Summary

The Zynq® UltraScale+™ MPSoC family is based on the Xilinx® UltraScale™ MPSoC architecture. The Zynq UltraScale+ RFSoCs are similar to the basic MPSoCs with the addition of key RF subsystems for multi-band, multi-mode cellular radios, and cable infrastructure. These MPSoC and RFSoC products integrate a feature rich 64-bit quad-core or dual-core Arm® Cortex™-A53 and dual core Arm Cortex R5 based processing system (PS) and Xilinx programmable logic (PL) UltraScale™ architecture in a single device. Nonvolatile memory (NVM) in the form of eFUSEs and battery-backed RAM (BBRAM) is used for advanced encryption standard (AES) keys, Rivest-Shamir-Adleman (RSA) key hashes, security control, and user-defined applications. This application note describes the programming of BBRAM and eFUSEs in Zynq UltraScale+ devices. The capability to program BBRAM and eFUSEs increases the field programmability of Xilinx FPGAs and SoCs. The programming of BBRAM and eFUSEs in Zynq UltraScale+ devices provides ease-of-use and security advantages over the programming capabilities of the Zynq-7000 SoC and UltraScale devices.

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

Introduction

The BBRAM and eFUSEs in Zynq UltraScale+ devices are principally used to store AES keys and the hashes of RSA keys. Self-programming of BBRAM and eFUSEs in Zynq UltraScale+ devices does not require an IP core. Software running on the Arm® Cortex-A53 or Cortex-R5 processor uses the Xilinx Secure Key (XiLSKey) library to program the BBRAM or the eFUSEs. Example C code is provided with the XiLSKey library to program the BBRAM or eFUSEs.

This application note provides two methods to program the BBRAM and eFUSEs:

• Create and run software built in the Vitis™ software platform to program the BBRAM and eFUSEs
• Use Xilinx Lightweight Provisioning (XLWP) command line tool to program the BBRAM and eFUSEs

Along with instructions to program BBRAM and eFUSEs, steps are provided to create and run an application software project in Vitis to test the cryptographic functionality, enabled by programmed memory. For example, run the `zcu102_program_bbram` software project to program the BBRAM, and then test the functionality using the standard hello_world software project.
The uses of programming the BBRAM and eFUSEs are listed as follows:

- AES Key
- RSA Support
- Security Control Support
- Physically Uncloneable Function (PUF) Support
- User Defined eFUSEs

For a complete list of programmable eFUSEs, see the *Zynq UltraScale+ MPSoC: Technical Reference Manual* (UG1085) [Ref 2].

Programming BBRAM and eFUSEs is a prerequisite for the secure boot functionality discussed in the *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [Ref 1].

**Note:** This application note applies to eFUSEs located in the processor system (PS) of Zynq UltraScale+ MPSoC/RFSoC devices, not the eFUSEs located in the programmable logic (PL). For details on programming the eFUSEs located in the programmable logic (PL) of Zynq UltraScale+ devices, see *Internal Programming of BBRAM and eFUSEs* (XAPP1283) [Ref 3].

## Hardware and Software Requirements

The hardware and software requirements for the reference systems are as follows:

- ZCU102 evaluation board or Avnet UltraZed-EG board
- AC power adapter (12 VDC)
- USB type-A to USB mini-B cables (for UART, JTAG communication)
- SD-Card (FAT32)
- Xilinx Vitis software platform 2020.1 or newer

**IMPORTANT:** Programming any of the noted eFUSE settings preclude Xilinx test access. Consequently, Xilinx may not accept return material authorization (RMA) requests. The eFUSEs are ENC_ONLY, JTAG_DIS, DFT_DIS, RSA_EN, and AES key. For further details, refer to Table 12-13 in the *Zynq UltraScale+ MPSoC: Technical Reference Manual* (UG1085) [Ref 2].

## Programming BBRAM and eFUSEs Using XilSKey Library

The software projects in Vitis uses the XilSKey library to program the BBRAM and eFUSEs. The XilSKey library is located at `<Vitis_install>/data/embeddedsw/lib/sw_services`.

The XilSKey library provides examples for programming Zynq UltraScale+ device eFUSEs and BBRAM. The high-level steps using the XilSKey library are explained in the following sections:
BBRAM and eFUSE Uses

This section describes the use cases of programming the BBRAM and eFUSEs in Zynq UltraScale+ devices:

AES Key

Zynq UltraScale+ devices use a hardened AES cryptographic block for AES encryption and decryption. The AES cryptographic block accepts keys from several sources to include storing a 256-bit AES key in BBRAM or eFUSEs. The AES key can also be stored in an obfuscated or black format either externally in flash or internally in eFUSEs. The AES key cannot be read out of BBRAM or eFUSEs once it has been programmed. The integrity of the key loading operation is verified using a 32-bit CRC. The Zynq UltraScale+ device also supports AES encryption and decryption using a PUF generated key, as discussed in Physically Uncloneable Function (PUF) Support. In the PUF eFUSEs mode, the CHASH, AUX data, and SYNDROME or PUF helper data are programmed into the eFUSEs.

eFUSE Uses

Additional eFUSE fields use cases are described in the following section.

