Zynq-7000 All Programmable SoC Secure Boot

Getting Started Guide

UG1025 (v1.0.1) March 18, 2014

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/12/2013	1.0	Initial Xilinx release.
03/18/2014	1.0.1	Corrected figure references to Figure 2-1 and Figure 2-2. Corrected SW16 setting (position 4) in Table 2-1. Added chapter numbers to PDF bookmarks and Feedback button.

Table of Contents

Introduction

Overview

The Zynq®-7000 All Programmable SoC (AP SoC) provides private key cryptography (AES/HMAC) and public key cryptography (RSA). This allows sensitive software to be encrypted, and it allows the software loaded into the Zynq-7000 AP SoC to be authenticated in a chain of trust. Booting with a chain of trust ensures that an adversary has not tampered with the software. Encrypting sensitive software ensures that it is confidential, i.e., not readable.

AES uses a private key. RSA uses public keys in the Zynq-7000 AP SoC, and the keys can be changed as often as desired, even for partitions within the same image. Cryptographic keys should be changed to limit the time an adversary has to attack the keys.

The Zynq-7000 AP SoC security features do not use programmable resources, so the incremental cost to use security in Zynq-7000 AP SoCs is the time spent learning to use secure boot. This getting started guide shows how to use secure boot on the ZC702 Evaluation Board. The same Xilinx tools are used for non-secure and secure boot, so the tool flow does not change. In the 14.6 release, Bootgen must be used as a command line tool for secure boot.

The reference systems which accompany this guide are available in the following link:

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

The organization of this guide is as follows:

- Chapter 2, ZC702 Evaluation Board Setup
- Chapter 3, Generating and Programming AES and RSA Keys
- Chapter 4, Creating Bootgen Image Format Files
- Chapter 5, Advanced Key Management Options
- Chapter 6, Multiboot
- Chapter 7, Loading a Data Partition
- Chapter 8, Using the User Defined Field in an Authentication Certificate

For additional information on the TRD, see the *Zynq-7000 All Programmable SoC ZC702 Evaluation Kit and Video and Imaging Kit Getting Started Guide* (UG926) [Ref 1].

For additional information on secure boot, see *Secure Boot of Zynq-7000 All Programmable SoC Application Note* (XAPP1175) [Ref 2].

ZC702 Evaluation Board Setup

Introduction

This chapter shows how to set up the ZC702 Evaluation Board.

Note: For a description of all the features on the ZC702 board, see *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide* (UG850) [Ref 3].

ZC702 Setup Requirements

The following is required to set up the ZC702 Evaluation Board.

- Hardware setup:
  - ZC702 Evaluation Board with XC7Z020 CLG484-1 part
  - USB Type-A to Mini-B cable (for UART)
  - AC power adapter (12 VDC)
- Windows software and driver setup:
  - Tera Term Pro (or similar) terminal program. (This program might already be installed. For more information, see the Tera Term website.)
  - Silicon Laboratories USB-UART driver (available for download from Silicon Laboratories USB-UART drivers website. This program might already be installed.)

ZC702 Setup Requirements

The default jumper and switch settings of the ZC702 board are shown in Figure 2-1 and Table 2-1.

To boot from Quad SPI device, set the SW16 switch as shown in Figure 2-1, where position 1, 2, 3, and 5 are switched to the right, and position 4 is switched to the left.

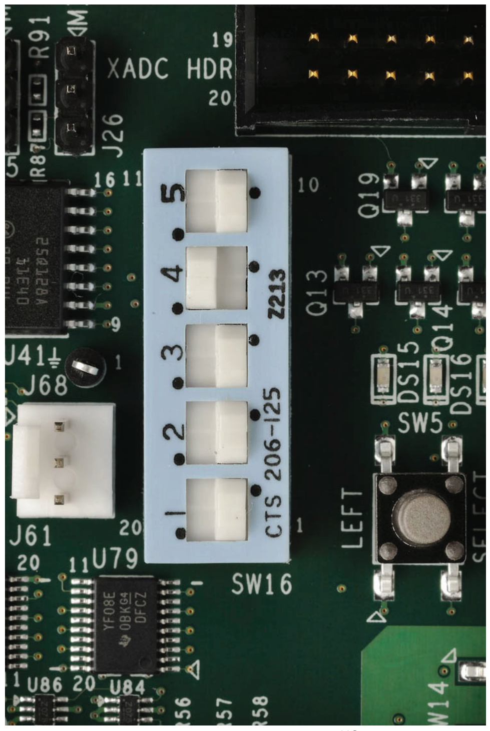


Figure 2-1: **Settings for the Mode Switch to Boot from Quad SPI Mode**

Table 2-1: **Switch Settings for the ZC702 Evaluation Board**

Switch	Position	Setting
SW10 (JTAG chain input select two-position DIP switch)	1	Off
	2	On
SW12 (Two-position DIP switch)	1	Off
	2	Off
SW15 (Two-position DIP switch)	1	Off
	2	Off
SW16 (Five-position DIP switch)	1	Right
	2	Right
	3	Right
	4	Left
	5	Right
SW11 (Power slide switch)	Off	Down

Hardware Setup

This section describes the steps for hardware set-up.

1. Set the SW16 Mode switch settings to those shown in Figure 2-1.
2. With the ZC702 board switched OFF (SW11 in the down position, as shown in Figure 2-2), plug the USB Mini-B cable into the mini-USB port J17 labeled USB-UART on the ZC702 board and the other end into an open USB port on the PC (Figure 2-2).

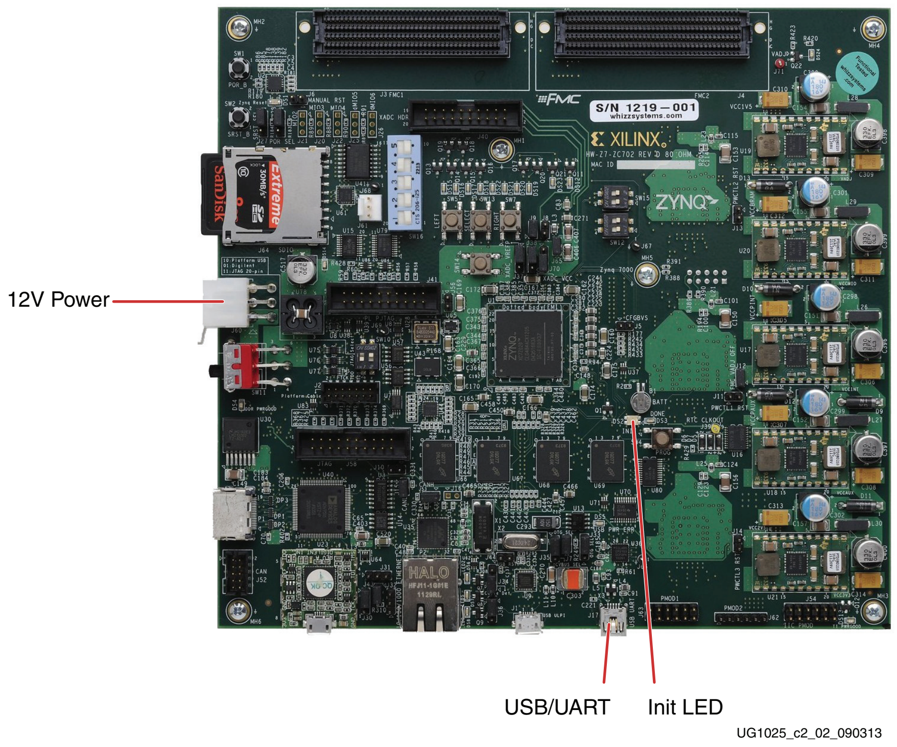
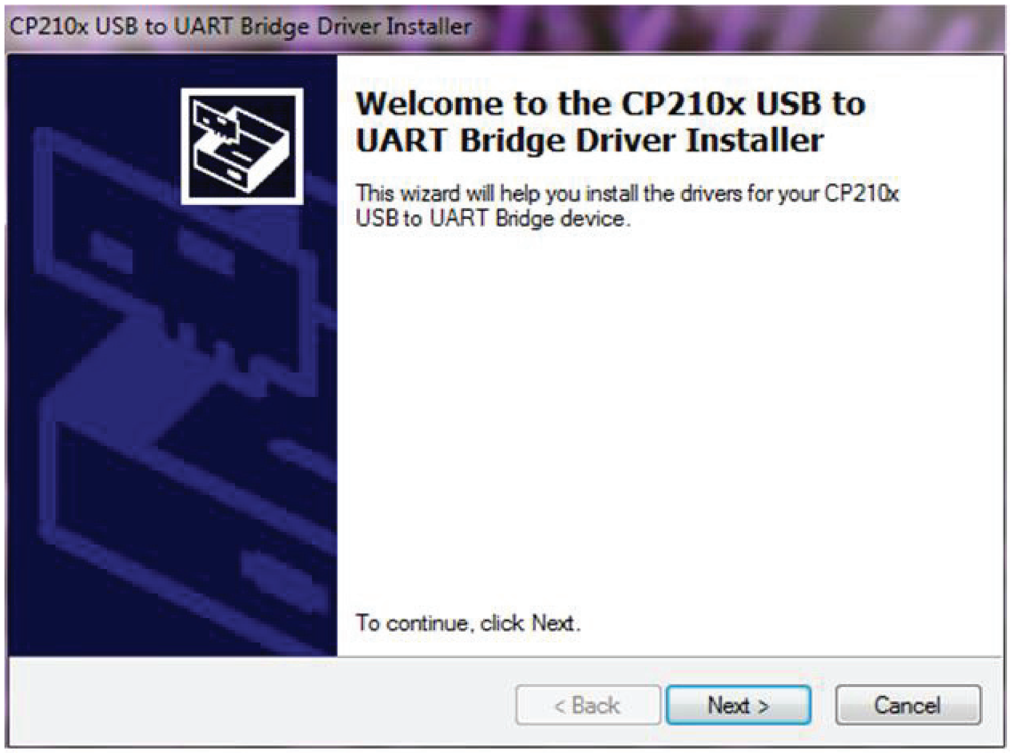


Figure 2-2: **ZC702 with the UART and Power Cable Attached**

3. Connect the power cable.
4. Switch the ZC702 board power to ON (SW11 to up position).

Install the USB-UART Driver

1. Run the downloaded executable UART-USB driver file, listed in ZC702 Setup Requirements, page 9. Running the executable file enables USB-to-UART communications with a host PC. This driver downloads and executes automatically when the board is powered up, or it can be downloaded from the Silicon Laboratories USB-UART drivers website.


CP210x USB to UART Bridge Driver Installer

Welcome to the CP210x USB to UART Bridge Driver Installer

This wizard will help you install the drivers for your CP210x USB to UART Bridge device.

To continue, click Next.

Figure 2-3: **UART Cable Driver Installation**

2. Set the USB-UART connection to a known COM Port and baud rate in the Device Manager.

   a. Left-click **Start Menu** and select **Control Panel** in Windows 7.

   b. Select **Device Manager**.

   c. Right-click the Silicon Labs device in the list and select **Properties.**

   d. Click the **Port Settings** tab. Click the **Advanced...** button.

   e. Select an open COM port between COM1 and COM4. This allows the computer to remember the assignment and not reassign it each time the board serial UART port is plugged in.

   f. Select the baud rate = **115200**, Data bits = **8**, Parity = **None**, Stop Bits = **1**, and Flow control = **None.** Click **OK**.

Note: Steps and diagrams refer to using a Windows XP or Windows 7 host PC.

Figure 2-4 through Figure 2-6 show the steps for setting the USB-UART port.

