Zynq-7000 All Programmable SoC のセキュア ブート

スタートアップ ガイド

UG1025 (v1.0.1) 2014 年 3 月 18 日

XILINX
ALL PROGRAMMABLE™

本資料は英語版 (v1.0.1) を翻訳したもので、内容に相違が生じる場合には原文を優先します。
資料によっては英語版の更新に対応していないものがあります。
日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com までお知らせください。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	内容
2013 年 11 月 12 日	1.0	初版
2014 年 3 月 18 日	1.0.1	図 2-1 および図 2-2 への参照を修正。表 2-1 の SW16 の設定 (位置 4) を修正。PDF のブックマークに章番号を追加、およびフィードバック ボタンを追加。

目次

第 1 章

はじめに

概要

Zynq®-7000 All Programmable SoC (AP SoC) は、非公開キー暗号 (AES/HMAC) と公開キー暗号 (RSA) を提供します。これによって、機密性の高いソフトウェアを暗号化したり、Zynq-7000 AP SoC へロードされるソフトウェアの認証をトラスト チェーン (信頼の連鎖) で行うことができます。トラスト チェーンを使用するブート プロセスでは、敵対者によるソフトウェアの改ざんが行われていないことが保証されます。機密性の高いソフトウェアは、暗号化することで解読不可となり信頼性が向上します。

AES は非公開キーを使用します。RSA は Zynq-7000 AP SoC 内の公開キーを使用し、このキーは必要に応じて何度でも変更でき、同じイメージ内のパーティションに対しても同様です。敵対者がキーを攻撃する時間を制限するために、暗号キーは変更する必要があります。

Zynq-7000 AP SoC のセキュリティ機能はプログラマブル リソースを使用しません。したがって、Zynq-7000 AP SoC 内のセキュリティ機能を使用する際のコストは、セキュア ブートの使用法を学ぶための時間くらいです。このスタートアップ ガイドでは、セキュア ブートの使用法を ZC702 評価ボードを用いて説明します。非セキュア ブートとセキュア ブートでは、同じザイリンクス ツールを使用するため、ツール フローの変更はありません。14.6 リリースの場合、セキュア ブート機能には Bootgen をコマンド ライン ツールとして使用する必要があります。

このユーザー ガイドに付属する参照システムは、次のリンクから入手可能です。

https://secure.xilinx.com/webreg/clickthrough.do?cid=345331

このユーザー ガイドの内容は次のとおりです。

- 第 2 章「ZC702 評価ボードのセットアップ」
- 第 3 章「AES/RSA キーの生成およびプログラミング」
- 第 4 章「Bootgen イメージ フォーマット ファイルの作成」
- 第 5 章「高度なキー管理オプション」
- 第 6 章「マルチブート」
- 第 7 章「データ パーティションのローディング」
- 第 8 章「認証証明のユーザー定義フィールドを使用」

TRD に関するその他の情報は、『Zynq-7000 All Programmable SoC ZC702 評価キットおよびビデオ/画像処理キット スタートアップ ガイド』(UG926) [参照 1] を参照してください。

セキュア ブートに関するその他の情報は、『Zynq-7000 All Programmable SoC のセキュア ブート (XAPP1175)』[参照 2] を参照してください。

第 2 章

ZC702 評価ボードのセットアップ

概要

この章では、ZC702 評価ボードのセットアップ方法について説明します。

注記 : ZC702 ボードの全機能の説明は、『Zynq-7000 XC7Z020 All Programmable SoC 向け ZC702 評価ボード ユーザー ガイド』(UG850) [参照 3] を参照してください。

ZC702 のセットアップ要件

ZC702 評価ボードのセットアップに必要なものを次に示します。

- ハードウェアのセットアップ :
  - ZC702 評価ボード (XC7Z020 CLG484-1 搭載)
  - USB Type-A/Mini-B ケーブル (UART 通信用)
  - AC 電源アダプター (12 VDC)
- Windows ソフトウェアおよびドライバーのセットアップ :
  - Tera Term Pro (または同等の) 端末プログラム (このプログラムはすでにインストールされている場合があります。詳細は、Tera Term のウェブサイトを参照してください。)
  - Silicon Laboratories 社製 USB-UART ドライバー (Silicon Laboratories 社の USB-UART ドライバー ダウンロード サイトからダウンロード可能です。このプログラムはすでにインストールされている場合があります。)

ZC702 のセットアップ要件

ZC702 ボードのジャンパーとスイッチのデフォルト設定を図 2-1 および表 2-1 に示します。

Quad SPI デバイスからブートする場合、図 2-1 のように SW16 スイッチを設定します。ポジション 1、2、3、および 5 を右にし、ポジション 4 を左にします。

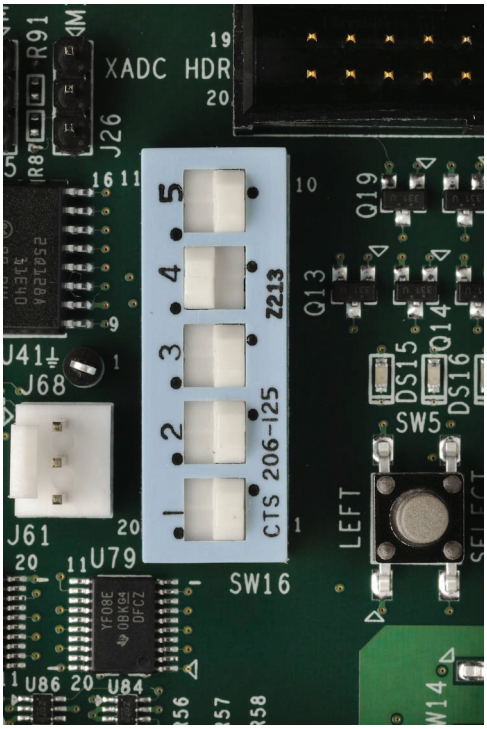


図 2-1 : Quad SPI モードでブートする場合のモード スイッチ設定

表 2-1 : ZC702 評価ボードのスイッチ設定

スイッチ	ポジション	設定
SW10 (JTAG チェーン入力選択用の 2 個の DIP スイッチ)	1	オフ
	2	オン
SW12 (2 個の DIP スイッチ)	1	オフ
	2	オフ
SW15 (2 個の DIP スイッチ)	1	オフ
	2	オフ
SW16 (5 個の DIP スイッチ)	1	右
	2	右
	3	右
	4	左
	5	右
SW11 (電源スライド スイッチ)	オフ	下シフト

ハードウェアのセットアップ

このセクションでは、ハードウェア セットアップの手順について説明します。

1. SW16 モード スイッチを図 2-1 のように設定します。
2. ZC702 ボードの電源をオフにした状態 (図 2-2 の SW11 は下) で、USB Mini-B ケーブルを ZC702 ボード上の USB-UART と表示されている mini-USB ポート J17 へ接続し、もう一方を PC 側の空いている USB ポートへ接続します (図 2-2)。

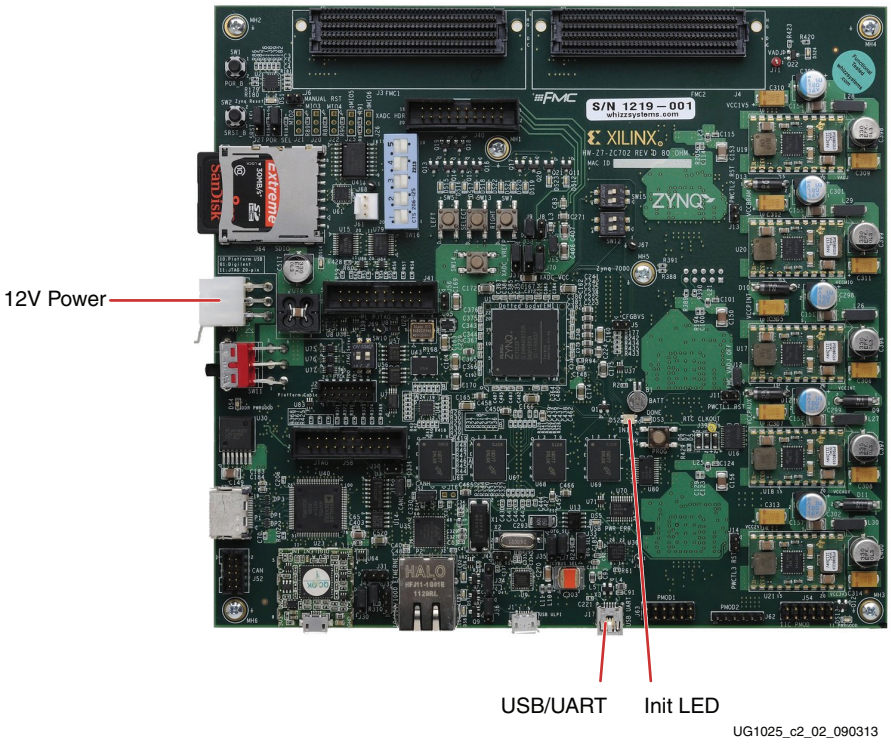


図 2-2 : UART と電源ケーブルを接続する ZC702

3. 電源ケーブルを接続します。
4. ZC702 ボードの電源をオンにします (SW11 スイッチを上にシフト)。

USB-UART ドライバーのインストール

1. ダウンロードした実行可能な UART-USB ドライバー ファイル (6 ページの「ZC702 のセットアップ要件」に記載) を実行します。この実行ファイルを実行することによって、ホスト PC との USB - UART 間通信が可能になります。このドライバーは、ボードに電源が投入されると自動的にダウンロードと実行を開始します。また、このドライバーは、Silicon Laboratories 社の USB-UART ドライバー ダウンロード サイトからダウンロードできます。

図 2-3 : UART ケーブル ドライバーのインストール

2. [デバイス マネージャー] で USB-UART 間の接続を既知の COM ポートおよびボー レートに設定します。
   a. Windows 7 の場合、[スタート] メニューを左クリックして、[コントロール パネル] を選択します。
   b. [デバイス マネージャー] を選択します。
   c. リストの中から Silicon Labs 社製のデバイスを右クリックして、[プロパティ] を選択します。
   d. [ポートの設定] タブをクリックします。[詳細設定] をクリックします。
   e. COM1 ～ COM4 の中で有効な COM ポートを選択します。これでコンピューターがポートの割り当てを記憶するため、ボードのシリアル UART ポートをプラグインするたびに再度割り当てる必要はありません。
   f. ビット/秒 = 115200、データ ビット = 8、パリティ = なし、ストップ ビット = 1、およびフロー制御 = なし に設定します。[OK] をクリックします。

