XAPP1333 (v1.2) April 12, 2022

External Secure Storage Using the PUF

Author: Nathan Menhorn

Summary

To store data in non-volatile memory (NVM) using a Zynq® UltraScale+™ device, data must be stored externally and should be encrypted if it is confidential. All Zynq UltraScale+ devices have a built-in physically unclonable function (PUF), which can generate a cryptographically strong, device-unique encryption key that can be used in combination with the built-in advanced encryption standard (AES) cryptographic core. This key cannot be read by a user, allowing for a heightened level of key security. Only if a Zynq UltraScale+ device is provisioned to store the PUF configuration information in eFUSEs and if Rivest-Shamir-Adleman (RSA) Authentication is registered and enabled in eFUSEs, then the PUF's device-unique encryption key can be used to encrypt and decrypt user data, which can then be stored and read from external non-volatile memory. 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 PUF takes advantage of silicon variations unique to Zynq UltraScale+ devices to generate a device-unique encryption key that cannot be read by anyone, including the user. Along with generating a unique encryption key, the PUF also generates the required helper data so that the PUF can exactly regenerate the encryption key later. The details of the PUF are described in the *Zynq UltraScale+ MPSoC: Technical Reference Manual* (UG1085) [Ref 2]. Normally, the PUF's encryption key, referred to as the Key Encryption Key (KEK), is used for encrypting a user's plain-text red key so that a user's red key can be stored encrypted in black key form in either eFUSES or the boot header. The black encryption key is then decrypted using the PUF's KEK to generate the red key, which in turn is used for decrypting the boot information during secure boot. This use of the PUF is shown in the following figure.

IMPORTANT: *The PUF characterization results confirm that over the life of the device, the PUF is expected to reliably regenerate the KEK across all voltages and temperatures* ***assuming registration at a nominal voltage and temperature***.

IMPORTANT: *The* ***RSA_EN eFUSE*** *must be programmed in order to use the PUF's device-unique encryption key to encrypt and decrypt user data. Once this is programmed, Boot Header based authentication (bh_auth_enable) can no longer be used.*

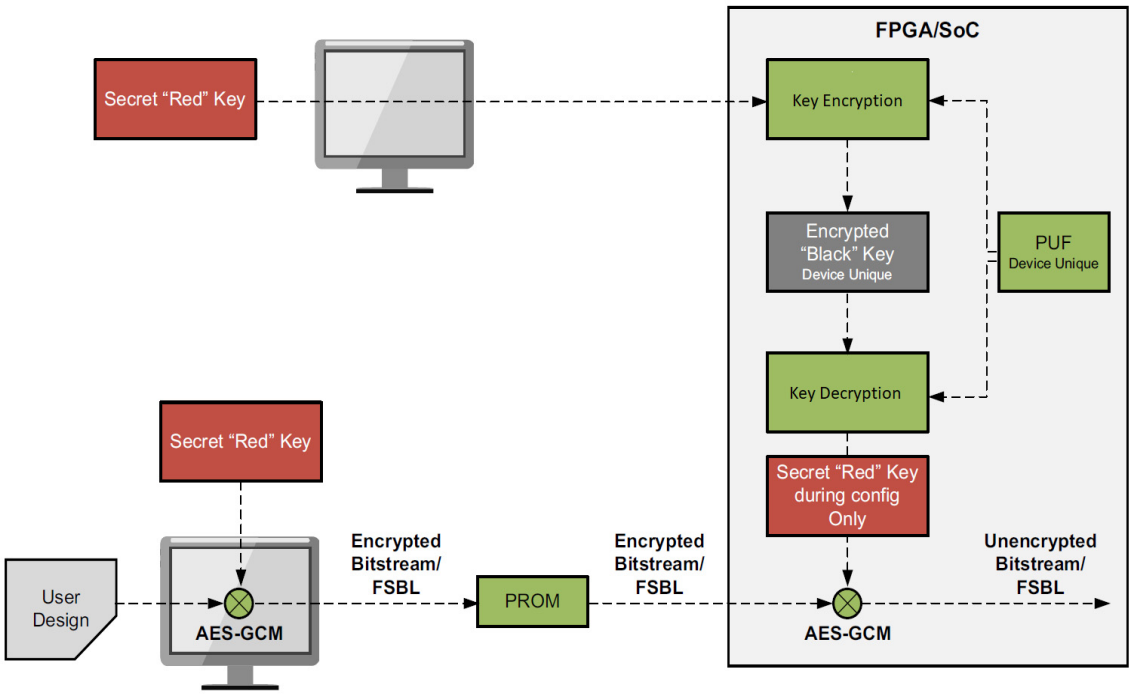


Figure 1: **Encrypting and Decrypting the Device Key Using PUF**

When the PUF is registered in eFUSEs and RSA authentication is enabled in eFUSEs, documented in *Programming BBRAM and eFUSEs* (XAPP1319) [Ref 3], the PUF's device-unique encryption key can be used to encrypt and decrypt any user data. This encrypted data can then be stored externally to the Zynq UltraScale+ device, which is the focus of this application note. The RSA authentication settings cannot be stored in the boot header when using the PUF to encrypt and decrypt user data.

IMPORTANT: *When the RSA_ENABLE eFUSEs are programmed, boot header authentication is no longer permitted.*

The process of using the PUF to encrypt user data is shown in Figure 2 and works as follows: a user generates data that must be encrypted and appends an optional ID. This optional ID can be used to validate that the correct version of data that is being used, such as when the data consists of encryption key information or configuration and is useful in preventing replay attacks. Even though the ID is optional, Xilinx highly recommends using it to ensure a more secure system. The optional ID enables key/data revocation as the user data packet can be revoked by burning one of the 256-bit user eFUSEs. Each of the 256-bit user eFUSEs can be mapped to 256 different 8-bit user IDs. Keep in mind that user eFUSEs are a shared resource as the fuses could be used for Enhanced Key Revocation software, a tamper log (see *Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices* (XAPP1323) [Ref 4], or any other user function.

Next, the PUF is enabled to regenerate the PUF's device-unique encryption key, which is loaded into the AES cryptographic core to encrypt the data. Xilinx recommends minimizing the use of the PUF's key by keeping the user data small or implementing an advanced key-rolling architecture where the PUF's device-unique key is only used to encrypt the first portion of a larger sized data, thereby minimizing its exposure. This helps to avoid differential power analysis (DPA) attacks. After the encrypted data is written to external memory, the data is read

back and decrypted to verify the process using the GCM authentication tag. If the data is authenticated, the user selected ID is safe to use. Conversely, if the data verification fails, a revocation penalty can take place, such as burning an associated user eFUSE.

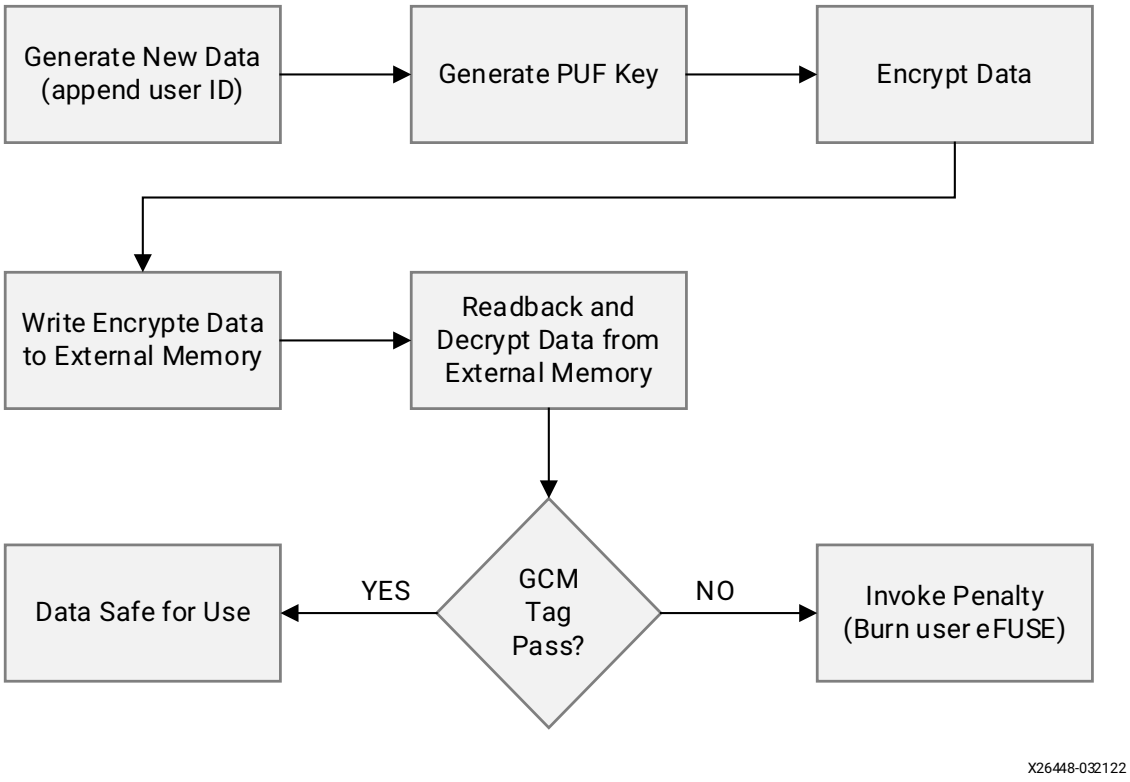


Figure 2: **Normal Encryption Process Using PUF**

Decrypting external data using the PUF is shown in Figure 3 and works as follows: the encrypted data packet is read from the external memory location followed by regeneration of the PUF decryption key. The data is then decrypted and authenticated via the GCM tag. If authentication passes and if the ID from the decrypted data has not been revoked in user eFUSEs, then the data is valid and can be used. Conversely, if the GCM tag authentication fails, then a penalty can be invoked and the decryption process could be stopped to avoid side channel attacks such as DPA. Furthermore, if the decryption process authenticates but the data's ID has been revoked in user eFUSEs, the data is invalid and should not be used.

IMPORTANT: *The PUF KEK isn't a FIPS legal key for storing data outside a cryptographic boundary. However, you can create a FIPS-legal KEK, encrypt the FIPS-legal KEK with the PUF KEK, store the encrypted FIPS-legal KEK in eFUSEs, and subsequently use the FIPS-legal KEK to store data outside the cryptographic boundary.*

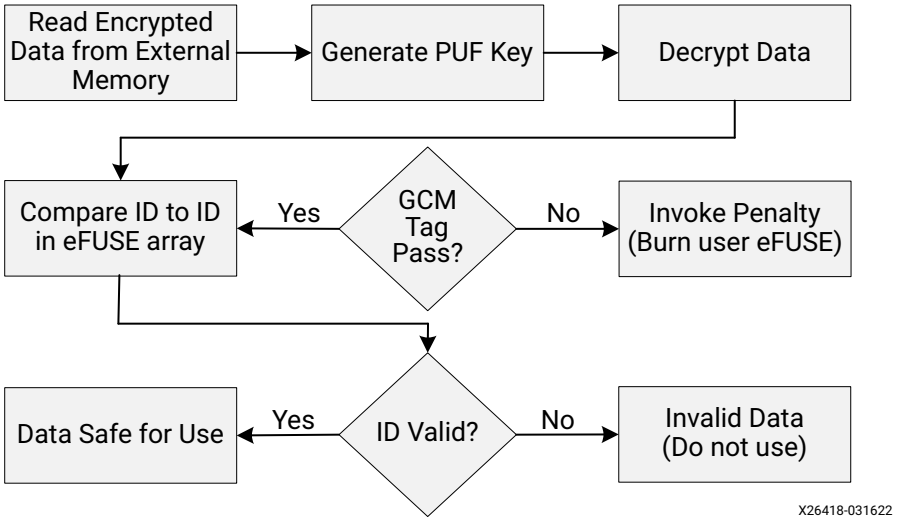


Figure 3: **Using the PUF for Decryption**

Hardware and Software Requirements

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

- ZCU102 Evaluation Board
- AC power adapter (12 VDC)
- USB type-A to USB mini-B cable x2
- Optional Platform JTAG hardware and associated cables
- Secure Digital (SD) card formatted using the FAT file system
- Xilinx Vitis™ Development Environment (Vitis IDE) 2021.2
- Required design files, which can be downloaded here.

