# Xilinx Standalone Library Documentation

## *XilSKey Library v6.8*

**UG1191 (2019.2) October 30, 2019**

XILINX

# Table of Contents


## Chapter 1: Overview

**Hardware Setup** . . . . . **9**

Hardware setup for Zynq PL . . . . . 9

Hardware setup for UltraScale or UltraScale+ . . . . . 10

**Source Files** . . . . . **11**

## Chapter 2: BBRAM PL API

**Overview** . . . . . **13**

**Example Usage** . . . . . **13**

**Function Documentation** . . . . . **14**

XilSKey_Bbram_Program . . . . . 14

## Chapter 3: Zynq UltraScale+ MPSoC BBRAM PS API

**Overview** . . . . . **15**

**Example Usage** . . . . . **15**

**Function Documentation** . . . . . **15**

XilSKey_ZynqMp_Bbram_Program . . . . . 15

XilSKey_ZynqMp_Bbram_Zeroise . . . . . 16

## Chapter 4: Zynq eFUSE PS API

**Overview** . . . . . **17**

**Example Usage** . . . . . **17**

**Function Documentation** . . . . . **17**

XilSKey_EfusePs_Write . . . . . 17

XilSKey_EfusePs_Read . . . . . 18

XilSKey_EfusePs_ReadStatus . . . . . 18

## Chapter 5: Zynq UltraScale+ MPSoC eFUSE PS API

**Overview** . . . . . **20**

**Example Usage** . . . . . **20**

**Function Documentation** . . . . . **21**

XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc . . . . . 21

XilSKey_ZynqMp_EfusePs_ReadUserFuse . . . . . 21

XilSKey_ZynqMp_EfusePs_ReadPpk0Hash . . . 22
XilSKey_ZynqMp_EfusePs_ReadPpk1Hash . . . 22
XilSKey_ZynqMp_EfusePs_ReadSpkId . . . 23
XilSKey_ZynqMp_EfusePs_ReadDna . . . 23
XilSKey_ZynqMp_EfusePs_ReadSecCtrlBits . . . 23
XilSKey_ZynqMp_EfusePs_Write . . . 24
XilSKey_ZynqMp_EfusePs_WritePufHelprData . . . 24
XilSKey_ZynqMp_EfusePs_ReadPufHelprData . . . 25
XilSKey_ZynqMp_EfusePs_WritePufChash . . . 25
XilSKey_ZynqMp_EfusePs_ReadPufChash . . . 26
XilSKey_ZynqMp_EfusePs_WritePufAux . . . 26
XilSKey_ZynqMp_EfusePs_ReadPufAux . . . 26
XilSKey_Write_Puf_EfusePs_SecureBits . . . 27
XilSKey_Read_Puf_EfusePs_SecureBits . . . 27
XilSKey_Puf_Debug2 . . . 28
XilSKey_Puf_Registration . . . 28
XilSKey_Puf_Regeneration . . . 29

## Chapter 6: eFUSE PL API

**Overview . . . 30**
**Example Usage . . . 30**
**Function Documentation . . . 30**
XilSKey_EfusePl_SystemInit . . . 30
XilSKey_EfusePl_Program . . . 31
XilSKey_EfusePl_ReadStatus . . . 31
XilSKey_EfusePl_ReadKey . . . 32

## Chapter 7: CRC Calculation API

**Overview . . . 33**
**Function Documentation . . . 33**
XilSKey_CrcCalculation . . . 33
XilSkey_CrcCalculation_AesKey . . . 34

## Chapter 8: User-Configurable Parameters

**Overview . . . 35**
**Zynq User-Configurable PS eFUSE Parameters . . . 35**
**Zynq User-Configurable PL eFUSE Parameters . . . 37**
Overview . . . 37
MIO Pins for Zynq PL eFUSE JTAG Operations . . . 38
MUX Selection Pin for Zynq PL eFUSE JTAG Operations . . . 40

