XAPP1225 (v1.0.1) October 24, 2014

Run Time Integrity and Authentication Check of Zynq-7000 AP SoC System Memory

Author: Lester Sanders

Summary

This application note provides a run time integrity checker (RTIC) for the Zynq®-7000 All Programmable SoC. The RTIC software runs on Zynq-7000 AP SoC devices and detects if there are any changes to external memory. When run periodically, the RTIC provides an effective means to ensure that nonvolatile or Double Data Rate (DDR) memory, two relatively vulnerable system components, have not been modified.

Included Systems

The `xapp1225-rtic` reference design contains the following software projects.

- fsbl
- hello_world
- hello
- rtic

The ELF/BINs of these software projects, cryptographic keys, and Bootgen Image Format (BIF) file are included in the `xapp1225-rtic/completed` directory. The source files are provided in the `hello_src` and `rtic_src` directories.

Introduction

A run time integrity checker is critical in the overall security of an embedded system. Modifications to embedded system memory can occur at boot and run time. When the security techniques described in *Secure Boot of Zynq-7000 All Programmable SoC* (XAPP1175) [Ref 1] are used, Zynq-7000 AP SoC devices are booted into a trusted state. Zynq-7000 AP SoC devices provide a high level of integration within their security perimeters. The external memory connected to a Zynq-7000 AP SoC device is a target when left unprotected. The RTIC ensures the security of external memory during operation, providing a necessary complement to the secure boot functionality.

This application note contains the following sections:

- RSA Authentication in Zynq-7000 AP SoC devices reviews Rivest, Shamir, Adleman (RSA) authentication used in Zynq-7000 AP SoC devices. The *Secure Boot of Zynq-7000 All Programmable SoC* application note (XAPP1175) [Ref 1] provides a detailed description of using RSA in Zynq-7000 AP SoC devices.
- RTIC Application discusses when the RTIC should be used and the type of information which should be checked for integrity.
- Developing the RTIC Reference Design shows how to create the RTIC. This section assumes Xilinx Software Development Kit (SDK) experience.
- RTIC System Test section shows how to run and review the results of the RTIC system on the ZC702 Evaluation Platform.
- Security Considerations in Using the RTIC discusses methods to run the RTIC securely.

Hardware and Software Requirements

The hardware requirements for updating and/or authentication of Zynq-7000 AP SoC devices are:

- ZC702, ZC706, Zed, or MicroZed evaluation boards
- AC power adaptor (12 VDC)
- USB Type-A to Micro-B cable
- Xilinx Platform Cable USB II
- Xilinx Vivado® Design Suite 2014.3
- Xilinx Software Development Kit 2014.3

RSA Authentication in Zynq-7000 AP SoC devices

RSA is public key cryptography used in Zynq-7000 AP SoC devices to ensure that an adversary has not modified software, data, or hardware (bitstream). RSA uses a private/public key pair for encryption and authentication. In RSA cryptography, the private key is not used in the embedded system. The Xilinx RSA library is used in secure boot and in the run time integrity check.

In RSA authentication, the SDK image writer (Bootgen) signs the partition(s) using the private key at the factory. The first stage boot loader (FSBL) and/or U-Boot use the RSA Library and the public RSA key(s) and signature(s) to authenticate the partition(s). For secure boot, this procedure is described in the *Secure Boot of Zynq-7000 All Programmable SoC* (XAPP1175) [Ref 1], which also includes details on generating and using RSA private and public keys, and the use of the Authentication Certificate(s) in the integrity check. The authentication methods used in secure boot are re-used in the RTIC.

RTIC Application

To detect tamper events on external memory during operation, the RTIC is run after boot. It can be started by the FSBL or a scheduler and run periodically uninterrupted until a system reset or power down. The frequency of the RTIC operation is user defined. Optimally, the integrity test is bound to the code which is to be executed, as this eliminates a "time of use - time of check" issue. If a compromise is detected quickly, corrective action might minimize the damage.

The RTIC checks a single partition, which is software, data, or hardware (bitstream). The partition can be plain text or encrypted. The RTIC tests data in external memory, NVM or DDR, which is read only, static data. The most common integrity check is on software (code). While the code in NVM/DDR is read/writable, an integrity check can only be done if the content does not change and is not relocated.

The bitstream in configuration memory cannot be authenticated. It can be authenticated when resident in NVM or DDR. The bitstream can then be copied to configuration memory. Since configuration memory resides within the security perimeter of a Zynq-7000 AP SoC device, it is much less accessible to an adversary than external memory. An SEU check can be done on configuration memory.

Developing the RTIC Reference Design

In the reference design, the `rtic` project checks the integrity of the `hello` partition. The `hello` partition is located at `0x20000000` in DDR.

