Renesas RA™ Family

Getting Started with the Wi-Fi Modules on FSP

Introduction

This application note provides steps for adding support for a new UART-based Wi-Fi module and its associated software/firmware for operation running on RA MCUs. The driver for the new module is developed while referencing the existing Wi-Fi driver provided by FSP as a starting point. Also, the app note provides an overview of the FSP, its associated pack files, and creating the custom user pack files for a new module driver.

Additionally, this document gives steps in developing/porting new Wi-Fi drivers to FSP by utilizing the existing Wi-Fi driver. Upon reading this document and following outlined procedures, you will be able to add support for a Wi-Fi module of your choice to your own design, configure it correctly for the target application and write code, and test the module using the included application project code as a reference and efficient starting point.

Required Resources

- e² studio version 2020-10 or later
- Flexible Software Package (FSP) v2.2.0 or later
- RA Flexible Software Package Documentation
- Sierra Wireless BX310x Development Board PN: BX3105 DEV KIT_6001182 (https://www.digikey.com/short/zv9m2w)
  - Firmware version 2.7.2 or later for the Bx310x module (https://source.sierrawireless.com/resources/airprime/software/bx310x-firmware/#sthash.VQRX8FUO.dpbs)
  - BX310x Development Board User Guide (https://source.sierrawireless.com/resources/airprime/hardware_specs_user_guides/bx310x-development-board-guide/#sthash.g50DSS2y.dpbs)
- EK-RA6M3 Kit Schematics (https://www.renesas.com/us/en/document/swr/ek-ra6m3v1-design-package?language=en)

Target Device

- Renesas RA MCUs. Tested on EK-RA6M3 kit

Contents

1. Introduction

This application note, along with the associated application project, is the starting point in developing/porting new UART based Wi-Fi drivers to FSP, by utilizing the existing UART based FSP Wi-Fi driver. In general, this App note can also be used for custom driver development for FSP.

On the same note, for more clarity on the Wi-Fi driver development, app note also gives overview of

- FSP Wi-Fi driver architecture and its components
- Data and control path of the Wi-Fi driver
- Supported application level APIs
- Interface to the secured and non-secured sockets for TCP/IP communication
- Module specific driver APIs to interface the Wi-Fi module
- FreeRTOS based stream buffer to handle the data size of arbitrary lengths
- UART drivers to communicate with the Wi-Fi modules

On top of all required prerequisite to add/modify the new Wi-Fi module to the FSP, high-level overview of the following topics are also provided.

- Brief introduction to FSP
- FSP pack files
- Organization of FSP pack files
- Creating the user pack files
- Overview of XML changes required for the new modules.

2. FSP Overview

The Renesas FSP is an enhanced software package designed to provide easy-to-use, scalable, high-quality software for embedded system designs using the Renesas RA™ family of Arm® microcontrollers. FSP provides a versatile way to build secure, connected IoT devices using production ready drivers, RTOS, and other middleware stacks.

FSP includes HAL drivers, middleware stacks with RTOS integration to ease implementation of complex modules like communication and security. FSP uses an open software ecosystem and provides flexibility in using bare-metal programming, as well as RTOS based applications.

Figure 1 below shows the block diagram of the FSP Architecture.

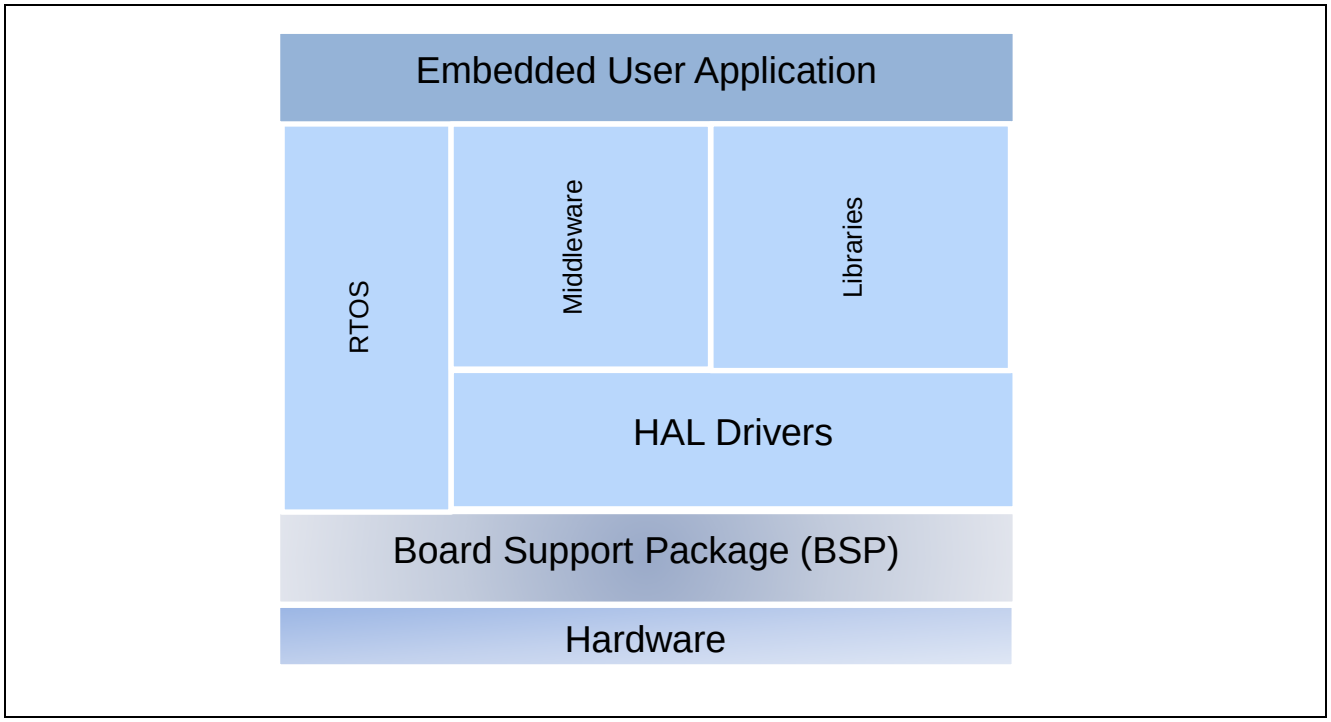


Figure 1. FSP Architecture

2.1 FSP Software Modules

As mentioned, the FSP is a comprehensive piece of software covering all aspects of an embedded systems software development. It includes the following parts:

2.1.1 Board Support Package

The **Board Support Package (BSP)**, customized for every RA hardware kit and Microcontroller. It includes the startup code for all supported blocks. Developers using custom hardware can take advantage of the BSP, as it can be tailored for end products and your own board by using the Custom BSP Creator built into e^2 studio.

2.1.2 HAL Drivers

The RTOS-independent **HAL Drivers**, providing efficient drivers for all peripherals and systems services. They eliminate a lot of deep study of the underlying hardware in the microcontroller as they abstract the bit-settings and register addresses for you.

2.1.3 Functional Libraries

The Functional **Libraries** containing, for example, specialized software for digital signal processing or security and encryption related functions also reduce development time and improve the stability of the end-application. Even the libraries can be in the form of middleware as well. For instance, the emWin graphical package from SEGGER is available in FSP in the form of libraries.

2.1.4 Real-Time Operating system

The **RTOS** (Real-Time Operating System), provides a multitasking real-time kernel with preemptive scheduling and a small memory footprint. Amazon FreeRTOS is used as the RTOS as part of the FSP.

2.1.5 Middleware

The **Middleware**, including TCP/IP communication, file systems, graphical user interfaces, and USB. Everything here is completely optimized for, and integrated into, the FSP.

In summary, FSP is a collection of software modules in Packs. More details on the FSP and its components can be found in the FSP User's manual.

2.2 FSP Packs

FSP packs are the delivery mechanism for software components, device parameters, and BSP which are in accordance with the CMSIS standard and can be used across RA ARM® Cortex®-M microcontroller devices.

When FSP is installed, a variety of Pack files, also known as packs, are extracted. The FSP packs can be classified into different categories.

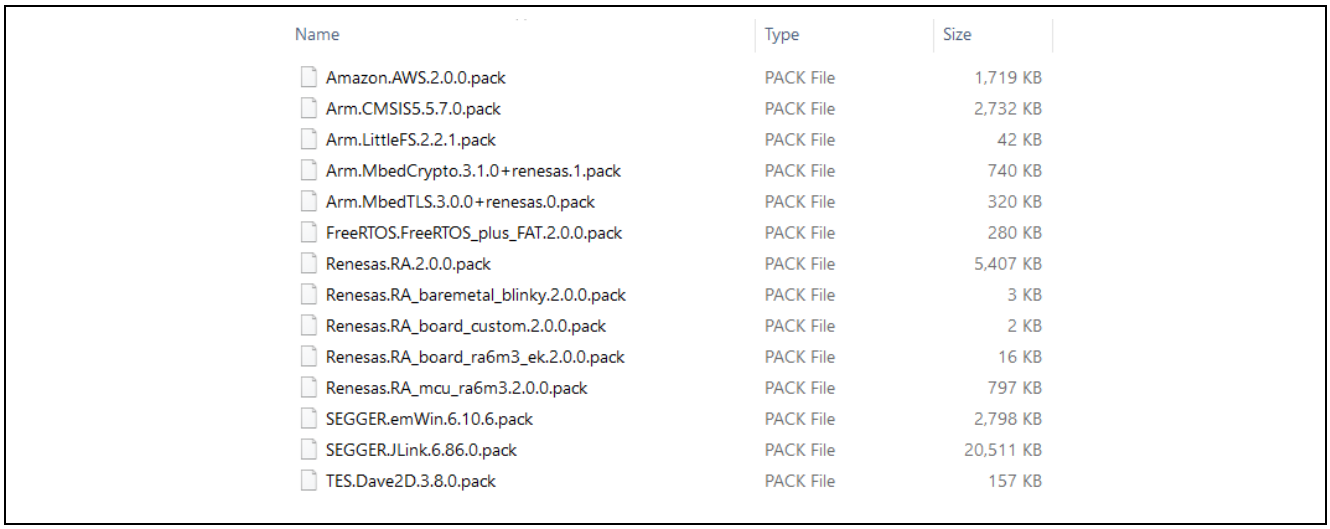
- Board Support Packs
- MCU Packs
- Middleware Packs
- Third party or vendor packs
- Sample Project packs

Note: For more information on the CMSIS packs refer to section 8 References of this document

2.2.1 Overview of the FSP Packs

The installed FSP Packs are available under the folder `e2_studio\internal\projectgen\ra\packs`. It contains all the required and supported FSP packs to create embedded application using RA MCUs.

Figure 2 shows a screenshot of different types of Pack files as part of the installation. You can see the pack files start with the name of the vendor such as Amazon, ARM, and SEGGER. All the Renesas pack files start with the vendor name as Renesas. The file name also contains the features and a version associated with the pack.



