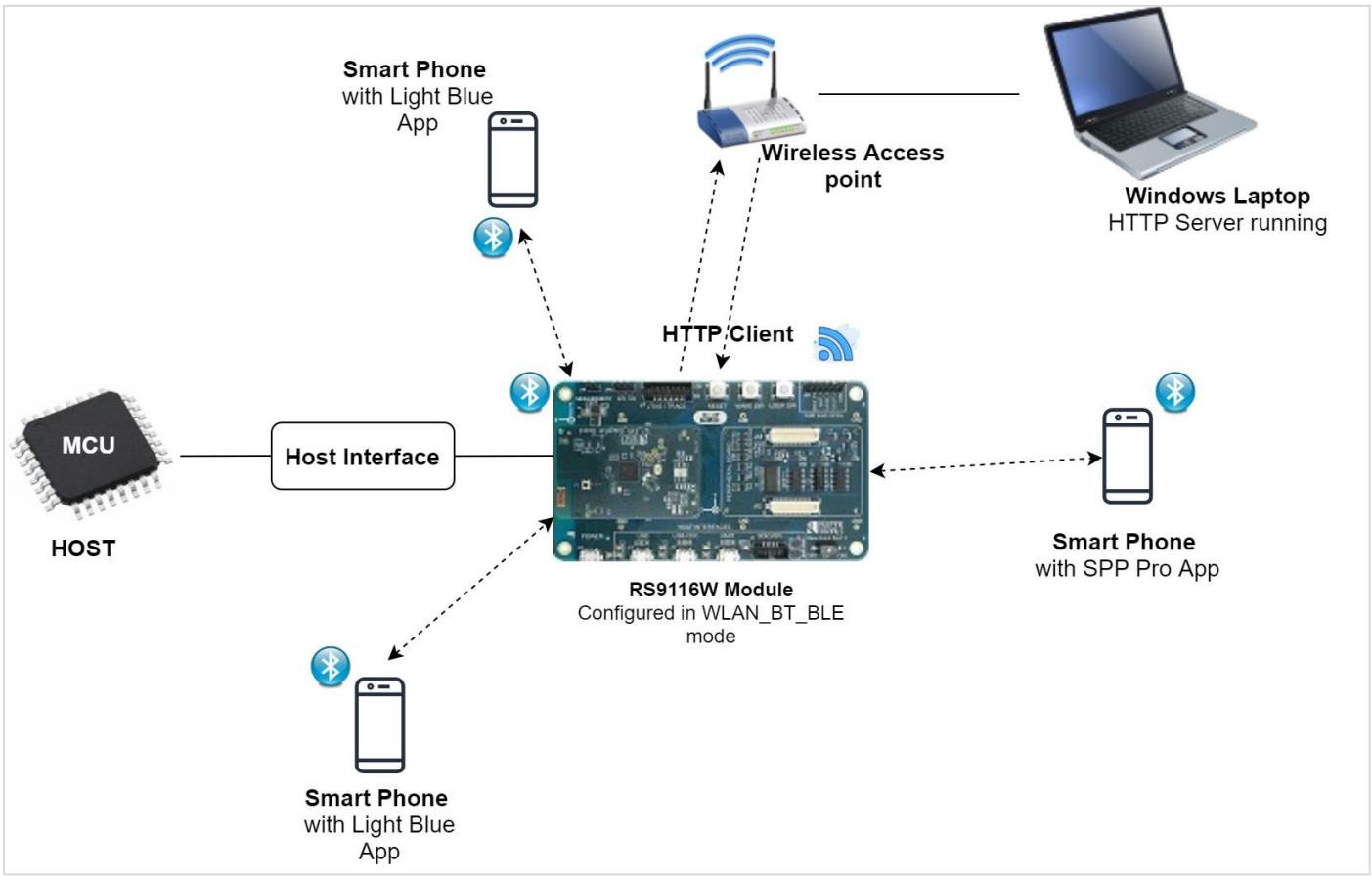


Figure 142: Setup Diagram for WLAN HTTP/HTTPs BT SPP BLE Dual Role Example

Configuration and Steps for Execution

Configuration of Application:

1. Open '**rsi_common_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\ wlan_http_s_bt_spp_ble_dual_role**' and configure below macros,

   set below macro to 1 to run **BLE** application

```
#define RSI_ENABLE_BLE_TEST     1 //Set this to 0 to disable BLE
```

   set below macro to 1 to run **BT** application

```
#define RSI_ENABLE_BT_TEST      1 //Set this to 0 to disable BT
```

   set below macro to 1 to run **WLAN** application

```
#define RSI_ENABLE_WLAN_TEST    1 //Set this to 0 to disable WLAN
```

> **Note:** By default, all protocols are enabled.

choose the required **operational mode** of RS9116W.

```
#define RSI_COEX_MODE        9
```

valid configurations:
0 - WLAN alone mode
5 - BT alone mode
9 - WLAN + BT + BLE mode
13 - BLE alone
mode

> **Note:** By default opermode set to WLAN+BT+BLE

2. open '**rsi_ble_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\ wlan_http_s_bt_spp_ble_dual_role**' and choose **BLE** application configurations.

   To select number of BLE connections, configure below macros.

   Set below macro to required slave connections.

```
#define RSI_BLE_MAX_NBR_SLAVES   1
```

   Set below macro to required master connections.

```
#define RSI_BLE_MAX_NBR_MASTERS  1
```

> **Note**: Maximum no. of RSI_BLE_MAX_NBR_MASTERS can be configured to '2' and RSI_BLE_MAX_NBR_SLAVES to '3'.

   If CONNECT_OPTION is set to CONN_BY_NAME, configure below macros.

```
#define CONNECT_OPTION CONN_BY_NAME //CONN_BY_NAME or CONN_BY_ADDR
```

   To identify remote device with BD Address/device name.

   Add the remote BLE device name to connect

```
#define RSI_REMOTE_DEVICE_NAME1                    "slave1"
#define RSI_REMOTE_DEVICE_NAME2                    "slave2"
#define RSI_REMOTE_DEVICE_NAME3                    "slave3"
```

   If CONNECT_OPTION is set to CONN_BY_ADDR, configure the below macros.

   Configure the address type of remote device as either Public Address or Random Address

```
#define RSI_BLE_DEV_ADDR_TYPE LE_PUBLIC_ADDRESS //!LE_PUBLIC_ADDRESS or LE_RANDOM_ADDRESS
```

   Add the BD Address of remote BLE device to connect

```
#define RSI_BLE_DEV_1_ADDR "88:DA:1A:FE:2A:2C"
#define RSI_BLE_DEV_2_ADDR "7E:E6:5E:30:77:6F"
#define RSI_BLE_DEV_3_ADDR "70:1A:69:32:7C:8E
```

Configure below macros to select the profile characteristics uuid for data transfer.

```
#define RSI_BLE_CLIENT_WRITE_SERVICE_UUID_M1                0x180D //! Heart Rate service uuid
#define RSI_BLE_CLIENT_WRITE_CHAR_UUID_M1                0x2A39 //! Heart Rate control Point
#define RSI_BLE_CLIENT_WRITE_NO_RESP_SERVICE_UUID_M1        0x1802 //! Immediate Alert service uuid
#define RSI_BLE_CLIENT_WRITE_NO_RESP_CHAR_UUID_M1        0x2A06 //! Alert level char uuid
#define RSI_BLE_CLIENT_INIDCATIONS_SERVICE_UUID_M1          0x1809 //! Health thermometer Alert service
uuid
#define RSI_BLE_CLIENT_INIDCATIONS_CHAR_UUID_M1             0x2A1C //! Temperature measurement
#define RSI_BLE_CLIENT_NOTIFICATIONS_SERVICE_UUID_M1     0x180D //! Heart Rate service uuid
#define RSI_BLE_CLIENT_NOTIFICATIONS_CHAR_UUID_M1        0x2A37 //! Heart Rate measurement
```

Configure below macros to select each connection configurations,

Master1 configurations: (where XX=M1)

Set below macro to enable secure connection between Silicon Labs device(peripheral) and remote ble device(central)

```
#define SMP_ENABLE_XX          0
```

//By default this macro is set to '0'

Set below macro to add remote device to whitelist

```
#define ADD_TO_WHITELIST_XX     0
```

//By default this macro is set to '0'

Set below macro to discover remote profiles.

```
#define PROFILE_QUERY_XX     1
```

//By default this macro is set to '1'

Set below macro to enable data transfer between devices

```
#define DATA_TRANSFER_XX     1
```

//By default this macro is set to '1'

To select the type of data transfer configure below macros

Set below macro to receive 'gatt notifications' from remote device.

```
#define RX_NOTIFICATIONS_FROM_XX     0
```

//By default this macro is set to '1'

Note:
Make sure to set below macros to 0

#define RX_INDICATIONS_FROM_XX 0 //Set this to 0

Set below macro to receive 'gatt indications' from remote device.

```
#define RX_INDICATIONS_FROM_XX     0
```

//By default this macro is set to '0'

Set below macro to Transmit 'gatt notifications' to remote device.

```
#define TX_NOTIFICATIONS_TO_XX     1
```

//By default this macro is set to '1'

Note:
Make sure to set below macros to 0

#define TX_WRITES_TO_XX 0 //Set this to 0
#define TX_WRITES_NO_RESP_TO_XX 0 //Set this to 0
#define TX_INDICATIONS_TO_XX 0 //Set this to 0

Set below macro to Transmit 'gatt write with response' to remote device.

```
#define TX_WRITES_TO_XX     0
```

//By default this macro is set to '0'

Set below macro to Transmit 'gatt write without response' to remote device.

```
#define TX_WRITES_NO_RESP_TO_XX     0
```

//By default this macro is set to '0'

Set below macro to Transmit 'gatt indications to remote device.

```
#define TX_INDICATIONS_TO_XX      0
```

//By default this macro is set to '0'

To select data length extension for each connection configure below macro

Set below macro to enable data length extension

```
#define DLE_ON_XX           0
```

//By default this macro is set to '0'

Configure below macros to set connection interval, connection latency and connection supervision timeout

Below configuration is for connection interval of 45ms, latency 0 and timeout:400ms

```
#define CONN_INTERVAL_XX        36
#define CONN_LATENCY_XX           0
#define CONN_SUPERVISION_TIMEOUT_XX  400
```

Note: Follow the above instructions to configure for remaining connections (slave1(XX = S1),slave2 (XX =S2),slave3(XX=S3) and master2(XX=M2))

3. Select BT configurations in '**rsi_bt_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\ wlan_http_s_bt_spp_ble_dual_role**'

   Enter the remote BT device address as the value to RSI_BT_REMOTE_BD_ADDR

```
#define     RSI_BT_REMOTE_BD_ADDR     (void *)"B8:D5:0B:9B:D6:B2"
```

   SPP_MODE refers to type of Module Mode, whether its MASTER/SLAVE

```
#define     SPP_MODE                SPP_SLAVE
```

   PIN_CODE refers 4 bytes string required for pairing process

```
#define  PIN_CODE         "0000"
```

   RSI_BT_LOCAL_NAME refers to name of Silicon Labs Module to appear during scanning by remote device

```
#define  RSI_BT_LOCAL_NAME    "SPP_SLAVE"
```

4. Select WLAN configurations in '**rsi_wlan_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\ wlan_http_s_bt_spp_ble_dual_role**'

   Enter the AP Connectivity essentials configs as the value to SSID, SECURITY_TYPE and PSK

```
#define     SSID                        "Hotspot"
#define     SECURITY_TYPE               RSI_WPA2 //RSI_OPEN
#define     PSK                         "12345678"
```

   To select the ip getting configure below macros

```
#define     DHCP_MODE               1 //0 enable or disable
#if !DHCP_MODE // Need to configure manually if dhcp disabled
#define   DEVICE_IP          0x6500A8C0  //192.168.0.101
#define   GATEWAY            0x0100A8C0  //192.168.0.1
#define   NETMASK            0x00FFFFFF  //255.255.255.0
#endif
```

   configure below macros to make Use of Local HTTP server to download the files.

```
#define  RSI_DNS_CLIENT   0              // set to '1' only if using server name instead of server ip
address, by default it is set to '0'
#define     RX_DATA         1              // set to '1' to RX data from remote server
#define    HTTPS_DOWNLOAD   0              // set to '0' to choose HTTP download
#define    SERVER_PORT      80           // by default http runs on port 80
#define     SERVER_IP_ADDRESS "192.168.0.101"    //Local server ip address
#define    DOWNLOAD_FILENAME"dltestdata32.txt"  // File to download, by default this file is provided
in the demo
#define    BYTES_TO_RECEIVE 1048576          // size of file configured under 'DOWNLOAD_FILENAME'
#define  CONTINUOUS_HTTP_DOWNLOAD  1       // set to '1' to download continuously, if reset download
happens only once.
```

   configure below macros to make Use of Local HTTPS server to download the files.

```
#define  RSI_DNS_CLIENT   0              // set to '1' only if using server name instead of server ip
address, by default it is set to '0'
#define     RX_DATA         1              // set to '1' to RX data from remote server
#define    HTTPS_DOWNLOAD   1              // set to '1' to choose HTTPs download
#define    SERVER_PORT      443          // by default https runs on port 443
#define     SERVER_IP_ADDRESS "192.168.0.101"    //Local server ip address
#define    DOWNLOAD_FILENAME"dltest.txt"  // File to download, by default this file is provided in the
demo
#define    BYTES_TO_RECEIVE 6144          // size of file configured under 'DOWNLOAD_FILENAME'
```

```
#define  CONTINUOUS_HTTP_DOWNLOAD  1        // set to '1' to download continuously, if reset download
happens only once.
```

Note:

BY default, when 'HTTPS_DOWNLOAD' is set, SSL and LOAD_CERTIFICATE will be set to '1' as it is required for HTTPS download

Follow below steps to configure local https server

1. Download and install SSL server from https://slproweb.com/products/Win32OpenSSL.html
2. Add the installed location (ex: "C:\Program Files\OpenSSL-Win64\bin") in environment variable 'PATH' and restart the pc to reflect the changes.

To select re-join feature, add below macros in wlan_config.h

```
//! RSI_ENABLE or RSI_DISABLE rejoin params
#define RSI_REJOIN_PARAMS_SUPPORT       RSI_ENABLE

//! Rejoin retry count. If 0 retries infinity times
#define RSI_REJOIN_MAX_RETRY           10

//! Periodicity of rejoin attempt
#define RSI_REJOIN_SCAN_INTERVAL       4

//! Beacon missed count
#define RSI_REJOIN_BEACON_MISSED_COUNT   40

//! RSI_ENABLE or RSI_DISABLE retry for first time join failure
#define RSI_REJOIN_FIRST_TIME_RETRY    RSI_DISABLE
```

Wlan register callback:

Regarding register a callback to handle join failure condition.

```
void rsi_wlan_app_callbacks_init(void)
{
   rsi_wlan_register_callbacks(RSI_JOIN_FAIL_CB, rsi_join_fail_handler);  //Initialize join fail call
back
}
```

Re-join call back handler:

The callback should update the system STATE as the error code received in the callback. Configuring the state for rejoin in the **rsi_join_fail_handler()** in rsi_wlan_http_s.c file (at 'RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_http_s_bt_spp_ble_dual_role\') is shown below,

```
void rsi_join_fail_handler(uint16_t status, uint8_t *buffer, const uint32_t length)
{
```

```
rsi_wlan_app_cb.state = RSI_WLAN_JOIN_STATE;//! update wlan Application state
}
```

Executing the Application

1. Compile the project and flash the binary onto STM32.
2. Copy the files 'dltestdata32.txt', 'dltest.txt' from below source path and paste in to the destination path.

   [source path:- ../host/sapis/examples/wlan_bt_ble/wlan_http_s_bt_spp_ble_dual_role/]

   [destination path:- ../host/sapis/examples/utilities/scripts/]
3. To download the files from local http server, navigate to below folder and run below command.

   [File path:- ../host/sapis/examples/utilities/scripts/]

   #python simple_http_server.py 80
4. To download the files from local https server, copy ssl certificates 'server-cert.pem' , 'server-key.pem' from below 'source path' and paste in to 'destination path'.

   [source path:- ../host/sapis/examples/utilities/certificates/]

   [destination path:- ../host/sapis/examples/utilities/scripts/]

   open command prompt, navigate to above destination path and run below command.

   #openssl s_server -accept 443 -cert server-cert.pem -key server-key.pem -tls1 -WWW
5. After the program gets executed, Module scans for the configured Accesspoint, connects to it and acquires the ip address
6. After acquiring ip address, initiates connection to remote server.(ex: simple_http_server.py running in same network where Module is also connected)
7. If connection is successful,

   a. Module starts advertising and scanning BLE
   b. Advertises BT and simultaneoulsy downloads http packets sent from remote server
8. If connection is not successful, step5 is repeated untill connection is success
9. While downloading is happening, user can initiate both BT SPP and BLE connections (both peripheral and central).
10. To check BLE peripheral connection, scan and initiate connection from nRF connect/dongles.
11. Module accepts the BLE connections if initiated by remote BLE device(max 2 master connections are accepted) and starts data transfer based on the user configuration.
12. To check data transfer, enable Gatt notifications of Module on service characteristic RSI_BLE_ATTRIBUTE_1_UUID,
13. If enabled module continuously transmits 20 notifications per connection interval of size 20bytes.
14. To check BLE central connection, advertise the remote ble devices using phone/dongles.

15. Module scans for advertised devices, crosschecks the ble device names/ble device address as configured in application, if matches initiate connection.
16. If BLE connection is successful, Module enables the Gatt notifications of remote device for RSI_BLE_CLIENT_NOTIFICATIONS_CHAR_UUID_M1 (Heart Rate measurement) and receives notifications/connection interval.

> **Note**: Steps 9 to 12 can be repeated for 2 peripheral connection and steps 13 to 15 can be repeated for 3 central connections based on the RSI_BLE_MAX_NBR_MASTERS and RSI_BLE_MAX_NBR_SLAVES.

17. while BLE and WLAN data transfer is happening, initiate BT SPP connection using "BT SPP manager app"
18. After successful BT connection, Module echos the data transmitted from BT SPP manager app.

> **Note**: Verify that all connections are stable and simultaneous data transfer is happening from all the radios of Module

9.2 WLAN HTTP/HTTPS BT SPP BLE Provisioning

Overview

This example demonstrates wlan connection using Access point details provided from Redpine BLE provisioning app, along with BT SPP data transfer and BLE data transfer.

Two BLE connections are supported, in which first connection is for provisioning and 2 connection is for data transfer.

Sequence of Events

WLAN task: Fetches the accesspoint details from Redpine BLE provisioning app, connects to remote server and starts http download

BLE task:

- Advertises the module and accepts the connection from Redpine BLE provisioning app (Android)

App location: "RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\utils\Redpine_Connect_v1.1.apk"

- Accepts the connection from Redpine BLE APP and sends the Accesspoint scan results to BLE APP using wlan task
- Sends the AP details selected in BLE APP to wlan task and sends the connection acknowledgement to it
- Accepts new connection if module gets connection request from remote BLE device
- Initiates connection request if module scans the configured BLE devices

BT task:

- Initializes BT SPP after wlan connection to remote server.
- Accepts the connection from BT device (BT SPP Manager APP) and retransmits the data sent by Manager APP.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BT Application (Ex: Bluetooth SPP Pro)
- Smart phone/tablet with BLE Application (Ex: Light Blue APP)
- WiFi client device (PC) with HTTP/HTTPS server.

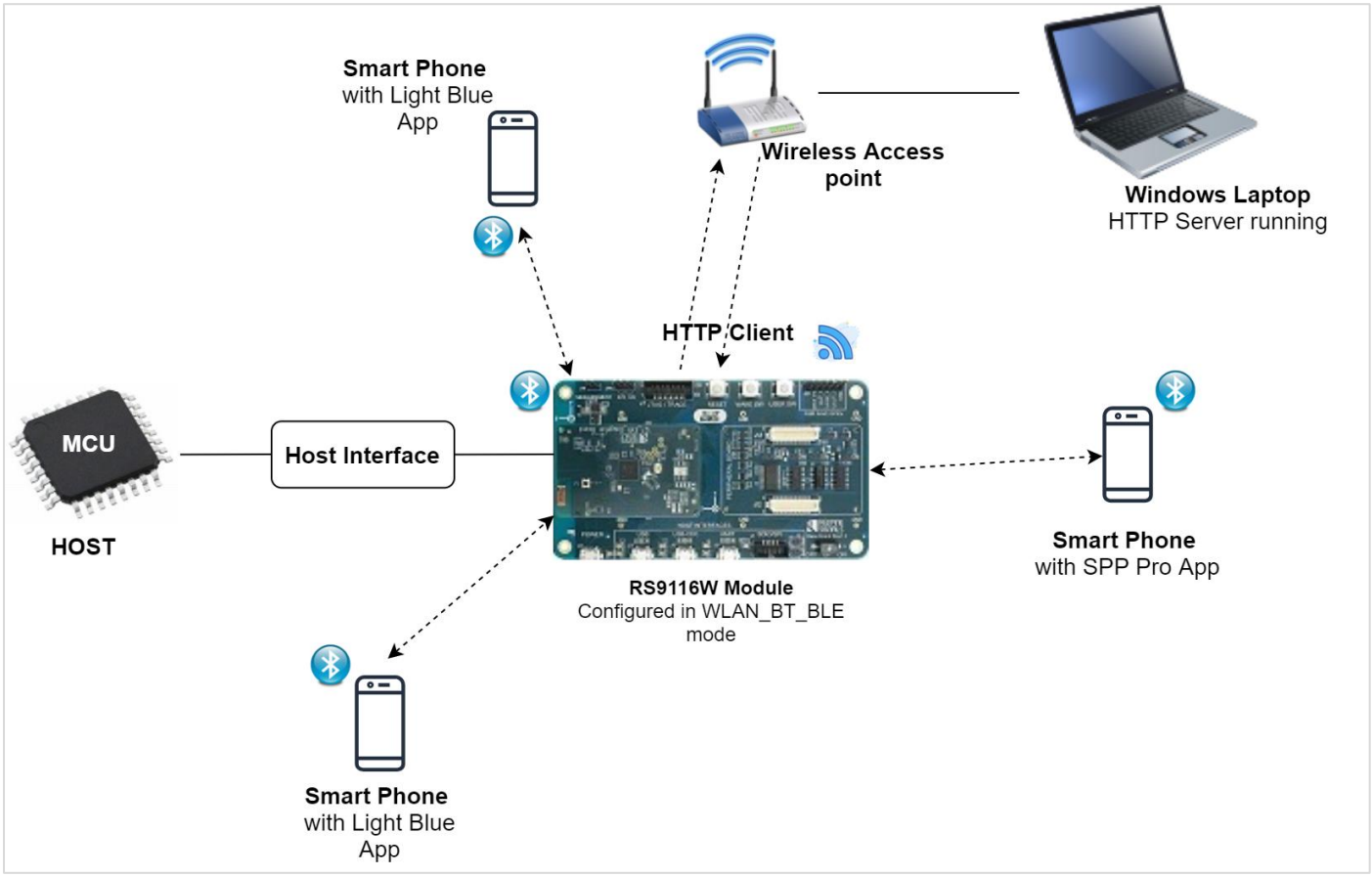


Figure 143: Setup Diagram for WLAN HTTP/HTTPs BT SPP BLE Provisioning Example

Configuration and Steps for Execution

Configuration of Application:

1. Open '**rsi_common_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_http_s_bt_spp_ble_provisioning**' and configure below macros.

   set below macro to 1 to run **BT** application along with WLAN and BLE

```
#define RSI_ENABLE_BT_TEST      1 //Set this to 0 to disable BT
```

> **Note:** By default, all protocols are enabled. It is mandatory to enable WLAN and BLE.

   choose the required **operational mode** of RS9116

```
#define RSI_COEX_MODE        9
```

   valid configurations:
   0 - WLAN alone mode
   5 - BT alone mode
   9 - WLAN + BT + BLE mode
   13 - BLE alone
   mode

> **Note:** By default, opermode is set to WLAN+BT+BLE

2. open **'rsi_ble_config.h'** file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_http_s_bt_spp_ble_provisioning**' and choose **BLE** application configurations
   To select number of BLE connections, configure below macros
   Set below macro to required slave connections

```
#define RSI_BLE_MAX_NBR_SLAVES   1
```

   Set below macro to required master connections

```
#define RSI_BLE_MAX_NBR_MASTERS  1
```

> **Note:**
>
> 1. Maximum no. of RSI_BLE_MAX_NBR_MASTERS can be configured to '2' and RSI_BLE_MAX_NBR_SLAVES to '3'
> 2. To run BLE provisioning application, always ensure min value of RSI_BLE_MAX_NBR_MASTERS is set to '1'

   Configure Module BLE advertise name

```
#define RSI_BLE_APP_GATT_TEST     (void *)"SI_COEX_MAX_DEMO"
```

   To identify remote device with BD Address/device name.

```
#define CONNECT_OPTION CONN_BY_NAME //CONN_BY_NAME or CONN_BY_ADDR
```

   If CONNECT_OPTION is set to CONN_BY_NAME, configure below macros.

   Add the remote BLE device name to connect

```
#define RSI_REMOTE_DEVICE_NAME1                    "slave1"
#define RSI_REMOTE_DEVICE_NAME2                    "slave2"
#define RSI_REMOTE_DEVICE_NAME3                    "slave3"
```

   If CONNECT_OPTION is set to CONN_BY_ADDR, configure the below macros.

   Configure the address type of remote device as either Public Address or Random Address

```
#define RSI_BLE_DEV_ADDR_TYPE LE_PUBLIC_ADDRESS //!LE_PUBLIC_ADDRESS or
```

   LE_RANDOM_ADDRESS

   Add the BD Address of remote BLE device to connect

```
#define RSI_BLE_DEV_1_ADDR "88:DA:1A:FE:2A:2C"
#define RSI_BLE_DEV_2_ADDR "7E:E6:5E:30:77:6F"
#define RSI_BLE_DEV_3_ADDR "70:1A:69:32:7C:8E
```

   Configure below macros to select the profile characteristics uuid for data transfer.

```
#define RSI_BLE_CLIENT_WRITE_SERVICE_UUID_M1              0x180D //! Heart Rate service uuid
#define RSI_BLE_CLIENT_WRITE_CHAR_UUID_M1               0x2A39 //! Heart Rate control Point
#define RSI_BLE_CLIENT_WRITE_NO_RESP_SERVICE_UUID_M1      0x1802 //! Immediate Alert service uuid
```

```
#define RSI_BLE_CLIENT_WRITE_NO_RESP_CHAR_UUID_M1         0x2A06 //! Alert level char uuid
#define RSI_BLE_CLIENT_INIDCATIONS_SERVICE_UUID_M1          0x1809 //! Health thermometer Alert service
uuid
#define RSI_BLE_CLIENT_INIDCATIONS_CHAR_UUID_M1             0x2A1C //! Temperature measurement
#define RSI_BLE_CLIENT_NOTIFICATIONS_SERVICE_UUID_M1      0x180D //! Heart Rate service uuid
#define RSI_BLE_CLIENT_NOTIFICATIONS_CHAR_UUID_M1         0x2A37 //! Heart Rate measurement
```

Configure below macros to select each connection configurations except for Master1 , as Master1 is configured by default to match with Redpine BLE provisioning app.

Master2 configurations: (where XX=M2)

Set below macro to enable secure connection between Silicon Labs device(peripheral) and remote ble device(central)

```
#define SMP_ENABLE_XX           0
```

//By default this macro is set to '0'

Set below macro to add remote device to whitelist

```
#define ADD_TO_WHITELIST_XX     0
```

//By default this macro is set to '0'

Set below macro to discover remote profiles.

```
#define PROFILE_QUERY_XX     1
```

//By default this macro is set to '1'

Set below macro to enable data transfer between devices

```
#define DATA_TRANSFER_XX     1
```

//By default this macro is set to '1'

To select the type of data transfer configure below macros

Set below macro to receive 'gatt notifications' from remote device

```
#define RX_NOTIFICATIONS_FROM_XX     0
```

//By default this macro is set to '0'

> **Note:**
> Make sure to set below macros to 0
>
> #define RX_INDICATIONS_FROM_XX 0 //Set this to 0

Set below macro to receive 'gatt indications' from remote device

```
#define RX_INDICATIONS_FROM_XX     0
```

//By default, this macro is set to '0'

Note:
Make sure to set below macros to 1

#define TX_NOTIFICATIONS_FROM_XX 1//Set this to 1

Set below macro to Transmit 'gatt notifications' to remote device

```
#define TX_NOTIFICATIONS_TO_XX     1
```

//By default, this macro is set to '1'

Note:
Make sure to set below macros to 0

#define TX_WRITES_TO_XX 0 //Set this to 0
#define TX_WRITES_NO_RESP_TO_XX 0 //Set this to 0
#define TX_INDICATIONS_TO_XX 0 //Set this to 0

Set below macro to Transmit 'gatt write with response' to remote device

```
#define TX_WRITES_TO_XX     0
```

//By default this macro is set to '0'

Set below macro to Transmit 'gatt write without response' to remote device

```
#define TX_WRITES_NO_RESP_TO_XX     0
```

//By default this macro is set to '0'

Set below macro to Transmit 'gatt indications to remote device

```
#define TX_INDICATIONS_TO_XX      0
```

//By default this macro is set to '0'

To select data length extension for each connection configure below macro

Set below macro to enable data length extension

```
#define DLE_ON_XX           0
```

//By default this macro is set to '0'

Configure below macros to set connection interval, connection latency and connection supervision timeout

Below configuration is for connection interval of 45ms, latency 0 and timeout:400ms

```
#define CONN_INTERVAL_XX        400
#define CONN_LATENCY_XX           0
#define CONN_SUPERVISION_TIMEOUT_XX  400
```

Note:
Follow the above instructions to configure for remaining connections (slave1(XX = S1), slave2 (XX =S2), slave3(XX=S3)

3. Select BT configurations in '**rsi_bt_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_http_s_bt_spp_ble_provisioning**'

   Enter the remote BT device address as the value to RSI_BT_REMOTE_BD_ADDR

```
#define     RSI_BT_REMOTE_BD_ADDR     (void *)"B8:D5:0B:9B:D6:B2"
```

   SPP_MODE refers to type of Module Mode, whether its MASTER/SLAVE

```
#define     SPP_MODE                SPP_SLAVE
```

   PIN_CODE refers 4 bytes string required for pairing process

```
#define  PIN_CODE          "0000"
```

   RSI_BT_LOCAL_NAME refers to name of RS9116W Module to appear during scanning by remote device

```
#define  RSI_BT_LOCAL_NAME    "SPP_SLAVE"
```

4. Select WLAN configurations in '**rsi_wlan_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_http_s_bt_spp_ble_provisioning**'

   configure below macros to make Use of Local HTTP server to download the files.

```
#define  RSI_DNS_CLIENT   0              // set to '1' only if using server name instead of server ip
address, by default it is set to '0'
#define     RX_DATA         1              // set to '1' to RX data from remote server
#define    HTTPS_DOWNLOAD   0              // set to '0' to choose HTTP download
#define     SERVER_PORT       80                    //! Server port number
#define     SERVER_IP_ADDRESS "192.168.0.10"  //Local server ip address
#define    DOWNLOAD_FILENAME"dltestdata32.txt"  // File to download, by default this file is provided
in the demo
#define    BYTES_TO_RECEIVE 1048576          // size of file configured under 'DOWNLOAD_FILENAME'
#define  CONTINUOUS_HTTP_DOWNLOAD  1       // set to '1' to download continuously, if reset download
happens only once.
```

   configure below macros to make Use of Local HTTPS server to download the files.

```
#define  RSI_DNS_CLIENT   0              // set to '1' only if using server name instead of server ip
address, by default it is set to '0'
#define     RX_DATA         1              // set to '1' to RX data from remote server
#define    HTTPS_DOWNLOAD   1              // set to '1' to choose HTTPs download
#define     SERVER_PORT       443                   //! Server port number
#define     SERVER_IP_ADDRESS "192.168.0.10"  //Local server ip address
#define    DOWNLOAD_FILENAME"dltest.txt"  // File to download, by default this file is provided in the
demo
#define    BYTES_TO_RECEIVE 6144          // size of file configured under 'DOWNLOAD_FILENAME'
#define  CONTINUOUS_HTTP_DOWNLOAD  1       // set to '1' to download continuously, if reset download
happens only once.
```

Note:

BY default, when 'HTTPS_DOWNLOAD' is set, SSL and LOAD_CERTIFICATE will be set to '1' as it is required for HTTPS download.

Follow below steps to configure local https server

1. Download and install SSL server from https://slproweb.com/products/Win32OpenSSL.html

2. Add the installed location (ex: "C:\Program Files\OpenSSL-Win64\bin") in environment variable 'PATH' and restart the pc to reflect the changes.

Executing the Application

1. Compile the project and flash the binary onto STM32
2. Copy the files 'dltestdata32.txt', 'dltest.txt' from below source path and paste in to the destination path.

   [source path:- ../host/sapis/examples/wlan_bt_ble/wlan_http_s_bt_spp_ble_provisioning/]

   [destination path:- ../host/sapis/examples/utilities/scripts/]
3. To download the files from local http server, navigate to below folder and run below command.

   [File path:- ../host/sapis/examples/utilities/scripts/]

   #python simple_http_server.py 80
4. To download the files from local https server, copy ssl certificates 'server-cert.pem' , 'server-key.pem' from below 'source path' and paste in to 'destination path'.

   [source path:- ../host/sapis/examples/utilities/certificates/]

   [destination path:- ../host/sapis/examples/utilities/scripts/]

   open command prompt navigate to above destination path and run below command.

   #openssl s_server -accept 443 -cert server-cert.pem -key server-key.pem -tls1 -WWW
5. Below steps are based on the default configurations provided in the application.
6. Module advertises and waits for connection from remote device.
7. Open "Redpine connect" application from mobile and scan for ble advertisement packets.
8. Initiate connection if packet name matches with the name configured in "RSI_BLE_APP_GATT_TEST".
9. After connection, list of available access points are displayed on the screen.
10. Connect to required access point and provide PSK if prompted.
11. If credentials are valid, Module connects to that access point, notifies to Redpine APP as "AP connection successful" and starts http/https download.
12. While downloading is happening, user can initiate both BT SPP and BLE second connection (either peripheral/central).
13. While WLAN data transfer is happening, initiate BT SPP connection using "BT SPP manager app"
14. After successful BT connection, Module echos the data transmitted from BT SPP manager app.
15. To check BLE peripheral connection, scan and initiate connection from nRF connect/dongles.
16. Module accepts the BLE connections if initiated by remote BLE device(max 2 master connections are accepted) and starts data transfer based on the user configuration.

17. To check data transfer, enable Gatt notifications of Module on service characteristic RSI_BLE_ATTRIBUTE_1_UUID (0x1AA1),
18. If enabled module continuously transmits 20 notifications per connection interval of size 20bytes.
19. To check BLE central connection, advertise the remote ble devices using phone/dongles.
20. Module scans for advertised devices, crosschecks the ble device names/ble device address as configured in application, if matches initiate connection.
21. If BLE connection is successful, Module enables the Gatt notifications of remote device for RSI_BLE_CLIENT_NOTIFICATIONS_CHAR_UUID_M1 (Heart Rate measurement) and receives notifications/connection interval.

Note: Steps 13 to 16 can be repeated for 2 peripheral connection and steps 17 to 19 can be repeated for 3 central connections based on the RSI_BLE_MAX_NBR_MASTERS and RSI_BLE_MAX_NBR_SLAVES.