MUX Parameter for Zynq PL eFUSE JTAG Operations . . . 40
AES and User Key Parameters . . . 41
**Zynq User-Configurable PL BBRAM Parameters . . . 42**
Overview . . . 42
MUX Parameter for Zynq BBRAM PL JTAG Operations . . . 43
AES and User Key Parameters . . . 43
**UltraScale or UltraScale+ User-Configurable BBRAM PL Parameters . . . 43**
Overview . . . 43
AES Keys and Related Parameters . . . 43
DPA Protection for BBRAM key . . . 48
GPIO Device Used for Connecting PL Master JTAG Signals . . . 49
GPIO Pins Used for PL Master JTAG Signals . . . 49
GPIO Channels . . . 50
**UltraScale or UltraScale+ User-Configurable PL eFUSE Parameters . . . 50**
Overview . . . 50
GPIO Device Used for Connecting PL Master JTAG Signals . . . 52
GPIO Pins Used for PL Master JTAG and HWM Signals . . . 53
GPIO Channels . . . 53
SLR Selection to Program eFUSE on MONO/SSIT Devices . . . 54
eFUSE PL Read Parameters . . . 54
AES Keys and Related Parameters . . . 55
USER Keys (32-bit) and Related Parameters . . . 57
RSA Hash and Related Parameters . . . 60
USER Keys (128-bit) and Related Parameters . . . 62
AES key CRC verification . . . 73
**Zynq UltraScale+ MPSoC User-Configurable PS eFUSE Parameters . . . 76**
Overview . . . 76
AES Keys and Related Parameters . . . 78
User Keys and Related Parameters . . . 79
PPK0 Keys and Related Parameters . . . 83
PPK1 Keys and Related Parameters . . . 84
SPK ID and Related Parameters . . . 85
**Zynq UltraScale+ MPSoC User-Configurable PS BBRAM Parameters . . . 87**
**Zynq UltraScale+ MPSoC User-Configurable PS PUF Parameters . . . 87**

## Chapter 9: Error Codes

**Overview . . . 90**
**PL eFUSE Error Codes . . . 90**
**PS eFUSE Error Codes . . . 93**

Zynq UltraScale+ MPSoC BBRAM PS Error Codes . . . . . 98

## Chapter 10: Status Codes

## Chapter 11: Procedures

Zynq eFUSE Writing Procedure Running from DDR as an Application . . . . . 100

Zynq eFUSE Driver Compilation Procedure for OCM . . . . . 100

UltraScale eFUSE Access Procedure . . . . . 101

UltraScale BBRAM Access Procedure . . . . . 101

## Appendix A: Additional Resources and Legal Notices

XILINX

Chapter 1

# Overview

The XilSKey library provides APIs for programming and reading eFUSE bits and for programming the battery-backed RAM (BBRAM) of Zynq®-7000 SoC, UltraScale™, UltraScale+™ and the Zynq UltraScale+ MPSoC devices.

- In Zynq-7000 devices:
  - PS eFUSE holds the RSA primary key hash bits and user feature bits, which can enable or disable some Zynq-7000 processor features.
  - PL eFUSE holds the AES key, the user key and some of the feature bits.
  - PL BBRAM holds the AES key.
- In Kintex/Virtex UltraScale or UltraScale+:
  - PL eFUSE holds the AES key, 32 bit and 128 bit user key, RSA hash and some of the feature bits.
  - PL BBRAM holds AES key with or without DPA protection enable or obfuscated key programming.
- In Zynq UltraScale+ MPSoC:
  - PUF registration and Regeneration.
  - PS eFUSE holds:
    - Programming AES key and can perform CRC verification of AES key
    - Programming/Reading User fuses
    - Programming/Reading PPK0/PPK1 sha3 hash
    - Programming/Reading SPKID
    - Programming/Reading secure control bits
  - PS BBRAM holds the AES key.
  - PL eFUSE holds the AES key, 32 bit and 128 bit user key, RSA hash and some of the feature bits.
  - PL BBRAM holds AES key with or without DPA protection enable or obfuscated key programming.

