XAPP1223 (v1.0.1) February 20, 2015

Changing the Cryptographic Key in Zynq-7000 AP SoC

Author: Lester Sanders

Summary

The cryptographic keys in secure embedded systems should be changed to increase security. System designers should first define the cryptographic requirements, including if and when to change the cryptographic keys. The definition of the cryptographic requirements analyzes the value of the information, use of the embedded system, and the expected threats to the embedded system. This note briefly discusses the cryptographic key requirements of embedded systems. A reference design provides methods for changing the AES key in the Zynq®-7000 All Programmable (AP) SoC device.

Download the Reference Design Files for this application note from the Xilinx website. For detailed information about the design files, see Reference Design, page 14.

Included Systems

The `zed_key_change` reference design contains four software projects:

- First Stage Boot Loader (FSBL)
- BBRAM key loader (BKL)
- Partition loader
- hello_world

The `bbram_key_loader_src` and `partition_loader_src` directories contain source code. The `completed` directory contains compiled projects and the `zed_key_change.bif`, `zed_key_change.mcs`, `zed_bbram1.nky`, and `zed_bbram2.nky` files.

Introduction

In modern cryptography, including Advanced Encryption Standard (AES), the cryptographic algorithm is public and the security of the system depends on the security of the cryptographic key. In Zynq-7000 AP SoC devices, the AES key can be stored in either one time programmable (OTP) eFUSE array or in battery-backed Random Access Memory (BBRAM). When BBRAM is used, the cryptographic key can be changed. Changing the key reduces the amount of time an adversary has to attack the key, and reduces the total amount of information protected by the key.

Prior to Zynq-7000 AP SoC devices, the AES cryptographic key was programmed into FPGAs using iMPACT or third-party software, usually before the system was fielded. The cryptographic key was used for the life of the device. With Zynq-7000 AP SoC devices, the key can be changed by a secure key driver running on a Zynq-7000 device CPU. This note shows how to change the BBRAM key for different partitions. The key can be changed at boot or run time.

The Define the Cryptographic Key Requirements section discusses security in general and provides reasons for changing the key in Zynq-7000 AP SoC devices. The reference design is run on the Zed or MicroZed evaluation platform. Board setup is discussed in Evaluation Boards, page 4, including the requirement for external connection to JTAG pins. A project development overview is given in Reference Design Development, page 5. The development of the software projects and their integration in Bootgen to create the image is discussed. The hardware verification of the key change on the Zed evaluation board is discussed in System Test, page 12.

Note: *Zed* stands for Zynq Evaluation and Development [Ref 1]

This application note assumes familiarity with the Xilinx Software Development Kit (SDK). For an introduction to using the SDK for secure applications, see *Secure Boot of Zynq-7000 All Programmable SoC Application Note* (XAPP1175) [Ref 2].

Hardware and Software Requirements

The hardware and software requirements for changing the cryptographic key in a Zynq-7000 device are:

- Zed or MicroZed evaluation board
- AC power adaptor (12 VDC)
- USB Type A to Micro Cable
- Xilinx Platform Cable USB II
- Four JTAG – MIO wire connectors
- Vivado® Design Suite 2014.3
- Xilinx Software Development Kit (SDK) 2014.3

Define the Cryptographic Key Requirements

The embedded system designer should start by specifying the cryptographic key requirements of the system, specifically defining if and when to change the key. The cryptographic key requirements are determined by the value of the information used by the embedded system, use of the embedded system, and the threat to the system.

Value of Information Used by the Embedded System

The three types of partitions used by Zynq-7000 AP SoC devices are software, data, and hardware (bitstream). The hardware and/or software might be valuable because the IP provides a competitive advantage. Data partitions might be valuable because they contain information which should be kept confidential, such as patient health records.

Use Model of the Embedded System

The use of the embedded system affects the cost and probability of an attack. If the embedded system is used within a security perimeter protected by security guards, it is subject to an insider attack, but usually less subject to theft and a physical attack. If the device is used in the field and is connected to the Internet with wired and/or wireless communication, an adversary might steal the device or intercept data in a man-in-the-middle attack. If a field-embedded system supports remote software update, a cryptographic key change is usually required.

Threat to the Embedded System

Two examples of threats to an embedded system are a physical attack and remote hacking. Threats are intrinsically related to the value of the information and the use model of the embedded system. The use model can help determine if the attack is most likely a physical attack on a system in-hand, such as a side channel differential power analysis (DPA) attack, or a remote attack over the Internet which creates a denial of service.

The value of the information might affect the protection needed. The protection needed might vary based on attacks to read confidential information, steal IP, or cause the system to be dysfunctional (Denial of Service).

