XAPP1224 (v1.1) February 12, 2015

Updating a System Securely in the Zynq-7000 AP SoC

Author: Lester Sanders

Summary

When connected to a network, software, hardware, and data files can be updated in the Zynq®-7000 All Programmable (AP) SoC based embedded systems during operation. Partitions updated over a network need to be protected to ensure security. This application note provides methods to securely update an embedded system while the Zynq-7000 AP SoC is operational.

You can download the reference design files for this application note from the Xilinx® website. For detailed information about the design files, see Reference Design.

Included Systems

The system_update reference system contains the following software projects:

- fsbl
- hello
- partition_loader
- u-boot
- hello_update

The ELFs for these software projects, cryptographic keys, and Bootgen Image Format (BIF) file are included in the `system_update/completed` directory. The source files are provided in the `hello_src` and `partition_loader_src` directories.

Introduction

The common method for re-programming an SoC or FPGA based embedded system is to:

1. Power the embedded system down.
2. Physically connect the system to a PC in which Xilinx software is loaded.
3. Download the image with all partitions into nonvolatile memory (NVM).
4. Power-up.

During this interval, the embedded system is not available. When connected to a network, the Zynq-7000 device allows software, hardware, or data to be loaded during run time using either a partition loader or U-Boot. The reference system shows how to use the partition loader and U-Boot to update a system.

This application note contains the following sections:

- The Reasons for System Updates section lists reasons for remotely updating an embedded system.
- The High Level Description of System Update section lists the tasks required for a system update. The tasks are divided into tasks done at the factory and tasks done on an embedded system in the field.
- The System Update Implementation section uses the Xilinx Software Development Kit (SDK) to build software projects. SDK is then used to create and program the image. If the reader is not familiar with using SDK to create software projects, *Secure Boot of Zynq-7000 All Programmable SoC* (XAPP1175) [Ref 1] provides SDK steps for creating systems. This section also discusses the use of the Partition Loader and U-Boot to update a system.
- The Test of System Update section shows how to run and view the results of the reference system. In a secure system update, AES encryption and RSA authentication are used.
- The Security in a System Update section provides methods to ensure confidentiality and integrity of the update.

Hardware and Software Requirements

The hardware and software requirements for updating software in Zynq AP SoC are:

- ZC702, ZED, or MicroZed evaluation board
- AC power adaptor (12 VDC)
- USB Type A to Micro cable
- Xilinx Platform Cable USB II
- Xilinx Software Development Kit 2014.3
- Xilinx Vivado® Design Suite 2014.3

Reasons for System Updates

Some advantages which can be realized from a system update are listed below.

- Reduce development cycle: the ease of adding new functionality means all features do not need to be in the original release
- Fault management: fix defects with a patch

- Software and/or hardware updates extend the life cycle of silicon
- Secure transfer of data records such as patient health records
- Increase system functionality of silicon by time multiplexing hardware and/or software operations
- Remote control of power consumption.
- Remote wipe in the event of a security breach or when an employee leaves.
- Extend the life cycle of embedded systems with updated software and/or hardware.

High Level Description of System Update

Traditionally, field programmable gate arrays (FPGAs) are programmed by the FPGA designer, and the end user does not change the functionality. In Zynq-7000 AP SoC devices, a user initiated system update increases product capability. The processing system (PS) and programmable logic (PL) sections are programmable. Since both the PS and PL are programmed using a conventional device driver, providing support for a user initiated update is feasible.

In addition to a user initiated update, third-party vendors provide device management software for system updates of a large number of embedded systems. These vendors provide a high level interfaces to the Open Mobile Alliance - Device Management (OMA DM) standard to provide efficient, secure updating of a large number of fielded embedded systems.

The methods used in this application note to implement a system update support either a user or device manager initiated system update.

A system update uses a wired or wireless network connection, with wireless the most convenient. In a system update, a single partition is updated. A partition is either software (ELF/BIN), hardware (bitstream), or data file. An update can be done in secure or non-secure mode. Figure 1 shows an overview of a system update. The term factory is used to mean the central site for developing and distributing software to fielded embedded systems. If used, the device manager can be an independent operation from the development organization. The factory creates the partition. The factory or device manager transmits the partition to the embedded system. The partition's initial destination is usually the embedded system DDR memory.

