

UG454: RS9116W with STM32 User's Guide

Version 1.2

10/21/2020

Table of Contents

1	About this Document	3
2	Getting Started with STM32 and RS9116W EVK	4
2.1	RS9116W Evaluation Kit Contents	4
2.2	RS9116W EVK Overview	5
2.2.1	The RS9116 WiSeConnect	5
2.2.2	Hardware Details	6
2.2.3	Supported Interfaces	6
2.3	RS9116W Hardware Requirements	7
2.4	RS9116W Software Requirements	8
2.5	RS9116W SPI headers on the EVK	9
2.5.1	Header Pin Orientations	9
2.5.2	Pin Description	9
2.6	Interfacing the STM32 NUCLEO with RS9116W EVK	10
2.6.1	Steps for Interfacing via SPI Interface	10
2.6.2	Steps for Interfacing via UART Interface	11
2.6.3	10-pin SPI Header Connection Details	11
2.6.4	UART Connection Details	11
2.6.5	GPIO Connection Details for ULP Power Save	12
2.6.6	GPIO Connection Details For LP Power Save	12
2.6.7	For Host MCU-based Reset	12
2.7	Getting Started with Keil IDE	12
2.7.1	Introduction to Keil	12
2.7.2	Steps for Executing STM32 Reference Projects on the Keil IDE	13
2.8	Getting Started with STM32CubeIDE	22
2.8.1	Please follow the below Steps for executing STM32 reference projects on STM32CubeIDE	22
3	STM32 Reference Projects	34
3.1	STM32 Reference Projects and Example Details	34
3.1.1	1. Reference projects details	34
3.1.2	2. Reference Examples Details	35
3.2	Steps for Executing STM32 Examples using Master Application (Sample Project)	35
3.2.1	Steps for Keil IDE:	35
3.2.2	Steps for STMCube IDE	40
3.3	Reference Projects for Keil Baremetal	44
3.3.1	Example1: Throughput Application	44
3.3.2	Example2: Wireless Firmware Upgradation	53
3.3.3	Example3: WLAN Station BLE Bridge	58
3.3.4	Example4: WLAN Station BLE Provisioning	73
3.3.5	Example5: WLAN Standby Associated Power Save	83
3.3.6	Example6: Enterprise Ping Client (eap)	91
3.3.7	Example7: BT_Alone	105
3.3.8	Example8: AWS_IoT	112
3.3.9	Example9: sample_project	116
3.3.10	Example10: udp_client	117
3.4	Reference Projects for Keil Freertos	123
3.4.1	Example1: mqtt_client	123
3.4.2	Example2 : wlan_https_bt_spp_ble_dual_role	130
3.4.3	Example3: wlan_https_bt_spp_ble_provisioning	137
3.4.4	Example4: wlan_throughput_bt_spp_ble_dual_role	143
3.4.5	Example5: sample_project	148
3.4.6	Example6: udp_client	148
3.5	Reference projects for Cube Baremetal	148
3.5.1	Example1: eap	148
3.5.2	Example2: Firmware_Upgrade	148
3.5.3	Example3: Power_save	148
3.5.4	Example4: Wlan_ble	149
3.5.5	Example5: sample_project	149
3.6	Reference Projects for Cube Freertos	149
3.6.1	Example1: wlan_https_bt_spp_ble_dual_role	149
3.6.2	Example2: wlan_https_bt_spp_ble_provisioning	149
3.6.3	Example3: wlan_throughput_bt_spp_ble_dual_role	149
3.6.4	Example4: sample_project	149
4	FreeRTOS Porting for STM32	150
5	AWS SDK Porting for RS9116W SAPIS in STM32	166
6	Revision History	173
7	References	174

1 About this Document

This document provides the following information:

1. The RS9116W Module's Evaluation Kit (EVK)
2. Evaluating RS9116W EVK in **WiSeConnect** mode with Host MCU as STM32 NUCLEO-F411RE
3. RS9116W Interface pin configurations (for SPI, UART, and Power Save) with STM32
4. STM32 projects execution on the KEIL and STM32CubeIDE

Note: STM32 NUCLEO-F411RE will be referred to as STM32 or STM32 NUCLEO in further documentation

2 Getting Started with STM32 and RS9116W EVK

2.1 RS9116W Evaluation Kit Contents

The RS9116 Module Evaluation Kit comes with the following components:

1. RS9116 I/O Baseboard
2. RS9116 Wireless Daughter Card
3. Micro A/B-type USB cable
4. SDIO Connector*
5. SPI Connector

Note: The SDIO host interface is currently not supported

1) Single Band EVK:

The below image is for the RS9116 connectivity Single Band (2.4Ghz) Evaluation kit. This contains the SDIO adaptor cable, SPI cable, IO baseboard, and Wireless daughter card.

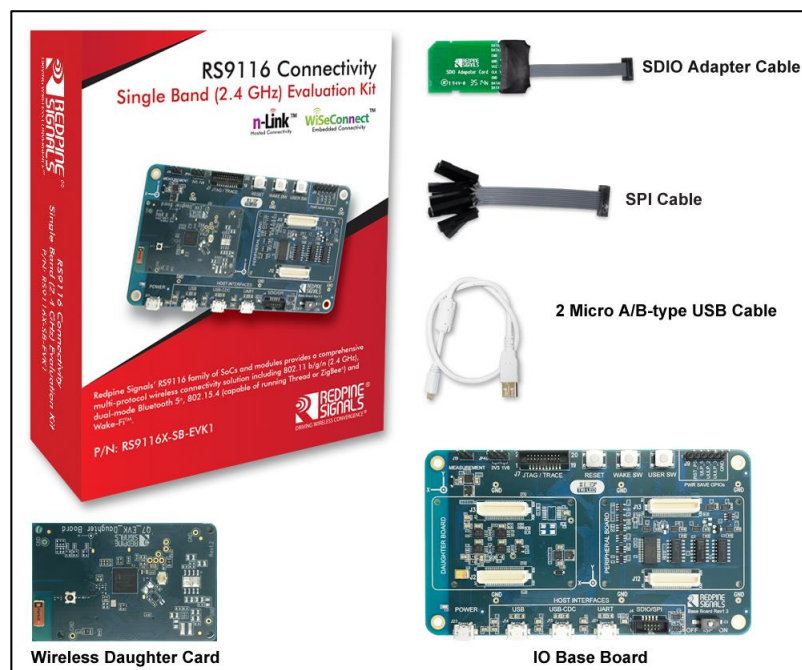


Figure 1: Single band Evaluation Kit Contents

2) Dual Band EVK:

The below image is for the RS9116 connectivity Dual Band (2.4/5 GHz) Evaluation kit. This contains the SDIO adaptor cable, SPI cable, IO baseboard, and Wireless daughter card.



Figure 2: Dual Band Evaluation Kit Contents

It is highly recommended to use the Micro A/B type USB cable that comes with the kit. If a longer cable is needed, ensure to use a USB-IF certified cable which can supply a peak current of at least 500mA.

Note:

For WiSeConnect, the user needs only the Micro A/B-type USB cable and the SPI connector.

For n-link, the user needs only the Micro A/B-type USB cable.

Latest EVK user guide, firmware, and reference projects can be downloaded from our website:
<https://docs.silabs.com/rs9116/>

2.2 RS9116W EVK Overview

2.2.1 The RS9116 WiSeConnect

The RS9116 WiSeConnect module family is based on Silicon Labs RS9116 ultra-low-power, single spatial stream, dual-band 802.11n + BT 5/BLE Convergence SoC. The RS9116 Module Evaluation Kit (EVK) is a platform for evaluating the RS9116 modules with multiple Host Processors/MCUs over interfaces like SDIO*, USB, USB-CDC, SPI and UART.

The EVK includes sample driver, supplicant, applications to test the following:

- Wireless Functionality for Wi-Fi, BT/BLE
- Wireless co-existence
- Security modes
- Throughputs
- Power Consumption
- Firmware Upgrade

Solution Highlights

- Offers WLAN, and Bluetooth protocols along with Wi-Fi Direct™, WPA/WPA2-PSK, WPA/WPA2-Enterprise (EAP-TLS, EAP-FAST, EAP-TTLS, PEAP-MS-CHAP-V2, LEAP).

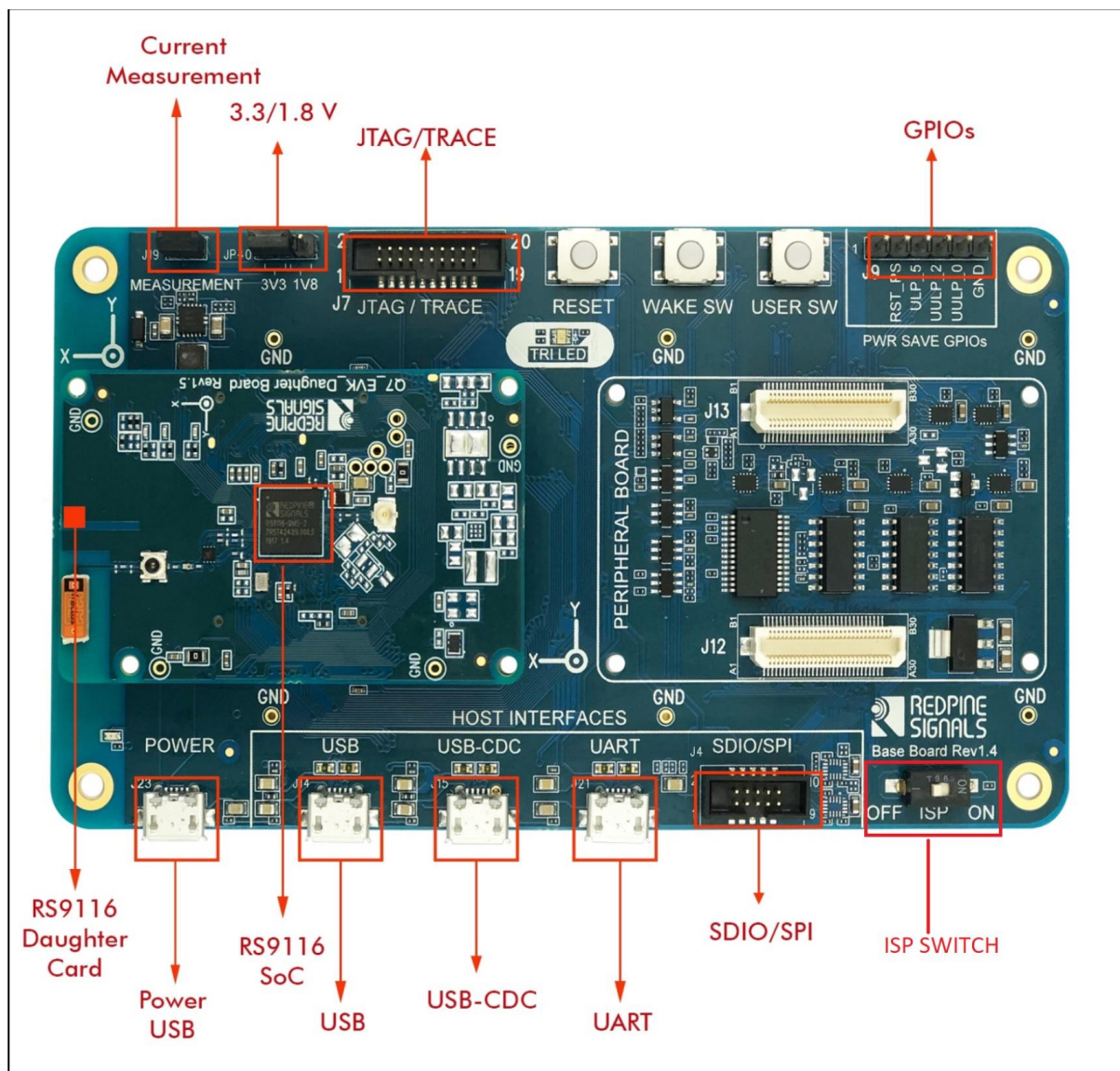
- Provides a feature-rich networking stack embedded in the device, thus providing a fully-integrated solution for embedded wireless applications.
- Can be interfaced to 8/16/32-bit host processors through SPI, SDIO*, UART, USB and USB-CDC interfaces.
- Integrates a multi-threaded MAC processor with integrated analog peripherals and support for digital peripherals, baseband digital signal processor, analog front-end, crystal oscillator, calibration OTP memory, dual-band RF transceiver, dual-band high-power amplifiers, baluns, diplexers, diversity switch and Quad-SPI Flash thus providing a fully-integrated solution for embedded wireless applications.
- Operates in industrial (-40°C to +85°C) temperature range.
- Choice of several module packages (with and without antenna) options depending on system requirements.
- Co-existence of multiple wireless protocols managed by an internal protocol arbitration manager.

2.2.2 Hardware Details

This section describes RS9116 EVK's various components and headers.

The OneBox-Embedded software for the WiSeConnect modules supports UART, SPI, USB and USB-CDC interfaces to connect to the Host MCU.

As shown in the image below, the RS9116 EVK has four USB connectors for the Power, USB, USB-CDC and UART connections. The UART signals of the module are converted to USB using an onboard circuit. The board also has SDIO*/SPI Header.



2.2.3 Supported Interfaces

The board is designed to configure the EVK module to use the interface on which the power supply is detected. The SDIO* and SPI interfaces require a power supply to be provided to the EVK module over the POWER port using a

USB cable. Hence, for these interfaces on the EVK module, it is required that the USB Power connection will be provided first, followed by the SDIO* or SPI connection.

Follow the below steps to use the EVK module with different interfaces:

1. USB, UART, USB-CDC Modes

- a. Connect the Micro A/B-type USB cable between a USB port of a PC/Laptop and the micro-USB port labeled USB, UART or USB-CDC on the EVK.

2. SPI Mode

- a. Connect the Micro A/B-type USB cable between a USB port of a PC/Laptop and the micro-USB port labeled POWER on the EVK.
- b. Connect the 10-pin header of the SPI Adaptor Cable to the EVK. Connect the other wires of this connector to the SPI signals of a Host MCU platform. The details of the SPI header are given in [RS9116W SPI headers on the EVK](#).

3. SDIO Mode*

- a. Connect the Micro A/B-type USB cable between a USB port of a PC/Laptop and the micro-USB port labeled POWER on the EVK.
- b. SDIO* adaptor cable connector has two ends. Connect the one end 10-pin header of the SDIO* Adaptor Cable to the EVK module. And Connect the other end of the connector to the SDIO* signals of the Host MCU platform. The details of the 10-pin Header are given in the following sections.

4. ISP Switch

ISP switch shall be used for In-system programming firmware downloading utility.

Make sure the ISP switch is in the OFF state. If it is ON state you will not get the boot loader messages.

There is a 2-pin inline jumper available for measuring the current being sourced by the module during different stages of operation. This is labeled as "MEASUREMENT" on the baseboard. The user may connect a power meter or an ammeter to this jumper to measure the current.

*The SDIO host interface is currently not supported.

Note






If the baseboard Rev is 1.1 or below, then follow the below procedure:

1. For SDIO*/SPI, insert the USB into the Power port first before the SDIO*/SPI connector is connected to the Host platform.
2. For USB and USB-CDC, please connect the USB port to the Host platform first before connecting the USB for the Power port.

2.3 RS9116W Hardware Requirements



RS9116W EVK

	<p>STM32 NUCLEO-F411RE board (not included in EVK Kit)</p> <p>Click here for Specifications</p>
	<p>SPI Cable</p>
	<p>Two Micro A/B-type USB Cables (One Micro A/B-type USB cable to connect between PC and STM32 board) (Second cable connect between PC and EVK power port)</p>
	<p>Access point (2.4GHz) 802.11 b/g/n compliant Wi-Fi AP</p>
	<p>For STM32CubeIDE: Integrated development environment PC with 64 bit Windows OS (x86 arch). Cube IDE works only on 64-bit systems. For Keil IDE: PC with 64 bit Windows OS (x86 arch).</p>

Note

All the user documents, firmware release packages, certifications of the module and other materials related to the RS9116 based Modules are available on the website <https://docs.silabs.com/rs9116>.

2.4 RS9116W Software Requirements

Requirement for STM32CubeIDE

1. STM32CubeIDE v1.0.2
2. Java Platform SE binary (x86 only)

Requirement for Keil IDE

1. Keil uVision5 v5.24.2.0

Other Third-Party Software

1. iperf
2. Python 2.7.0
3. STMCubeMX

2.5 RS9116W SPI headers on the EVK

2.5.1 Header Pin Orientations

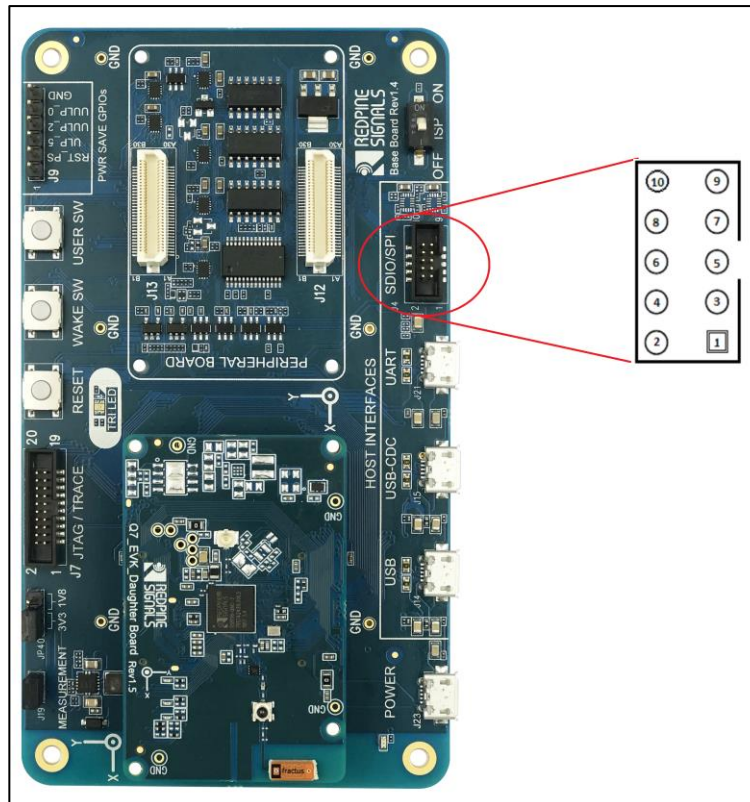
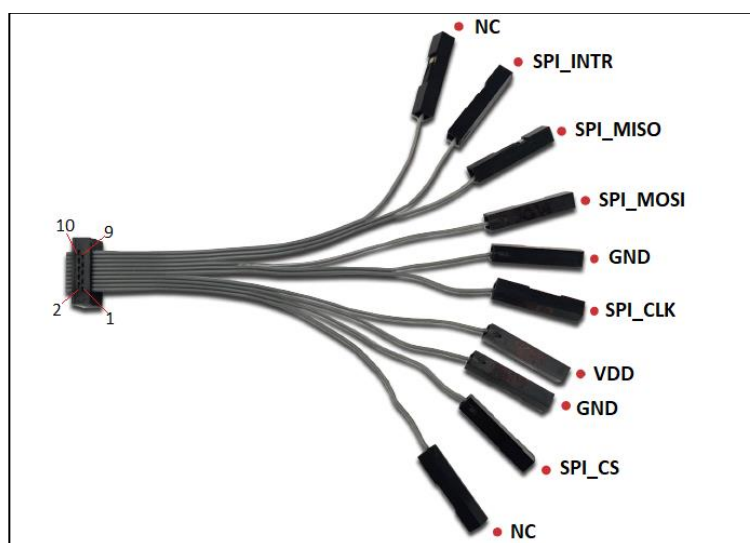


Figure 3: SPI header Pin Orientations



2.5.2 Pin Description

Table 1: SPI Header Pins

Pin Number	Pin Name	Direction	Description
1	NC	-	This pin must be left open.

Pin Number	Pin Name	Direction	Description
2	SPI_CS	Input	SPI slave select from the host (active low)
3	GND	-	Ground
4	VDD	-	Supply voltage
5	SPI_CLK	Input	Serial clock input from the host. The clock can be up to 80 MHz
6	GND	-	Ground
7	SPI_MOSI	Input	SPI data input
8	SPI_MISO	Output	SPI data output
9	SPI_INTR	Output	Active high-level triggered interrupt, used in SPI mode. The interrupt is raised by the EVB to indicate there is data to be read by the Host.
10	NC	-	No Connection

Note

Signal Integrity Guidelines for SPI/SDIO interface: Glitches in the SPI/SDIO clock can potentially take the SPI/SDIO interface out of synchronization. The quality and integrity of the clock line need to be maintained. In case a cable is used for the board to board connection, the following steps are recommended (please note that this is not an exhaustive list of guidelines and depending on individual cases additional steps may be needed.):

1. Minimize the length of the SPI/SDIO bus cable to as small as possible, preferably to within an inch or two.
2. Increase the number of ground connections between the EVB and the Host processor PCB.

2.6 Interfacing the STM32 NUCLEO with RS9116W EVK

2.6.1 Steps for Interfacing via SPI Interface.

1. Connect the RS9116W EVK to the STM32 NUCLEO board using the 10-pin SPI header connector.
2. Connect the STM32 with Mini USB cable to the Windows PC.
With this connection, STM32 is powered-up and also power-up the 9116EVK via SPI connected.
3. It is also recommended to power-up the RS9116 EVK by connecting the USB cable between the PC and POWER interface of the 9116 EVK.

Now the STM32 NUCLEO board is operational to execute projects using STM32CubeIDE or Keil IDE on the Windows PC.

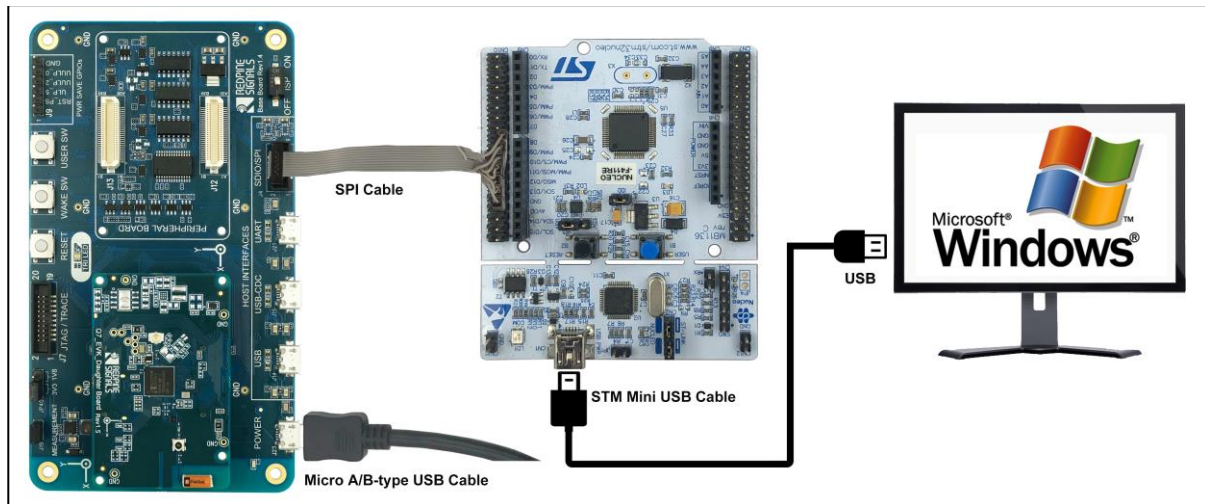


Figure 4: STM32 NUCLEO Board with RS9116W EVK

2.6.2 Steps for Interfacing via UART Interface

1. Connect the RS9116W EVK to the STM32 NUCLEO board using the GPIO peripheral card.
2. Connect the STM32 with Mini USB cable to the Windows PC.
With this connection, STM32 is powered-up and also power-up the 9116EVK via the UART connected.
3. It is also recommended to power-up the RS9116 EVK by connecting the USB cable between the PC and POWER interface of the 9116 EVK.

Pin Interfacing details - RS9116 EVK to STM32

2.6.3 10-pin SPI Header Connection Details

Table 2: SPI Connections

RS9116 EVK SPI J4 Header Pin #	STM32 Nucleo CN10 Header Pin #	Function
2	17	CS
3	9	GND
4	7	VCC
5	11	CLK
7	15	MOSI
8	13	MISO
9	21	INT

2.6.4 UART Connection Details

With GPIO peripheral Card on RS9116W EVK and with RS9116W Module

Table 3: UART Connections

RS9116 EVK GPIOs peripheral J1 Header Pin #	RS9116 Module	STM32 Nucleo CN10 Header Pin #	Function
11	GPIO_8	21	TX

RS9116 EVK GPIOs peripheral J1 Header Pin #	RS9116 Module	STM32 Nucleo CN10 Header Pin #	Function
13	GPIO_9	33	RX
GND	GND	GND	GND

2.6.5 GPIO Connection Details for ULP Power Save

Table 4: ULP Power Save Connections

RS9116 PWR SAVE GPIOs peripheral J9 Header Pin	STM32 Nucleo CN10 Header Pin #	Function	Comment
UULP_2	2	GPIO	
UULP_0	4	GPIO	RS9116 EVK Rev 1.4 Base Board + Rev 1.5 Daughter Board and above version
UULP_3			RS9116 EVK Below Rev 1.4 Base Board + Rev 1.5 Daughter Board

2.6.6 GPIO Connection Details For LP Power Save

Table 5: LP Power Save Connections

RS9116 PWR SAVE GPIOs peripheral J9 Header Pin	STM32 Nucleo CN10 Header Pin #	Function	Comment
UULP_5	2	GPIO	
UULP_0	4	GPIO	RS9116 EVK Rev 1.4 Base Board + Rev 1.5 Daughter Board and above version
UULP_3			RS9116 EVK Below Rev 1.4 Base Board + Rev 1.5 Daughter Board

2.6.7 For Host MCU-based Reset

Table 6: MCU-based Reset

RS9116 PWR SAVE GPIOs peripheral J9 Header Pin	STM32 Nucleo CN10 Header Pin #	Function	Comment
RST_PS	6	RESET	Reset from the host via GPIO

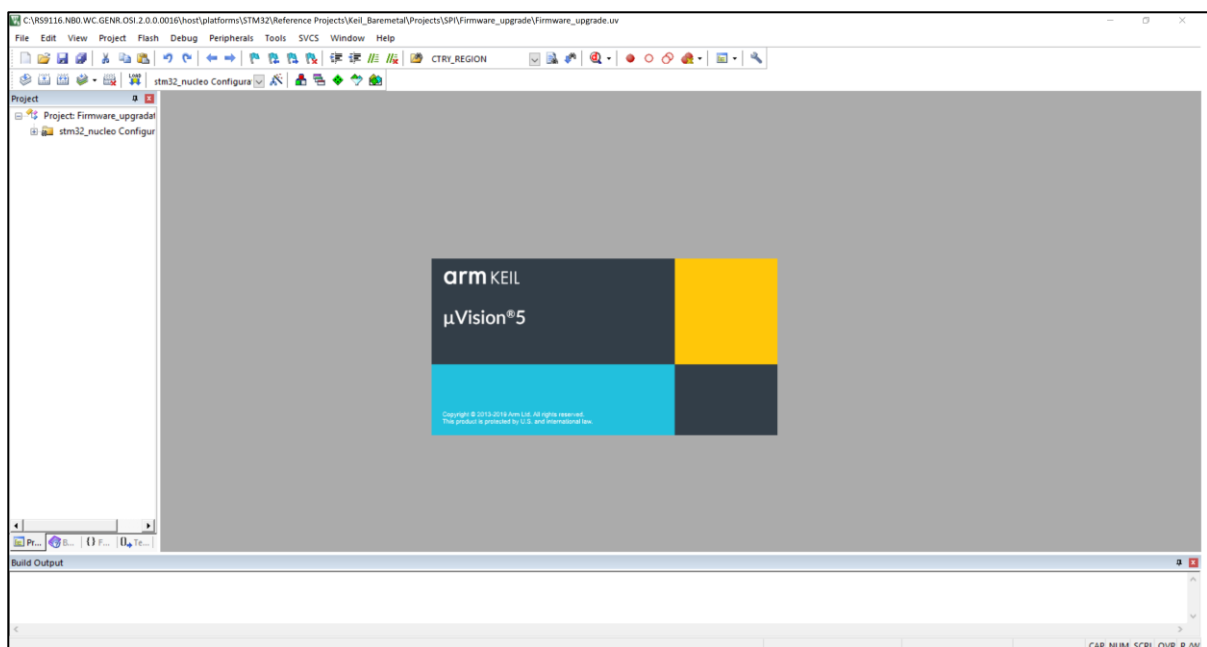
2.7 Getting Started with Keil IDE

2.7.1 Introduction to Keil

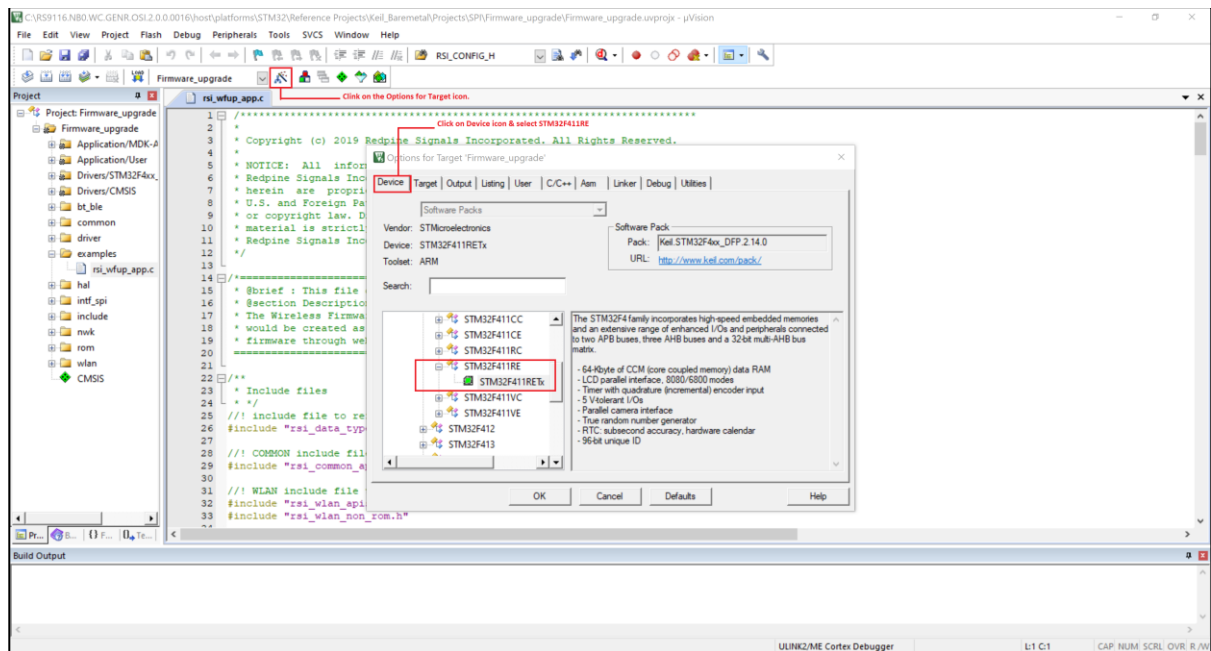
Keil MDK is the software development environment for a wide range of Arm Cortex-M based microcontroller devices. It includes the µVision IDE and debugger, ARM C/C++ compiler, and essential middleware components. KEIL IDE supports Windows OS only.

2.7.2 Steps for Executing STM32 Reference Projects on the Keil IDE

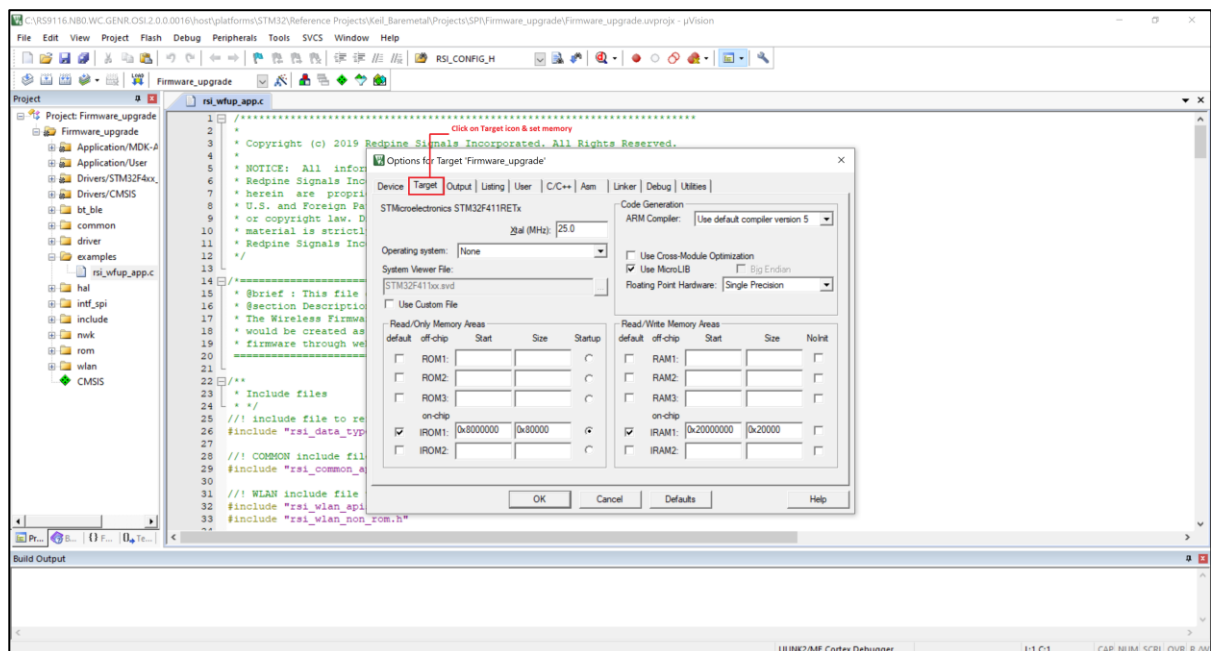
1. Connect the setup environment of RS9116 EVK and STM32 NUCLEO with the PC
 - a. Make sure SPI cable connected between RS9116 EVK and STM32 board
 - b. Connect the STM32 board power port to the PC (via USB cable)
 - c. It is recommended to connect the POWER port of RS9116 EVK to the PC (via USB cable)
2. Download and Install KEIL IDE from the below link.
<https://www.keil.com/demo/eval/arm.htm#/DOWNLOAD>
 Please note the above downloaded Keil IDE is for **Evaluation purposes only** and the usage is limited to 32KB program sizes. This Keil Evaluation version is not sufficient to compile and build STM32 projects.
 So Please get the **licensed version** of the KEIL IDE to execute STM32 Reference projects and Examples in the package.
3. Navigate to the below Release package link for accessing Keil projects.
RS9116.NB0.WC.GENR.OSI.x.x.x.xx/host/platforms/STM32/Reference_Projects/Keil_Baremetal/Projects/SPI
 Open any project folder (in the SPI folder) as per the user requirement and double-click the ".uVision5 Project" and then it will be redirected to KEIL IDE for viewing the project source.
Note: Keil_Baremetal projects are non-OS based projects.



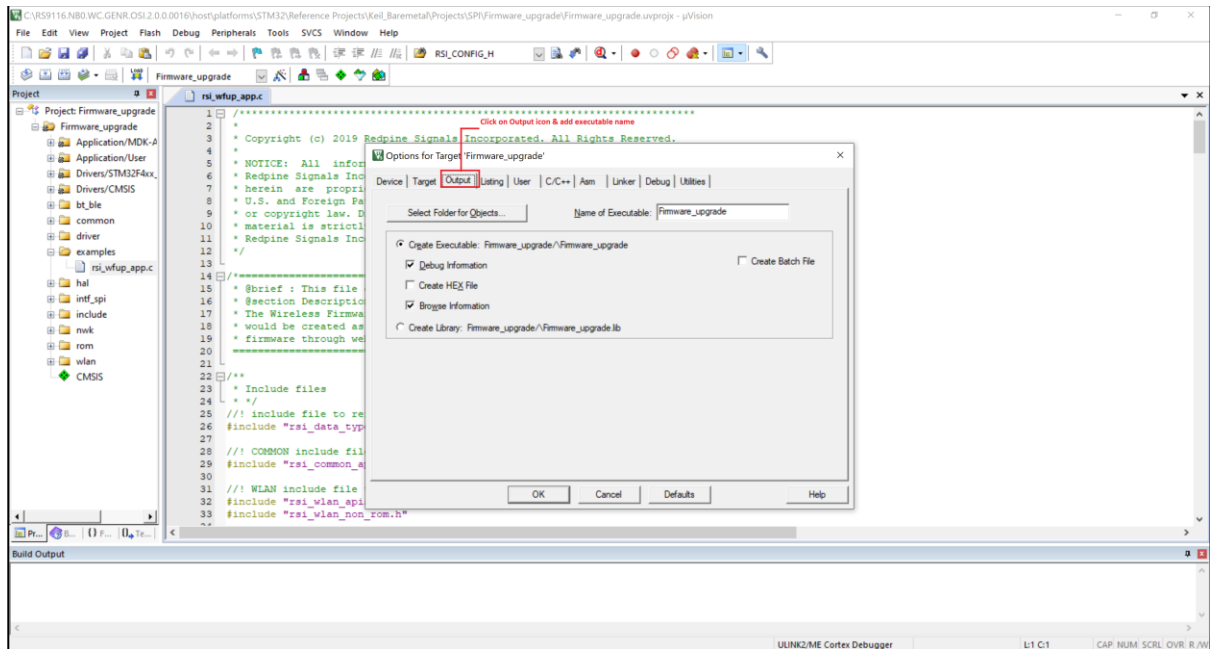
4. Please follow Steps 5 to 13 for settings project configuration for any Keil project.
 Please follow Steps 14- 16 for Compile/building the project and then debug on the STM32 target.
5. Click on the 'Options for Target icon'. after a click, one popup window will open. and then click on the 'Device' icon and select the device STM32F411RE. Only need to select device STM32F411RE if earlier not included in the project.



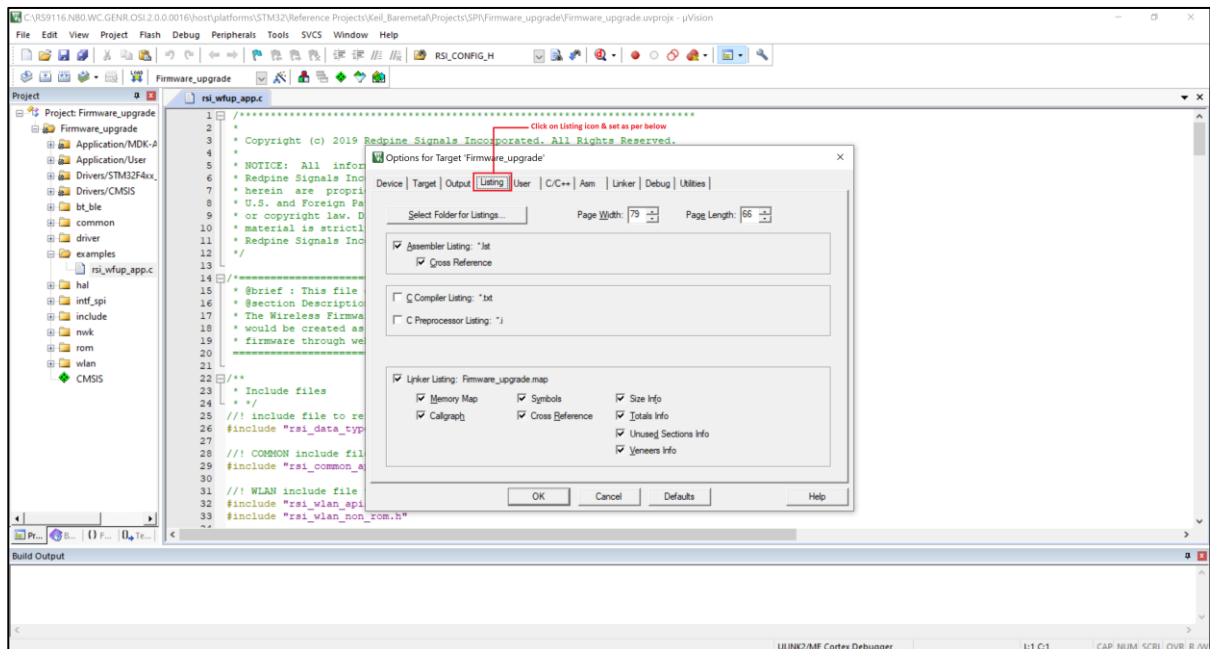
6. Then click on the 'Target' icon. Update the Xtal frequency according to requirements. by default it is 25MHz. Also, verify the Read/Write Memory Area section.



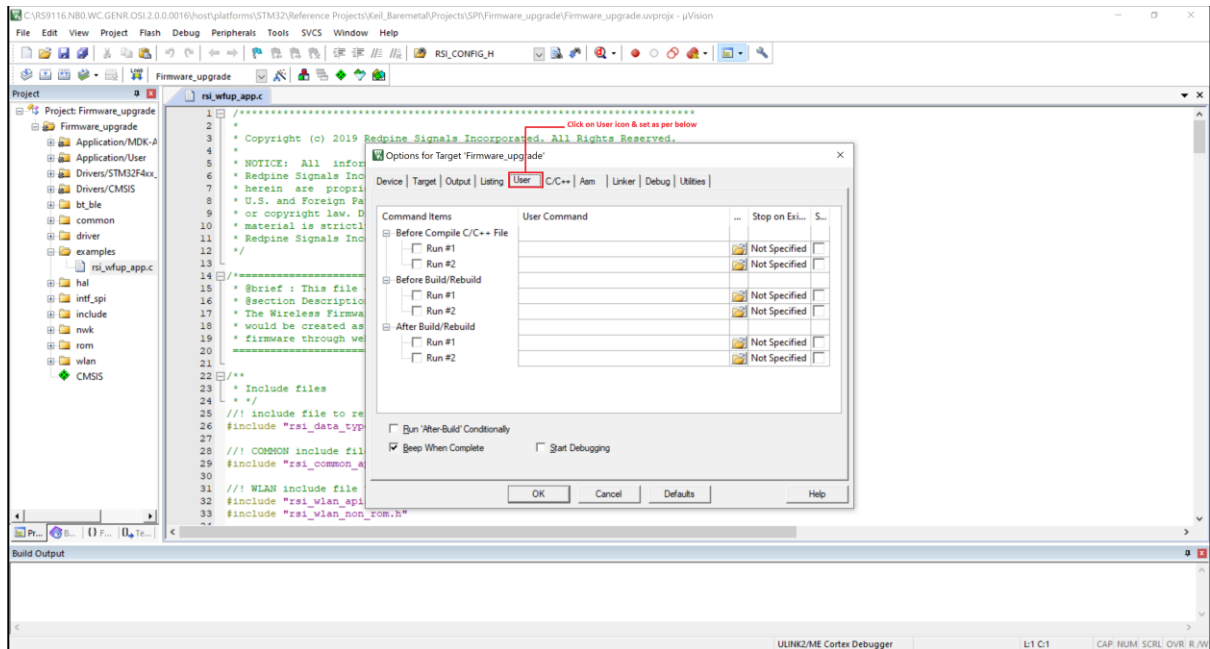
7. Then click on the 'Output' icon. Enter the Executable name and then select the 'Create Executable:' option as shown below Image.



8. Then click on the 'Listing' icon. Select the 'Assembler Listing' option and also select the 'Linker Listing' section as shown below Image.



9. Then click on the 'User' icon and set as per the below image.

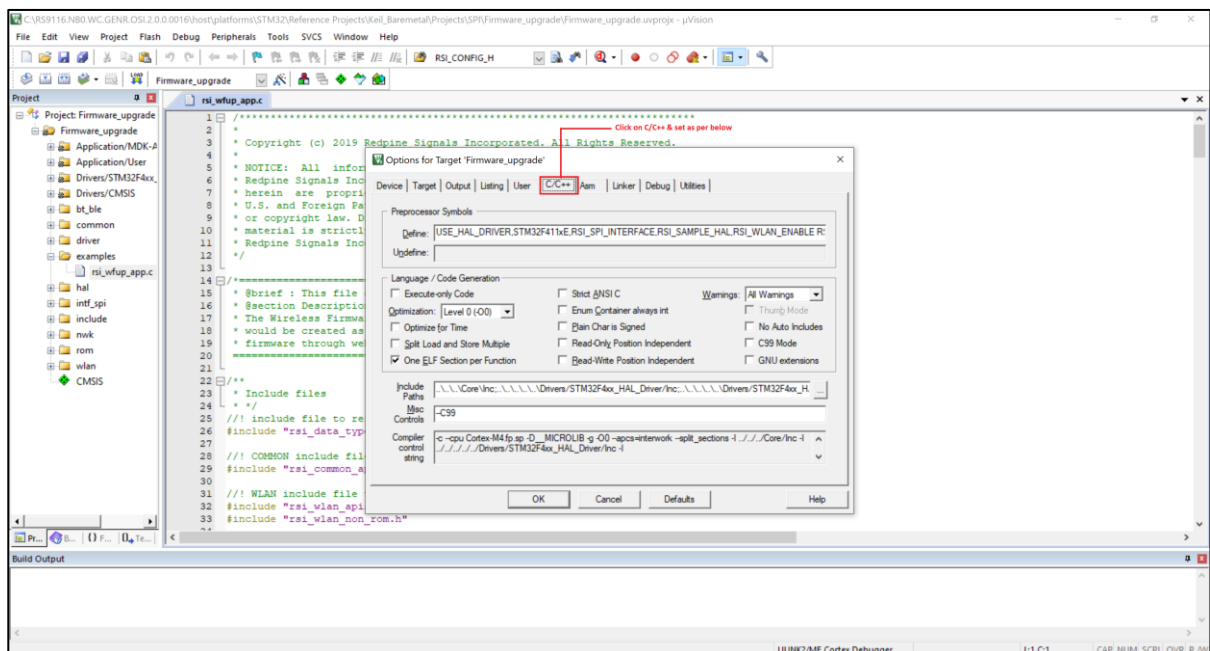


10. Then click on the 'C/C++' icon. Refer to the Preprocessor symbols, Update the pre-defined Macros provided in the 'Define' section if needed. It's depending on the application requirement.
For Ex: If anyone working with SPI communication then needs to enable 'RSI_SPI_INTERFACE', for WIFI needs to enable 'RSI_WLAN_ENABLE' etc.

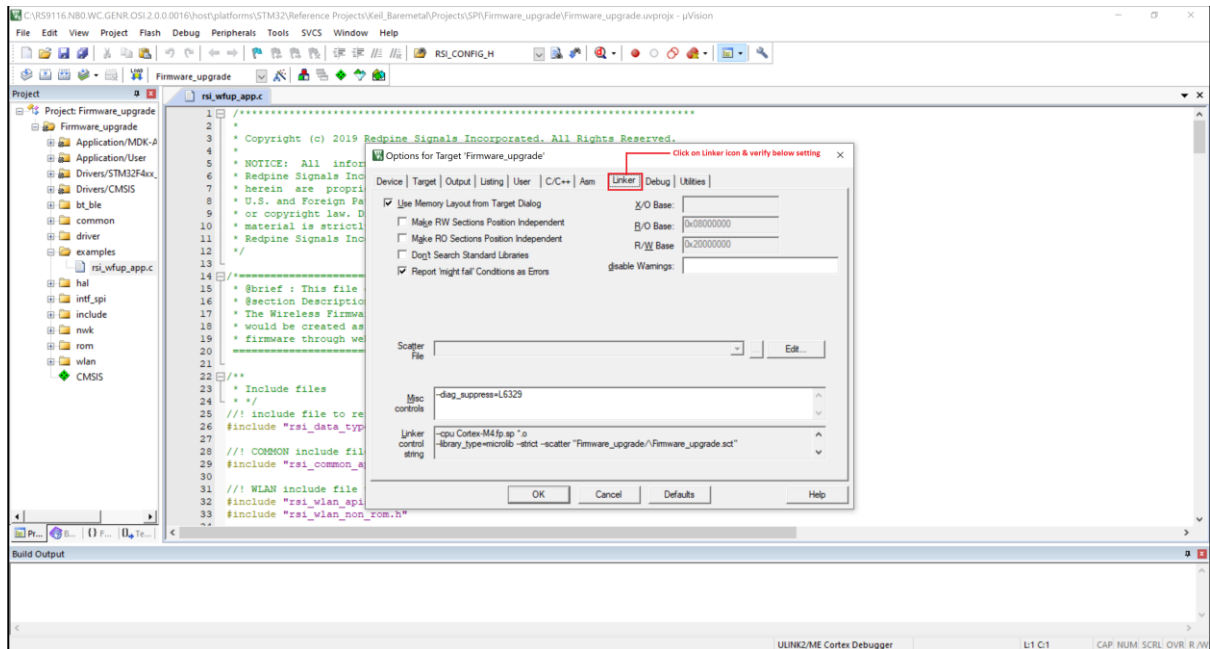
Optimization settings:

If the user doesn't have a valid Keil license but has an Evaluation copy then KEIL might produce a Code size limit error while building STM32 projects/examples.

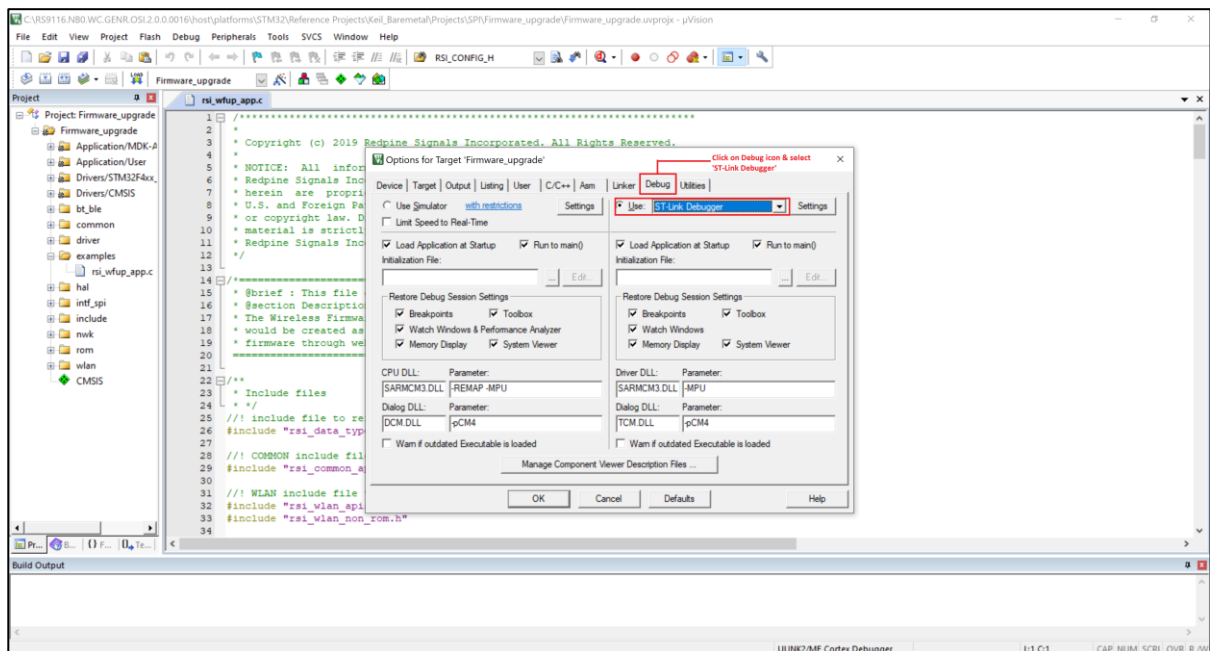
With the Evaluation version, the user can try with optimization level (level 2/3) to avoid the Code size limit error. Still, the optimization level will not ensure the proper building of the STM32 projects. The final recommendation is to get a valid Keil license.



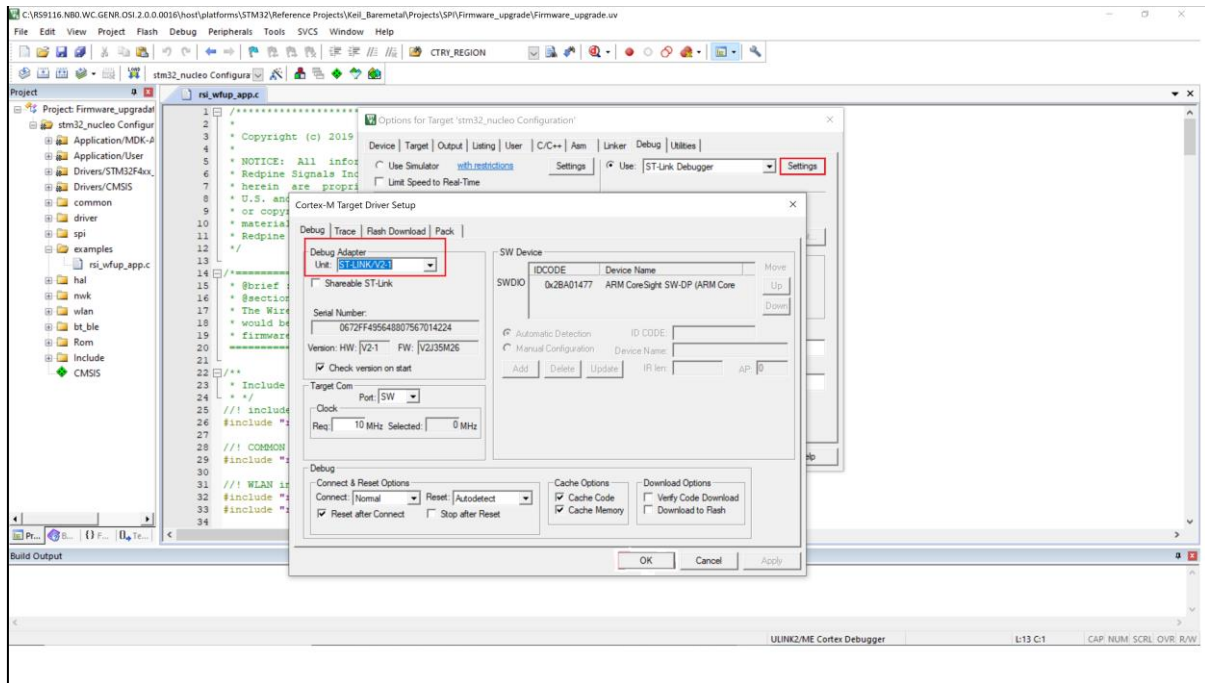
11. Then click on the 'Linker' icon and verify it as per the below image.



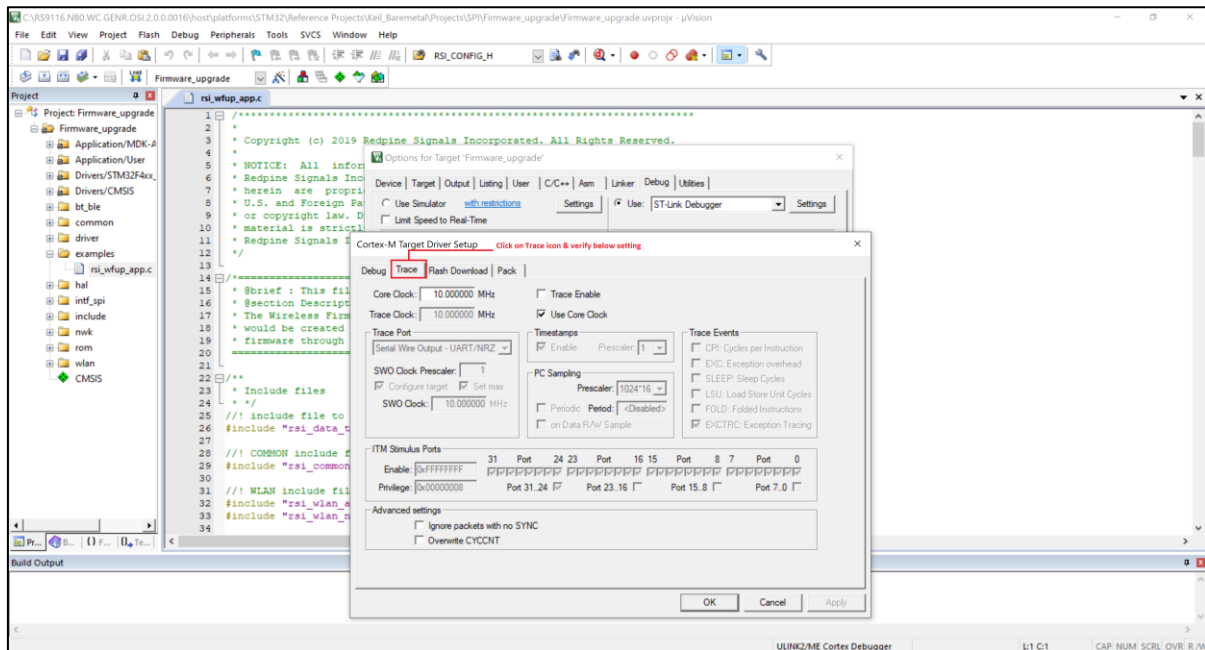
12. Then click on the Debug icon, and select the 'ST-Link Debugger' for debugging.



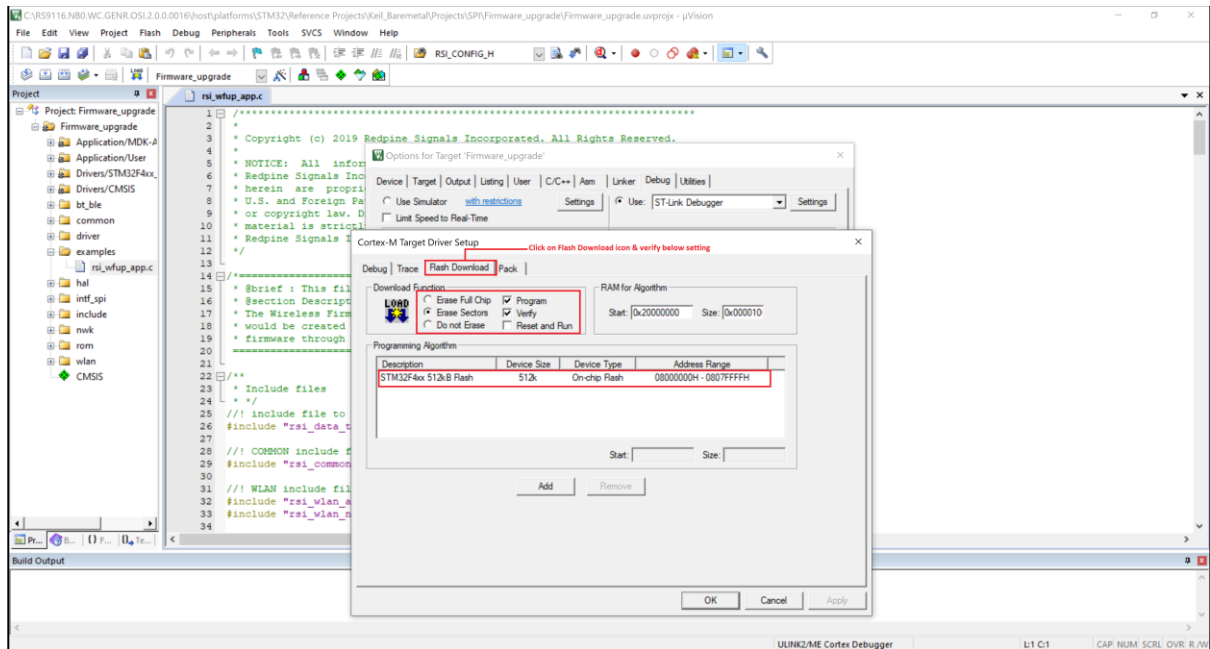
Then click on the Settings. the 'Cortex-M Target Driver setup' window will open. In this window, 'Debug Adapter' sub-section, select 'Unit' as 'ST-LINK/V2-1'.



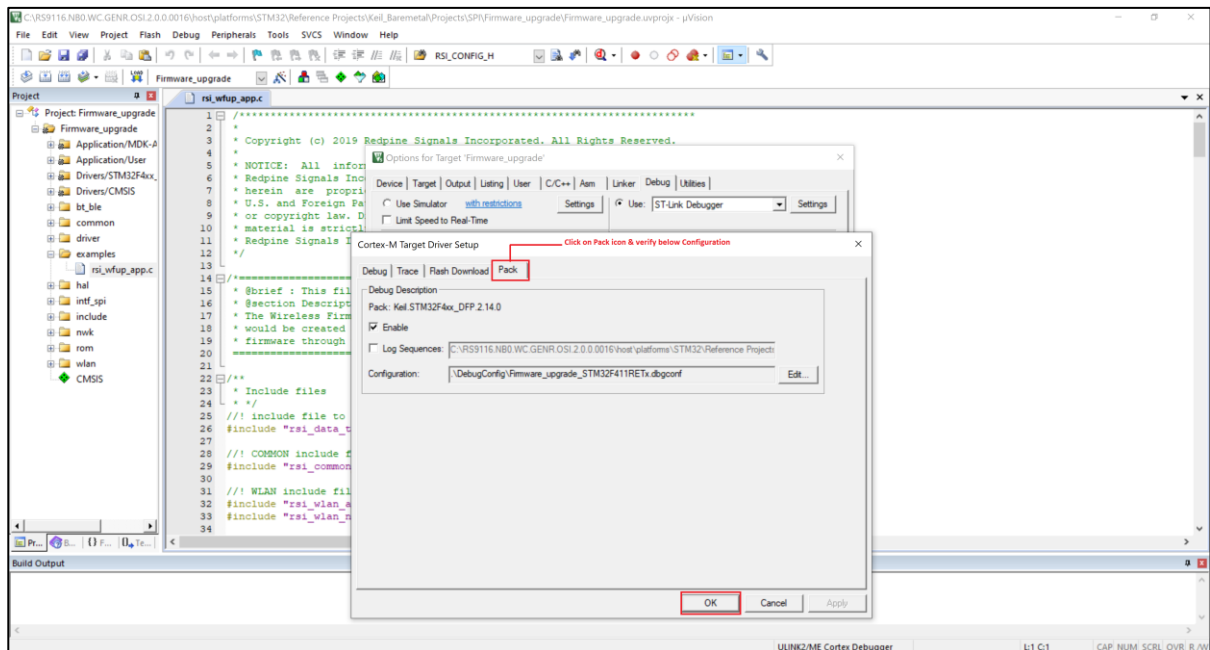
Then click on the 'Trace' icon and verify the below display setting in the image.



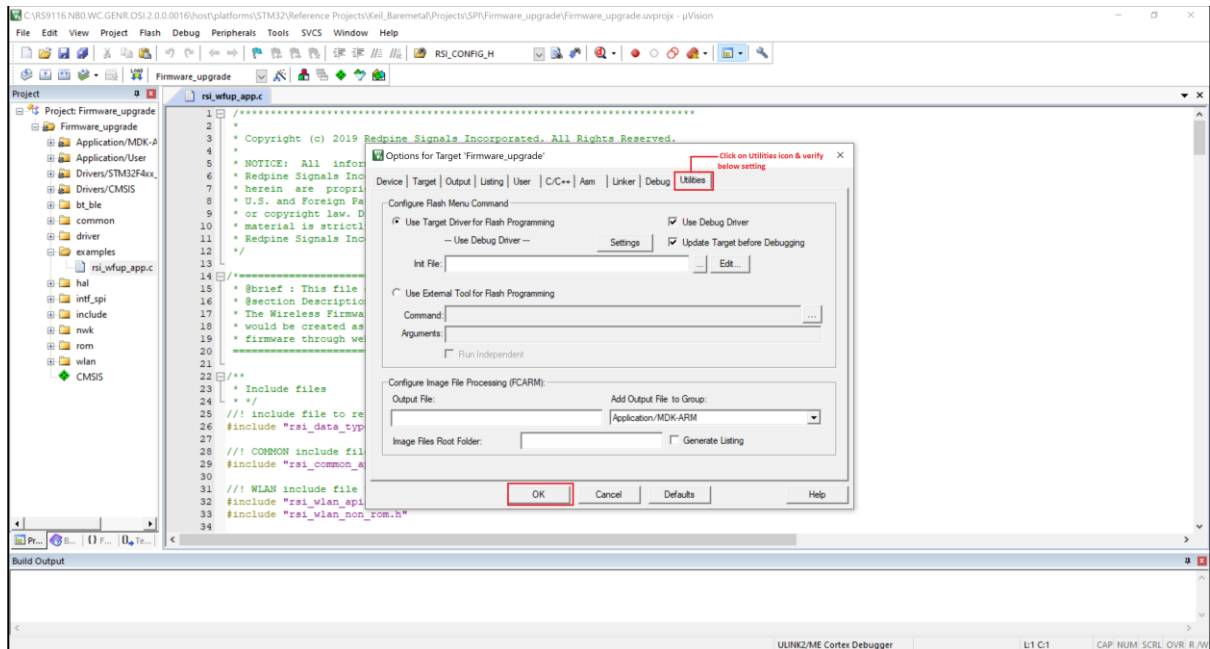
Then click on the 'Flash Download' icon and verify the setting as per the below image.



Then click on the 'Pack' icon and verify the setting as per the below image. and then click on 'OK'.

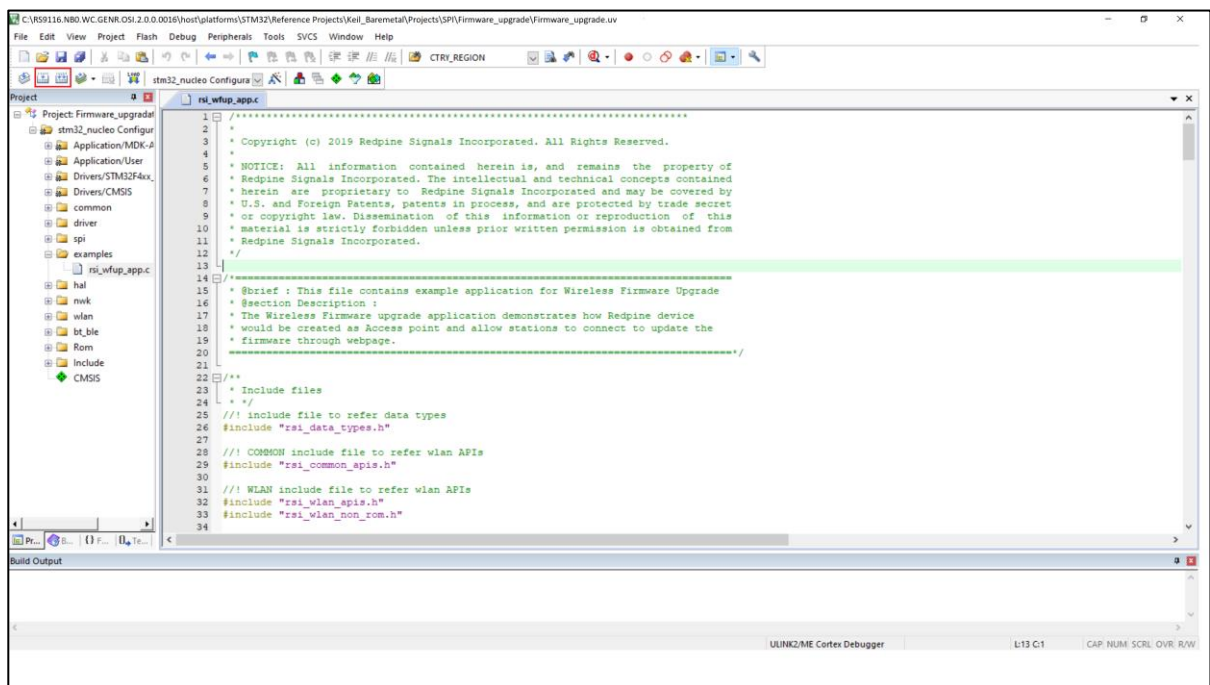


13. Then click on the 'Utilities' icon and verify the setting as per the below image. and then click on 'OK'.



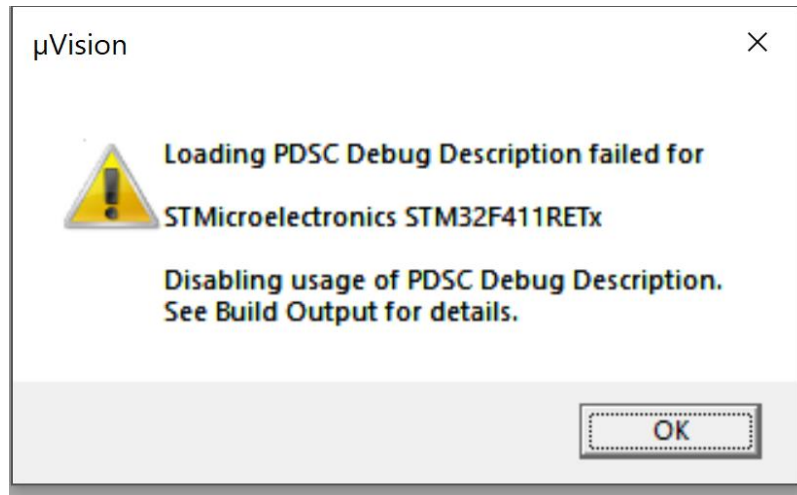
14. In the Project workspace, click on the Build icon to build the project. Refer to the highlighted part in the screenshot.