If the embedded system is used in a mobile application, the owner might know if the device is stolen. If the device is replaceable at a reasonable cost and the cryptographic key is zeroized, a physical attack might not be a significant concern.

If the threat is that an adversary steals information by remotely hacking into the device, changing the key might provide protection.

Reasons to Change the Cryptographic Key

The following are reasons to change the cryptographic key:

- Best security practice – Even in closed systems, the key should be changed periodically.
- Open systems – Different software suppliers for an embedded system should use different cryptographic keys.
- Connected systems – Embedded systems are connected using wired and wireless Ethernet. When connected, systems are much more powerful, but an access point for a remote hacker is introduced.
- Multiple private peer-to-peer communication – Transferring sensitive information to multiple remote entities should use different cryptographic keys.

Evaluation Boards

The reference design uses the Xilinx and Avnet evaluation boards. Two Xilinx Zynq-7000 AP SoC device evaluation platforms are the ZC702 and ZC706. Two Avnet Zynq-7000 device evaluation platforms are the Zed and MicroZed boards. The early Zed boards did not support security. The ZC702 and ZC706 boards include a mounted battery for the BBRAM.

The board setup requirements are to make an external connection from the GPIO pins to the JTAG port. This connection is straightforward to define and implement on new boards. The inexpensive Avnet boards are used in this reference design.

Figure 1 shows the Zed board setup. The JTAG connections are to the J15 connector. The JTAG TMS, TCK, TDO, TDI ports are labeled on the board. The MIO connections are to the JE1 connector. The MIO (GPIO) to JTAG connection on the Zed and MicroZed boards is relatively straightforward.

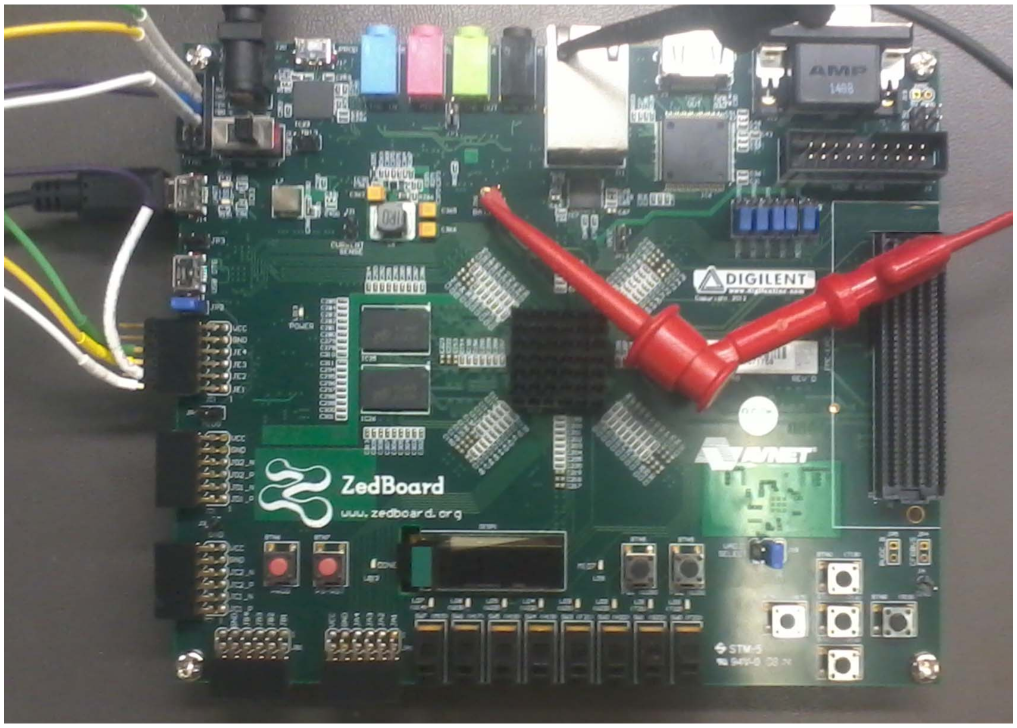


Figure 1: **Connecting MIO to JTAG on Zed Board**

The Zed or MicroZed boards do not have the battery for BBRAM. For testing a prototype, a power supply can be used instead of a battery for the BBRAM. For the Zed board, connect a power supply to the BATT/GND terminals on J16. For the MicroZed board, connect the power supply to S12. A photo of the MicroZed connection is provided in the reference design.

After prototyping using an evaluation board, a custom board can be developed in which the GPIO and JTAG ports are connected using a board trace.

Reference Design Development

The development of a system in which the BBRAM key is changed uses four software projects: First Stage Boot Loader (FSBL), BBRAM key loader (BKL), partition loader, and hello_world. In the reference design, the original cryptographic key is used by the FSBL, BBRAM key loader, and partition loader, and a second cryptographic key is used for the hello_world partition.