Figure 1 provides the steps to build the RTIC reference design. The first two steps, generating and programming the RSA keys, are discussed in *Secure Boot of Zynq-7000 All Programmable SoC* (XAPP1175) [Ref 1]. The `fsbl`, `hello_world`, `hello`, and `rtic` software projects are created. SDK provides application project templates for all projects. The templates for `hello`, `hello_world`, and the `rtic` projects are modified to enable the integrity check.

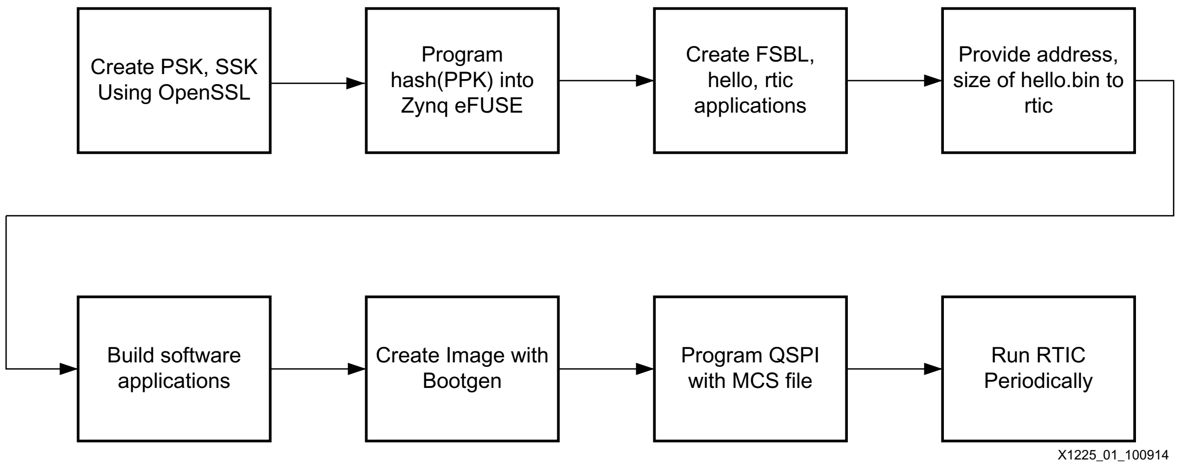

Create PSK, SSK Using OpenSSL
Program hash(PPK) into Zynq eFUSE
Create FSBL, hello, rtic applications
Provide address, size of hello.bin to rtic
Build software applications
Create Image with Bootgen
Program QSPI with MCS file
Run RTIC Periodically
X1225_01_100914

Figure 1: **Runtime Integrity Check Flow**

In the reference design, the FSBL loads the `hello_world`, `hello`, and `rtic` software projects. The `rtic` periodically checks the contents of hello.