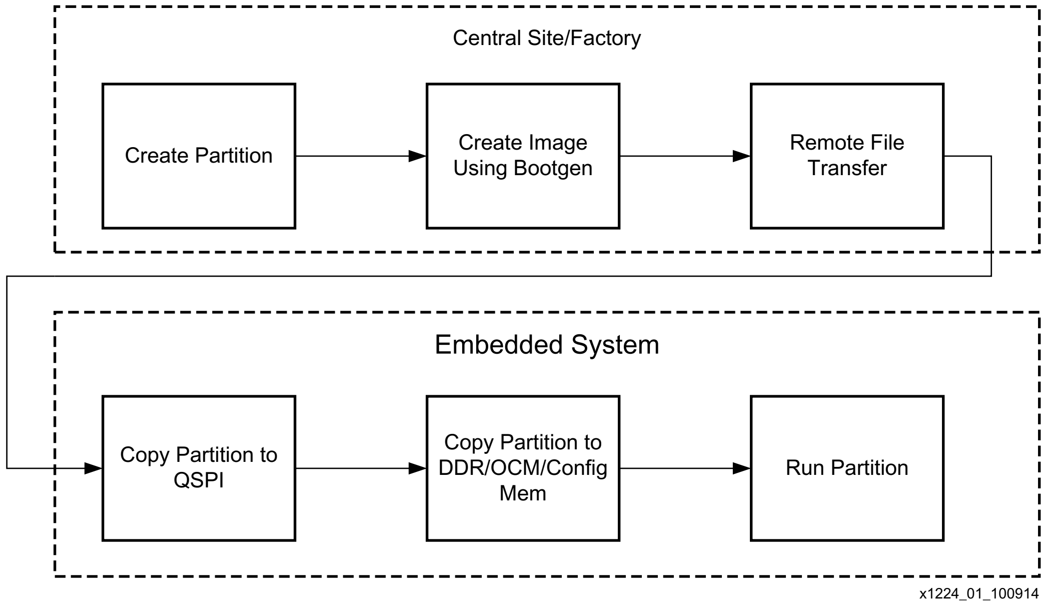


Figure 1: **System Update Flow Overview**

This application note does not address the transfer of the partition from the factory to the embedded system. The transfer typically uses wired or wireless Ethernet.

After the partition is transferred to the embedded system, either the Partition Loader or U-Boot copies the partition from DDR to another location in DDR, On-Chip Memory (OCM), or configuration memory. If U-Boot is used, the partition in DDR can be written to Quad-SPI. Writing Quad-SPI is required if the embedded system is to load the updated partition when it is re-booted. The Partition Loader cannot write Quad-SPI.

The Partition Loader is smaller and easier to use than U-Boot. U-Boot provides more functionality. Both the Partition Loader and U-Boot copy software, hardware, or data partitions, and both support AES decryption and RSA authentication. If custom modifications are needed, the modifications to the Partition Loader can remain proprietary. U-Boot is open source. Some embedded systems will use both the Partition Loader and U-Boot.

When run on Zynq devices, Linux can load partitions, including the bitstream. Linux can also write partitions to Quad-SPI. As of the Vivado Design Suite 2014.3 release, Linux bitstream configuration operations have not been tested in secure mode.

Note: To make a system update persistent on boot, U-Boot must write the update to NVM. If Quad-SPI is used as NVM, the U-Boot `sf write` command is used.

System Update Implementation

Figure 2 shows the functionality of the system update flow. The First Stage Boot Loader (FSBL) copies the `hello` software project from nonvolatile memory (NVM) to DDR memory. The `hello` software displays "Hello World" in the communication terminal. After the `hello` program, the partition loader is run. For the system update, the partition loader loads `hello_update`. When run, the "Hello World Update" output is displayed on the communication terminal.

In an actual system, a scheduler is responsible for running programs. The software projects in this application note provide a heuristic reference system which emulates boot and run time operation. In real systems, system updates occur asynchronously. The load of the FSBL and `hello` occur at boot time. In the reference system, run time is modeled when the CPU transitions from `hello` to `partition_loader`.