Name	Type	Size
Amazon.AWS.2.0.0.pack	PACK File	1,719 KB
Arm.CMSIS5.5.7.0.pack	PACK File	2,732 KB
Arm.LittleFS.2.2.1.pack	PACK File	42 KB
Arm.MbedCrypto.3.1.0+renesas.1.pack	PACK File	740 KB
Arm.MbedTLS.3.0.0+renesas.0.pack	PACK File	320 KB
FreeRTOS.FreeRTOS_plus_FAT.2.0.0.pack	PACK File	280 KB
Renesas.RA.2.0.0.pack	PACK File	5,407 KB
Renesas.RA_baremetal_blinky.2.0.0.pack	PACK File	3 KB
Renesas.RA_board_custom.2.0.0.pack	PACK File	2 KB
Renesas.RA_board_ra6m3_ek.2.0.0.pack	PACK File	16 KB
Renesas.RA_mcu_ra6m3.2.0.0.pack	PACK File	797 KB
SEGGER.emWin.6.10.6.pack	PACK File	2,798 KB
SEGGER.JLink.6.86.0.pack	PACK File	20,511 KB
TES.Dave2D.3.8.0.pack	PACK File	157 KB

Figure 2. FSP Pack files

The contents of the pack files typically contain the package description (`.pdsc`) file at the root, which is an XML-based file describing the content of a software pack. The Pack also contains a software component under the subfolder which includes:

- Source code, header files, and software libraries
- Documentation and source code templates
- Device parameters along with startup code and programming algorithms
- Sample projects code

Users can unzip this pack file to see the contents. The contents of the Renesas.RA pack file and its folder/sub folder structure are listed below in the Figure 3.

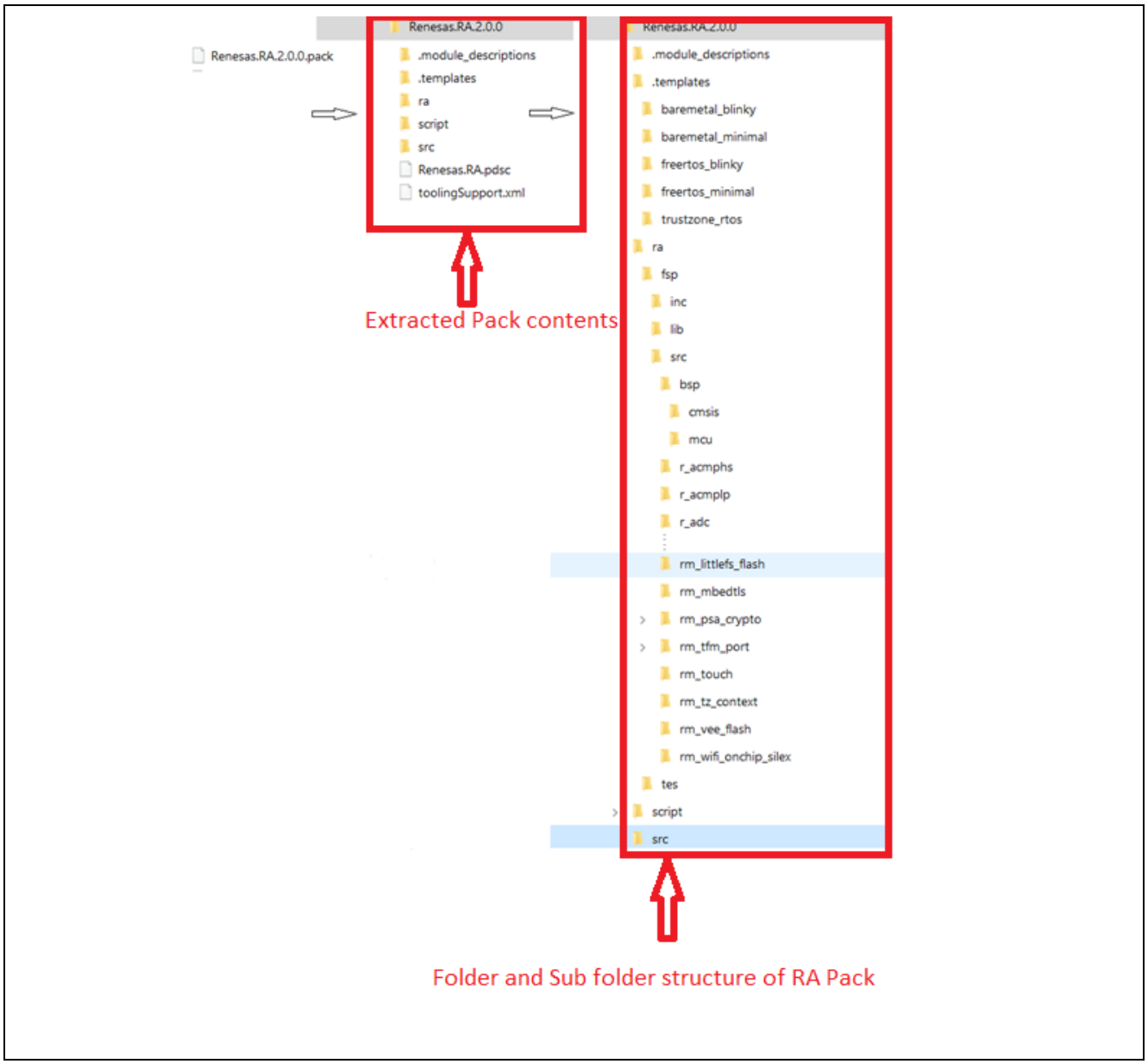


Figure 3. FSP Pack contents

2.2.2 User-Creatable FSP Packs

Users can create the packs to support user defined modules in addition to those available from FSP. For example, if a company wishes to create a custom board representing their microcontroller-based product, then a BSP can be created, verified, and then distributed to application developers for speeding development. In the case of a custom communication module if support is not available in FSP, a separate pack file is required to work with new module. These pack files play an important role in updating the code between different releases and across the projects. In this app note, we will be creating user packs for a Wi-Fi module in section 3.15. The driver for the new Wi-Fi module will be distributable as a user pack.

Note: The driver pack is not part of the FSP Pack distribution. But it is bundled as part of the application project.

2.2.3 User Pack Creation Tools

Pack creation can be done in different ways. It can be created through integrated pack creation utility with e^2 studio, or pack creation scripts, or by manually modifying the existing FSP packs for the new module. In this app note we will be using the manual user pack creation method to showcase the creation of the user pack for the newly added Wi-Fi module.

3. FSP Wi-Fi Driver Module Architecture

Modules are the core building block of FSP. Modules can do many different things, but all modules share the basic concept of providing functionality upwards and requiring functionality from below.

Modules can be layered on top of one another, building an FSP stack. The stacking process is performed by matching what one module provides with what another module requires. The Wi-Fi module, for example, requires a physical layer communication interface for data transfer, which can be provided by the UART driver module. It provides functionality to the secure sockets through the API created by the Wi-Fi driver to the secure sockets at the upper layer.

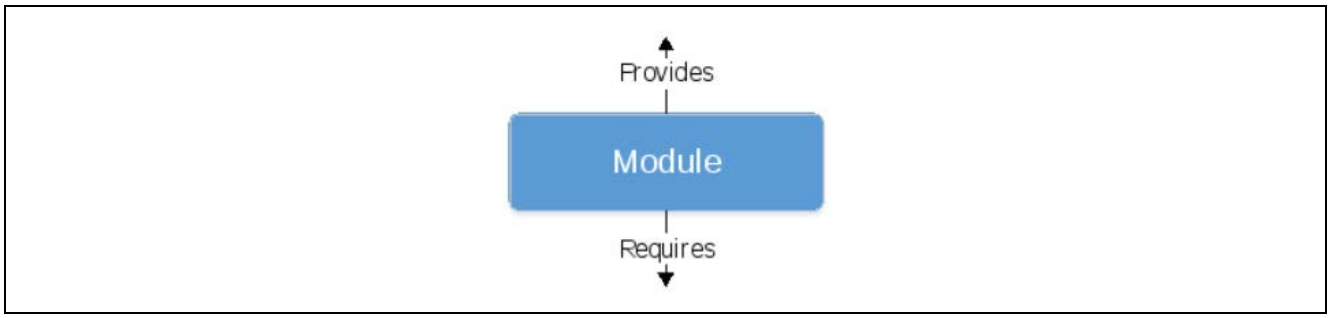


Figure 4. Provides Functionality to the Caller and Requires Functionality from the Lower Level

3.1 FSP Wi-Fi Driver Module Overview

Figure 5 below shows the overview of the FSP Wi-Fi module architecture and its components

