

Co-Ex Power Save Profile Application

User guide

Version 0.2

May 2016

Redpine Signals, Inc.

2107 N. First Street, #540

San Jose, CA 95131.

Tel: (408) 748-3385

Fax: (408) 705-2019

Email: info@redpinesignals.com

Website: www.redpinesignals.com

About this Document

This document describes the process of bringing up the RS9113 based module as a WiFi STA+BTLE co-ex mode with power save.

Disclaimer:

The information in this document pertains to information related to Redpine Signals, Inc. products. This information is provided as a service to our customers, and may be used for information purposes only. Redpine assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at Redpine's sole discretion without any prior notice to anyone. Redpine is not committed to updating this document in the future.

Copyright © 2015 Redpine Signals, Inc. All rights reserved.

Table of Contents

1	Introduction	4
1.1	Application Overview	4
1.1.1	Overview	4
1.1.2	Sequence of Events	4
1.1.2.1	WLAN Task	4
1.1.2.2	BLE Task	4
1.2	Application Setup	5
1.2.1	SPI based Setup Requirements	5
1.2.2	UART/USB-CDC based Setup Requirements	5
2	Configuration and Execution of the Application	7
2.1	Initializing the Application	7
2.1.1	SPI Interface	7
2.1.2	UART/USB-CDC Interface	7
2.2	Configuring the Application	7
2.2.1	Configuring the WLAN task	7
2.2.2	Configuring the BLE Application	12
2.3	Executing the Application	14

Table of Figures

Figure 1 Setup to demonstrate WLAN station BLE Power save Application	6
--	----------

Table of Tables

No table of figures entries found.

1 Introduction

This project is applicable to all the WiSeConnect variants like WiSeConnect Plus, WiSeMCU and WyzBee. The term WiSeConnect refers to its appropriate variant.

1.1 Application Overview

1.1.1 Overview

The coex application demonstrates how information can be exchanged seamlessly using two wireless protocols (WLAN and BLE) running in the same device using different connected power save profiles.

In this coex application, WiSeConnect BTLE device connects with remote BTLE device (Smart Phone) and WiSeConnect WiFi interface connects with an Access Point in station mode and configures device in Power Save profile mode 2 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 WiSeConnect 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 WiSeConnect device, the message will be sent to Smartphone via BLE task.

Thus messages can be seamlessly transferred between PC and Smartphone.

1.1.2 Sequence of Events

1.1.2.1 WLAN Task

This Application explains user how to:

- Configure device in station mode
- Connect WiSeConnect device to Access point
- Configure device in power save profile mode 2
- Open TCP client socket over SSL
- Establish TCP connection with remote TCP server using SSL
- Receive TCP data sent by connected station and forward to BLE task
- Send data received by BLE task to connected station using TCP protocol

1.1.2.2 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

1.2 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:

1.2.1 SPI based Setup Requirements

- Windows PC1 with CoCoX IDE
- Spansion (MB9BF568NBGL) micro controller

Note: If user does not have Spansion (MB9BF568NBGL) host platform, please go through the SPI-Porting guide [\sapis\docs\RS9113-WiSeConnect-SAPI-Porting-Guide-vx.x.pdf](#) for SAPIs porting to that particular platform.

- WiSeConnect device
- 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.

- Agilent power analyzer to measure power consumption

1.2.2 UART/USB-CDC based Setup Requirements

- Windows PC1 with Dev-C++ IDE
- WiSeConnect device
- 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.