Note: Verify that all connections are stable and simultaneous data transfer is happening from all the radios of Module

9.3 WLAN Throughput BT SPP BLE Dual Role

Overview

This example demonstrates the throughput measurements of wlan along with BLE (master)/BT (SPP) connections and also provides user options to choose individual/combined protocols while measuring throughput of Wlan.

Sequence of Events:

WLAN Task:

This application can be used to configure Silicon Labs module in UDP client / server or TCP client / server or SSL client/server.

To measure throughput, following configuration can be applied.

To measure SSL Tx throughput, module should configured as SSL client.

To measure SSL Rx throughput, module should configured as SSL server.

To measure UDP Tx throughput, module should configured as UDP client.

To measure UDP Rx throughput, module should configured as UDP server.

To measure TCP Tx throughput, module should configured as TCP client.

To measure TCP Rx throughput, module should configured as TCP server.

BLE task:

This application can be used to configure module in scanning and advertising modes.

Manages the connections and datatransfer between remote ble devices and module.

BT task:

This application can be configure module in slave or master modes.

In slave mode, accepts the connection from remote device and retransmits the data sent by remote device.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module
- Smart phone/tablet with BT Application (Ex: Bluetooth SPP Pro)

- Smart phone/tablet with BLE Application (Ex: Light Blue APP)
- WiFi client device (PC) with UDP client application.

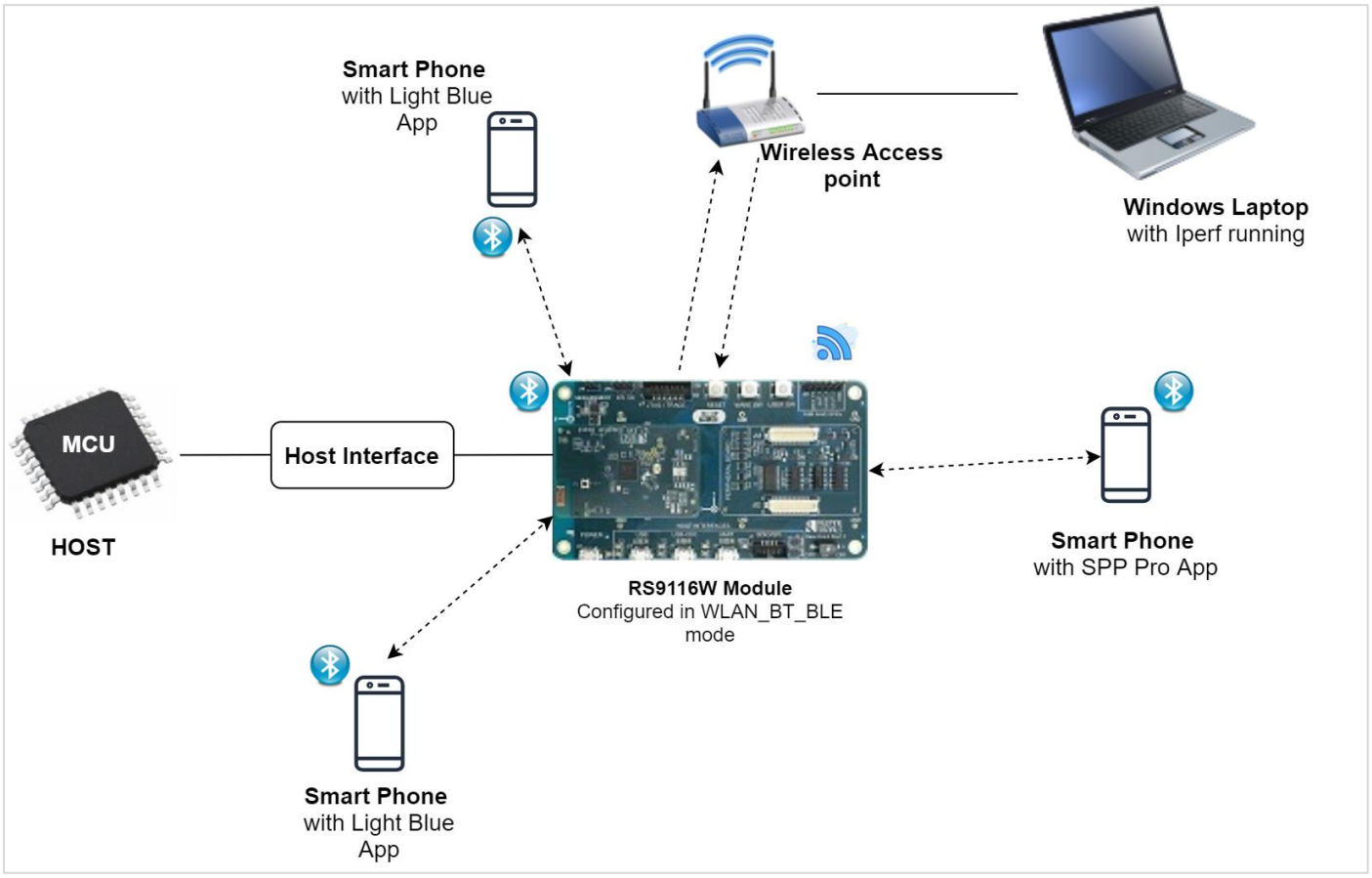


Figure 144: Setup Diagram for wlan Throughput BT SPP BLE Dual Role Example

Configuration and Steps for Execution

Configuration of Application:

1. Open '**rsi_common_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_throughput_bt_spp_ble_dual_role**' and configure below macros,

   set below macro to 1 to measure **WLAN** alone throughput

```
#define RSI_ENABLE_WLAN_TEST    1 //Set this to 0 to disable WLAN
```

   set below macro to 1 to measure **WLAN** throughput along with **BLE** connection

```
#define RSI_ENABLE_BLE_TEST     1 //Set this to 0 to disable BLE
```

   set below macro to 1 to measure **WLAN** throughput with **BT** connection

```
#define RSI_ENABLE_BT_TEST      1 //Set this to 1 to enable BT
```

> **Note:** By default, all macros are enabled.

   Configure below macro when BT/BLE is enabled along with WLAN

```
#define WLAN_THROUGHPUT_AFTER_BT_BLE_CONN    1  //! Measure wlan throughput after BT and BLE
```

connections

```
#define WLAN_THROUGHPUT_AFTER_BT_BLE_CONN    0  //! Measure wlan throughput independent of BT
```

and BLE connections

valid
choose the required **operational mode** of RS9116

```
#define RSI_COEX_MODE        9
```

valid configurations:
0 - WLAN alone mode
5 - BT alone mode
9 - WLAN + BT + BLE mode
13 - BLE alone
mode

Note: By default, opermode set to WLAN+BT+BLE.

2. Select WLAN configurations in '**rsi_wlan_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_throughput_bt_spp_ble_dual_role**'

   Enter the AP Connectivity essentials configs as the value to SSID, SECURITY_TYPE and PSK

```
#define     SSID                        "Hotspot"
#define     SECURITY_TYPE               RSI_WPA2
#define     PSK                         "12345678"
```

choose the throughput type by configuring below macro

```
#define     THROUGHPUT_TYPE          UDP_TX
```

valid configurations are

UDP_TX → UDP transmit
UDP_RX → UDP receive
TCP_TX → TCP transmit
TCP_RX → TCP receive
SSL_TX → SSL transmit
SSL_RX → SSL receive

Average time required to measure UDP_TX/TCP_TX throughputs

```
#define     THROUGHPUT_AVG_TIME    60000    //60sec of throughput numbers average
```

Note:

1. While measuring UDP/TCP TX throughput with ble/bt connections/data transfer, ensure the connections/data transfer happens in above configured time.
2. Please increase the THROUGHPUT_AVG_TIME to check the throughput in long running scenarios.

Maximum no. of packets required to measure UDP_RX/TCP_RX/SSL_TX

```
#define     MAX_TX_PKTS         10000
```

Note:

1. While measuring SSL TX/RX throughput with ble/bt connections/data transfer, MAX_TX_PKTS (10000) packet count is not sufficient so we recommended increase the MAX_TX_PKTS and also increase the packet count in "SSL_Server_throughput_d.py" & "SSL_tx_throughput.py" located in "..\host\sapis\examples\utilities\scripts".
2. Present IoT Package scripts supports only 10,000 packets receive/transmit to/from the Module.

Port number of remote server

```
#define     SERVER_PORT         5001
```

IP address of remote server

```
#define     SERVER_IP_ADDRESS    "192.168.0.102"
```

To select the ip getting configure below macros

```
#define      DHCP_MODE                1 //0 enable or disable
#if !DHCP_MODE // Need to configure manually if dhcp disabled
#define    DEVICE_IP          0x6500A8C0  //192.168.0.101
#define    GATEWAY            0x0100A8C0  //192.168.0.1
#define    NETMASK            0x00FFFFFF  //255.255.255.0
#endif
```

3. Open '**rsi_ble_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_throughput_bt_spp_ble_dual_role**' and choose **BLE** application configurations

   BLE Advertise name

```
#define RSI_BLE_APP_GATT_TEST    (void *)"SI_COEX_MAX_DEMO"
```

   Configure BLE advertising interval

```
#define RSI_BLE_ADV_INT_MIN 0x06a8 //! 1065ms
#define RSI_BLE_ADV_INT_MAX 0x06a8 //! 1065ms
```

   Configure below macros to set connection interval, connection latency and connection supervision timeout

```
#define CONN_INTERVAL_M1        400      // connection interval:500ms
#define CONN_LATENCY_M1           0       // latency : 0
#define CONN_SUPERVISION_TIMEOUT_M1   400     // supervision timeout : 400ms
```

4. Select BT configurations in '**rsi_bt_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\sapis\examples\wlan_bt_ble\wlan_throughput_bt_spp_ble_dual_role**'

   Enter the remote BT device address as the value to RSI_BT_REMOTE_BD_ADDR

```
#define     RSI_BT_REMOTE_BD_ADDR     (void *)"B8:D5:0B:9B:D6:B2"
```

   SPP_MODE refers to type of Module Mode, whether its MASTER/SLAVE

```
#define     SPP_MODE               SPP_SLAVE
```

   PIN_CODE refers 4 bytes string required for pairing process

```
#define  PIN_CODE         "0000"
```

   RSI_BT_LOCAL_NAME refers to name of Silicon Labs Module to appear during scanning by remote device

```
#define  RSI_BT_LOCAL_NAME    "SPP_SLAVE"
```

Executing the Application

1. Compile the project and flash the binary onto STM32
2. To measure throughput, following configurations can be applied
   a. To measure UDP Tx throughput, module should be configured as UDP client and open UDP server in remote port using below command
      iperf.exe -s -u -p <SERVER_PORT> -i 1
   Ex: iperf.exe -s -u -p 5001 -i 1
   b. To measure UDP Rx througput, module should be configured as UDP server and open UDP client in remote port using below command
      iperf.exe -c <Module_IP> -u -p <PORT_NUM> -i 1 -b<Bandwidth>
   Ex: iperf.exe -c 192.168.0.1 -u -p 5001 -i 1 -b50M
   c. To measure TCP Tx throughput, module should be configured as TCP client and open TCP server in remote port using below command
      iperf.exe -s -p <SERVER_PORT> -i 1
   Ex: iperf.exe -s -p 5001 -i 1
   d. To measure TCP Rx througput, module should be configured as TCP server and open TCP client in remote port using below command
      iperf.exe -c <Module_IP> -p <PORT_NUM> -i 1
   Ex: iperf.exe -c 192.168.0.1 -p 5001 -i 1
   e. To measure SSL Tx throughput, configure module in SSL client and follow below steps to run SSL server in windows
   → Copy SSL_Server_throughput_d.py from host/sapis/examples/utilities/scripts/
   to host/sapis/examples/utilities/certificates/
      → Open command prompt in folder host/sapis/examples/utilities/certificates/ and run below command
         python SSL_Server_throughput_d.py
   f. To measure SSL Rx throughput, module should be configured as SSL server and follow below steps to run SSL client in
   windows →
   copy SSL_tx_throughput.py from host/sapis/examples/utilities/scripts/ to host/sapis/examples/utilities/certificates/
      → Open command prompt in folder host/sapis/examples/utilities/certificates/ and run below command
         python SSL_tx_throughput.py
3. After the program gets executed, Module scans for the configured Access point, connects to it.
4. Acquires the ip address and waits for bt/ble connections.
5. Open "Bluetooth SPP Manager"(android app) from mobile, scan for 'RSI_BT_LOCAL_NAME' and initiate connection if found.
6. After Successful BT connection, scan for BLE advertise name (RSI_BLE_APP_GATT_TEST) using nRF connect (Android app)/ BLE dongles and initiate ble connection if found.
7. If both the connections are successful, module starts transmitting/receiving wlan packets and throughput measurement is calculated.

   Note: Verify that all connections are stable, and throughput is as expected.

8. To check BLE data transfer along wlan, enable Gatt notifications of Module on service characteristic RSI_BLE_ATTRIBUTE_1_UUID (0x1AA1) using nRF connect.
9. If enabled module continuously transmits 20 notifications per connection interval of size 20bytes.
10. To check BT data transfer along with WLAN/BLE data transfer, open Bluetooth SPP Manager app and send the data.
11. Module receives the data transmitted by app and retransmits the same to BT SPP manager app.

10 Crypto

Following is the list of examples described in this section.

S.NO	Example	Description	Example Source Path
1	AES Example	This application demonstrates how to Encrypt and decrypt data using AES in WiSeConnect.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\crypto\AES
2	ECDH Example	Elliptic-curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol that allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel. This shared secret may be directly used as a key, or to derive another key. The key, or the derived key, can then be used to encrypt subsequent communications using a symmetric-key cipher. It is a variant of the Diffie–Hellman protocol using elliptic-curve cryptograph	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\crypto\ECDH
3	Exponential Example	Diffie-Hellman key agreement protocol (also called exponential key agreement) was developed by Diffie and Hellman [DH76] in 1976.The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\crypto\DH
4	HMAC SHA Example	In cryptography, a keyed-hash message authentication code (HMAC) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\crypto\HMAC
5	SHA Example	In cryptography, SHA (Secure Hash Algorithm) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST SHA produces a message digest based on principles similar to those used by Ronald L. Rivest of MIT in the design of the MD4 and MD5 message digest algorithms, but has a more conservative design.	RS9116.NB0.WC.GENR.OSI.xxxx\host\sapis\examples\crypto\SHA

10.1 AES

Overview

The Advanced Encryption Standard (AES), also known by its original name Rijndael is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits.

AES has been adopted by the U.S. government and is now used worldwide. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

This application demonstrates how to Encrypt and decrypt data using AES in WiSeConnect.

Sequence of Events

This application explains user the following:

- AES CBC mode encryption(256bit key)
- AES CBC mode decryption(256bit key)
- AES ECB mode encryption(256bit key)
- AES ECB mode decryption(256bit key)
- AES CTR mode encryption(256bit key)
- AES CTR mode decryption(256bit key)

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module

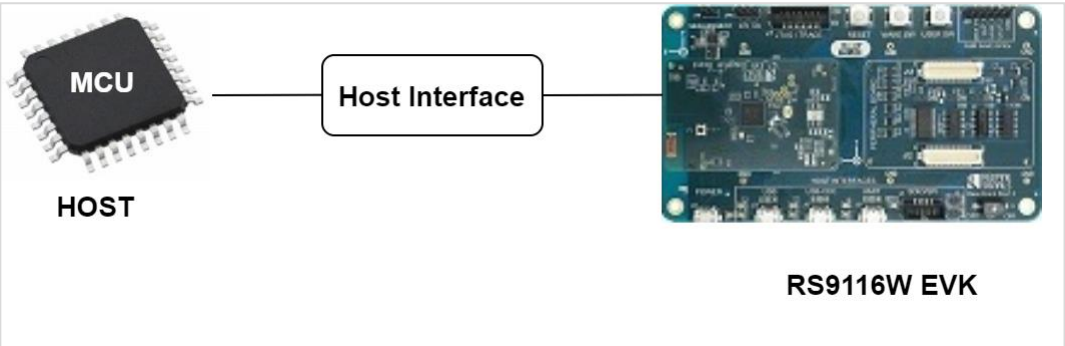


Figure 145: Setup Diagram for AES Example

Configuration and Steps for Execution

Configuring the Application

1. Edit the **rsi_aes_app.c** file

- From given configuration,the following fields can be edited

"**msg"** refers to plain data which is fed to AES engine for encryption with all sizes of key.
"**key"** refers to key which is in turn used for encryption/decryption with AES engine.
"**iv"** refers to IV used in AES-CBC/AES-CTR mode.

- Depending on key size/mode of encryption/decryption provided input data is processed using AES engine.

2. Open ***rsi_wlan_config.h*** file and update/modify following macros

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_HTTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_IEEE_80211J
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Executing the Application

1. Configure the **key**, **msg**, **iv** in the file **rsi_aes_app.c**
2. Build and launch the application.
3. After the program gets executed, Silicon Labs device's AES Encryption / Decryption operations are performed by providing key to AES engine.

10.2 DH

Overview

Diffie-Hellman key agreement protocol (also called exponential key agreement) was developed by Diffie and Hellman [DH76] in 1976.The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.

The protocol has two system parameters p and g. They both are public and may be used by all the users in a system. Parameter p is a prime number and parameter g (usually called a generator) is an integer less than p, with the following property: for every number n between 1 and p-1 inclusive, there is a power k of g such that n = gk module p.

Suppose Alice and Bob want to agree on a shared secret key using the Diffie-Hellman key agreement protocol. They proceed as follows:

- First, Alice generates a random private value **"a"** and Bob generates a random private value **"b"**. Both a and b are drawn from the set of integers.
- They then derive their public values using parameters p and g and their private values. Alice's public value is ga modulo p and Bob's public value is gb modulo p. When finished, they exchange their public values.
- Finally, Alice computes gab = (gb)a modulo p, and Bob computes gba = (ga)b modulo p. Since gab = gba = k, Alice and Bob now have a shared secret key 'k'.
- The desired operation to be performed for generating the key is XE mod P.(analogous to gamod p where E is the Exponent, X is the generator and P is the prime).

Sequence of Events

This application demonstrates

- Diffie-Hellman algorithm to compute exponentiation value or shared secret key for the given input data.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module

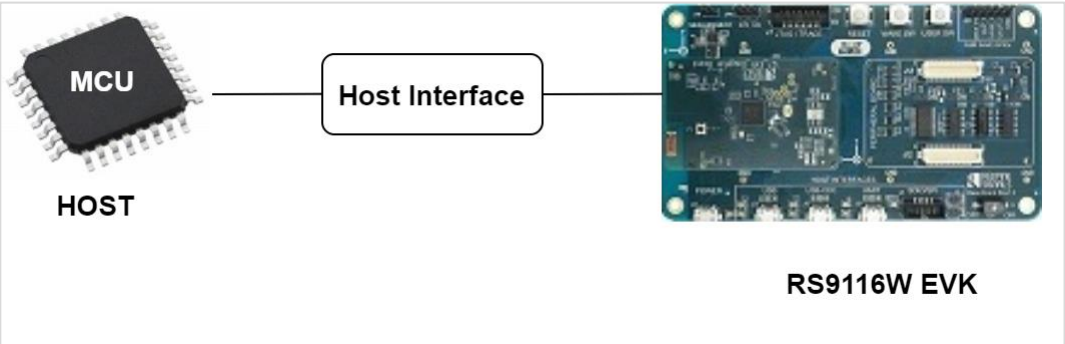


Figure 146: Setup Diagram for Exponentiation Example

Configuration and Steps for Execution

Configuring the Application

1. Edit the following fields in **rsi_exp_app.c** file.

- From given configuration,

"**prime"** refers to the prime value data which is given as input to DH for computing exponentiation value.
"**exp"** refers to the exponent value which is given as input to DH for computing exponentiation value.
"**base"** refers to the base value which is given as input to DH for computing exponentiation value.

- Depending on given input data prime, exponent and base values corresponding DH key is generated.

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_HTTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_IEEE_80211J
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Executing the Application

1. Connect Silicon Labs device to the Windows PC with Keil or IAR IDE in running state.
2. Configure the **DH** in the file **rsi_exp_app.c**.
3. Build and launch the application.
4. After the program gets executed, Silicon Labs device generates DH key for the given input data.

10.3 ECDH

Overview

Elliptic-curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol that allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel. This shared secret may be directly used as a key, or to derive another key. The key, or the derived key, can then be used to encrypt subsequent communications using a symmetric-key cipher. It is a variant of the Diffie–Hellman protocol using elliptic-curve cryptography.

Sequence of Events

This Application demonstrates the various ECDH operations like

- ECDH point addition
- ECDH point subtraction
- ECDHpoint multiplication
- ECDH point double
- ECDH affinity

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module

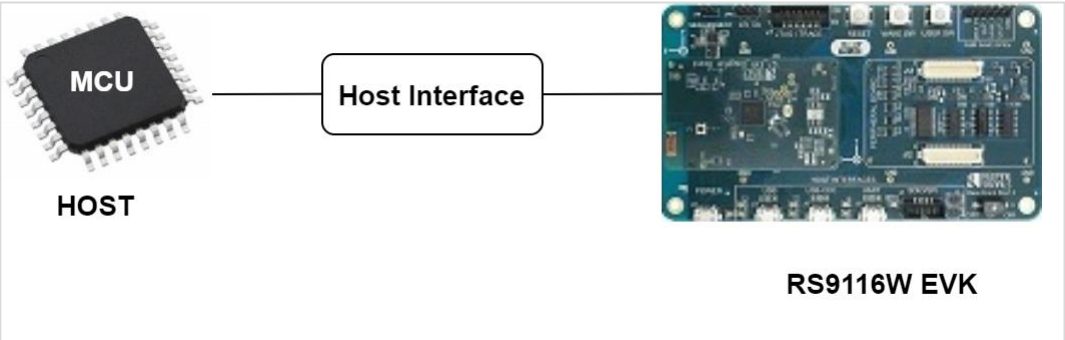


Figure 147: Setup Diagram of ECDH Example

Configuration and Steps for Execution

Configuring the Application

1. Edit the **rsi_ecdh_app.c** file:

- From given configuration,
  Input vectors sz, sy, sz and tx, ty, tz along with the scalar multiplier d are given to the ECDH.
- Depending on ECDH mode and ECDH operation respective outputs are computed for the given input data.

2. Open ***rsi_wlan_config.h*** file update / modify following macros,

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_HTTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_IEEE_80211J
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Executing the Application

1. Connect Silicon Labs device to the Windows PC with Keil or IAR IDE in running state.
2. Configure the **ECDH** in the file **rsi_ecdh_app.c.**
3. Build and launch the application.
4. After the program gets executed, Silicon Labs Device generates the corresponding output for the given input data.

10.4 HMAC

Overview

In cryptography, a keyed-hash message authentication code (HMAC) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC.

The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

HMAC generation uses two passes of hash computation. The secret key is first used to derive two keys - inner and outer. The first pass of the algorithm produces an internal hash derived from the message and the inner key. The second pass produces the final HMAC code derived from the inner hash result and the outer key. Thus the algorithm provides better immunity against Length extension attacks.

HMAC does not encrypt the message. Instead, the message (encrypted or not) must be sent alongside the HMAC hash. Parties with the secret key will hash the message again themselves, and if it is authentic, the received and computed hashes will match.

The configuration application demonstrates how to compute digest using HMAC SHA. This application computes a digest of 64 bytes using HMAC SHA.

Sequence of Events

The application demonstrates:

- Computation of 64 bytes digest for the given input data using HMAC SHA.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module

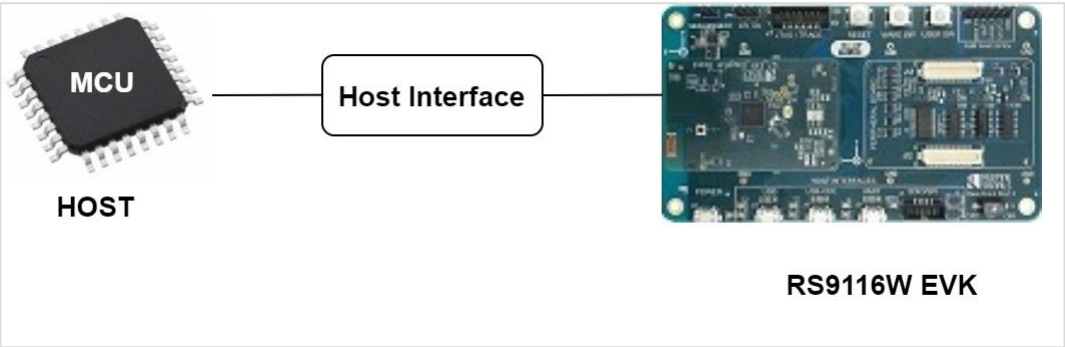


Figure 148: Setup Diagram for HMAC SHA Example

Configuration and Steps for Execution

Configuring the Application

1. Edit the following fields in **rsi_hmac_sha_app.c** file in the following path:

- From given configuration,

"**msg"** refers to data which is given as input to HMAC SHA for computing digest.
"**key"** refers to the key input for HMAC SHA.

#define HMAC_BUFF_LEN 4000

Length of the HMAC SHA buffer. It depends on the input message length and key length. It should be greater than the addition of message length and key length.

- Depending on HMAC SHA mode, digest of respective size is computed for the given input data.

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_HTTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_IEEE_80211J
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Executing the Application

1. Connect Silicon Labs device to the Windows PC with Keil or IAR IDE in running state.
2. Configure the **HMAC SHA** in the file **rsi_hmac_sha_app.c**
3. Build and launch the application.
4. After the program gets executed, Silicon Labs Device generates the digest of 64 bytes for the given input data.

10.5 SHA

Overview

In cryptography, SHA (Secure Hash Algorithm) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST

SHA produces a message digest based on principles similar to those used by Ronald L. Revest of MIT in the design of the MD4 and MD5 message digest algorithms, but has a more conservative design.

SHA forms part of several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec. Those applications can also use MD5; both MD5 and SHA are descended from MD4.

Sequence of Events

This Application demonstrates user how to:

- Compute a digest of 64 bytes using SHA.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silicon Labs module

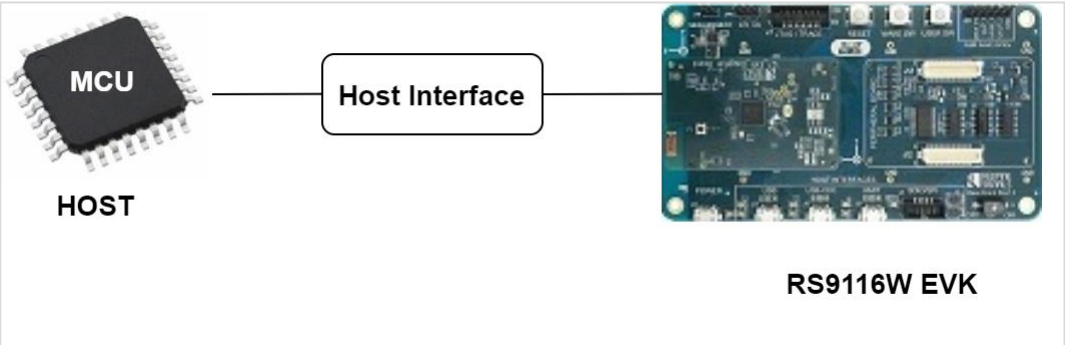


Figure 149: Setup Diagram for SHA Example

Configuration and Steps for Execution

Configuring the Application

1. Edit the **rsi_sha_app.c** file

- From given configuration,
  "**SHA"** refers to data which is given as input to SHA for computing digest.
- Depending on SHA mode digest of respective size is computed for the given input data.

2. Open ***rsi_wlan_config.h*** file and update/modify following macros,

```
#define CONCURRENT_MODE                 RSI_DISABLE
#define RSI_FEATURE_BIT_MAP             FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS               RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP      (TCP_IP_FEAT_DHCPV4_CLIENT | TCP_IP_FEAT_HTTP_CLIENT)
#define RSI_CUSTOM_FEATURE_BIT_MAP      FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP  EXT_FEAT_IEEE_80211J
#define RSI_BAND                        RSI_BAND_2P4GHZ
```

Executing the Application

1. Connect Silicon Labs device to the Windows PC with Keil or IAR IDE in running state.
2. Configure the **SHA** in the file
   - **rsi_sha_app.c**
3. Build and launch the application.
4. After the program gets executed, Silicon Labs Device generates the digest of 64 bytes for the given input data.

11 Debug Utils

This section provides examples in order to easily carry out the configuration and execution of the debug applications of RS9116 module.

11.1 RAM Dump

Overview

The RAM dump application is use to read the RAM content of RS9116 module for a given address.

Example Setup

The WiSeConnect parts require that the host processor is connected to the WiSeConnect using either SPI, USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (SPI/ USB) in case of WiSeConnect
- Silicon Labs module

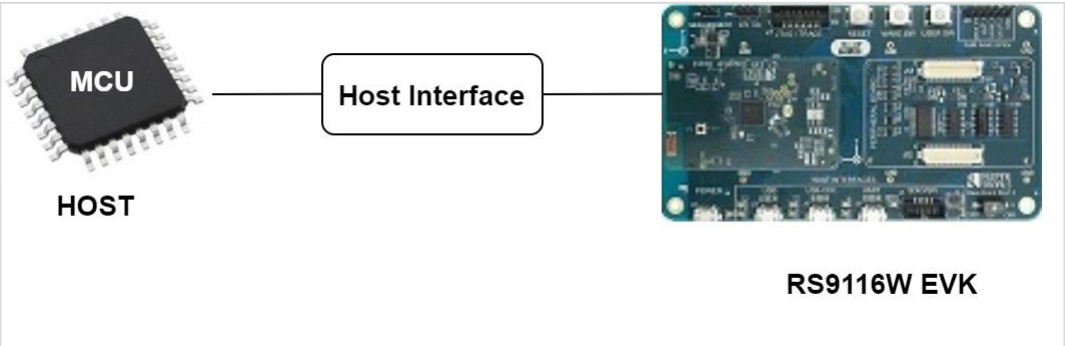


Figure 150: Setup Diagram for RAM Dump Example

Configuration and Steps for Execution

Configuring the Application

Open ***rsi_ram_dump.c*** file and update/modify following macros,

Path : /host/sapis/examples/debug_utils/ram_dump

This define should be ENABLE only in case of linux platform with USB interface by default it is DISABLE

```
#define LINUX_PLATFORM          DISABLE
```

Address in Silicon Labs module

```
#define READ_ADDRESS            0x0
```

Executing the Application

1. Build and launch the application.

2. After the program gets executed RAM content is saved in ram_content buffer. In case of USB interface RAM content is saved in file dump.txt.

12 PER Test

This section provides example application for PER Test.

This section describes:

- The different commands supported by the WiSeConnect™ PER application in an automated manner.
- Support for PER test for different protocols like WLAN, BLE, BT. Operational mode for each protocol has to be set individually for conducting each PER Test.
- The DUT uses the UART Interface for communicating with the Master(External Application like MATLAB) for configuring the Wireless System and for logging the Wireless performance
- Also, describes the constraints in using each command.

12.1 PER Test App

Introduction

This example is applicable to WiSeConnect™. The feature(s) used in this example may or may not be available on your part number. Refer to the product datasheet to verify the features available.

Overview

This section describes:

- The different commands to be used for enabling the Master to configure and log the Wireless performance.
- The commands to be given for different protocols like WLAN, BT and BLE/BLR and also a few common commands for initialization of Wireless System

All the command needs to be send to UART interface in a string format.

Sequence of Events

- Connect any serial terminal (Teraterm/cutecom) to FRDM-K28 serial interface for debug prints and to send UART commands.
- Issue the UART commands (In APP Note below) on serial terminal (Teraterm/Cutecom/Docklight) .
- Start executing the PER_TEST demo as per above steps in Steps to Execute Demo 54

Steps to execute WI-FI PER

- Configure operational and coex modes.
- Configure WLAN features.
- WIFI PER Transmit sequence.
    - Tx Start
    - Tx Stop
- WIFI PER Receive sequence.
    - Rx Start
    - Rx Statistics
    - Rx stop

The detailed description of the steps is mentioned below.

Example Setup

The WiSeConnect parts require that the host processor should be connected to the WiSeConnect either using SPI, UART or USB host interface. The host processor firmware needs to properly initialize the selected host interface. The Silicon Labs Wireless SAPI framework provides necessary HAL APIs to enable variety of host processors.

WiSeConnect based Setup Requirements

- Windows / Linux PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect

- RS9116W Module

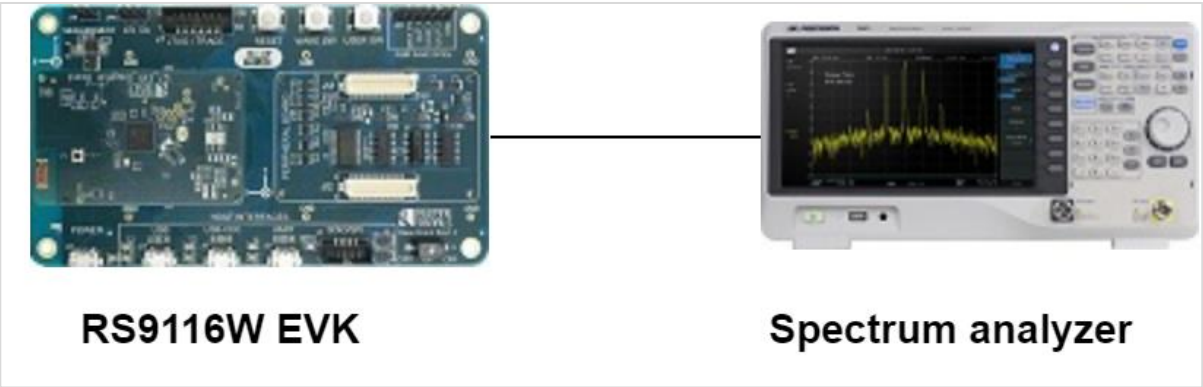


Figure 151: Setup Diagram for PER Tx Example

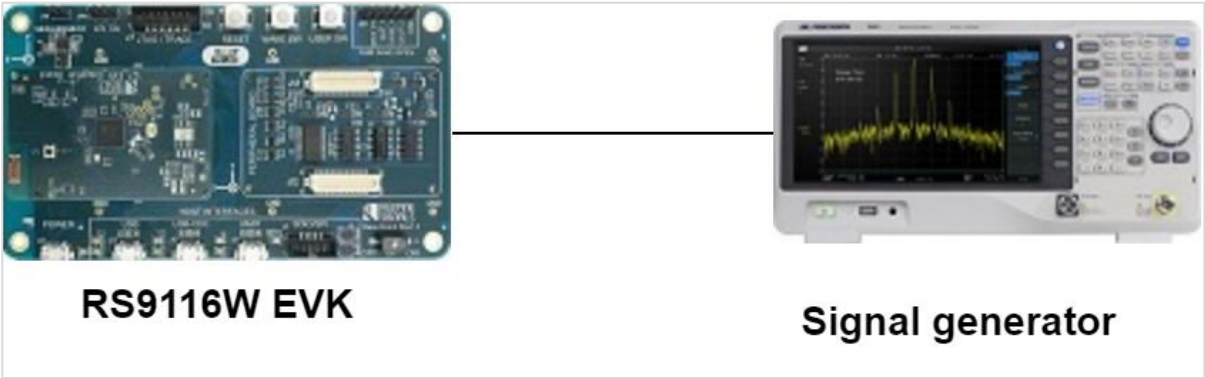


Figure 152: Setup Diagram for PER Rx Example

Execution of the Application

Operational Mode configuration

This command initializes the Wireless Processor for the Operational Mode and COEX Mode specified in the command. The Wireless system will be reset before initializing to the selected Operational Mode. Description of these parameters is provided below:

Command: {1 <oper_mode> <coex_mode>\n}

Example: 1 8 0\n

WLAN Features Configuration

This command enables the Master to configure different features supported by the WLAN system to be used for Transmit/Receive. Description of these parameters is provided below:

Command: {2 <pll_mode> <rf_type> <wireless_mode> <enable_ppp> <afe_type> <features>\n}

Example: 2 0 1 0 0 1 0\n

Description:

Parameter	Value
pll_mode	0 : PLL Mode 0 1: PLL Mode 1 2: PLL Mode 2
rf_type	0 : External RF 1: Internal RF
wireless_mode	0: WLAN HP Chain 12:WLAN LP Chain
enable_ppp	0: Disables Tx Per-packet programming 1:Enables Tx Per-packet programming
afe_type	0: External AFE 1: Internal AFE
Features	0: Disable [duty cycling & end of frame] 1: Duty cycling Enabled 2: End_of_Frame 3. Enable [Duty_cycling& End_of_Frame]

WIFI PER Antenna Select

antenna selection command

This command enables the Master to start the WLAN transmit. Description of these parameters is provided below:

Command: {S <ant_sel> <gain_2g> <gain_5g> \n}

Description: S 1 0 0\n

Parameter	Value
ant_sel	1 : Select Internal antenna or uFL connector
gain_2g	0 : Antenna gain in 2.4GHz band
gain_5g	0 : Antenna gain in 5GHz band

WIFI PER Transmit Sequence

Tx Start Commands

This command enables the Master to start the WLAN transmit. Description of these parameters is provided below:

Command: {3 <chan_idx> <tx_mode> <pkt_length> <rate> <tx_power> \n}

Description: 3 1 0 1000 6 10\n

Parameter	Value
chan_idx	Indicates the channel Index as specified by IEEE 802.11.
tx_mode	0 : Burst Mode 1 : Continuous Mode 2 : Continuous wave Mode (non modulation) in DC mode

Parameter	Value
	3 : Continuous wave Mode (non modulation) in single tone mode (center frequency -2.5MHz) 4: Continuous wave Mode (non modulation) in single tone mode (center frequency +5MHz)pkt_length
pkt_length	Indicates the length of the packet to be transmitted in bytes
Rate	Indicates the rate at which the packet needs to be transmitted 0 : 1 Mbps 2 : 2 Mbps 4 : 5.5 Mbps 6 : 11 Mbps 139 : 6 Mbps 143 : 9 Mbps 138 : 12 Mbps 142 : 18 Mbps 137 : 24 Mbps 141 : 36 Mbps 136 : 48 Mbps 140 : 54 Mbps 256 : MCS0 257 : MCS1 258 : MCS2 259 : MCS3 260 : MCS4 261 : MCS5 262 : MCS6 263 : MCS7
tx_power	Indicates the Output power at which the packet needs to be transmitted.