IMPORTANT: *Programming any of the eFUSE settings noted in Table 12-13 in Zynq UltraScale+ MPSoC: Technical Reference (UG1085) [Ref 2] precludes Xilinx test access. Consequently, Xilinx may not accept return material authorization (RMA) request.*

Create a New Embedded Project for the Zynq UltraScale+ MPSoC

Perform the following steps to create a new embedded project for the Zynq UltraScale+ MPSoC. A brief description is covered in this section. Step-by-step instructions can be found in Appendix A. For detailed elaboration on each step, refer to the *UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [Ref 5] for further details.

1. Open up Vivado® Design Suite and create the hardware design required for the Zynq UltraScale+ ZCU102 Evaluation Board. The PL is not required for this lab so all the PS-PL interfaces are disabled and no bitstream is exported.
2. Export the hardware and launch Xilinx Vitis® IDE from within the Vivado Design Suite.
3. Create a platform project using the XSA file exported from Vivado. The platform projects will automatically create `ZCU102_XAPP1333` platform that includes standalone domain BSP, first stage boot loader projects called **zynqmp_fsbl** and **zynqmp_pmufw** along with their associated Board Support Packages named **zynqmp_fsbl_bsp** and **zynqmp_pmufw_bsp** running respectively on the `ARM Cortex-A53 processor` in the APU domain and TMR `MicroBaze processor` in the PMU domain.
4. Build the platform, including **zynqmp_fsbl and zynqmp_pmufw,** by right-clicking on the `ZCU102_XAPP1333` platform and select **Project -> Build Project** from the main menu.
5. Create a **HelloWorld** project to verify the hardware and software setup before proceeding.

Key Generation

Key generation is covered in detail in the Secure Boot section of *UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [Ref 5] so only a summary pertaining to this application note is documented here.

AES Key Generation

Create a new directory in the Xilinx Vitis workspace root directory (called `Keys`). The Vitis root directory can be found the same level as the `HelloWorld` folder. Generate a device key and its associated IV, an operational key, and one partition block key and its associated IV. Combine these keys and IVs into a file named `multiple_keys.nky`. Alternatively, copy the `Keys` folder found in the reference design documents to use for this lab or, if desired, use them as a template and insert your own key and IV values.

```
Device   zcu9eg;
Key 0    0123456789012345678901234567890123456789012345678901234567890123;
IV       01DBD60260A7EC34DE5F6A494;
Key Opt E070C542B6680A855724793A75222391E663CBD35F45D070F22F703A5CA31B45;
Key 1    00000001000000010000000100000001000000010000000100000001000000001;
IV 1     00000001000000100000001;
```

Encrypting the boot image is not required to use the PUF for encrypting user data. However, Xilinx highly recommends doing so, which is used throughout this application note.

IMPORTANT: *Be sure to use your own AES keys and associated IVs for operational devices. The keys provided in this lab are for demonstration purposes and are not cryptographically strong. Per the NIST Special Publication (SP) 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM), new IVs need to be used each time a key is used to encrypt new data. This means that if the boot image is updated, a new IV needs to be selected and provided to Bootgen.*

RSA Asymmetric Key Generation

For this application note, generate a pair of RSA keys called **psk0.pem** and **ssk0.pem**. Alternatively, these keys are provided in the design documents in the Keys folder. RSA authentication is required to use the PUF for encrypting and decrypting user data. While this application note does not require the use of a secondary key set, Xilinx highly recommends doing so in an operational application.

Generate SHA3 of Public RSA Asymmetric Key

Generate the associated SHA3 hash of the RSA PPK and name the output file **sha3.txt**. Alternatively, this hash can be found in the design documents in the `Keys` folder.

PUF eFUSE Configuration

IMPORTANT: *THESE INSTRUCTIONS MODIFY THE EFUSES ON THE ZCU102 DEVELOPMENT BOARD AND MAY LIMIT FUTURE USE OF THE DEVELOPMENT BOARD FOR NON-SECURE TESTING AND DEBUGGING!*

IMPORTANT: *Programming any of the noted eFUSE settings noted in Table 12-13 Zynq UltraScale+ MPSoC: Technical Reference Manual (UG1085) [Ref 2] preclude Xilinx test access. Consequently, Xilinx might not accept return material authorization (RMA) requests.* See the important note below Table 12-13 *of the Zynq UltraScale+ MPSoC: Technical Reference (UG1085) [Ref 2].*

PUF eFUSE Settings

PUF registration is covered in detail in *Using the PUF* in the *UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [Ref 5] so only a summary pertaining to this application note is documented here.

1. In the Vitis workspace for this application note, right-click on the **platform.spr** that is located under **ZCU102_XAPP1333** platform in the **Explorer** view and click **Open**.

2. Select **Board Support Package** under **standalone on psu_cortexa53_0** in the **ZCU102_XAPP1333** platform view and click **Modify BSP settings**.

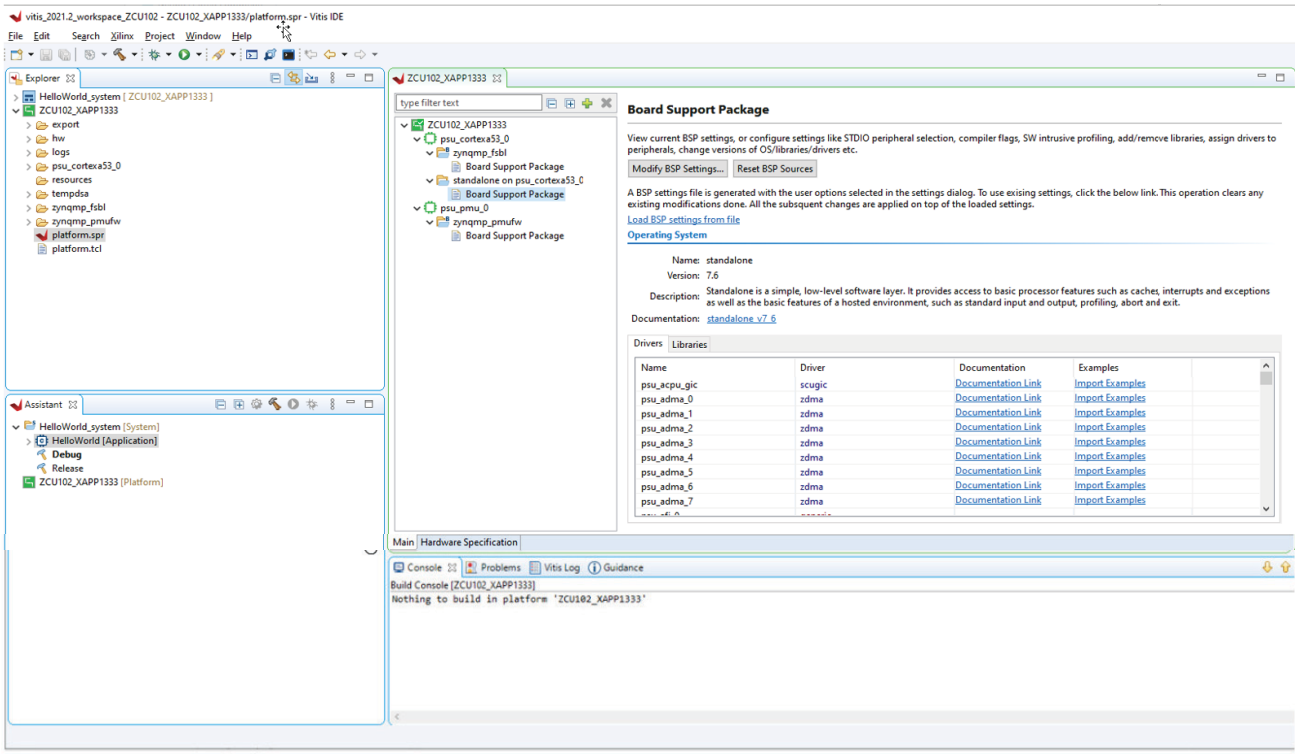


Figure 4: **Standalone BSP for PUF eFUSE registration**

3. In the **Supported Libraries**, select **xilsecure** and **xilskey**.
4. Click **OK** to close the window.
5. Right-click the **ZCU102_XAPP1333** platform in the **Explorer** view, which is now marked out-of-date, and click **Build Project**.
6. Select **Board Support Package** under **standalone on psu_cortexa53_0** in the **platform view** that just opened and select **Libraries** tab in **Operating Systems** section.
7. Scroll to the bottom of the **Libraries** tab and click **Import Examples** for the **xilskey** library.
8. Check the **xilskey_puf_registration** example and click **OK**. This adds the associated project to your workspace.
9. Open the **xilskey_puf_registration.h** file in the **src** folder under the fully expanded **xilskey_puf_registration_example_1_system** in the **Project Explorer** tab.
10. Change the definition of **XSK_PUF_INFO_ON_UART** to **TRUE**. This setting is extremely important to verify the PUF registration completed successfully.
11. Ensure the definition of **XSK_PUF_PROGRAM_EFUSE** is set to **TRUE**.
12. Change the definition of **XSK_PUF_PROGRAM_SECUREBITS** to **TRUE**.
13. Change the definition of **XSK_PUF_SYN_WRLK** to **TRUE**.
14. Set the **XSK_PUF_AES_KEY** to the **Key 0** value in the **aes_key.nky** file.

15. Set the **XSK_PUF_BLACK_KEY_IV** to a value that is user choice. This IV is not related to the IV created in **aes_key.nky** and can be any user generated value. This IV is used by encryption when encrypting the red key with the PUF's KEK.
16. Create a file named **puf_iv.txt** with the ASCII-HEX string of the PUF IV used in **XSK_PUF_BLACK_KEY_IV** as this is needed during boot. Alternatively, use the one provided in the design documents in the `Keys` folder.
17. Verify all the required changes are made before continuing as shown in the figure below. The **xilskey_puf_registration.h** file with the example keys, shown in the figure below, is included in the reference design in the `puf_registration` folder.
18. To save changes to **xilskey_puf_registration.h** click File -> Save in the main toolbar.

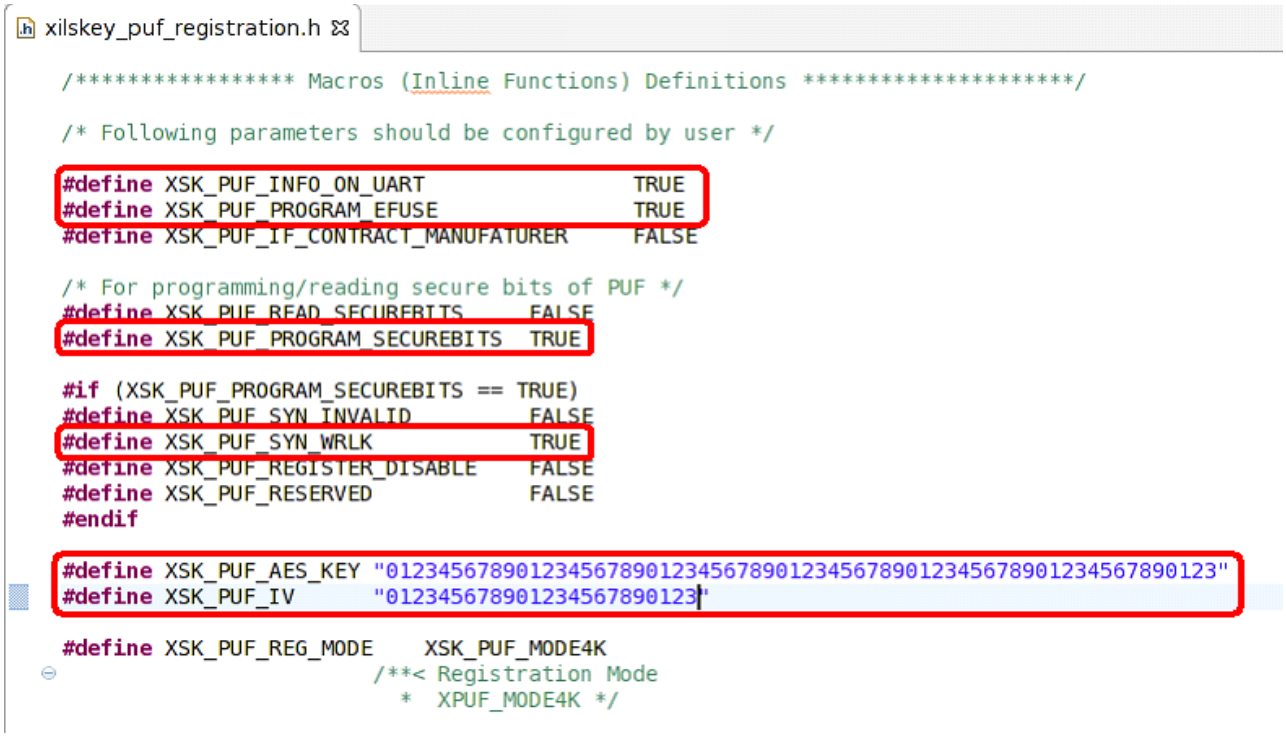


Figure 5: **PUF Registration File Required for eFUSE**

PUF Registration into eFUSEs

IMPORTANT: *THESE INSTRUCTIONS MODIFY THE EFUSES ON THE ZCU102 DEVELOPMENT BOARD AND MIGHT LIMIT FUTURE USE OF THE DEVELOPMENT BOARD FOR TESTING AND DEBUGGING!*

To register the PUF into the eFuse, perform the following steps:

1. Right-click on the **platform.spr** that is located under `ZCU102_XAPP1333` platform in the **Explorer** view and click **Open**.
2. Select **Board Support Package** under **standalone on psu_cortexa53_0** in the `ZCU102_XAPP1333` platform view and click **Modify BSP settings**.

