External Secure Storage Using the PUF

XAPP1333 (v1.3) June 20, 2025

Summary

To store data in non-volatile memory (NVM) using an AMD 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, can the PUF's device-unique encryption key 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 AMD website. For detailed information about the design files, see Reference Design.

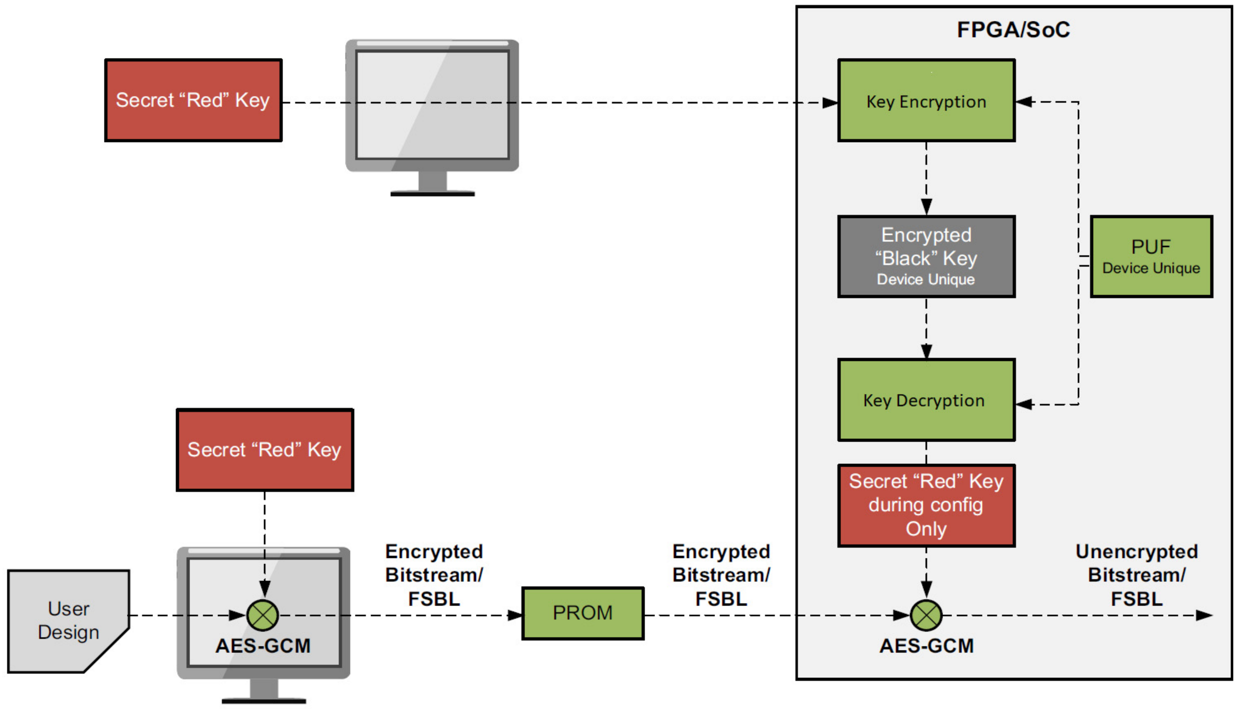
Introduction

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+ Device Technical Reference Manual* (UG1085). 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 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**


FPGA/SoC
Secret "Red" Key
Key Encryption
Encrypted "Black" Key Device Unique
PUF Device Unique
Key Decryption
Secret "Red" Key
Secret "Red" Key during config Only
User Design
AES-GCM
Encrypted Bitstream/ FSBL
PROM
Encrypted Bitstream/ FSBL
Unencrypted Bitstream/ FSBL
AES-GCM

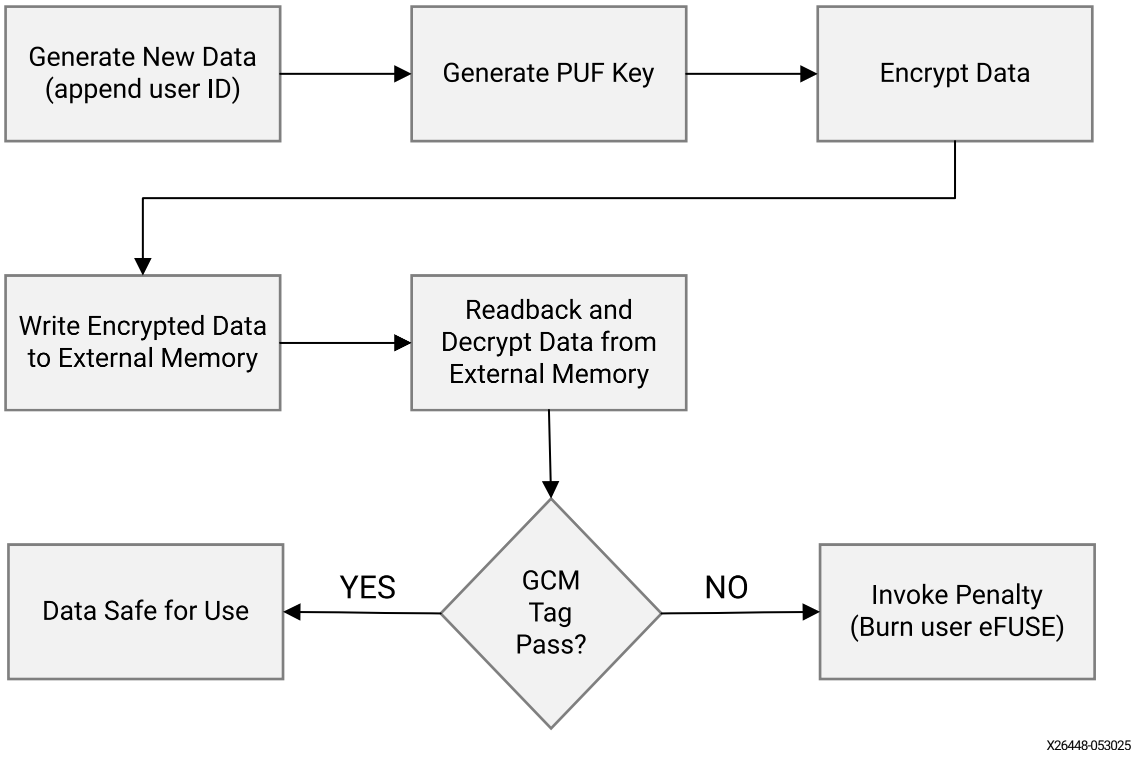
When the PUF is registered in eFUSEs and RSA authentication is enabled in eFUSEs, documented in *Programming BBRAM and eFUSEs* (XAPP1319), the PUF's device-unique encryption key can be used to encrypt and decrypt any user data. This encrypted data can then be stored externally to the Zynq UltraScale+ 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 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, AMD 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. 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)), 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. AMD 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.

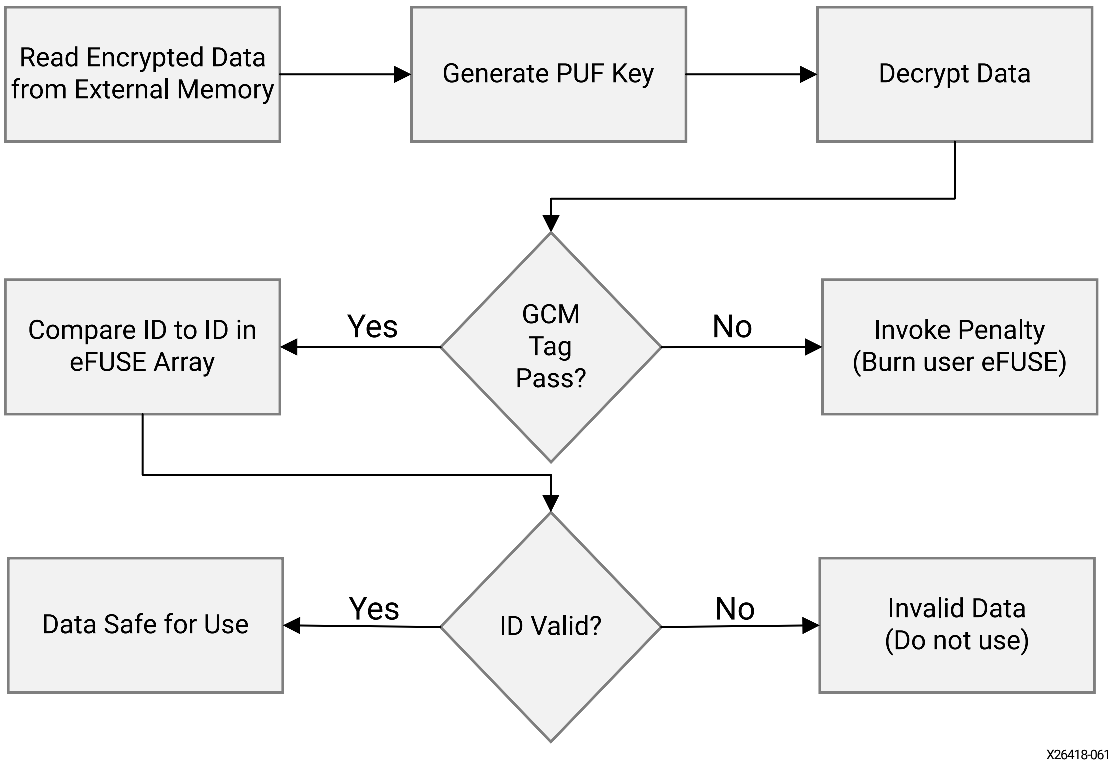
Figure 2: **Normal Encryption Process Using PUF**


Generate New Data (append user ID)
Generate PUF Key
Encrypt Data
Write Encrypted Data to External Memory
Readback and Decrypt Data from External Memory
GCM Tag Pass?
YES
NO
Data Safe for Use
Invoke Penalty (Burn user eFUSE)
X26448-053025

Decrypting external data using the PUF is shown in the following figure 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, the data is valid and can be used. Conversely, if the GCM tag authentication fails, 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**


Read Encrypted Data from External Memory
Generate PUF Key
Decrypt Data
GCM Tag Pass?
Yes
No
Compare ID to ID in eFUSE Array
Invoke Penalty (Burn user eFUSE)
ID Valid?
Yes
No
Data Safe for Use
Invalid Data (Do not use)
X26418-061025

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 x 2
- Optional platform JTAG hardware and associated cables
- Secure Digital (SD) card formatted using the FAT file system
- AMD Vitis™ integrated design environment (Vitis IDE) 2021.2
- Required design files, which can be downloaded here.

IMPORTANT! *Programming any of the eFUSE settings noted in the Zynq UltraScale+ MPSoC Security eFUSEs table in the Zynq UltraScale+ Device Technical Reference Manual (UG1085) precludes AMD test access. Consequently, AMD may not accept return material authorization (RMA) requests when these eFUSEs are programmed.*

Create a New Embedded Project for the Zynq UltraScale+ MPSoC

Perform the following steps to create a new embedded project for the AMD 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 *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209).

1. Open AMD 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 Vitis IDE from within the Vivado Design Suite.
3. Create a platform project using the XSA file exported from Vivado. The platform projects automatically create the 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 *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) so only a summary pertaining to this application note is documented here.