Tx Stop Commands

This command enables the Master to stop the WLAN transmit operation.

Command: {6 0 \n}

Example: 6 0\n

WIFI PER Receive Sequence

Rx Start

This command enables the Master to start the WLAN receive.

Command: {4 <chan_idx>\n}

Example: 4 1\n

Description:

chan_idx: Indicates the channel Index as specified by IEEE 802.11.

Statistics:

This command enables the Master to get the performance statistics(CRC_PASS, CRC_FAIL, RSSI) from the Wireless system.Description of these parameters is provided below:

Command: {7 <N>\n}

Example: 7 8\n

Description: N: Indicates the number of seconds for which the statistics are obtained.

Response:

- This command returns the following data,each data shown below is printed N times
  {<crc_pass_cnt > <crc_fail_cnt > <rssi >}
  Ex: crc_pass_cnt =260 crc_fail_cnt =0 rssi =0
- If **AVG_STATS_REQ** is enable, this command returns the following data (4*N*3 characters).

  Each data shown below is represented by 4 characters.

  {<crc_pass_1> <crc_fail_1> <rssi_1> <crc_pass_2> <crc_fail_2> <rssi_2> <crc_pass_3> <crc_fail_3> <rssi_3>..... <crc_pass_N> <crc_fail_N> <rssi_N>}

Rx Stop:

This command enables the Master to stop the WLAN receive operation.

Command: {5 0 \n}

Example: 5 0\n

Steps to execute BLE PER

- Configure Operational and coex modes.
- BLE PER Transmit sequence.
  - Tx Start
  - Tx Stop
- BLE Receive sequence.
  - Rx Start
  - Rx Statistics
  - Rx Stop

The detailed description of the steps is mentioned below.

Operational Mode configuration:

This command initializes the Wireless Processor for the Operational Mode and COEX Mode specified in the command. The Wireless system will be reset before initializing to the selected Operational Mode. Description of these parameters is provided below:

Command: {1 <oper_mode> <coex_mode>\n}

Example: 1 0 13\n

BLE PER Transmit

Start:

This command enables the Master to start the BLE/BLR transmit with the transmit variables and the wireless features provided through this command. Description of these parameters are provided below.

Command: {8 19 1 <access_addr> <phy_rate> <rx_chnl_num> <tx_chnl_num> <scrambler_seed> <le_chnl_type> <freq_hop_en> <ant_sel> <pll_mode> <rf_type> <rf_chain> <pkt_len> <payload_type> <tx_power> <transmit_mode> <inter_pkt_gap>\n}

Example: 8 19 1 0x71764129 1 39 39 0 1 0 2 0 1 2 255 6 10 0 0\n

Description:

Parameters	Value
access_addr	It is a 32-bit address in Hex format (Access address of BLE PER packet : 0x71764129)
phy_rate:	Indicates the rate of transmission 1 : LE_1Mbps 2 : LE_2Mbps 4 : LR_125Kbps 8 : LR_500Kbps
rx_chnl_num	Receive channel index, as per the Bluetooth standard.i.e, 0 to 39
tx_chnl_num	Transmit channel index, as per the Bluetooth standard. i.e, 0 to 39.
scrambler_seed	Initial seed to be used for whitening. It should be set to '0' in order to disable whitening.
le_chnl_type	0 : Advertising channel 1 : Data channel
freq_hop_en	0 : No hopping 1 : Fixed hopping 2 : Random hopping
ant_sel	2 : On-chip antenna 3 : External u.f.l antenna
pll_mode	0 : PLL_MODE0 1 : PLL_MODE1 2 : PLL_MODE2
rf_type	0 : External RF 1 : Internal RF
rf_chain	0 : WLAN_HP_CHAIN 1 : WLAN_LP_CHAIN 2 : BT_HP_CHAIN 3 : BT_LP_CHAIN
pkt_len	Length of the packet, in bytes, to be transmitted(range : 0 to 255 bytes).
payload_type	Type of payload to be transmitted 0 : PRBS9 Sequence 1 : 0x0F 2 : 0x55 3 : PBRS15 sequence 4 : 0xFF 5 : 0x00 6 : 0xF0 7 : 0xA0
tx_power	Transmit power value should be transmitted. [1-31] : BLE-LP Chain 0DBM Mode [33-63] : BLE-LP Chain 10DBM Mode Note: For BLE-LP Chain tx_power value 0 and 32 are invalid.

Parameters	Value
transmit_mode	0 : Burst mode 1 : Continuous mode
inter_pkt_gap	Number of slots to be skipped between two packets - Each slot will be 1250usec.

Stop

This command enables the Master to stop the BLE PER transmit operation.

Command: {B 0\n}

Example: B 0\n

BLE PER Receive

Start

This command enables the Master to start the BLE/BLR receive with the receive variables and the wireless features provided through this command.

Command: {9 20 1 <access_addr> <phy_rate> <rx_chnl_num> <tx_chnl_num> <scrambler_seed> <le_chnl_type> <freq_hop_en> <ant_sel> <pll_mode> <rf_type> <rf_chain> <ext_data_len_indication> <loop_back_mode> <pwrsave_options>\n}

Example: 9 20 1 0x71764129 2 39 39 0 1 0 2 0 1 3 1 0 0\n

Description:

Parameters	Value
access_addr	It is a 32-bit address in Hex format
phy_rate	Indicates the rate at which BLE transmits 1 : LE_1Mbps 2 : LE_2Mbps 4 : LR
rx_chnl_num	Receive channel index, as per the Bluetooth standard.i.e, 0 to 39
tx_chnl_num	Transmit channel index, as per the Bluetooth standard. i.e, 0 to 39
scrambler_seed	Initial seed to be used for whitening. It should be set to ‘0’ in order to disable whitening
le_chnl_type	0 : Advertising channel 1 : Data channel
freq_hop_en	0 : No hopping 1 : Fixed hopping 2 : Random hopping
ant_sel	2 : On-chip antenna 3 : External u.f.l antenna
pll_mode	0 : PLL_MODE0 1 : PLL_MODE1 2 : PLL_MODE2

Parameters	Value
rf_type	0 : External RF 1 : Internal RF
rf_chain	0 : WLAN_HP_CHAIN 1 : WLAN_LP_CHAIN 2 : BT_HP_CHAIN 3 : BT_LP_CHAIN
ext_data_len_indication	0 : Disable(<=37 Bytes) 1 : Enable(<=255 Bytes)
loop_back_mode	0 : Disable 1 : Enable
pwrsave_options	0 : Duty-cycling and 4x Mode are disabled. 1 : Duty-cycling is enabled and 4x Mode are disabled. 128 : Duty-cycling is disabled and 4x Mode are enabled. 129 : Duty-cycling and 4x Mode are enabled.

Statistics

This command enables the Master to get the performance statistics(CRC_PASS, CRC_FAIL, RSSI) from the Wireless system.

Command: {C <N>\n}

Example: C 8\n

Description:

N: Indicates the number of seconds for which the statistics are obtained.

- This command returns the following data,each data shown below is printed N times
  {<crc_pass_cnt > <crc_fail_cnt > <rssi >}
  Ex: crc_pass_cnt =260 crc_fail_cnt =0 rssi =0
- If **AVG_STATS_REQ** is enabled, this command returns the following data (4*N*3 characters).

  Each data shown below is represented by 4 characters.

  {<crc_pass_1> <crc_fail_1> <rssi_1> <crc_pass_2> <crc_fail_2> <rssi_2> <crc_pass_3> <crc_fail_3> <rssi_3>..... <crc_pass_N> <crc_fail_N> <rssi_N>}

  Note:

  By default, AVG_STATS_REQ is disabled

  To get logs in above format, enable below macro

  #define AVG_STATS_REQ 1

  File path:
  RSI_WSDK_vX/host/sapis/examples/wsdk_apps/PER_TEST_DEMO_54/rsi_per_app_DEMO_54.h

Stop

This command enables the Master to stop the BLE receive operation.

Command: {A 0\n}

Example: A 0\n

Steps to execute BT PER

These are the steps to be followed for executing BT PER. The detailed description of the steps is mentioned below.

- Configure operational and coex modes.
- BT PER Transmit sequence.
    - Tx Start
    - Tx Stop
- BT PER Receive sequence.
    - Rx Start
    - Rx Statistics
    - Rx Stop

The detailed description of the steps is mentioned below.

Operational Mode configuration:

This command initializes the Wireless Processor for the Operational Mode and COEX Mode specified in the command. The Wireless system will be reset before initializing to the selected Operational Mode. Description of these parameters is provided below:

Command: {1 <oper_mode> <coex_mode>\n}

Example: 1 0 5\n

BT PER Transmit

Start

This command enables the Master to start the BT-Classic transmit with the transmit variables and the wireless features provided through this command.

Command: {I 21 1 <access_addr_lsb> <access_addr_msb> <pkt_len> <pkt_type> <br_edr_mode> <rx_chnl_num> <tx_chnl_num> <link_type> <scrambler_seed> <payload_type> <tx_power> <tx_mode> <hopping_type> <ant_sel> <inter_pkt_gap> <pll_mode> <rf_type> <rf_chain> \n}

Example: I 21 1 0x12345678 0x0000 1021 3 2 78 78 1 0 4 10 1 0 2 0 0 1 2\n

Description:

Parameters	Value
access_addr_lsb, access_addr_msb	It is a 48-bit address in hexadecimal format, e.g.,0x000012345678 access_addr_lsb = 0x12345678 access_addr_msb = 0x0000
pkt_type	Type of the packet to be transmitted, as per the Bluetooth standard. 3: BT_DM1_PKT_TYPE 4: BT_DH1_PKT_TYPE 10: BT_DM3_PKT_TYPE 11: BT_DH3_PKT_TYPE 14: BT_DM5_PKT_TYPE 15: BT_DH5_PKT_TYPE

Parameters	Value
	4: BT_2DH1_PKT_TYPE 10: BT_2DH3_PKT_TYPE 14: BT_2DH5_PKT_TYPE 8: BT_3DH1_PKT_TYPE 11: BT_3DH3_PKT_TYPE 15: BT_3DH5_PKT_TYPE 5: BT_HV1_PKT_TYPE 6: BT_HV2_PKT_TYPE 7: BT_HV3_PKT_TYPE 8: BT_DV_PKT_TYPE 7: BT_EV3_PKT_TYPE 6: BT_2EV3_PKT_TYPE 7: BT_3EV3_PKT_TYPE 12: BT_EV4_PKT_TYPE 12: BT_2EV5_PKT_TYPE 13: BT_EV5_PKT_TYPE 13: BT_3EV5_PKT_TYPE
pkt_length	Length of the packet, in bytes, to be transmitted. 17: BT_DM1_PAYLOAD_MAX_LEN 121: BT_DM3_PAYLOAD_MAX_LEN 224: BT_DM5_PAYLOAD_MAX_LEN 27: BT_DH1_PAYLOAD_MAX_LEN 183: BT_DH3_PAYLOAD_MAX_LEN 339: BT_DH5_PAYLOAD_MAX_LEN 54: BT_2DH1_PAYLOAD_MAX_LEN 367: BT_2DH3_PAYLOAD_MAX_LEN 679: BT_2DH5_PAYLOAD_MAX_LEN 83: BT_3DH1_PAYLOAD_MAX_LEN 552: BT_3DH3_PAYLOAD_MAX_LEN 1021:BT_3DH5_PAYLOAD_MAX_LEN 10: BT_HV1_VOICE_PAYLOAD_LEN 20: BT_HV2_VOICE_PAYLOAD_LEN 30: BT_HV3_VOICE_PAYLOAD_LEN 30: BT_EV3_VOICE_PAYLOAD_LEN 60: BT_2EV3_VOICE_PAYLOAD_LEN 90: BT_3EV3_VOICE_PAYLOAD_LEN 120: BT_EV4_VOICE_PAYLOAD_LEN 180: BT_EV5_VOICE_PAYLOAD_LEN 360: BT_2EV5_VOICE_PAYLOAD_LEN 540: BT_3EV5_VOICE_PAYLOAD_LEN
br_edr_mode	1: basic rate

Parameters	Value
	2/3: enhanced_rate
rx_channel_index	Receive channel index, as per the Bluetooth standard.i.e, 0 to 78
tx_channel_index	Transmit channel index, as per the Bluetooth standard. i.e, 0 to 78
link_type	0: sco 1: acl 2: esco
scrambler_seed	Initial seed to be used for whitening. It should be set to '0' in order to disable whitening.
no_of_packets	Number of packets to be transmitted. It is valid only when the <tx_mode> is set to Burst mode
payload_type	Type of payload to be transmitted 0: Payload consists of all zeros 1: Payload consists of all 0xFF's 2: Payload consists of all 0x55's 3: Payload consists of all 0xF0's 4: Payload consists of PN9 sequence.
tx_power	Transmit power value should be between 0 and 18
tx_mode	0: Burst mode 1: Continuous mode
hopping type	0: no hopping 1: fixed hopping 2: random hopping
ant_sel	2: onchip antenna 3: u.f.l
inter_pkt_gap	Number of slots to be skipped between two packets
pll_mode	0: PLL_MODE0 1: PLL_MODE1 2: PLL_MODE2
rf_type	0: External RF 1: Internal RF
rf_chain	0: WLAN_HP_CHAIN 1: WLAN_LP_CHAIN 2: BT_HP_CHAIN 3: BT_LP_CHAIN

Stop

This command enables the Master to stop the BT-Classic transmit operation.

Command: {L 0\n}

Example: L 0\n

BT PER Receive

Start

This command enables the Master to start the BT-Classic receive with the receive variables and the wireless features provided through this command.

Command: {J 22 1 <access_addr_lsb> <access_addr_msb> <pkt_len> <pkt_type> <br_edr_mode> <rx_chnl_num> <tx_chnl_num> <link_type> <scrambler_seed> <hopping_type> <ant_sel> <pll_mode> <rf_type> <rf_chain> <loop_back_mode>\n}

Example: J 22 1 0x12345678 0x0000 1021 3 2 78 78 1 0 0 2 0 1 3 0\n

Description:

Parameters	Value
access_addr_lsb, access_addr_msb	It is a 48-bit address in hexadecimal format, e.g.,0x000012345678
pkt_type	Type of the packet to be transmitted, as per the Bluetooth standard.
pkt_length	Length of the packet, in bytes, to be transmitted.
br_edr_mode	1: basic rate 2 or 3: enhanced_rate
rx_channel_index	Receive channel index, as per the Bluetooth standard.i.e, 0 to 78
tx_channel_index	Transmit channel index, as per the Bluetooth standard. i.e, 0 to 78
ink_type	0: sco 1: acl 2: esco
scrambler_seed	Initial seed to be used for whitening. It should be set to ‘0’ in order to disable whitening.
hopping type	0: no hopping 1: fixed hopping 2: random hopping
ant_sel	2: onchip antenna 3: u.f.l
inter_pkt_gap	Number of slots to be skipped between two packets
pll_mode	0: PLL_MODE0 1: PLL_MODE1 2: PLL_MODE2
rf_type	0: External RF 1: Internal RF
rf_chain	0: WLAN_HP_CHAIN 1: WLAN_LP_CHAIN 2: BT_HP_CHAIN 3: BT_LP_CHAIN
loop_back_mode	0: Disable 1: Enable

Statistics

This command enables the Master to get the performance statistics(Packet Contents) from the Wireless system so that the Master can compute PER over the received packets.

Command: {M <N>\n}

Example: M 8\n

Description:

N: Indicates the number of seconds for which the statistics are obtained.

- This command returns the following data,each data shown below is printed N times
  {<crc_pass_cnt > <crc_fail_cnt > <rssi >}
  Ex: crc_pass_cnt =260 crc_fail_cnt =0 rssi =0
- If **AVG_STATS_REQ** is enabled, this command returns the following data (4*N*3 characters).

  Each data shown below is represented by 4 characters.

  {<crc_pass_1> <crc_fail_1> <rssi_1> <crc_pass_2> <crc_fail_2> <rssi_2> <crc_pass_3> <crc_fail_3> <rssi_3>..... <crc_pass_N> <crc_fail_N> <rssi_N>}

  Note:

  By default, AVG_STATS_REQ is disabled

  To get logs in above format, enable below macro

  #define AVG_STATS_REQ 1

  File path:
  RSI_WSDK_vX/host/sapis/examples/wsdk_apps/PER_TEST_DEMO_54/rsi_per_app_DEMO_54.h

Stop

This command enables the Master to stop the BT-Classic receive operation.

Command: {K 0\n}

Example: **K 0\n**

Steps to execute BT BER

These are the steps to be followed for executing BT BER. The detailed description of the steps is mentioned below.

- Configure operational and coex modes.
- BT BER Transmit sequence.
  - TX Start
  - TX Stop
- BT BER Receive sequence.
  - RX Start
  - RX Statistics
  - RX Stop

The detailed description of the steps is mentioned below.

Operational Mode configuration

This command initializes the Wireless Processor for the Operational Mode and COEX Mode specified in the command. The Wireless system will be reset before initializing to the selected Operational Mode. Description of these parameters is provided below:

Command: {1 <oper_mode> <coex_mode>\n}

Example: 1 0 5\n

BT BER Transmit

Start

This command enables the Master to start the BT-Classic transmit with the transmit variables and the wireless features provided through this command.

Command: {D 21 1 <access_addr_lsb> <access_addr_msb> <pkt_len> <pkt_type> <br_edr_mode> <rx_chnl_num> <tx_chnl_num> <link_type> <scrambler_seed> <payload_type> <tx_power> <tx_mode> <hopping_type> <ant_sel> <inter_pkt_gap> <pll_mode> <rf_type> <rf_chain> \n}

Example: D 21 1 0x12345678 0x0000 1021 3 2 78 78 1 0 4 18 1 0 2 0 0 1 2\n

Description:

Parameters	Value
access_addr_lsb, access_addr_msb	It is a 48-bit address in hexadecimal format, e.g.,0x000012345678 access_addr_lsb = 0x12345678 access_addr_msb = 0x0000
pkt_type	Type of the packet to be transmitted, as per the Bluetooth standard. 3: BT_DM1_PKT_TYPE 4: BT_DH1_PKT_TYPE 10: BT_DM3_PKT_TYPE 11: BT_DH3_PKT_TYPE 14: BT_DM5_PKT_TYPE 15: BT_DH5_PKT_TYPE 4: BT_2DH1_PKT_TYPE 10: BT_2DH3_PKT_TYPE 14: BT_2DH5_PKT_TYPE 8: BT_3DH1_PKT_TYPE 11: BT_3DH3_PKT_TYPE 15: BT_3DH5_PKT_TYPE 5: BT_HV1_PKT_TYPE 6: BT_HV2_PKT_TYPE 7: BT_HV3_PKT_TYPE 8: BT_DV_PKT_TYPE 7: BT_EV3_PKT_TYPE 6: BT_2EV3_PKT_TYPE 7: BT_3EV3_PKT_TYPE 12: BT_EV4_PKT_TYPE 12: BT_2EV5_PKT_TYPE 13: BT_EV5_PKT_TYPE 13: BT_3EV5_PKT_TYPE
pkt_length	Length of the packet, in bytes, to be transmitted. 17: BT_DM1_PAYLOAD_MAX_LEN

Parameters	Value
	121: BT_DM3_PAYLOAD_MAX_LEN 224: BT_DM5_PAYLOAD_MAX_LEN 27: BT_DH1_PAYLOAD_MAX_LEN 183: BT_DH3_PAYLOAD_MAX_LEN 339: BT_DH5_PAYLOAD_MAX_LEN 54: BT_2DH1_PAYLOAD_MAX_LEN 367: BT_2DH3_PAYLOAD_MAX_LEN 679: BT_2DH5_PAYLOAD_MAX_LEN 83: BT_3DH1_PAYLOAD_MAX_LEN 552: BT_3DH3_PAYLOAD_MAX_LEN 1021: BT_3DH5_PAYLOAD_MAX_LEN 10: BT_HV1_VOICE_PAYLOAD_LEN 20: BT_HV2_VOICE_PAYLOAD_LEN 30: BT_HV3_VOICE_PAYLOAD_LEN 30: BT_EV3_VOICE_PAYLOAD_LEN 60: BT_2EV3_VOICE_PAYLOAD_LEN 90: BT_3EV3_VOICE_PAYLOAD_LEN 120: BT_EV4_VOICE_PAYLOAD_LEN 180: BT_EV5_VOICE_PAYLOAD_LEN 360: BT_2EV5_VOICE_PAYLOAD_LEN 540: BT_3EV5_VOICE_PAYLOAD_LEN
br_edr_mode	1: basic rate 2/3: enhanced_rate
rx_channel_index	Receive channel index, as per the Bluetooth standard.i.e, 0 to 78
tx_channel_index	Transmit channel index, as per the Bluetooth standard. i.e, 0 to 78
link_type	0: sco 1: acl 2: esco
scrambler_seed	Initial seed to be used for whitening. It should be set to ‘0’ in order to disable whitening.
no_of_packets	Number of packets to be transmitted. It is valid only when the <tx_mode> is set to Burst mode
payload_type	Type of payload to be transmitted 0: Payload consists of all zeros 1: Payload consists of all 0xFF’s 2: Payload consists of all 0x55’s 3: Payload consists of all 0xF0’s 4: Payload consists of PN9 sequence.
tx_power	Transmit power value should be between 0 and 18
tx_mode	0: Burst mode

Parameters	Value
	1: Continuous mode
hopping type	0: no hopping 1: fixed hopping 2: random hopping
ant_sel	2: onchip antenna 3: u.f.l
inter_pkt_gap	Number of slots to be skipped between two packets
pll_mode	0: PLL_MODE0 1: PLL_MODE1 2: PLL_MODE2
rf_type	0: External RF 1: Internal RF
rf_chain	0: WLAN_HP_CHAIN 1: WLAN_LP_CHAIN 2: BT_HP_CHAIN 3: BT_LP_CHAIN

Stop

This command enables the Master to stop the BT-Classic transmit operation.

Command: {G 0\n}

Example: G 0\n

BT BER Receive

Start

This command enables the Master to start the BT-Classic receive with the receive variables and the wireless features provided through this command.

Command: {E 22 1 <access_addr_lsb> <access_addr_msb> <pkt_len> <pkt_type> <br_edr_mode> <rx_chnl_num> <tx_chnl_num> <link_type> <scrambler_seed> <hopping_type> <ant_sel> <pll_mode> <rf_type> <rf_chain> <loop_back_mode>\n}

Example: E 22 1 0x12345678 0x0000 1021 3 2 78 78 1 0 0 2 0 1 3 0\n

Description:

Parameters	Value
access_addr_lsb, access_addr_msb	It is a 48-bit address in hexadecimal format, e.g.,0x000012345678
pkt_type	Type of the packet to be transmitted, as per the Bluetooth standard.
pkt_length	Length of the packet, in bytes, to be transmitted.
br_edr_mode	1 : basic rate 2 or 3 : enhanced_rate
rx_channel_index	Receive channel index, as per the Bluetooth standard.i.e, 0 to 78

Parameters	Value
tx_channel_index	Transmit channel index, as per the Bluetooth standard. i.e, 0 to 78
ink_type	0 : sco 1 : acl 2 : esco
scrambler_seed	Initial seed to be used for whitening. It should be set to '0' in order to disable whitening.
hopping type	0 : no hopping 1 : fixed hopping 2 : random hopping
ant_sel	2 : onchip antenna 3 : u.f.l
inter_pkt_gap	Number of slots to be skipped between two packets
pll_mode	0 : PLL_MODE0 1 : PLL_MODE1 2 : PLL_MODE2
rf_type	0 : External RF 1 : Internal RF
rf_chain	0 : WLAN_HP_CHAIN 1 : WLAN_LP_CHAIN 2 : BT_HP_CHAIN 3 : BT_LP_CHAIN
loop_back_mode	0 : Disable 1 : Enable

Statistics

This command enables the Master to get the performance statistics (Packet Contents) from the Wireless system so that the Master can compute BER over the received packets.

Command: {H <N>\n}

Example: H 8\n

Description:

N: Indicates the number of BER statistics to be obtained.

This command returns the following data (N*((1030*2) + 6) characters) -{<pkt_index[N*1:N*2]> <pkt_length[N*3:N*6]> <payload[N*7:N*(1030*2)]>}

Stop

This command enables the Master to stop the BT-Classic receive operation.

Command: {F 0\n}

Example: F 0\n

PER Examples

WIFI PER Burst mode Transmit

For transmitting a WIFI-11MBPS packet with packet length of 1000bytes at 2412MHZ and tx power of 10dbm on High power chain, configure the following parameters

pll_mode	0
rf_type	1
wireless_mode	0
enable_ppp	0
afe_type	1
Features	0
chan_idx	1
tx_mode	0
pkt_length	1000
Rate	6
tx_power	10

Observations in spectrum analyzer with above parameters:

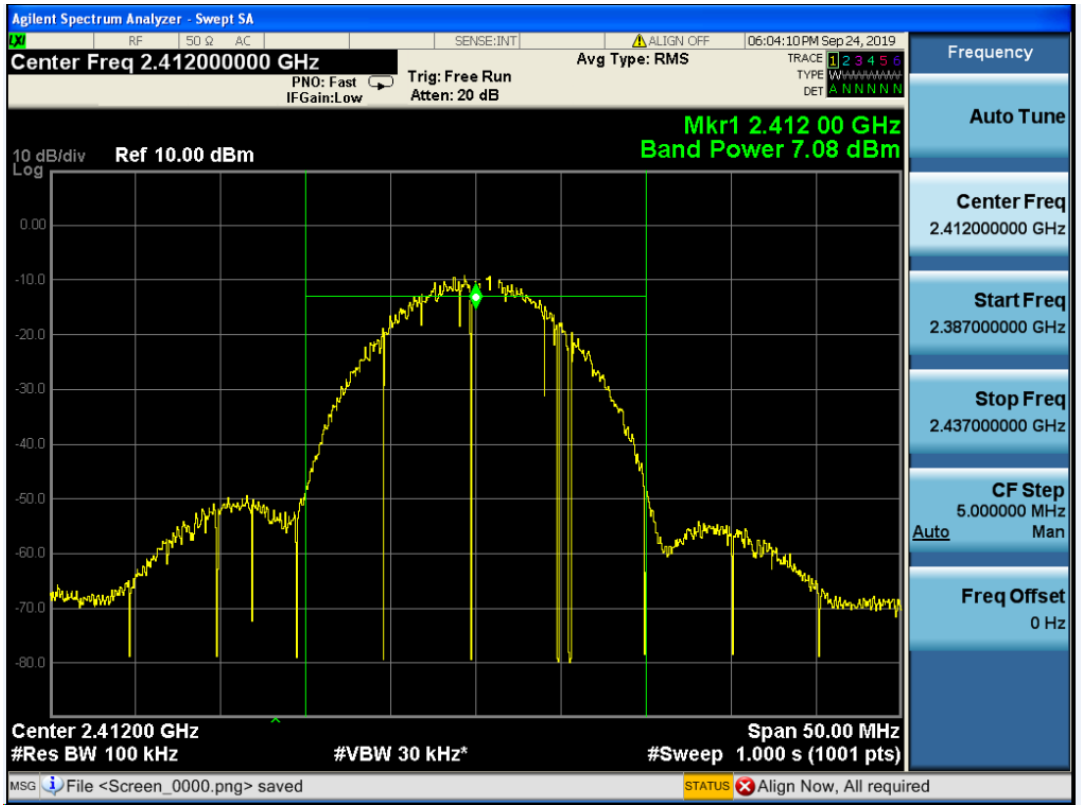


Figure 153: Observations in Spectrum Analyzer

For receiving a WIFI-1Mbps with Access Address of 0x71764129 on 2412MHz configure the following parameters

pll_mode	0
rf_type	1
wireless_mode	0

enable_ppp	0
afe_type	1
Features	0
chan_idx	1
tx_mode	0
pkt_length	1000
Rate	6
tx_power	10

BLE PER Burst mode Transmit

For transmitting a BLE-1Mbps packet with Access Address of 0x71764129 and packet length of 255bytes on 2480MHz and tx o/p power of 10dbm on the High power chain, configure the following parameters

Access_Addr	0x71764129
pkt_length	255
ble_phy_rate	1
rx_channel_index	39
tx_channel_index	39
scrambler_seed	0
no_of_packets	0
payload_type	6
le_channel_type	1
tx_power	10
tx_mode	0
hopping_type	0
ant_sel	2
inter_pkt_gap	0
pll_mode	0
rf_type	1
rf_chain	2

Observations in spectrum analyzer with above parameters:

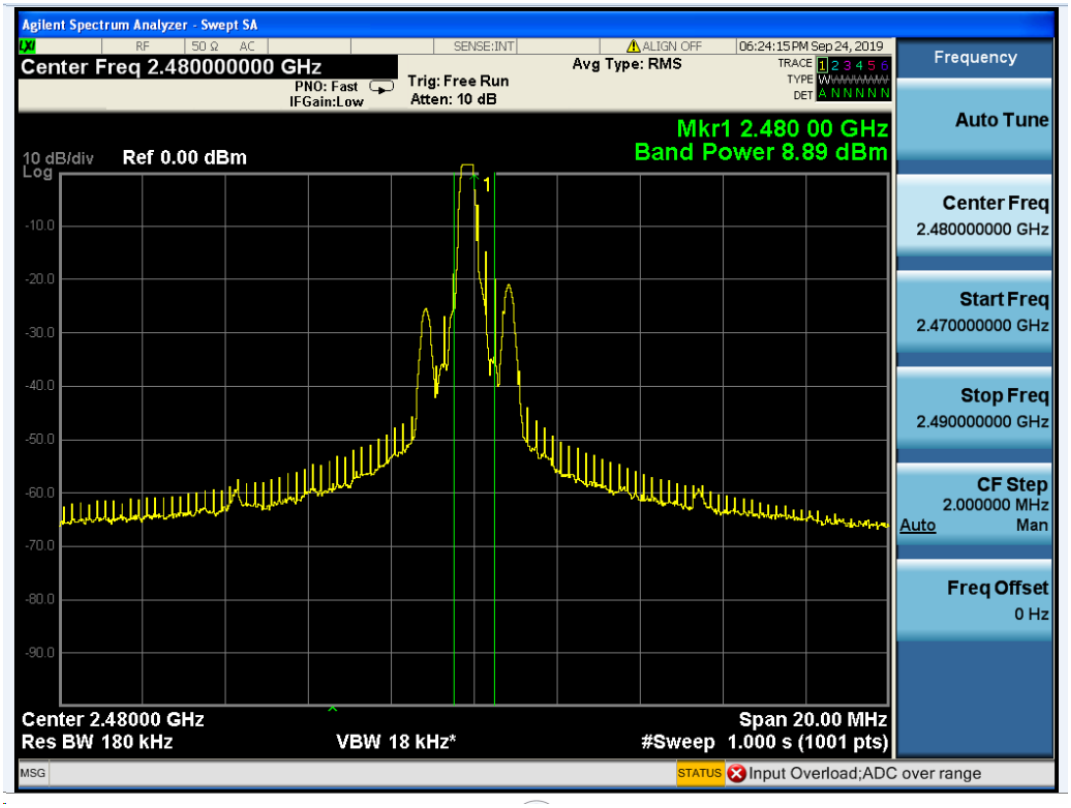


Figure 154: Observations in Spectrum Analyzer

For receiving a BLE-1Mbps with Access Address of 0x71764129 on 2480MHz frequency with the following parameters

Parameters	value
Access_Addr	71764129
data_length_indication	1
ble_phy_rate	1
rx_channel_index	39
tx_channel_index	39
scrambler_seed	0
le_channel_type	1
loop_back_mode	0
freq_hop_en	0
ant_sel	2
duty_cycling_en	0
pll_mode	0
rf_type	1
rf_chain	4

Note:

RF type should be always one for all the modules.

13 Wireless Features And Mechanisms

This document briefs about the wireless features and mechanisms like Power Save modes, Wake on Wireless and Wireless Firmware Upgrade.

13.1 Power Save Modes

Power Save

Description

This feature configures the Power Save mode of the module and can be issued at any time after Opermode command. Power Save is disabled by default.
There are three different modes of Power Save.

1. Power Save mode 0
2. Power Save mode 1
3. Power Save mode 2
4. Power Save mode 3
5. Power Save mode 8
6. Power Save mode 9

Note:

1. RS9116-WiSeConnect doesn't support power save modes while operating in AP or group owner mode.
2. Power Save modes 2 and 8 are not supported in USB / USB-CDC interface. Instead, they are supported in UART / SPI interfaces.
3. In SPI interface, when Power Save mode is enabled, after wakeup from sleep, host has to re-initialize SPI interface of the module.

Power Save Operations

The behavior of the module differs according to the power save mode it is configured.

The following terminology can be used in the below section in order to describe the functionality.

