



wolfSSL サンプルプログラム実行マニュアル

ターゲット : Renesas RX72N Envision Kit
RTOS: FreeRTOS+ IoT libraries

内容

本ドキュメントの目的	3
サンプルプロジェクトの概要	3
サンプルプロジェクト動作の要件	5
サンプルプロジェクトの作成手順	5
1.executable プロジェクトの作成	5
新規プロジェクトの作成	5
2.デバイス情報の設定	9
3.FIT モジュールの追加と更新	9
4.wolfSSL パッケージのコピー	14
5.セクション設定	15
6.wolfSSL ライブラリプロジェクトと wolfSSL デモプロジェクトを追加	16
wolfSSL ライブラリプロジェクトのインポート	16
WOLFSSL デモアプリケーションファイルの追加	17
AWS_DEMOS プロジェクトヘインクルードファイルパスの追加	20
AWS_DEMOS プロジェクトヘプリプロセッサ・マクロの追加	20
AWS_DEMOS プロジェクトヘリンクするライブラリファイルの追加	21
7. wolfSSL デモタスクの追加	21
デモアプリケーションの実行	23
Crypto-test デモ	24
Crypto-Benchmark デモ	25
TLS-Client デモ	25

ユーザーが用意した Root CA 証明書を利用する場合に必要なこと	28
クライアント認証を行うための必要事項	28
制限事項	29
リソース	30
RENESAS SITES.....	30
WOLFSSL SITES	30
サポートとコンタクト.....	30

本ドキュメントの目的

このドキュメントは wolfSSL TLS ライブラリおよびサンプルプログラムを Renesas 社製 RX72N Envision Kit 上で動作させるためのインストラクションである。対象 MCU は製品搭載時にはリアルタイム OS とともに利用されることが想定される。そこで本サンプルプログラムは FreeRTOS および FreeRTOS+TCP を使用する構成で提供する。以下ではサンプルプログラムを Renesas 社製 IDE である e² studio のプロジェクトとして新規作成、実行する手順を説明する。

サンプルプロジェクトの概要

本サンプルプログラムの実行には FreeRTOS カーネルと FreeRTOS+TCP プロトコルスタックが必要だが、それらは e² studio のプロジェクトの新規作成時に自動的に用意される。プロジェクト作成時に自動的に実行されるスクリプトによって、GitHub 上の FreeRTOS 関連のソースファイルがダウンロードされ評価ボード上での動作に必要な設定が行われる。ダウンロードされる FreeRTOS IoT Libraries にはいくつかのデモアプリケーションが含まれており、その中から選択したデモアプリケーションが実行されるような構成になっている。

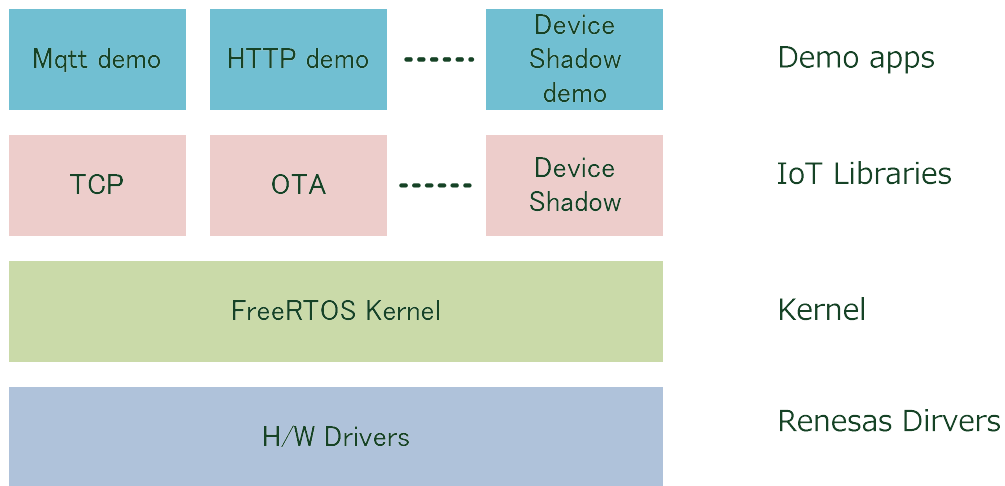


図 1. プロジェクト新規作成時のソフトウェア構成

wolfSSL サンプルプログラムはこの構成に、wolfSSL ライブラリ、wolfSSL デモアプリケーションと H/W ドライバとして必要な FIT コンポーネントを追加し図 2 に示すような構成にする。

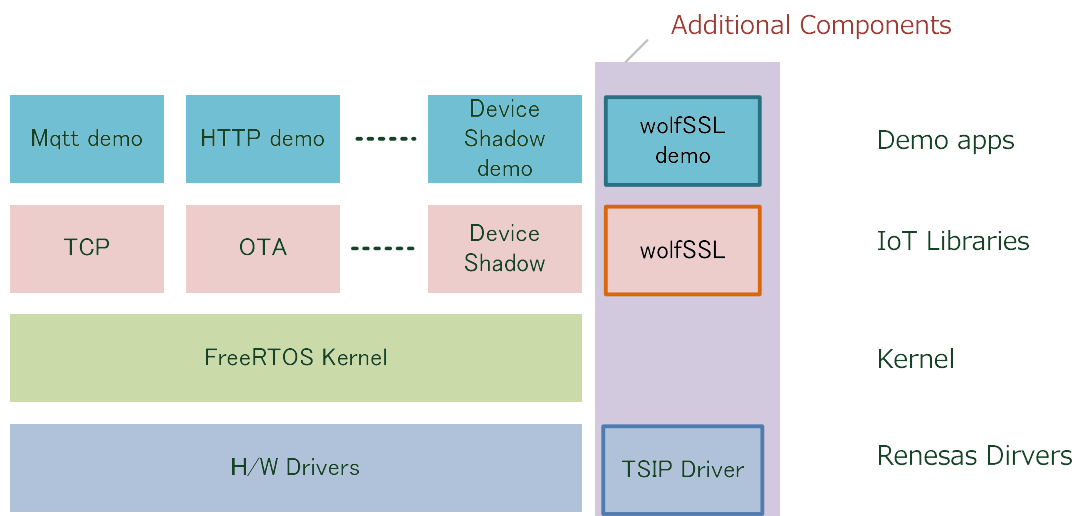


図 2. wolfSSL デモアプリケーション追加時のソフトウェア構成

追加される wolfSSL デモアプリケーションは FreeRTOS カーネル上のタスクとして動作し、通信機能として TCP プロトコルスタックを利用する。加えて、wolfSSL ライブラリは TSIP をサポートしている。wolfSSL ライブラリがソフトウェアとして実装している暗号化機能と TLS 機能の一部を H/W (TSIP) に置き換えることにより処理速度を大幅に向上させることが可能である。

サンプルプロジェクト動作の要件

本サンプルプログラムの動作に必要なツール、コンポーネント：

1. e² Studio Version 2021-10 以降
2. CC-RX Tool Chain V3.04 以降
3. TSIP v1.15 以降
4. RTOS v202012.00-rx-1.0.0 以降
5. wolfSSL v5.3.0 以降

サンプルプロジェクトの作成手順

サンプルプログラムの実行には大まかに以下のステップが必要である：

1. e²Studio のプロジェクト新規作成で Executable プロジェクトを作成
2. デバイス情報の設定
3. FIT モジュールの追加と更新
4. wolfSSL パッケージのコピー
5. セクション設定
6. wolfSSL ライブラリプロジェクトと wolfSSL デモプロジェクトを追加
7. wolfSSL デモタスクの選択と実行

以下順に説明する。

1.EXECUTABLE プロジェクトの作成

新規プロジェクトの作成

e² studio を起動し、ファイルメニュー > インポート > 一般 > **Renesas GitHub FreeRTOS(with IoT libraries) プロジェクト** を選択する。

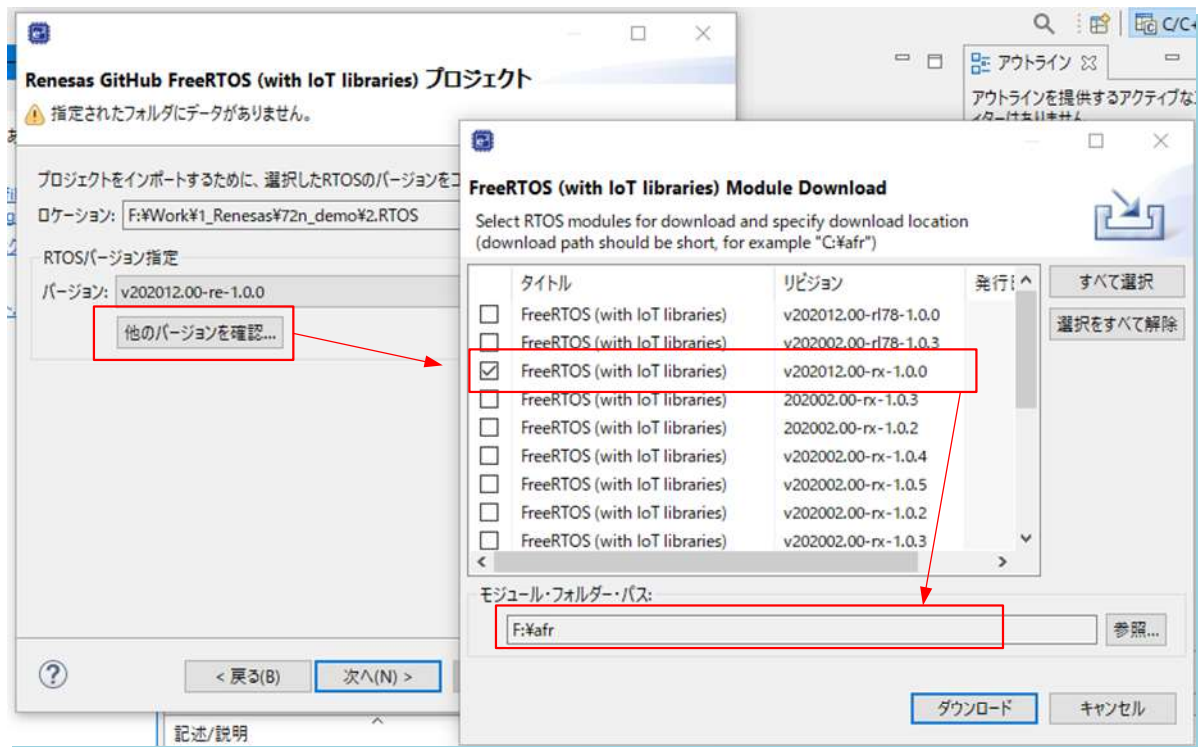


図 3. 新規プロジェクトの作成手順

Renesas GitHub FreeRTOS(with IoT libraries) プロジェクトダイアログが表示される。RTOS バージョン指定のセクションで “他のバージョンを確認..” ボタンを押す。

FreeRTOS(with IoT libraries)をダウンロードしてくるのだが、そのリビジョンとダウンロード先を指定する必要がある。リビジョンは“**v202012.00-rx-1.0.0**”を指定すること。リビジョン名が類似しているものが多いので注意すること。

さらに、ダウンロード先のフォルダ（モジュール・フォルダー・パス）はパス名の長さの制約で処理に失敗する可能性があるため、できるだけパス名の短いフォルダを指定すること。

“ダウンロード”ボタンを押すとモジュールのダウンロードが行われる。

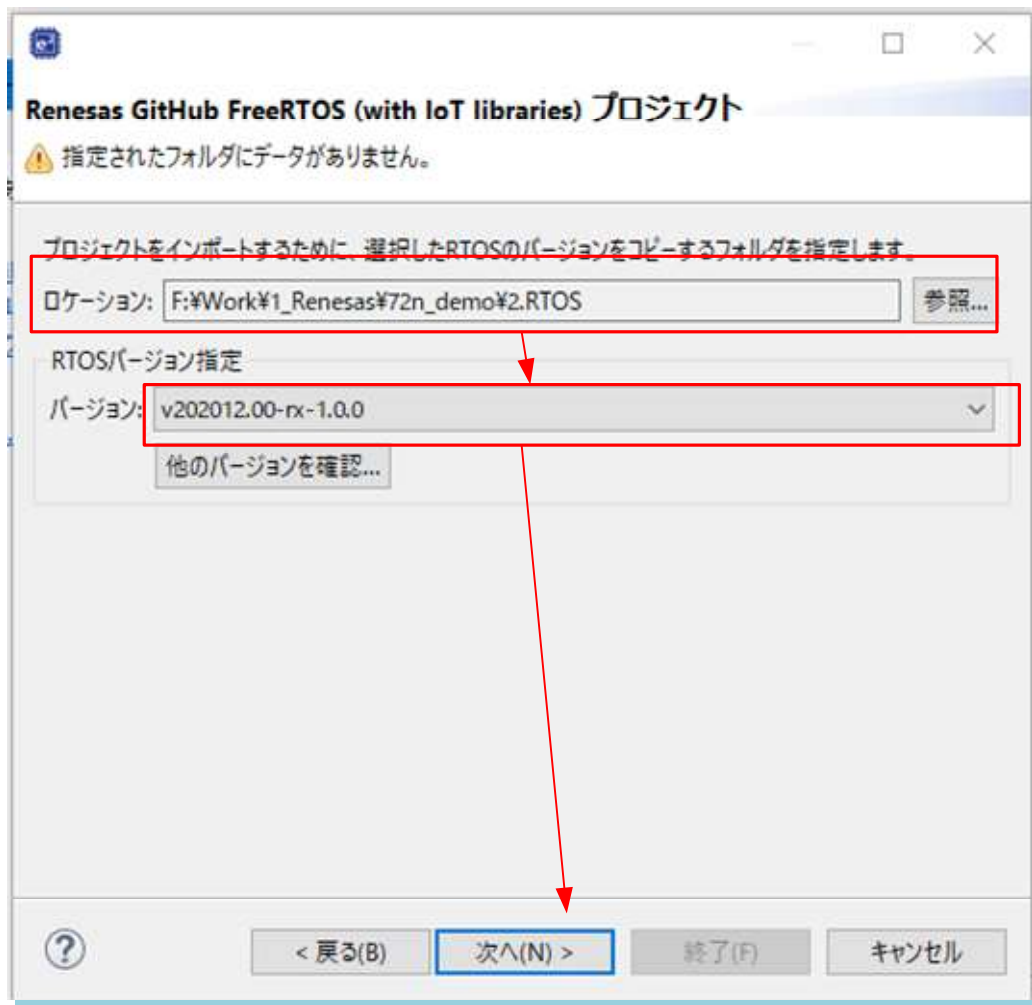


図 4. 新規プロジェクトの作成ロケーションとバージョン

図 4 のダイアログでロケーション欄にプロジェクトを作成するベースフォルダを指定する。以降、本ドキュメントではこのフォルダをbaseと称する。

RTOS バージョン指定欄には先ほど選択したバージョンが選択されていることを確認し、“次へ”ボタンを押す。

内部で処理が実行された後に、図 5 に示すプロジェクトのインポートダイアログが表示される。プロジェクト欄には複数の似通った名前のプロジェクトがリストアップされている。aws_demos, aws_tests, boot_loader の 3 種類のデモプロジェクトがボードとコンパイラ別のフォルダに存在しているので

“aws_demos(…¥projects¥renesas¥rx72n-envision-kit¥e2studio¥aws_demos)”

となっているものを探して選択すること。

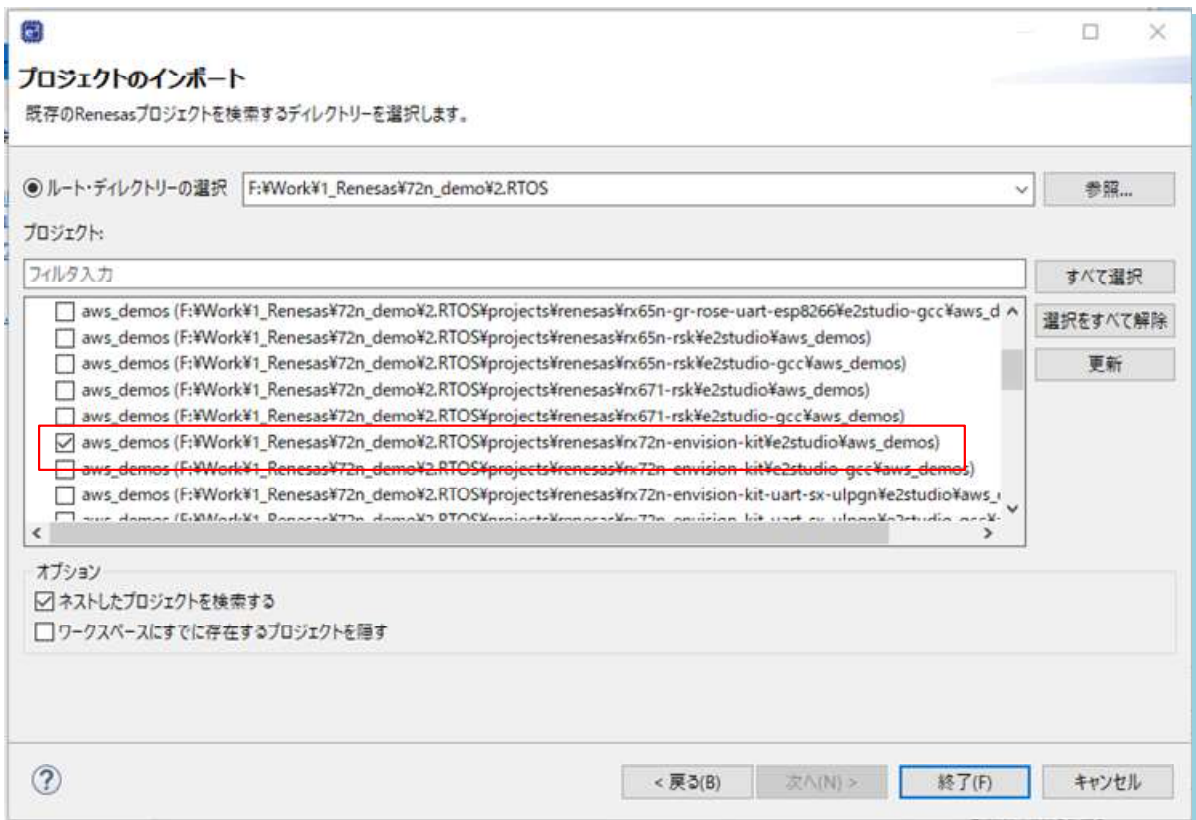


図 5. プロジェクトのインポートダイアログ

プロジェクトのインポートが完了すると、プロジェクト・エクスプローラー上には図 6 の様に aws_demos プロジェクトが出現する。

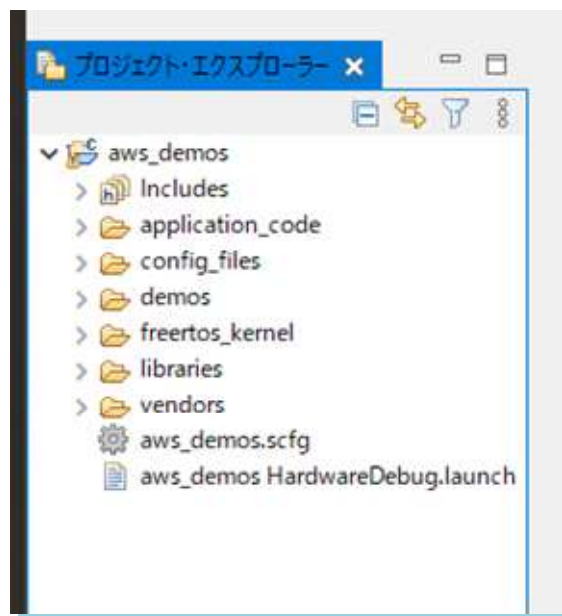


図 6. インポートされた aws_demos プロジェクト

2. デバイス情報の設定

FIT モジュールの追加に先立ち、ボード・デバイスの設定を行っておく。プロジェクト・エクスプローラー上の **aws_demos.scfg** をダブルクリックしてスマートコンフィギュレータパースペクティブを開き、下部のボードタブを選択してソフトウェアコンポーネント設定ペインを表示させる。ボード欄で

“RX72NEnvisionKit(V1.10)”を選択する。もし、選択リストにこのボードが出現しない場合は、下の”ボード情報をダウンロードする..”のリンクを押してボード情報を取得する。

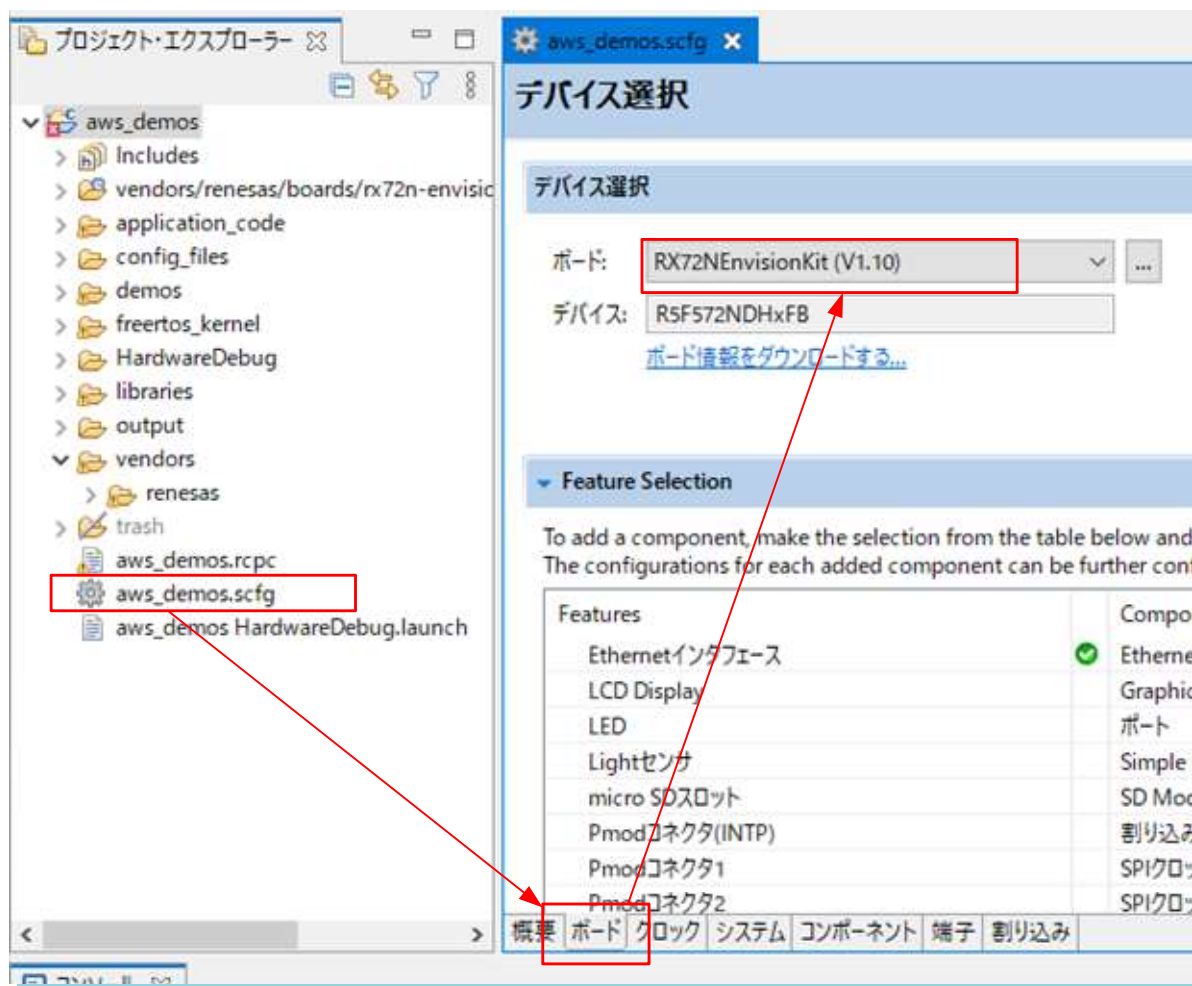


図 7. デバイス選択

3. FIT モジュールの追加と更新

この時点でプロジェクトには FreeRTOS と IoT ライブラリおよびデモアプリケーションのソースファイルが登録されている。また必要な FIT コンポーネント（Renesas 社製ドライバー）群のソースファイルも生成済みとなっている。しかし、いくつかの FIT コンポーネントライブラリは Renesas 社のサイトからダウンロードして取得する必要がある。

プロジェクト・エクスプローラー上の **aws_demos.scfg** をダブルクリックしてスマートコンフィギュレータパースペクティブを開き、下部の“コンポーネント”タブを選択してソフトウェアコンポーネント設定ペインを表示させる。

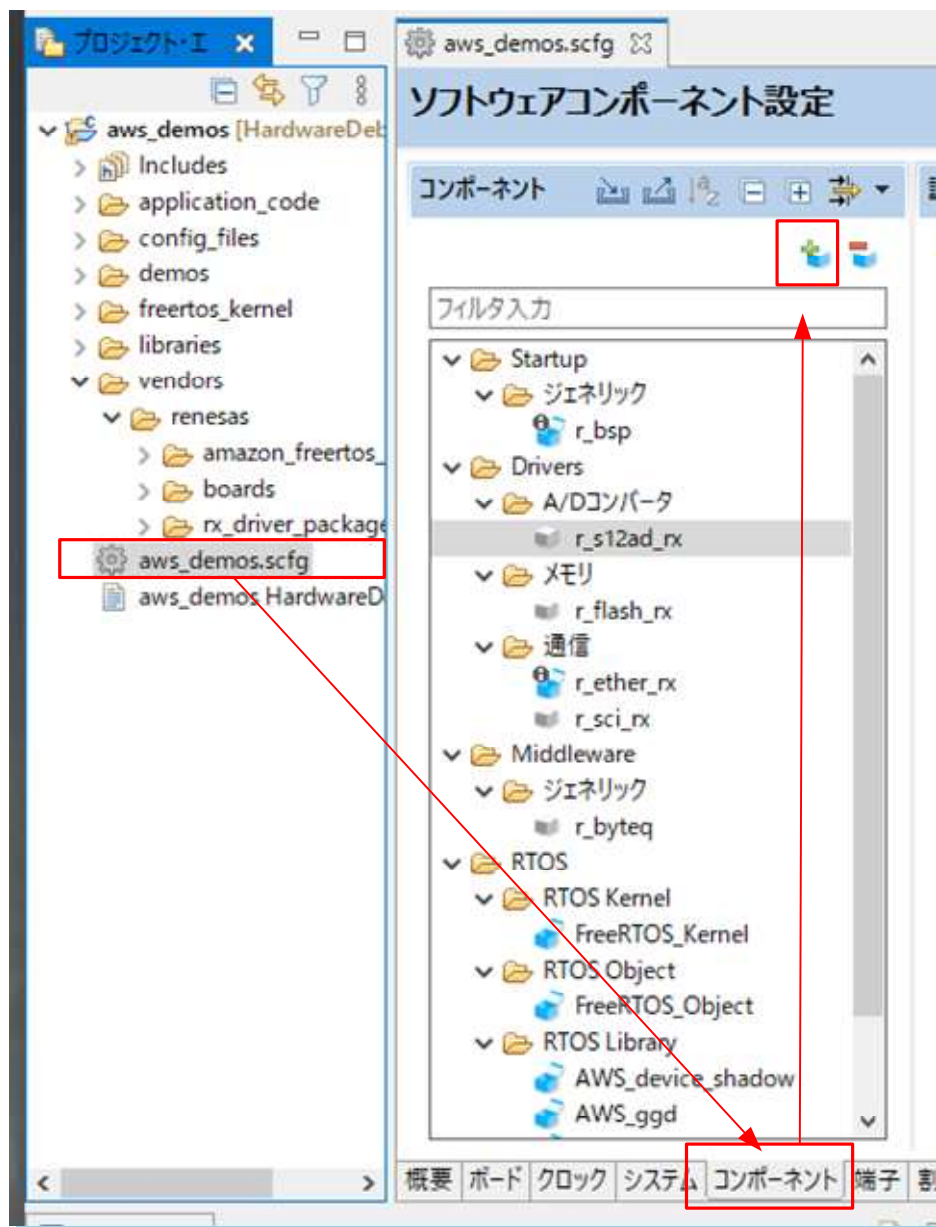


図 8. ソフトウェアコンポーネント設定

その後、上部のコンポーネント追加アイコンを押す。図 9 のダイアログが表示される。ダイアログ中ほどのコンポーネントリストに何も表示されていない場合は“最新版の FIT ドライバとミドルウェアをダウンロードする”のリンクを押して PC 上にダウンロードしておく。ダウンロード後には一覧に多くのコンポーネントがリストアップされるが、この中から所望のコンポーネントを目視で選択するのは難しいので、機能を指定して関係したコンポーネントを絞りこませる。

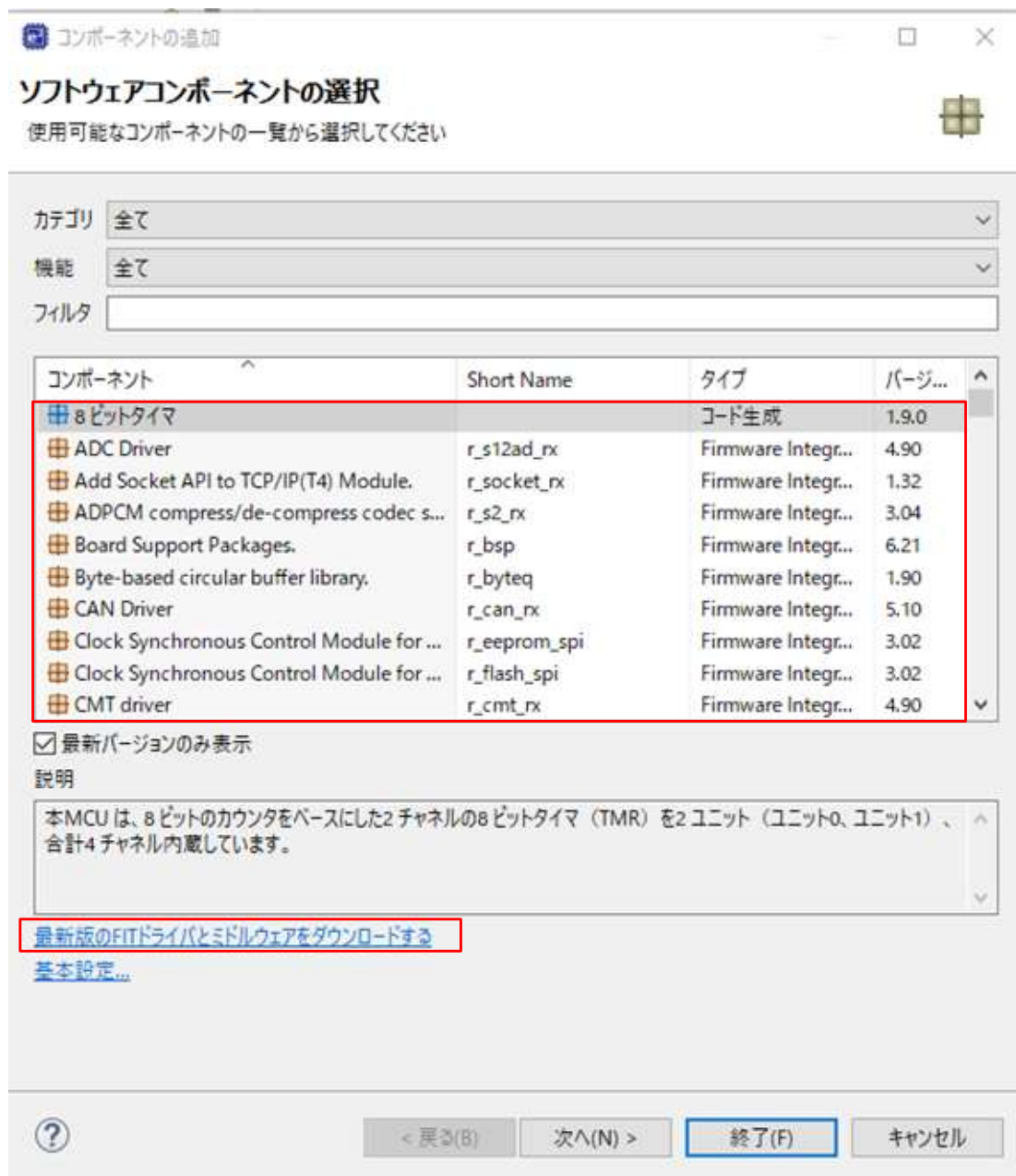


図 9. ソフトウェアコンポーネントの選択ダイアログ

wolfSSL サンプルプログラムでは次の FIT コンポーネントの追加が必要である：

1. TSIP コンポーネント(r_tsip_rx)
2. CMT コンポーネント(r_cmt_rx)

“機能”のリストボックスから“セキュリティ”を選択すると、コンポーネント欄に“TSIP”がリストアップされるので選択し“終了”ボタンを押す。

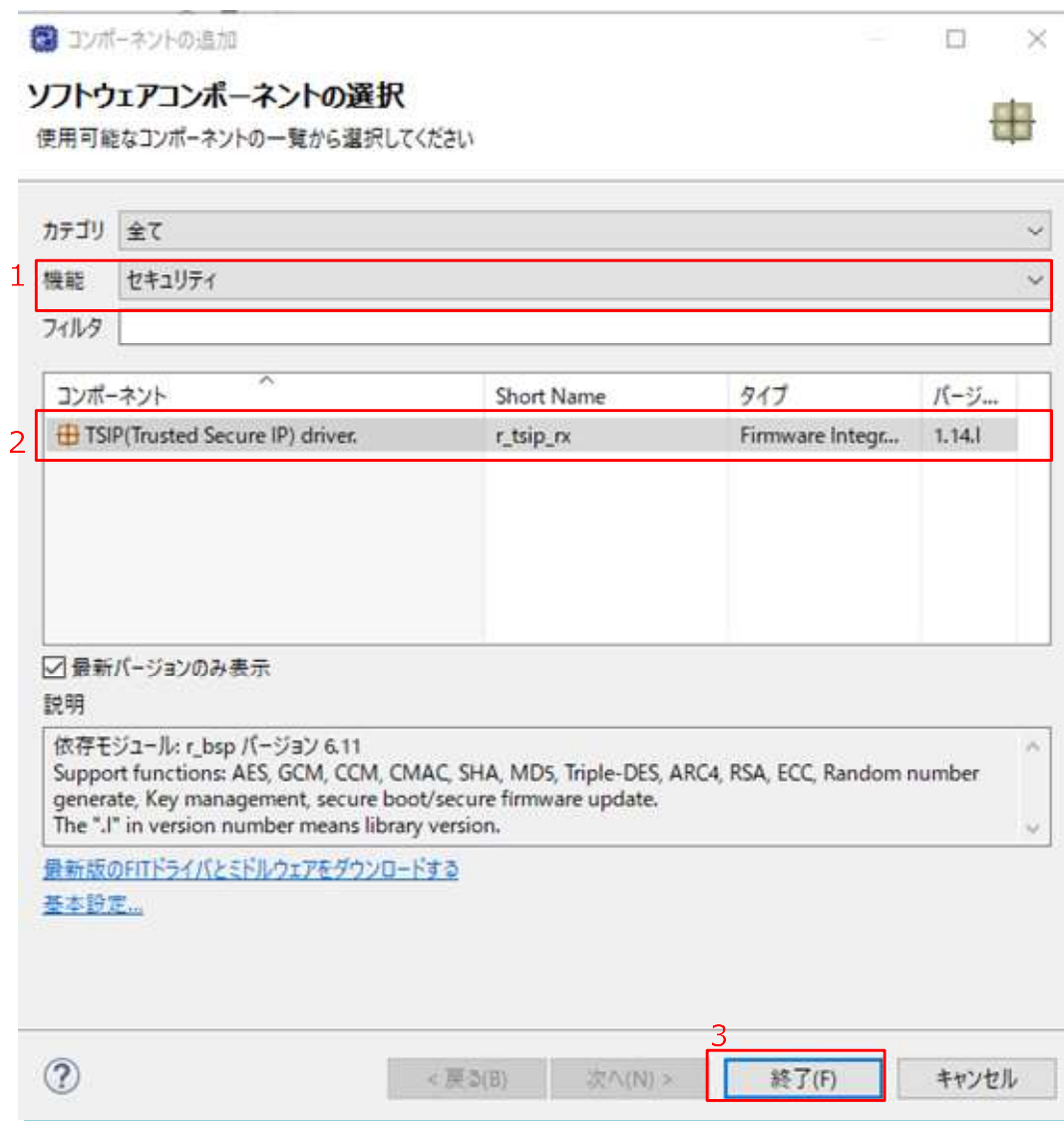


図 10. TSIP の追加

同様にして CMT Driver(r_cmt_rx)を追加する。コンポーネントの追加が終わると使用されるコンポーネントの状況が図 11 のようになる。セキュリティコンポーネント“r_tsip_rx”とタイマコンポーネント“r_cmt_rx”が追加されていることが確認できる。

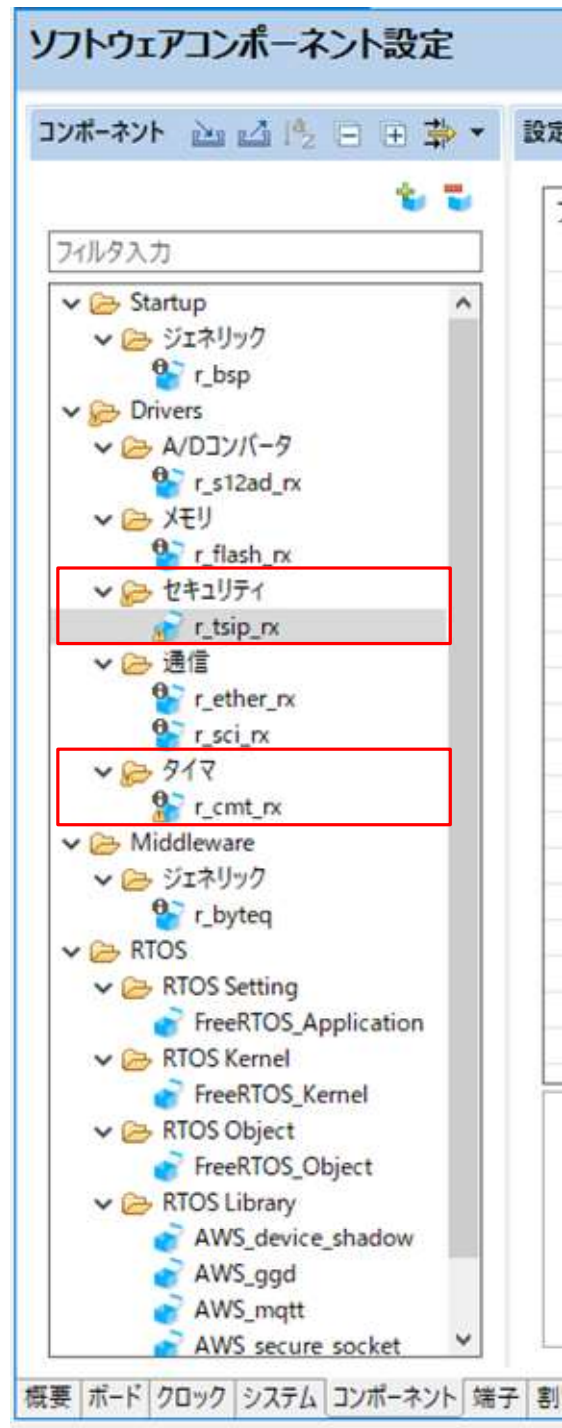


図 11. 2 つのコンポーネントが追加されたリスト

必要な FIT コンポーネントの登録が終了したので、SMC にソースファイルを生成させる。ソフトウェアコンポーネント設定画面の右上の“コードの生成”ボタンを押してコードを生成させる。



4. WOLFSSL パッケージのコピー

GitHub または wolfSSL ダウンロードページからダウンロードした wolfSSL パッケージがある場合、図 12 の右側のボックスのように、最上位のフォルダー名にバージョン文字列が含まれている（wolfssl-5.1.1-stable など）。パッケージ全体を<base>フォルダーの下に「wolfssl」という名前でコピーすること。wolfSSL デモと aws_demos の両方がパス名を相互に参照するため、wolfssl のトップフォルダの名前と場所は、図と同様にすること。

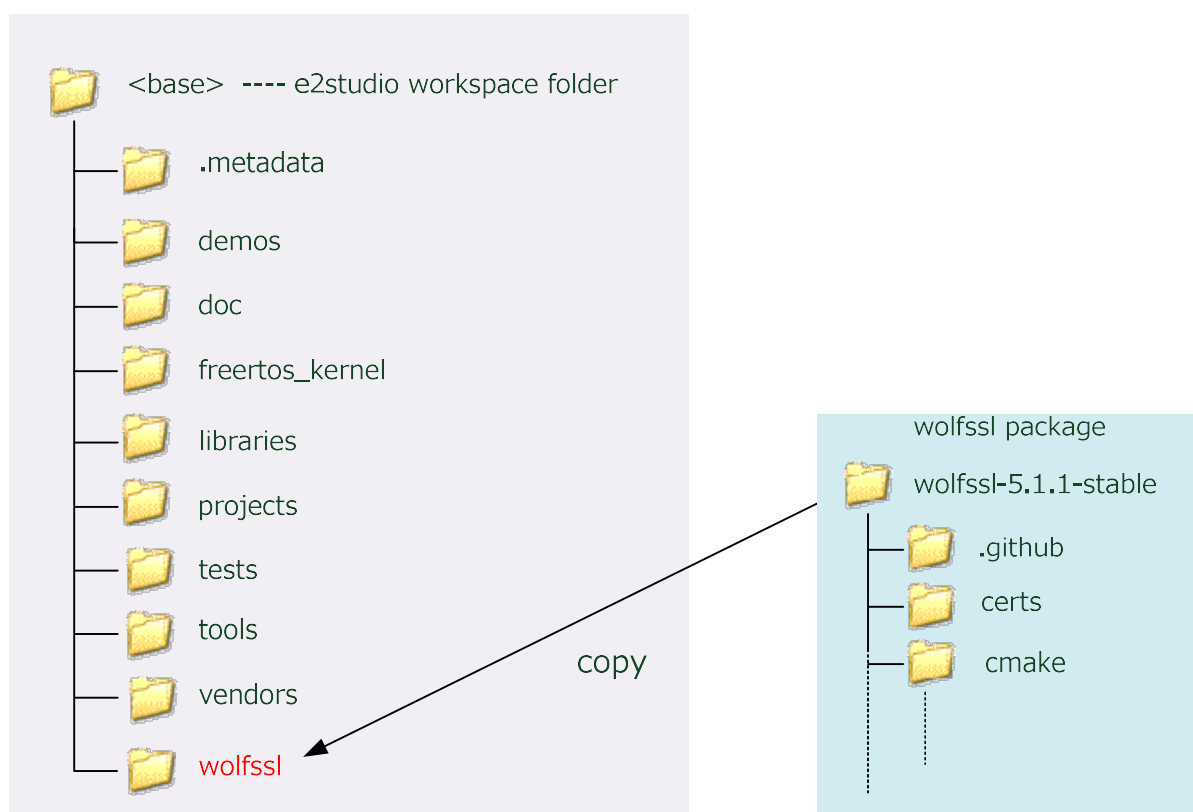


図 12. wolfSSL フォルダ追加後のフォルダ構成

5. セクション設定

追加した FIT コンポーネントが要求しているメモリマップ上のセクションの設定を行う。aws_demos プロジェクトのプロパティを表示させ、C/C++ビルド > 設定 > ツール設定 > Linker > セクション を選択して下図に示す様なセクション設定画面を表示させる。セクション設定欄の右端の“...”ボタンを押す。

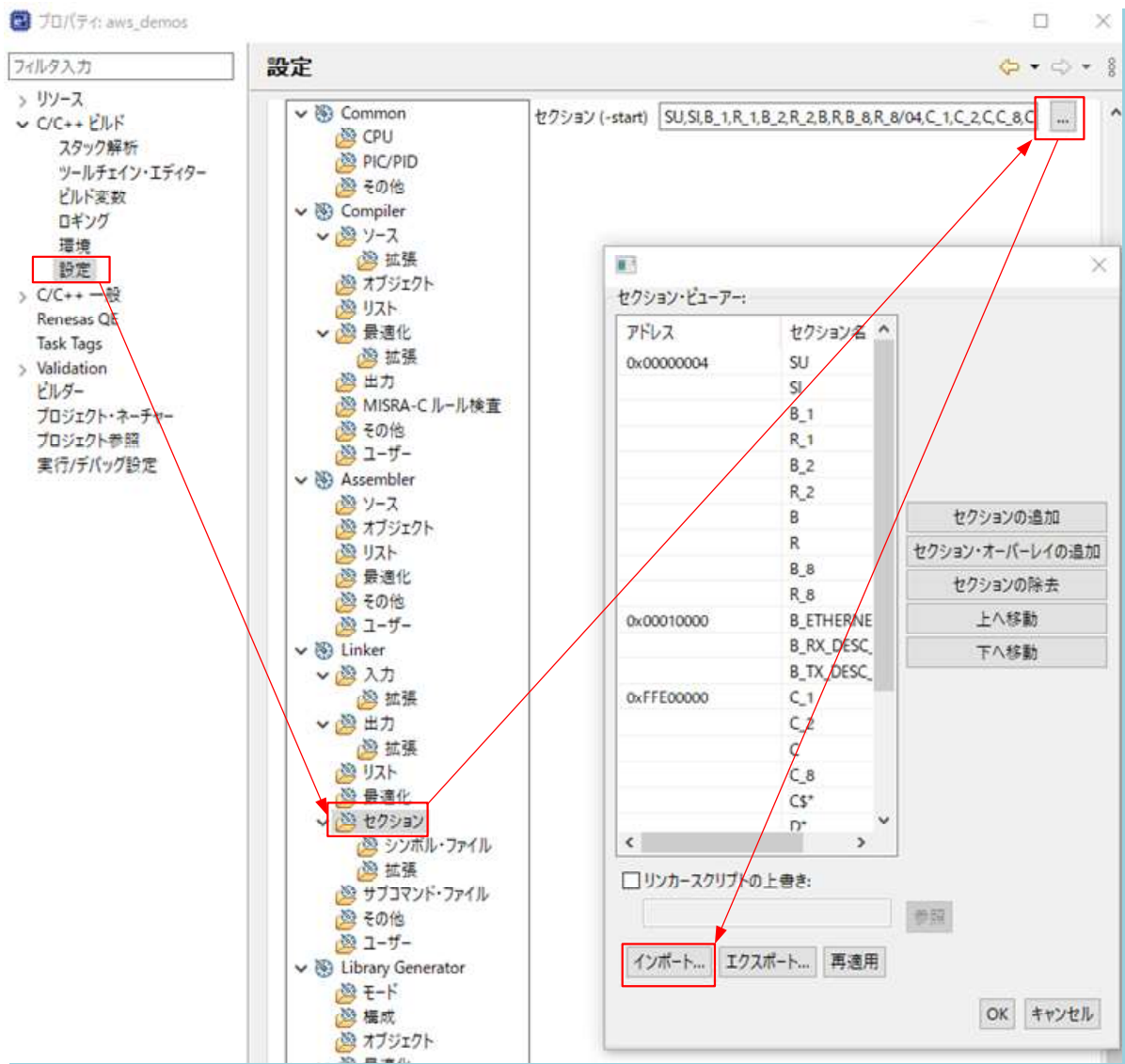


図 13. セクション設定画面

セクション・ビューアダイアログが表示されるので、下部の“インポート”ボタンを押す。リンカー・セクション・ファイルを指定するダイアログが表示されるのでここに、下記の section.esi ファイルを指定するとセクション設定が更新される。

<base>%wolfSSL%IDE%Renesas%e2studio%RX72N%EnvisionKit%resource%section.esi

6. WOLFSSL ライブラリプロジェクトと WOLFSSL デモプロジェクトを追加

次に必要な作業は wolfSSL ライブラリと wolfSSL デモアプリケーションのコードをプロジェクトに追加する作業になる。

WOLFSSL ライブラリプロジェクトのインポート

e² studio 上で aws_demos プロジェクトに wolfSSL ライブラリをビルドするプロジェクトをインポートする。wolfSSL ライブラリプロジェクトは既に wolfSSL パッケージ内に e² studio のプロジェクトとして用意してある。

e² studio のファイルメニュー > ファイル・システムからプロジェクトを開く > ディレクトリ を選択し以下のフォルダを指定する。

<base>%wolfssl%IDE%Renesas%e2studio%RX72N%EnvisionKit%wolfssl

プロジェクト欄に wolfssl プロジェクトがリストアップされるので“終了”ボタンを押す。

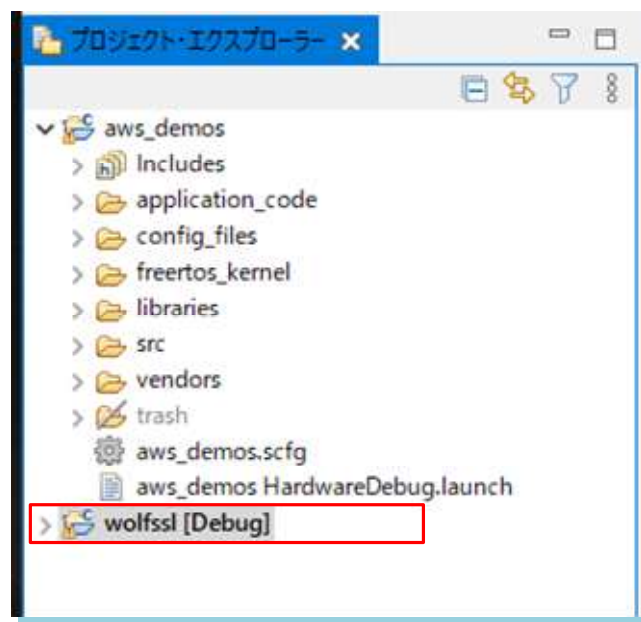


図 14. wolfSS ライブラリプロジェクト追加後のプロジェクト構成

プロジェクト・エクスプローラー上に wolfssl プロジェクトが追加されていることが確認できる。

追加された wolfssl ライブラリプロジェクトには aws_demos プロジェクト内に生成されたインクルードファイルへのパスの設定は指定済みなので設定すべき項目はない。

WOLFSSL デモアプリケーションファイルの追加

FreeRTOS のデモタスクとして wolfSSL デモタスクを追加する。プロジェクト・エクスプローラー上で aws_demos フォルダをマウスの右ボタンでクリックし、新規 > ソース・フォルダ を選択する。

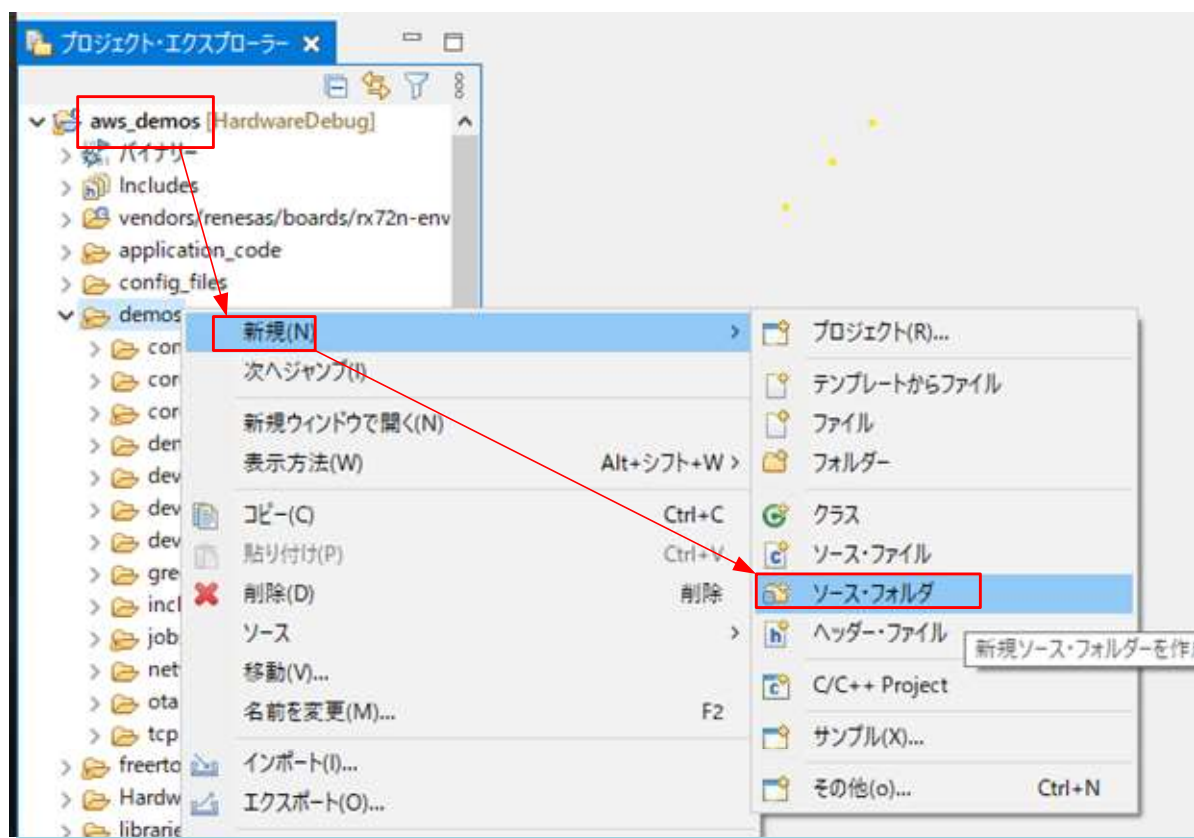


図 15.wolfssl_demo ソース・フォルダの追加

追加するフォルダ名を“**wolfssl_demo**”に設定する。次に、エクスプローラで

<base>%wolfssl%IDE%Renesas%e2studio%RX72N%EnvisionKit%wolfssl_demo

フォルダを表示させ。内部のファイルをすべて選択し、プロジェクト・エクスプローラー上の wolfssl_demo フォルダにドロップする。

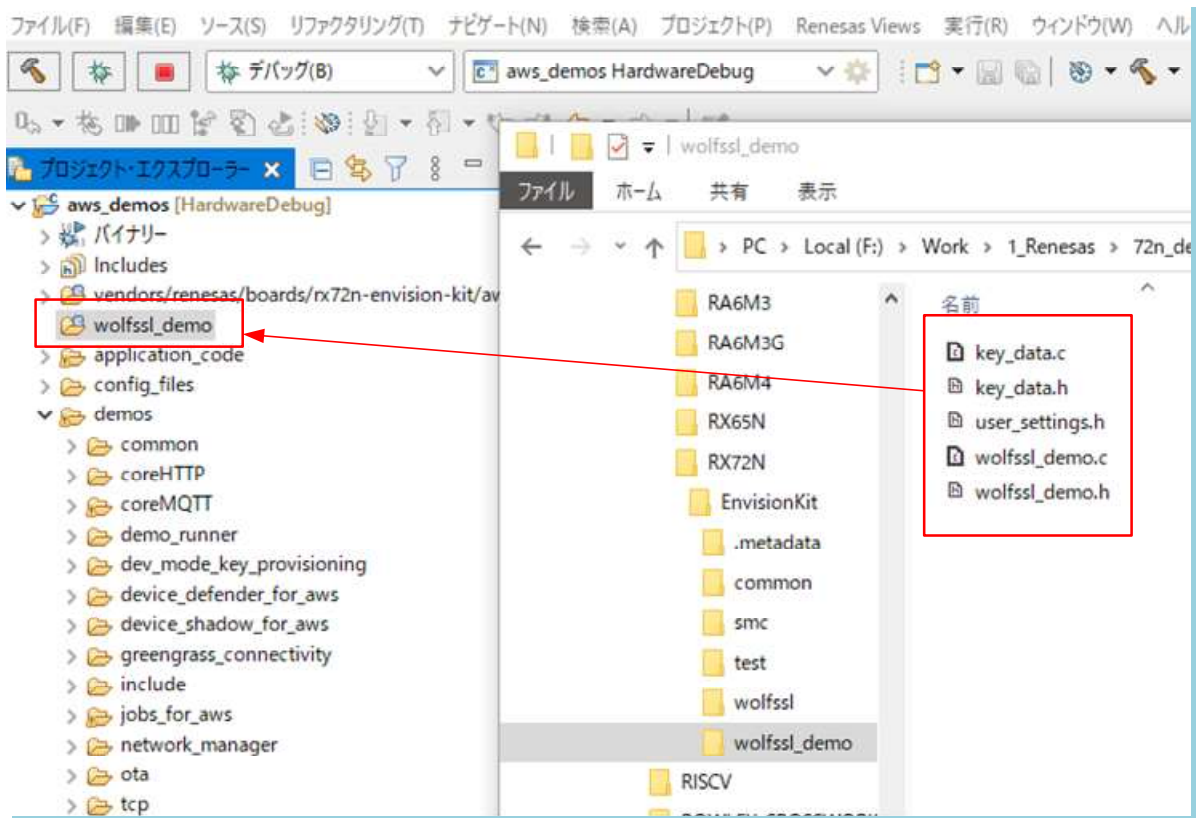


図 16.wolfssl_demo ソースファイルの追加

その際、図 17 に示す様にファイルをどのように追加するか（コピーするかリンクするか）問われるので、**リンクする**を選択すること。

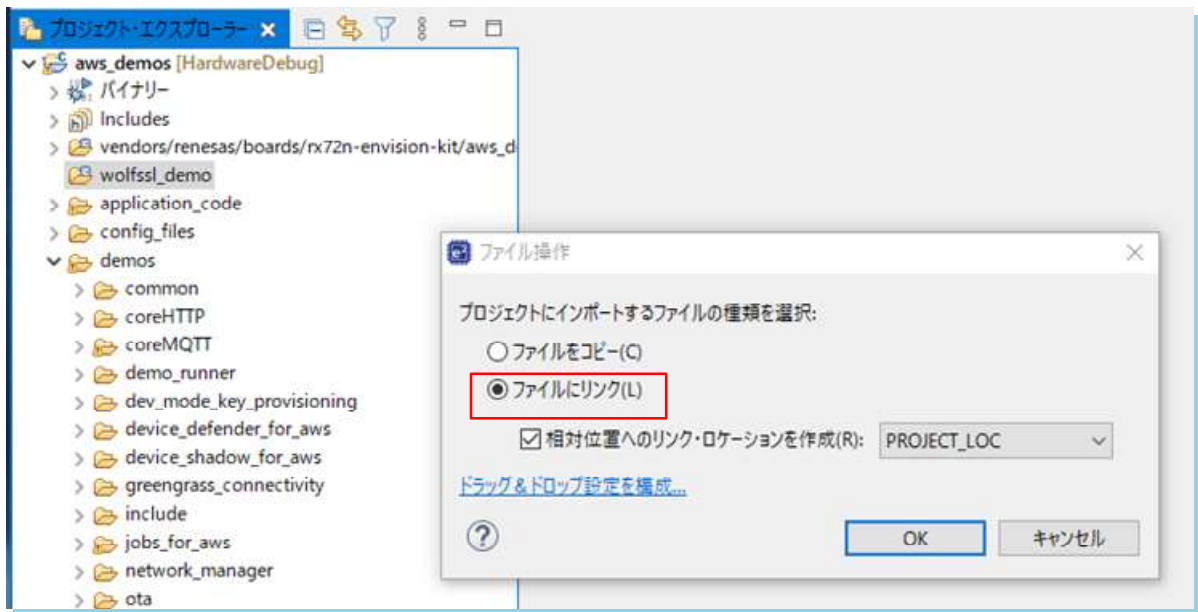


図 17. ファイルのリンクを追加

wolfssl_demo フォルダにはさらに次のファイルを同様にリンクとして追加する：

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

ファイルの追加が完了すると wolfssl_demo フォルダは図 18 に示す様になる。

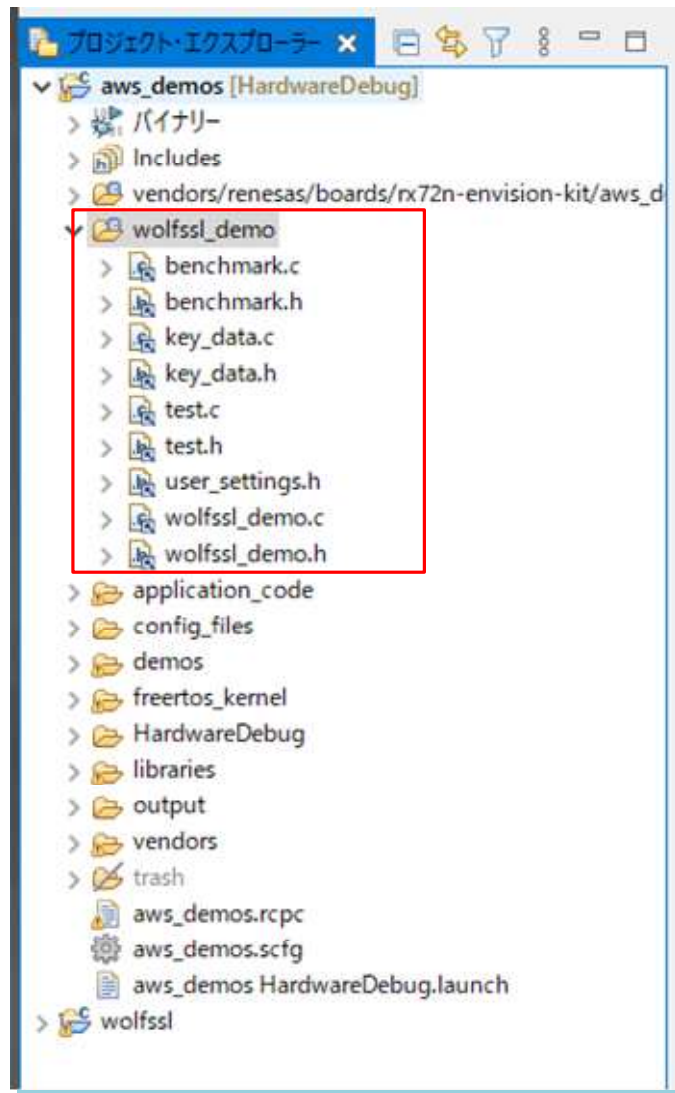


図 18. wolfssl_demo フォルダ

AWS_DEMOS プロジェクトヘインクルードファイルパスの追加

aws_demos プロジェクトのプロパティで、C/C++ ビルド > 設定 > Compiler > ソース > インクルードファイルを検索するフォルダ に以下を追加する。

- ◆ `${ProjDirPath}/../..../..../wolfssl`
- ◆ `${ProjDirPath}/../..../..../wolfssl/IDE/Renesas/e2studio/RX72N/EnvisionKit/wolfssl_demo`

AWS_DEMOS プロジェクトヘプリプロセッサ・マクロの追加

aws_demos プロジェクトのプロパティで、C/C++ ビルド > 設定 > Compiler > ソース > プリプロセッサ・マクロ に以下を追加する。

◆ WOLFSSL_USER_SETTINGS

AWS_DEMOS プロジェクトへリンクするライブラリファイルの追加

aws_demos プロジェクトのプロパティで、C/C++ ビルド > 設定 > Linker > 入力 > リンクするライブラリ・ファイル に以下を追加する。

◆ \${ProjDirPath}/../..../wolfssl/IDE/Renesas/e2studio/RX72N/EnvisionKit/wolfssl/Debug/wolfssl.lib

7. WOLFSSL デモタスクの追加

wolfSSL_demo はデモアプリケーションの一つとしてプロジェクトに追加された。さらにこのデモアプリケーションを実行するための設定を行う。設定は

```
<base>%venders%renesas%boards%rx72n-envision-  
kit%aws_demos%config_files%aws_demo_config.h
```

にをオープンし、現在

```
#define CONFIG_CORE_MQTT_DEMO_ENABLED
```

となっている箇所をコメントアウトし

```
#define CONFIG_WOLFSSL_DEMO_ENABLED
```

に変更する。

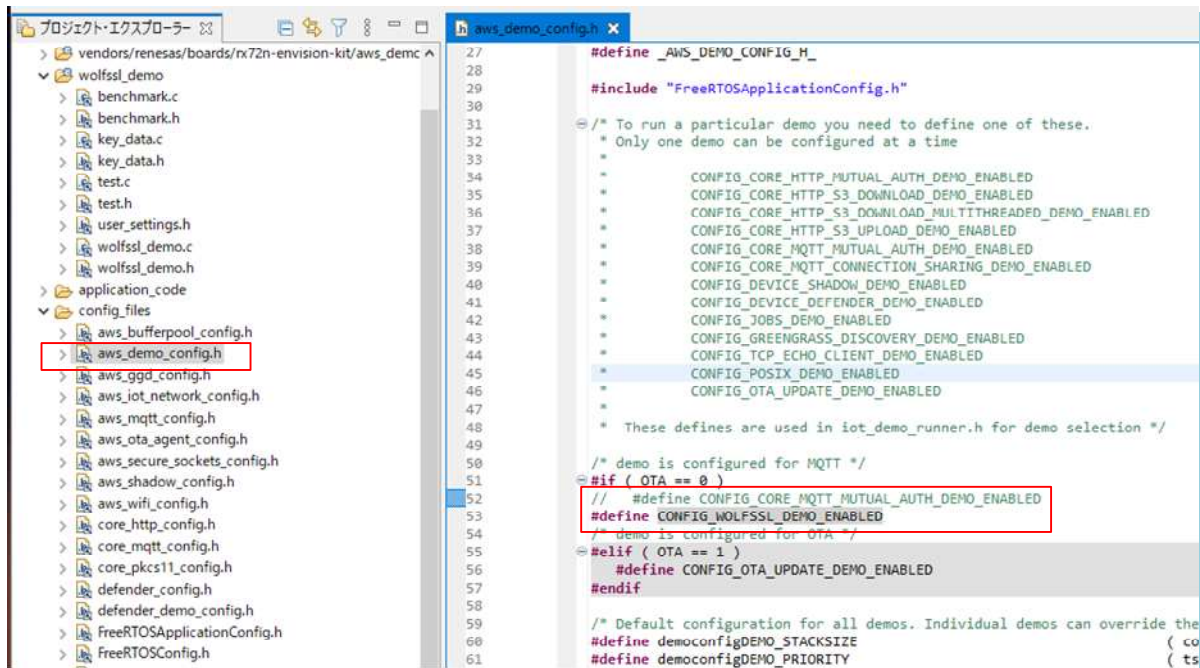


図 19. マクロの追加

さらに、<base>%demos%include%iot_demo_runner.h

をオープンし、最終行付近の#else の行の直前に

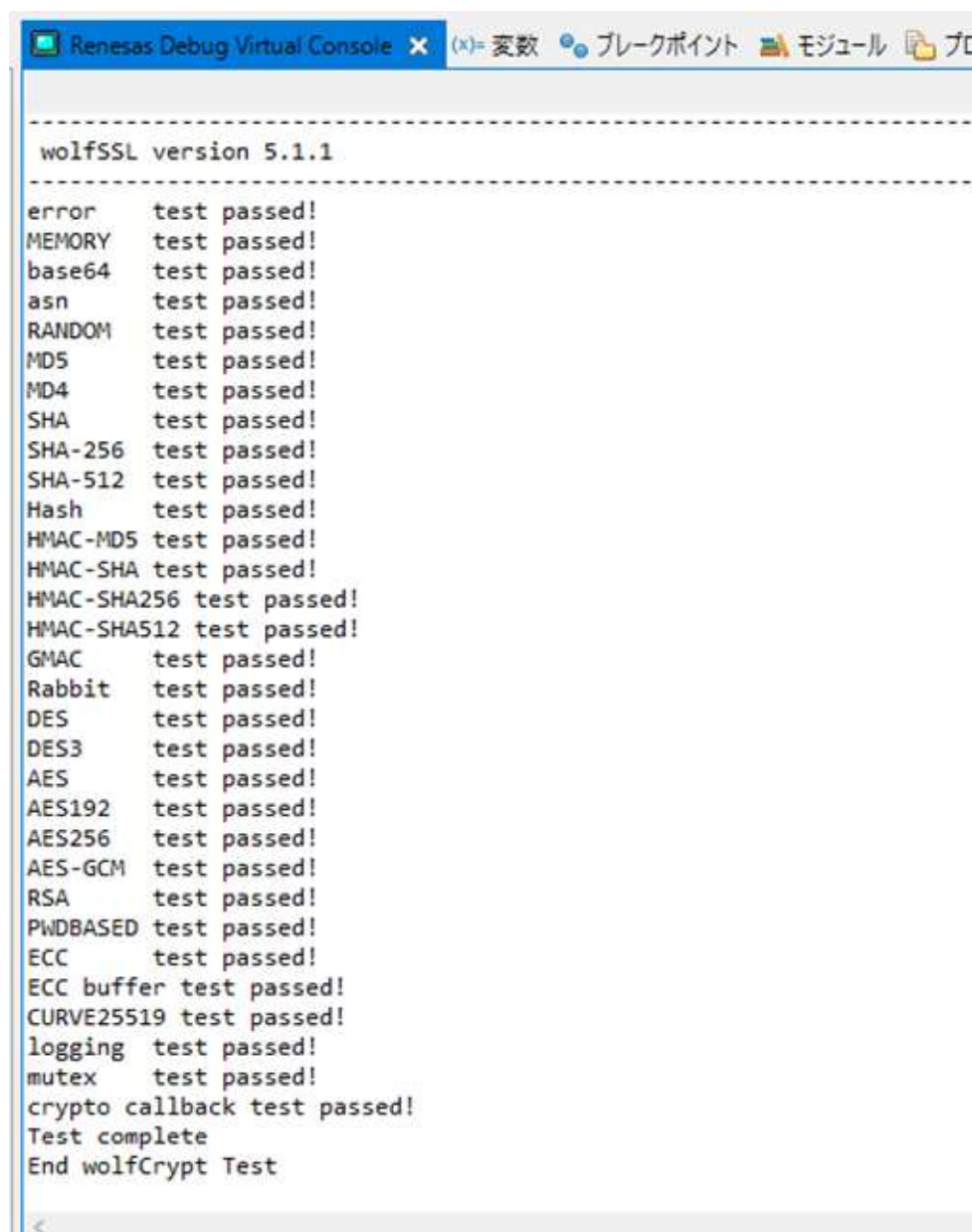
```
#elif defined(CONFIG_WOLFSSL_DEMO_ENABLED)
    #define DEMO_entryFUNCTION          wolfSSL_demo_task
```

の 2 行を追加する。

デモアプリケーションの実行はデバッグ機能をつかって行う。デモ実行中の表示は Renesas Debug Virtual Console で確認できる。

CRYPTO-TEST デモ

Renesas Debug Virtual Console をオープンしておいてデバッグを実行すると、CryptTest でもが実行され結果が表示される。

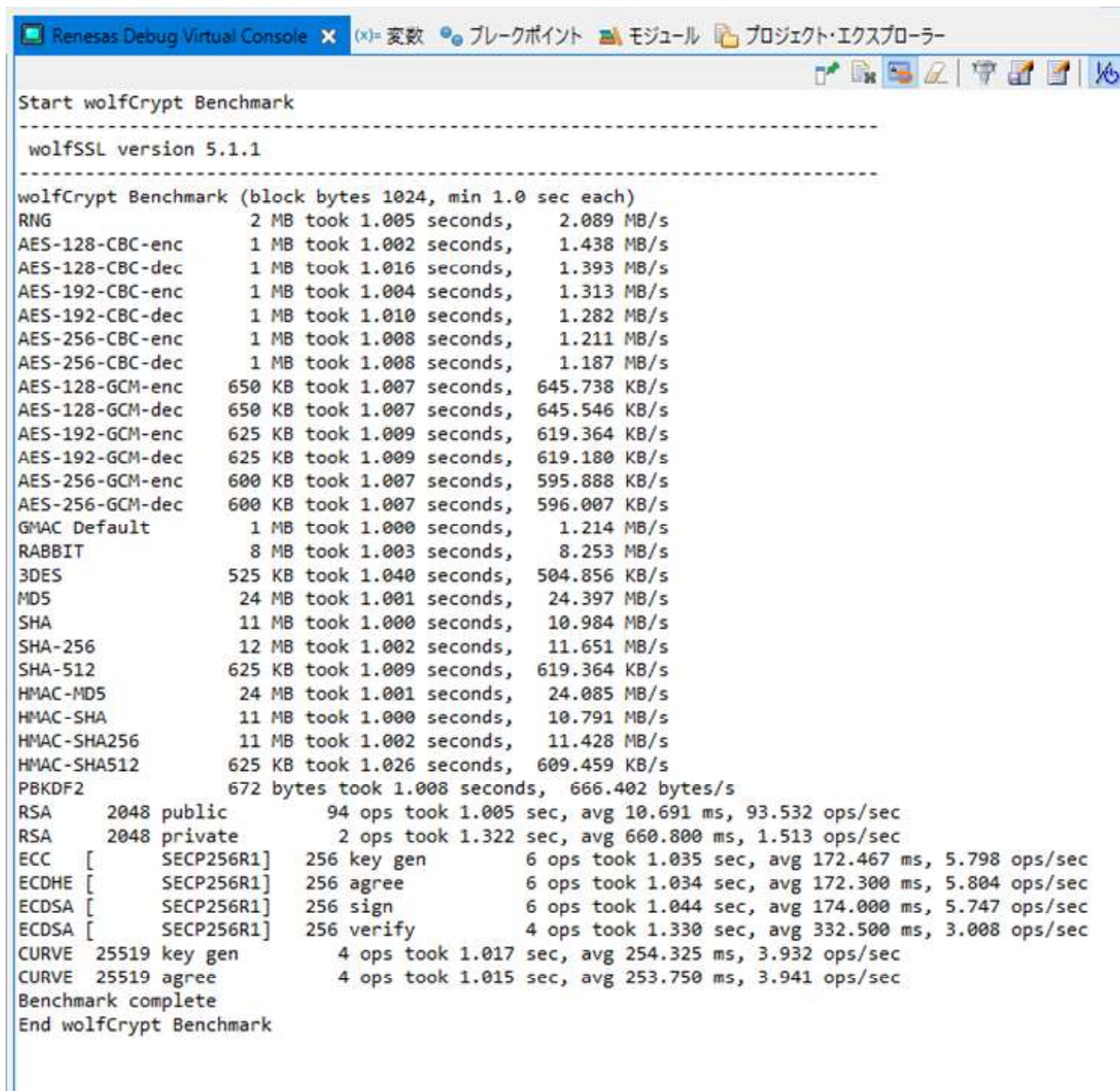


```
wolfSSL version 5.1.1

error      test passed!
MEMORY    test passed!
base64     test passed!
asn        test passed!
RANDOM      test passed!
MD5        test passed!
MD4        test passed!
SHA        test passed!
SHA-256    test passed!
SHA-512    test passed!
Hash       test passed!
HMAC-MD5   test passed!
HMAC-SHA   test passed!
HMAC-SHA256 test passed!
HMAC-SHA512 test passed!
GMAC       test passed!
Rabbit     test passed!
DES        test passed!
DES3       test passed!
AES        test passed!
AES192     test passed!
AES256     test passed!
AES-GCM    test passed!
RSA        test passed!
PWDBASED   test passed!
ECC        test passed!
ECC buffer test passed!
CURVE25519 test passed!
logging    test passed!
mutex      test passed!
crypto callback test passed!
Test complete
End wolfCrypt Test
```

図 21. Crypt-test デモの出力

CRYPTO-BENCHMARK デモ



```
Start wolfCrypt Benchmark
-----
wolfSSL version 5.1.1
-----
wolfCrypt Benchmark (block bytes 1024, min 1.0 sec each)
RNG                2 MB took 1.005 seconds, 2.089 MB/s
AES-128-CBC-enc    1 MB took 1.002 seconds, 1.438 MB/s
AES-128-CBC-dec    1 MB took 1.016 seconds, 1.393 MB/s
AES-192-CBC-enc    1 MB took 1.004 seconds, 1.313 MB/s
AES-192-CBC-dec    1 MB took 1.010 seconds, 1.282 MB/s
AES-256-CBC-enc    1 MB took 1.008 seconds, 1.211 MB/s
AES-256-CBC-dec    1 MB took 1.008 seconds, 1.187 MB/s
AES-128-GCM-enc    650 KB took 1.007 seconds, 645.738 KB/s
AES-128-GCM-dec    650 KB took 1.007 seconds, 645.546 KB/s
AES-192-GCM-enc    625 KB took 1.009 seconds, 619.364 KB/s
AES-192-GCM-dec    625 KB took 1.009 seconds, 619.180 KB/s
AES-256-GCM-enc    600 KB took 1.007 seconds, 595.888 KB/s
AES-256-GCM-dec    600 KB took 1.007 seconds, 596.007 KB/s
GMAC Default       1 MB took 1.000 seconds, 1.214 MB/s
RABBIT             8 MB took 1.003 seconds, 8.253 MB/s
3DES               525 KB took 1.040 seconds, 504.856 KB/s
MD5                24 MB took 1.001 seconds, 24.397 MB/s
SHA                11 MB took 1.000 seconds, 10.984 MB/s
SHA-256            12 MB took 1.002 seconds, 11.651 MB/s
SHA-512            625 KB took 1.009 seconds, 619.364 KB/s
HMAC-MD5           24 MB took 1.001 seconds, 24.085 MB/s
HMAC-SHA           11 MB took 1.000 seconds, 10.791 MB/s
HMAC-SHA256        11 MB took 1.002 seconds, 11.428 MB/s
HMAC-SHA512        625 KB took 1.026 seconds, 609.459 KB/s
PBKDF2             672 bytes took 1.008 seconds, 666.402 bytes/s
RSA 2048 public    94 ops took 1.005 sec, avg 10.691 ms, 93.532 ops/sec
RSA 2048 private    2 ops took 1.322 sec, avg 660.800 ms, 1.513 ops/sec
ECC [ SECP256R1] 256 key gen 6 ops took 1.035 sec, avg 172.467 ms, 5.798 ops/sec
ECDHE [ SECP256R1] 256 agree 6 ops took 1.034 sec, avg 172.300 ms, 5.804 ops/sec
ECDSA [ SECP256R1] 256 sign 6 ops took 1.044 sec, avg 174.000 ms, 5.747 ops/sec
ECDSA [ SECP256R1] 256 verify 4 ops took 1.330 sec, avg 332.500 ms, 3.008 ops/sec
CURVE 25519 key gen 4 ops took 1.017 sec, avg 254.325 ms, 3.932 ops/sec
CURVE 25519 agree 4 ops took 1.015 sec, avg 253.750 ms, 3.941 ops/sec
Benchmark complete
End wolfCrypt Benchmark
```

図 22. Crypt-benchmark デモの出力

TLS-CLIENT デモ

TLS_CLIENT としてのデモを行う場合には、TLS 通信の相手方となる対向アプリケーションが必要である。wolfSSL パッケージにはこの用途に使用できるサンプルプログラムが用意されている。このプログラムは wolfSSL をビルドすることで生成される。wolfSSL のビルドには gcc がインストールされている Linux (MacOS、WSL も含む) でのビルドと VisualStudio をつかったのビルドが可能である。以下では WSL 上でのビルドを紹介する。

(1) ECDSA 証明書を使用する場合

TLS_CLIENT 側は user_settings.h 内に USE_ECC_CERT が定義されているため、既定で、ECDSA 証明書を使用する設定となっている。これに合わせてサーバー側のサンプルプログラムを以下のコンフィギュレーションオプションを指定してビルドする。“-DNO_RSA”を付与することを忘れないようにすること。

```
$ 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) RSA 証明書を使用する場合

また、TLS_CLIENT 側で RSA 証明書を使用する設定を行う場合には、user_settings.h 内で USE_ECC_CERT 定義をコメントアウトして再ビルドを行う。対応して、サーバー側のサンプルでは以下のコンフィギュレーションオプションを指定してビルドする。

```
$ 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
```

make コマンドの実行でエラーが発生せず終了したら、TLS サーバーアプリケーションも生成されており実行可能な状態になっている。TLS サーバーアプリケーションの実行の前に、“ifconfig”コマンドを実行して実行マシンの IP v4 アドレスを取得しておく。この取得したアドレスを wolfSSL_demo.c 内の TLSSERVER_IP マクロに設定する。

一方、ボード側の IP アドレスは以下のマクロへの設定値を変更することで行える。

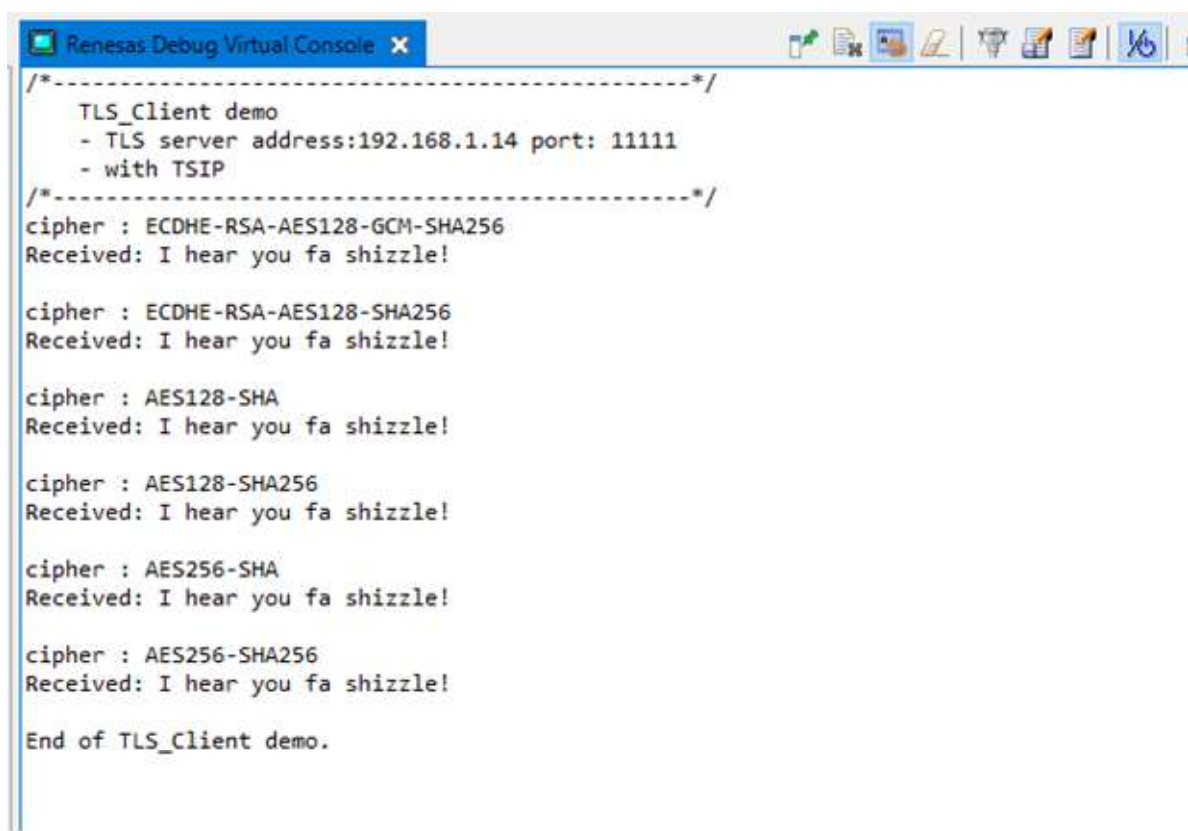
- `ipconfigUSE_DHCP` (FreeRTOSIPConfig.h 内)
- `configIP_ADDR0 ~ configIP_ARRE3` (FreeRTOSConfig.h 内)

デバッグ目的で実行する場合あるいはサーバーとの TCP 接続で問題がある場合には、上記設定を変更して、静的 IP アドレスをボードに設定することを勧める。

TLS サーバーアプリケーションの実行は次のコマンドとオプションで行う。オプションの “-v4” は TLS1.3 プロトコルを使用することを指定する。“-v3” を指定すると TLS1.2 プロトコルを使用する。

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

上記のサーバーアプリケーションを実行させた状態で TLS__CLIENT のデモを実行すると TLS 通信が行われ下図の様な出力が得られる。TLS クライアントは cipher-suite を変えながら繰り返しサーバアプリケーションに TLS 接続を行う。TLS クライアントの出力に含まれている暗号化スイートは TSIP がサポートしているものが TLS バージョン、使用する証明書の種類の設定に応じて変わる。



```

/*-----*/
  TLS_Client demo
  - TLS server address:192.168.1.14 port: 11111
  - with TSIP
/*-----*/
cipher : ECDHE-RSA-AES128-GCM-SHA256
Received: I hear you fa shizzle!

cipher : ECDHE-RSA-AES128-SHA256
Received: I hear you fa shizzle!

cipher : AES128-SHA
Received: I hear you fa shizzle!

cipher : AES128-SHA256
Received: I hear you fa shizzle!

cipher : AES256-SHA
Received: I hear you fa shizzle!

cipher : AES256-SHA256
Received: I hear you fa shizzle!

End of TLS_Client demo.

```

図 23. TLS-Client デモ実行時のボード側の出力

ユーザーが用意した ROOT CA 証明書を利用する場合に必要なこと

本サンプルプログラムでは、TLS_Client として動作する際に必要な RootCA 証明書とサンプルサーバーアプリケーションが使用するサーバー証明書、クライアント証明書などは評価用でのみ利用可能な証明書である。機能評価を超えた目的で利用する場合にはそれら証明書をユーザー自身で用意する必要がある。それに伴い、

1. Provisioning Key
2. RootCA 証明書の検証の為に必要な RSA 鍵ペア
3. RootCA 証明書を上記の RSA 秘密鍵で生成した署名
4. クライアント証明書に含まれている公開鍵に対応した秘密鍵

が必要になる。それらの生成方法は Renesas 社提供のマニュアルを参照のこと。

クライアント認証を行うための必要事項

クライアント認証機能は以下のようにサポートしています。

- TLS1.3 では ECDSA 証明書は TSIP を使って処理し、RSA 証明書はソフトウェアで処理します。
- TLS1.2 では ECDSA 証明書と RSA 証明書は共に TSIP を使って処理します。

(1) クライアント証明書のロード

wolfSSL_CTX_use_certificate_buffer あるいは
wolfSSL_CTX_use_certificate_chain_buffer_format を使ってクライアント証明書をロードしてください。

(2) クライアント秘密鍵/公開鍵のロード

クライアント証明書の種類に応じてロードすべき鍵が決まります。以下に従って必要な鍵をロードしてください。

ECDSA 証明書の場合：

- ・ `tsip_use_PrivateKey_buffer` を使って秘密鍵をロードしてください。

RSA 証明書の場合：

- ・ `tsip_use_PrivateKey_buffer` を使って秘密鍵をロードしてください。
- ・ `tsip_use_PublicKey_buffer` を使って公開鍵をロードしてください。

RSA 証明書の場合には署名処理を内部で検証する目的で公開鍵も使用します。その為に、公開鍵のロードが必要です。

(3) encrypted key の作成

鍵ロードすべき秘密鍵あるいは公開鍵は Rensas Secure Flash Programmer あるいは SecurityKeyManagementTool を使って出力された encrypted key を渡してください。encrypted key の作成方法はアプリケーションノート“RX ファミリ TSIP モジュール Firmware Integration Technology”の 7.1.4“encrypted key, encrypted provisioning key の使用方法”に説明されています。

(4) 必要なマクロ定義

`user_settings.h` に `WOLF_PRIVATE_KEY_ID` の定義を行ってください。

制限事項

TSIPv1.15 をサポートした wolfSSL では以下の機能制限があります。

1. TLS ハンドシェイク中にサーバーと交換したメッセージパケットが平文でメモリ上に蓄積されています。これはハンドシェイクメッセージのハッシュ計算に使用されます。内容はセッション終了時に削除されます。
2. TLS1.3 では TSIP を使ったクライアント認証機能は ECDSA クライアント証明書の場合にのみサポートされます。RSA 証明書の場合はソフトウェアでの処理となります。
3. TLS1.3 では TSIP を使ったサーバー認証機能のうち、CertificateVerify メッセージの検証はソフトウェアでの処理となります。
4. TSIP を使ったセッション再開および early data はサポートされません。

上記制限事項 1~4 は次版以降の TSIP によって改善が見込まれています。

リソース

下記は開発に有用な情報（ボード、MCU、TSIP と wolfSSL）を含んだサイトへのリンクである。

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/ja/products/microcontrollers-microprocessors/rx-32-bit-performance-efficiency-mcus/>)
- Renesas Trusted Secure IP Driver(TSIP) (<https://www.renesas.com/us/ja/software-tool/trusted-secure-ip-driver/>)

WOLFSSL SITES

- wolfSSL Website (<http://www.wolfssl.jp/>)
- wolfSSL Renesas page (<https://www.wolfssl.com/docs/renesas/>)
- wolfSSL TSIP support page (<https://www.wolfssl.com/docs/wolfssl-renesas-tsip/>)

サポートとコンタクト

ご質問は、info@wolfssl.jp までお問い合わせください。テクニカルサポートについては、support@wolfssl.com にお問い合わせください。