The project compiles and builds without errors then moves to the next step. In case of build errors, then correct them and rebuild the project.



Note:

While building the project, Incase if the user observes the below template of the error then the issue is there with a longer project path. To avoid this error during the building, please keep the Release package content in the shorter path.



15. User need to press the RESET button on the RS9116 EVK before getting into the project debugging in the Keil

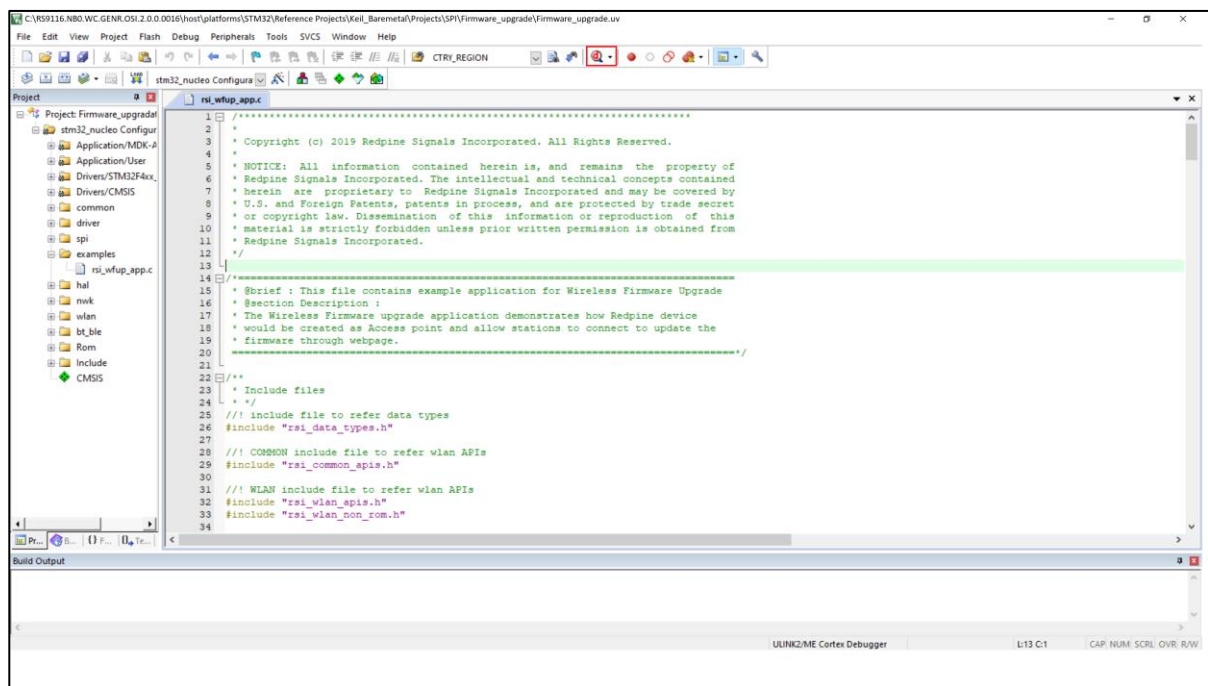
Debug the code by using the 'Debug' icon or press Ctrl+F5 in the Keil IDE. Refer to the highlighted part in the screenshot.

After clicking on the Debug icon,

if the user using Evaluation KEIL IDE then an EVALUATION MODE popup appears. then Click OK.

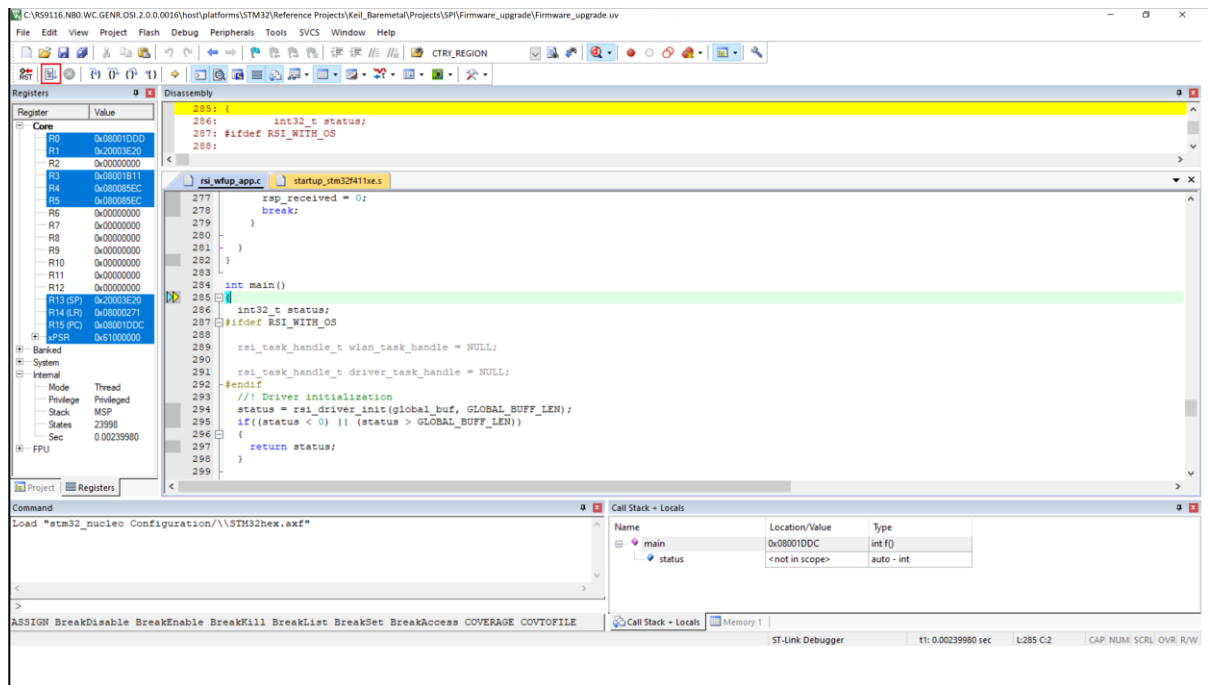
if the user using License version KEIL IDE then the EVALUATION MODE popup disappears.

Note: If user finds "rsi_device_init() failed error on the Log window, that means RESET is not pressed before entering debug state.



16. User needs to press the RESET button on the RS9116 EVK before getting into the Project Free-run in the Keil

Free-Run the application using the 'Run' icon as shown in the highlighted part in the below screenshot or press F5.



17. Please follow Steps 3 to 16 for executing any new project on the STM32 target. Need to press the RESET button on the RS9116 EVK before debugging/running any project on the STM32.

Note

In case of debugging errors, check if your setup is connected correctly, and verify configurations.

2.8 Getting Started with STM32CubeIDE

Introduction to STM32CubeIDE:

STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. It is supported on multiple OS such as Windows, Linux and iOS for Mac (64-bit versions only). The execution steps provided below are for Windows OS.

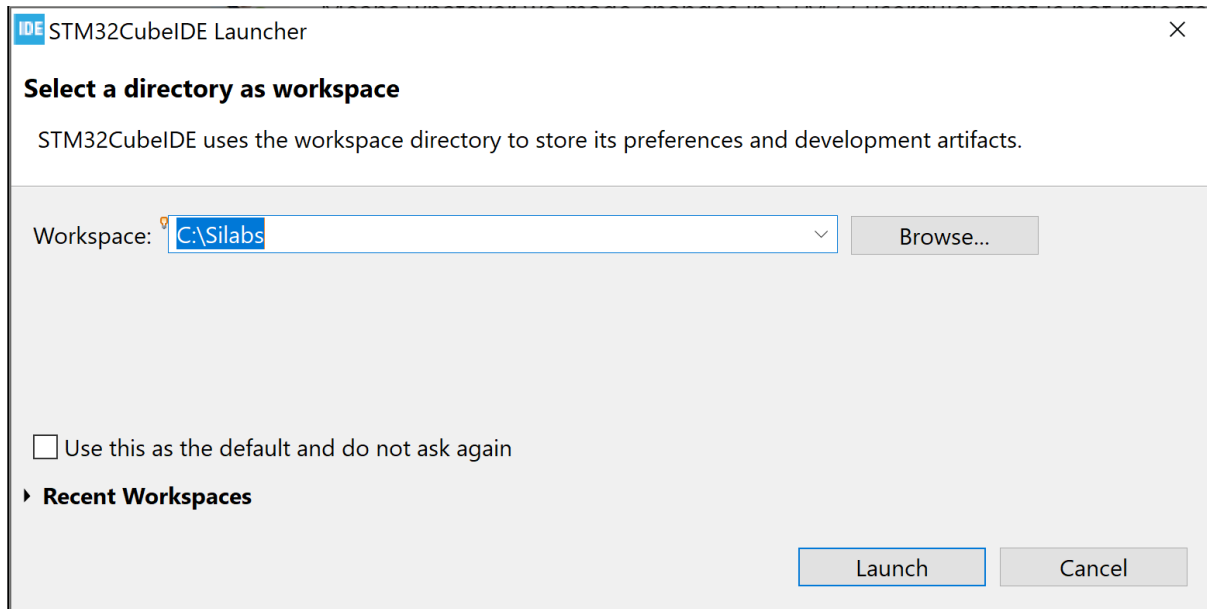
2.8.1 Please follow the below Steps for executing STM32 reference projects on STM32CubeIDE

1. Setup-up the environment of connecting the RS9116 EVK and STM32NUCLEO to the PC
 - a. Ensure EVK and STM32 board are connected via the SPI cable.
 - b. Connect the power port of the STM32 board to the PC via USB cable.
 - c. It is recommended to connect the POWER port of RS9116 EVK to the PC (via USB cable).
2. Download and Install STM32Cube IDE from the link: <https://www.st.com/en/development-tools/stm32cubeide.html>

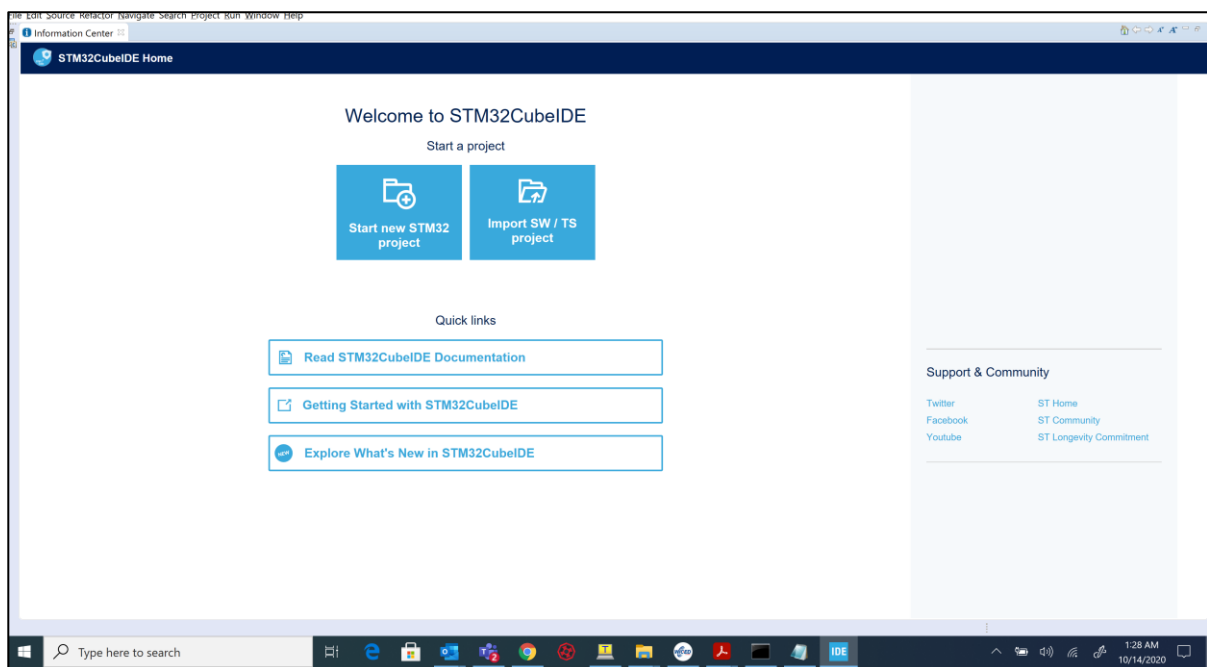
Note

Make sure to download the IDE for the latest version of Windows OS. (Ex:-STM32CubeIDE-Win for windows in the above-mentioned Link).

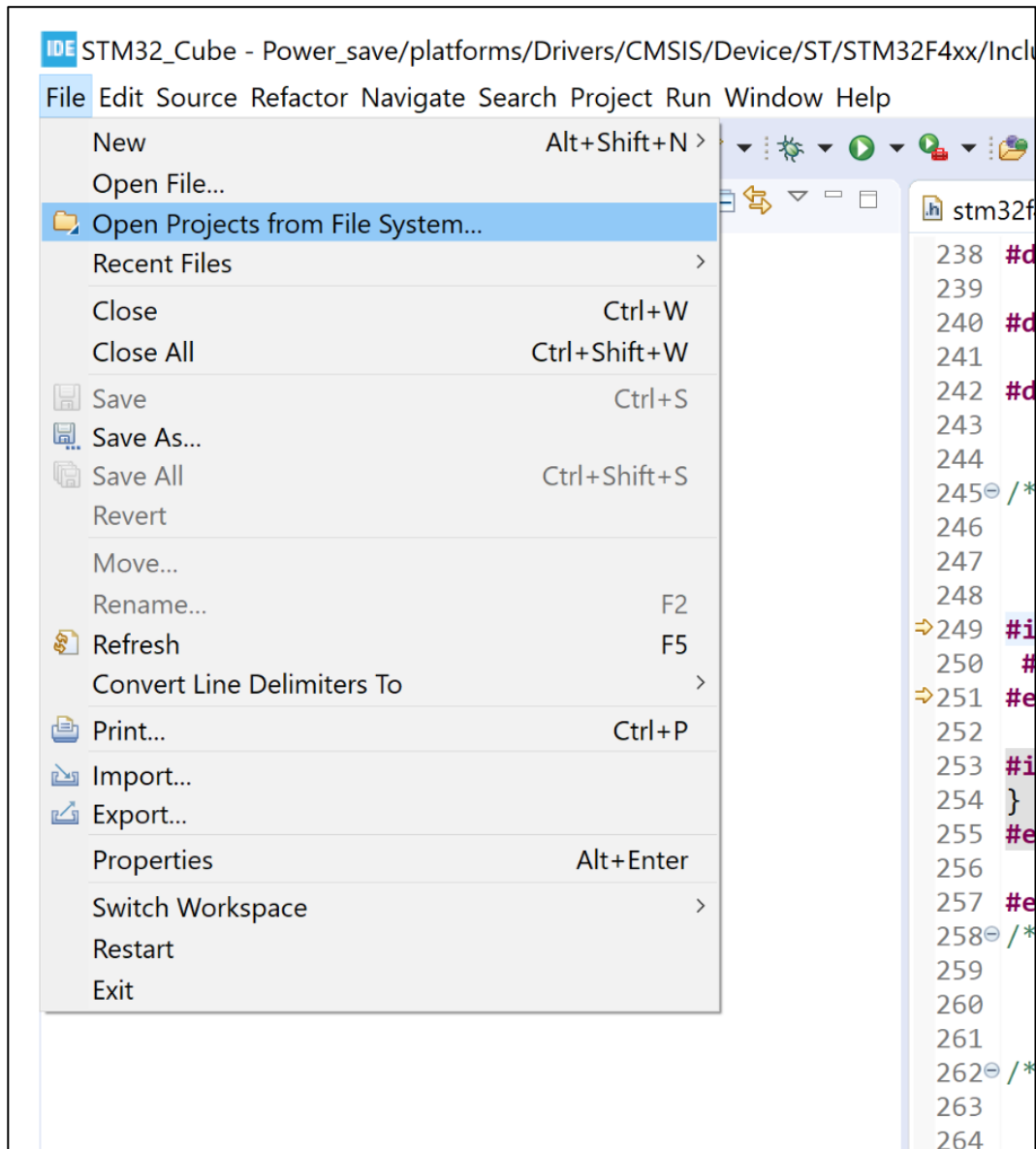
3. Open the STM32CubeIDE tool and give any user choice path for the Workspace and click on Launch, as below.



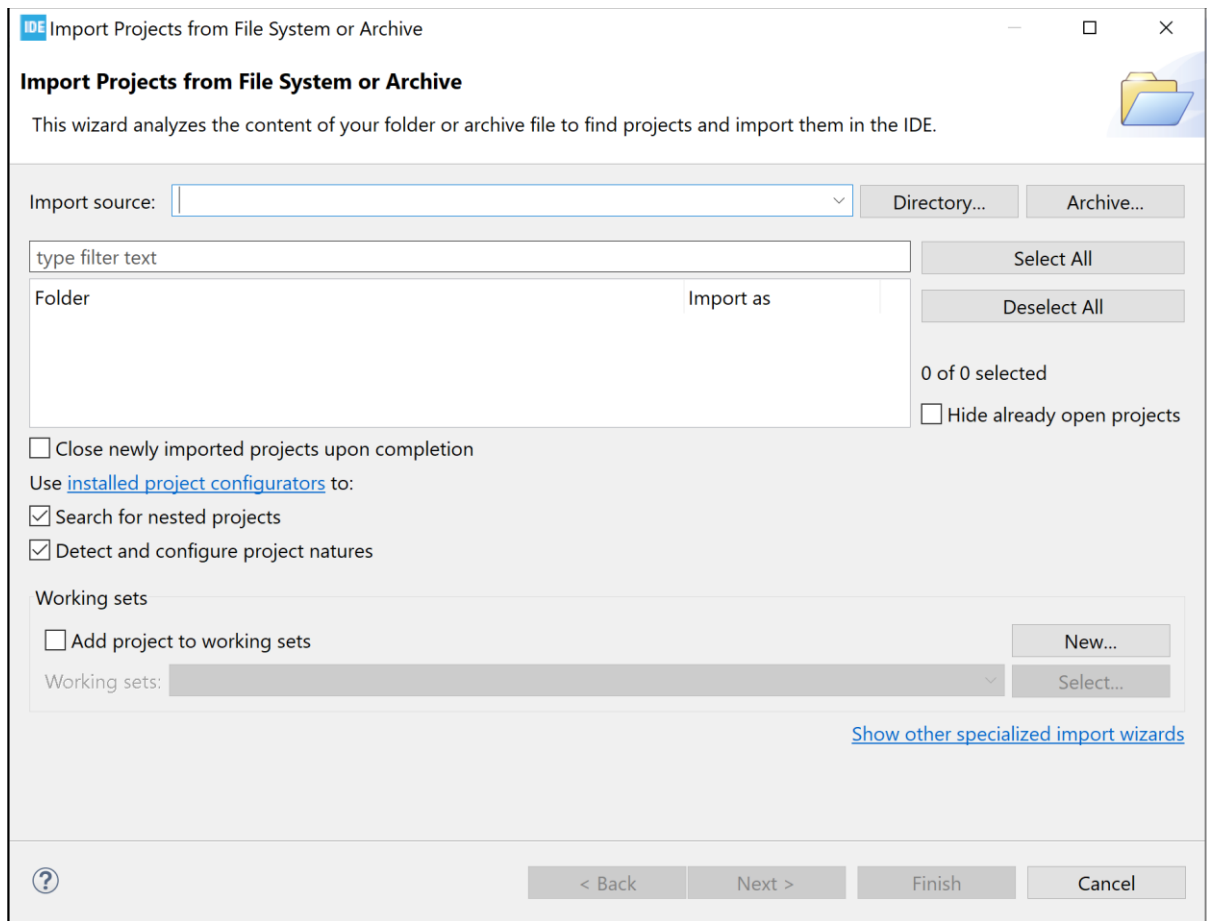
4. After launching, the user can see the below information center window "Welcome to STM32CubeIDE". Close this information center tab at the top left of the window.



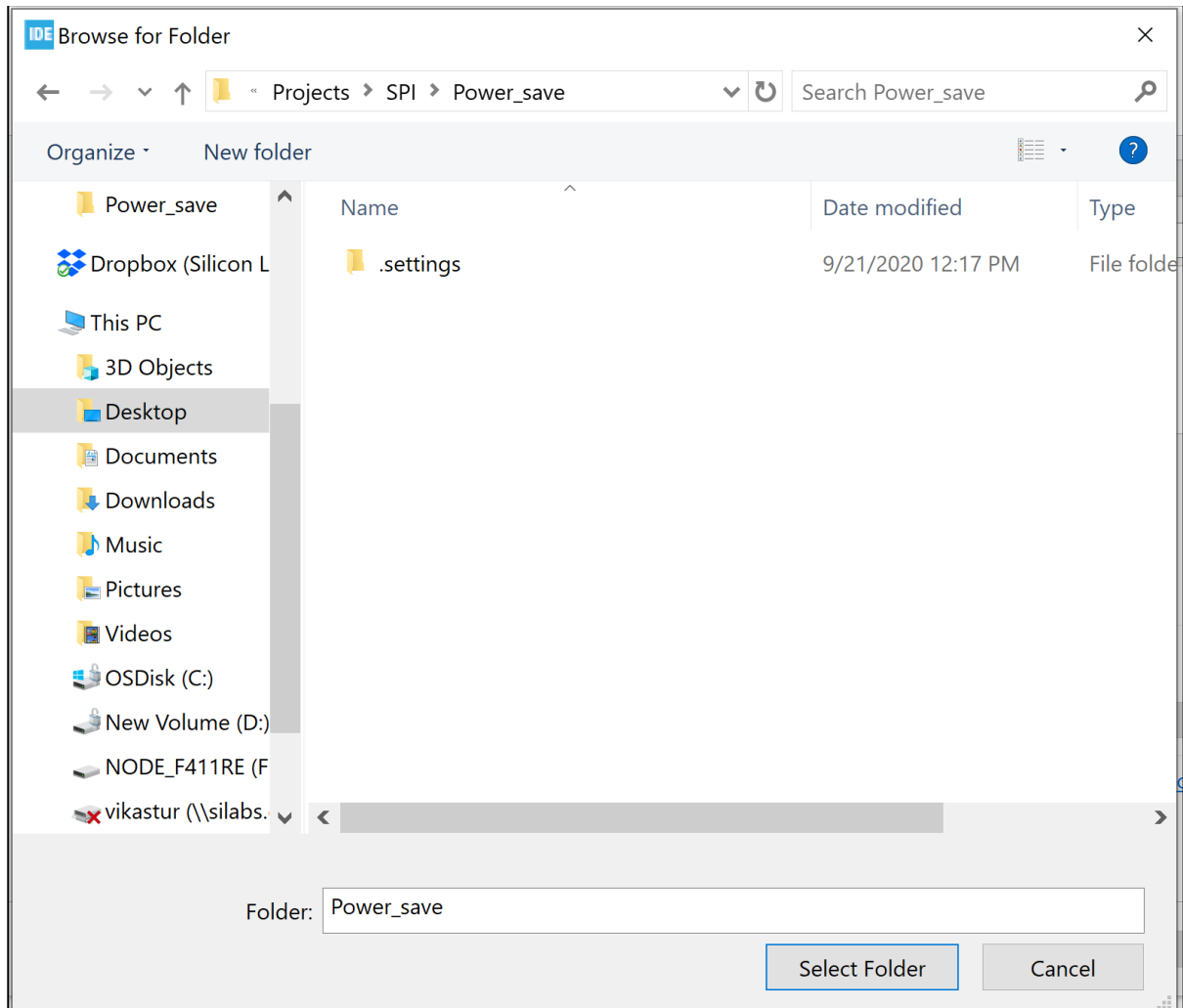
5. Navigate to File→ Open projects from the File system (Import project from File system popup appears).



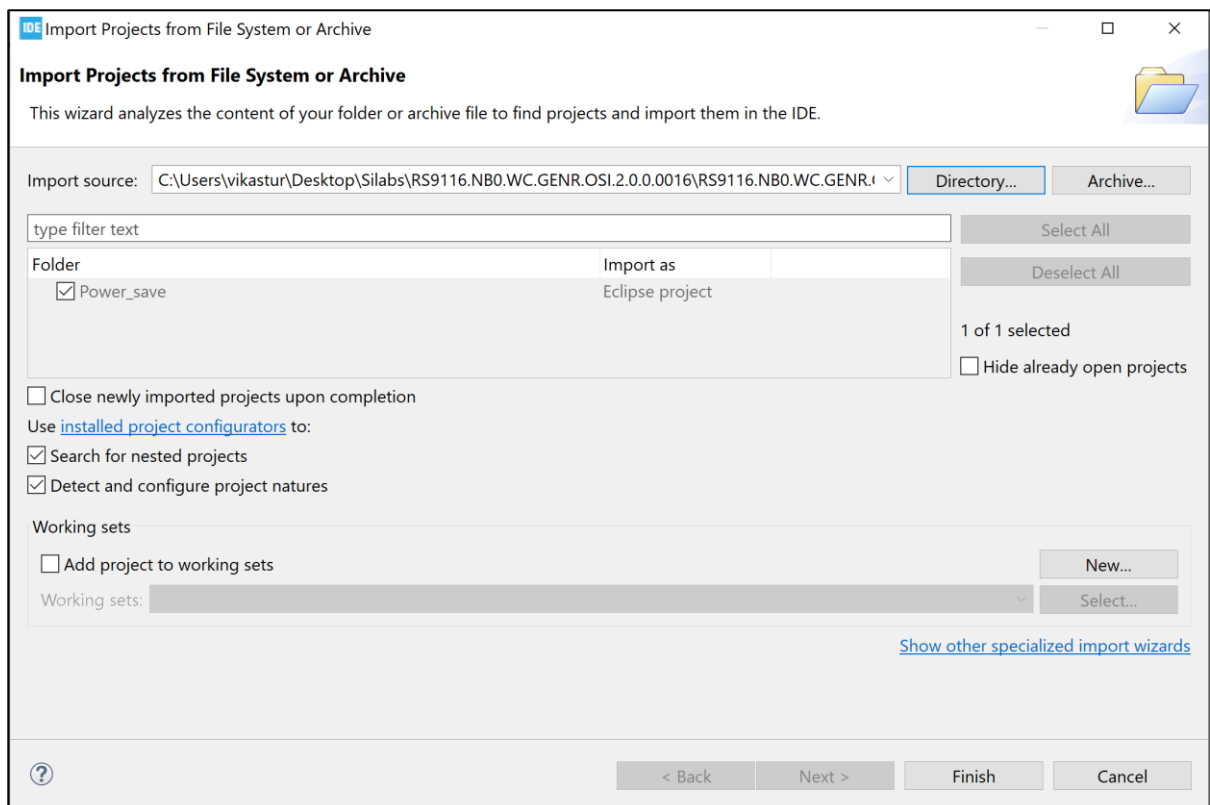
6. Click on the 'Directory' and select the desired STM32Cube project from the below link
RS9116.NB0.WC.GENR.OSI.x.x.x.xx\host\platforms\STM32\Reference_Projects\Cube_Baremetal\Projects\SPI



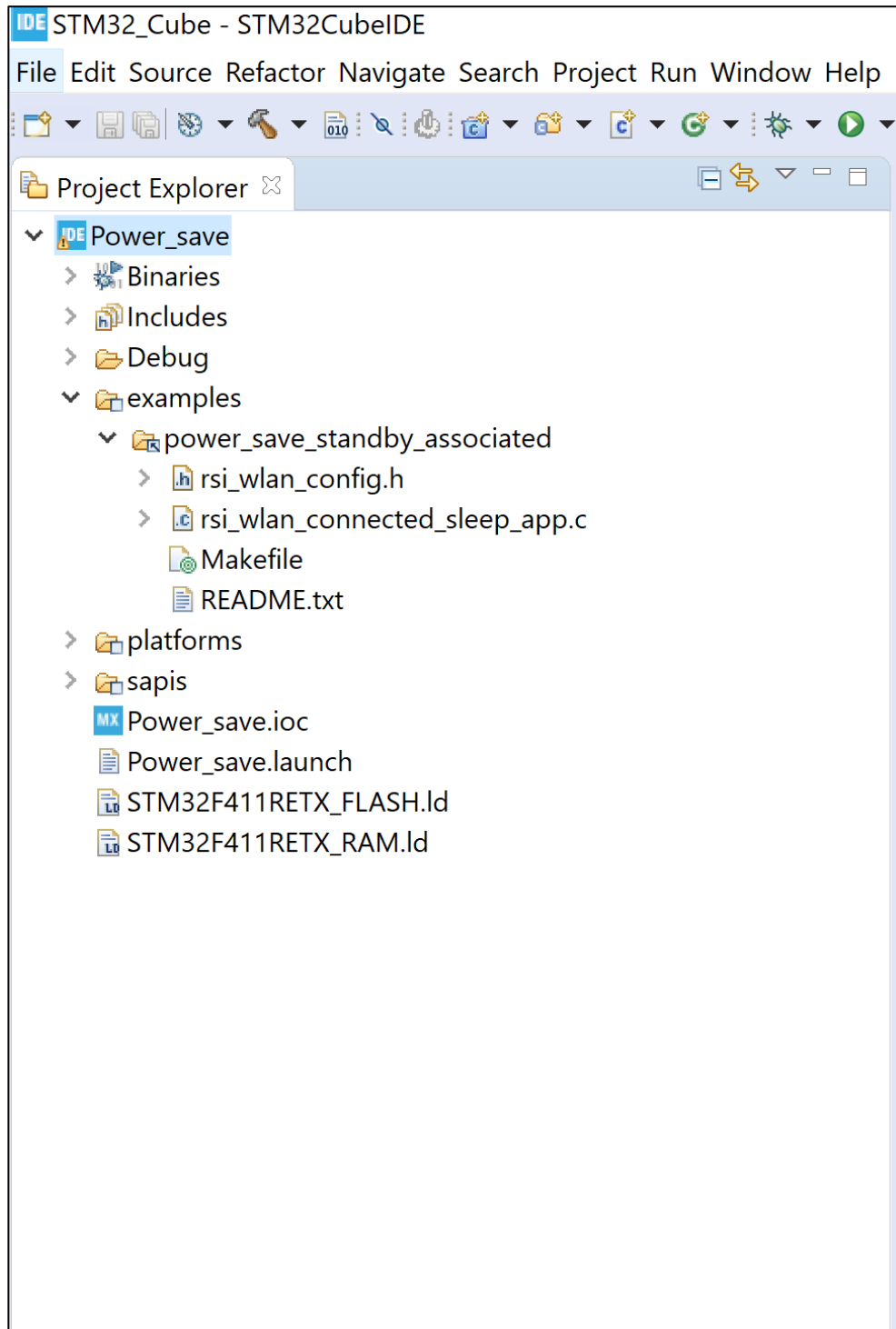
Select Any Reference project as shown below (For ex: 'Power_save' project is selected).



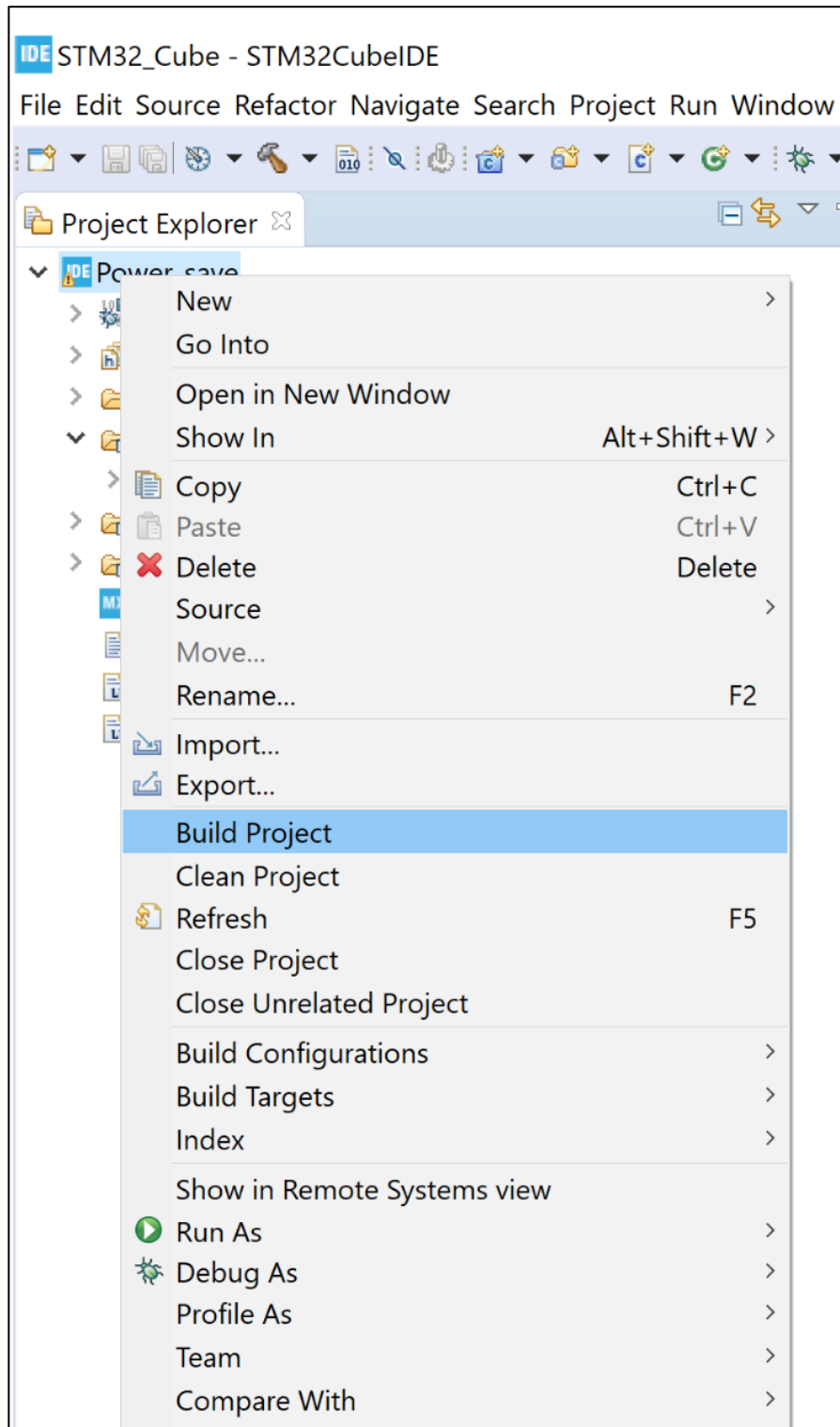
Click 'Finish' then the STM32Cube IDE sub-screen appears.



Then the "Power_save" project appears in the project explorer as below.



7. Right-click on the project (Power_save) and Build the project.



After a successful building, please move onto the next step of project debugging.

8. Before entering the debug state for every project, User need to press the RESET button on the RS9116 EVK for successful execution.

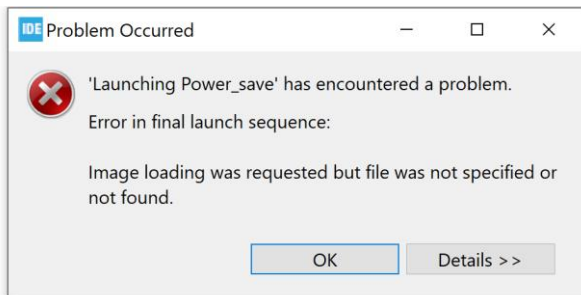
Note: If user finds "rsi_device_init() failed error on the Log window, that means RESET is not pressed before entering debug state.

9. Debugging the selected project for the first time, User need to select the debug file as below
 - a. Right-click on Project->Debug As->Debug configurations.
 - b. Double click on "STM32 Cortex-M C/C++ Application".
 - c. You will see a "Project_name.elf" file. (Ex: power_save.elf). Please refer to the following captures.

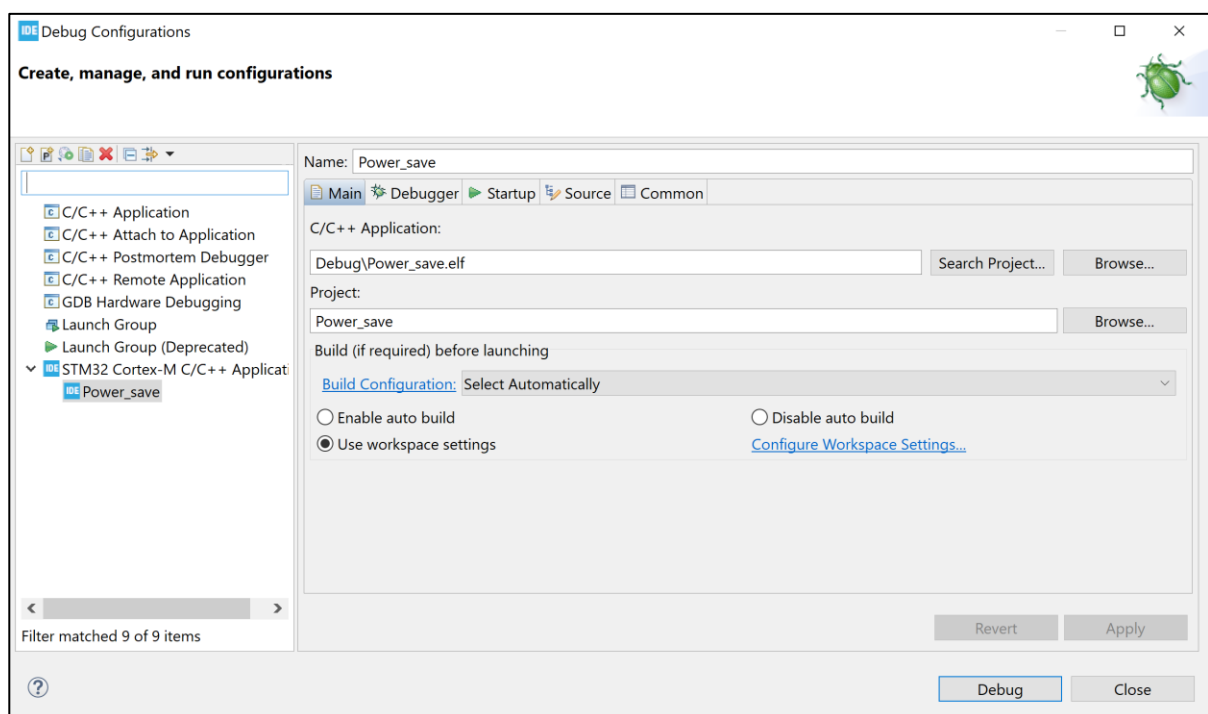
Note

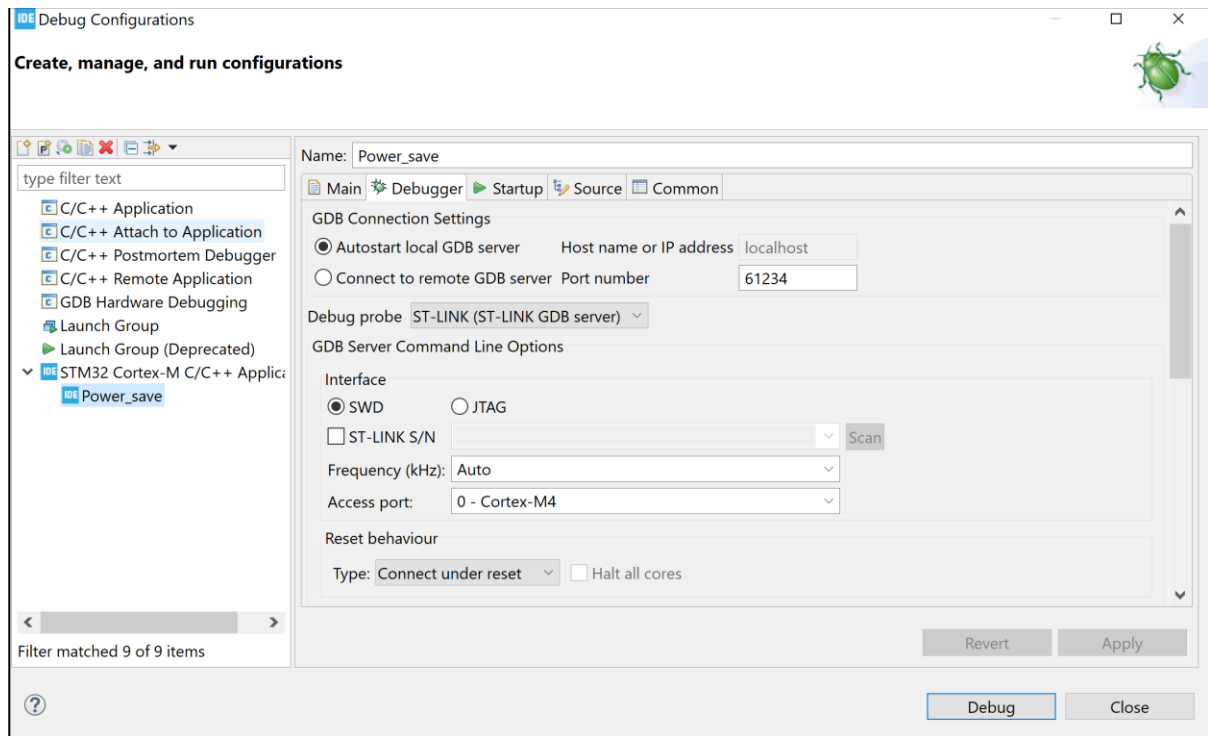
It may or may not show the '.elf' file, if the '.elf' file is not generated click on **project_name debug** which generated after above Step 9-b

Note: User can find the below error while launching the debugger of any project first time, so please follow the Step-9 for avoiding the below error.



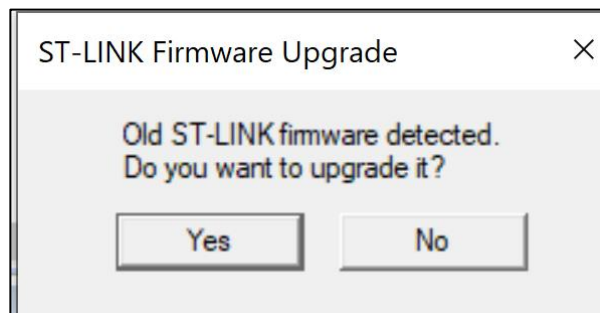
- Click on the '.elf' file or Project_name (if '.elf' not displayed) and click debug. Please make sure all the configurations are the same as Below.



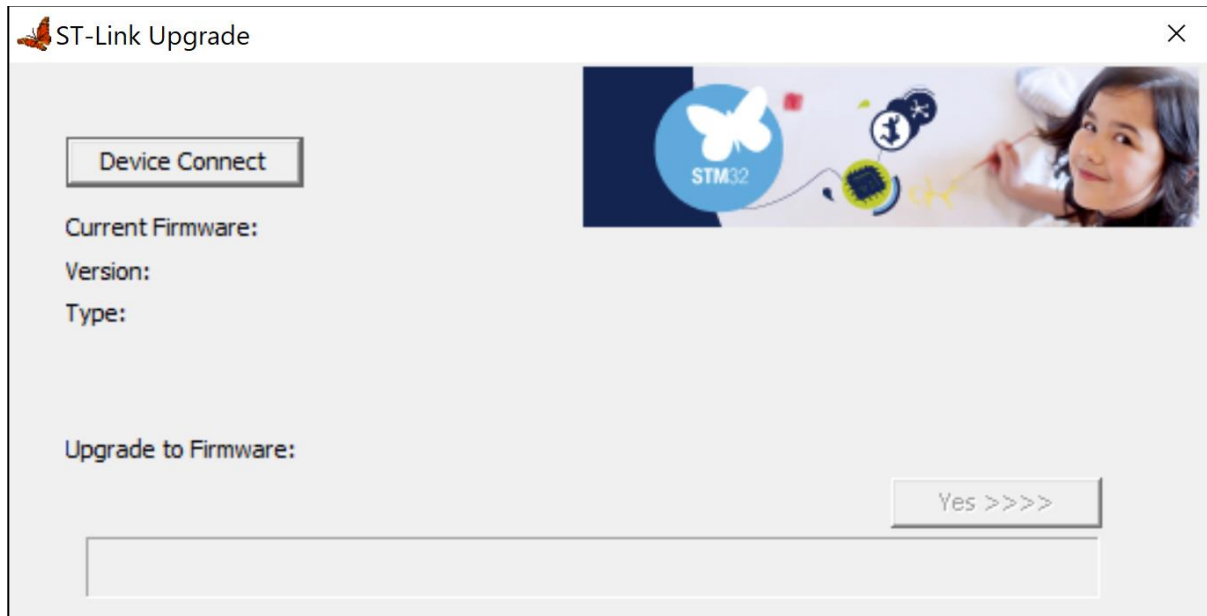


11. For the **First time debugging** with New STM32 or Old STM32 boards which are never used before with SPI, it gives a debug error and also displays a popup to upgrade drivers with reference to STM32Cube IDE. Please Upgrade them and restart Debugging.
Please refer to the captures for better understanding.

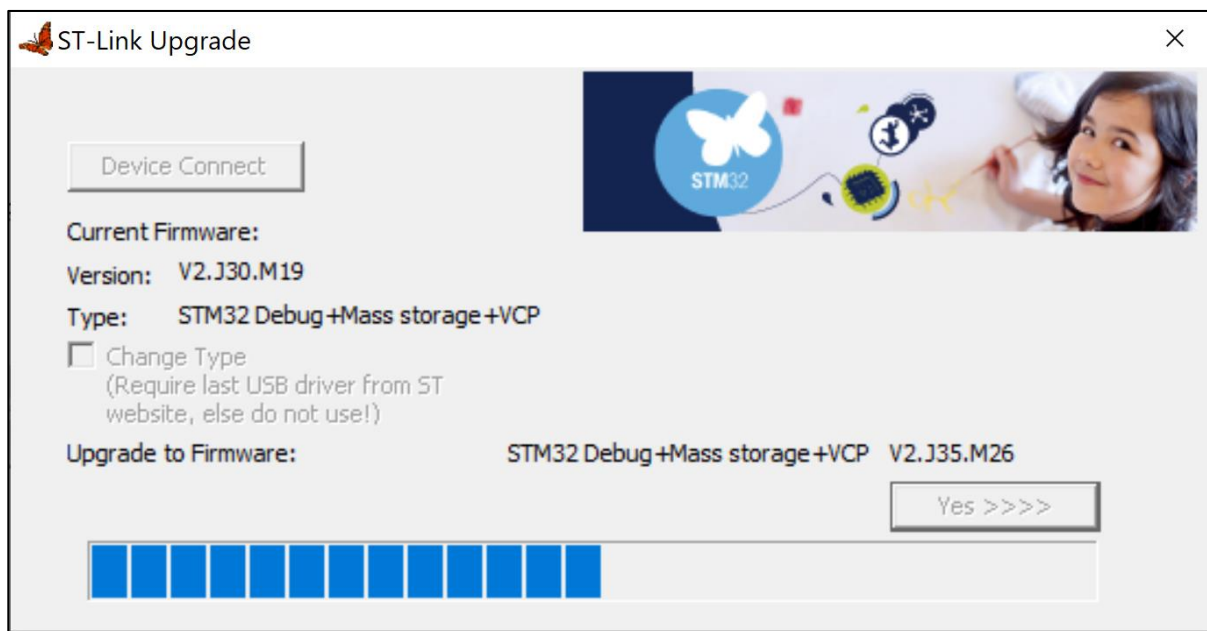
When asks for 'ST-Link firmware detected', Click on 'OK'.



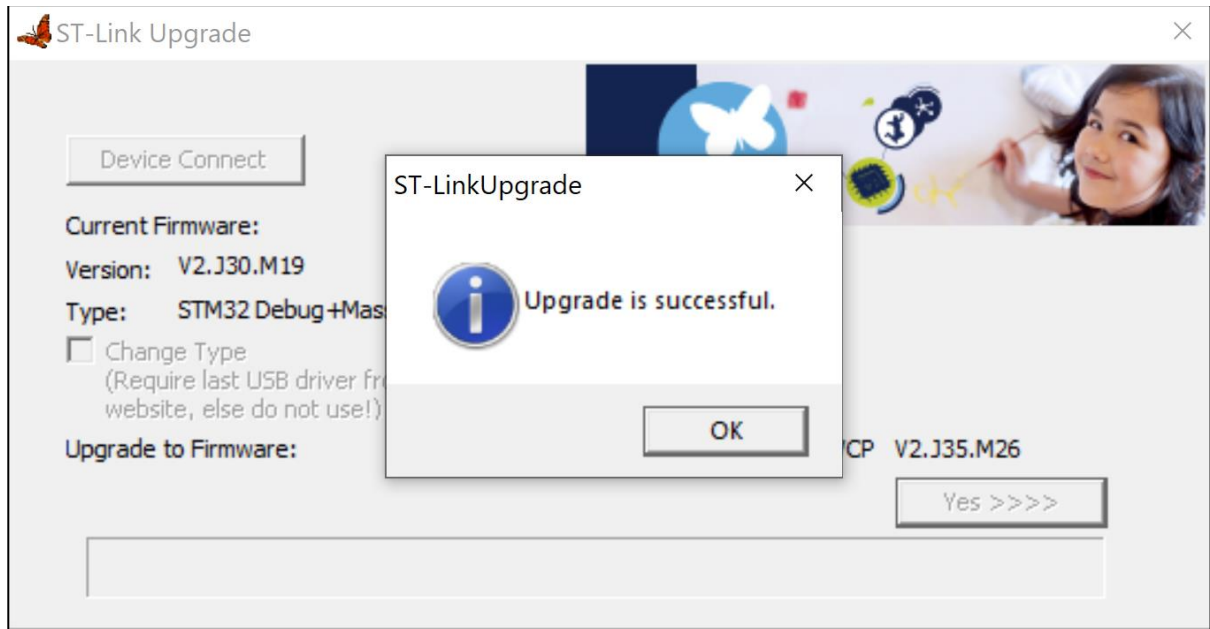
An 'ST-Link Upgrade' pop-up appears, Click on Device connect.



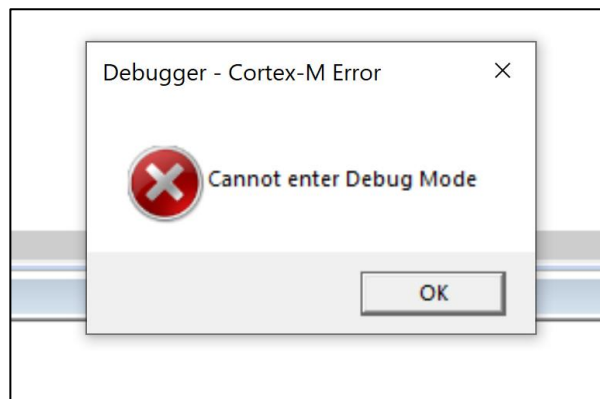
After Clicking on Device Connect, Device gets noticed and Version and type appear.
Click on Yes, the driver starts upgrading.



Once ST-Link driver done Upgrading, a pop up appears as "Upgrade is Successful", Click on 'OK'.

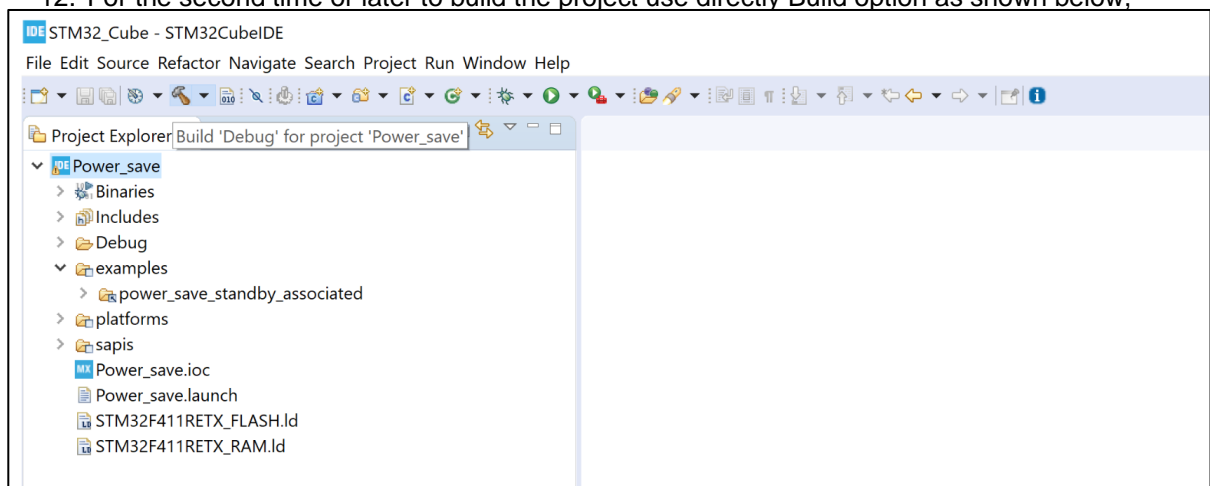


After Successful Upgradation, one pop-up window appears which displays "Cannot enter Debug Mode" (Debug error).

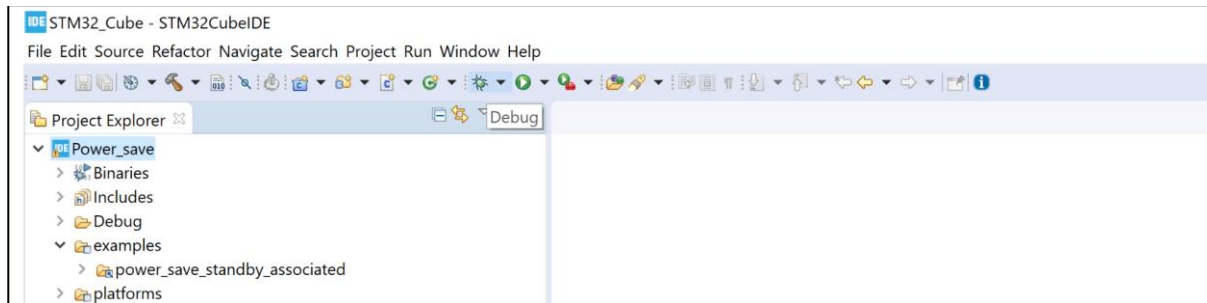


Now click on 'OK' and Re-insert STM and EVK, Now restart the Debug session again.

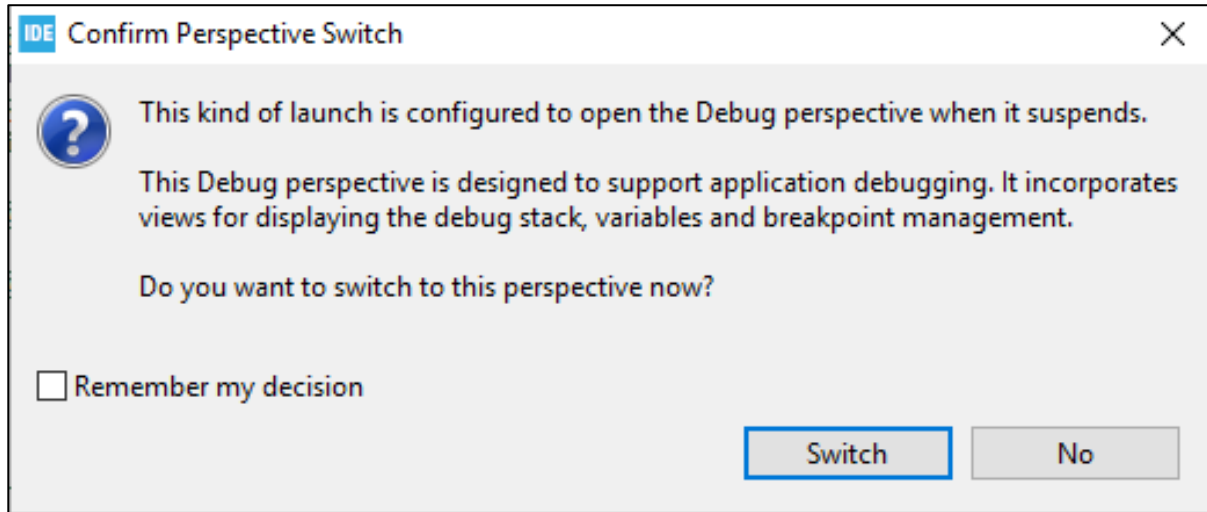
12. For the second time or later to build the project use directly Build option as shown below,



13. Also Debug the code as shown below, (No need to select the Debug file again as mentioned in the Step9)



14. While the project is debugging, a 'perspective' popup window will appear. Click on 'Switch'.



15. After completion of Debug procedures, click on Run(F8).

Note

- In case of debugging errors, check if your Setup is Connected correctly, and verify configurations.
- This document is valid only on WINDOWS OS and for STM32F411RE.
- In case of failure in running the application, please try the following solutions.
 - a) Reset the module properly each time after or before debugging, but compulsorily before running.
 - b) After multiple iterations of debugging, building or editing the project, cleaning the project is recommended.

3 STM32 Reference Projects

This section provides a list of:

1. STM32 reference projects and examples provided in RS9116W release package,
2. Steps for executing Examples using 'Master Application'

CAUTION

Long Keil/STMCube project directory paths can result in build failures with “**No such file or directory**” error message. The maximum length of a file path on Microsoft Windows is 260 characters.

Recommended to keep the RS9116W release package in the shorter directory paths of the PC drives.

CAUTION

By default, SPI interface with DMA is enabled for all reference projects under SPI projects. SPI_DMA only works on NUCLEO F411RE boards.

For older STM32 boards need to disable 'DMA_ENABLED' macro in the project. Please add #define DMA_ENABLED 0 in main project source file.

3.1 STM32 Reference Projects and Example Details

RS9116 package contains STM32 based reference projects and examples that can be executed on Keil IDE and STM32Cube IDE.

Nature of projects are divided into two parts.

1. Baremetal projects are non-OS based
2. FreeRTOS projects are FreeRTOS based

3.1.1 1. Reference projects details

RS9116W release package provided STM32 reference projects in the below path.

"RS9116.SW.1610.x.x.x.xx\host\platforms\STM32\Reference_Projects"

The below figure describes the structure of the reference project in the release package.

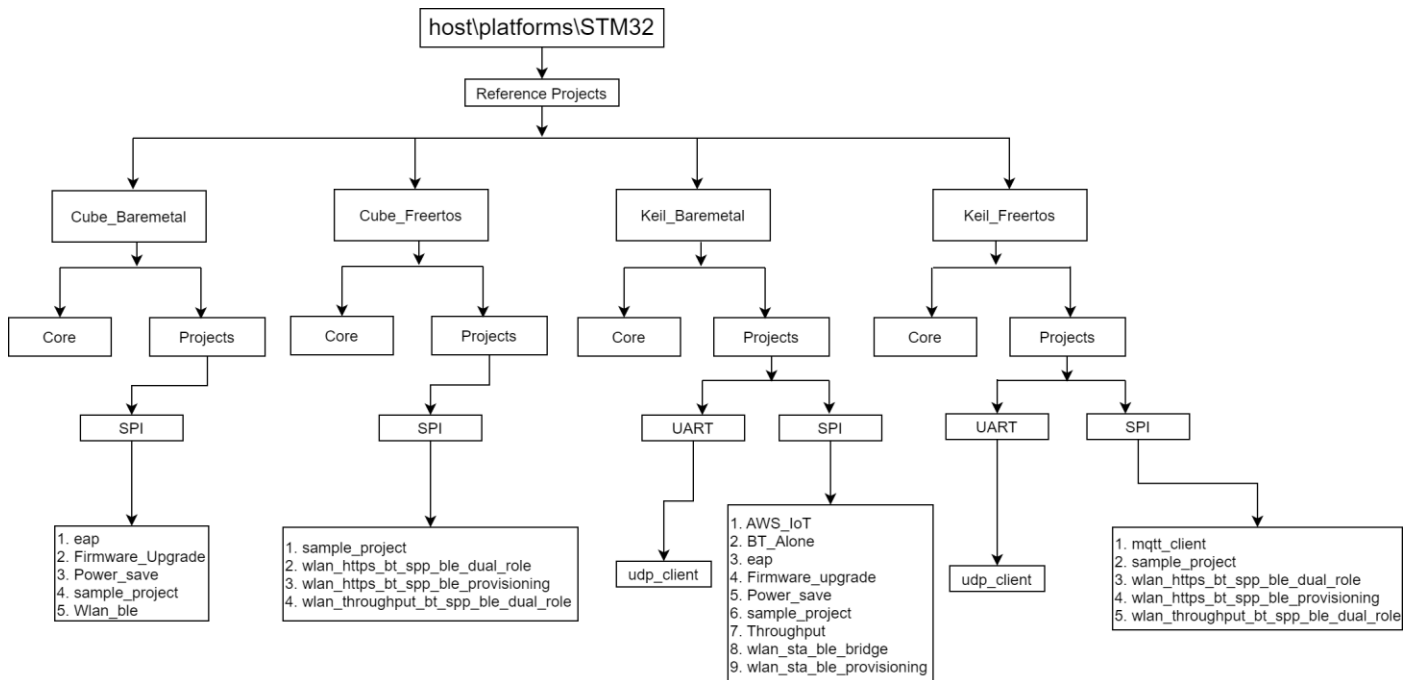


Figure 5: Examples List

3.1.2 2.Reference Examples Details

In addition to reference project above, the RS9116W release package is also provided many WiFi, BT, BLE examples in the below path.

[RS9116.NB0.WC.GENR.OSI.x.x.x.xx\host\sapis\examples\](#)

Please refer to the [Getting Started with Keil IDE](#) and [Getting Started with STM32CubeIDE](#) for executing the these examples.

3.2 Steps for Executing STM32 Examples using Master Application (Sample Project)

This section describes steps to be followed for executing example (provided in the package) using Mater application.

Examples path: [RS9116.NB0.WC.GENR.OSI.x.x.x.xx\host\sapis\examples\](#)

Master Application path: [RS9116.NB0.WC.GENR.OSI.x.x.x.xx\host\sapis\examples\master_application\](#)

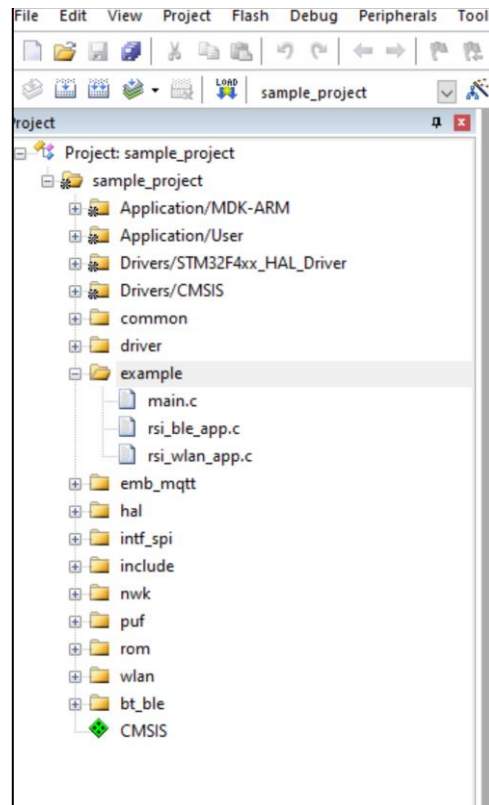
The master application provided in the package is not RTOS compatible by default. If the user wants to use this application with FreeRTOS, the application needs to be modified.

3.2.1 Steps for Keil IDE:








3.2.1.1 1.Keil_Baremetal Projects (non-OS type)

Please follow the below steps for executing sample project.

1. Sample project is provided for compiling and building Wlan, BT, BLE examples in the release package. Open "**sample_project.uvprojx**" located in the below folder
"[RS9116.NB0.WC.GENR.OSI.x.x.x.xx\host\platforms\STM32\Reference Projects\Keil Baremetal\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.xx\host\sapis\examples\master_application".
To test specific examples from the "RS9116.NB0.WC.GENR.OSI.x.x.x.xx\host\sapis\examples" Navigate to the master_application folder (shown in the below fig) and delete the existing code, copy and paste the specific example code into the master_application folder.