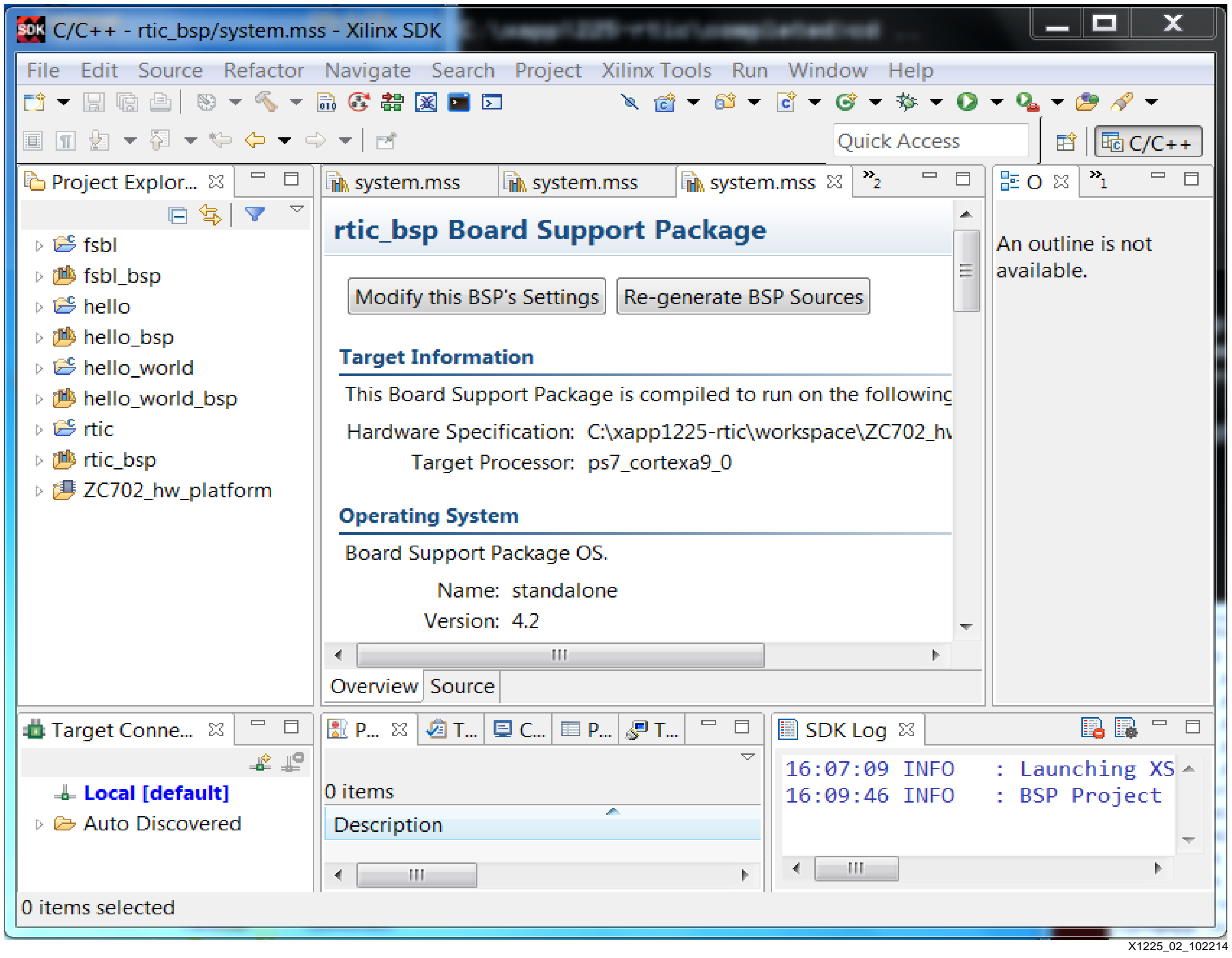
To check the integrity of `hello`, `rtic` needs address and size information of the `hello` partition. Bootgen is used to create a standalone `hello.bin` partition with an authentication certificate. The address and size of the partition and the authentication certificate are determined from `hello.bin`. These parameters are input into the `rtic` project to enable the integrity check.

SDK is used to create the `fsbl`, `hello_world`, `hello`, and `rtic` software project templates.

1. Create the `xapp1225-rtic` directory.
2. Run the following commands to create a FSBL software project.
    a. **File > New > Application Project**
    b. Type `fsbl` in the Project Name text box.
    c. Select **ZC702_hw_platform(predefined)** as the hardware platform under Target Hardware
    d. Click **Next**
    e. Select **Zynq FSBL**
    f. Click **Finish**
3. Right-click `fsbl`, and select **C/C++ Build Settings**.
4. Click **Symbols**. Click '**+**' in the **Defined Symbols** pane.
5. Enter **RSA Support** in the text box. Click **OK**, **Apply**, **OK**.
6. Repeat the steps used to create `fsbl` to create the `hello` and `hello_world` software projects, selecting **Hello World** in the Templates window for both projects.
7. Edit the `hello_world` project by copying `helloworld.c` and `handoff.S` from `xapp1225-rtic/hello_world_src` to the src area. With this edit, the CPU runs the `rtic` software after `hello_world` is finished.
8. Repeat the steps used to create the `fsbl` project to create the `rtic` software project, selecting **RSA Authenticate App** in the Templates window.
9. After the `rtic` project builds, right-click the **rtic_bsp** in the Project Explorer window, select **Board Support Package Settings**, check **xilrsa**, and click **OK**.
10. Click **Finish**.

In the Project Explorer pane, the `rsa_auth_app.c` and `rsa_auth_app.h` files are placed in the `src` directory under `rtic`.

The Project Explorer pane in Figure 2 shows the `fsbl`, `hello_world`, `hello`, and `rtic` software projects.



rtic_bsp Board Support Package

Target Information

This Board Support Package is compiled to run on the following

Hardware Specification: C:\xapp1225-rtic\workspace\ZC702_hw

Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone

Version: 4.2

An outline is not available.

```
16:07:09 INFO   : Launching XS
16:09:46 INFO   : BSP Project
```

Figure 2: **SDK with FSBL, Hello, Hello_World and RTIC Projects**

Edit the `hello` and `rtic` projects as outlined below.

The SDK tool Bootgen is used twice in this project. When an image is created for the boot stage, the Bootgen Image Format (BIF) input into Bootgen includes the FSBL, bitstream, and all software partitions. Bootgen generates a single file image, in either BIN or MCS format.

In the case of RSA authentication of the `hello` partition, the second use of Bootgen is to create an image which contains only the `hello` partition. As noted, this is to extract location and size parameters from `hello.bin`.