AES Key Generation

1. Create a new directory in the Vitis IDE workspace root directory (called `Keys`). The Vitis IDE root directory can be found at the same level as the `HelloWorld` folder.
2. Generate a device key and its associated IV, an operational key, and one partition block key and its associated IV.
3. 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
0000000100000001000000010000000100000001000000010000000100000001;
IV 1    000000010000000100000001;
```

Encrypting the boot image is not required to use the PUF for encrypting user data. However, AMD 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.

RECOMMENDED: *While this application note does not require the use of a secondary key set, AMD 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!*

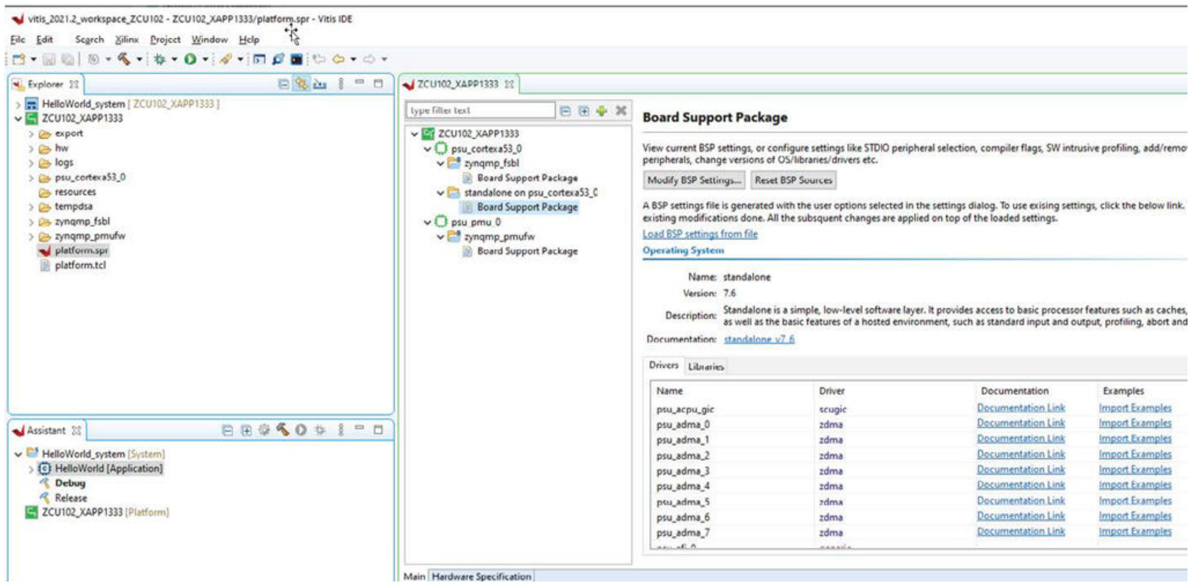
IMPORTANT! *Programming any of the noted eFUSE settings noted in the Zynq UltraScale+ MPSoC Security eFUSEs table in the Zynq UltraScale+ Device Technical Reference Manual (UG1085) preclude AMD test access. Consequently, AMD might not accept return material authorization (RMA) requests. See the important note below the Zynq UltraScale+ MPSoC Security eFUSEs table in the Zynq UltraScale+ Device Technical Reference Manual (UG1085).*

PUF eFUSE Settings

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

1. In the Vitis IDE workspace for this application note, right-click on the **platform.spr** that is located under the 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**



Board Support Package

View current BSP settings, or configure settings like STDIO peripheral selection, compiler flags, SW intrusive profiling, add/remo
peripherals, change versions of OS/libraries/drivers etc.

Modify BSP Settings... Reset BSP Sources

A BSP settings file is generated with the user options selected in the settings dialog. To use exising settings, click the below link.
existing modifications done. All the subsquent changes are applied on top of the loaded settings.

Load BSP settings from file

Operating System

Name: standalone
Version: 7.6
Description: Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and
Documentation: standalone_v7_6

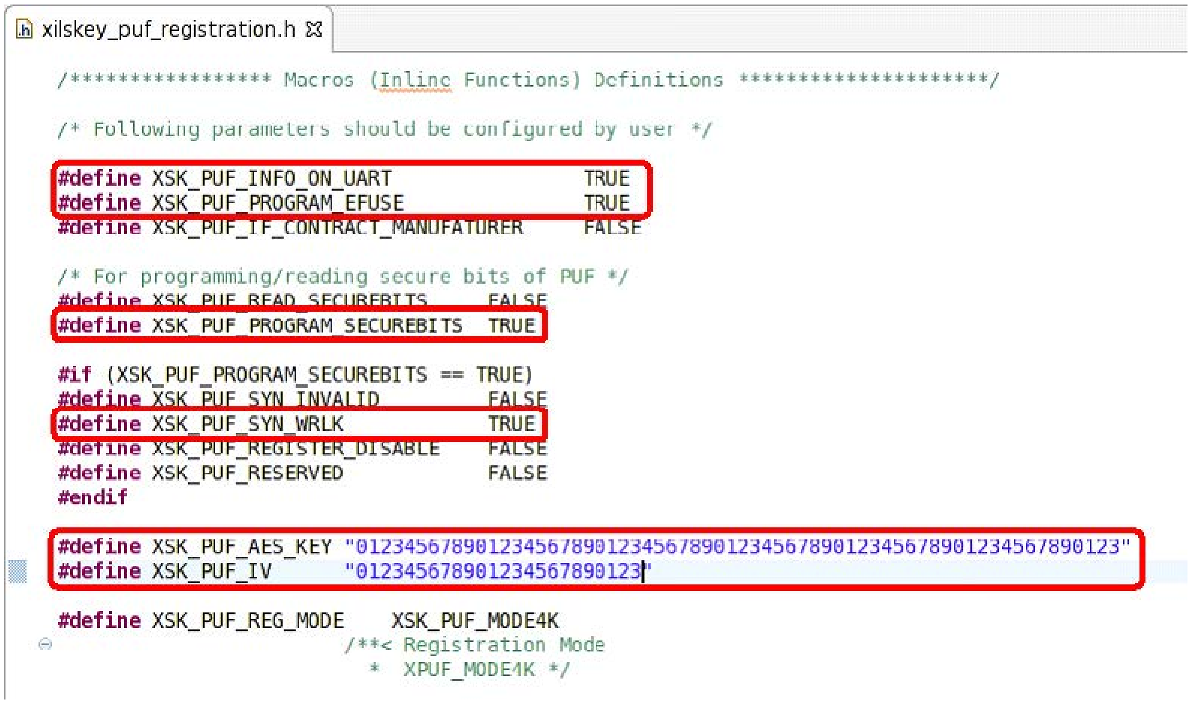
Drivers Libraries

Name	Driver	Documentation	Examples
psu_acpu_gic	scugic	Documentation Link	Import Examples
psu_adma_0	zdma	Documentation Link	Import Examples
psu_adma_1	zdma	Documentation Link	Import Examples
psu_adma_2	zdma	Documentation Link	Import Examples
psu_adma_3	zdma	Documentation Link	Import Examples
psu_adma_4	zdma	Documentation Link	Import Examples
psu_adma_5	zdma	Documentation Link	Import Examples
psu_adma_6	zdma	Documentation Link	Import Examples
psu_adma_7	zdma	Documentation Link	Import Examples

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 the **Libraries** tab in the 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**



```
/***************** Macros (Inline Functions) Definitions *********************/

/* Following parameters should be configured by user */

#define XSK_PUF_INFO_ON_UART                TRUE
#define XSK_PUF_PROGRAM_EFUSE               TRUE
#define XSK_PUF_IF_CONTRACT_MANUFATURER     FALSE

/* For programming/reading secure bits of PUF */
#define XSK_PUF_READ_SECUREBITS       FALSE
#define XSK_PUF_PROGRAM_SECUREBITS    TRUE

#if (XSK_PUF_PROGRAM_SECUREBITS == TRUE)
#define XSK_PUF_SYN_INVALID           FALSE
#define XSK_PUF_SYN_WRLK              TRUE
#define XSK_PUF_REGISTER_DISABLE      FALSE
#define XSK_PUF_RESERVED              FALSE
#endif

#define XSK_PUF_AES_KEY "0123456789012345678901234567890123456789012345678901234567890123"
#define XSK_PUF_IV      "012345678901234567890123"