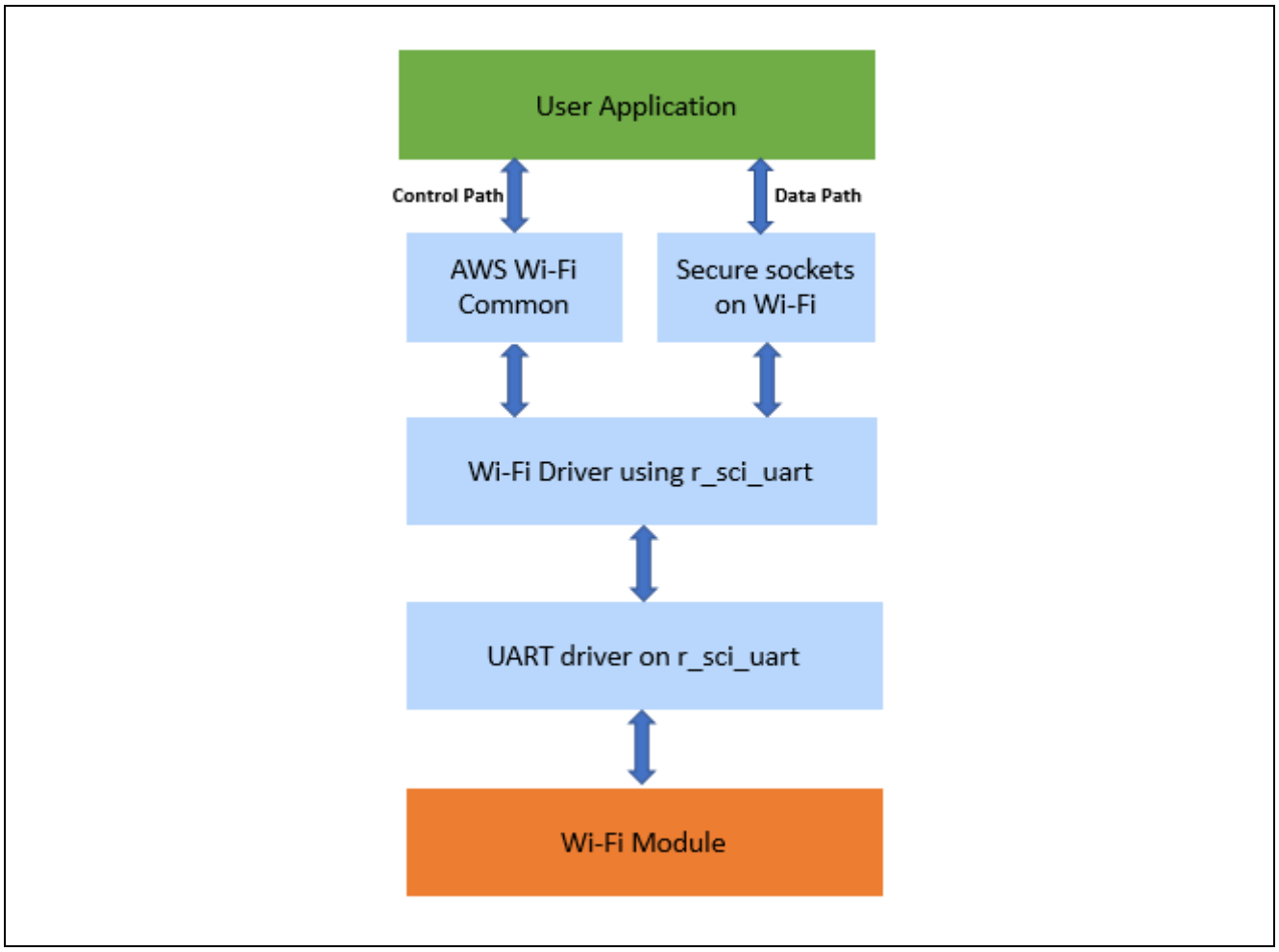


Figure 5. FSP Wi-Fi Driver Architecture

3.2 Wi-Fi Module Directory Structure.

FSP Wi-Fi driver for the Silex ULPGN Wi-Fi module is packaged as part of the FSP pack(`Renesas.RA.x.x.x.pack`). When the FSP provided Wi-Fi drivers are used for the application development, the generated code is arranged in multiple folders. The details of the files and folder structure of the code are shown below in the Figure 6.

For the Silex ULPGN Wi-Fi module, the configuration related header file is generated/stored under `ra_cfg\fsp_cfg\rm_wifi_onchip_silex_cfg.h`. This file's contents are generated based on the configurations made from the configurator.

Header file `rm_wifi_onchip_silex.h` present inside the `ra\fsp\inc\instances` folder contains the data structure, driver function prototypes for the Silex ULPGN module.

The `rm_wifi_onchip_silex` folder under `ra\fsp\src` contains the secure socket interface, application level interface, FreeRTOS specific configuration and driver level Interface code for the module.

When porting or modifying the Wi-Fi drivers for the new module, it is a good practice to maintain the code structure in the same format.

Note: The new driver pack for the Sierra Wireless BX310x module is bundled in the same directory format.

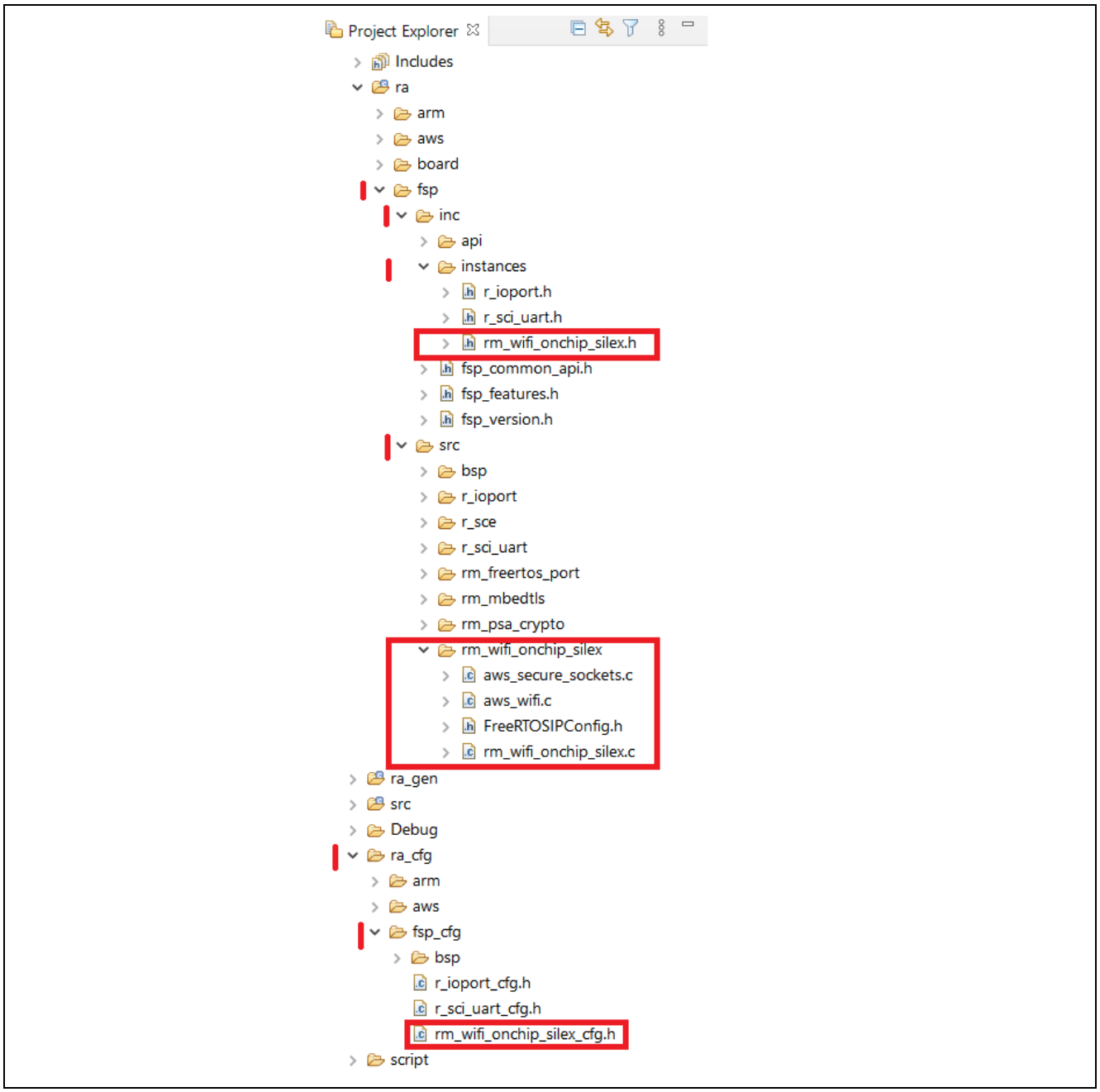


Figure 6. Wi-Fi driver File/folder structure for Silex ULPGN Wi-Fi module

3.3 Supported APIs for the Application from AWS Wi-Fi Library.

Libraries provided by amazon AWS Wi-Fi library includes the application level APIs which can be used for the application development. These APIs use glue logic interface to the Wi-Fi driver level APIs. As a result, it can easily be ported across different Wi-Fi modules supported with FSP. The AWS supported list of APIs or listed below. These APIs are found inside the file `aws_wifi.c`.

- WIFI_On
- WIFI_Off
- WIFI_ConnectAP
- WIFI_Disconnect
- WIFI_Reset
- WIFI_Scan
- WIFI_Ping
- WIFI_GetIP
- WIFI_GetMAC
- WIFI_GetHostIP
- WIFI_IsConnected

3.4 AWS Sockets

The AWS Secure Sockets library provides socket interface to the embedded applications to communicate securely. The sockets library is based on the Berkeley sockets interface with additional secure communication option provided by mbedTLS.

FSP provides a simple interface to connect using securely or send data in plaintext. The selection is user programable during the project creation. When developing Wi-Fi applications, these socket APIs are made available to the user from the FSP. The following APIs are available as part of the file `aws_secure_sockets.c`.

- SOCKETS_Init
- SOCKETS_Socket
- SOCKETS_Connect
- SOCKETS_Recv
- SOCKETS_Send
- SOCKETS_Shutdown
- SOCKETS_Close
- SOCKETS_SetSockOptu
- SOCKETS_GetHostByName

Note: Usage restriction:

- Only TCP sockets are supported by the FreeRTOS Secure Sockets library. UDP sockets are not supported.
- Only client APIs are supported by the FreeRTOS Secure Sockets library. Server APIs, including Bind, Accept, and Listen, are not supported.

3.5 Wi-Fi Driver API

Driver level APIs are the entry point to access the module through the AT commands. Depending on the module and its supported features, the AT commands are grouped under the driver API. Individual or group of AT commands are used under an API to interact with the module. The driver level APIs uses these AT commands to interact with the Wi-Fi module for Wi-Fi configurations, network configurations, and even for sending and receiving the data. Before making changes to these APIs, individual AT commands needs to be validated to confirm the working behavior as documented. The command and response for some of the modules varies depending on the Wi-Fi Chip vendor. If a different chipset from the same vendor is used, then most of the AT commands may be reused supporting the operation, without making major changes to the driver level APIs.

3.5.1 Supported Driver-Level APIs for the Wi-Fi Module

The FSP Wi-Fi driver module supports the following APIs for the Silex ULPGN module. For supporting operation of a new Wi-Fi module similar APIs needs to be developed and tested.

Note: Some of the APIs may not be applicable to the new module or in some cases a newer set of APIs is needed to support the new feature present in the module. It is up to the user to add/delete the APIs as required. But from the driver development perspective, the APIs provided by FSP can be used as reference for the new Wi-Fi module.

- rm_wifi_onchip_silex_open
- rm_wifi_onchip_silex_version_get
- rm_wifi_onchip_silex_close
- rm_wifi_onchip_silex_connect
- rm_wifi_onchip_silex_mac_addr_get
- rm_wifi_onchip_silex_scan
- rm_wifi_onchip_silex_ping
- rm_wifi_onchip_silex_ip_addr_get
- rm_wifi_onchip_silex_avail_socket_get
- rm_wifi_onchip_silex_socket_status_get
- rm_wifi_onchip_silex_socket_create
- rm_wifi_onchip_silex_tcp_connect
- rm_wifi_onchip_silex_tcp_send
- rm_wifi_onchip_silex_tcp_recv
- rm_wifi_onchip_silex_tcp_shutdown
- rm_wifi_onchip_silex_socket_disconnect
- rm_wifi_onchip_silex_disconnect
- rm_wifi_onchip_silex_dns_query
- rm_wifi_onchip_silex_socket_connected

3.6 Stream Buffer

Stream buffers are RTOS objects for Inter-Task communication available from FreeRTOS, they are optimized for single reader, single writer scenarios, such as passing data from an interrupt service routine to a task.

The stream buffer implementation uses direct-to-task notifications. Therefore, calling a stream buffer API function that places the calling task into the Blocked state can change the calling task's notification state and value.

In the FSP Wi-Fi driver implementation, Stream buffers allow a stream of bytes to be passed from an interrupt service routine to a task. A byte stream can be of arbitrary length and does not necessarily have a beginning or end. Any number of bytes can be written at once, and any number of bytes can be read at once. Data is passed by copy – the data is copied into the buffer by the sender and out of the buffer by the read operation.

3.7 UART Drivers

The UART driver module provides a simple communication interface using the standard UART protocol between the MCU and Wi-Fi Module. The UART module on the MCU side uses the SCI module to communicate with the SCI peripherals and data-transfer (DTC) peripherals on a RA MCU.

The UART HAL Module manages data flow using the standard UART protocol. Flow Control support is also available for synchronization. DTC support can be optionally added to the module during the configuration so that DTC takes care of the Data transfers without using the much of MCU cycles.

Note: In the Sierra Wi-Fi module implementation, asynchronous UART communication is used without the flow control. For achieving higher data throughput, flow control can be used.

4. Adding Support for New Wi-Fi module

This section covers considerations to be made when identifying the suitable Wi-Fi module and modifying the existing FSP Wi-Fi drivers to support the new module. This includes creating and modifying the Wi-Fi driver, Socket level API, Application level API modification, and the user pack contents for the new driver pack.

4.1 Identifying the New Wi-Fi mModule

When adding a new Wi-Fi module to FSP by leveraging existing Wi-Fi driver implementation, first identify the new Wi-Fi module that uses the on-module TCP/IP stack, supports AT commands, and supports UART interface to communicate with the module. On top of the basic minimum criteria, the module also needs to support the TCP/UDP socket interface, DHCP Client, and DNS client. In addition to the bare minimum features, it is up to the users to choose other features supported by the module as value addition features.

Note: This application note only shows how to add/modify the driver for the new Wi-Fi module which has a UART interface, AT command support, and on-module TCP/IP support. This doesn't mean that the other modules which have the SPI/i2c interface cannot be added/supported in FSP. With the addition of interface drivers, changes to control path and data path, and addition of glue logic to the TCP/IP stack present on the MCU, different Wi-Fi modules can be added.

Note: Module manufacturers usually categorize modules by specific parameters such as IEEE 802.11 a/b/g/n , transmit power, data rate, RF compliance, secure Wi-Fi etc. These play an important role in identifying the Wi-Fi module. As part of this app note we don't cover much on this topic. However, it is advisable to choose the right module for your application with good technical support from module vendors.

4.2 Identify the Driver Related Changes to New Module

After identifying the module which can be supported with FSP, list the features which are common between the new module vs the FSP supported Silex ULPGN Wi-Fi module. Also list the new set of features which needs to be included.

List what features you need to support in your driver implementation. For example, some Wi-Fi modules support both AP mode and Station mode. Choose modes your driver needs to support, both Access point mode and station mode, or just station mode. You can select the appropriate AT commands for the driver modification or feature addition.

For basic Wi-Fi connectivity, Wi-Fi and network-specific basic AT commands for the new Sierra module are listed below in Table 1 and Table 2. These AT commands are used in the new driver.

Table 1. Wi-Fi specific AT Commands

AT Commands	Description
+SRWAPCFG	Configure local device's Wi-Fi access point
+SRWSTACFG	Configure/display Wi-Fi station connection information
+SRWSTACON	Connect/disconnect to Wi-Fi access point
+SRWSTANETCFG	Configure/ local device Wi-Fi station interface network IP address
+SRWSTASCN	Scan for Wi-Fi access points

Table 2. Network specific AT Commands

AT Commands	Description
+KTCPCFG	TCP Session (Connection) Configuration
+KTCPCNX	Start TCP Connection
+KTCPCLOSE	Close TCP Connection
+SRWSTANETCFG	Configure/ local device Wi-Fi station interface network IP address
+KTCPSND	Send Data through a TCP Connection
+KPING	Ping an IP Address

These are some of the AT commands which must be added or modified in the driver APIs. Details of the modification is shown in the next section.

Note: The Silex ULPGN Wi-Fi module uses the Qualcomm QCA4010 System-on-Chip. If the identified module is also based on the Qualcomm SOC, the changes to the driver can be minimized by leveraging the existing driver. But if the module identified is different, the AT commands and response data/strings may be different and must be handled differently in the driver APIs.

4.3 Modify the Driver APIs

Refer to the module datasheet and user's manual for the Sierra Wi-Fi and Silex Wi-Fi modules. List the AT commands for the Identified features on the new Wi-Fi modules and compare to the existing FSP supported module. Check the basic API support for the new module are available in the existing Wi-Fi driver as part of FSP (reference Section 3.5). If available, identify the API, and change the AT command and response code/string for the API to support the new module. These AT commands can be found in the module's AT command reference manual of the new module.

If the equivalent APIs are not available in the FSP, create a new API to accommodate the new feature and its associated AT commands and response.

Also, it is important to identify all the APIs have the supporting equivalent AT commands required by the module. For instance, for some of the modules the AT command support may be very limited, or a feature may not be supported (for example, the stack in the Sierra Wireless module does not have support for an AT command to perform DNS lookup which is provided by the Silex WiFi and used frequently by any application using socket programming). Such limitations can be solved by using the workaround suggested by the module vendor.

Note: The existing FSP Wi-Fi driver module has string parsing routines. Leveraging the string parsing routines can be beneficial for easy porting for the new module.

Now let us look into the header file and source code related changes for the new module.

For changing the driver APIs in the header file, open `rm_wifi_onchip_silex.h` under the `/ra/fsp/inc/instances` folder, add/modify the macros, enums, and driver-specific data structures as applicable. Rename the function prototypes, or any reference made to the Silex module to Sierra specific as required.

For instance, some of the sample changes are shown here.

1. Change the enums
   `WIFI_ONCHIP_SILEX_SECURITY_WPA2` can be changed to `WIFI_ONCHIP_SIERRA_SECURITY_WPA2`
2. Change the Data structures

```
/** User configuration structure, used in open function */
typedef struct st_wifi_onchip_cfg
{
    const uint32_t          num_uarts;
    const uint32_t          num_sockets;
    const bsp_io_port_pin_t reset_pin;
    const uart_instance_t * uart_instances[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS];
    void const            * p_context;
    void const            * p_extend;
} wifi_onchip_silex_cfg_t;

typedef struct st_wifi_onchip_cfg
{
    const uint32_t          num_uarts;
    const uint32_t          num_sockets;
    const bsp_io_port_pin_t reset_pin;
    const uart_instance_t * uart_instances[WIFI_ONCHIP_SIERRA_CFG_MAX_NUMBER_UART_PORTS];
    void const            * p_context;
    void const            * p_extend;
} wifi_onchip_sierra_cfg_t;
```

3. Change the API function prototypes name
   `fsp_err_t rm_wifi_onchip_silex_open(wifi_onchip_silex_cfg_t const * const p_cfg);`
   to
   `fsp_err_t rm_wifi_onchip_sierra_open(wifi_onchip_sierra_cfg_t const * const p_cfg);`

For the changes related to driver API inside the source file, open `rm_wifi_onchip_silex.c` under the `/ra/fsp/src/ rm_wifi_onchip_silex` folder, and add/modify the API carefully as applicable. Some of the code may be removed as it is not required for the new module. In some cases, additional code can be added to support the new AT commands for the driver APIs.

A sample screenshot of the source code changes from Silex module to Sierra module is shown below. This is just for reference and as a user you are required to change all the code as desired for your module.

```
if (FSP_SUCCESS == err)
{
    err = rm_wifi_onchip_silex_send_basic(p_instance_ctrl,
                                          p_instance_ctrl->curr_cmd_port,
                                          "ATS108=1\r",
                                          WIFI_ONCHIP_SILEX_TIMEOUT_100MS,
                                          WIFI_ONCHIP_SILEX_TIMEOUT_8SEC,
                                          WIFI_ONCHIP_SILEX_RETURN_OK);
}
```

```
if (FSP_SUCCESS == err)
{
    err = rm_wifi_onchip_sierra_send_basic(p_instance_ctrl,
                                           p_instance_ctrl->curr_cmd_port,
                                           "AT+SRWCFG=1,2\r",
                                           WIFI_ONCHIP_SIERRA_TIMEOUT_3MS,
                                           WIFI_ONCHIP_SIERRA_TIMEOUT_200MS,
                                           WIFI_ONCHIP_SIERRA_RETURN_OK);
}
```

Figure 7. Sample Driver Code Change for the Sierra Wi-Fi Module

Note: There are many more required changes inside this file. The sample shown here is for reference only.

Note: Change the file name and the folder name for the new set drivers. These are required to differentiate the Silex module driver vs the Sierra module drivers. Also maintain the same directory structure.

Note: After all the changes are done, make a backup of these files and folders. Also make the changes to the files and folders as read-only. This will help prevent the file form getting overwritten by FSP during the Project Generate.

4.4 Files and Folder Structure for the New Module

The screenshot in Figure 8 shows the directory structure and the list of files modified as part of the new Wi-Fi module driver.

The screenshot are similar to the Silex module packs available as part of the FSP packs. When you create a pack for the new (Sierra Wireless module) Wi-Fi driver, these will not be part of the FSP pack. The pack will be an independent user pack. This makes it easy to port across different FSP releases.

The file and folder structure is kept similar to Silex module to ensure compatibility and to maintain the same structure for ease of use.

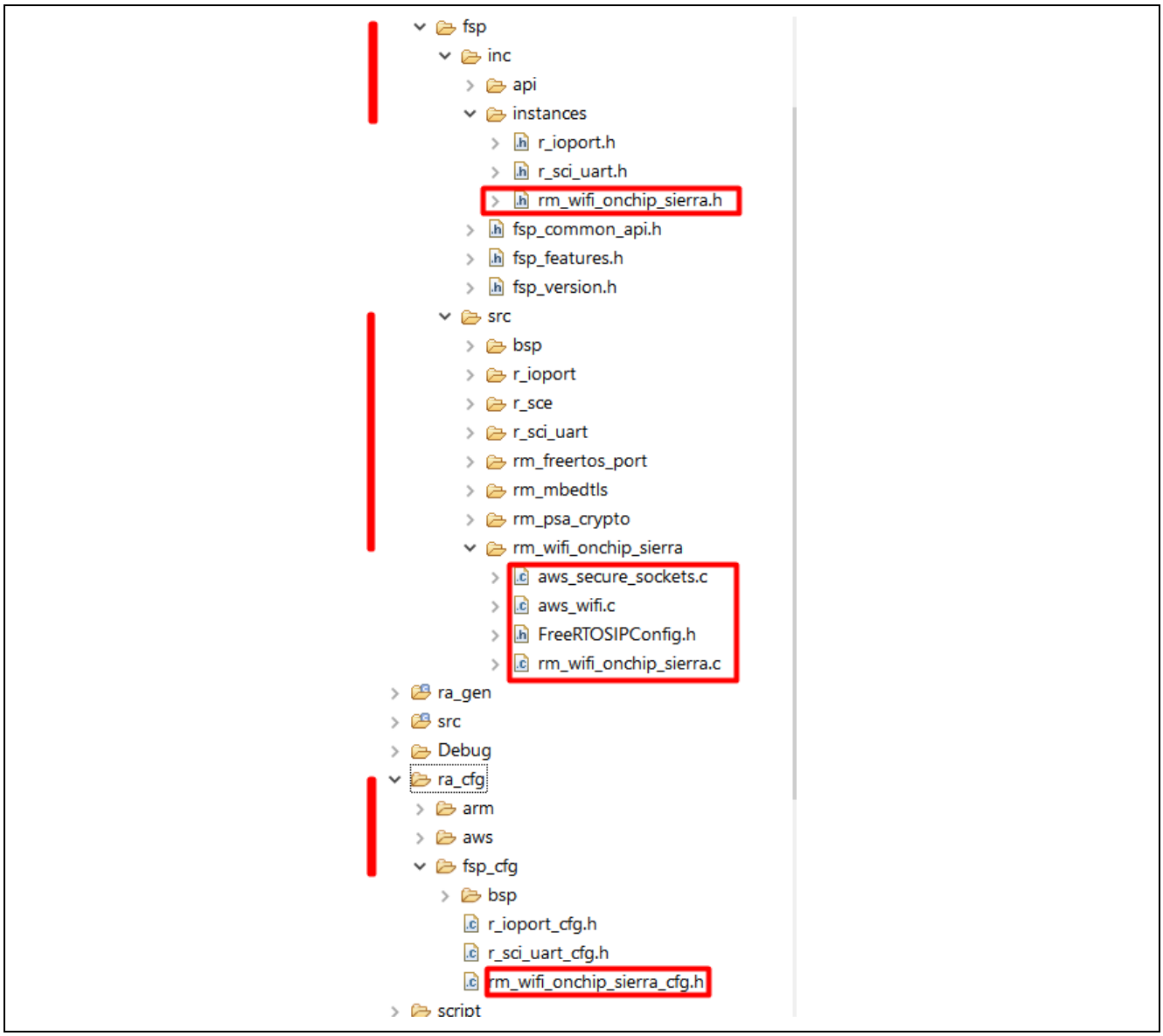


Figure 8. Wi-Fi Driver File/Folder Structure for Sierra Wi-Fi Module

4.5 Modifying the AWS API

To accommodate the application level Wi-Fi APIs calling the right driver level APIs, you need to make changes in the `aws_wifi.c` file under the folder `ra\fsp\inc\rm_wifi_onchip_silex` and call the driver API as applicable.

For instance, the `WIFI_On` API calls the Driver API `rm_wifi_onchip_silex_open` for the Silex module. This needs to be changed to `rm_wifi_onchip_sierra_open` to support the sierra Wi-Fi module. Change it for all the applicable APIs as required.

The APIs to be changed is listed in the section 3.3.

Note: Some of the Wi-Fi APIs may not have the support at the driver level interface. If the support in the module is present, you can take advantage of it by adding the new driver APIs and linking to the application level APIs.

4.6 Modifying the AWS Sockets API

To accommodate the socket-level APIs calling the right driver-level APIs, you need to make changes in the `aws_secure_sockets.c` file under the folder `ra\fsp\inc\rm_wifi_onchip_silex` and call the driver API as applicable.

For instance, SOCKETS_Connect API calls the driver API rm_wifi_onchip_silex_tcp_connect for the Silex module. This needs to be changed to rm_wifi_onchip_sierra_tcp_connect for the Sierra module. Such calls need to be changed for all the applicable APIs as required.

The APIs to be changed is listed in the section 3.4.

4.7 Modifying the XML Files for the New Wi-Fi Module

Xml file changes under the .module_descriptions folder are required for the FSP configurator. Before you start to make the changes on this xml file, we need to understand the organization of this file. The xml file contains the following listed tags, under <raModuleDescription> you will see the <module> and its associated <config>

Under the <module> tag you will see the ID, display data, version and url details. It also has the tags for <requires> and <provides>

Under the <config> tag you will see the you will notice the property and contents for the FSP configuration. The module tag also contains the constrain, header, includes etc.

This XML file needs to be copied and modified for the new module to prepare the pack file.

Note: For more details refer to the xml file for the Silex module/Sierra module as part of the pack files.

Part of the FSP RA Pack folder contains the xml files under the .module_descriptions folder. The xml file Renesas##HAL Drivers##all##rm_wifi_onchip_silex####x.y.z.xml is key for the configuration parameters for the Wi-Fi module configuration on the FSP configurator. Users are required to modify this xml file for the new Wi-Fi module specific configurations.

Copy Renesas##HAL Drivers##all##rm_wifi_onchip_silex####x.y.z.xml from the RA packs folder to a temp location. Change the name of the file, the resulting file will be Renesas##HAL Drivers##all##rm_wifi_onchip_sierra####x.y.z.xml.

Note: Make sure the version numbers match the FSP version. In this case ‘x.y.z’ should be ‘2.1.0’.

Open the file and change the contents. Find and replace instances of ‘silex’ with ‘sierra’ and ‘SILEX’ with ‘SIERRA’.

Save the file and continue to the next steps of pack creation

4.8 Pack Creation for the New Wi-Fi Module

Pack files can be created in 2 ways

- Manual pack creation
- Using the e^2 studio RA Pack Creator utility

4.8.1 Manual User Pack Creation

To create the user pack manually, you need to validate the modified driver code on the new module. Once the code is validated and ready, it can be added as part of the user pack, to make it available for configuration using the FSP configurator and for application use. Creating the Pack involves the code to be arranged with proper folder structure format as referred below in the Figure 9.

Once the driver code changes are ready, new changes are also needed to be added to the xml file under **.module_description** folder (Explained in the section 3.14), pack descriptor file (.pdsc) file for the description of the pack contents, and the tooling support file for tooling info. These are the pack specific changes.

Note: Refer the Silex Wi-Fi module/Sierra Wi-Fi module pack for detailed information on the contents of the pack file.

- Create a folder with the format Vendor.VendorFeature.Version. Here for the Sierra Wi-Fi module we have Renesas.RA_wifi_onchip_sierra.2.1.0 as the folder name. Place the modified

`toolingSupport.xml` and `Reneas.RA_wifi_onchip_sierra.pdsc` file inside the root of the newly created folder (`Renesas.RA_wifi_onchip_sierra.2.1.0`).

- Create a folder with a name like `.module_descriptions`, and place the modified module description file under this folder. Here in the case of Sierra Wi-Fi module `Renesas##HAL Drivers##all##rm_wifi_onchip_sierra####2.1.0.xml` is used as module description file.
- For source and header files create a set of folders in the same hierarchy `ra\fsp\inc\instances` and `ra\fsp\src\rm_wifi_onchip_sierra` under `Renesas.RA_wifi_onchip_sierra.2.1.0` as shown in the snapshot and place the modified header files and source files.

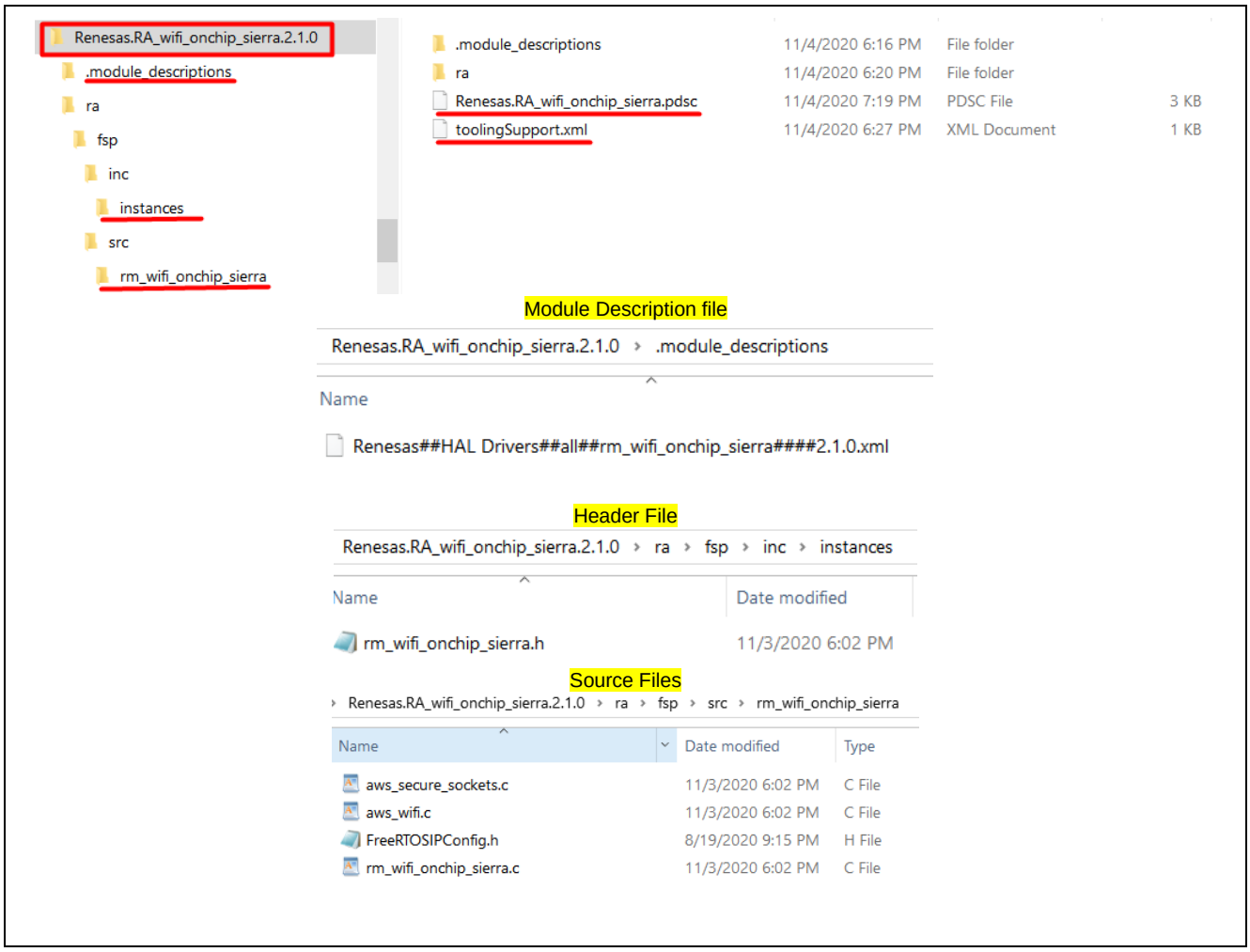


Figure 9. Folder Structure and File info for the new Wi-Fi driver Pack

- After all the changes are in place, you need to create a zip file using the 7-zip utility. The method of creating the `.zip` file and later renaming as, `.pack` file is shown in Figure 10.

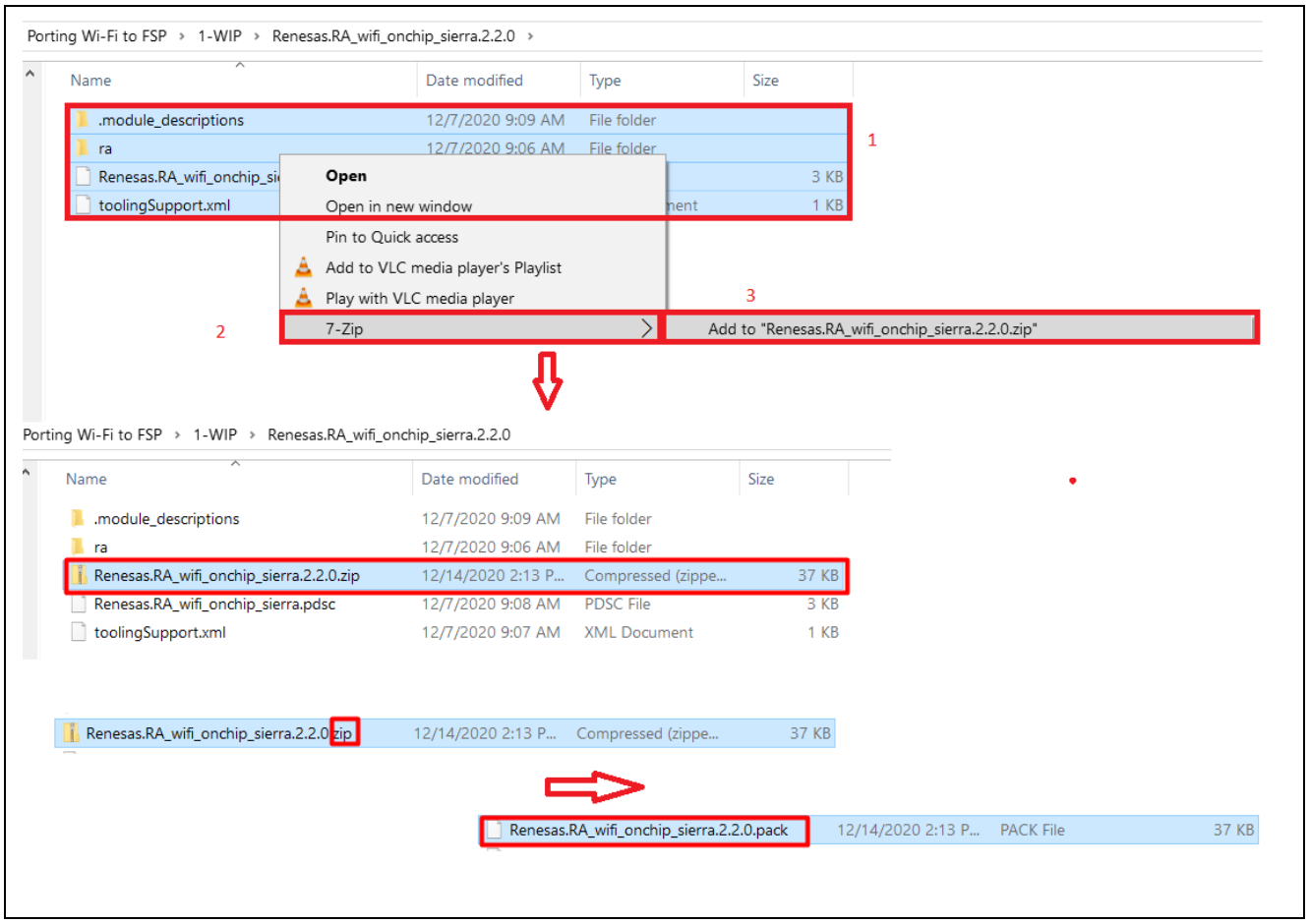


Figure 10. Pack creation from the zip file for the new Wi-Fi driver Pack

4.8.2 Creating User Pack using e² studio Utility

User pack creation using e² studio can be done for the new Wi-Fi driver via the utility tool. This app note will not be covering the pack creation using the e² studio utility tool. The steps for creating the user pack using the e² studio utility is available as part of the e² studio **Help** tab. More details on the usage steps can be found under the e² studio help section by searching for “Creating a RA CMSIS user pack”.

4.9 Importing New User Pack to the Project

Once the final user pack is built, it needs to be copied to the packs folder. Installing the packs file can be done in two different ways.

- Copy manually the new user pack to the packs folder of the e² studio installation. The Packs folder under the e² studio installation folder is `e2_studio\internal\projectgen\ra\packs`.
- Use the Import feature of e² studio to import the pack (**Project→Right-Click→Import→CMSIS Pack**).

After the packs are installed via manually copying or importing via the e² studio tool, you will notice the e² studio IDE detects the new packs and updates its database.

Note: If the pack files are detected by the e² studio IDE, close and restart the e² studio for the installed packs to take effect.

You can check whether the e² studio installed the pack file successfully, you can check by accessing through **e2 studio →Help → About e2 studio →Installation Details → Support folders → e2 studio Support Area link**. It is available at the support area under the `internal\projectgen\ra\packs` folder. he screenshot for the same is given below in Figure 11.

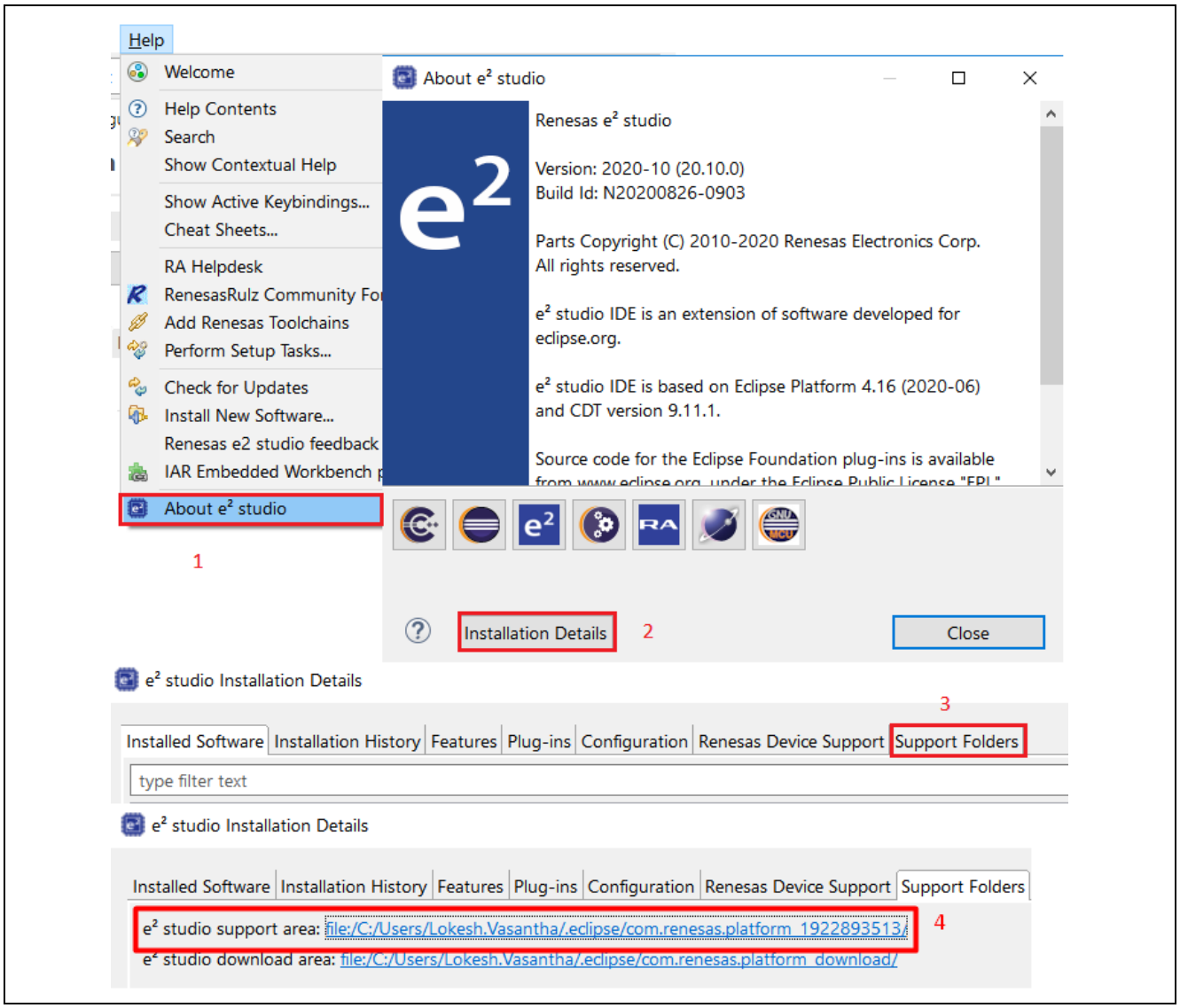


Figure 11. Location of Cached e² studio Packs

In the next section, we show how this Wi-Fi module can be included to the project using the FSP configurator and configure the new module using the property window and changing its configuration parameter values.

5. Building Application with New Wi-Fi Module

After the pack is installed successfully, the new application project can be created by adding the new Wi-Fi module and configuring the module via the FSP configurator.

5.1 FSP Configuration

FSP Configuration for the project is done through the e² studio's graphically guided tool. Refer the FSP User's manual, Section Starting Development, e2 studio User Guide, Configuring a Project which has the details on how the FSP configuration can be done for the individual modules and different configuration settings as needed.

5.2 Including the Module to the Project

To include the WiFi Module in an RA Project, in the FSP configurator, under the created thread, choose the **Stacks** tab, **New Thread →New stack →FreeRTOS→Secure Sockets →Secure Sockets on Sierra Wi-Fi**. This will add the Sierra Wi-Fi module to the thread. The screenshot of adding the Wi-Fi module and added Wi-Fi module to the project is shown below in the Figure 12 and Figure 13.

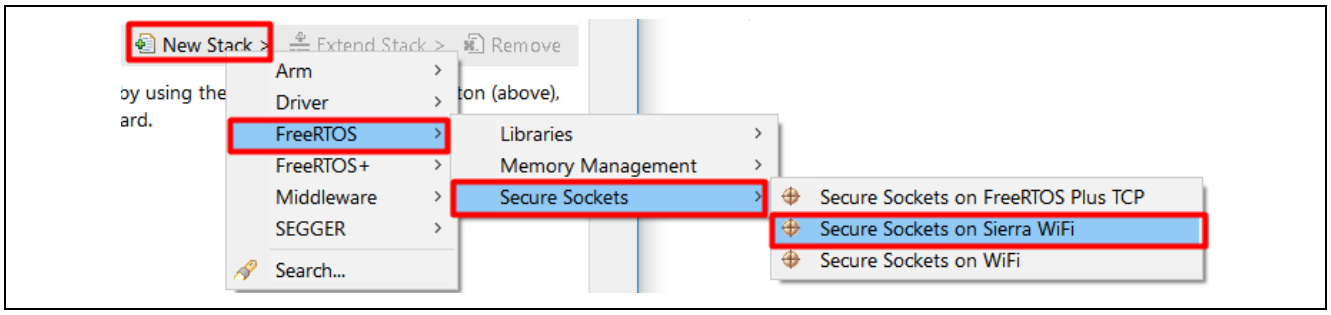


Figure 12. Adding Sierra Wi-Fi Module to the Project

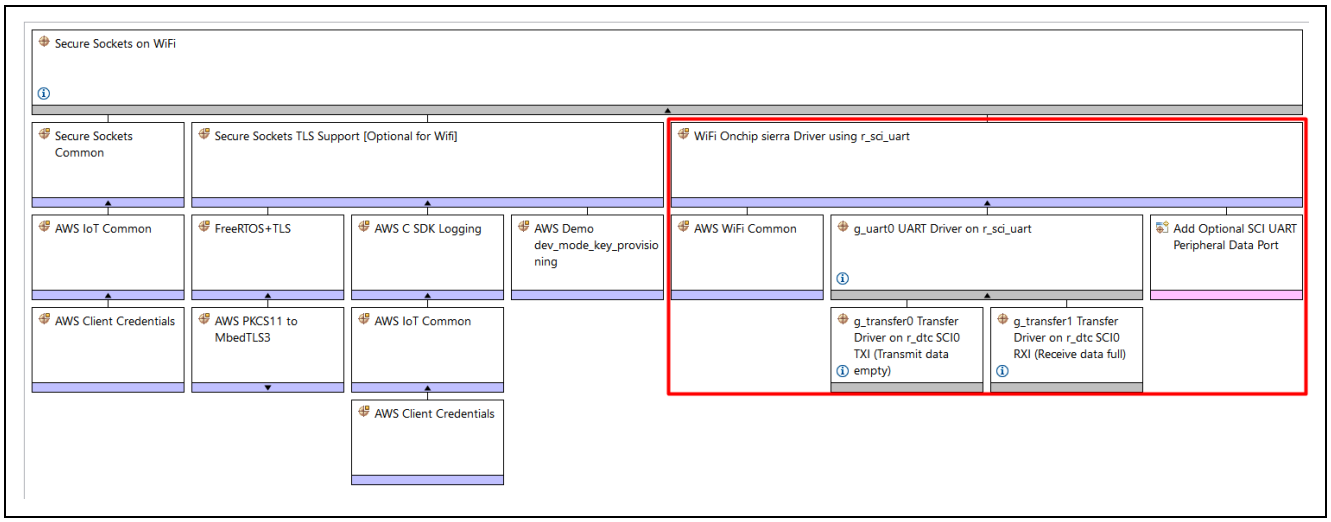


Figure 13. Sierra Wi-Fi Module Added to the Project

5.3 Module Configuration

Adding the module to the project is the first step. Module-specific configuration is the next step needed to make the module operate in the desired way. For each block of the module, there is associated property window where in the configuration values can be changed from the default values.

The details of individual module configuration for each block are not explained here. More details can be found from the included project. Also, for more details, refer to the FSP user manual for the Wi-Fi module configuration.

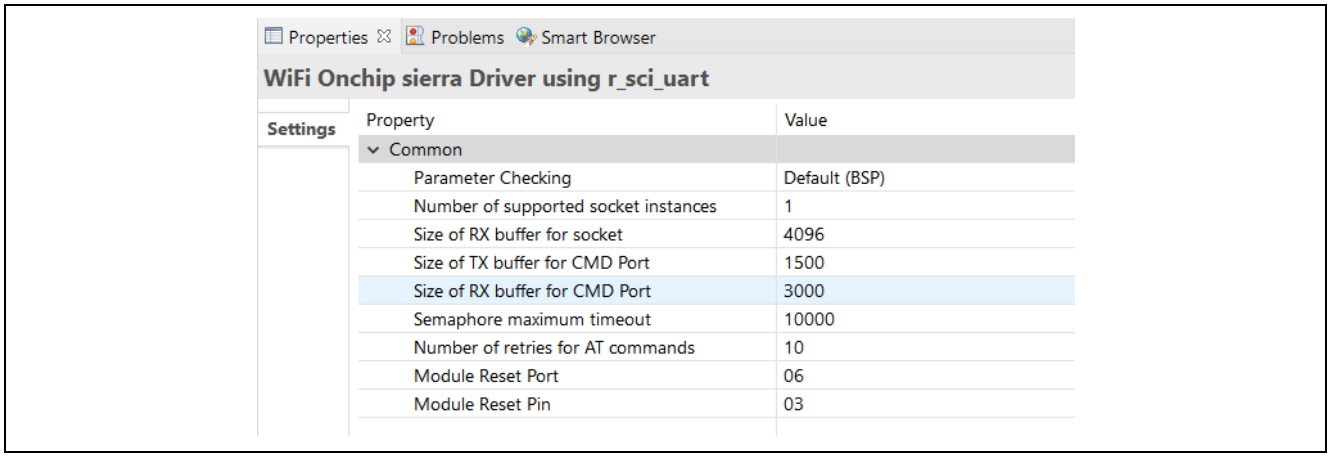


Figure 14. Properties Window for Modifying the Wi-Fi Module's Configuration

6. Importing and Building the Project

Prior to importing the project bundled as part of the application note, make sure that the new pack is installed to the packs folder

- Add the included user pack to the packs folder `e2studio\internal\projectgen\ra\packs`
- To import the included project, follow the standard Import steps documented as part of the *RA Flexible Software Package Documentation*, Starting Development, e² studio Users Guide , Importing an Existing Project into e² studio.

After the importing the project, make sure the FSP configurator shows the new module without any warning or errors.

Open the `wifi_app.h` header file under `src` folder make the changes to SSID and password as applicable to your access point.

```
#define WIFI_SSID "Renesas"
#define WIFI_PW   "@Renesas123"
```

Generate the Project content and build the project. The build should be error free and warning free with reference to the application code.

This completes the successful importing and building the module.

7. Running the Application

Before running the application on EK-RA6M3, it is necessary to make the connection shown in this section.

7.1.1 Board Setups

Before running the project make sure the Sierra Wireless BX310x Development Board is connected to the PMOD1 of the EK-RA6M3 board using the connection diagram as shown below in Table 5.

Note: Sierra Wireless BX310x Development Board has a jumper setting to select the UART or the USB (connecting to PC). Make sure the UART setting is selected.

The BX310x development board provides two sources for accessing the BX310x modules.

Table 3. 4-wire UART interface:

USB—Accessed via USB (UART FTDI IC)	Can be used for Debugging and testing in the initial stages of validation of the module
UART header	Accessed via UART Header – This is the interface used with RA6M3

A 3-pin header is used to select the UART interface.

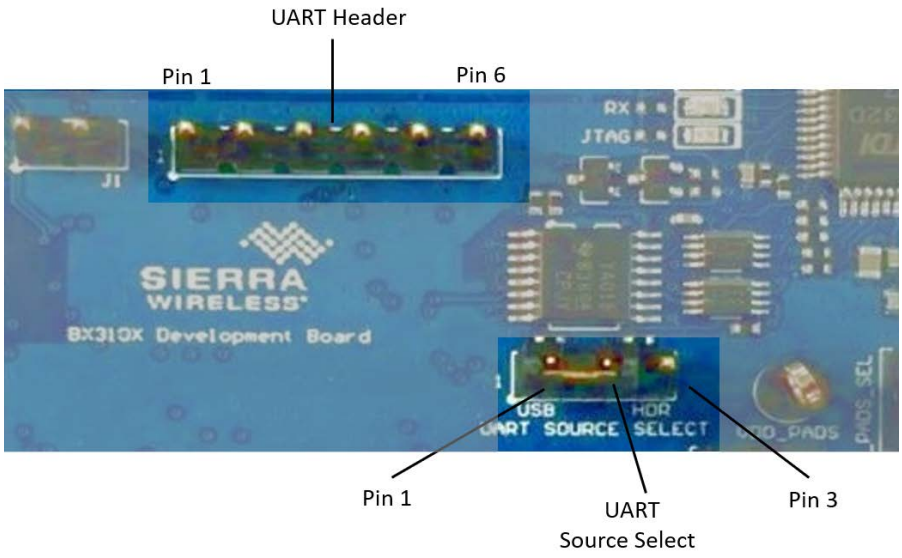


Figure 15. UART Header and UART interface selection

Table 4. UART Details of the Sierra Wireless Module

Component	PCB Label	Description
3-pin header	UART SOURCE SELECT	Selects the hardware source used to access the BX310x module's UART interface, based on jumper position: • Pins 1&2—USB via FTDI USB UART IC • Pins 2&3—UART header
UART HEADER Pins		
6 pin UART header Pins		
1	None	Ground
2	None	UART Clear to Send
3	None	No connect
4	None	UART Transmit Data
5	None	UART Receive Data
6	None	UART Ready to Send

Note: UART signals named from host perspective, with module acting as slave device (e.g. UART_HD_TXD is Host Transmit/Module Receive)

Table 5. UART Connection diagram from BX310x Development Board to RA6M3 PMOD1

UART Header Pin	PMOD1 Pin
1- (Ground)	5- (Ground)
2- (CTS)	
3- (No Connection)	
4- Transmit	2- Receive
5- Receive	3- Transmit
6- RTS	

To power the RA6M3 and Sierra Wireless BX310x Development Board, connect the USB cable to the micro USB connector of EK-RA6M3 kit (J10) and Sierra Wireless BX310x Development Board (micro-B USB connector).

7.1.2 User Interface

Once the application is running, open the RTT viewer to see the banner message and initialization sequence log messages at each step. If the Wi-Fi module is connected to the access point, the DHCP client gets the IP address and it will be displayed on console. User can ping from the PC to the new IP address and validate the connectivity.

Also, upon successful connection you can notice the blinking green LED, indicating the connection to the access point.

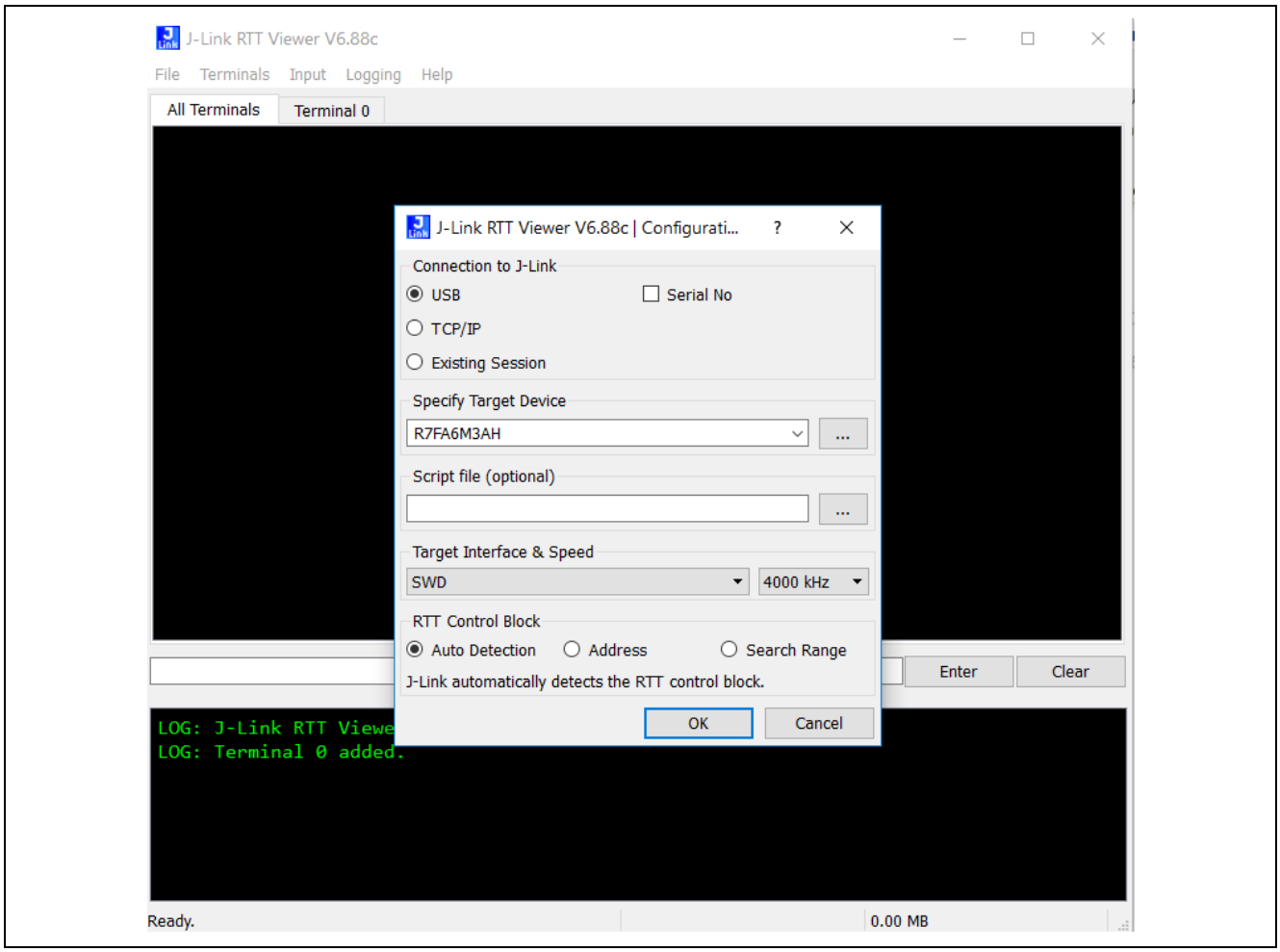


Figure 16. Snapshot of RTT viewer settings

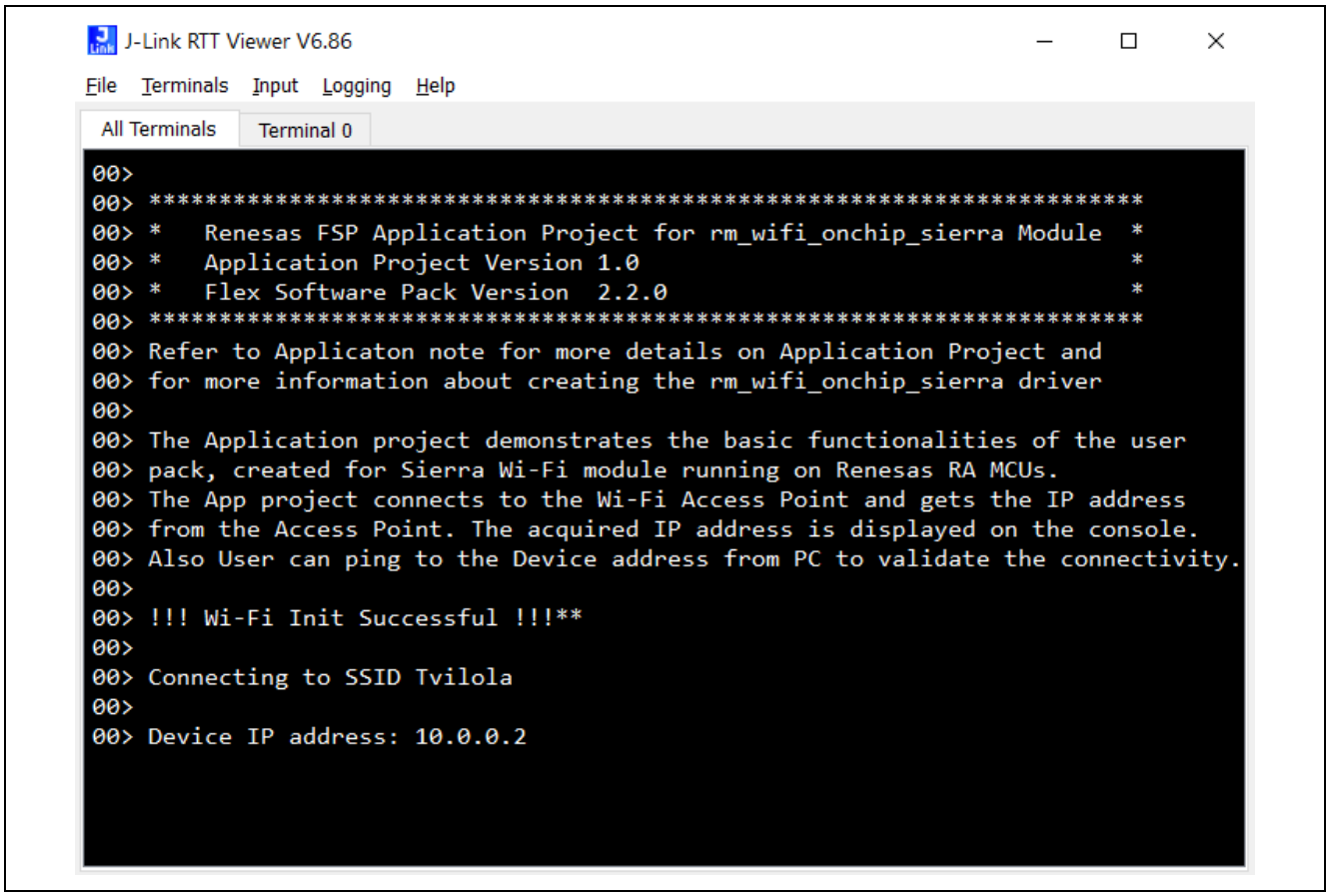


Figure 17. Snapshot of the User Interface for the Application

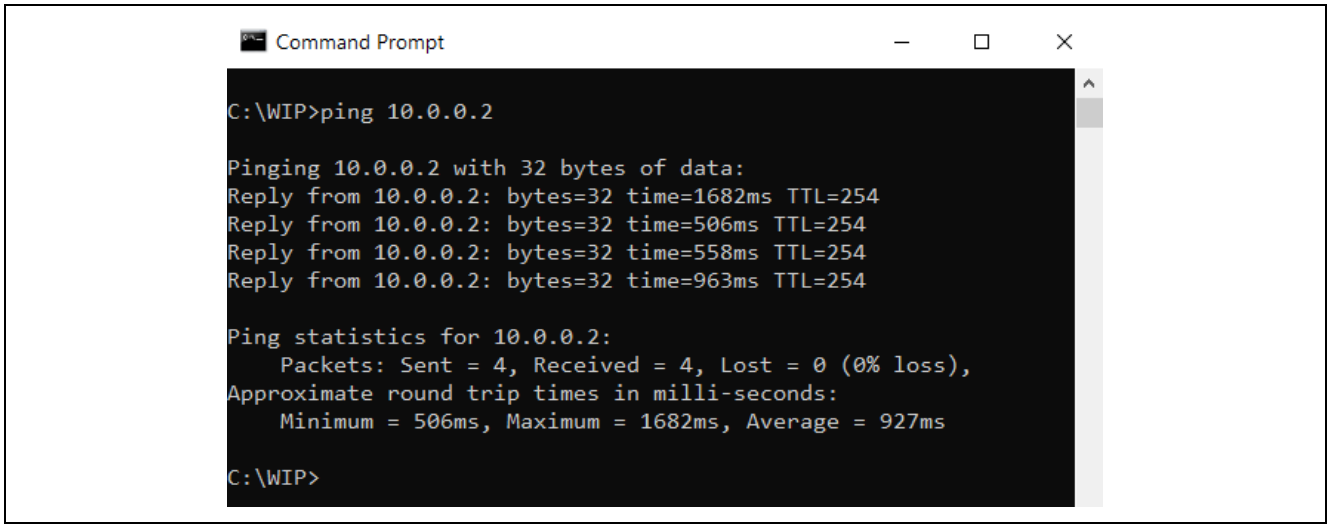


Figure 18. Snapshot of Validating the Connectivity to the Module

8. Known Issues

The Sierra Wireless BX310x Development Board doesn't provide a AT command for a DNS feature. But there is a workaround to make it available for the application given as part of Section 8 References.

9. References

- RA Flexible Software Package Documentation: https://renesas.github.io/fsp
- CMSIS-Pack Documentation: https://www.keil.com/pack/doc/cmsis/Pack/html/index.html
- FreeRTOS Stream Buffer: https://www.freertos.org/RTOS-stream-message-buffers.html
- Suggested Workaround for DNS Client : *Application Note for Setting Up a DNS Service in a Private Network-Rev1.1*: https://forum.sierrawireless.com/uploads/short-url/cDpnrH63tlv7jBIsy0s5jjcgNPn.pdf
- Custom BSP Creation : https://en-support.renesas.com/knowledgeBase/19427072
- RA6M3 MCU : https://www.renesas.com/us/en/products/microcontrollers-microprocessors/ra-cortex-m-mcus/ra6m3-32-bit-microcontrollers-120mhz-usb-high-speed-ethernet-and-tft-controller

Website and Support

Visit the following vanity URLs to learn about key elements of the RA family, download components and related documentation, and get support.

RA Product Information	www.renesas.com/ra
RA Product Support Forum	www.renesas.com/ra/forum
RA Flexible Software Package	www.renesas.com/FSP
Renesas Support	www.renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec.16.20	-	Initial version

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.