注記 : 手順および図は、Windows XP または Windows 7 ホスト PC を使用した場合を示しています。

図 2-4 ～図 2-6 は、USB-UART ポートの設定手順を示しています。

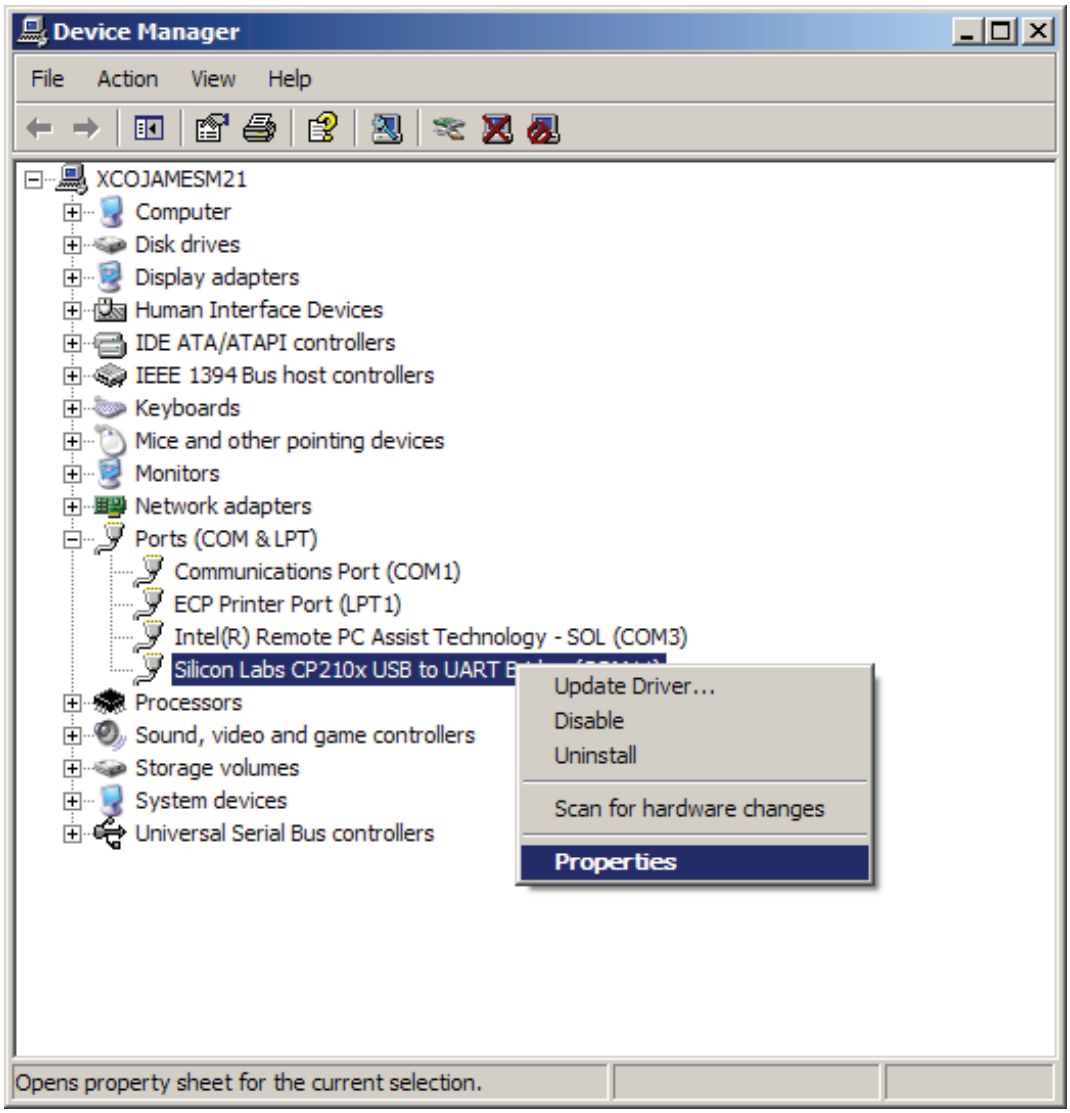


図 2-4 : ドライバーの設定

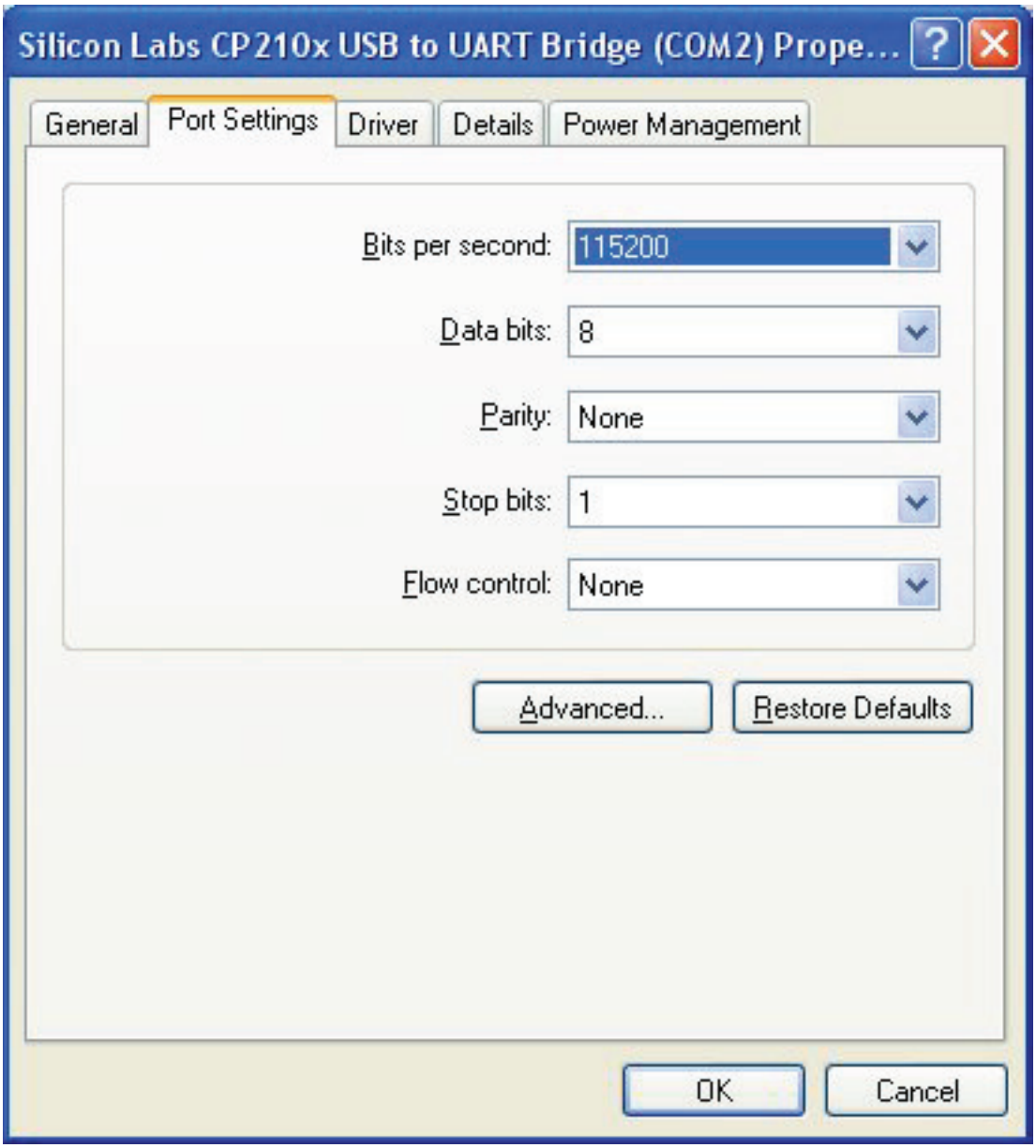


図 2-5 : UART の [ポートの設定] タブ

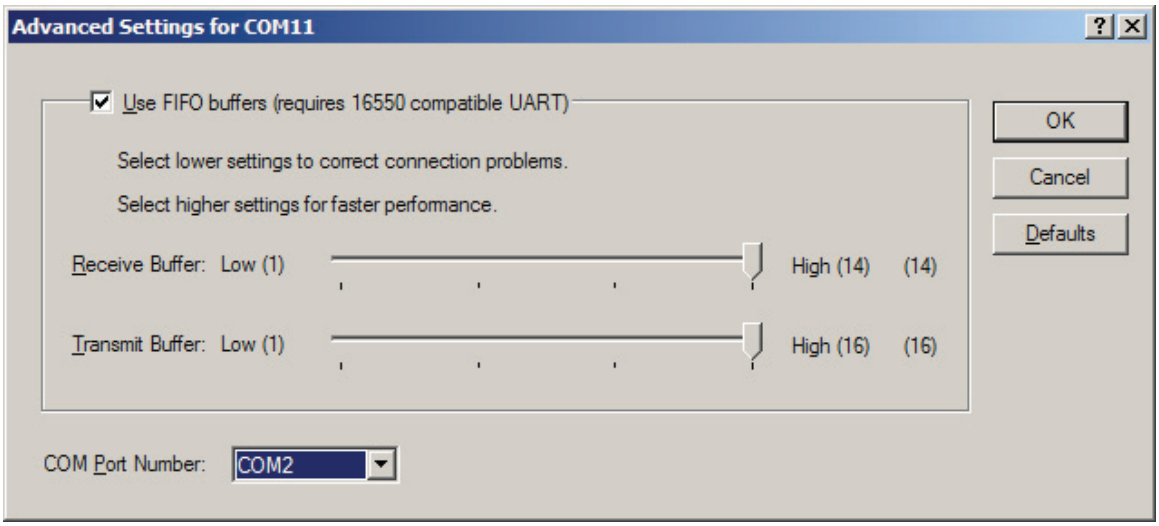
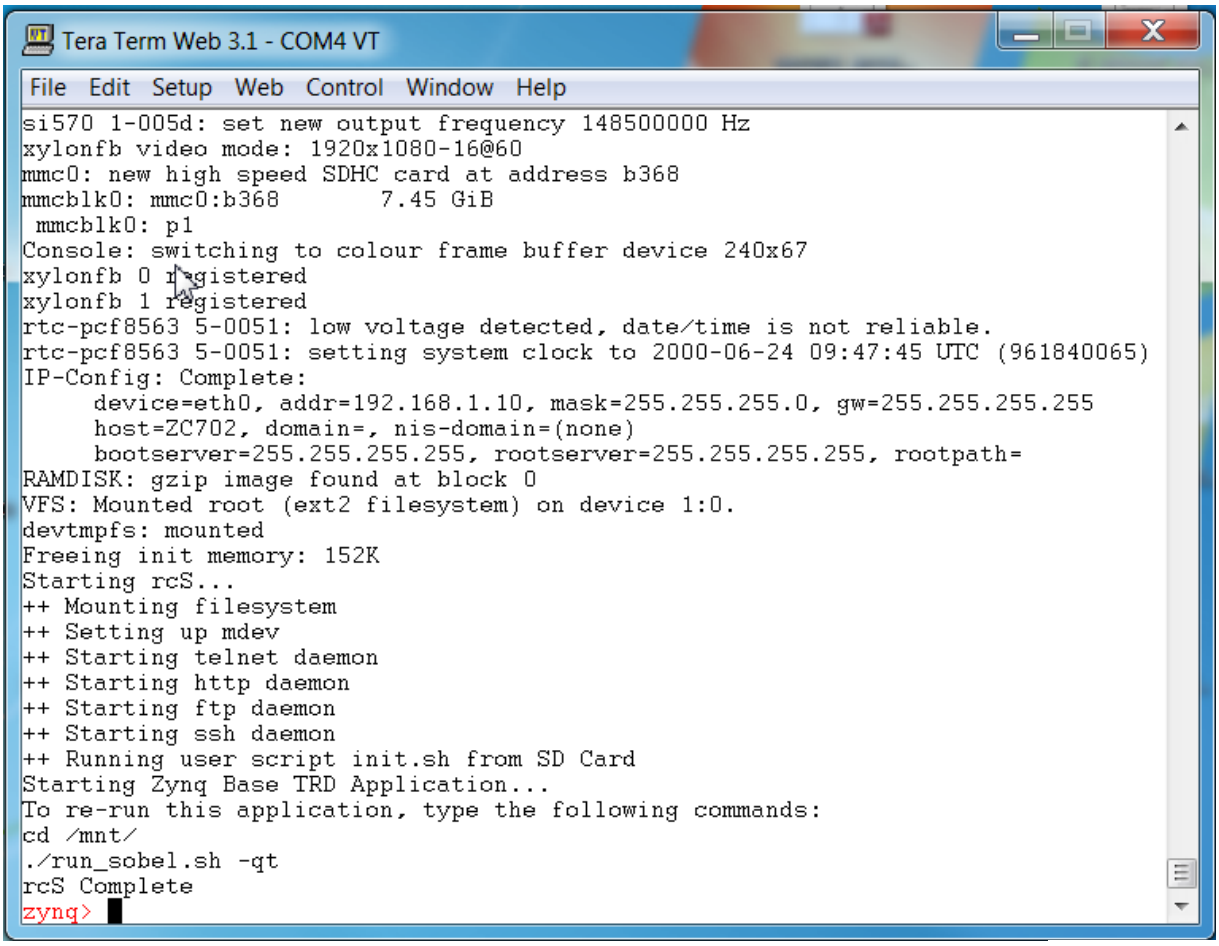


図 2-6 : COM ポートの選択 (COM1 ～ COM4)

非セキュア TRD アプリケーションの実行

1. インストール済みの Tera Term または同等の端末プログラムを開始し、次のように設定します。
   ビット/秒 = 115200、データ ビット = 8、パリティ = なし、ストップ ビット = 1、フロー制御 = なし
2. ZC702 評価ボード上でブート モード スイッチ (J16) を JTAG モードにします。
3. [SDK] → [Xilinx Tools] → [Program Flash] を呼び出して、`ug1025/zc702_linux_trd/ready_for_download/zc702_linux_trd_ns.mcs` を参照します。オフセットに「`0x0`」と入力して、[Finish] をクリックします。QSPI がプログラムされた後に電源を切ります。

4. 電源を入れて、QSPI モードで TRD をブートします。図 2-7 に、TRD のブートを示します。



UG1025_c4-01_082913

図 2-7 : zc702_linux_trd のブート

ZC702 ボードに関する詳細情報は、『Zynq-7000 XC7Z020 All Programmable SoC 向け ZC702 評価ボード ユーザー ガイド』(UG850) [参照 3] を参照してください。図 2-1 に、工場出荷時の ZC702 ボードのジャンパーおよびスイッチの設定が示されています。スイッチとジャンパーのデフォルト設定一覧は表 2-1 を参照してください。

ZC702 評価キットの最新情報については、Zynq-7000 All Programmable SoC ZC702 評価キットの資料ウェブサイトを参照してください。ZC702 ボードを使用した基本的なハードウェアおよびソフトウェア フローの詳細は、『Zynq-7000 All Programmable SoC : コンセプト、ツール、テクニック ガイド』(UG873) [参照 4] を参照してください。その他の情報は、Zynq-7000 AP SoC の資料ウェブサイトを参照してください。

第 3 章

AES/RSA キーの生成およびプログラミング

概要

この章では、キーの生成およびプログラミングについて説明します。図 3-1 に、ここで説明するトピックを示します。

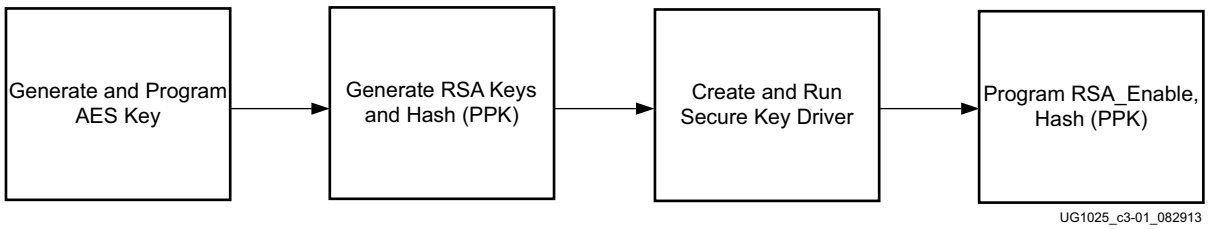


図 3-1 : キーの生成およびプログラミング

Zynq®-7000 All Programmable SoC (AP SoC) で使用される暗号キーを次に示します。

- 256 ビットの AES 暗号キー
- 256 ビットの HMAC 暗号キー
- RSA プライマリ非公開キー (PSK)
- RSA プライマリ公開キー (PPK)
- RSA セカンダリ非公開キー (SSK)
- RSA セカンダリ公開キー (SPK)

推奨されるセキュア ブート フローを実行するため、AES および RSA キーを生成して Zynq AP SoC 内にプログラムしてください。AES キーは、Bootgen を使用して生成します。AES キーは、バッテリ バックアップ RAM (BBRAM) または eFUSE にプログラムできます。BBRAM は、再プログラム可能でゼロ クリアなメモリです。eFUSE は 1 回のみプログラム可能 (OTP) です。

Zynq AP SoC では、プライマリ RSA キーを使用してセカンダリ キーを認証します。そしてセカンダリ キーを使用してパーティション (ELF、BIT ファイルなど) を認証します。RSA プライマリおよびセカンダリ キーの生成には、OpenSSL を使用します。OpenSSL は入手が簡単と言う理由で使用していますが、その他の方法でも可能です。OpenSSL で生成される RSA キーは、非公開キーと公開キーの 2 つです。公開キーは、非公開キーのサブセットとなります。セキュリティ上、非公開キーの保護が非常に重要です。RSA の場合、非公開キーを使用して製造サイトで各パーティションに署名し、その署名を検証するために、Zynq AP SoC エンベデッド デバイスに公開キーがプログラムされます。フィールドにあるエンベデッド デバイスには非公開の RSA キーが含まれていないため安全です。

このスタートアップ ガイドでは、iMPACT を使用してデバイスに AES キーがプログラムされます。RSA セキュリティは、ザイリンクスのセキュア キー ドライバーを使用してデバイスにプログラムされます。セキュア キー ドライバーは、AES キーをプログラムできます。

AES キーの生成およびプログラム

次の手順で Bootgen を使用して AES キーを生成します。

1. 次のコマンドを使用して `generate_aeskey.bif` ファイルを生成します。

   ```
   generate_aeskey:
   {
   [aeskeyfile] bbram.nky
   [bootloader, encryption=aes] fsbl.elf
   }
   ```

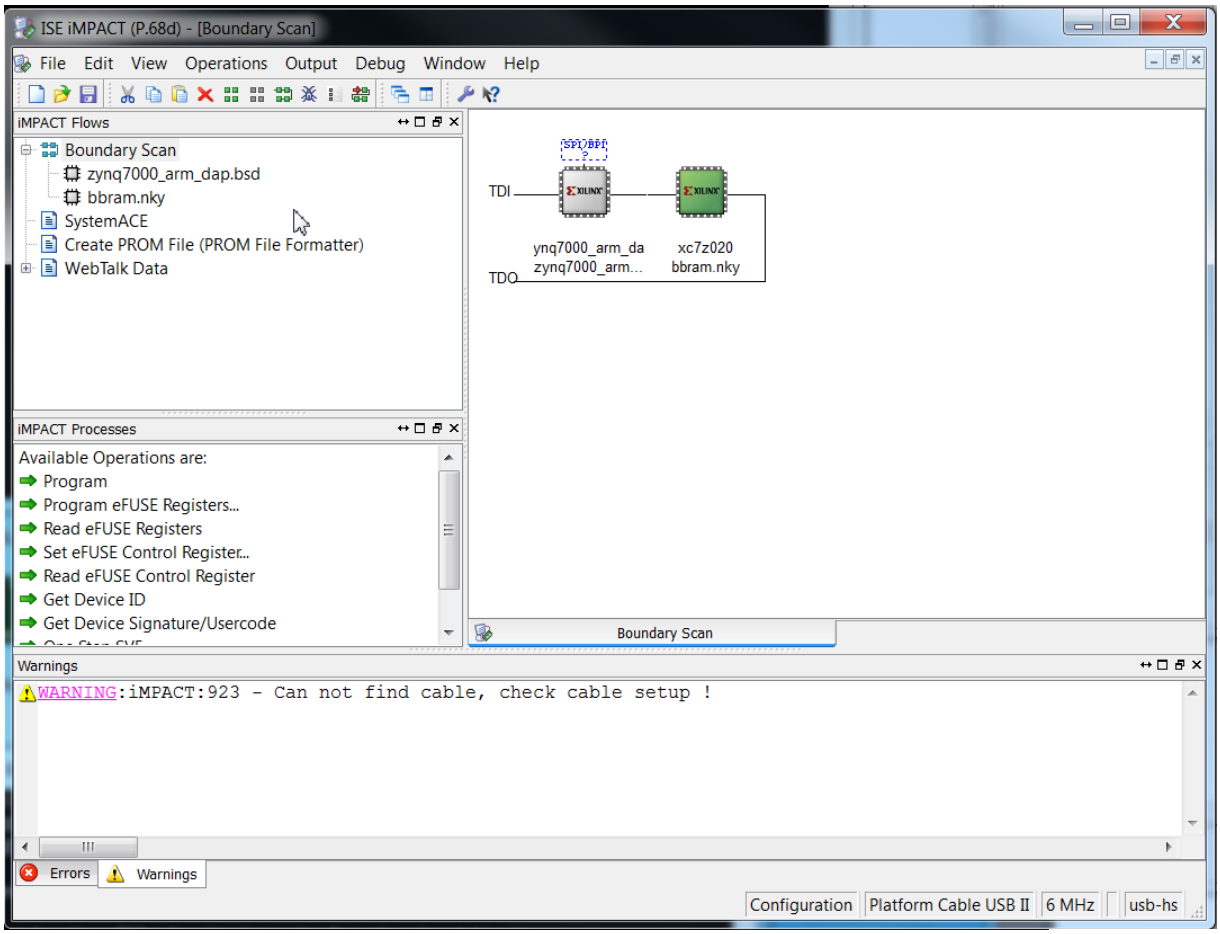
2. 次の Bootgen コマンドを使用して AES キーを生成します。

   ```
   bootgen -image generate_aeskey.bif -o temp.mcs -encrypt bbram
   ```

   指定した AES キーがない場合は、Bootgen が `generate_aeskey.bif` 内にある名前 (この場合は `bbram.nky`) を使用してキーを生成します。

3. AES キーをプログラムするために、iMPACT を起動します。Linux の場合は、impact & を実行します。Windows の場合は、スタート メニューから [ISE Design Suite 14.6] → [ISE Design Tools] → [64-Bit Tools] → [iMPACT] を選択します。プロジェクト ファイルを保存する指示画面が表示された場合は [No] をクリックします。新しい iMPACT プロジェクトのダイアログ ボックスが表示された場合は [Cancel] をクリックします。
4. [iMPACT Flows] ウィンドウで Boundary Scan をダブルクリックします。右クリックして Initialize Chain を選択します。JTAG チェーンに xc7z020 FPGA の ARM® コアが 1 つしか表示されない場合は、ブート モード スイッチを JTAG に変更してチェーンを再び初期化します。コンフィギュレーション ファイルの割り当て指示画面が表示されたら [Yes] をクリックします。
5. DAP が緑色表示しているときに [Bypass] をクリックします。xc7z020 を選択して、[Assign New Configuration File] ダイアログ ボックスに手順 1 で生成した AES キーを 入力します。[Device Programming Properties] が表示された場合は [Cancel] をクリックします。

図 3-2 に、iMPACT の JTAG チェーンを示します。

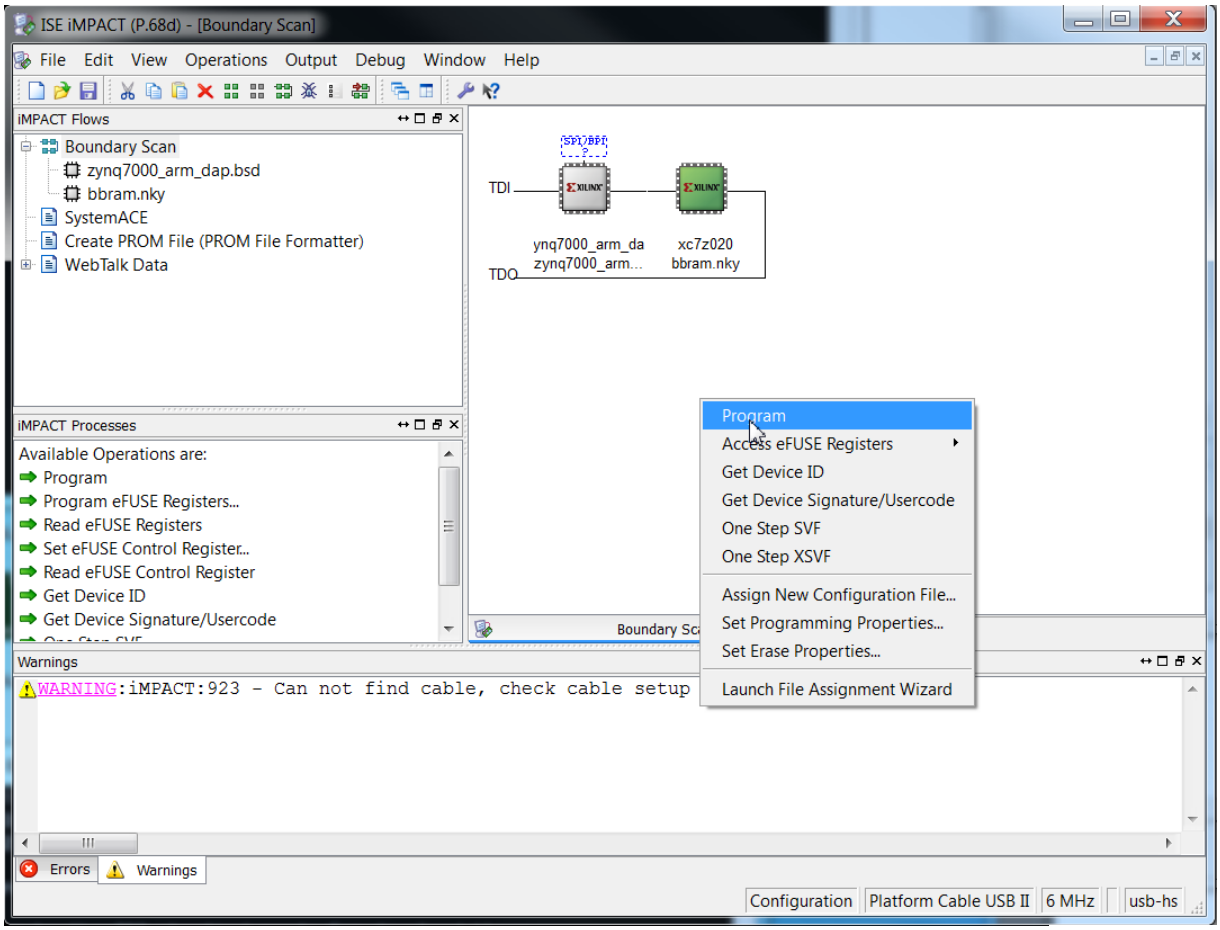


UG1025_c3-02_082913

図 3-2 : iMPACT - 連結されたチェーンの検出

6. 図 3-3 に示すように、iMPACT を使用して BBRAM キーをプログラムします。[ZC7Z020] を右クリックして、[Program] をクリックします。iMPACT では eFUSE キーもプログラムできます。eFUSE キーと BBRAM キーは両方とも Zynq AP SoC へプログラムできます。Bootgen で -encrypt コマンド オプションの引数 efuse | bbram を使用して、使用するキーを指定します。Bootgen がイメージの Boot Header 領域にキー ソースを書き込みます。電源投入時に BootROM コードが Boot Header を読み取り、使用するキー ソースを判断します。

eFUSE の制御も [iMPACT Processes] ウィンドウの [Available Operations] を使用してプログラムできます。eFUSE 制御レジスタは、キーの切り替えを制限したり、JTAG ポートを永久的に無効にするなどの機能に使用されます。



UG1025_c3-03_082913

図 3-3 : iMPACT を使用して BBRAM キーをプログラム

RSA キーの生成

このセクションでは、プライマリおよびセカンダリ RSA キーを生成します。これらのキーを使用して、プライマリ公開キーのハッシュが生成されます。

1. このユーザー ガイドでは、OpenSSL を使用して RSA キーを生成します。(RSA キーの生成手段はほかにもあります。) プライマリおよびセカンダリ非公開 RSA キーは、次の OpenSSL コマンドを使用して生成されます。

   ```
   openssl genrsa -out psk.pk1 2048
   openssl genrsa -out ssk.pk1 2048
   ```

   RSA の場合、公開キーは非公開キーの中に含まれます。次の OpenSSL コマンドを使用して、非公開キーから公開キーを抽出します。

   ```
   openssl rsa -pubout -in psk.pk1 -out ppk.pub
   openssl rsa -pubout -in ssk.pk1 -out spk.pub
   ```

2. プライマリ公開キー (PPK) のハッシュを生成します。openssl を使用して RSA キーを生成した後、Bootgen を使用して PPK のハッシュを生成します。次のコマンドを使用して gen_hash_ppk.bif を生成します。

```
gen_hash_ppk:
{
[pskfile] psk.pk1
[sskfile] ssk.pk1
[bootloader, authentication=rsa] fsbl.elf
}
```

3. 次を実行します。

```
bootgen -image gen_hash_ppk.bif -efuseppkbits hash_ppk.txt
```

hash_ppk.txt ファイルに PPK のハッシュが含まれます。

セキュア キー ドライバーを生成して RSA 機能を有効化

RSA を使用するには、PS eFUSE アレイに RSA Enable 制御ビットと PPK のハッシュをプログラムする必要があります。これを行うには、ソフトウェア開発キット (SDK) を使用してセキュア キー ドライバー ソフトウェアのプロジェクトを作成します。このプロジェクトは、ug1025/zc702_secure_driver ディレクトリにあります。セキュア キー ドライバーの機能は、xilskey_efuse_example.c および xilskey_input.h ファイルを編集して操作します。

RSA を有効にして PPK のハッシュをプログラムするには、xilskey_input.h ファイルを編集して PS eFUSE、PL eFUSE、またはそれらの両方をプログラムします。eFUSE は OTP であるため、すでにプログラム済みの場合には、PPK ハッシュの読み取りのみ可能となるように xilskey_efuse_example.c ファイルを編集できます。

次のように xilskey_input.h ファイルを編集します。

1. XSK_EFUSEPS_DRIVER を定義します。
2. コメントを使用して XSK_EFSUSEPL_DRIVER を無効にします。
3. hash_ppk.txt から XSK_EFUSEPS_RSA_KEY_HASH_VALUE に PPK ハッシュをコピーします。
4. XSK_EFUSEPS_ENABLE_RSA_KEY_HASH を TRUE に設定します。
5. XSK_EFUSEPS_ENABLE_RSA_AUTH を TRUE に設定します。

SDK を使用してセキュア キー ドライバーをコンパイルします。Linux の場合、xsdk & を実行して SDK を起動します。Windows の場合、Xilinx SDK 2013.2 をダブルクリックするか、[スタート] → [ISE Design Suite 14.6] → [EDK] → [Xilinx Software Development Kit] を選択して SDK を起動します。図 3-4 のようにワークスペースを選択します。

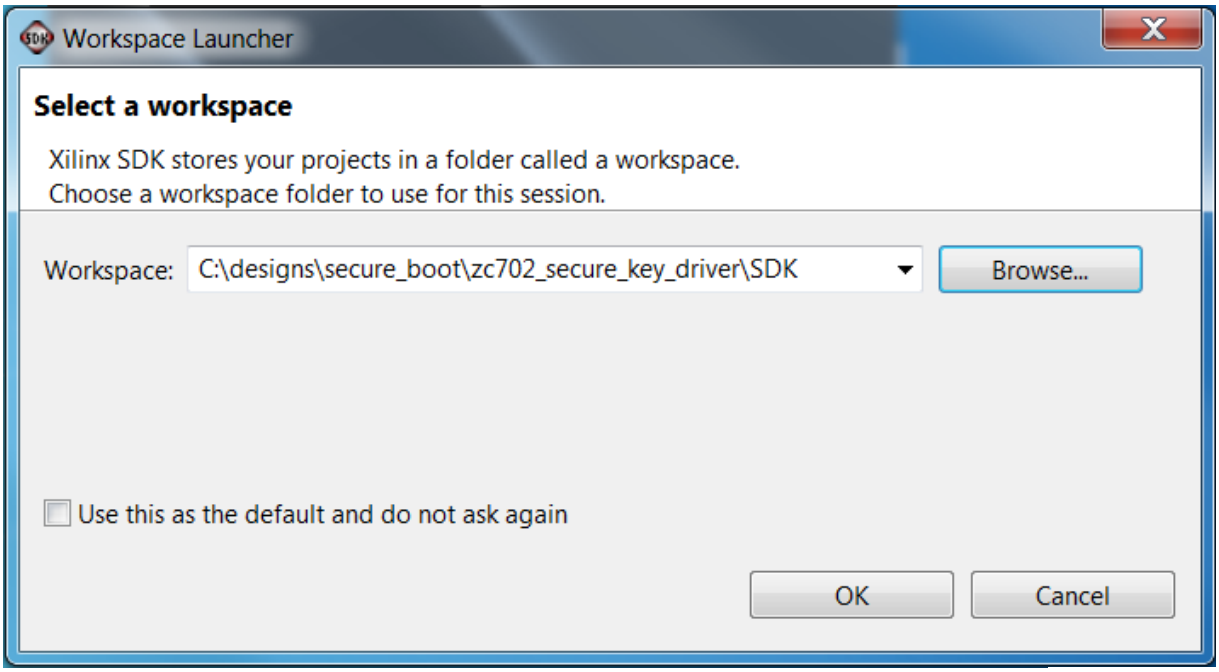


図 3-4 : セキュア キー ドライバーのワークスペースを選択

6. 図 3-5 のように新しいアプリケーション プロジェクトを作成します。[Project name] に「secure_key_driver」と入力します。その他のオプションはデフォルト値のままで使用します。[Next] をクリックします。

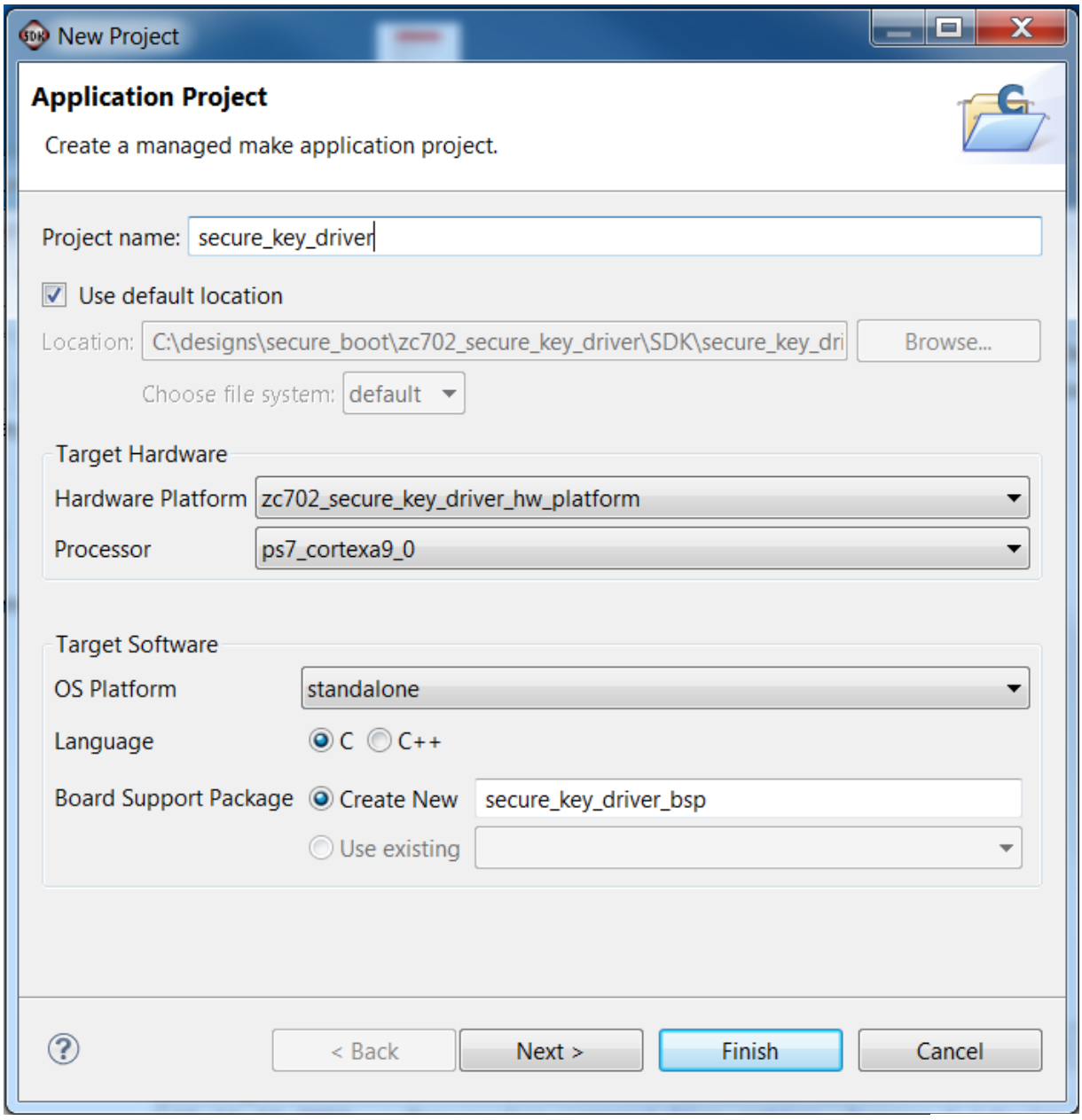


図 3-5 : セキュア キー ドライバーの作成

7. 図 3-6 のように [Empty Application] を選択して、ダイアログ ボックスが表示されるたら、secure_key_driver プロジェクトを指定して [Finish] をクリックします。

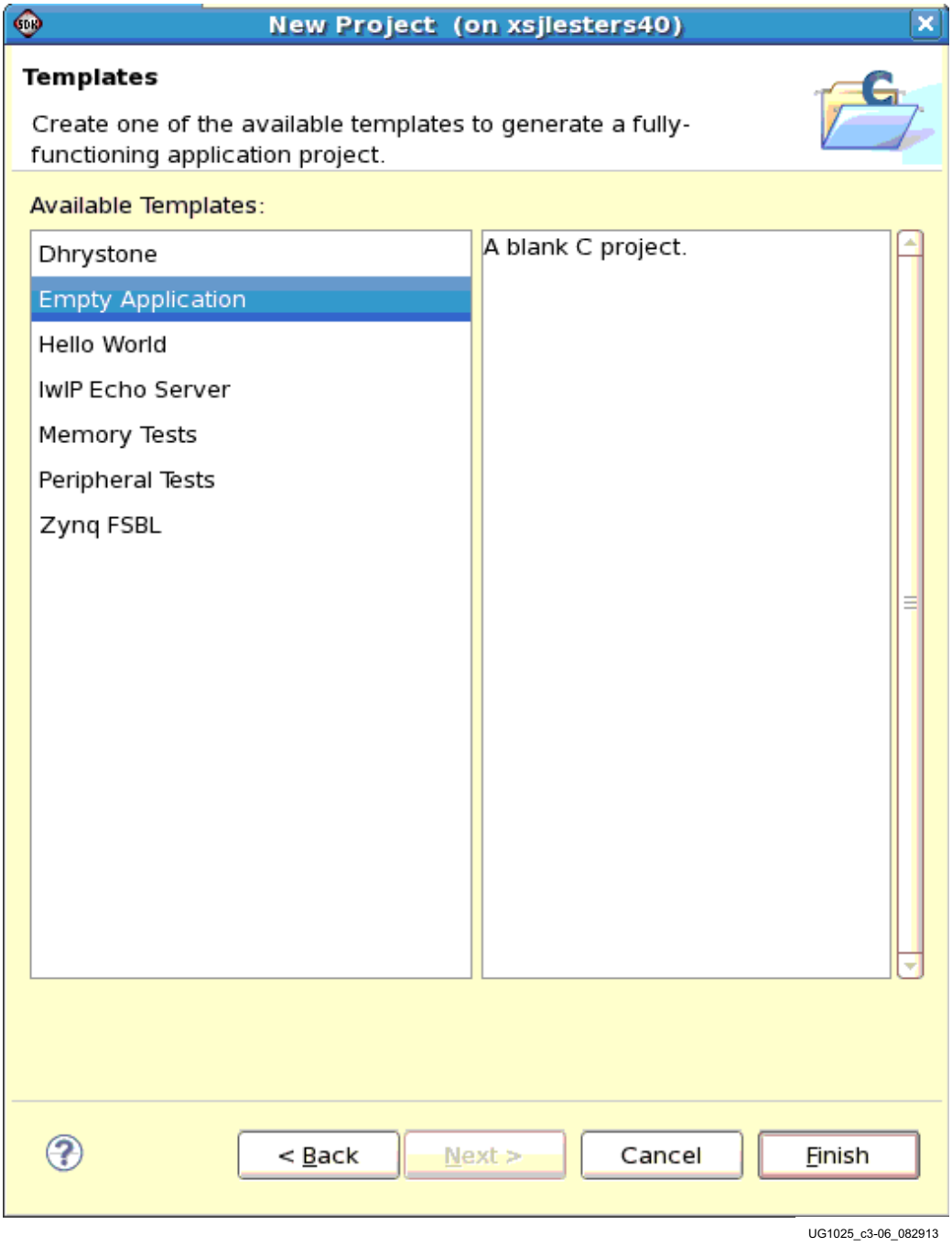


図 3-6 : 空のアプリケーションの作成

8. 図 3-7 のように [secure_key_driver_bsp] を右クリックして、[Board Support Package Settings] をクリックします。

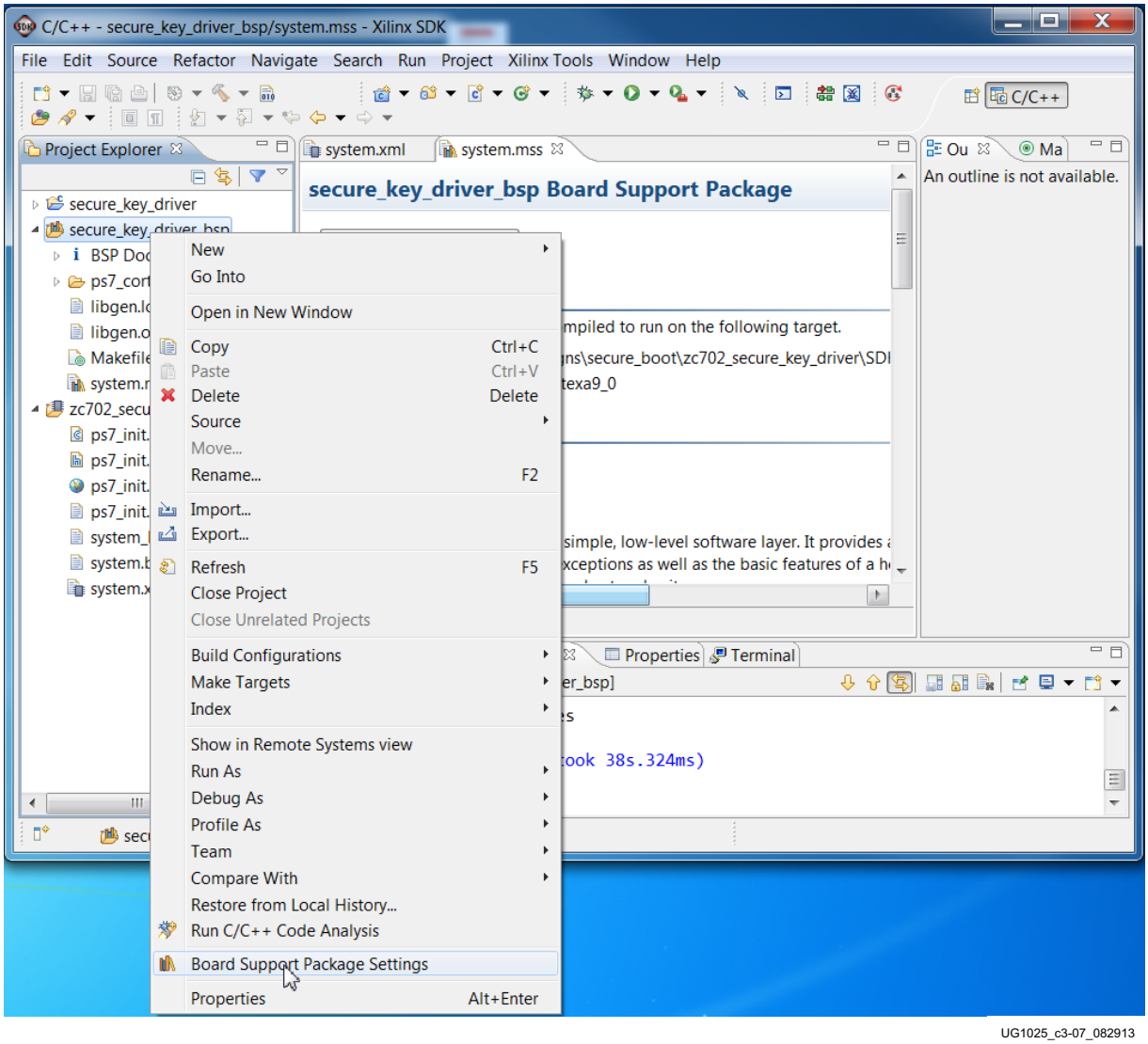


図 3-7 : ボード サポート パッケージの設定

9. 図 3-8 のように [xilskey library] を選択して、ボード サポート パッケージ (BSP) を再構築します。[OK] をクリックします。

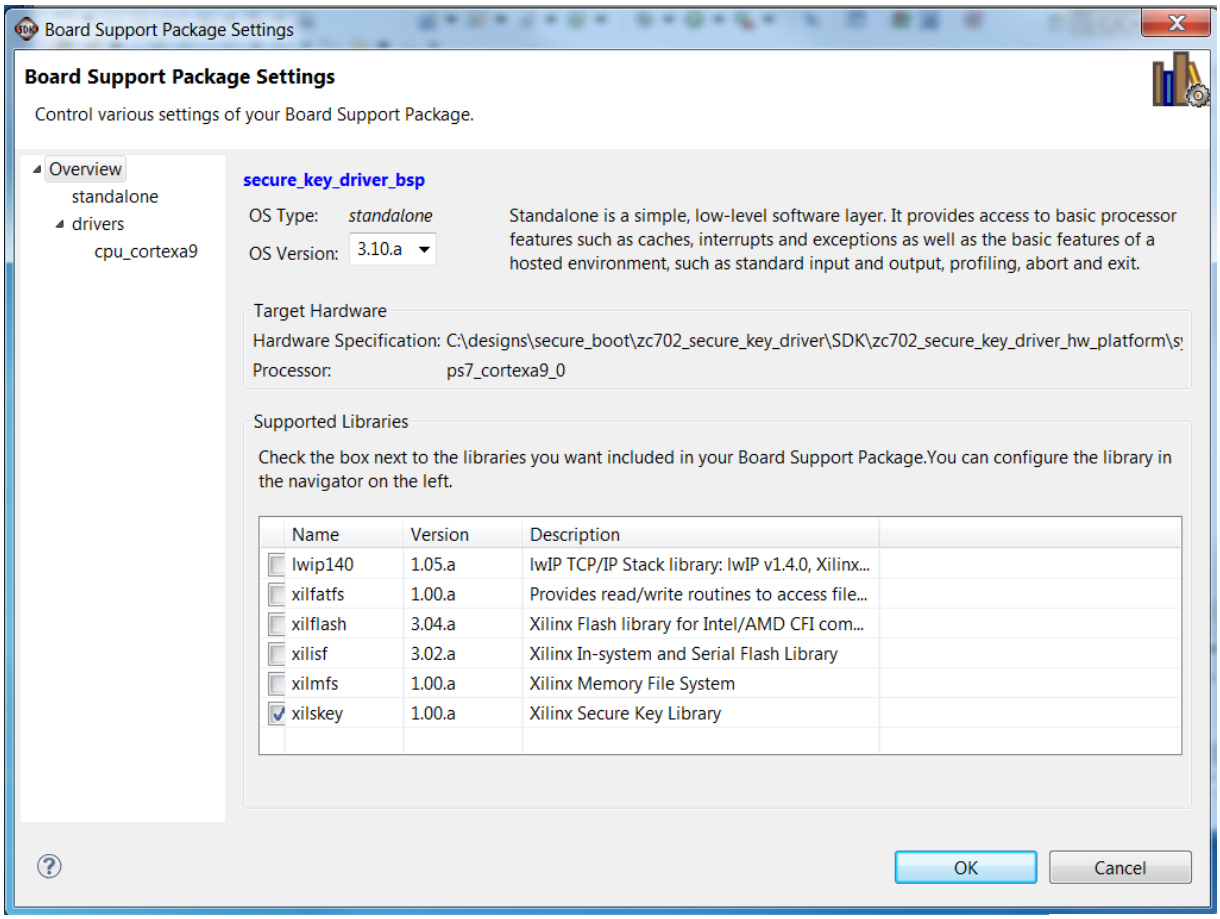


図 3-8 : セキュア キー ライブラリの選択

10. xilskey ライブラリは、secure_key_driver_bsp/ps7_cortexa9_0/lib の下にコンパイルされ、[Project Explorer] ウィンドウに表示されます。図 3-9 を参照してください。

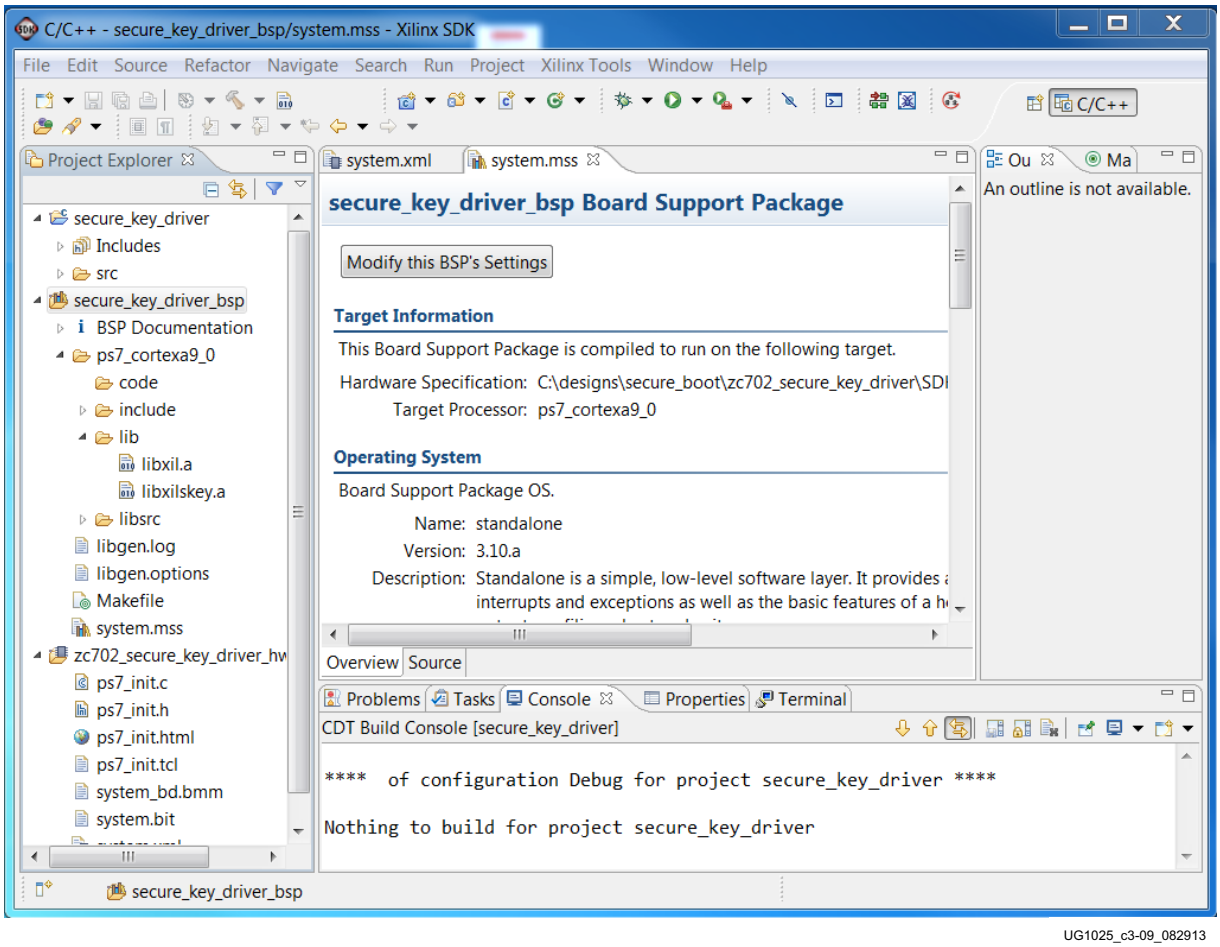
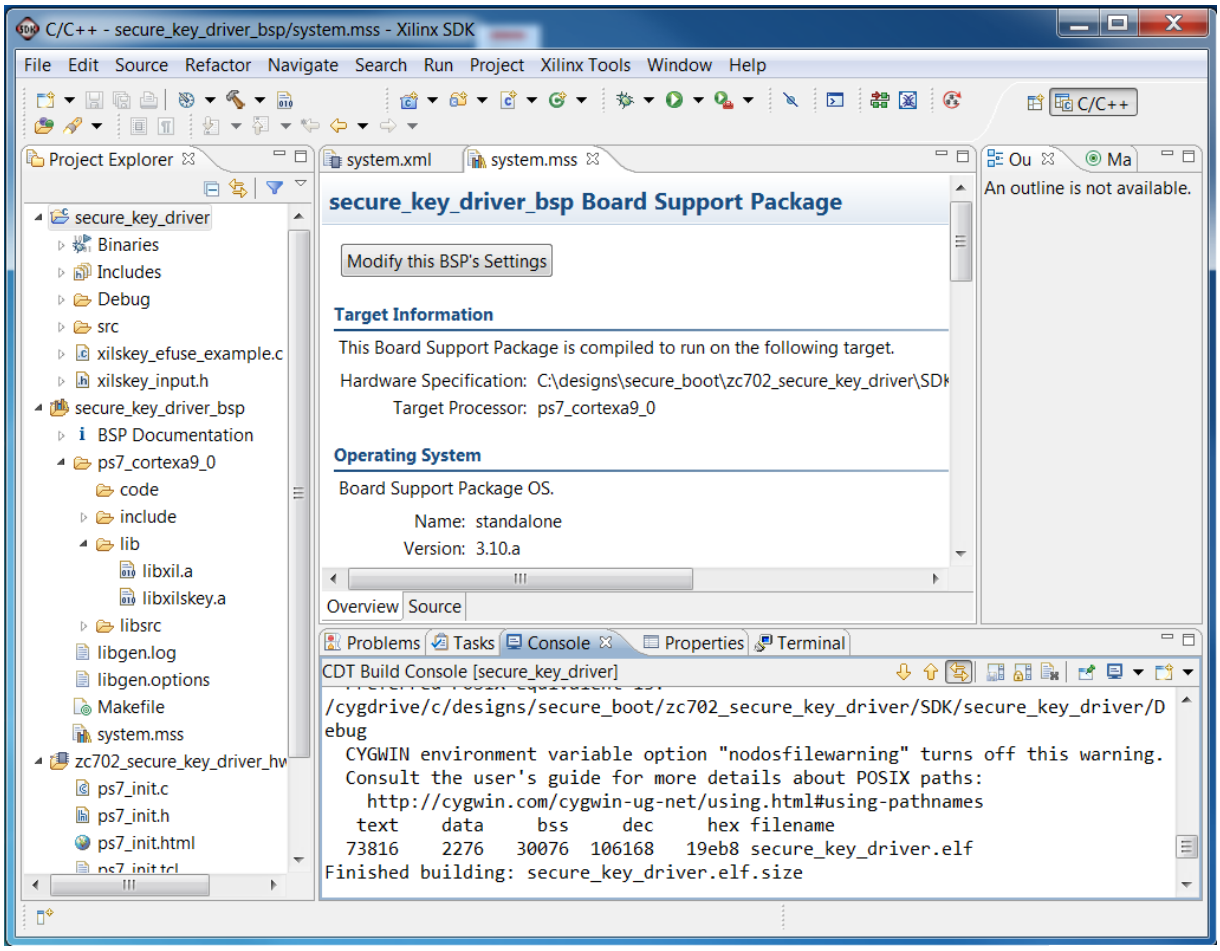


図 3-9 : コンパイルされた xilskey ライブラリ

11. secure_key_driver を選択し、[File] → [Import] → [General Files] を選択して、アプリケーション プロジェクトに `xilskey_efuse_example.c` および `xilskey_input.h` をインポートします。secure_key_driver プロジェクトを選択して、[Project] → [Build Project] を実行します。図 3-10 に、コンパイルされた secure_key_driver プロジェクトを示します。



UG1025_c3-10_082913

図 3-10 : コンパイルされたセキュア キー ドライバー プロジェクト

セキュア キー ドライバーを実行して RSA 機能を有効化

セキュア キー ドライバーは、いかなるブート モードでも実行できます。最もシンプルな方法は JTAG であり、次の XMD コマンドを使用します。コマンド プロンプトに「xmd」と入力するか、SDK で [Xilinx Tools] → [XMD Console] をクリックして XMD コンソールを開きます。

```
connect arm hw
dow fsbl.elf
con
stop
dow secure_key_driver.elf
```

```
con
stop
```

読み取り可能に設定して secure_key_driver をコンパイルすると、通信端末を開いて PPK のハッシュ値を確認できます。

第 4 章

Bootgen イメージ フォーマット ファイルの作成

概要

Bootgen を使用して、Zynq®-7000 All Programmable SoC (AP SoC) の起動に使用されるファイル イメージを生成します。Bootgen へ入力されるファイルは、イメージ内に含まれるパーティションをリストした Bootgen Image Format (BIF) ファイルです。パーティションとは、イメージを構成するソフトウェア、データ、およびビットストリーム ファイルです。たとえば、TRD のパーティションは、`fsbl.elf`、`system.bit`、`u-boot.elf`、`uImage.bin`、`uramdisk.image.gz`、`devicetree.dtb`、および `sobel_cmd.elf` です。

BIF では、属性を使用してパーティションの暗号化 [encryption] や認証 [authentication] を指定します。また、不揮発性メモリ (NVM) のオフセット アドレス [offset] やダブルデータレート (DDR) メモリのロード アドレス [load] を指定する場合にも属性を使用します。

Bootgen は、イメージ ファイルを MCS フォーマットまたは BIN フォーマットで出力します。Zynq Flash Programmer を使用してクワッド シリアル ペリフェラル インターフェイス (QSPI) へ書き込みを実行する場合は、Intel MCS フォーマットを使用します。U-Boot を使用して QSPI をプログラムする場合や SD ブート モードを使用する場合には、BIN フォーマットを使用します。

14.6 では、RSA およびパーティション ベースで暗号/認証を指定する機能は、コマンド ラインで Bootgen を使用する場合のみ有効です。14.6 の Bootgen GUI では、セキュリティ機能のサポートに制限があります。また、14.6 では、Bootgen コマンド ラインは Linux 上でのみ利用できます。

TRD を安全に起動

TRD を安全に起動するために使用される BIF を次に示します。

trd_image:{

```
[aeskeyfile] bbram.nky
[pskfile] psk.pk1
[sskfile] ssk.pk1
[bootloader, encryption=aes, authentication=rsa] fsbl.elf
[encrytion=aes, authentication=rsa] system.bit
[authentication=rsa] u-boot.elf
[authentication=rsa, load= 0x3000000, offset=0x100000] uImage.bin
[authentication=rsa, load=0x2A000000, offset=0x600000] devicetree.dtb
[authentication=rsa, load=0x2000000, offset=0x620000] uramdisk.image.gz
[encryption=aes, authentication=rsa] sobel_cmd.elf
```

この BIF では、トラスト チェーンを使用してソフトウェアがロードされるように、すべてのパーティションが認証されます。FSBL は暗号化されます。`system.bit` や `sobel_cmd.elf` などの機密性の高いパーティションは暗号化されますが、U-Boot や Linux などのオープン ソース パーティションは暗号化されません。

ビットストリーム パーティションが含まれているため、この BIF を使用して作成されたイメージは大きすぎて ZC702 QSPI 内 (16MB) に収まりません。BIF から 4.4MB の `system.bit` を削除すると格納可能になります。

[aeskeyfile]、[pskfile]、および [sskfile] 属性で指定した暗号キーの生成については、第 3 章を参照してください。

この BIF では、FSBL が指定された DDR アドレスへパーティションをロードできるように、[offset] 属性と [load] 属性を使用します。ほとんどのシステムでは、U-Boot を使用してパーティションをロードします。14.6 リリースの時点では、パーティションの認証に使用される RSA ライブラリが U-Boot に含まれていません。このため、FSBL を使用してパーティションを認証/ロードします。すべてのパーティションが FSBL で認証されるフローをサポートするために、U-Boot `zynq_common.h` は変更されています。

注記 : U-Boot の認証機能は、14.7 リリースでサポートされる予定です。

トラスト チェーンを使用してデバイスがブートされるよう、すべてのパーティションが認証される必要があります。つまり、各パーティションはロード前に認証されます。ほとんどの場合、すべてのパーティションは暗号化されていません。暗号は機密性を与えるため、機密情報を含むパーティションは暗号化すべきです。一般的には、Linux や U-Boot などのオープン ソース パーティションを暗号化することはあまり意味がないと考えられています。暗号化はブート時間を増加する可能性がありますが、これまでにテストした NVM コンフィギュレーションでは、セキュア ブートと非セキュア ブートのブート時間に差はありません。

AES は既知のプレーン テキスト攻撃に対して脆弱ではありませんが、それ以外の攻撃を受ける可能性があるため、必要ない場合に暗号化を使用すべきではありません。暗号化は機密情報を保護するために使用してください。

QSPI モードで TRD を安全に起動するには、次の手順を実行します。

1. 第 2 章で説明したとおりに、ZC702 評価ボードを設定します。
2. 第 2 章で説明したとおりに、Tera Term を設定します。
3. UG1025/zc702_secure_key_driver/SDK ディレクトリへ移動します。第 3 章で説明したとおりに `secure_key_driver.elf` プロジェクトを作成します。PS RSA_Enable および PPK eFUSE のハッシュをプログラムするために、次の xmd コマンドを実行します。

   ```
   connect arm hw
   dow fsbl.elf
   con
   stop
   dow secure_key_driver.elf
   con
   exit
   ```

4. ug1025/zc702_linux_trd ディレクトリへ移動します。このプロジェクトの BIF 機能を確認する場合は、`zc702_linux_trd.bif` を開いて属性を確認します。次を実行して TRD イメージを生成します。

   ```
   bootgen -image zc702_linux_trd.bif -o zc702_linux_trd.mcs -encrypt bbram
   ```

   注記 : SDK 14.6 リリースの場合、Bootgen コマンド ラインは Linux 上で実行する必要があります。Windows は、SDK 14.7 でサポートされる予定です。

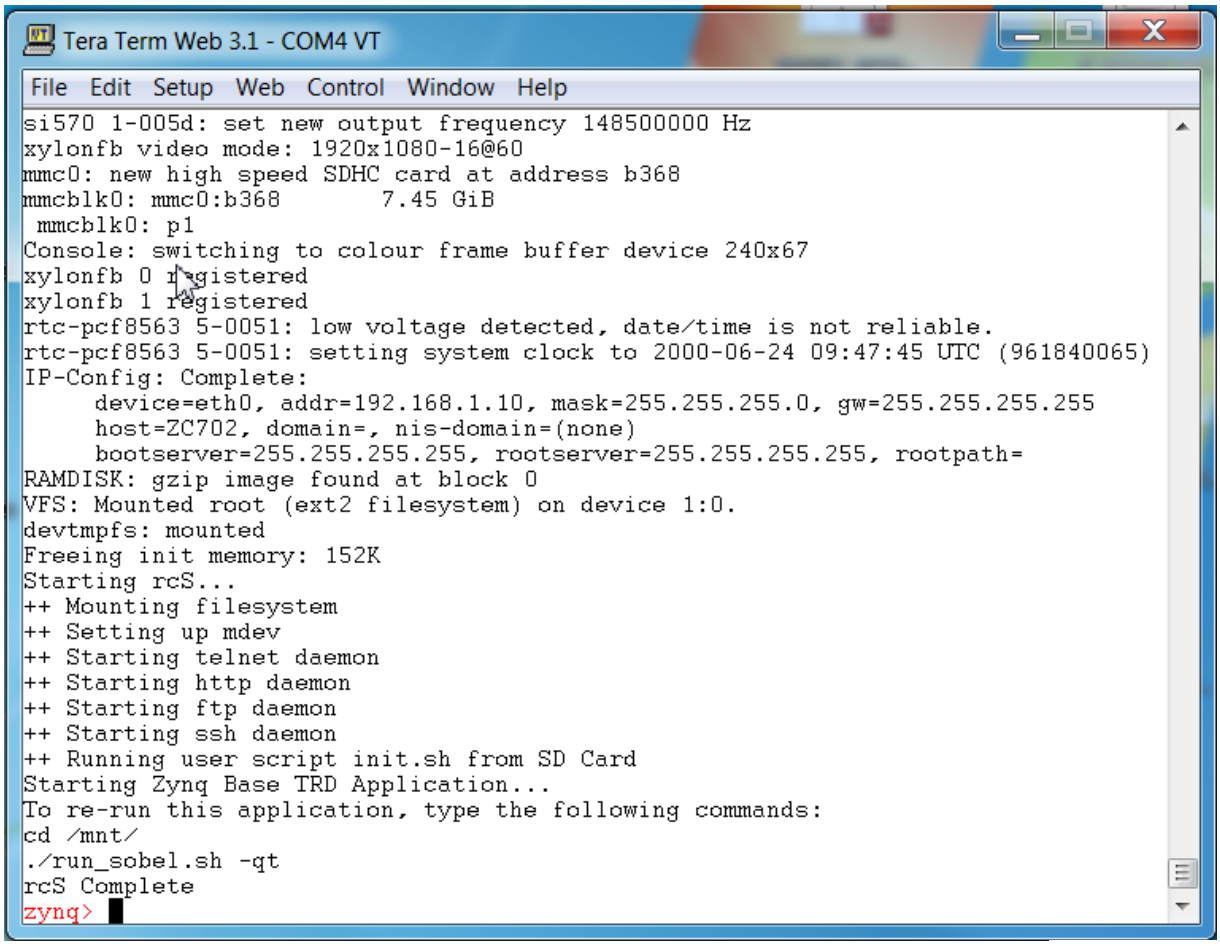
5. Linux の場合、xsdk & を実行して SDK を起動します。ワークスペースを ug1025/zc702_linux_trd/SDK に指定します。
6. SDK で [Xilinx Tools] → [Program Flash] をクリックします。電源を切ります。
7. 次のイメージを指定します。

   ```
   ug1025/zc702_linux_trd/zc702_linux_trd.mcs
   ```

   前に述べたとおり、この MCS イメージには TRD ビットストリームが含まれていません。TRD をブートするには、ビットストリームを含む BIN イメージを生成して、SD モードでブートします。

8. J25 を 1 へ移動、または評価ボードが SW16 を使用している場合にはスイッチ 4 を 1 へ移動して、ZC702 評価ボード上のブート モードを QSPI に変更します。

9. 電源を一旦切ってから再度オンにします。Petalinux プロンプト画面で Zynq AP SoC のブートを検証します。ログインには、ユーザー名 root、パスワード root を使用します。図 4-1 に、予想される TRD 出力を示します。



UG1025_c4-01_082913

図 4-1 : 予想される TRD 出力

10. 非セキュアとセキュアのブート時間を比較するため、非セキュア ブート用に QSPI へ zc702_linux_trd_ns.mcs をプログラムして手順 7 ～手順 9 を繰り返します。電源を投入してから、通信端末に「zynq>」と表示されるまでの時間を計測します。

ブート時間は、NVM の種類やコンフィギュレーション、セキュリティ、およびイメージのサイズによって異なります。ブート時間の概算が必要な場合は、ザイリンクスの FAE (フィールド アプリケーション エンジニア) へお問い合わせください。ザイリンクス FAE が Boot Timer Estimator を使用して、さまざまなコンフィギュレーションの概算を行います。

プレーンの非セキュア ブートと RSA Enable がプログラムされた非セキュア ブートでは、結果が異なります。RSA Enable がプログラムされている場合は、BIF 内に [pskfile] 属性と [sskfile] 属性が指定されている必要があり、また FSBL パーティションの認証が必要です。

セキュア ストレージに機密性の高いパーティションをプログラムする

機密性の高い情報を保護するためのもう 1 つの手段として、Zynq-7000 AP SoC のセキュリティ境界内にあるセキュア ストレージ (オンチップ メモリ [OCM]、AXI BRAM) に格納する方法があります。これは、リンカー スクリプトまたは Bootgen [load] 属性で指定できます。

DDR ではなく OCM にパーティションを格納するには、リンカー スクリプトを図 4-2 のように変更します。

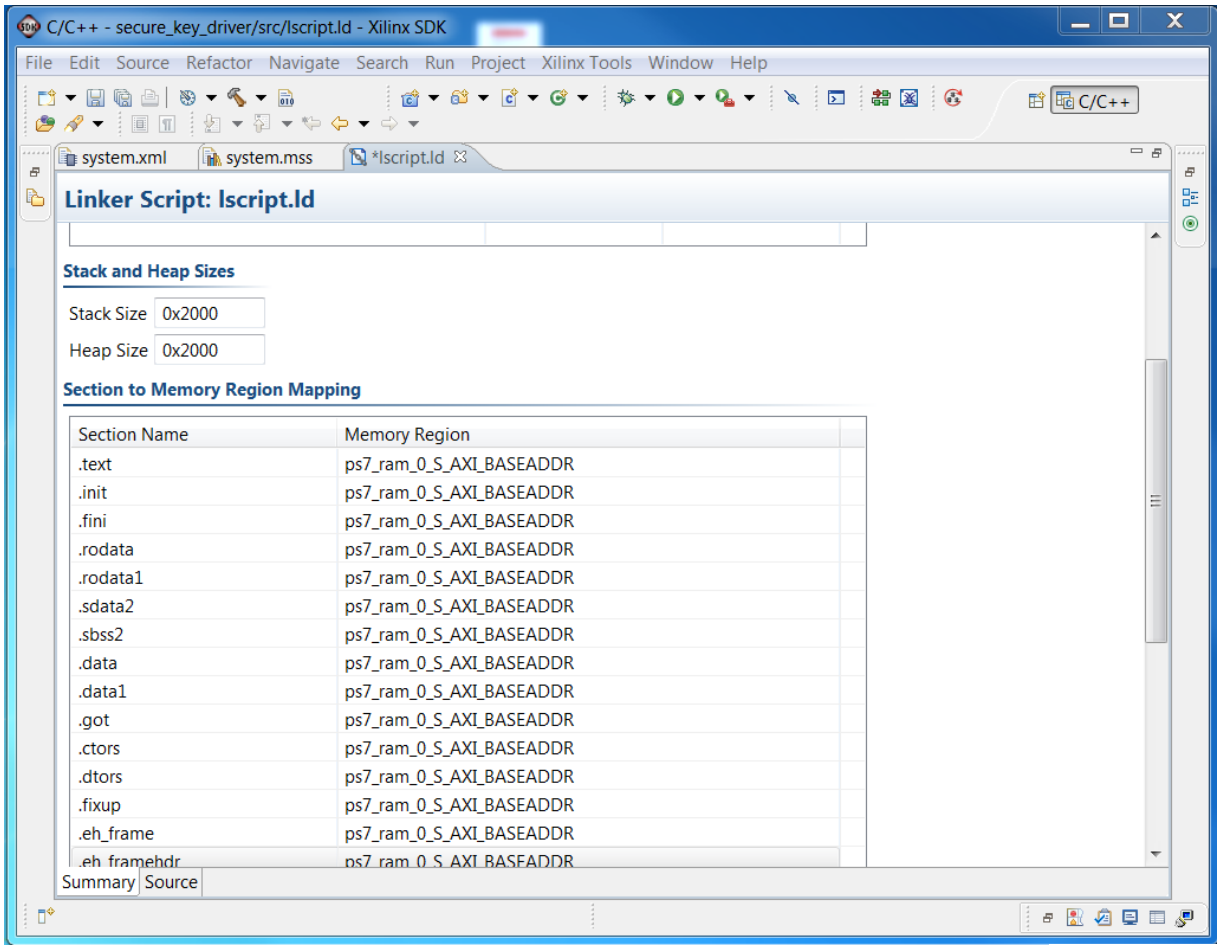


図 4-2 : セキュリティ キー ドライバーを OCM に格納

FSBL は OCM から実行するため、通常、パーティションは FSBL を使用して OCM へロードされることはありません JTAG モードの場合は、デバイスを ps7_init で初期化できるため、XMD コマンドを使用して ELF を OCM へロードできます。その他のブート モードを使用する場合は、FSBL で U-Boot をロードし、U-Boot を使用して OCM へ機密性の高いパーティションをロードします。

OCM は、高速で安全なストレージであるため、機密性の高いソフトウェアやデータ パーティションを格納するのに最適な場所です。データ パーティションはコンパイルされていないソフトウェアです。したがってリンカー スクリプトはありません。OCM 内のデータ パーティションを特定するには、BIF ファイルで [load] 属性を使用します。

第 5 章

高度なキー管理オプション

概要

製造サイトで Bootgen が使用されます。Bootgen の動作モードには Debug と Release があります。この章では、Release モードで Bootgen を実行する方法について説明します。Release モードは、Debug モードとは異なる方法で RSA キーを使用します。Debug モードの場合は非公開キーを使用します。Release モードの場合は、最終的な BIF の中に公開キーとパーティションの署名が含まれます。

最新の暗号技術としてはアルゴリズムが主流であり、非公開キーの保護には不可欠です。Bootgen の Release モードは、非公開キーの保護に役立ちます。この非公開キーの保護は、製造サイトで実施されます。使用したモード (Debug/Release) に関わらず、フィールドへ出されるエンベデッド デバイスには同じ公開キーとコンテンツが含まれます。

非公開キーを保護する 1 つの方法として、エンベデッド製品の最終リリース時にセキュリティ スタッフ (Infosec group) を利用することが挙げられます。これによって、暗号キーを取り扱う社員数を削減できます。多くの場合、ハードウェア セキュリティ モジュール (HSM) がセキュア エリア内で使用されます。HSM は、RSA の 2 種類のキー (非公開と公開) と署名を生成します。HSM に格納された非公開キーは決して漏洩しないため、セキュリティも強化されます。

Boorgen Release モードでは、Bootgen ともう一つのザイリンクス ツール (xil_rsa_sign) が HSM と同じ機能を果たし、ハッシュと署名を生成します。この章では、Bootgen と **xil_rsa_sign** を使用して、非公開キーの保護をさらに強化する方法について説明します。

xil_rsa_sign をコンパイルする際には、ug1025/zc702_secure_key/xil_rsa_sign_src ディレクトリへ移行します。Linux の場合は、`makefile_linux` を makefile にコピーします。Windows の場合は、`makefile_xp` を makefile にコピーします。そして **make** を実行します。

Windows では、makefile 内の `gcc.exe` が正しくない場合に **make** でエラーが生じます。$XILINX_EDK ソフトウェアには `gcc.exe` ファイルが多数あります。Windows を使用しているときは、`$XILINX/gnu/MinGW/5.0.0/nt/bin/gcc.exe` または `cygwin gcc.exe` への完全なパスを使用して `makefile` を編集してください。

```
CC=c:/cygwin/bin/gcc.exe
```

または

```
CC=c:/Xilinx/14.6/ISE_DS/SE/gnu/MinGW/5.0.0/nt/bin/gcc.exe
```

Bootgen の Release モードを使用する手順を次に示します。システムは、ug1025/zc702_secure_key ディレクトリにあります。

1. 次のコマンドと BIF を使用して SPK のハッシュを生成します。

```
bootgen -image spk.bif -generate_hashes
```

`spk.bif` は、次のとおりです。

```
spk:
{
[spkfile] spk.pk1
```

```
}
```

Bootgen が SPK ハッシュ ファイル (spk.pk1.sha256) を生成します。

2. **xil_rsa_sign** を使用して署名を生成します。

```
xil_rsa_sign -gensig -sk psk.pk1 -data spk.pk1.sha256 -out spk.pk1.sha256.sig
```

3. 次の Bootgen コマンドを使用して、パーティションのハッシュを生成します。

```
bootgen -image bootimage_partitions.bif -o temp.txt -encrypt efuse -generate_hashes
```

BIF は次のとおりです。

```
bootimage_partitions:
{
[ppkfile] ppk.pk1
[spkfile] spk.pk1
[spksignature] spk.pk1.sha256.sig
[bootloader, encryption=aes, authentication=rsa] fsbl.elf
[encryption=aes, authentication=rsa] system.bit
[authentication=rsa] u-boot.elf
}
```

パーティションのハッシュが生成され、ハッシュ ファイルは、fsbl.elf.0.sha256、system.bit.0.sha256、および u-boot.elf.0.sha256 となります。

4. 作成されたハッシュの署名を生成します。

```
xil_rsa_sign -gensig -sk ssk.pk1 -data fsbl.elf.0.sha256 -out fsbl.elf.0.sha256.sig

xil_rsa_sign -gensig -sk ssk.pk1 -data system.bit.0.sha256 -out
system.bit.0.sha256.sig

xil_rsa_sign -gensig -sk ssk.pk1 -data u-boot.elf.0.sha256 -out
u-boot.elf.0.sha256.sig
```

5. 次の Bootgen コマンドを使用して、イメージを生成します。

```
bootgen -image bootimage_presign.bif -o zc702_uboot_rm.mcs -encrypt efuse
```

BIF は次のとおりです。

```
bootimage_presign:

{

[aeskeyfile] efuse.nky
[ppkfile] ppk.pk1
[sskfile] spk.pk1
[spksignature] spk.pk1.sha256.sig
[bootloader, encryption=aes, authentication=rsa, presign=fsbl.elf.0.sha256.sig]
fsbl.elf
[encryption=aes, authentication=rsa, presign=system.bit.0.sha256.sig] system.bit
[authentication=rsa, presign=u-boot.elf.0.sha256.sig] u-boot.elf
}
```

6. Bootgen の Debug モードと Bootgen の Release モードが同じイメージ ファイルを生成していることを確認します。これを行うには、Bootgen Debug モードでイメージを生成し、2 つのイメージを比較します。次の Bootgen コマンドを使用して、Debug モードで Bootgen を実行します。

```
bootgen -image bootimage_dm.bif -o zc702_uboot_dm.mcs -encrypt efuse
```

bootimage_dm.bif は次のとおりです。

```
bootimage_dm:
{
[aeskeyfile] efuse.nky
[pskfile] psk.pk1
[sskfile] ssk.pk1
[bootloader, encryption=aes, authentication=rsa] fsbl.elf
[encryption=aes, authentication=rsa] system.bit
[authentication=rsa] u-boot.elf
}
```

次を実行します。

```
diff zc702_uboot_dm.mcs zc702_uboot_rm.mcs
```

これで、2 つのファイルが同じであることを確認できます。

本章のまとめ

- RSA の安全性は、公開 (非公開ではない) キーを使用するフィールド デバイスに基づいており、このキーは必要に応じて変更できます。
- Bootgen の Release モードは、製造サイトで内部攻撃から非公開キーを守るためのオプションです。