RSA Support

Zynq UltraScale+ devices use silicon-based RSA and SHA3 cryptographic blocks for RSA authentication. RSA uses a 4096-bit private/public key pair. Only the 384 bit hash of the primary public key (PKK) is stored in the eFUSEs to save area on the device. The configuration security unit (CSU) reads the public key from external memory, calculates its cryptographic checksum using the SHA-3/384 engine, and compares it to the value stored in eFUSEs. For further details, you can refer to Zynq UltraScale+ MPSoC: Technical Reference Manual (UG1085) [Ref 2].

You can also program eFUSEs for revoking public keys of applications and partitions. For further details, you can refer to Key Revocation Lab (XAPP1344) [Ref 8].

Security Control Support

Zynq UltraScale+ devices provide eFUSEs that increase the security of the device. For example, some eFUSE bits can permanently disable the JTAG and design for testing (DFT) functionality of the device which is useful during the development phase. When a device moves to the production phase, disabling the JTAG and DFT ensures these ports are no longer accessible, preventing an adversary from gaining access to the hardware.
Physically Uncloneable Function (PUF) Support

The principle use of the PUF in Zynq UltraScale+ devices is black key storage. Black key storage stores the user's AES key in the eFUSEs or in the Bootheader in an encrypted format. At the time of use, the encrypted key in the eFUSEs or Bootheader is decrypted, and the resulting plaintext key is used for the encryption and decryption operation. As a secondary usage of the PUF, the PUF along with the AES engine, can be used to encrypt and decrypt user data. Refer to External Secure Storage Using the PUF (XAPP1333) [Ref 9].

The PUF registration software commands the PUF to generate values and program the eFUSEs used by the PUF. This software is included in the XilSKey library. The registration software commands the PUF to generate the following values:

- Helper data
- Black key (encrypted user key)

The values generated by the PUF registration software are programmed into the eFUSEs. In the PUF Bootheader mode, the values are included in the Bootheader by Bootgen.

The steps used to generate and program PUF values into the eFUSEs are explained in Programming PUF-related eFUSEs. The use of the PUF Bootheader mode is discussed in Secure Boot of Zynq UltraScale+ MPSoC: Embedded Design Tutorial (UG1209) [Ref 1].

User Defined eFUSEs

Zynq UltraScale+ devices provides eight 32-bit registers for user-defined eFUSEs. These eFUSEs can be written to and read from, for various user-defined functions. They can be used for enhanced secondary key revocation.

XilSKey

The XilSKey library provides a convenient structure to program the BBRAM and eFUSEs. Table 1 provides a summary of the user-defined macros used in BBRAM and eFUSE programming. These are used in the xilskey_bbramps_zynqmp.c, xilskey_efuseps_zynqmp_input.h, and xilskey_puf_registration.h files used in the example designs.