Protocol	Non Connected State	Connected State
WLAN	This mode is significant when module is not connected with any AP	This mode is significant when module is in associated state with AP
BT Classic	This mode is significant when module is in Idle (standby) state.	This mode is significant when module is in Connected sniff mode, Discoverable mode (ISCAN) and Connectable mode (PSCAN)
BLE	This mode is significant when module is in Idle (standby) state.	This mode is significant when module is in Advertising state, Scan state or Connected state.

Note:

1. In case of WLAN, wake up period will be calculated based on DTIM interval.
2. In case of BT-Classic, wake up period will be calculated based on inquiry scan interval in discoverable mode, page scan interval in connectable mode and sniff interval in connected mode.
3. In case of BLE, wake up period will be calculated based on advertise interval in advertising state, scan interval in scanning state and connection interval in connected state.
4. If incase BT/BLE wakeup period is lesser then the WLAN wake up period, the module will wakeup and servs BT/BLE and go back to the sleep again.

Power Save Mode 0

In this mode module is active and power save is disabled. It can be configured at any time while power save is enable with Power Save mode 2 or Power Save mode 8.

Power save Mode 1

Once the module is configured to power save mode 1, it wakes itself up periodically based upon the DTIM interval configured in connected AP. In power mode 1, only the RF of the module is in power save while SOC continues to work normally. After successful execution of command, confirmation is received in response. This command has to be given only when module is in connected state (with the AP).

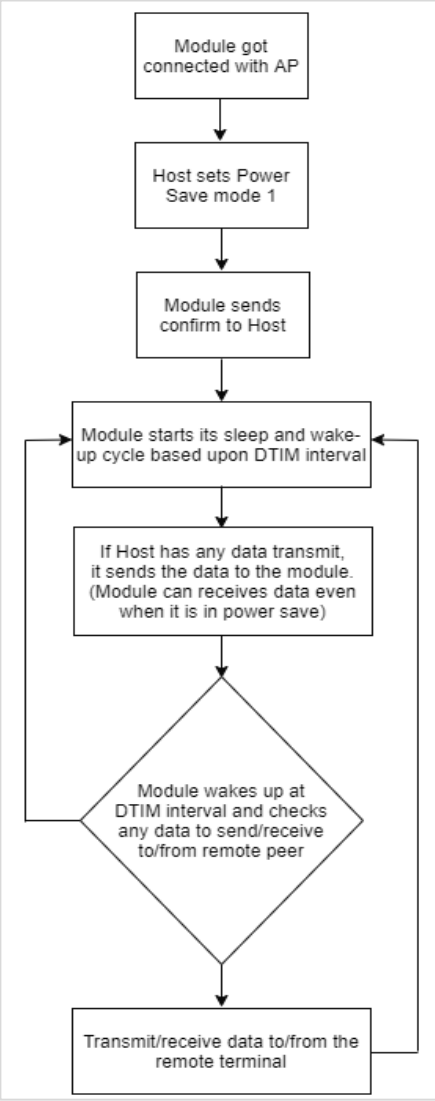


Figure 155: Setting Power Save Mode 1

After having configured the module to power save mode, the Host can issue subsequent commands. In power save mode 1 the module can receive data from host at any point of time but it can send/receive the data to/from remote terminal only when it is awake at DTIM intervals.

Power Save Mode 2

Once the module is configured to power save mode 2, it can be woken up either by the Host or periodically during its sleep-wakeup cycle. Power Save mode 2 can be either GPIO based or message based.

GPIO based mode:
In case of GPIO based mode, whenever host wants to send data to module, it gives wakeup indication by setting UULP GPIO #2. After wakeup, if the module is ready for data transfer, it sends wakeup indication to host by setting UULP GPIO #3. Host is required to wait until module gives wakeup indication before sending any data to the module. After the completion of data transfer, host can give sleep permission to module by resetting UULP GPIO #2. After recognizing sleep permission from host, module gives confirmation to host by resetting UULP GPIO #3 and again gets back to its sleep-wakeup cycle.
Module can send received packets or responses to host at any instant of time. No handshake is required on Rx path.

Message based mode:
In case of message based power save, both radio and SOC of the module are in power save mode. Module wakes up periodically upon every deep sleep duration and gives wakeup message ("WKP") to host. Module can not be woken up asynchronously. Every time module intends to go to sleep it sends a sleep request message ("SLP") to the host and expects host to send the ack ("ACK") message. Host either sends ack ("ACK") or any other pending message. But once ack ("ACK") is sent, Host should not send any other message unless next wakeup message from module is received.
Module shall not go into complete power-save state if ack is not received from host for given sleep message. Module can send received packets or responses to host at any instant of time. No handshake is required on Rx path.

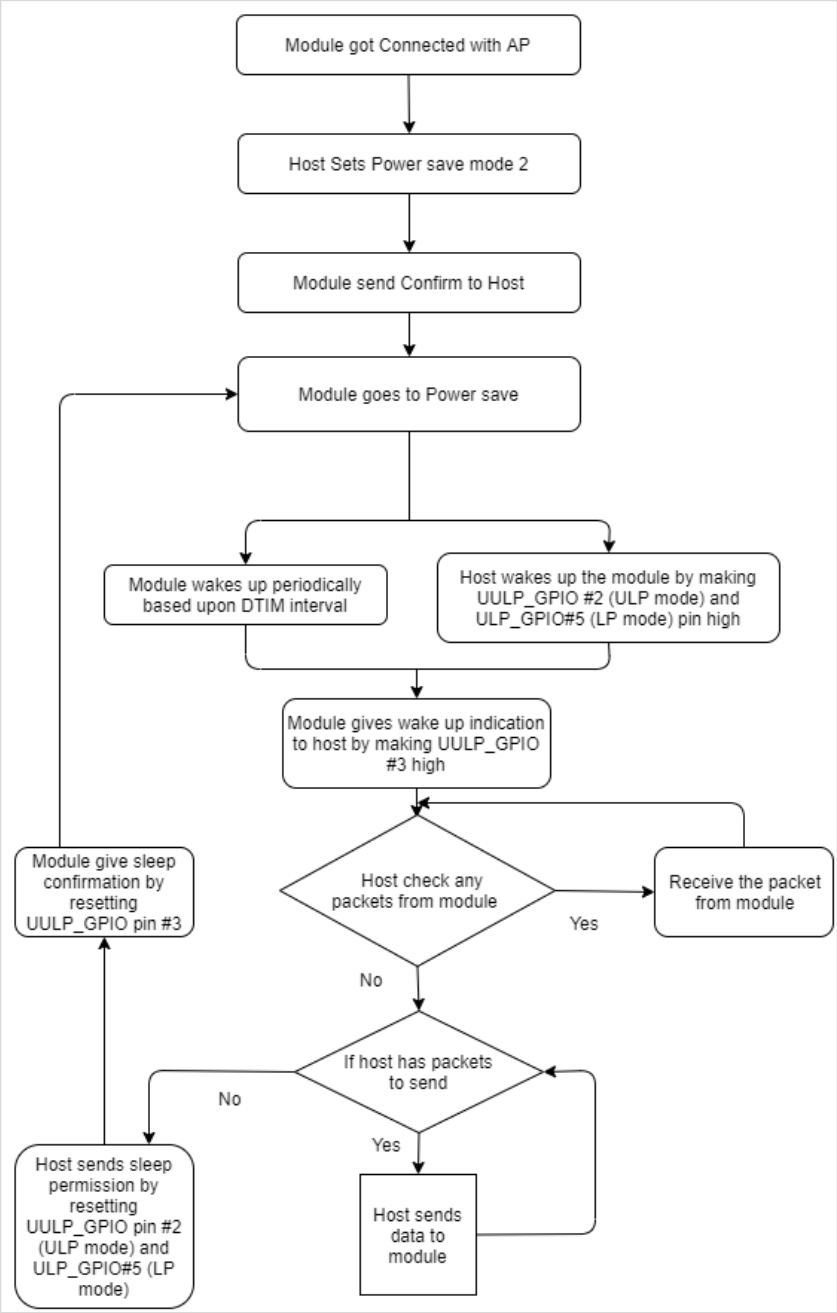


Figure 156: Power Save Mode 2

Table 3 Messages from module in Power Save mode 2

Command Description	Binary Mode
"WKP"	0xDD
"SLP"	0xDE

Table 4 Message from host in Power Save mode 2

Command Description	Binary Mode
"ACK"	0xDE

Usage in BT-Classic Mode:
In Classic, Power Save mode 2 can be used during Discoverable / Connectable / Connected sniff states.

- **Discoverable Mode State:** In this state, module is awake during Inquiry Scan window duration and sleeps till Inquiry Scan interval.

Default Inquiry scan window value is 11.25 msec, and Inquiry scan interval is 320 msec.

- **Connectable Mode State:** In this state, module is awake during Page Scan window duration and sleeps till Page Scan interval.

Default Page scan window value is 11.25 msec, and Page scan interval is 320 msec.

- **Connected Sniff State:** While the module is in connected state as a master or slave, once the module has configured with Power Save mode 2 with GPIO based or message based then the module will goes into power save mode in connected state. This will be work when the module and peer device supports sniff feature. And module should configure with sniff command after a successful connection, before configure with power save command.

Module will goes into power save after serving a sniff anchor point, and wakes up before starting a sniff anchor point. Sniff connection anchor point may varies based on the remote device t_sniff value.

Usage in BT-LE Mode:
In LE, Power Save mode 2 can be used during Advertise / Scan / Connected states.

- **Advertise State:** In this state, module is awake during advertising event duration and sleeps till Advertising interval.
- **Scan State:** In this state, module is awake during Scanning window and sleeps till Scanning Interval. Default scan window is 50 msec, default scan interval is 160 msec.
- **Connected state:** In this state, module wakes up for every connection interval. Default connection interval is 200 msec which was configurable.

Power Save mode 3

Power Mode 3 is message based power save. In Power Mode 3 like Power mode 2 both radio and SOC of RS9116-WiSeConnect are in power save mode. This mode is significant when module is in associated state with AP. Module wakes up periodically upon every DTIM and gives wakeup message ("WKP") to host. Module can not be woken up asynchronously. Every time module intends to go to sleep it sends a sleep request message ("SLP") to the host and expects host to send the ack message. Host either send ack ("ACK") or any other pending message. But once ack is sent, Host should not send any other message unless next wakeup message from module is received.
Module shall not go into complete power-save state if ack is not received from host for given sleep message. Module can send received packets or responses to host at any instant of time. No handshake is required on Rx path.

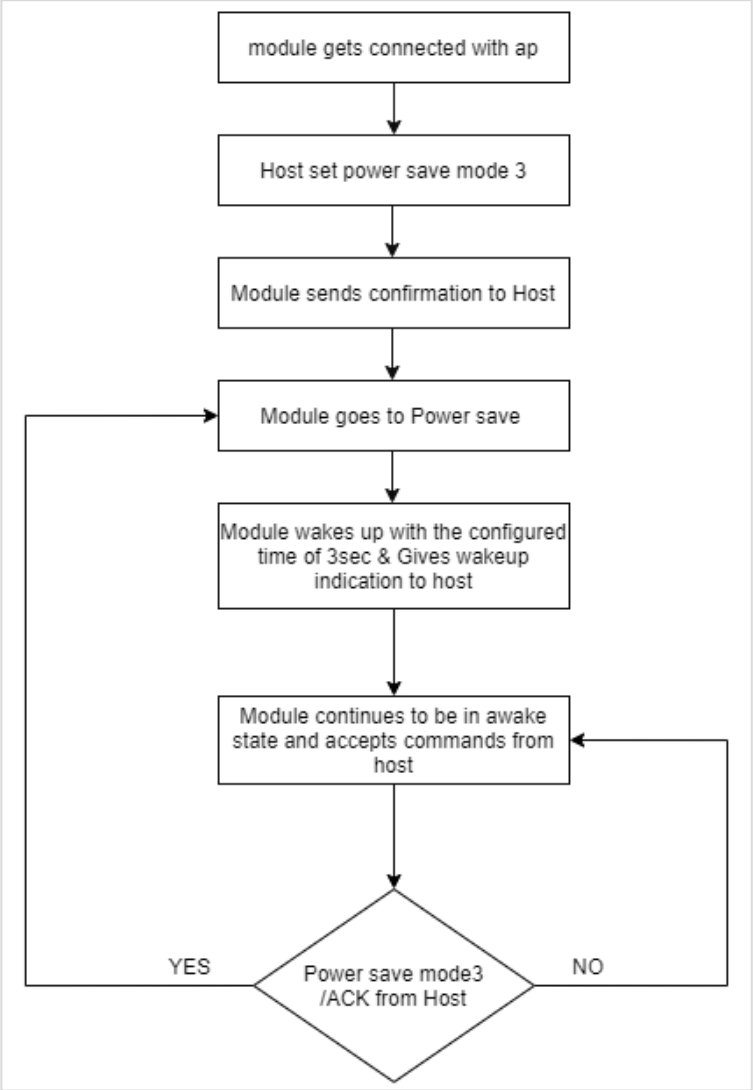


Figure 157: Power Save Mode 3

Power Save mode 8

In Power save mode 8, both RF and SOC of the module are in complete power save mode. This mode is significant only when module is in standby mode. Power mode 8 is GPIO based/message based. Power Save mode 8 can be either GPIO based or message based.

GPIO based mode:
In case of GPIO based, host can wakeup the module from power save by making UULP-GPIO #2 high.
Once the module wakes up, it continues to be in wakeup state until it gets power mode 8 commands from host.

Message based mode:
In case of message based, module goes to sleep immediately after issuing power save command and wakes up after 3sec. Upon wakeup, module sends a wakeup message "WKP" to the host and expects host to give ack "ACK" before it goes into next sleep cycle. Host can either send ack or any other messages. But once ACK is sent, no other packet should be sent before receiving next wakeup message.

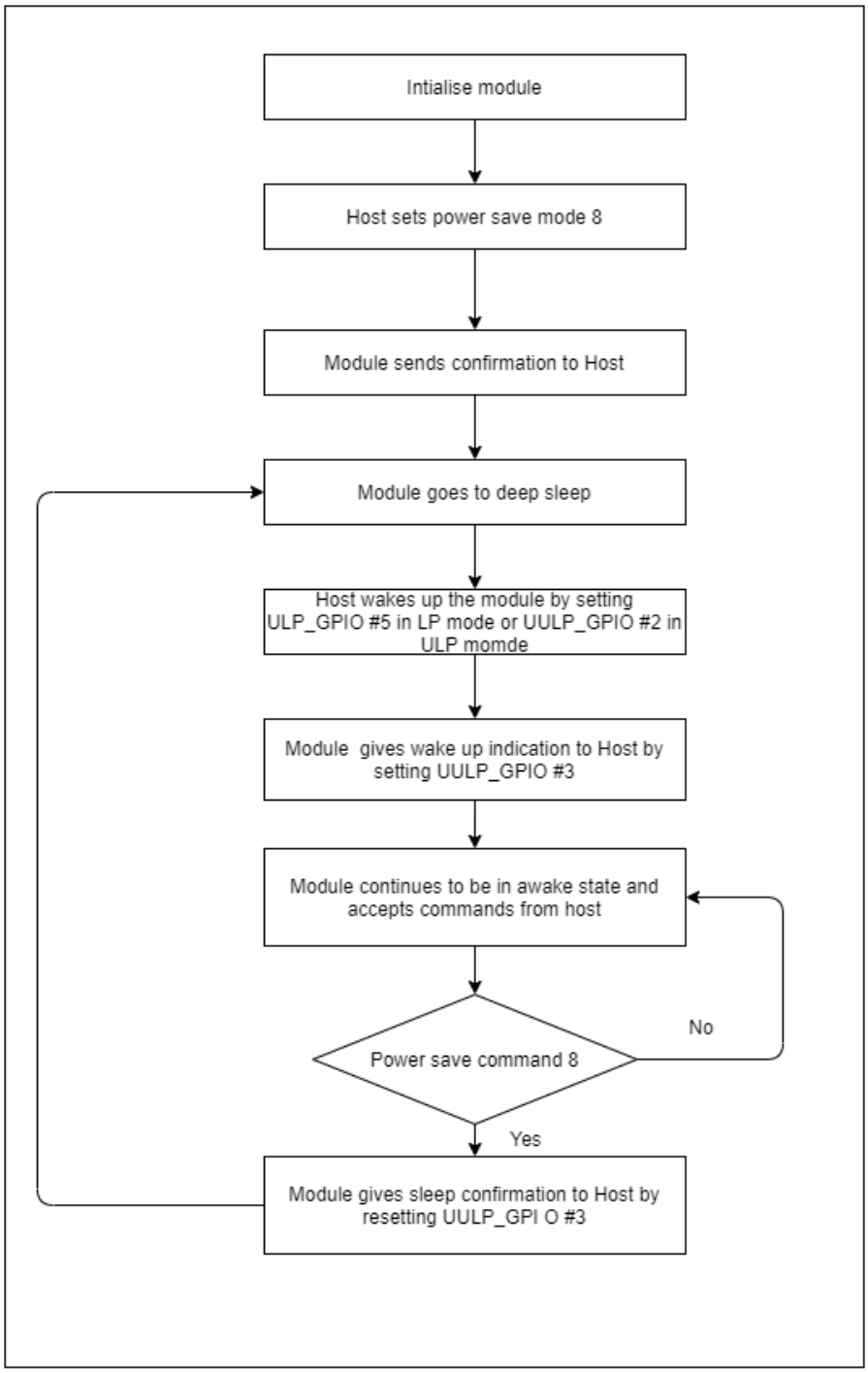


Figure 158: Power Save Mode 8

Table 5 Messages from module in Power Save mode 8

Command Description	Binary Mode
"WKP"	0xDD

Table 6 Message from host in Power Save mode 8

Command Description	Binary Mode
"ACK"	0xDE

Note:

- In BT Classic/LE, Power Save mode 8 can be used in Standby (idle) state.

Example: Suppose if Power Save is enabled in advertising state, to move to Scanning state, first Power Save disable command need to be issue before giving Scan command.

- For Page scan, Inquiry scan, sniff parameters related information, please verify Bluetooth protocol specification document.
- When the module is configured in a co-ex mode and WLAN is in INIT_DONE state, powersave mode 2 & 3 are valid after association in the WLAN. Where as in BT & BLE alone modes, it will enter into power save mode (2&3) in all states (except in standby state).
- Power save disable command has to be given before changing the state from standby to remaining states and vise-versa.

Power save mode 9

In Power Mode 9 both Radio and SOC of RS9116-WiSeConnect are in complete power save mode. This mode is significant when module is not connected with any AP. Once power mode 9 command is given, the module goes to sleep immediately and wakes up after sleep duration configurable by host by set sleep timer command. If host does not sets any default time, then the module wakes up in 3sec by default. Upon wakeup module sends a wakeup message to the host and expects host to give ack before it goes into next sleep cycle. Host either send ack or any other messages but once ACK is sent no other packet should be sent before receiving next wakeup message. When ulp_mode_enable is set to '2', after waking up from sleep, the module sends following message to host when RAM retention is not enabled. After receiving this message, host needs to start giving commands from beginning (opermode) as module's state is not retained.

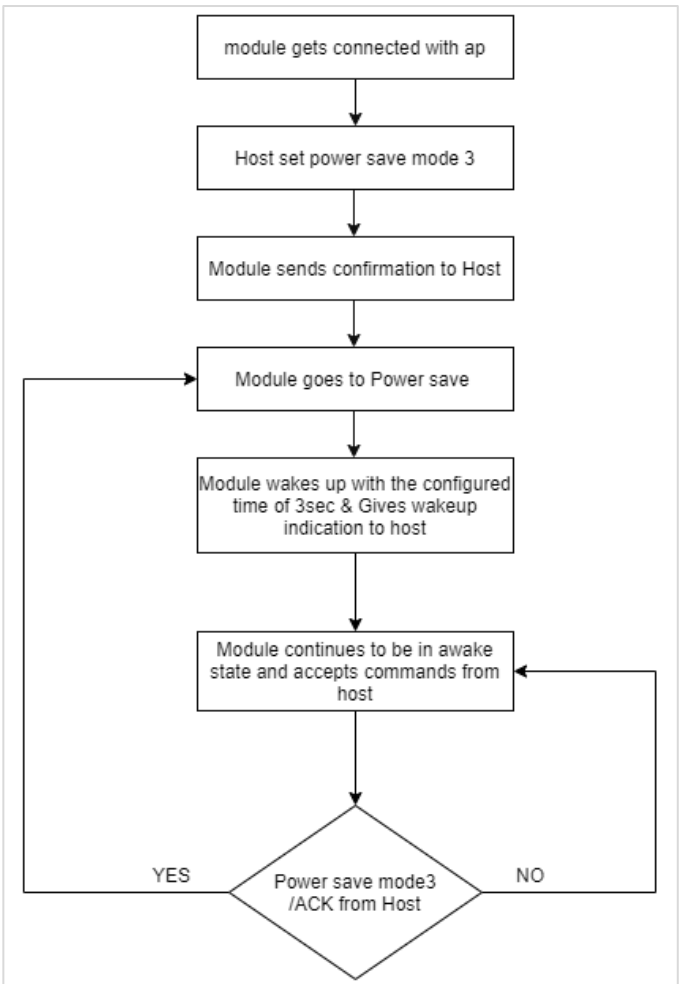


Figure 159: Power Save Mode 9

13.2 SAPI Wake On Wireless

RS9116 Module wants to send packets to host with wake on wireless mode.It has two method as below ,

Active High Interrupt Mode

If BIT(16) of config_feature_bit_map is enable in opermode command then active high interrupt mode is enable.

Whenever RS9116 Module wants to send packets to host, it will assert WAKEUP_FROM_DEV pin.

In UART mode following is the sequence:

1. RS9116 Module assert WAKEUP_FROM_DEV pin when data is pending from module and polls for ack (HOST_WAKEUP_INDICATION pin to be high) before starting the transfer.
2. After recognizing WAKEUP_FROM_DEV pin asserted, host should ack the request from module by asserting HOST_WAKEUP_INDICATION pin and poll WAKEUP_FROM_DEV pin for module confirmation (WAKEUP_FROM_DEV is low).
3. After recognizing ack (HOST_WAKEUP_INDICATION pin high) from host, module will de-assert WAKEUP_FROM_DEV pin and start transfer.
4. Once WAKEUP_FROM_DEV is de-asserted, host should de-assert HOST_WAKEUP_INDICATION pin and receive the packet from module.

Refer the below flow chart to receive data from RS9116 Module in wake on wireless mode of active high interrupt mode :

Active Low Interrupt Mode

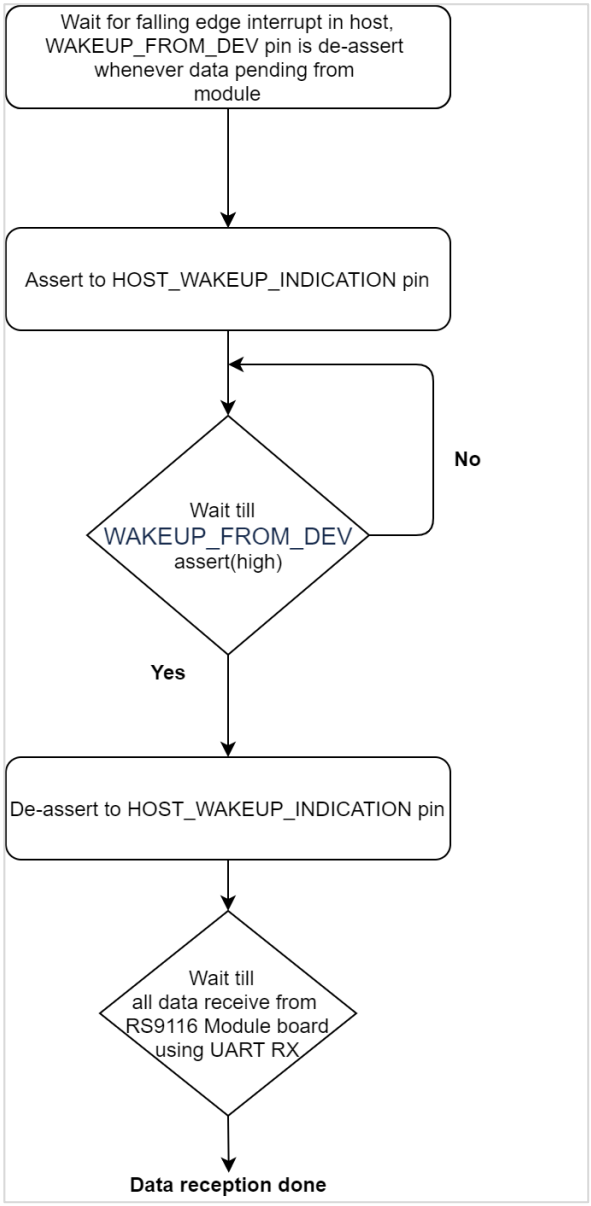
If BIT(16) of config_feature_bit_map is not enable in opermode command then active low interrupt mode is enable.

Whenever RS9116 Module wants to send packets to host, it will de-assert WAKEUP_FROM_DEV pin.

In UART mode following is the sequence:

1. RS9116 Module de-assert WAKEUP_FROM_DEV pin when data is pending from module and polls for ack (HOST_WAKEUP_INDICATION pin to be high) before starting the transfer.
2. After recognizing WAKEUP_FROM_DEV pin de-asserted, host should ack the request from module by asserting HOST_WAKEUP_INDICATION pin and poll WAKEUP_FROM_DEV pin for module confirmation (WAKEUP_FROM_DEV pin is high).
3. After recognizing ack (HOST_WAKEUP_INDICATION pin high) from host, module will assert WAKEUP_FROM_DEV pin and start transfer.
4. Once WAKEUP_FROM_DEV is asserted, host should de-assert HOST_WAKEUP_INDICATION pin and receive the packet from module.

Refer the below flow chart to receive data from RS9116 Module in wake on wireless mode of active low interrupt mode:



Note:

Since UART is Asynchronous interface, Polling mechanism is suggested.

Polling for HOST_WAKEUP_INDICATION is valid only if BIT(11) in custom feature bit map of opermode command is enabled in UART mode.

If BIT(11) of custom_feature_bit_map is not enabled in opermode command.The WOW feature in UART follows the below sequence like other interfaces

For Other host interfaces like SDIO / SPI / USB / USB-CDC:

Whenever RS9116W want to send packets to host,

1. RS9116W will drive WAKEUP_FROM_DEV pin low ('0').
2. Host should wakeup and take the packet from RS9116W.
3. Once the packet is received by host, RS9116W will drive WAKEUP_FROM_DEV pin high ('1').

Note:

BIT(11) of custom_feature_bit_map is not valid in SPI/SDIO/USB/USB-CDC interfaces.

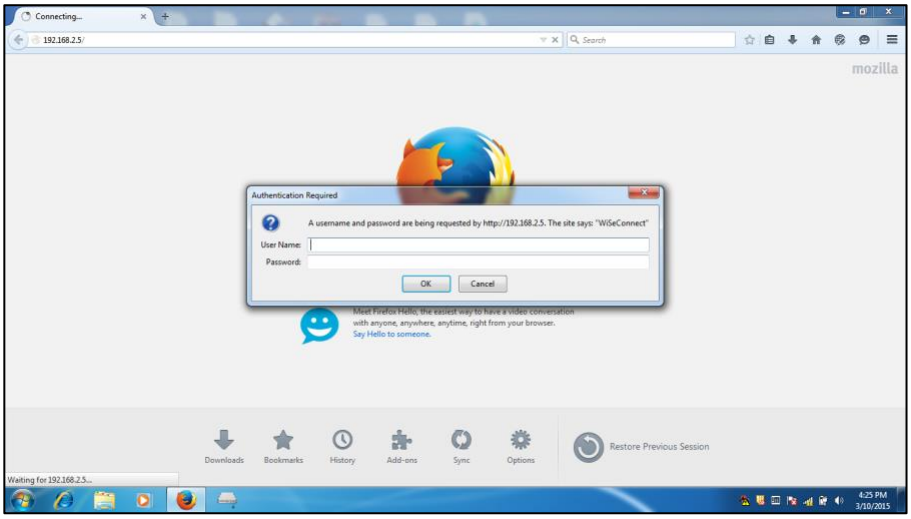
13.3 SAPI Wireless Firmware Upgrade

The firmware of the module can be upgraded in different ways shown below

- Wirelessly Through Web Server
- Using UDB-CDC
- Using Host
- OTA(Over The Air)

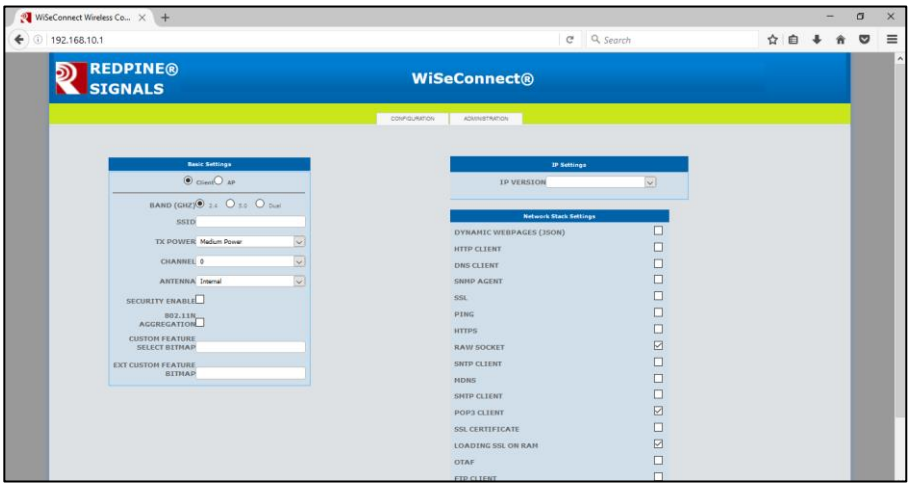
Wirelessly Through Web Server:

To upgrade the firmware wirelessly user has to open configuration page. In the given example, module is in WLAN client mode. Some other host and module connected to an AP. Module got IP 192.168.2.5.When opened the module's webpage on the other host, it asks for the login credentials. The credentials for Username should be given as "redpine" and password should be given as "admin" to open the modules configuration page.

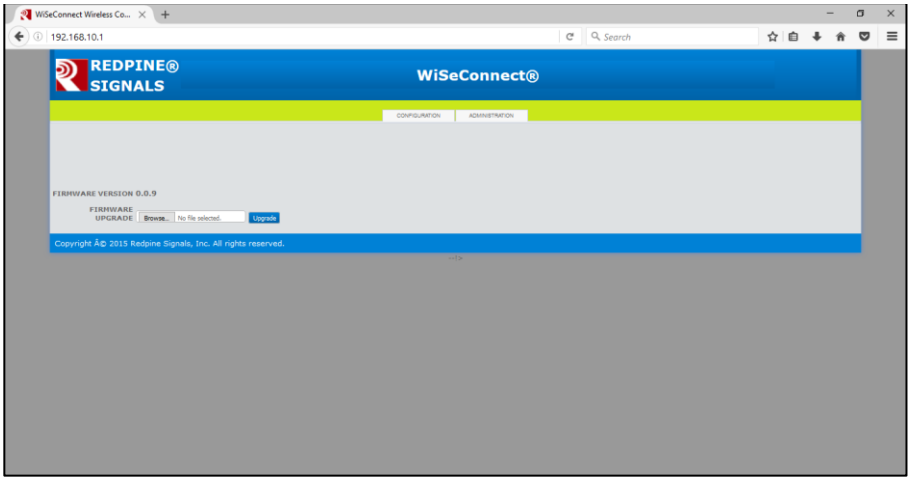


After giving the login credentials, the module's configuration page is opened as shown in the figure below.

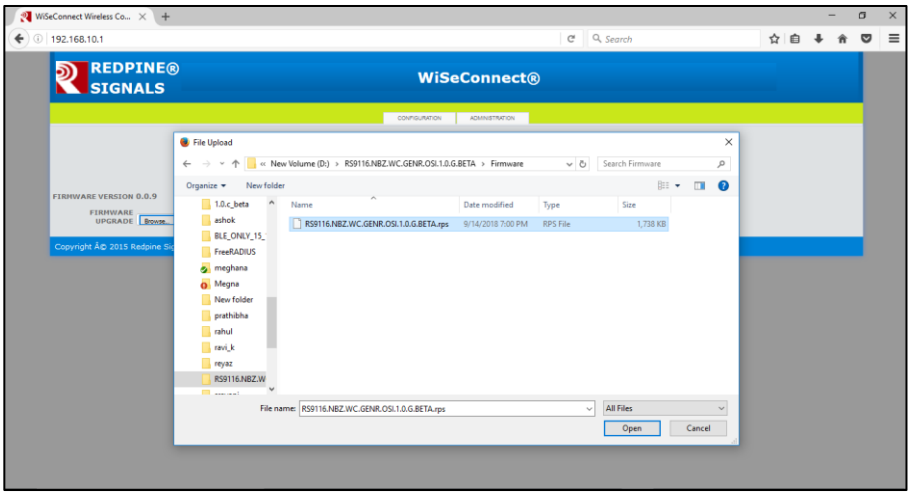
Note:
'Authentication Required' pop up window will only appear if BIT[23] in Custom feature bitmap is enabled.



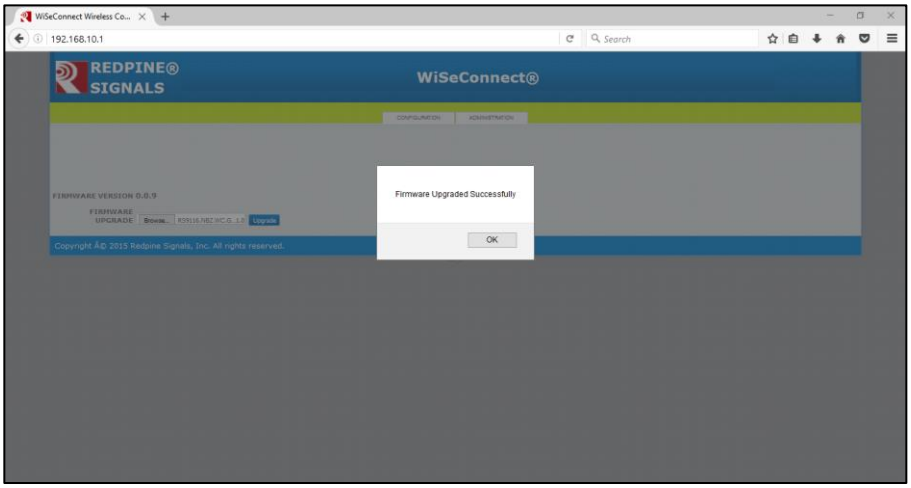
Click on ADMINISTRATION button to go to the wireless firmware up-gradation page.



Browse for the rps file (RS9116.WC.GEN.OSI.x_x_x.rps) on the host to upgrade the module firmware, and click on UPGRADE, as shown in the below screen shot.



Once the remote peer has pressed upgrade button on the webpage, if module is connected through UART or USB-CDC interface host will get asynchronous notification as"AT+RSI_FWUPREQ". So host has to issue AT+RSI_FWUPOK. For SPI and USB interfaces, an asynchronous message with response id 0x59 is sent to host and then host has to respond with request id 0x59 (through API) . If host failed to reply within specified timeout(~20 seconds), request will expire and upgradation process terminates.
After the firmware upgraded successful, one popup will come on remote peer screen to intimate the process complete . Similarly asynchronous success message(for UART/USB-CDC "AT+RSI_FWUPSUCCESS" and for SPI/USB response id 0x5A) will be forward to host connected with the module.



Note:

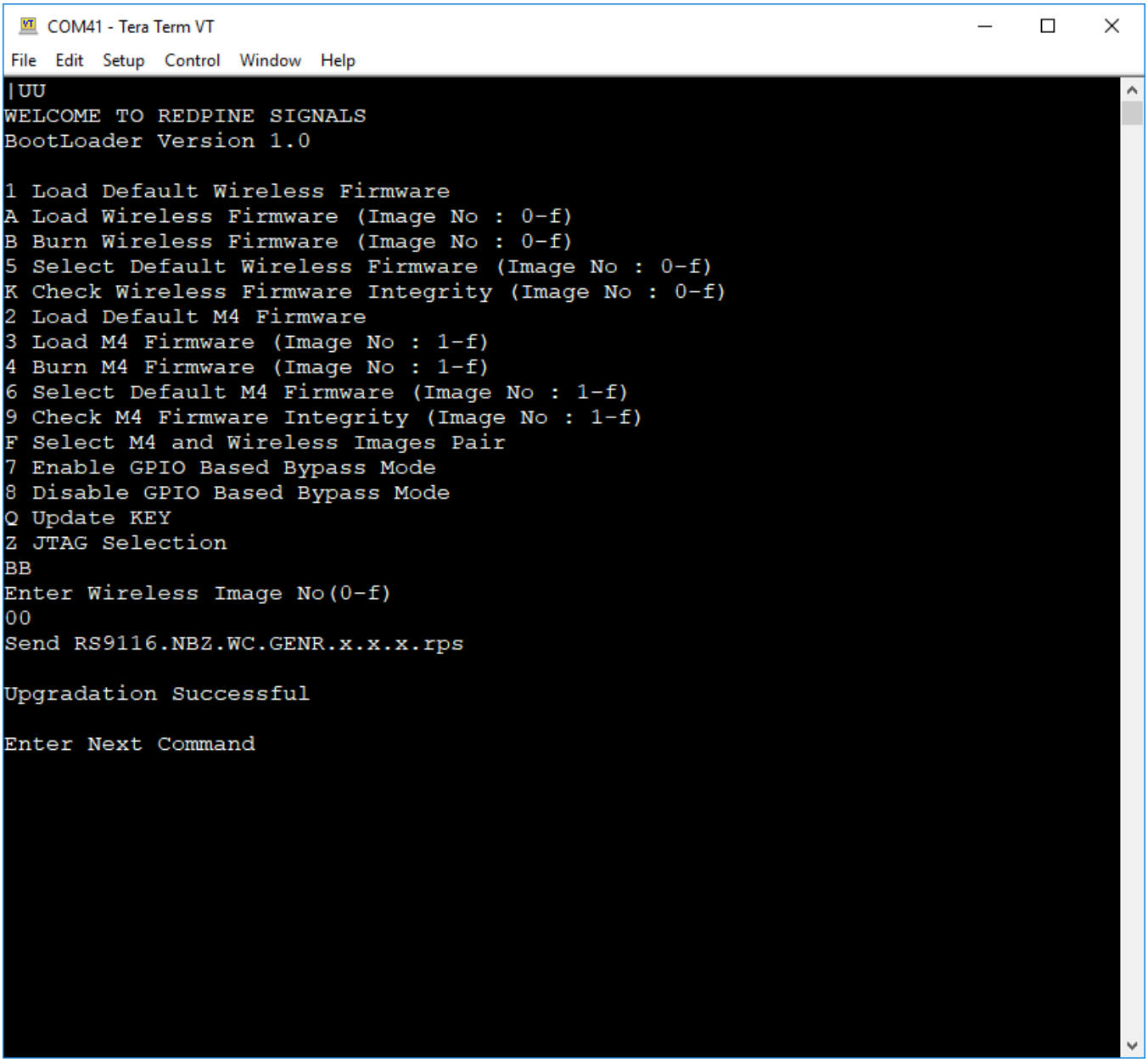
1. Wireless firmware upgradation is supported only for the latest versions of Firefox and Google chrome.
2. When user clicks on upgrade button, module starts erasing flash for storing image. This may take few seconds, then upgradation automatically starts.
3. After wireless firmware upgrade, after reboot user needs to wait for few minutes (~ 1.5 minutes) so that bootloader will copy upgraded image into actual flash location.

Using UDB-CDC:

Using UDB-CDC via boot loader we can upgrade the firmware.

Steps:

1. Power the device off and switch ISP to ON.
2. Plug-in a USB cable to USB-CDC and another cable to POWER, in the same order. The device should appear on the Windows PC as a COM port.
3. Open Tera Term and select the COM Port used by the RS14100.
4. Enter the pipe key |. Tera Term should echo back a U. Enter a capital U. This will make the bootloader menu appear. This process is called Auto Baud Rate Detection (ABRD) and is used to set the baud rate of the RS14100.
5. Choose option B and select image 0.
6. Go to File-> Transfer-> Kermit-> Send.
7. Select the image **RS14100.NB0.WM.GENR.X.Y.Z.rps** in <Package>\NWP\Firmware. Tera Term will begin sending this image.
8. RS14100 will send the message **"Upgradation Successful"** once the flashing process is completed.
9. Turn off the device and switch ISP back to OFF. Your device is now running the latest wireless firmware.



Using Host:

Host application demonstrates how to upgrade new firmware to Silicon Labs device using remote TCP server. In this application, the device connects to access point and establishes TCP client connection with TCP server opened on remote peer. After successful TCP connection, application sends the firmware file request to remote TCP server and server responds with Firmware file and waits for the next firmware file request. Once firmware file receives from the TCP server, application loads the firmware file into device using firmware upgrade API and gets next firmware file from TCP server. After successful firmware upgrade, firmware upgrade API returns 0x03 response.

For execution steps please refer below page: Firmware Upgrade.

OTA(Over The Air):

This application demonstrates how to upgrade new firmware to Silicon Labs using remote TCP server.

In this application Silicon Labs connects to Access Point and using OTAF command establishes TCP client connection with TCP server opened on remote peer. After successful TCP connection, module sends the firmware file request to remote TCP server and server responds with Firmware file and waits for the next firmware file request. Once firmware file receives from the TCP server, Module loads the firmware file into on to the modules flash. After successful firmware upgrade, OTAF api returns success response.

For execution steps please refer below page: OTAF

14 Revision Record

S.No.	Version number	Date	Change
1	1.0	October 2017	Initial version
2	1.1	June 2018	Added New API's
3	1.2	July 2018	Added rsi_wlan_pmk_generate API
4	1.3	July 2018	Added rsi_setsockopt API
5	1.4	August 2018	Added rsi_accept_async API
6	1.5	October 2018	Added PMK feature in **station ping** Application Added a2dp burst mode command and more data req event A2DP_BURST_MODE should be 1 Added BT A2DP Sink Example
7	1.6	April 2019	1. Added Send Bulk Data Example 2. Remove the Zigbee 3. Updated switch_m4_frequency api usage in wlan , wlan_ble and wlan_bt throughput examples. 4. Added AWS cloud examples 5. Modified the WLAN-AP, BT & BLE Bridge & WLAN-AP, BT or BLE Bridge examples with corrected port numbers. 6. Modified BT SPP Master Example, Corrected the diagram of application setup 7. Modified BT TestMode example, Corrected the wrong file name mentioned & Deleted the RSI_SEL_ANTENNA Macro which is not being used in application.
8	1.7	May 2019	1. Added BT HID Device Example 2. Added WLAN Asychronous Statistics Example 3. Added RAM dump example 4. Added WLAN Concurrent BLE 5. Added Chip manufacturing utility example
9	1.8	September 2019	1. Added select socket API 2. Removed rsi_bl_module_power_off and rsi_bl_module_power_on APIs 3. Added BT IAP2 example
10	1.10	September 2019	1. Added scan results provision to host example. 2. Added TCP server socket close wait example
11	1.11	October 2019	1. Updated prototypes of various SAPIs
12	1.12	February 2020	Added SSL client example with certificates loading in to RAM
13	1.13	February 2020	Updated SSL client example with certificates loading in to RAM
14	1.14	May 2020	1. Updated scan results, tcp ip bypass, transmit test and wlan async stats applications.
15	1.15	May 2020	1. Updated HTTP_Client example user guide for HTTP_PUT response
16	1.16	May 2020	1. Added the note in provisioning demo guide, which specifies the host interface as SPI.

S.No.	Version number	Date	Change
17	1.17	May 2020	1. Copied the BLE secure connections and test mode examples from the master page to the slave page. 2. Modifying the WLAN+BLE provisioning application as per the review comments.
18	1.18	May 2020	Updated documents with BT_GLOBAL_BUFF_LEN from 10000 to 15000 as per the examples.
19	1.19	May 2020	Removed WLAN_ZIGBEE and ZIGBEE Section
20	1.20	June 2020	Added User config gain table example
21	1.21	July 2020	1. Added flag to enable HTTP status code for HTTP examples 2. Added A2DP source & sink examples in BT Classic examples.
22	2.0	Sep 2020	1. Added WLAN_BLE Power save example. 2. Added WLAN_BT Power save example. 3. Added BT Power save example. 4. Added list of examples in each section. 5. **Changed the Document name to RS9116W Guide for SAPI Application Examples.** 6. Added following new sections o Cloud User Manual in Appendix A. o Wireless Features and Mechanisms - Power Save modes - SAPI Wake on Wireless - SAPI Wireless Firmware Upgrade 7. Added 'SAPI Example Directory Structure' section. 8. Removed old setup diagrams and added IO diagrams in all the examples. 9. Removed 'Manufacturing Util' example. 10. Changed Example names w.r.t Examples names in release package. 11. Removed 'Store Configuration Profile' example. 12. Added 'Auto Join' example
23	2.1	Oct 2020	Added info for rejoin handling in wlan bt ble dual role example

15 Appendix A: Cloud User Manual

ALIBABA CLOUD

Alibaba IoT account is required to create resources required to send, receive, and process MQTT messages from devices using Alibaba IoT. Visit the page https://www.alibabacloud.com/ and create an account.

Create a Product

The first step when you start using IoT Platform is to create products. A product is a collection of devices that typically have the same features. For example, a product can refer to a product model and a device is then a specific device of the product model.

IoT Platform supports two editions of products: Basic Edition and Pro Edition. This document introduces how to create a Pro Edition product.

- Log on to the IoT Platform console : https://iot.console.aliyun.com/product
- In the left-side navigation pane, click Devices > Product, and then click Create Product.
- Select **Pro Edition** and click **Next** as shown in Figure **Create Product - Step_1**.
- Enter required information like product name and product information as shown in Figure **Create Product - Step_2** and click **OK.**

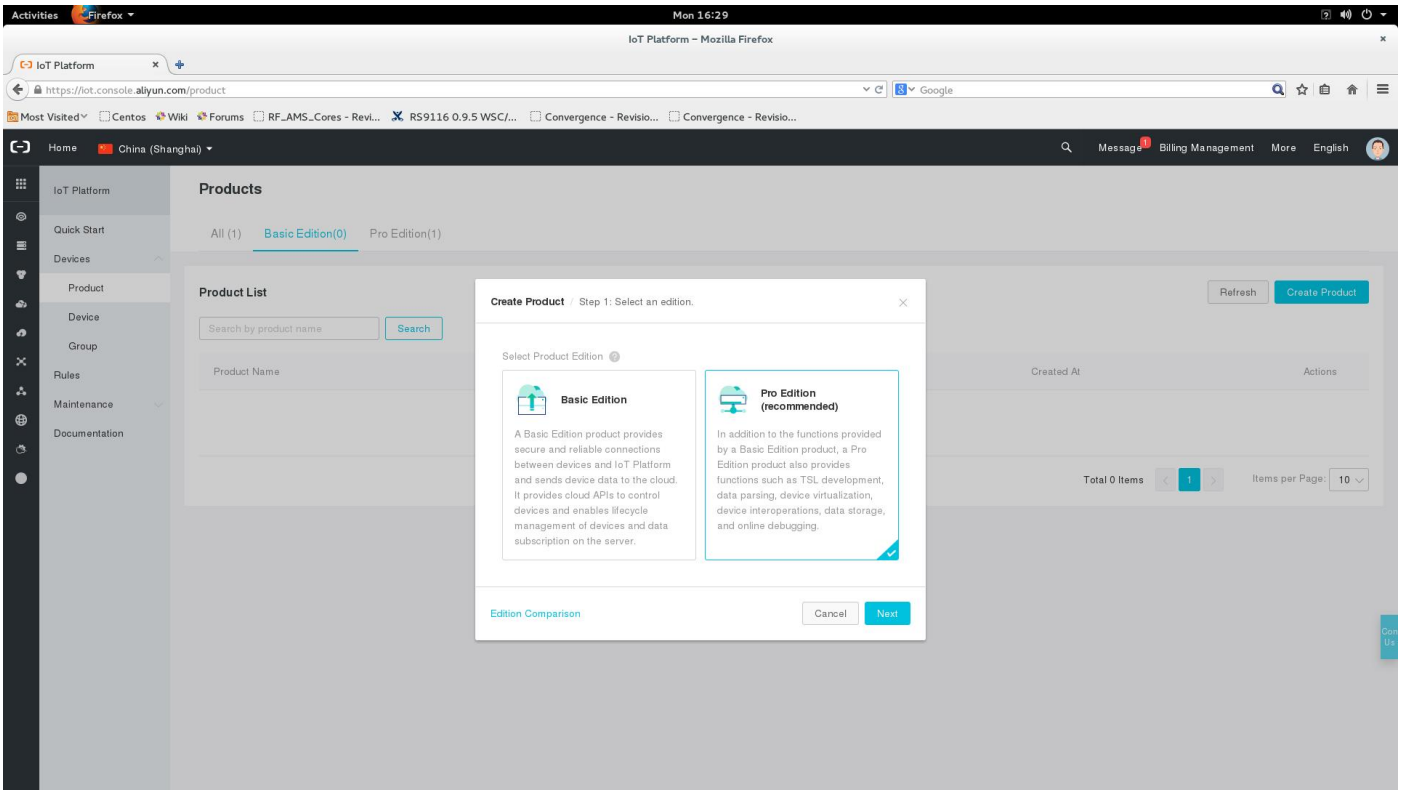


Figure 160 Create Product - Step_1

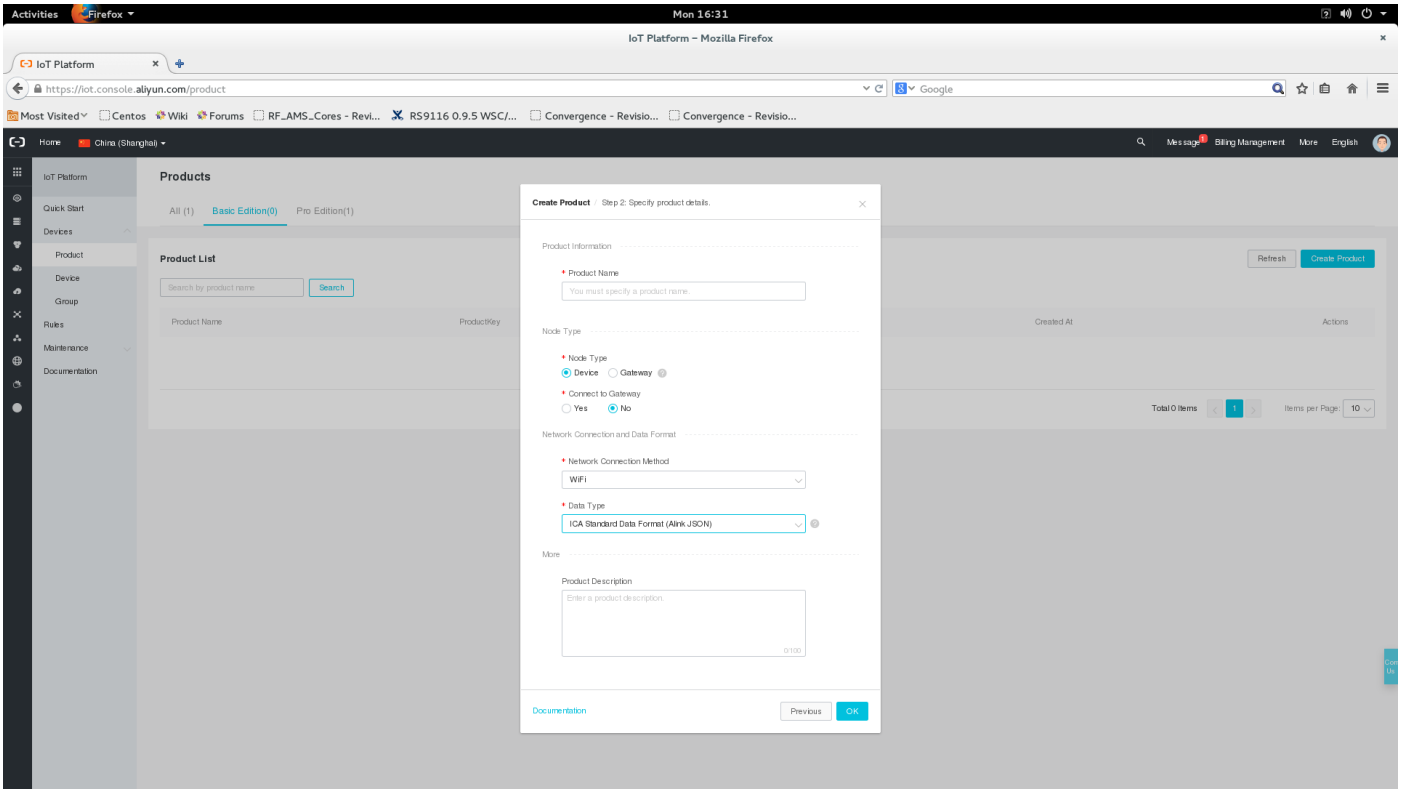


Figure 161 Create Product - Step_2

The parameters of Step_2 are described as follows:

Parameter	Description
Product Name	The name of the product that you want to create. The product name must be unique within the account. For example, you can enter the product model as the product name. A product name is 4 to 30 characters in length, and can contain English letters, digits and underscores.
Node Type	Options are Device and Gateway. • Device: Indicates that devices of this product cannot be mounted with sub-devices. This kind of devices can connect to IoT Platform directly or as sub-devices of gateway devices. • Gateway: Indicates that devices of this product connect to IoT Platform directly and can be mounted with sub-devices. A gateway can manage sub-devices, maintain topological relationships with sub-devices, and synchronize topological relationships to IoT Platform.
Network connection Method	Select a network connection method for the devices: • WiFi • Cellular (2g/3g/4G) • Ethernet • Other
Data type	Select a format in which devices exchange data with IoT Platform. Options are ICA Standard Data Format (Alink JSON) and Do not parse/Custom. • ICA Standard Data Format (Alink JSON): The standard data format defined by IoT Platform for device and IoT Platform communication. • Do not parse/Custom: If you want to customize the serial data format, select Do not parse/Custom.
Product Description	Describe the product information. You can enter up to 100 characters.

After the product is created successfully, you are automatically redirected to the Products page. You can then view or edit the product information.

Create a Device

A product is a collection of devices. After you have created products, you can create devices of the product models. You can create one device or multiple devices at a time. This document introduces how to create a single device.

- Log on to the IoT Platform console : https://iot.console.aliyun.com/product
- n the left-side navigation pane, click Devices > Device, and then click Add Device
- Select a product that you have created. The device to be created will be assigned with the features of the selected product.
- (Optional) Enter a name for the device. If you do not enter a device name for the device, the system will automatically generate one for the device as shown in Figure below.

> **Note**:
>
> 'DeviceName' must be unique within a product. It is used as a device identifier when the device communicates with IoT Platform.

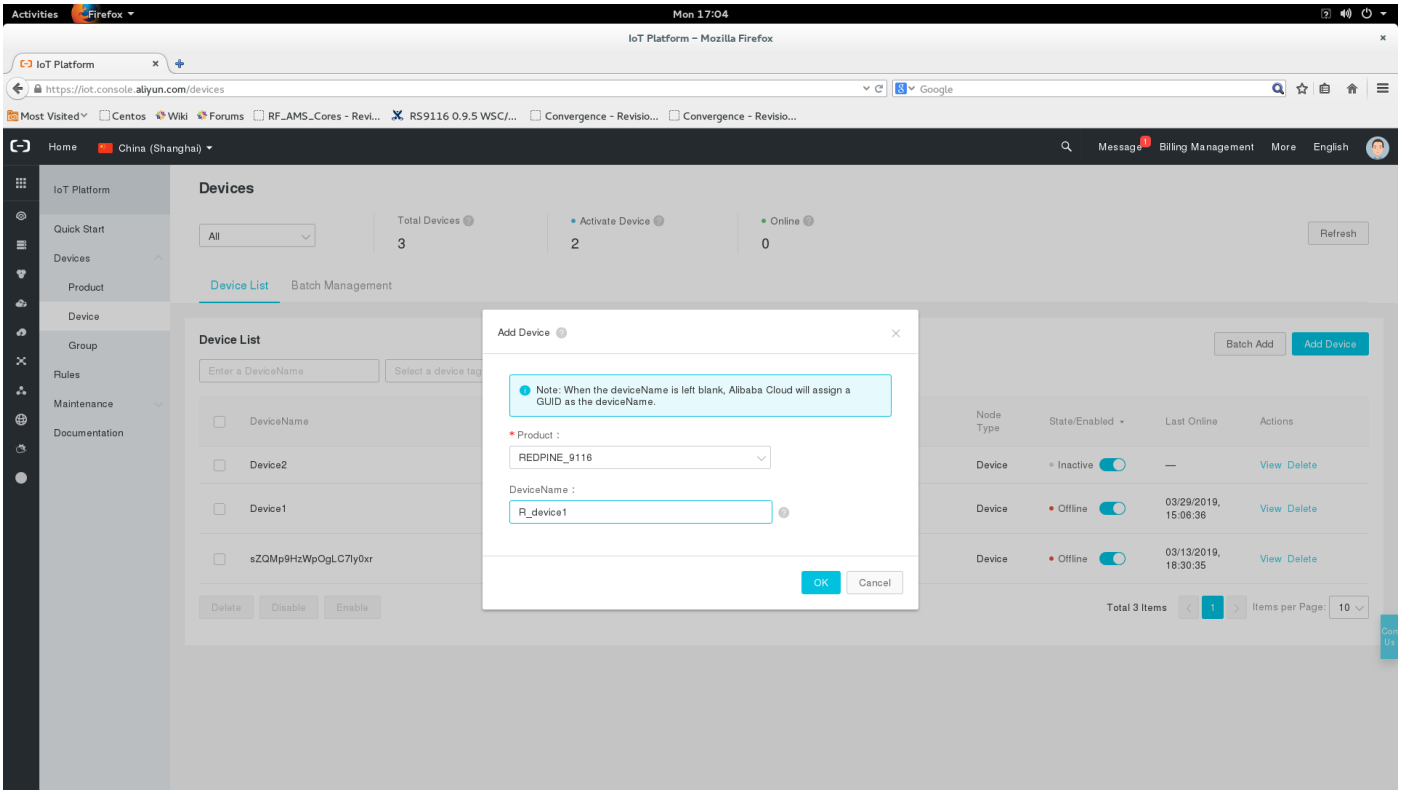


Figure 162 Create a Device

- Click OK to create the device.

  After the device has been successfully created, the View Device Certificate box is displayed. There, you can view and copy the device certificate information. A device certificate is the authentication certificate of a device when the device is communicating with IoT Platform. It contains three key fields: ProductKey, DeviceName, and DeviceSecret as shown in figure below.

- ProductKey: The globally unique identifier issued by IoT Platform for a product.

- DeviceName: The identifier of a device. It must be unique within a product and is used for device authentication and message communication.
- DeviceSecret: The secret key issued by IoT Platform for a device. It is used for authentication encryption and must be used in pairs with the DeviceName

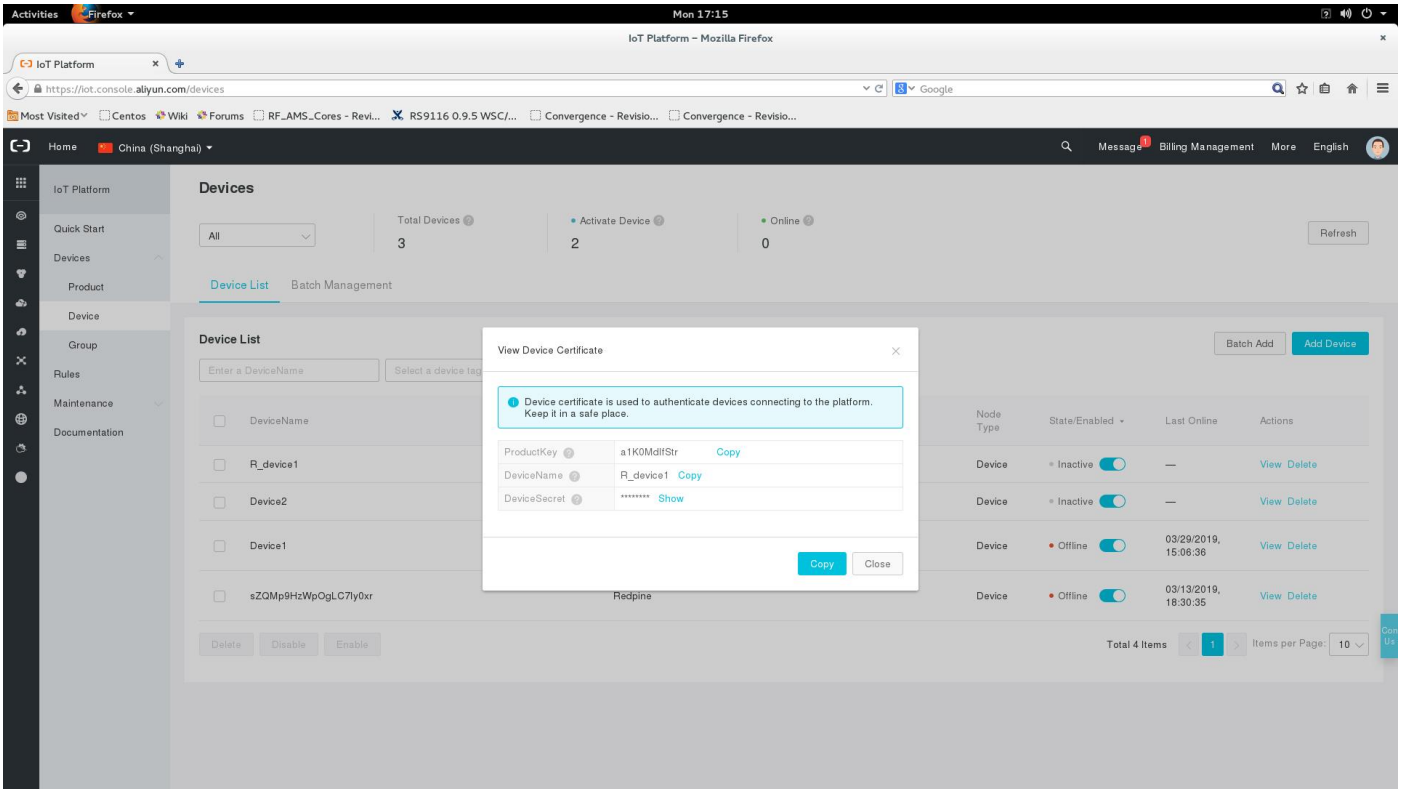


Figure 163 Device Certificate

On the device list page, find the device and click View. On the Device Details page, you can view the information of this device.

Topic creation

To create a topic please refer the following documents in the Alibaba cloud home page

- Documentation → IoT → IoT Platform → User Guide → Create products and devices → Topics

Establishing MQTT connection over TCP

- For the information to update Domain Name, client ID, user name and passwords that are needed to establish mqtt connection please refer the following section of Documentation in the Alibaba Cloud home page.
  Documentation → IoT → IoT Platform → Developer Guide (Devices) → Protocols for connecting devices → Establish MQTT connections over TCP.
- Please download root ca certificate also from the above mentioned page.

Publish and subscribe data on Topics

- User can observe published data from the device in Upstream Analysis tab in the page IoT platform→Maintenance→Device log as shown in the below screen shot.

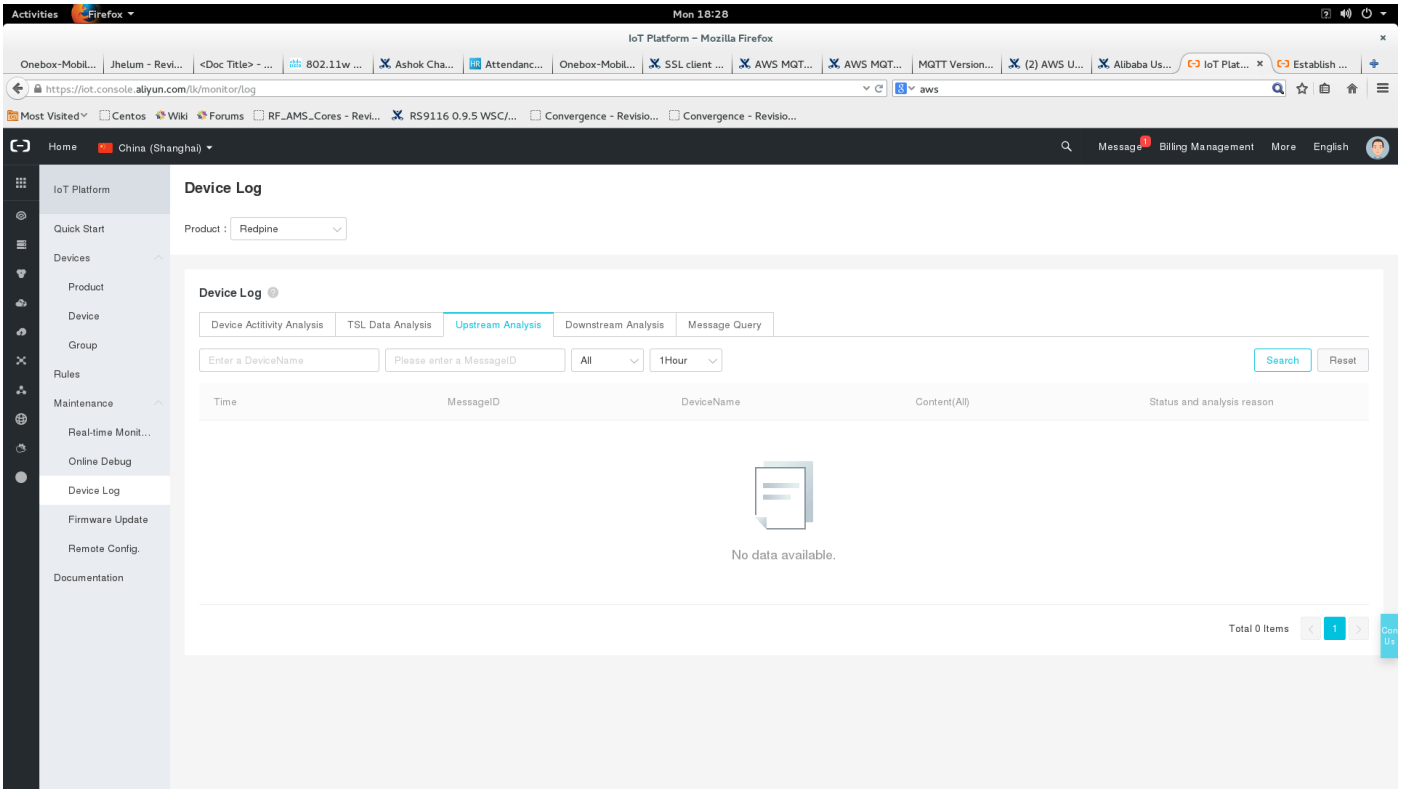


Figure 164 Published Data from MQTT Client

- User can publish to a topic to which client has subscribed from the cloud by clicking on 'publish' in the page "Topic list" under IoT platform→Device as shown in the below screen shot.

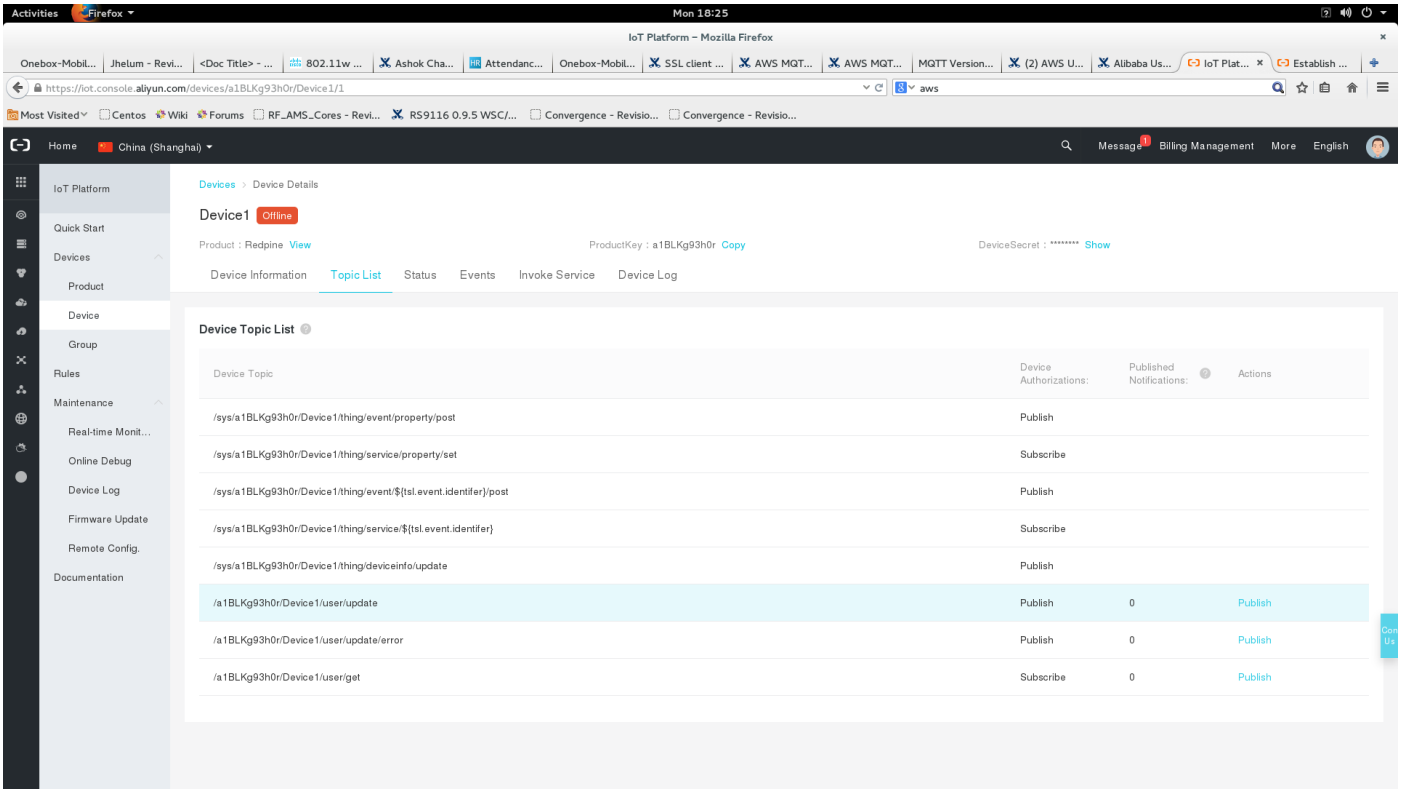


Figure 165 Publish Data from Cloud

AWS CLOUD

AWS account creation

AWS IoT account is required to create resources required to send, receive, and process MQTT messages from devices using AWS IoT.