#define XSK_PUF_REG_MODE     XSK_PUF_MODE4K
                        /**< Registration Mode
                          *  XPUF_MODE4K */
```

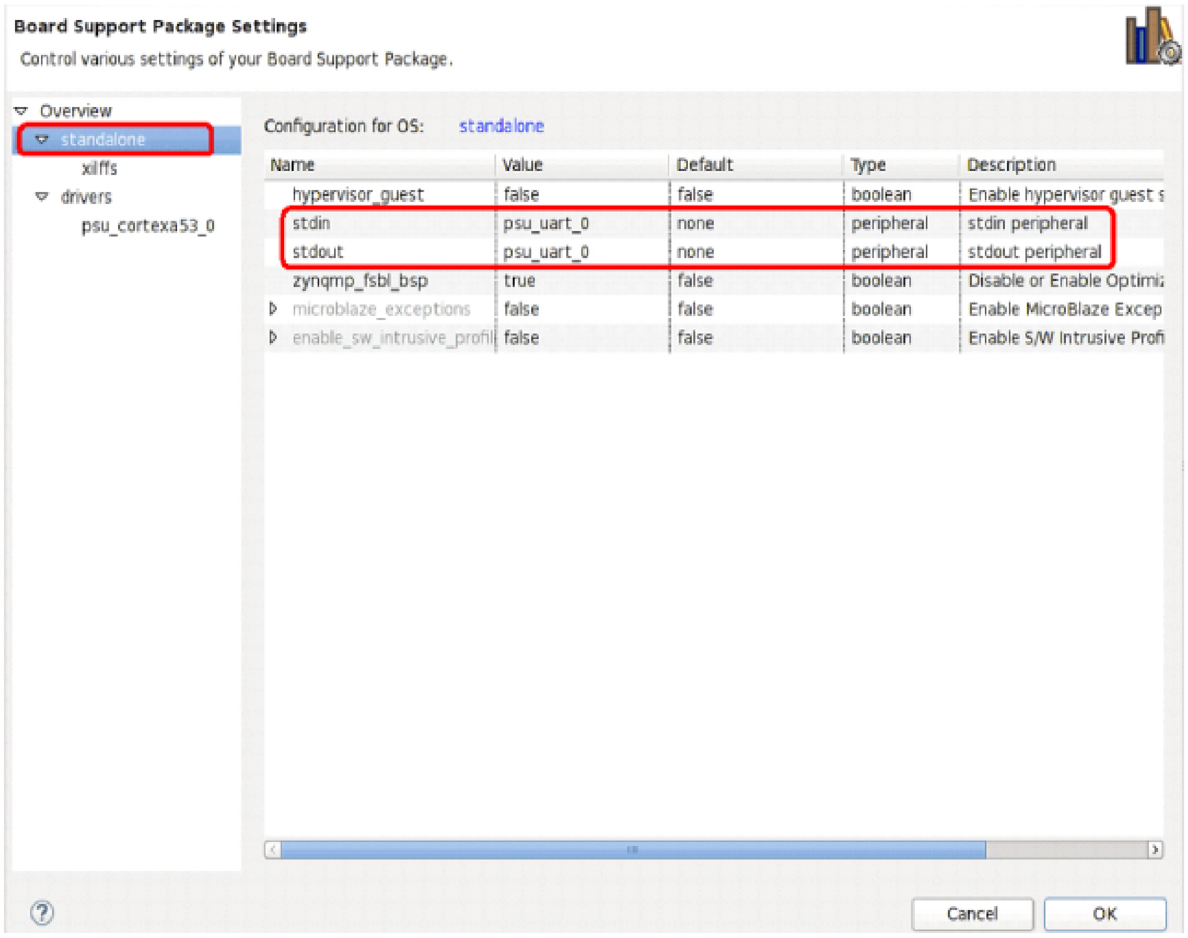
PUF Registration into eFUSEs

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 the 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 following figure.

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


Board Support Package Settings

Control various settings of your Board Support Package.

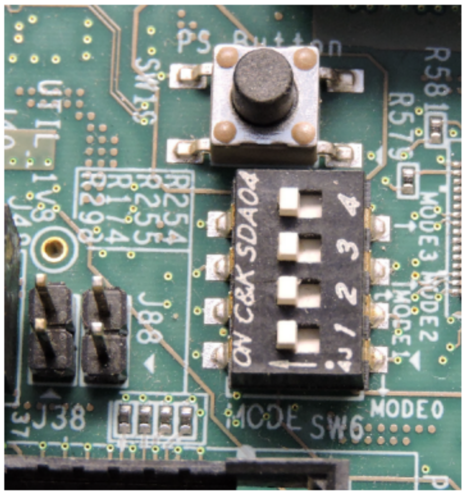
Overview
- standalone
  - xilffs
- drivers
  - psu_cortexa53_0

Configuration for OS: standalone

Name	Value	Default	Type	Description
hypervisor_guest	false	false	boolean	Enable hypervisor guest s
stdin	psu_uart_0	none	peripheral	stdin peripheral
stdout	psu_uart_0	none	peripheral	stdout peripheral
zynqmp_fsbl_bsp	true	false	boolean	Disable or Enable Optimi:
microblaze_exceptions	false	false	boolean	Enable MicroBlaze Excep
enable_sw_intrusive_profil	false	false	boolean	Enable S/W Intrusive Profi

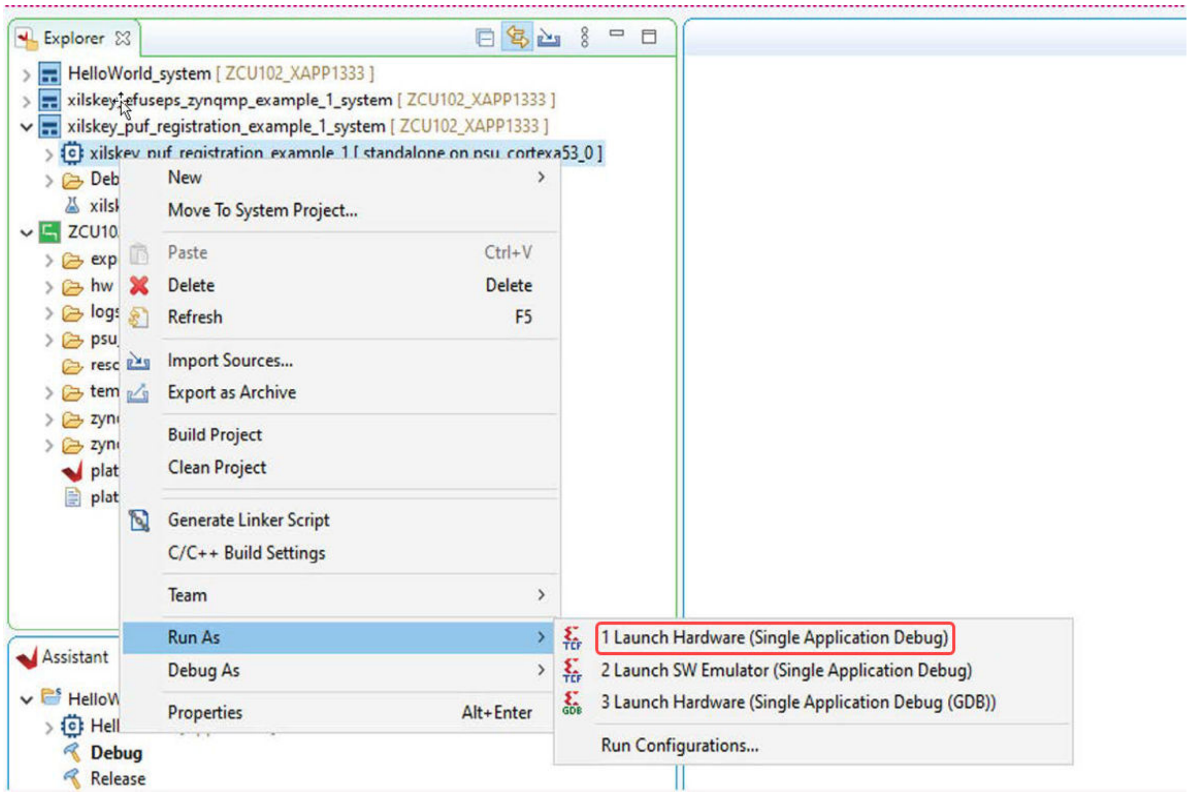
4. Ensure the stdin and stdout functions are mapped to psu_uart_0 and click **OK**.
5. In the Vitis IDE explorer view, on the left, right-click **xilskey_puf_registration_example_1_system** and select **Build Project**.
6. Turn power off to the ZCU102 board.
7. Connect either the USB JTAG connector J2 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 following figure.

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



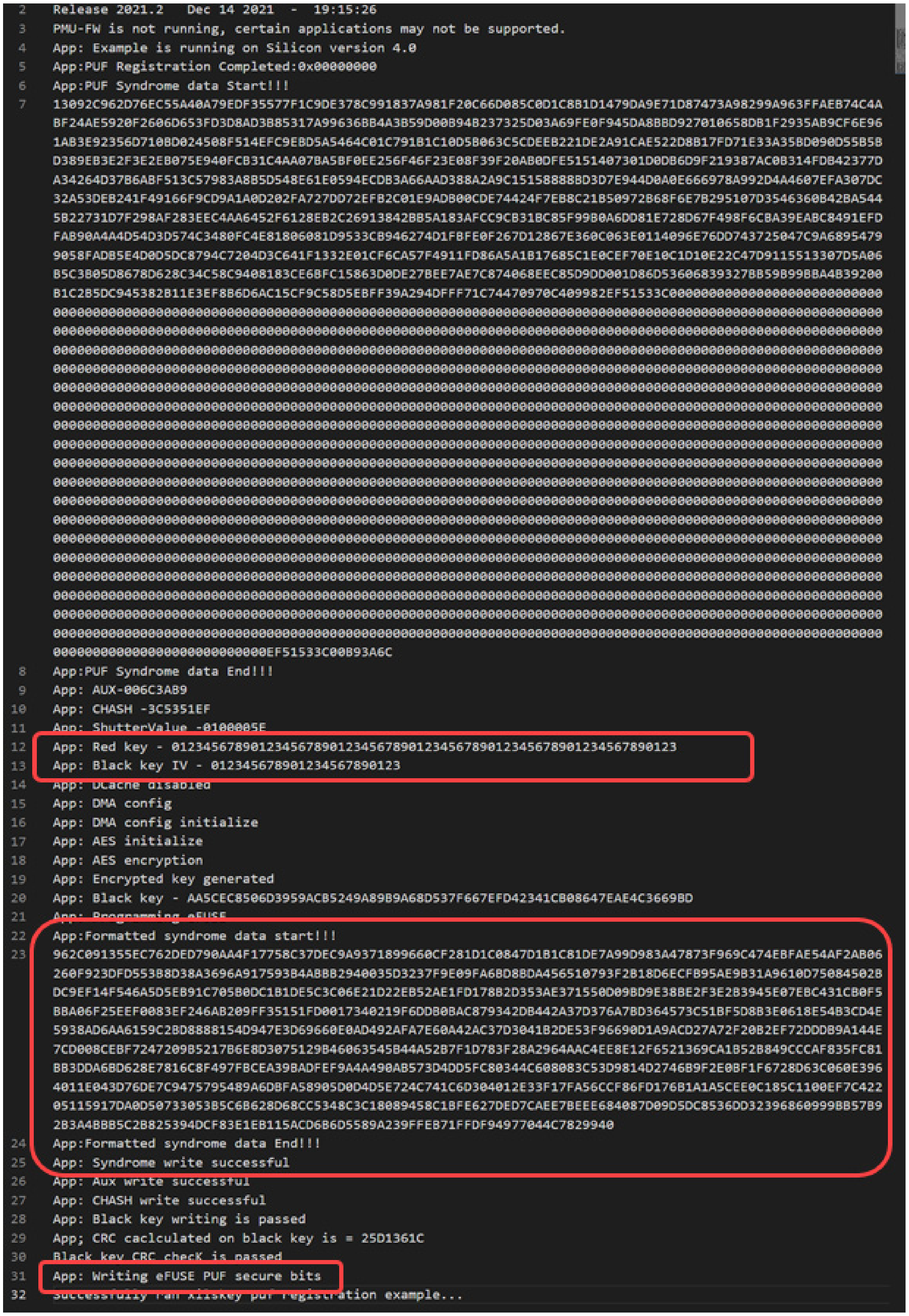
11. Power on the ZCU102 board using switch SW1.
12. Right-click **xilskey_puf_registration_example_1** → **Run As** → **Launch Hardware (Single Application Debug)** as shown in the following figure.

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


Explorer
HelloWorld_system [ZCU102_XAPP1333]
xilskey_efuseps_zynqmp_example_1_system [ZCU102_XAPP1333]
xilskey_puf_registration_example_1_system [ZCU102_XAPP1333]
xilskey_puf_registration_example_1 [standalone on psu_cortexa53_0]
New
Move To System Project...
Paste Ctrl+V
Delete Delete
Refresh F5
Import Sources...
Export as Archive
Build Project
Clean Project
Generate Linker Script
C/C++ Build Settings
Team
Run As
Debug As
Properties Alt+Enter
1 Launch Hardware (Single Application Debug)
2 Launch SW Emulator (Single Application Debug)
3 Launch Hardware (Single Application Debug (GDB))
Run Configurations...
Assistant
Debug
Release

The PUF registration application starts running and outputs information to the terminal as shown in the following figure. 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**