Table 1: Zynq UltraScale+ User-configurable PS eFUSE Parameters

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td></td>
</tr>
<tr>
<td>XSK_EFUSEPS_WRITE_AES_KEY</td>
<td>Commands to write the value defined in the XSK_EFUSEPS_AES_KEY macro to the AES eFUSE</td>
</tr>
<tr>
<td>XSK_EFUSEPS_AES_KEY</td>
<td>256-bit AES key for use in eFUSE</td>
</tr>
<tr>
<td>XSK_ZYNQMP_BBRAMPS_AES_KEY</td>
<td>256-bit AES key for use in BBRAM</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>XSK_EFUSEPS_AES_RD_LOCK</td>
<td>Enables the AES key cyclic redundancy check (CRC) for eFUSE key storage</td>
</tr>
<tr>
<td>XSK_EFUSEPS_AES_WR_LOCK</td>
<td>Enables write to AES eFUSEs</td>
</tr>
<tr>
<td>XSK_EFUSEPS_ENC_ONLY</td>
<td>Requires that the boot image be encrypted with eFUSE AES key, when programmed. It only applies to the encryption status and is independent of the RSA_EN.</td>
</tr>
<tr>
<td>XSK_EFUSEPS_BBRAM_DISABLE</td>
<td>Enables use of AES key from BBRAM permanently</td>
</tr>
<tr>
<td><strong>RSA</strong></td>
<td></td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK0_IS_SHA3</td>
<td>Specifies secure hash algorithm-2 (SHA-2) or SHA-3 of PPK0</td>
</tr>
<tr>
<td>XSK_EFUSEPS_WRITE_PPK0_HASH</td>
<td>Causes hash of public-private key 0 (PPK0) to be programmed into eFUSEs</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK0_HASH</td>
<td>PPK0 hash</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK1_IS_SHA3</td>
<td>Specifies SHA-2 or SHA-3 of PPK1</td>
</tr>
<tr>
<td>XSK_EFUSEPS_WRITE_PPK1_HASH</td>
<td>Causes hash of PPK1 to be programmed into eFUSEs</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK1_HASH</td>
<td>PPK1 hash</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK0_WR_LOCK</td>
<td>Enables writing to PPK0 eFUSEs permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK0_INVLD</td>
<td>Revokes PPK0 permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK1_INVLD</td>
<td>Revokes PPK1 permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PPK1_WR_LOCK</td>
<td>Enables writing to PPK1 eFUSEs permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_RSA_ENABLE</td>
<td>Forces RSA authentication during boot permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_SPK_ID</td>
<td>Selects SPK to use</td>
</tr>
<tr>
<td>XSK_EFUSEPS_WRITE_SPK</td>
<td>Enables control for SPK selection</td>
</tr>
<tr>
<td><strong>Security Control</strong></td>
<td></td>
</tr>
<tr>
<td>XSK_EFUSEPS_ERR_DISABLE</td>
<td>Prohibits error messages from being read via JTAG (ERROR_STATUS register)</td>
</tr>
<tr>
<td>XSK_EFUSEPS_JTAG_DISABLE</td>
<td>Enables JTAG. IDCODE and BYPASS are the only allowed commands.</td>
</tr>
<tr>
<td>XSK_EFUSEPS_DFT_DISABLE</td>
<td>Enables DFT boot mode permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PROG_GATE_DISABLE</td>
<td>When programmed, these fuses prohibit the PROG_GATE feature from being engaged. If any of these are programmed, the PL is always reset when the PS is reset.</td>
</tr>
<tr>
<td>XSK_EFUSEPS_SECURE_LOCK</td>
<td>When programmed, the device does not enable boundary scan (BSCAN) capability while in secure lockdown.</td>
</tr>
<tr>
<td>XSK_EFUSEPS_LBIST_EN</td>
<td>Enables logic built-in self-test (BIST) to be run during boot permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_LPD_SC_EN</td>
<td>Enables zeroization of registers in low-power domain (LPD) during boot permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_FPD_SC_EN</td>
<td>Enables zeroization of registers in full-power domain (FPD) during boot permanently</td>
</tr>
<tr>
<td>XSK_EFUSEPS_PBR_BOOT_ERR</td>
<td>Enables boot halt after a platform management unit (PMU) error permanently</td>
</tr>
</tbody>
</table>
This section provides step-by-step procedures to program a user AES key in BBRAM for the ZCU102 evaluation board. The main steps are as follows:

- Launch Vitis and create a platform project.
- Import the example BBRAM project in Vitis and generate the elf file.
- Generate the boot image and program the BBRAM.
- Test the functionality of the programmed BBRAM using a hello_world software project.

Note: These steps demonstrated here are for a Windows machine but could be performed in a Linux machine (either physical or virtual).

Launch Vitis and create a platform project:

1. Launch the Vitis IDE in Windows by using the desktop shortcut or by double-clicking the C:\Xilinx\Vitis\2020.1\bin\vitis.bat file. The Eclipse Launcher dialog box opens.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSK_PUF_PROGRAM_EFUSE</td>
<td>Programs syndrome data, CHASH, AUX, and black key into eFUSEs</td>
</tr>
<tr>
<td>XSK_PUF_IF_CONTRACT_MANUFACTURER</td>
<td>Checks whether or not RSA is enabled prior to issuing PUF commands</td>
</tr>
<tr>
<td>XSK_PUF_READ_SECUREBITS</td>
<td>Reads status of secure bits (SYN_INVALID, REGISTER_DISABLE, SYN_WRLK) for display on communication terminal</td>
</tr>
<tr>
<td>XSK_PUF_PROGRAM_SECUREBITS</td>
<td>Programs PUF secure bits (SYN_INVALID, REGISTER_DISABLE, SYN_WRLK)</td>
</tr>
<tr>
<td>XSK_PUF_SYN_INVALID</td>
<td>Invalidates the programmed helper data permanently</td>
</tr>
<tr>
<td>XSK_PUF_REGISTER_DISABLE</td>
<td>Disables PUF regeneration permanently</td>
</tr>
<tr>
<td>XSK_PUF_AES_KEY</td>
<td>Red key value used in black key generation</td>
</tr>
<tr>
<td>XSK_PUF_IV</td>
<td>User-provided initialization vector</td>
</tr>
<tr>
<td>XSK_PUF_SYN_WRLK</td>
<td>Locks PUF helper data from future programming</td>
</tr>
<tr>
<td>XSK_EFUSEPS_USER[0-7].FUSES</td>
<td>User eFUSE value</td>
</tr>
<tr>
<td>XSK_EFUSEPS_WRITE_USER[0-7].FUSE</td>
<td>Burns value in XSK_EFUSEPS_USER[0-7] to user eFUSES</td>
</tr>
<tr>
<td>XSK_EFUSEPS_USER_WRLK_[0-7]</td>
<td>Locks the corresponding user eFUSE register so it cannot be written to again</td>
</tr>
</tbody>
</table>

Table 1: Zynq UltraScale+ User-configurable PS eFUSE Parameters (Cont’d)
2. Select the workspace location (for example: D:\designs\workspace). Create the workspace folder if it is not already created.

![Eclipse Launcher](image)

*Figure 1: Vitis IDE Eclipse Launcher Dialog Box*

3. Click **Launch**. The Vitis integrated design environment (IDE) opens.

4. Click **File > New > Platform Project** to create platform project using ZCU102 Vivado Xilinx Shell Archive (XSA).
5. Enter the project name as hw_platform, when the New Platform Project dialog box opens as shown in Figure 2. Click Next.
6. Choose Create from hardware specification (XSA) in the Platform Project dialog box, and click **Next**.

![Image of Platform Project dialog box with options](image)

*Figure 3: Setting Custom Hardware*
7. Browse and select `zcu102.xsa` from the Vitis installation folder. To create the platform based on your selection, the tool automatically selects the appropriate operating system and processor.

![Hardware and Software Specification in Vitis](image)

*Figure 4: Hardware and Software Specification in Vitis*
8. Click **Finish** to create your platform project. The platform project editor opens as shown in **Figure 5**.

![Figure 5: Platform View in Vitis](image)

9. Right-click **hw_platform** and select **Platform Build > Project** to build the hardware platform.

When the platform is generated, the dialog box shows the status of platform generation and the Board Support Package settings dialog box opens.

10. Select **Board Support Package** under **psu_cortexa53_0**. The **Board Support Package** opens as shown in **Figure 6**.

![Figure 6: Board Support Package View in Vitis](image)
11. In the Board Support Package page, click **Modify BSP Settings**.

12. Select **xilskey** under Supported Libraries in **Board Support Package Settings window**, and click **OK**.

![Image of Board Support Package Settings](image)

**Figure 7:** Board Support Package Settings - Xilinx Secure Key Library
Import the example BBRAM project in Vitis and generate the elf project:

1. Select the Libraries tab under **Board Support Package** and click **Import Examples** to the right of the xilskey libraries to view the xilskey example projects.

![Board Support Package](image)

*Figure 8: Import Example Project for Xilinx Secure Key Library*

2. Select the check box for xilskey_bbramps_zynqmp_example. Click **OK**.

![Import Examples](image)

*Figure 9: Import Xilskey BBRAM Example Project*
The `xilskey_bbramps_zynqmp_example_1` project gets added to the workspace. This project contains the example C code to program a key into BBRAM and verify the key.

3. Double-click `xilskey_bbramps_zynqmp_example.c` so that it is displayed in the text window.

4. From the aes.nky file in the reference design files (see Reference Design), copy the AES key to line 41 in the `xilskey_bbramps_zynqmp_example.c` file as shown in Figure 10.

![Figure 10: xilskey_bbramps_zynqmp_example.c file](image)

5. Save the file.

6. Right-click `xilskey_bbramps_zynqmp_example_1` in the Project Explorer, and select Build Project to build the application.
Generate the boot image and program the BBKRAM fuse:

1. Right-click `xilskey_bbramps_zynqmp_example_1` and select **Create Boot Image**. The **Create Boot Image** dialog box opens with pre-populated setting as shown in Figure 11.

   ![Create Boot Image](image)

   **Figure 11:** Create Boot Image in Vitis

2. Click **Create Image** to create boot image (.Bin) file.
The console window (Figure 12) shows the boot generation command line setting and result.