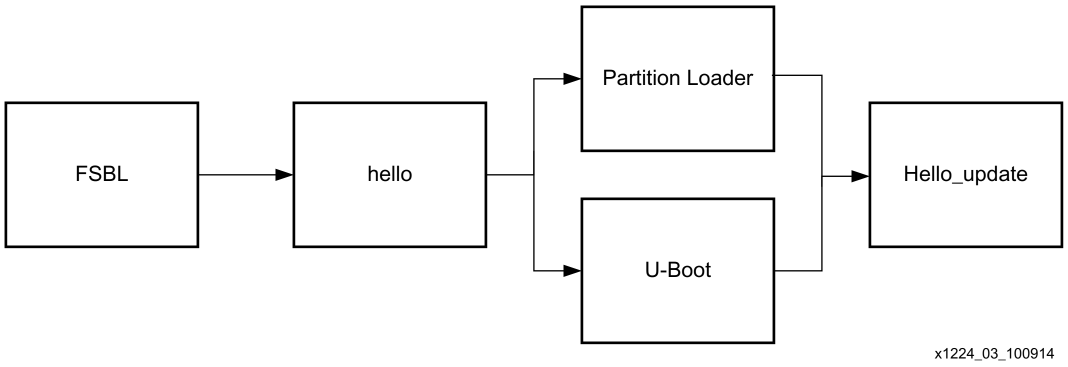


Figure 2: **System Update Flow Functionality**

The following steps use SDK to implement the system update.

1. Create `fsbl`, `hello`, `partition_loader` and/or U-Boot, and `hello_update` software projects.
2. Create the `system_update.mcs` image using Bootgen.
3. Write Quad-SPI using Flash Writer.
4. SDK is used to create FSBL and hello project templates. Create a directory named `system_update` and invoke SDK.
5. In the Workspace Launcher text box, browse to `<path>\system_update`. Click **OK**.
6. Click **File > New > Application Project**.
7. Enter **`fsbl`** as the Project Name.
8. Change **Target Hardware > Hardware Platform** to **ZC702_hw_platform**. Click **Next**.
9. In the New Project Templates dialog box, click **Zynq FSBL**. Click **Finish**.
10. After fsbl compilation is complete, right-click on **fsbl** in the Project Explorer pane.
11. Click **C/C++ Build Settings**.

12. Click **Symbols**. Click '**+**'.
13. Type `FSBL_DEBUG_INFO` in the text box. Click **OK**.
14. Repeat the steps used to create the FSBL project to create the `hello` and `hello_update` projects. In the New Project Template dialog box, select **Hello World**.
15. Modify the `hello` project in the Project Explorer pane. The source files for the change are in the `hello_src` directory.
    a. Use **File Import -> General -> File System** to add `system update/hello_src/handoff.S` to `hello/src`.
    b. Replace the `hello/src/helloworld.c` with `system_update/hello_src/helloworld.c`.

       The new `helloworld.c` adds three lines to the code so that the Partition Loader is executed after the `hello` project displays "Hello World."

       The following two lines are located just before **int main()**:

       ```
       u32 PartitionLoaderStartAddress = 0x200000;
       void HandoffExit(u32 PartitionLoaderStartAddress);
       ```

       The following line is placed after `cleanup_platform();`:

       ```
       HandoffExit(0x200000);
       ```
    c. Modify the `hello_update` project in the Project Explorer pane. Edit the print statement in `helloworld.c` to

       ```
       print("Hello World from updated hello");
       ```
    d. In the Project Explorer pane, modify the `hello_update/src/lscript.ld` linker script to locate `hello_update` at `0x300000`.

Partition Loader

The Partition Loader copies software, data, or hardware partitions. The Partition Loader is a wrapper for the devcfg device driver. The devcfg driver source code is located in the SDK install area, under the `XilinxProcessorLib/driver/devcfg_v*` directories.

The `PcapDataTransfer` and `PcapLoadPartition` functions are located in `pcap.c`. The `PcapDataTransfer` function copies a software or data partition. If the bitstream is copied to configuration memory, the `PcapLoadPartition` function is used. The `PcapDataTransfer` function can be used to copy the bitstream from NVM to DDR without decryption.