```
Release 2021.2   Dec 14 2021  -  19:15:26
PMU-FW is not running, certain applications may not be supported.
App: Example is running on Silicon version 4.0
App:PUF Registration Completed:0x00000000
App:PUF Syndrome data Start!!!
13092C962D76EC55A40A79EDF35577F1C9DE378C991837A981F20C66D085C0D1C8B1D1479DA9E71D87473A98299A963FFAEB74C4A
BF24AE5920F2606D653FD3D8AD3B85317A99636BB4A3B59D00B94B237325D03A69FE0F945DA8BBD927010658DB1F2935AB9CF6E96
1AB3E92356D7108D024508F514EFC9EBD5A5464C01C791B1C10D5B063C5CDEEB221DE2A91CAE522D8B17FD71E33A35BD090D55B58
D389EB3E2F3E2EB075E940FCB31C4AA07BA5BF0EE256F46F23E08F39F20AB0DFE5151407301D0DB6D9F219387AC0B314FD842377D
A34264D37B6ABF513C57983A8B5D548E61E0594ECDB3A66AAD388A2A9C15158888BD3D7E944D0A0E666978A992D4A4607EFA307DC
32A53DEB241F49166F9CD9A1A0D202FA727DD72EFB2C01E9ADB00CDE74424F7EB8C21B50972B68F6E7B295107D3546360B42BA544
5B22731D7F298AF283EEC4AA6452F6128EB2C26913842BB5A183AFCC9CB31BC85F99B0A6DD81E728D67F498F6CBA39EABC8491EFD
FAB90A4A4D54D3D574C3480FC4E81806081D9533CB946274D1FBFE0F267D12867E360C063E0114096E76DD743725047C9A6895479
9058FADB5E4D0D5DC8794C7204D3C641F1332E01CF6CA57F4911FD86A5A1B17685C1E0CEF70E10C1D10E22C47D9115513307D5A06
B5C3B05D8678D628C34C58C9408183CE68FC15863D0DE27BEE7AE7C874068EEC85D9DD001D86D53606839327BB59B99BBA4B39200
B1C2B5DC945382B11E3EF8B6D6AC15CF9C58D5EBFF39A294DFFF71C74470970C409982EF51533C00000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000EF51533C00B93A6C
App:PUF Syndrome data End!!!
App: AUX-006C3AB9
App: CHASH -3C5351EF
App: ShutterValue -0100005E
App: Red key - 0123456789012345678901234567890123456789012345678901234567890123
App: Black key IV - 012345678901234567890123
App: DCache disabled
App: DMA config
App: DMA config initialize
App: AES initialize
App: AES encryption
App: Encrypted key generated
App: Black key - AA5CEC8506D3959ACB5249A89B9A68D537F667EFD42341CB08647EAE4C3669BD
App: Programming eFUSE
App:Formatted syndrome data start!!!
962C091355EC762DED790AA4F17758C37DEC9A9371899660CF281D1C0847D1B1C81DE7A99D983A47873F969C474EBFAE54AF2AB06
260F923DFD553B8D38A3696A917593B4A8BB2940035D3237F9E09FA6BD88DA456510793F2B18D6ECFB95AE9B31A9610D750845028
DC9EF14F546A5D5EB91C705B0DC1B1DE5C3C06E21D22EB52AE1FD178B2D353AE371550D09BD9E38BE2F3E2B3945E07EBC431CB0F5
BBA06F25EEF0083EF246AB209FF35151FD0017340219F6DDB0BAC879342DB442A37D376A7BD364573C51BF5D8B3E0618E54B3CD4E
5938AD6AA6159C2BD8888154D947E3D69660E0AD492AFA7E60A42AC37D3041B2DE53F96690D1A9ACD27A72F20B2EF72DDD89A144E
7CD008CEBF7247209B5217B6E8D3075129B46063545B44A52B7F1D783F28A2964AAC4EE8E12F6521369CA1B52B849CCCAF835FC81
BB3DDA6BD628E7816C8F497FBCEA39BADFEF9A4A490AB573D4DD5FC80344C608083C53D9814D2746B9F2E0BF1F6728D63C060E396
4011E043D76DE7C9475795489A6DBFA58905D0D4D5E724C741C6D304012E33F17FA56CCF86FD176B1A1A5CEE0C185C1100EF7C422
05115917DA0D5073305385C6B628D68CC5348C3C18089458C1BFE627DED7CAEE7BEEE684087D09D5DC8536DD32396860999BB5789
2B3A4BBB5C2B825394DCF83E1EB115ACD6B6D5589A239FFEB71FFDF94977044C7829940
App:Formatted syndrome data End!!!
App: Syndrome write successful
App: Aux write successful
App: CHASH write successful
App: Black key writing is passed
App; CRC caclculated on black key is = 25D1361C
Black key CRC check is passed
App: Writing eFUSE PUF secure bits
Successfully ran Xilskey puf registration example...
```

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!*

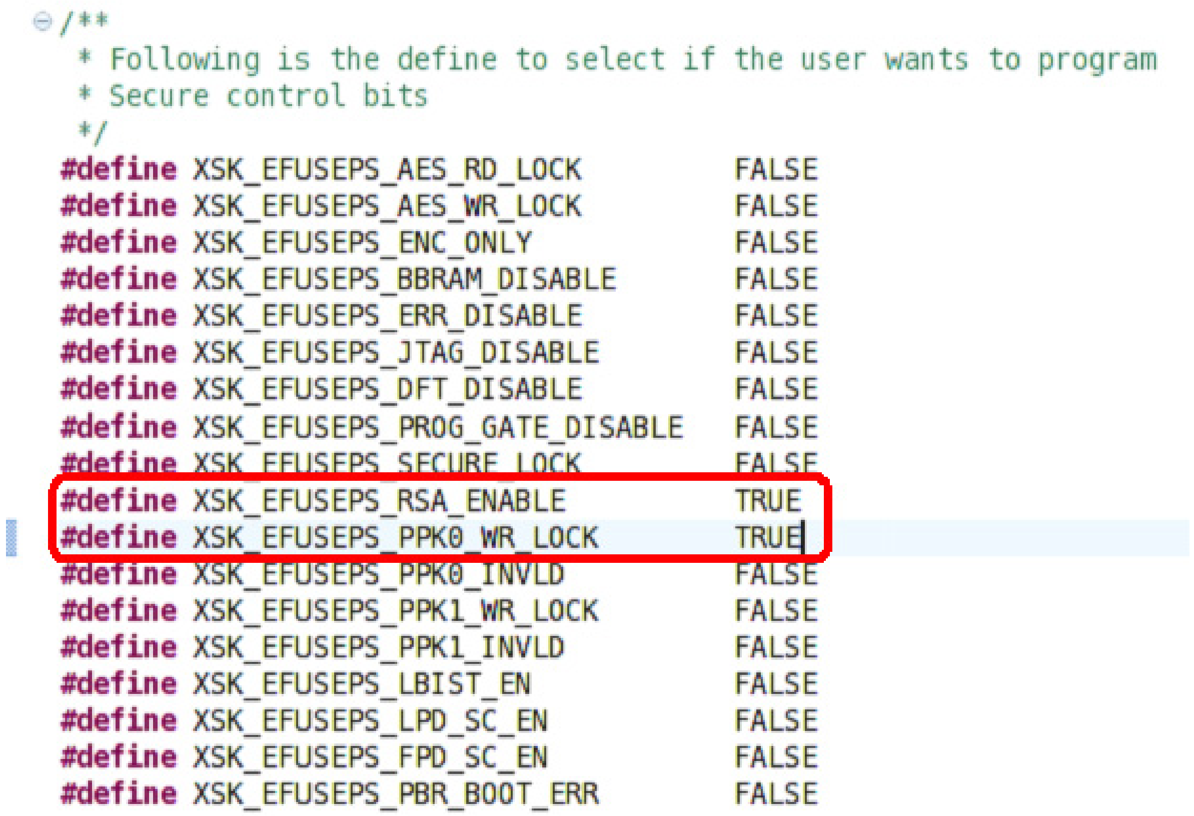
IMPORTANT! *Programming any of the RSA_EN eFUSE settings preclude AMD test access. Consequently, AMD might not accept return material authorization (RMA) requests.*

RSA eFUSE Settings

RSA eFUSE registration is covered in detail in Programming eFUSEs for AES and RSA Cryptographic Functions in the *Programming BBRAM and eFUSEs* (XAPP1319), so only a summary pertaining to this application note is covered here.

1. Right-click on the **platform.spr** that is located under the 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 opened and select the **Libraries** tab in the 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 following figure:

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

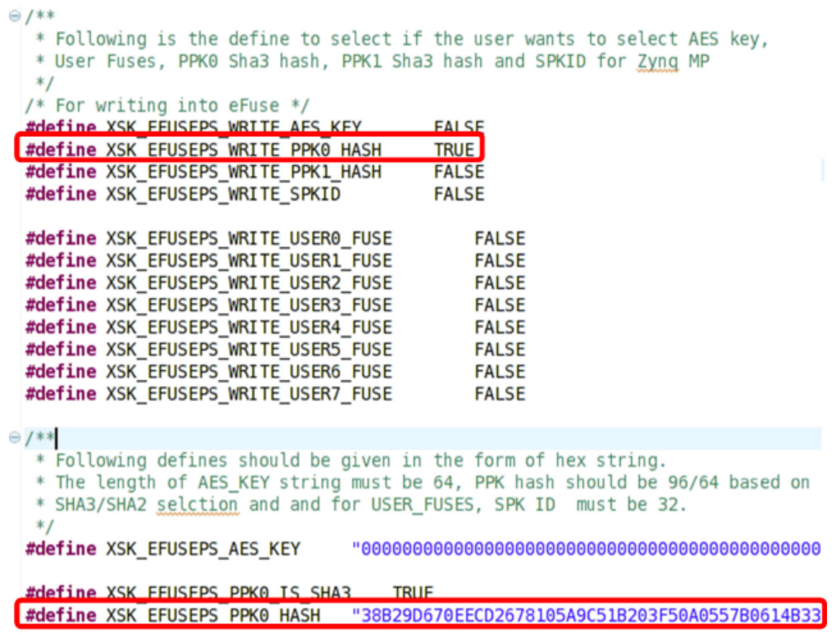


```
/**
 * Following is the define to select if the user wants to program
 * Secure control bits
 */
#define XSK_EFUSEPS_AES_RD_LOCK          FALSE
#define XSK_EFUSEPS_AES_WR_LOCK          FALSE
#define XSK_EFUSEPS_ENC_ONLY             FALSE
#define XSK_EFUSEPS_BBRAM_DISABLE        FALSE
#define XSK_EFUSEPS_ERR_DISABLE          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
```

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 is shown in Figure 11. These settings using the example 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.

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



```
/**
 * 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     "000000000000000000000000000000000000000000000

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

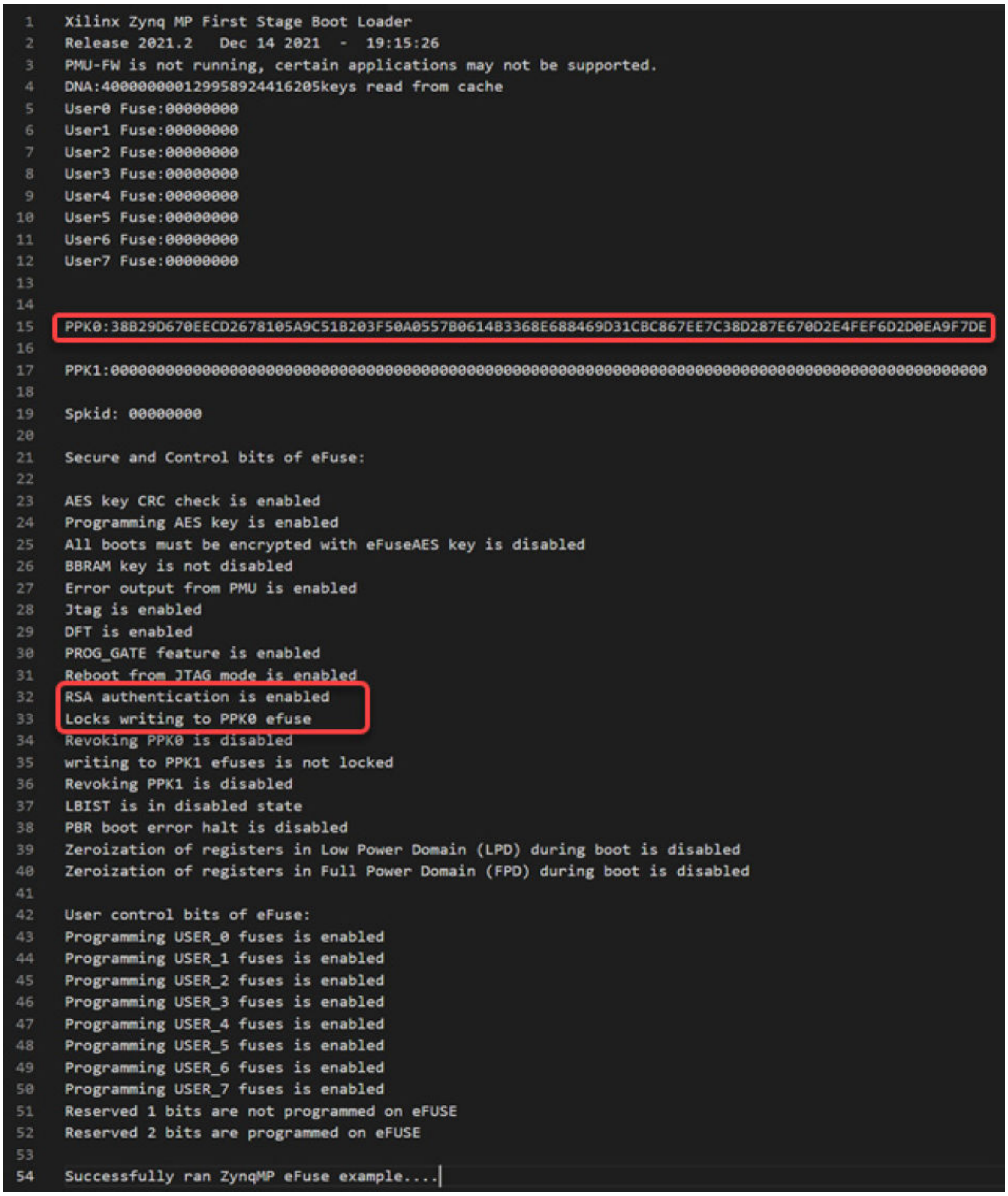
Programming RSA eFUSEs

Program the RSA eFUSEs by performing the following steps:

1. Right-click on the **platform.spr** that is located under the ZCU102_XAPP1333 platform in the Explorer view and click **Open**.
2. Select **Board Support Package** under the 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 the Vitis IDE explorer view, on the left, right-click **xilskey_efuseps_zynqmp_example_1_system** and select the **Build Project** option. This might 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. Line 32 from the terminal matches the SHA3 output that was generated and stored in the `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.