> **Note:**
>
> For further support in creating thing please visit https://docs.aws.amazon.com/iot/latest/developerguide/create-iot-policy.html

Follow the steps given below to create an AWS account:

- Open the AWS home page https://aws.amazon.com and choose Create an AWS Account.
- Follow the online instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using your phone's keypad.
- Sign into the AWS Management Console.
- In the Console Home page, select your AWS Region as shown in the Figure below

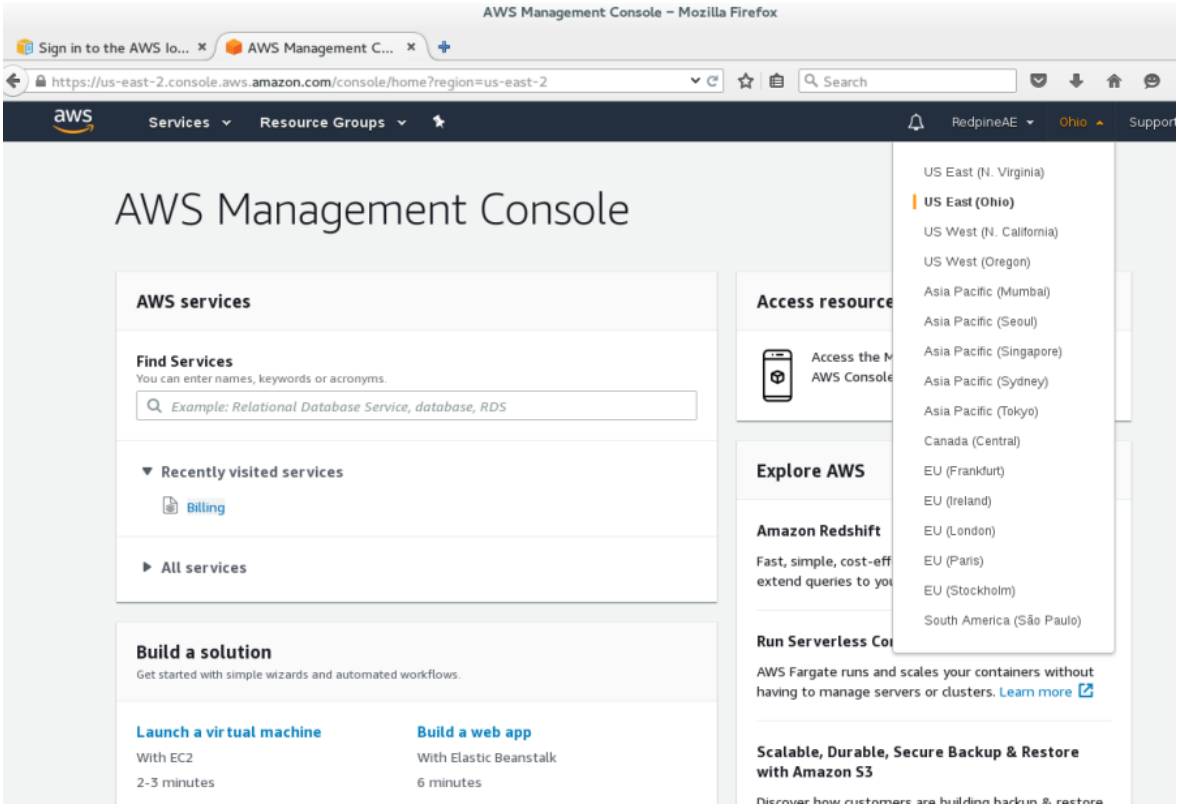


Figure 166 AWS Management Console

IoT thing creation in AWS

Create a Thing

- Choose the AWS IoT service. The AWS IoT Console window appears as shown in the above figure.
- In IoT Select Manage > Things and click on Create button as shown in the above figure.

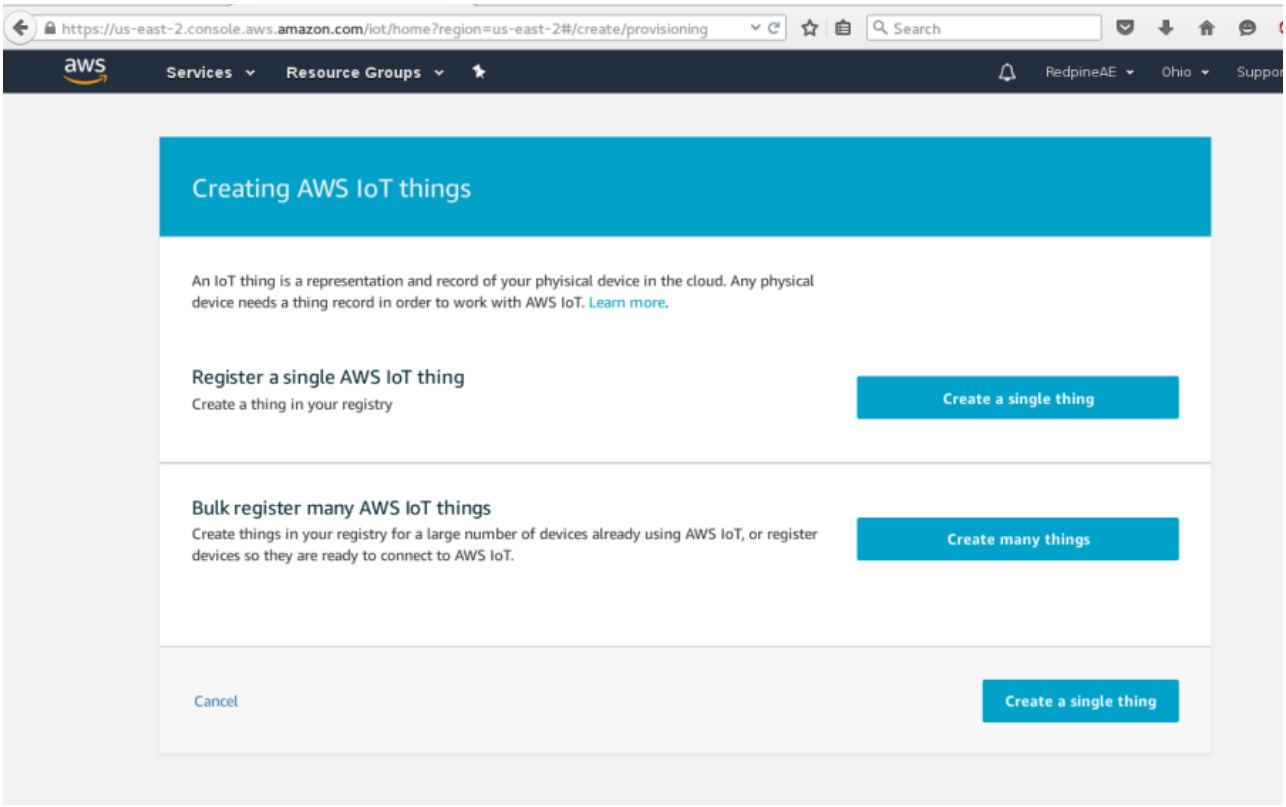


Figure 167 Things

- A page "Creating IoT Things " appears and click on ton "Create a single Thing" as shown in the above figure. If the page "You don't have any things yet" appears click on the button "Register a Thing".

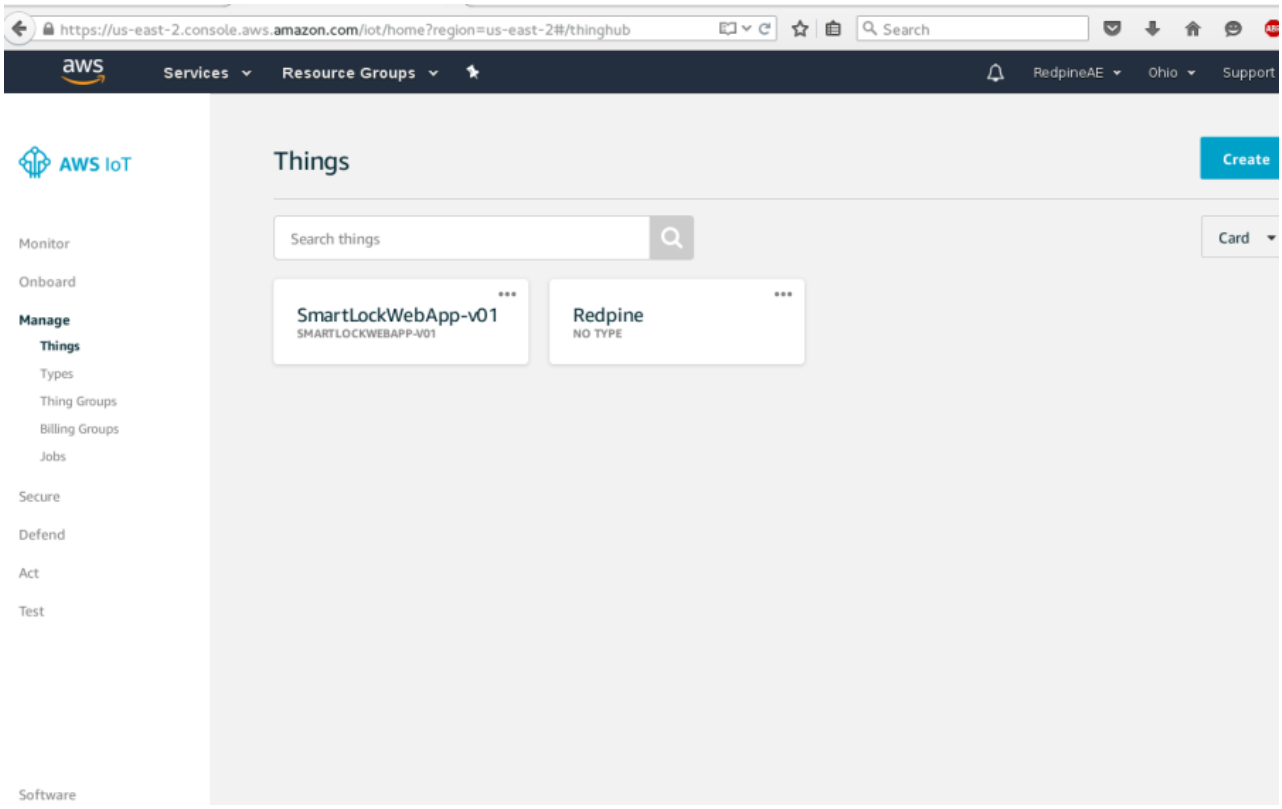


Figure 168 Create AWS IOT things

- In the next page enter the name of the thing as shown in the figure below, and the name should be unique for each thing created. Click on "Next" button.

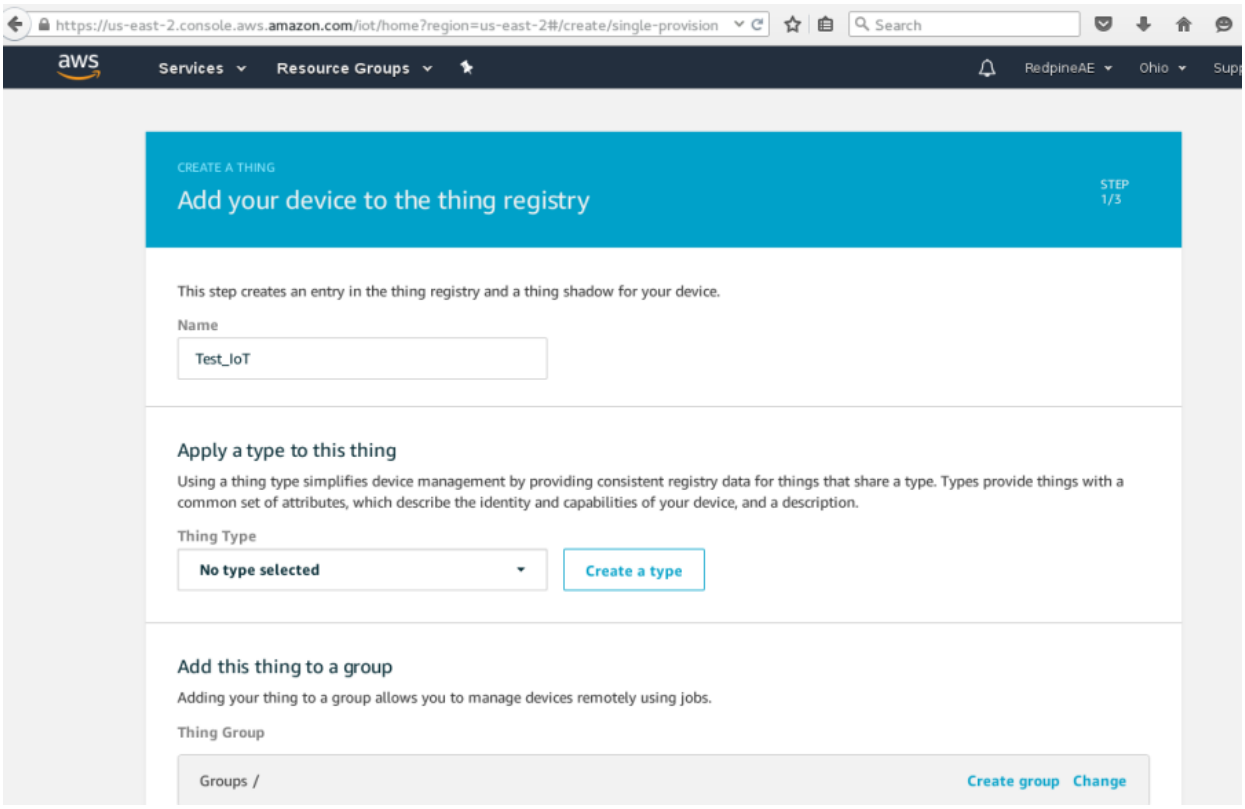


Figure 169 Add device to thing Registry

- In the page "Add a certificate to your thing" as shown in the figure below, click on the button "Create certificate".

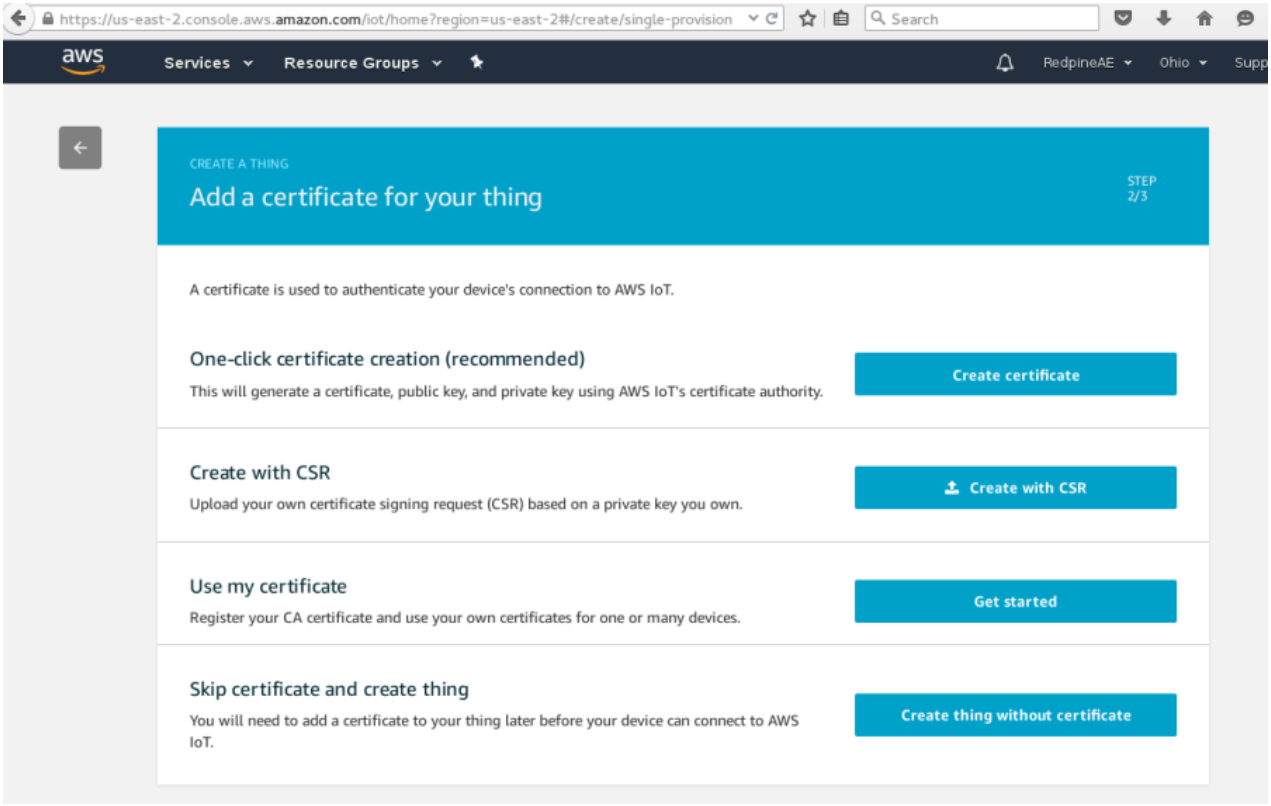


Figure 170 Add Certificate to Thing

- A page "Certificate created " appears, On the Certificate created page, download your public and private keys, certificate, and root certificate authority (CA). Save them on your computer, you will copy them to a different directory when you run the sample applications. Choose Activate to activate the X.509 certificate, and then

choose Attach a policy as shown in the figure below. For root CA certificate click on download button and download the CA certificate from the page "X.509 Certificates and AWS IOT".

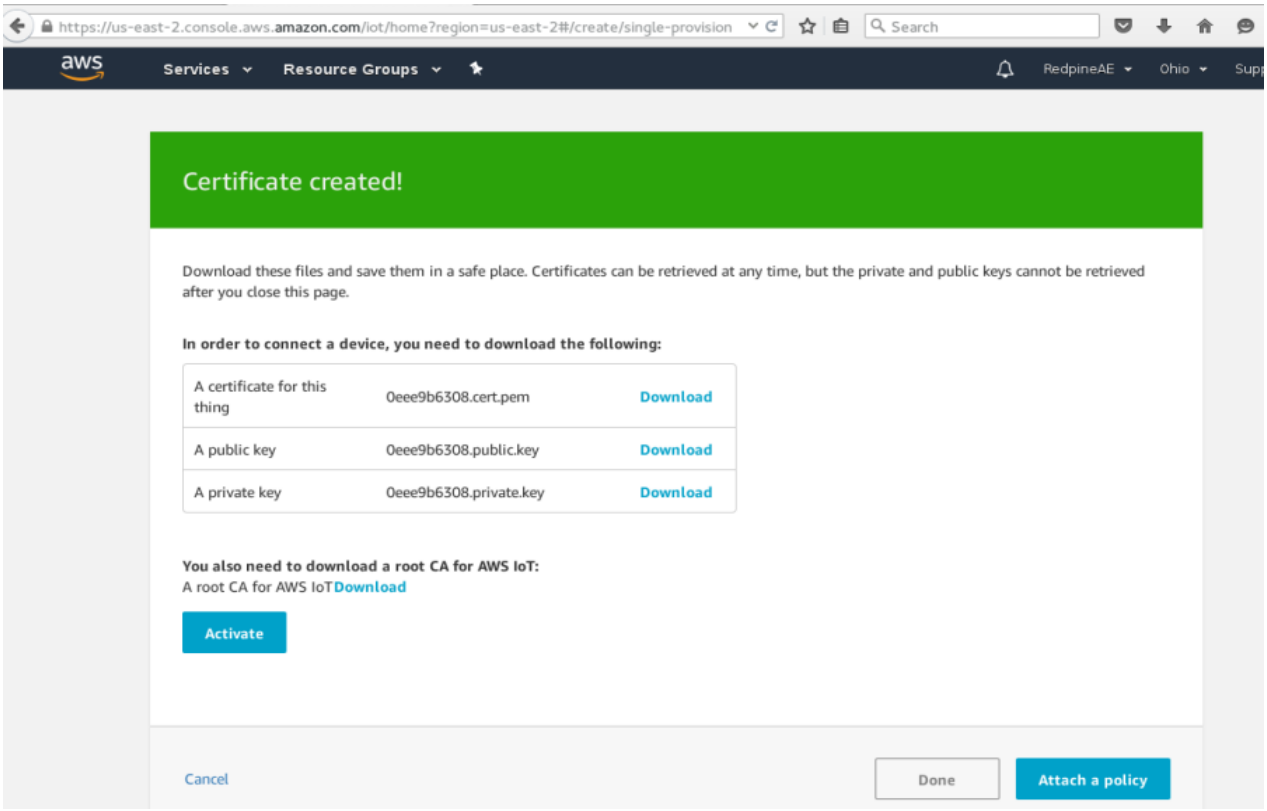


Figure 171 Certificate Created

- In the page "Add policy for your thing" as shown in the figure below, click on "Register Thing".

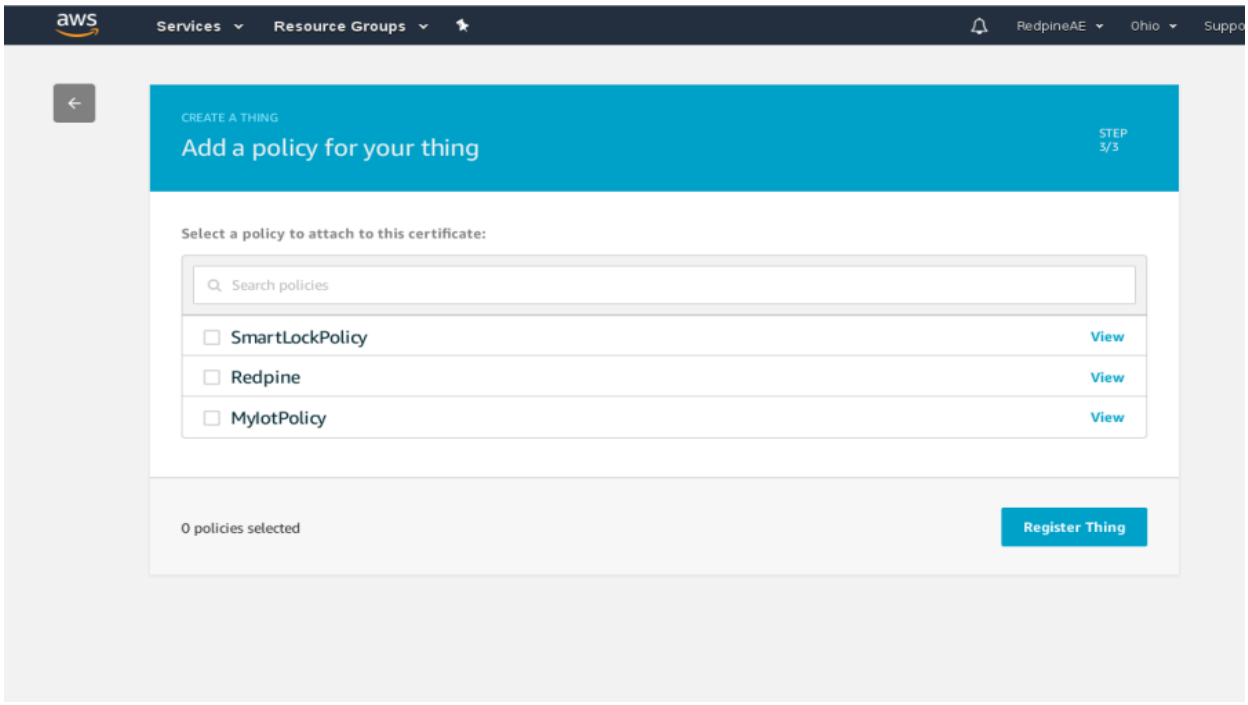


Figure 172 Add Policy to Thing

- Now the Thing is successfully created and appears in the page "Things" as shown in the figure below.

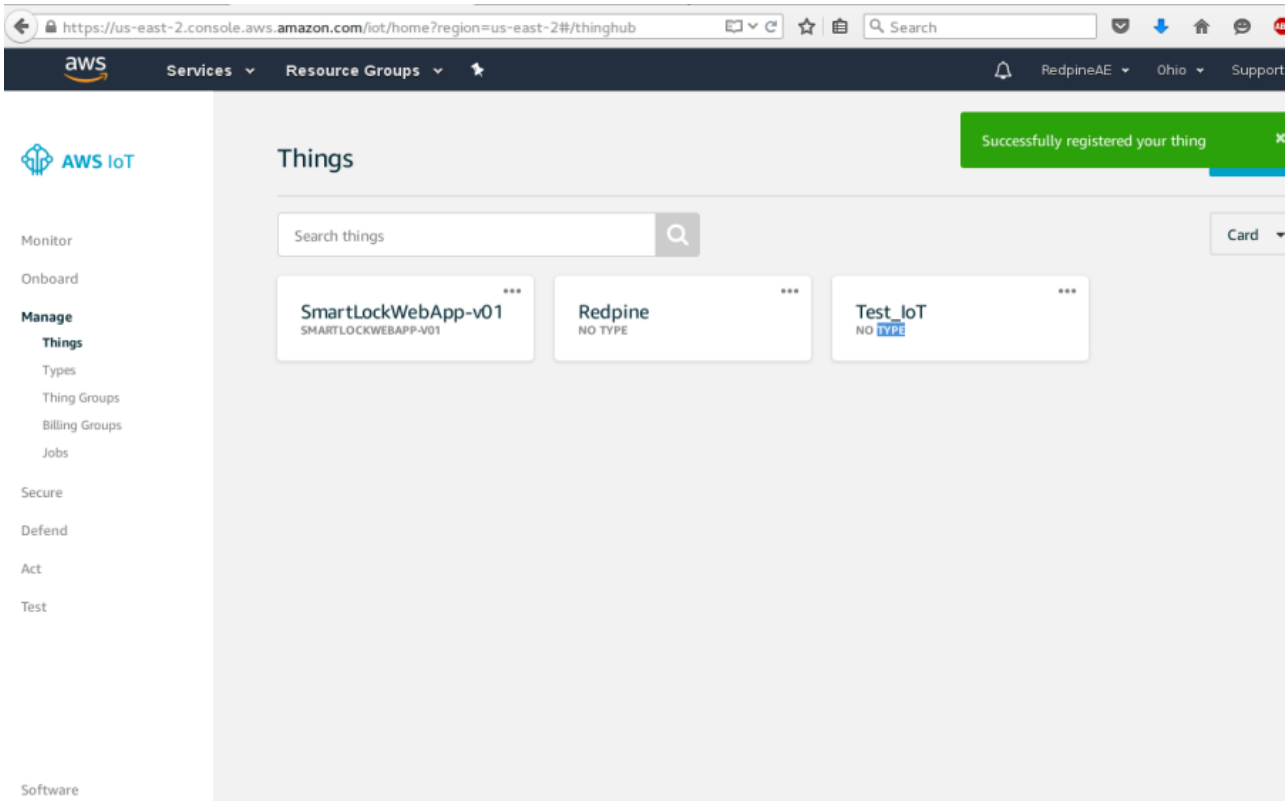


Figure 173 Thing Created

Create AWS IOT policy

- In the AWS IOT console select secure > policies as shown in the figure below and click on the button "Create". And if the "you don't have any policies yet " appears click on the button "Create a policy".

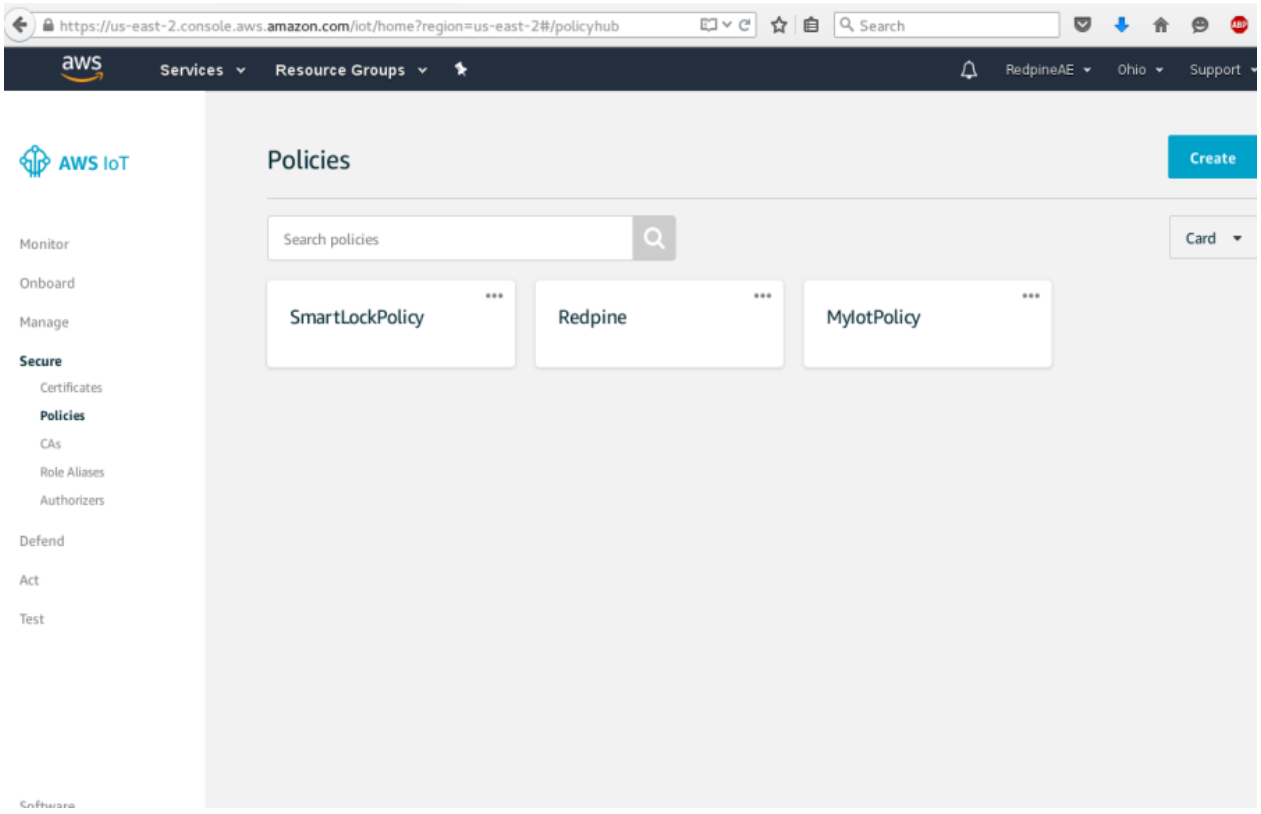


Figure 174 Create policy

- As shown in the above figure, on the page "Create a policy"
- Enter a Name for the policy, such as MyIotPolicy.
- For Action, enter iot:*. For Resource ARN, enter *.
- Under Effect, choose Allow, and then choose Create.

This policy allows your device to perform all AWS IoT actions on all AWS IoT resources.

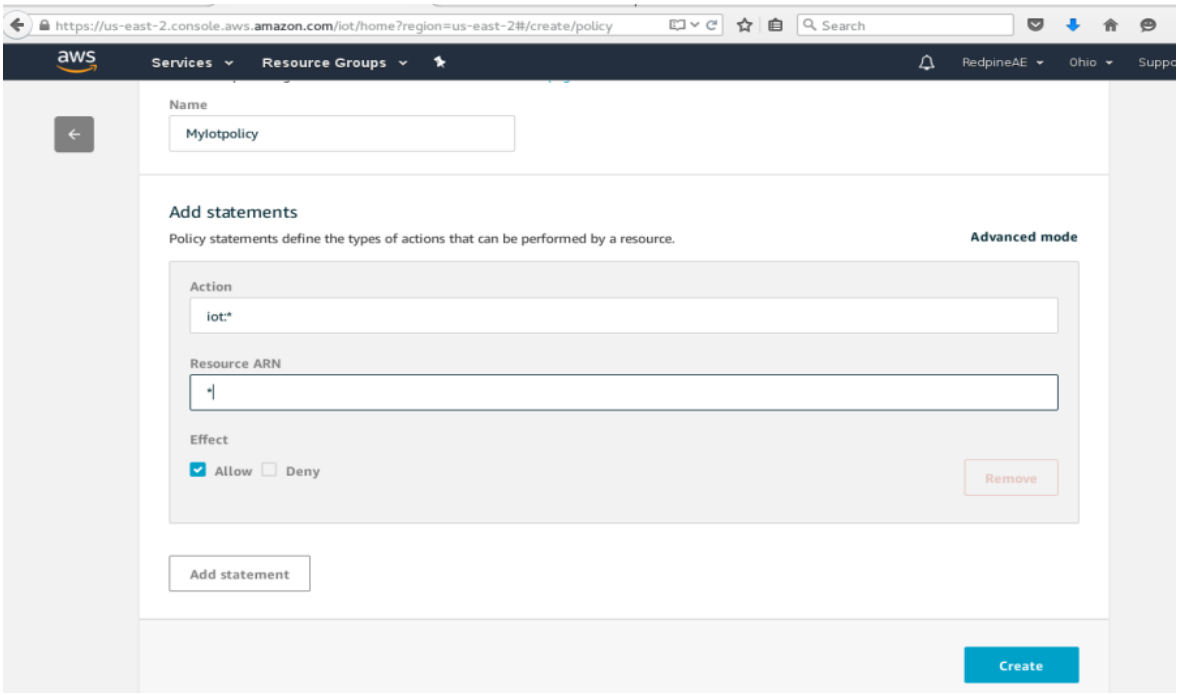


Figure 175 Policy Details

Note:

These settings are overly permissive. In a production environment narrow the scope of the permissions to that which are required by your device. For more information, see Authorization.

- On the "Policies" page, the policy name created appears as shown in the figure below.

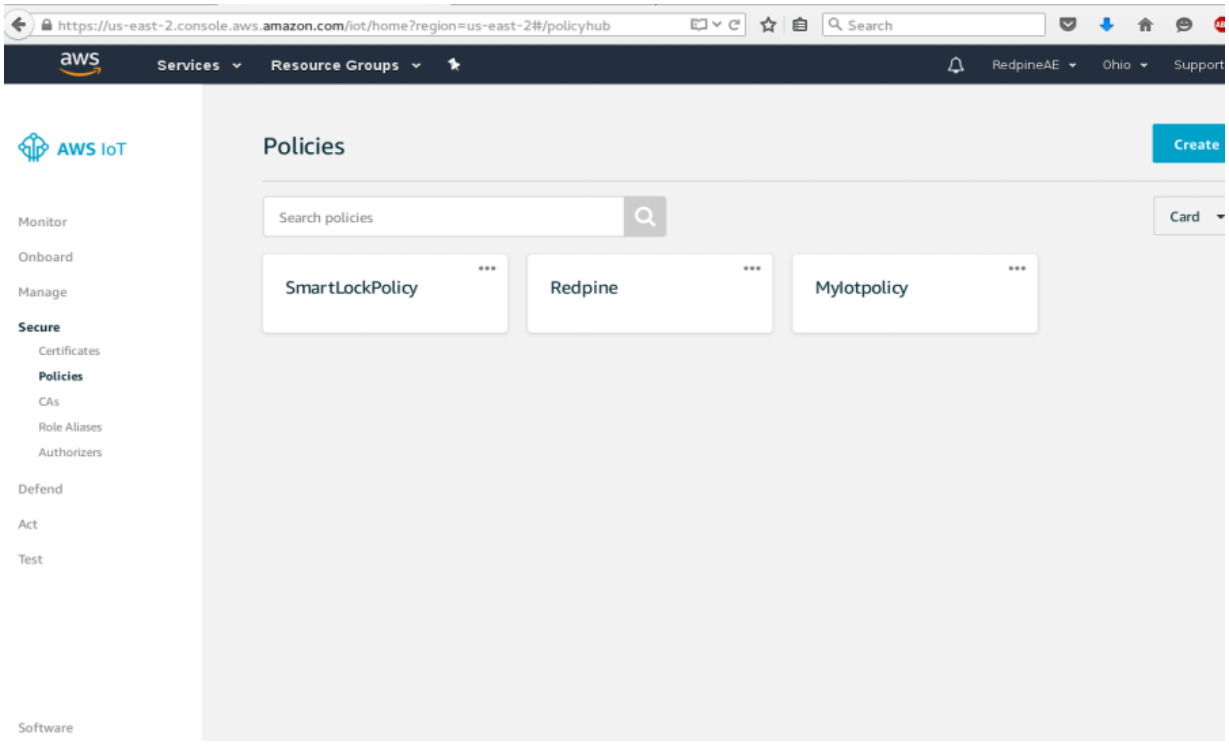


Figure 176 Policies Page

- Choose manage and select the AWS IOT thing created as shown in the figure below.

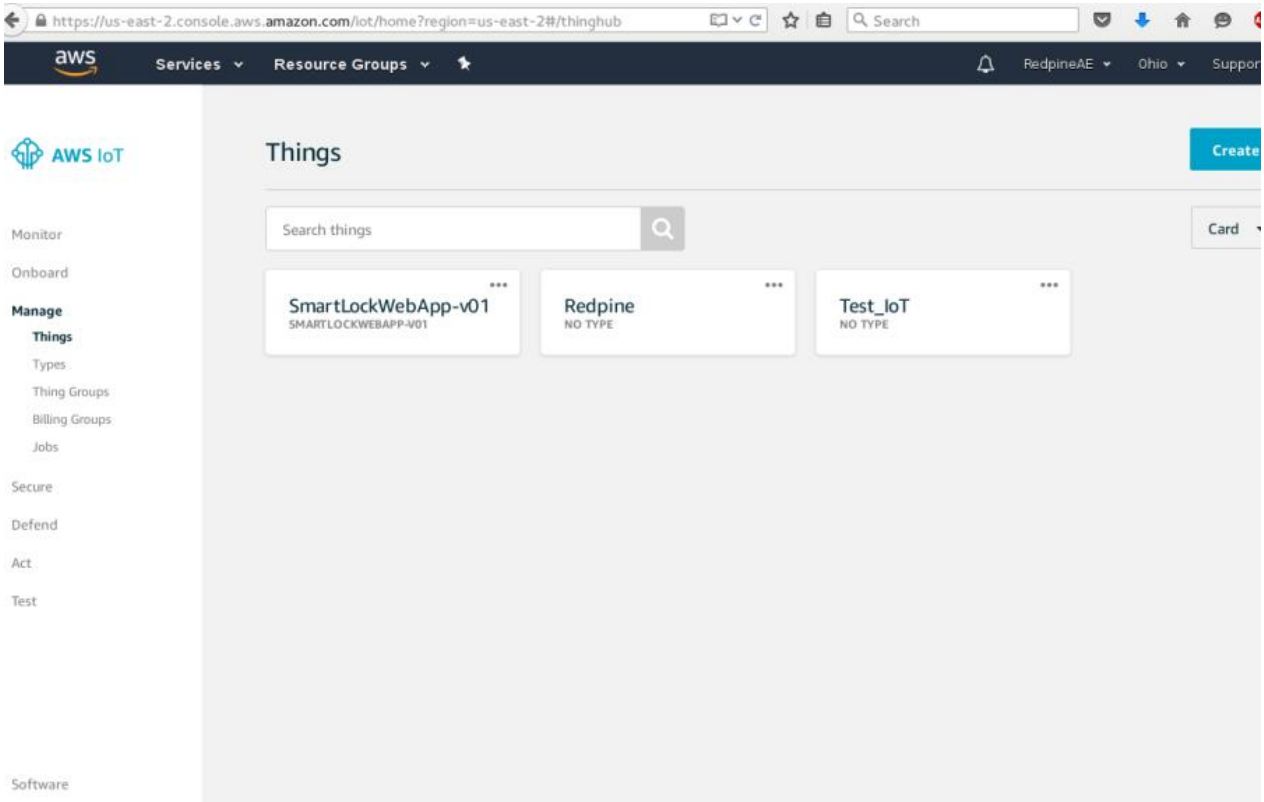


Figure 177 AWS IOT

- On the next page choose "security" as shown in the figure below and click on the certificate.

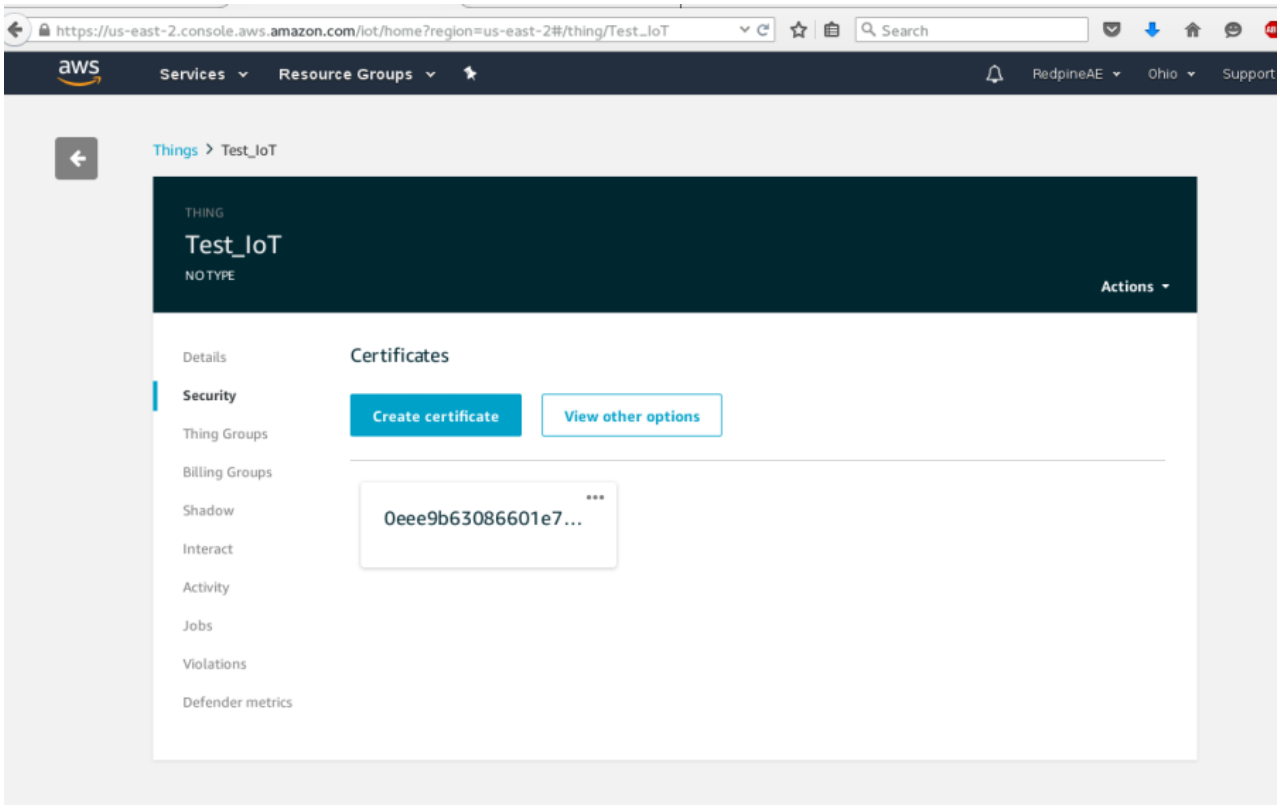


Figure 178 Create Certificate

- In the certificate page click on "Actions" drop down and select attach policy as shown in the figure below.

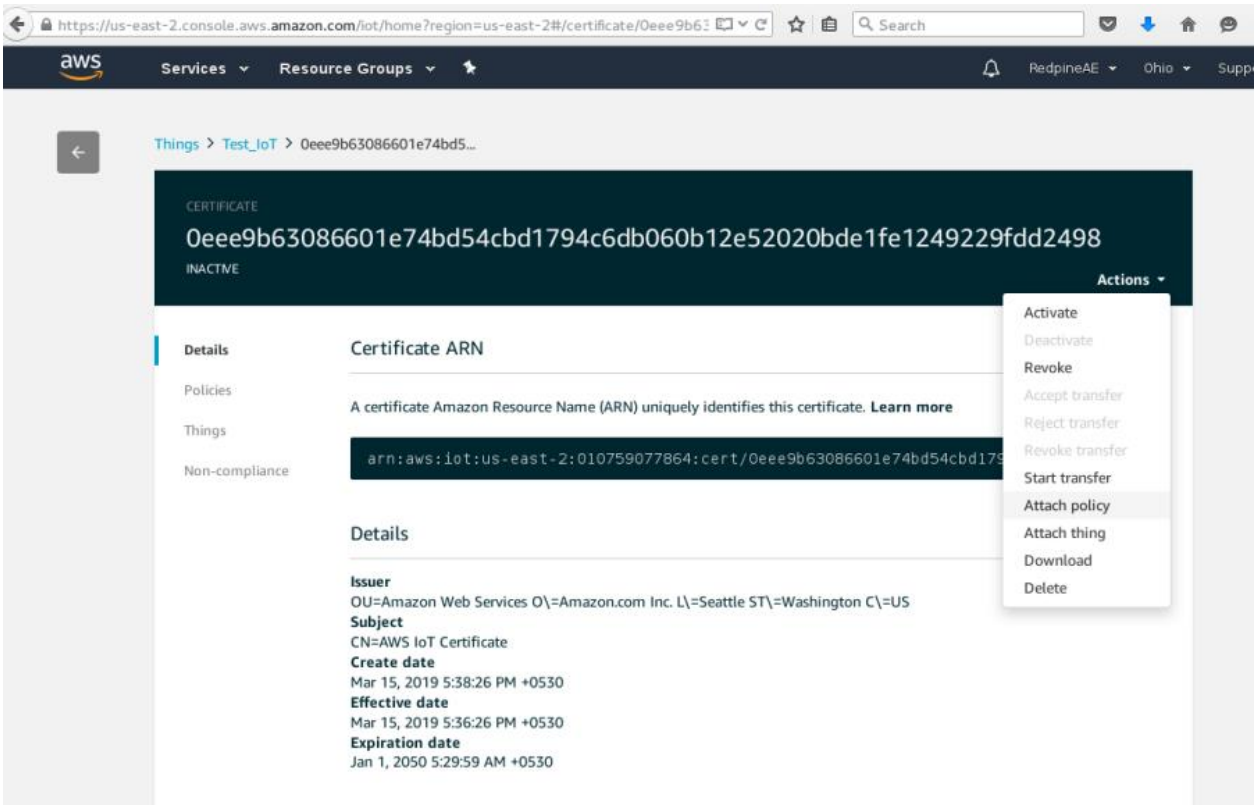


Figure 179 Actions

- Choose the policy you have created as shown in the figure below and click on "Attach" button.

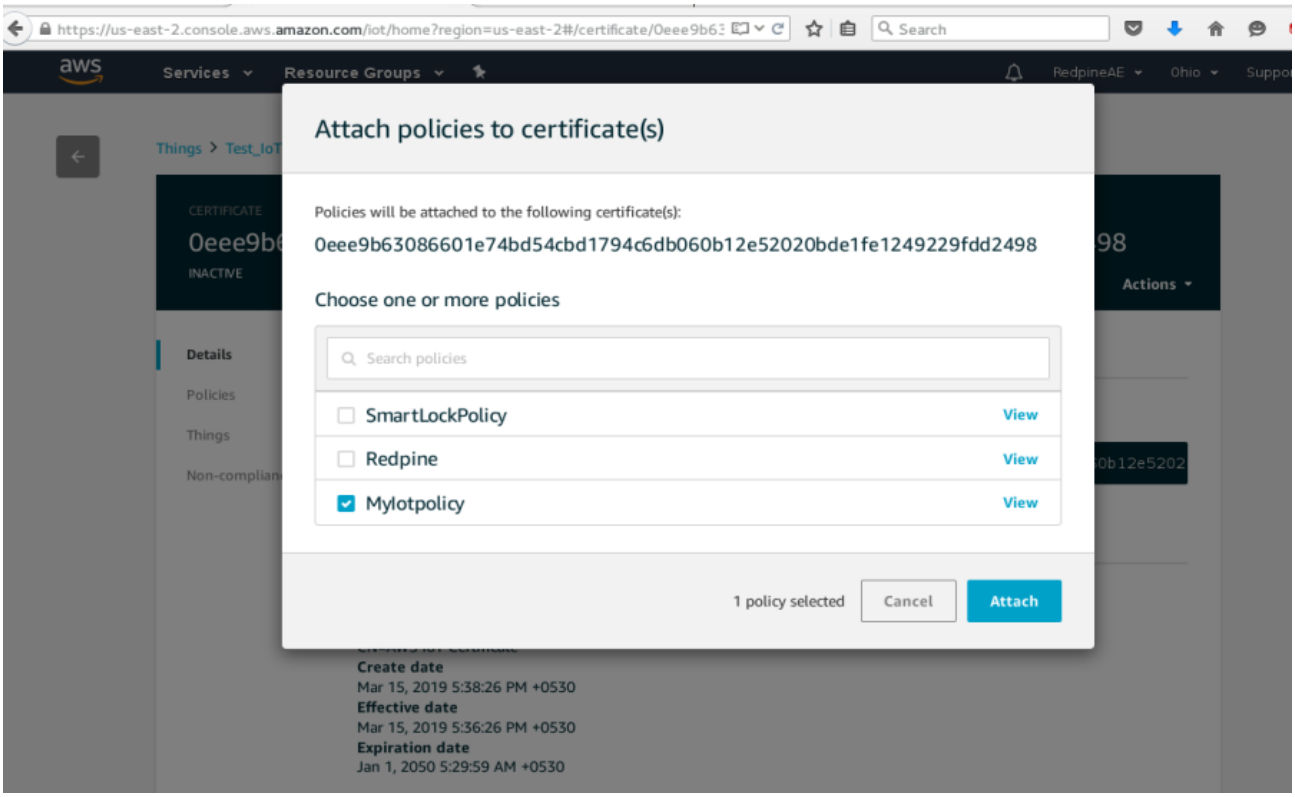


Figure 180 Attach Policy

- The AWS IOT Thing is successfully created.
- Now go back to AWS IOT console. And click on security > certificates. Select the certificate created. If it is in Inactive state, select options for the certificate and click on Activate button as shown in the figure below.

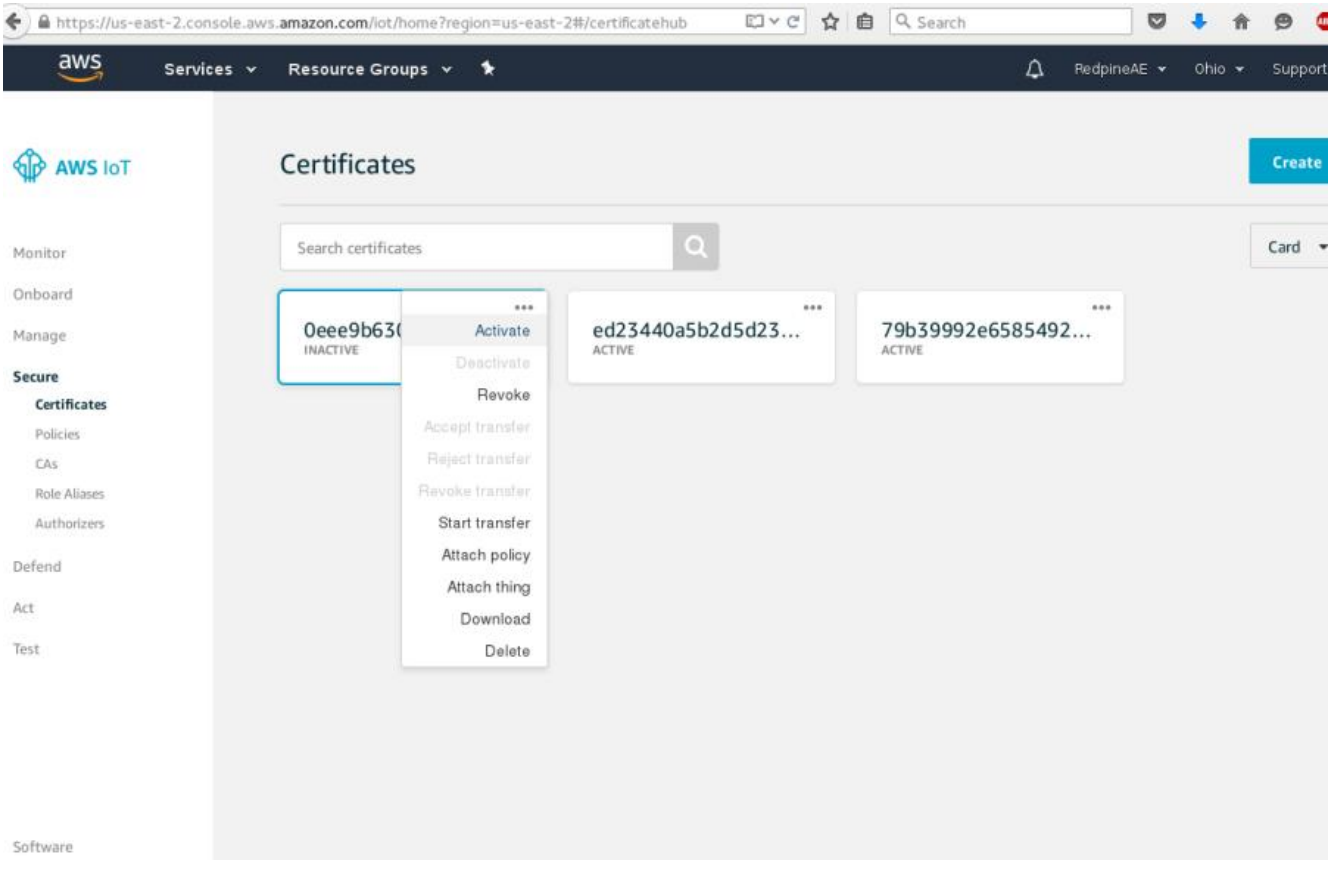


Figure 181 Activate Certificate

Client configurations

Converting formats of the certificated generated in step 6 before loading to module using API's

1. Copy the certificates generated in step 6 to the folder /Examples/Wireless_Examples/utilities/certificates/ in the release package.
2. Convert the certificate formats to linear buffer formats as explained in the 'Cloud' section in WLAN examples.

Running the application

Note:

Please refer "Cloud" section in WLAN Examples.

- In AWS IoT page go to Manage->things and click on the "thing" created.
- Click on Interact, a page will appear as shown in figure below.

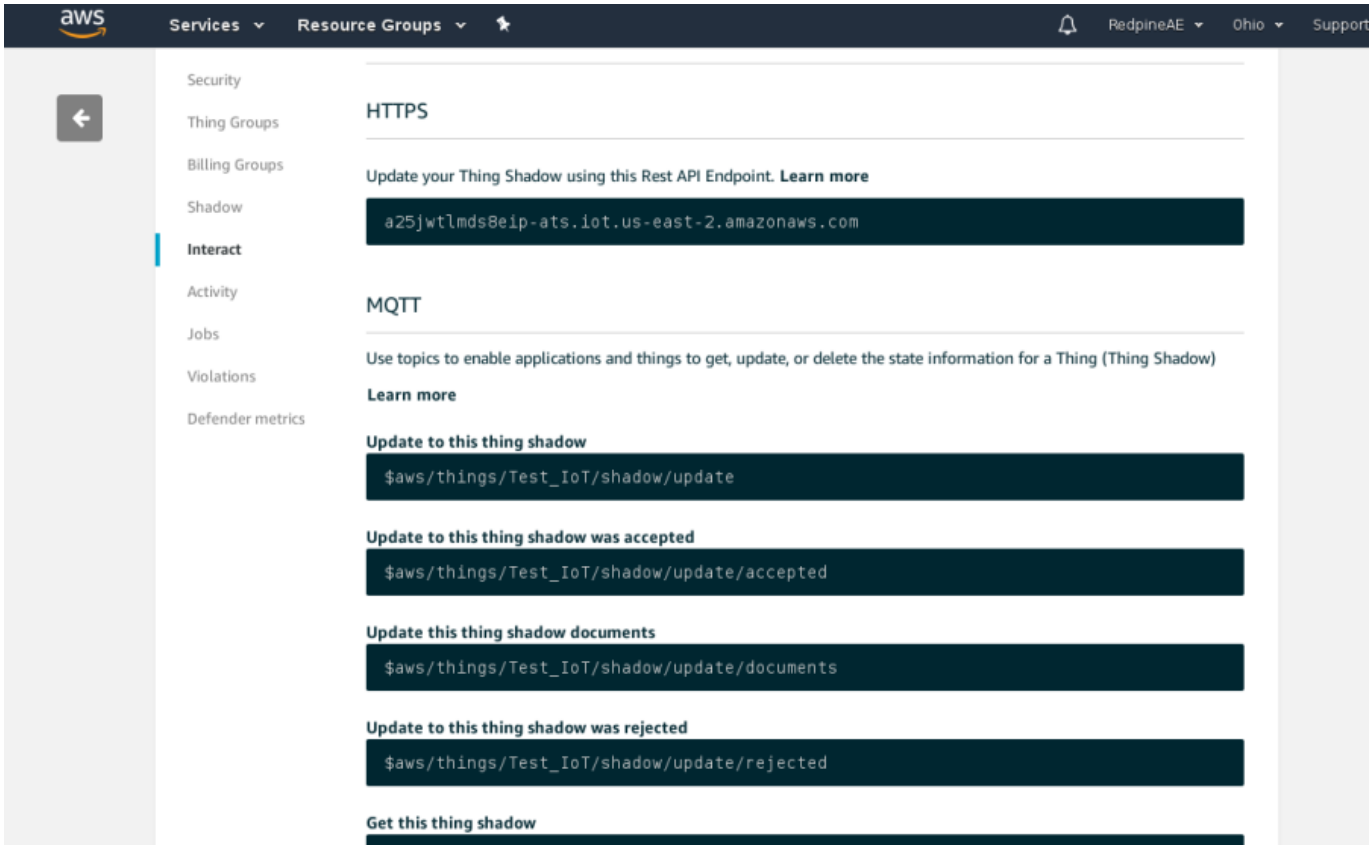


Figure 182 Interact Page

- Open the file rsi_mqtt.c in the folder Examples/Wireless_Examples/wlan/cloud/mqtt/ in release package.
- update the filed AWS_DOMAIN_NAME in rsi_mqtt.c to the filed in the box under HTTPS in the page interact.
- Update SERVER_PORT in rsi_mqtt.c to 8883.
- Update clientID in rsi_mqtt.c to a uniqe name of users choice.
- Update RSI_MQTT_TOPIC to the topic for which user wants to publish , by copying the desired topic under MQTT in Interact page.in rsi_mqtt.c. This will be the topic to which we publish messages.

- In rsi_mqtt.c update. AWS_UPDATE_TOPIC to the topic to which user wants to subscribe , by copying the desired topic under MQTT in Interact page.in rsi_mqtt.c. This will be the topic to which we subscribe and receive messages.
- Now compile the mqtt client application in the device and run the application.
- Open the Activity page under the 'thing' created in AWS IOT and go to Activity filed as shown in the figure below. This page shows the activity of the client after successful connection to the server.

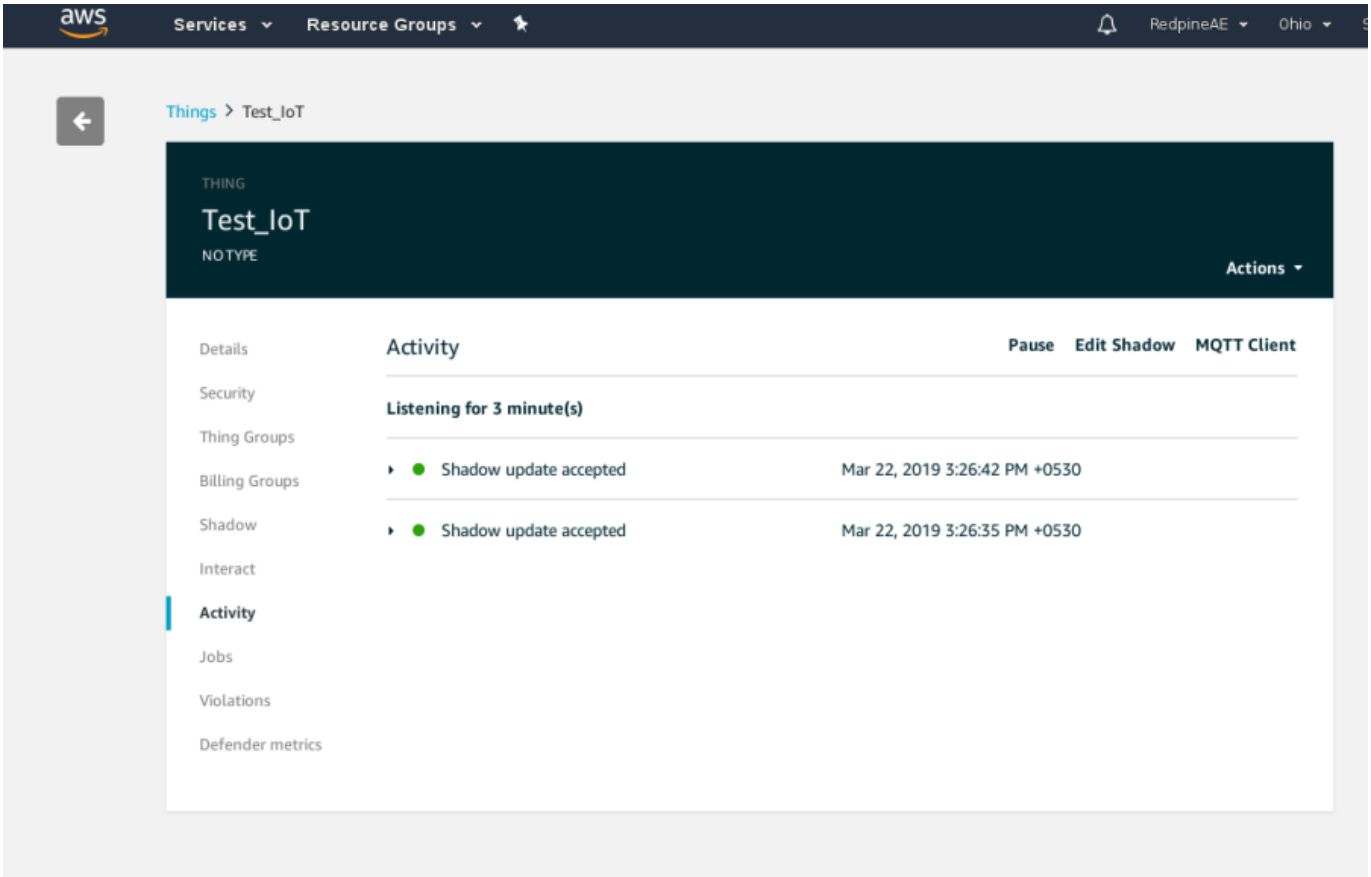


Figure 183 Activity Page

- In the shadow page under the "Thing" created user can observe the messages posted to the thing.
- Go to AWS IoT page and click on 'Test', a page appears as shown below in figure below.

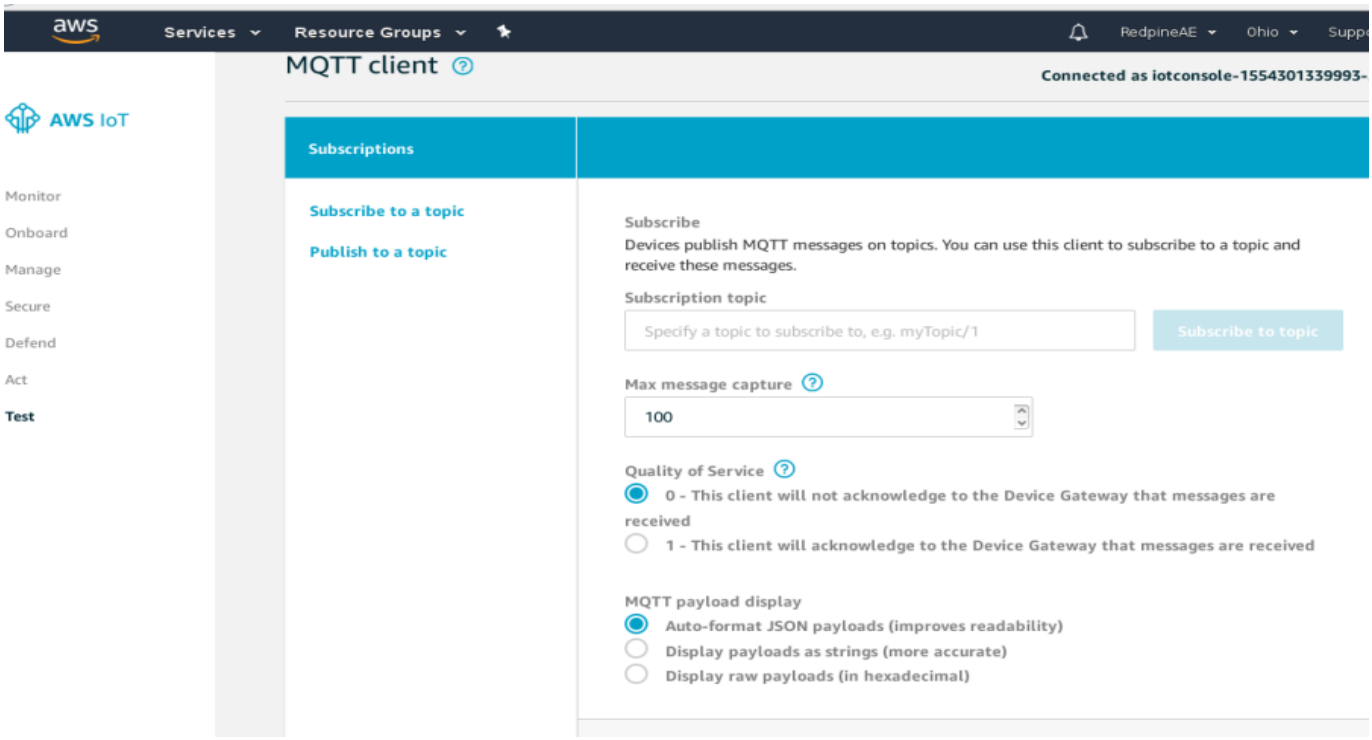


Figure 184 AWS Management Console

- In the Test page user can subscribe to a topic by entering the topic name to subscribe and can see messages sent from the client in the same page.
- And similarly, user can publish to a topic by entering the topic name under 'Publish to a topic'. And the published topic should be subscribed by the client to receive the messages from cloud as shown in the figure below.

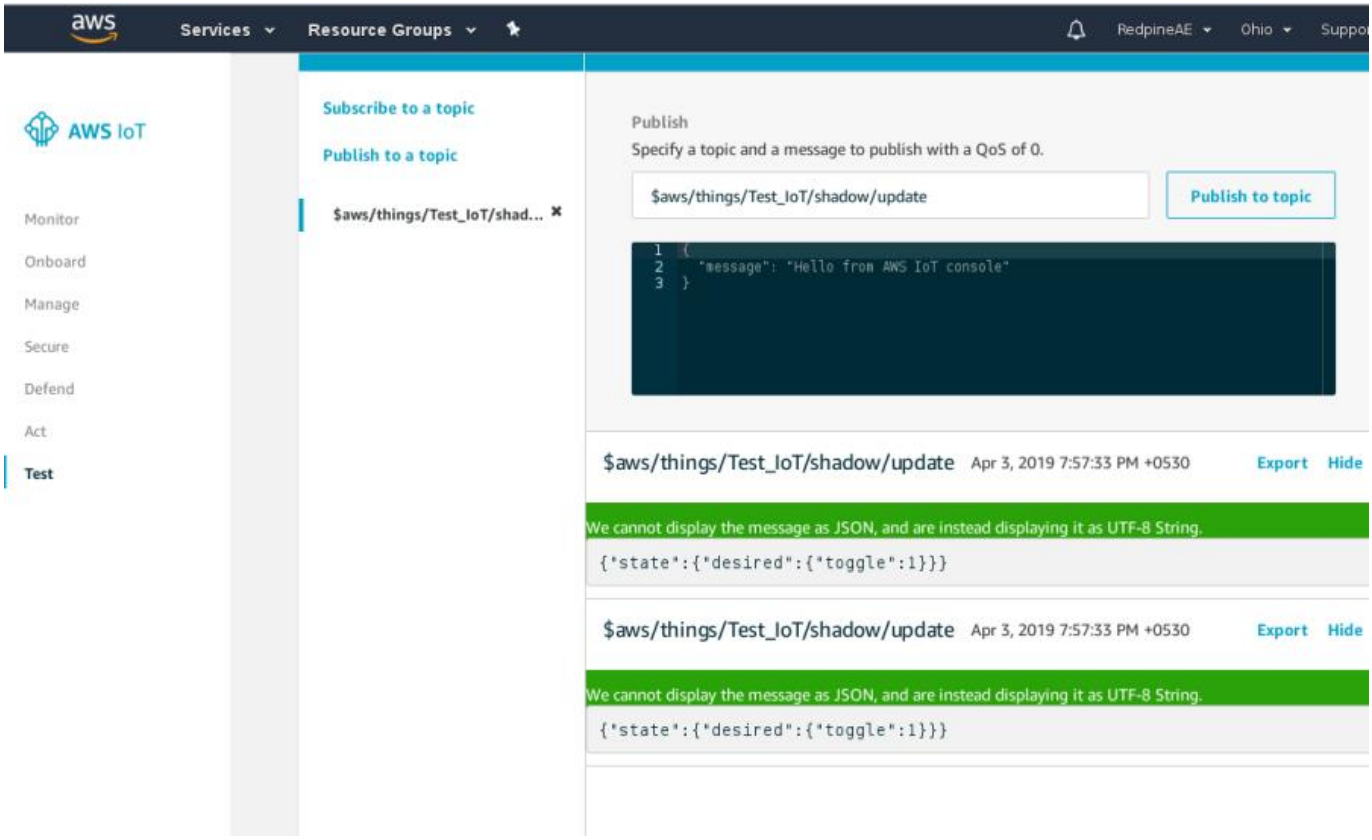


Figure 185 AWS Management Console

Instance Creation in AWS cloud

1. Login to AWS account
    - After successful log into AWS, select region (top right preceded by support) to create an instance.
    - Click on Services then select EC2 present under Compute section then it will show Amazon EC2 resources.
2. Select instances then it will show already existed instances states. To create a new instance click on Launch Instance.
3. Choose an Amazon Machine Image (AMI)An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs. Ex: Selected "Ubuntu Server 18.04 LTS (HVM), SSD Volume Type ", it is a free tier eligible.
4. Choose an Instance Type: Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Ex: Selected free tier eligible General purpose t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GB memory, EBS only)then click on Next: Configure Instance Details.
5. Configure Instance Details: Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more. keep default configuration unless it's necessary to change. Then click on Next: Add storage
6. Add Storage: Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. It shows storage details, keep default configuration unless it's necessary to change. Then click on Next: Add Tags.
7. Add Tags: A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. Default Tag adding is not required unless it explicitly required. Then click on Next Configure Security Group.
8. Configure Security Group: A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Select, Create a new security group and change Type to All traffic and Source to Anywhere. Then click on Review and Launch.
9. Review Instance Launch: Please review your instance launch details. You can go back to edit changes for each section. Click Launch to assign a key pair to your instance and complete the launch process. Then one pop up window will come as Select an existing key pair or create a new key pair. Change 'Choose an existing key pair' to 'Create a new key pair' and give key pair name. Ex: Tokyo_instance4_key then click on Download Key Pair then save that key (for further operations) and click on Launch Instances.

Your instances are launching, and it may take a few minutes until they are in the running state, when they will be ready for you to use. Usage hours on your new instances will start immediately and continue to accrue until you stop or terminate your instances. Click View Instances to monitor your instances' status. Once your instances are in the running state, you can connect to them from the Instances screen.

16 Appendix B: Steps to Test Different Examples under Single Project

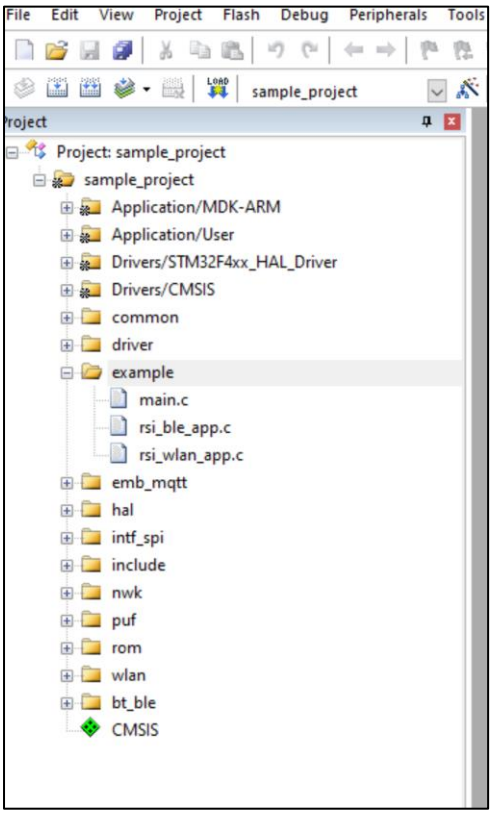
This section describes the steps to be followed to compile and execute different examples on STM32 MCU using keil and STMCube IDE.

16.1 Steps for Keil IDE

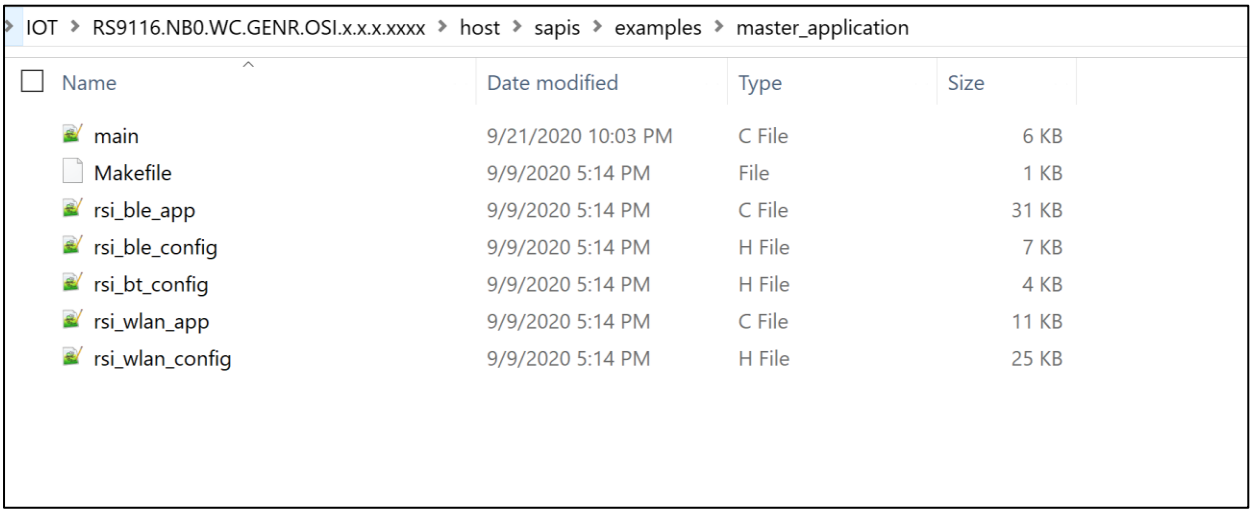
1. Open "sample_project.uvprojx " located in below folder

   "RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI\sample_project"

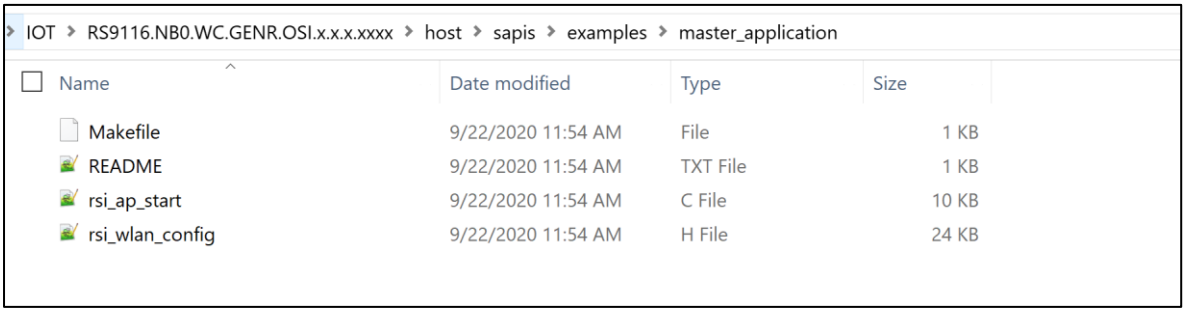
2. By default this project maps to sample example code located in folder "RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\sapis\examples\master_application".



3. To test specific example, navigate to above folder.

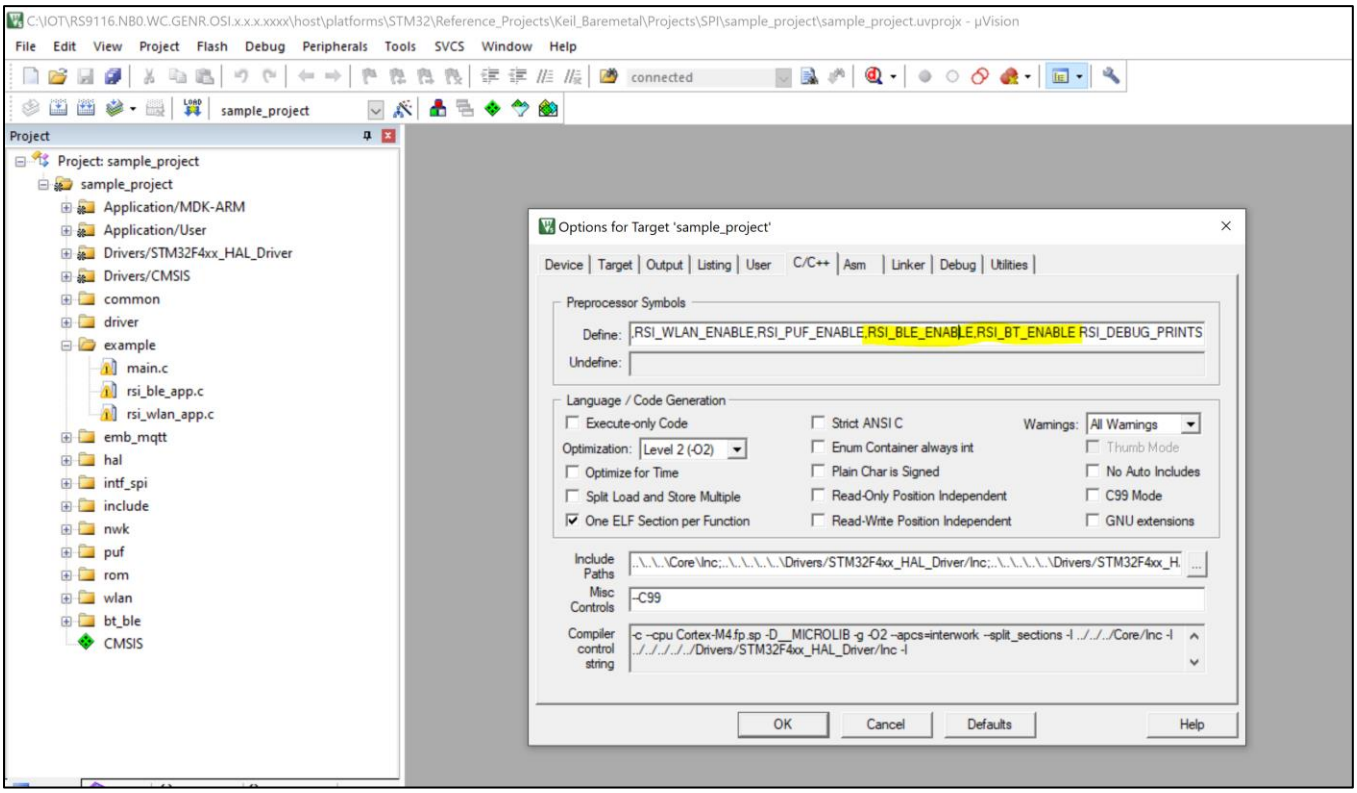


4. Delete the existing code, copy and paste the specific example code in to above folder.
   Ex: Below screenshot shows the example code copied from
   "RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\sapis\examples\wlan\access_point\" to
   "RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\sapis\examples\master_application

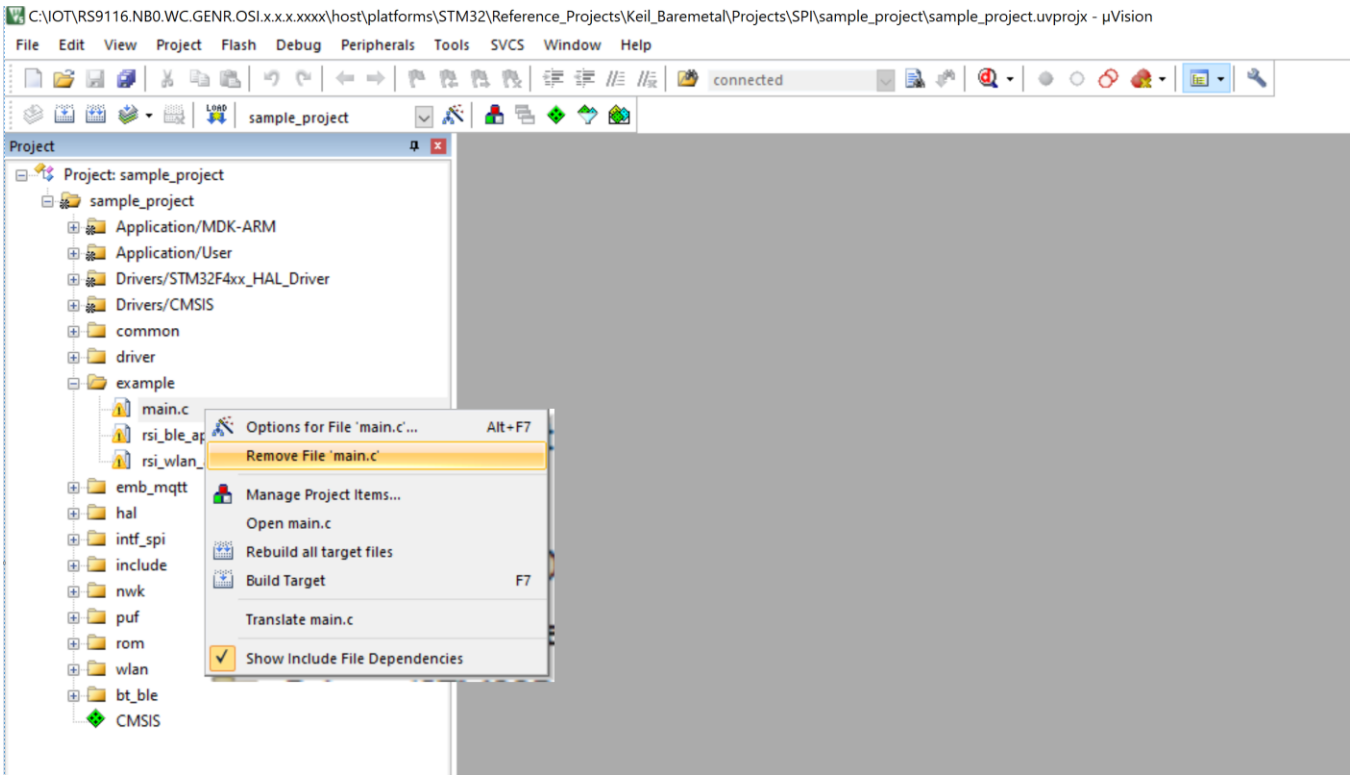


Note:

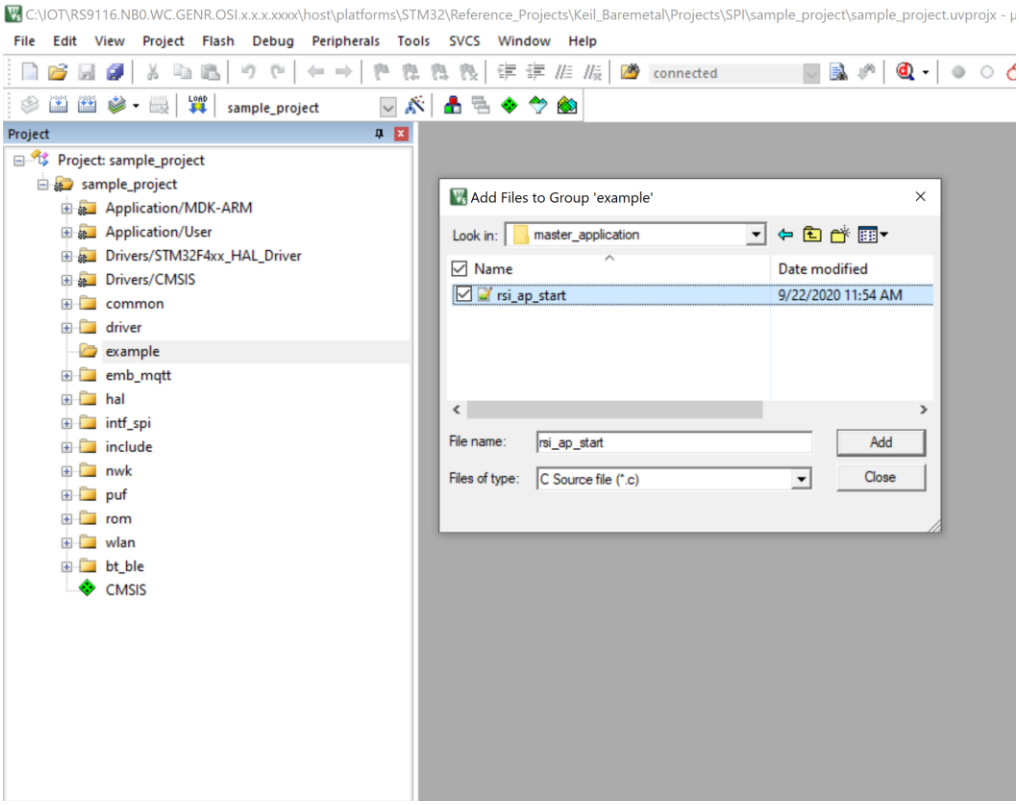
- Sample_project compiles successfully only if all the header files rsi_wlan_config.h, rsi_bt_config.h and rsi_ble_config.h are present in "master_application" folder.
- In case if example code doesn't contain any of those header files, use the default files present in the "master_application" folder
  (or)
  Remove the corresponding protocol macros in keil project settings (Ex: As above example code doesn't have rsi_ble_config.h and rsi_bt_config.h, remove RSI_BLE_ENABLE, RSI_BT_ENABLE)



5. Next, go to keil IDE and remove the already existing files from examples folder



6. Add new files (only '.c') and compile the project



7. Repeat the same steps to compile any example with this project.

Note:

Follow same steps to compile the **'sample_project.uvprojx'** located in **"RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI\sample_project"**

STM32 projects for following examples are already provided at 'RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI'. Efforts to create STM32 projects for these examples using 'master_application' can be saved.

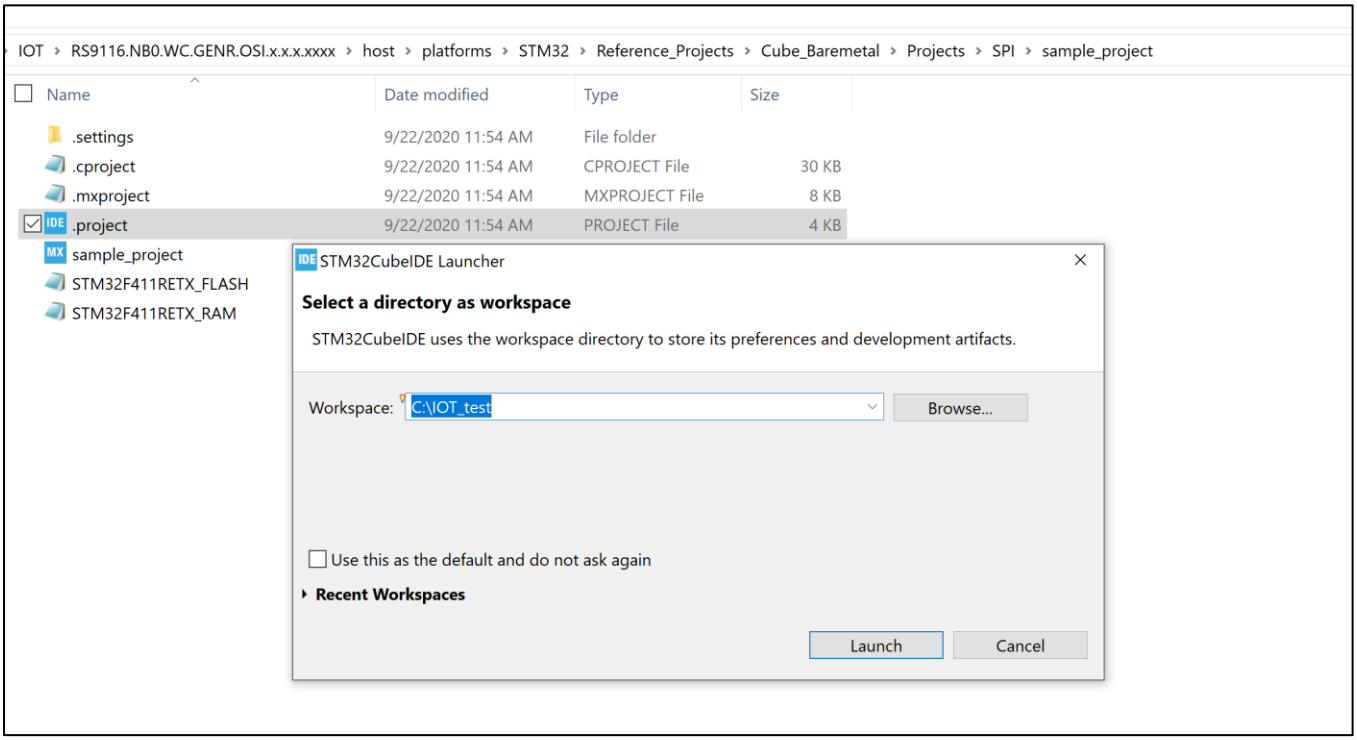
- AWS_IoT
- BT_Alone
- eap
- Firmware_upgrade
- Power_save
- Throughput
- wlan_sta_ble_bridge
- wlan_sta_ble_provisioning

Below are the applications need to include the files from the nwk (\host\sapis\nwk\applications) folder for successful compilation of sample project.

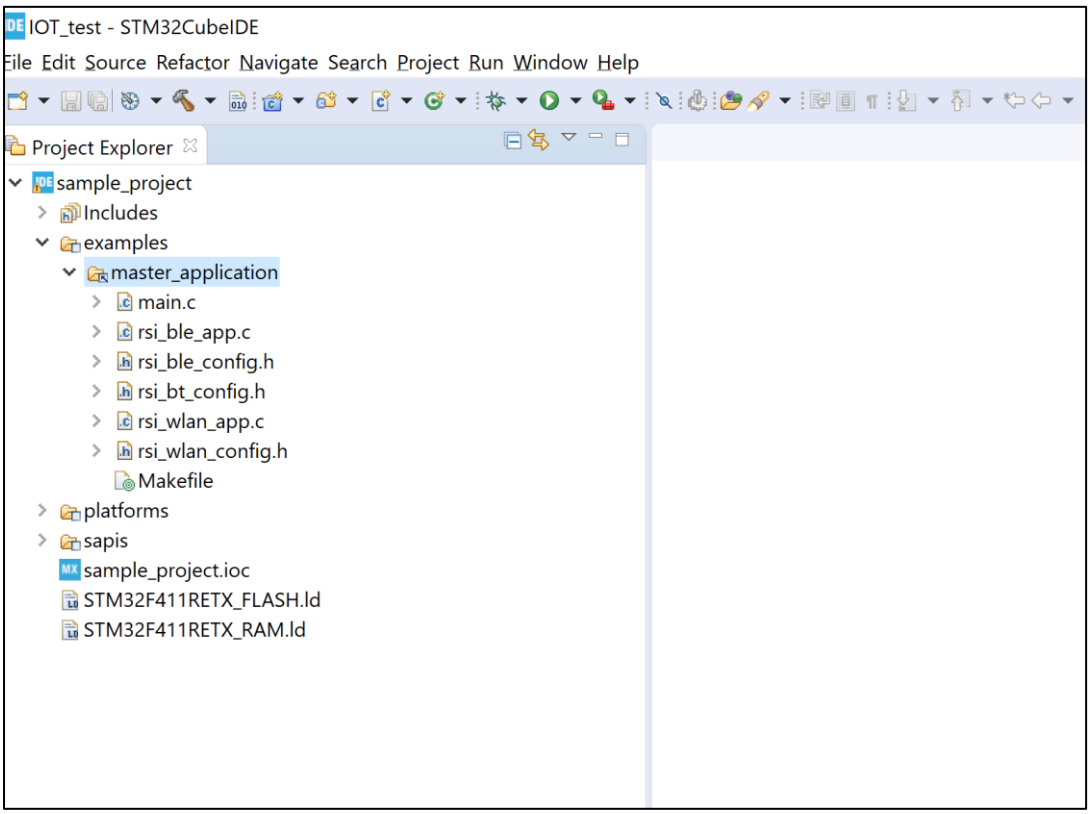
S.No	Application	Nwk folder files need to add for successfull execution($\host\sapis\nwk\applications)
1	cloud	Files present in the complete AWS-SDK folder
2	mqtt_client	Files present in the complete mqtt_Client folder and rsi_mqtt_client.c present in the application folder
3	Sntp_client	rsi_sntp_client.c($\host\sapis\nwk\applications)
4	smtp_client	rsi_smtp_client.c($\host\sapis\nwk\applications)
5	DNS_Client	rsi_dns.c($\host\sapis\nwk\applications)
6	web_socket	rsi_web_socker.c
7	wireless_firmware_upgrade	rsi_ota_fw_up.c
8	http_Client/http_client_post_data	rsi_http_client.c
9	multicast	rsi_multicast.c
10	ftp_client	rsi_ftp.c
11	emb_mqtt	rsi_emb_mqtt.c
12	dhcp_dns_fqdn	rsi_dhcp_dns_fqdn.c
13	otap	rsi_firmware_upgrade.c
14	dhcp_user_class	rsi_dhcp_user_class.c
15	Raw data	rsi_raw_data.c

16.2 Steps for STMCube IDE

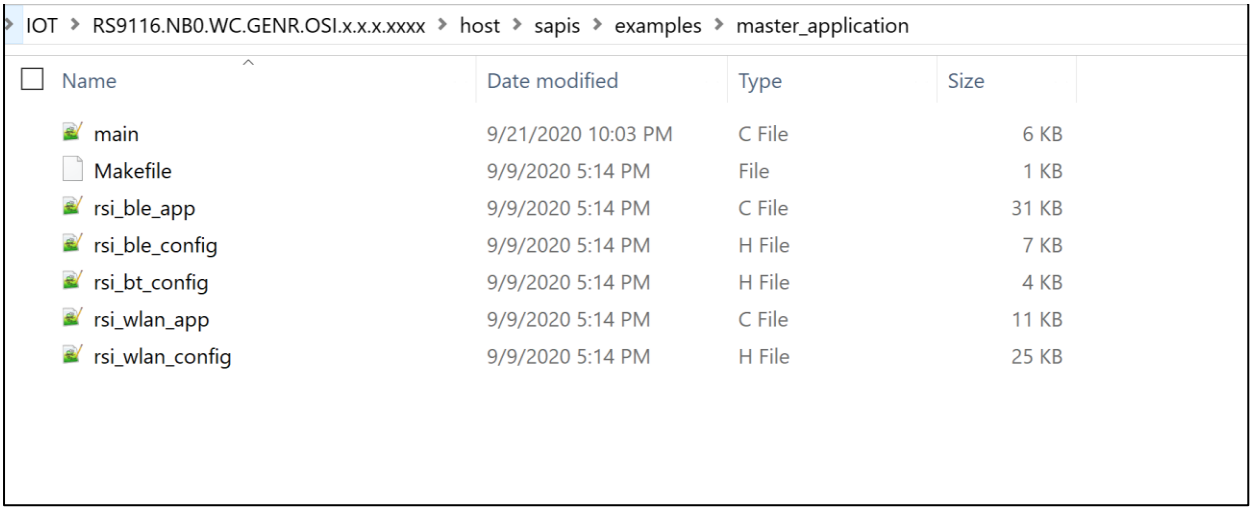
1. Double click on ".project" located in below folder using CubeIDE.
"C:\IOT\RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\platforms\STM32\Reference_Projects\Cube_Baremetal\Projects\SPI\sample_project"



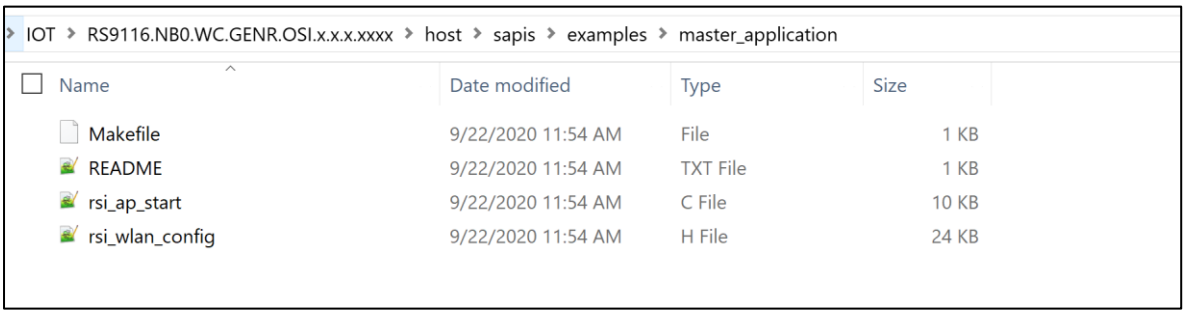
2. By default, this project maps to dummy example code located in
"C:\IOT\RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\sapis\examples\master_application"



3. To test specific example, navigate to folder
"C:\IOT\RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\sapis\examples\master_application"

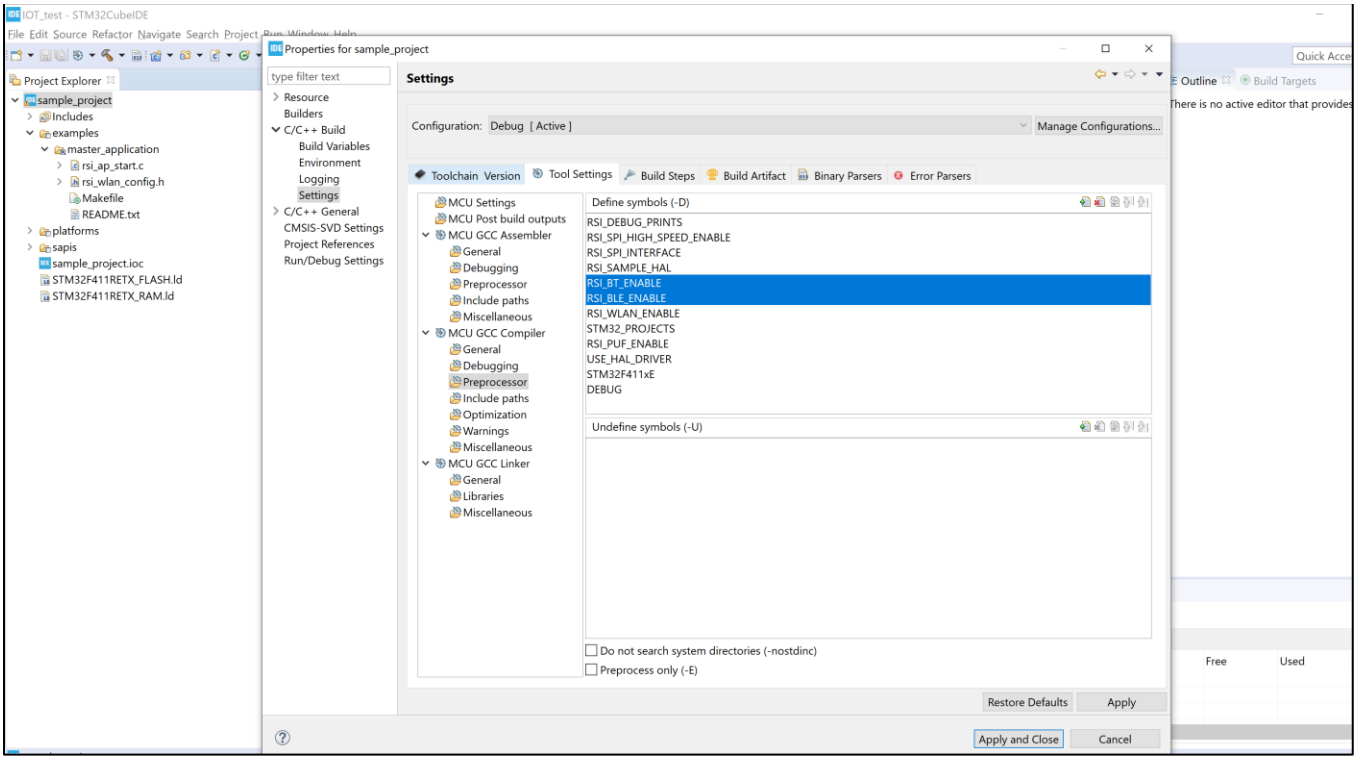


4. Delete the existing code, copy and paste the specific example code to above folder.
   Ex: Below screenshot shows the example code copied from "RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\sapis\examples\wlan\access_point\" to "RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\sapis\examples\master_application



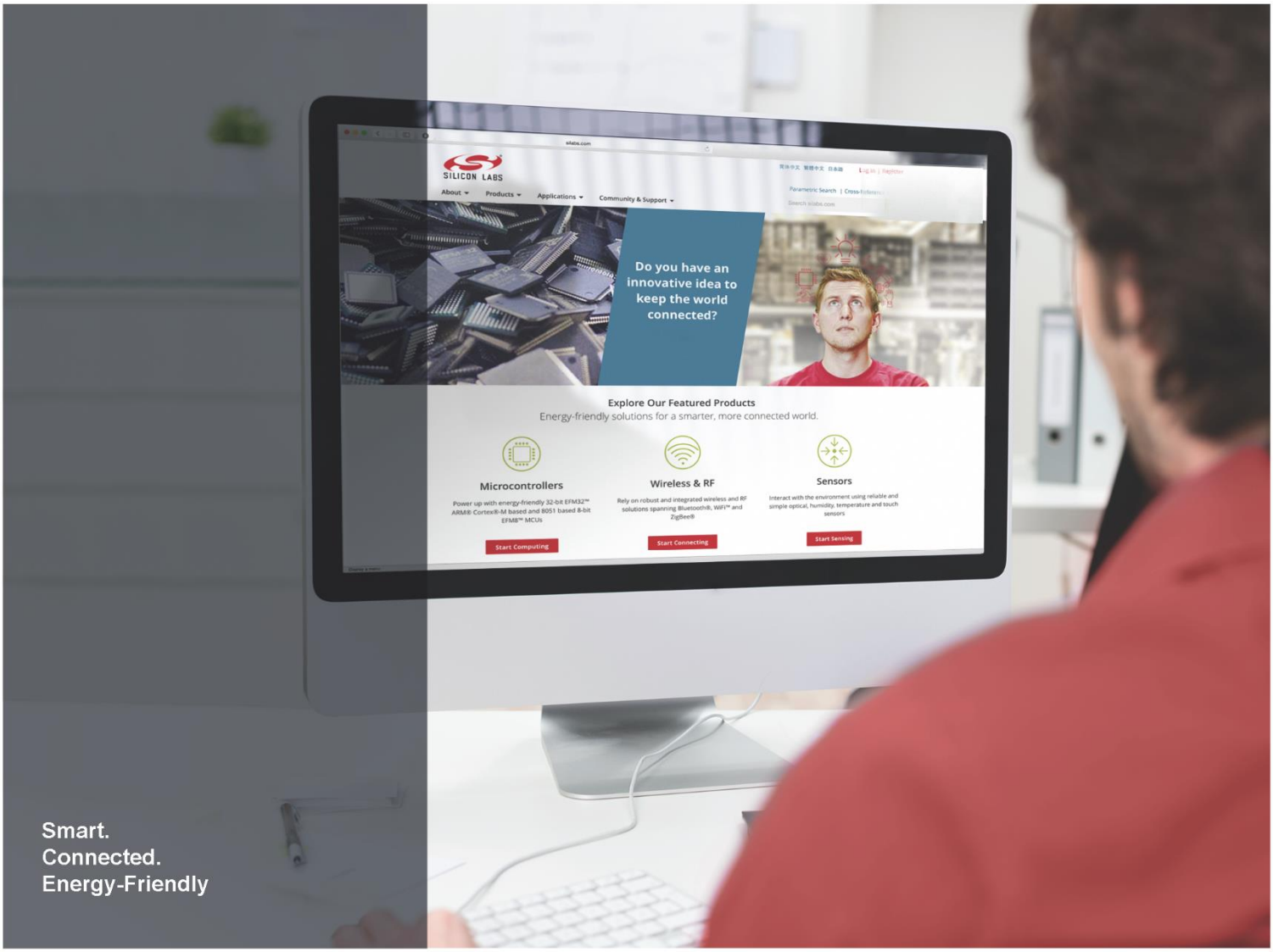
Note:

- sample_project compiles successfully only if all the header files rsi_wlan_config.h, rsi_bt_config.h and rsi_ble_config.h are present in "master_application" folder.
- In case if example code doesn't contain any of those header files, use the default files present in the "master_application" folder
  (or)
  Remove the corresponding protocol macros in STM32CubeIDE project settings. To change the project settings, right click on project → properties → C/C++ Build → settings → Tool Settings → MCU GCC Compiler → Preprocessor. As above example code doesn't have rsi_ble_config.h and rsi_bt_config.h, so remove RSI_BLE_ENABLE, RSI_BT_ENABLE. Refer to below screenshot for reference.



5. Save the settings and compile the project.
6. Repeat the same steps to compile different examples using this project.

> **Note:**
>
> Follow same steps to compile the sample_project located in "C:\IOT\RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\platforms\STM32\Reference_Projects\Cube_Freertos\Projects\SPI\sample_project"



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com