![Console window showing boot generation command line setting and result](image)

**Figure 12: Create a Zynq MP Boot Image for the BBRAM AES Key**

3. Verify that the BOOT.BIN and program_aes_bbram_key.bif files are written to the specified directory D:/Designs/workspace/xilskey_bbramps_zynqmp_example_1/_ide/bootimage.

4. Use a text editor to review the program_bbram_aes_key.bif file.

5. Insert an SD card into the SD card slot of the PC. Copy BOOT.BIN to the SD card.

6. Set up the ZCU102 or UltraZed-EG evaluation board.

7. Set up a communication terminal such as Tera Term using Interface 0, 115200 baud rate, and default settings.

8. Move the SD card from the PC to the SD card slot on the ZCU102 or UltraZed-EG evaluation board. The ZCU102 evaluation board uses a standard SD card. The UltraZed-EG board uses a mini SD card.

9. Configure the board to boot in SD-Boot mode by setting switch SW6 as shown in Figure 13 below.

![SD-Boot Mode for Switch SW6](image)

**Figure 13: SD-Boot Mode for Switch SW6**
10. Apply power to the evaluation board.

11. Verify that the output on the communication terminal indicates successful programming of the BBRAM AES key (Figure 14).

Test the functionality of the programmed BBRAM by using a hello software project:
1. In the Vitis IDE, select File > New > Application Project. The New Application Project wizard welcome screen opens.

2. Click Next, and use the information in Table 2 to make your selections in the “New Application Project” wizard screen.

Table 2: New Application Project Settings for Standalone APU Application

<table>
<thead>
<tr>
<th>Wizard Screen</th>
<th>System Properties</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Select platform from repository</td>
<td>hw_platform</td>
</tr>
<tr>
<td>Application project details</td>
<td>Application project name</td>
<td>hello_world</td>
</tr>
<tr>
<td></td>
<td>System project name</td>
<td>hello_world_system</td>
</tr>
<tr>
<td></td>
<td>Target processor</td>
<td>psu_cortexa53_0</td>
</tr>
<tr>
<td>Domain</td>
<td>Domain</td>
<td>standalone on psu_cortexa53_0</td>
</tr>
<tr>
<td>Templates</td>
<td>Available templates</td>
<td>Hello World</td>
</tr>
</tbody>
</table>

The hello_world project gets created.
3. Right-click **hello_world** under hello_world_system and select **Build Project** to build the project.

4. Right-click **hello_world** once more and select **Create Boot Image**.

5. Rename the output BIF file as **test_encrypted_hello.bif** (Figure 16), in the Output BIF file path field.

![Create Boot Image](image)

**Figure 16:** Create Boot Image for hello_world
6. Click **Security** tab, then click **Encryption** tab, and check **Use Encryption**. Under Key store select BBRAM RED (Figure 17).

![Create Boot Image](image)

*Figure 17: Create Boot Image Encryption*

7. Double-click `fsbl` to open edit partition window and change the encrypted option to AES, and browse to `aes_fsbl.nky` for key file (Figure 18). Click **OK**.
Note: The file aes_fsbl.nky is available in the Reference Design file.

Figure 18: Encryption for fsbl Boot Image
8. Next, double-click the hello_world.elf, change the encrypted option to AES, and browse to `aes.nky` file as Key file as shown in Figure 19. Click **OK**.

![Figure 19: Encryption for hello_world Boot Image](image-url)
Figure 20 shows the encrypted boot image creation GUI for hello_world project.

9. Click Create Image to create a boot image (.Bin) file for hello_world.
10. Insert an SD card into the SD card slot of the PC. Copy the BOOT.BIN to the SD card.
11. Move the SD card from the PC to the SD card slot on the evaluation board.
12. Apply power to the evaluation board.
13. Verify that Hello World is displayed on the communication terminal (Figure 21).

![Figure 21: Verified Hello World](image)

**Programming eFUSEs for AES and RSA Cryptographic Functions**

The following section provides step-by-step procedures to program the AES eFUSEs and hash of the primary public key. These steps can be used to program any of the other eFUSEs as well. Additionally, it includes steps to test the functionality of the programmed eFUSEs, using a hello_world software project.

1. Follow steps from [Launch Vitis and create a platform project](#) to create a platform project.
2. Expand Libraries under **Board Support Package** and click **Import Examples** to the right of the xilskey.
3. Select the check box for xilskey_efuseps_zynqmp_example (Figure 22) in the import examples pane, and click **OK**.

![Figure 22: Import Xilskey eFUSEs Examples Project](image)
4. Expand **xilskey_efuseps_zynqmp_example_1** in the project explorer pane, and double-click **puf_registration > src > xilskey_efuseps_zynqmp_input.h** to invoke the file in the Vitis source editor (Figure 23).