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



```
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

PPK1:000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

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....
```

PUF Encryption and Decryption

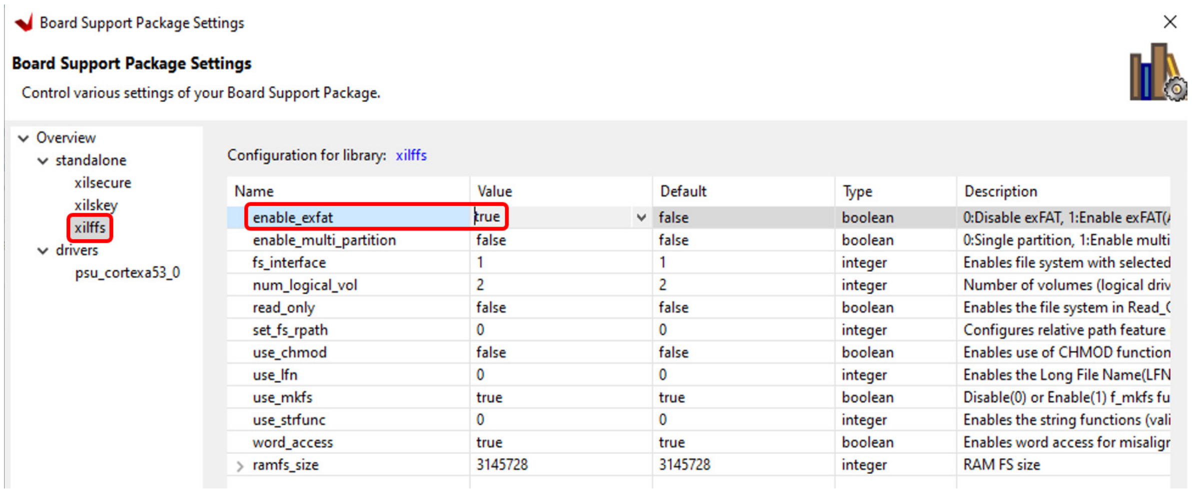
PUF Encryption Decryption Demo Application

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 the 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 the **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 the Vitis IDE – Standalone Library Configuration**



Board Support Package Settings

Control various settings of your Board Support Package.

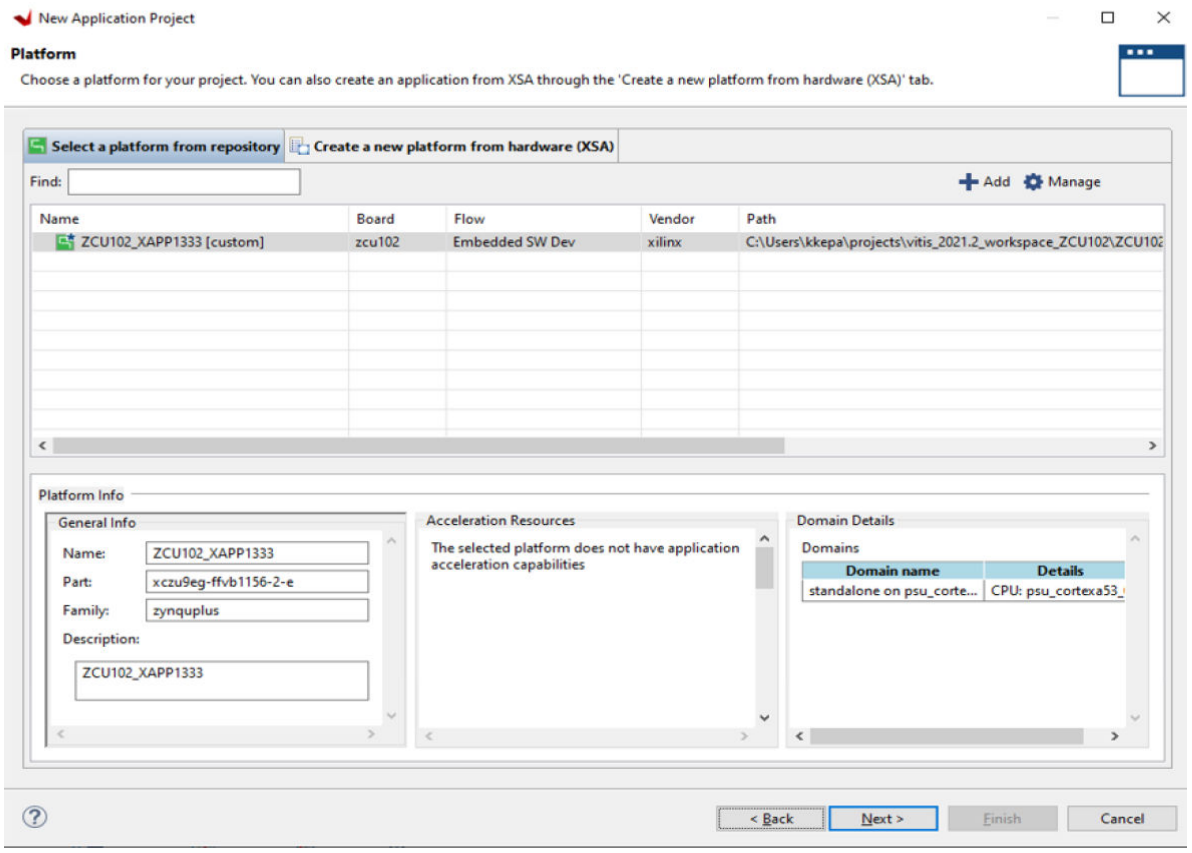
Configuration for library: xilffs

Name	Value	Default	Type	Description
enable_exfat	true	false	boolean	0:Disable exFAT, 1:Enable exFAT(
enable_multi_partition	false	false	boolean	0:Single partition, 1:Enable multi
fs_interface	1	1	integer	Enables file system with selected
num_logical_vol	2	2	integer	Number of volumes (logical driv
read_only	false	false	boolean	Enables the file system in Read_(
set_fs_rpath	0	0	integer	Configures relative path feature
use_chmod	false	false	boolean	Enables use of CHMOD function
use_lfn	0	0	integer	Enables the Long File Name(LFN
use_mkfs	true	true	boolean	Disable(0) or Enable(1) f_mkfs fu
use_strfunc	0	0	integer	Enables the string functions (vali
word_access	true	true	boolean	Enables word access for misaligr
ramfs_size	3145728	3145728	integer	RAM FS size

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 the Vitis IDE, click **File** → **New** → **Application Project**. If the Create a New Application Project window appears, click **Next**.
6. Select the **ZCU102_XAPP1333** platform in the Platform Window and click **Next**.
7. Type in `ExternalKeyStorage` in the Application project name.
8. Leave the 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 theExternalKeyStorage 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 the Vitis IDE– Platform Selection**


New Application Project
Platform
Choose a platform for your project. You can also create an application from XSA through the 'Create a new platform from hardware (XSA)' tab.
Select a platform from repository
Create a new platform from hardware (XSA)
Find:
Add
Manage

Name	Board	Flow	Vendor	Path
ZCU102_XAPP1333 [custom]	zcu102	Embedded SW Dev	xilinx	C:\Users\kkepa\projects\vitis_2021.2_workspace_ZCU102\ZCU102

Platform Info

General Info
Name: ZCU102_XAPP1333
Part: xczu9eg-ffvb1156-2-e
Family: zynquplus
Description: ZCU102_XAPP1333

Acceleration Resources
The selected platform does not have application acceleration capabilities

Domain Details
Domains

Domain name	Details
standalone on psu_corte...	CPU: psu_cortexa53_

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

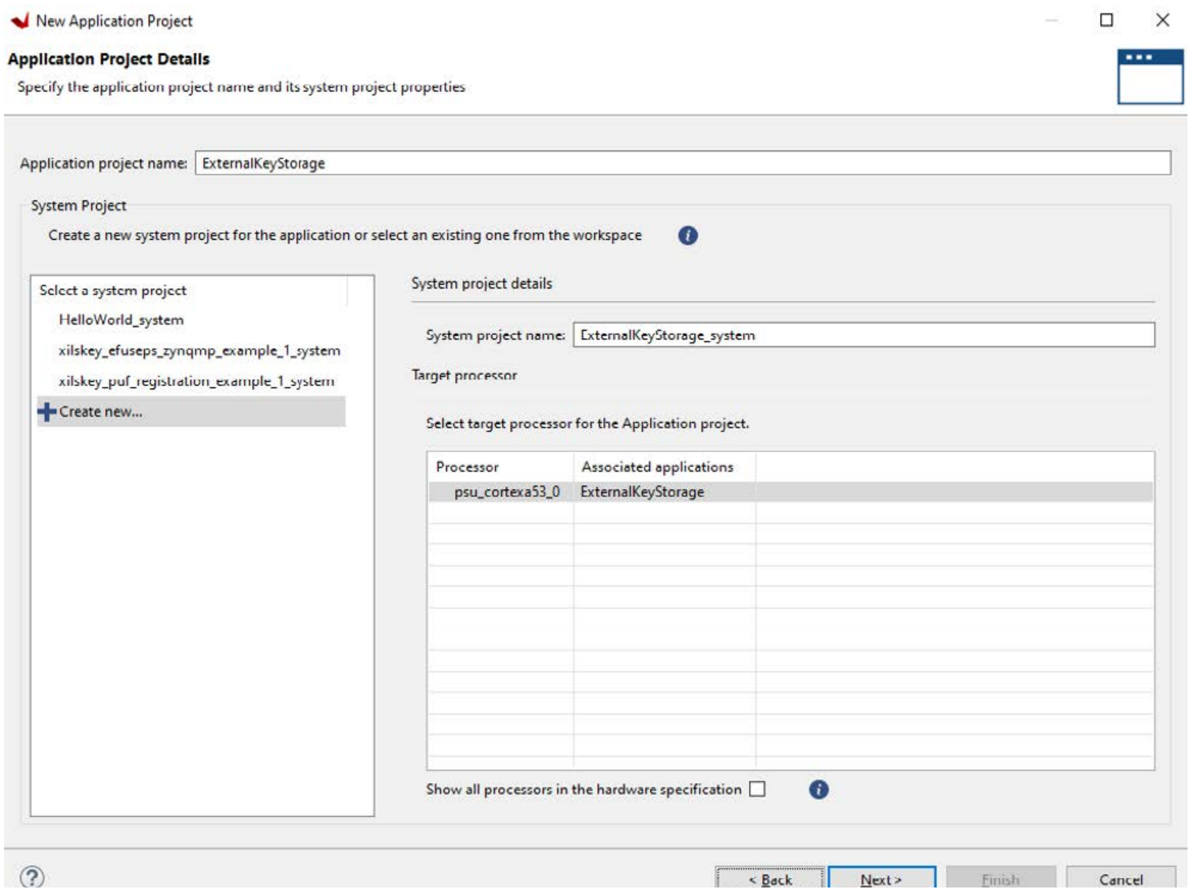
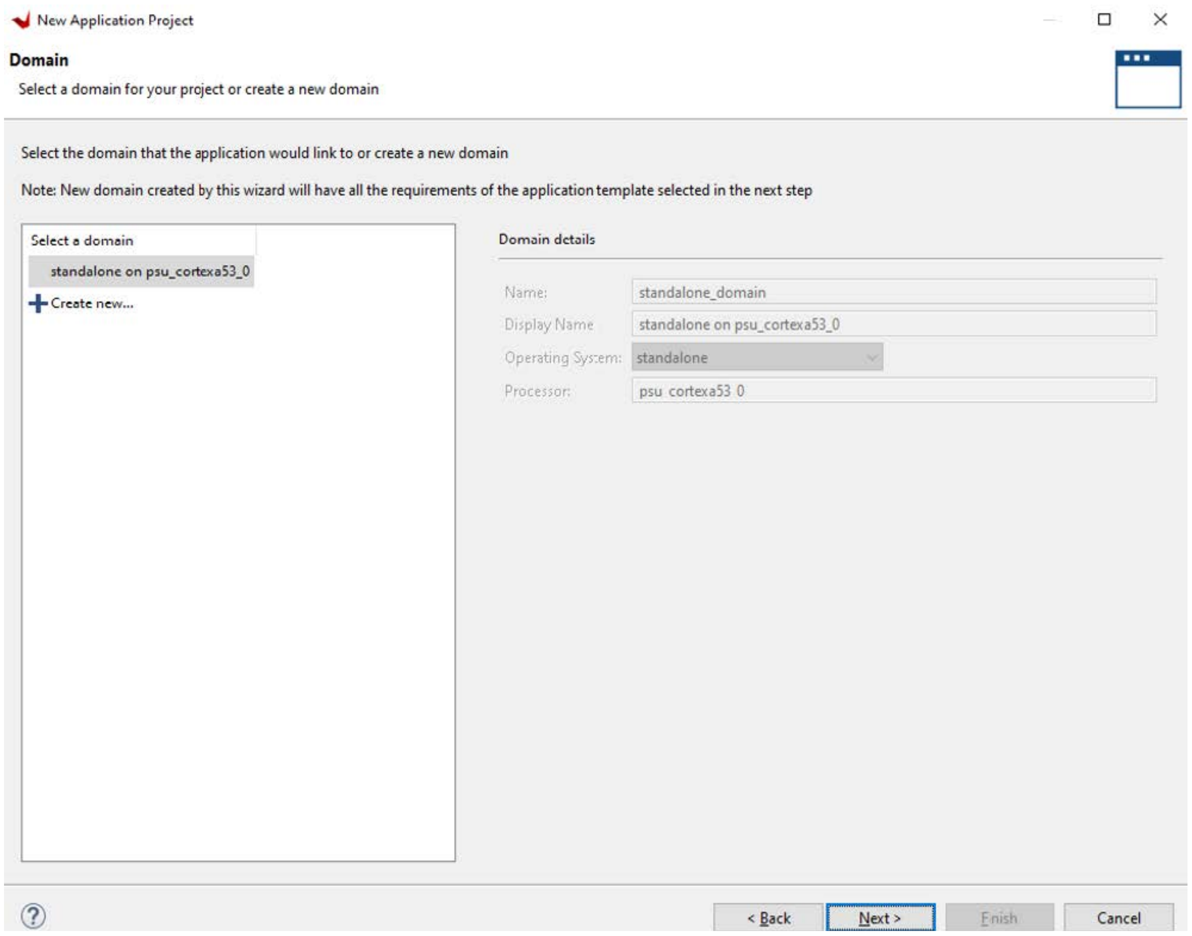