第 6 章

マルチブート

概要

オリジナル イメージのブートに失敗した場合、Zynq®-7000 All Programmable SoC (AP SoC) を既知の正常な状態へ確実にブートするためにマルチブートが使用されます。この章では、マルチブートを使用する方法について説明します。

マルチブートでは、3 つのイメージを使用します。FSBL パーティションのみ含むオリジナル イメージは 0x0 に配置されています。標準/アップデート イメージは 0x400000 に配置されています。そしてゴールデン イメージは 0xA00000 に配置されています。

マルチブート例の主な構成要素は、ブートの失敗を引き起こすエラーを定義しています。この例では、`system.bit` 内にエラーが含まれています。

マルチブートは、次の手順で実行されます。

1. FSBL、アップデート イメージ、およびゴールデン イメージを生成します。

   ```
   bootgen -image bootimage_fsbl.bif -o fsbl.bin -encrypt efuse
   bootgen -image bootimage_update.bif -o update.bin -encrypt efuse
   bootgen -image bootimage_golden.bif -o golden.bin -encrypt efuse
   ```

2. `update.bin` をコピーして、アップデート イメージのオリジナル版を保存します。

   ```
   cp update.bin update.bin$
   ```

3. アップデート イメージ内にエラーを挿入するため、HEX エディター (gvim または hd など) を使用します。

   ```
   gvim update.bin
   ```

4. `update.bin` にエラーを挿入します。ビットストリームの開始位置は 0x194C0 です。19570 行目の 1 文字を変更します。`update.bin` を保存します。

5. diff コマンドを使用して、`update.bin` にエラーが含まれていることを確認します。

   ```
   diff update.bin update.bin$
   ```

6. **Tera Term** を開きます。**[File]** → **[Log]** をクリックします。ダイアログ ボックスに「**`multiboot.log`**」と入力します。

7. ug1025/zc702_multiboot/ready_for_download から、`BOOT.bin`、`fsbl.bin`、`update.bin` および `golden.bin` を SD カードへコピーします。ZC702 評価ボードでブート モードを SD に設定します。電源を一旦切ってから再度オンにします。

8. U-Boot のプロンプト画面で次のコマンドを入力します。

   ```
   mmcinfo
   fatload mmc 0 0x100000 fsbl.bin
   sf probe 0 0 0
   sf write 0x100000 0 0x20000
   fatload mmc 0 0x100000 update.bin
   sf write 0x100000 0x400000 ${filesize}
   fatload mmc 0 0x100000 golden.bin
   sf write 0x100000 0xA00000 ${filesize}
   ```

9. 電源を切ります。ブート モードを SD から QSPI へ変更します。電源を入れます。
10. `multiboot.log` を読み込んで、マルチブート動作を検証します。

`multiboot.log` 内で、`update.bin` のロード エラーの原因となるエラーが予想どおりであり、`golden.bin` イメージがロードされていることを確認します。

第 7 章

データ パーティションのローディング

概要

Bootgen は、ELF や BIT ファイルのほかにもデータ パーティションをロードできます。代表的なデータ パーティション アプリケーションには、デジタル信号処理 (DSP) 係数、保健記録、財務記録があります。ELF、BIN、および BIT などのパーティション タイプと同じように、Bootgen の属性でデータ パーティションを暗号化/認証できます。

データ パーティションをロードする BIF は、次のようになります。

```
image:{
{
[bootloader, encryption=aes, authentication=rsa] fsbl.elf
[encryption=aes, authentication=rsa] hello.elf
[encryption=aes, authentication=rsa, load=0xFFFFC000] coefficients.bin
}
```

このデータ ファイル例 (`coefficients.bin`) には、`0101010111001100` が含まれています。

データ ファイルが OCM にロードされているかを確認するには、次の XMD コマンドを実行します。

```
mrd 0xFFFFC000 8
```

一般的に、データ パーティションの格納場所としては OCM が最適です。上記の BIF に問題はありませんが、FSBL が OCM から実行するため、FSBL を使用してパーティションを OCM へロードすべきではありません。データ パーティションを OCM へロードするには、U-Boot を使用する方法を推奨しています。

第 8 章

認証証明のユーザー定義フィールドを使用

概要

RSA を使用する場合は、[authentication] 属性がある各パーティションに認証証明 (AC) が生成されます。AC には、キーや署名のほかに、ロードされるソフトウェアのバージョンを示すために使用できるユーザー定義フィールド (UFD) が含まれます。この UDF は、パーティション サプライヤーの識別やタイム スタンプ情報の提供など、さまざまなユーザー定義機能に使用できます。

UDF は、HEX ファイルで定義されます。HEX ファイルのサイズは最大 56 バイトです。UDF 内の情報を使用するには、FSBL または U-Boot にコードを追加する必要があります。

AC に UDF を作成する手順は次のとおりです。完成プロジェクトは、ug1025/zc702_udf にあります。

1. 16 進数を使用する uboot_v10.hex を生成します。たとえば、16 進数値として「12345678abcdef」を使用します。
2. udf_data 属性を使用して、次のように BIF を生成します。

```
the_image
{
[pskfile] psk.pk1
[sskfile] ssk.pk1
[aeskeyfile] efuse.nky
[bootloader, authentication=rsa, encryption=aes] fsbl.elf
[authentication=rsa, udf_data=uboot_v10.hex] u-boot.elf
}
```

3. 次の Boorgen コマンドを実行します。

```
bootgen -image bootimage.bif -o zc702_udf.mcs -encrypt bbram
```

4. HEX エディター (gvim または hd) を使用して UDF を特定します。図 8-1 では、0025480 行目に完成した zc702_udf デザインの UDF を示しています。

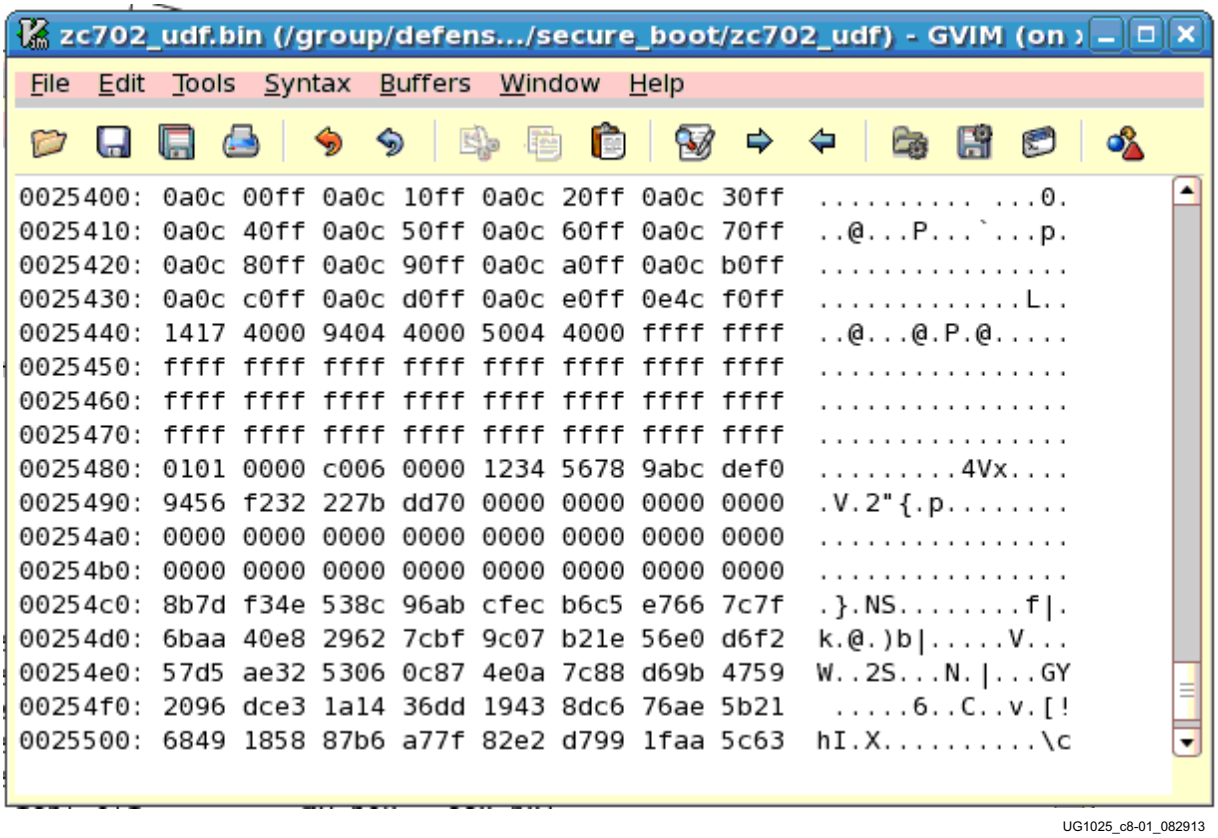


図 8-1 : 認証証明のユーザー定義フィールド

付録 A

その他のリソース

ザイリンクスのリソース

アンサー データベース、資料、ダウンロード、フォーラムなどのサポート リソースを利用するには、次のウェブサイトにアクセスしてください。

japan.xilinx.com/support

常に最新情報を受け取るには、マイ プロフィールからアラートにアンサーを追加してください。

japan.xilinx.com/support/myalerts

ザイリンクスの資料で使用されている技術用語については、次の用語集を参照してください。

japan.xilinx.com/company/terms.htm

ソリューション センター

デバイス、ツール、IP のサポートについては、ザイリンクスのソリューション センターを参照してください。設計に関するアシスタンス、アドバイス、問題解決のヒントなどを記載しています。

参考資料

ZC702 および関連資料の最新情報は、次のウェブサイトから入手できます。

Zynq-7000 All Programmable SoC ZC702 評価キットのウェブサイト

Zynq-7000 All Programmable SoC ZC702 評価キットの資料ウェブサイト

これらのザイリンクス資料およびウェブサイトでは、このユーザー ガイドで役立つ情報を提供しています。

1. 『Zynq-7000 All Programmable SoC ZC702 評価キットおよびビデオ/画像処理キット スタートアップ ガイド』(UG926)
2. 『Zynq-7000 All Programmable SoC のセキュア ブート』(XAPP1175)
3. 『Zynq7000 XC7Z020 All Programmable SoC 向け ZC702 評価ボード ユーザー ガイド』(UG850)
4. 『Zynq-7000 All Programmable SoC : コンセプト、ツール、テクニック ガイド』(UG873)
5. 『AMS101 評価カード ユーザー ガイド』(UG886)
6. 『Zynq-7000 All Programmable SoC テクニカル リファレンス マニュアル』(UG585)

7. 『Zynq-7000 All Programmable SoC ソフトウェア開発者向けガイド』(UG821)
8. Tera Term のウェブサイト : en.sourceforge.jp/projects/ttssh2/releases/
9. Silicon Laboratories 社の USB-UART ドライバー ウェブサイト : www.silabs.com/Support%20Documents/Software/CP210x_VCP_Windows.zip

付録 B

Warranty

THIS LIMITED WARRANTY applies solely to standard hardware development boards and standard hardware programming cables manufactured by or on behalf of Xilinx (“Development Systems”). Subject to the limitations herein, Xilinx warrants that Development Systems, when delivered by Xilinx or its authorized distributor, for ninety (90) days following the delivery date, will be free from defects in material and workmanship and will substantially conform to Xilinx publicly available specifications for such products in effect at the time of delivery. This limited warranty excludes: (i) engineering samples or beta versions of Development Systems (which are provided “AS IS” without warranty); (ii) design defects or errors known as “errata”; (iii) Development Systems procured through unauthorized third parties; and (iv) Development Systems that have been subject to misuse, mishandling, accident, alteration, neglect, unauthorized repair or installation. Furthermore, this limited warranty shall not apply to the use of covered products in an application or environment that is not within Xilinx specifications or in the event of any act, error, neglect or default of Customer. For any breach by Xilinx of this limited warranty, the exclusive remedy of Customer and the sole liability of Xilinx shall be, at the option of Xilinx, to replace or repair the affected products, or to refund to Customer the price of the affected products. The availability of replacement products is subject to product discontinuation policies at Xilinx. Customer may not return product without first obtaining a customer return material authorization (RMA) number from Xilinx.

THE WARRANTIES SET FORTH HEREIN ARE EXCLUSIVE. XILINX DISCLAIMS ALL OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, AND ANY WARRANTY THAT MAY ARISE FROM COURSE OF DEALING, COURSE OF PERFORMANCE, OR USAGE OF TRADE. (2008.10)

Do not throw Xilinx products marked with the “crossed out wheeled bin” in the trash. Directive 2002/96/EC on waste electrical and electronic equipment (WEEE) requires the separate collection of WEEE. Your cooperation is essential in ensuring the proper management of WEEE and the protection of the environment and human health from potential effects arising from the presence of hazardous substances in WEEE. Return the marked products to Xilinx for proper disposal. Further information and instructions for free-of-charge return available at: http:\\www.xilinx.com\ehs\weee.htm.