This application note shows the methodology to modify, on the device, the load/execution order so that the cryptographic key load software is executed prior to loading/decryption of the hello_world partition. This is done with the BBRAM key loader and partition loader.

The reason the standard flow of loading partitions is modified is that a key change requires changing the load/execution order of partitions. If unmodified, the FSBL loads all partitions and transfers execution to the last partition loaded. As noted, the hello_world partition is encrypted using the new cryptographic key. If the BBRAM key loader is loaded but not executed, the BBRAM key is not updated for decryption of the hello_world partition.

Figure 2 shows the standard flow for booting an embedded system.

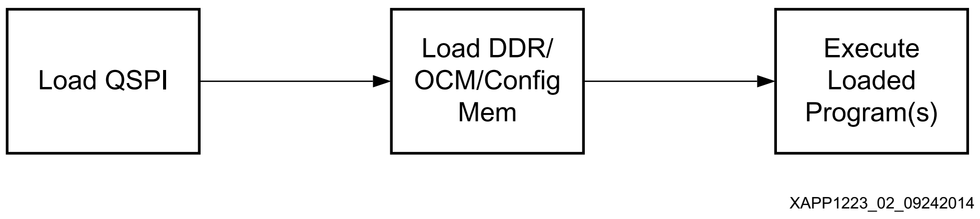


Figure 2: **Typical Steps in System Development**

Figure 3 shows the flow for the reference design in which the cryptographic key is changed. The load/execution order of the partitions is changed so that the partition encrypted with the new key can be decrypted.

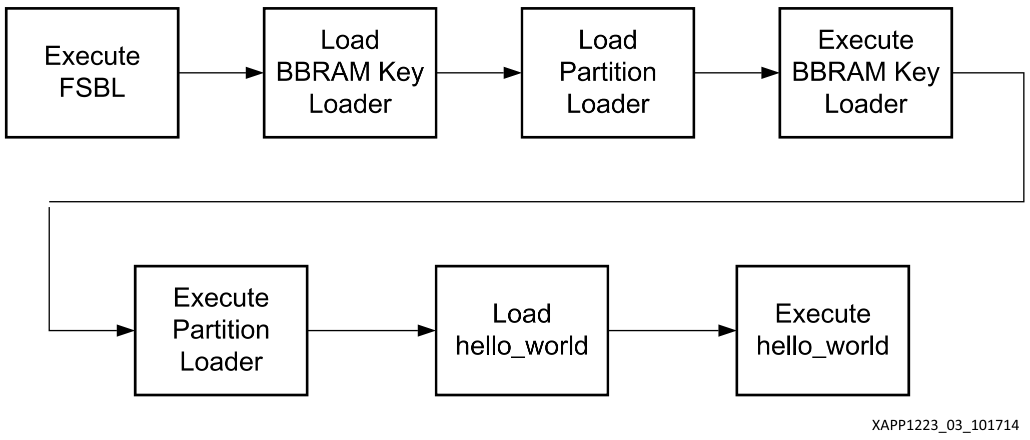


Figure 3: **Load/Execute Order for Changing a Cryptographic Key**

In the reference design, the following tasks are used to change the AES key:

- Block RAM key loader and partition loader software projects
- Enabling/disabling the JTAG port
- Handoff code so that all software projects run.
- Modifications to the Bootgen Image Format file

Address Map and Execution Handoff

The reference design requires the FSBL, BBRAM key loader, partition loader, and hello_world software to be executed sequentially. In the standard FSBL flow, only the last partition loaded is executed. To transition execution of one software partition to the next, a HandoffExit function is included in the software projects. The HandoffExit function uses the load address of the next partition to be executed. Table 1 shows the address map used for the software projects used in the reference design.

Table 1: **Address Map**

Software	Address
Block RAM key loader	`0xFFFFC000`
Partition loader	`0x200000`
hello_world	`0x300000`

With this address map, the HandoffExit functions, and the BIF partition_owner attribute discussed later, the execution order is **FSBL > BBRAM Key Loader > Partition Loader > hello_world**.

First Stage Boot Loader and Hello_World Projects

The SDK New Project Templates are used to compile the FSBL and hello_world projects. The hello_world project is compiled without changes. After the initial compilation, minor changes to the FSBL project are required.

Since this reference design is run in secure mode, the JTAG port is disabled. The Secure Key Driver used by the BBRAM key loader uses the JTAG port. The `fsbl_src/main.c` code is modified to enable the JTAG port. To enable the DAP/JTAG port, code in `main.c` writes CTRL(23).