The arguments to the Partition Loader functions are source and destination address, source and destination size, and whether to route the partition through the decryptor. The partition address and size parameters are hard coded.

The Partition Loader in the reference system illustrates basic system update capability. To provide the user or Device Manager with an interface to update a system, modify the Partition Loader functions to allow the address and size parameters to be provided as arguments or memory addresses.

There are two methods to get a partition's address and size.

- The first method is to run Bootgen using the `-debug` option:

  ```
  bootgen -image hello_update.bif -o hello_update.bin -w on -debug
  ```

  Bootgen also provides address and size information when the BIF includes all partitions.

- The second method to obtain address and size information is to use the log output of the FSBL. The log is provided when the FSBL is compiled with the FSBL_DEBUG_INFO option.

The Partition Loader does not currently use RSA authentication. *Run Time Integrity and Authentication Check of Zynq-7000 All Programmable SoC System Memory* (XAPP1225) [Ref 2] shows how to use the Xilinx RSA library to authenticate a partition.

Create the Partition Loader software project as follows:

1. Click **File > New > Application Project**
2. Type **`partition_loader`** in the **Project Name** text box. Click **Next**.
3. In the New Project Templates, select **Empty Application**. Click **Finish**.
4. In the Project Explorer pane, right-click **partition_loader > src**
5. Select **Import > General -> File System**.
6. Browse to `system_update/partition_loader_src`, and select all files. Click **OK**.
7. In the Project Explorer pane, select `partition_loader/src/lscript.ld`
8. Modify the linker script to locate `partition_loader` at `0x200000`.

The Partition Loader is included in the BIF used for initial boot, so that it is resident in memory at run time.

U-Boot

U-Boot is a widely used universal boot loader, commonly used to load the Linux kernel, device tree, root file system and Linux applications. [Ref 3] provides documentation, including the user manual, for U-Boot. [Ref 4] provides Xilinx specific documentation on configuring and building U-Boot. Most texts on embedded Linux include a section on U-Boot.

1. To download U-Boot from the Xilinx Git server, enter:

   ```
   git clone git://github.com/Xilinx/u-boot-xlnx.git
   cd u-boot-xlnx
   ```

2. Add the following `define` to `include/configs/zynq-common.h`.

   ```
   #define CONFIG_CMD_ZYNQ_AES
   ```

3. Configure and build U-Boot by entering:

   ```
   make arch=ARM zynq_zc70x (zynq_zed)
   ```

The following set of commands illustrates how to use U-Boot to configure the PL.

```
promgen -w -b -p bin -o system.bin -u 0 system.bit -datawidth 32
Copy system.bin to a SD card.
```

4. Invoke XMD and run the following XMD commands:

```
connect arm hw
dow fsbl.elf
con
stop dow u-boot.elf
con
```

5. Press a key to prevent autoboot.
6. At the U-Boot prompt, run the following commands in the communication terminal:

```
mmcinfo
fatload mmc 0 0x100000 system.bin
fpga load 0 0x100000 0x3DBAFC
```

7. To route the bitstream through the decryptor, replace the `fpga` command with the `zynqaes` command:

```
zynqaes <srcaddr> <srclen> <destaddr> <destlen>
```

The `include/configs/zynq-common.h` section for the qspiboot loads the Linux kernel, device tree, and root file system. The commands can be changed to load `hello_update` using `zynqaes`. The `system.bin` partition is encrypted by Bootgen.

Bootgen

Invoke the SDK Bootgen GUI by entering:

1. **Xilinx Tools > Create Zynq Boot Image**
2. Specify the path and `system_update.bif` file name in the BIF File Path dialog box.
3. Click '**+**' and add `fsbl.elf`.
4. Click **Add**, and add the `hello.elf`, `partition_loader.elf` and `hello_update.elf` files.
5. Specify `system_update.mcs` as the output file.
6. Click **Create Image**.