This file contains the #define statements that specifies the eFUSE functionality.

![xilskey_efuseps_zynqmp_input.h File in Source Editor](image)

**Figure 23: xilskey_efuseps_zynqmp_input.h File in Source Editor**

5. Change the **xilskey_efuseps_zynqmp_input.h** as defined in **Table 3**.

<table>
<thead>
<tr>
<th>Line No</th>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>416</td>
<td>XSK_EFUSEPS_WRITE_AES_KEY TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>417</td>
<td>XSK_EFUSEPS_WRITE_PPK0_HASH TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>441</td>
<td>XSK_EFUSEPS_WRITE_PPK1_HASH TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

6. Open the supplied xilskey_efuseps_zynqmp_input.h file in the **Reference Design** files, update the following keys and hash value, and save the xilskey_efuseps_zynqmp_input.h file:
   - XSK_EFUSEPS_AES_KEY (Line 455)
   - XSK_EFUSEPS_PPK0_HASH (Line 458)
In the project explorer pane, right-click xilskey_efuseps_zynqmp_example_1, and select **Build Project**.

8. Right-click xilskey_efuseps_zynqmp_example_1, and select **Xilinx > Create Boot Image**.

9. Use default settings and **Click Create Image**.

10. Insert an SD card into the SD card slot of the PC. Copy
    <workspace>/xilskey_efuseps_zynqmp_example_1/_ide/bootimage
    /BOOT.BIN to the SD card.

11. Move the SD card from the PC to the SD card slot on the ZCU102 or UltraZed-EG evaluation board.

12. Set the **Boot Mode** switch to SD mode.

13. Apply power to the board.

14. Verify that the log in the communication terminal indicates that programming of eFUSEs works as expected.

To test the functionality of the programmed eFUSEs, create a hello_world software project:

15. In the Vitis IDE, select **File > New Application Project**. The **Create New Application** project wizard welcome screen opens.

16. Click **Next** and use the information in **Table 2** to make your selections in the **New Application Project** wizard screens.

17. Right-click **hello_world** and select **Build Project**.

18. Right-click **hello_world** and select **Create Boot Image**.
19. Rename the BIF file as test_encrypted_hello.bif in the Output BIF file path field.

20. Click Security > Encryption tab and check Use Encryption. Select EFUSE RED under key store.

21. Double-click the zynqmp_fsbl.elf and change the Encrypted option to AES, and browse to aes_fsbl.nky (Figure 18). Click OK.

22. Next, double-click the hello_world.elf and change the encrypted option to AES, and browse to aes.nky file (Figure 19). Click OK.

23. In the Create Boot Image GUI, click Create Image.

24. Insert an SD card into the SD card slot of the PC. Copy the BOOT.BIN to the SD card.

25. Move the SD card from the PC to the SD card slot on the evaluation board.

26. Apply power to the evaluation board.

27. Verify that Hello World is displayed on the communication terminal. If the RSA_EN eFUSE is programmed, every boot is required to be authenticated.

### Programming PUF-related eFUSEs

This section provides step-by-step procedure to program eFUSEs used by the PUF. In Programming eFUSEs for AES and RSA Cryptographic Functions, a red AES key is programmed in the eFUSEs. This section show the user how to program a black key in the eFUSEs.

**Note:** The users cannot run this section after Programming eFUSEs for AES and RSA Cryptographic Functions, as the eFUSEs have already been programmed with the red AES key. Also, the users who want a Black Key cannot program the AES key in the earlier section.

1. Follow steps from Launch Vitis and create a platform project:

2. Select Board Support Package under psu_cortexa53_0 and the Board Support Package page opens.

3. In the Board Support Package window, click Modify BSP Settings.

4. Select xilkskey and xilsecure under Supported Libraries and click OK.

5. Expand Libraries under Board Support Package and click Import Examples to the right of the xilkskey.

6. Select the check box for xilkskey_puf_registration in the examples for xilkskey pane, and click OK.

7. Expand xilkskey_puf_registration_1 in the project explorer pane.

8. Double-click xilkskey_puf_registration_1 > src >xilkskey_puf_registration.h in the project explorer pane, to invoke the file in the source editor.
9. Edit the xilskey_puf_registration.h as shown in Table 4.