The standard FSBL enforces a load order in which the bitstream must be loaded as the partition immediately following the FSBL. In some systems, it might be necessary to change the FSBL code to relax this requirement.

1. At the command prompt, enter **xsdk &.**
2. Click **Create Application Project.**
3. In the New Project dialog box, type **fsbl** as the Project Name, and select the **Hardware Platform**. Click **Next**.

4. In the New Project Templates, select **Zynq FSBL** and click **Finish**. In `fsbl/src`, replace `main.c` with the **`fsbl_src/main.c`** in the reference design.
5. In the Project Explorer pane, right-click the **fsbl** project, and select **C/C++ Build Settings > Symbols > Defined Symbols +**. Type **FSBL_DEBUG_INFO** in the text box. Click **OK**.
6. Repeat these steps to create a hello_world software project. The hello_world software project does not require code changes.
7. In the Project Explorer pane, double-click **hello_world > src > lscript.ld**, and enter **`0x00300000`** as the Base Address for ps7_ddr_0_S_AXI_BASEADDR.

BBRAM Key Loader

The BBRAM key loader uses the Secure Key Driver (SKD) library in the SDK `lib/sw_services` directory. The SKD can be used to program either the eFUSE and/or the BBRAM key. Traditionally, iMPACT or a device programmer such as one available from BP Microsystems was used to program the cryptographic key.

To create the BBRAM key loader software project, first create the BBRAM key loader Board Support Package using the following commands.

1. Click **File > New> Board Support Package**.
2. Enter **bbram_key_loader_bsp_0** in the Project Name text box. Select **zed_hw_platform** for the **Hardware Platform**. Click **Finish**.
3. In the Board Support Package Settings window, check the **xilskey** box. Click **Finish**.
4. Click **File > New > Application Project**.
5. Enter **bbram_key_loader** in the Project Name text box. Select **zed_hw_platform** for the **Hardware Platform**. Select **Use Existing > bbbram_key_loader_bsp_0** BSP. Click **Next**.
6. Select **Empty Application** in the New Project Templates window.
7. In the Project Explorer pane, right-click **bbram_key_loader > src** and select **Import > General > File System**. Click **Next**.
8. Use the Browse box to select the **xilskey_v2_0/examples** directory in the reference design. Click **OK**.
9. Select the `xilskey_bbram_example.c`, `xilskey_input.h`, and `handoff.S` files.

   The xilskey source files are also in the SDK install area, under `lib/sw_services`. In the reference design copy of the xilskey_v2_0/examples directory, a sample AES key is added to `xilskey_input.h`, and handoff code is added to `xilskey_bbram_example.c.` After adding the source files, click **Finish**.
10. In the Project Explorer pane, double-click **bbram_key_loader > src > lscript.ld**, and enter **0xFFFFC000** as the Base Address for ps7_ram_0_S_AXI_BASEADDR.

Table 2 shows the connections used for the ZC702 and Zed/MicroZed boards.

Table 2: **MIO to JTAG Connection**

Pin Name	ZC702 MIO	Zed MIO	MicroZed MIO
TDI	17	13	13
TDO	18	10	10
TCK	19	11	11
TMS	20	12	12

The MIO to JTAG connection is defined in the `xilskey_input.h` file.

Secure Key Driver documentation is provided in

```
$ Xilinx/../SDK/2014.3/data/embeddedsw/lib/sw_services/xilskey_v2_0/doc
```

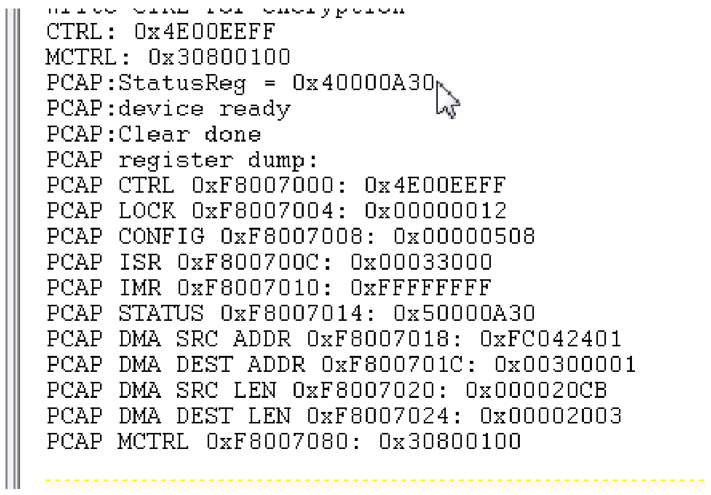
Partition Loader

The partition loader is a small code set which copies a single partition from a source to a destination address. The partition loader provides an easy-to-use interface to two hardware functions on the Zynq-7000 AP SoC: the Device Configuration Direct Memory Access Controller (DevC DMAC) and the AES Decryptor. The devcfg device driver is located in the `SDK/../data/embeddedsw/XilinxProcessorIPLib/drivers/devcfg_v3_1` directory. The devcfg device driver functions are used by the partition loader to control the DevC DMAC and AES Decryptor. If the partition copied is a software or data partition, the PcapDataTransfer function is used. If the partition copied is hardware (bitstream), the PcapLoadPartition function is used.