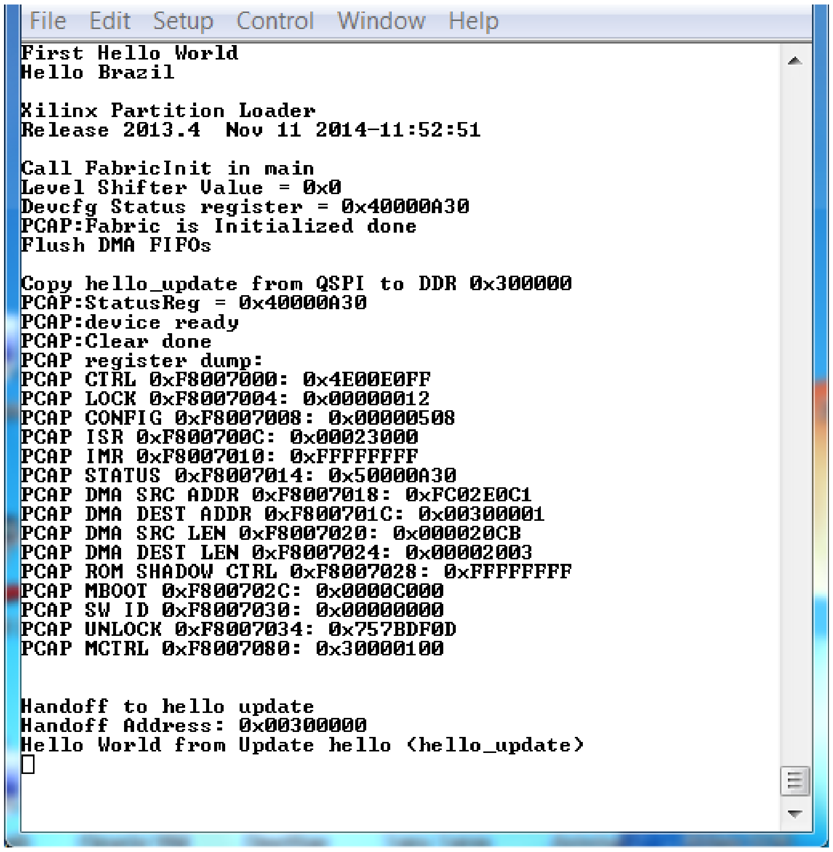
Programming Quad Serial Peripheral Interface (Quad-SPI)

1. Invoke SDK Program Flash by selecting **Xilinx Tools > Program Flash**
2. Browse to `system_update.mcs` to complete the `Image File` input.
3. Type **`0x0`** in the Offset dialog box.
4. Click **Program**.

Test of System Update

1. Invoke a communication terminal as Teraterm using a 115200 baud rate.
2. On the evaluation board, change the boot mode select switch MIO5 to 1 to boot using Quad-SPI mode.

As shown in Figure 3, the message in the communication terminal shows the activity of the FSBL, hello, partition loader, and hello update program. The initial hello software displays "Hello World" and "Hello Brazil." After the code transitions to the partition loader, "Xilinx Partition Loader" is displayed. The hello system update is shown by the "Hello World Update" message.



```
First Hello World
Hello Brazil

Xilinx Partition Loader
Release 2013.4  Nov 11 2014-11:52:51

Call FabricInit in main
Level Shifter Value = 0x0
Devcfg Status register = 0x40000A30
PCAP:Fabric is Initialized done
Flush DMA FIFOs

Copy hello_update from QSPI to DDR 0x300000
PCAP:StatusReg = 0x40000A30
PCAP:device ready
PCAP:Clear done
PCAP register dump:
PCAP CTRL 0xF8007000: 0x4E00E0FF
PCAP LOCK 0xF8007004: 0x00000012
PCAP CONFIG 0xF8007008: 0x00000508
PCAP ISR 0xF800700C: 0x00023000
PCAP IMR 0xF8007010: 0xFFFFFFFF
PCAP STATUS 0xF8007014: 0x50000A30
PCAP DMA SRC ADDR 0xF8007018: 0xFC02E0C1
PCAP DMA DEST ADDR 0xF800701C: 0x00300001
PCAP DMA SRC LEN 0xF8007020: 0x000020CB
PCAP DMA DEST LEN 0xF8007024: 0x00002003
PCAP ROM SHADOW CTRL 0xF8007028: 0xFFFFFFFF
PCAP MBOOT 0xF800702C: 0x0000C000
PCAP SW ID 0xF8007030: 0x00000000
PCAP UNLOCK 0xF8007034: 0x757BDF0D
PCAP MCTRL 0xF8007080: 0x30000100


Handoff to hello update
Handoff Address: 0x00300000
Hello World from Update hello (hello_update)
```