3. In the **Board Support Package Settings** window, expand the Overview tree, then click **standalone** as shown in the figure below.

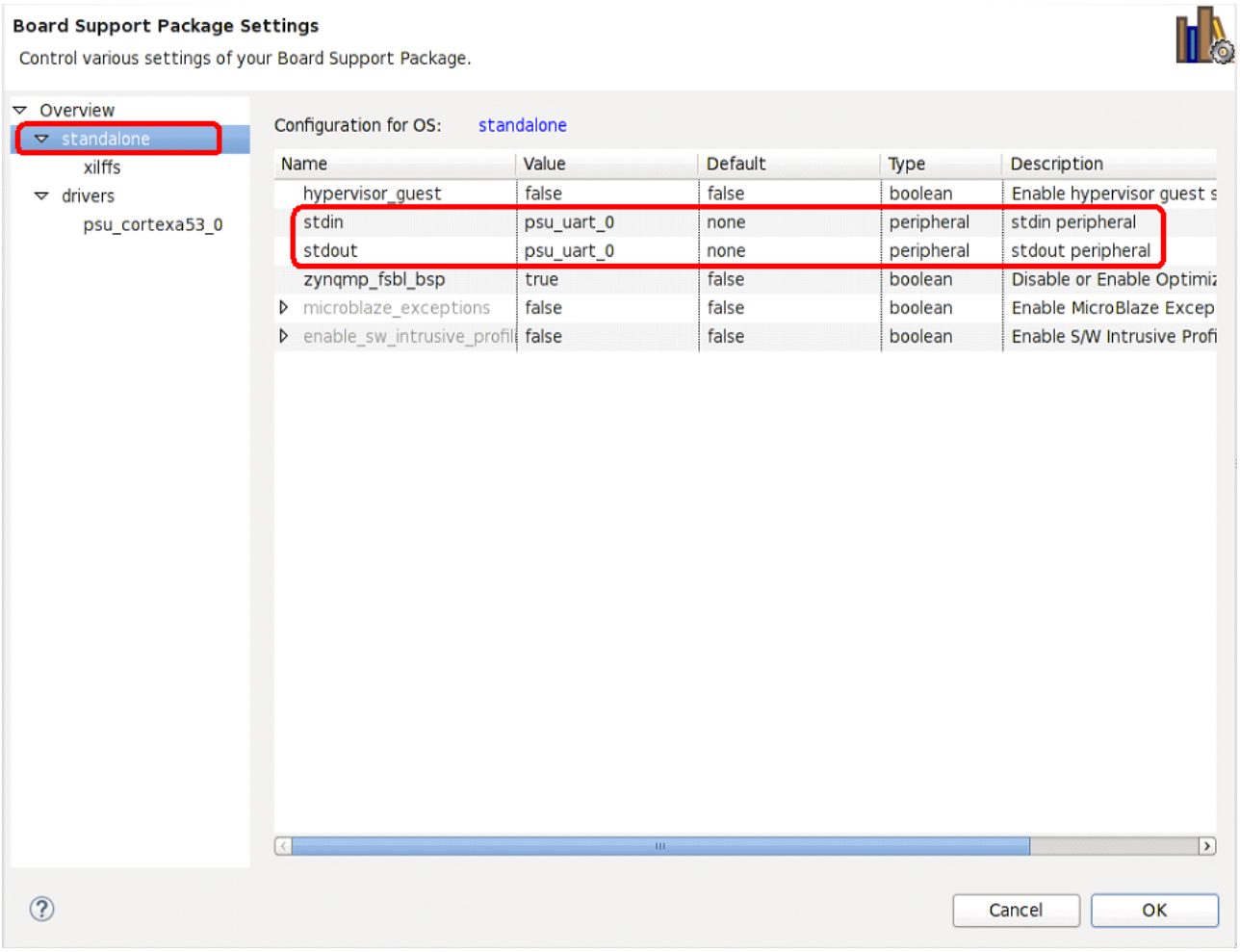


Figure 6: **Setting Up the UART Output Using the BSP Settings**

4. Ensure the **stdin** and **stdout** functions are mapped to `psu_uart_0` and click **OK**.
5. In Xilinx Vitis Explorer view, on the left, right-click **xilskey_puf_registration_example_1_system** and select **Build Project**.
6. Turn power off to the ZCU102 board.
7. Connect either the USB JTAG connector J2 to the ZCU102 development board and then a computer or connect the Platform JTAG to the ZCU102 and the associated hardware to a computer.
8. Connect a USB cable from the USB Serial port connector J83 on the ZCU102 board to a computer and note which COM port was enumerated with the Silicon Labs Quad CP2108 USB to UART Bridge: Interface 0.
9. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud. Enable terminal logging and select a file name and location.

10. On the ZCU102 development board, set the dip switch SW6 to configure the board for JTAG boot mode as shown in the figure below.

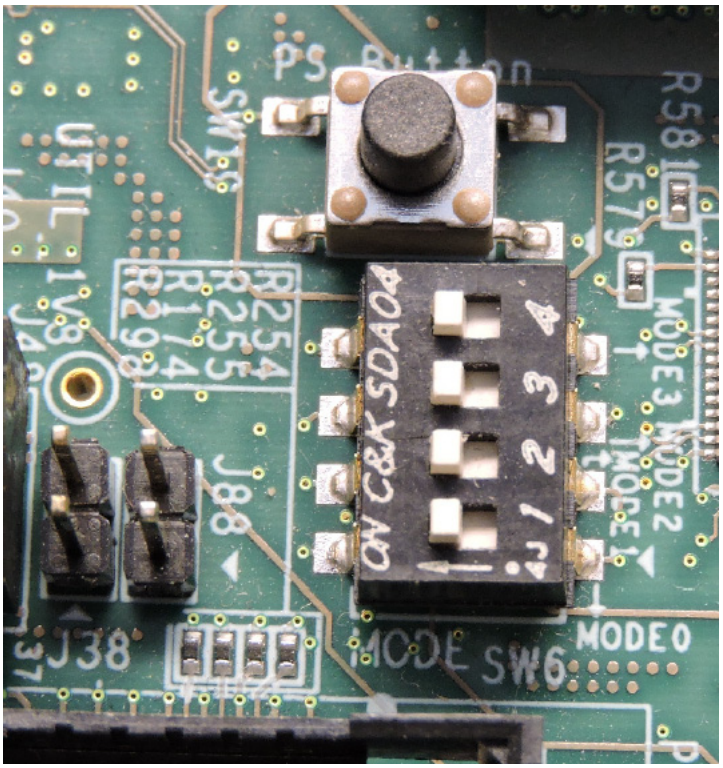


Figure 7: **ZCU102 JTAG Boot Mode Switch**

11. Power on the ZCU102 board using switch SW1.

12. Right-click **xilskey_puf_registration_example_1 > Run As > Launch Hardware (Single Application Debug)** as shown in the figure below.

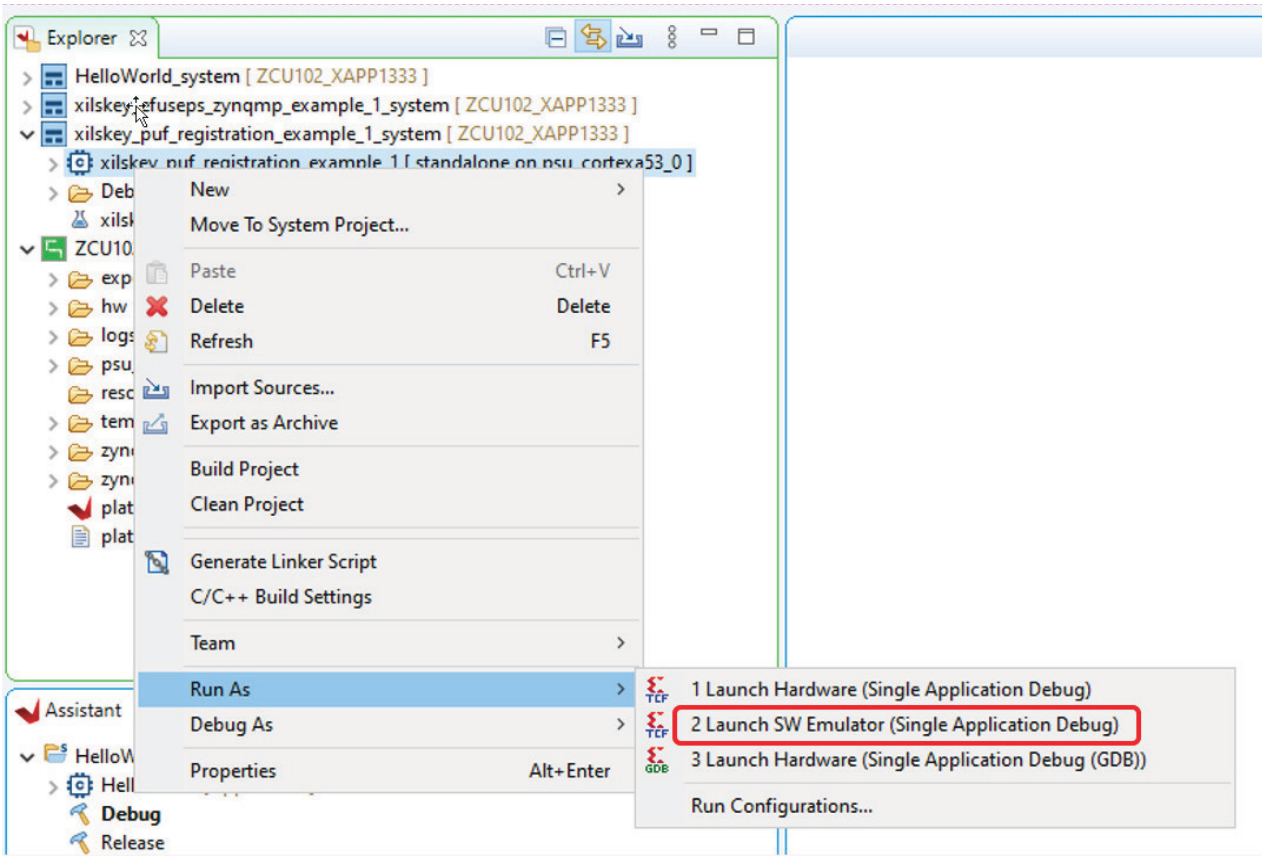


Figure 8: **Running the PUF Registration on the ZCU102 Board**

The PUF registration application starts running and outputs information to the terminal as shown in the figure below. An example log of the PUF registration is included in the design files in the `Logs` folder called **puf_registration_log.log**.