- Agilent power analyzer to measure power consumption

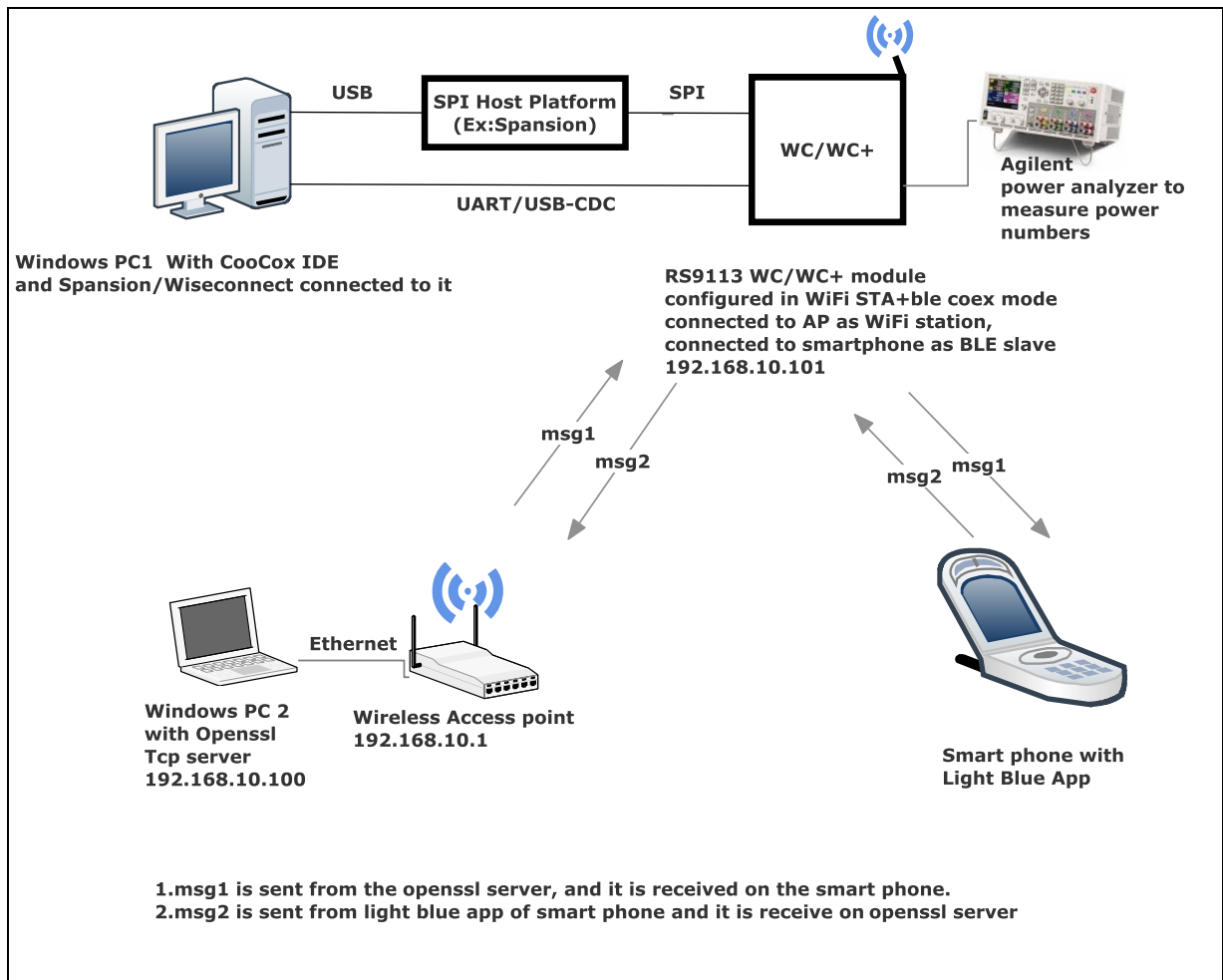


Figure 1 Setup to demonstrate WLAN station BLE Power save Application

2 Configuration and Execution of the Application

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.

2.1 Initializing the Application

2.1.1 SPI Interface

If User using SPI interface, Please refer the document *sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf* for opening the *power_save* example in CoCoX IDE.