The partition loader accepts five arguments: source address, destination address, source size, destination size, and an argument which specifies if the partition is routed through the decryptor when it is copied. The source address is the partition's Bootgen generated Quad Serial Peripheral Interface (Quad SPI) flash memory address. This address is written by the SDK Flash Writer. The destination address is the address in DDR, on-chip memory (OCM), or configuration memory in the Zynq-7000 AP SoC device. If encryption is used, the partition source length is larger than the destination length.

There are two methods to determine a partition source and destination address and size. When the symbol FSBL_DEBUG_INFO is a compilation option, the FSBL debug log file provides the address and size information.

The tail section of the FSBL debug log file in Figure 4 shows the source and destination address and size of the bbram_key_loader partition.



```
CTRL: 0x4E00EEFF
MCTRL: 0x30800100
PCAP:StatusReg = 0x40000A30
PCAP:device ready
PCAP:Clear done
PCAP register dump:
PCAP CTRL 0xF8007000: 0x4E00EEFF
PCAP LOCK 0xF8007004: 0x00000012
PCAP CONFIG 0xF8007008: 0x00000508
PCAP ISR 0xF800700C: 0x00033000
PCAP IMR 0xF8007010: 0xFFFFFFFF
PCAP STATUS 0xF8007014: 0x50000A30
PCAP DMA SRC ADDR 0xF8007018: 0xFC042401
PCAP DMA DEST ADDR 0xF800701C: 0x00300001
PCAP DMA SRC LEN 0xF8007020: 0x000020CB
PCAP DMA DEST LEN 0xF8007024: 0x00002003
PCAP MCTRL 0xF8007080: 0x30800100
```

Figure 4: **Log File with Source, Destination Address of bbram_key_loader Partition**

The partition address and size is also provided when bootgen is run with the `–debug` command option.

If the DevC DMAC hangs during partition loader execution, verify that the size (SRC LEN, DEST LEN) is correct. The partition size information can be presented or required in terms of the number of bytes or words. To translate between words and bytes, use the following python code:

```
int("f6d20",16)
1010976
1010976 * 4
4043904
hex(4043904)
0x3db480
```

In this reference design, the partition loader copies the hello_world software from Quad SPI flash memory to DDR, with routing through the AES Decryptor.

Create the Partition Loader software project as an **Empty Application**.

1. In the Project Explorer pane, right-click **partition_loader > src > Import**.
2. In the **Import** dialog box, select **General > File System**, and click **Next**.
3. Use the **Browse** box to select the `partition_loader_src` directory in the reference design. Click **OK**.
4. Click **Select All**. Click **Finish**.
5. In the Project Explorer pane, double-click **partition_loader >src > lscript.ld**.
6. Enter **0x00200000** as the Base Address for ps_ddr_0_S_AXI_BASEADDR.

Following the SDK Flash Writer load, the `bbram_key_loader` partition is encrypted in Quad SPI flash memory. The standard FSBL operation of decrypting/copying the bbram_key_loader partition to DDR cannot be used because the cryptographic key is included in `xilskey_input.h`. The partition loader copies the `bbram_key_loader` partition to DDR without routing it through the decryptor. When the BBRAM key loader is used, the partition is decrypted and copied to OCM. The partition loader is executed from OCM.

In addition to copying and decrypting partitions, the partition loader supports tasks such as storing an encrypted cryptographic key in the Zynq-7000 device's OCM and decrypting the key just prior to use.

Bootgen

Bootgen is the SDK tool which integrates partitions and writes the image in a BIN or MCS format. To use Bootgen, create a Bootgen Image Format (BIF) file using either the Bootgen GUI or a text editor. The BIF file is a list of partitions included in the image. For each partition in the list, the BIF file optionally includes partition attributes which specify such options as load address, encryption, and authentication for each partition. The partition_owner attribute controls whether the FSBL loads the partition. The partition_owner attribute is used when a second loader such as U-Boot, or in the reference design case, the partition loader, loads the partition.