X1224_05_010615

Figure 3: **Communication Terminal Output Showing System Update**

Security in a System Update

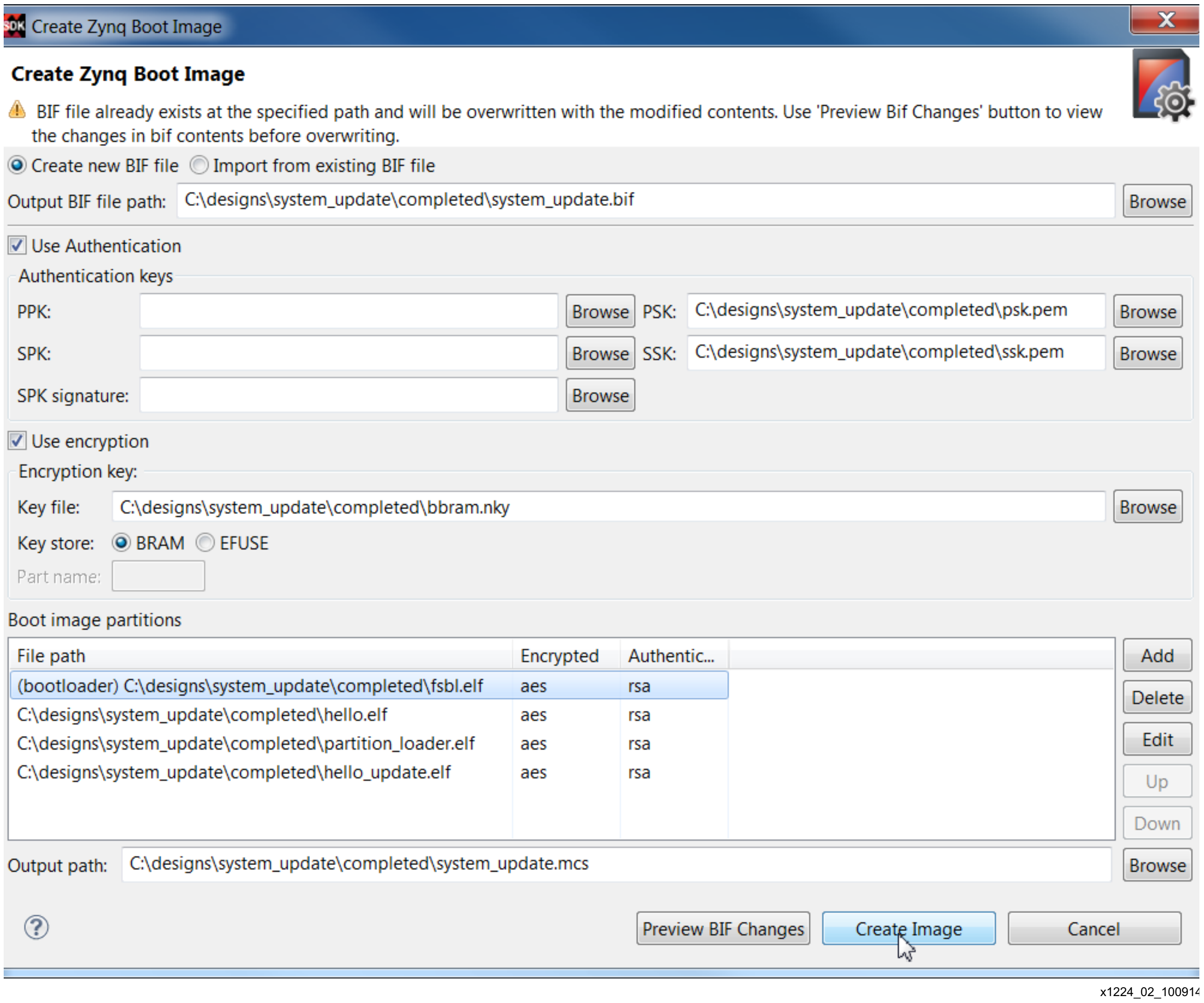
Systems remotely updated over a network are subject to *man in the middle* attacks. Xilinx provides AES encryption to protect the confidentiality and RSA authentication to protect the integrity of sensitive information.