## BOARD Support Package Settings

There are few configurable parameters available under bsp settings, which can be configured during compilation of board support package.

# Configurations For Adding New device

The below configurations helps in adding new device information not supported by default. Currently, MicroBlaze™, Zynq UltraScale™ and Zynq UltraScale+™ MPSoC devices are supported.

| Parameter Name | Description |
|---|---|
| device_id | Mention the device ID |
| device_irlen | Mention IR length of the device. Default is 0 |
| device_numslr | Mention number of SLRs available. Range of values can be 1 to 4. Default is 1. If no slaves are present and only one master SLR is available then only 1 number of SLR is available. |
| device_series | Select the device series. Default is FPGA_SERIES_ZYNQ. The following device series are supported: XSK_FPGA_SERIES_ZYNQ - Select if the device belongs to the Zynq®-7000 family. XSK_FPGA_SERIES_ULTRA - Select if the device belongs to the Zynq UltraScale family. XSK_FPGA_SERIES_ULTRA_PLUS - Select if the device belongs to Zynq UltraScale MPSoC family. |
| device_masterslr | Mention the master SLR number. Default is 0. |

# Configurations For Zynq UltraScale+ MPSoC devices

| Parameter Name | Description |
|---|---|
| override_sysmon_cfg | Default = TRUE, library configures sysmon before accessing efuse memory. If you are using the Sysmon library and XilSkey library together, XilSkey overwrites the user defined sysmon configuration by default. When override_sysmon_cfg is set to false, XilSkey expects you to configure the sysmon to read the 3 ADC channels - Supply 1 (VPINT), Supply 3 (VPAUX) and LPD Temperature. XilSkey validates the user defined sysmon configuration is correct before performing the eFuse operations. |

**Note**

On Ultrascale and Ultrascale plus devices there can be multiple or single SLRs and among which one can be master and the others are slaves, where SLR 0 is not always the master SLR. Based on master and slave SLR order SLRs in this library are referred with config order index. Master SLR is mentioned with CONFIG ORDER 0, then follows the slaves config order, CONFIG ORDER 1,2 and 3 are for slaves in order. Due to the added support for the SSIT devices, it is recommended to use the updated library with updated examples only for the UltraScale and the UltraScale+ devices.

# Hardware Setup

This section describes the hardware setup required for programming PL BBRAM or PL eFUSE.

## Hardware setup for Zynq PL

This chapter describes the hardware setup required for programming BBRAM or eFUSE of Zynq PL devices. PL eFUSE or PL BBRAM is accessed through PS via MIO pins which are used for communication PL eFUSE or PL BBRAM through JTAG signals, these can be changed depending on the hardware setup.
A hardware setup which dedicates four MIO pins for JTAG signals should be used and the MIO pins should be mentioned in application header file (xilskey_input.h). There should be a method to download this example and have the MIO pins connected to JTAG before running this application. You can change the listed pins at your discretion.

### MUX Usage Requirements

To write the PL eFUSE or PL BBRAM using a driver you must:

- Use four MIO lines (TCK,TMS,TDO,TDI)
- Connect the MIO lines to a JTAG port

If you want to switch between the external JTAG and JTAG operation driven by the MIOs, you must:

- Include a MUX between the external JTAG and the JTAG operation driven by the MIOs
- Assign a MUX selection PIN

To rephrase, to select JTAG for PL EFUSE or PL BBRAM writing, you must define the following:

- The MIOs used for JTAG operations (TCK,TMS,TDI,TDO).
- The MIO used for the MUX Select Line.
- The Value on the MUX Select line, to select JTAG for PL eFUSE or PL BBRAM writing.

The following graphic illustrates the correct MUX usage.