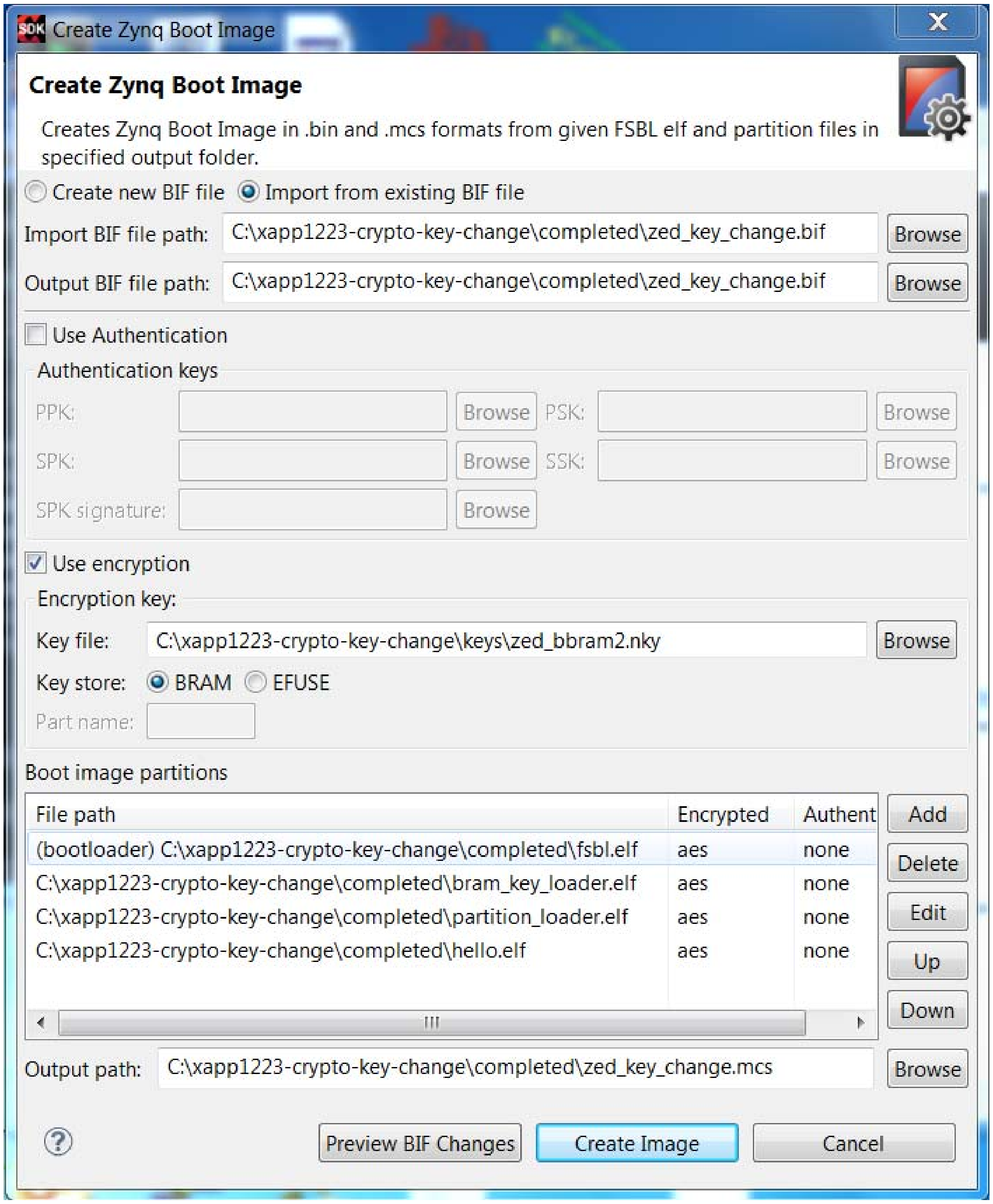
The reference design shows how to create the image in which load/execution order requirements are met.

Bootgen Image Format File

Specifying multiple cryptographic keys is not currently supported in the Bootgen GUI. To create the BIF for the reference design, the Bootgen GUI creates a template BIF, and a text editor is used to define the keys in the BIF.

1. In SDK, click **Tools > Create Zynq Boot Image**.
2. Check **Use encryption**.
3. In the Key File text box, browse to `zed_bbram1.nky` (provided in the reference design keys directory). Click **Add**.
4. In the File path text box, browse to `fsbl.elf`.
5. Select **bootloader** as the Partition type. Select **aes** for Encryption. Click **OK**.
6. Repeat the **Add** steps for `fsbl.elf` to add the `bbbram_key_loader.elf`, `partition_loader.elf`, and `hello_world.elf` partitions.
7. Change the partition type to **datafile**.
8. Use **Browse** to specify the **Output BIF file path**: to `zed_key_change.bif`.
9. Use **Browse** to specify the **Output path**: to `zed_key_change.mcs`.
10. Click **Create Image**.