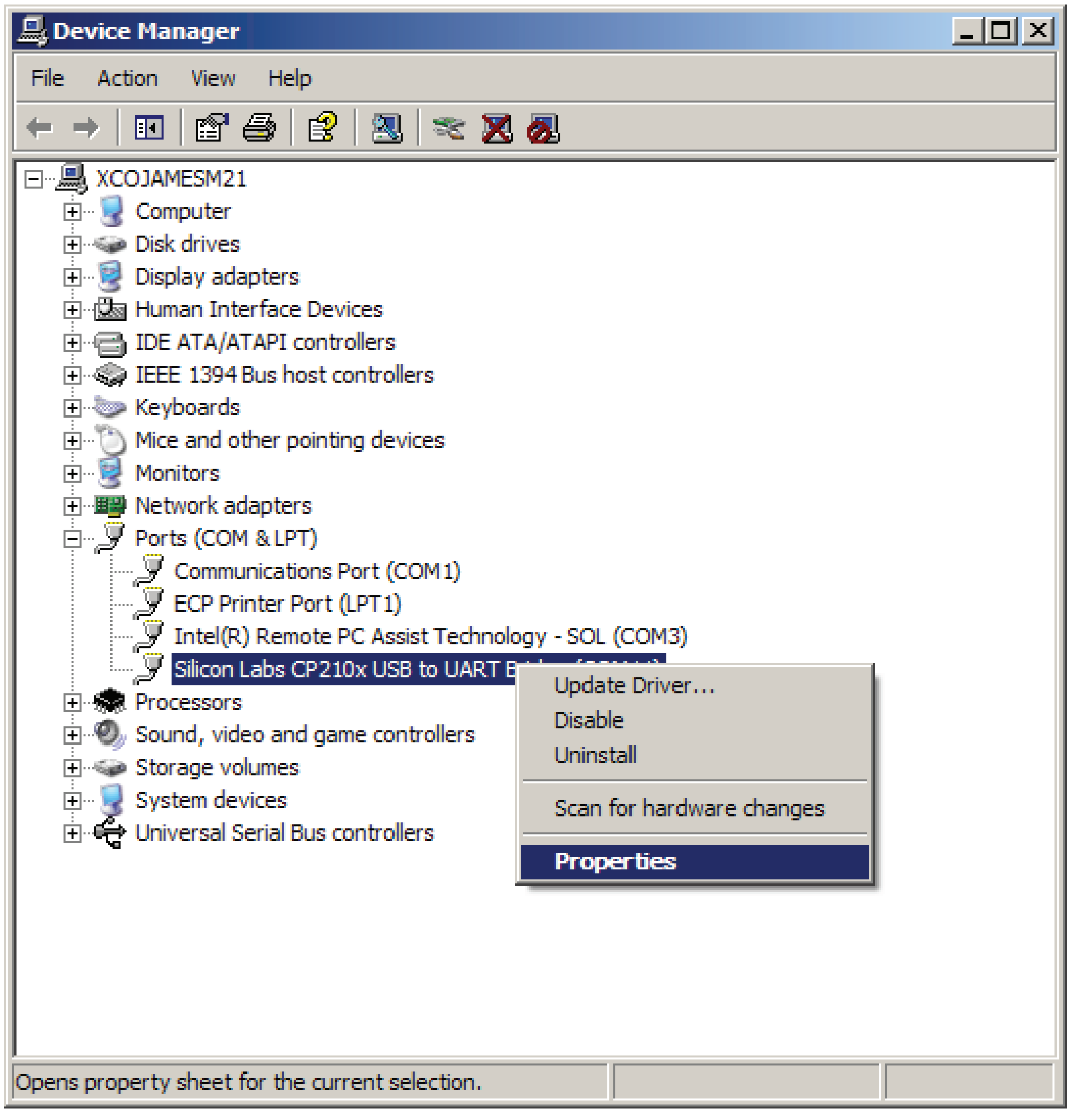
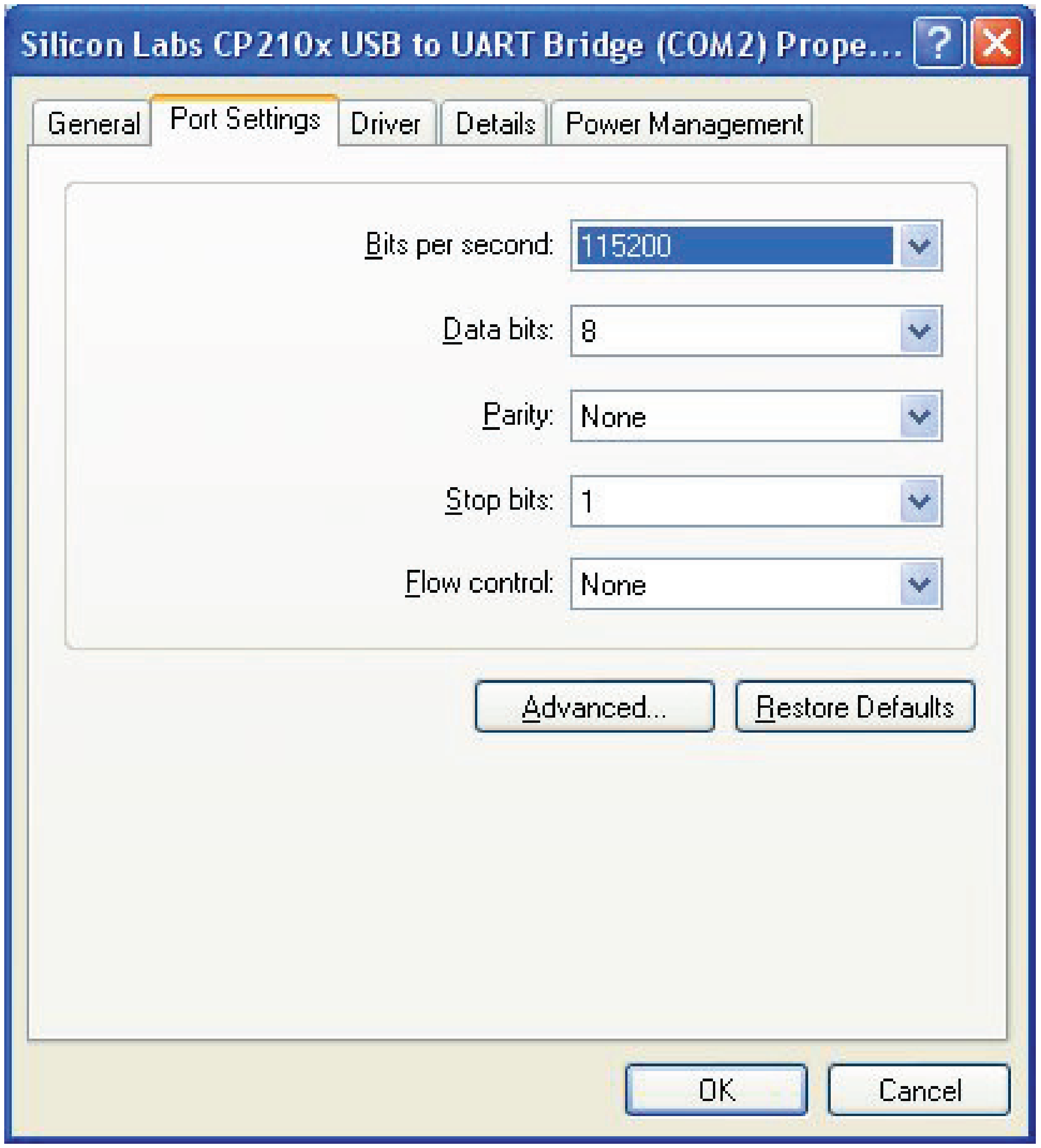

Device Manager
File Action View Help
XCOJAMESM21
Computer
Disk drives
Display adapters
Human Interface Devices
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
Communications Port (COM1)
ECP Printer Port (LPT1)
Intel(R) Remote PC Assist Technology - SOL (COM3)
Silicon Labs CP210x USB to UART B
Processors
Sound, video and game controllers
Storage volumes
System devices
Universal Serial Bus controllers
Update Driver...
Disable
Uninstall
Scan for hardware changes
Properties
Opens property sheet for the current selection.
UG1025_c2_04_090313

Figure 2-4: **Configuring the Driver**



Bits per second: 115200

Data bits: 8

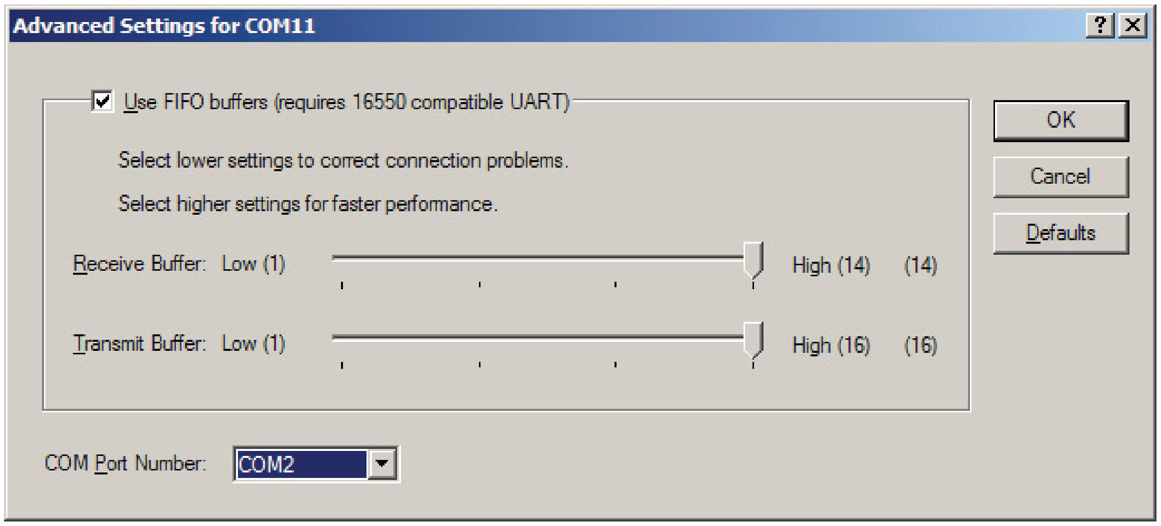
Parity: None

Stop bits: 1

Flow control: None

UG1025_c2_05_090313

Figure 2-5: **UART Port Setting Tab**



☑ Use FIFO buffers (requires 16550 compatible UART)

Select lower settings to correct connection problems.

Select higher settings for faster performance.

Receive Buffer: Low (1) High (14) (14)

Transmit Buffer: Low (1) High (16) (16)

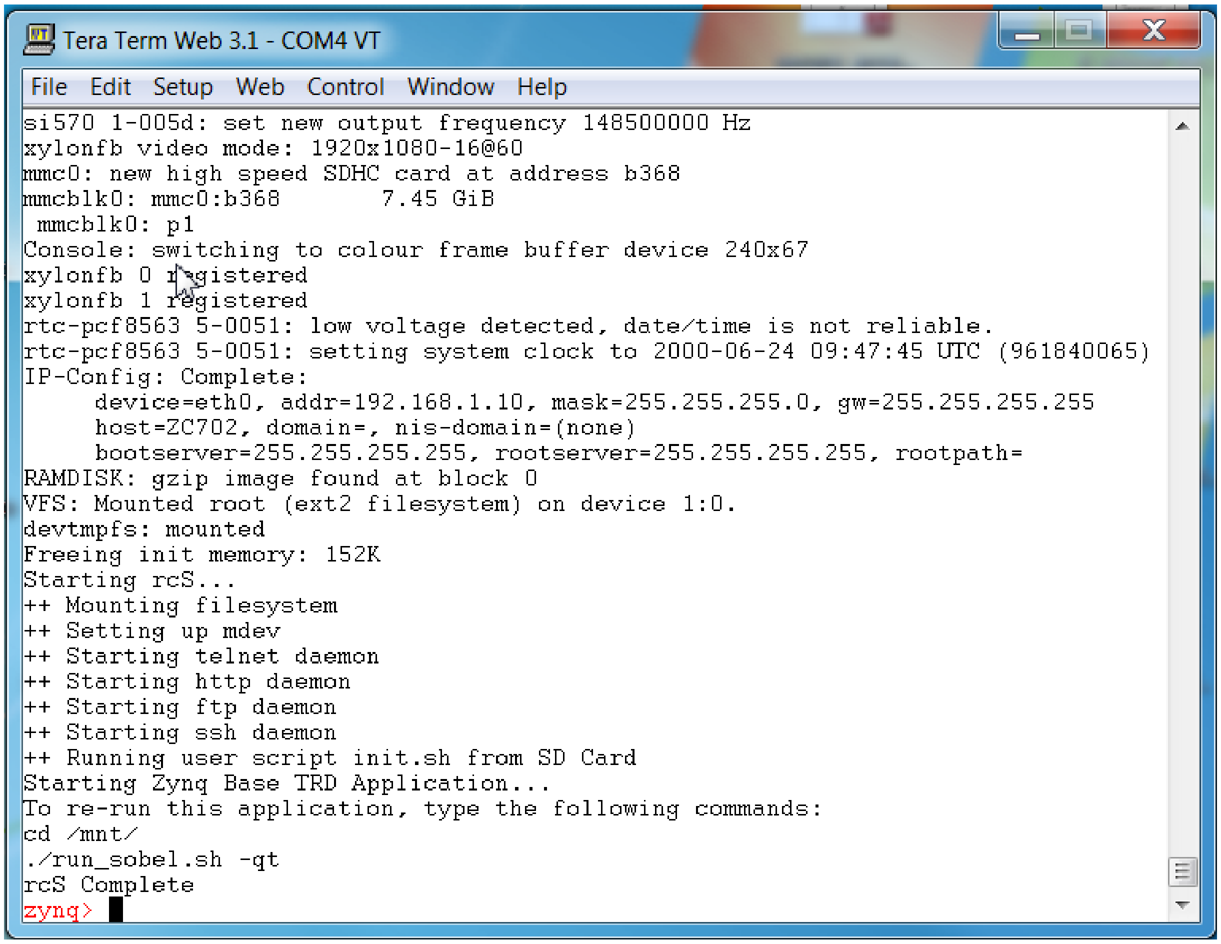
COM Port Number: COM2

Figure 2-6: **Select a COM Port (between COM1 and COM4)**

Run the Non-secure TRD Application

1. Start Tera Term or a comparable installed terminal program. Configure it to have the following settings:
   Baud = **115200**, Data = **8**, Parity = **None**, Stop = **1**, and Flow = **None**.
2. On the ZC702 Evaluation Board, set the Boot mode switch (J16) to **JTAG** mode.
3. Invoke **SDK > Xilinx Tools > Program Flash** and browse to `ug1025/zc702_linux_trd/ready_for_download/zc702_linux_trd_ns.mcs`. Enter **`0x0`** as the offset. Click **Finish**. Power down after the QSPI is programmed.