New Application Project
Application Project Details
Specify the application project name and its system project properties
Application project name: ExternalKeyStorage
System Project
Create a new system project for the application or select an existing one from the workspace
Select a system project
HelloWorld_system
xilskey_efuseps_zynqmp_example_1_system
xilskey_puf_registration_example_1_system
Create new...
System project details
System project name: ExternalKeyStorage_system
Target processor
Select target processor for the Application project.
Processor
Associated applications
psu_cortexa53_0
ExternalKeyStorage
Show all processors in the hardware specification
< Back
Next >
Finish
Cancel

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



Domain

Select a domain for your project or create a new domain

Select the domain that the application would link to or create a new domain

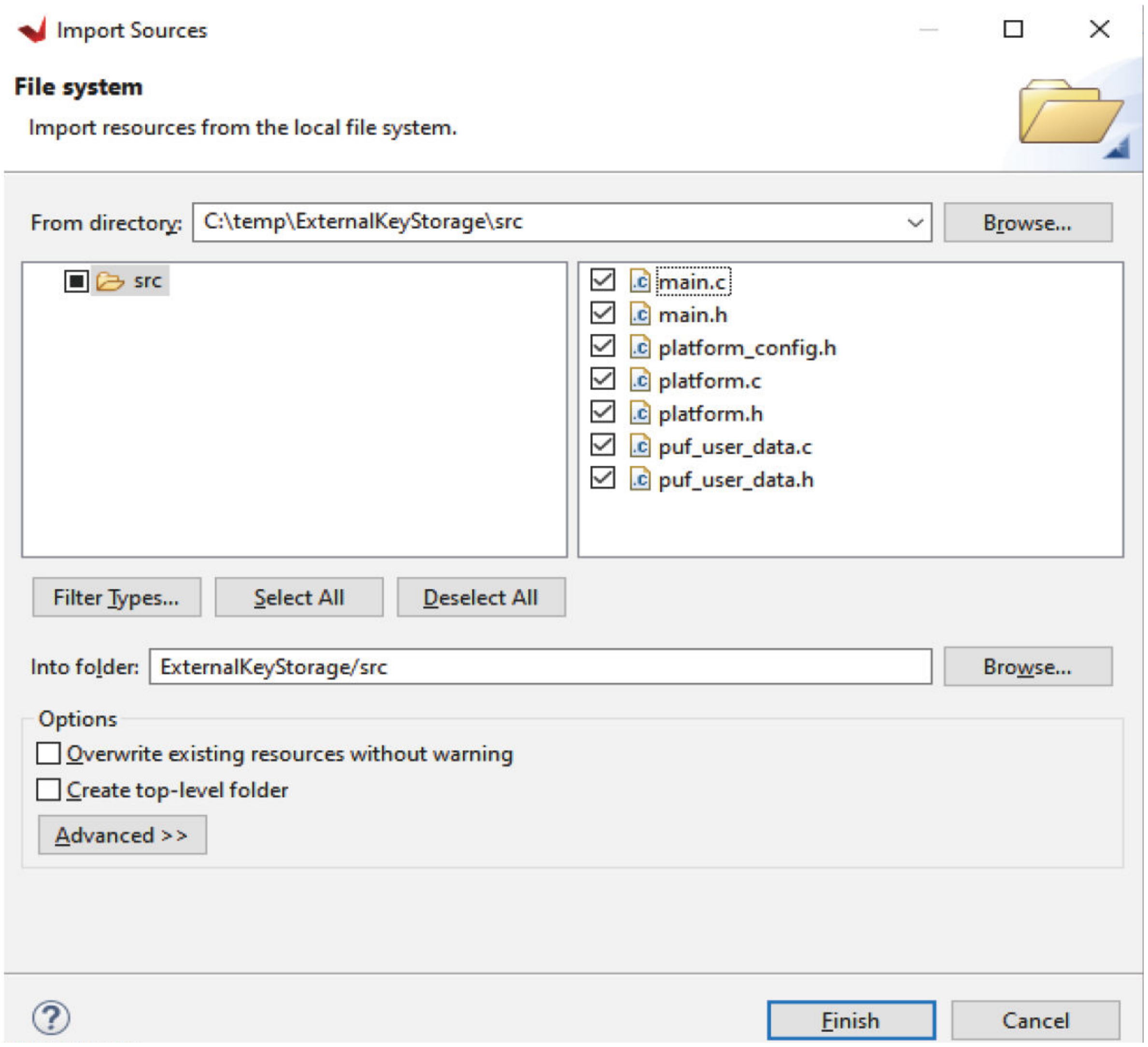
Note: New domain created by this wizard will have all the requirements of the application template selected in the next step

Select a domain
- standalone on psu_cortexa53_0
- Create new...

Domain details

Name:	standalone_domain
Display Name	standalone on psu_cortexa53_0
Operating System:	standalone
Processor:	psu_cortexa53_0

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



From directory: C:\temp\ExternalKeyStorage\src

- [✓] src
- [✓] main.c
- [✓] main.h
- [✓] platform_config.h
- [✓] platform.c
- [✓] platform.h
- [✓] puf_user_data.c
- [✓] puf_user_data.h

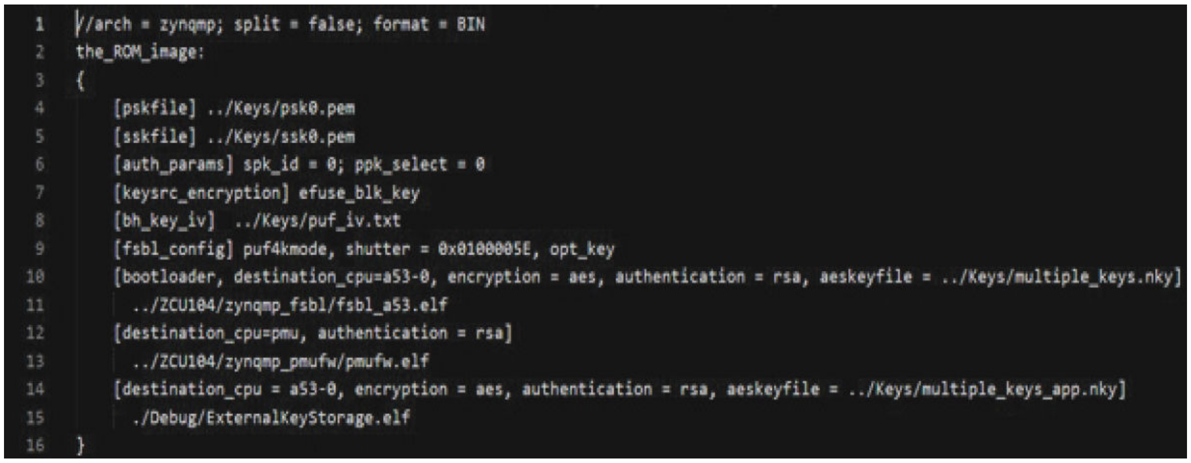
Into folder: ExternalKeyStorage/src

Options
- [ ] Overwrite existing resources without warning
- [ ] Create top-level folder

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 the Vitis IDE does not currently support this feature.