Table 4: Cryptographic Macros in xilskey_puf_registration.h

<table>
<thead>
<tr>
<th>Line No</th>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>131</td>
<td>XSK_PUF_PROGRAM_EFUSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>145</td>
<td>XSK_PUF_AES_KEY</td>
<td>45195DE9B5B80119D8DD4E7DF032736D53CF75AD1DCE61C5BA681CFA0724E8</td>
</tr>
<tr>
<td>146</td>
<td>XSK_PUF_BLACK_KEY_IV</td>
<td>62A4B57D0F121CCB02CB8336</td>
</tr>
</tbody>
</table>

10. Save the file and exit.

11. In the project explorer, right-click the xilskey_puf_registration_1 project, and select Build Project.

12. Right-click the xilskey_puf_registration_1 project, and select Create Boot Image.

13. In the Create Boot Image GUI, ensure that Zynq MP is selected in the architecture field.

14. Use default settings and Click Create Image. Click Create Image to create boot image (.Bin) file.

15. Insert an SD card into the SD card slot of the PC. Copy
    
        D:/Designs/workspace/xilskey_puf_registration_1/ide/boot image/BOOT.bin
    
    to the SD card.

16. Move the SD card from the PC to the SD card slot on the ZCU102 or UltraZed-EG evaluation board.

17. Set the Boot Mode switch to SD mode.

18. Set up a communication terminal.

19. Power cycle the board.

20. Verify that the log displayed in the communication terminal indicates that the security control eFUSEs are programmed as expected.
Use XLWP Command Line Tool to Program the BBRAM and eFUSEs

Xilinx Lightweight Provisioning (XLWP) Tool is a simple way to program BBRAM and eFUSEs. It performs all of its functionality via JTAG. It allows an interactive user mode via JTAG UART to terminal app or via Tcl script mode. Contact your local sales team to download the XLWP tool. For windows, it is recommended to install the TeraTerm terminal program to use the XLWP tool. TeraTerm is an open-source and free terminal emulator for Windows available from Tera Term Home Page.

There are three Tcl files and one .ELF file that comprise the XLWP tool:

- **xlwp_tool.tcl**: Main XLWP Tcl that gets called by XSCT -- sets up JTAG connection downloads .elf, launches terminal -or- user_script, etc.
- **xlwp_zup_cmds.tcl**: Contains all the commands to allow reading/programming of Zynq UltraScale+ eFUSEs and load the AES key into BBRAM when running in script mode
- **xlwp_zup_script.tcl**: Template modified by user to define which ZU+ eFUSEs to read/program and the AES key to load into BBRAM when running in script mode
- **xlwp_zup_<nn>mhz.elf**: Binary files based on xilskey and xilsecure libraries that is downloaded to Zynq UltraScale+ A53_0 via JTAG - where <nn> must match the external PS_REF_CLK frequency on the customer's board

There are two batch (.bat) files and one initialization file for Windows and sample script for Linux included here. Here are the windows batch files:

- **xlwp_tool.bat**: Starts the hw_server application and then calls xlwp_tool.tcl via XSDB with command-line arguments (ensure the path setting matches your system)
- **xlwp_term.bat**: Starts the TeraTerm terminal program in user mode and passes in the hw_server hostname and TCP port number (ensure the path setting matches your system)
- **tt_jtag_uart.ini**: TeraTerm initialization file to setup proper operation with the XLWP tool via the JTAG UART over TCP
Launching XLWP Tool in Command Line Mode to Program the BBRAM

1. Double-click `xlwp_tool.bat` file to run the batch file.

2. Press any key and XLWP tool will open in TeraTerm terminal. Figure 25 shows XLWP tool running in Teraterm GUI.

3. Press 4 to load BBRAM AES key, and enter your 256-bit AES key (you can use the same one that is used in Vitis flow).

4. Press y for “Is the key correct” instruction, and press y once more for Load key into BRAM instruction.
The AES key will be programmed in BBRAM and a CRC value will be calculated. Refer to Figure 26.

5. Press any key to continue.

*Figure 26: BBRAM AES Programming Using XLWP Tool*
6. Press x to quit XLWP tool and then y for “Quit XLWP tool” instruction (Figure 27).

![Exiting XLWP Tool](image)

Figure 27: Exiting XLWP Tool

7. Close any terminal that remains open.

**Note:** You can follow similar steps and use a script to program BBRAM or eFUSE using the supplied script.

---

**Conclusion**

BBRAM in Zynq UltraScale+ devices can be used to store the 256-bit AES image decryption key. This AES key can also be stored in the eFUSEs with the important difference that it can neither be modified nor erased. Additionally, various custom use cases are also available for eFUSEs.

This application note provides a straightforward and secure method to self-program BBRAM and eFUSEs in the Zynq UltraScale+ devices. It also shows a hello_world design with an encrypted image to verify the BBRAM key.
Reference Design

Download the reference design files for this application note from the Xilinx website.