4. Power up to boot the TRD using QSPI mode. Figure 2-7 shows the boot of the TRD.


Tera Term Web 3.1 - COM4 VT
File Edit Setup Web Control Window Help

```
si570 1-005d: set new output frequency 148500000 Hz
xylonfb video mode: 1920x1080-16@60
mmc0: new high speed SDHC card at address b368
mmcblk0: mmc0:b368       7.45 GiB
 mmcblk0: p1
Console: switching to colour frame buffer device 240x67
xylonfb 0 registered
xylonfb 1 registered
rtc-pcf8563 5-0051: low voltage detected, date/time is not reliable.
rtc-pcf8563 5-0051: setting system clock to 2000-06-24 09:47:45 UTC (961840065)
IP-Config: Complete:
     device=eth0, addr=192.168.1.10, mask=255.255.255.0, gw=255.255.255.255
     host=ZC702, domain=, nis-domain=(none)
     bootserver=255.255.255.255, rootserver=255.255.255.255, rootpath=
RAMDISK: gzip image found at block 0
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing init memory: 152K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting ssh daemon
++ Running user script init.sh from SD Card
Starting Zynq Base TRD Application...
To re-run this application, type the following commands:
cd /mnt/
./run_sobel.sh -qt
rcS Complete
zynq>
```

UG1025_c4-01_082913

Figure 2-7: **zc702_linux_trd Boot**

See *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide* (UG850) [Ref 3] for more detailed information about the ZC702 board. Default factory settings of jumpers and switches on the ZC702 board are highlighted in Figure 2-1. Default switch and jumper settings are listed in Table 2-1.

For the most up to date information on the content provided with the ZC702 evaluation kit, see the Zynq-7000 All Programmable SoC ZC702 Evaluation Kit documentation website. See *Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques* (UG873) [Ref 4] for the basic hardware and software flow using the ZC702 board. For additional information, see the Zynq-7000 AP SoC documentation website.

Generating and Programming AES and RSA Keys

Introduction

This chapter provides instructions on generating and programming keys. Figure 3-1 shows the topics covered.

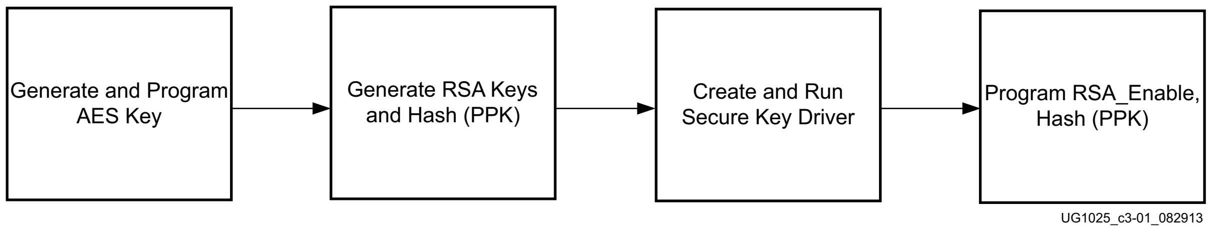

Generate and Program AES Key
Generate RSA Keys and Hash (PPK)
Create and Run Secure Key Driver
Program RSA_Enable, Hash (PPK)
UG1025_c3-01_082913

Figure 3-1: **Generating and Programming Keys**

The cryptographic keys used by the Zynq®-7000 All Programmable SoC (AP SoC) are:

- AES 256 bit key
- HMAC 256 bit key
- RSA Primary Secret Key (PSK)
- RSA Primary Public Key (PPK)
- RSA Secondary Secret Key (SSK)
- RSA Secondary Public Key (SPK)

For the recommended secure boot flow, AES and RSA keys need to be created and programmed into the Zynq AP SoC. The AES key is created using Bootgen. The AES key can be programmed into Battery Backed Random Access Memory (BBRAM) or eFUSE. BBRAM is re-programmable and zeroizable. eFUSE is one-time programmable (OTP).

In the Zynq AP SoC, the primary RSA key is used to authenticate the secondary keys. Secondary keys are used to authenticate partitions (i.e., ELF and BIT files). OpenSSL is used to create RSA primary and secondary keys. OpenSSL is used because it is readily available. Other methods of generating keys can be used. The RSA key generated by OpenSSL is a private/public key pair. The public key is a subset of the private key. For security, it is critical to protect the private key. In RSA, the private key is used to sign the partitions at the manufacturing site, and the public key is programmed into the Zynq AP SoC embedded device to verify the signature. The fielded embedded device is secure because it does not contain the private RSA key.

For this getting started guide, the AES key is programmed into the device using iMPACT. RSA security is programmed into the device using the Xilinx Secure Key Driver. The Secure Key Driver can program the AES key.

Generate and Program the AES Key

The AES key is generated with Bootgen using these steps:

1. Create a `generate_aeskey.bif` file with the following content:

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

2. Use the following Bootgen command to generate an AES key.

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

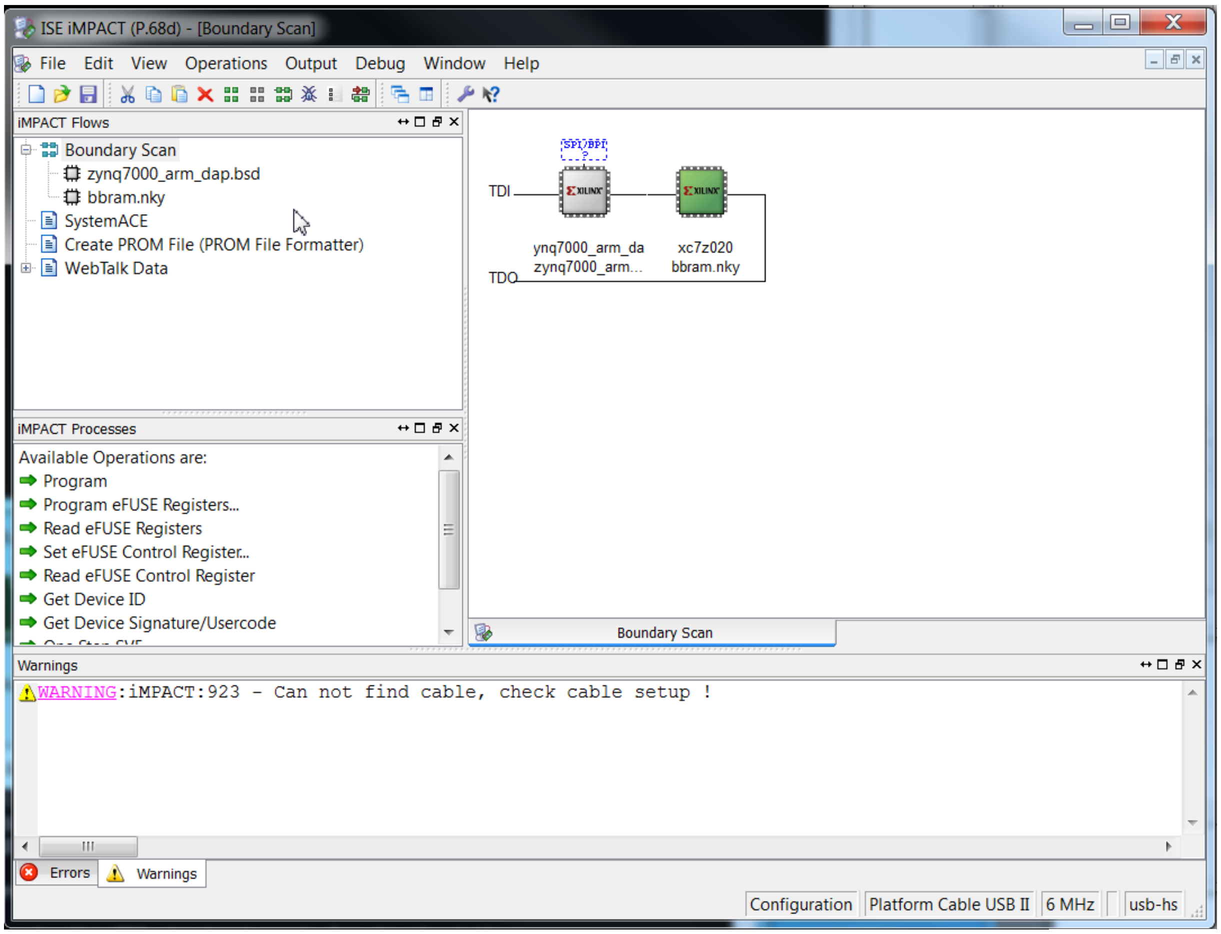
   If the specified AES key does not exist, Bootgen generates the key using the name in the `generate_aeskey.bif` file (`bbram.nky` in this case).

3. To program the AES key, invoke iMPACT. If using Linux, run **impact &**. If using Windows, select from the Program Start **ISE Design Suite 14.6 > ISE Design Tools > 64-Bit Tools > iMPACT**. Click **No** when prompted to save the project file. Click **Cancel** when the New iMPACT Project dialog box is displayed.

4. Double-click **Boundary Scan** in the iMPACT Flows pane. Right-click and select **Initialize Chain**. If only one of the ARM® core of xc7z020 FPGA is displayed on the JTAG chain, change the **Boot Mode selection** switch to **JTAG** and re-initialize the chain. Click **Yes** when asked to assign a configuration file.

5. With the DAP displayed in green, click **Bypass**. With the xc7z020 selected, enter the AES key generated in step 1 in the Assign New Configuration File dialog box. Click **Cancel** when the Device Programming Properties is displayed.

Figure 3-2 shows the JTAG Chain in iMPACT.

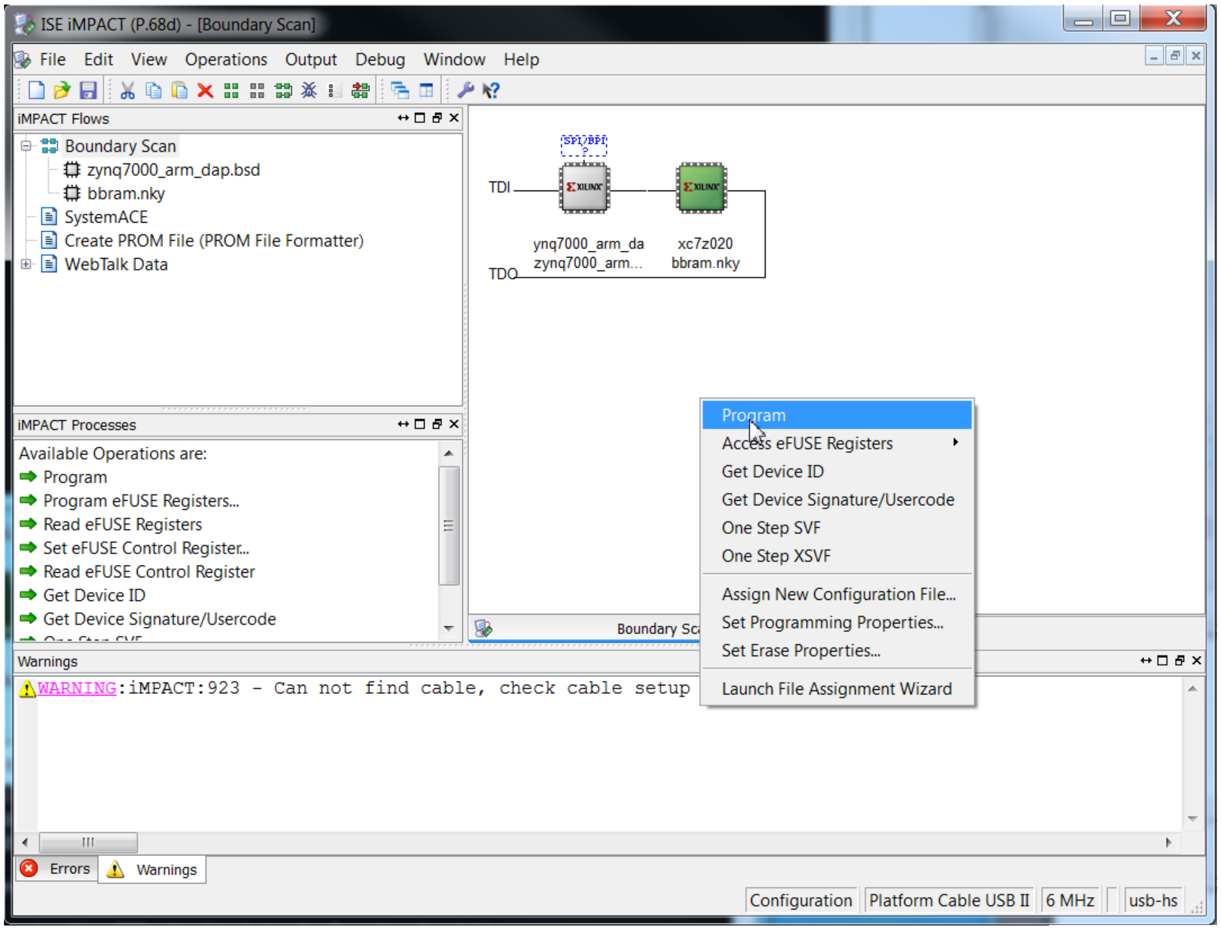


```
WARNING:iMPACT:923 - Can not find cable, check cable setup !
```

Figure 3-2: **iMPACT - Detecting the Concatenated Chain**

6. As shown in Figure 3-3, use iMPACT to program the BBRAM key. Right-click on **ZC7Z020** and click **Program**. iMPACT can also program the eFUSE key. Both eFUSE and BBRAM keys can be programmed into the Zynq AP SoC. The bootgen -encrypt efuse | bbram argument specifies which key is used. Bootgen writes the key source to the Boot Header region of the image. At power-up, the BootROM code reads the Boot Header to determine which key source to use.

The eFUSE control is also programmable using Available Operations in iMPACT, located in the iMPACT Processes pane. The eFUSE control register is used for functions such as restricting key swapping and permanently disabling the JTAG port.


ISE iMPACT (P.68d) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
zynq7000_arm_dap.bsd
bbram.nky
SystemACE
Create PROM File (PROM File Formatter)
WebTalk Data
iMPACT Processes
Available Operations are:
Program
Program eFUSE Registers...
Read eFUSE Registers
Set eFUSE Control Register...
Read eFUSE Control Register
Get Device ID
Get Device Signature/Usercode
TDI
TDO
SPI/BPI
ynq7000_arm_da
zynq7000_arm...
xc7z020
bbram.nky
Program
Access eFUSE Registers
Get Device ID
Get Device Signature/Usercode
One Step SVF
One Step XSVF
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Boundary Sc
Warnings
Errors
Warnings
Configuration
Platform Cable USB II
6 MHz
usb-hs
UG1025_c3-03_082913

```
WARNING:iMPACT:923 - Can not find cable, check cable setup
```

Figure 3-3: **Using iMPACT to Program the BBRAM Key**

Generating RSA Keys

In this section, primary and secondary RSA keys are generated. Using these keys, the hash of the Primary Public Key is generated.

1. In this document, OpenSSL is used to generate RSA keys. (There are other methods of generating RSA keys.) The primary and secondary secret RSA keys are generated using the following OpenSSL command:

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

   In RSA, the public key is contained in the private key. The OpenSSL command to extract the public key from the private key is:

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

2. Generate the hash of the Primary Public Key. After generating the RSA keys using openssl, use Bootgen to generate the hash of the PPK. Create a `gen_hash_ppk.bif` with following content:

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

3. Run

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

The `hash_ppk.txt` file contains the hash of the PPK.

Create the Secure Key Driver to Enable RSA Functionality

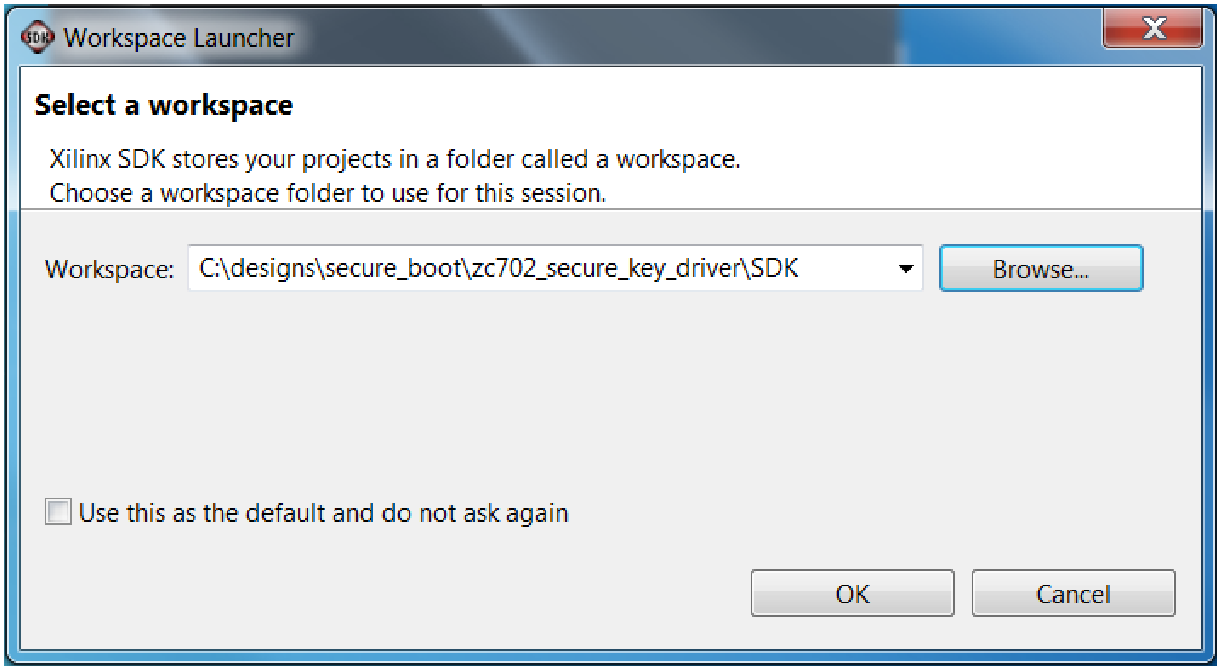
To use RSA, the RSA Enable control bit and the hash of the PPK must be programmed in the PS eFUSE array. To do this, a Secure Key Driver software project is created with Software Development Kit (SDK). The Secure Key Driver software project is in the ug1025/zc702_secure_driver directory. The functionality of the Secure Key Driver is controlled by editing the `xilskey_efuse_example.c` and `xilskey_input.h` files.

To enable RSA and program the hash of the PPK, edit `xilskey_input.h` file to program the PS eFUSEs, PL eFUSEs, or both. The eFUSE is OTP, so if it is already programmed, the `xilskey_efuse_example.c` file can be edited to only read the hash of the PPK.

Edit the `xilskey_input.h` file as follows:

1. Define XSK_EFUSEPS_DRIVER.
2. Disable XSK_EFSUSEPL_DRIVER using a comment.
3. Copy the PPK hash from `hash_ppk.txt` to XSK_EFUSEPS_RSA_KEY_HASH_VALUE.
4. Set XSK_EFUSEPS_ENABLE_RSA_KEY_HASH TRUE.
5. Set XSK_EFUSEPS_ENABLE_RSA_AUTH TRUE.

Compile the Secure Key Driver using SDK. If using Linux, invoke SDK by running **xsdk &**. If using Windows, double-click on **Xilinx SDK 2013.2** or use the **Program Start > ISE Design Suite 14.6 > EDK > Xilinx Software Development Kit**. Select a workspace as shown in Figure 3-4.



Workspace Launcher

Select a workspace

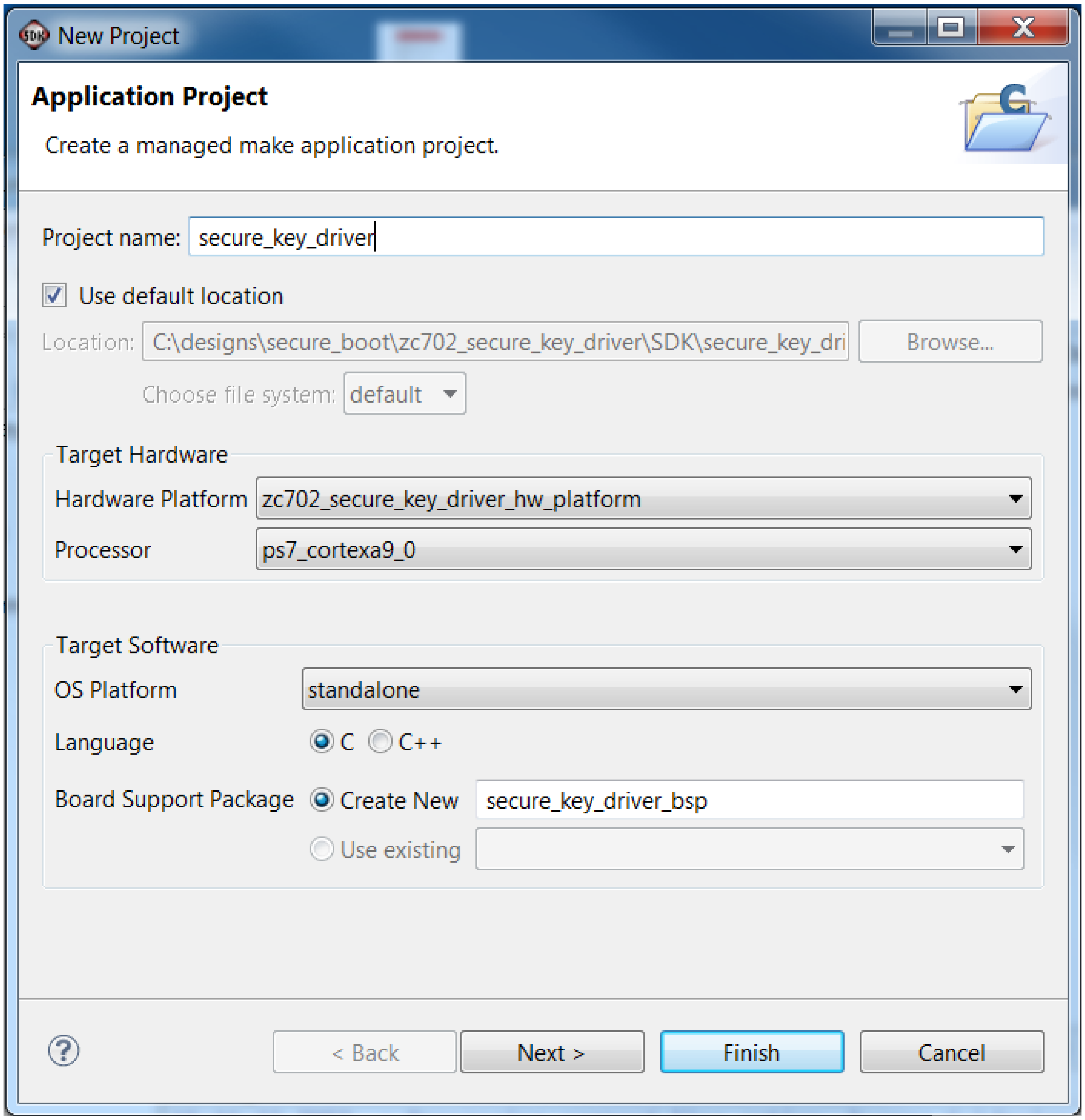
Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\designs\secure_boot\zc702_secure_key_driver\SDK

☐ Use this as the default and do not ask again

Figure 3-4: **Launching the Secure Key Driver Workspace**

6. Create a new application project as shown in Figure 3-5. Define the Project Name as **secure_key_driver**. Leave other options at their default values. Click **Next**.



Application Project

Create a managed make application project.

Project name: secure_key_driver

[✓] Use default location

Location: C:\designs\secure_boot\zc702_secure_key_driver\SDK\secure_key_dri

Choose file system: default

Target Hardware

Hardware Platform: zc702_secure_key_driver_hw_platform

Processor: ps7_cortexa9_0

Target Software

OS Platform: standalone

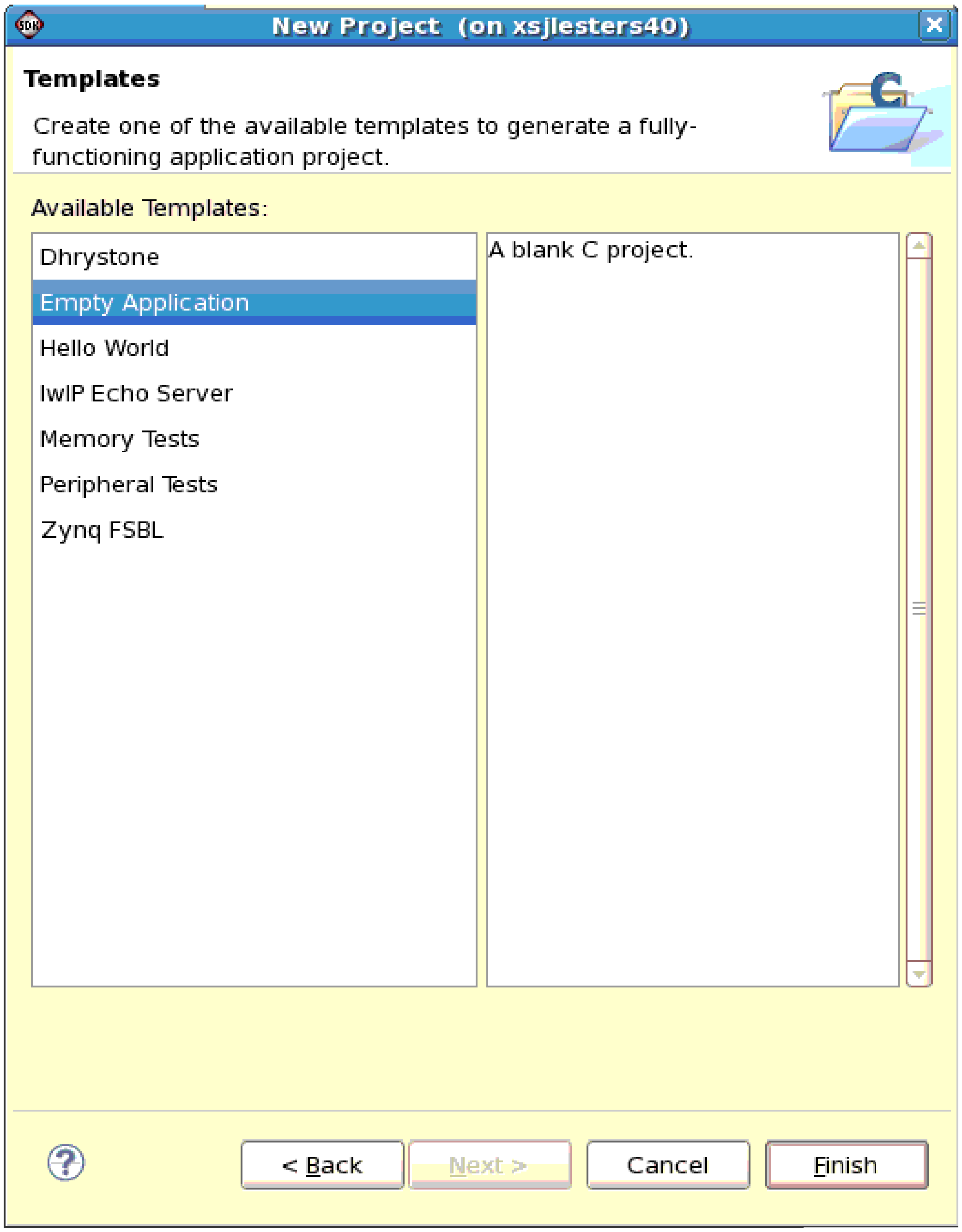
Language: [● C] ○ C++

Board Support Package: [● Create New] secure_key_driver_bsp

○ Use existing

Figure 3-5: **Create the Secure Key Driver Project**

7. As shown in Figure 3-6, select **Empty Application**, and when the dialog box is displayed, name the project **secure_key_driver**. Click **Finish**.


New Project (on xsjlesters40)

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- Peripheral Tests
- Zynq FSBL

A blank C project.

Figure 3-6: **Create Empty Application**

8. As shown in Figure 3-7, right-click the **secure_key_driver_bsp** board support package and click **Board Support Package Settings.**

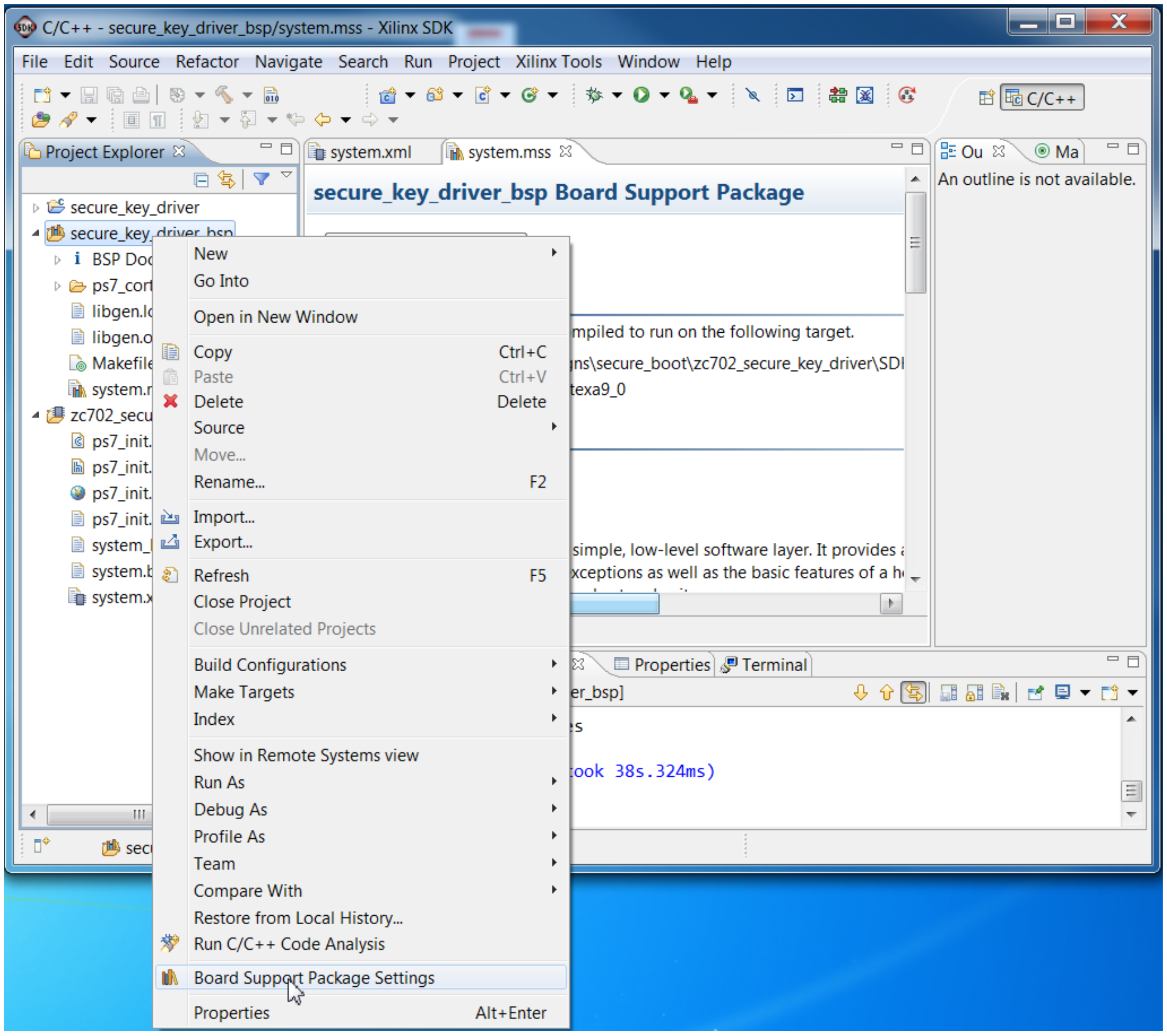
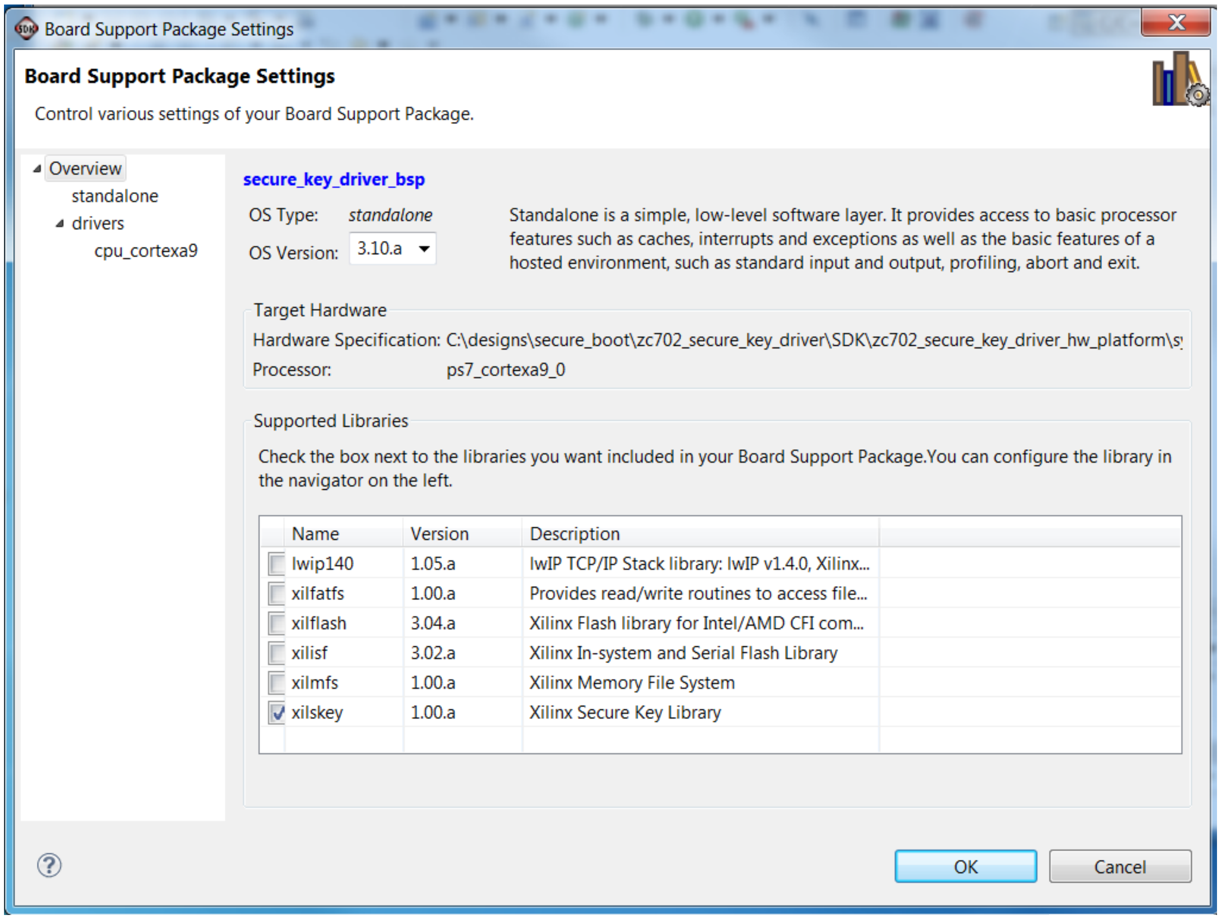


Figure 3-7: **Board Support Package Settings**

9. As shown in Figure 3-8, select the **xilskey library** and rebuild the Board Support Package (BSP). Click **OK**.


Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

secure_key_driver_bsp

OS Type: *standalone*

OS Version: 3.10.a

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Target Hardware

Hardware Specification: C:\designs\secure_boot\zc702_secure_key_driver\SDK\zc702_secure_key_driver_hw_platform\s

Processor: ps7_cortexa9_0

Supported Libraries

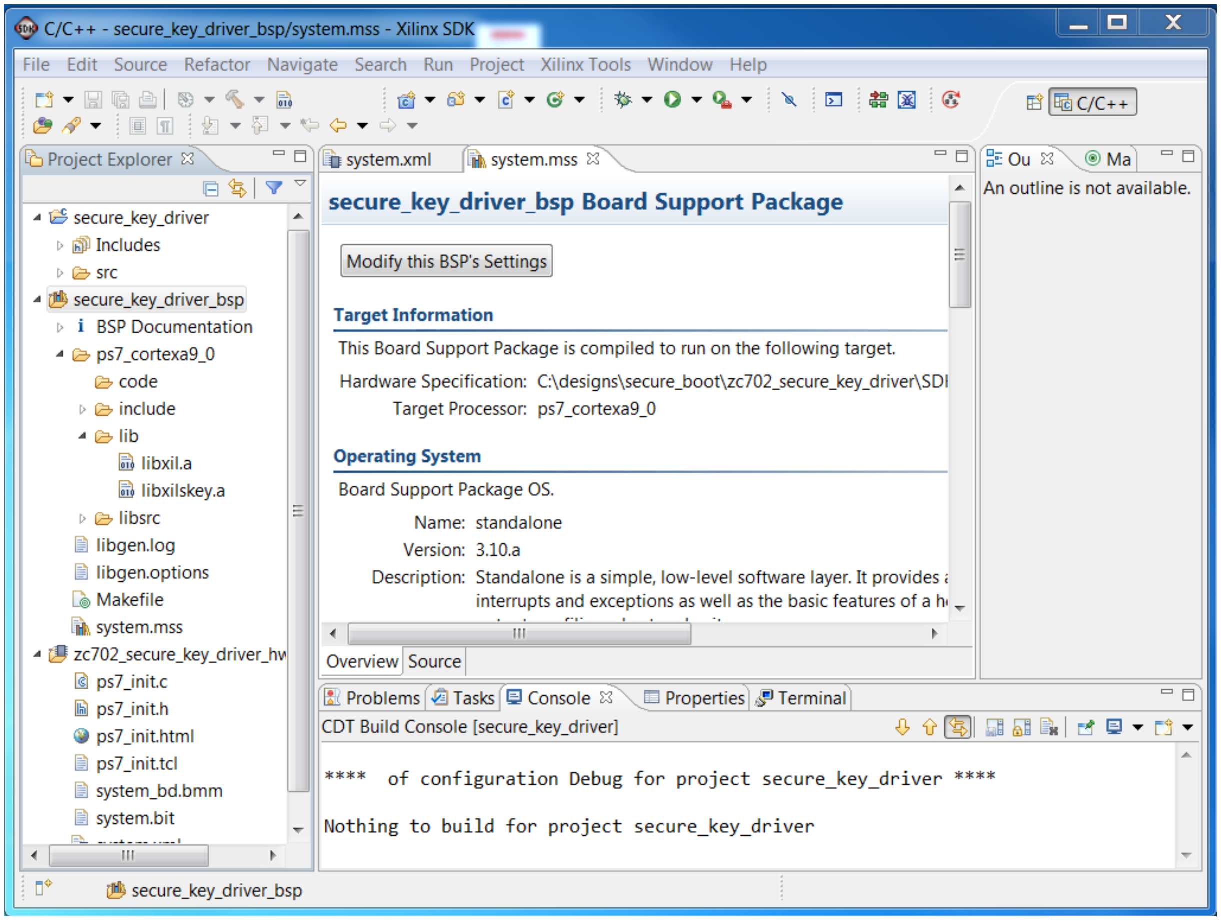
Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.

	Name	Version	Description
☐	lwip140	1.05.a	lwIP TCP/IP Stack library: lwIP v1.4.0, Xilinx...
☐	xilfatfs	1.00.a	Provides read/write routines to access file...
☐	xilflash	3.04.a	Xilinx Flash library for Intel/AMD CFI com...
☐	xilisf	3.02.a	Xilinx In-system and Serial Flash Library
☐	xilmfs	1.00.a	Xilinx Memory File System
✓	xilskey	1.00.a	Xilinx Secure Key Library

UG1025_c3-08_082913

Figure 3-8: **Selecting the Secure Key Library**

10. The xilskey library is compiled and displayed in the Project Explorer pane under **secure_key_driver_bsp/ps7_cortexa9_0/lib**. This is shown in Figure 3-9.



secure_key_driver_bsp Board Support Package

Modify this BSP's Settings

Target Information

This Board Support Package is compiled to run on the following target.

Hardware Specification: C:\designs\secure_boot\zc702_secure_key_driver\SDI

Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone

Version: 3.10.a

Description: Standalone is a simple, low-level software layer. It provides

interrupts and exceptions as well as the basic features of a h

An outline is not available.

CDT Build Console [secure_key_driver]

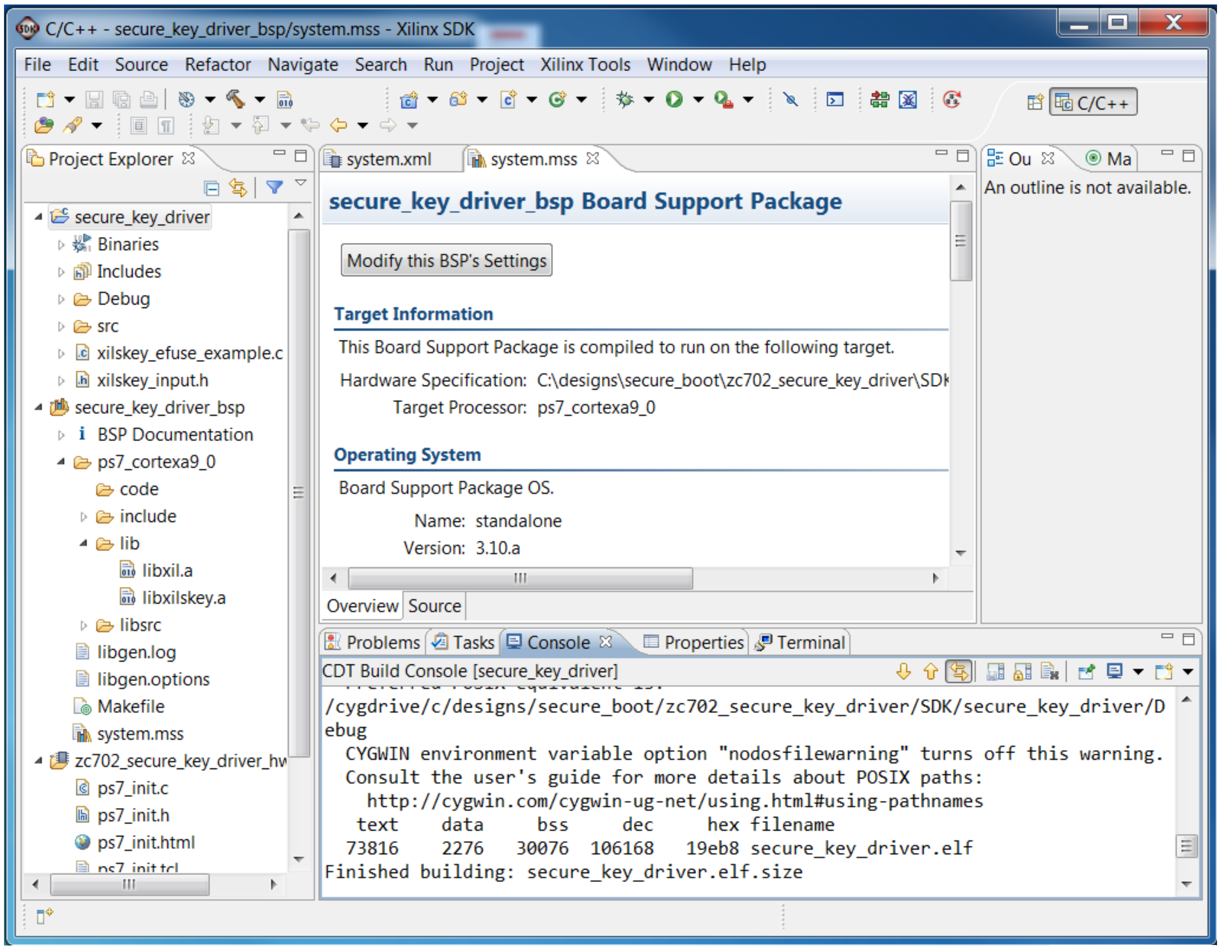
```
****  of configuration Debug for project secure_key_driver ****

Nothing to build for project secure_key_driver
```

UG1025_c3-09_082913

Figure 3-9: **Compiled xilskey Library**

11. With the **secure_key_driver** selected, select **File > Import > General Files**, and import **`xilskey_efuse_example.c`** and **`xilskey_input.h`** into the application project. Select the **secure_key_driver** project, and run **Project > Build Project**. Figure 3-10 shows a compiled secure_key_driver project.



secure_key_driver_bsp Board Support Package

Target Information

This Board Support Package is compiled to run on the following target.

Hardware Specification: C:\designs\secure_boot\zc702_secure_key_driver\SDK

Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone

Version: 3.10.a

CDT Build Console [secure_key_driver]

```
/cygdrive/c/designs/secure_boot/zc702_secure_key_driver/SDK/secure_key_driver/D
ebug
  CYGWIN environment variable option "nodosfilewarning" turns off this warning.
  Consult the user's guide for more details about POSIX paths:
    http://cygwin.com/cygwin-ug-net/using.html#using-pathnames
   text     data     bss     dec     hex filename
  73816     2276   30076  106168   19eb8 secure_key_driver.elf
Finished building: secure_key_driver.elf.size
```

Figure 3-10: **Compiled Secure Key Driver Project**

Run the Secure Key Driver to Enable RSA Functionality

The Secure Key Driver can be run using any boot mode. The simplest is JTAG, using the following XMD commands. Invoke XMD by entering **xmd** at the command prompt or in SDK enter **Xilinx Tools > XMD Console**.

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

```
stop
dow secure_key_driver.elf
con
stop
```

If secure_key_driver is compiled with read enabled, open a communication terminal to view the value of the hash of the PPK.

Creating Bootgen Image Format Files

Introduction

Bootgen creates a single file image which is used to boot the Zynq®-7000 All Programmable SoC (AP SoC). The input into Bootgen is a Bootgen Image Format (BIF) file which lists the partitions that are to be included in the image. Partitions are the software, data, and bitstream files which comprise the image. For example, the partitions in the TRD are `fsbl.elf`, `system.bit`, `u-boot.elf`, `uImage.bin`, `uramdisk.image.gz`, `devicetree.dtb`, and `sobel_cmd.elf`.

In the BIF, attributes are used to specify if the partition is to be encrypted [encryption] and/or authenticated [authentication]. Attributes also are used to specify the offset address [offset] in nonvolatile memory (NVM) and the load address [load] in the double-date-rate (DDR) memory.

Bootgen outputs the image file in MCS or BIN format. The Intel MCS format is used if the Zynq Flash Programmer is used to write Quad Serial Peripheral Interface (QSPI). The BIN format is used if U-Boot is used to program QSPI or if the SD boot mode is used.

In 14.6, RSA and the ability to specify encryption/authentication on a partition basis are only supported when Bootgen is used at the command line. The 14.6 Bootgen GUI has limited support for security features. In 14.6, the Bootgen command line is only supported on Linux.

Booting the TRD Securely

The following is a BIF which is used to boot the TRD securely.

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
```

In this BIF, all partitions are authenticated so that software is loaded using a chain of trust. The FSBL is encrypted. Sensitive partitions, including the `system.bit` and `sobel_cmd.elf`, are encrypted. The open source partitions, U-Boot and Linux, are not encrypted.

With the bitstream partition included, the image created using this BIF does not fit into the ZC702 QSPI, which is 16 MB. The image fits if the 4.4 MB `system.bit` is removed from the BIF.

The cryptographic keys specified by the [aeskeyfile], [pskfile], and [sskfile] attributes are generated in Chapter 3.

This BIF uses the [offset] and [load] attributes so that the FSBL loads the partitions into the specified DDR address. In many systems, U-Boot is used to load partitions. As of the 14.6 release, U-Boot does not contain the RSA libraries used to authenticate partitions, so the FSBL is used to authenticate/load the partitions. To support the flow in which all partitions are authenticated by the FSBL, the U-Boot `zynq_common.h` has been edited.

Note: Authentication in U-Boot is scheduled for the 14.7 release.

All partitions should be authenticated so that the device boots with a chain of trust. This means that each partition is authenticated before it is loaded. In most cases, all partitions are not encrypted. Encryption provides confidentiality, so partitions with sensitive information should be encrypted. It is not generally constructive to encrypt partitions like Linux and U-Boot which are open source. Encryption can increase boot time, but in the NVM configurations tested to date, secure and non-secure boot time have been equivalent.

Although AES is not susceptible to the known plaintext attack, encryption should not be used when it is not needed, as other attacks might be attempted. Use encryption to protect sensitive information.

Run the following steps to boot the TRD securely using QSPI mode.

1. Set up the ZC702 Evaluation Board as defined in Chapter 2.
2. Set up Tera Term as defined in Chapter 2.
3. Change to the UG1025/zc702_secure_key_driver/SDK directory. In Chapter 3, the `secure_key_driver.elf` software project is created. To program the PS RSA_Enable and hash of the PPK eFUSEs, run the following **xmd** commands:

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

4. Change to the ug1025/zc702_linux_trd directory. To learn about the BIF functionality in this project, open `zc702_linux_trd.bif` and review the attributes. To create the TRD image, run:

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

   Note: In the SDK 14.6 release, command-line Bootgen must be run on Linux. Windows support is scheduled for SDK 14.7.

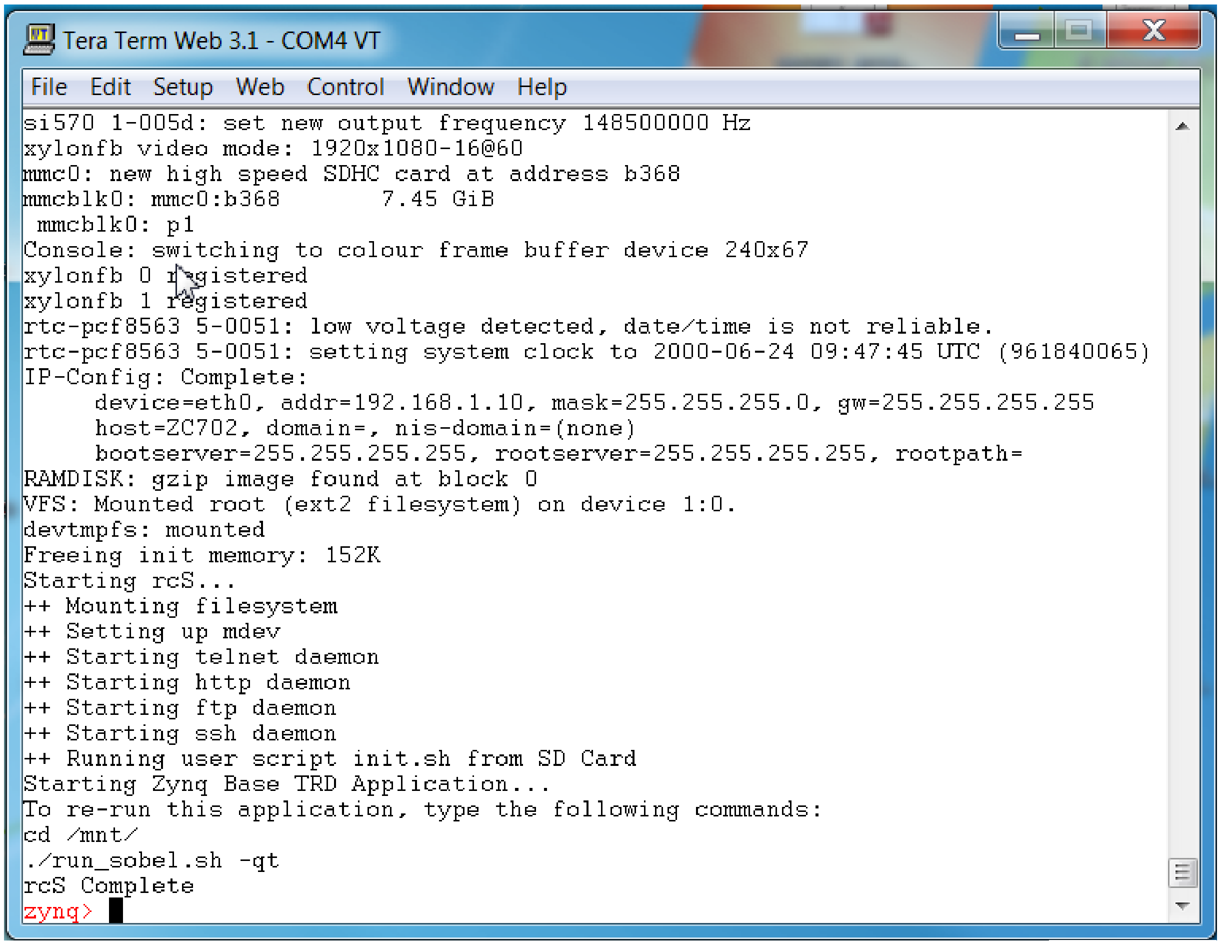
5. If using Linux, invoke SDK using **xsdk &**. Select the workspace as ug1025/zc702_linux_trd/SDK.
6. In SDK, enter **Xilinx Tools > Program Flash**. Power down.
7. Specify the image by browsing to:

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

   As noted, this MCS image does not contain the TRD bitstream file. To boot the TRD, generate a BIN image which includes the bitstream, and boot using SD mode.

8. On the ZC702 Evaluation Board, change the boot mode switch to QSPI boot mode by moving J25 to 1, or moving switch 4 to 1 if the evaluation board uses SW16.

9. Power cycle. Verify that the Zynq AP SoC boots to the Petalinux prompt. To login, use **root** for the user name and **root** for the password. Figure 4-1 shows the expected TRD output.


Tera Term Web 3.1 - COM4 VT
File Edit Setup Web Control Window Help

```
si570 1-005d: set new output frequency 148500000 Hz
xylonfb video mode: 1920x1080-16@60
mmc0: new high speed SDHC card at address b368
mmcblk0: mmc0:b368       7.45 GiB
 mmcblk0: p1
Console: switching to colour frame buffer device 240x67
xylonfb 0 registered
xylonfb 1 registered
rtc-pcf8563 5-0051: low voltage detected, date/time is not reliable.
rtc-pcf8563 5-0051: setting system clock to 2000-06-24 09:47:45 UTC (961840065)
IP-Config: Complete:
     device=eth0, addr=192.168.1.10, mask=255.255.255.0, gw=255.255.255.255
     host=ZC702, domain=, nis-domain=(none)
     bootserver=255.255.255.255, rootserver=255.255.255.255, rootpath=
RAMDISK: gzip image found at block 0
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing init memory: 152K
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting ssh daemon
++ Running user script init.sh from SD Card
Starting Zynq Base TRD Application...
To re-run this application, type the following commands:
cd /mnt/
./run_sobel.sh -qt
rcS Complete
zynq>
```

UG1025_c4-01_082913

Figure 4-1: **Expected TRD Output**

10. To compare non-secure and secure boot time, repeat step 7-step 9, programming QSPI with `zc702_linux_trd_ns.mcs` for the non-secure boot. Measure the time from power on to when the "zynq>" prompt is displayed in the communication terminal.

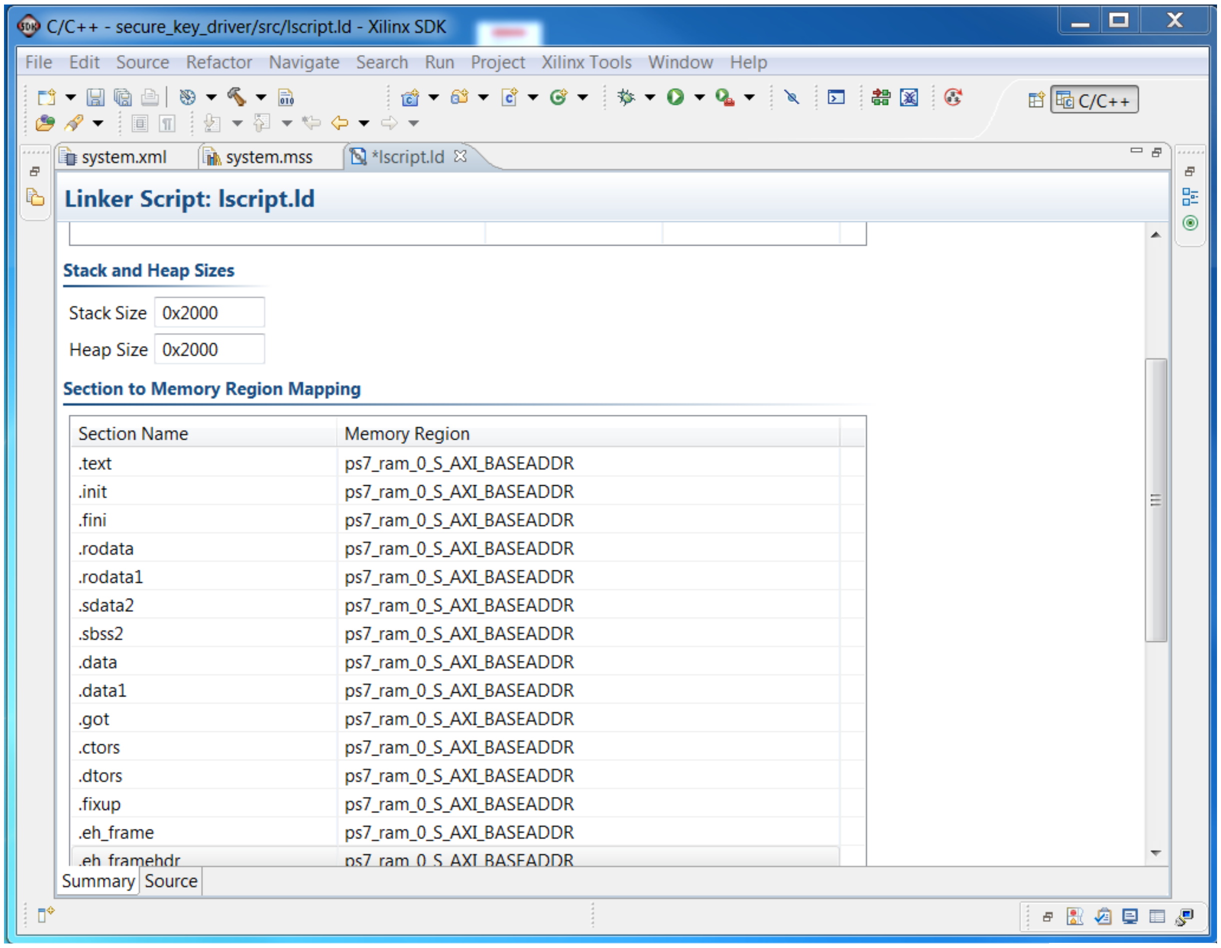
Boot time varies based on factors such as the type and configuration of the NVM, security, and size of the image. If a boot time estimate is needed, contact a Xilinx Field Application Engineer (FAE). Xilinx FAEs use an internal Boot Time Estimator for various configurations.

There is a difference in a plain non-secure boot and a non-secure boot with RSA Enable programmed. When RSA Enable is programmed, the [pskfile] and [sskfile] attributes must be included in the BIF and the FSBL partition must be authenticated.

Loading Sensitive Partitions in Secure Storage

A second method of protecting sensitive information is to locate sensitive information in secure storage (on-chip memory [OCM], AXI BRAM), within the security perimeter of the Zynq-7000 AP SoC. This is done with the linker script or the Bootgen [load] attribute.

To locate a partition in the OCM rather than DDR, edit the linker script as shown in Figure 4-2.


C/C++ - secure_key_driver/src/lscript.ld - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help
C/C++
system.xml
system.mss
*lscript.ld
Summary
Source
UG1025_c3-11_082913

Linker Script: lscript.ld

Stack and Heap Sizes

Stack Size	0x2000
Heap Size	0x2000

Section to Memory Region Mapping

Section Name	Memory Region
.text	ps7_ram_0_S_AXI_BASEADDR
.init	ps7_ram_0_S_AXI_BASEADDR
.fini	ps7_ram_0_S_AXI_BASEADDR
.rodata	ps7_ram_0_S_AXI_BASEADDR
.rodata1	ps7_ram_0_S_AXI_BASEADDR
.sdata2	ps7_ram_0_S_AXI_BASEADDR
.sbss2	ps7_ram_0_S_AXI_BASEADDR
.data	ps7_ram_0_S_AXI_BASEADDR
.data1	ps7_ram_0_S_AXI_BASEADDR
.got	ps7_ram_0_S_AXI_BASEADDR
.ctors	ps7_ram_0_S_AXI_BASEADDR
.dtors	ps7_ram_0_S_AXI_BASEADDR
.fixup	ps7_ram_0_S_AXI_BASEADDR
.eh_frame	ps7_ram_0_S_AXI_BASEADDR
.eh_framehdr	ps7_ram_0_S_AXI_BASEADDR

Figure 4-2: **Locating the Secure Key Driver in OCM**

Because the FSBL runs from OCM, partitions are generally not loaded into the OCM with the FSBL. If JTAG mode is used, the device can be initialized with ps7_init, and the ELF can be loaded into OCM using an XMD command. For other boot modes, load U-Boot with the FSBL and use U-Boot to load the sensitive partition into OCM.

Because the OCM is fast secure storage, it is a good location to store sensitive software and data partitions. Data partitions are not compiled software partitions, so there is no linker script. To locate data partitions in OCM, use the [load] attribute in the BIF file.

Advanced Key Management Options

Introduction

Bootgen is used at the manufacturing site. Bootgen operates in Debug and Release modes. This chapter shows how to run Bootgen in the Release mode. In the Release mode, Bootgen uses the RSA keys differently than in the Debug mode. In Bootgen Debug mode, private keys are used. In Bootgen Release mode, the final BIF contains public keys and signatures of the partitions.

In current cryptography, algorithms are known, and it is critical to protect the private key. Bootgen Release mode helps protect the private key. This protection of the private key occurs at the manufacturing site. Independent of whether Debug or Release mode is used, the fielded embedded device contains the same public key and content.

One approach to protecting the private key is to use a security staff, or Infosec group, for the final release of the embedded product. This reduces the number of employees who have access to the cryptographic key. Often, a Hardware Security Module (HSM) is used in a secure area. The HSM generates the RSA private/public key pair and signatures. Security is also enhanced because the private key never leaves the HSM.

In the Bootgen Release mode, Bootgen and a second Xilinx tool, **xil_rsa_sign**, perform the same function as the HSM in generating hashes and signatures. This chapter shows how to use Bootgen and **xil_rsa_sign** to provide additional protection of the private key.

To compile **xil_rsa_sign**, change to the ug1025/zc702_secure_key/xil_rsa_sign_src directory. If using Linux, copy `makefile_linux` to Makefile. If using Windows, copy `makefile_xp` to Makefile. Run **make**.

The **make** fails in Windows if the `gcc.exe` in the Makefile is incorrect. There are many `gcc.exe` files in the $XILINX_EDK software. If using Windows, edit the Makefile to use the full path to `$XILINX/gnu/MinGW/5.0.0/nt/bin/gcc.exe` or the `cygwin gcc.exe`:

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

or

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

The steps to use Bootgen Release mode are as follows. The system is in the ug1025/zc702_secure_key directory.

1. Create the SPK hash using the following command and BIF.

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

   The `spk.bif` is

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

   Bootgen generates the `spk.pk1.sha256` SPK hash file.

2. Use **xil_rsa_sign** to generate the signature.

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

3. Generate the partition hashes using the following Bootgen command:

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

   where the BIF is

   ```
   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
   }
   ```

   The hashes for the partitions are generated. The hash files are `fsbl.elf.0.sha256`, `system.bit.0.sha256`, and `u-boot.elf.0.sha256`.

4. Generate the signatures of the hashes just created:

   ```
   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. Create the image using the following Bootgen command:

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

   The BIF is

   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. Verify that Bootgen Debug mode and Bootgen Release mode generate identical image files. To do this, create the image using Bootgen Debug mode, and then diff the two images. Run Bootgen Debug mode using the following bootgen command:

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

   where `bootimage_dm.bif` is:

   ```
   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
   }
   ```

   Run:

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

   to verify that the files are identical.

Chapter Summary

- RSA security is based on the fielded device using a public (not private) key, and that the key can be changed as often as desired.
- Bootgen Release Mode is an option which protects the private key against an insider attack at the manufacturing site.

Multiboot

Introduction

Multiboot is used to ensure that the Zynq®-7000 All Programmable SoC (AP SoC) boots to a known good state in the event of an error booting the original image. This chapter shows a methodology for using multiboot.

The multiboot method uses three images. One image includes the FSBL partition only, located at 0x0. The standard or update image is located at 0x400000. The golden image is located at 0xA00000.

A key element in a multiboot example is defining the error which causes a failure. In this example, an error is induced in the `system.bit` file.

The following steps are used in multiboot.

1. Create the fsbl, update, and golden images.

   ```
   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. Copy the `update.bin` image to preserve an original version of the update image.

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

3. Use a hex editor such as gvim or hd to create the error in the update image.

   ```
   gvim update.bin
   ```

4. Corrupt `update.bin`. The bitstream starts at 0x194C0. Change a character in line 19570. Save `update.bin`.
5. Verify that `update.bin` has been corrupted using diff.

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

6. Open **Tera Term**. Click **File > Log**. Enter **`multiboot.log`** in the dialog box.
7. From ug1025/zc702_multiboot/ready_for_download, copy `BOOT.bin`, `fsbl.bin`, `update.bin`, and `golden.bin` to a SD card. Set the Boot mode on the ZC702 Evaluation Board to SD. Power cycle.
8. Enter the following commands at the U-Boot prompt:

```
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. Power down. Change from SD to QSPI boot mode. Power up.
10. Read `multiboot.log` to verify multiboot operation.

In the `multiboot.log`, verify that the error causing the failure to load `update.bin` is the correct error, and that the `golden.bin` image is loaded successfully.

Loading a Data Partition

Introduction

In addition to ELF and BIT partitions, Bootgen can load data partitions. Typical data partition applications are Digital Signal Processing (DSP) coefficients, health records, and financial records. As with ELF, BIN, and BIT partition types, Bootgen attributes allow data partitions to be encrypted and/or authenticated.

An BIF which loads a data partition is:

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

The example data file, `coefficients.bin`, contains `0101010111001100`.

To verify that the data file is loaded into OCM, run the following XMD command:

```
mrd 0xFFFFC000 8
```

In general, the OCM is a good location for a data partition. The aforementioned BIF works, but because the FSBL runs from OCM, the FSBL should not be used to load partitions into OCM. A better solution is to load the data partition into OCM using U-Boot.

Using the User Defined Field in an Authentication Certificate

Introduction

When RSA is used, an Authentication Certificate (AC) is generated for each partition with the [authentication] attribute. The AC contains keys, signatures, and a user defined field (UDF) which can be used as to indicate the version of the software being loaded. The UDF can also be used to identify the supplier of the partition, provide timestamp information, or any user defined function.

The UDF is defined in a hex file. The size of the hex file is up to 56 bytes. You need to add code to the FSBL or U-Boot to use the information in the UDF.

Use the following steps to create a UDF in an AC. The completed project is in ug1025/zc702_udf.

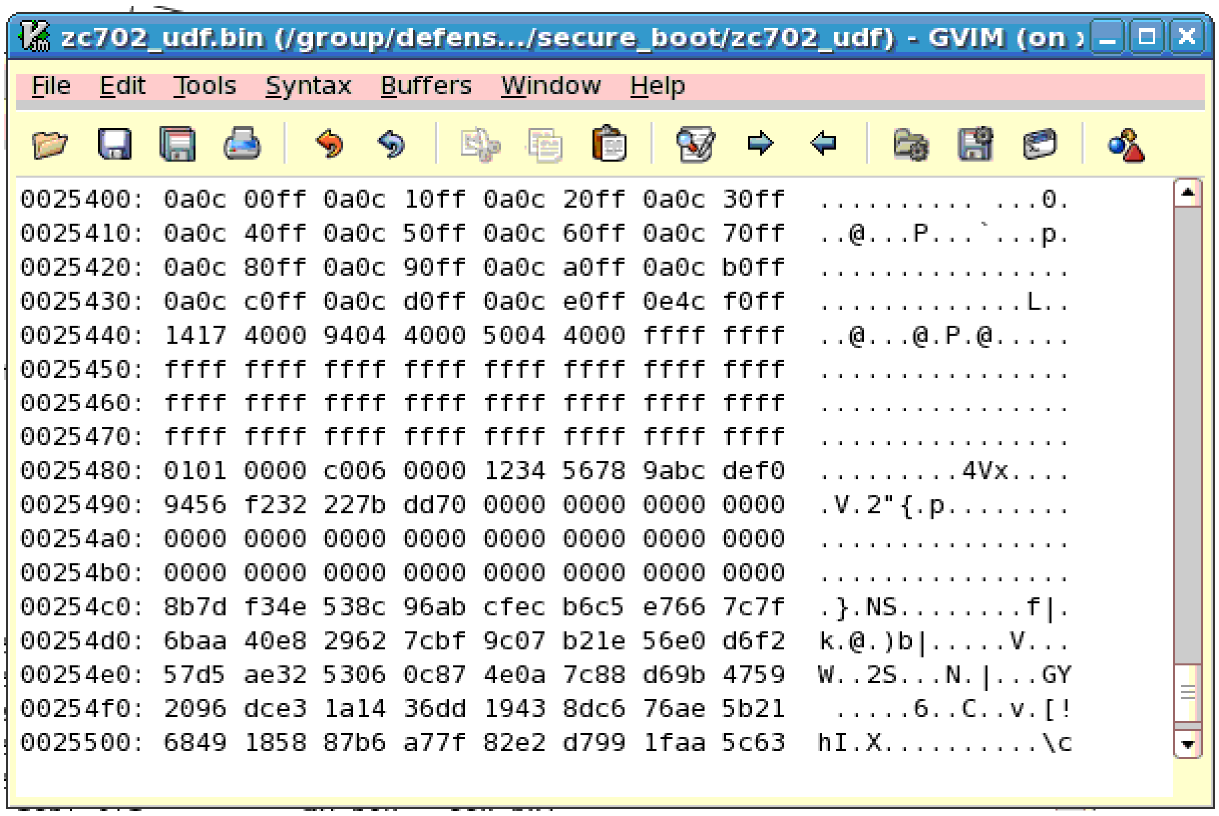
1. Create a `uboot_v10.hex` with hex values. For example, use "12345678abcdef" as the hex values.
2. Generate a BIF as follows with the udf_data attribute:

```
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. Run the following Bootgen command:

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

4. Use a hex editor as gvim or hd to locate the UDF. On line 0025480, Figure 8-1 shows the UDF in the completed zc702_udf design.


zc702_udf.bin (/group/defens.../secure_boot/zc702_udf) - GVIM (on
File Edit Tools Syntax Buffers Window Help

```
0025400: 0a0c 00ff 0a0c 10ff 0a0c 20ff 0a0c 30ff  .......... ...0.
0025410: 0a0c 40ff 0a0c 50ff 0a0c 60ff 0a0c 70ff  ..@...P...`...p.
0025420: 0a0c 80ff 0a0c 90ff 0a0c a0ff 0a0c b0ff  ................
0025430: 0a0c c0ff 0a0c d0ff 0a0c e0ff 0e4c f0ff  .............L..
0025440: 1417 4000 9404 4000 5004 4000 ffff ffff  ..@...@.P.@.....
0025450: ffff ffff ffff ffff ffff ffff ffff ffff  ................
0025460: ffff ffff ffff ffff ffff ffff ffff ffff  ................
0025470: ffff ffff ffff ffff ffff ffff ffff ffff  ................
0025480: 0101 0000 c006 0000 1234 5678 9abc def0  .........4Vx....
0025490: 9456 f232 227b dd70 0000 0000 0000 0000  .V.2"{.p........
00254a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00254b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00254c0: 8b7d f34e 538c 96ab cfec b6c5 e766 7c7f  .}.NS........f|.
00254d0: 6baa 40e8 2962 7cbf 9c07 b21e 56e0 d6f2  k.@.)b|.....V...
00254e0: 57d5 ae32 5306 0c87 4e0a 7c88 d69b 4759  W..2S...N.|...GY
00254f0: 2096 dce3 1a14 36dd 1943 8dc6 76ae 5b21   .....6..C..v.[!
0025500: 6849 1858 87b6 a77f 82e2 d799 1faa 5c63  hI.X..........\c
```

UG1025_c8-01_082913

Figure 8-1: **User Defined Field in Authentication Certificate**

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For continual updates, add the Answer Record to your myAlerts:

www.xilinx.com/support/myalerts.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

The most up to date information related to the ZC702 and its documentation is available on these websites:

Zynq-7000 All Programmable SoC ZC702 Evaluation Kit product website

Zynq-7000 All Programmable SoC ZC702 Evaluation Kit documentation website

These Xilinx documents and sites provide material useful with this guide:

1. *Zynq-7000 All Programmable SoC ZC702 Evaluation Kit and Video and Imaging Kit Getting Started Guide* (UG926)
2. *Secure Boot of Zynq-7000 All Programmable SoC Application Note* (XAPP1175)
3. *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide* (UG850)
4. *Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques* (UG873)
5. *AMS101 Evaluation Card User Guide* (UG886)
6. *Zynq-7000 All Programmable AP SoC Technical Reference Manual* (UG585)
7. *Zynq-7000 All Programmable SoC Software Developer's Guide* (UG821)
8. Tera Term website: en.sourceforge.jp/projects/ttssh2/releases/
9. Silicon Laboratories USB-UART drivers website: www.silabs.com/Support%20Documents/Software/CP210x_VCP_Windows.zip

Warranty