Figure 5 shows the Bootgen GUI interface with four partitions added.



Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

( ) Create new BIF file (●) Import from existing BIF file

Import BIF file path: C:\xapp1223-crypto-key-change\completed\zed_key_change.bif

Output BIF file path: C:\xapp1223-crypto-key-change\completed\zed_key_change.bif

☐ Use Authentication

Authentication keys

PPK: PSK:

SPK: SSK:

SPK signature:

☑ Use encryption

Encryption key:

Key file: C:\xapp1223-crypto-key-change\keys\zed_bbram2.nky

Key store: (●) BRAM ( ) EFUSE

Part name:

Boot image partitions

File path	Encrypted	Authent
(bootloader) C:\xapp1223-crypto-key-change\completed\fsbl.elf	aes	none
C:\xapp1223-crypto-key-change\completed\bram_key_loader.elf	aes	none
C:\xapp1223-crypto-key-change\completed\partition_loader.elf	aes	none
C:\xapp1223-crypto-key-change\completed\hello.elf	aes	none

Output path: C:\xapp1223-crypto-key-change\completed\zed_key_change.mcs

Figure 5: **Creating BIF Using Bootgen GUI**

11. Use a text editor to add the second cryptographic key, `zed_bbram2.nky`, to the Bootgen GUI generated BIF. Add the partition_owner attribute to the hello_world partition. The final BIF is in the completed /zed_key_change.bif.

12. At the command prompt, run the following command to create a MCS file.

```
bootgen -image zed_change_key.bif -o zed_change_key.mcs -encrypt bbram -w on
```

Alternately, in SDK, the manually edited `zed_change_key.bif` can be read into the Bootgen GUI, and the MCS file can be created using the GUI.

System Test

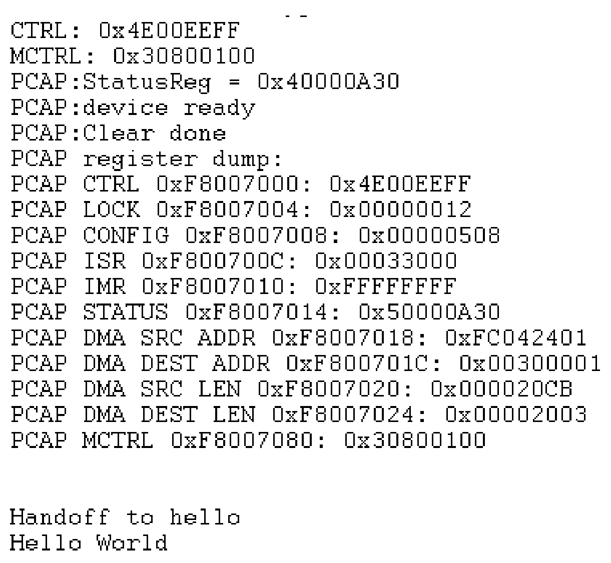
To test the cryptographic key change, use the SDK Flash Writer to write `zed_key_change.mcs` into Quad SPI flash memory. If there are problems writing QSPI, set the Zed/MicroZed board switches to JTAG programming mode for this operation. All partitions are loaded into Quad SPI flash memory.

1. In SDK, click **Xilinx Tools > Program Flash**.
2. In the Program Flash Memory dialog box, use **Browse** to select `zed_key_change.mcs` as the Image file.
3. In the Offset text box, specify **0x0**.
4. Click **Program**.
5. If the Zed board is used and a battery is not added, connect a power supply to the J16 BATT/GND terminal. Program the initial `zed_bbram1.nky` key into BBRAM using iMPACT.

   impact -batch xapp1223-crypto-key-change/keys/bbram1_key_load.cmd

6. Replace the Platform Cable USB II with the connector from JE1 as shown in Figure 1 and defined in Table 2.
7. After loading Quad SPI flash memory, power the board off, change the Mode Select pins to QSPI mode, and power the board up. Start a communication terminal such as Tera Term. Press the PS_RST (RST) button on the board.

   The tail of the expected debug log output is shown in Figure 6.



```
CTRL: 0x4E00EEFF
MCTRL: 0x30800100
PCAP:StatusReg = 0x40000A30
PCAP:device ready
PCAP:Clear done
PCAP register dump:
PCAP CTRL 0xF8007000: 0x4E00EEFF
PCAP LOCK 0xF8007004: 0x00000012
PCAP CONFIG 0xF8007008: 0x00000508
PCAP ISR 0xF800700C: 0x00033000
PCAP IMR 0xF8007010: 0xFFFFFFFF
PCAP STATUS 0xF8007014: 0x50000A30
PCAP DMA SRC ADDR 0xF8007018: 0xFC042401
PCAP DMA DEST ADDR 0xF800701C: 0x00300001
PCAP DMA SRC LEN 0xF8007020: 0x000020CB
PCAP DMA DEST LEN 0xF8007024: 0x00002003
PCAP MCTRL 0xF8007080: 0x30800100


Handoff to hello
Hello World
```

Figure 6: **Log of Changing the BBRAM Key**

8. Using the log in the reference design, verify that the BBRAM key loader writes the BBRAM and hands execution off to the partition loader. Verify that the partition loader copies the hello_world partition to DDR, through the AES decryptor, and hands off execution to hello_world, which writes "Hello World" to the communication terminal.

Notes on Implementation

This application note provides methods to change the cryptographic key. In an actual system, there are other considerations which are discussed in this section.

The change of the cryptographic key in Zynq-7000 AP SoC devices provides a limited defense against an adversary who steals the embedded system and monitors signals on the connection from MIO/EMIO to the JTAG port. The external JTAG connection must be protected to protect the key. Anti-tamper information is provided in *Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs* (XAPP1084) [Ref 3].

In Zynq-7000 AP SoC devices, a BBRAM write transaction clears the programmable logic (FPGA) section, so the bitstream must be re-written. As noted, the partition loader is used to write partitions.

The reference design uses two cryptographic keys. The methodology is extendable to any number of keys. Any number of BBRAM key loader/partition loader(s) can be included in a BIF.

The reference design uses a combination of boot and run time processes, since the handoff is done in run time.

Using the partition loader, the methodology is extendible to remote re-keying. This application note does not address key exchange. To communicate with a remote entity (peer, host), the cryptographic key needs to be exchanged, it must be encrypted, and it should be signed. Bootgen supports encrypting/signing a single partition. A 256-bit cryptographic key is a data partition. The key to be stored is encrypted in NVM, DDR, or on-chip, within the Zynq-7000 device security perimeter, and copied/decrypted by the partition loader when the key is needed.

The Secure Key Driver currently writes the cryptographic key, which is hard-coded in the `xilskey_input.h` file, into BBRAM. In some applications, the key(s) is stored encrypted in OCM or AXI block RAM, at a memory address. Using a memory address for the key rather than the hard-coded key in `xilskey_input.h` requires minor code changes to the Secure Key Driver.

The BBRAM key is secure since it is within the security perimeter of the Zynq-7000 device. If the key is no longer required, a security option is to zeroize it by using 0s as the key value in `xilskey_input.h`. The zeroization can then be done using a BBRAM key loader/partition loader pair. After the cryptographic key is loaded, disable the JTAG port in `xilskey_bram_example.c` by writing CTRL(23).

The cryptographic key requirements specification should define the key requirements for the life cycle of the device. The cryptographic key used by the partitions loaded at boot must be present in the BBRAM at boot time.

Note: After executing a BBRAM key loader/partition loader operation to change the cryptographic key during run time, it is necessary to reload the cryptographic key used at power-up.

Conclusion

This application note shows how to change the BBRAM key in the Zynq-7000 All Programmable SoC. The techniques used in this reference design can be re-used to change the key as many times as required. The key can be changed for software, data, or hardware partitions. The BBRAM key loader and partition loader can change the key for several key requirements.

Reference Design

Download the Reference Design Files for this application note from the Xilinx website.

Table 3 shows the reference design matrix.

Table 3: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Lester Sanders
Target devices	Zynq-7000 AP SoC
Source code provided	Yes
Source code format	VHDL/Verilog
Design uses code and IP from existing Xilinx application note and reference designs or third-party	No
Simulation	
Functional simulation performed	No
Timing simulation performed	N/A
Test bench used for functional and timing simulations	N/A
Test bench format	Verilog
Simulator software/version used	Vivado simulator
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado synthesis
Implementation software tools/versions used	Vivado implementation
Static timing analysis performed	Yes

Table 3: **Reference Design Matrix *(Cont'd)***

Parameter	Description
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	Zed board

References

1. *ZedBoard Getting Started Guide*
2. *Secure Boot of Zynq-7000 All Programmable SoC Application Note* (XAPP1175)
3. *Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs* (XAPP1084)
4. *ZedBoard Configuration and Booting Guide:* zedboard.org/support/design/1521/11
5. *ZedBoard Hardware User's Guide:* zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf
6. *MicroZed and System on Module Hardware User Guide:* zedboard.org/support/documentation/1519

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/06/2014	1.0	Initial Xilinx release.
02/20/2015	1.0.1	Revised the Reference Design link.

Please Read: Important Legal Notices