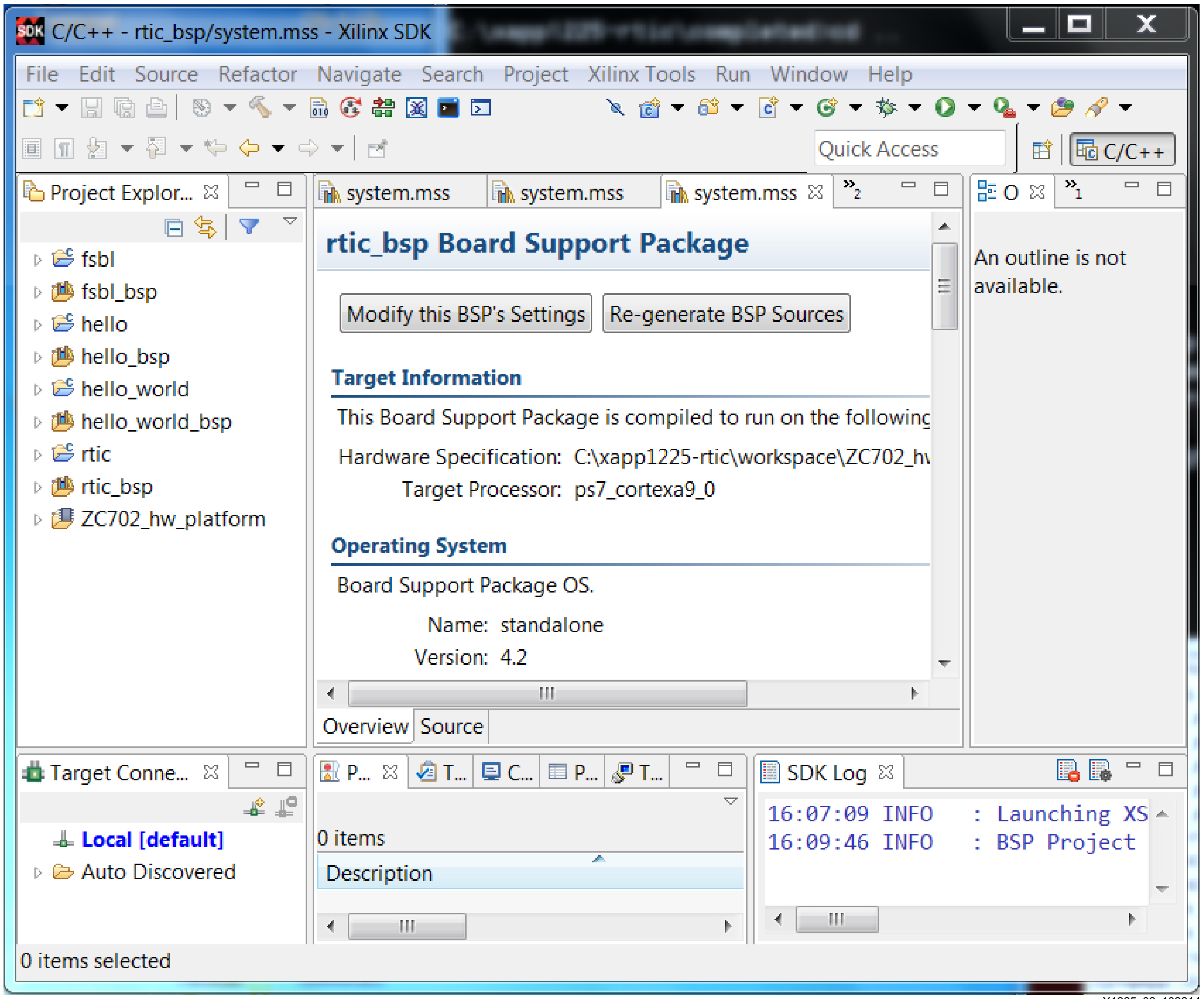
1. In SDK, select **Xilinx Tools > Create Zynq Boot Image**.
2. In the Create Zynq Boot Image window, browse to `xapp1225-rtic` and type **`hello.bif`** for the BIF File path.
3. Click **Use Authentication**.

4. In the **Authentication Keys** section, browse to `psk.pem` for the **PSK** entry. Sample Primary Secret Key (PSK), Secondary Secret Key (SSK) keys and the hash of the Primary Public Key (PPK) are provided in the `xapp1225-rtic/completed` directory. Browse to `ssk.pem` for the **SSK** entry.
5. Click **Add**.
6. In the Add Partition window, browse to `xapp1225-rtic/hello/Debug/hello.elf` to enter the File path.
7. Select **datafile** as the Partition type.
8. Select **rsa** as Authentication.
9. For the Output Path entry, browse to the `xapp1225-rtic/hello` and enter `hello.bin`.
10. Click **Create Image**.
11. As shown in Figure 3, open `hello.bin` in a text editor such as Textpad.
12. Search for three values starting at address `C80`:

    Partition Start Address - `C0 05 00 00` (second word on C90 line)

    Partition Size - `03 20 00 00` (second word on C80 line)

    Authentication Certificate (AC) Start Address - `D0 25 00 00` (third word on CA0 line)



rtic_bsp Board Support Package

Target Information

This Board Support Package is compiled to run on the following

Hardware Specification: C:\xapp1225-rtic\workspace\ZC702_hv

Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone

Version: 4.2

An outline is not available.

```
16:07:09 INFO    : Launching XS
16:09:46 INFO    : BSP Project
```

Figure 3: **Locating the Partition Start Address, Size, and AC Start Address**

13. Calculate the size value using hex multiplication. Byte swapping is required for this operation.

    a. After byte swapping, the value `0xC0050000` translates to `0x5C0`. Since there are four bytes per word, multiply by 4 to get `0x1700`.

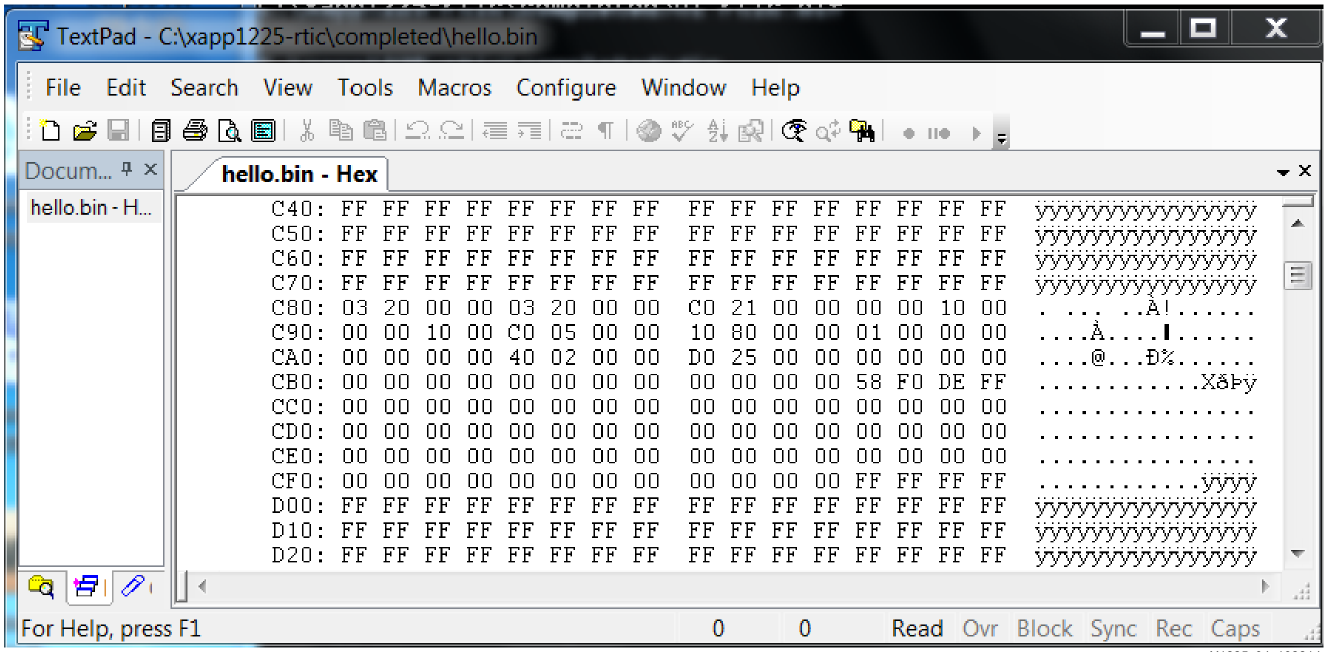
    b. As an example, to calculate the start address using python, invoke python, and enter the following commands.

```
int("5C0",16)
1472
1472 * 4
5888
hex(5888)
0x1700
```

The values calculated are given below:

PARTITION START ADDRESS - `0x1700`
PARTITION SIZE - `0x8700`
AUTHENTICATION CERTIFICATE START ADDRESS - `0x9740`

14. As shown in Figure 4, open `rsa_auth_app.h`, and provide the values just calculated to the #define(s). When the authentication test is run, the image is loaded at an offset of `0x2000000`. Add the `0x20000000` base address to the APPLICATION START ADDRESS and CERTIFICATE_START_ADDRESS.



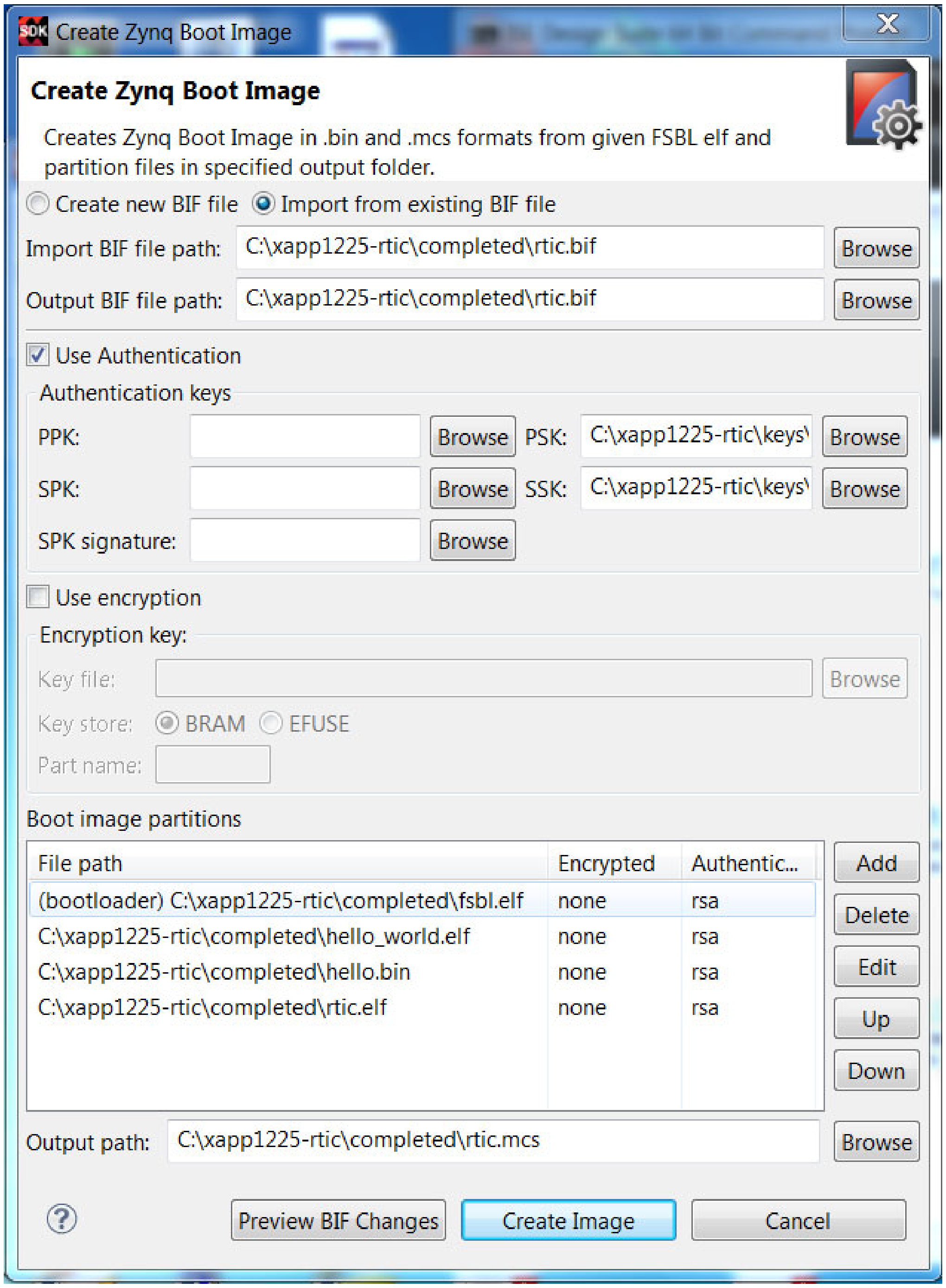
```
C40: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
C50: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
C60: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
C70: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
C80: 03 20 00 00 03 20 00 00  C0 21 00 00 00 00 10 00  . ... ..À!......
C90: 00 00 10 00 C0 05 00 00  10 80 00 00 01 00 00 00  ....À....▌......
CA0: 00 00 00 00 40 02 00 00  D0 25 00 00 00 00 00 00  ....@...Ð%......
CB0: 00 00 00 00 00 00 00 00  00 00 00 00 58 F0 DE FF  ............XðÞÿ
CC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
CD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
CE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
CF0: 00 00 00 00 00 00 00 00  00 00 00 00 FF FF FF FF  ............ÿÿÿÿ
D00: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
D10: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
D20: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

Figure 4: **Updating Parameters in rsa_auth_app.h**

15. Highlight the **rtic** project, right-click, and click **Build Project**.
16. To use Bootgen to create the image for the `xapp1225-rtic` project, enter **Xilinx Tools > Create Zynq Boot Image**
17. To add the four partitions, follow the steps used for the `hello.bin` partition.

Alternately, click **Import from existing BIF file** and browse to `xapp1225-rtic/completed/rtic.bif`.

18. With the Bootgen content as shown in Figure 5, click **Create Image** to write `rtic.mcs`.



Create Zynq Boot Image

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:\xapp1225-rtic\completed\rtic.bif

Output BIF file path: C:\xapp1225-rtic\completed\rtic.bif

[✓] Use Authentication

Authentication keys

PPK:
PSK: C:\xapp1225-rtic\keys\
SPK:
SSK: C:\xapp1225-rtic\keys\
SPK signature:

[ ] Use encryption

Encryption key:

Key file:

Key store: [✓ BRAM] EFUSE

Part name:

Boot image partitions

File path	Encrypted	Authentic...
(bootloader) C:\xapp1225-rtic\completed\fsbl.elf	none	rsa
C:\xapp1225-rtic\completed\hello_world.elf	none	rsa
C:\xapp1225-rtic\completed\hello.bin	none	rsa
C:\xapp1225-rtic\completed\rtic.elf	none	rsa

Output path: C:\xapp1225-rtic\completed\rtic.mcs

Figure 5: **Use Bootgen to Create RTIC Image**

RTIC System Test

This section shows two methods to run a system test of the `xapp1225-rtic` reference design. The simplest test uses XMD commands. The second test runs the reference design from Quad SPI flash memory.

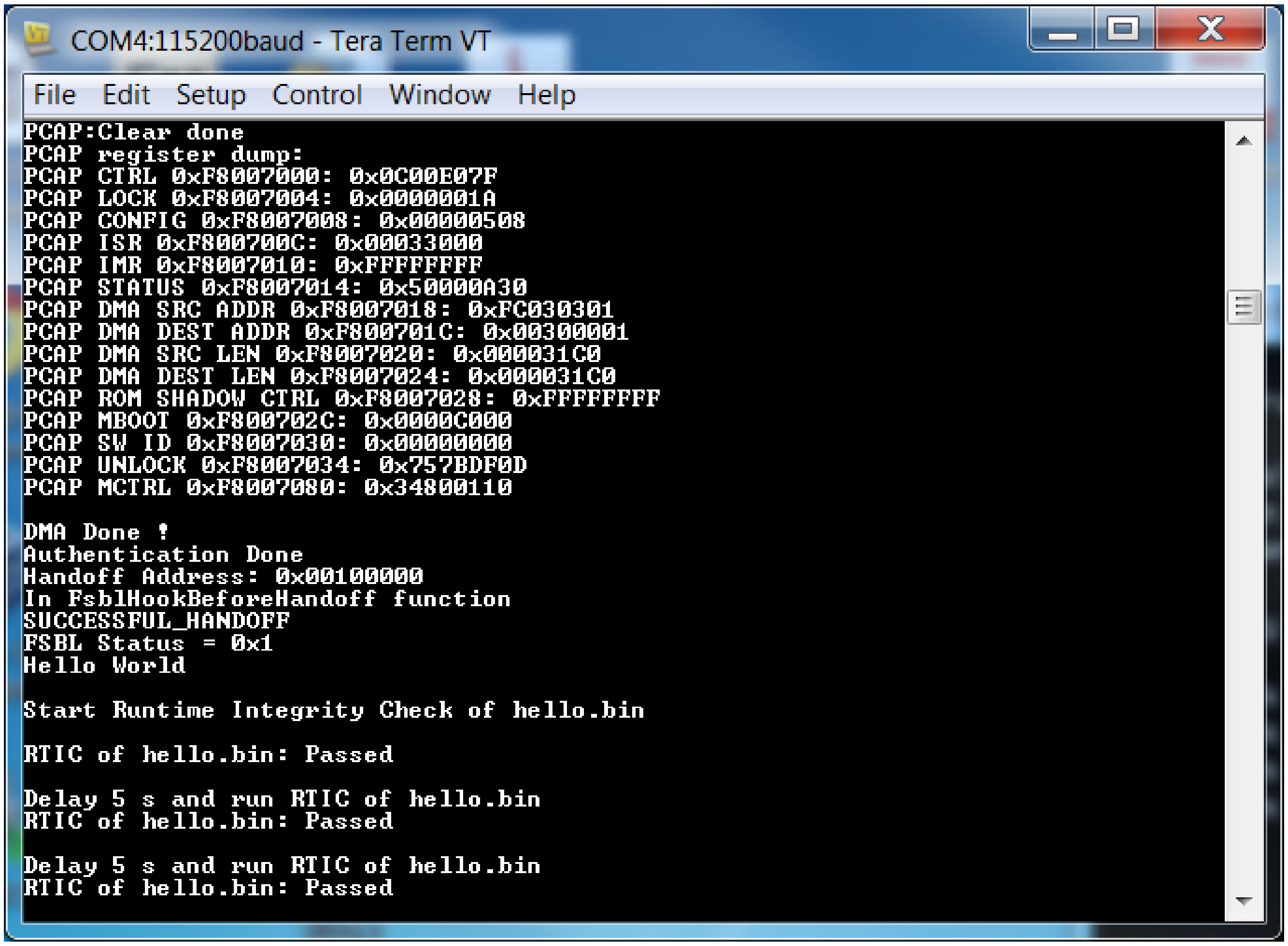
1. To view the system test results, set up a communication terminal such as Tera Term using a 115,200 baud rate.
2. To run the simple test, invoke XMD and run the following commands

```
connect arm hw
dow fsbl.elf
run
stop
dow -data hello.bin 0x20000000
dow rtic.elf
con
```

3. To run the reference design from Quad SPI flash memory, set the ZC702 Mode Select pins to JTAG mode.
4. From SDK, click **Xilinx Tools > Program Flash**
5. To enter the **Image File** text box, browse to `xapp1225-rtic/completed/rtic.mcs`.
6. Type **`0x0`** in the **Offset** text box.
7. Click **Program**
8. On the ZC702 board, change the Mode Select pins to select the Quad SPI flash memory mode.
9. Power cycle the board.

Figure 6 shows the tail of the expected output from the system test. A RTIC is run periodically. In the system test, it is run three times, in five second intervals.

The complete `rtic.log` file is provided in the `xapp1225-rtic/completed` directory. The FSBL is compiled with the FSBL_DEBUG_INFO Build setting, so detailed information is provided on the loading of the partitions. The log file contains address location, size, and authentication information on each partition. Following this, the system test is run three times.


COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
PCAP:Clear done
PCAP register dump:
PCAP CTRL 0xF8007000: 0x0C00E07F
PCAP LOCK 0xF8007004: 0x0000001A
PCAP CONFIG 0xF8007008: 0x00000508
PCAP ISR 0xF800700C: 0x00033000
PCAP IMR 0xF8007010: 0xFFFFFFFF
PCAP STATUS 0xF8007014: 0x50000A30
PCAP DMA SRC ADDR 0xF8007018: 0xFC030301
PCAP DMA DEST ADDR 0xF800701C: 0x00300001
PCAP DMA SRC LEN 0xF8007020: 0x000031C0
PCAP DMA DEST LEN 0xF8007024: 0x000031C0
PCAP ROM SHADOW CTRL 0xF8007028: 0xFFFFFFFF
PCAP MBOOT 0xF800702C: 0x0000C000
PCAP SW ID 0xF8007030: 0x00000000
PCAP UNLOCK 0xF8007034: 0x757BDF0D
PCAP MCTRL 0xF8007080: 0x34800110

DMA Done !
Authentication Done
Handoff Address: 0x00100000
In FsblHookBeforeHandoff function
SUCCESSFUL_HANDOFF
FSBL Status = 0x1
Hello World

Start Runtime Integrity Check of hello.bin

RTIC of hello.bin: Passed

Delay 5 s and run RTIC of hello.bin
RTIC of hello.bin: Passed

Delay 5 s and run RTIC of hello.bin
RTIC of hello.bin: Passed
```

X1225_06_100914

Figure 6: **RTIC of hello.bin**

Security Considerations in Using the RTIC

The reference design shows the basics of how to run the RTIC periodically. In an actual system, the period should be unending, and additional security measures should be used. When not used, the RTIC should be located in system memory, either in NVM or DDR. When the RTIC is scheduled to run, the RTIC should be authenticated.

The RTIC should run in the most privileged mode from OCM. If Trustzone is used, the RTIC should reside in the secure world.

Conclusion

This application note provides a method to check that there have not been any modifications to sections of system memory which contain code and/or static data. The run time integrity checker provides an effective method of detecting an attack on system memory which might occur during operation.

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-7000 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	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
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ZC702, ZC706, or Zed evaluation board

References

1. *Secure Boot of Zynq-7000 All Programmable SoC* (XAPP1175)
2. *ZC702 Evaluation Board for Zynq-7000 XC7020 AP SoC*(UG850)
3. *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821)

4. *Secure System Update of Zynq-7000 All Programmable SoC* (XAPP1224)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/23/2014	1.0	Initial Xilinx release.
10/24/2014	1.0.1	Corrected board number in [Ref 2] to ZC702.

Please Read: Important Legal Notices

© Copyright 2014 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.