18. Update the contents of the file to the contents shown in the following figure using the correct paths.

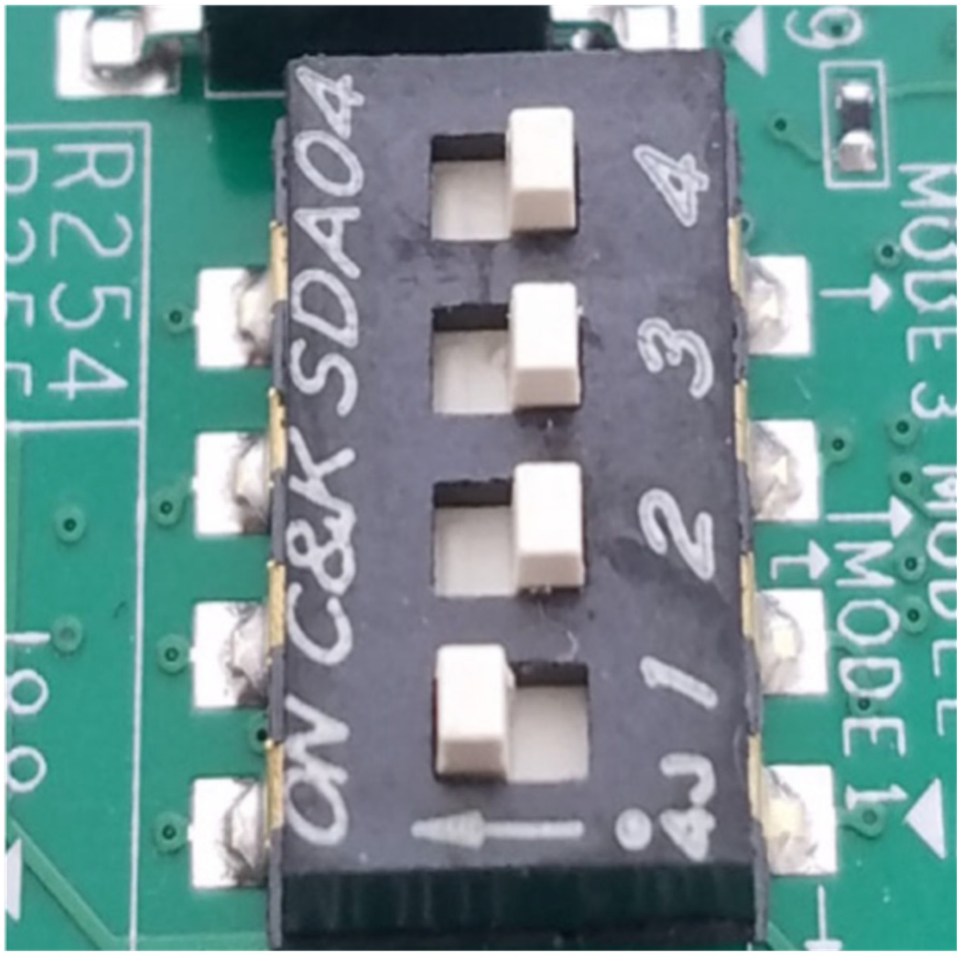
Figure 18: **ExternalKeyStorage.bif File**



```
//arch = zynqmp; split = false; format = BIN
the_ROM_image:
{
    [pskfile] ../Keys/psk0.pem
    [sskfile] ../Keys/ssk0.pem
    [auth_params] spk_id = 0; ppk_select = 0
    [keysrc_encryption] efuse_blk_key
    [bh_key_iv]  ../Keys/puf_iv.txt
    [fsbl_config] puf4kmode, shutter = 0x0100005E, opt_key
    [bootloader, destination_cpu=a53-0, encryption = aes, authentication = rsa, aeskeyfile = ../Keys/multiple_keys.nky]
      ../ZCU104/zynqmp_fsbl/fsbl_a53.elf
    [destination_cpu=pmu, authentication = rsa]
      ../ZCU104/zynqmp_pmufw/pmufw.elf
    [destination_cpu = a53-0, encryption = aes, authentication = rsa, aeskeyfile = ../Keys/multiple_keys_app.nky]
      ./Debug/ExternalKeyStorage.elf
}
```

19. Build the ExternalKeyStorage project in the Vitis IDE.
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.

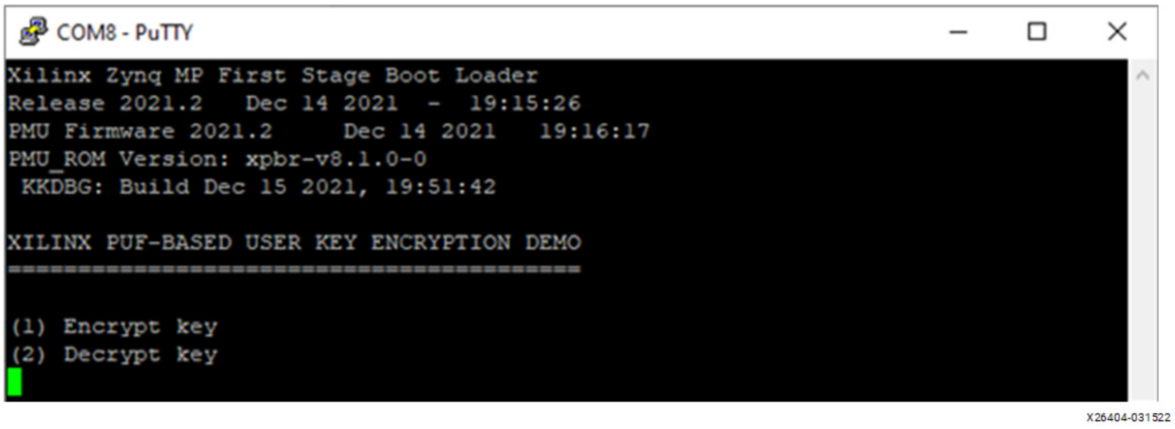
Figure 19: **ZCU102 SD Boot Mode Switch Setting**



26. On the ZCU102 development board, set 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**

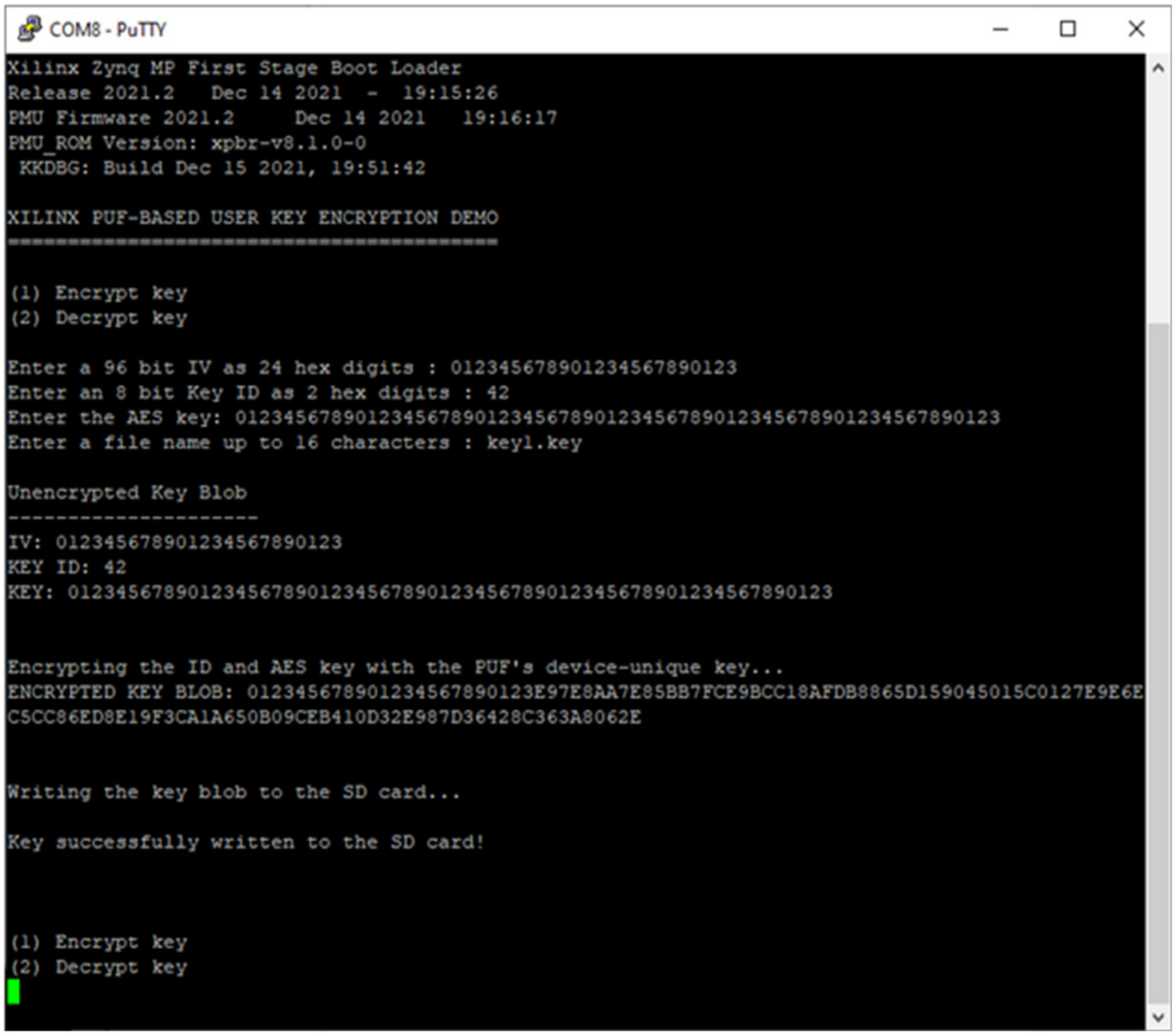


```
Xilinx Zynq MP First Stage Boot Loader
Release 2021.2   Dec 14 2021  -  19:15:26
PMU Firmware 2021.2     Dec 14 2021   19:16:17
PMU_ROM Version: xpbr-v8.1.0-0
 KKDBG: Build Dec 15 2021, 19:51:42

XILINX PUF-BASED USER KEY ENCRYPTION DEMO
=========================================

(1) Encrypt key
(2) Decrypt key
```

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.

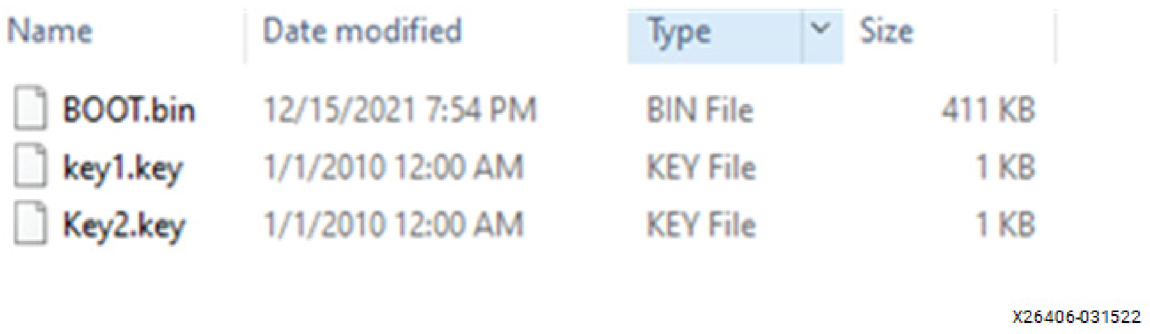
       Note: *Do not reuse IV. Per the AES-GCM standard, the IV should be a new one per every use.*
    b. Enter an 8-bit key ID. Use an ID of 42 for this key. An ID of 0 is mapped to user eFUSE 0 bit 0, an ID of 1 is mapped to user eFUSE 0 bit 1, ... , an ID of 255 is mapped to user eFUSE 7 bit 31.
    c. 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 (for example, `Key2.key`).
32. Power off the ZCU102 board.
33. Remove the SD card and insert the card into an 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**


COM8 - PuTTY

```
Xilinx Zynq MP First Stage Boot Loader
Release 2021.2   Dec 14 2021  -  19:15:26
PMU Firmware 2021.2     Dec 14 2021   19:16:17
PMU_ROM Version: xpbr-v8.1.0-0
 KKDBG: Build Dec 15 2021, 19:51:42

XILINX PUF-BASED USER KEY ENCRYPTION DEMO
=========================================

(1) Encrypt key
(2) Decrypt key

Enter a 96 bit IV as 24 hex digits : 012345678901234567890123
Enter an 8 bit Key ID as 2 hex digits : 42
Enter the AES key: 0123456789012345678901234567890123456789012345678901234567890123
Enter a file name up to 16 characters : key1.key

Unencrypted Key Blob
---------------------
IV: 012345678901234567890123
KEY ID: 42
KEY: 0123456789012345678901234567890123456789012345678901234567890123


Encrypting the ID and AES key with the PUF's device-unique key...
ENCRYPTED KEY BLOB: 012345678901234567890123E97E8AA7E85BB7FCE9BCC18AFDB8865D159045015C0127E9E6E
C5CC86ED8E19F3CA1A650B09CEB410D32E987D36428C363A8062E


Writing the key blob to the SD card...

Key successfully written to the SD card!



(1) Encrypt key
(2) Decrypt key
```