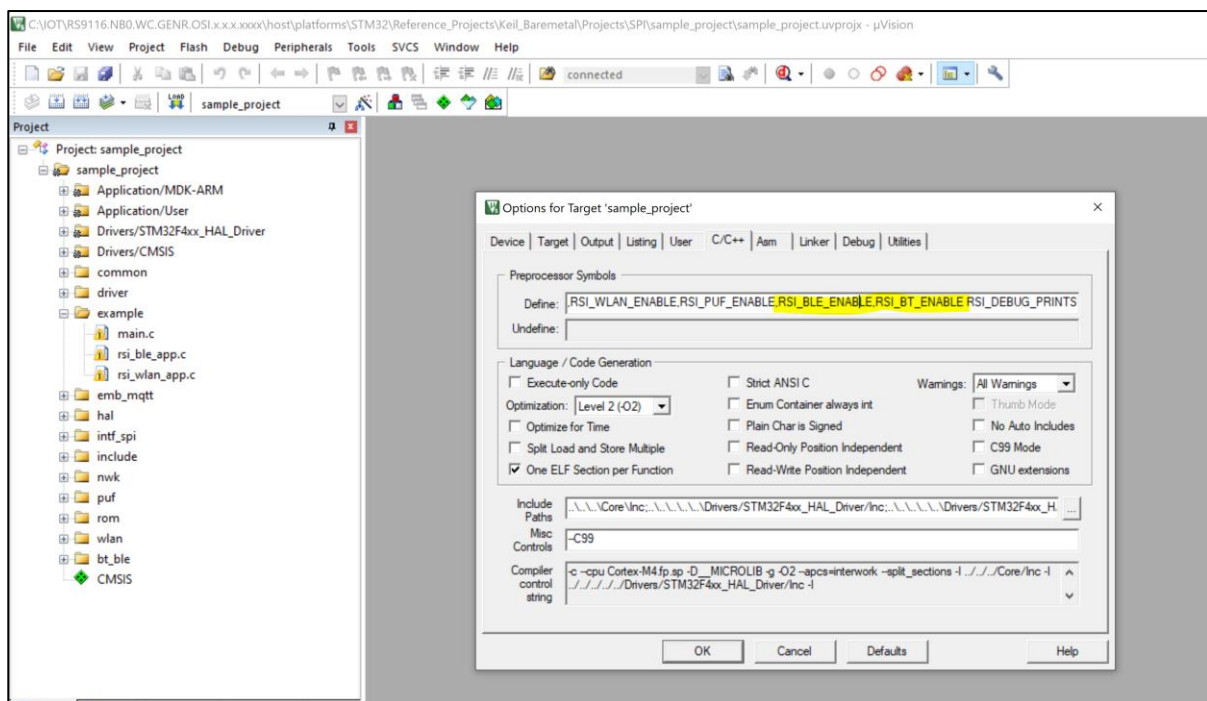
IOT > RS9116.NB0.WC.GENR.OSI.x.x.x.xxx > host > sapis > examples > master_application				
<input type="checkbox"/> Name	Date modified	Type	Size	
 main	9/21/2020 10:03 PM	C File	6 KB	
 Makefile	9/9/2020 5:14 PM	File	1 KB	
 rsi_ble_app	9/9/2020 5:14 PM	C File	31 KB	
 rsi_ble_config	9/9/2020 5:14 PM	H File	7 KB	
 rsi_bt_config	9/9/2020 5:14 PM	H File	4 KB	
 rsi_wlan_app	9/9/2020 5:14 PM	C File	11 KB	
 rsi_wlan_config	9/9/2020 5:14 PM	H File	25 KB	

Ex: Below screenshot shows the example code copied from "RS9116.NB0.WC.GENR.OSI.x.x.x.xx\host\sapis\examples\wlan\access_point" to "RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\sapis\examples\master_application"

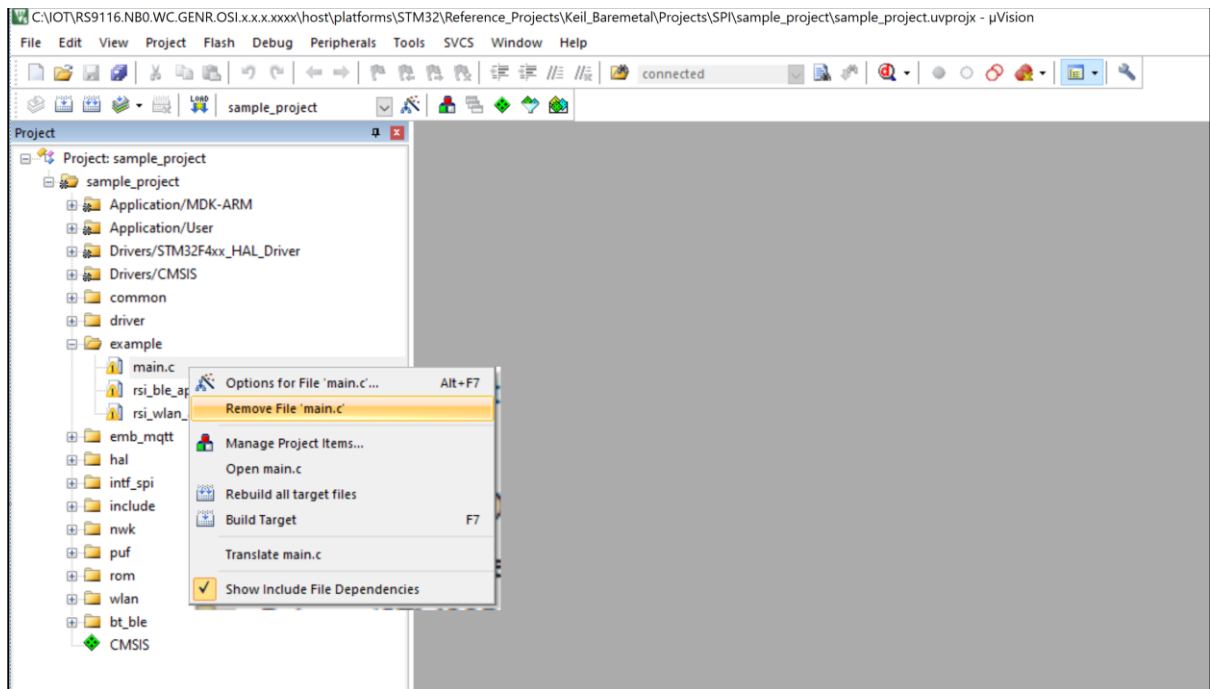
IOT > RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx > host > sapis > examples > master_application				
<input type="checkbox"/> Name	Date modified	Type	Size	
Makefile	9/22/2020 11:54 AM	File	1 KB	
README	9/22/2020 11:54 AM	TXT File	1 KB	
rsi_ap_start	9/22/2020 11:54 AM	C File	10 KB	
rsi_wlan_config	9/22/2020 11:54 AM	H File	24 KB	

Note:

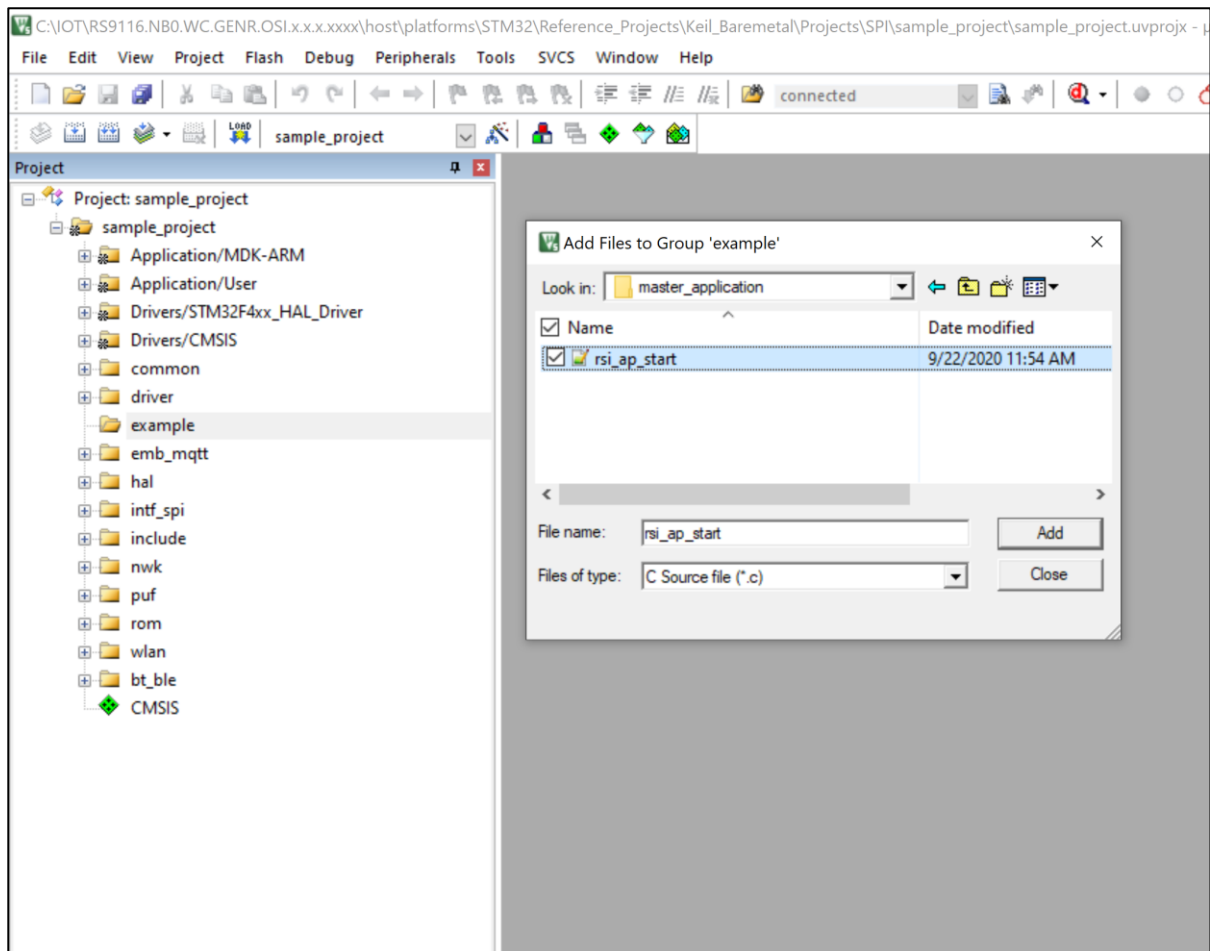
- sample_project compiles successfully only if all the header files rsi_wlan_config.h, rsi_bt_config.h and rsi_ble_config.h are present in "master_application" folder.
For all examples, please ensure rsi_wlan_config.h, rsi_bt_config.h and rsi_ble_config.h files in the master_application for building the sample project.
- Any example folder doesn't contain all three configuration header files, in that case please use the default configuration headers files present in the "master_application" folder (or)
Remove the corresponding protocol macros in Keil project settings (Ex: As the above example code doesn't have rsi_ble_config.h and rsi_bt_config.h, so remove RSI_BLE_ENABLE, RSI_BT_ENABLE)



3. Next, go to Keil project workspace and remove the already existing files from the examples folder



Add new files (only '.c') from the master_application and compile the project



4. Repeat the Steps1 to 4 for compile/building any example with the sample project.

2. Keil_FreeRTOS Projects (OS type)

Sample project is provided (in the below path) for compiling and building Wlan, BT, BLE examples in the release package. Please refer examples which are OS based.

Please follow the below steps for executing sample project.

1. The release package doesn't include FreeRTOS source. So users has to follow steps in the section [FreeRTOS Porting for the STM32](#) for generating the FreeRTOS source.

The generated FreeRTOS source shall be placed under
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx\host\platforms\STM32\Reference Projects\Keil Freertos\"

2. After the above Step1, Please follow the Keil_Baremetal Steps1 to 4 (from the above section) for compiling and building 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"

Note: The sample project will get compiled only if the FreeRTOS code is placed as per the Step1.

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 examples needs more source files support from the nwk folder (located in \host\sapis\nwk\applications) for the successful compilation of the sample project.

User need to ensure the supporting source files present in the Sample project in the Keil project work space.

For Example: **Cloud** example needs supporting source files present in the "host\sapis\nwk\applications\aws_sdk" for successful compilation.

S.No	Examples	Nwk folder files need to add for successful 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_snmp_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

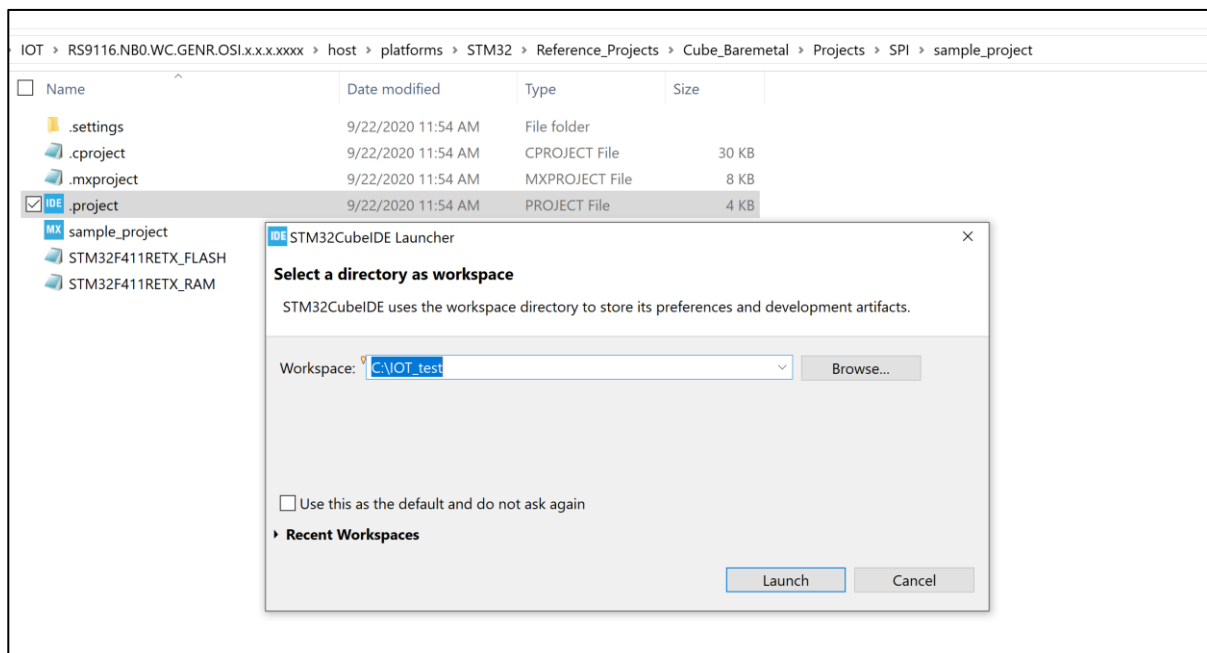
8	http_Client/http_client_post_data	rsi_http_client.c
9	multicast	rsi_multicast.c
10	ftp_client	rsi_ftp.c
11	emb_mqtt	rsi_emb_mqtt.c
12	dhcp_dns_fqdn	rsi_dhcp_dns_fqdn.c
13	otap	rsi_firmware_upgrade.c
14	dhcp_user_class	rsi_dhcp_user_class.c
15	Raw data	rsi_raw_data.c

3.2.2 Steps for STMCube IDE

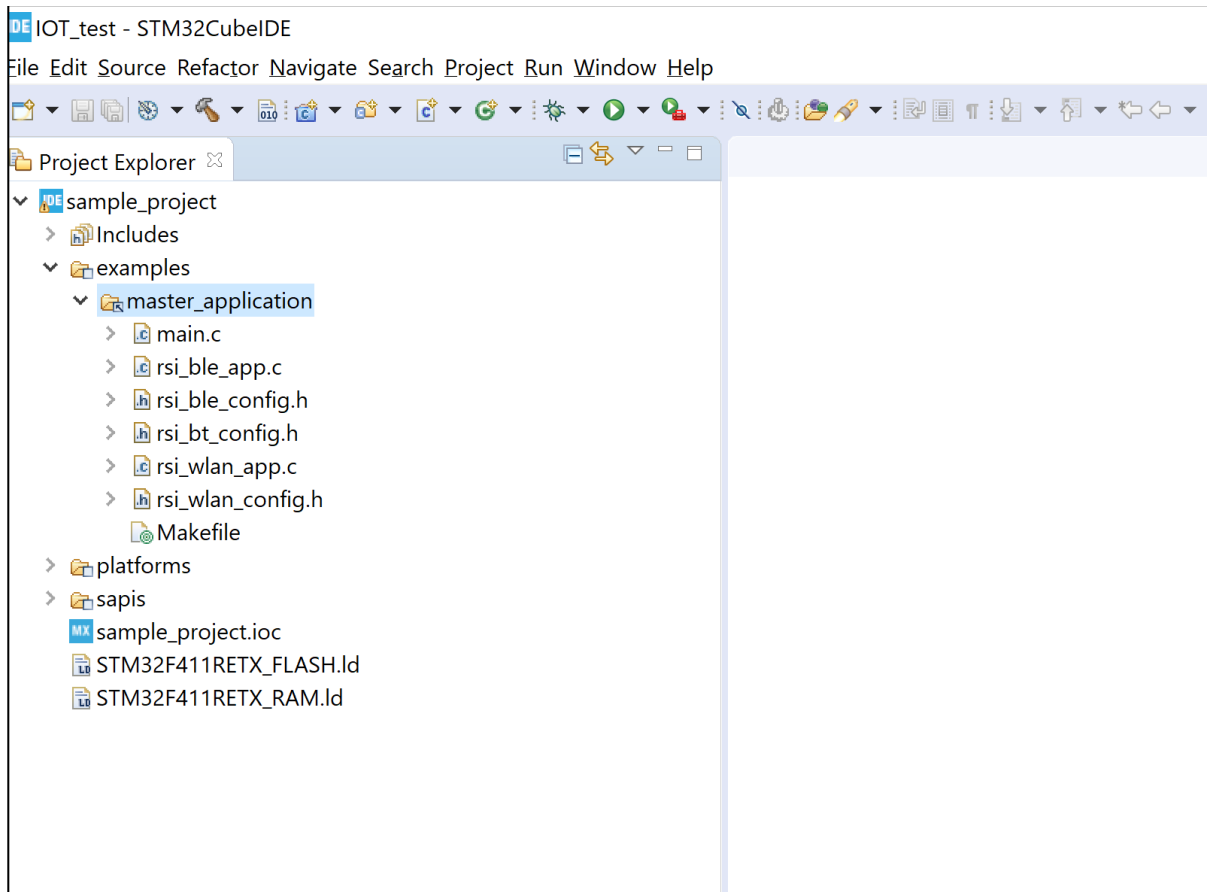
3.2.2.1 1.Cube_Baremetal Projects (non-OS type)

Please follow the below steps for executing sample project.

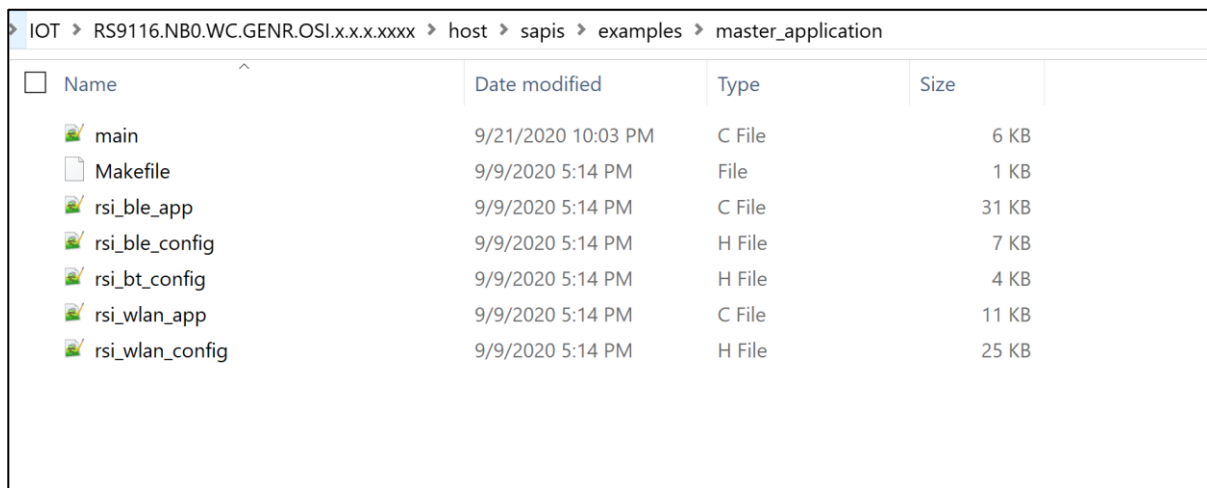
1. Double-click on **".project"** located in the below folder using CubeIDE.
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\platforms\STM32\Reference_Projects\Cube_Baremetal\Projects\SP\l\sample_project"



2. By default this project maps to dummy example code located in
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\sapis\examples\master_application"



3. To test specific example, navigate to folder
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\sapis\examples\master_application"



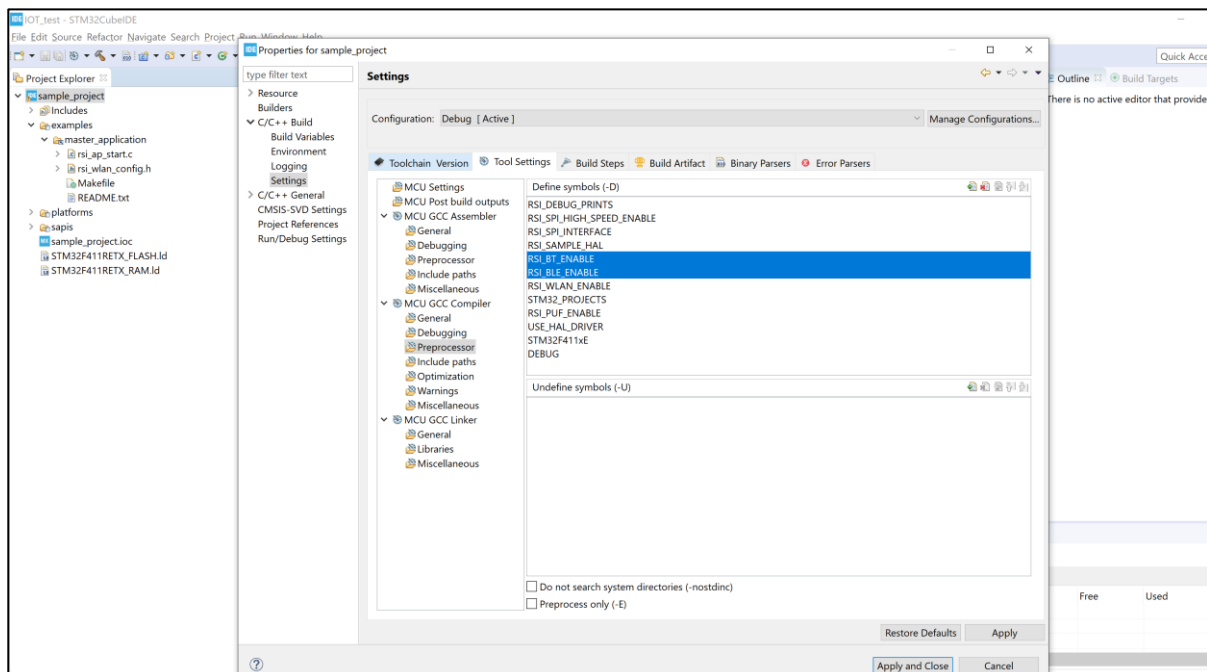
4. Delete the existing code, copy and paste the specific example code into the master_application folder.
 Ex: Below screenshot shows the example code copied from
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\sapis\examples\wlan\access_point" to
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\sapis\examples\master_application"

IOT > RS9116.NB0.WC.GENR.OSI.x.x.x.xxxx > host > sapis > examples > master_application				
<input type="checkbox"/> Name	Date modified	Type	Size	
Makefile	9/22/2020 11:54 AM	File	1 KB	
README	9/22/2020 11:54 AM	TXT File	1 KB	
rsi_ap_start	9/22/2020 11:54 AM	C File	10 KB	
rsi_wlan_config	9/22/2020 11:54 AM	H File	24 KB	

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.
For all examples, please ensure rsi_wlan_config.h, rsi_bt_config.h and rsi_ble_config.h files in the master_application for building the sample project.
- Any example folder doesn't contain all three configuration header files, in that case please use the default configuration headers 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 below screenshot for reference.



- Save the settings and compile the project.
- Repeat the same steps to compile different examples using this project.

2. Cube_FreeRTOS Projects (OS type)

Sample project is provided for compiling and building Wlan, BT, BLE examples in the release package. Please refer examples which are OS based.

Please follow the below steps for executing the sample project.

- The release package doesn't include FreeRTOS source for CubeIDE. So users has to follow steps in the section [FreeRTOS Porting for the STM32](#) for generating the FreeRTOS source for **CubeIDE specific**.

The generated FreeRTOS source shall be placed under
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\platforms\STM32\Reference Projects\Cube_Freertos\"

2. After the above Step1, Please follow the Cube_Baremetal Steps1 to 4 (from the above section) for compiling and building the sample_project located in

"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\platforms\STM32\Reference Projects\Cube_Freertos\Projects\SPI\sample_project"

Note: The sample project will get compiled only if the FreeRTOS code is placed as per the Step1.

Note:

Follow same steps to compile the sample_project located in
"RS9116.NB0.WC.GENR.OSI.x.x.x.xxx\host\platforms\STM32\Reference_Projects\Cube_Freertos\Projects\SP\sample_project"

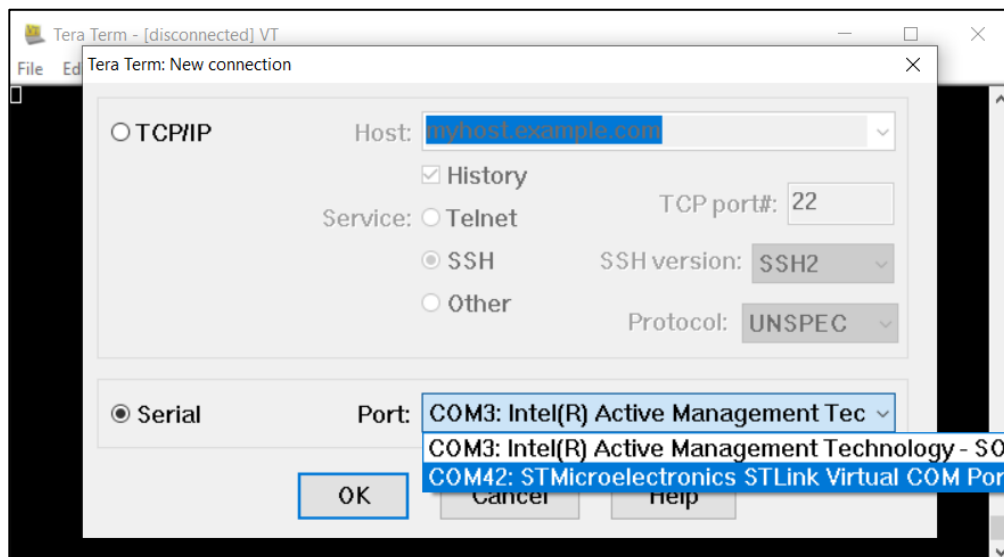
This project will get compiled only if the FreeRTOS package is present. To port the FreeRTOS package, refer [FreeRTOS Porting for STM32](#) section.

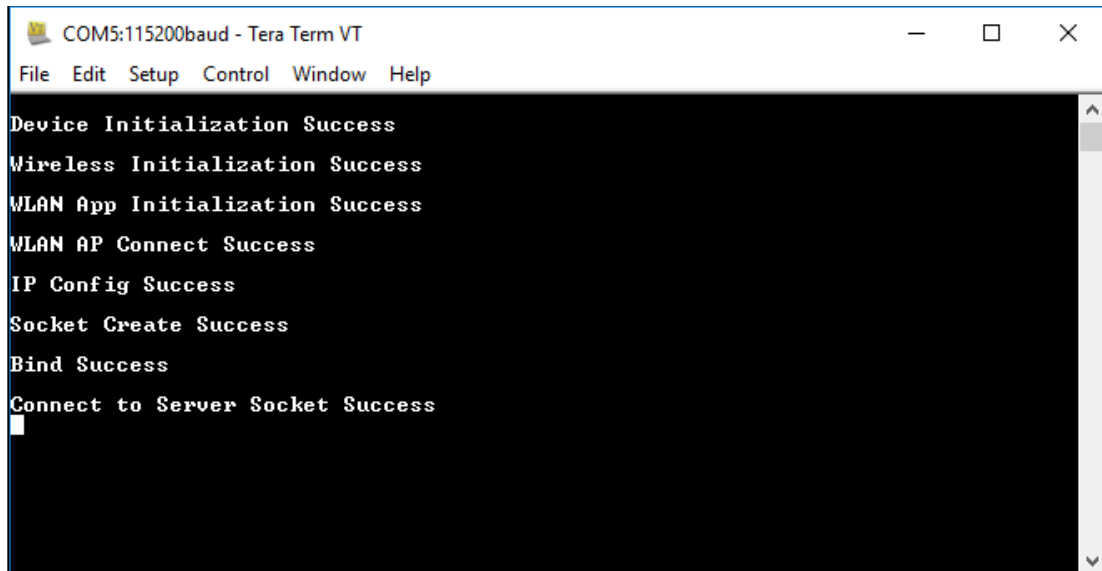
Note:

Debug prints are enabled by default in all the reference projects for Keil and Cube IDE.

In order to view the debug prints:

- Select the following Serial port settings: Baud rate - 115200, Data - 8 bit, Parity - none, Stop - 1 bit, Flow control - none
- Below is an example of debug prints for the wlan_station_ble_bridge reference project using Teraterm:

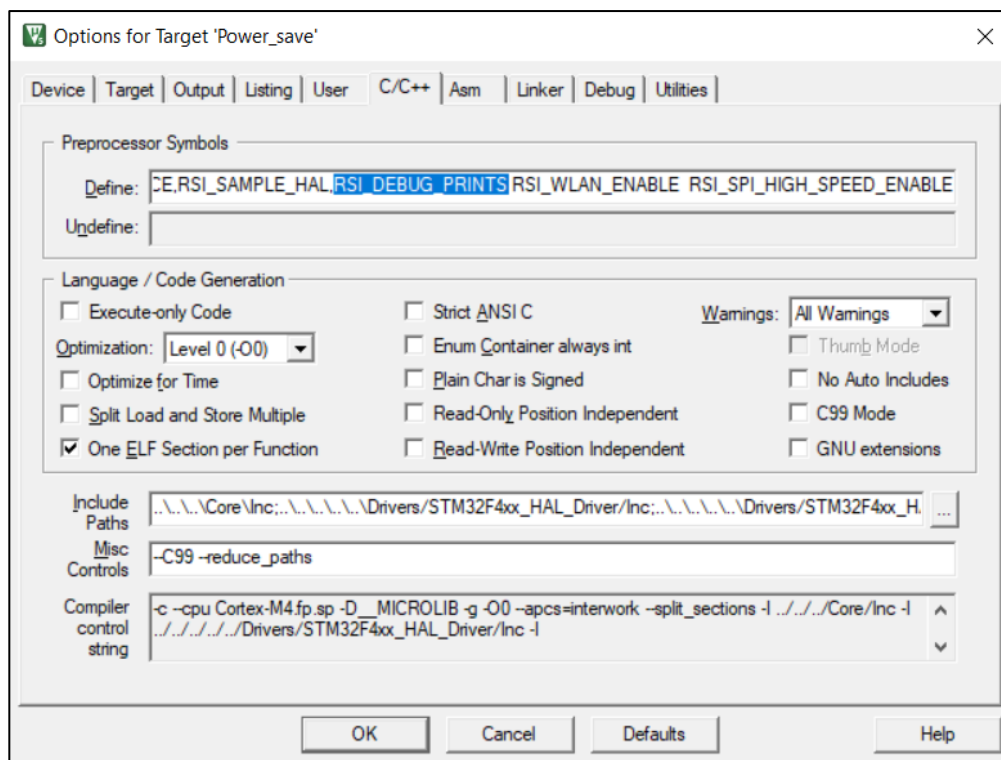




```
COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

Device Initialization Success
Wireless Initialization Success
WLAN App Initialization Success
WLAN AP Connect Success
IP Config Success
Socket Create Success
Bind Success
Connect to Server Socket Success
```

- To disable the debug prints, remove the Macro **RSI_DEBUG_PRINTS** from the project settings as below:



3.3 Reference Projects for Keil Baremetal

To run all the reference projects with Cube IDE, follow the section : [Getting Started with Keil IDE](#) in this document.

3.3.1 Example1: Throughput Application

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 Silabs Wireless SAPI framework provides necessary HAL APIs to enable a variety of host processors.

Note:

For this application refer to - "RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference Projects\Keil_Baremetal\Projects\SPI\Throughput"

WiSeConnect based Setup Requirements

- Windows PC with KEIL IDE
- Windows PC with Host interface(SPI/ UART/ USB-CDC/ USB) in case of WiSeConnect
- Silabs module connected through SPI with STM32 board
- Windows PC with application program like iperf
- Wireless Access point

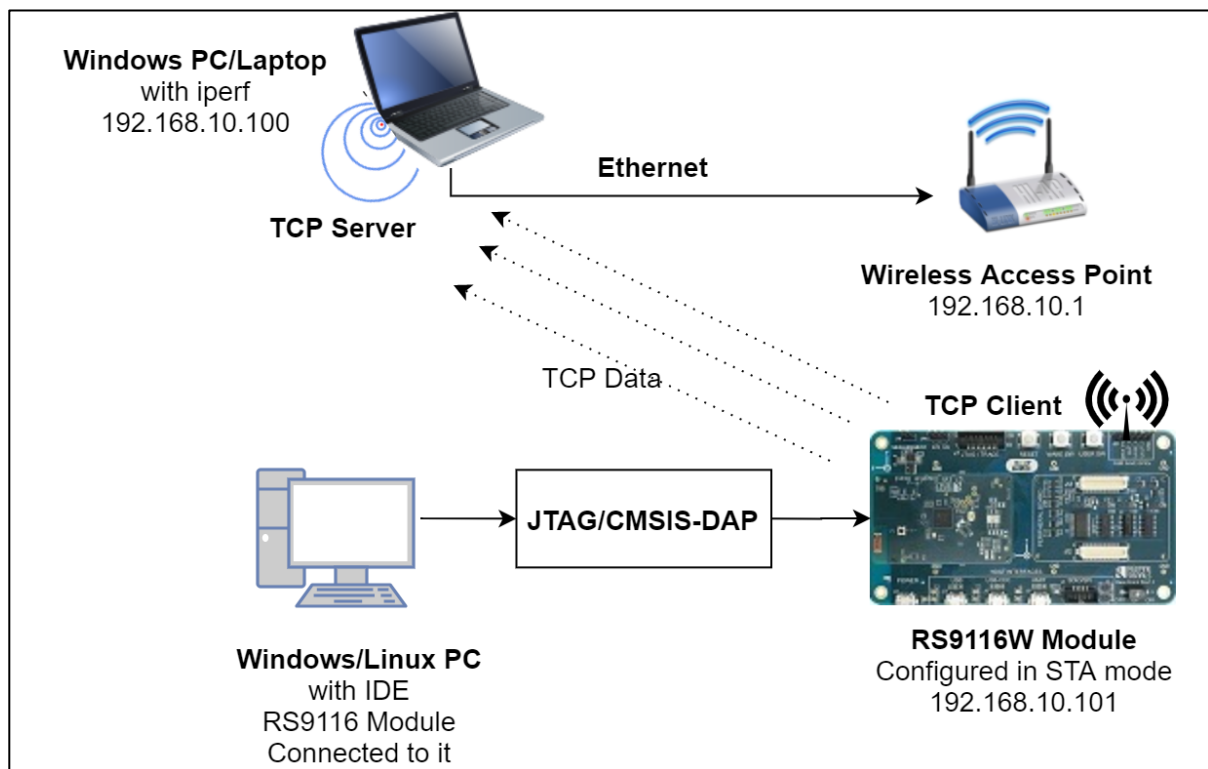


Figure 6: Module Configured in TCP Client Mode

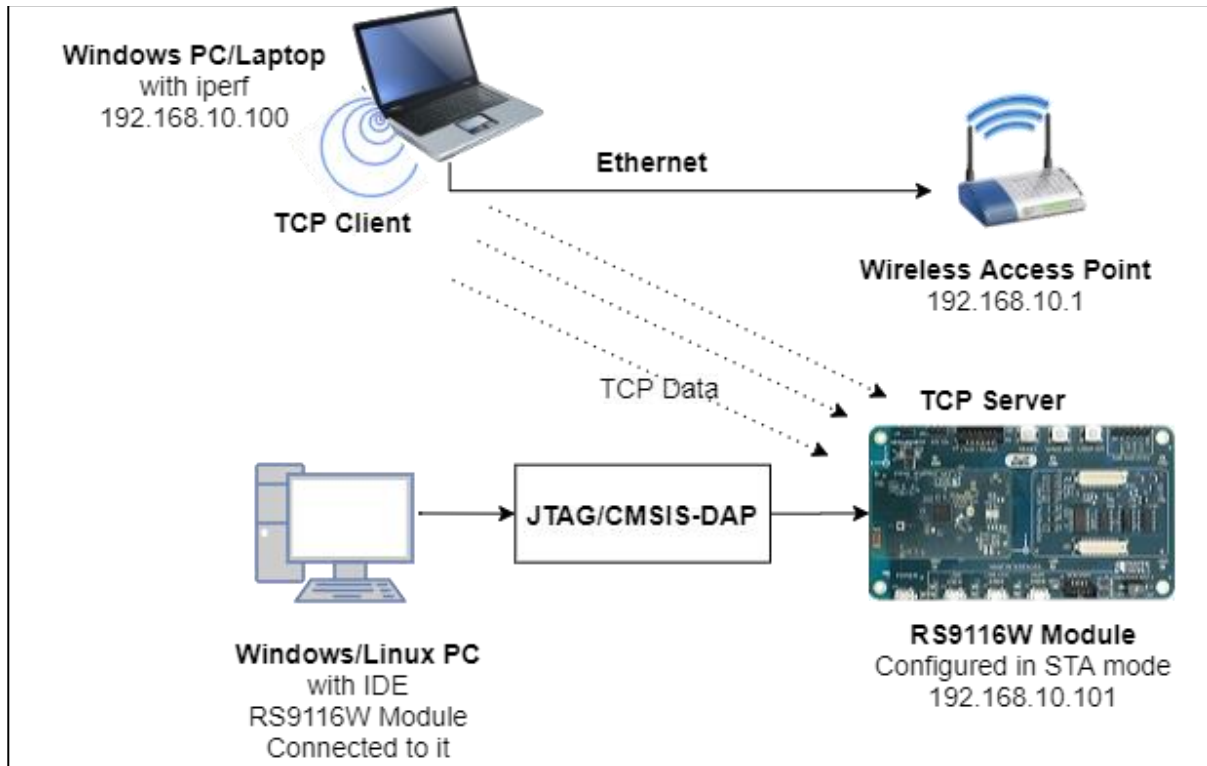


Figure 7: Module Configured in TCP Server Mode

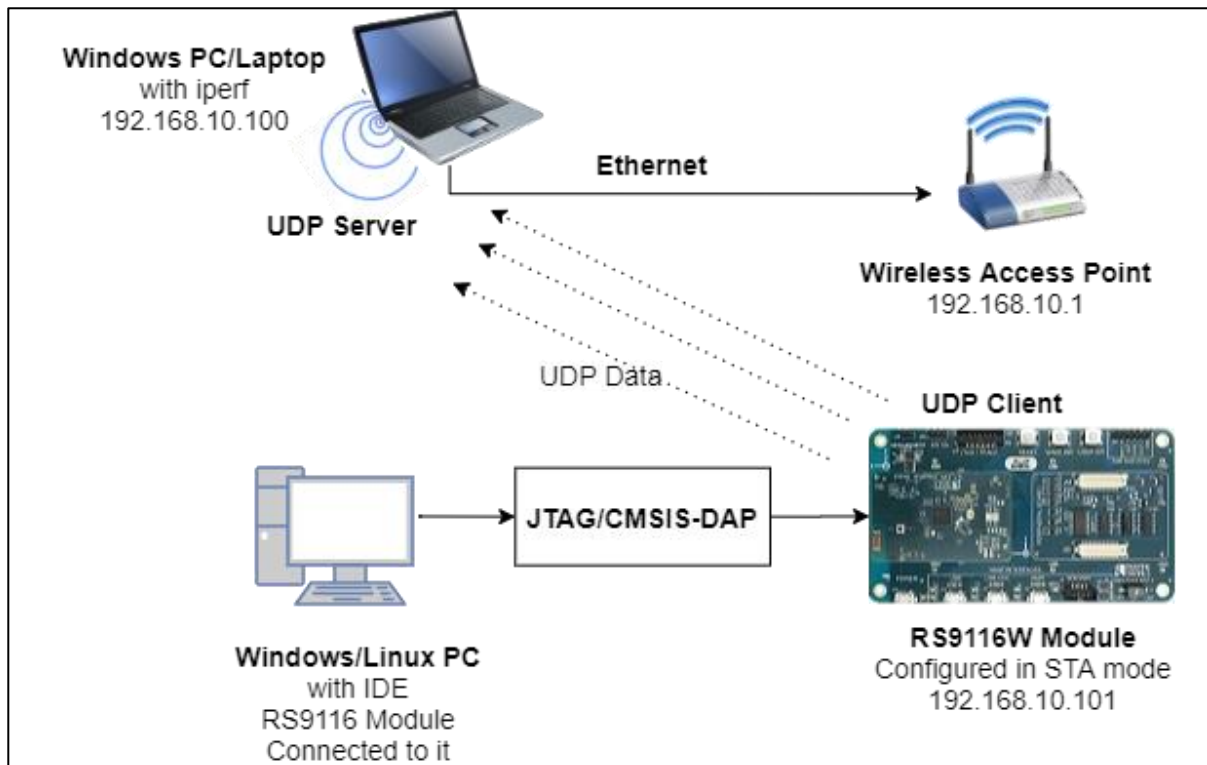


Figure 8: Module Configured in UDP Client Mode

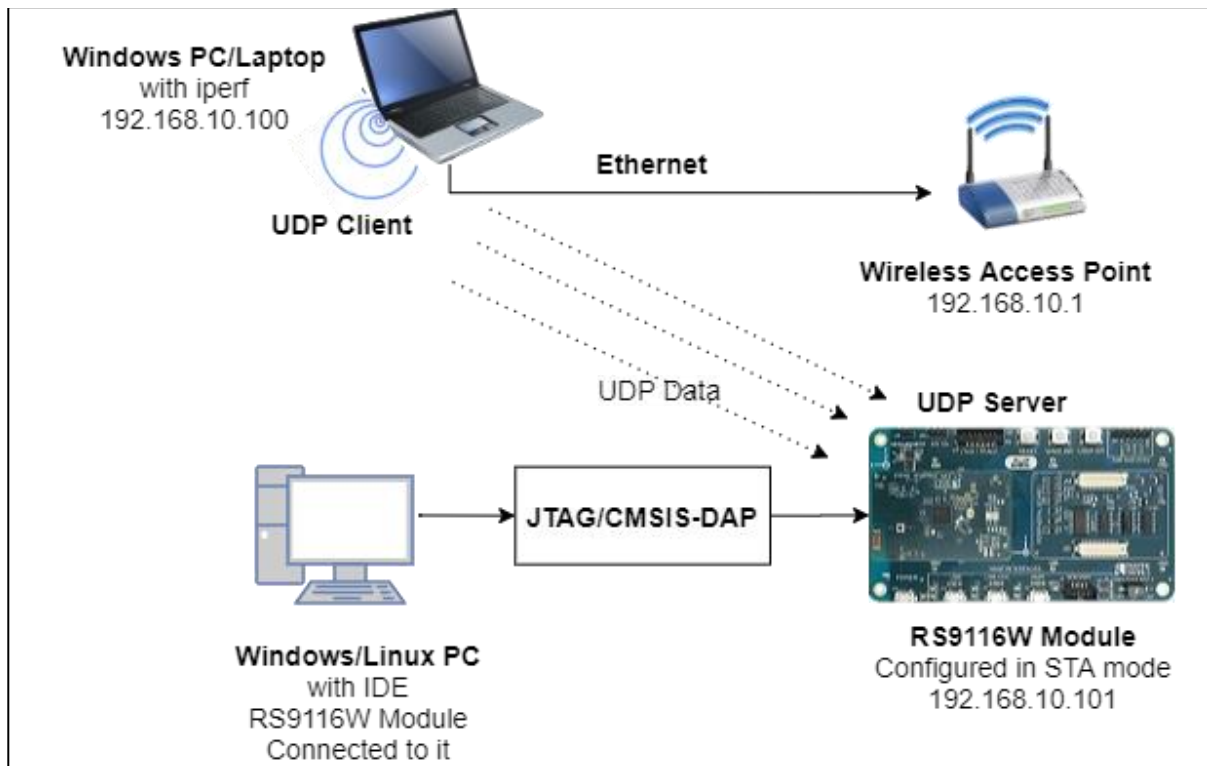


Figure 9: Module Configured in UDP Server Mode

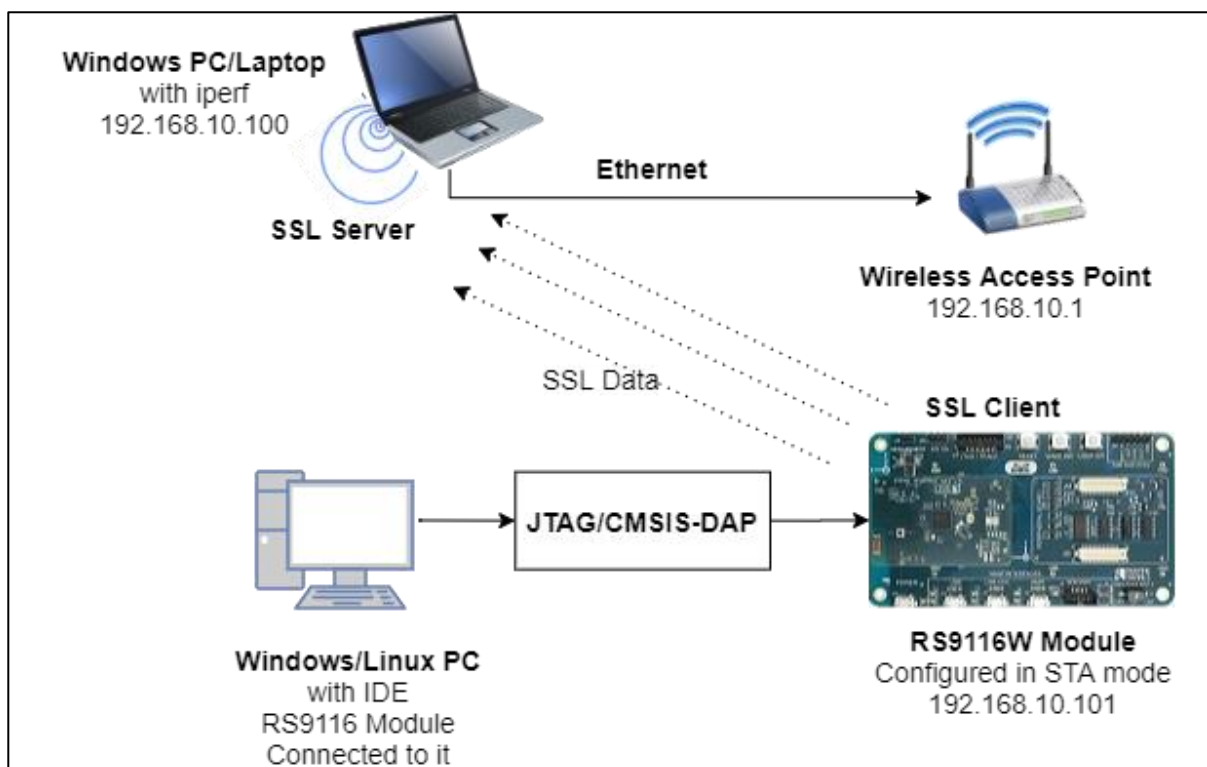


Figure 10: Module Configured in SSL Client Mode

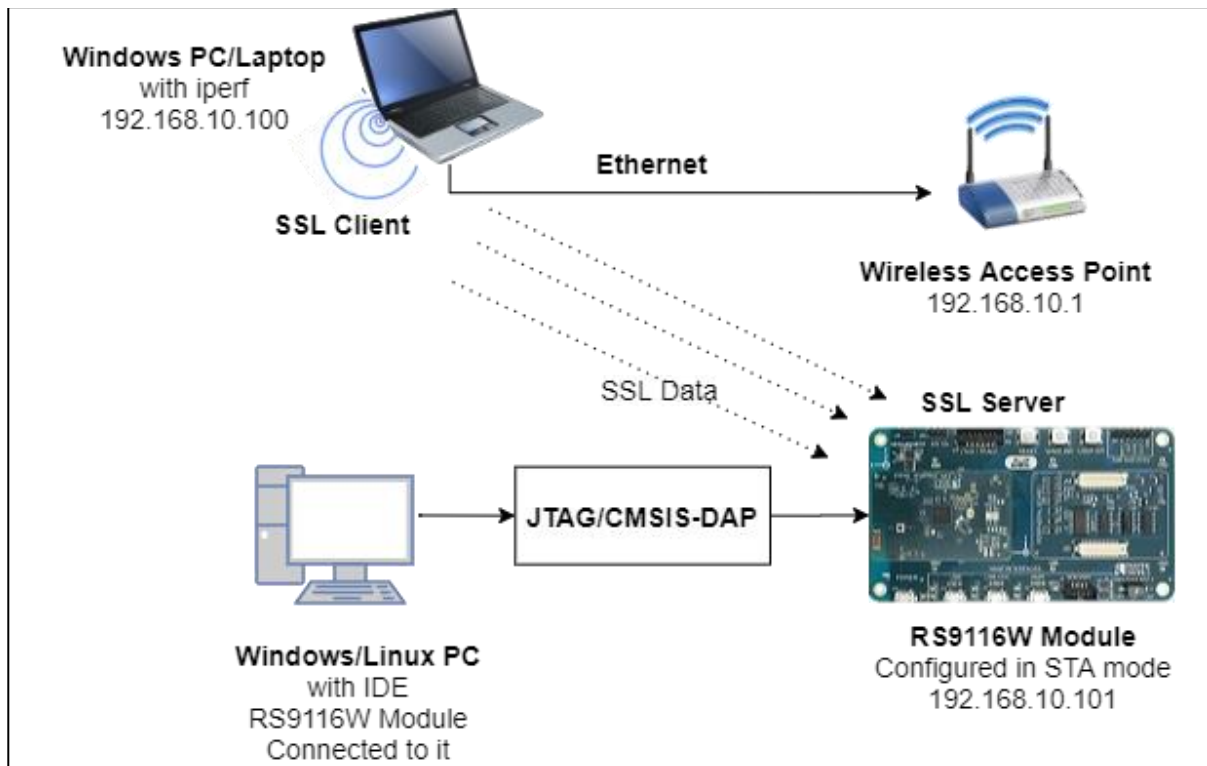


Figure 11: Module Configured in SSL Server Mode

Description:

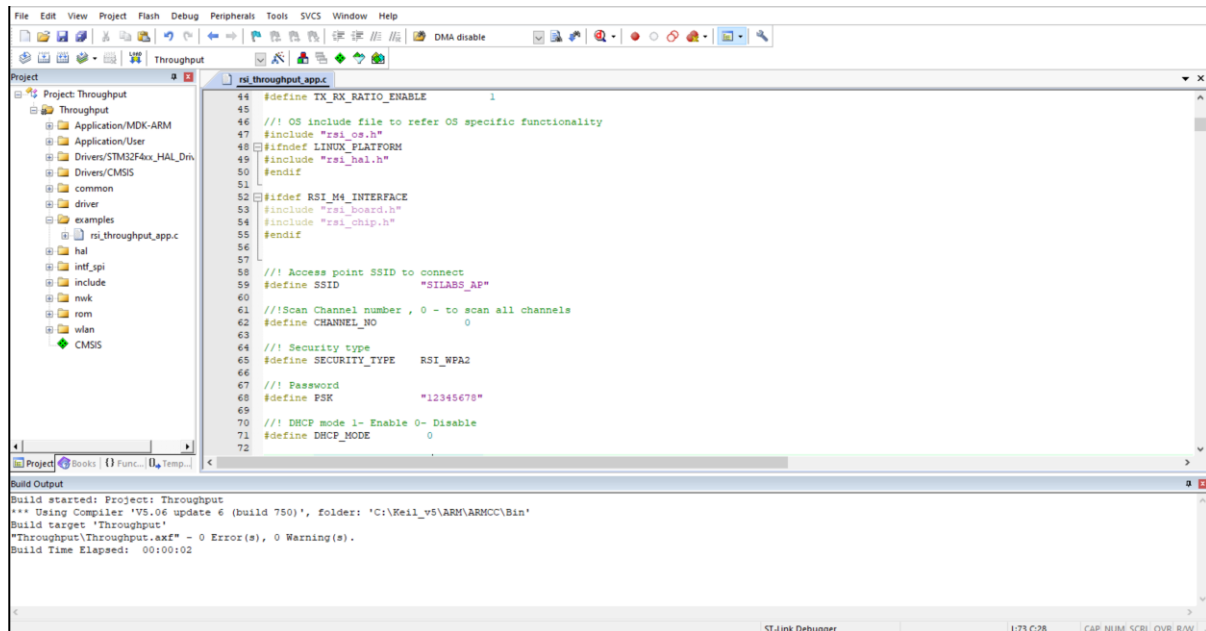
This application can be used to configure the Silicon Labs module in UDP client/server or TCP client/server SSL client/server. To measure throughput, the following configuration can be applied.

1. To measure TCP Tx throughput, the module should be configured as a TCP client.
2. To measure TCP Rx throughput, the module should be configured as a TCP server.
3. To measure UDP Tx throughput, the module should be configured as a UDP client.
4. To measure UDP Rx throughput, the module should be configured as a UDP server.
5. To measure SSL Tx throughput, the module should be configured as an SSL client.
6. To measure SSL Rx throughput, the module should be configured as an SSL server.

Configuration and Steps for Execution

Configuring the Application:

1. Open Project **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\Throughput** and then double click on **Throughput** (uVision5 Project).
Then open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\throughput_app\rsi_throughput_app.c** file and update/modify the following macros:



SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

CHANNEL_NO refers to the channel in which the device should scan. If it is 0, the device will scan all channels.

```
#define CHANNEL_NO 0
```

Note

Valid values for CHANNEL_NO in the 2.4GHz band are 1 to 11 and the 5GHZ band is 36 to 48 and 149 to 165. In this example default configured band is 2.4GHz.

So, if the user wants to use a 5GHz band then the 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. The access point supports Open, WPA, WPA2 securities. The 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

The IP address to be configured to the device should be in a 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
```

The 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
```

The 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 5001
```

The 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
```

The 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.

The application can select throughput type as UDP Tx, UDP Rx, TCP Tx, TCP Rx, SSL Tx, SSL Rx. The following is a 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
#define SSL_TX          4
#define SSL_RX          5
```

Note

In AP mode, configure the same IP address for both DEVICE_IP and GATEWAY macros.

3. Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\throughput_app\rsi_wlan_config.h** file and update/modify the 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 |
TCP_IP_FEAT_SSL | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP (FEAT_CUSTOM_FEAT_EXTENSION_VALID |
CUSTOM_FEAT_SOC_CLK_CONFIG_160MHZ)
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP EXT_FEAT_256K_MODE

#define RSI_EXT_TCPIP_FEATURE_BITMAP EXT_TCP_IP_WINDOW_DIV

#define RSI_BAND                  RSI_BAND_2P4GHZ
```

3.3.1.1 Executing the Application

1. Connect the Silabs module through SPI to the STM32 board with the provided SPI header.
2. Configure the macros in the files **rsi_wlan_config.h** as mentioned above in Configuring the Application
3. To measure throughput, the following configuration can be applied.

Note

To Install the iperf software click on the link: <https://iperf.fr/iperf-download.php>

Please note that for this example, we have used iperf version 2.0.9.

4) Make sure that the iperf is running before running the project. (Except Throughput UDP RX and TCP RX which should be running after running the project).

a) To measure UDP Tx throughput, the module should be configured as a UDP client. Open UDP server at the remote port

```
iperf.exe -s -u -p <SERVER_PORT> -i 1
```

b) To measure UDP Rx throughput, the module should be configured as a UDP server. Open UDP client at the remote port

```
iperf.exe -c <Module_IP> -u -p <Module_Port> -i 1 -b <Bandwidth>
```

c) To measure TCP Tx throughput, the module should be configured as a TCP client. Open the TCP server at the remote port.

```
iperf.exe -s -p <SERVER_PORT> -i 1
```

d) To measure TCP Rx throughput, the module should be configured as a TCP server. Open TCP client at the remote port.

```
iperf.exe -c <Module_IP> -p <module_PORT> -i 1 -t 100
```

To get the Silabs module IP, first run the program so that the module connects to the access point and then find the Silabs module IP through ip_buff in the code, and re-run the application by using that IP to run command in iperf.

e) To measure SSL Tx throughput, the module should be configured as an SSL client. Open the SSL server at a 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, the module should be configured as an SSL server. Open SSL client at a 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>
```

5. Build the project and flash the binary into the STM32 board.

6. After the program gets executed, the device would be connected to an Access point having the configuration the same as that of in the application and get.

7. 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

Note: If M4 frequency needs to Switch higher clock then follow the below steps.

step1 : Call **switch_m4_frequency()** api after device initialization.

step2: Update systics to higher clock as ; **SysTick_Config(SystemCoreClock /1000)**.

3.3.2 Example2: Wireless Firmware Upgradation

Overview

The Wireless Firmware upgrade application demonstrates how the WiSeConnect device would be created as an Access point and allow stations to connect to update the firmware through the webpage.

Sequence of Events

This Application explains user how to:

- Silabs Device acts as an Access point
- Allows stations to connect to it to update the firmware through a webpage.
- A PC having a WiFi card can be used as a Wireless station.
- Silabs device creates a network and acts as a 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 Silabs Wireless SAPI framework provides necessary HAL APIs to enable a variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with Keil IDE
- Silabs module connected through SPI with STM32 board

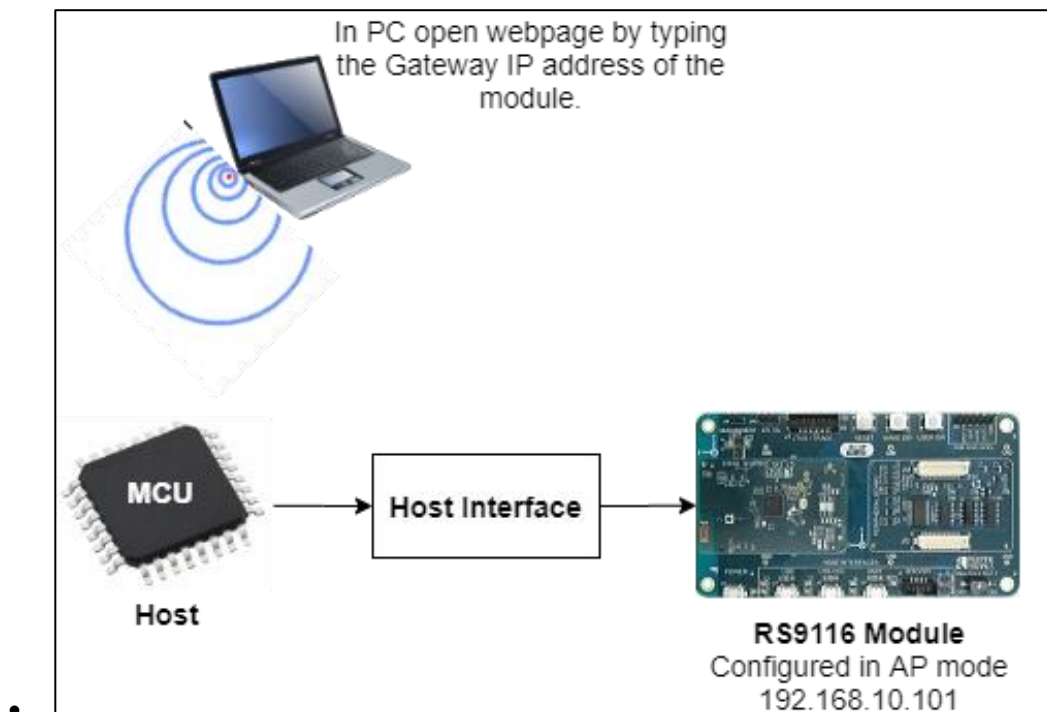
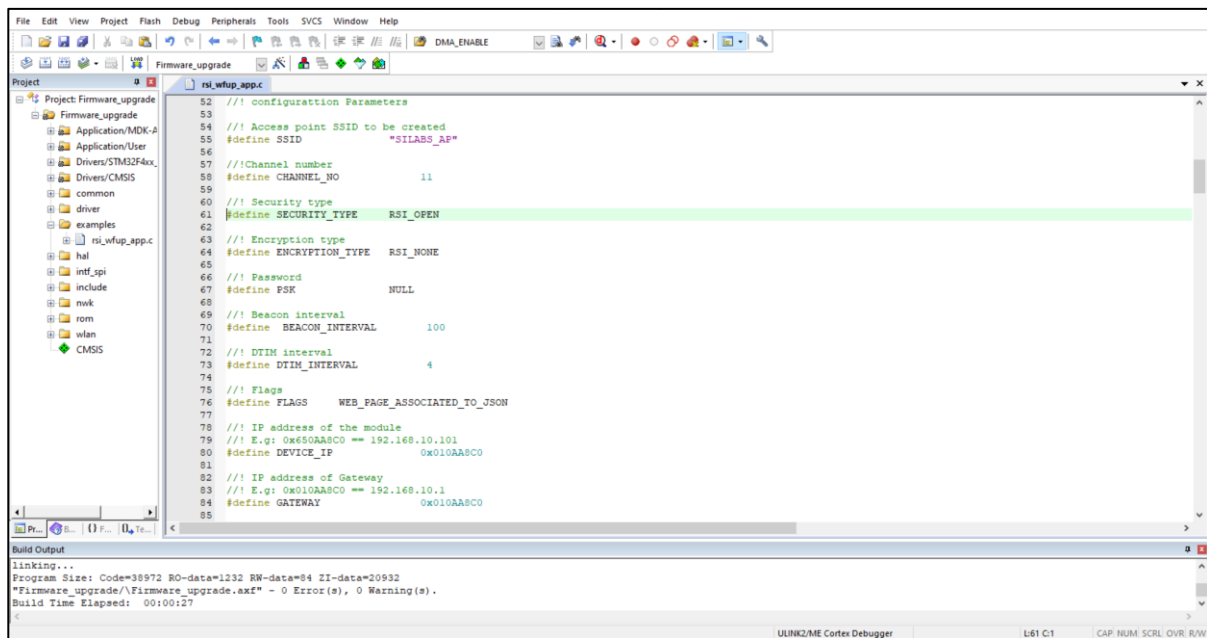


Figure 12: Setup Diagram for Wireless Firmware Upgrade

Configuration and Steps for Execution

Configuring the Application

1. Open Project `RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\Firmware_upgrade\` and then double click on `Firmware_upgrade` (uVision5 Project).
Then
open `RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\wireless_firmware_upgradation\rsi_wfup_app.c` file and update/modify following macros:



SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

CHANNEL_NO refers to the channel in which the device should scan. If it is 0, the device will scan all channels.

```
#define CHANNEL_NO 11
```

Note

Valid values for CHANNEL_NO in the 2.4GHz band are 1 to 11 and the 5GHz band is 36 to 48 and 149 to 165. In this example default configured band is 2.4GHz. So, if the user wants to use a 5GHz band then the 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. The access point supports Open, WPA, WPA2 securities

The 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. The access point supports Open, TKIP, CCMP methods.

The valid configuration is: RSI_CCMP - For CCMP encryption RSI_TKIP - For TKIP encryption RSI_NONE - For open encryption

```
#define ENCRYPTION_TYPE <encryption_type>
```

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 in milliseconds. Allowed values are integers from 100 to 1000 which are multiples of 100.

```
#define BEACON_INTERVAL 100
```

DTIM_INTERVAL refers to the DTIM interval of the Access Point. Allowed values are from 1 to 255.

```
#define DTIM_INTERVAL 4
```

FLAGS :

```
#define FLAGS WEB_PAGE_ASSOCIATED_TO_JSON
```

- To configure the IP address
The IP address to be configured to the device should be in a 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
```

The 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
```

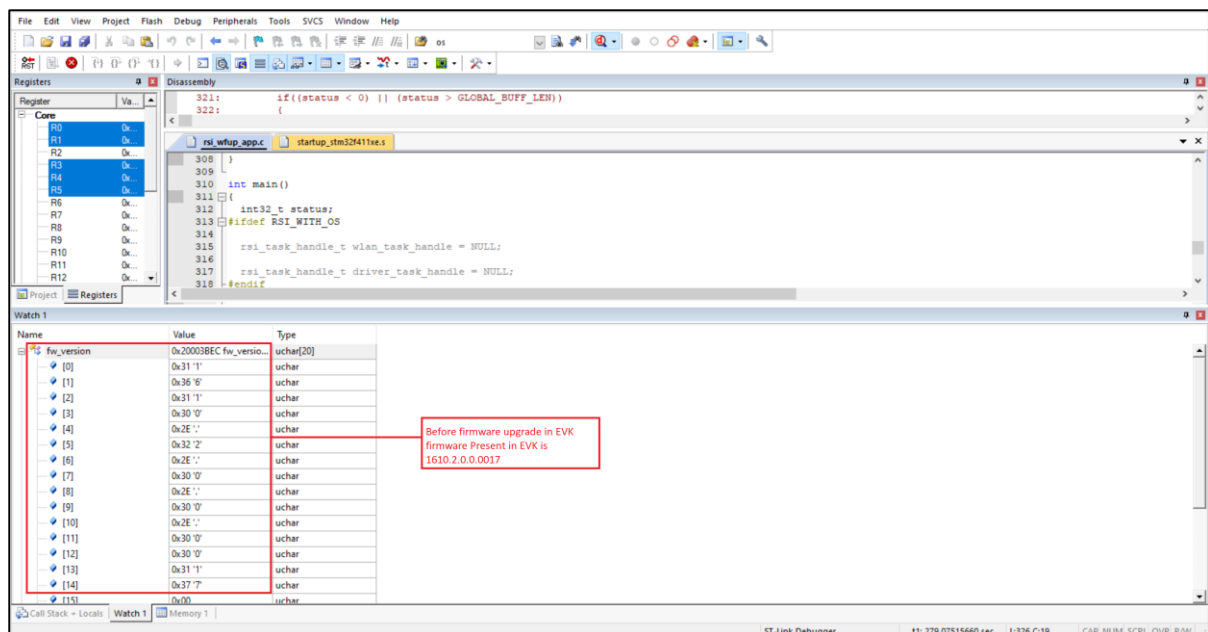
The IP address of the network mask should also be in long format and in little-endian byte order
Example: To configure "255.255.255.0" as a network mask, update the macro NETMASK as 0x00FFFFFF

```
#define NETMASK 0x00FFFFFF
```

- Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\examples\wlan\wireless_firmware_upgradation\rsi_wlan_config.h** file and update/modify following macros:

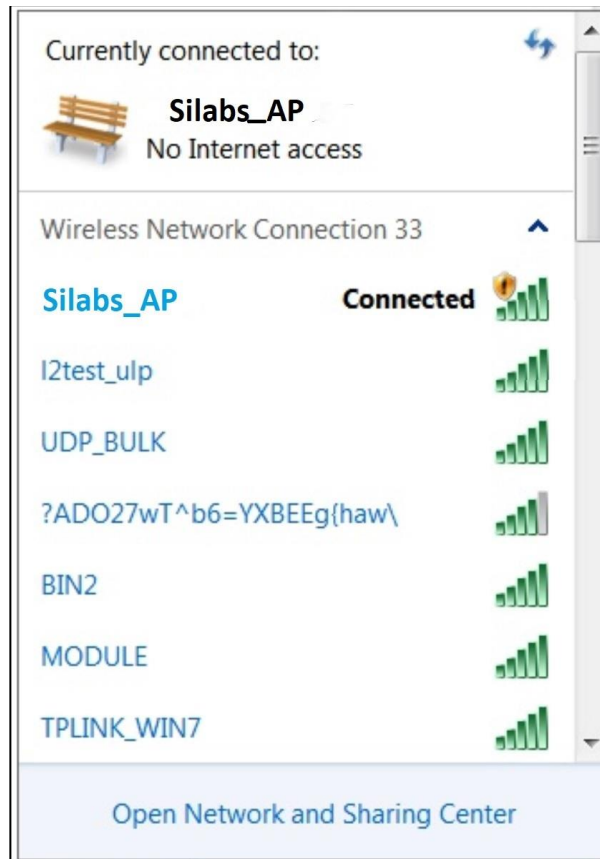
```
#define CONCURRENT_MODE RSI_DISABLE
#define RSI_FEATURE_BIT_MAP FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP (TCP_IP_FEAT_DHCPV4_SERVER | TCP_IP_FEAT_HTTP_SERVER)
#define RSI_CUSTOM_FEATURE_BIT_MAP FEAT_CUSTOM_FEAT_EXTENTION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP EXT_FEAT_256K_MODE
#define RSI_BAND RSI_BAND_2P4GHZ
```

Firmware Version check: Before firmware up-gradation check the Firmware Version of the Silabs Device by running the project once, you could see the firmware version through fw_version in the code. In this figure left hand corner (fw_version = 1.1.2) is displayed.

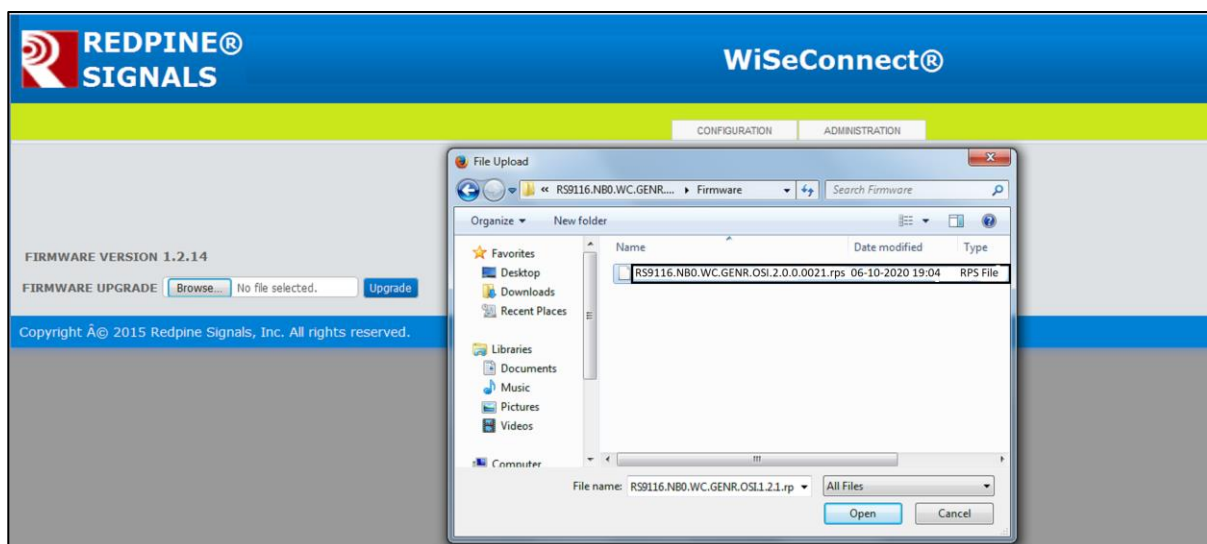


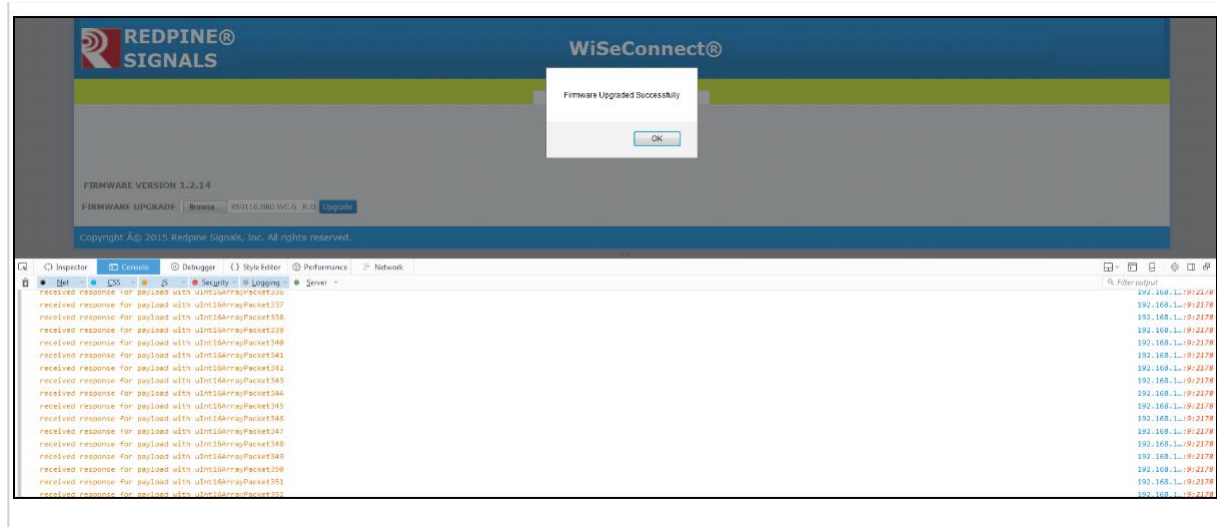
Executing the Application

- Connect the Silabs module through SPI to the STM32 board with the provided SPI header.
- Configure the macros in the files rsi_wfup_app.c and rsi_wlan_config.h as per above Configuring the Application.
- Build the project and flash the binary into the STM32 board.
- After the program gets executed, the Silabs device will act as an Access point and starts beaconing.

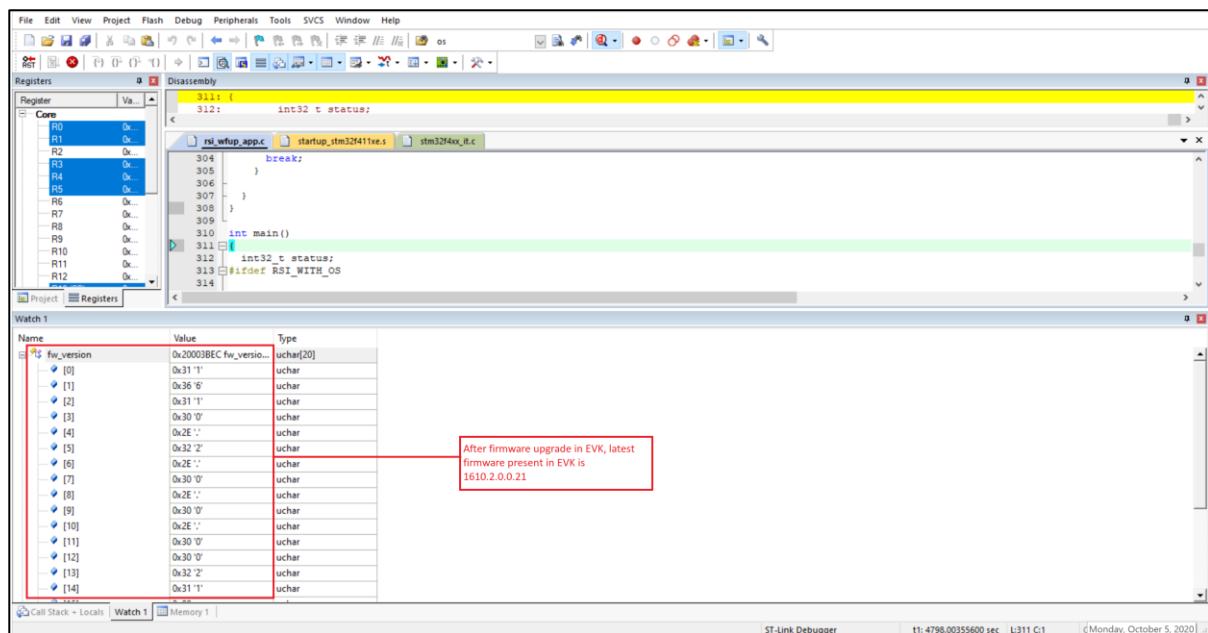


5. After the Silabs module act as an access point connect with the Wi-Fi station(PC) and open the Webpage by typing the Gateway IP address of the module. (Example: 192.168.10.1 this Gateway IP is write in a new tab and press Enter. After, open the webpage, it shows two-parameters like Configuration and Administration.
6. Click on Administration:-- In this part just browse the .rps file in which you want to update the firmware in your module. then click on the Update button. Next press Ctrl+shift+k to show the firmware package. After an update the firmware successfully it displays "Upgradation successfully".





7. After Firmware up-gradation, the Device needs to be a reboot to get effective of the new firmware file. After the first reboot, the device will take a few minutes(~2 min) to load the new firmware.8. Check the updated Firmware version using fw_version.



3.3.3 Example3: 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, the Silabs BTLE device connects with a remote BTLE device (Smart Phone) and the Silabs Wi-Fi interface connects with an Access Point in station mode and does data transfer in BTLE and Wi-Fi 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 the Access point. Smartphone interacts with the BLE task, while remote PC interacts with the WLAN task. When Smartphone connects and sends a message to the Silabs device, the BLE task accepts and sends to the WLAN task, which in turn sends to the remote PC which is connected to Access Point. Similarly, when a remote PC sends a message to the Silabs device, the message will be sent to smartphones via the BLE task.

Thus messages can be seamlessly transferred between PC and Smartphone.

Sequence of Events

WLAN Task

This Application explains user how to:

- Create a Silabs device as Station
- Connect Silabs station to the remote Access point
- Receive TCP data sent by the connected station and forward to BT task
- Send data received by BT task to the connected station using TCP over SSL client protocol.

BLE Task

This Application explains user how to:

- Create chat service
- Configure Silabs device in advertise mode
- Connect from Smartphone to Silabs device
- Receive data sent by Smartphone and forward to WLAN task
- Send data received by WLAN task and send to Smartphone

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 Silabs Wireless SAPI framework provides necessary HAL APIs to enable a variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with KEIL IDE
- Windows PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silabs module connected through SPI with STM32 board
- Wireless Access point
- Windows PC with SSL server application (OpenSSL)
- BTLE supported smartphones with GATT client applications.

Note

Install the BLE scanner for the GATT client application.

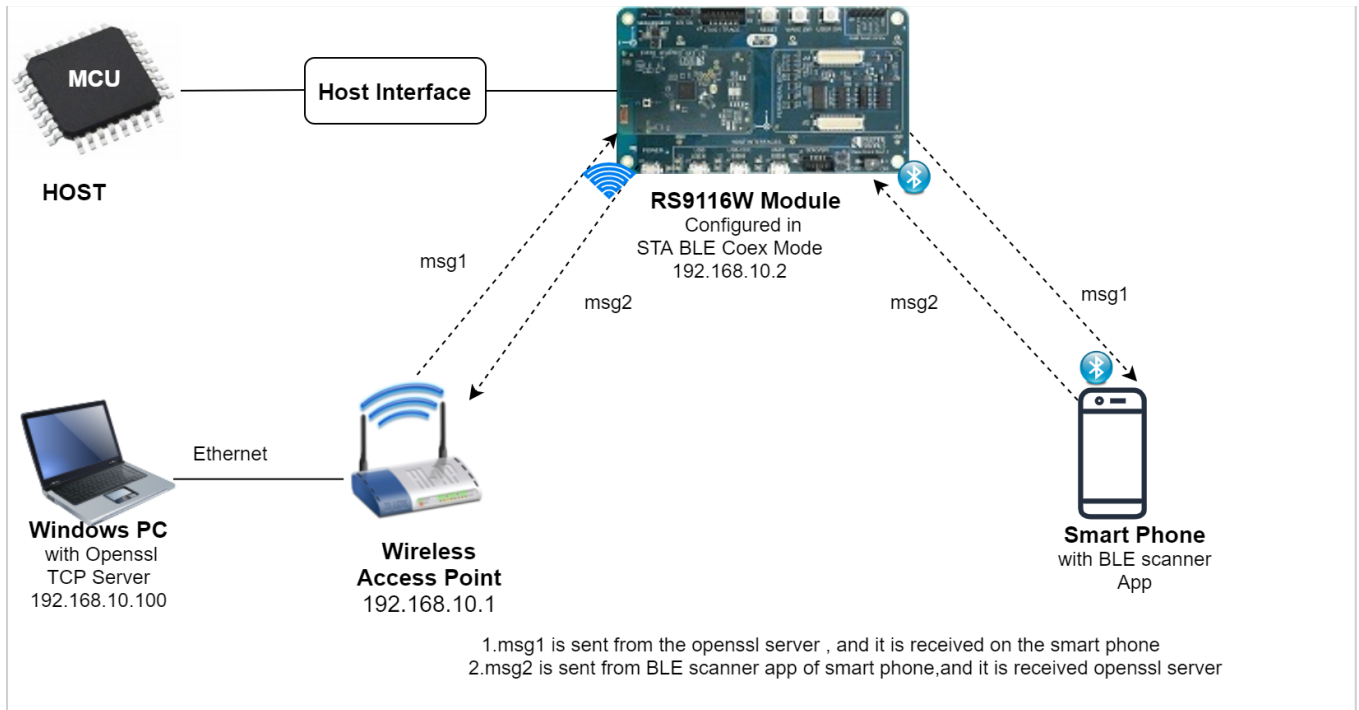


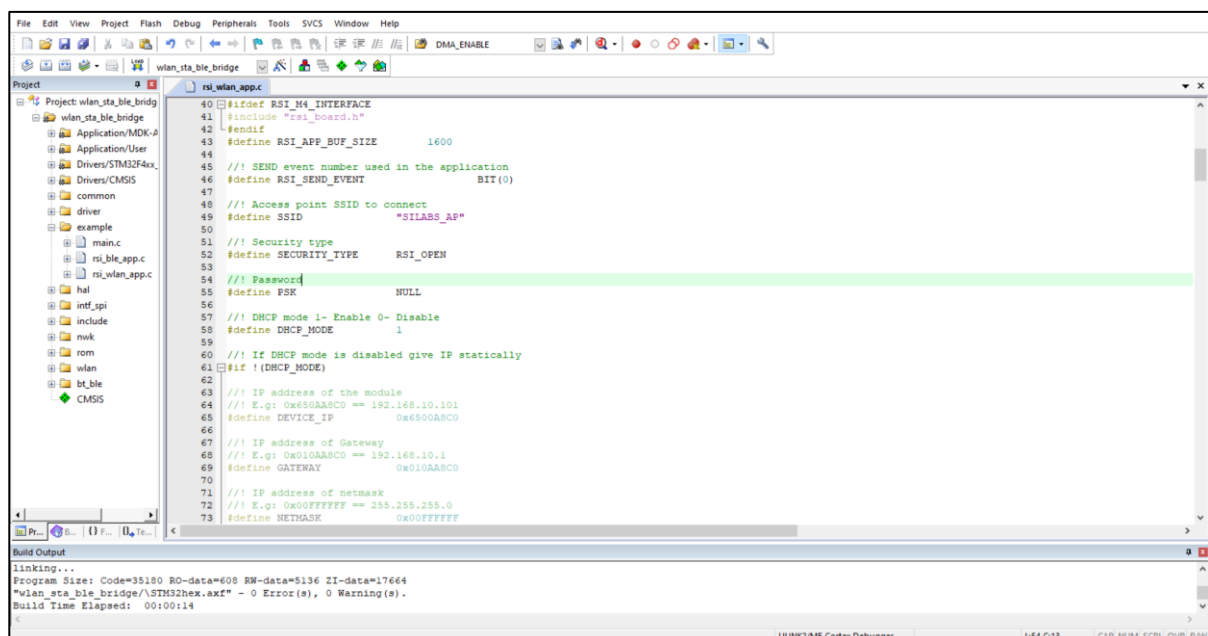
Figure 13: Setup to Demonstrate WLAN Station BLE Bridge Application

Configuration and Steps for Execution

Configuring the Application

Configuring the WLAN task

1. Open Project
RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\wlan_sta_ble_bridge and then double click on **wlan_sta_ble_bridge** (uVision5 Project).
Then
open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan_ble\wlan_station_ble_bridge\rsi_wlan_app.c** file and update/modify the following macros:



SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

SECURITY_TYPE refers to the type of security. In this application, STA supports Open, WPA-PSK, WPA2-PSK securities.

The 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, the application will load the certificate which is included using `rsi_wlan_set_certificate` API.

By default, the application loading the "cacert.pem" certificate if **LOAD_CERTIFICATE** enables. In order to load a different certificate, the user has to follow the following steps:

- `rsi_wlan_set_certificate` API expects the certificate in the form of a 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 cacert.pem**

The script will generate wifiuser.pem in which one linear array named cacert contains the certificate.

- After the conversion of the certificate, update the `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 the certificate loads into the device, it will write into the device flash. So, users need not load the certificate for every boot up unless the certificate change.

So define **LOAD_CERTIFICATE** as 0, if the certificate is already present in the Device.

Note

All the certificates are given in the release package *certificates path is RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\utilities\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 the SSL server socket. The IP address should be in a 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 to whether the IP address configured through DHCP or STATIC

```
#define DHCP_MODE 1
```

Note

If the user wants to configure the 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 the STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following **DEVICE_IP**, **GATEWAY**, and **NETMASK** macros.

The IP address to be configured to the device in STA mode should be in a 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
```

The 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
```

The 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**

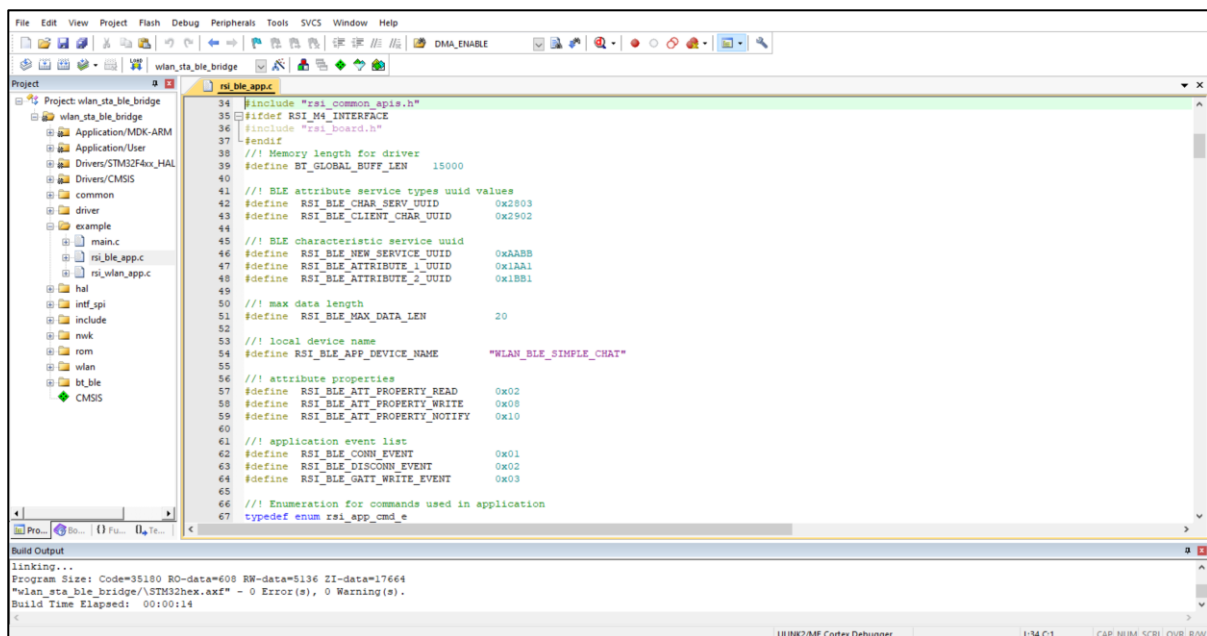
```
#define NETMASK 0x00FFFFFF
```

- Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\examples\wlan_ble\wlan_station_ble_bridge\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_CLIENT |
TCP_IP_TOTAL_SOCKETS_1 | TCP_IP_FEAT_SSL | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP FEAT_CUSTOM_FEAT_EXTENSION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BITMAP EXT_DYNAMIC_COEX_MEMORY
#define RSI_BAND RSI_BAND_2P4GHZ
```

Configuring the BLE Application

- Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\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 to the name of the Silabs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_DEVICE_NAME "WLAN_BLE_SIMPLE_CHAT"
```

The 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 to the number of bytes required by the application and the driver.

```
#define BT_GLOBAL_BUFF_LEN 10000
```

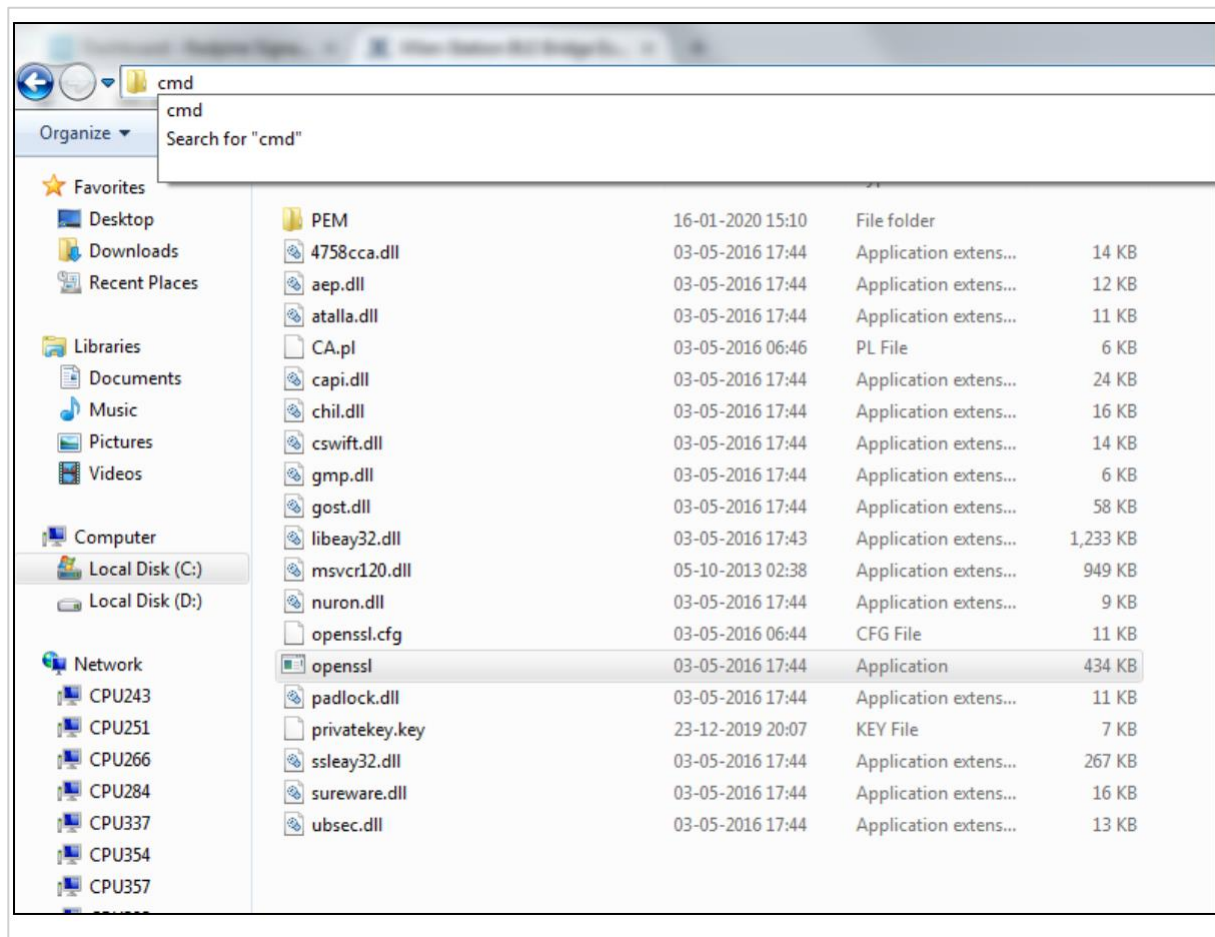
Executing the Application

1. Connect the Silabs module through SPI to the STM32 board with the provided SPI header.
2. Find the Openssl Window32/64 bit installation link :
<https://siproweb.com/products/Win32OpenSSL.html>

Note

Application was tested in Win32OpenSSL_Light-1_1_1d and Win64OpenSSL_Light-1_1_1d.

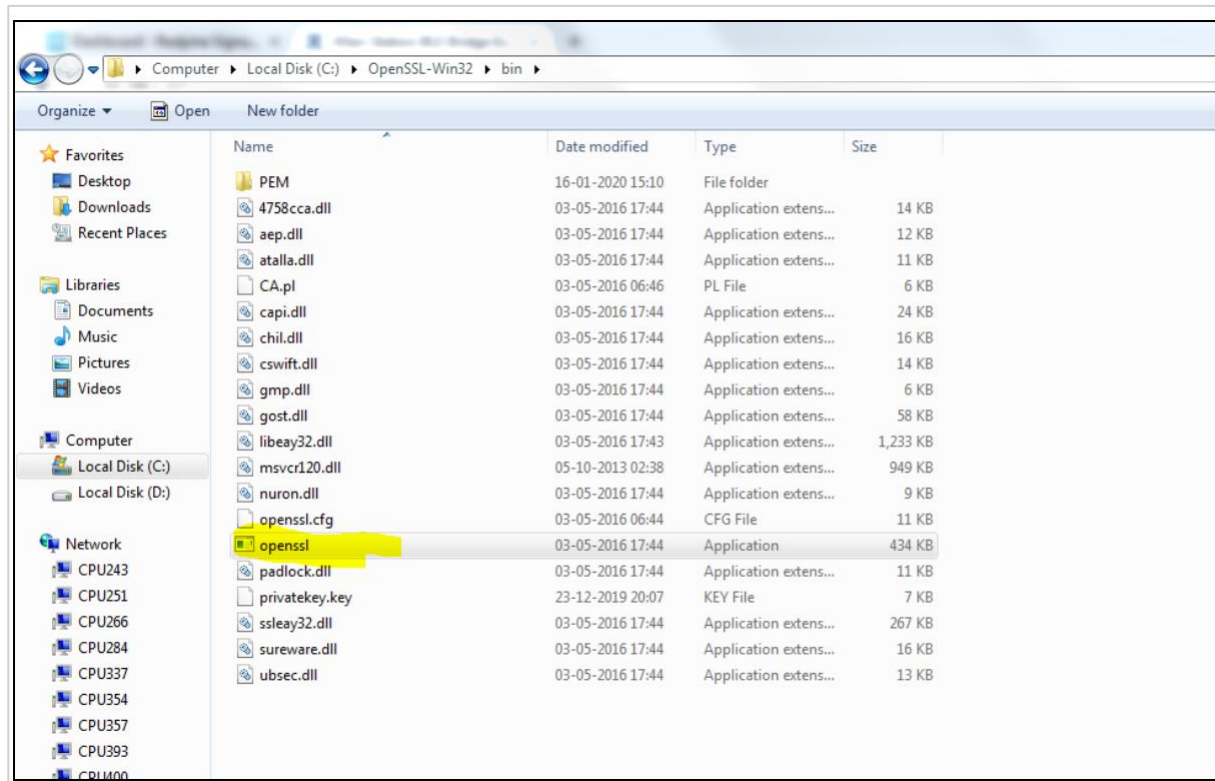
3. After installing the software of Openssl, copy all certificates from this path (RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\utilities\certificates) to paste on this path (C:\OpenSSL-Winx\bin).
4. Navigate to the installed Openssl Application folder, follow this path : C:\OpenSSL-Winx.x\bin and open the command prompt by the typing cmd in root path of the directory then after 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>

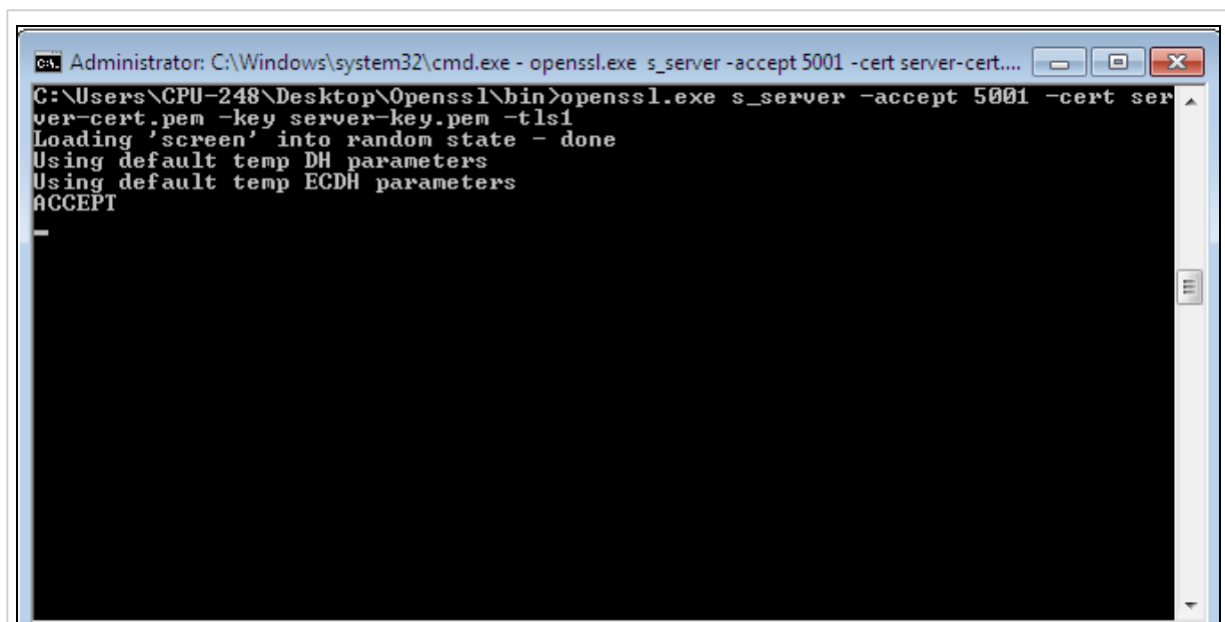
(OR)

Directly click on the Openssl application. (refer to figure below)

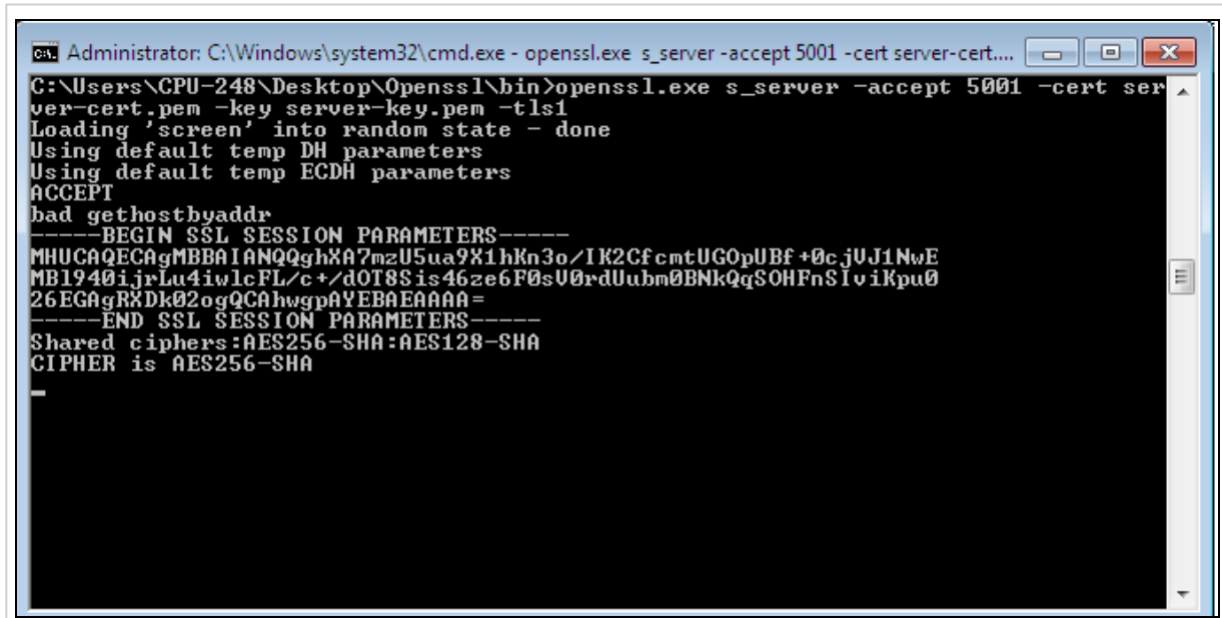


s_server -accept<SERVER_PORT> -cert <server_certificate_file_path> -key <server_key_file_path> -tls<tls_version>

Example: s_server -accept 5001 -cert server-cert.pem -key server-key.pem -tls1



5. After the program gets executed, Silabs 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.



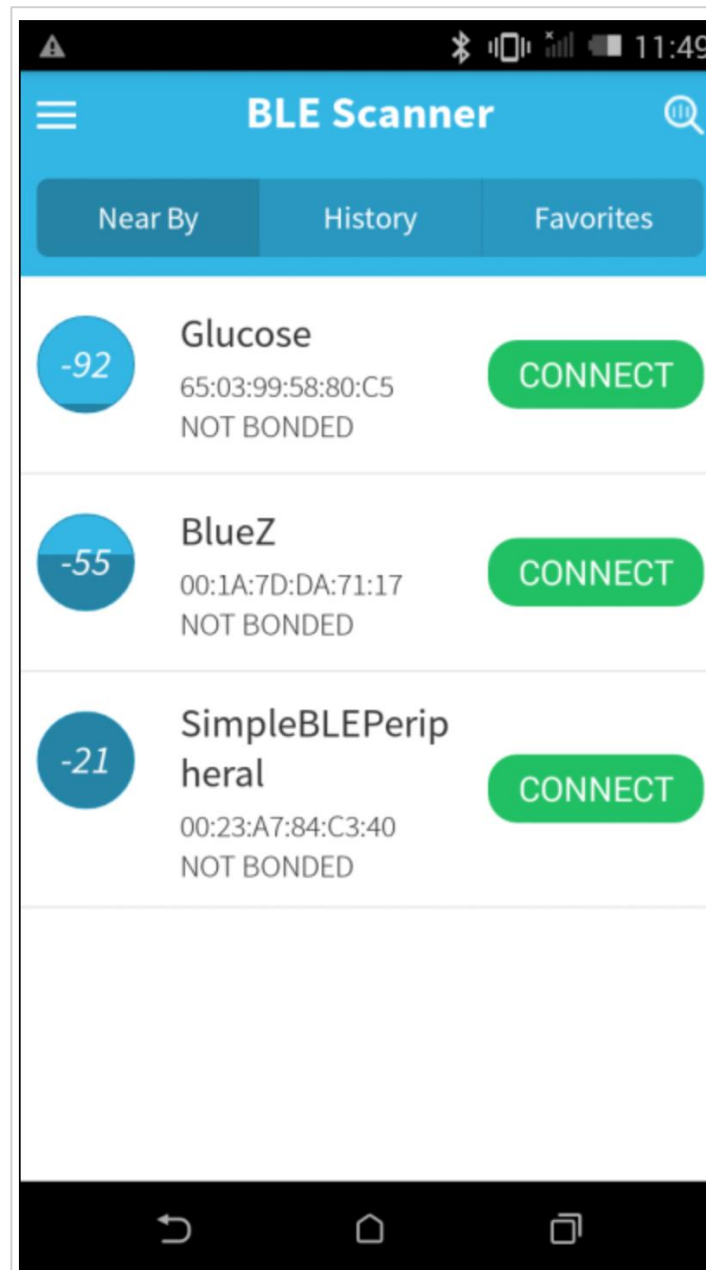
```

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-----
MHUCAQECagMBBAIANQQghXA7mzU5ua9X1hKn3o/IK2CfcmtUGOpUBf+0cjUJ1NwE
MB1940ijrLu4iwlcFL/c+/dOT8Sis46ze6F0sV0rdUubm0BNkQqSOHPnSivikpu0
26EGAgRXDk02ogQCAhwgpAYEBAAAA=
-----END SSL SESSION PARAMETERS-----
Shared ciphers:AES256-SHA:AES128-SHA
CIPHER is AES256-SHA

```

6. Open BLE scanner App in the Smartphone and scan for the Silabs device.

7. In the App, the Silabs device will appear with the name configured in the macro **RSI_BLE_APP_SIMPLE_CHAT** (Ex: "**WLAN_BLE_SIMPLE_CHAT**") or sometimes observed as an internal name "**SimpleBLEPeripheral**".

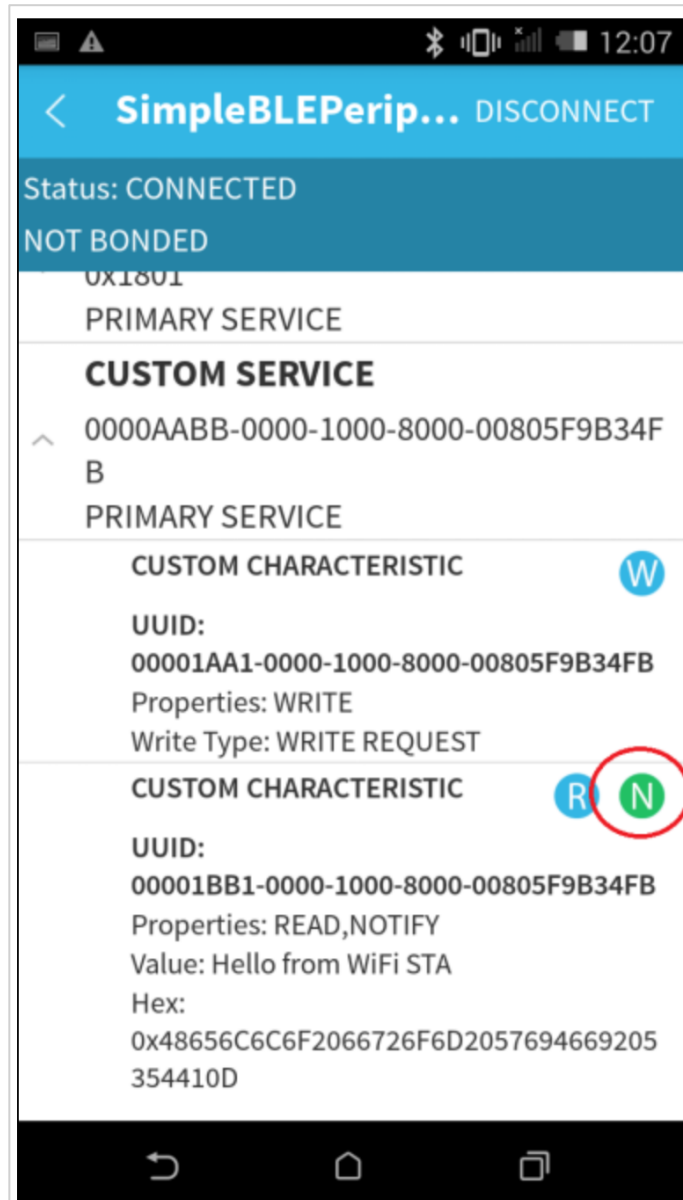


8. Initiate the BLE connection from the App to the Silabs device.

9. After a successful connection, the BLE scanner displays the supported services of the Silabs device.

10. Select the attribute service which is added **RSI_BLE_NEW_SERVICE_UUID**

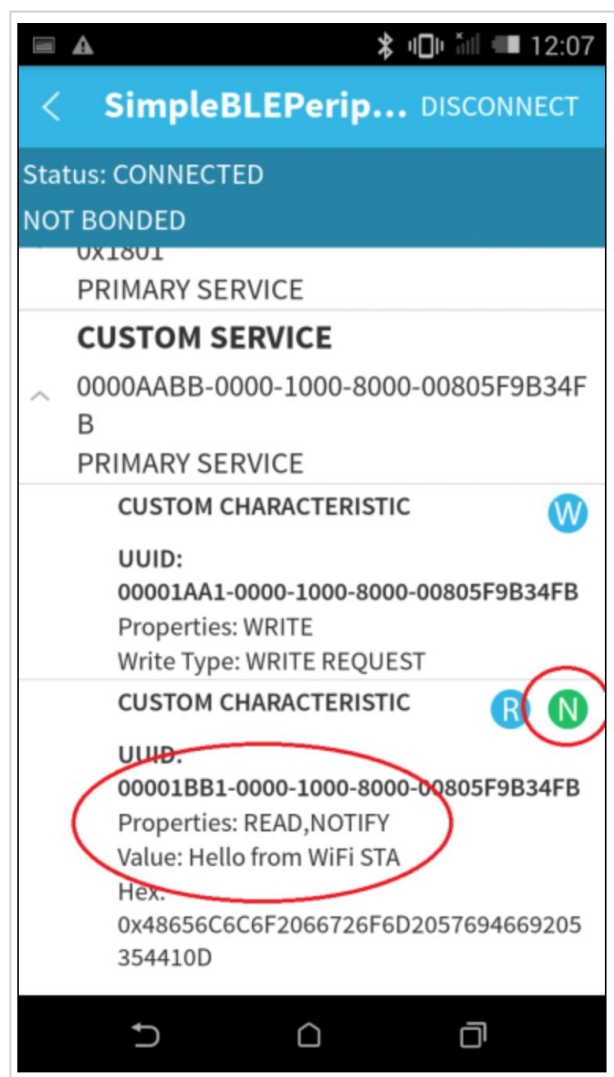
(Ex: 0xAABB) and enable Notification for attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID (Ex: 0x1BB1)** to receive data sent by Wi-Fi STA.



11. Now from the SSL server (windows PC1), send a message (Ex: "Hello from WiFi STA") to the Silabs device. Silabs device forwards the received message from the SSL server to a remote BTLE device which is connected to the Silabs BTLE device over BTLE protocol. Users can observe the message notification on attribute UUID **RSI_BLE_ATTRIBUTE_2_UUID (Ex: 0x1BB1)** in the BTLE scanner app.

```

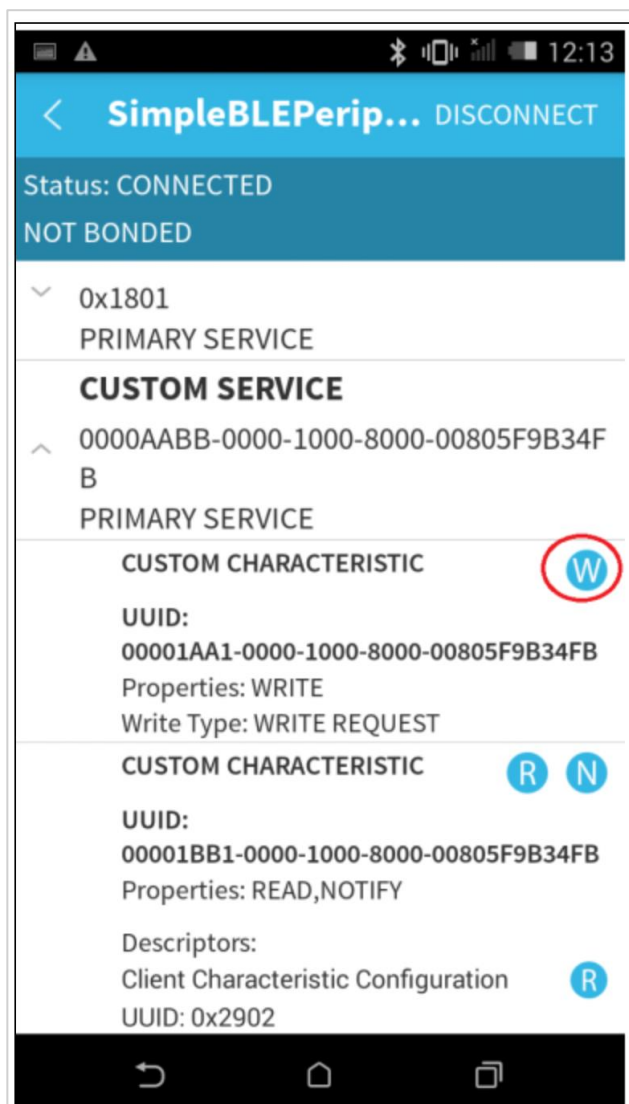
C:\Users\CPU-248\Desktop\Openssl\bin>openssl.exe s_server -accept 5001 -cert server-cert.pem -key server-key.pem -tls1
Loading 'screen' into random state - done
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
bad gethostbyaddr
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECagMBBAIANQQghXA7mzU5ua9X1hKn3o/IK2CfcntUGOpUBf+0cjUJ1NwE
MB1940ijrLu4iwlcFL/c+/dOT8Sis46ze6F0sU0rdUubm0BNkQqSOHFnSlviKpu0
26EGAgRXDk02ogQCAhwgpAYEBAAAA=
-----END SSL SESSION PARAMETERS-----
Shared cipher: AES256-SHA:AES128-SHA
Cipher is AES256-SHA
Hello from WiFi STA
  
```

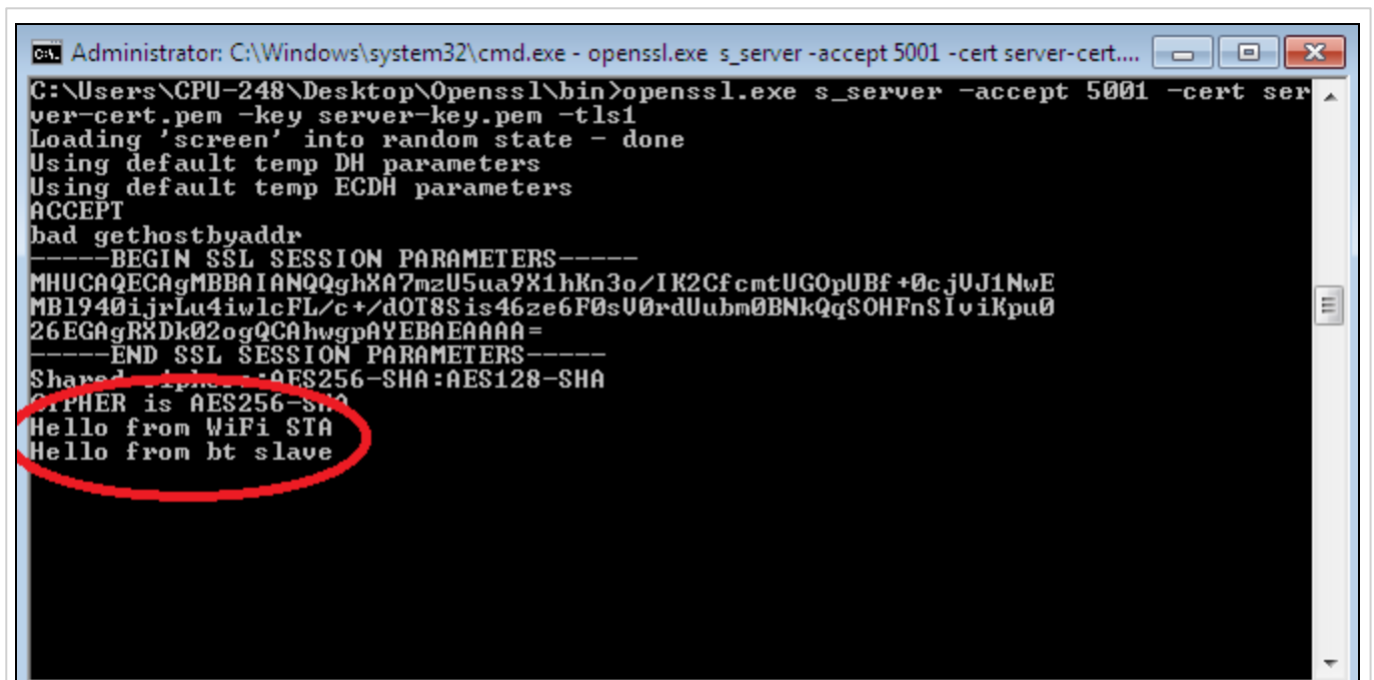
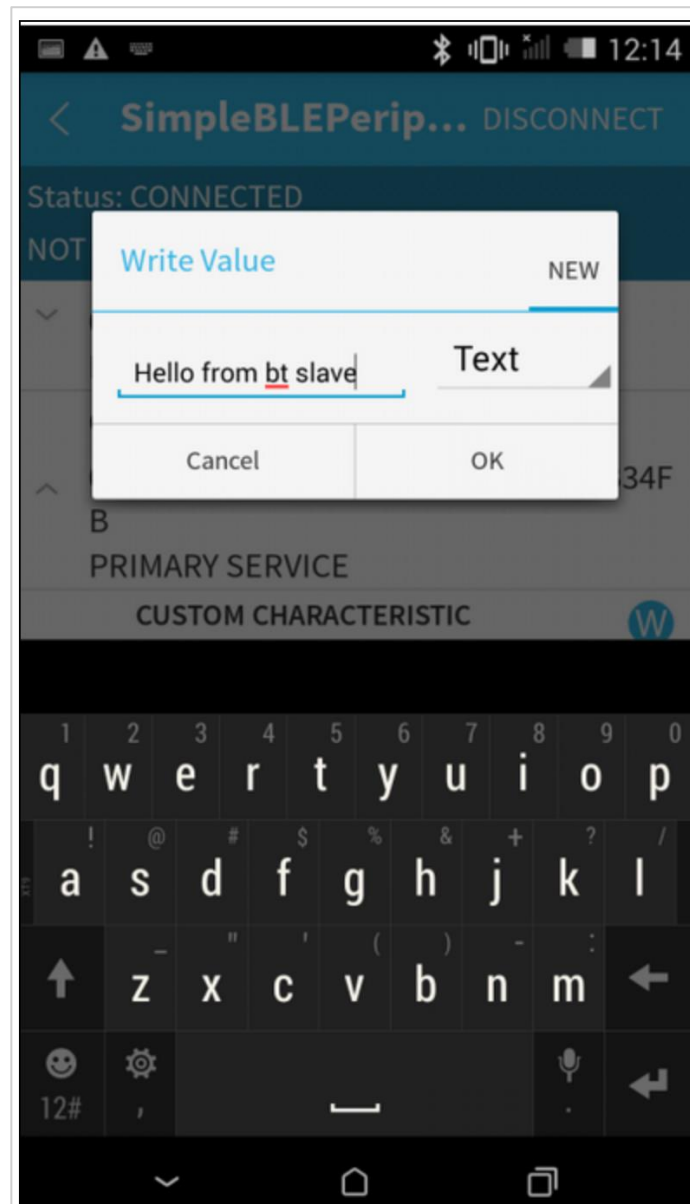


Note

rsi_wlan_app_send_to_btble() 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 smartphone BLE scanner app) using attribute **RSI_BLE_ATTRIBUTE_1_UUID** (Ex: 0x1AA1) to Silabs device. Silabs device forwards the received message from BTLE remote device to the SSL server over Wi-Fi protocol. Users can observe the message on the UDP socket application.





Note

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

3.3.4 Example4: 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
- Connect from Smartphone

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 Silabs Wireless SAPI framework provides necessary HAL APIs to enable a variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with KEIL IDE
- Windows PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silabs module connected through SPI with STM32 board
- Wireless Access point
- BTLE supported smartphones

Note

Install the Redpine Connect application in Android smartphones.

Find the Redpine_Connect_v1.1.apk file from this path: RS9116.NB0.WC.GENR.OSI.x.x.xx\utils\.

Block Diagram

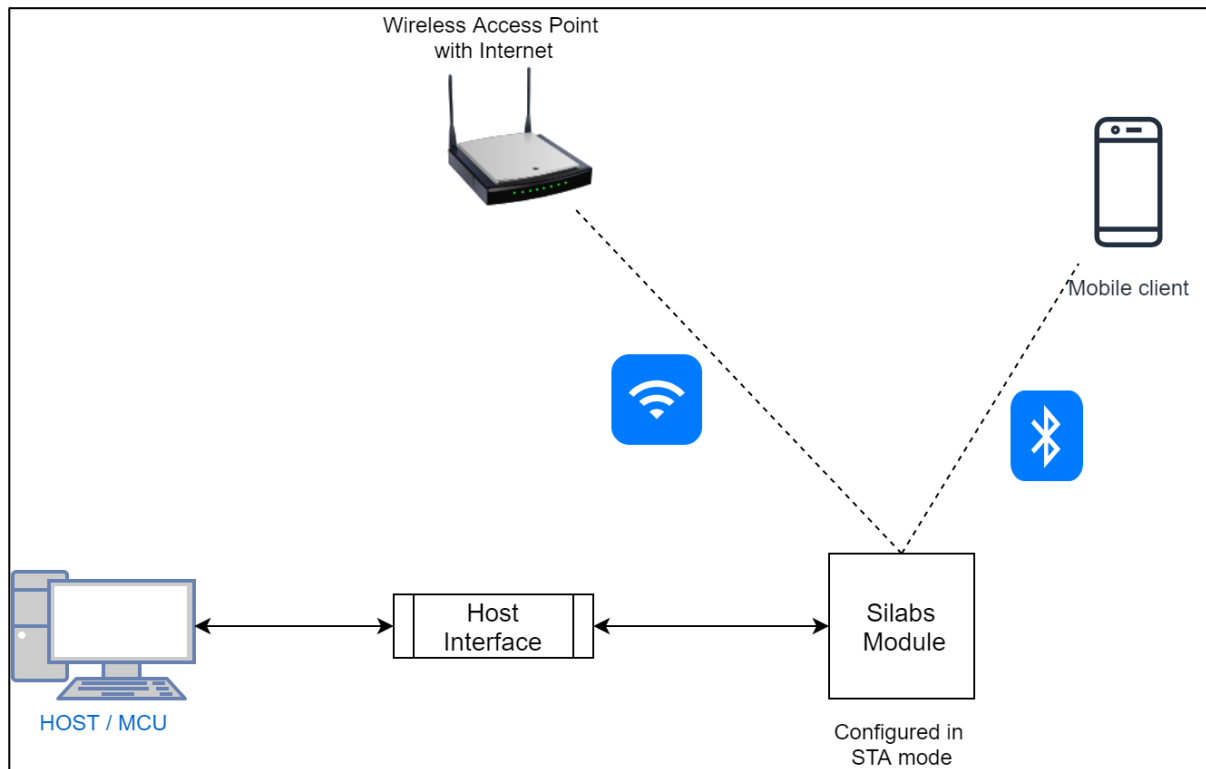
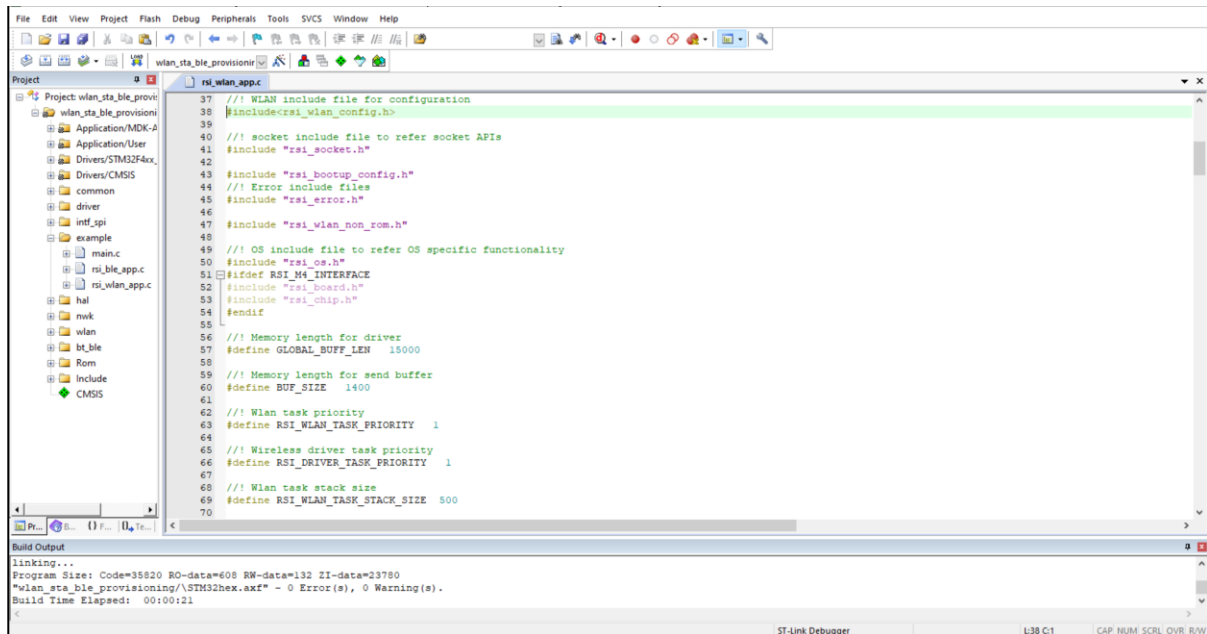


Figure 14: WLAN Station BLE Provisioning Setup Diagram

Configuring the Application Configuration and Steps for Execution

Configuring the WLAN task

1. Open Project `RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\wlan_sta_ble_provisioning` and then double click on `wlan_sta_ble_provisioning` (uVision5 Project). Then open `RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan_ble\wlan_station_ble_provisioning\rssi_wlan_app.c` file and update/modify following macros:



To configure the IP address

Memory length for driver

```
#define GLOBAL_BUFF_LEN 15000
```

Memory length for the send buffer

```
#define BUF_SIZE 1400
```

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 the 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 **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\examples\wlan_ble\wlan_station_ble_provisioning\rsi_wlan_config.h** file and update/modify the following macros:

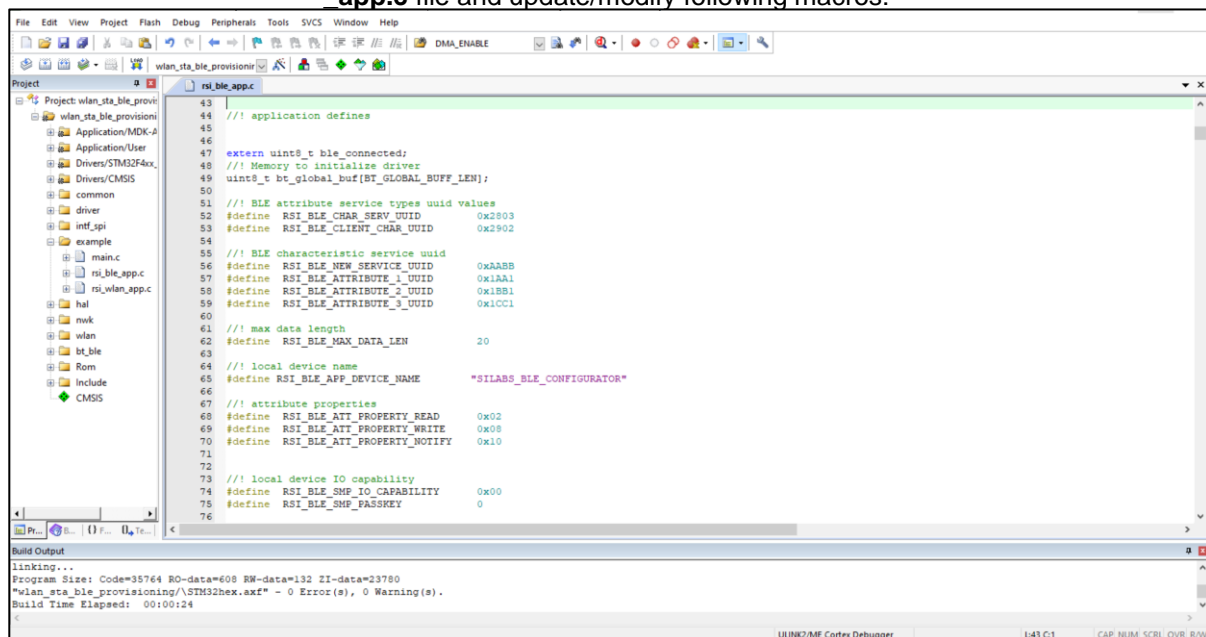
```
#define CONCURRENT_MODE RSI_DISABLE
#define RSI_FEATURE_BIT_MAP FEAT_SECURITY_OPEN
#define RSI_TCP_IP_BYPASS RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP (TCP_IP_FEAT_DHCPV4_CLIENT |
TCP_IP_FEAT_SSL | TCP_IP_FEAT_DNS_CLIENT | TCP_IP_FEAT_EXTENSION_VALID)
#define RSI_CUSTOM_FEATURE_BIT_MAP FEAT_CUSTOM_FEAT_EXTENSION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP EXT_FEAT_384K_MODE
#define RSI_EXT_TCPIP_FEATURE_BITMAP 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.

Configuring the BLE Application:

Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan_ble\wlan_station_ble_provisioning\rsi_ble_app.c** file and update/modify following macros:



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_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_ATTRIBUTE_3_UUID refers to the attribute type of the third attribute under this service (**RSI_BLE_NEW_SERVICE_UUID**).

```
#define RSI_BLE_ATTRIBUTE_3_UUID 0x1CC1
```

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 to the name of the Silabs device to appear during scanning by remote devices.

```
#define RSI_BLE_APP_DEVICE_NAME "REDPINE_BLE_CONFIGURATOR"
```

The following are the **non-configurable** macros in the application.

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.

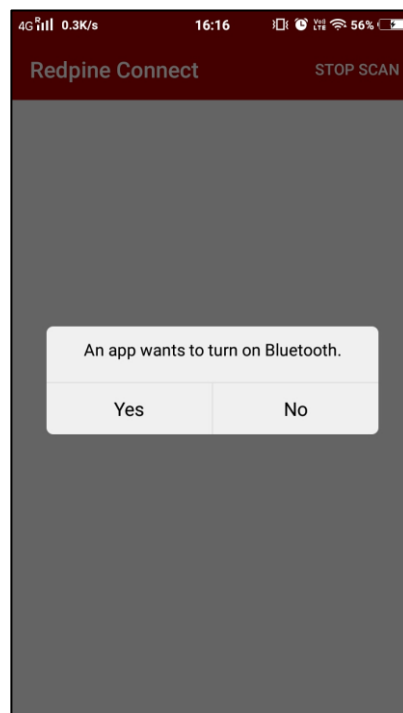
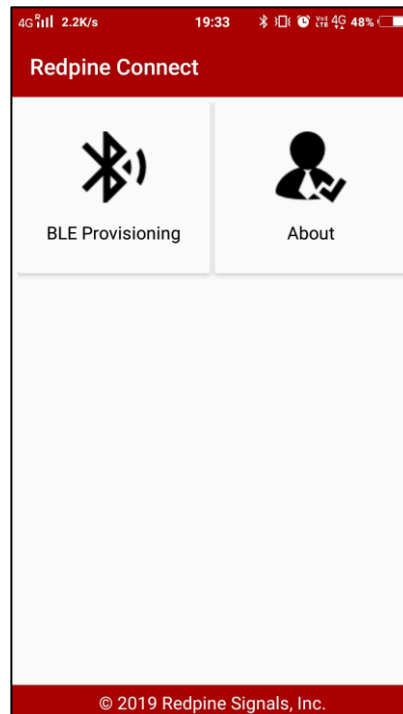
```
#define RSI_BLE_ATT_PROPERTY_NOTIFY 0x10
```

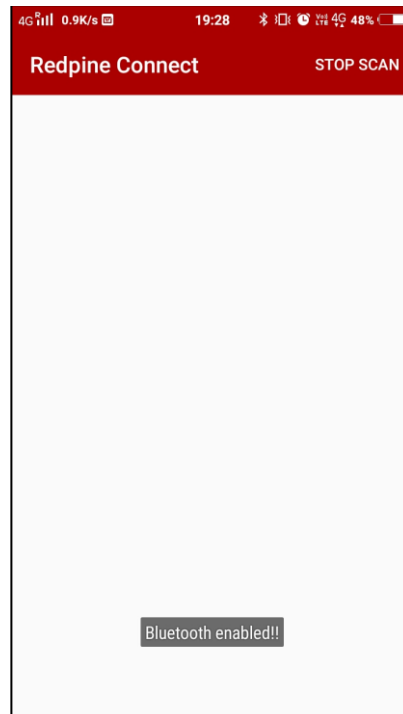
BT_GLOBAL_BUFF_LEN refers to the number of bytes required by the application and the driver.

```
#define BT_GLOBAL_BUFF_LEN 15000
```

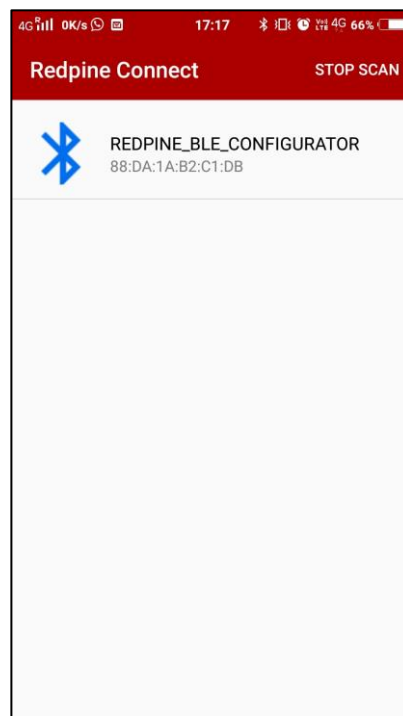
Executing the Application

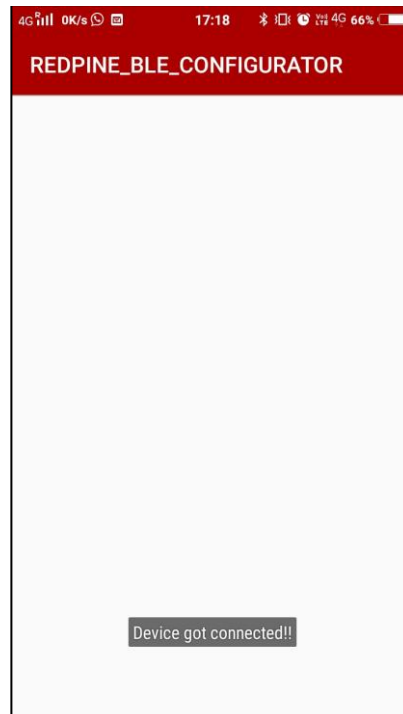
1. Connect the Silabs module through SPI to the STM32 board with the provided SPI header.
2. After executing the program, Silabs device is in Advertising state
3. Open a Redpine connect App in the Smartphone.
4. Click on BLE Provisioning, after a click on that "An app wants to turn on Bluetooth" type of pop-up display click on "Yes" and then do the Scan.



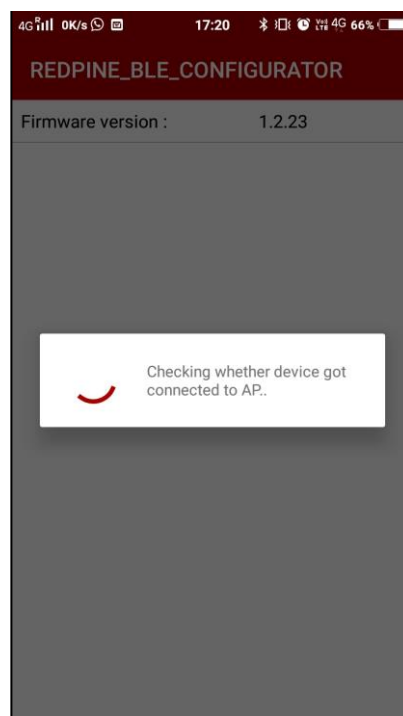


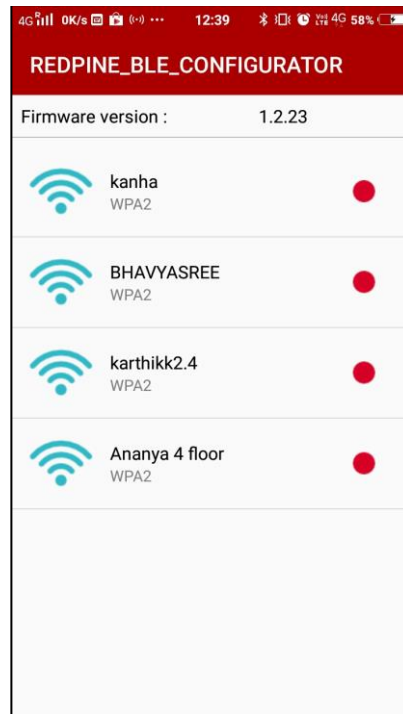
4. In the App, the Silabs module device will appear with the name configured in the macro **REDPINE_BLE_CONFIGURATOR**.
5. Initiate the BLE connection from the App to the Silabs device.



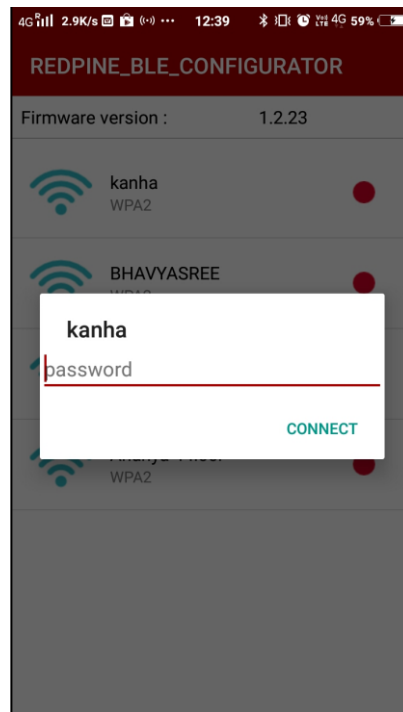


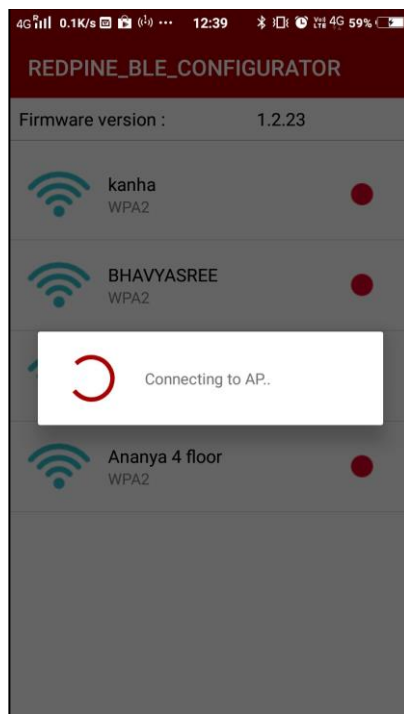
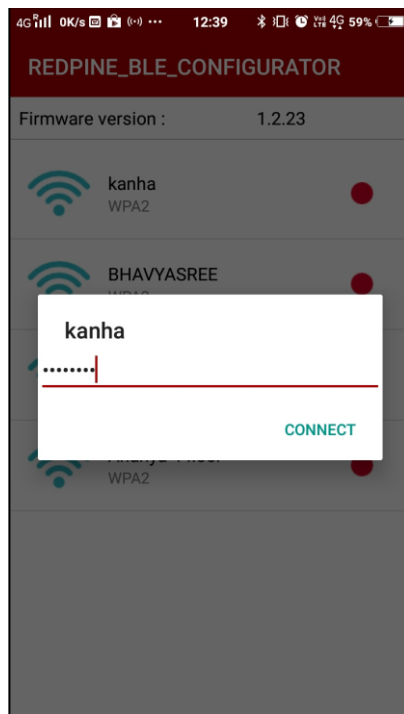
6. Once the device gets connected, a list of available Access Points to get displayed on the screen.

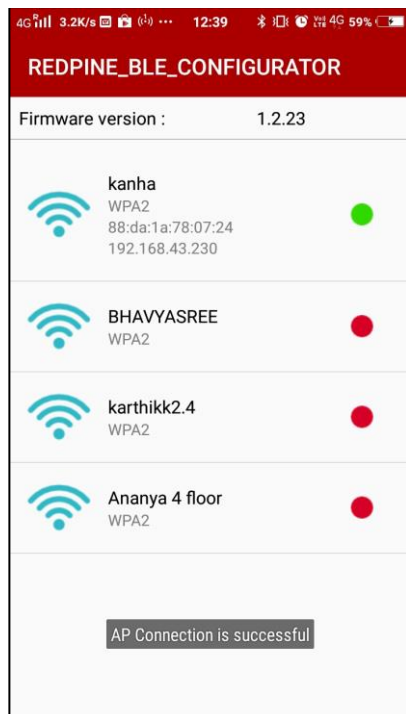




7. Connect to any Access Point.







3.3.5 Example5: WLAN Standby Associated Power Save

Overview

The application demonstrates the process of configuring the device in power save profile mode 2 after successful connection with the Access point in station mode and provides the steps to send UDP data from the 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:

- Enable Silabs device in station mode.
- Connect the Silabs device station to the Access point.
- Configure device in Power Save profile mode 2.

If needed, data transfer is possible in power save mode 2 in the following steps:

- Open the UDP client socket in the Silabs device.
- Send UDP data from the Silabs device to the remote peer.
- Analyze power save profile while it is in an 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 a variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with Host interface(UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silabs EVK or module
- Wireless Access point
- Windows PC2 with UDP server application (iperf)
- Agilent power analyzer

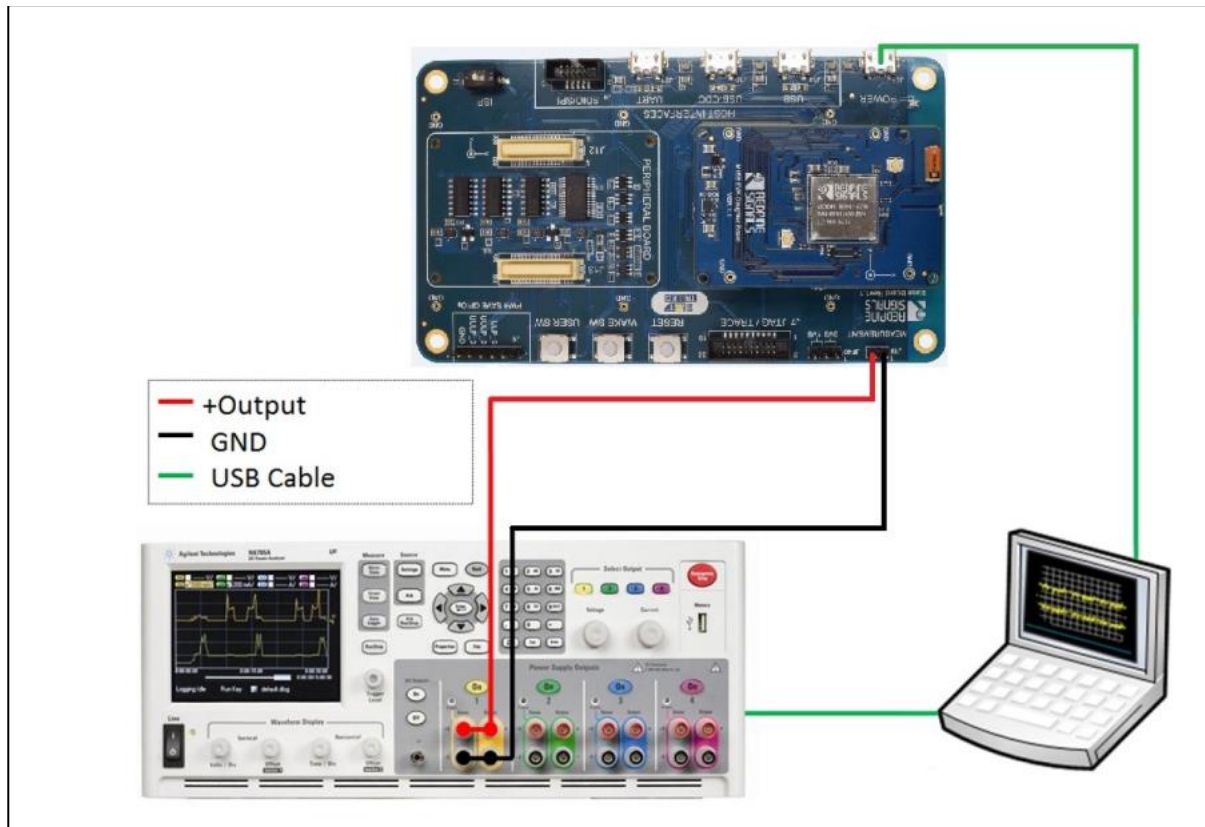


Figure 15: Setup Diagram

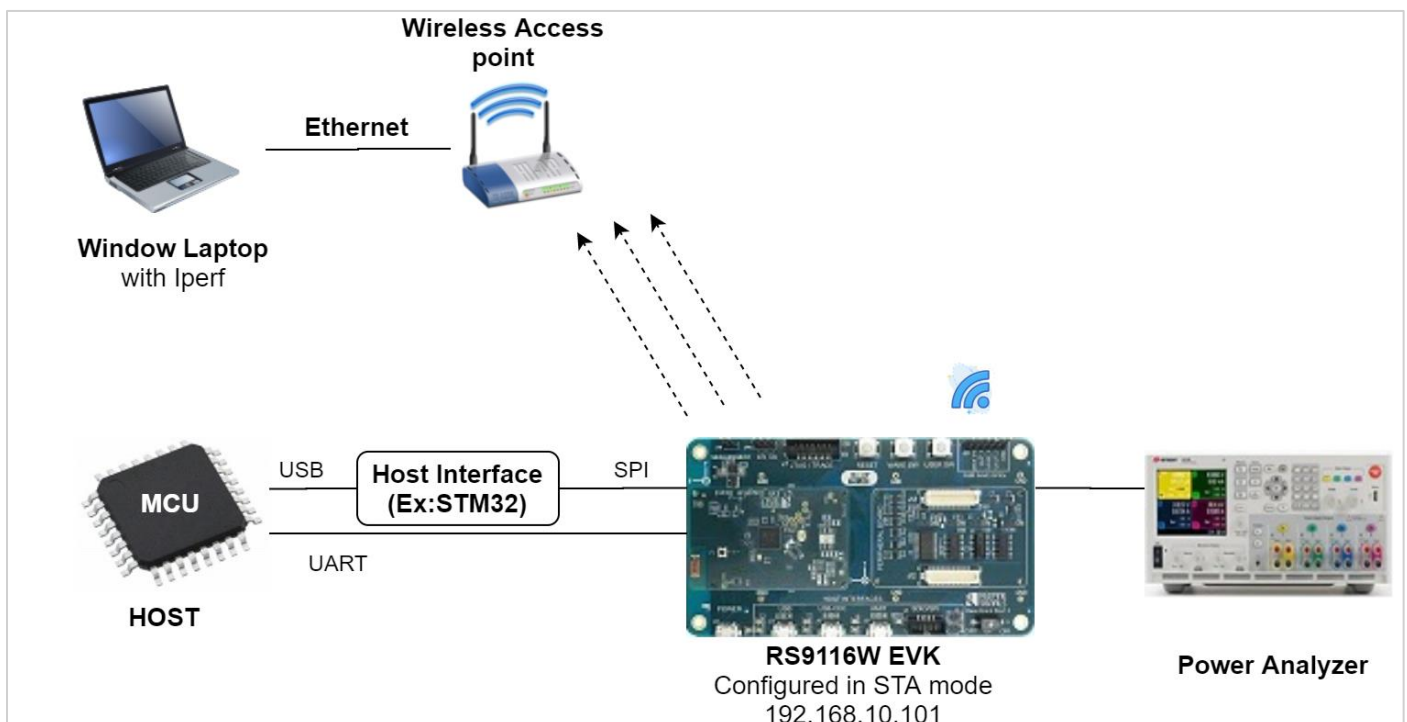


Figure 16: Setup Diagram of Standby Associated Power Save Example

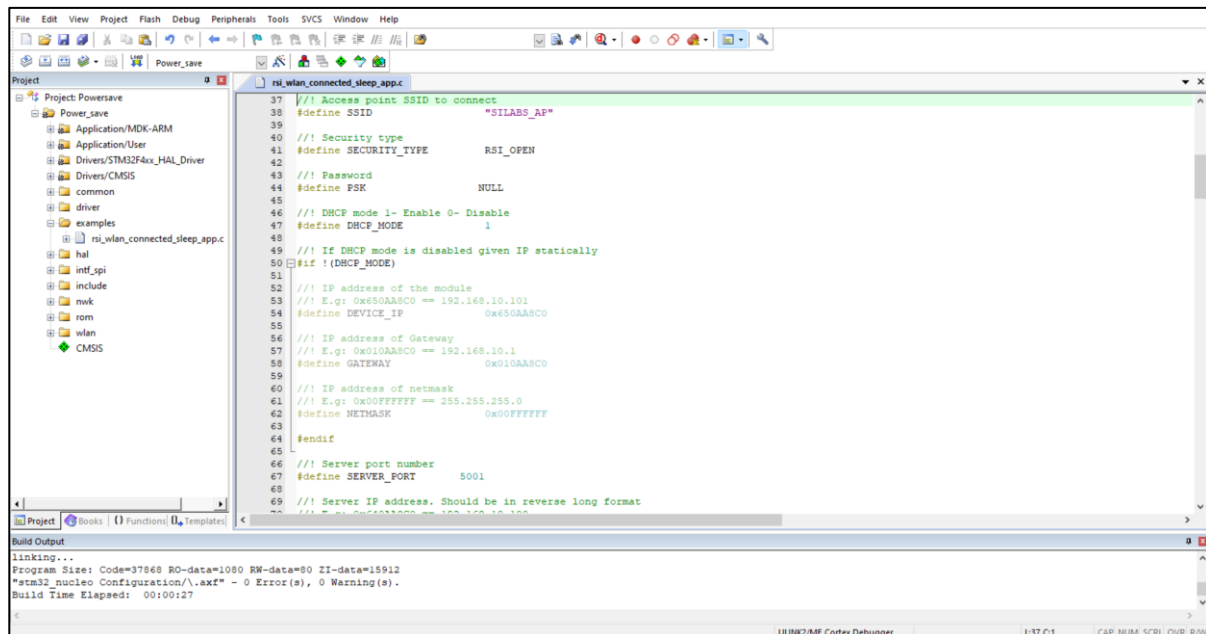
Configuration and Steps for Execution

Configuring the Application

1. Open Project **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPIPower_save** and then double click on **Powersave (uVision5 Project)**.

Then open

RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\power_save_standby_associated\rsi_wlan_connected_sleep_app.c file and update/modify the following macros:



SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

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 the UDP server socket. The IP address should be in a 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 the device to the 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, which is whether through DHCP or STATIC.

```
#define DHCP_MODE 1
```

Note

If the user wants to configure the 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 the STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following DEVICE_IP, GATEWAY and NETMASK macros.

The IP address which is to be configured to the device in STA mode should be in a 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
```

The 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
```

The 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
```

/// Power Save Profile Mode

In this application, the 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 a message-based handshake.

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_MODE RSI_SLEEP_MODE_2
#define PSP_TYPE RSI_MAX_PSP
```

2. Open

RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\power_save_standby_associated\rsi_wlan_config.h file and update/modify the following macros:

```
#define CONCURRENT_MODE RSI_DISABLE
#define RSI_FEATURE_BIT_MAP (FEAT_SECURITY_OPEN | FEAT_AGGREGATION |
FEAT_ULP_GPIO_BASED_HANDSHAKE)
#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_EXTENSION_VALID
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP EXT_FEAT_LOW_POWER_MODE |
EXT_FEAT_BYPASS_CLK_ON_UULP_GPIO_3
```

Note: The above macro is for silicon version 1.4, if the silicon version is 1.3, then define the macro as:

```
#define RSI_EXT_CUSTOM_FEATURE_BIT_MAP (EXT_FEAT_LOW_POWER_MODE |
EXT_FEAT_XTAL_CLK_ENABLE)
#define RSI_EXT_TCPIP_FEATURE_BITMAP CONFIG_FEAT_EXTENSION_VALID
#define RSI_CONFIG_FEATURE_BITMAP RSI_FEAT_SLEEP_GPIO_SEL_BITMAP
#define RSI_BAND RSI_BAND_2P4GHZ
#define RSI_HAND_SHAKE_TYPE GPIO_BASED
#define RSI_SELECT_LP_OR_ULP_MODE RSI_ULP_WITH_RAM_RET

#define PLL_MODE 0
#define RF_TYPE 1
#define WIRELESS_MODE 0
#define ENABLE_PPP 0
#define AFE_TYPE 1
#define FEATURE_ENABLES (RSI_FEAT_FRAME_PREAMBLE_DUTY_CYCLE |
RSI_FEAT_FRAME_LP_CHAIN | RSI_FEAT_FRAME_IN_PACKET_DUTY_CYCLE)
```

3.

Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\power_save_standby_associated\rsi_wlan_config.h** file and update/modify the following macros for **WMM_PS** :

```

//! set wmm enable or disable
#define RSI_WMM_PS_ENABLE           RSI_ENABLE
//! 0 -TX_BASED 1- PERIDIC
#define RSi_WMM_PS_TYPE             0
//! set wmm wake up interval
#define RSI_WMM_PS_WAKE_INTERVAL   20
//! set wmm UAPSD bitmap
#define RSI_WMM_PS_UAPSD_BITMAP    15

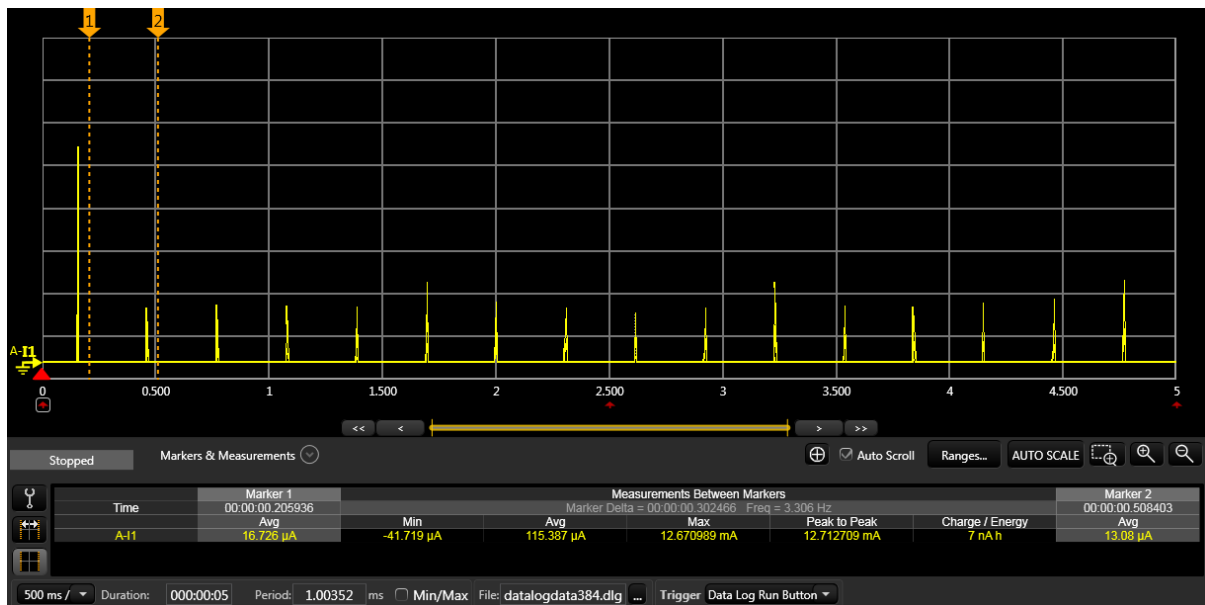
```

Note

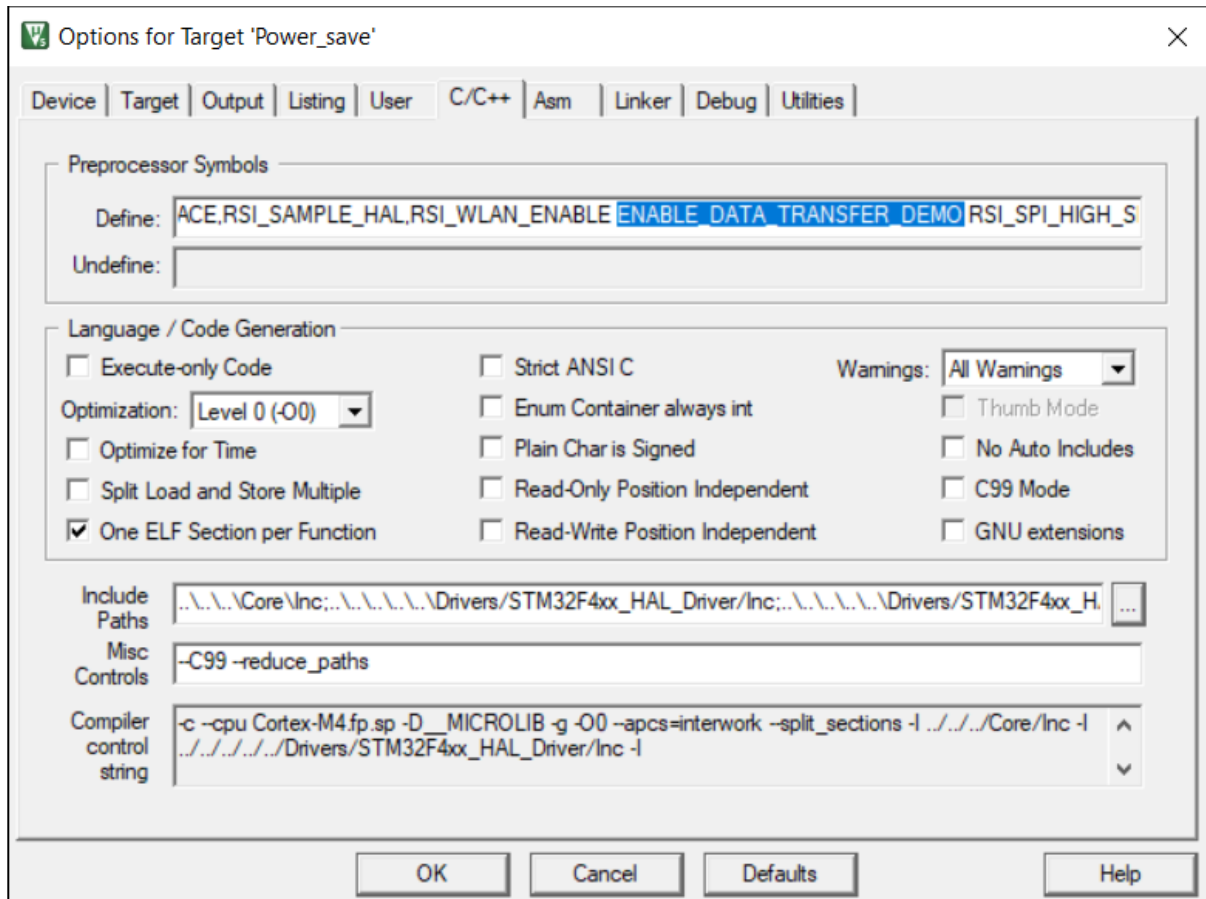
1. psp_type is only valid in psp_mode 1 and 2.
2. psp_type UAPSD is applicable only if RSI_WMM_PS_ENABLE is enabled in rsi_wlan_config.h file

Executing the Application

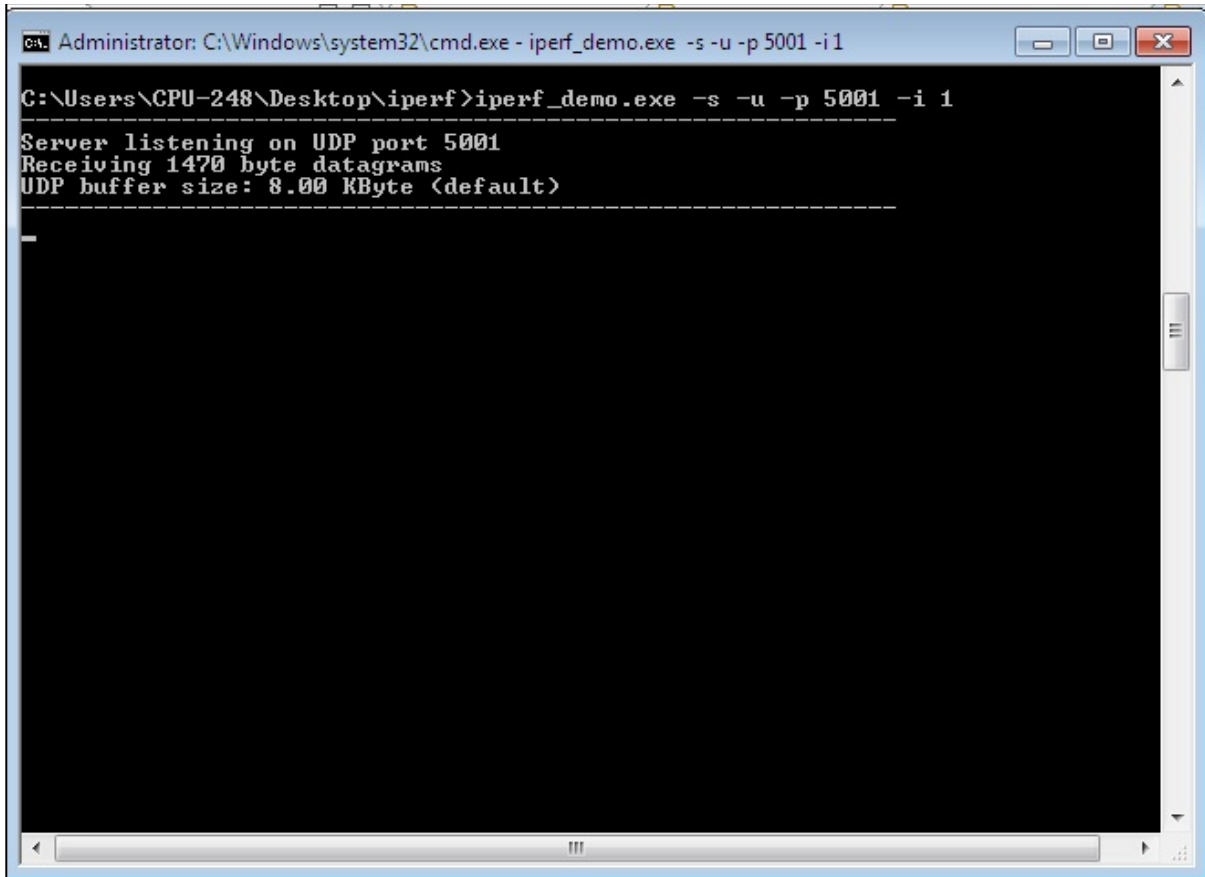
1. Switch on the power meter, go to settings and select measurement as the Current Measure.
2. Connect power probes of the power meter to the Silabs device J19 pins.
3. Connect the Silabs module through SPI to the STM32 board with the provided SPI header.
4. Give power to the Silabs device on the power interface.
5. Update Access point settings with DTIM count as 1/3/10 and beacon interval as 100.
6. Compile and run the application on PC/Laptop.
7. After a successful connection with the access point then module enters into a connected sleep mode.
8. Note down power measurement as shown below for DTIM count 1,3,10.
9. For example, below is the power measurement capture for DTIM 3.



10. In case, if you want to do data transfer in the connected sleep mode, enable the Macro **ENABLE_DATA_TRANSFER_DEMO** in project settings as below:



11. Open UDP server application using iperf in Windows PC2 which is connected to the access point through LAN.
12. Download the iperf application from the link below:
<https://iperf.fr/iperf-download.php#windows>
 And give this command: **iperf_demo.exe -s -u -p <SERVER_PORT> -i 1**



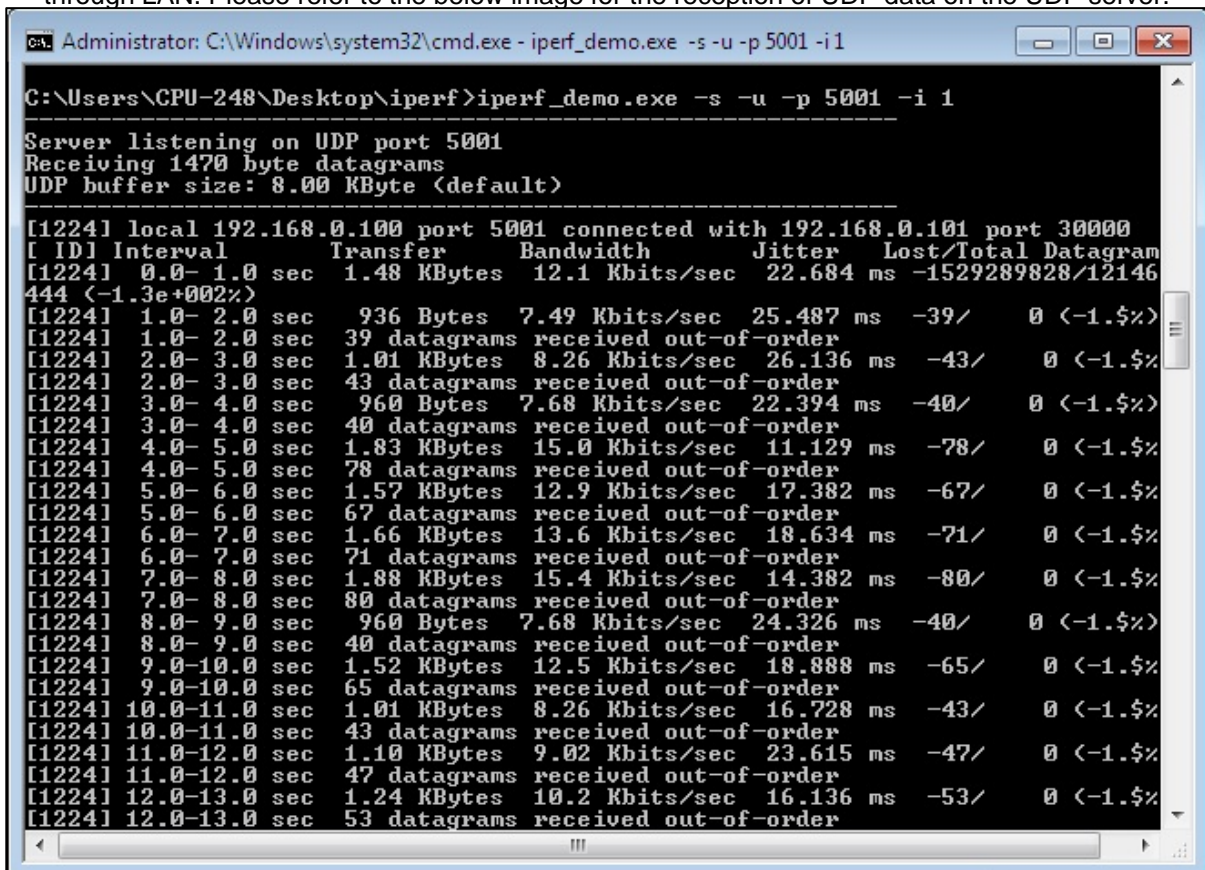
```
Administrator: C:\Windows\system32\cmd.exe - iperf_demo.exe -s -u -p 5001 -i 1

C:\Users\CPU-248\Desktop\iperf>iperf_demo.exe -s -u -p 5001 -i 1

Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)

-----
```

13. After the program gets executed, Silabs Device will scan and connect to the access point and get an IP address.
14. After a successful connection, the device goes into configured power save mode 2 and sends a configured number of (**NUMBER_OF_PACKETS**) UDP packets to a remote peer which is connected to the access point through LAN. Please refer to the below image for the reception of UDP data on the UDP server.



```
Administrator: C:\Windows\system32\cmd.exe - iperf_demo.exe -s -u -p 5001 -i 1

C:\Users\CPU-248\Desktop\iperf>iperf_demo.exe -s -u -p 5001 -i 1

Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)

-----
[1224] local 192.168.0.100 port 5001 connected with 192.168.0.101 port 30000
[ ID] Interval      Transfer    Bandwidth  Jitter    Lost/Totl  Datagram
[1224] 0.0- 1.0 sec    1.48 KBytes 12.1 Kbits/sec 22.684 ms -1529289828/12146
444 <-1.3e+002%>
[1224] 1.0- 2.0 sec    936 Bytes  7.49 Kbits/sec 25.487 ms -39/    0 <-1.5%>
[1224] 1.0- 2.0 sec    39 datagrams received out-of-order
[1224] 2.0- 3.0 sec    1.01 KBytes 8.26 Kbits/sec 26.136 ms -43/    0 <-1.5%>
[1224] 2.0- 3.0 sec    43 datagrams received out-of-order
[1224] 3.0- 4.0 sec    960 Bytes  7.68 Kbits/sec 22.394 ms -40/    0 <-1.5%>
[1224] 3.0- 4.0 sec    40 datagrams received out-of-order
[1224] 4.0- 5.0 sec    1.83 KBytes 15.0 Kbits/sec 11.129 ms -78/    0 <-1.5%>
[1224] 4.0- 5.0 sec    78 datagrams received out-of-order
[1224] 5.0- 6.0 sec    1.57 KBytes 12.9 Kbits/sec 17.382 ms -67/    0 <-1.5%>
[1224] 5.0- 6.0 sec    67 datagrams received out-of-order
[1224] 6.0- 7.0 sec    1.66 KBytes 13.6 Kbits/sec 18.634 ms -71/    0 <-1.5%>
[1224] 6.0- 7.0 sec    71 datagrams received out-of-order
[1224] 7.0- 8.0 sec    1.88 KBytes 15.4 Kbits/sec 14.382 ms -80/    0 <-1.5%>
[1224] 7.0- 8.0 sec    80 datagrams received out-of-order
[1224] 8.0- 9.0 sec    960 Bytes  7.68 Kbits/sec 24.326 ms -40/    0 <-1.5%>
[1224] 8.0- 9.0 sec    40 datagrams received out-of-order
[1224] 9.0-10.0 sec    1.52 KBytes 12.5 Kbits/sec 18.888 ms -65/    0 <-1.5%>
[1224] 9.0-10.0 sec    65 datagrams received out-of-order
[1224] 10.0-11.0 sec    1.01 KBytes 8.26 Kbits/sec 16.728 ms -43/    0 <-1.5%>
[1224] 10.0-11.0 sec    43 datagrams received out-of-order
[1224] 11.0-12.0 sec    1.10 KBytes 9.02 Kbits/sec 23.615 ms -47/    0 <-1.5%>
[1224] 11.0-12.0 sec    47 datagrams received out-of-order
[1224] 12.0-13.0 sec    1.24 KBytes 10.2 Kbits/sec 16.136 ms -53/    0 <-1.5%>
[1224] 12.0-13.0 sec    53 datagrams received out-of-order
```

3.3.6 Example6: Enterprise Ping Client (eap)

Overview

This application demonstrates how to configure the device in Enterprise Client, and to connect with Enterprise Secured AP, and check data traffic in Enterprise Security Mode. In this application, the device connects to Enterprise Secured AP using EAP-TLS/TTLS/PEAP/FAST method. After connecting successfully, the application pings to the AP.

EAP Overview

In wireless communications using EAP, a user requests connection to a WLAN through an AP, which 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 the AP gets from the user. The AP then sends back the proof of identity 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.

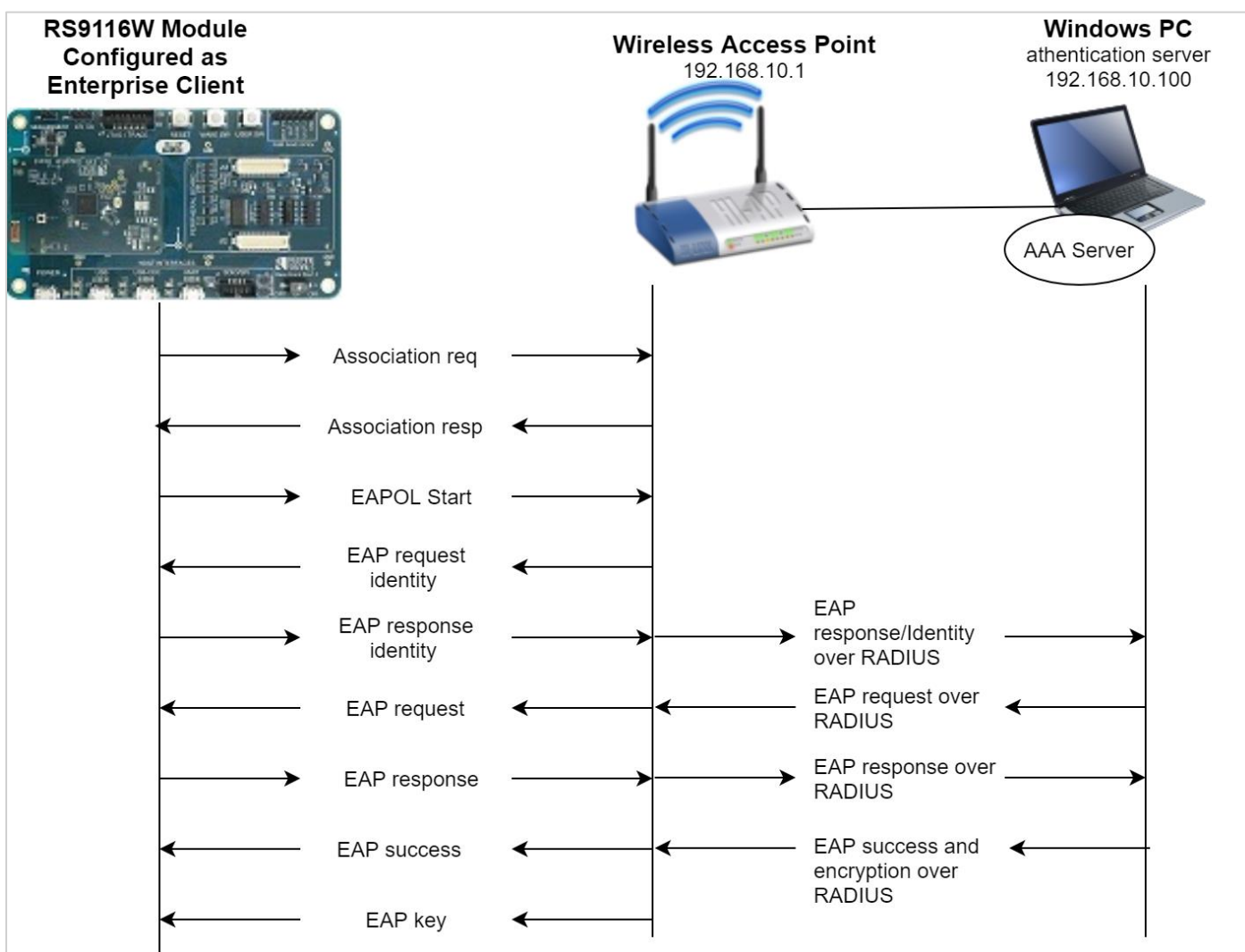


Figure 17: EAPOL-Keys Exchange

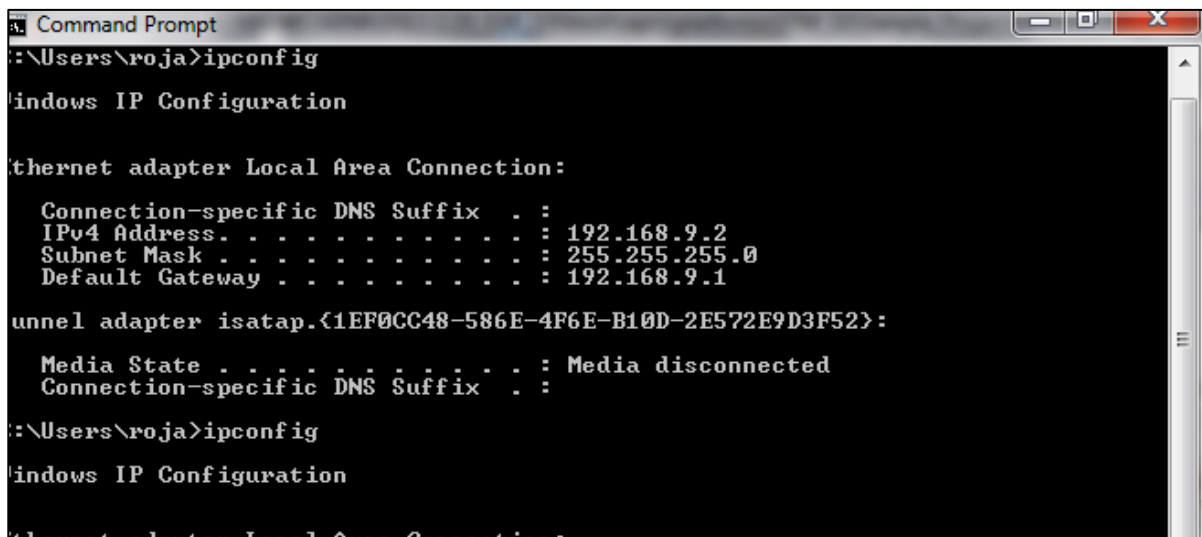
Note:

Ensure the LAN connection is removed from your PC. Remove any proxy server settings as well.

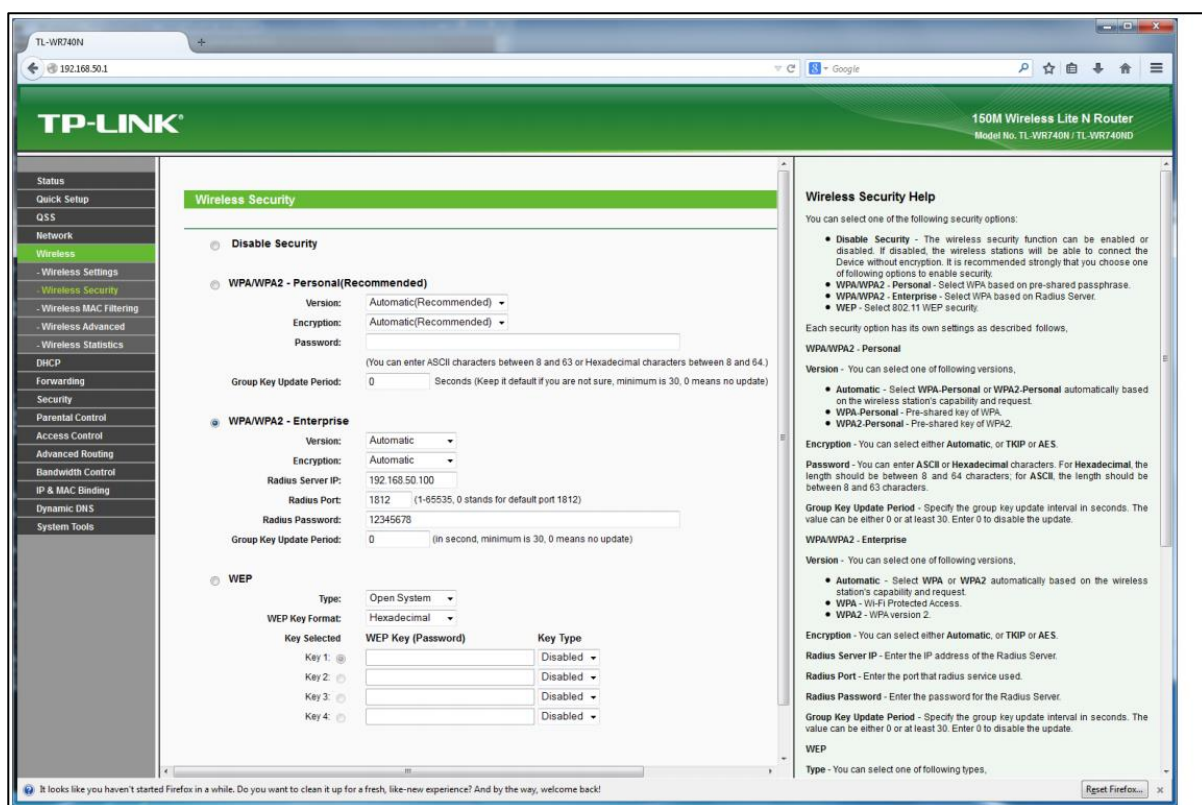
TP-link setup :

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 gives "ipconfig" command to know the IP and gateway address of the Radius server. The below image is for reference purposes.



2. Connect the Access Point to PC over Ethernet and open the Access Point page in the 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.

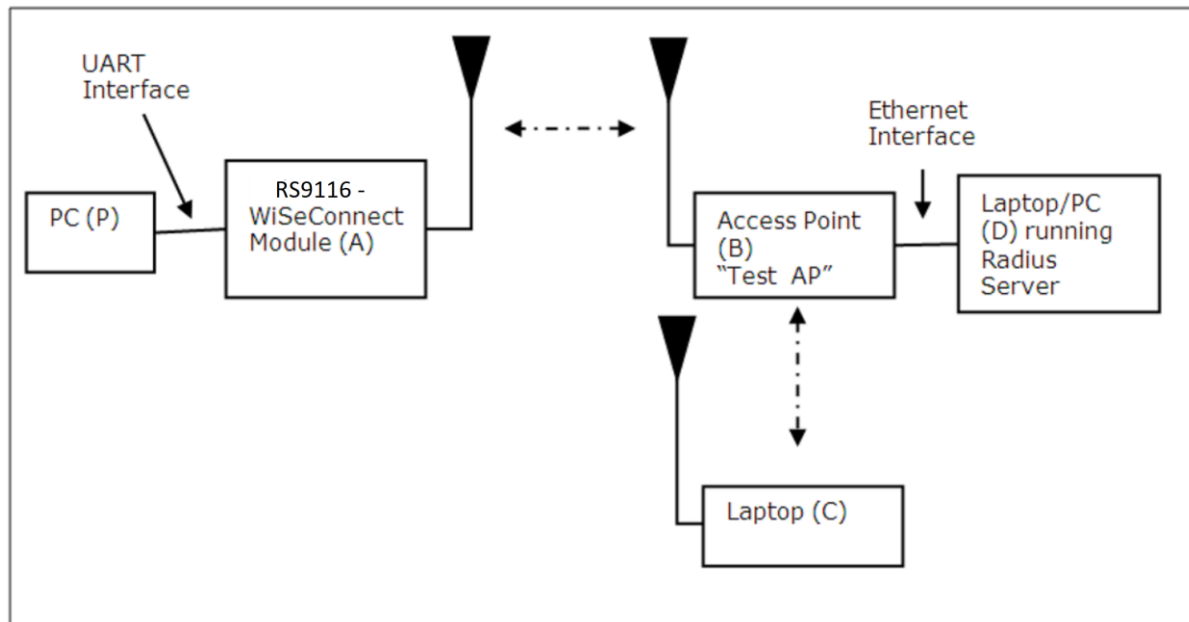


Figure 18: Setup for Wi-Fi Client in Enterprise Security Mode

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, a similar process may be followed for other OS.

1. Free Radius Server installation link:
<https://freeradius.org/>
<http://xperientech.com/download/radius-free-download.asp>

Note

The application was tested in FreeRADIUS-server-2.2.3-x86.

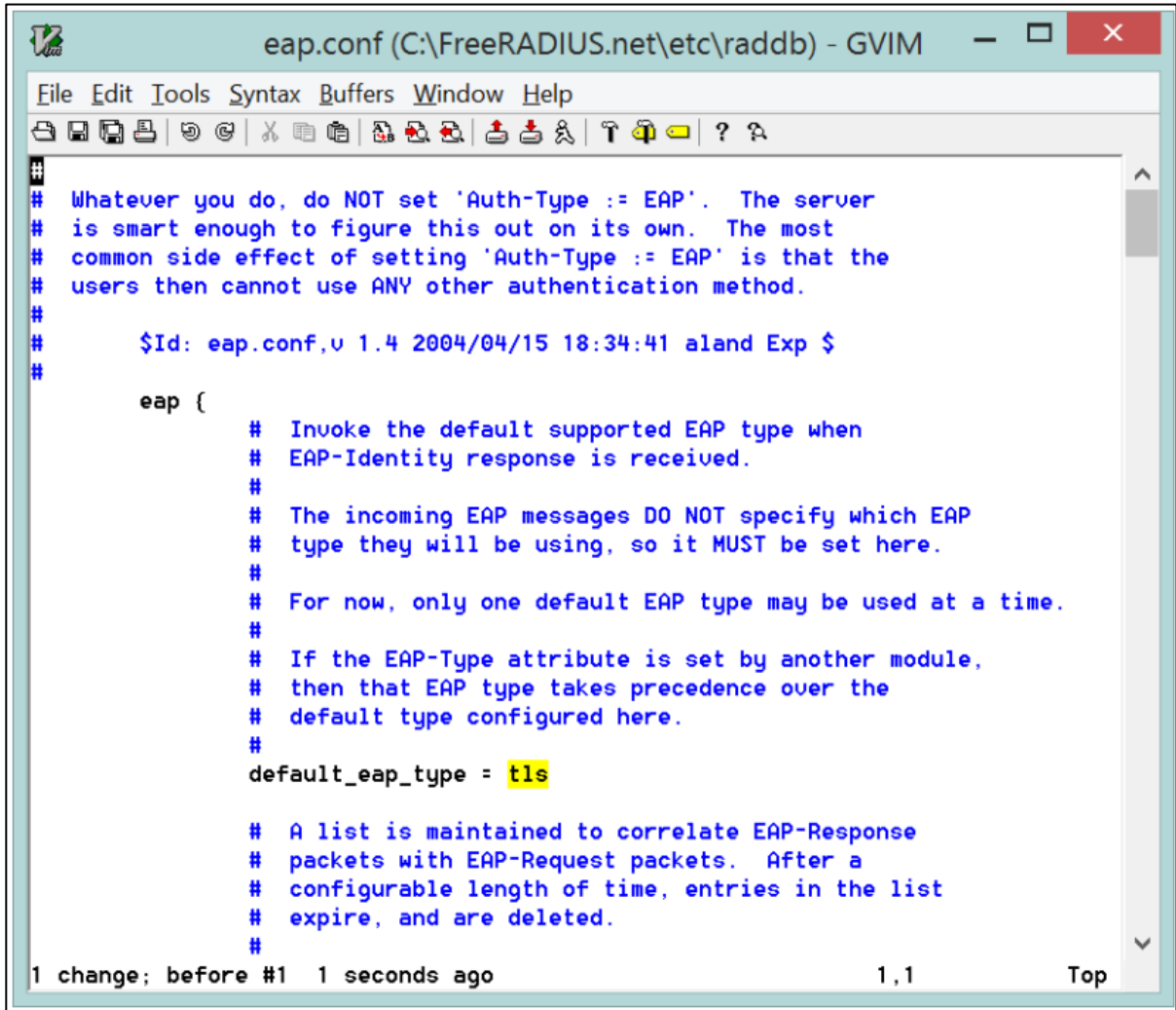
2. Once installed, go to the C:\FreeRADIUS\etc\raddb folder and make the following modifications.
3. Open the 'clients.conf' file and add the following lines at the end of the file.

```

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.



```
# Whatever you do, do NOT set 'Auth-Type := EAP'. The server
# is smart enough to figure this out on its own. The most
# common side effect of setting 'Auth-Type := EAP' is that the
# users then cannot use ANY other authentication method.
#
# $Id: eap.conf,v 1.4 2004/04/15 18:34:41 aland Exp $
#
eap {
    # Invoke the default supported EAP type when
    # EAP-Identity response is received.
    #
    # The incoming EAP messages DO NOT specify which EAP
    # type they will be using, so it MUST be set here.
    #
    # For now, only one default EAP type may be used at a time.
    #
    # If the EAP-Type attribute is set by another module,
    # then that EAP type takes precedence over the
    # default type configured here.
    #
    default_eap_type = tls

    # A list is maintained to correlate EAP-Response
    # packets with EAP-Request packets. After a
    # configurable length of time, entries in the list
    # expire, and are deleted.
    #
    1 change; before #1 1 seconds ago
```

- 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

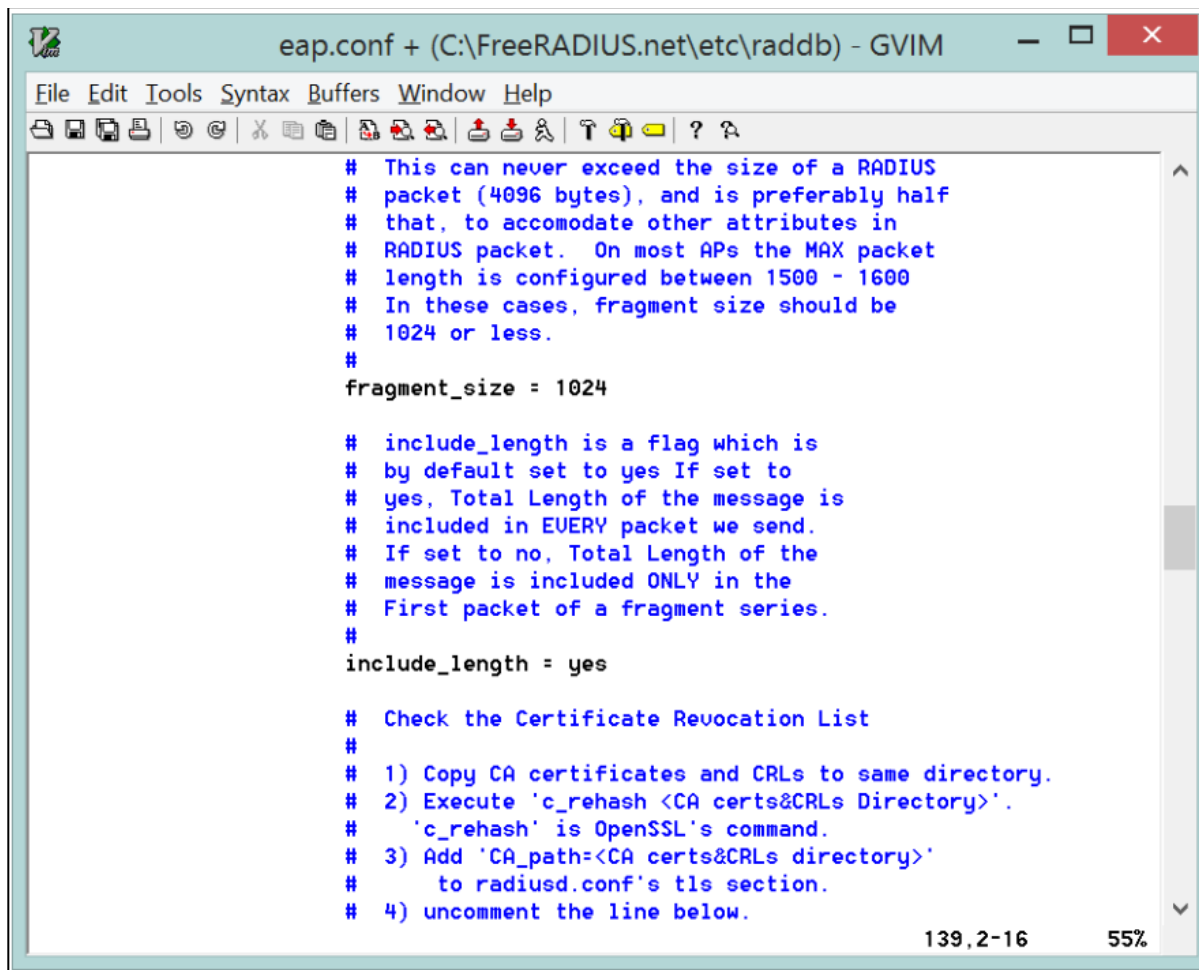
    # If Private key & Certificate are located in
    # the same file, then private_key_file &
    # certificate_file must contain the same file
    # name.
    #
    # If CA_file (below) is not used, then the
    # certificate_file below MUST include not
    # only the server certificate, but ALSO all
    # 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.



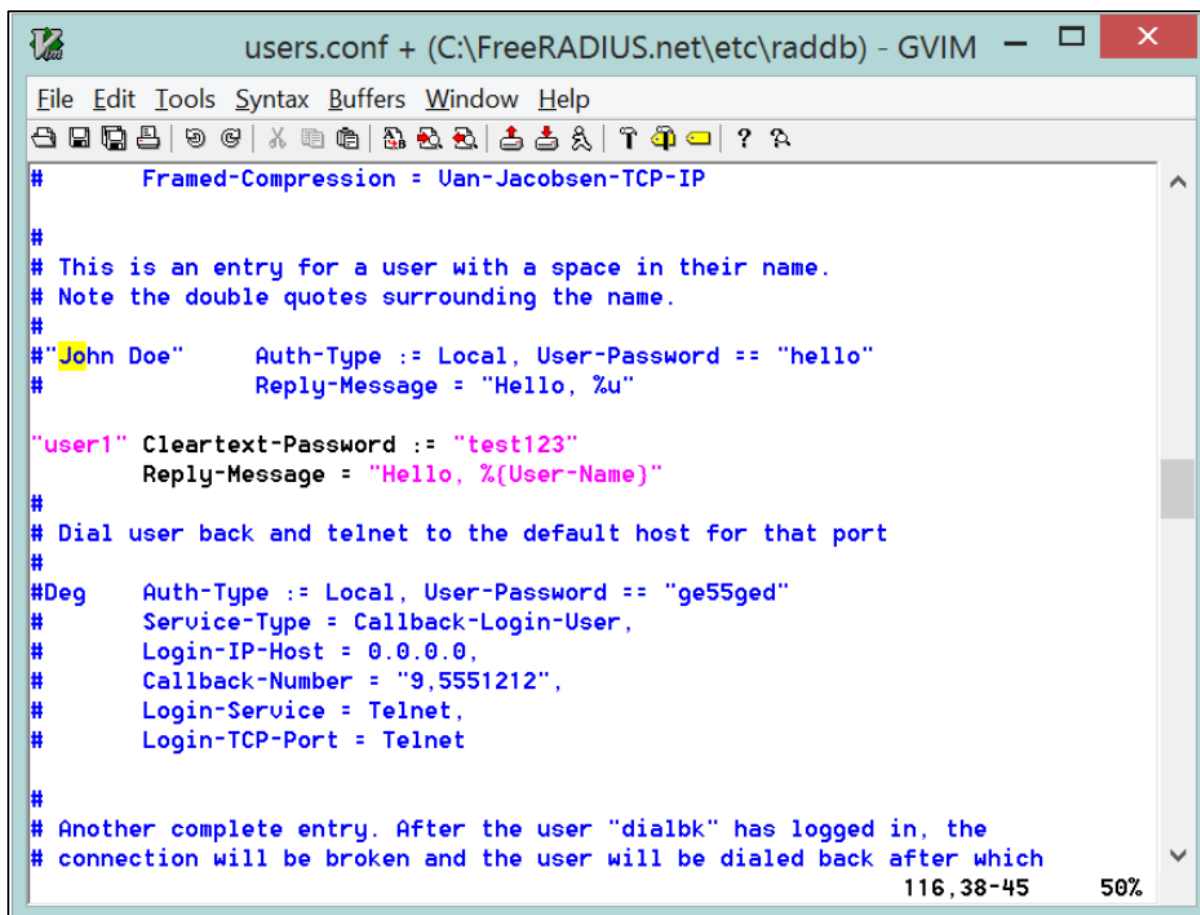
```
# This can never exceed the size of a RADIUS
# packet (4096 bytes), and is preferably half
# that, to accomodate other attributes in
# RADIUS packet.  On most APs the MAX packet
# length is configured between 1500 - 1600
# In these cases, fragment size should be
# 1024 or less.
#
fragment_size = 1024

# include_length is a flag which is
# by default set to yes If set to
# yes, Total Length of the message is
# included in EVERY packet we send.
# If set to no, Total Length of the
# message is included ONLY in the
# First packet of a fragment series.
#
include_length = yes

# Check the Certificate Revocation List
#
# 1) Copy CA certificates and CRLs to same directory.
# 2) Execute 'c_rehash <CA certs&CRLs Directory>'.
# 'c_rehash' is OpenSSL's command.
# 3) Add 'CA_path=<CA certs&CRLs directory>'
# to radiusd.conf's tls section.
# 4) uncomment the line below.
```

139,2-16 55%

- 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".



```
# Framed-Compression = Van-Jacobson-TCP-IP

#
# This is an entry for a user with a space in their name.
# Note the double quotes surrounding the name.
#
#"John Doe"      Auth-Type := Local, User-Password == "hello"
#                Reply-Message = "Hello, %u"

"user1" Cleartext-Password := "test123"
        Reply-Message = "Hello, %{User-Name}"

#
# Dial user back and telnet to the default host for that port
#
#Deg      Auth-Type := Local, User-Password == "ge55ged"
#         Service-Type = Callback-Login-User,
#         Login-IP-Host = 0.0.0.0,
#         Callback-Number = "9,5551212",
#         Login-Service = Telnet,
#         Login-TCP-Port = Telnet

#
# Another complete entry. After the user "dialbk" has logged in, the
# connection will be broken and the user will be dialed back after which
```

116,38-45 50%

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:

- Configure the device as an Enterprise client.
- Connect with Enterprise secured AP using EAP-TLS/TTLS/PEAP/FAST method.
- Establish TCP connection from connected Silabs device to TCP server opened on the remote peer.
- Send TCP data from the device to the 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 Silabs Wireless SAPI framework provides necessary HAL APIs to enable a variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with Keil IDE
- Windows PC with Host interface (UART/ USB-CDC/ SPI/ USB) in case of WiSeConnect
- Silabs module SPI connection with STM32 board
- Windows PC2 with AAA Radius Server or Free Radius server
- TP-link router module connect with CPU through a LAN cable
- Wireless Access point
- TCP server application running on Windows PC2 (This example uses iperf for windows)

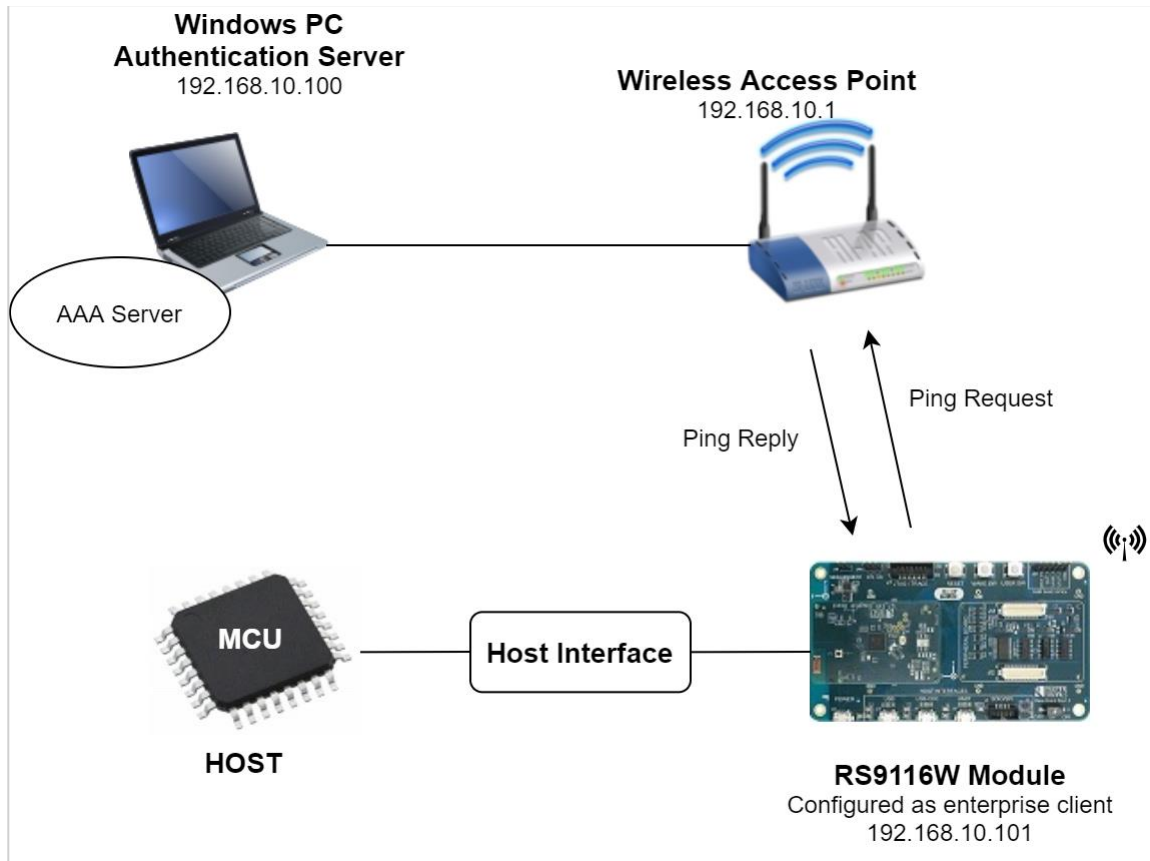
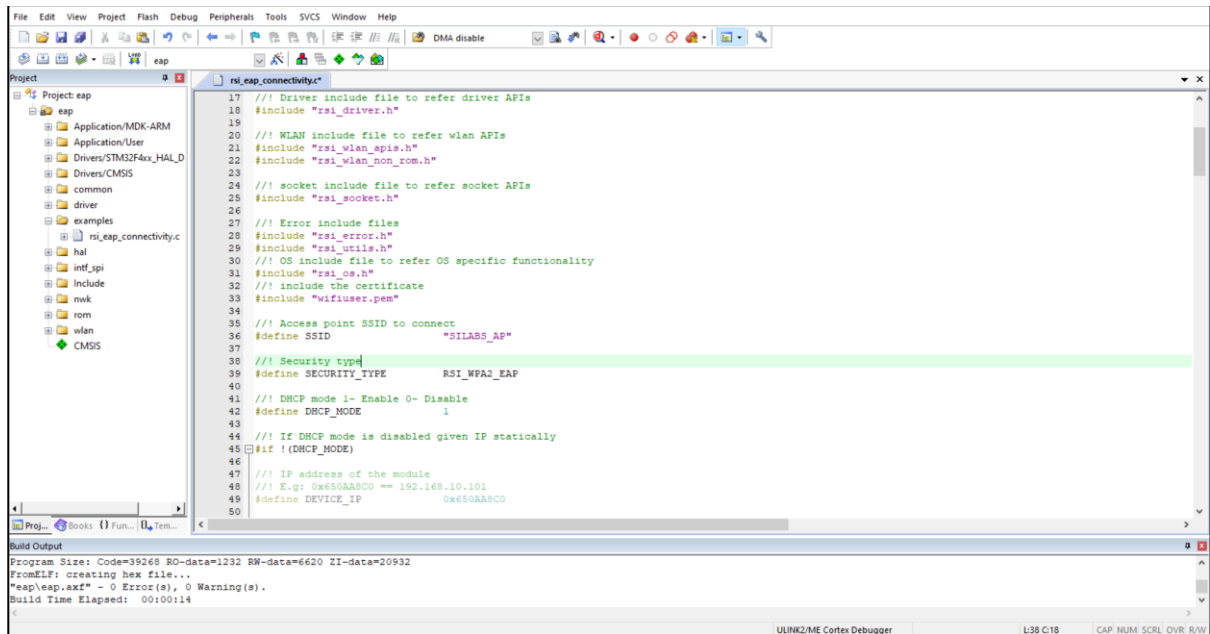


Figure 19: Setup for Wi-Fi Client in Enterprise Security Mode

Configuration and Steps for Execution

Configuring the Application

1. Open Project **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\leap** and then double click on **eap** (uVision5 Project). Then open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\examples\wlan\leap\rsi_eap_connectivity.c** file and update/modify following macros:



SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

SECURITY_TYPE refers to the type of security. In this application, STA supports WPA-EAP, WPA2-EAP securities.

The 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 to whether a certificate to load into the module or not.

```
#define LOAD_CERTIFICATE 1
```

Note

If **LOAD_CERTIFICATE** set to 1, the application will load the certificate which is included using **rsi_wlan_set_certificate** API.

By default, the application is loading the "wifiuser.pem" certificate when **LOAD_CERTIFICATE** enables. In order to load a different certificate, the user has to do the following steps:

- **rsi_wlan_set_certificate** API expects the certificate in the form of a linear array. So, convert the pem certificate into linear array form using python script provided in the release package

Ex: If the certificate is wifi-user.pem. Give the command in the following way :

```
python certificate_script.py wifi-user.pem
```

The script will generate wifiuser.pem in which one linear array named wifiuser contains the certificate.

- After the conversion of the 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.
- Once the certificate loads into the device, it will write into the device flash. So, users need not load the certificate for every boot up unless the certificate change.

So define **LOAD_CERTIFICATE** as 0, if the 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, the password is "test123"

```
#define PASSWORD                      "\"test123\""
```

DEVICE_PORT port refers TCP client port number

```
#define DEVICE_PORT                   5001
```

SERVER_PORT port refers TCP server port number

```
#define SERVER_PORT                   5001
```

Server IP address. Should be in reverse long format E.g: 0x640AA8C0 == 192.168.10.100

```
#define SERVER_IP_ADDRESS 0x640AA8C0
```

NUMBER_OF_PACKETS refers to how many numbers of pings to send from the device.

```
#define NUMBER_OF_PACKETS 1000
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN 1500
```

To configure IP address in STA mode

DHCP_MODE refers to whether the IP address configured through DHCP or STATIC in STA mode

```
#define DHCP_MODE 1
```

Note

If the user wants to configure the STA IP address through DHCP then skip configuring the following **DEVICE_IP**, **GATEWAY**, and **NETMASK** macros.

(Or)

If the user wants to configure the STA IP address through STATIC then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP**, **GATEWAY**, and **NETMASK** macros.

The IP address to be configured to the device in STA mode should be in a long format and in little-endian byte order.

Example: To configure "192.168.0.10" as IP address, update the macro **DEVICE_IP** as **0x0A00A8C0**.

```
#define DEVICE_IP 0X0A00A8C0
```

The IP address of the gateway should also be in a 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
```

The IP address of the network mask should also be in a 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 of the remote peer (AP IP address).

Example: To configure "192.168.10.1" as REMOTE_IP, update the macro REMOTE_IP as 0x010AA8C0.

```
#define REMOTE_IP                                0x010AA8C0
```

PING_SIZE refers to the size of the ping packet.

```
#define PING_SIZE                                100
```

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.

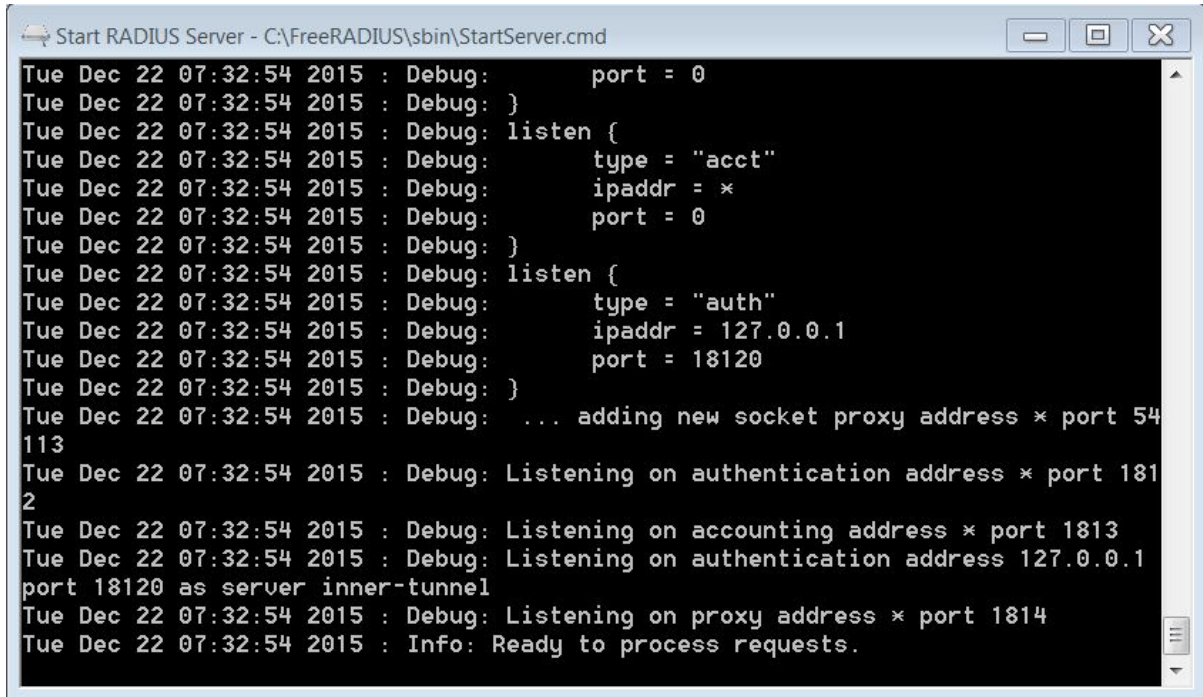
2. Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\examples\wlan\leap\rsi_wlan_config.h** file and update/modify the following macros:

```
#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 |
TCP_IP_FEAT_ICMP )

#define RSI_CUSTOM_FEATURE_BIT_MAP              0
#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
```

Executing the Application

1. Connect the WiSeConnect device (Silabs module + STM32 Board) to the Windows PC running Keil IDE.
2. Configure the Access point in WPA-EAP/WPA2-EAP mode to connect the Silabs device in enterprise secured mode.
3. Run Radius server in Windows PC2 which is connected to AP through LAN by providing required certificate and credentials.

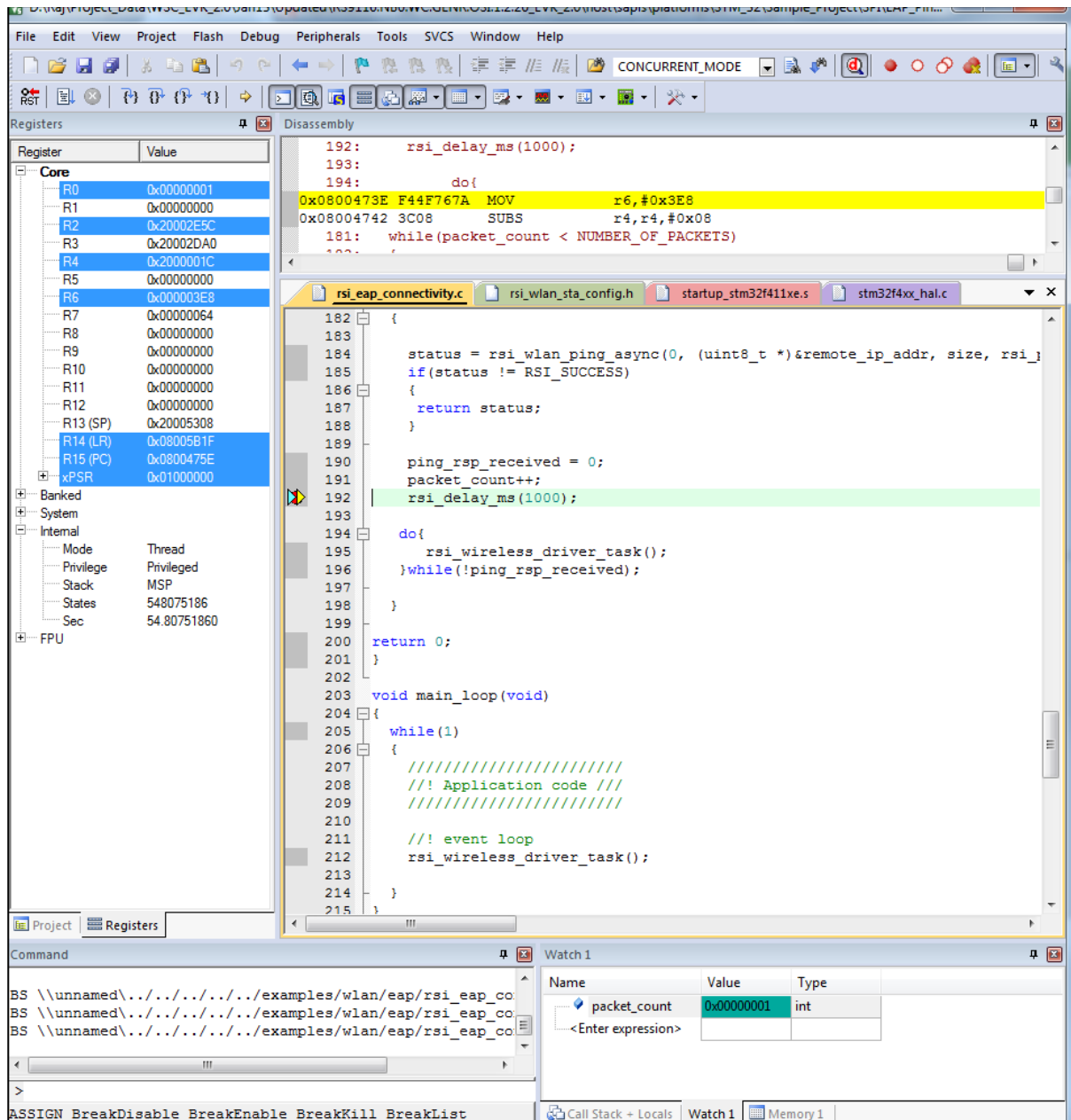


```

Start RADIUS Server - C:\FreeRADIUS\sbin\StartServer.cmd
Tue Dec 22 07:32:54 2015 : Debug:      port = 0
Tue Dec 22 07:32:54 2015 : Debug:    }
Tue Dec 22 07:32:54 2015 : Debug:  listen {
Tue Dec 22 07:32:54 2015 : Debug:      type = "acct"
Tue Dec 22 07:32:54 2015 : Debug:      ipaddr = *
Tue Dec 22 07:32:54 2015 : Debug:      port = 0
Tue Dec 22 07:32:54 2015 : Debug:    }
Tue Dec 22 07:32:54 2015 : Debug:  listen {
Tue Dec 22 07:32:54 2015 : Debug:      type = "auth"
Tue Dec 22 07:32:54 2015 : Debug:      ipaddr = 127.0.0.1
Tue Dec 22 07:32:54 2015 : Debug:      port = 18120
Tue Dec 22 07:32:54 2015 : Debug:    }
Tue Dec 22 07:32:54 2015 : Debug:    ... adding new socket proxy address * port 54
113
Tue Dec 22 07:32:54 2015 : Debug: Listening on authentication address * port 181
2
Tue Dec 22 07:32:54 2015 : Debug: Listening on accounting address * port 1813
Tue Dec 22 07:32:54 2015 : Debug: Listening on authentication address 127.0.0.1
port 18120 as server inner-tunnel
Tue Dec 22 07:32:54 2015 : Debug: Listening on proxy address * port 1814
Tue Dec 22 07:32:54 2015 : Info: Ready to process requests.

```

4. After the program gets executed, the Silabs device would be connected to the access point which is in enterprise security having the configuration the same as that of in the application and get 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 figure 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.



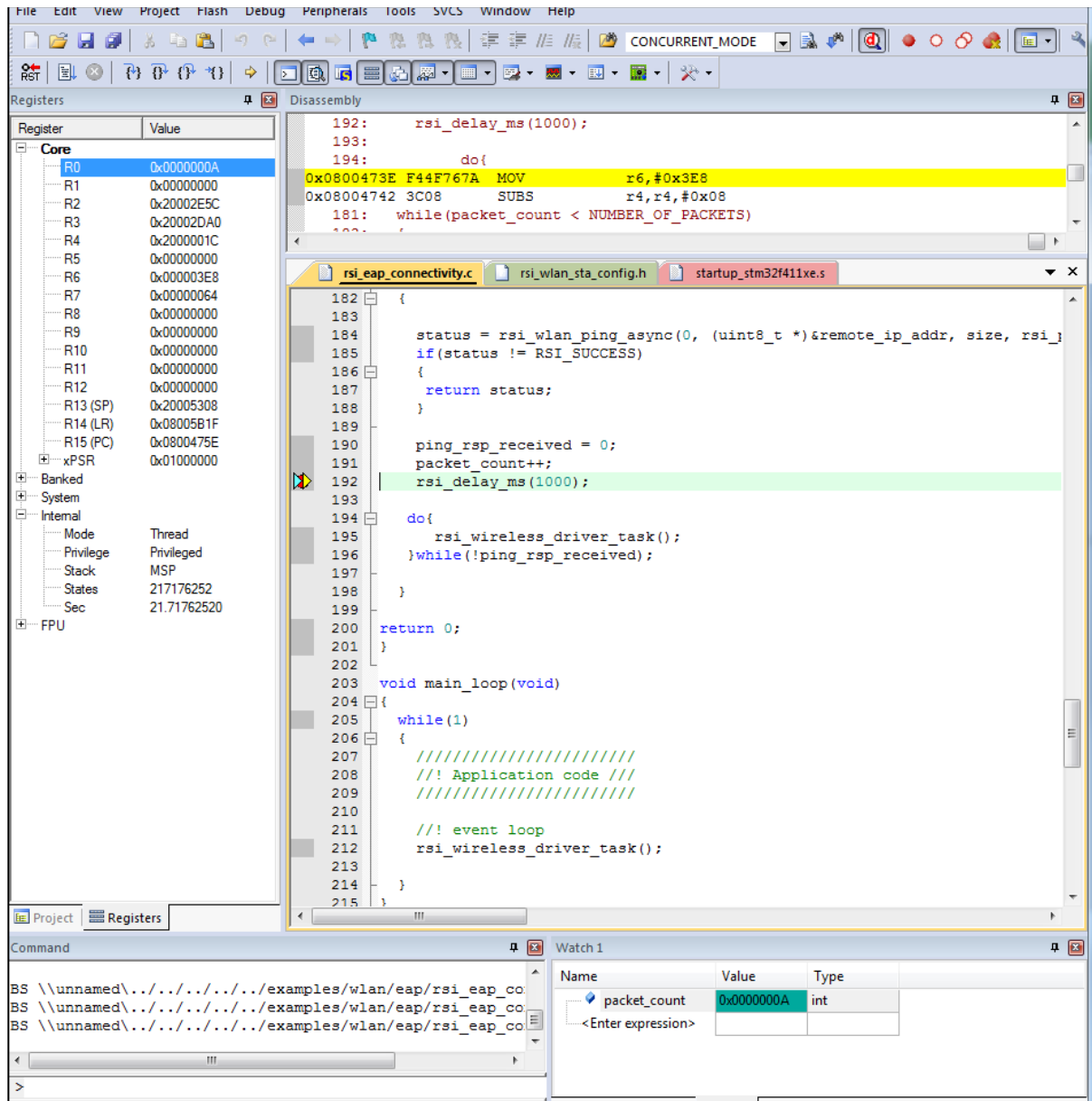
The screenshot displays the Silicon Labs IDE interface for debugging an STM32 microcontroller. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The main window is divided into several panes.

Registers Pane: Shows a list of registers (R0-R15, xPSR) with their current values. The registers are grouped into Core, Banked, System, Internal, and FPU sections.

Disassembly Pane: Displays the disassembled code for the 'rsi_eap_connectivity.c' file. The code includes instructions such as 'rsi_delay_ms(1000);', 'do{', 'rsi_wireless_driver_task();', and 'while(!ping_rsp_received);'.

Command Pane: Shows a series of 'BS' commands used for debugging, such as 'BS \\unnamed\\...\\examples\\wlan\\eap\\rsi_eap_co'.

Watch 1 Pane: Displays the value of the variable 'packet_count' as 0x00000001, with its type listed as 'int'.



3.3.7 Example7: BT_Alonge

Overview

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

Sequence of Events

This Application explains user how to:

- Configure the Silicon Labs module to act as Master
- Connect the Silicon Labs module with the Slave
- Accept SPP level connection from the Smartphone
- Loopback 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

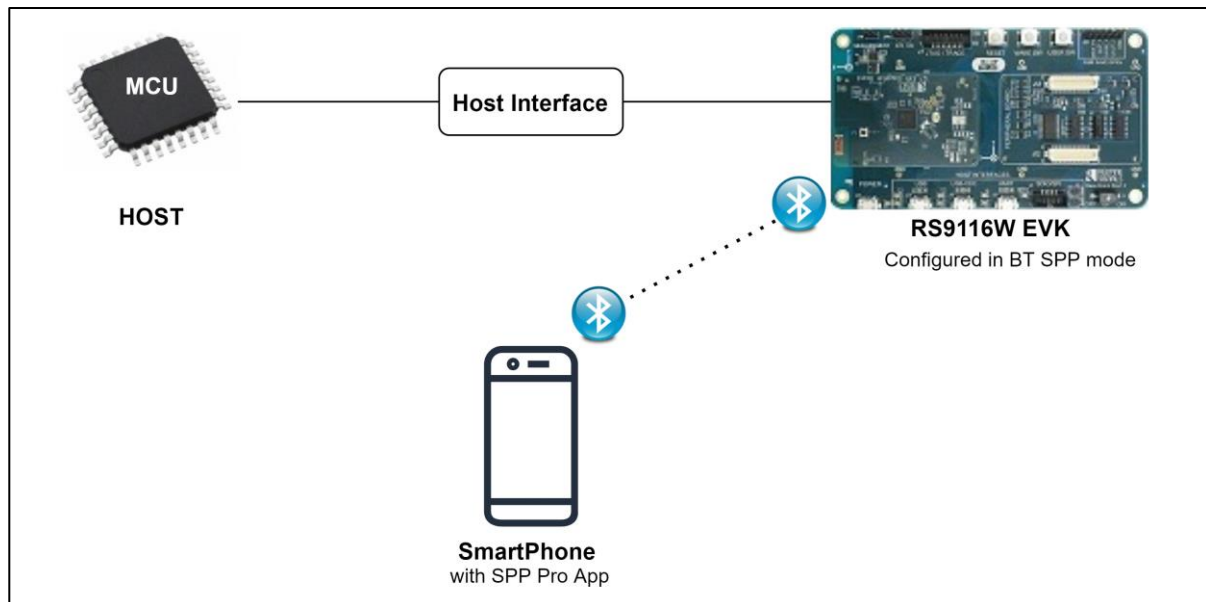


Figure 20: Setup Diagram for BT SPP Master Example

Configuration and Execution of the Application

Configuring the Application

1. Open Project
RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\BT_Alone and then double click on wlan_sta_ble_bridge (uVision5 Project).
 Then open
RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\bt\spp_master_slave\rsi_spp_master_slave.c file
 and update/modify the following macros:

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

```
#define SPP_MODE SPP_MASTER
```

PIN_CODE refers to four bytes string required for the pairing process.

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

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

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

Note:

In the smartphone, the User Can check the BD address of the Bluetooth device by searching for the "Pair new device"

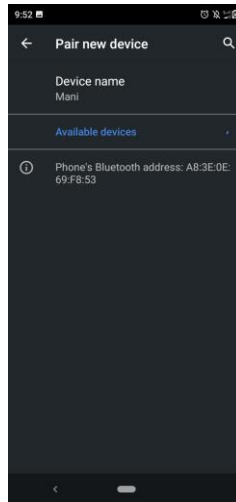


Figure 21: Bluetooth Address

The following are the **non-configurable** macros in the application.

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

```
#define BT_GLOBAL_BUFF_LEN 15000
```

- Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\bt\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_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);

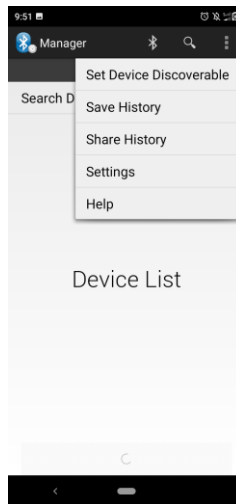
//To set the device role to either Master or Slave.(set_role = 0 -->Master, set_role = 1 -->Slave)
rsi_bt_set_local_device_role((int8_t *)str_conn_bd_addr, set_role, &device_state);

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

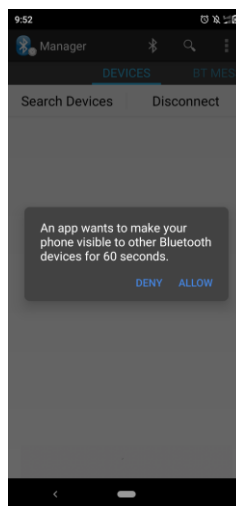
The status of the `role_change` function should return success for successful role switch, otherwise, fail.

Executing the Application

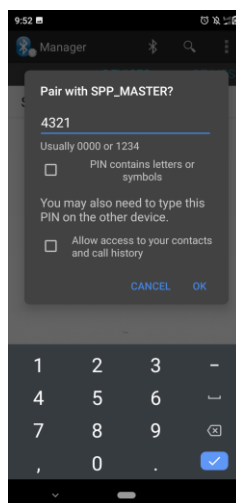
- Power on Bluetooth in the smartphone and put it in visible mode to all Bluetooth devices(Can be done through "Bluetooth SPP Manager" app. As shown in the below image



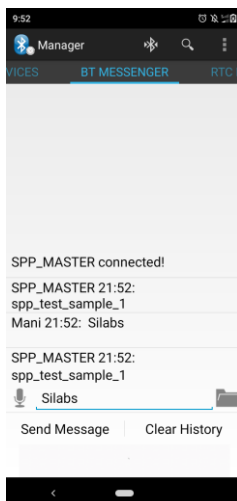
2. Tap the "Set Device Discoverable". By tapping the mode as per the app the module will be discoverable for 60Seconds.



3. After the program gets executed, the Silicon Labs module initiates a basic connection with the remote device (Smartphone). The user has to provide **PIN_CODE** at the remote device for successful connectivity. Please find below images for connection at the remote device.



4. After a successful connection, In smartphone Silicon Labs module lists under Paired devices.
5. After a successful connection, Open the Sena BT term "Bluetooth SPP Manager app" on mobile and wait for connection which will initiate from "Silicon Labs Module". After a successful scan, the Silicon Labs module can initiate a connection to the already bonded device as we have already completed basic pairing from the Silicon Labs module.



6. At the remote device, a Bluetooth pairing request will pop-up for SPP connection success. providing a secret key (PIN_CODE) for SPP connection success.
7. After successful SPP Connection in mobile side, go to BT_MESSENGER tab which is next to DEVICES tab. Please write some data (Ex: "Silicon Labs signals") and call "Send Message" option in that app. we can see that the same message has been exchanged between two modules which is kind of loopback test.

BT SPP Slave mode

Overview

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

Sequence of Events

This Application explains user how to:

- Configure the Silicon Labs module to act as Slave
- Configure device in discoverable and connectable mode
- Accept SPP level connection from the Smartphone
- Loopback 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 a 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
- BT master device

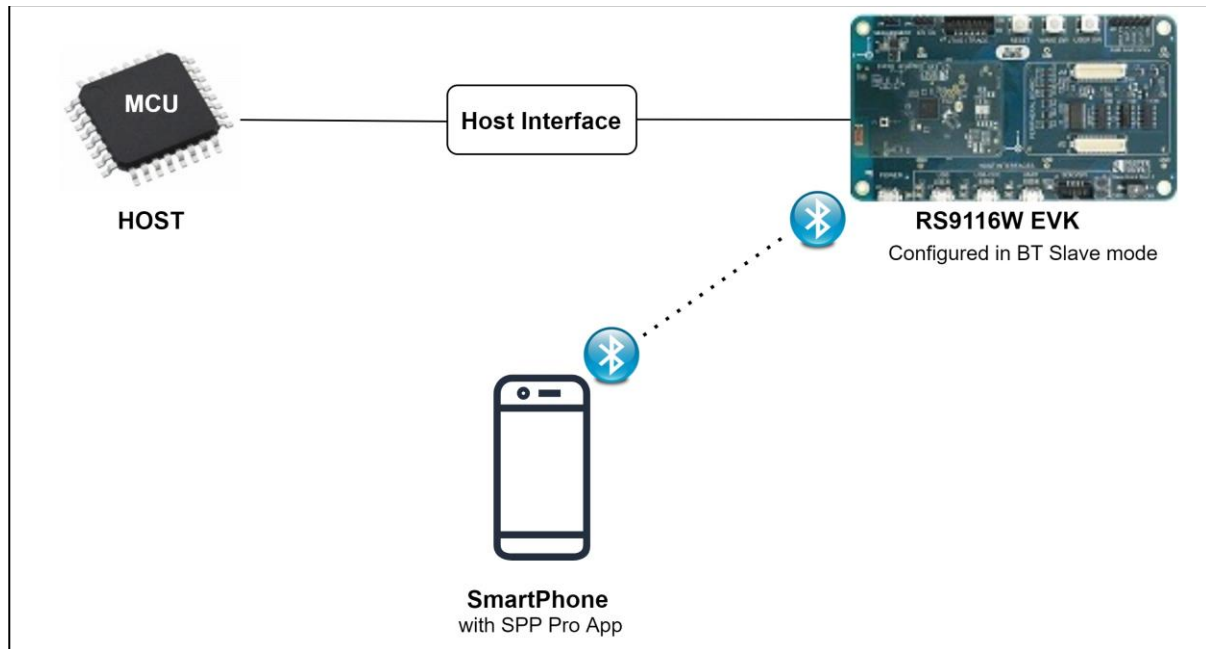


Figure 22: Setup Diagram for BT SPP Slave

Configuration and Execution of the Application

- Open Project **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\BT_Alone** and then double click on **wlan_sta_ble_bridge** (uVision5 Project).
Then open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\bt\spp_master_slave\rsi_spp_master_slave.c** file and update/modify the following macros:

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

```
#define SPP_MODE SPP_SLAVE
```

PIN_CODE refers to four bytes string required for the pairing process.

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

The following are the **non-configurable** macros in the application.

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

```
#define BT_GLOBAL_BUFF_LEN 15000
```

- Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\bt\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_CLIENT
#define RSI_CUSTOM_FEATURE_BIT_MAP 0
#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);

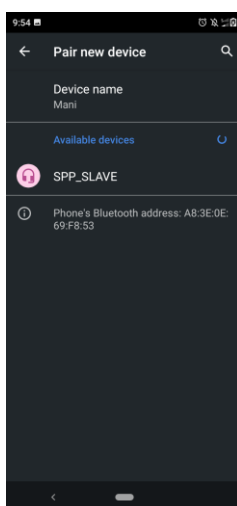
//To set the device role to either Master or Slave.(set_role = 0 -->Master, set_role = 1 -->Slave)
rsi_bt_set_local_device_role((int8_t *)str_conn_bd_addr, set_role, &device_state);

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

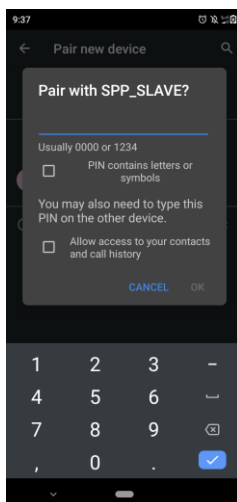
The status of the role_change function should return success for successful role switch, otherwise, fail.

Executing the Application

1. After the program gets executed, the Silicon Labs module initializes the SPP profile and waits for the incoming connection.
2. Open the Bluetooth setting present in the mobile.



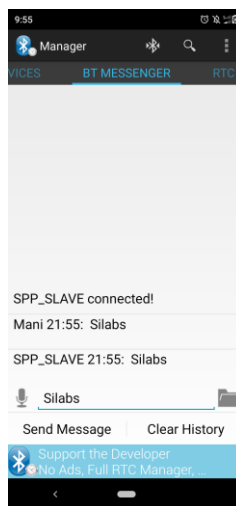
3. Open the Bluetooth SPP pro app on mobile and do the scan until the Silicon Labs module (Ex: "SPP_SLAVE") gets present in the scan list.
4. After the successful scan, select the device and initiate pairing to Silicon Labs module.



5. After initiating pairing, the Pairing request will pop-up at the smartphone side and issue a secret key which is given at the Silicon Labs module (PIN_CODE) side. Then the module successfully establishes the physical level connection. By using the Bluetooth SPP manager spp profile can be established. Tap on the scan option visible in the app you will get scan results as shown below



6. After a Tapping on the SPP_SLAVE successful pair, initiate SPP connection is successfully established.



7. After successful SPP Connection in mobile side, go to BT_MESSENGER tab which is next to DEVICES tab. Please write some data (Ex: "Silicon Labs signals") and call "Send Message" option in that app. we can see that the same message has been exchanged between two modules which is kind of loopback test.

3.3.8 Example8: AWS_IoT

Protocol Overview

MQTT is a publish-subscribe based "lightweight" 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

An MQTT client is any device from a microcontroller to a full-fledged server, that has an 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 the 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 to 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.

Example Overview

Overview

User has to create an IoT Thing in the AWS cloud before running this application and this procedure is described in the section "Appendix A: Cloud User Manual" in RS9116W_Guide_for_SAPI_Application_Examples.pdf

In this application, the RS9116W device configured as a WiFi station and connects to Access Point. After a successful WiFi connection, the application connects to the MQTT AWS server and subscribe to a topic.

Device thing shadow thing 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. You can use the shadow to get and set the state of a device over MQTT.

This example will update the device thing shadow document, after successfully update we can see the updates in thing shadow in the cloud.

Setup Requirements

- Windows PC1 with Keil or IAR IDE
- RS9116W Module
- Access point with an Internet connection
- AWS account

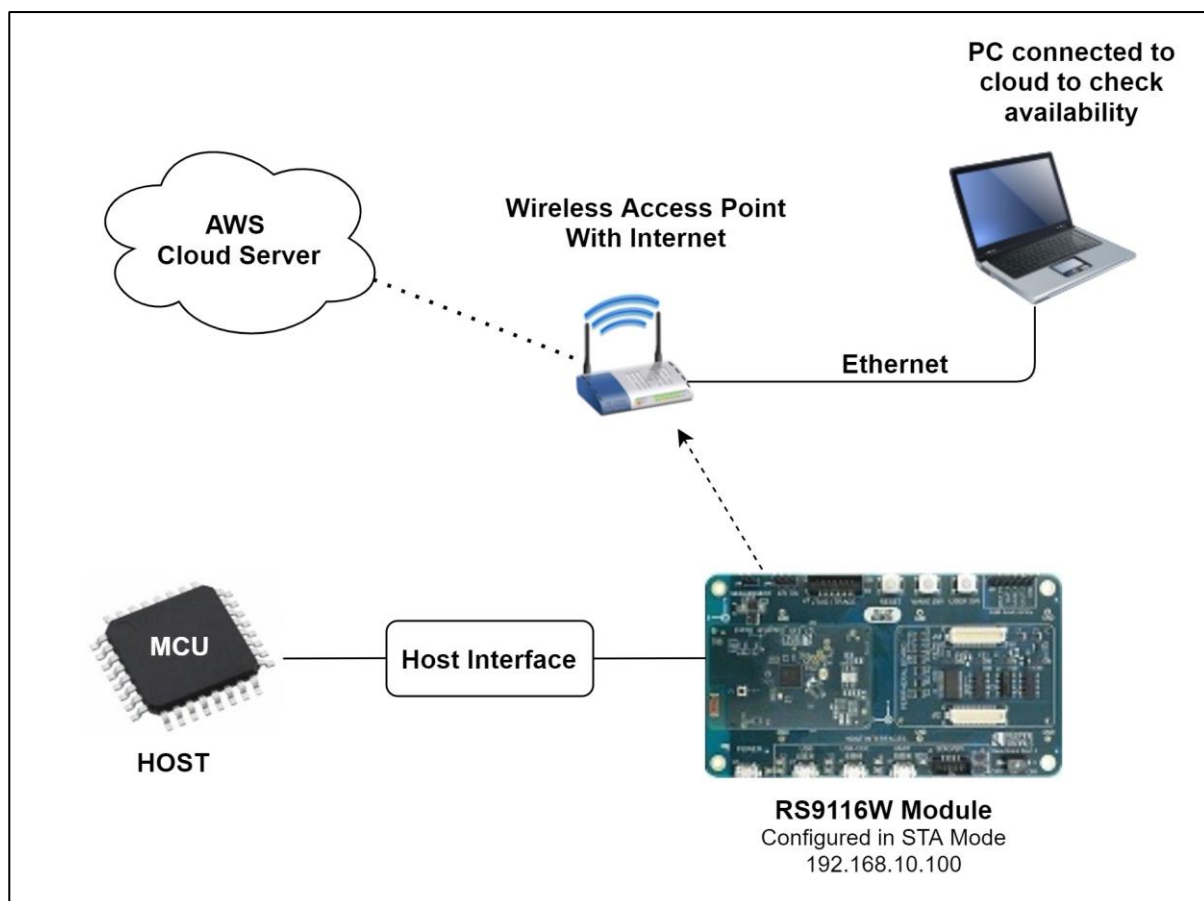


Figure 23: Setup Diagram for Device Shadow Example

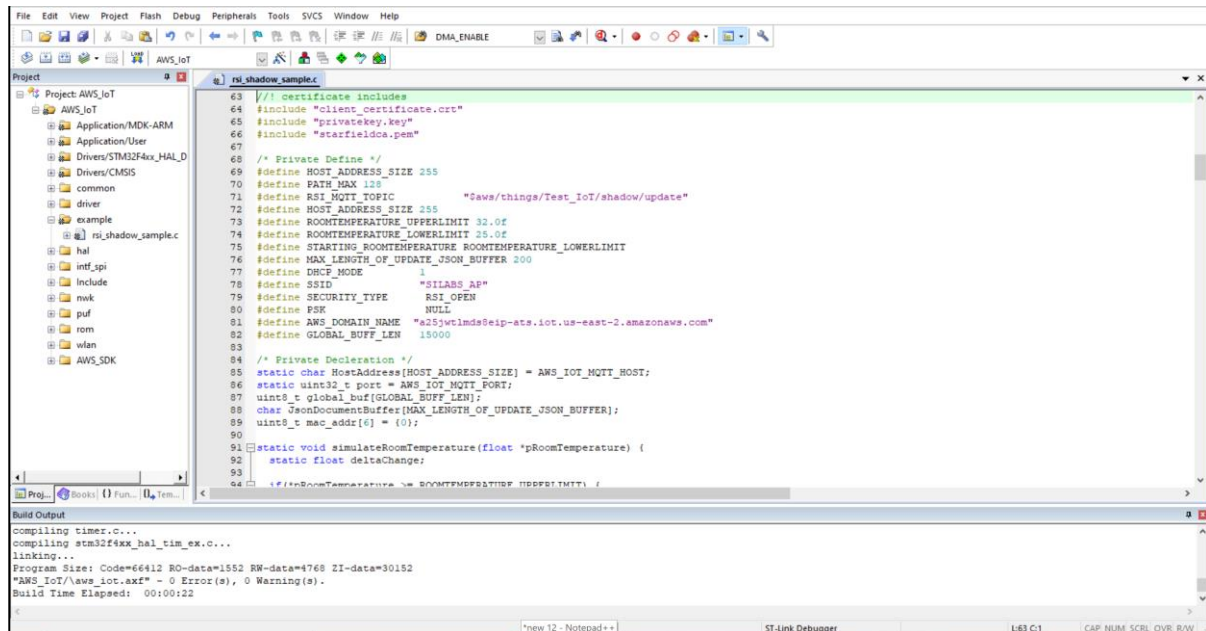
Configuration and Steps for Execution

Configuring the Application

1. Open Project `RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPIAWS_IoT` and then double click on `AWS_IoT` (uVision5 Project).

Then

open `RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\cloud\aws_iot\device_shadow\rsi_shadow_sample.c` file and update/modify the following macros:



SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

SECURITY_TYPE refers to the type of security. In this application, STA supports Open, WPA-PSK, WPA2-PSK securities.

The 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>"
```

Note

To Load certificate

rsi_wlan_set_certificate API expects the certificate in the form of a linear array. So, convert the pem certificate into a 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 starfieldca.pem

The script will generate wifiuser.pem in which one linear array named **starfieldca** contains the certificate.

- After the conversion of the certificate, update the *rsi_shadow_sample.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, users need not load the certificate for every boot up unless the certificate change. So, define **LOAD_CERTIFICATE** as 0, if the certificate is already present in the device.

Global buffer or memory which is used for MQTT client initialization. This buffer is used for the MQTT client information storage.

uint8_t mqtt_client_buffer[MQTT_CLIENT_INIT_BUFF_LEN];

QOS indicates the level of assurance for the 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 the WiSeConnect MQTT client is supposed to subscribe.

```
#define RSI_MQTT_TOPIC "$aws/things/Test_IoT/shadow/update" (Update
with a Topic name of IoT Thing in AWS cloud)
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN 1500
```

To configure the IP address

DHCP_MODE refers to whether the IP address configured through DHCP or STATIC

```
#define DHCP_MODE 1
```

Note:

If the user wants to configure the 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 the STA IP address through **STATIC** then set **DHCP_MODE** macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

The IP address to be configured to the device in STA mode should be in a 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
```

The 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
```

The 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 **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\cloud\aws_iot\device_shadow\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 the desired configuration in respective example folders user need not change for each example.

3. Configure below parameter in

RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\cloud\aws_iot\device_shadow\aws_iot_config.h file:

```
#define AWS_IOT_MQTT_HOST "a25jwtmlmds8eip-ats.iot.us-east-2.amazonaws.com" ///< Customer specific MQTT HOST. The same will be used for Thing Shadow
#define AWS_IOT_MQTT_PORT 8883 ///< default port for MQTT/S
#define AWS_IOT_MQTT_CLIENT_ID "Silabs" ///< MQTT client ID
should be unique for every device
#define AWS_IOT_MY_THING_NAME "Test_IoT" ///< Thing Name of the
Shadow this device is associated with
#define AWS_IOT_ROOT_CA_FILENAME "rootCA.crt " ///< Root CA file name
#define AWS_IOT_CERTIFICATE_FILENAME "cert.pem" ///< device signed
certificate file name
#define AWS_IOT_PRIVATE_KEY_FILENAME "privkey.pem " ///< Device private key
filename
```

Executing the Application

1. Configure the Access point with Internet connection in OPEN/WPA-PSK/WPA2-PSK mode to connect the RS9116W device in STA mode.
2. Login to the AWS account and open IoT Thing created.
3. Update Domain name and topics name to connect to the AWS MQTT cloud. We will subscribe to the update/delta topic to get a shadow update
4. Update the device thing shadow document, after successfully update the thing, we can observe the updates in thing shadow in the cloud.
5. User can see the updated messages in cloud shadow under the Thing

→ Follow this below link for API description

<http://aws-iot-device-sdk-embedded-c-docs.s3-website-us-east-1.amazonaws.com/index.html>

3.3.9 Example9: sample_project

Refer to section "Steps for Keil IDE" in [Steps for executing STM32 examples using Master Application \(Sample Project\)](#)

3.3.10 Example10: 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 is done in UDP Protocol because it is a connection-less protocol.

Note

This example application (UDP_Client) is based on the UART host interface between the STM32 and Silabs EVK through the GPIO header card. Pin configuration is defined in the "RS9116 WC Assembling and Accessing STM NUCLEO-F411RE Board" section.

UART INTERFACE_BAREMETAL:

1. This path "RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\UART" contains the reference UART SAPIs project (udp_client) without OS.
2. The Baud rate used in this project is 115200, For higher baud rate Hardware flow control has to be used.
3. By default UDP client example is present in the project, In order to try any other example, you can exclude the current example from the build and include the required example in the Keil IDE and make sure corresponding examples rsi_wlan_config.h file is included.
4. Open "RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\include\rsi_uart.h" file and update/modify the following macros:
The parameters need to be changed in the application to enable different baud rates and to select the UART HW flow control option:

```
UART Baud rate to be set
For baud rates greater than 115200, it is mandatory to enable UART hardware flow control.
#define BAUDRATE                B115200

Enable UART hardware flow control 0 - disable, 1- Enable
#define UART_HW_FLOW_CONTROL    0
```

uart_hw_flowcontrol_enable:

a.

Enable or Disable UART hardware flow control.

1/2 -Enable and Pinset to be used for RTS/CTS purposes.

0 - Disable If uart_hw_flowcontrol_enable parameter is 0, UART flow control is disabled. If the parameter is given as 1 or 2, It means UART hardware flow control is enabled and Pin set to be used.

If the parameter is given as 1: Pin set used for RTS/CTS functionality will be

UART_CTS : GPIO - 11

UART_RTS : GPIO - 7

If the parameter is given as 2: Pin set used for RTS/CTS functionality will be

UART_CTS : GPIO - 15

UART_RTS : GPIO - 12

uart_hw_flow_control_enabled(1 byte) : If uart flow control is enabled, then response parameter will 1 else 0.

RTS/CTS Pin connection between the RS9116W and STM32 board:

1) RS9116W board side UART CTS (GPIO15) (J1_27) --> STM32 board side UART RTS (PA12) (CN10_12)

2) RS9116W board side UART RTS (GPIO12) (J1_21) --> STM32 board side UART CTS (PA11) (CN10_14)

Overview

The UDP client application demonstrates how to open and use a standard UDP client socket and sends data to the UDP server socket. Once it is configured as a UDP client, it can establish and maintain a network conversation by means of an application program for exchanging data.

Sequence of Events

This Application explains to the user how to:

- Connect the Device to an Access point and get an IP address through DHCP.
- Open UDP Server socket at Access point using iperf application.
- Open the UDP client socket in the device.
- Send data from Silicon Labs device to remote peer using an 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 a variety of host processors.

WiSeConnect based Setup Requirements

- Windows PC with Host interface (UART/ USB-CDC/ 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)

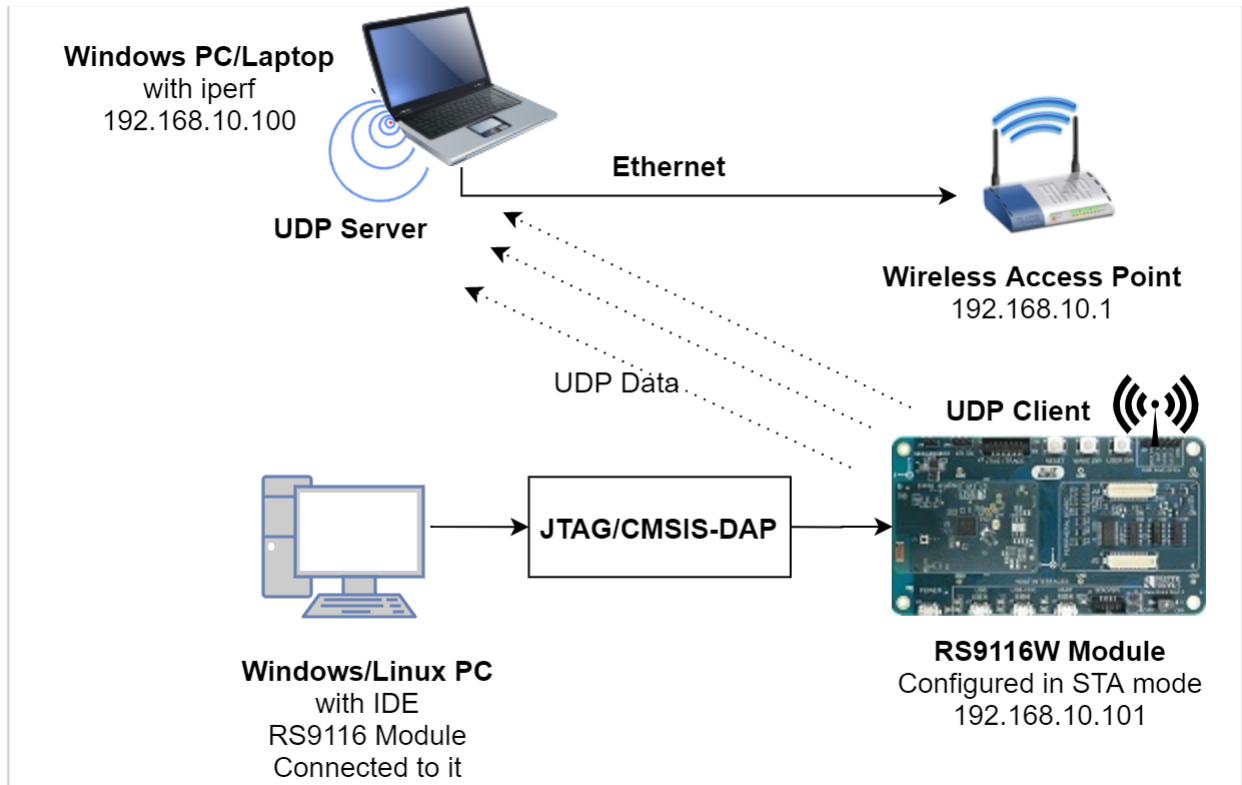


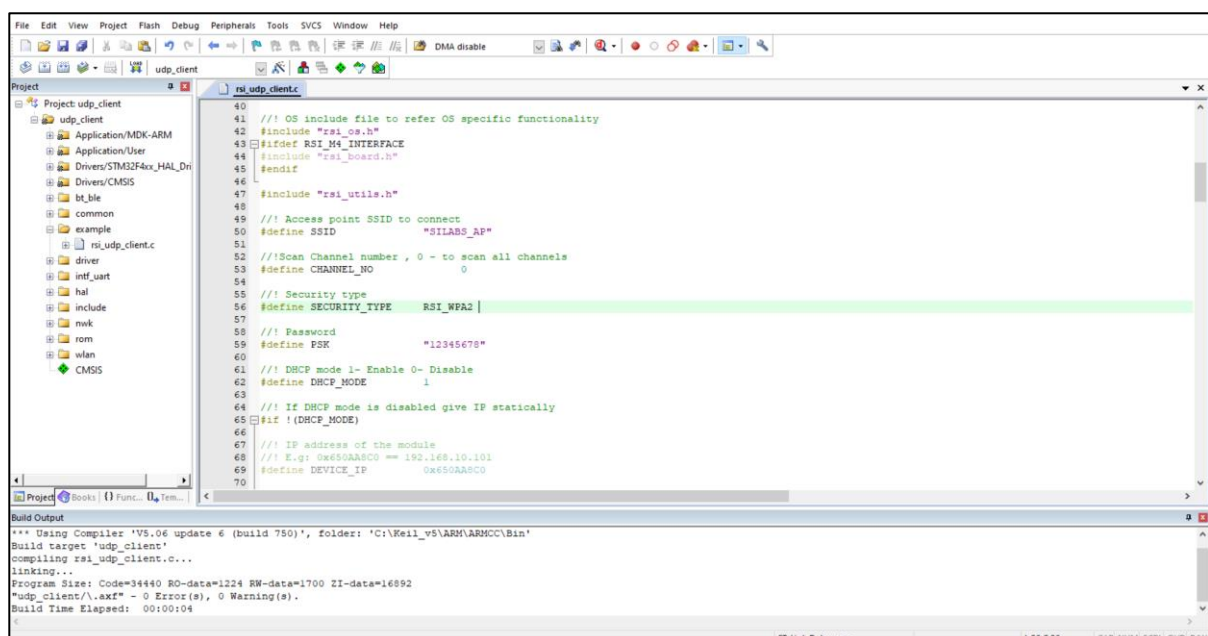
Figure 24: Setup Diagram for UDP Client Socket

Configuration and Steps for Execution

Configuring the Application

1. Open Project
RS9116.NB0.WC.GENR.OSI.x.x.xx\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\UART\udp_client and then double click on **udp_client** (uVision5 Project).

Then open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\udp_client\rsi_udp_client.c** file and update/modify following macros:



```

40
41 // OS include file to refer OS specific functionality
42 #include "rsi_os.h"
43 #ifndef RSI_M4_INTERFACE
44 #include "rsi_board.h"
45 #endif
46
47 #include "rsi_utils.h"
48
49 // Access point SSID to connect
50 #define SSID "SILABS_AP"
51
52 // Scan Channel number, 0 - to scan all channels
53 #define CHANNEL_NO 0
54
55 // Security type
56 #define SECURITY_TYPE RSI_WPA2
57
58 // Password
59 #define PSK "12345678"
60
61 // DHCP mode 1- Enable 0- Disable
62 #define DHCP_MODE 1
63
64 // If DHCP mode is disabled give IP statically
65 #if ! (DHCP_MODE)
66
67 // IP address of the module
68 // E.g: 0x650A8C0 == 192.168.10.101
69 #define DEVICE_IP 0x650A8C0
70
  
```

Build Output

```

*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'udp_client'
Compiling rsi_udp_client.c...
Linking...
Program Size: Code=34440 RO-data=1224 RW-data=1700 ZI-data=16892
"udp_client\*.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:04
  
```

SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

CHANNEL_NO refers to the channel in which the 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 and WPA2-PSK securities.

The 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 to 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 the TCP server socket.

The IP address should be in a 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 how many packets to send from the device to the UDP server.

```
#define NUMBER_OF_PACKETS          1000
```

Application memory length which is required by the driver

```
#define GLOBAL_BUFF_LEN            8000
```

To configure IP address:

DHCP_MODE refers whether the IP address configured through DHCP or STATIC

```
#define DHCP_MODE                   1
```

Note:

If the user wants to configure the 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 the STA IP address through STATIC then set DHCP_MODE macro to "0" and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

The IP address which is to be configured to the device in STA mode should be in a 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
```

The IP address of the gateway should also be in a 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
```

The 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 0x0FFFFFFF
```

- Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\sapis\examples\wlan\udp_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
```

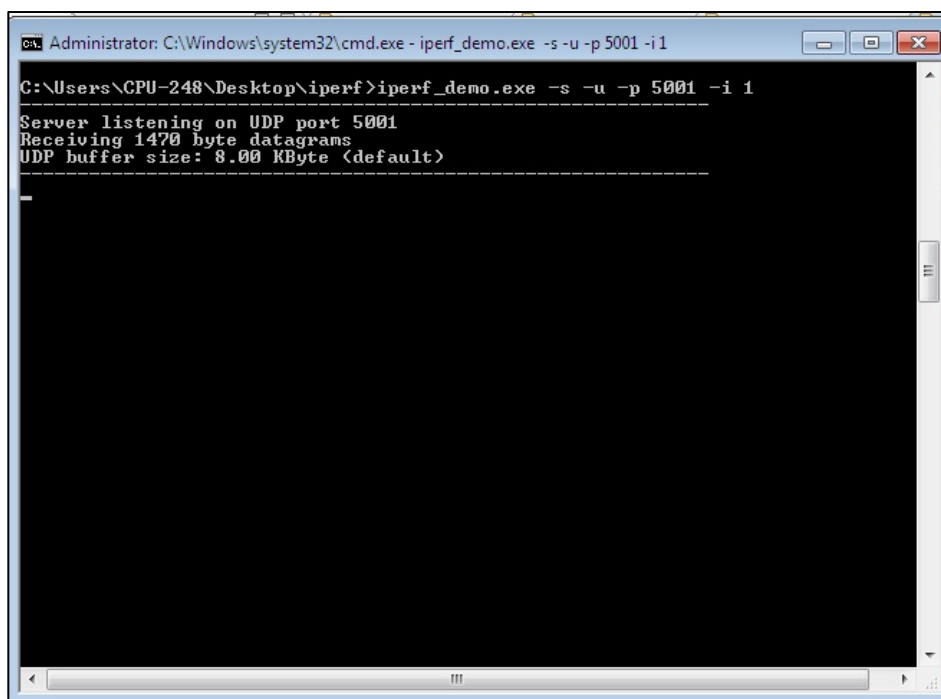
Note:

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 in OPEN / WPA-PSK/WPA2-PSK mode in order to connect the Silicon Labs device in STA mode.
- Open UDP server application using iperf application in Windows PC2 which is connected to the Access point through LAN.

iperf_demo.exe -s -u -p <SERVER_PORT> -i 1



- After the program gets executed, The Silicon Labs device would scan and connect to the Access point and get IP.
- After a successful connection, the device STA connects to the UDP server socket opened on Windows PC2 using UDP client socket and sends configured **NUMBER_OF_PACKETS** to the remote UDP server. Please refer to the below image for the reception of UDP data on the UDP server.

```

Administrator: C:\Windows\system32\cmd.exe - iperf_demo.exe -s -u -p 5001 -i 1

C:\Users\CPU-248\Desktop\iperf>iperf_demo.exe -s -u -p 5001 -i 1

Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)

-----
[1224] local 192.168.0.100 port 5001 connected with 192.168.0.101 port 30000
[ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagram
[1224] 0.0- 1.0 sec  1.48 KBytes   12.1 Kbits/sec  22.684 ms   -1529289828/12146
444 (-1.3e+002%)
[1224] 1.0- 2.0 sec    936 Bytes    7.49 Kbits/sec  25.487 ms    -39/    0 (-1.5%)
[1224] 1.0- 2.0 sec    39 datagrams received out-of-order
[1224] 2.0- 3.0 sec    1.01 KBytes   8.26 Kbits/sec  26.136 ms    -43/    0 (-1.5%)
[1224] 2.0- 3.0 sec    43 datagrams received out-of-order
[1224] 3.0- 4.0 sec    960 Bytes    7.68 Kbits/sec  22.394 ms    -40/    0 (-1.5%)
[1224] 3.0- 4.0 sec    40 datagrams received out-of-order
[1224] 4.0- 5.0 sec    1.83 KBytes   15.0 Kbits/sec  11.129 ms    -78/    0 (-1.5%)
[1224] 4.0- 5.0 sec    78 datagrams received out-of-order
[1224] 5.0- 6.0 sec    1.57 KBytes   12.9 Kbits/sec  17.382 ms    -67/    0 (-1.5%)
[1224] 5.0- 6.0 sec    67 datagrams received out-of-order
[1224] 6.0- 7.0 sec    1.66 KBytes   13.6 Kbits/sec  18.634 ms    -71/    0 (-1.5%)
[1224] 6.0- 7.0 sec    71 datagrams received out-of-order
[1224] 7.0- 8.0 sec    1.88 KBytes   15.4 Kbits/sec  14.382 ms    -80/    0 (-1.5%)
[1224] 7.0- 8.0 sec    80 datagrams received out-of-order
[1224] 8.0- 9.0 sec    960 Bytes    7.68 Kbits/sec  24.326 ms    -40/    0 (-1.5%)
[1224] 8.0- 9.0 sec    40 datagrams received out-of-order
[1224] 9.0-10.0 sec    1.52 KBytes   12.5 Kbits/sec  18.888 ms    -65/    0 (-1.5%)
[1224] 9.0-10.0 sec    65 datagrams received out-of-order
[1224] 10.0-11.0 sec    1.01 KBytes   8.26 Kbits/sec  16.728 ms    -43/    0 (-1.5%)
[1224] 10.0-11.0 sec    43 datagrams received out-of-order
[1224] 11.0-12.0 sec    1.10 KBytes   9.02 Kbits/sec  23.615 ms    -47/    0 (-1.5%)
[1224] 11.0-12.0 sec    47 datagrams received out-of-order
[1224] 12.0-13.0 sec    1.24 KBytes   10.2 Kbits/sec  16.136 ms    -53/    0 (-1.5%)
[1224] 12.0-13.0 sec    53 datagrams received out-of-order

```

3.4 Reference Projects for Keil Freertos

To run all the reference projects with Cube IDE, follow the section : [Getting Started with Keil IDE](#) in this document.

Refer to section [FreeRTOS Porting for STM32](#), for porting FreeRTOS source to existing reference projects.

3.4.1 Example1: 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:

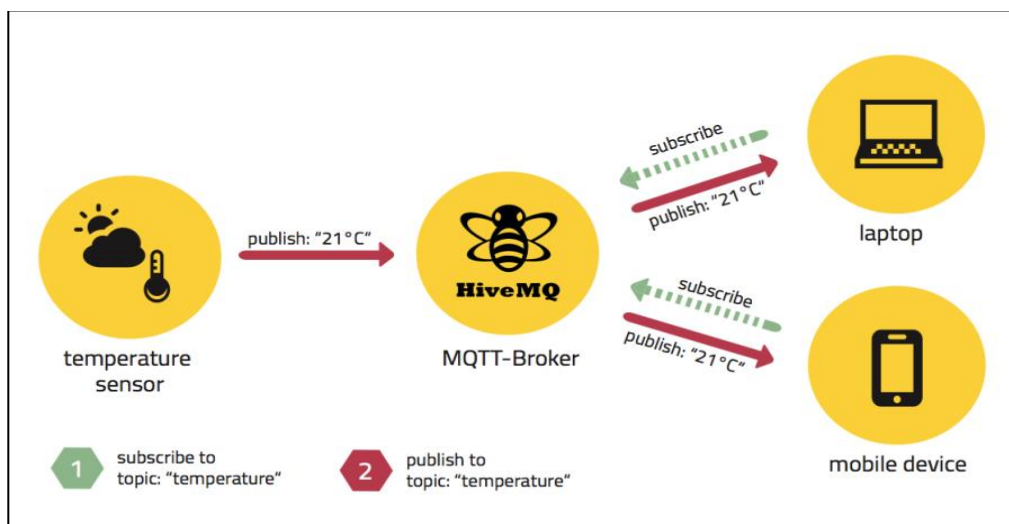


Figure 25: Demonstration of MQTT Protocol

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>

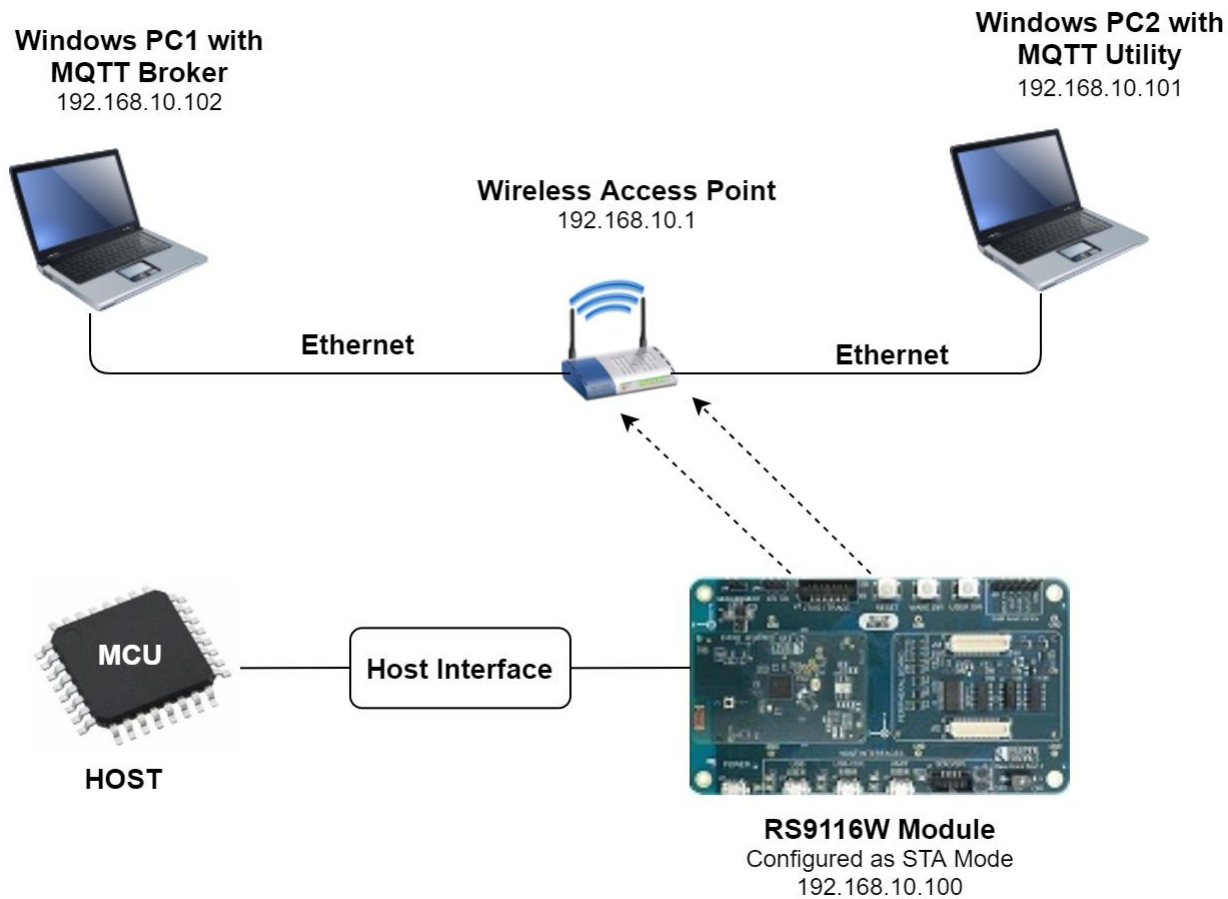


Figure 26: Setup Diagram for MQTT Client Example

Configuration and Steps for Execution

Configuring the Application

1. Open `RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\examples\wlan\mqtt_client\rsi_mqtt.c` file and update/modify the following macros:

SSID refers to the name of the Access point.

```
#define SSID "SILABS_AP"
```

SECURITY_TYPE refers to the type of security. In this application, STA supports Open, WPA-PSK, WPA2-PSK securities.

The 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 to 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 mqtt_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
```

2. Open **RS9116.NB0.WC.GENR.OSI.x.x.xx\host\apis\examples\wlan\mqtt_client\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_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
```

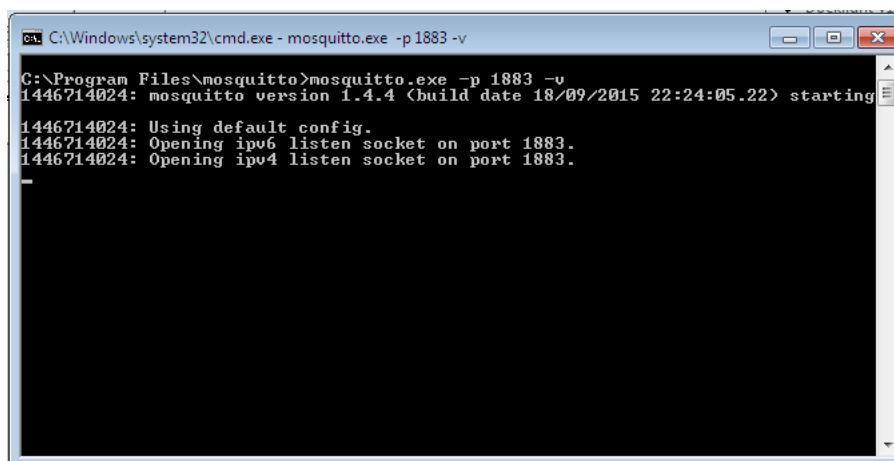
Note:

- **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 the following command. Open Command prompt and go to MQTT installed folder (Ex: C:\Program Files\mosquitto) and run the following command:

mosquitto.exe -p 1883 -v

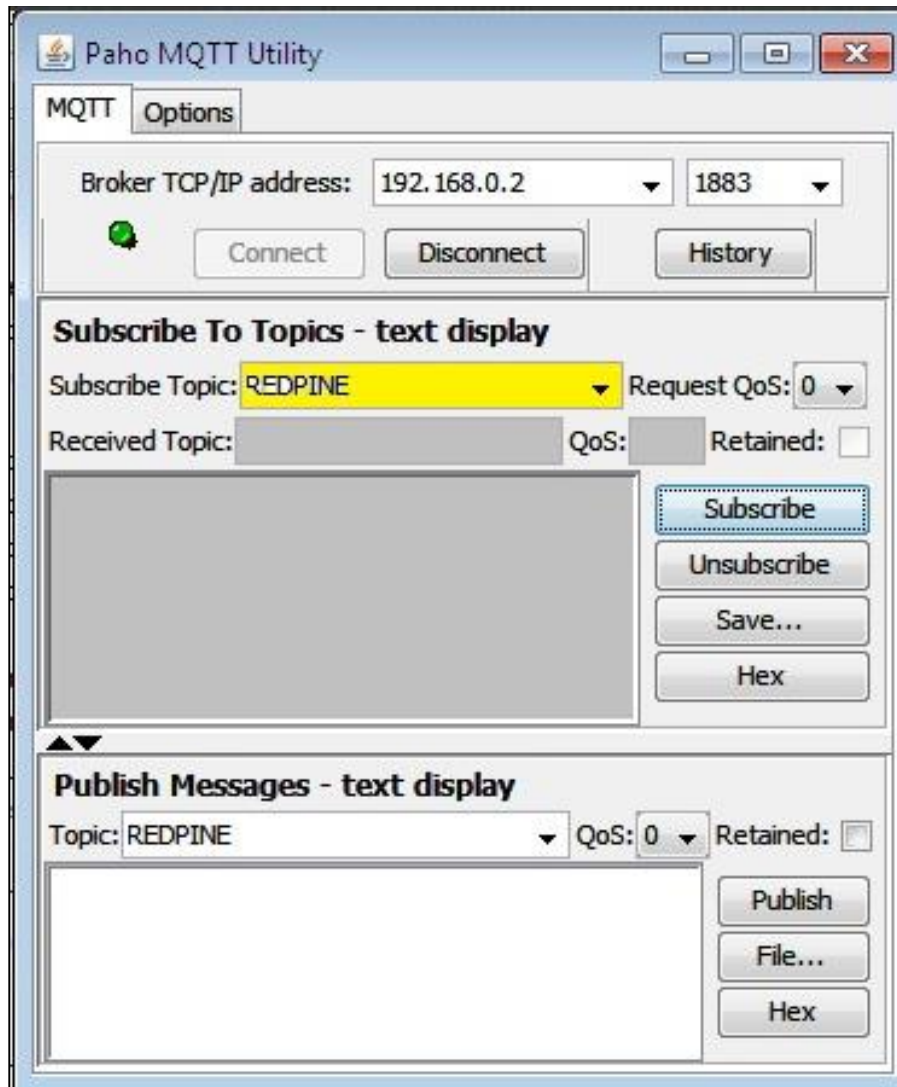


```
C:\Windows\system32\cmd.exe - mosquitto.exe -p 1883 -v
C:\Program Files\mosquitto>mosquitto.exe -p 1883 -v
1446714024: mosquitto version 1.4.4 (build date 18/09/2015 22:24:05.22) starting
1446714024: Using default config.
1446714024: Opening ipv6 listen socket on port 1883.
1446714024: Opening ipv4 listen socket on port 1883.
```


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.

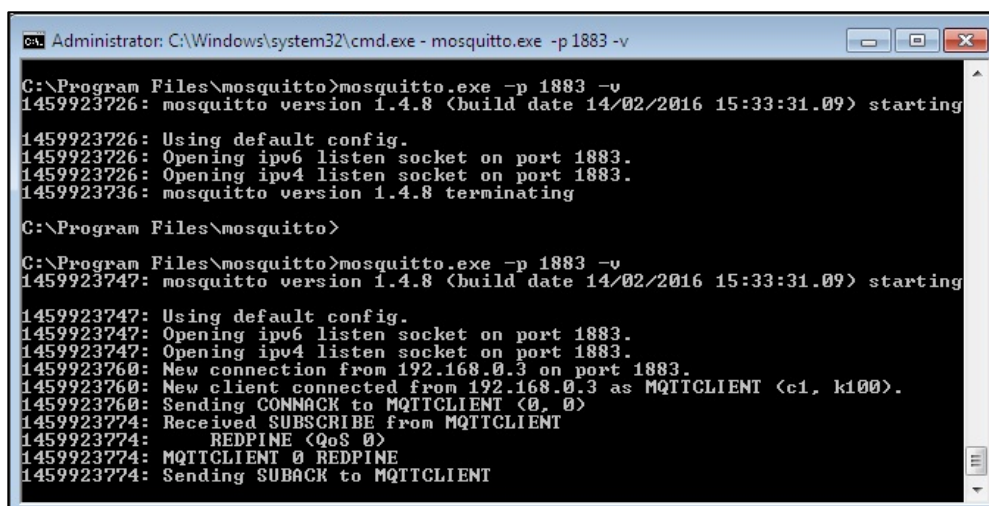


5. After successful connection, subscribe to the topic from MQTT client utility.



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.

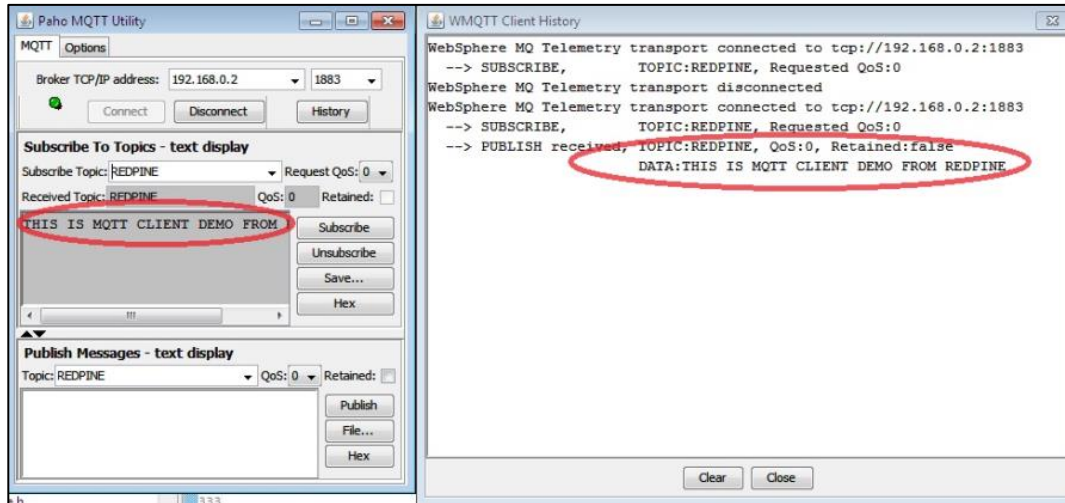


8. After successful subscription to the topic **RSI_MQTT_TOPIC** (Ex: "REDPINE"), the device publishes a message which is given in **publish_message** array (Ex: "THIS IS MQTT CLIENT DEMO FROM REDPINE") on the subscribed topic.

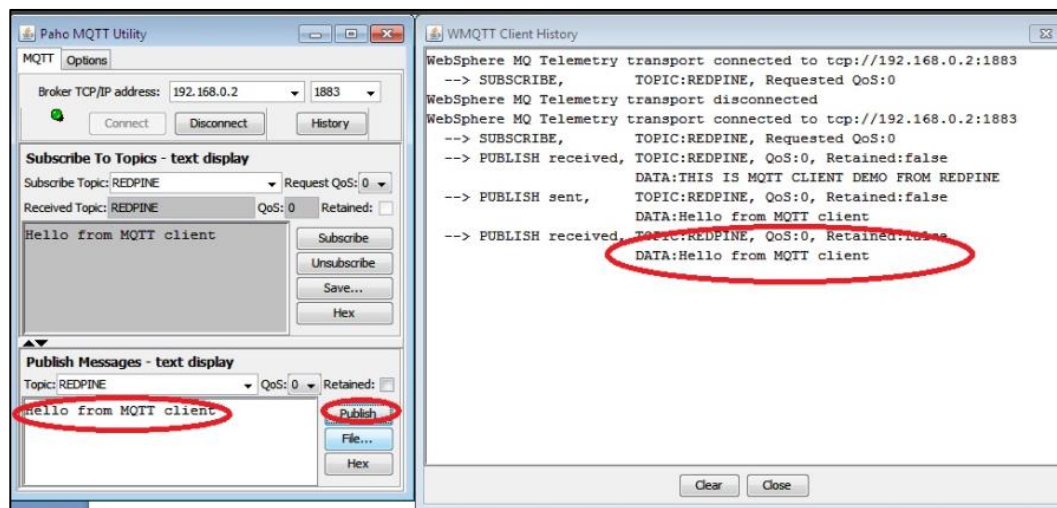
9. MQTT client utility which is running on Windows PC3 will receive the message published by the Silicon Labs device

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_rcv_data` API.

3.4.2 Example2 : wlan_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).

The 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 the user with the ability 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.

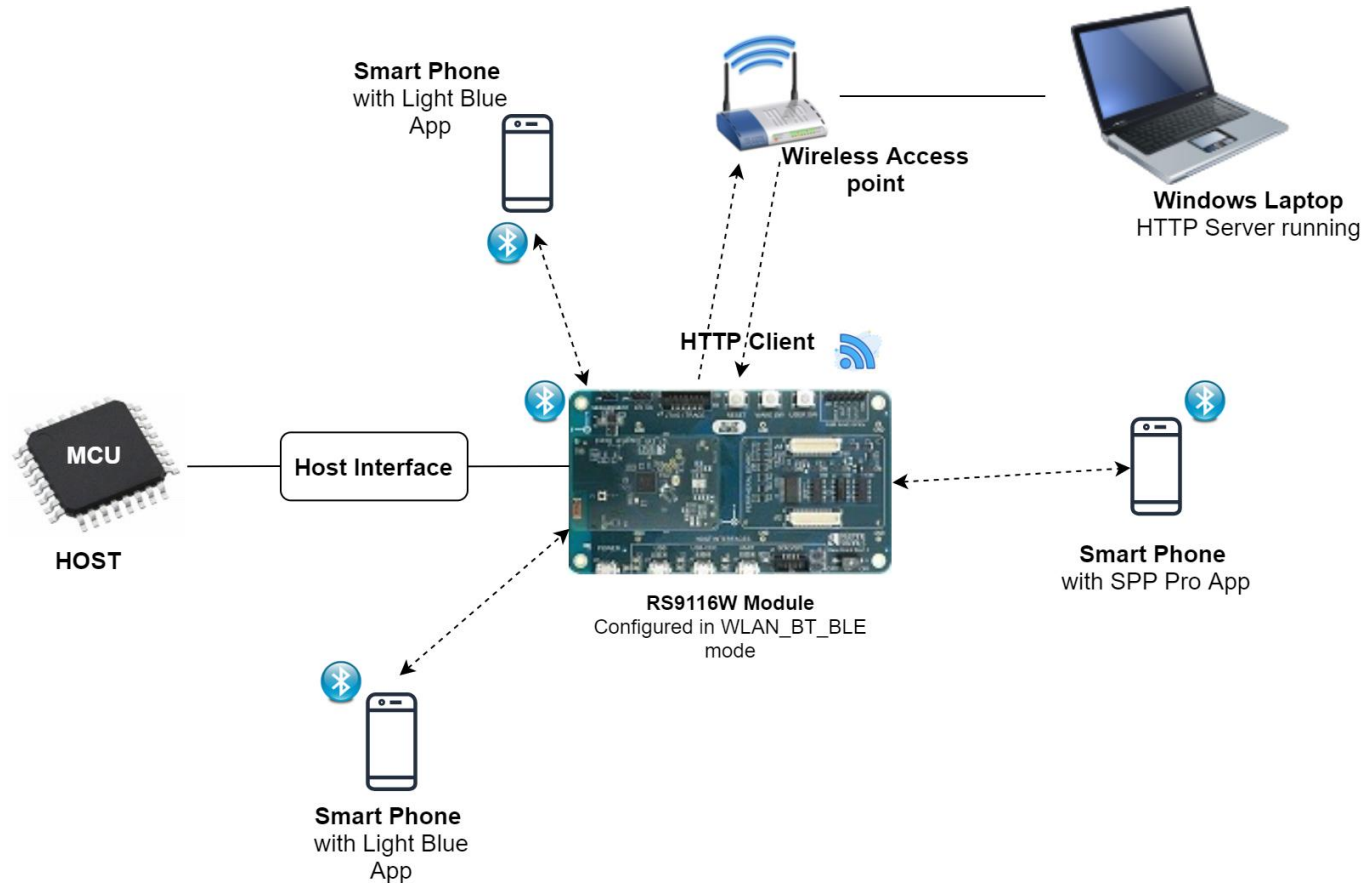


Figure 27: Setup Diagram for WLAN HTTP/HTTPS BT SPP BLE Dual Role Example

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_bt_ble\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

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_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))

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_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"
```

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_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 manual or auto IP, 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://siproweb.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 rejoin feature, configure below macros

```
//! 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\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 simultaneously 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 initiates 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

3.4.3 Example3: wlan_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 the first connection is used for provisioning and the second connection is used for data transfer.

Sequence of Events

WLAN task: Fetches the access point 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.x\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.

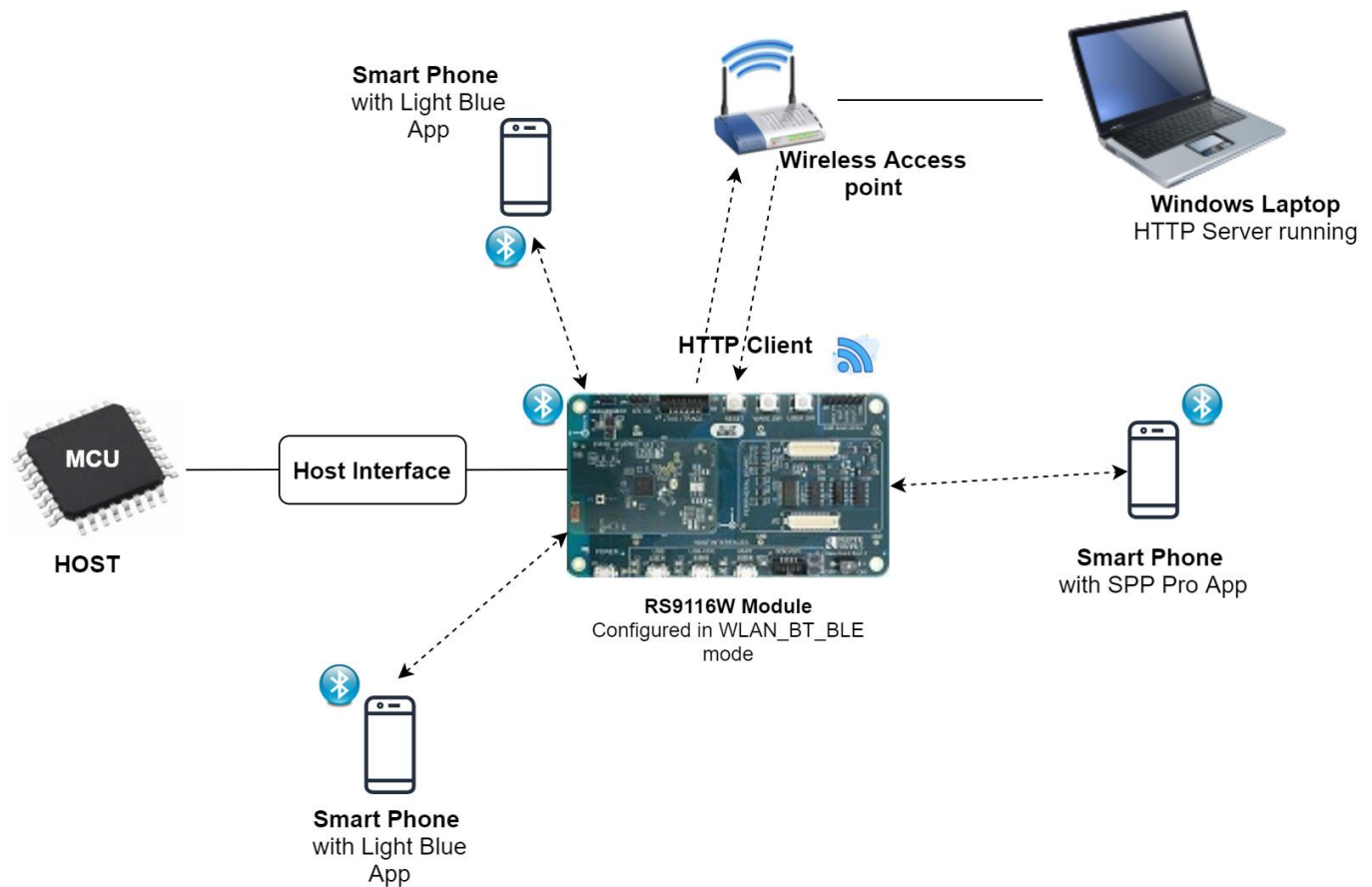


Figure 28: Setup Diagram for WLAN HTTP/HTTPS BT SPP BLE Provisioning Example

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\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

- open '**rsi_ble_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.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:

- Maximum no. of RSI_BLE_MAX_NBR_MASTERS can be configured to '2' and RSI_BLE_MAX_NBR_SLAVES to '3'
- 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_INIDICATIONS_SERVICE_UUID_M1 0x1809 //! Health thermometer Alert
service uuid
#define RSI_BLE_CLIENT_INIDICATIONS_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 RS9116WModule 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 accesspoints are displayed on the screen.
10. Connect to required accesspoint and provide PSK if prompted.
11. If credentials are valid, Module connects to that accesspoint, notifies to Redpine APP as "AP connection successfull" 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 successfull 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 initiates connection.
21. If BLE connection is successfull, 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

3.4.4 Example4: 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.

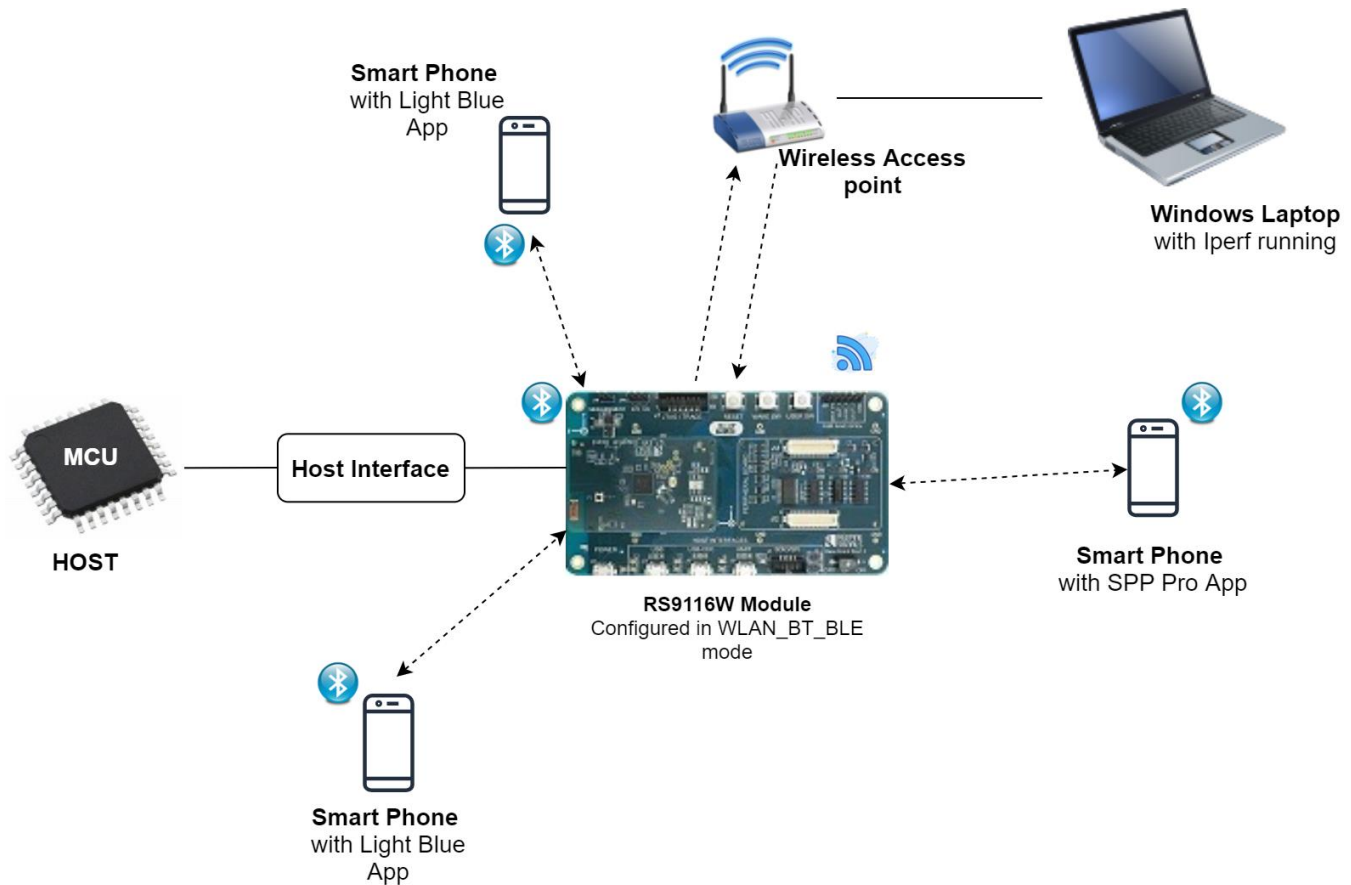


Figure 29: Setup Diagram for WLAN Throughput BT SPP BLE Dual Role Example

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\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
```

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
```

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
```

- Open '**rsi_ble_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.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
```

- Select BT configurations in '**rsi_bt_config.h**' file provided in the release package at '**RS9116.NB0.WC.GENR.OSI.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

- Compile the project and flash the binary onto STM32
- To measure throughput, following configurations can be applied
 - 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
 - To measure UDP Rx throughput, 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
 - 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
 - To measure TCP Rx throughput, 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
 - 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.

3.4.5 Example5:sample_project

Refer to section "Steps for Keil IDE" in [Steps for executing STM32 examples using Master Application \(Sample Project\)](#) document

3.4.6 Example6: udp_client

Refer to section [udp_client](#) in this document.

3.5 Reference projects for Cube Baremetal

To run all the reference projects with Cube IDE, follow the section : [Getting Started with STM32CubeIDE](#) in this document.

3.5.1 Example1:eap

Refer to steps [Getting Started with CUBE_IDE](#) to run with Cube IDE and for application execution refer to section [Enterprise Ping Client \(eap\)](#)

3.5.2 Example2: Firmware_Upgrade

Refer to steps [Getting Started with CUBE_IDE](#) to run with Cube IDE and for application execution refer to section [Wireless Firmware Upgradation](#)

3.5.3 Example3: Power_save

Refer to steps [Getting Started with CUBE_IDE](#) to run with Cube IDE and for application execution refer to section [WLAN Standby Associated Power save](#)

3.5.4 Example4: Wlan_ble

Refer to steps [Getting Started with CUBE IDE](#) to run with Cube IDE and for application execution refer to section [WLAN Station BLE Bridge](#)

3.5.5 Example5: sample_project

Refer to section "Steps for STMCube IDE" in [Steps for executing STM32 examples using Master Application \(Sample Project\)](#)

3.6 Reference Projects for Cube Freertos

To run all the reference projects with Cube IDE, follow the section : [Getting Started with STM32CubeIDE](#) in this document.

Refer to section [FreeRTOS Porting for STM32](#), for porting FreeRTOS source to existing reference projects.

3.6.1 Example1: wlan_https_bt_spp_ble_dual_role

Refer to steps [Getting Started with STM32CubeIDE](#) to run with Cube IDE and for application execution refer to section [wlan_https_bt_spp_ble_dual_role](#)

3.6.2 Example2: wlan_https_bt_spp_ble_provisioning

Refer to steps [Getting Started with STM32CubeIDE](#) to run with Cube IDE and for application execution refer to section [wlan_https_bt_spp_ble_provisioning](#)

3.6.3 Example3: wlan_throughput_bt_spp_ble_dual_role

Refer to steps [Getting Started with STM32CubeIDE](#) to run with Cube IDE and for application execution refer to section [wlan_throughput_bt_spp_ble_dual_role](#)

3.6.4 Example4: sample_project

Refer to section "Steps for STMCube IDE" in [Steps for executing STM32 examples using Master Application \(Sample Project\)](#)

4 FreeRTOS Porting for STM32

This section describes steps to compile FreeRTOS based example projects on STM32 provided in the RS9116W release.

The procedure includes the following steps for executing the STM32 FreeRTOS Projects using Keil and STM32Cube IDE

- Download the RS9116W release and navigate to the FreeRTOS projects directory as below
 1. RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects
 2. RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Cube_Freertos\Projects
- Generate '**Middlewares**' folder using STM32CubeMX software tool by selecting FreeRTOS.
Add the CubeMX generated "Middlewares" folder to the project directory paths as
(\Reference_Projects\Keil_Freertos & \Reference_Projects\Cube_Freertos).
Please follow the section "**Steps for Compiling STM32 FreeRTOS Example**" for generating "**Middlewares**" for both Keil and CubeIDE.
- Launch the project, compile and execute

Sample SAPI examples with FreeRTOS are provided in RS9116W release package at '*RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects*' and '*RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Cube_Freertos\Projects*'. These example projects cannot be compiled directly. It requires the FreeRTOS source to be added in these projects for successful compilation.

Prerequisites

- Windows PC (64-bit preferred)
- STM32CubeMx can be downloaded from <https://www.st.com/en/development-tools/stm32cubemx.html>
- Keil IDE
- STM32Cube IDE
- ST-Link driver (<https://www.st.com/en/development-tools/stsw-link009.html>)

Steps for Compiling STM32 FreeRTOS Example

1. Launch STM32CubeMx.

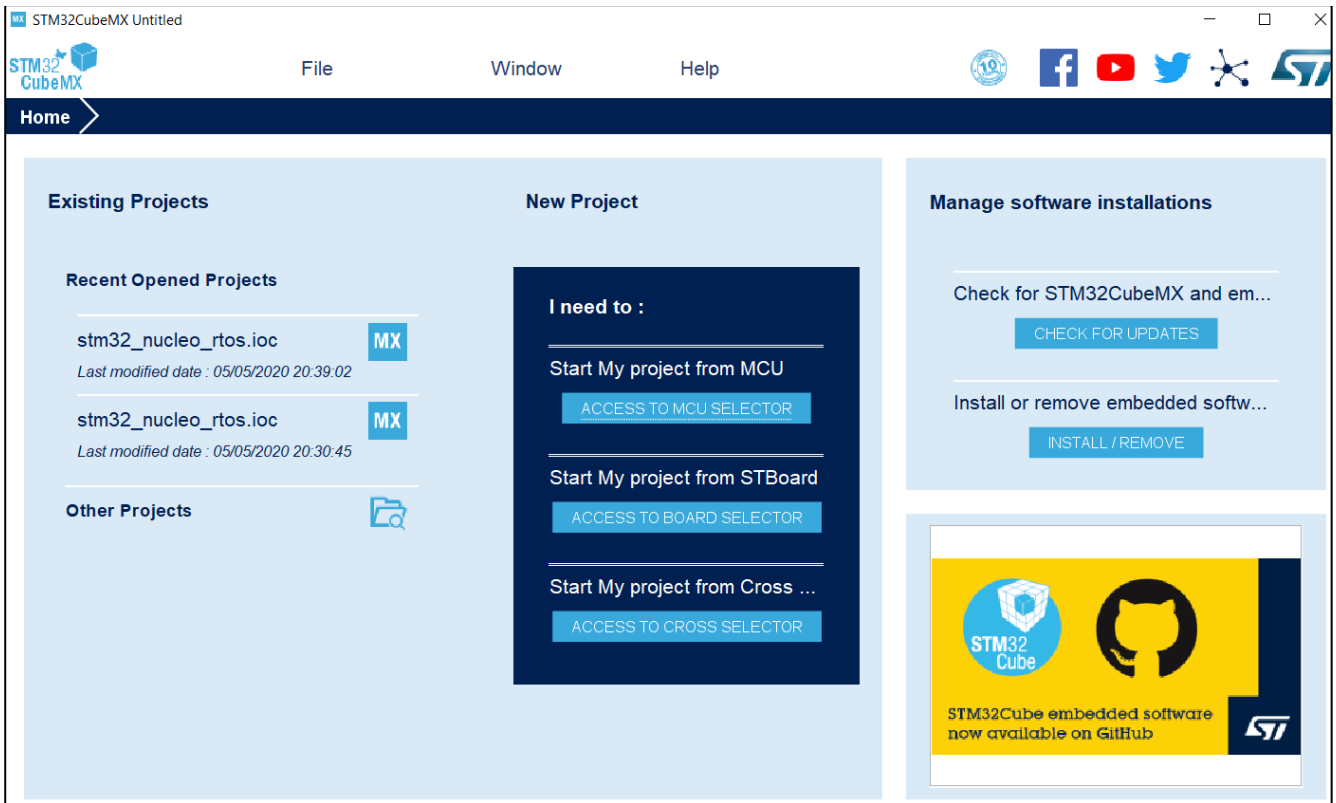


Figure 30: STM32CubeMX - Launch

- Click on the 'File' tab and select 'New Project'.

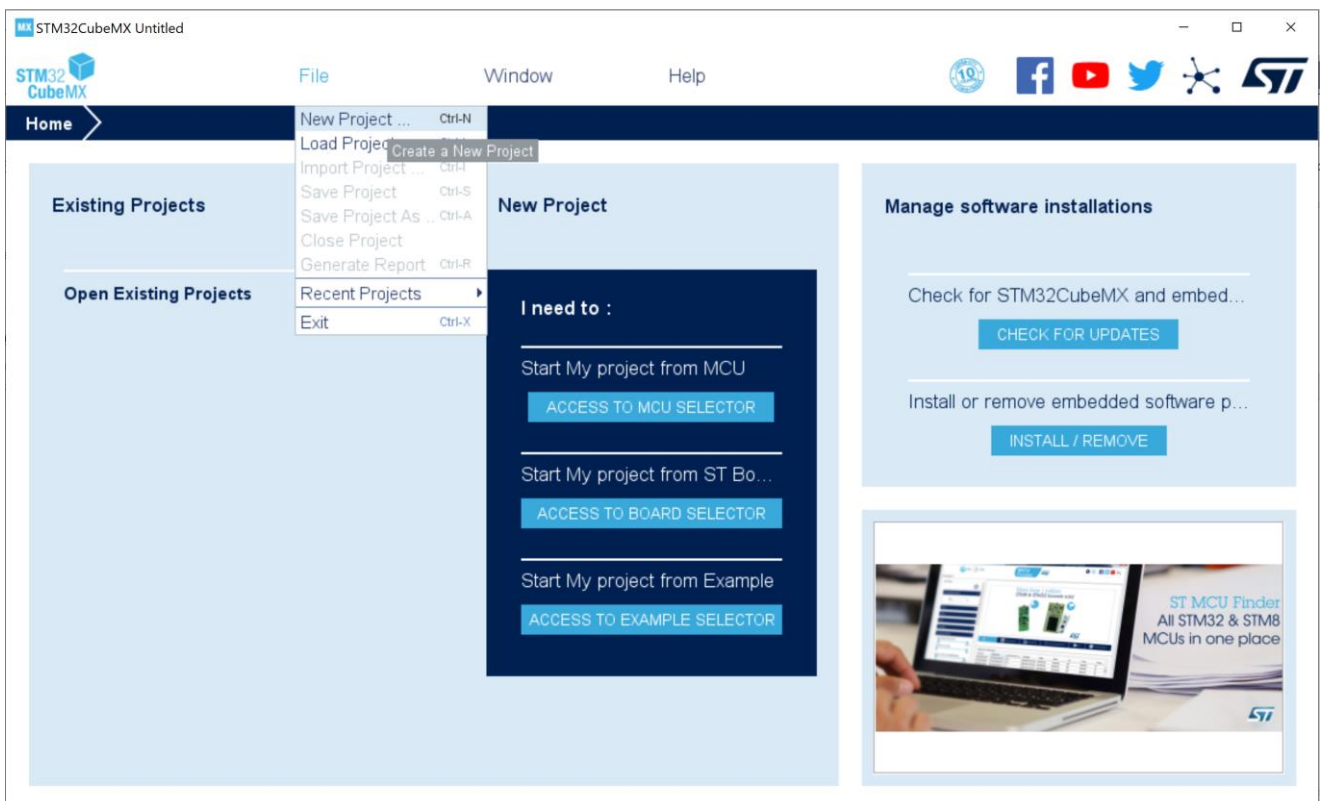


Figure 31: STMCubeMX-New Project

- Select '**STM32F411RE**' from the MCU's/MPU's list as shown below.

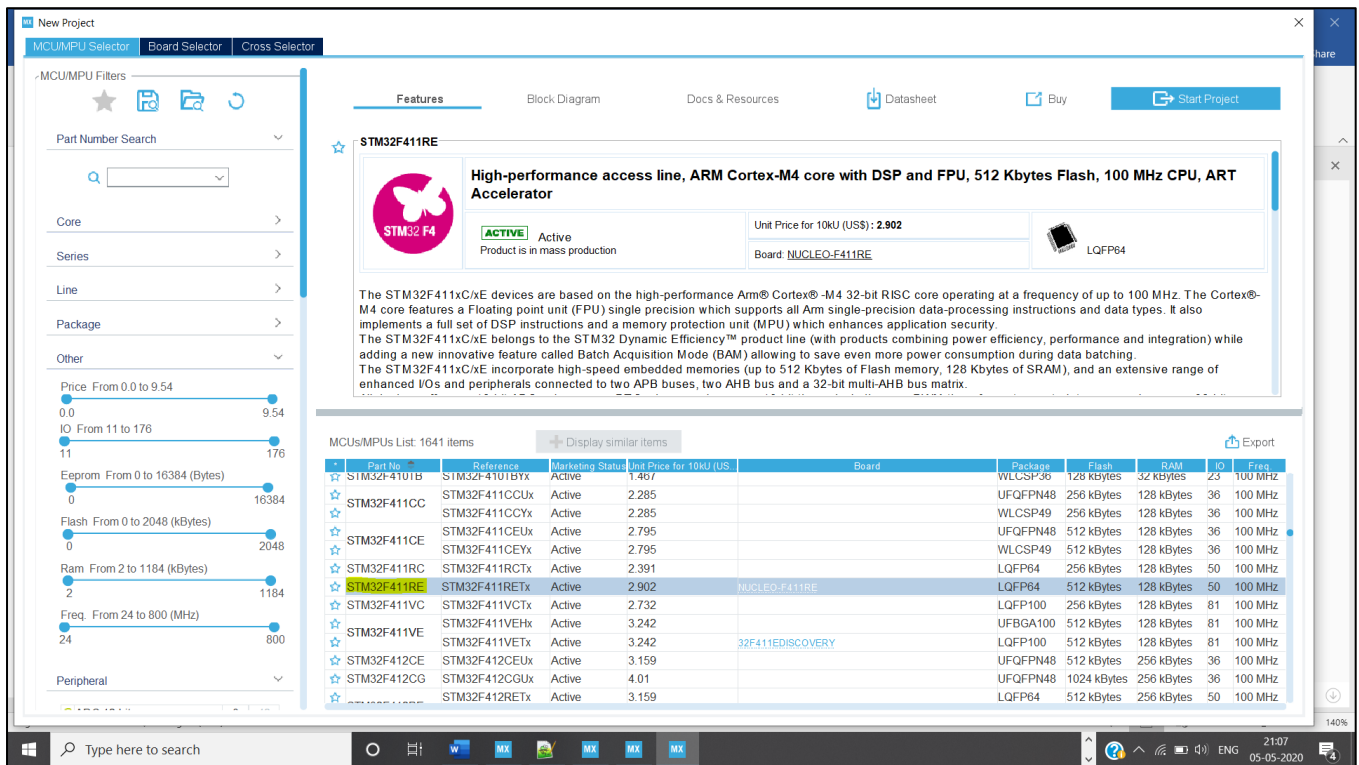


Figure 32: STM32CubeMX - MCU (STM32f411RE) Selection

- Double click on the '**STM32f411RE**' and then a window will appear as shown below with multiple tabs for different configurations.

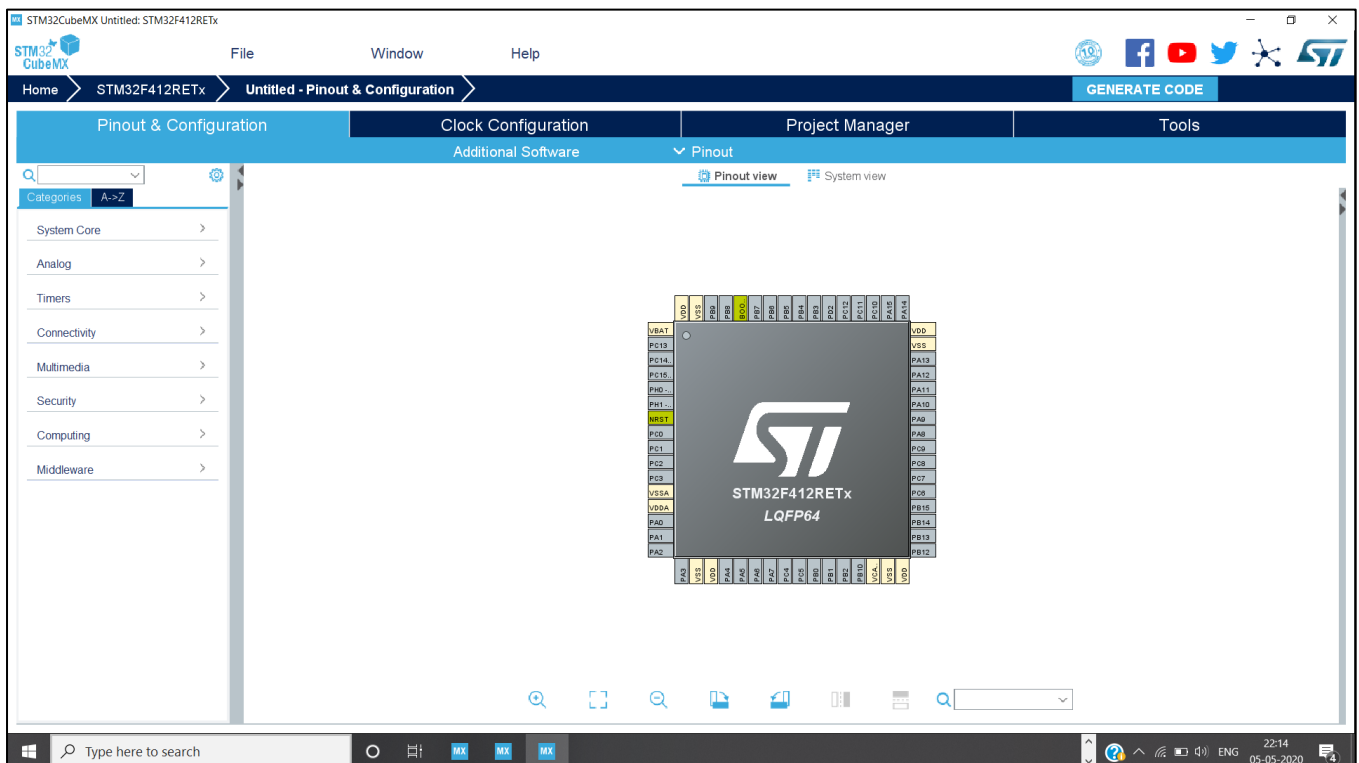


Figure 33: STM32CubeMX - Pinout View

- Click on the 'System view' panel, then the following window will appear as shown below.

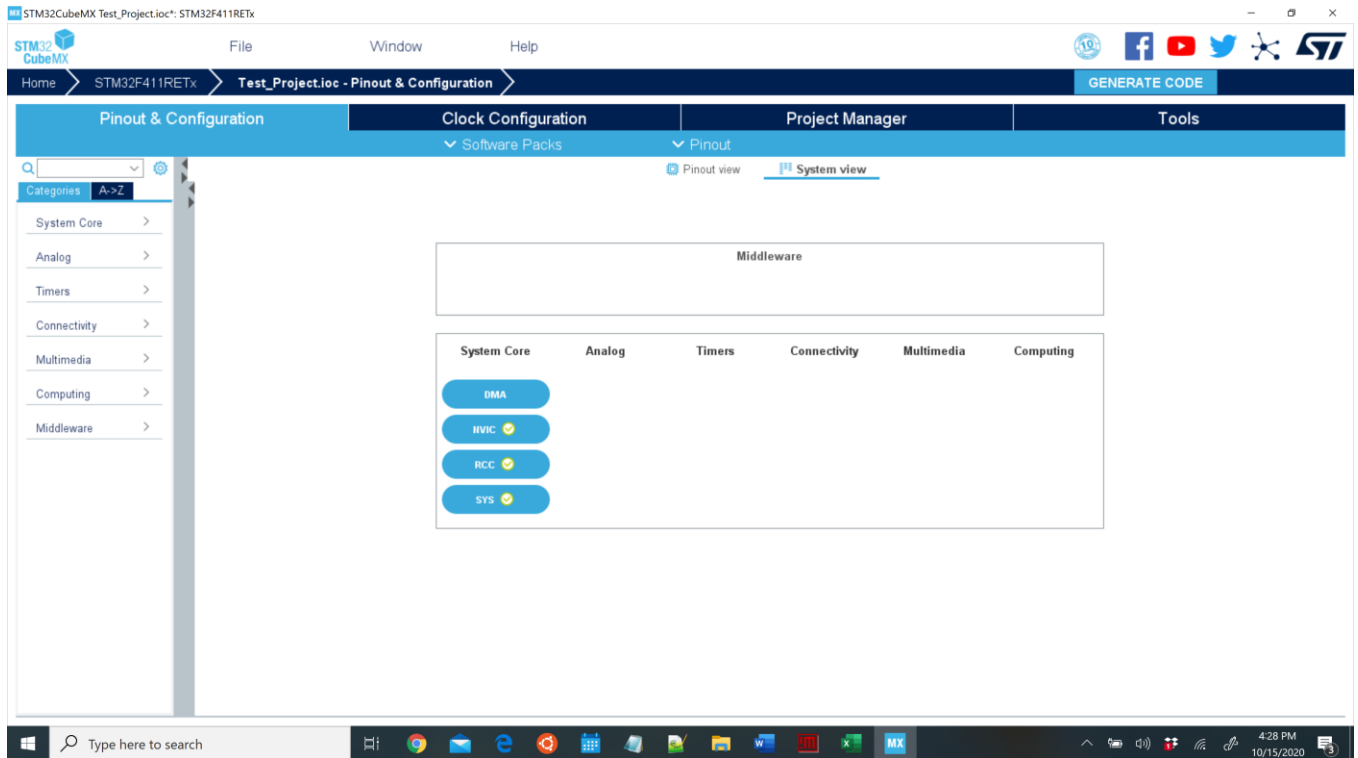


Figure 34: STM32CubeMX - System View

- In the 'SYS' option, select 'TIM1' in 'Timerbase Source' field in the 'Mode' panel, this is recommended by STM32CUBE for FreeRTOS.

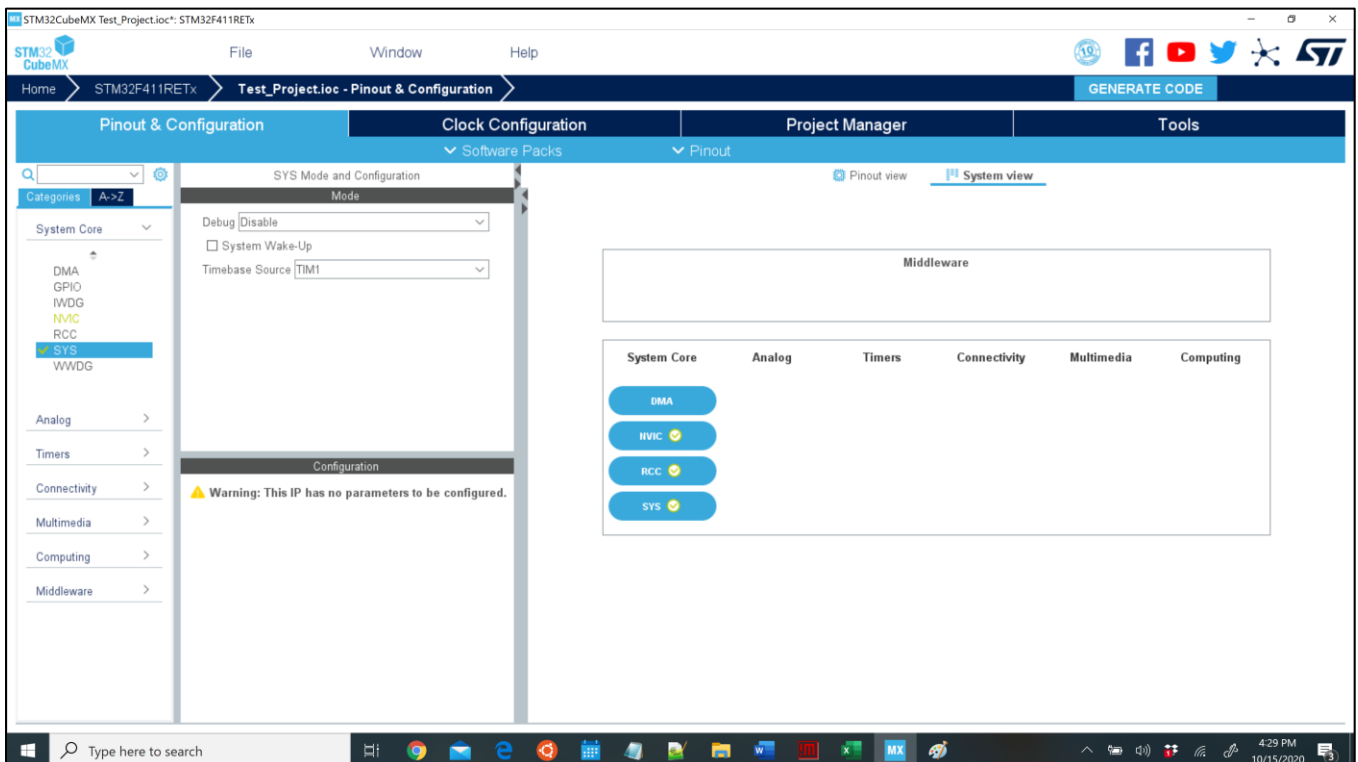


Figure 35: STM32CubeMX - Timerbase Source

- In the 'Middleware' option, choose 'FREERTOS' as shown below.

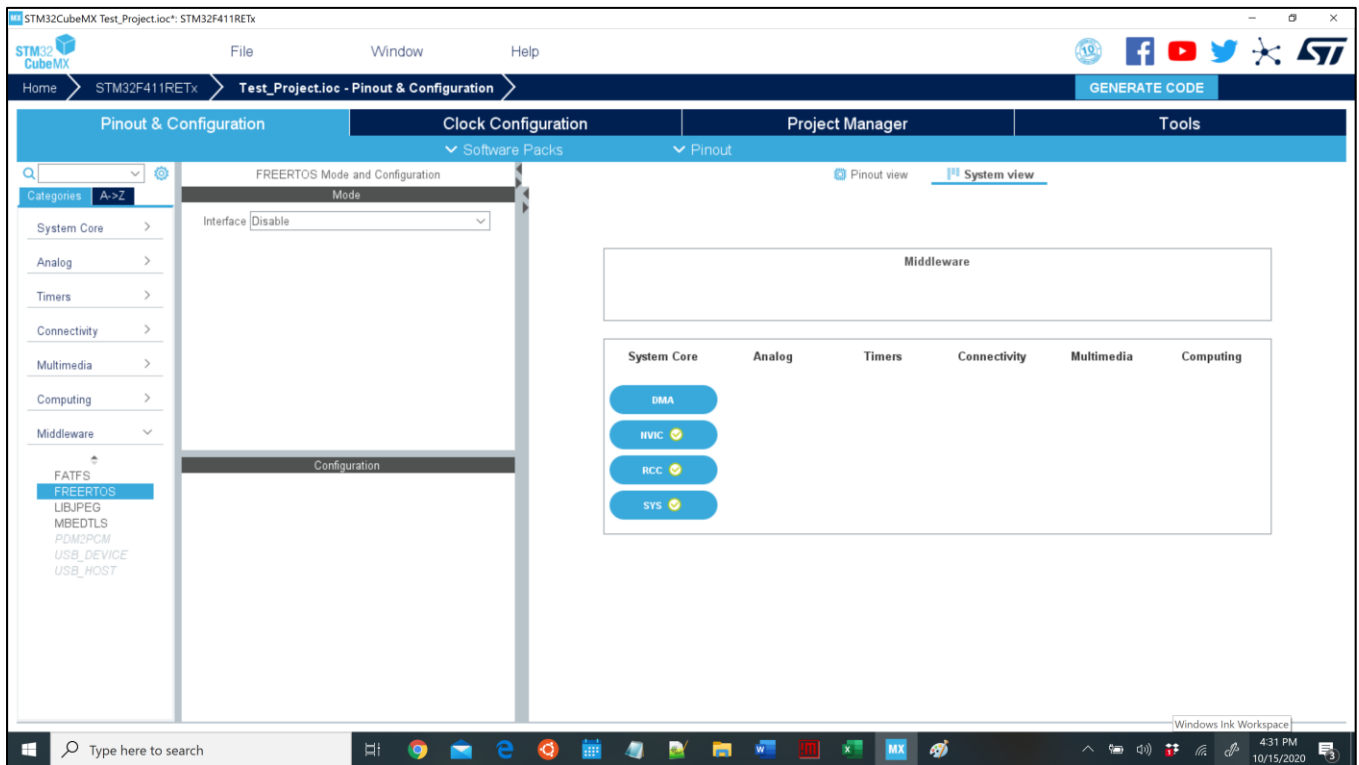


Figure 36: STM32CubeMX - FreeRTOS Selection in Middleware

- In the 'Mode' panel, select 'CMSIS_V2' in the 'Interface' field.

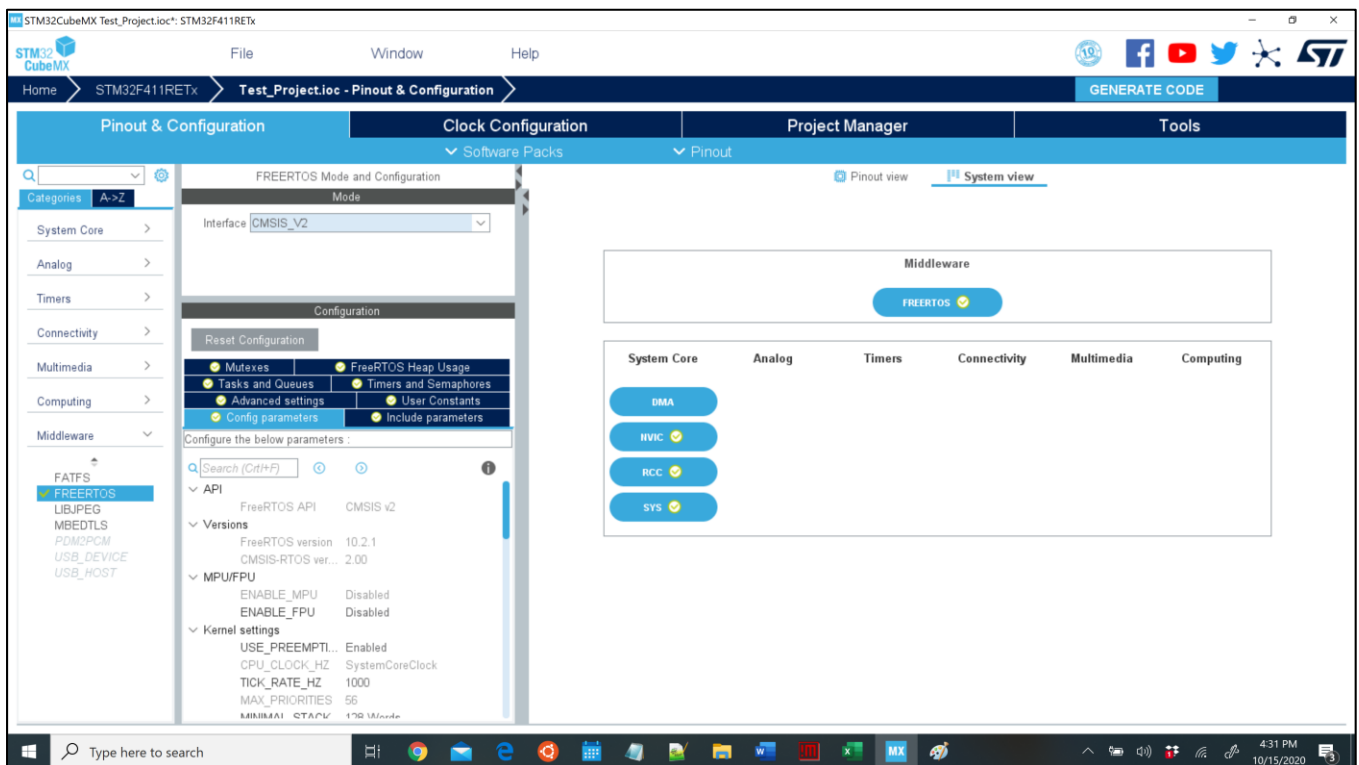


Figure 37: STM32CubeMX - CMSIS_V2 Selection in FreeRTOS Interface

9. Complete settings in the 'System view' panel will be as shown below.

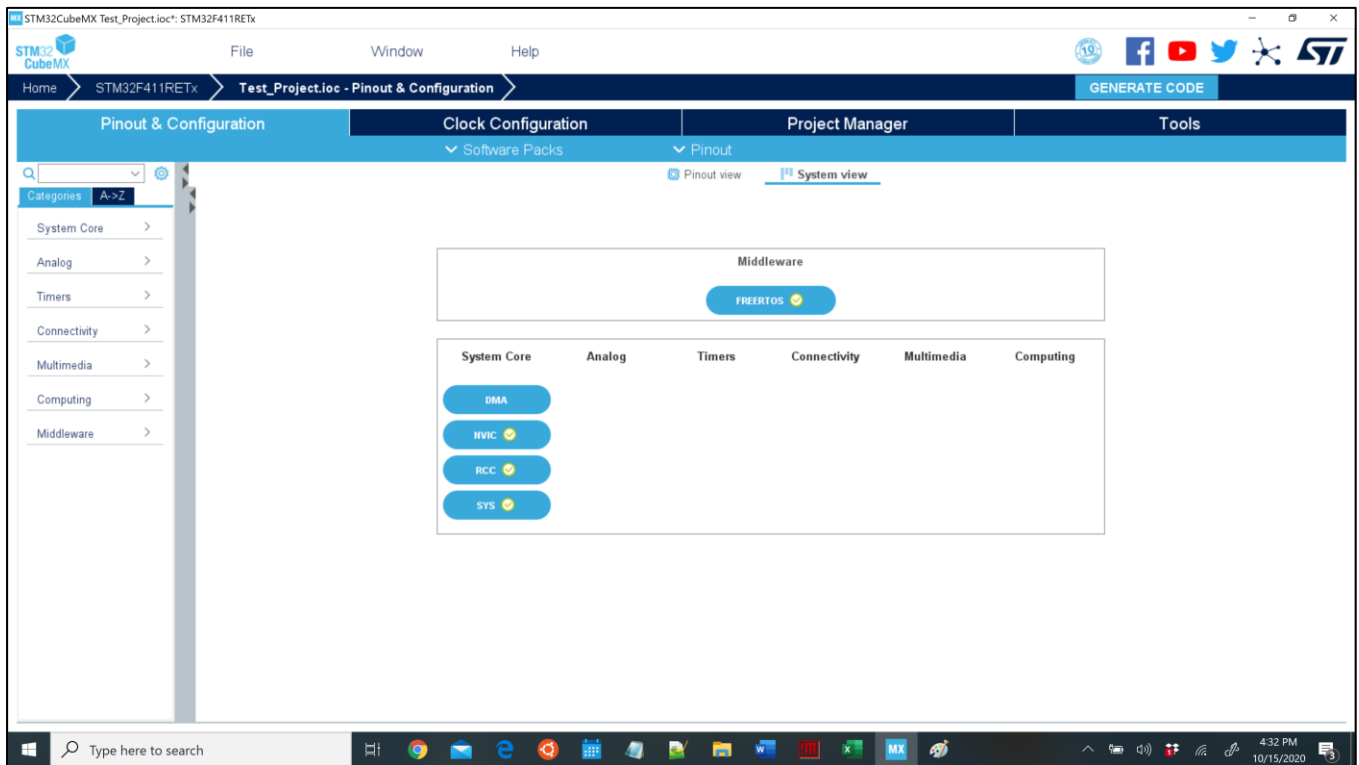


Figure 38: STM32CubeMX - Complete System View

10. After configuring all settings, click on the 'Project Manager' tab. Provide a new project path and project name in settings.

For generating Middlewares for **Keil IDE**, select the '**MDK-ARM**' as 'Toolchain/IDE' and 'Version' as 'V5'.

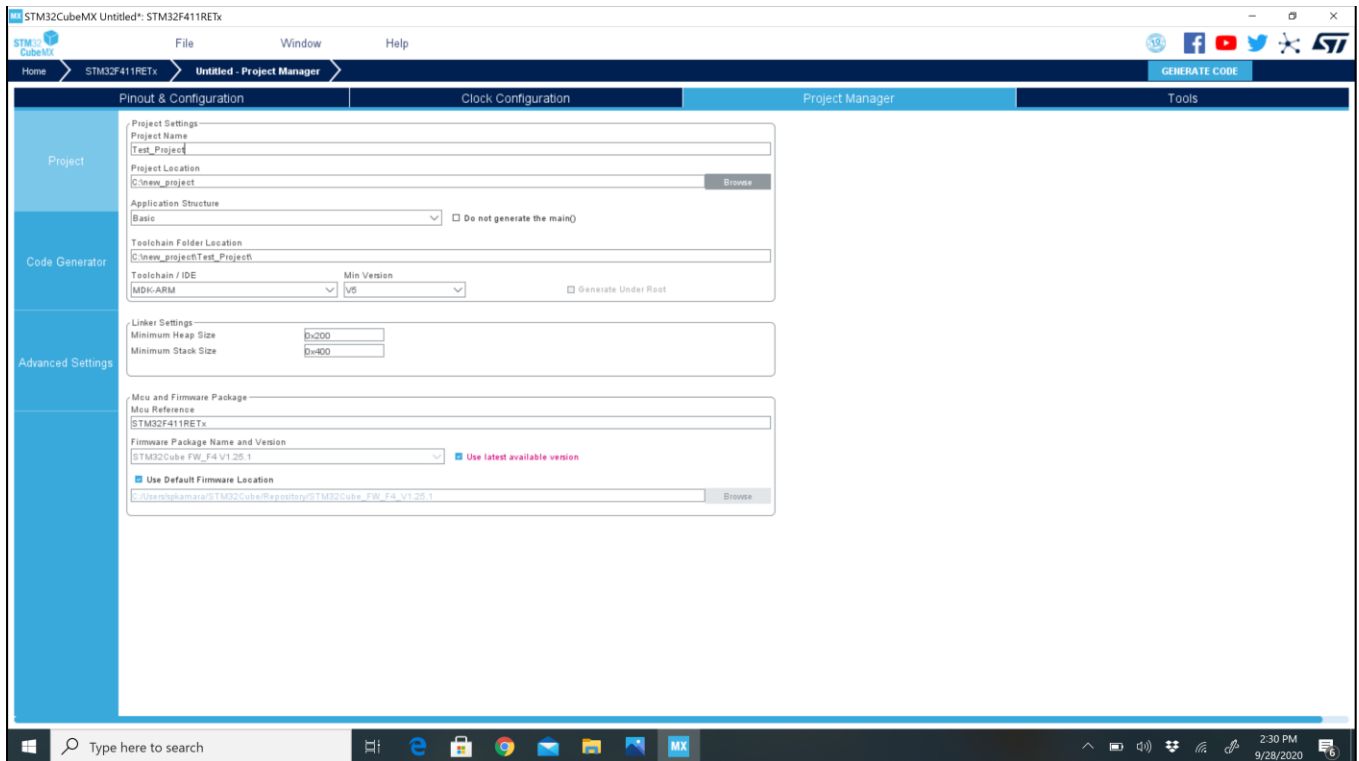


Figure 39: STM32CubeMX - Toolchain Configuration for Keil

For generating Middlewares for **STM32CubeIDE**, select the '**STM32CubeIDE**' as 'Toolchain/IDE'.

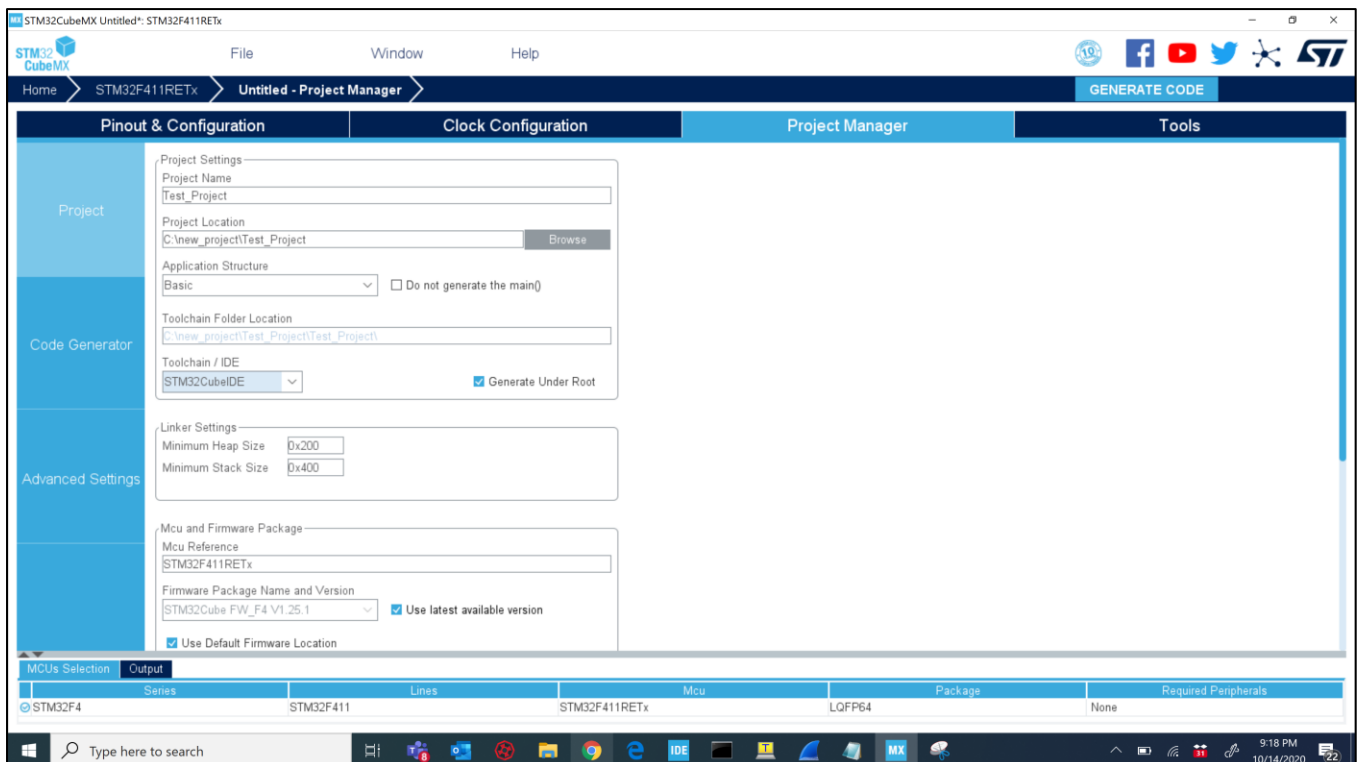


Figure 40: STM32CubeMX - Toolchain Configuration for STM32Cube

11. Now click on the "GENERATE CODE" option to create a project

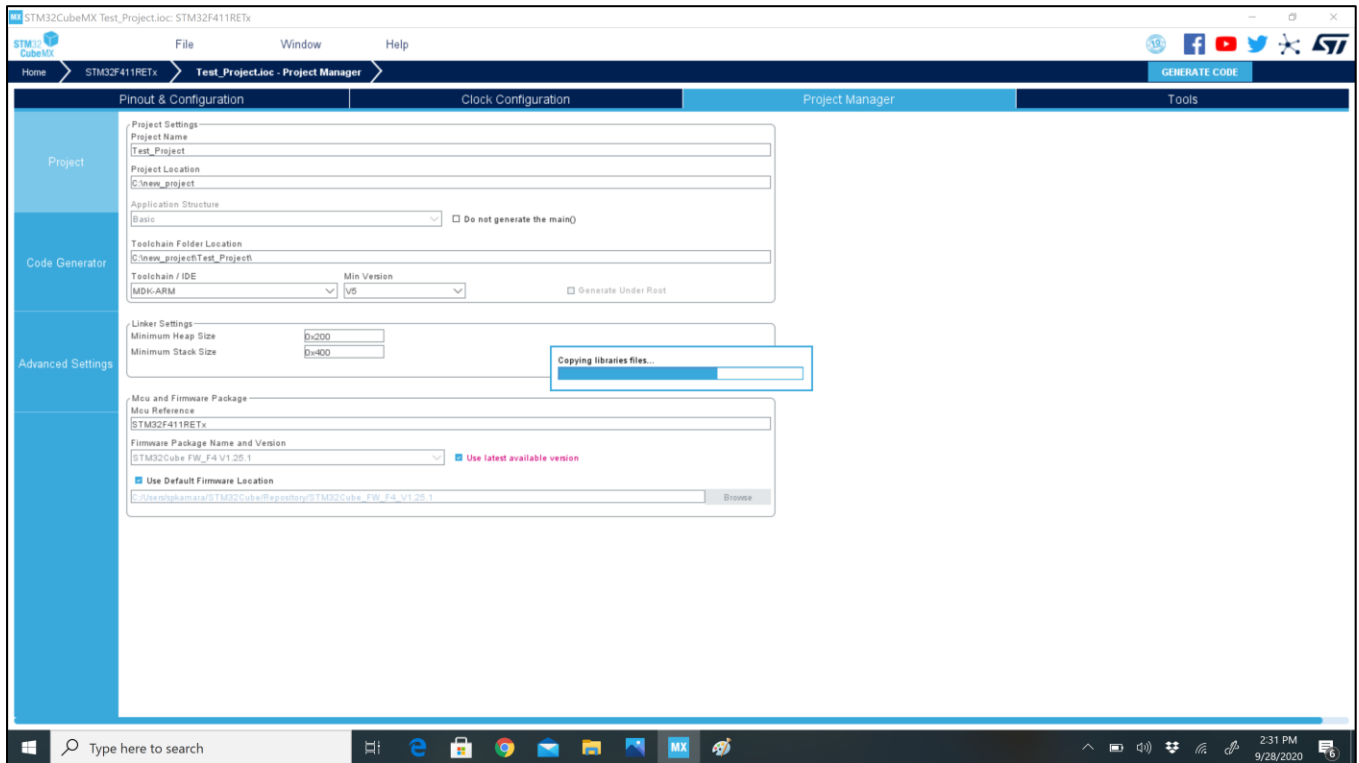


Figure 41: Generate Code

12. After the successful project creation in STM32CubeMX, the following folders will be generated.

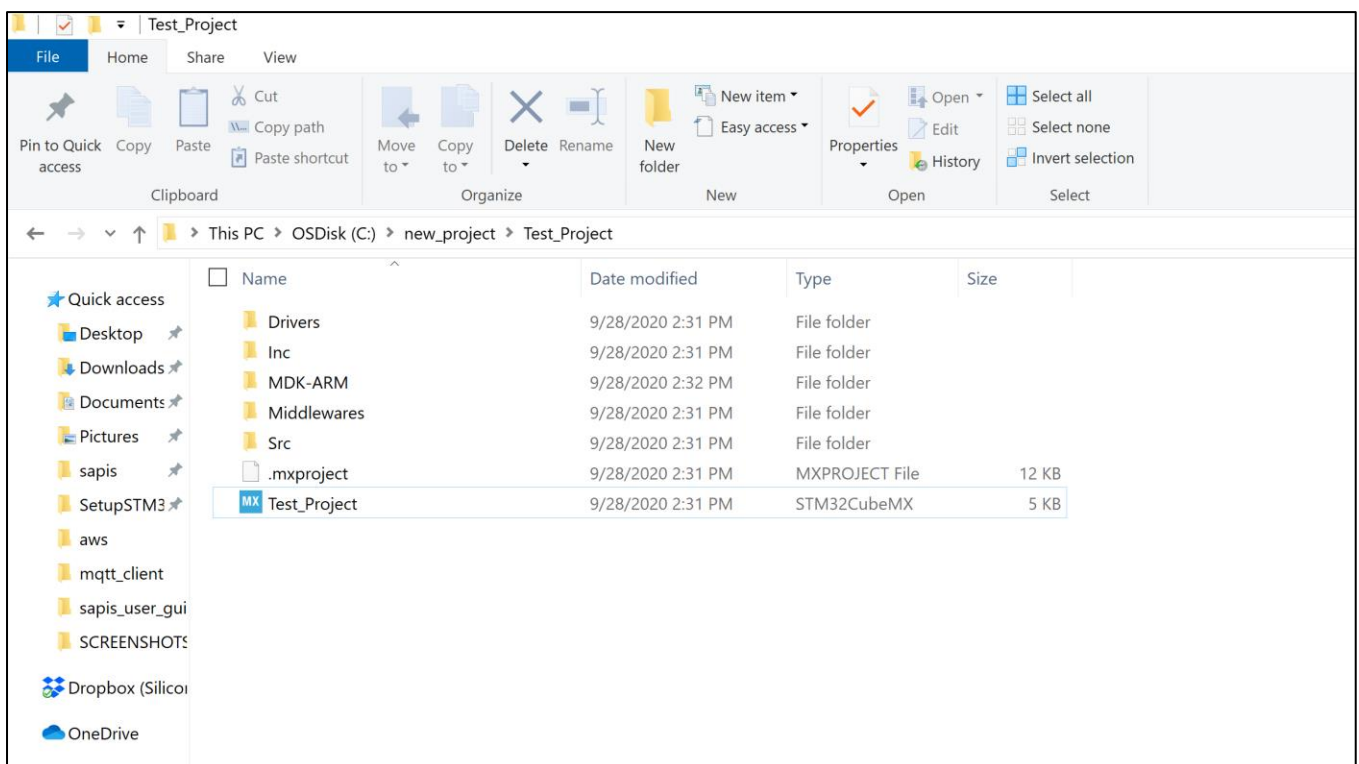


Figure 42: Folders View of Generated Code

13. For **Keil**, Copy the '**Middlewares**' folder to '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Keil_Freertos**' in the release package directory.

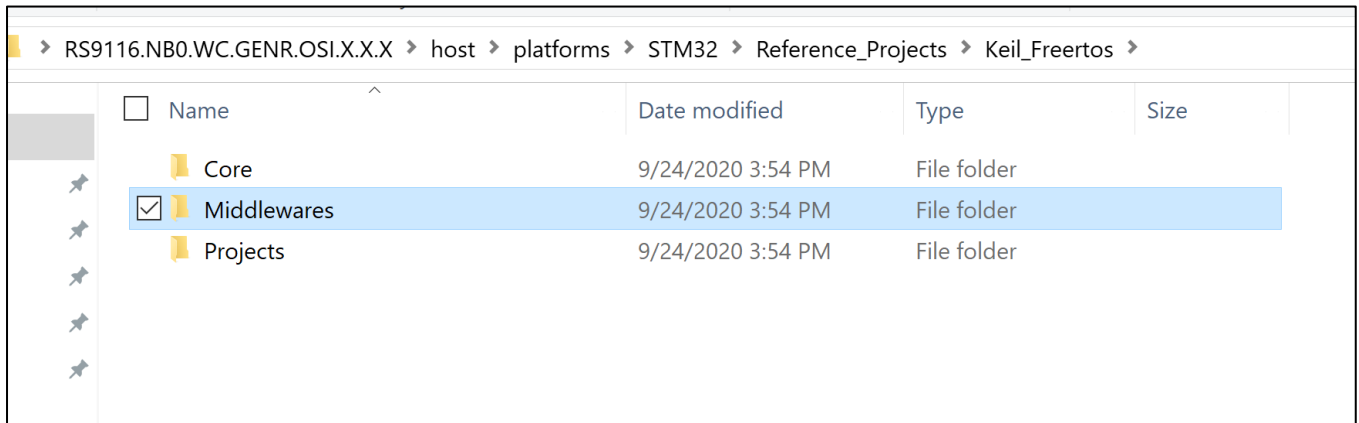


Figure 43: Keil - Copy Middlewares Folder to Project Space

For **STM32Cube**, copy the '**Middlewares**' folder to '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Cube_Freertos**' in the release package directory.

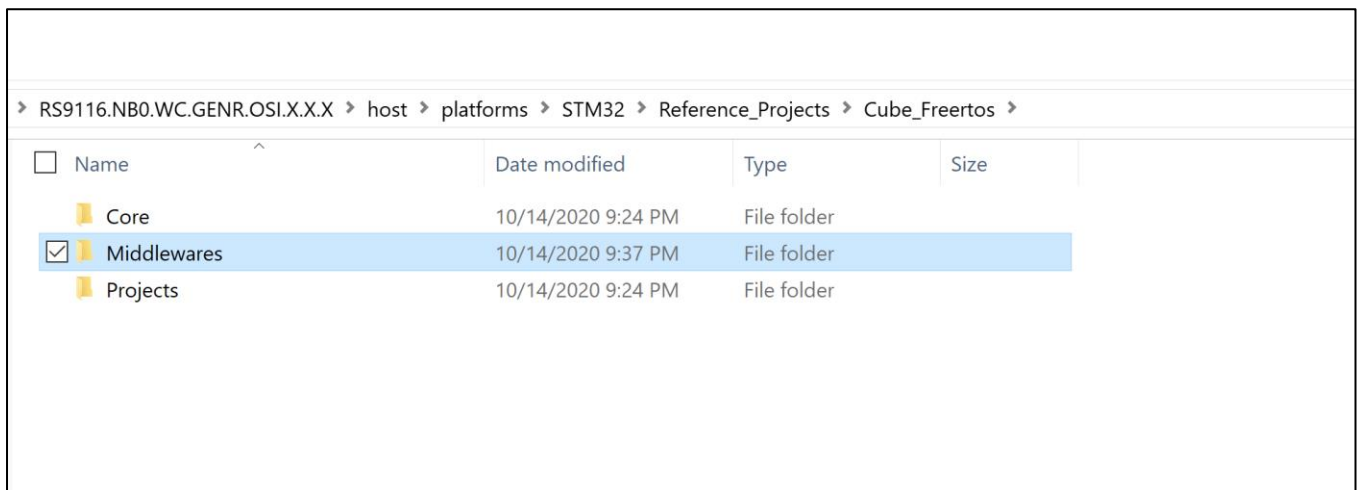


Figure 44: STM32Cube - Copy Middlewares Folder to Project

14. To Compile the Cube Freertos projects, navigate to '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Cube_Freertos\Projects\SPI\sample_project**', double click on '.project' and compile using STM32CubeIDE.

Here, the 'sample_project' is used as a reference only. Users can select any project present at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Cube_Freertos\Projects\SPI**'.

To compile the Keil Freertos project, navigate to '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI\mqtt_client**' and double-click on 'mqtt_client' Keil project.

Here, the 'mqtt_client' project is used as a reference only. Users can select any project present at '**RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\SPI**'.

\ or
'RS9116.NB0.WC.GENR.OSI.X.X.X\host\platforms\STM32\Reference_Projects\Keil_Freertos\Projects\UA
RT'.

15. Keil IDE will be launched with the 'mqtt_client' project displayed in the 'Project' window.

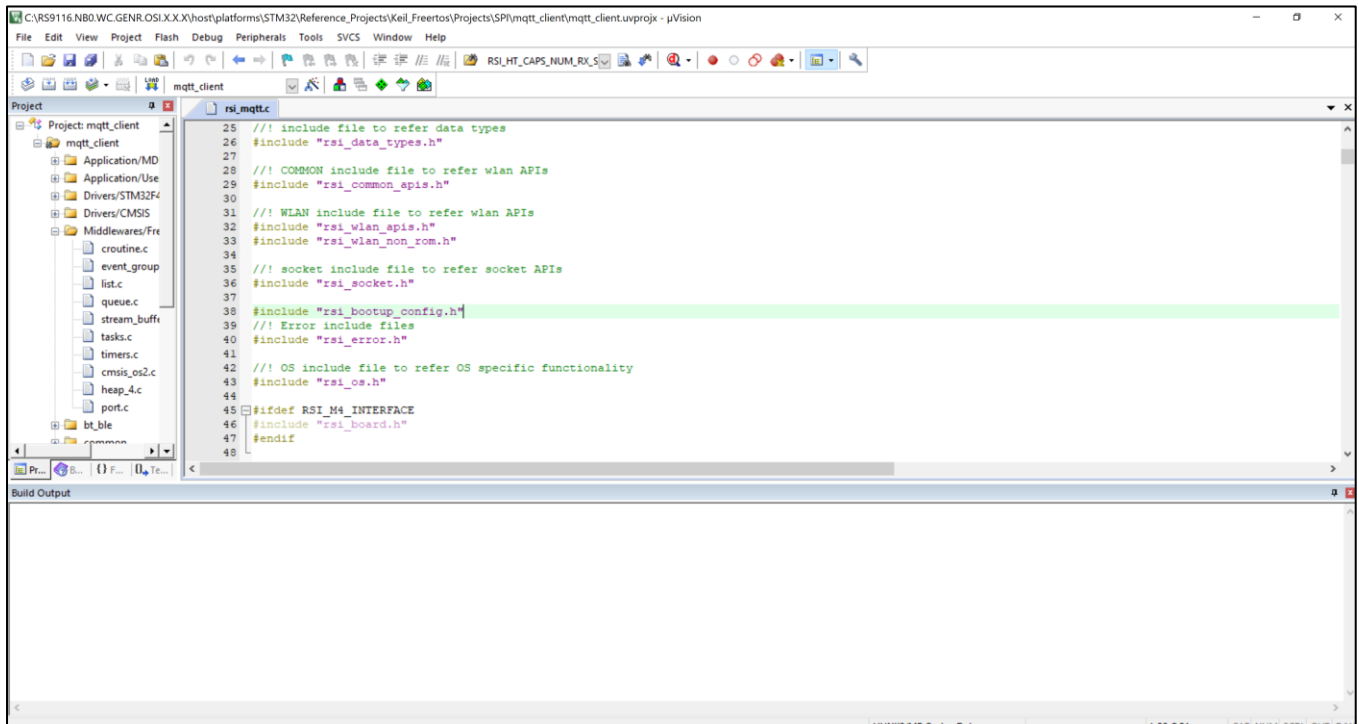


Figure 45: Project Windows

16. Click the 'Build' icon to compile the 'mqtt_client' project.

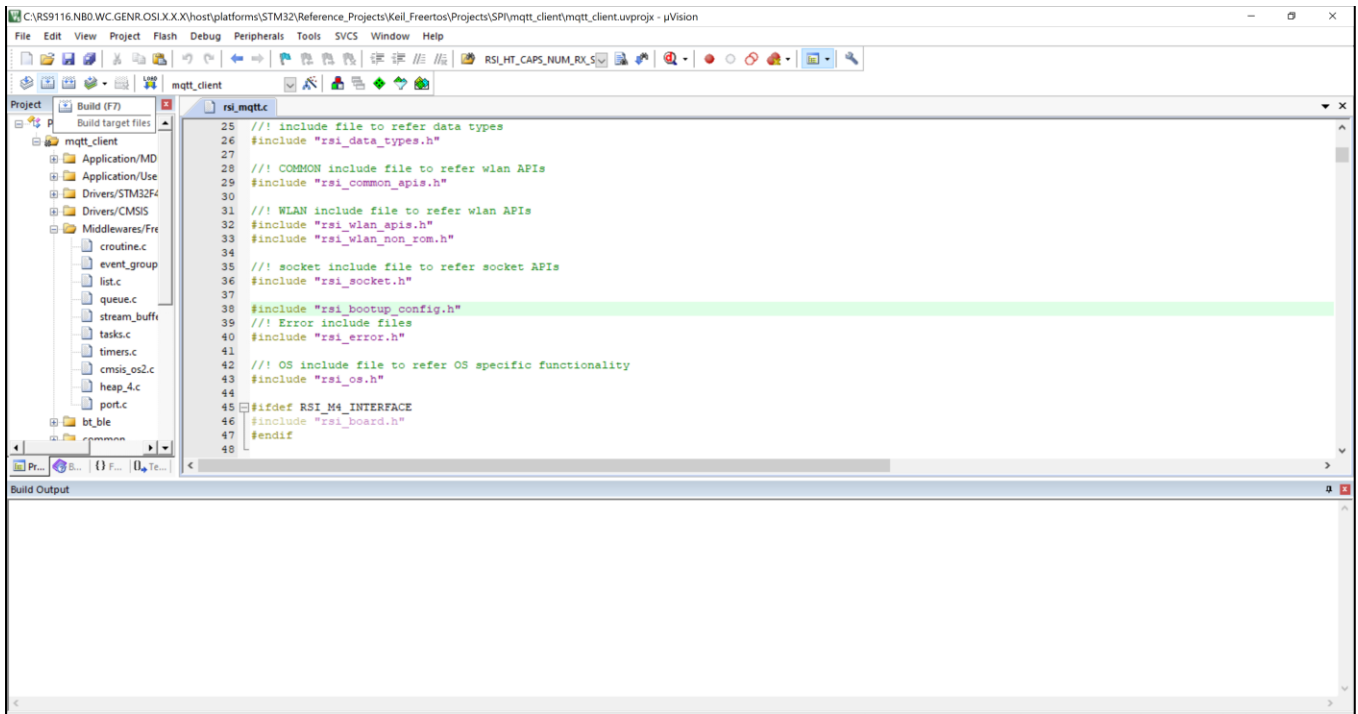


Figure 46: Build Project

17. After successful compilation, the following log messages are displayed in the 'Build Output' window.

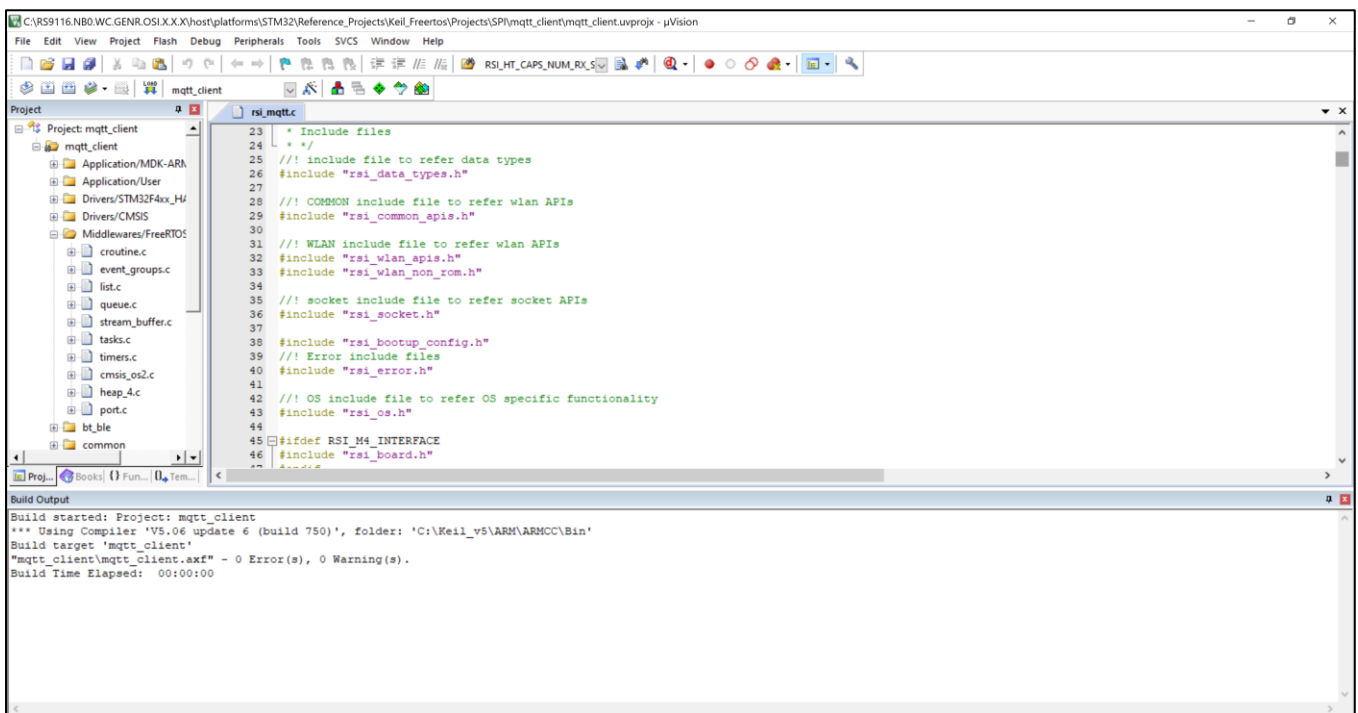


Figure 47: Compilation Success

Follow steps mentioned in STM32 UserGuide to bring up STM32 and RS9116W setup over the SPI interface. Connect STM32 to PC via USB where the above Keil project is compiled. Make sure ST-LINK USB drivers are installed.

18. Click on "options for target" in Keil.

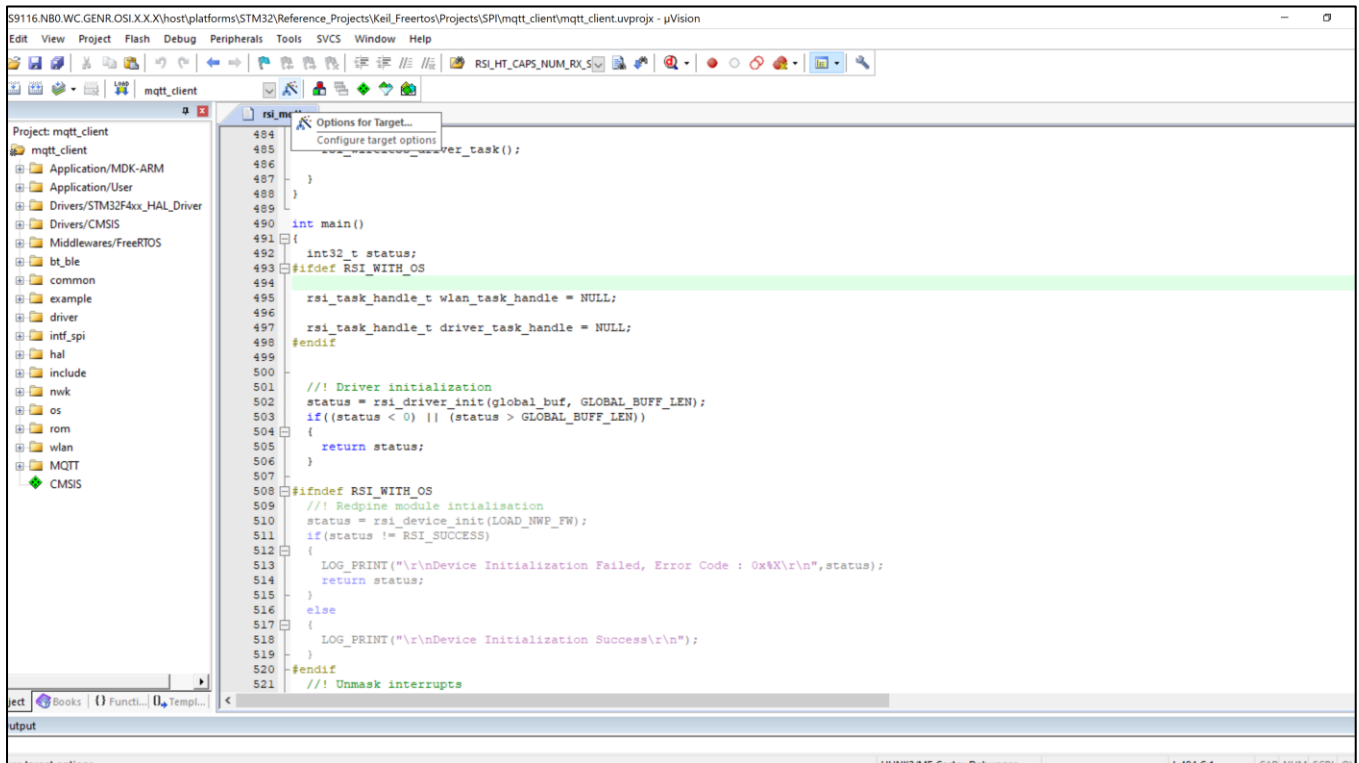


Figure 48: Options Selection

19. Click on the Device tab and make sure that the below DFP package is selected.

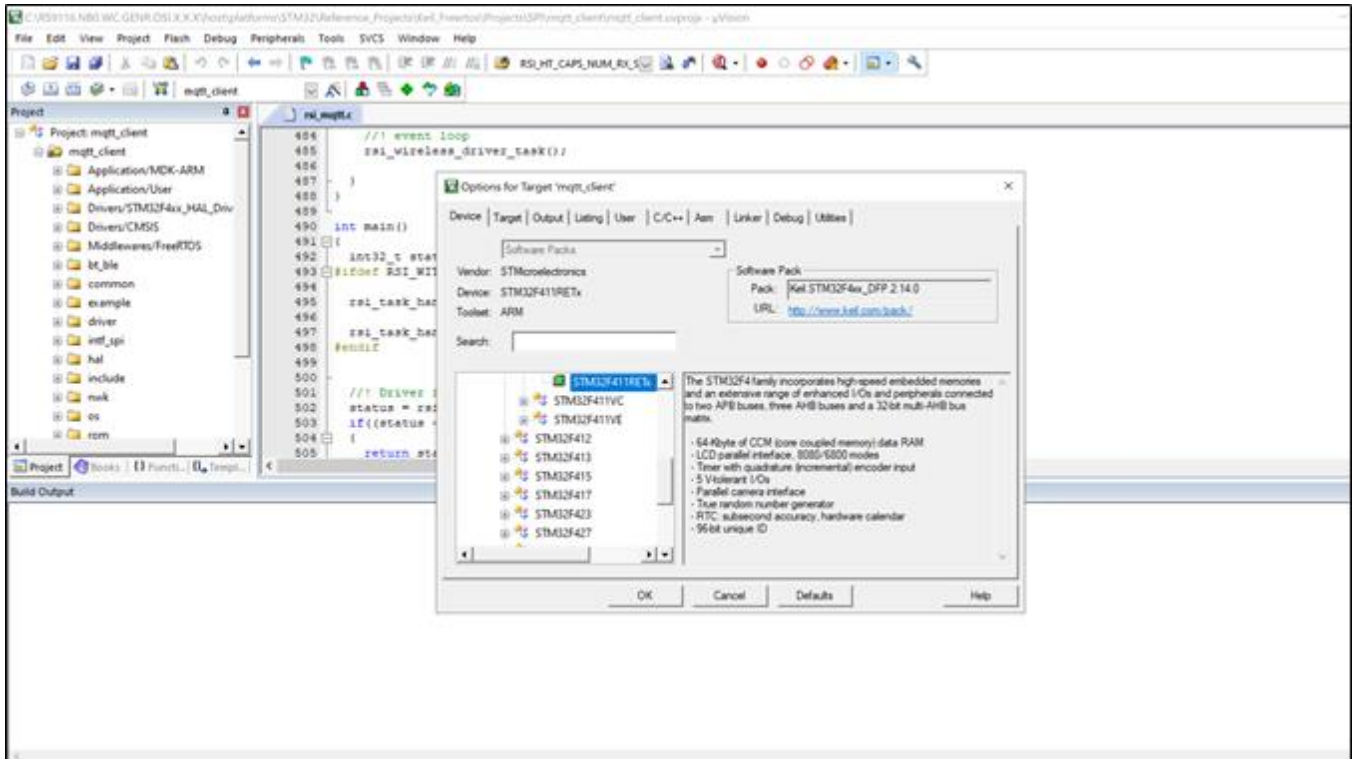


Figure 49: DFP Package

20. Click on the Debug tab and make sure that ST-Link Debugger is selected.

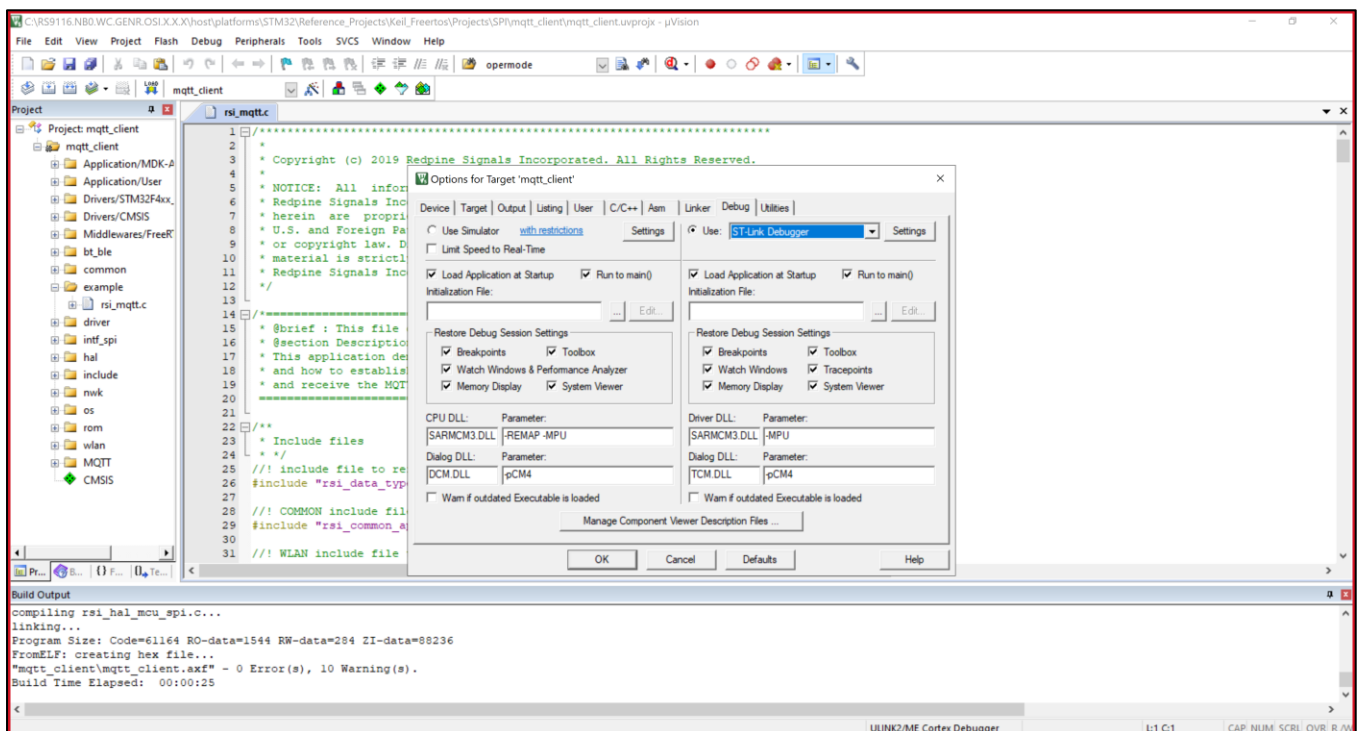


Figure 50: ST-Link Debugger selection

21. Check debugger detection status, it should detect like below.

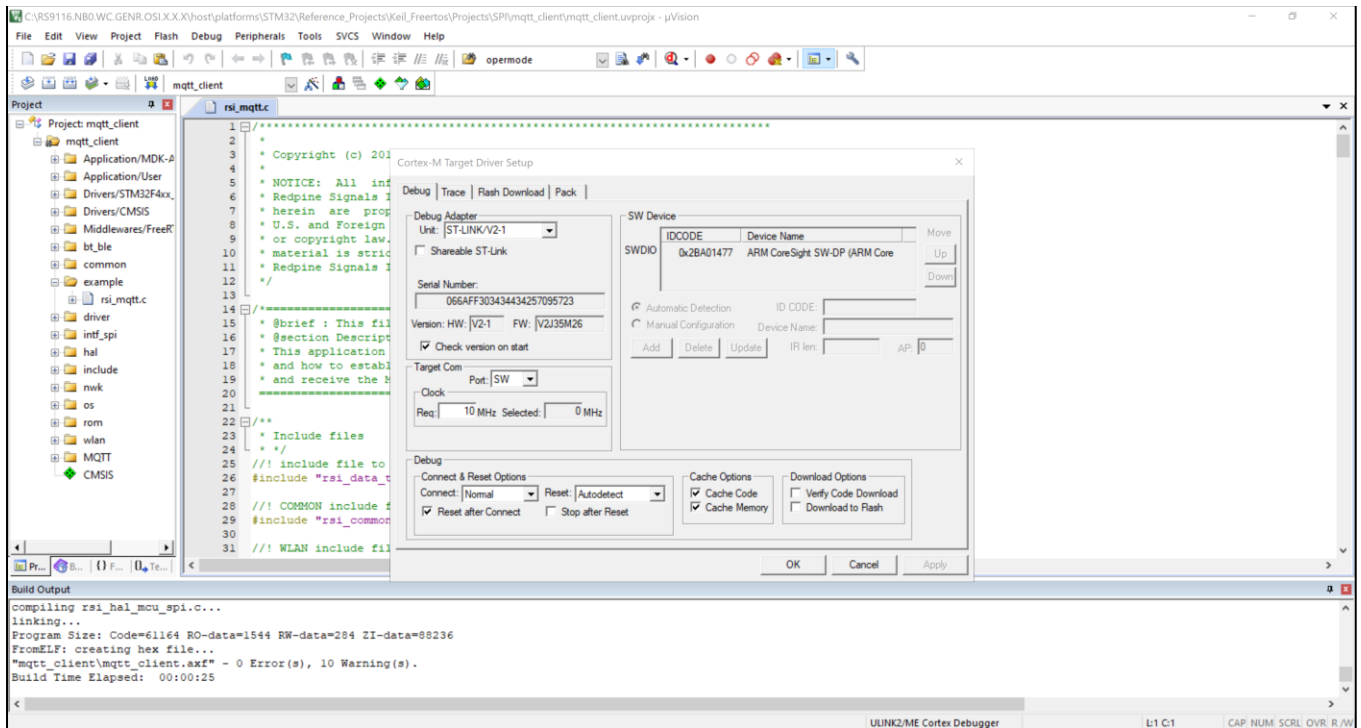


Figure 51: Debug Detection

22. Click on the Flash Download tab and make sure the below settings are configured.

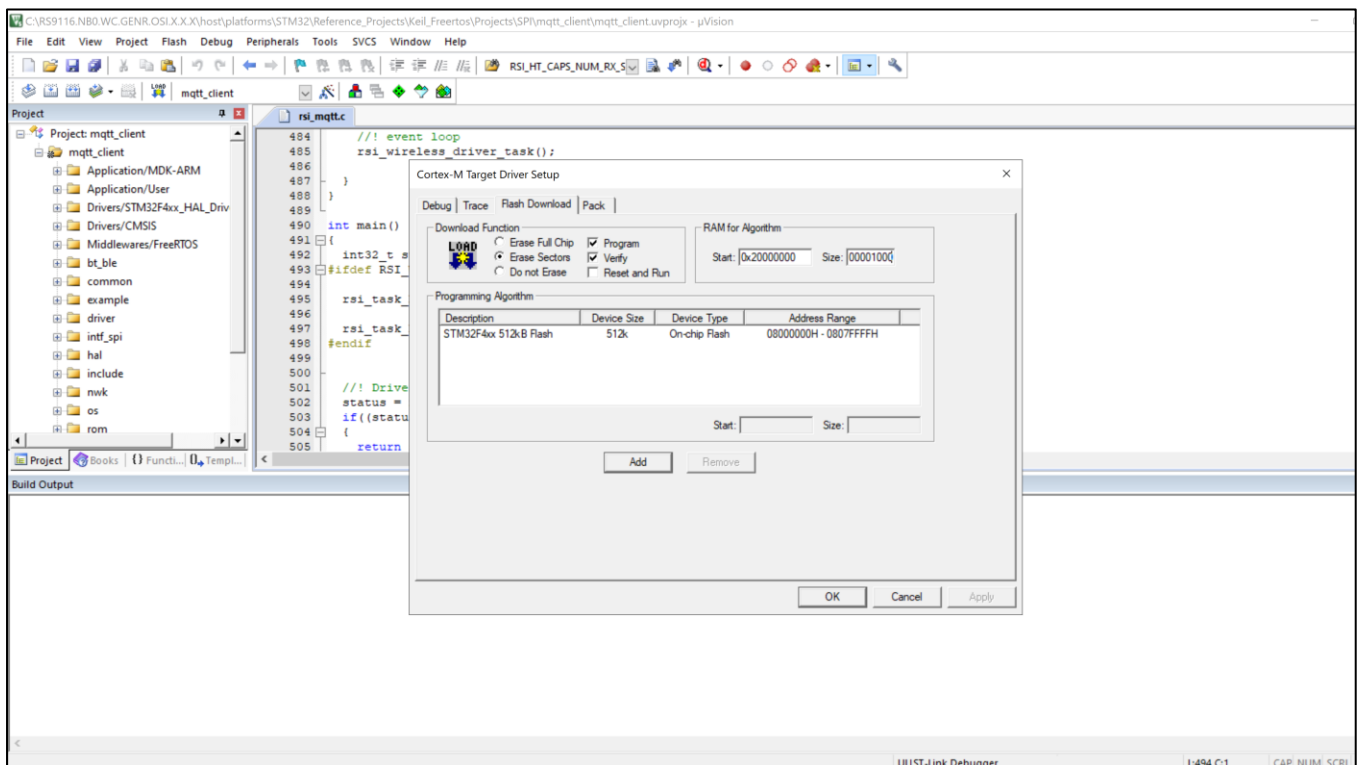


Figure 52: Flash Download

23. Click the 'Download and Debug' icon to flash the compiled binary to the STM32 platform.

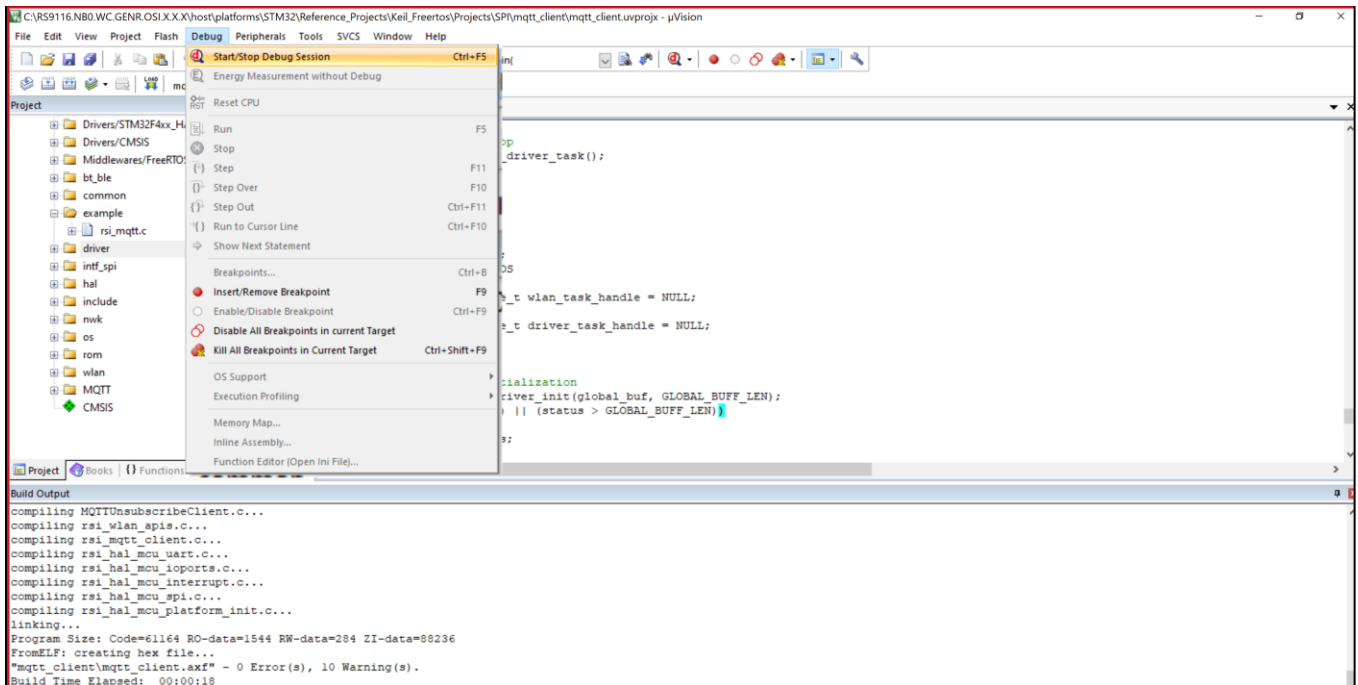


Figure 53: Download and Debug

24. After successful download, project execution control will be at the main() as shown below.

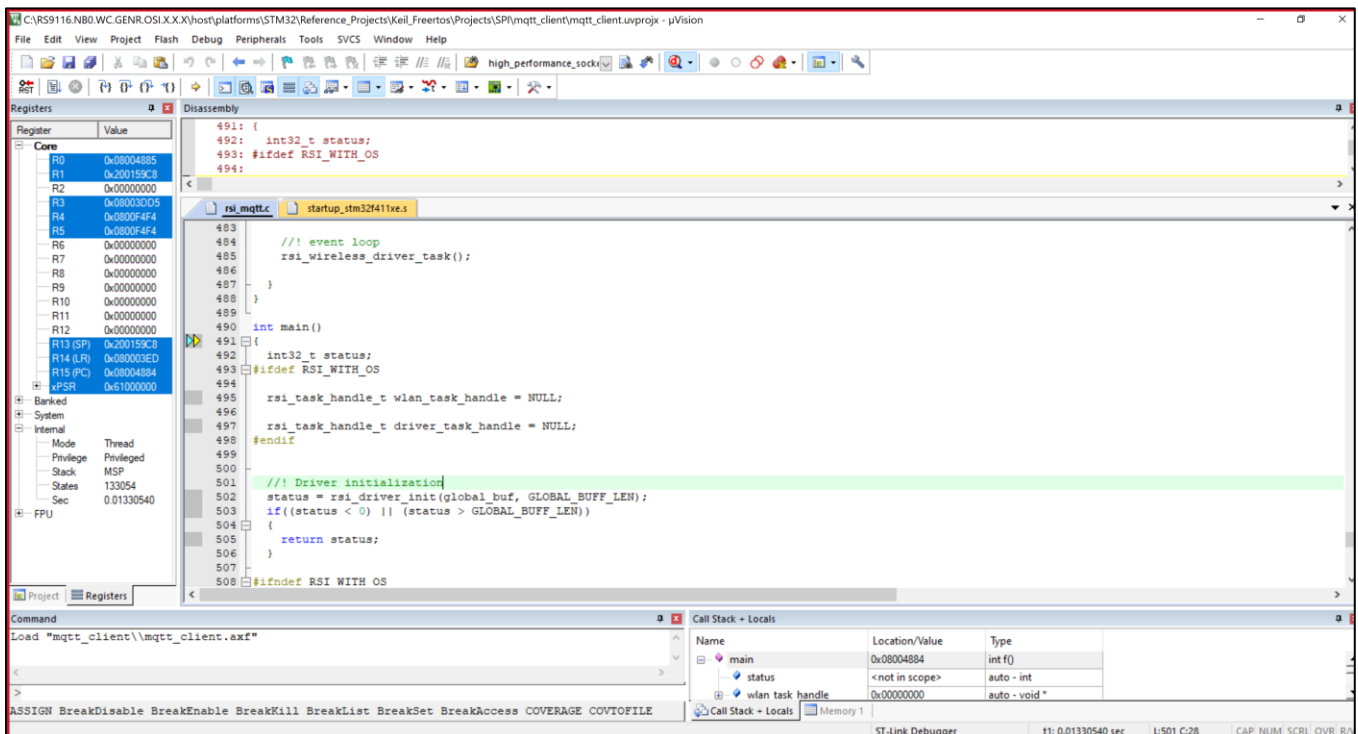


Figure 54: Control at Main

25. Execute the application by clicking the run option.

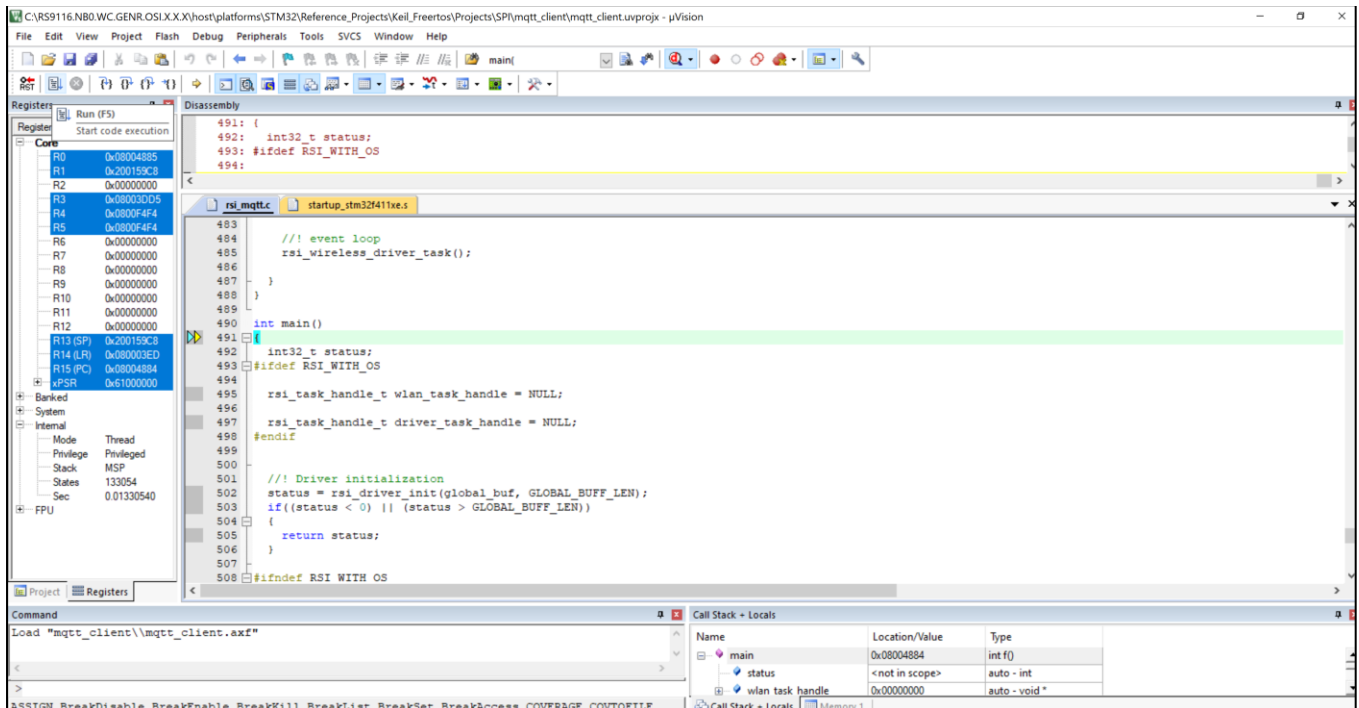


Figure 55: Execute Project

26. Refer to section **Steps for executing STM32 examples using Master Application (Sample Project)** in UG454: RS9116W with STM32 User's Guide.pdf at <https://docs.silabs.com/rs9116> for executing project.

5 AWS SDK Porting for RS9116W SAPIs in STM32

This section describes the steps for porting AWS SDK to RS9116W SAPI library.

Introduction

- **AWS IoT** Core is a cloud platform which connects devices across **AWS** cloud services. **AWS IoT** provides a interface which allows the devices to communicate securely and reliably in bi-directional ways to the **AWS** touch-points, even when the devices are offline.
- The AWS IoT Device SDK allow applications to securely connect to the AWS IoT platform. It includes an [MQTT 3.1.1 client](#), as well as libraries specific to AWS IoT, such as [Thing Shadows](#).
- It is distributed in source form and may be build into firmware along with application code.
- AWS SDK can be ported to different platforms.

Block Diagram

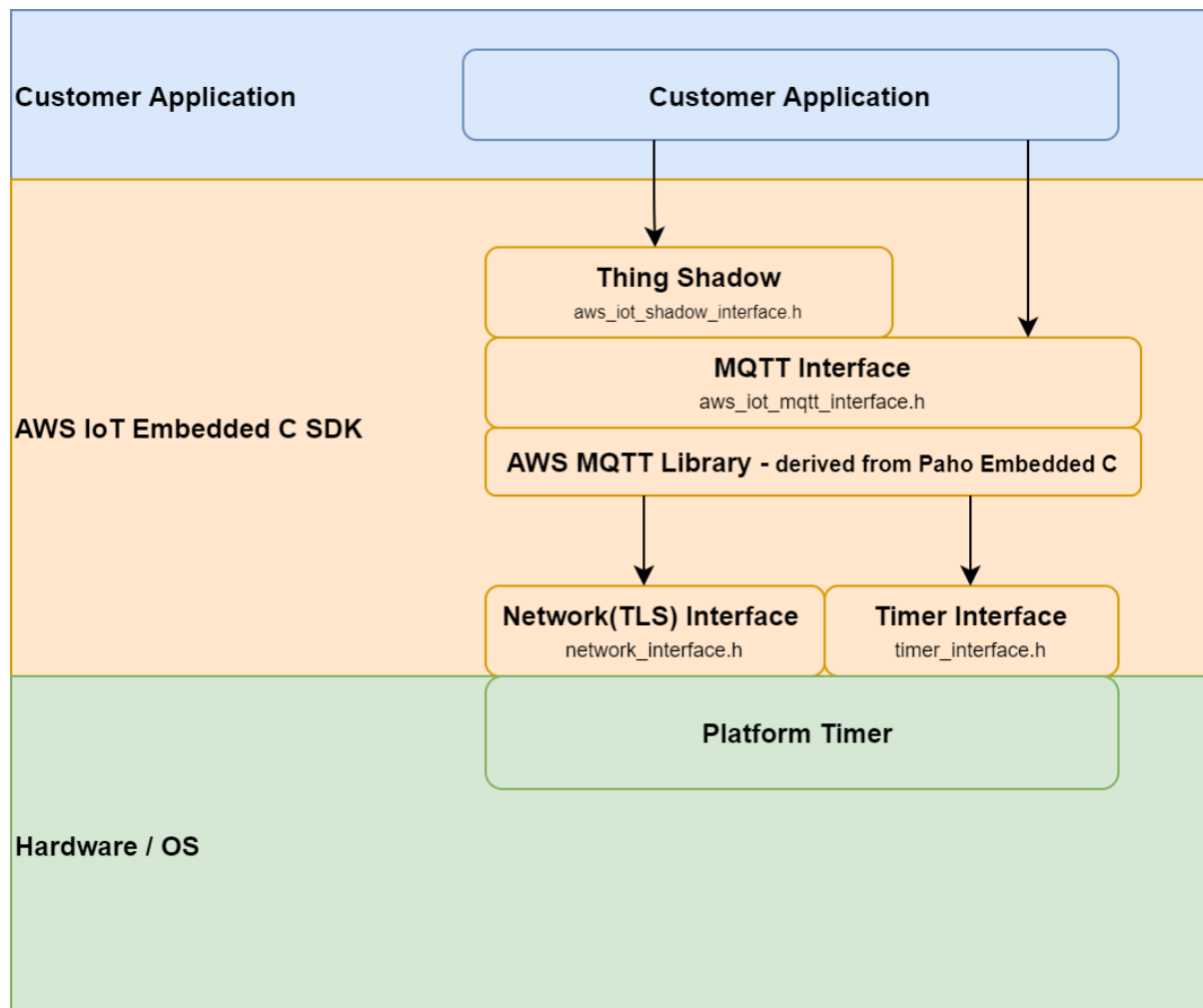


Figure 56: AWS IOT SDK Block Diagram

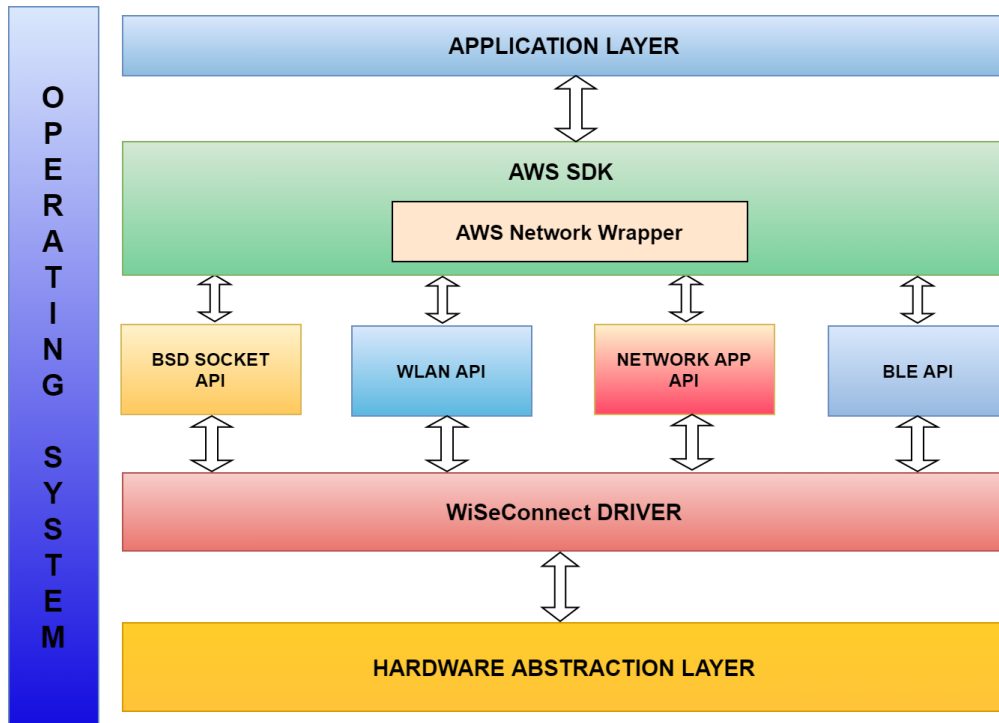


Figure 57: AWS Porting Block Diagram

AWS SDK Porting Steps

1. Download AWS SDK from <https://github.com/aws/aws-iot-device-sdk-embedded-C/tree/v3.1.0>

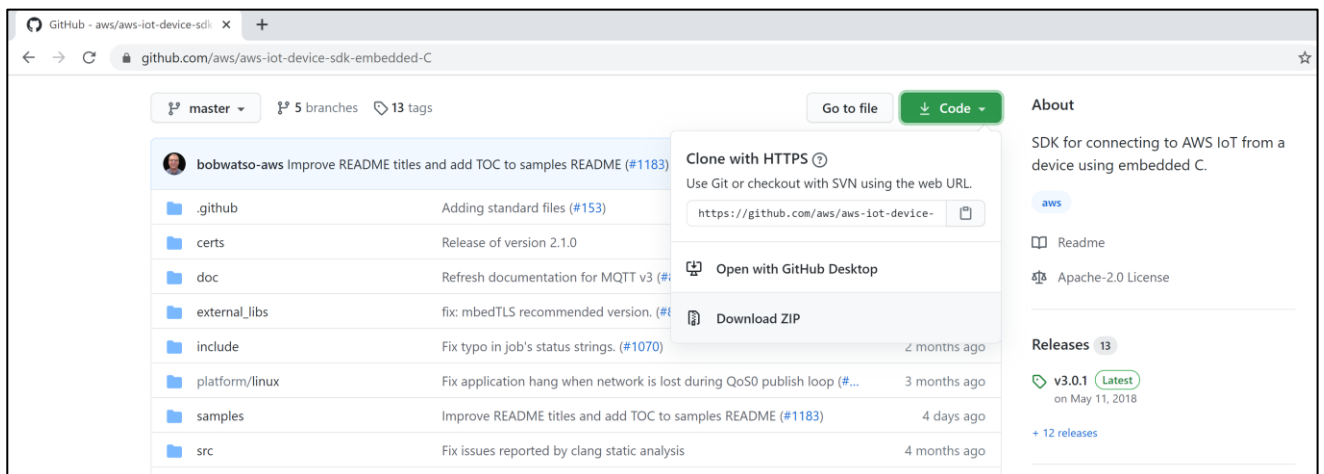


Figure 58: Download Package

2. Extract (Extracted folder would be visible as aws-iot-device-sdk-embedded-C-master)

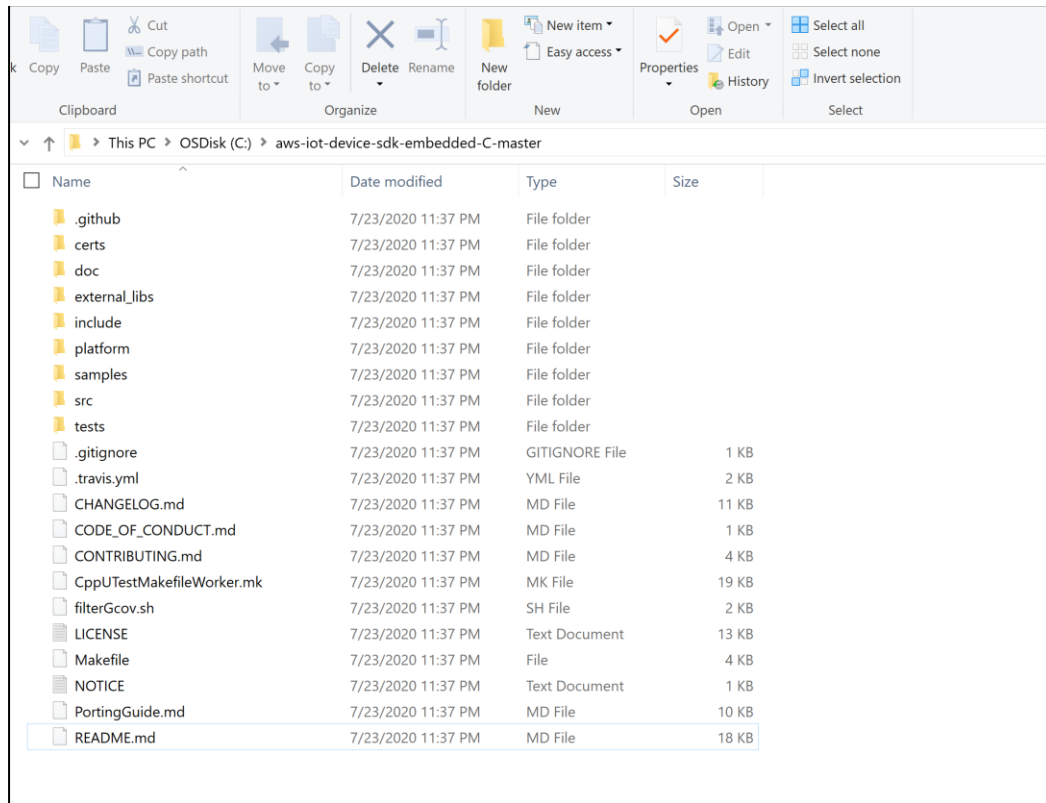


Figure 59: Extract aws-iot-device-sdk-embedded-C-master Package

- Download and Extract the WiSeConnect Release Package (RS9116.NB0.WC.GENR.OSI.XXX)
- Copy **include**(except timer_interface.h, threads_interface.h, network_interface.h) and **src** folders present in the sdk folder(aws-iot-device-sdk-embedded-C-master) to release package (host\sapis\nwk\applications\aws_sdk).

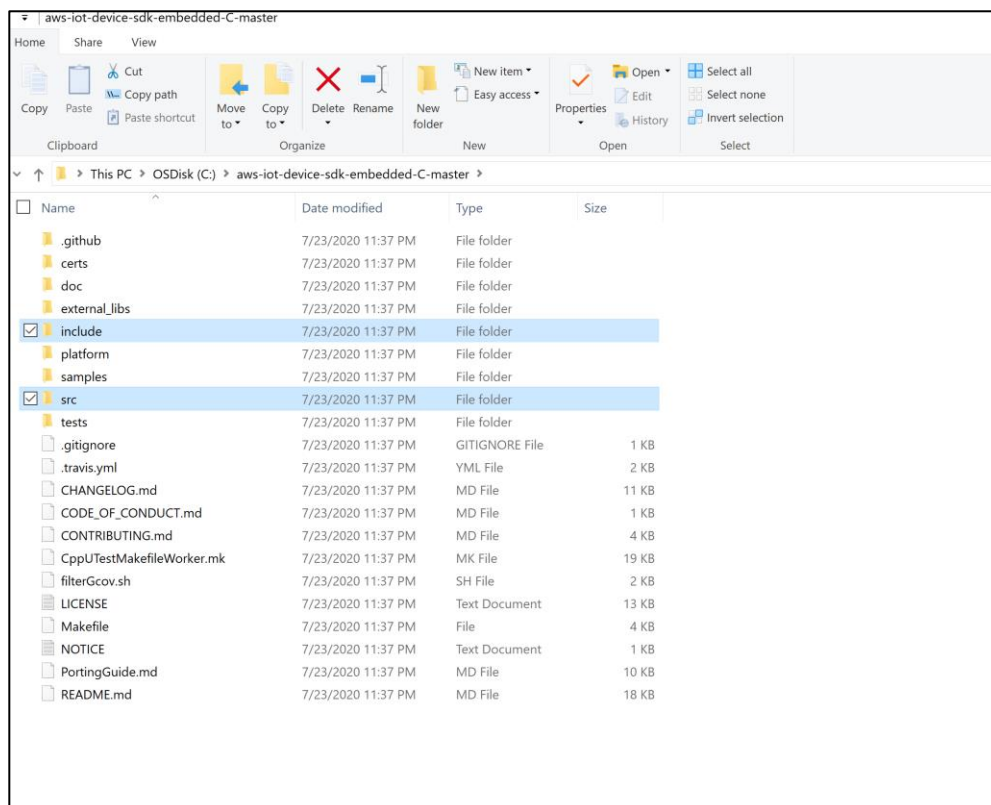


Figure 60: Copy Src and Include Folders

5. Copy jsnm.c file present at aws-iot-device-sdk-embedded-C-master\external_libs\jsnm to the location host\sapis\nwk\applications\aws_sdk\src in the RS9116W release package.

Copy aws-iot-device-sdk-embedded-C-master\external_libs\jsnm\jsnm.h file to the location \Libraries\Wireless_Library\nwk\applications\aws_sdk\include in the release package.

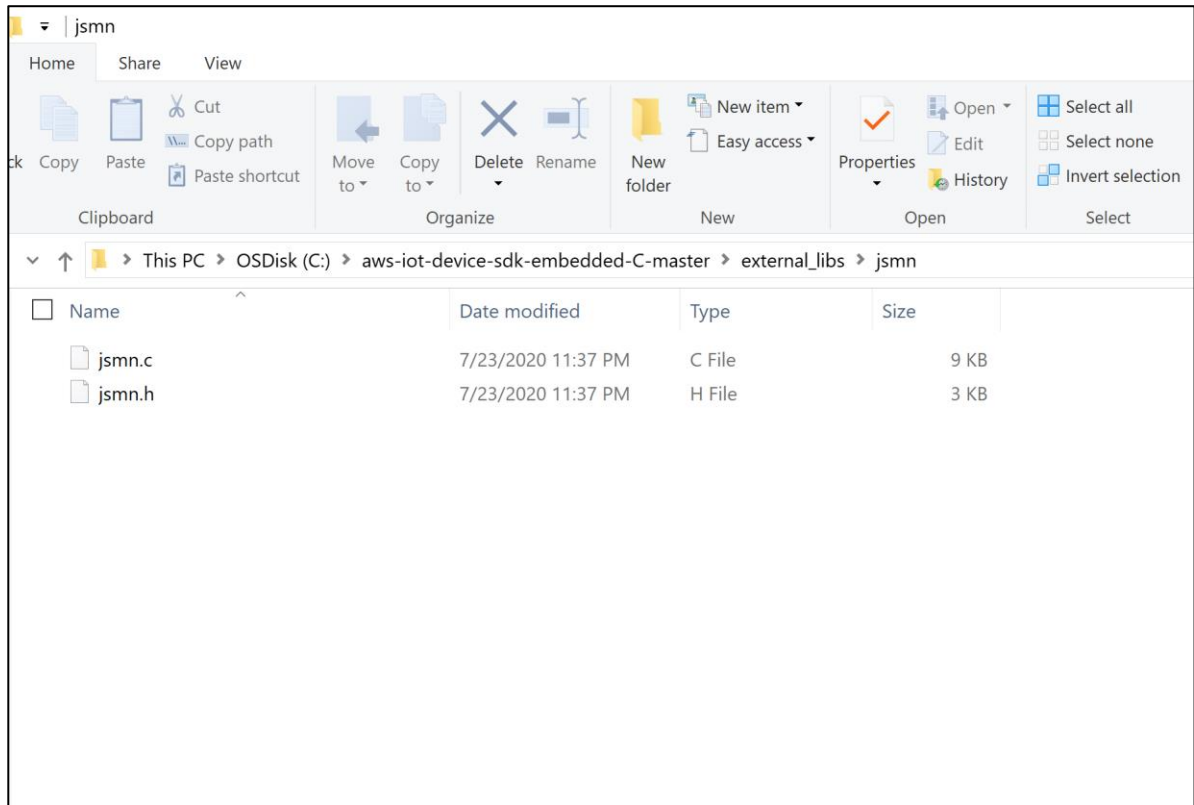


Figure 61: Copy Jsmn Files

6. Add AWS_SDK folder to the project ,copy source files from host\sapis\nwk\applications\aws_sdk\src and host\sapis\nwk\applications\aws_sdk\platform\src to the project

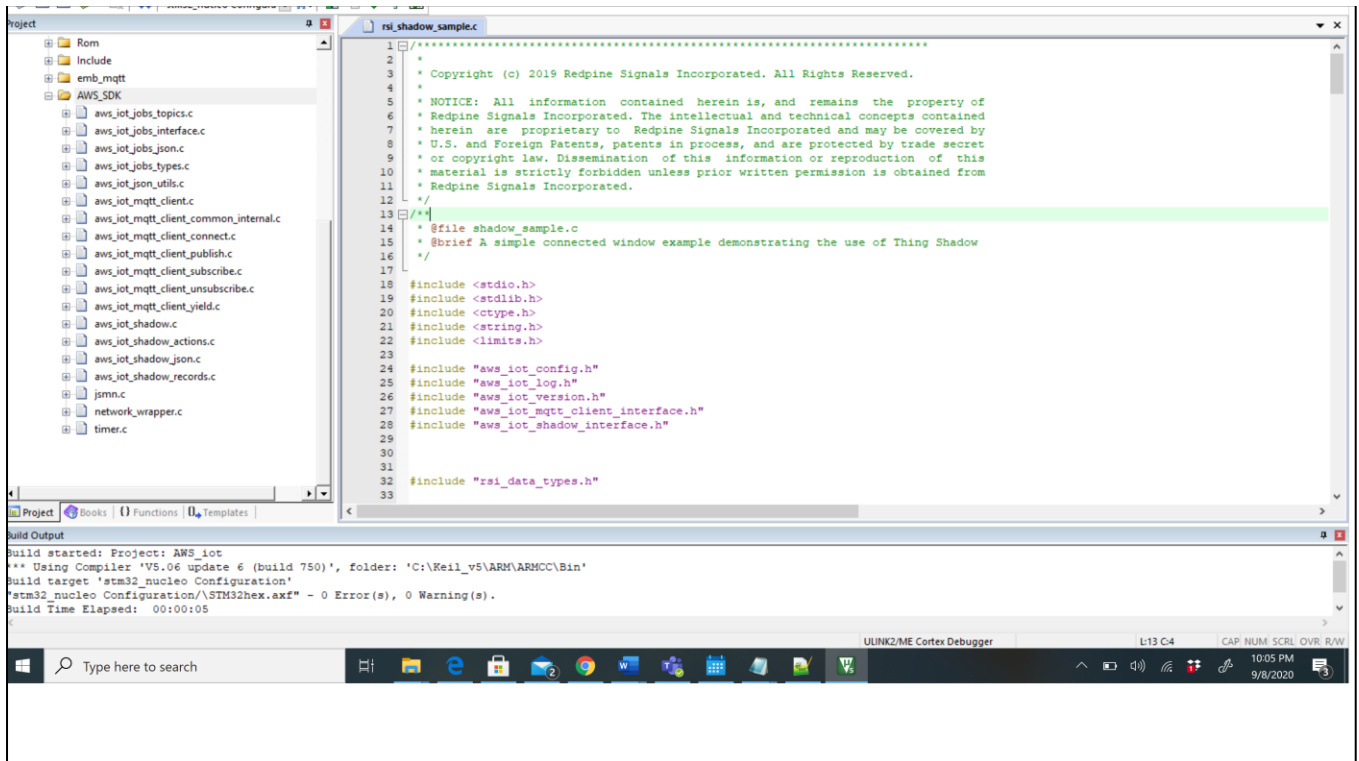


Figure 62: Add AWS_FOLDER to Project

7. Add below include paths in project settings.
 host\sapis\nwk\applications\aws_sdk\include
 host\sapis\nwk\applications\aws_sdk\platform\inc

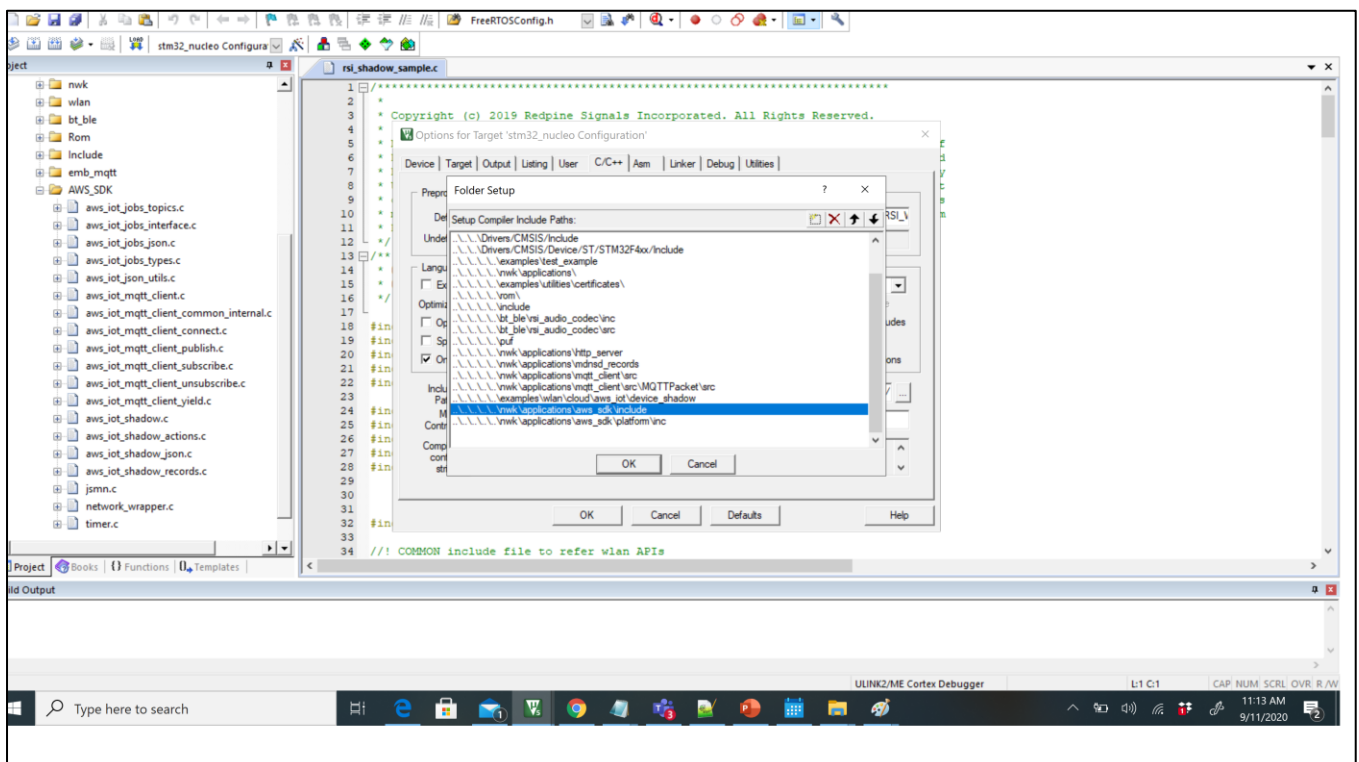


Figure 63: Add Include Paths to Project

8. The package is now ready to test the aws_iot project.

9. For aws iot reference application refer to release (RS9116.NB0.WC.GENR.OSI.XXX\host\platforms\STM32\Reference_Projects\Keil_Baremetal\Projects\SPI\AWS_IoT).

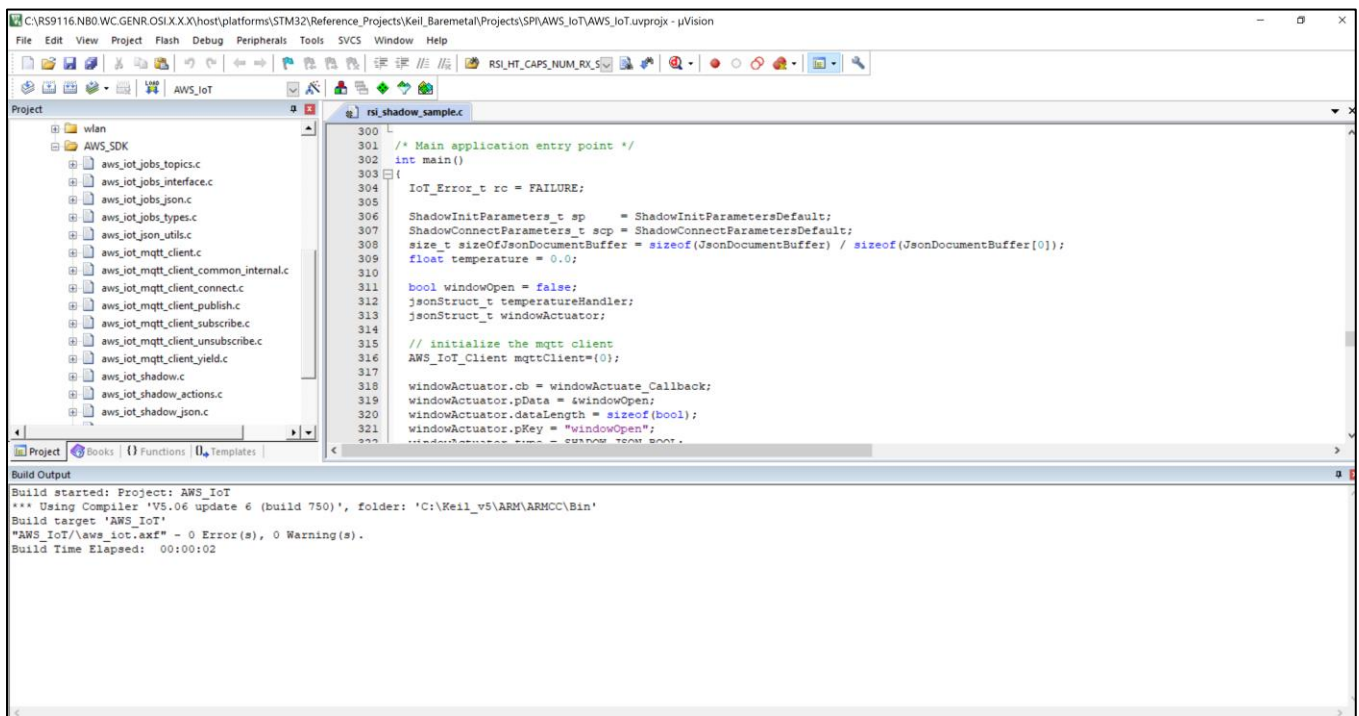


Figure 64: AWS Iot Project

10. To execute this project please follow section [Cloud->AWS IoT SDK->in RS9116W Guide for SAPI Application Examples.pdf](#) at <https://docs.silabs.com/rs9116>.

Integrating the SDK to Wrapper

Timer Functions

Timer implementation is necessary to handle request timeouts (sending MQTT connect, subscribe, etc. commands) as well as connection maintenance (MQTT keep-alive pings). Timers need millisecond resolution and are polled for expiration. So these can be implemented using a "milliseconds since startup" free-running counter if desired. Below timer functions are implemented with APIs present in rsi_hal_mcu_timer.c file to achieve the required functionality.

1. void init_timer(Timer *); init_timer - A timer structure is initialized to a clean state.
2. bool has_timer_expired(Timer *); has_timer_expired - a polling function to determine if the timer has expired.
3. void countdown_ms(Timer *, uint32_t); countdown_ms - set the timer to expire in x milliseconds and start the timer.
4. void countdown_sec(Timer *, uint32_t); countdown_sec - set the timer to expire in x seconds and start the timer.
5. uint32_t left_ms(Timer *); left_ms - query to find the time left in milliseconds for the timer to expire.
6. void delay(unsigned milliseconds) delay - sleep for the specified number of milliseconds.

Network Functions

In order for the MQTT client stack to be able to communicate via the TCP/IP network protocol stack using a mutually authenticated TLS connection, the following API calls are implemented for RS9116 platform.

Define the TLSDataParams Struct as in network_platform.h This is used for data specific to the TLS library being used.

1. IoT_Error_t iot_tls_init(Network *pNetwork, char *pRootCALocation, char *pDeviceCertLocation, char *pDevicePrivateKeyLocation, char *pDestinationURL, uint16_t DestinationPort, uint32_t timeout_ms, bool ServerVerificationFlag);

This API Initialize the network client / structure and used as given in SDK with out modifications.

2. IoT_Error_t iot_tls_connect(Network *pNetwork, TLSConnectParams *TLSPParams);

This API create a TLS TCP socket to the configure address using the credentials provided via the NewNetwork API

call. This will include DNS request to cloud, socket create, bind and connect functionality by using wireless SAPs.

3. `IoT_Error_t iot_tls_write(Network*, unsigned char*, size_t, Timer *, size_t *)`;

This API Write to the TLS network buffer by calling `int32_t rsi_send(int32_t sockID, const int8_t *msg, int32_t msgLength, int32_t flags)` API.

4. `IoT_Error_t iot_tls_read(Network*, unsigned char*, size_t, Timer *, size_t *)`;

This API Read from the TLS network buffer by calling `int32_t rsi_rcv(int32_t sockID, void *rcvBuffer, int32_t bufferLength, int32_t flags)` API.

5. `IoT_Error_t iot_tls_disconnect(Network *pNetwork)`;

This API Disconnects API by calling `int32_t rsi_shutdown(int32_t sockID, int32_t how)` API.

6. `IoT_Error_t iot_tls_is_connected(Network *pNetwork)`;

This API is used to check if the TLS layer is still connected.

6 Revision History

Version No	Date	Changes
1.0	May, 2020	Advance Version
1.1	Jul, 2020	Images, sentences, procedures were corrected based on out of box experience validation for all examples.
1.2	Sep, 2020	<ol style="list-style-type: none"> 1. Added new reference projects <ol style="list-style-type: none"> a. AWS IoT b. Sample Project c. BT_Alone d. wlan_https_bt_spp_ble_dual_role e. wlan_https_bt_spp_ble_provisioning f. wlan_throughput_bt_spp_ble_dual_role 2. Added examples list and relevant description for each reference example 3. Added FreeRTOS porting section 4. Added AWS SDK Porting section

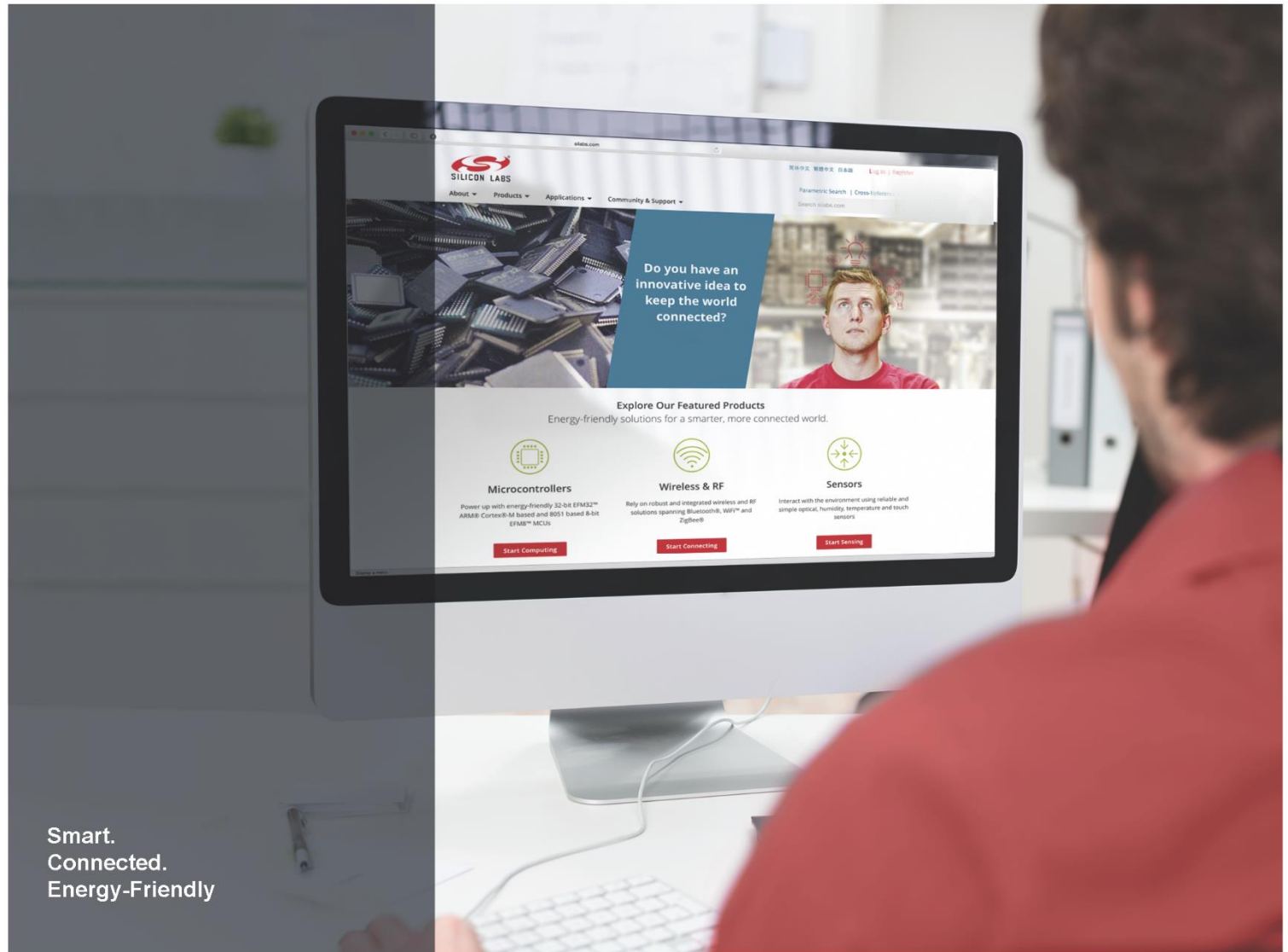
7 References

Refer to "Technical Documents" section in the link given below for Reference Manuals, and other collaterals,

<https://www.silabs.com/development-tools/wireless/wi-fi/rs9116x-db-evk-development-kit>

Toolchains for STM board can be downloaded from STM website with the link given below :

[NUCLEO-F411RE Tools & Software](#)



Smart.
Connected.
Energy-Friendly



Products
www.silabs.com/products



Quality
www.silabs.com/quality



Support and Community
community.silabs.com

Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.



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

<http://www.silabs.com>