2.1.2 UART/USB-CDC Interface

If User using UART interface, Please refer the document *sapis/platforms/windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf* for opening the *power_save* example in Dev-C++ IDE

2.2 Configuring the Application

2.2.1 Configuring the WLAN task

1. Open *sapis/examples/wlan_ble/power_save/rsi_wlan_app.c* file and update/modify following macros,

SSID refers to the name of the Access point.

```
#define SSID                "<ap name>"
```

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
“sapis/examples/utilities/certificates/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.
- 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 *sapis/examples/utilities/certificates*

DEVICE_PORT port refers SSL client port number

```
#define DEVICE_PORT <local port>
```

SERVER_PORT port refers remote SSL server port number

```
#define SERVER_PORT <remote port>
```

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 <server ip address>
```

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 ultra low power mode 2 (**RSI_SLEEP_MODE_2**) with maximum power save (**RSI_MAX_PSP**) with message based hand shake.

```
#define PSP_MODE            RSI_SLEEP_MODE_2  
#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  
                                     | TCP_IP_FEAT_SSL  
                                     | TCP_IP_FEAT_SINGLE_SSL_SOCKET  
                                     | TCP_IP_TOTAL_SOCKETS_2)  
#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_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
```

3. If user wants to select different power save mode profiles, Please go through the step #4 and #5 other wise skip step #4 and #5
4. Open *sapis/examples/wlan_ble/power_save/rsi_wlan_app.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 hand shake 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 hand shake or Message exchange, therefore handshake is required before sending the command to the module.

```
#define PSP_MODE RSI_SLEEP_MODE_2
```

Note1: For **RSI_SLEEP_MODE_2** and **RSI_SLEEP_MODE_8** modes, GPIO or Message based hand shake can be selected using **RSI_HAND_SHAKE_TYPE** macro which is define in *sapis/include/rsi_wlan_config.h*

Note2: In this example user can verify **RSI_SLEEP_MODE_2** with Message based hand shake. If user wants to verify other power modes, user has to change the application as well as GPIO hand shake 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/include/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
```

5. Open *sapis/include/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.

2.2.2 Configuring the BLE Application

1. Open *sapis/examples/wlan_ble/wlan_station_ble_bridge/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 WiSeConnect 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 10000
```

To enable power save

PSP_MODE refers power save profile mode. WiSeConnect device supports following power modes in BTLE,

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

RSI_SLEEP_MODE_2 (1) : This mode is applicable when module is in Advertising state as well as in connected state. 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 power mode, module goes to power save when it is in unassociated state with the remote device. In this sleep mode, SoC will go to sleep based on GPIO hand shake or Message exchange, therefore handshake is required before sending the command to the module.

```
#define PSP_MODE RSI_SLEEP_MODE_2
```

Note1:

For **RSI_SLEEP_MODE_2** and **RSI_SLEEP_MODE_8** modes, GPIO or Message based hand shake can be selected using **RSI_HAND_SHAKE_TYPE** macro which is defined in *sapis/include/rsi_wlan_config.h*

Note2:

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

PSP_TYPE refers power save profile type. WiSeConnect device supports following power save profile types in BTLE mode,

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.

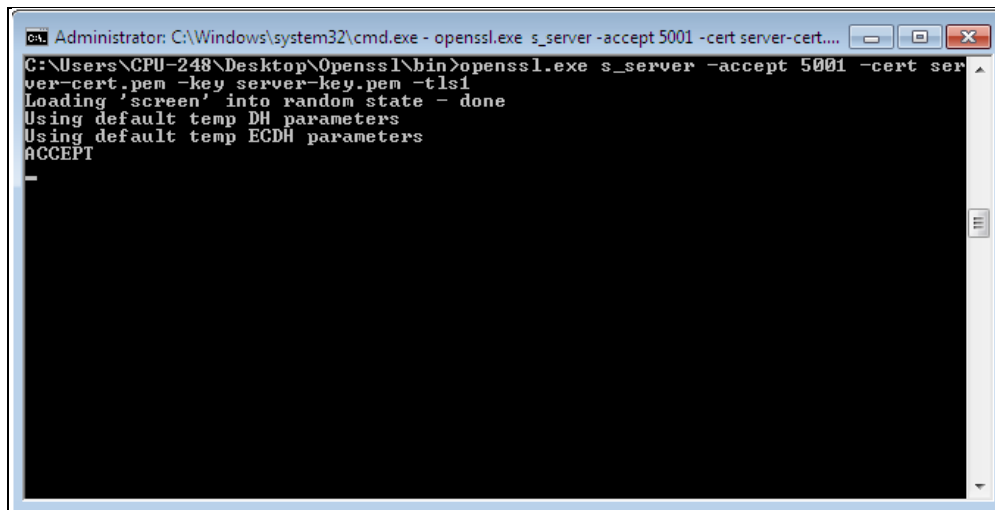
```
#define PSP_TYPE RSI_MAX_PSP
```

2.3 Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect WiSeConnect 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**



```
Administrator: C:\Windows\system32\cmd.exe - openssl.exe s_server -accept 5001 -cert server-cert...
C:\Users\CPU-248\Desktop\Openssl\bin>openssl.exe s_server -accept 5001 -cert server-cert.pem -key server-key.pem -tls1
Loading 'screen' into random state - done
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
```

3. SPI Interface

If User using SPI interface, Please refer the document [*sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*](#) for executing the **power_save** example in CooCox IDE.

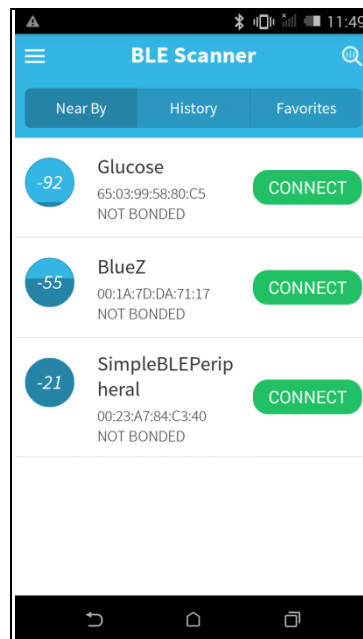
4. UART/USB-CDC Interface

If User using UART interface, Please refer the document [*sapis/platforms/windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*](#) for executing the **power_save** example in Dev-C++ IDE

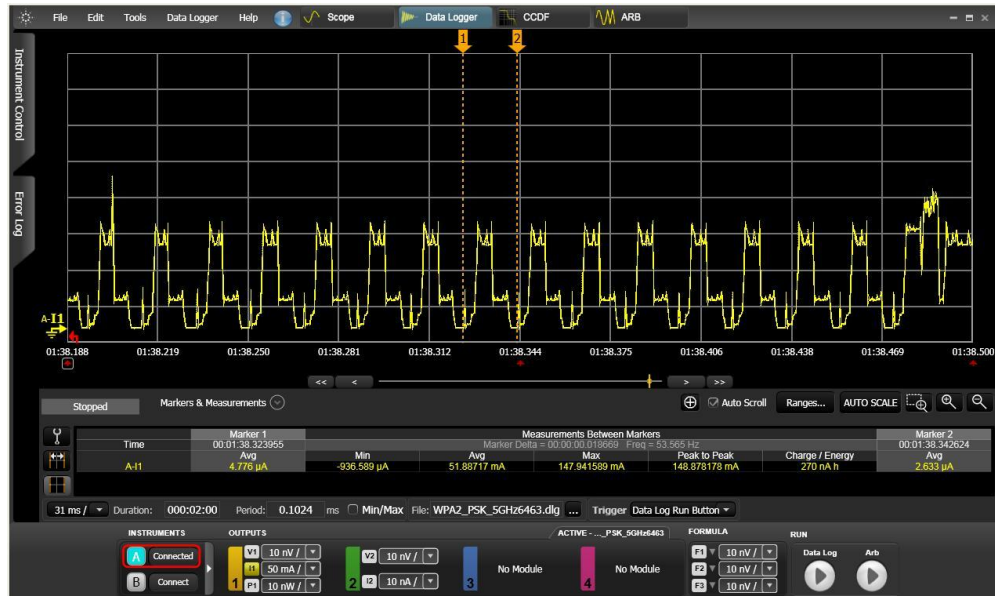
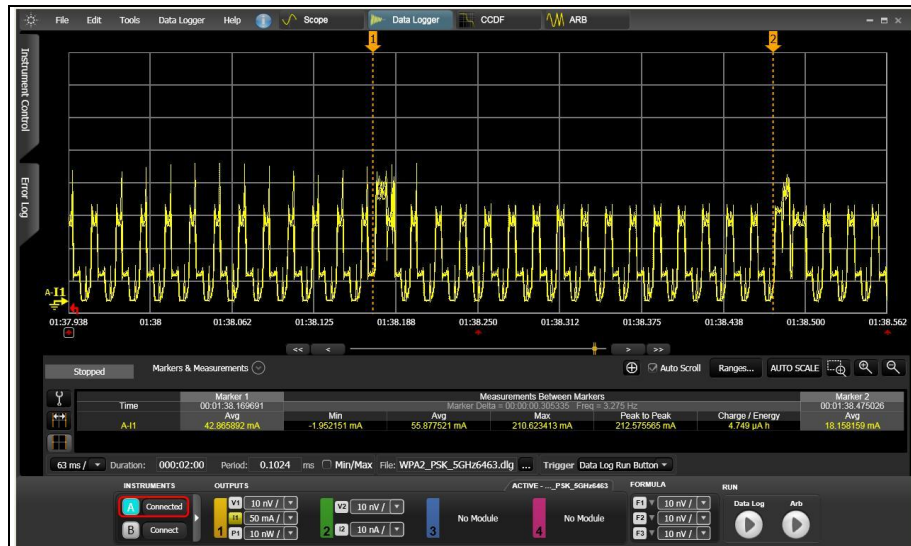
5. After the program gets executed, WiSeConnect BLE is in Advertising state and WLAN connects to Access Point and establishes SSL connectivity with SSL server opened on Windows PC1. Please refer the given below image for connection establishment at windows PC1,

```
Administrator: C:\Windows\system32\cmd.exe - openssl.exe s_server -accept 5001 -cert server-cert...
C:\Users\CPU-248\Desktop>openssl bin>openssl.exe s_server -accept 5001 -cert server-cert.pem -key server-key.pem -tls1
Loading 'screen' into random state - done
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
bad gethostbyaddr
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECAgMBBAIANQgghXA7mzU5ua9X1hKn3o/IK2CfcmUGOpUBf+0cjUJ1NwE
MB1940ijrLu4iwlcFL/c+/d0T8Sis46ze6F0sU0rdUubm0BNkQqSOHFnS1vIkpu0
26EGAgRXDk02ogQCAhwgPAYEBAEAAA=
-----END SSL SESSION PARAMETERS-----
Shared ciphers: AES256-SHA: AES128-SHA
CIPHER is AES256-SHA
```

6. Open a BLE scanner App in the Smartphone and do the Scan.
7. In the App, WiSeConnect 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 WiSeConenct device as internal name **"SimpleBLEPeripheral"**.



8. Initiate BLE connection from the App.
9. After successful connection of both WiFi and BTLE, Application issues power save profile command and module goes to sleep and wakes up for every DTIM interval of Access Point and for every BTLE connection interval. Please refer the given below image for power profile cycle after successful connection of both WiFi and BTLE.



Note1:

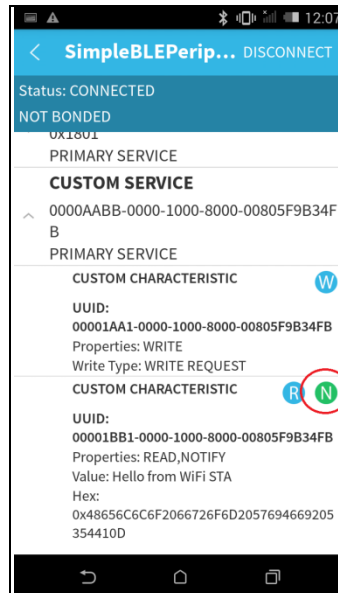
In WiFi Access Point, Beacon interval and DTIM are configured to 100 and 3 and In BTLE, Connection interval of Master device is 18ms. So, Module waking up for every DTIM interval (i.e 300ms) as well as every BTLE connection interval (i.e 18ms) in above power save profile pictures.

Note2:

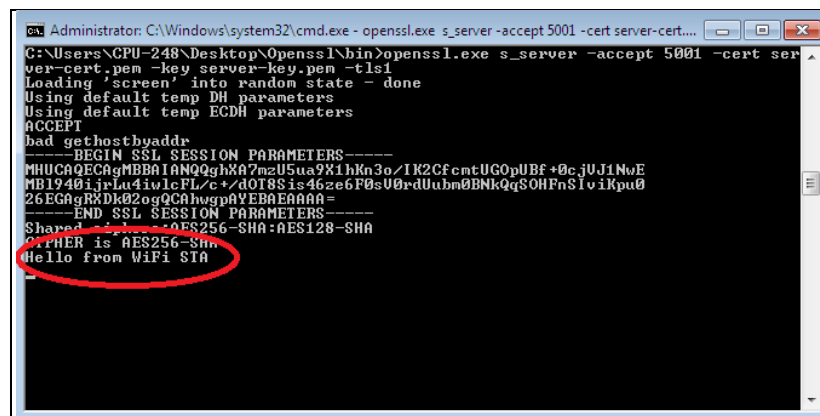
Power save profile (Or Avg power consumption) may vary based on the data traffic. So, user does not see exact image as shown above while running data traffic.

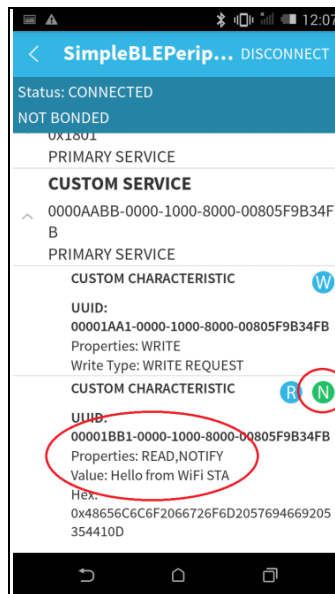
10. After successful BTLE connection, BLE scanner displays the supported services of WiSeConnect module.
11. Select the attribute service which is added **RSI_BLE_NEW_SERVICE_UUID**

(Ex: 0xAABB) and enable Notification for attribute UUID
RSI_BLE_ATTRIBUTE_2_UUID (Ex: 0x1BB1) to receive data sent
by WiFi STA.



12. Now from SSL server (windows PC1), send a message (Ex: “Hello from WiFi STA”) to WiSeConnect device. WiSeConnect device forwards the received message from SSL server to remote BTLE device which is connected to WiSeConnect 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.

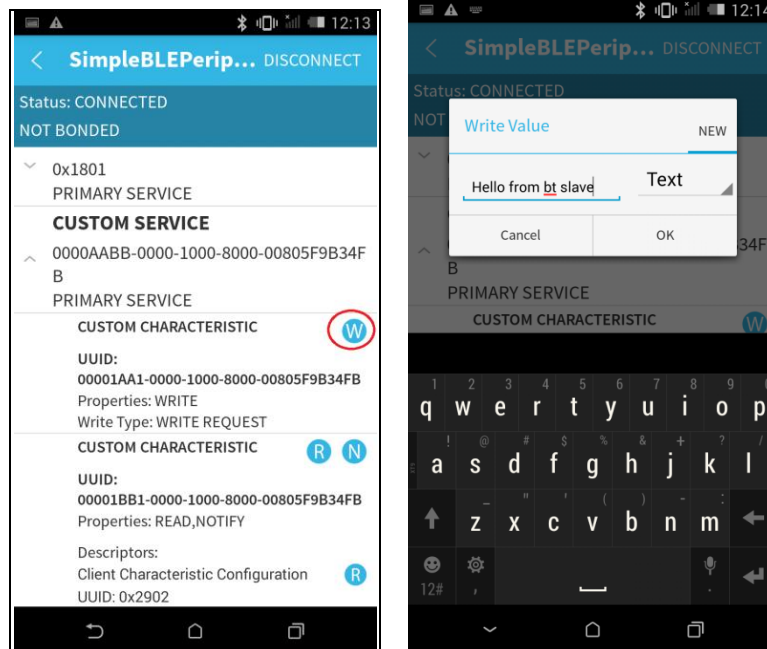


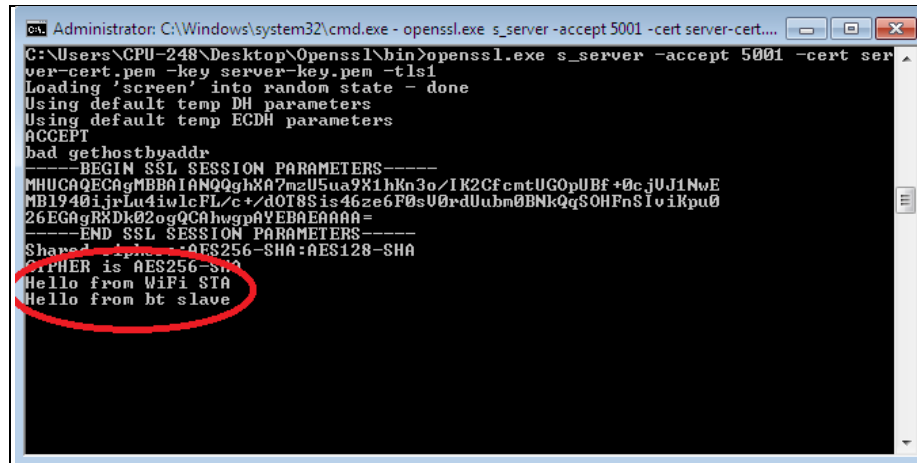


Note:

rsi_wlan_app_send_to_btble() function defined in rsi_ble_app.c to send message from WLAN task to BTLE task

13. 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 WiSeConnect device. WiSeConnect device forwards the received message from BTLE remote device to SSL server over WiFi protocol. User can observe the message on UDP socket application.





```
C:\Users\CPU-248\Desktop\Openssl\bin>openssl.exe s_server -accept 5001 -cert server-cert.pem -key server-key.pem -tls1
Loading 'screen' into random state - done
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
bad gethostbyaddr
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECagMBBBIANQghXA7mzU5ua9X1hKn3o/LK2CfcmtUGOpUBf+0cjUJ1NwE
MB1940i jrLu4iwlcFL/c+/dOT8Sis46ze6F0sU0rdUubm0BNkQqSOHFnSivikpu0
26EGAgRXdK02ogQCAhwgpaVEBAEAAAA=
-----END SSL SESSION PARAMETERS-----
Shared secret: AES256-SHA:AES128-SHA
Cipher is AES256-SHA
Hello from WiFi STA
Hello from bt slave
```

Note:

rsi_bt_app_send_to_wlan() function defined in rsi_wlan_app.c is to send message from BTLE task to WLAN task