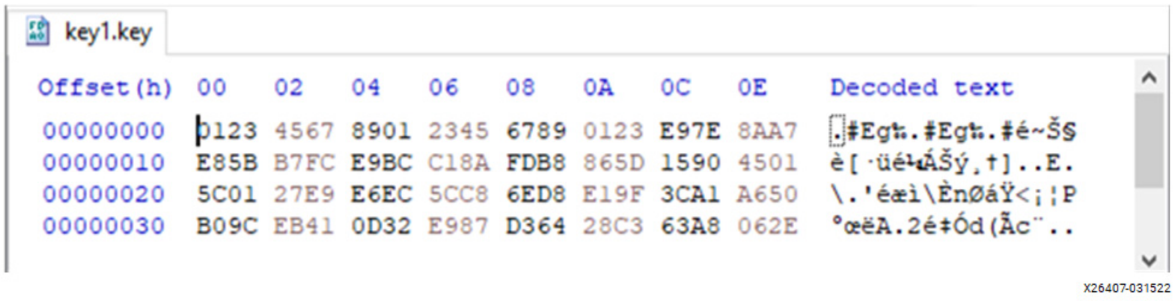
Figure 22: **Directory Contents of the SD Card after Writing the Encrypted Key**



Name	Date modified	Type	Size
BOOT.bin	12/15/2021 7:54 PM	BIN File	411 KB
key1.key	1/1/2010 12:00 AM	KEY File	1 KB
Key2.key	1/1/2010 12:00 AM	KEY File	1 KB

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**



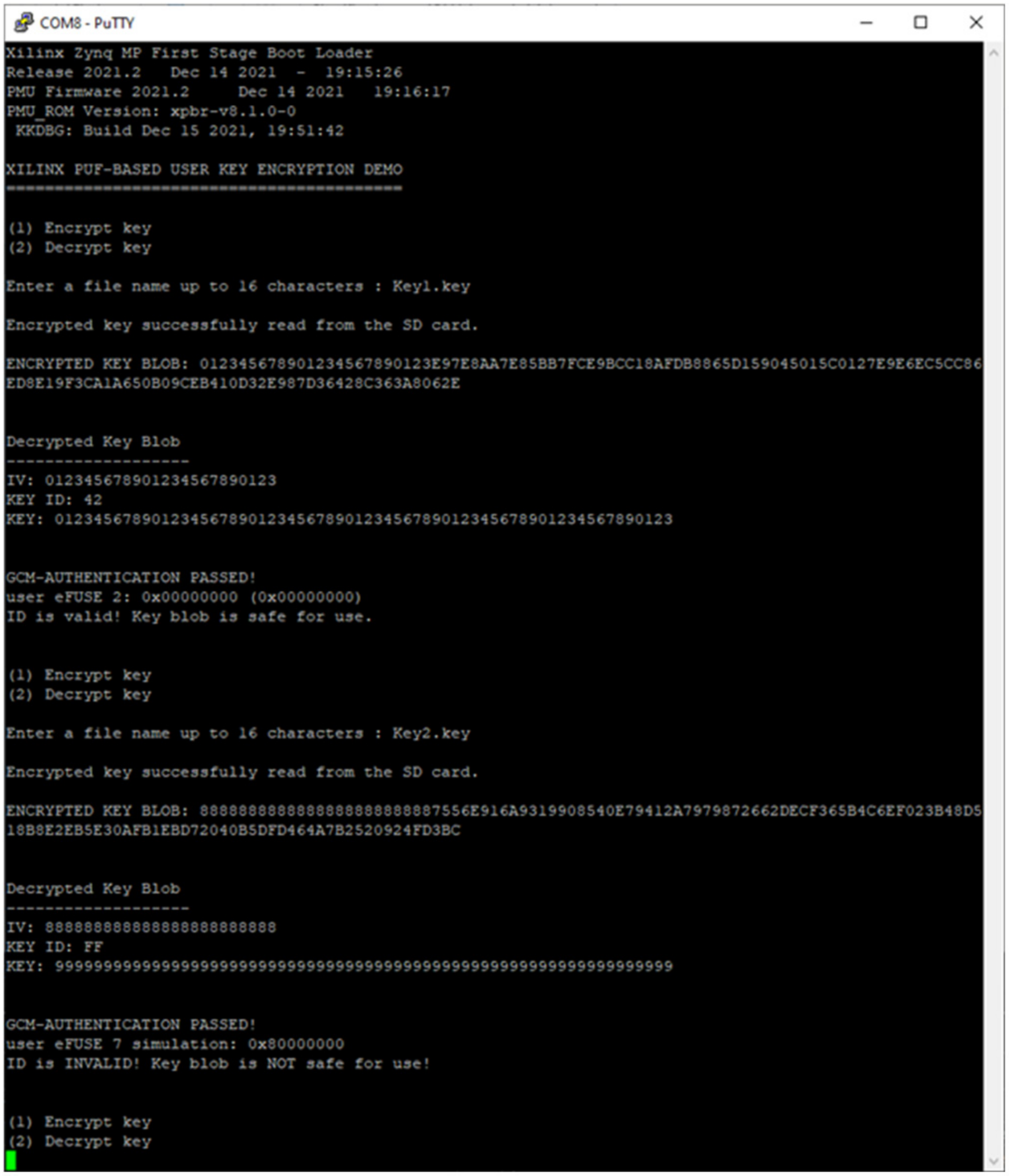
key1.key

```
Offset(h) 00   02   04   06   08   0A   0C   0E    Decoded text
00000000  0123 4567 8901 2345 6789 0123 E97E 8AA7  .#Eg‰.#Eg‰.#é~Š§
00000010  E85B B7FC E9BC C18A FDB8 865D 1590 4501  è[·üé¼ÁŠý¸†]..E.
00000020  5C01 27E9 E6EC 5CC8 6ED8 E19F 3CA1 A650  \.'éæì\ÈnØáŸ<¡¦P
00000030  B09C EB41 0D32 E987 D364 28C3 63A8 062E  °œëA.2é‡Ód(Ãc¨..
```

X26407-031522

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. Because 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.

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



```
Xilinx Zynq MP First Stage Boot Loader
Release 2021.2   Dec 14 2021  -  19:15:26
PMU Firmware 2021.2     Dec 14 2021   19:16:17
PMU_ROM Version: xpbr-v8.1.0-0
 KKDBG: Build Dec 15 2021, 19:51:42

XILINX PUF-BASED USER KEY ENCRYPTION DEMO
=========================================

(1) Encrypt key
(2) Decrypt key

Enter a file name up to 16 characters : Key1.key

Encrypted key successfully read from the SD card.

ENCRYPTED KEY BLOB: 012345678901234567890123E97E8AA7E85BB7FCE9BCC18AFDB8865D159045015C0127E9E6EC5CC86
ED8E19F3CA1A650B09CEB410D32E987D36428C363A8062E


Decrypted Key Blob
-------------------
IV: 012345678901234567890123
KEY ID: 42
KEY: 0123456789012345678901234567890123456789012345678901234567890123


GCM-AUTHENTICATION PASSED!
user eFUSE 2: 0x00000000 (0x00000000)
ID is valid! Key blob is safe for use.


(1) Encrypt key
(2) Decrypt key

Enter a file name up to 16 characters : Key2.key

Encrypted key successfully read from the SD card.

ENCRYPTED KEY BLOB: 8888888888888888888888887556E916A9319908540E79412A7979872662DECF365B4C6EF023B48D5
18B8E2EB5E30AFB1EBD72040B5DFD464A7B2520924FD3BC


Decrypted Key Blob
-------------------
IV: 888888888888888888888888
KEY ID: FF
KEY: 9999999999999999999999999999999999999999999999999999999999999999


GCM-AUTHENTICATION PASSED!
user eFUSE 7 simulation: 0x80000000
ID is INVALID! Key blob is NOT safe for use!


(1) Encrypt key
(2) Decrypt key
```

X26408-031522

Ordering

Because of the additional screening required to ensure entropy, AMD offers two versions of the PUF, a 128-bit one (SCD #4687) and a 256-bit one (SCD #4560). In both cases, the KEK length is 256 bits. These devices require special ordering codes (SCD). The PUF is fully supported with the standard ordering codes but entropy is not guaranteed, except for development and evaluation, because 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) (available in the Design Security Lounge) which is an AMD proprietary report.

Contact your local AMD field application engineer (FAE) or sales person to obtain a copy of the report. Use of the PUF does not require third party licensing fees. However, when ordering Zynq UltraScale+ MPSoCs with guaranteed entropy, keep in mind that SCD #4560 and SCD #4687 add additional costs. Reach out to your AMD FAE and/or sales team for additional details.

Conclusion

This application note guides you in how to use the PUF's device-unique encryption key with the AES-GCM hardware 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 with the PUF's device-unique key. In addition, this application note shows how to perform data validation of decrypted data packets by using values stored in the user programmable section of eFUSEs.

Reference Design

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

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Jim Wesselkamper, Nathan Menhorn, Krzysztof Kepa
Target devices	Zynq UltraScale+ devices
Source code provided?	Y
Source code format (if provided)	C
Design uses code or IP from existing reference design, application note, third party or Vivado software? If yes, list.	
Simulation	
Functional simulation performed	N
Timing simulation performed?	N
Test bench provided for functional and timing simulation?	N
Test bench format	N/A

Table 1: **Reference Design Matrix** *(cont'd)*

Parameter	Description
Simulator software and version	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/versions used	Vivado synthesis
Implementation software tool(s) and version	Vitis IDE 2021.2
Static timing analysis performed?	N
Hardware Verification	
Hardware verified?	Y
Platform used for verification	ZCU102 evaluation board

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Note: For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD 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 DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Appendix A

Creating the Zynq UltraScale+ ZCU102 Evaluation Board Hardware Design

1. Open the 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. Press `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 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 the Vitis IDE Launcher window opens, in the workspace, provide the appropriate working directory to create a Vitis IDE workspace and click **Launch**.
8. In the Welcome tab click **Create Platform Project** under the Project column. If no Welcome tab is present, in the main toolbar, click **File**→**New**→**Platform Project**.
9. Type in `ZCU102 XAPP1333` as the platform project name in the Create new platform window and click **Next**.
10. In Hardware Specification section of the Platform window, click **Browse** to search the XSA file location.
11. In the Create Platform from XSA window, navigate to the location that was provided in step 3, select the previously exported XSA file, and click **Open**. Back in the Hardware Specification section of the Platform window the file appears in the XSA File selection.

12. Leave the 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 the Vitis IDE.
13. To build the platform in the Vitis IDE, 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 Vitis IDE 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 the 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 the 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

These documents provide supplemental material useful with this application note:

1. Design Security Lounge
2. *Zynq UltraScale+ 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
06/20/2025 Version 1.3	
AES Key Generation	Restructured into multiple steps for clarity.
Introduction	Updated Figure 2 and Figure 3.
PUF Registration into eFUSEs	Updated figure in Step 12.
PUF Encryption Decryption Demo Application	Updated titles for Figure 13, Figure 14, Figure 15, and Figure 16.
Ordering	• Clarified information about third party licensing. • Added information about additional costs for SCD #4560 for 256-bit guaranteed entropy and SCD #4687 for 128-bit guaranteed entropy.
Finding Additional Documentation	Updated topic title.
04/12/2022 Version 1.2	
Throughout document	Updated SDK information to Vitis IDE across all content.
Introduction	Updated two block diagrams.
PUF eFUSE Configuration	Added one new figure, replaced two and removed one.
PUF Encryption and Decryption	Added two new figures, replaced six and removed two.
Ordering	Updated SDK steps to Vitis IDE 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 release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright