
# Instruction Manual for the wolfSSL Example Application

Target: Renesas RSK+RX65N-2MB
RTOS: FreeRTOS+ IoT libraries

## TABLE OF CONTENTS

## WHAT IS THIS DOCUMENT FOR?

This document is an instruction to add wolfSSL TLS library and to run an example program on the Renesas RSK+RX65N-2MB. The target MCU is expected to be used with a real-time OS when the product is installed. Therefore, this example program is provided in a configuration that uses FreeRTOS and FreeRTOS + TCP. The steps for generating and executing the program as a new project of e² studio, an IDE made by Renesas, is explained below.

## EXAMPLE PROGRAM STRUCTURE

The FreeRTOS kernel and FreeRTOS + TCP protocol stack are required to execute this example program, but they are automatically prepared when creating a new e² studio project. A script that is automatically executed when the project is created, downloads the FreeRTOS-related source files from the GitHub and configures the settings necessary for operation on the evaluation board. The downloaded FreeRTOS IoT Libraries include several demo applications, and the demo application selected from them is configured to be executed.

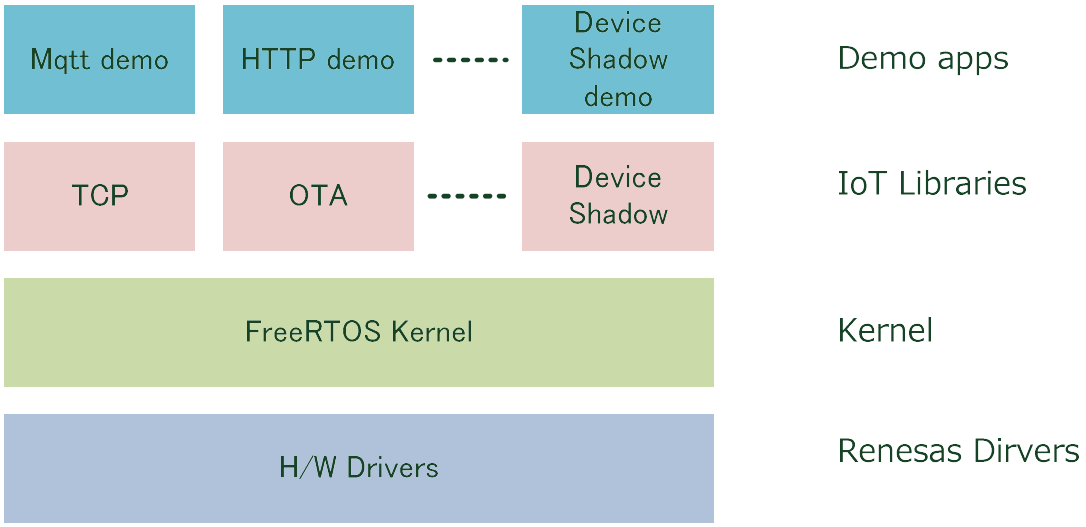


fig.1. Original Structure of the FreeRTOS+IoT libraries demos

The wolfSSL example program adds the wolfSSL library, the wolfSSL demo application, and the FIT components required as the H/W driver to this configuration, and configures it as shown in fig.2.

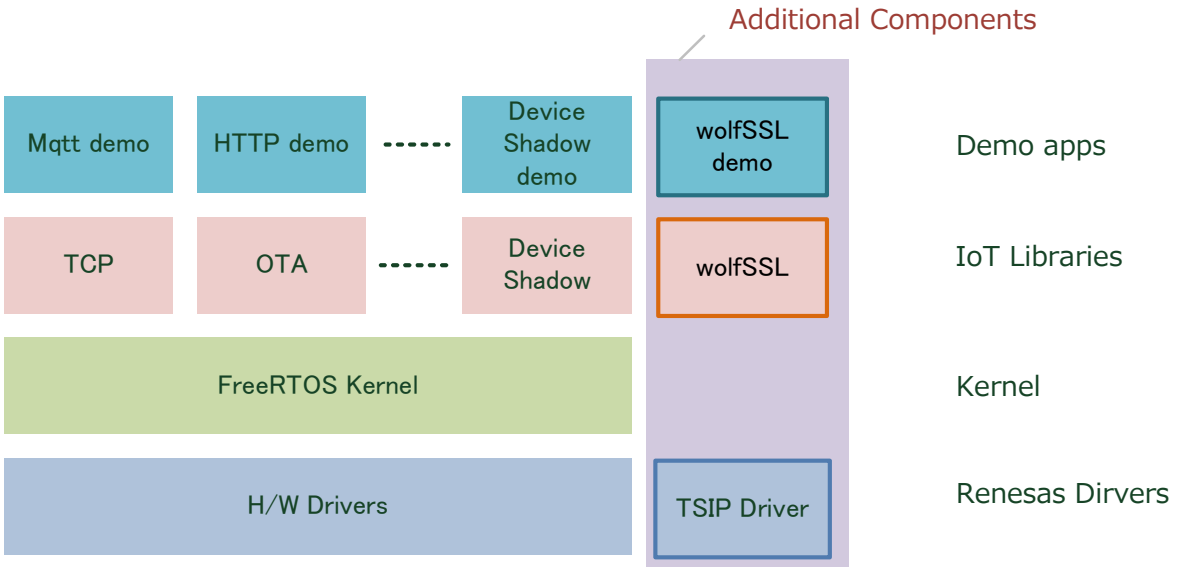


fig.2. The extended structure by adding wolfSSL demo

The added wolfSSL demo application runs as a task on the FreeRTOS kernel and utilizes the TCP protocol stack as a communication channel. In addition, the wolfSSL library supports TSIP. By replacing some of the encryption and TLS functions implemented by the wolfSSL library as software with H/W (TSIP), it is possible to significantly improve the processing speed.

## REQUIREMENTS FOR BUILDING AND RUNNING THE EXAMPLE PROJECT

Tools and components required for the building and execution of this example program:

1. e² studio Version 2021-10 or later
2. CC-RX Tool Chain V3.04 or later
3. TSIP v1.17 or later
4. RTOS v202107.00-rx-1.0.1 or later
5. wolfSSL v5.5.4 or later

## PROCEDURE FOR CREATING A WOLFSSL EXAMPLE APPLICATION PROJECT

The following steps are roughly required to execute this example program：

1. Create an executable project including FreeRTOS+IoT libraries on e² studio

2. Settings for the target MCU and the evaluation board
3. Adding FIT components and their update
4. Copy wolfSSL package
5. Section settings
6. Adding wolfSSL library project and wolfSSL demo files
7. Execution of wolfSSL demo

From now on, the above steps will be described in that order.

## 1. CREATE A NEW EXECUTABLE PROJECT

Launch e² studio and specify a folder to be its workspace. The folder will be the base folder of the project. Here after, the folder is referred as **<base>** in this document.

### CREATE A NEW FREERTOS PROJECT

Selecting "File" menu > "Import..." > "General" > "Renesas GitHub FreeRTOS(with IoT libraries) Project" will show you dialogs below.

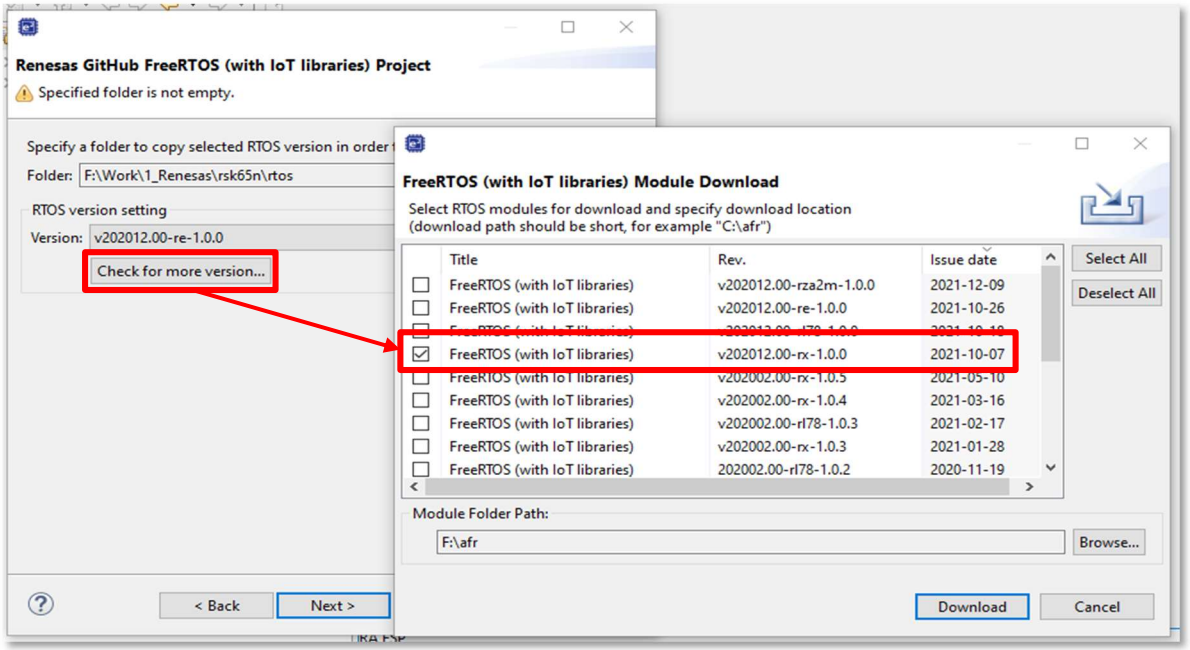


fig.3. Steps to choose FreeRTOS module to download

On a FreeRTOS(with IoT libraries) Module Download dialog, select FreeRTOS with revision "v202107.00-rx-1.0.1" or later. Next, specify the download destination folder. Note that the folder path should be short enough to avoid errors where the path length is too long.

Once the download is complete, you will be able to see the location of your project and the version of FreeRTOS, shown in fig.4.

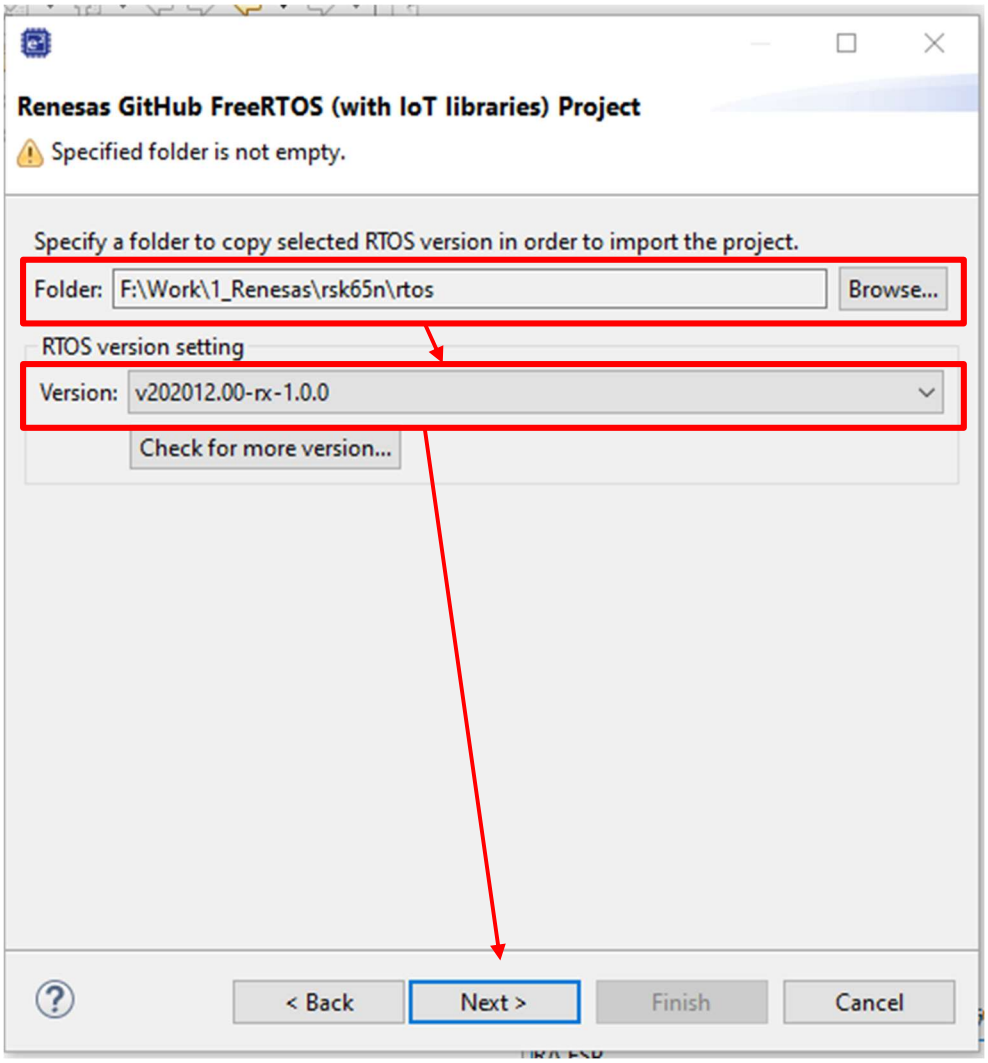


fig.4. The project location and the version of FreeRTOS to use

The folder is identical to the workspace folder and is to be referred as <base>. For the next step, you need to choose a demo type for the target MCU and the compiler.

The dialog show in fig.5 lists up products ready for import to your <base> folder. The list contains three types (aws_demo, aws_test and boot_loader). Pick up

"aws_demos(...\projects\renesas\rx65n-rsk\e2studio\aws_demos)"

from the list.

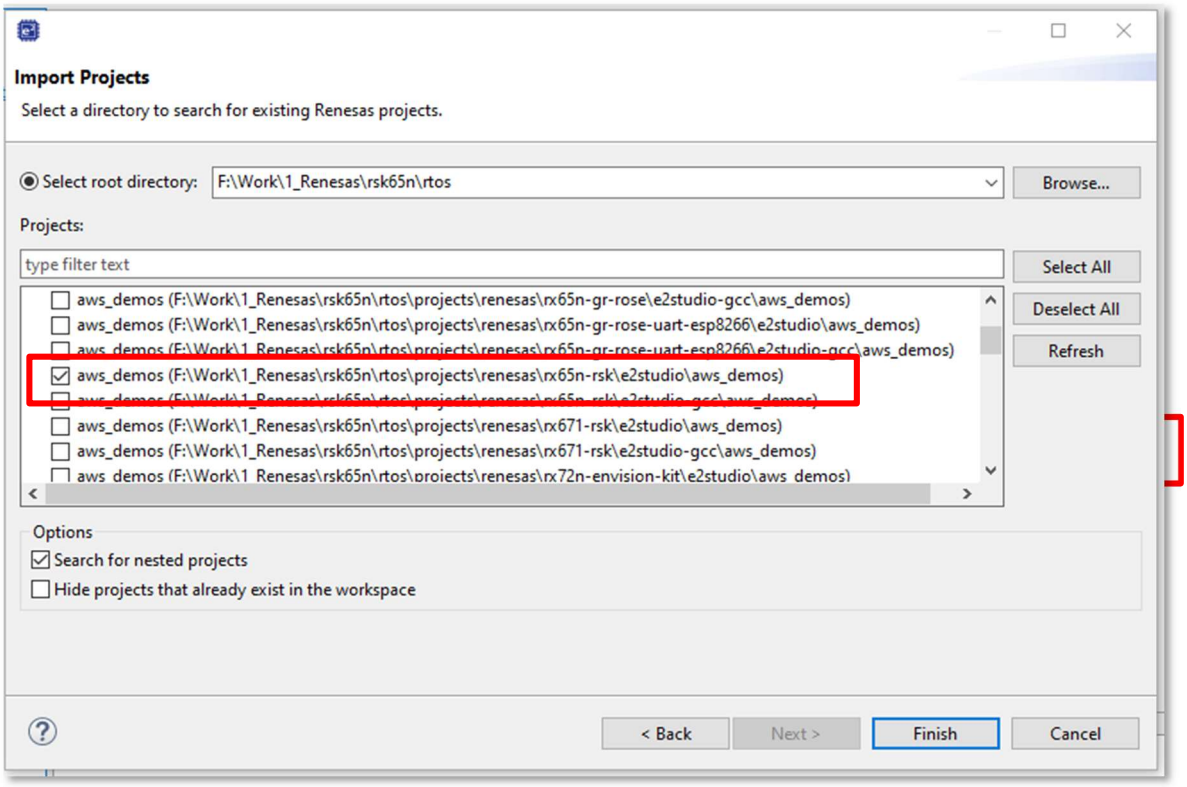


fig.5. Import Projects dialog

Script runs to extract selected source files and organize project folders in the project explore pane.

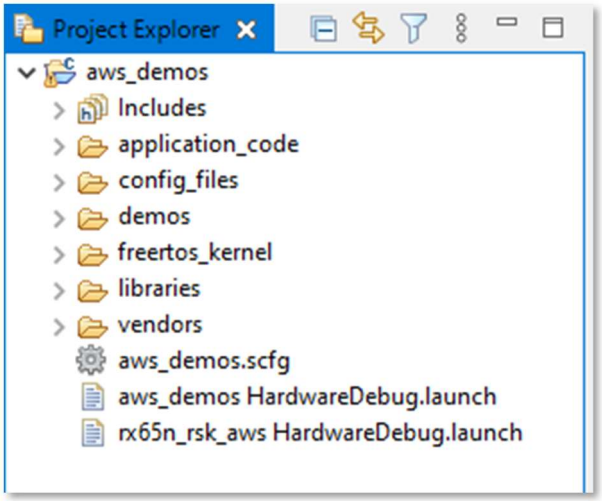


fig.6. Imported and organized aws_demos project

## 2.DEVICE INFO SETTINGS

Before adding FIT modules, set the board and device information. Double-click aws_demos.scfg on the Project Explorer to open the Smart Configurator Perspective and select the Board tab at the bottom to display the "Device selection" settings pane.

In the "Board" type selection list, choose "RSKRX65N-2MB(TSIP)(V1.00)"or later. If no board type listed, you can get them by clicking the link named "Download more boards...". There are several similar files to download, so be sure to select one that has "TSIP" in the file name.

When you choose the board, "Device" is filled with automatically.

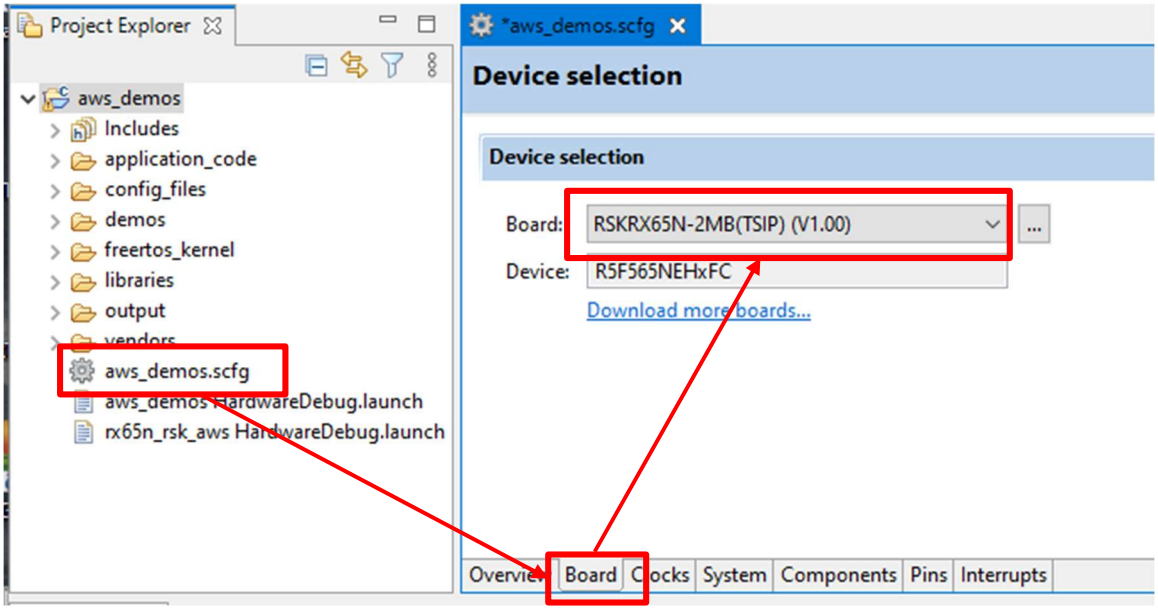


fig.7. Device selection

## 3.ADDING FIT MODULES

At this point, the project has the source files for FreeRTOS, the IoT library and the demo application. In addition, the source files of the necessary FIT components (drivers provided by Renesas) have already been generated. However, some FIT component libraries need to be downloaded and obtained from the Renesas site.

Double-click aws_demos.scfg on the Project Explorer to open the Smart Configurator Perspective and select the "Components" tab at the bottom to display the "Software components configuration" pane. Then push the icon to show "Software Component Selection" dialog shown in fig.8.

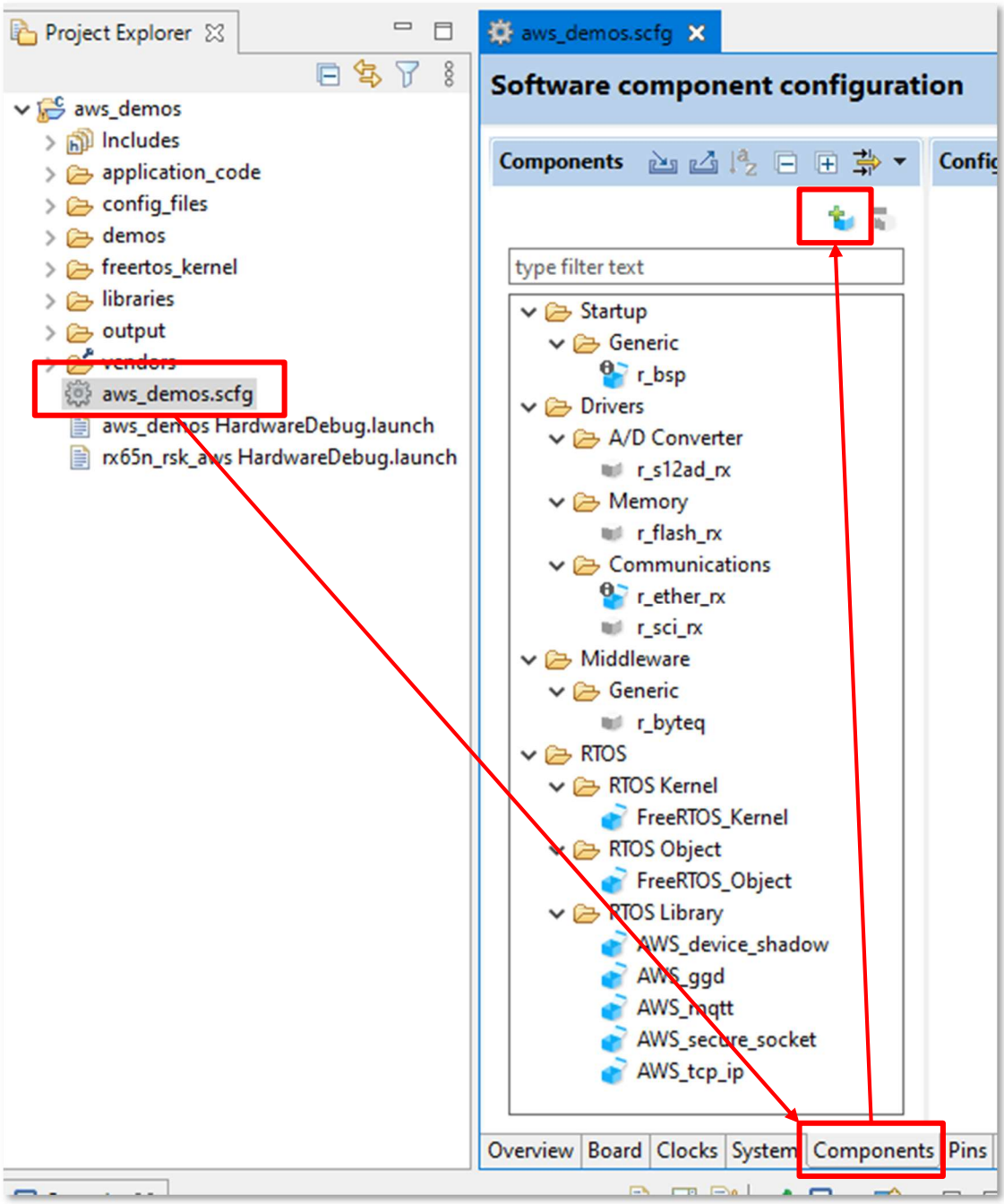


fig.8. Software components configuration

In the "Software Component Selection" dialog, find and choose one FIT component to add the project. If no components are listed in the dialog, it means that you need to download FIT components from Renesas site into your PC. Click the "Download the latest FIT drivers and middleware" link.

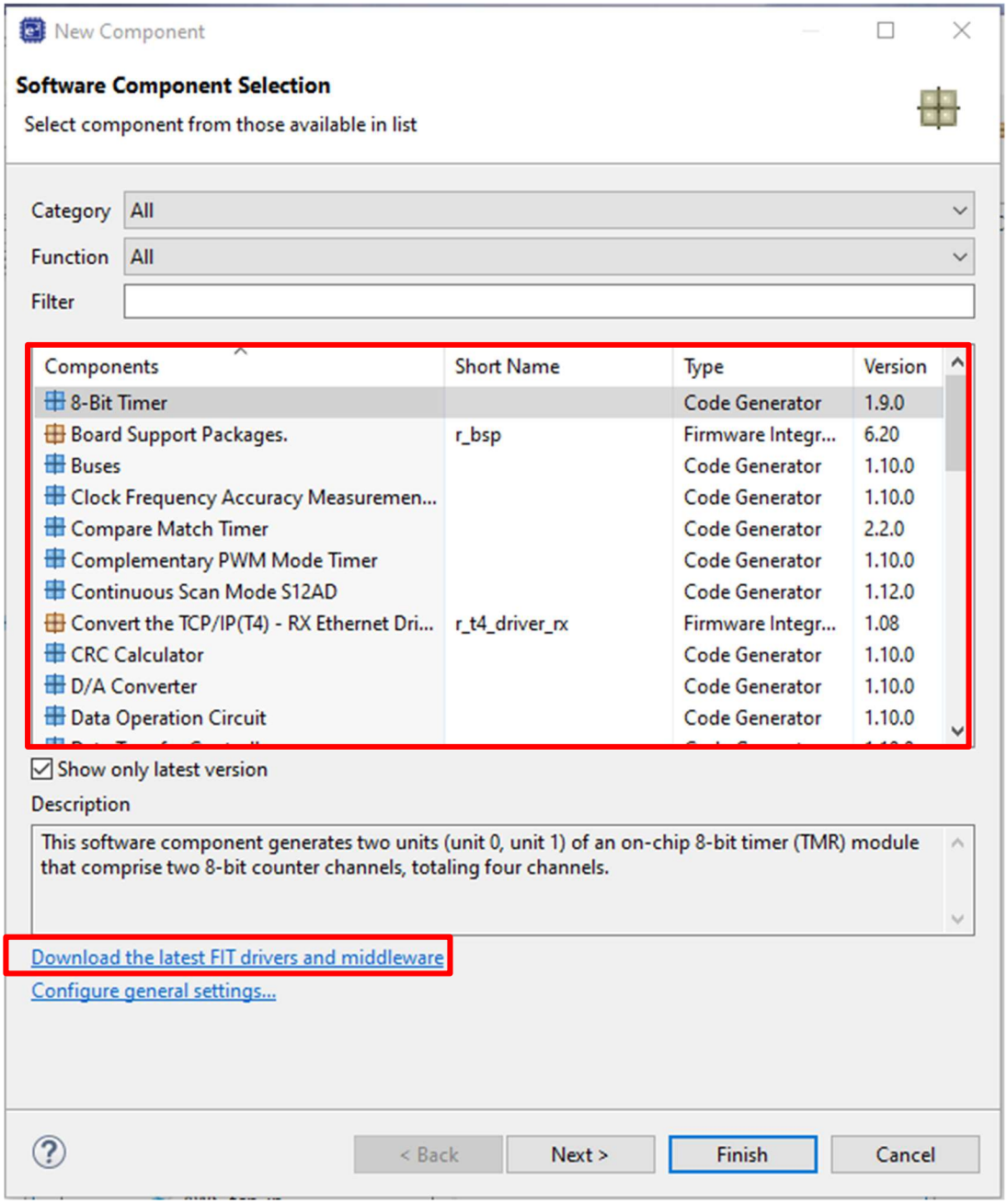


fig.9. Software Component Selection Dialog

If you have downloaded the latest FIT components in your PC, you can extract the components to add in the list by specifying the function type of the component.

wolfSSL demo requires following FIT components to add the project:

1. TSIP component(r_tsip_rx)
2. CMT component(r_cmt_rx)

Let's take an example of how to add a TSIP component. Select “Security” function category in the following dialog lists up TSIP in the component list. Click the TSIP in the list and then push "Finish" button to add the component. Since you can add only one FIT component at a time, repeat the same steps to add other components.

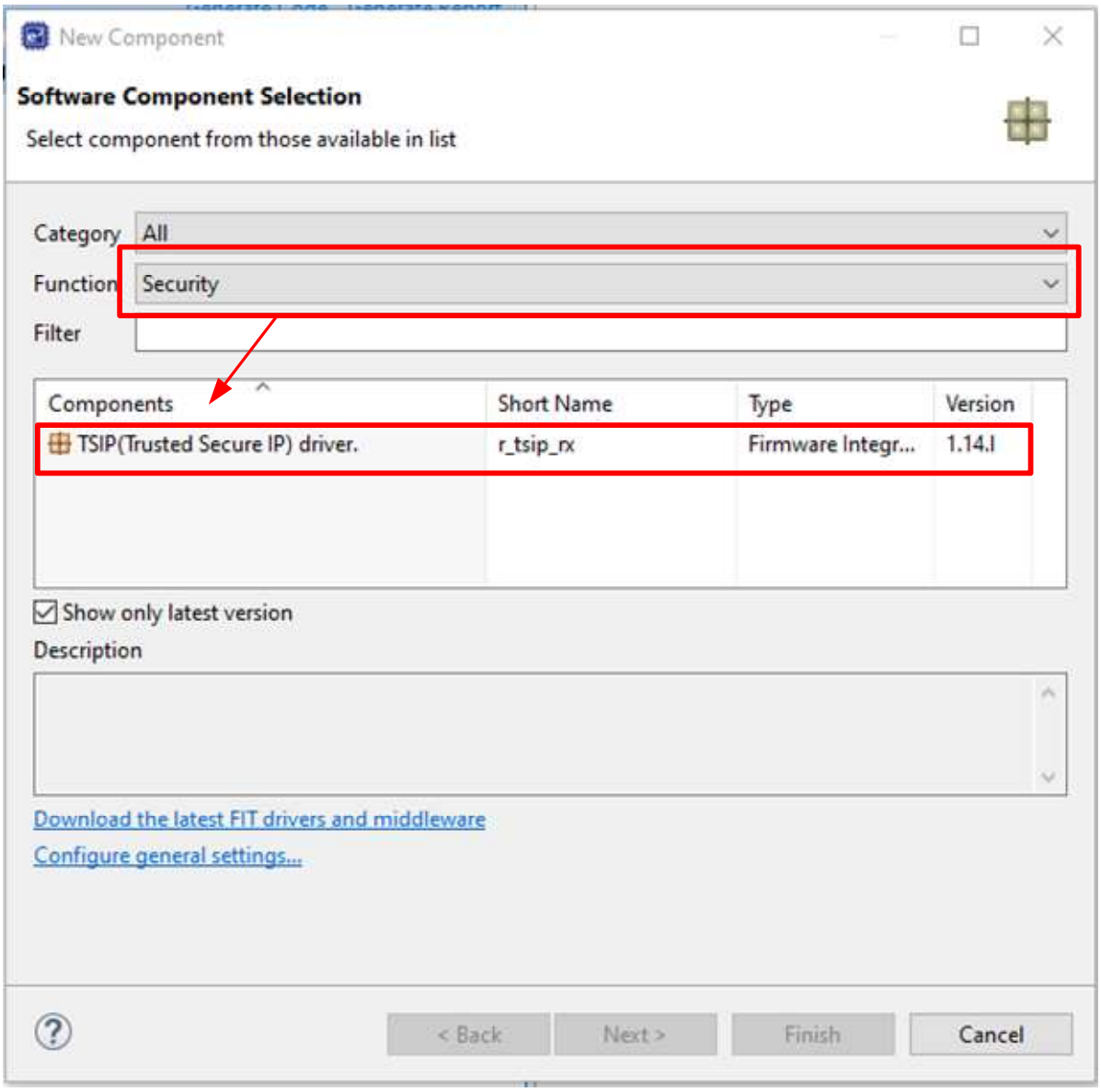


fig.10. How to add TSIP

After adding TSIP and CMT driver to the project, you can see those components are listed in the components pane.

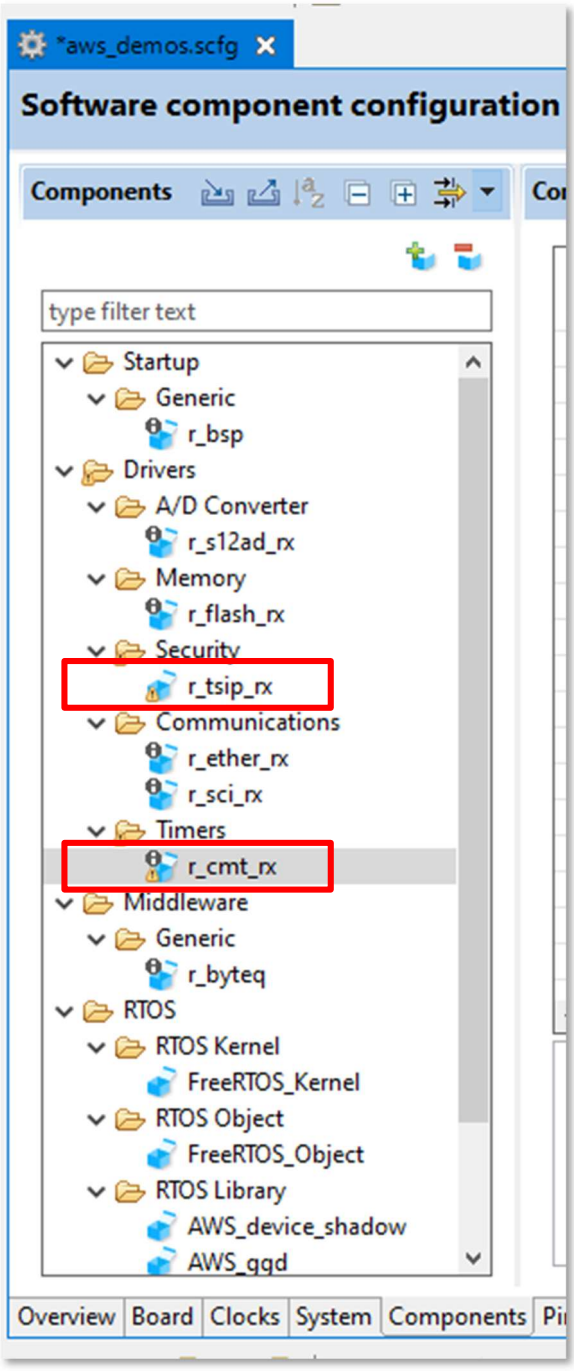


fig.11. Component list

After specifying the required FIT components, let SMC(SMart Configurator) generate source files. Push the button on the top right of the "Software component configuration" pane. Generated files are added to the aws_demos project.

## 4.COPY WOLFSSL PACKAGE

If you have a wolfSSL package downloaded from the GitHub or wolfSSL download page, it has version string in its top folder name (such as wolfssl-5.1.1-stable) like as the right box of fig.12. Copy the entire package under the <base> folder with the name "wolfssl".

This is important because both wolfSSL demo and aws_demos refer each other by traversing their path names. Therefore, name of wolfssl top folder and the location should be exact the same as fig.12.

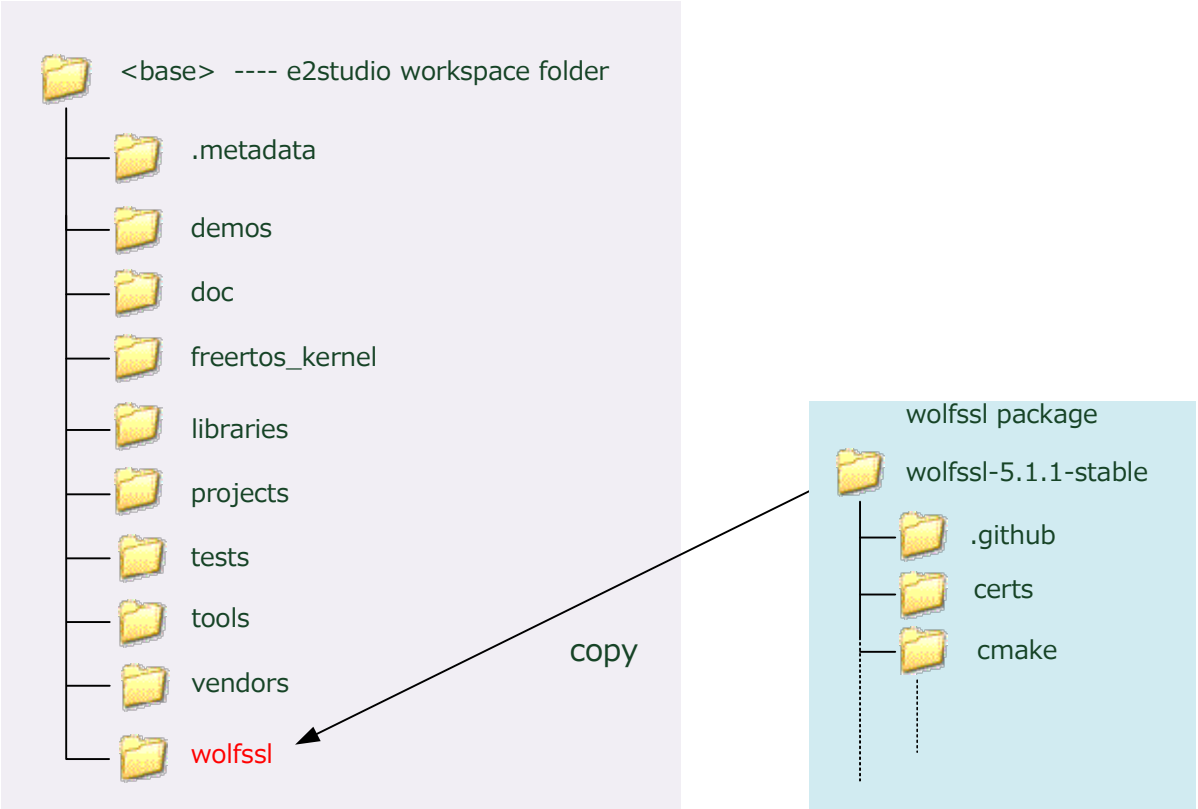


fig.12. folder structure after wolfssl package added

## 5.SECTION SETTING

Section setting in the memory map is necessary. Open the property page of the aws_demos project, then choose "C/C++ Build" > "Settings" > "Linker" > "Section" to show section setting pane. Push the button located in the right most of the pane to show the "Section Viewer" dialog.

Push the "Import..." button to show the dialog for specifying the section setting file to import.

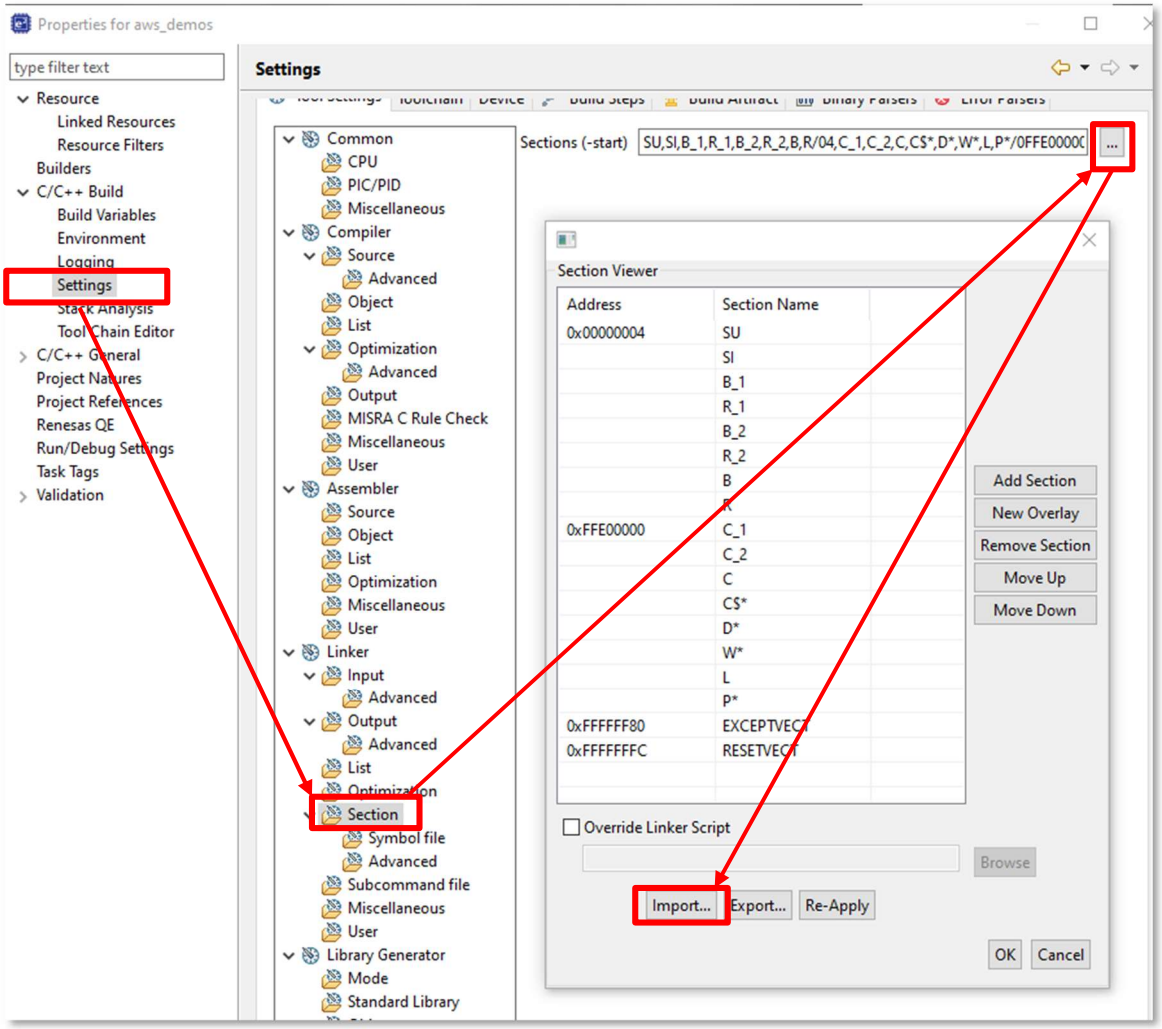


fig.13. How to show the "Section Viewer" dialog

In the dialog for a section file, specify the following file:

<base>\wolfSSL\IDE\Renesas\e2studio\RX65N\RSK\resource\section.esi

## 6.ADDING WOLFSSL LIBRARY AND WOLFSSL DEMO FILES

The next thing you need to do is add the wolfSSL library project and the code for the wolfSSL demo application to the aws_demos project.

### IMPORTING WOLFSSL LIBRARY PROJECT

Then import the project that builds the wolfSSL library into the project explorer on e² studio. The wolfSSL library project is already available as an e² studio project in the wolfSSL package.

On e² studio, selecting "File" menu > "Open a project from the file system" > "Directory" pops up the dialog for specifying a folder including a project file. Specify the following folder:

<base>\wolfssl\IDE\Renesas\e2studio\RX65N\RSK\wolfssl

The dialog finds out a wolfssl project to import.

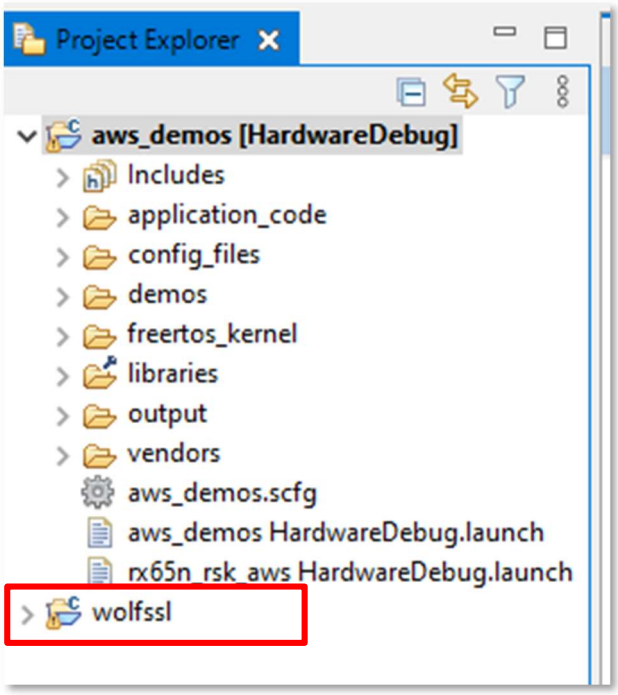


fig.14. Project structure after wolfssl project imported

You can see that the wolfssl project has been added in the project explorer. In the added wolfssl library project, there is nothing to set because the path to the include files generated by the aws_demos project is already set.

## ADDING WOLFSSL DEMO APPLICATION FILES

Add wolfSSL demo application files which work as a kind of FreeRTOS demo task. Point to the aws_demos folder on the project explorer pane, then open the floating dialog by right button click to create a new source file folder named "wolfSSL demo".

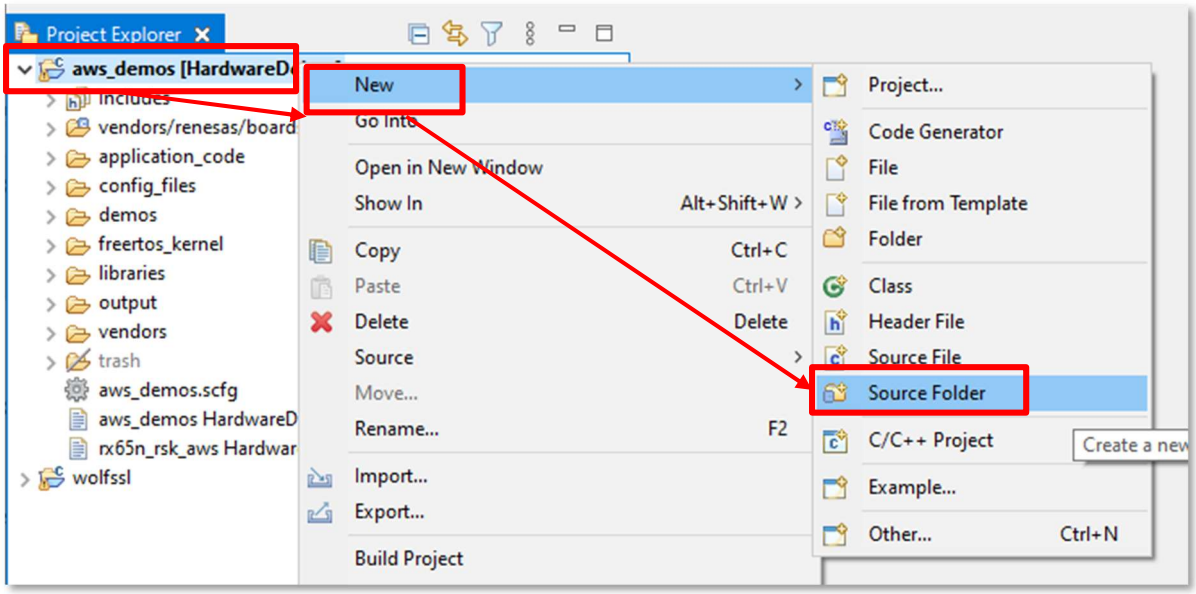


fig.15. Creating a source folder for wolfssl_demo

Open the following folder with explorer and grab all the files (*.c, *.h) in there and drop them on the created "wolfssl_demo" folder in the project explorer pane of the e² studio.

<base>\wolfssl\IDE\Renesas\e2studio\RX65N\RSK\wolfssl_demo

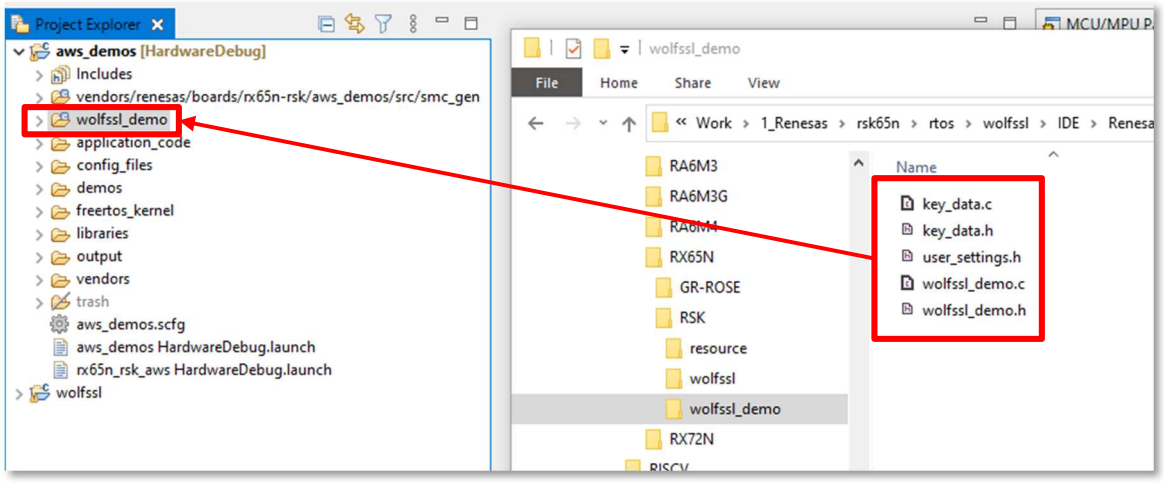


fig.16. Adding wolfssl_demo source files

You will be asked whether you want to process these files by copying or linking. Choose linking.

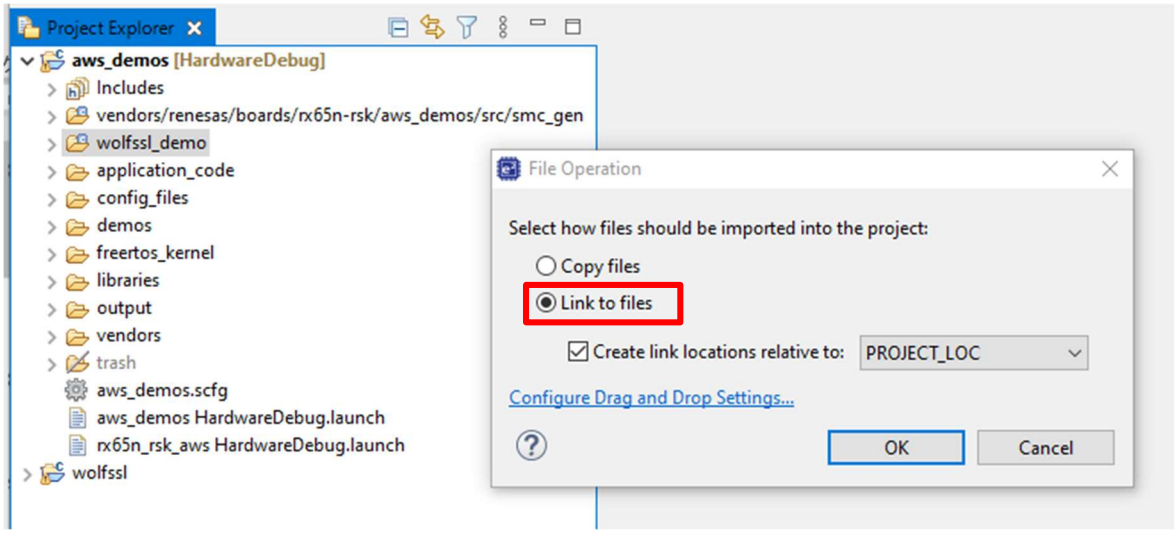


fig.17. Copying files by linking

Add more files below to the wolfssl_demo folder by linking:

1. <base>\wolfssl\wolfcrypt\benchmark\benchmark.c
2. <base>\wolfssl\wolfcrypt\benchmark\benchmark.h
3. <base>\wolfssl\wolfcrypt\test\test.c
4. <base>\wolfssl\wolfcrypt\test\test.h

Finally, you should see the wolfssl_demo folder in the project explorer, as shown in fig.18.

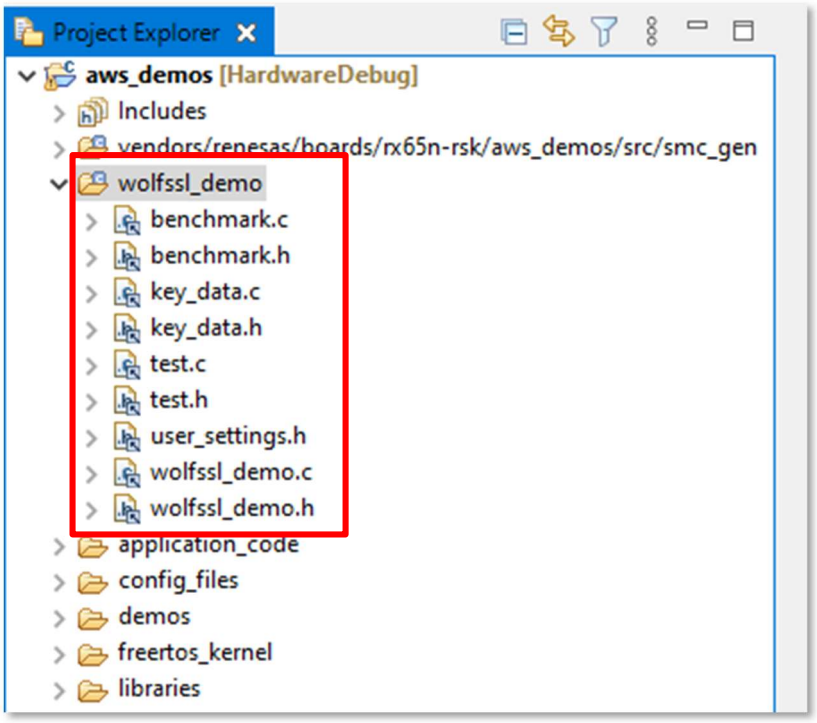


fig.18. files in the wolfssl_demo folder

## ADDING INCLUDE FILE PATHS TO THE PROJECT

Open aws_demos project property setting dialog, then select "C/C++ build" > "Settings" > "Compiler" > "Source" to show "include file directories" pane. Add following include file paths:

- ${ProjDirPath}/../../../../../wolfssl
- ${ProjDirPath}/../../../../../wolfssl/IDE/Renesas/e2studio/RX65N/RSK/wolfssl_demo

## ADDING PREPROCESSOR MACRO

Open aws_demos project property setting dialog, then select "C/C++ build" > "Settings" > "Compiler" > "Source" to show "Macro definition" pane. Add following macro definition:

- WOLFSSL_USER_SETTINGS

This macro definition lets wolfSSL demo application refer the user_settings.h file.

## ADDING LINK LIBRARY

Open aws_demos project property setting dialog, then select "C/C++ build" > "Settings" > "Linker" > "Input" to show "Relocateable files, objects files and library files" pane. Add following library file:

◆ ${ProjDirPath}/../../../../../wolfssl/IDE/Renesas/e2studio/RX65N/RSK/wolfssl/Debug/wolfssl.lib

## 7.ADDING WOLFSSL DEMO AS A TASK

wolfSSL_demo has been added as one of the demo applications to the project but not enabled yet. To do this, enable the demo and register its entry function to the demo runner environment. Open the following configuration file with editor.

<base>\venders\renesas\boards\rx65n-rsk\aws_demos\config_files\aws_demo_config.h

In the file, find "CONFIG_CORE_MQTT_DEMO_ENABLED" macro definition and make it commented out. Instead add definition of "CONFIG_WOLFSSL_DEMO_ENABLED" macro to set wolfssl demo enable.

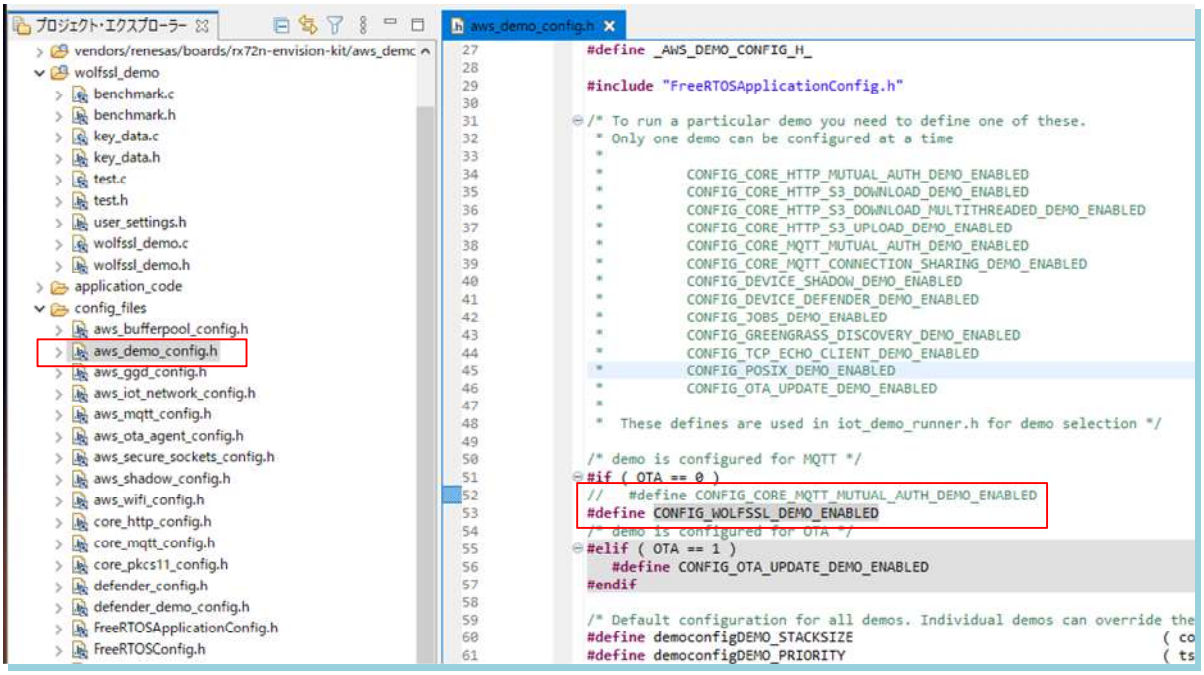


fig.19. Add macro in the configuration file

Moreover, open the following file.

<base>\demos\include\iot_demo_runner.h

Then add the two lines of code below at just before the last #else statement of the file.

```
#elif defined(CONFIG_WOLFSSL_DEMO_ENABLED)
      #define DEMO_entryFUNCTION            wolfSSL_demo_task
```

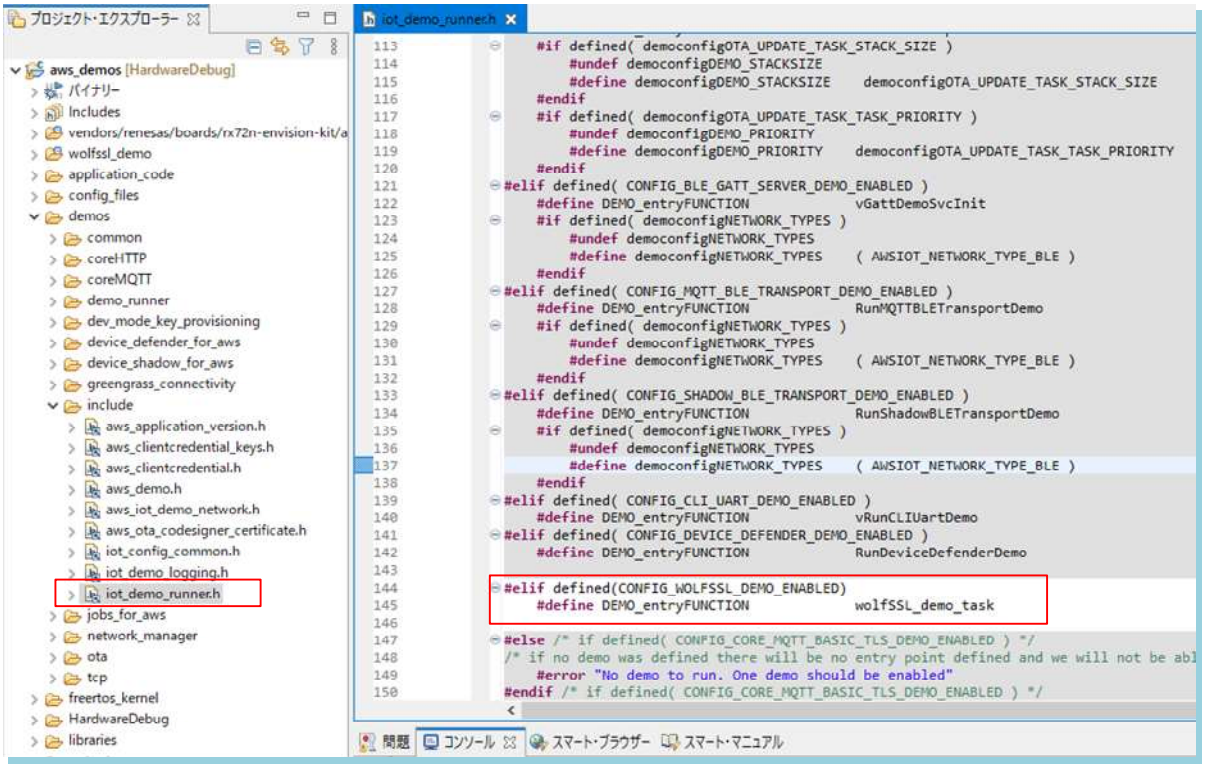


fig.20. Add definition of entry function

Those addition registers the wolfSSL_demo to run.

## EXECUTION OF THE WOLFSSL DEMO APPLICATION

Internally, wolfssl_demo has three different types of demos which is selectable by the following definitions in the wolfssl_demo.h.

- #define CRYPT_TST
- #define BENCHMARK
- #define TLS_CLIENT

By enabling one of the three definition and rebuild of aws_demo project changes the demo.

Whenever you make any change in user_settings.h, rebuild wolfssl project followed by aws_demos project. The execution of the demo can use debugger in the board and monitor the output from demo through “Renesas debug virtual console”.

### CRYPTO-TEST DEMO

You will see the following output in the Renesas Debug Virtual Console when you choose crypto test demo.

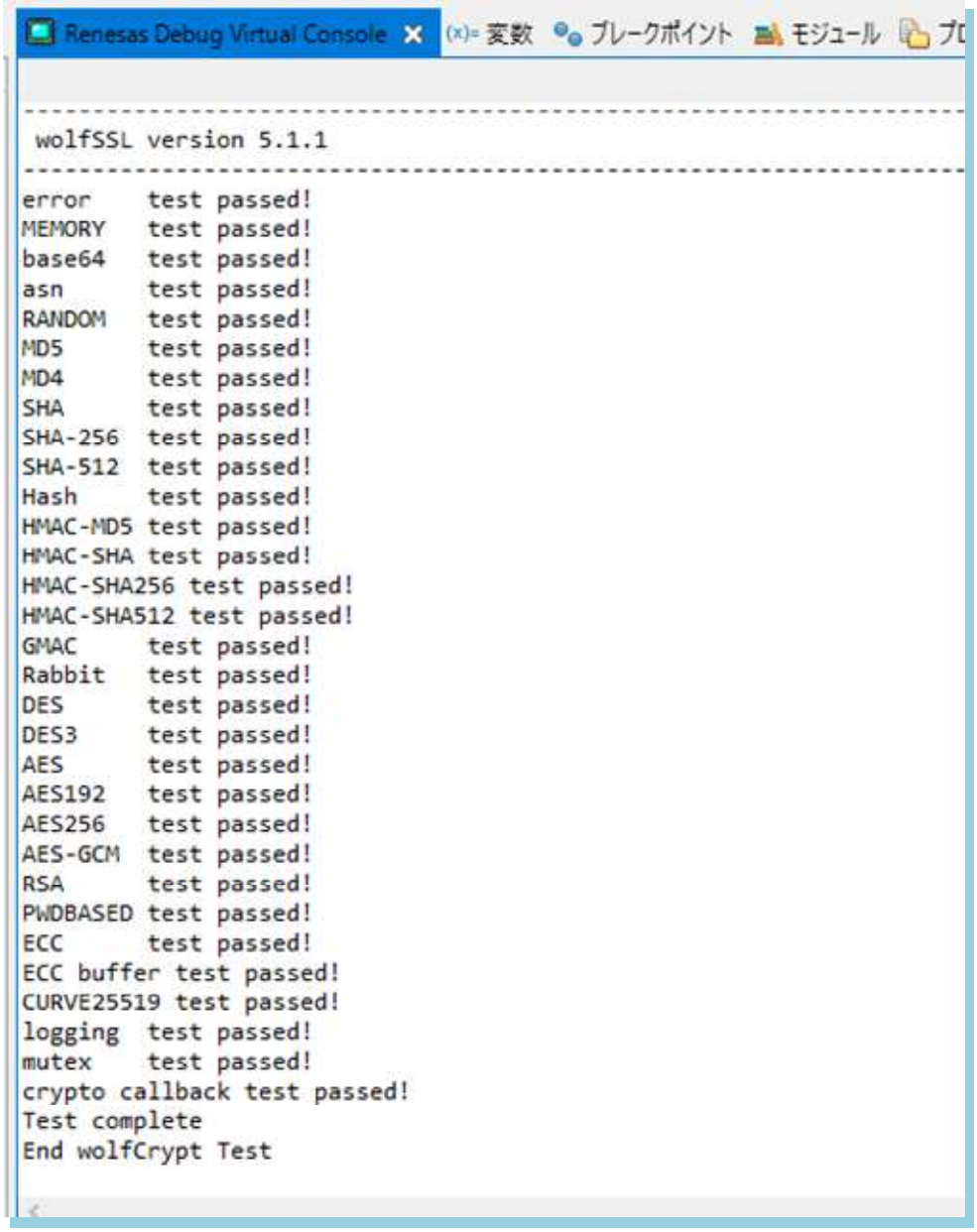


fig.21. Output from Crypt-test demo

## CRYPTO-BENCHMARK DEMO

You will see the following output in the Renesas Debug Virtual Console when you choose crypto-benchmark demo.