At the factory, Bootgen is used to create an update partition which is AES encrypted and/or RSA signed. Bootgen can create an image with all partitions for boot or a single partition for run time update. The use of security attributes in the Bootgen Image Format (BIF) file is discussed in *Secure Boot of Zynq-7000 All Programmable SoC* (XAPP1175) [Ref 1].

The Partition Loader supports security using an argument in the `PcapLoadPartition` and `PcapDataTransfer` functions. U-Boot supports security with the `zynqaes` and `zynqrsa` commands. The reference system shows the use of security when using the Partition Loader and U-Boot.

In the embedded system, Zynq SoC provides a hard AES decryptor and soft RSA libraries for decryption and verification of partitions.

The difference between the non-secure mode and the secure mode is the BIF file. Figure 4 shows a BIF for a secure system update. Provide AES and RSA keys as shown in the figure.



Create Zynq Boot Image

Create Zynq Boot Image

⚠ BIF file already exists at the specified path and will be overwritten with the modified contents. Use 'Preview Bif Changes' button to view the changes in bif contents before overwriting.

● Create new BIF file ○ Import from existing BIF file

Output BIF file path: C:\designs\system_update\completed\system_update.bif

☑ Use Authentication

Authentication keys

PPK:		PSK:	C:\designs\system_update\completed\psk.pem
SPK:		SSK:	C:\designs\system_update\completed\ssk.pem
SPK signature:			

☑ Use encryption

Encryption key:

Key file: C:\designs\system_update\completed\bbram.nky

Key store: ● BRAM ○ EFUSE

Part name:

Boot image partitions

File path	Encrypted	Authentic...
(bootloader) C:\designs\system_update\completed\fsbl.elf	aes	rsa
C:\designs\system_update\completed\hello.elf	aes	rsa
C:\designs\system_update\completed\partition_loader.elf	aes	rsa
C:\designs\system_update\completed\hello_update.elf	aes	rsa

Output path: C:\designs\system_update\completed\system_update.mcs

Figure 4: **BIF for Secure System Update**

Reference Design

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

Table 1 shows the reference design matrix.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Lester Sanders
Target devices	Zynq All Programmable 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	No
Test bench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado synthesis
Implementation software tools/versions used	Vivado implementation
Static timing analysis performed	No
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ZC702

The reference design uses Quad Serial Peripheral Interface (Quad-SPI) as the nonvolatile memory (NVM). Quad-SPI is provided on all Zynq-7000 evaluation boards. Quad-SPI is commonly used on production boards. The system update methods used with Quad-SPI can be used with NAND, NOR, or SD NVM.

Conclusion

The hardware and software programmability of Zynq All Programmable SoC can provide distinct advantages during the life cycle of the embedded system. The ability of a user or device manager to update a system during run time increases the functionality, availability, and reliability of the embedded system. Two methods, a Partition Loader and U-Boot, are provided which can be used to update a system while a Zynq device is operational.

References

1. *Secure Boot of Zynq-7000 All Programmable SoC* (XAPP1175)
2. *Run Time Integrity and Authentication Check of Zynq-7000 All Programmable SoC System Memory* (XAPP1225)
3. www.denx.de/wiki/U-Boot
4. www.wiki.xilinx.com/Build+U-Boot#Zynq
5. *ZC702 Evaluation Board for Zynq-7000 XC7020 AP SoC* (UG850)
6. *Zynq-7000 All Programmable Software Developers Guide* (UG821)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/22/2014	1.0	Initial Xilinx release.
02/12/2015	1.1	Replaced Figure 2 and Figure 3. Changed address in step 15b from `0x00400000` to `0x200000` (in two places). Changed address in step 15d from `0x400000` to `0x300000`. Changed address in step 8 from `0x600000` to `0x200000`. Modified third sentence under Test of System Update (paragraph following step 2) by deleting "and Handoff Address `0x00600000`".

Please Read: Important Legal Notices