Table 5 shows the reference design matrix.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
</tr>
<tr>
<td>Target devices</td>
<td>Zynq UltraScale+ devices</td>
</tr>
<tr>
<td>Source code provided</td>
<td>Yes</td>
</tr>
<tr>
<td>Source code format</td>
<td>C</td>
</tr>
<tr>
<td>Design uses code and IP from existing Xilinx application note and reference designs or third party</td>
<td>No</td>
</tr>
<tr>
<td>Static code analysis/MISRA C</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Simulation</strong></td>
<td></td>
</tr>
<tr>
<td>Functional simulation performed</td>
<td>No</td>
</tr>
<tr>
<td>Timing simulation performed</td>
<td>No</td>
</tr>
<tr>
<td>Test bench used for functional and timing simulations</td>
<td>No</td>
</tr>
<tr>
<td>Test bench format</td>
<td>N/A</td>
</tr>
<tr>
<td>Simulator software/version used</td>
<td>N/A</td>
</tr>
<tr>
<td>SPICE/IBIS simulations</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td></td>
</tr>
<tr>
<td>Synthesis software tools/versions used</td>
<td>N/A</td>
</tr>
<tr>
<td>Implementation software tools/versions used</td>
<td>N/A</td>
</tr>
<tr>
<td>Static timing analysis performed</td>
<td>No</td>
</tr>
<tr>
<td><strong>Hardware Verification</strong></td>
<td></td>
</tr>
<tr>
<td>Hardware verified</td>
<td>Yes</td>
</tr>
<tr>
<td>Hardware platform used for verification</td>
<td>Avnet UltraZed-EG and ZCU102 evaluation boards</td>
</tr>
</tbody>
</table>

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select Help > Documentation and Tutorials.
- On Windows, select Start > All Programs > Xilinx Design Tools > DocNav.
• At the Linux command prompt, enter docnav.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

• In the Xilinx Documentation Navigator, click the Design Hubs View tab.
• On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. Zynq UltraScale+ MPSoC: Embedded Design Tutorial (UG1209)
3. Internal Programming of BBRAM and eFUSEs (XAPP1283)
4. Secure Boot of Zynq-7000 SoC (XAPP1175)
6. Changing the Cryptographic Key in Zynq-7000 SoC (XAPP1223)
7. Zynq UltraScale+ MPSoC Software Developer Guide (UG1137)
8. Key Revocation Lab (XAPP1344)
9. External Secure Storage using the PUF (XAPP1333)
10. Zynq UltraScale+ MPSoC: Embedded Design Tutorial (UG1209)
Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/23/2020</td>
<td>2.1</td>
<td>Updated the flow and design example using Vitis IDE 2020.1. Updated the Introduction section to clarify the BBRAM and eFUSEs programming options. Added Figure 10. Modified Figure 2, Figure 3, Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 11, Figure 12, Figure 13, Figure 14, Figure 16, Figure 17, Figure 18, Figure 19, Figure 20, Figure 21. Modified Table 2, Table 3, Table 4. Updated application projection creation flow for Vitis IDE 2020.1 in Test the functionality of the programmed BBRAM by using a hello software project: section.</td>
</tr>
<tr>
<td>08/31/2020</td>
<td>2.0</td>
<td>Updated for Vitis™ unified software platform. Validated for Vitis IDE 2019.2. Updates describe how to program eFUSEs in the field. Added new section Programming BBRAM and eFUSEs Using XilSKey Library. Added procedural steps in section Programming the AES Key in BBRAM. Added Figure 1, Figure 2, Figure 3, Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 11, Figure 12, Figure 14, Figure 15, Figure 16, Figure 17, Figure 18, Figure 19, Figure 20, and Figure 21 in section Programming the AES Key in BBRAM. Added procedural steps in section Programming eFUSEs for AES and RSA Cryptographic Functions. Added Figure 22, Figure 23, and Figure 25. Added Table 3. Added procedural steps in Programming PUF-related eFUSEs. Added Table 4. Added new section Use XLWP Command Line Tool to Program the BBRAM and eFUSEs. Added Launching XLWP Tool in Command Line Mode to Program the BBRAM. Added procedural steps in section Launching XLWP Tool in Command Line Mode to Program the BBRAM. Added Figure 25, Figure 26, and Figure 27.</td>
</tr>
<tr>
<td>07/26/2017</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
</tbody>
</table>
AUTOMOTIVE PRODUCTS (IDENTIFIED AS “XA” IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE (“SAFETY APPLICATION”) UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD (“SAFETY DESIGN”). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2017-2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, ISE, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.