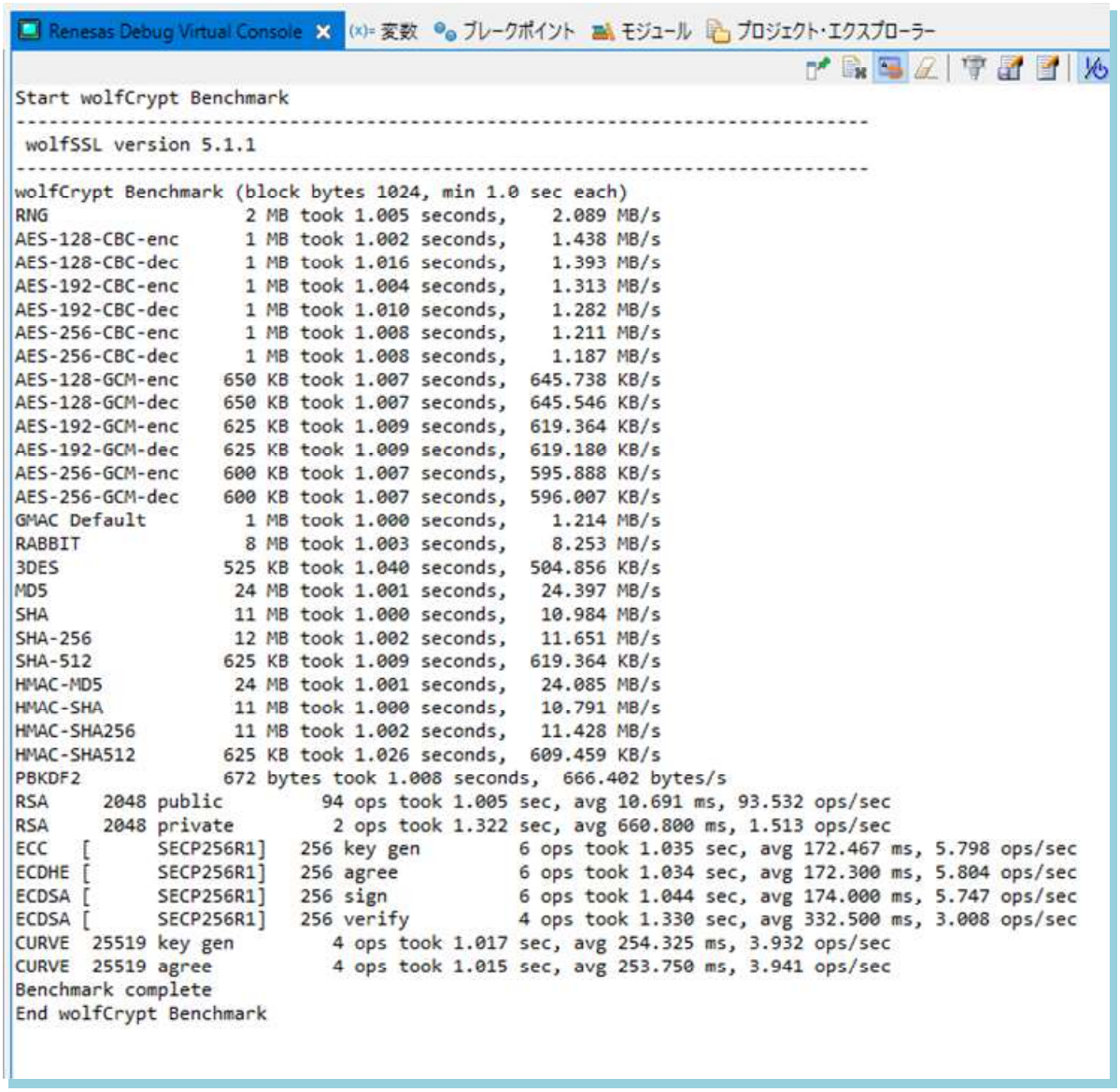


fig.22. output from Crypt-benchmark demo

## TLS-CLIENT DEMO

When you attempt to run TLS_Client demo, prepare the communication opponent program (TLS server program). wolfSSL package provides TLS server example application for this purpose. The application is generated by building wolfSSL package. You can build wolfSSL on Linux (including MacOS and WSL) with gcc installed or build using Visual Studio. The following introduces the build on WSL.

### (1) When using ECDSA certificates

Since USE_ECC_CERT is defined in user_settings.h on the TLS_CLIENT side, it is set to use the Build the server-side sample program accordingly with the following configuration options. Don't forget to give "-DNO_RSA".

```
$ cd <base>/wolfssl

$ ./autogen.sh

$./configure --enable-ecc --enable-dsa  CFLAGS="-DWOLFSSL_STATIC_RSA -DHAVE_DSA
-DHAVE_ALLCURVES -DHAVE_ECC -DHAVE_AESCCM -DNO_RSA"

$ make

$ ifconfig
```

### (2) When using RSA certificates

Also, when setting to use RSA certificates on the TLS_CLIENT side, comment out the USE_ECC_CERT definition in user_settings.h and rebuild. Correspondingly, the server-side sample builds with the following configuration options

```
$ cd <base>/wolfssl

$ ./autogen.sh

$./configure --enable-ecc --enable-dsa  CFLAGS="-DWOLFSSL_STATIC_RSA -DHAVE_DSA
-DHAVE_ALLCURVES -DHAVE_ECC -DHAVE_AES_CBC -DHAVE_AESCCM"

$ make

$ ifconfig
```

If "make" command reports no error, TLS server application is ready to run. Before running the server application, get IP address of the server by typing "ifconfig". You could see IP v4 address in the console. Set the address to the TLSSERVER_IP macro defined in wolfSSL_demo.c.

The IP address of the target board could be set by changing value of the following macros:

- ipconfigUSE_DHCP defined in FreeRTOSIPCOnfig.h
- configIP_ADDR0 ~ configIP_ADDR3 defined in FreeRTOSConfig.h

For debugging purpose or when you get trouble in TCP connection, try to use static IP address for the board.

Run the TLS server application with the following command and options. The option "-v4" specifies to use the TLS1.3 protocol. If "-v3" is specified, the TLS1.2 protocol is used.

```
$ ./examples/server/server -b -v4 -i
```

The sever application waits for the client connection. Run the demo on the board to establish TLS communication with the server application. You will see the following output in the Renesas Debug Virtual Console.

TLS Client attempt to establish TLS connection six times with TLS server using six different cipher suites respectively. The cipher suites included in the output of the TLS client depends on the TLS version and the certificate type setting.

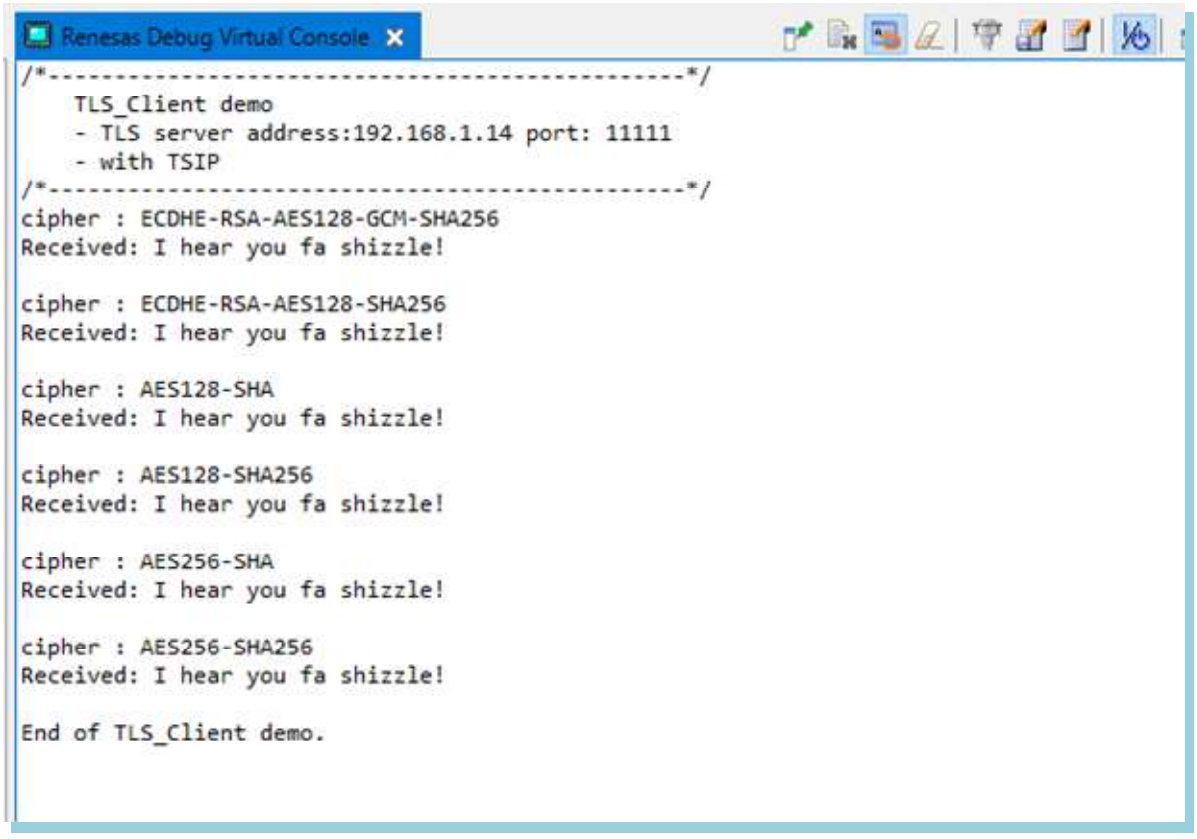


fig.23. output from TLS-Client demo

## THINGS TODO WHEN USING USER'S ROOT CA CERT

The Root CA certificate, the server certificate and the client certificate used in this example application, can be used only for evaluation. If you want to use your own certificate, prepare following items:

1. Provisioning Key
2. RSA key pair for validating RootCA certificate
3. The signature generated by the RootCA certificate with the private ky in 2 above.
4. Private key corresponding to the public key included in the client certificate.

Refer to the manual provided by Renesas for how to generate them.

## REQUIREMENTS FOR CLIENT AUTHENTICATION

wolfSSL supports client authentication as follows:

- Both ECDSA certificates and RSA certificates are handled by TSIP.

(1) Loading client certificate

Use wolfSSL_use_certificate_buffer or wolfSSL_CTX_use_certificate_buffer to load client certificate.

(2) Loading client private key/public key

Type of the client certificate decides the keys to be loaded.

a) ECDSA certificate:

Load private key using tsip_use_PrivateKey_buffer.

b) RSA certificate:

Load private key using tsip_use_PrivateKey_buffer.

Load public key using tsip_use_PublicKey_buffer.

Note. In case of RSA certificate, the public key will be used for internal verification of signature process.

(3) How to generate encrypted keys

The keys (private and public keys) to be loaded should be encrypted-key format. Those keys could be generated with Renesas Secure Flash Programmer or

SecurityKeyManagementTool. Refer the section 7.5 and 7.6 of the application note named "RX Family TSIP Module Firmware Integration technology" how to operate above key wrapping tool.

(4) Macro to be defined

Define "WOLF_PRIVATE_KEY_ID" in your user_settings.h.

## LIMITATIONS

WolfSSL, which supports TSIPv1.17, has the following functional restrictions.

1. Message packets exchanged with the server during the TLS handshake are stored in plaintext in memory. This is used to calculate the hash of handshake messages. The content will be deleted at the end of the session.
2. Session resumption and early data using TSIP are not supported.

The above restrictions are expected to be improved by TSIP and wolfSSL from the next version onwards.

# RESOURCES

Followings are the links to the sites that contain helpful information regarding board, MCU, TSIP and wolfSSL .

## RENESAS SITES

- Renesas wiki page for RX72N Envision Kit (https://github.com/renesas/rx72n-envision-kit/wiki)
- Renesas RX MCUs( https://www.renesas.com/us/en/products/microcontrollers-microprocessors/rx-32-bit-performance-efficiency-mcus/
- Renesas Trusted Secure IP Driver(TSIP) (Renesas Trusted Secure IP Driver(TSIP)(https://www.renesas.com/us/en/software-tool/trusted-secure-ip-driver/)

## WOLFSSL SITES

- wolfSSL Website (www.wolfssl.com)
- wolfSSL Renesas page (https://www.wolfssl.com/docs/renesas/)
- wolfSSL TSIP support page (https://www.wolfssl.com/docs/wolfssl-renesas-tsip/)
- wolfSSL Renesas GitHub repo (https://github.com/wolfSSL/Renesas/)

# SUPPORT AND CONTACT

For support inquiries and questions, please email support@wolfssl.com. Feel free to reach out to info@wolfssl.jp.