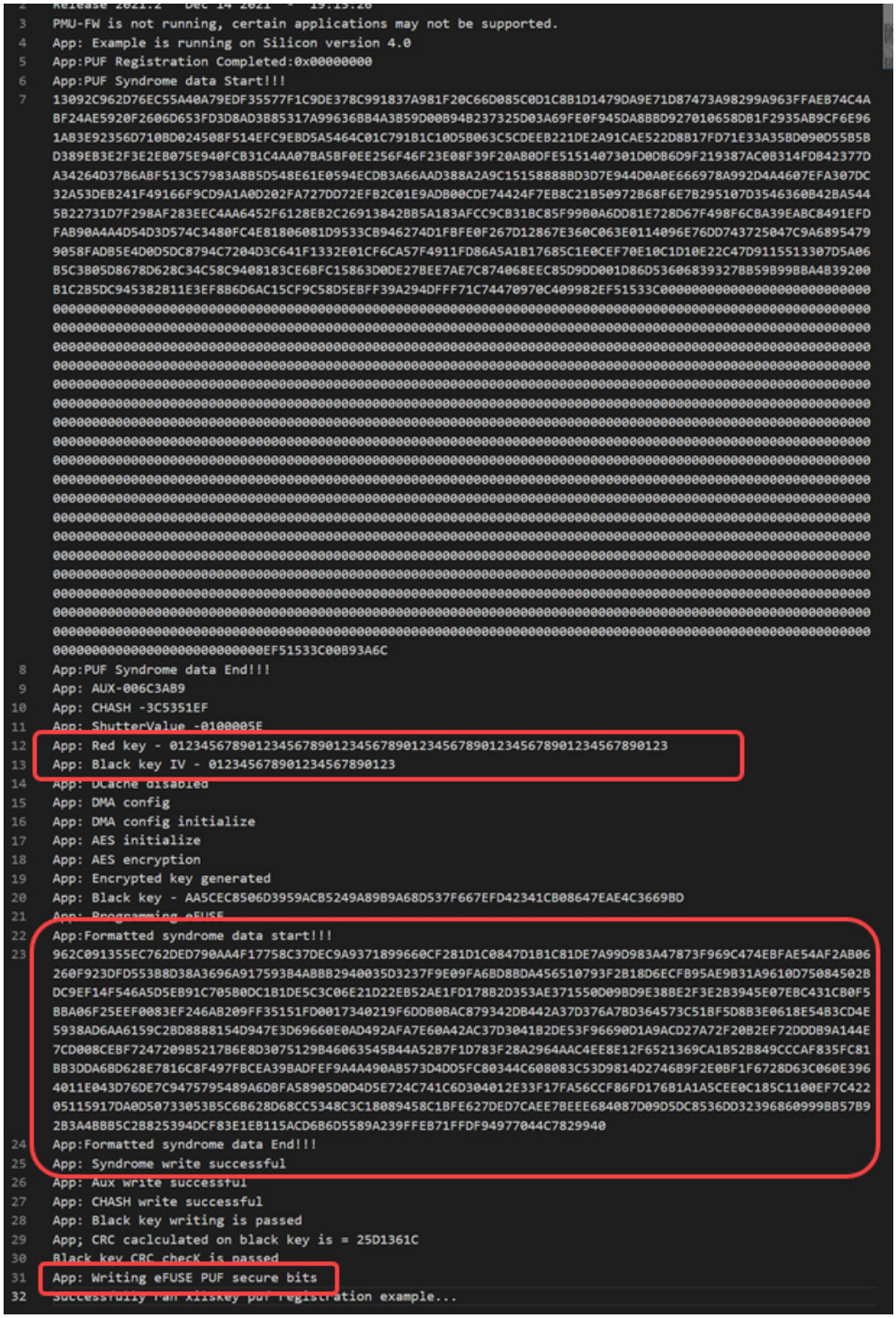


Figure 9: **Terminal Output Registering PUF to eFUSEs**

13. Verify line 12 of the UART output is the Red Key that was configured in **XSK_PUF_AES_KEY** in **xilskey_puf_registration.h**.
14. Verify line 13 of the UART output is the Black Key IV that was configured in **XSK_PUF_BLACK_KEY_IV** in **xilskey_puf_registration.h**.

- Line 20 of the UART output is the **Black Key** generated by the AES encryption engine using the PUF as a KEK.

- Line 21 shows that the **Black Key** was burned into eFUSEs.
- Line 23 of the UART output is the required syndrome data that the PUF uses to regenerate its device-unique encryption key. It is the data that is being programmed into the eFUSEs.
- Lines 31 shows that the PUF information has been burned into eFUSEs.

15. Power off the **ZCU102** development board.

RSA eFUSE Configuration

IMPORTANT: *THESE INSTRUCTIONS MODIFY THE EFUSES ON THE ZCU102 DEVELOPMENT BOARD AND MIGHT LIMIT FUTURE USE OF THE DEVELOPMENT BOARD FOR NON-SECURE TESTING AND DEBUGGING!*

IMPORTANT: *Programming any of the RSA_EN eFUSE settings preclude Xilinx test access. Consequently, Xilinx might not accept return material authorization (RMA) requests.*

RSA eFUSE Settings

RSA eFUSE registration is covered in detail in *Programming eFUSEs for AES and RSA Cryptographic Functions* in the *Programming BBRAM and RSA_EN eFUSEs* [Ref 3], so only a summary pertaining to this application note is covered here.

1. Right-click on the **platform.spr** that is located under **ZCU102_XAPP1333** platform in the **Explorer** view and click **Open**.
2. Select **Board Support Package** under **standalone on psu_cortexa53_0** in the **platform view** that just opened and select **Libraries** tab in **Operating Systems** section.
3. Scroll to the bottom of the libraries tab and click Import Examples for the **xilskey** library.
4. Check the **xilskey_efuseps_zynqmp_example** project and click **OK**. This adds the associated project to your workspace.
5. Open the **xilskey_efuseps_zynqmp_input.h** file in the `src` folder under the fully expanded `xilskey_efuseps_zynqmp_example_1_system` in the **Project Explorer** tab.
6. Change the definition of **XSK_EFUSEPS_RSA_ENABLE** to **TRUE**. This permanently forces the use of RSA authentication.
7. Change the definition of **XSK_EFUSEPS_PPK0_WR_LOCK** to **TRUE**. This prevents any modifications to the PPK0 hash stored in eFUSEs.

   The first set of settings are shown in the figure below:

```
/**
 * Following is the define to select if the user wants to program
 * Secure control bits
 */
#define XSK_EFUSEPS_AES_RD_LOCK          FALSE
#define XSK_EFUSEPS_AES_WR_LOCK          FALSE
#define XSK_EFUSEPS_ENC_ONLY             FALSE
#define XSK_EFUSEPS_BBRAM_DISABLE        FALSE
#define XSK_EFUSEPS_ERR_DISABLE          FALSE
#define XSK_EFUSEPS_JTAG_DISABLE         FALSE
#define XSK_EFUSEPS_DFT_DISABLE          FALSE
#define XSK_EFUSEPS_PROG_GATE_DISABLE    FALSE
#define XSK_EFUSEPS_SECURE_LOCK          FALSE
#define XSK_EFUSEPS_RSA_ENABLE           TRUE
#define XSK_EFUSEPS_PPK0_WR_LOCK         TRUE
#define XSK_EFUSEPS_PPK0_INVLD           FALSE
#define XSK_EFUSEPS_PPK1_WR_LOCK         FALSE
#define XSK_EFUSEPS_PPK1_INVLD           FALSE
#define XSK_EFUSEPS_LBIST_EN             FALSE
#define XSK_EFUSEPS_LPD_SC_EN            FALSE
#define XSK_EFUSEPS_FPD_SC_EN            FALSE
#define XSK_EFUSEPS_PBR_BOOT_ERR         FALSE
```

Figure 10: **Settings for RSA Authentication When Using eFUSEs - 1**

8. In the next section of the configuration, change the definition of **XSK_EFUSEPS_WRITE_PPK0_HASH** to **TRUE**.
9. Change the definition of **XSK_EFUSEPS_PPK0_HASH** to the value stored in **sha3.txt** that was created by `bootgen` (or copied form the `Keys` directory) from the previous section.

   The second set of settings are shown in Figure 11. These settings using the examples keys are included in the design files in the **xilskey_efuseps_zynqmp_input.h** file in the `rsa_registration` folder. The second RSA authentication key (PPK1) is not written for this application note but it can be done by changing the value of `XSK_EFUSEPS_PPK1_WR_LOCK` and `XSK_EFUSEPS_PPK1_HASH`.
10. To save changes to **xilskey_efuseps_zynqmp_input.h** click **File -> Save** in the main toolbar.

```
/**
 * Following is the define to select if the user wants to select AES key,
 * User Fuses, PPK0 Sha3 hash, PPK1 Sha3 hash and SPKID for Zynq MP
 */
/* For writing into eFuse */
#define XSK_EFUSEPS_WRITE_AES_KEY       FALSE
#define XSK_EFUSEPS_WRITE_PPK0_HASH     TRUE
#define XSK_EFUSEPS_WRITE_PPK1_HASH     FALSE
#define XSK_EFUSEPS_WRITE_SPKID         FALSE

#define XSK_EFUSEPS_WRITE_USER0_FUSE        FALSE
#define XSK_EFUSEPS_WRITE_USER1_FUSE        FALSE
#define XSK_EFUSEPS_WRITE_USER2_FUSE        FALSE
#define XSK_EFUSEPS_WRITE_USER3_FUSE        FALSE
#define XSK_EFUSEPS_WRITE_USER4_FUSE        FALSE
#define XSK_EFUSEPS_WRITE_USER5_FUSE        FALSE
#define XSK_EFUSEPS_WRITE_USER6_FUSE        FALSE
#define XSK_EFUSEPS_WRITE_USER7_FUSE        FALSE

/**
 * Following defines should be given in the form of hex string.
 * The length of AES_KEY string must be 64, PPK hash should be 96/64 based on
 * SHA3/SHA2 selction and and for USER_FUSES, SPK ID  must be 32.
 */
#define XSK_EFUSEPS_AES_KEY     "0000000000000000000000000000000000000000000000000

#define XSK_EFUSEPS_PPK0_IS_SHA3    TRUE
#define XSK_EFUSEPS_PPK0_HASH   "38B29D670EECD2678105A9C51B203F50A0557B0614B33
```

Figure 11: **Settings for RSA Authentication When Using eFUSEs - 2**

Programming RSA eFUSEs

Program the RSA eFUSEs by performing the following steps:

1. Right-click on the **platform.spr** that is located under **ZCU102_XAPP1333** platform in the **Explorer** view and click **Open**.
2. Select **Board Support Package** under **standalone on psu_cortexa53_0** in the **ZCU102_XAPP1333** platform view and click **Modify BSP settings**.
3. In the **Board Support Package Settings** window, expand the **Overview** tree and then click **standalone**, as shown in Figure 4 in step 2 of PUF Registration into eFUSEs.
4. Ensure the **stdin** and **stdout** functions are still mapped to `psu_uart_0` and click **OK**.
5. In Xilinx Vitis Explorer view, on the left, right-click **xilskey_efuseps_zynqmp_example_1_system** and select the **Build Project** option. This may have already been completed if your SDK environment is set up to build automatically.
6. Power off the **ZCU102** board.

7. Connect either the USB JTAG connector J2 to the **ZCU102** development board and then a computer or connect the **Platform JTAG** to the **ZCU102** and the associated hardware to a computer.
8. Connect a USB cable from the USB Serial port connector J83 on the ZCU102 board to a computer and make note of which COM port was enumerated with the *Silicon Labs Quad CP2108 USB to UART Bridge: Interface 0*.
9. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud. Enable terminal logging and select a file name and location.
10. On the **ZCU102** development board, set the dip switch **SW6** to configure the board for **JTAG** boot mode as shown in Figure 7.
11. Power on the **ZCU102** board using switch **SW1**.
12. Right-click **xilskey_efuseps_zynqmp_example_1 > Run As > Launch on Hardware (Single Application Debug)**.
13. The RSA eFUSE application starts running and outputs information to the terminal as shown in Figure 12. An example log of the writing the RSA eFUSEs is included in the design files in the *Logs* folder called **write_rsa_enable_log.log**.
14. Verify line 15 from the output terminal matches the SHA3 output that was generated and stored in the **sha3.txt** file.
15. Notice that line 32 from the terminal matches the SHA3 output that was generated and stored in **sha.txt** file.

- Line 32 confirms that RSA authentication is enabled and now required for use because this was burned into the eFUSEs.
- Line 33 shows that the PPK0 eFUSE has been programmed and the PPK0 SHA3 value cannot be changed.

16. Power off the **ZCU102** development board.

```
Xilinx Zynq MP First Stage Boot Loader
Release 2021.2   Dec 14 2021  -  19:15:26
PMU-FW is not running, certain applications may not be supported.
DNA:400000000129958924416205keys read from cache
User0 Fuse:00000000
User1 Fuse:00000000
User2 Fuse:00000000
User3 Fuse:00000000
User4 Fuse:00000000
User5 Fuse:00000000
User6 Fuse:00000000
User7 Fuse:00000000


PPK0:38B29D670EECD2678105A9C51B203F50A0557B0614B3368E688469D31CBC867EE7C38D287E670D2E4FEF6D2D0EA9F7DE

PPK1:0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Spkid: 00000000

Secure and Control bits of eFuse:

AES key CRC check is enabled
Programming AES key is enabled
All boots must be encrypted with eFuseAES key is disabled
BBRAM key is not disabled
Error output from PMU is enabled
Jtag is enabled
DFT is enabled
PROG_GATE feature is enabled
Reboot from JTAG mode is enabled
RSA authentication is enabled
Locks writing to PPK0 efuse
Revoking PPK0 is disabled
writing to PPK1 efuses is not locked
Revoking PPK1 is disabled
LBIST is in disabled state
PBR boot error halt is disabled
Zeroization of registers in Low Power Domain (LPD) during boot is disabled
Zeroization of registers in Full Power Domain (FPD) during boot is disabled

User control bits of eFuse:
Programming USER_0 fuses is enabled
Programming USER_1 fuses is enabled
Programming USER_2 fuses is enabled
Programming USER_3 fuses is enabled
Programming USER_4 fuses is enabled
Programming USER_5 fuses is enabled
Programming USER_6 fuses is enabled
Programming USER_7 fuses is enabled
Reserved 1 bits are not programmed on eFUSE
Reserved 2 bits are programmed on eFUSE

Successfully ran ZynqMP eFuse example....
```

Figure 12: **Terminal Output While Writing the RSA Settings to eFUSEs**

PUF Encryption and Decryption

PUF Encryption Decryption Demo Application

The PUF can now be used for encrypting and decrypting user data because the ZCU102 development board has been provisioned. Specifically, this section uses a reference design to show how to encrypt and decrypt user generated AES keys that are stored on an SD card.

1. To support the SD card storage the **xilffs** library has to be added to application BSP. Right-click on the **platform.spr** that is located under **ZCU102_XAPP1333** platform in the **Explorer** view and click **Open**.
2. Select **Board Support Package** under **standalone on psu_cortexa53_0** in the **ZCU102_XAPP1333** platform view and click **Modify BSP settings**.
3. Select **xilffs** library in the Board Support Package Settings window. Click on the **xilffs** library that appears on the left in **Overview -> standalone** and set the **enable_exfat** configuration parameter to **true**. Click **OK.**

   These settings are shown in the following figure:

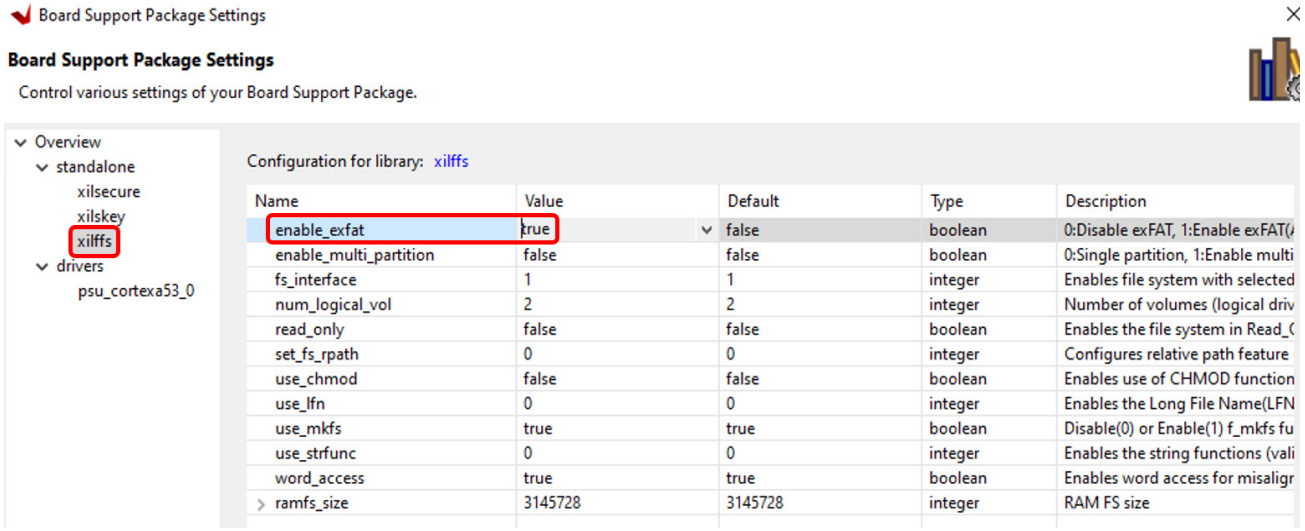


Figure 13: **Configuring Board Support Package Settings in Xilinx Vitis – Standalone Library Configuration**

4. Right-click the `ZCU102_XAPP1333` platform in the **Explorer** view, which is now marked as out-of-date, and click **Build Project**.
5. In Xilinx Vitis, click **File > New > Application Project**. If **Create a New Application Project** window appear click **Next**.
6. Select `ZCU102_XAPP1333` platform in the Platform Window and click **Next**.
7. Type in **ExternalKeyStorage** in the Application project name:
8. Leave remaining parameters at their default value and click **Next.**
9. Leave the domain as **standalone on psu_cortexa53_0**. These settings are shown in Figure 14, Figure 15, and Figure 16.
10. Select **Next**.
11. Select **Empty Application (C)**.
12. Click **Finish**.
13. Expand the src folder in **ExternalKeyStorage** of the Project explorer window.
14. Right-click **src** and select **Import Sources**.

15. Click **Browse** in the File system window.
16. Navigate to the `ExternalKeyStorage/src` folder in the reference design file directory and check all ".c" and ".h" files and then click **Finish** as shown in Figure 17.

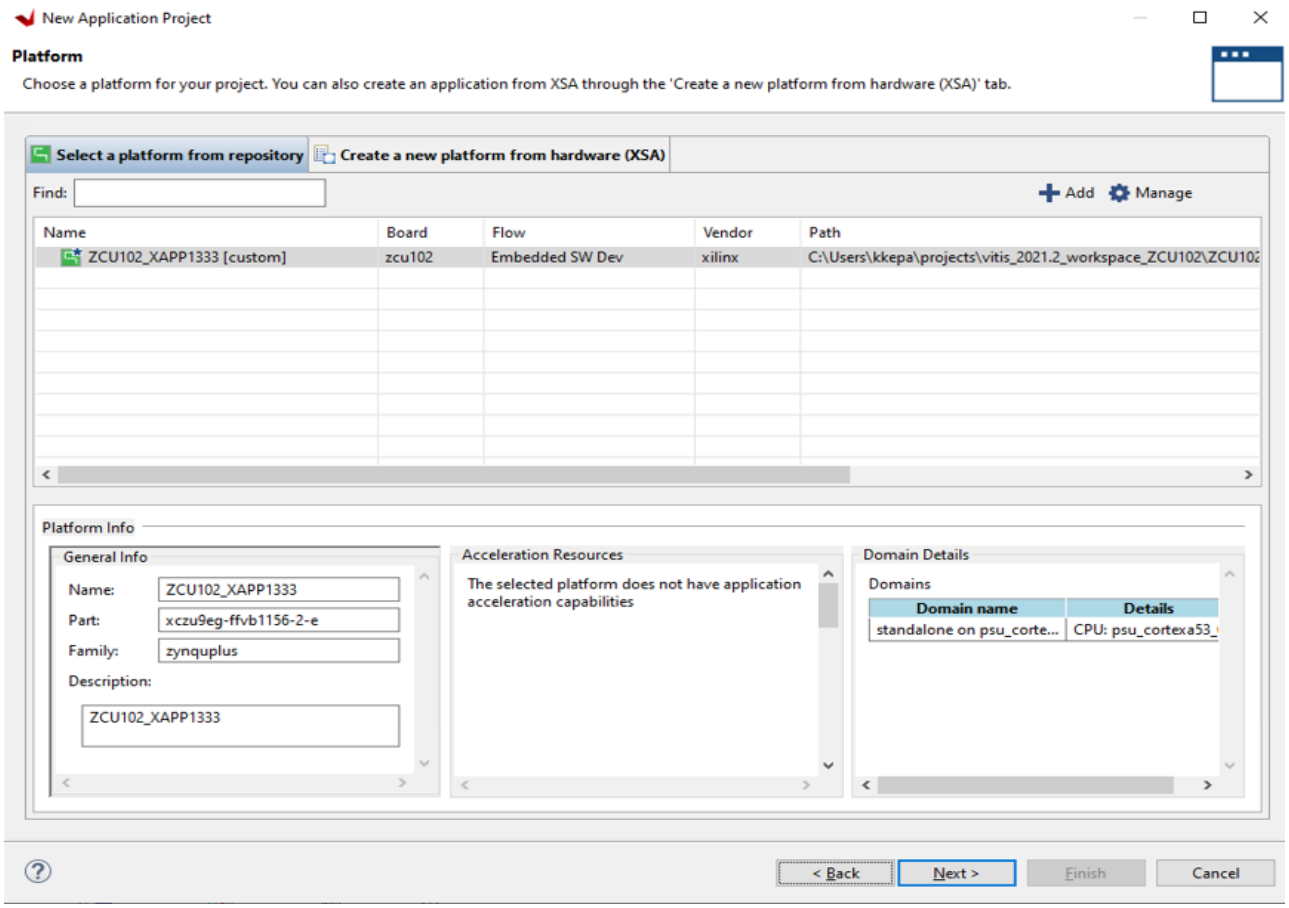


Figure 14: **Creating the ExternalKeyStorage Project in Xilinx Vitis – Platform Selection**

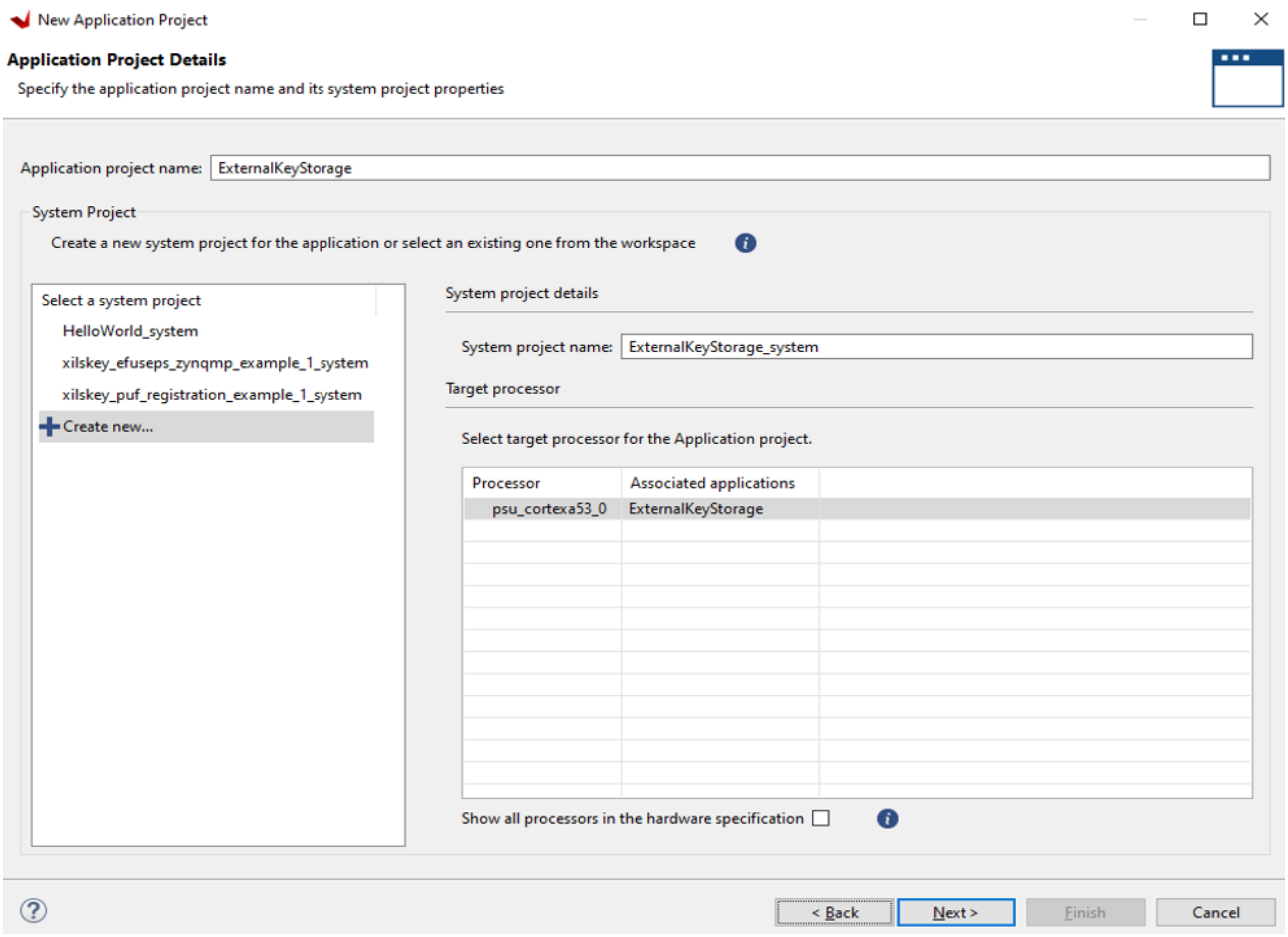


Figure 15: **Creating the ExternalKeyStorage Project in Xilinx Vitis – System Selection**

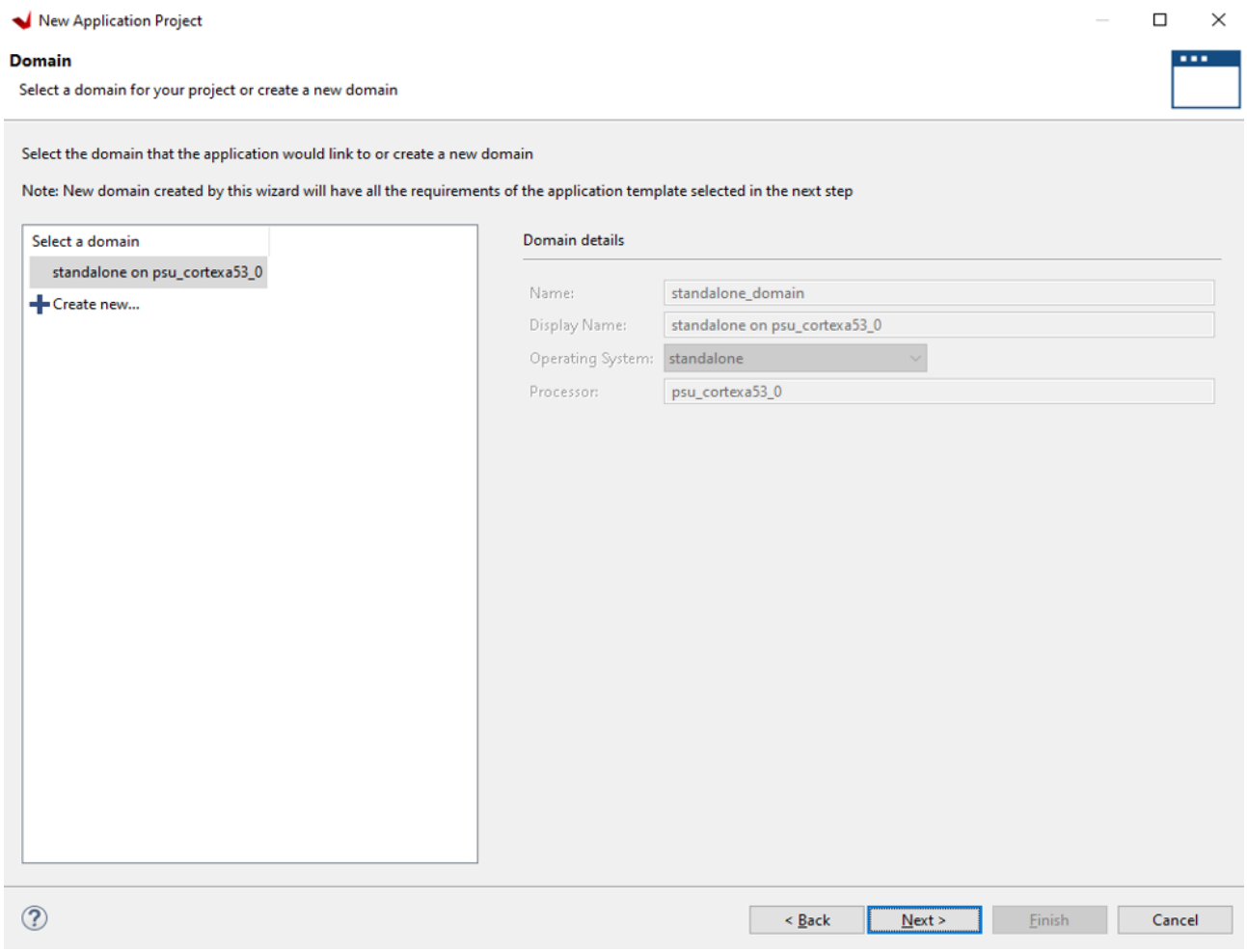


Figure 16: **Creating the ExternalKeyStorage Project in Xilinx Vitis – Domain Selection**

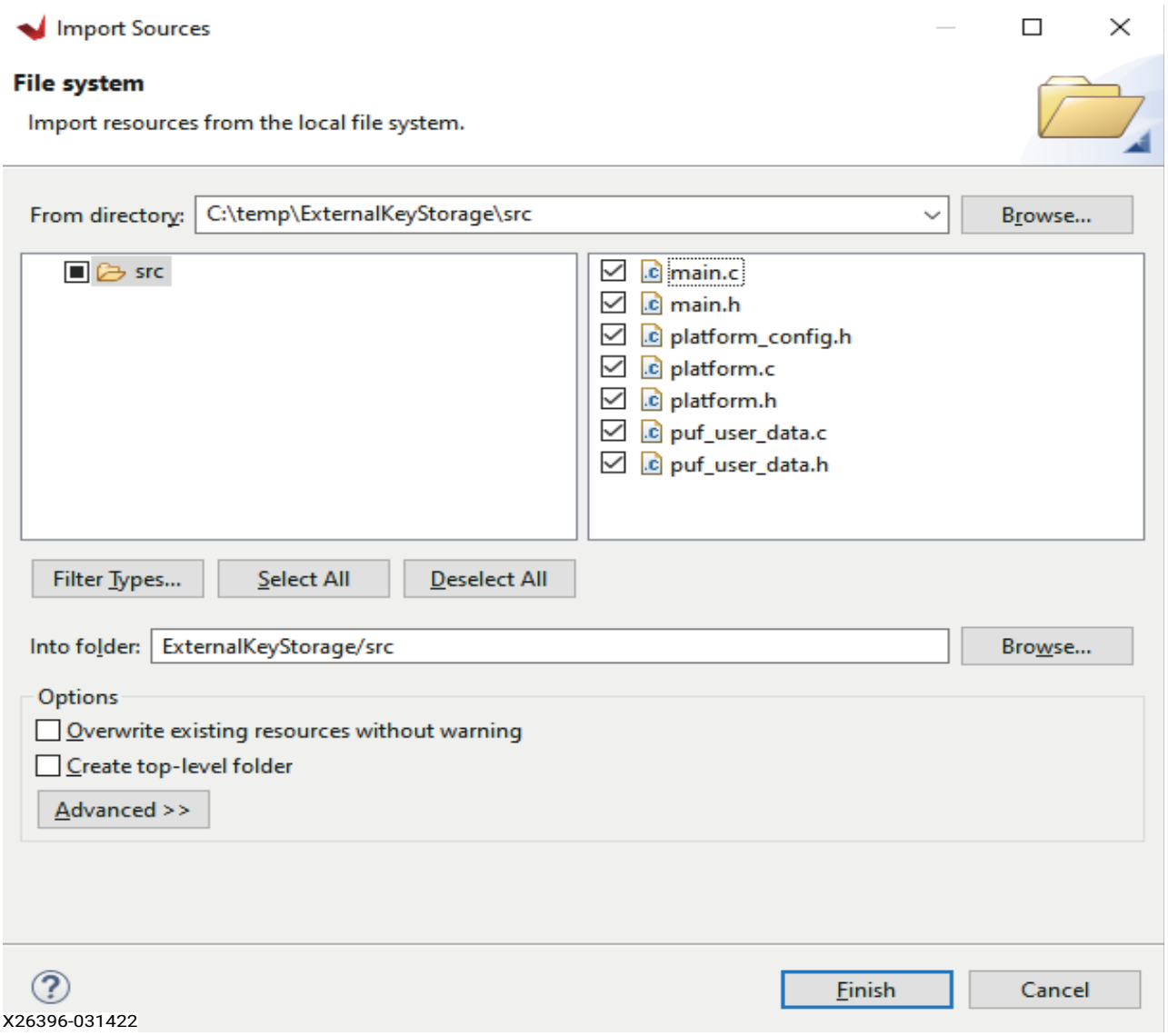


Figure 17: **Importing Files from the Reference Design into the ExternalKeyStorage Project**

17. Create a new file called **ExternalKeyStorage.bif** in the **ExternalKeyStorage** folder. This file is also included with the design files and can be copied into the project folder but the paths must be updated to point to the correct folders. Manual creation of the BIF file is necessary to use the Black Key during boot as the Create Boot Image tool within Xilinx Vitis does not currently support this feature. Future revisions of Xilinx Vitis may support this feature.
18. Update the contents of the file to the contents shown in the following figure using the correct paths.

```
1  //arch = zynqmp; split = false; format = BIN
2  the_ROM_image:
3  {
4      [pskfile] ../Keys/psk0.pem
5      [sskfile] ../Keys/ssk0.pem
6      [auth_params] spk_id = 0; ppk_select = 0
7      [keysrc_encryption] efuse_blk_key
8      [bh_key_iv]  ../Keys/puf_iv.txt
9      [fsbl_config] puf4kmode, shutter = 0x0100005E, opt_key
10     [bootloader, destination_cpu=a53-0, encryption = aes, authentication = rsa, aeskeyfile = ../Keys/multiple_keys.nky]
11       ../ZCU104/zynqmp_fsbl/fsbl_a53.elf
12     [destination_cpu=pmu, authentication = rsa]
13       ../ZCU104/zynqmp_pmufw/pmufw.elf
14     [destination_cpu = a53-0, encryption = aes, authentication = rsa, aeskeyfile = ../Keys/multiple_keys_app.nky]
15       ./Debug/ExternalKeyStorage.elf
16 }
```

Figure 18: **ExternalKeyStorage.bif File**

19. Build the ExternalKeyStorage project in Xilinx Vitis.
20. From the command prompt in the ExternalKeyStorage folder run the following command: `bootgen -p zcu9eg -arch zynqmp -image ExternalKeyStorage.bif -w -o BOOT.bin`
21. Power off the **ZCU102** board.
22. Copy **BOOT.bin** to a blank SD card.
23. Load the SD card into the J100 SD slot on the ZCU102 development board.
24. Connect a USB cable from the USB Serial port J83 on the ZCU102 board to a computer and make note of which COM port was enumerated with the Silicon Labs Quad CP2108 USB to UART Bridge: Interface 0.
25. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud. Enable terminal logging and select a file name and location.

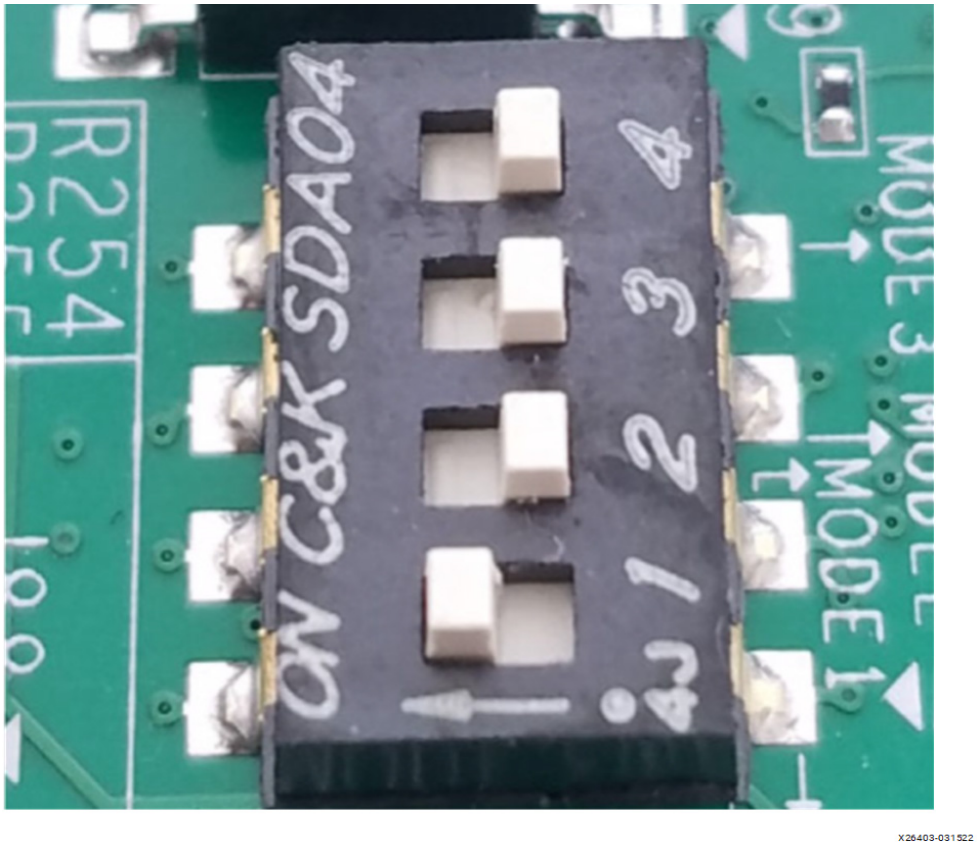


Figure 19: **ZCU102 SD Boot Mode Switch Setting**

26. On the **ZCU102** development board, set the dip switch **SW6** to configure the board for SD boot mode as shown in the previous figure.
27. Load the SD card into the **J100 SD** slot on the **ZCU102** development board.
28. Power on the **ZCU102** board using switch **SW1**.

In the terminal program, a menu appears as shown in the following figure:

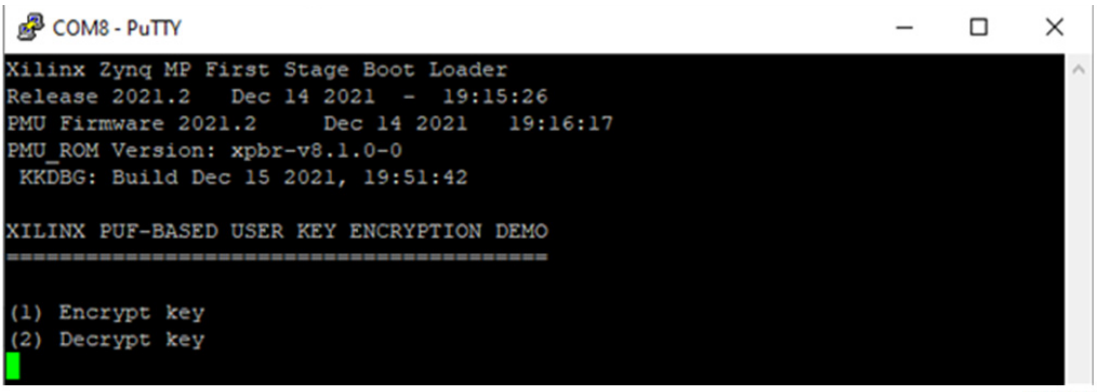


Figure 20: **Main Menu of External Key Storage Demo**

29. Press **1** to encrypt a user key and to save the encrypted key to the external SD card and follow the prompts, as illustrated in Figure 21.

a. Enter a 96-bit IV. **Please note: Do not reuse IV.** Per the AES-GCM standard the IV should be a new one per every use.

b. Enter an 8-bit key ID. Use an ID of 42 for this key. An ID of 0 is mapped to user eFUSE 0 bit 0, an ID of 1 is mapped to user eFUSE 0 bit 1, … , an ID of 255 is mapped to user eFUSE 7 bit 31.

c. Enter a 256-bit AES key.

d. Enter a file name including a file extension (for example, Key1.key) for the key up to 16 characters long and then press enter when complete.

30. After entering the file name, the program displays the unencrypted key blob which consists of the IV, Key's ID, and the key itself. Afterwards, the ID and AES key are encrypted using the PUF's device-unique KEK, the entire 61 byte encrypted key blob is displayed, and the entire encrypted key blob is written to the SD card.
31. Repeat the entire encryption process and encrypt another key and new IV (as per AES-GCM standard), using step 29. However, select an ID that is equal to `0xFF` and create a unique key file name (e.g., Key2.key).
32. Power off the **ZCU102** board.
33. Remove the SD card and insert the card into a SD card reader on a computer.
34. Using a browser or the command line, display the contents of the SD card.
35. Make sure both key files generated in step 29 and step 31 appear on the SD card as shown in Figure 22.

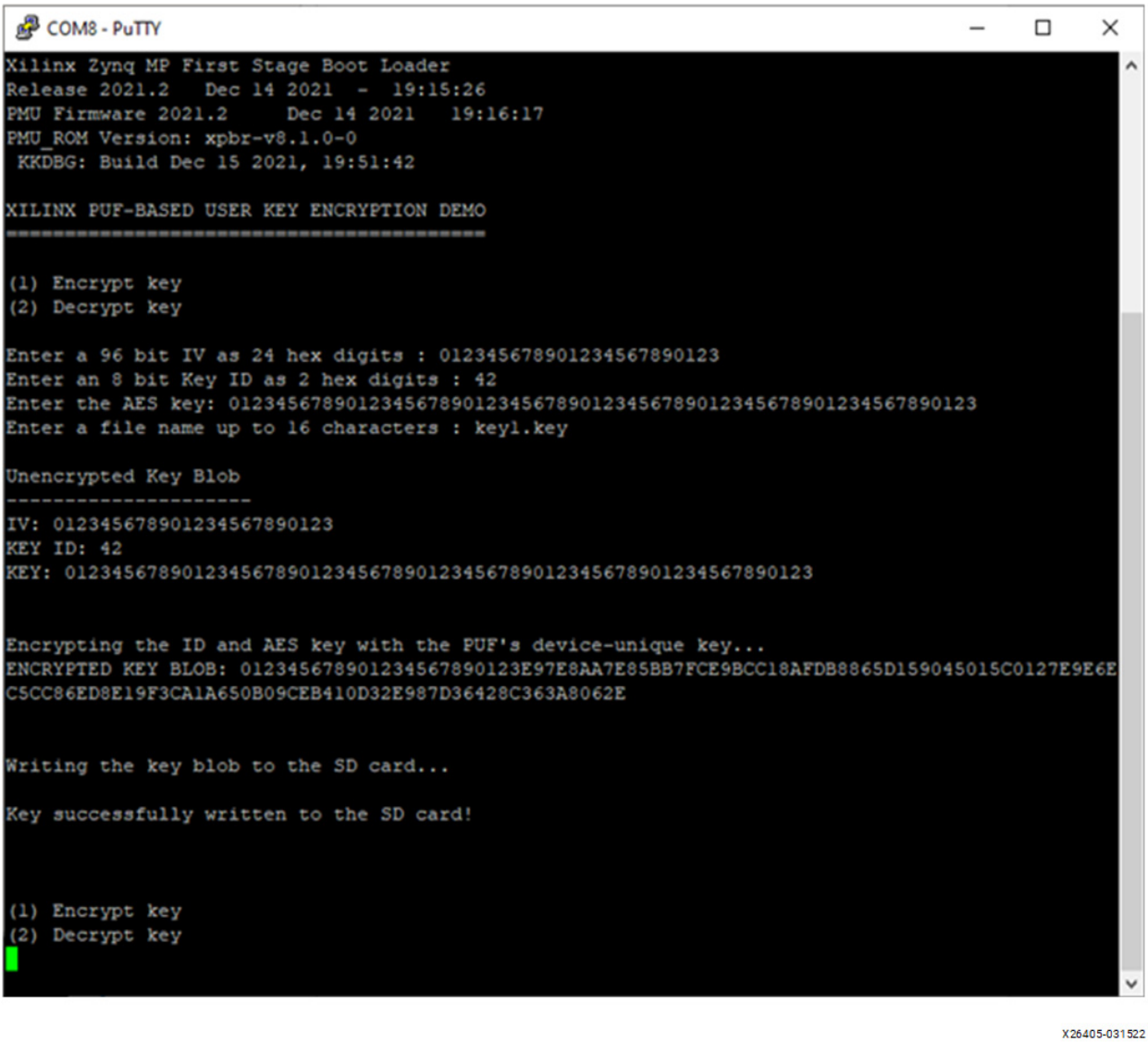


Figure 21: **External Key Storage Encryption**

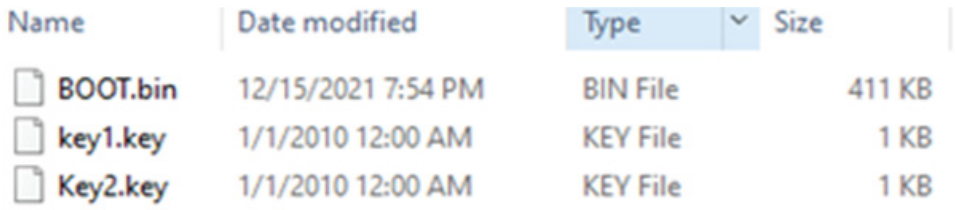


Figure 22: **Directory Contents of the SD Card after Writing the Encrypted Key**

36. Open both keys in a hex editor and confirm that they match the encrypted key blobs displayed in the user application. KEY1.KEY is shown in the following figure and matches the output generated in Figure 21.

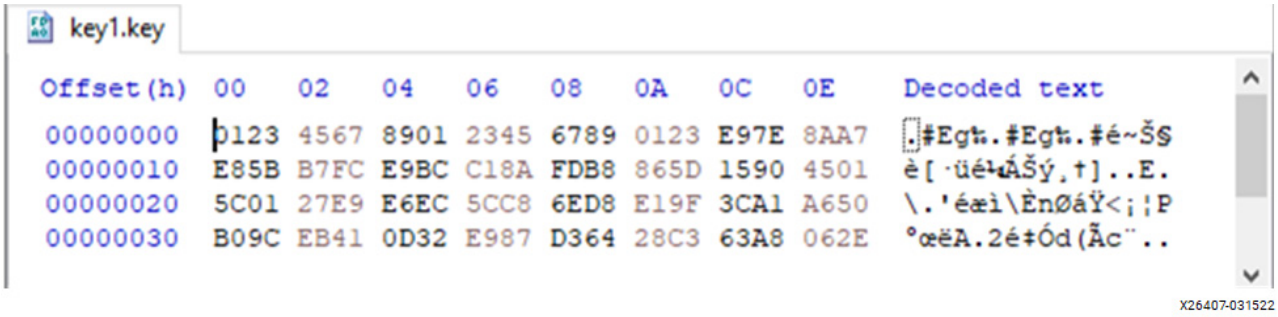
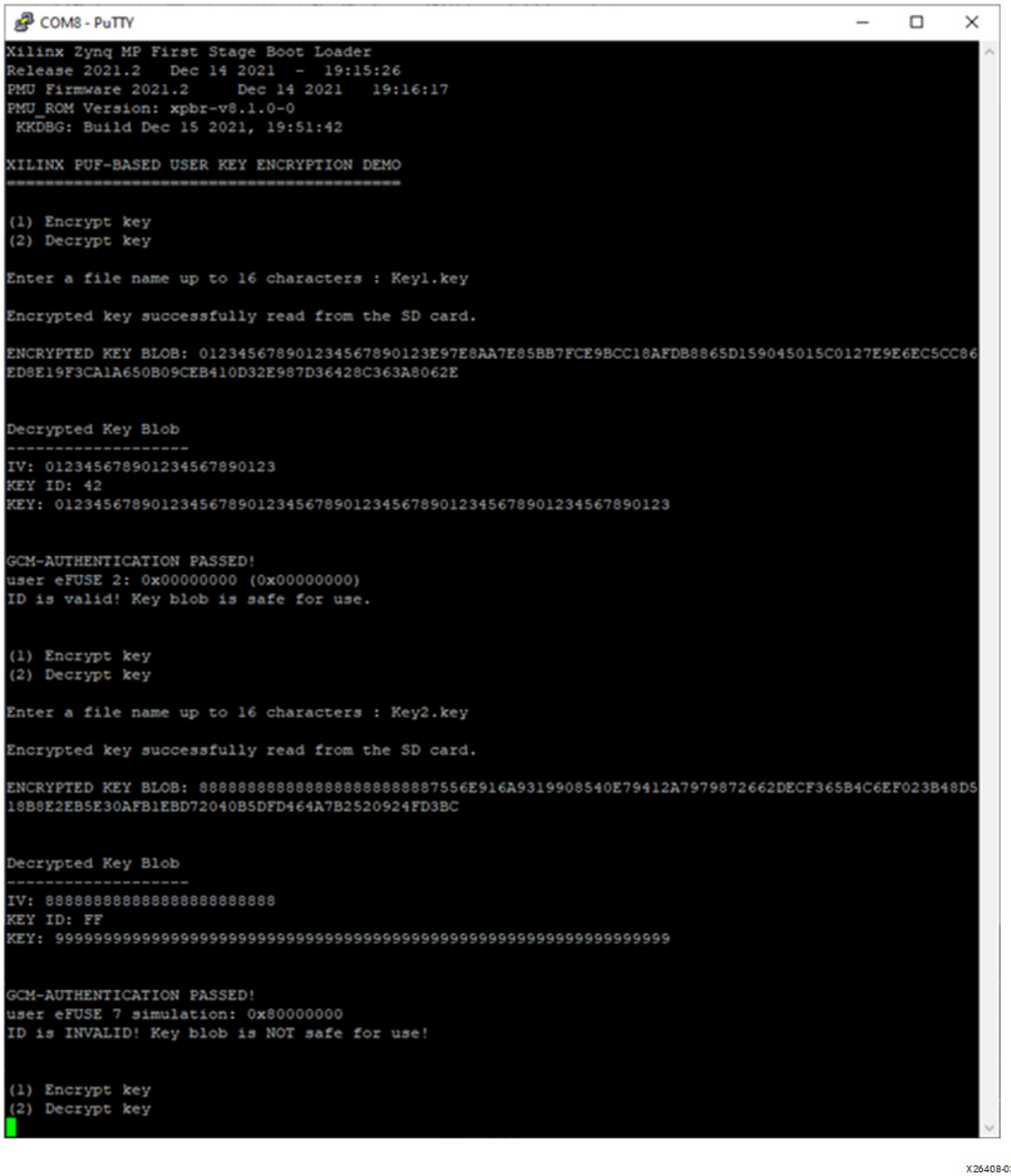


Figure 23: **Encrypted Key Data Stored in KEY1.KEY Read from the SD Card**

37. Remove the SD card from the computer and insert the card into the **ZCU102** development board.
38. Apply power to the **ZCU102** development board. The menu shown in Figure 20 appears.
39. Press **2** to decrypt the data that is stored externally on the SD card.
40. Type in the name of the key file and the file extension used in step 29 (Key1.key).
    a. The key is read from the SD card and placed into OCM for processing.
    b. The encrypted key blob is displayed.
    c. The decryption process of the key blob takes place and the decrypted information is displayed showing the IV, key ID, and key.
    d. The decrypted GCM tag is compared to the GCM tag stored in the encrypted key blob and the software indicates if they match.
    e. Lastly, the key ID is mapped to and compared to the associated bit stored in the user eFUSEs and the software indicates if the IDs match. In this case, the IDs match. An ID of 0 is mapped to user eFUSE 0 bit 0, an ID of 1 is mapped to user eFUSE 0 bit 1, … , an ID of 255 is mapped to user eFUSE 7 bit 31.
41. Repeat the process and decrypt the second key that was created in step 29.
42. All of the same information from step 40 is displayed and the key is decrypted and passes authentication. However, the software simulates ID 255 being revoked and should not be used. When ID 255 is read from a decrypted key file, the software replaces the actual value read in from User eFUSE 7, 0x0000_0000, with a simulated value of 0x8000_0000. Since bit 31 of User eFUSE 7 is now set and appears to be burned, this simulates ID 255 as being revoked. Decrypting the two test keys is shown in the following figure.



X26408-03152

Figure 24: **External Key Storage Decryption - Decrypting Two Keys and Simulating a Revocation of Key with ID 255**

Ordering

Because of the additional screening required to ensure entropy, Xilinx offers two versions of the PUF, a 128-bit and a 256-bit. In both cases, the KEK length is 256 bits. These devices require special ordering codes (SCD). The PUF is not supported for the standard ordering codes, except for development and evaluation, as there is no assurance that there is sufficient entropy in the KEK. Entropy is measured as described in *Zynq UltraScale+ MPSoC PUF Characterization Report* (RPT236) [Ref 6] which is a Xilinx proprietary report. Contact your local Xilinx FAE or sales person to obtain a copy of the report. Use of the PUF does not require additional licensing fees.

Conclusion

This application note guides a user on how to use the PUF's device-unique encryption key in conjunction with the AES-GCM hardware in order to encrypt user generated data and store the encrypted data externally. The encrypted data can then be read from external storage and decrypted using the AES-GCM hardware in conjunction with the PUF's device-unique key. In addition, this application note shows how to perform data validation of decrypted data packets by utilizing values stored in the user programmable section of eFUSEs.

Reference Design

Download the reference design files for this application note from the Xilinx website. The table below displays the reference design matrix.

Table 1: **Reference Design Checklist**

Parameter	Description
General	
Developer Name(s)	Jim Wesselkamper, Nathan Menhorn, Krzysztof Kepa
Target Devices	Zynq UltraScale+ devices
Source code provided?	Yes
Source code format (if provided)	C
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	
Simulation	
Functional simulation performed	No
Timing simulation performed?	No

Parameter	Description
Testbench provided for functional and timing simulation?	No
Testbench format	N/A
Simulator software and version	N/A
SPICE/IBIS simulations	N/A
Implementation software tool(s) and version	Vitis 2021.2
Static timing analysis performed?	No
Hardware Verification	
Hardware verified?	Yes
Platform used for verification	ZCU102 evaluation board

Document 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 o the Xilinx website.

Appendix A

Creating the Zynq UltraScale+ ZCU102 Evaluation Board Hardware Design

1. Open **Vivado Design Suite**.
2. In the **Quick Start** tab click **Create Project**.
3. Click **Next** in the **Create a New Vivado Project** page.

4. Enter **ZCU102** in the Project name.
5. Enter or select an appropriate working directory in the Project location.
6. Click **Next** on the **Project Name** page.
7. In **Project Type**, select **RTL Project** and uncheck two boxes for **Do not specify sources at this time** and **Project is an extensible Vitis platform**.
8. Click **Next** on the **Project Type** page.
9. Click **Next** on the **Add Sources** page.
10. Click **Next** on the **Add Constraints (optional)** page.
11. On the Default Part page, click the **Boards** tab.
12. Type in **ZCU102** in the Search.
13. Click the **Zynq UltraScale+ ZCU102 Evaluation Board**.
14. Click **Next** on the **Default Part** page.
15. Click **Finish** on the **New Project Summary** Page and wait while the project is being created.
16. In the Project Manager tab located on the left of the **Vivado** workspace, click **IP INTEGRATOR > Create Block Design**.
17. When the **Create Block Design** window appears, type in **ZCU102** in **Design name**. Leave everything else set to default.
18. Click **OK** and wait while the design is created.
19. In the **Diagram** section of the workspace, located on the top right, click the **+** button to add IP.
20. When the Search box appears, type in **ZYNQ**.
21. Double-click **Zynq UltraScale+ MPSoC** and wait while the part is added to the design.
22. Click **Run Block Automation** at the top of the **Diagram** window.
23. After the **Run Block Automation** window appears, select **All Automation** and **Apply Board Preset**, click **OK** and wait while the automation takes place.
24. Double-click the Zynq UltraScale+ part in the Diagram window.
25. Click **Page Navigator > PS-PL Configuration** located on the left of the Zynq UltraScale+ (3.3) window.
26. Click **PS-PL Interfaces** located in the PS-PL Configuration window.
27. Click **Master Interface** and uncheck the **AXI HPM0 FPD** and **AXI HPM1 FPD** parameters.
28. Click **OK** to close the window.
29. Pres **F6** to validate the design.
30. Click **OK** when the **Validate Design** window opens indicating the validation was successful.
31. In the **BLOCK DESIGN** window, click the **Sources** tab in the upper left-hand corner.

32. Right-click **ZCU102** under **Design Sources** and select **Create HDL Wrapper**.
33. When the **Create HDL Wrapper** window opens, click **Let Vivado manage wrapper and auto-update**, then click **OK** and wait while the sources are created.
34. In the **BLOCK DESIGN** window in the upper left corner on the **Sources** tab, expand the ZCU102_wrapper.
35. Right-click **ZCU102_i: ZCU102** and select **Generate Output Products**.
36. Leave the default settings in the **Generate Output Products** window. Click **Generate** and wait while the IP is being generated.
37. Click **OK** when the **Generate Output Products** window displays **Out-of-context module run was launched for generating output products**.

Exporting the ZCU102 Hardware and Launching Xilinx Vitis IDE

1. In the main **Vivado Design Suite** toolbar select **File > Export > Export Hardware**.
2. Click **Next** in the **Export Hardware Platform** window.
3. Select **Pre-synthesis output** and then click **Next** in **Output** window.
4. Leave the XSA file name and location default values and click **Next** in **Files** window.
5. Click **Finish** in **Exporting Hardware Platform** window.
6. In the main **Vivado Design Suite** toolbar select **Tools -> Launch Vitis IDE**.
7. If **Vitis IDE Launcher** window opens then in the Workspace provide appropriate working directory to create **Vitis workspace** and click **Launch**.
8. In the Welcome tab click **Create Platform Project** under Project column. If no Welcome tab is present then in the main toolbar click **File -> New -> Platform Project**.
9. Type in **ZCU102 XAPP1333** as Platform project name in **Create new platform** window and click **Next**.
10. In **Hardware Specification** section of **Platform window** click **Browse** to search the XSA file location.
11. In **Create Platform from XSA** window navigate to location that was provided in step 3, select the previously experted XSA file and click **Open**. Back in **Hardware Specification** section of **Platform window** the file should appear in the XSA File selection.
12. Leave other settings with default values, and then click **Finish**. After importing the hardware, you should see a project named `ZCU102_XAPP1333` that was automatically created based upon the **ZCU102** evaluation board. The platform is marked *out-of-date* because the software components are not yet built by Vitis.
13. To build the platform in **Vitis** Select the **ZCU102 XAPP1333** platform in the explorer view on the upper, left side and in the main toolbar click **Project -> Build Project**.

Validate the Hardware and Software with the Hello World Application

1. In the main **Xilinx Vitis** toolbar, select **File > New > Application Project**. If the **Create a New Application Project** window appears, click **Next**..
2. Select `ZCU102_XAPP1333` platform in the **Platform** window and then click **Next**.
3. In the **Application Project Details** type in **HelloWorld** in the Application project name. Leave remaining parameters at their default value and click **Next**.
4. Leave the domain as **standalone on** `psu_cortexa53_0` and click **Next**.
5. On the **Template** page of the **New Project** window, select **Hello World**.
6. Click **Finish**.
7. Right-click on the **platform.spr** that is located under `ZCU102_XAPP1333` platform in the **Explorer** view and click **Open.**
8. Select **Board Support Package Settings** under **standalone on** `psu_cortexa53_0` in the `ZCU102_XAPP1333` platform view and click **Modify BSP settings.**
9. In the **Board Support Package Settings** window expand the **Overview** tree and then click **standalone**.
10. Make sure the **stdin** and **stdout** functions are mapped to `psu_uart_0` and click **OK**.
11. Right-click the `HelloWorld` project and select **Build Project**.
12. Connect either the USB JTAG connector J2 to the ZCU102 development board and then to a computer or connect the Platform JTAG to the ZCU102 via J8 and the associated hardware to a computer.
13. Connect a USB cable from the USB Serial port connector J83 on the ZCU102 board to a computer and make note of which COM port was enumerated with the **Silicon Labs Quad CP2108** USB to UART Bridge: Interface 0.
14. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud.
15. On the **ZCU102** development board set the dip switch to configure the board for JTAG boot mode as shown in Figure 7.
16. Right-click the `HelloWorld` project and select **Run As > Launch on Hardware (Single)**.
17. Verify that “Hello World” is output on the terminal screen. The hardware and software is properly configured and is now ready for use.

References

1. Design Security Lounge
2. *Zynq UltraScale+ MPSoC Device: Technical Reference Manual* (UG1085)
3. *Programming BBRAM and eFUSEs* (XAPP1319)
4. *Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices* (XAPP1323)
5. *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209)
6. *Zynq UltraScale+ MPSoC PUF Characterization Report* (RPT236). Available in Design Security Lounge.
7. *NIST Special Publication (SP) 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM)*

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
04/12/2022 Version 1.2	
Throughout document	Updated SDK information to Vitis across all content
Introduction	Updated 2 block diagrams
PUF eFUSE Configuration	Added 1 new figure, replaced 2 and removed 1
PUF Encryption and Decryption	Added 2 new figures, replaced 6 and removed 2
Ordering	Updated SDK steps to Vitis steps and removed sub-section: Creating the First Stage Boot Loader (FSBL) and Board Support Package (BSP)
05/28/2021 Version 1.1	
Introduction	• Added a note for further clarity about boot header permissibility • Added a note about the PUF Key
06/26/2018 Version 1.0	
Initial Xilinx release.	N/A

Please Read: Important Legal Notices