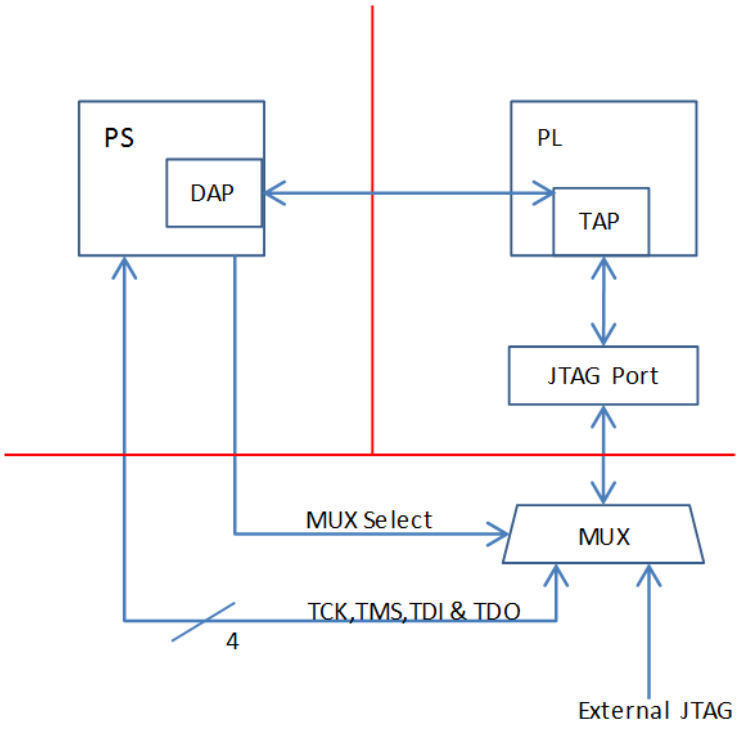


Figure 1.1: MUX Usage

**Note**

If you use the Vivado® Device Programmer tool to burn PL eFUSEs, there is no need for MUX circuitry or MIO pins.

# Hardware setup for UltraScale or UltraScale+

This chapter describes the hardware setup required for programming BBRAM or eFUSE of UltraScale devices. Accessing UltraScale MicroBlaze eFuse is done by using block RAM initialization. UltraScale eFUSE programming is done through MASTER JTAG. Crucial Programming sequence will be taken care by Hardware module. It is mandatory to add Hardware module in the design. Use hardware module's vhd code and instructions provided to add Hardware module in the design.

- You need to add the Master JTAG primitive to design, that is, the MASTER_JTAG_inst instantiation has to be performed and AXI GPIO pins have to be connected to TDO, TDI, TMS and TCK signals of the MASTER_JTAG primitive.
- For programming eFUSE, along with master JTAG, hardware module(HWM) has to be added in design and it's signals XSK_EFUSEPL_AXI_GPIO_HWM_READY , XSK_EFUSEPL_AXI_GPIO_HWM_END and XSK_EFUSEPL_AXI_GPIO_HWM_START, needs to be connected to AXI GPIO pins to communicate with HWM. Hardware module is not mandatory for programming BBRAM. If your design has a HWM, it is not harmful for accessing BBRAM.

- All inputs (Master JTAG's TDO and HWM's HWM_READY, HWM_END) and all outputs (Master JTAG TDI, TMS, TCK and HWM's HWM_START) can be connected in one channel (or) inputs in one channel and outputs in other channel.
- Some of the outputs of GPIO in one channel and some others in different channels are not supported.
- The design should contain AXI BRAM control memory mapped (1MB).

**Note**

MASTER_JTAG will disable all other JTAGs.

For providing inputs of MASTER JTAG signals and HWM signals connected to the GPIO pins and GPIO channels, refer GPIO Pins Used for PL Master JTAG Signal and GPIO Channels sections of the UltraScale User-Configurable PL eFUSE Parameters and UltraScale User-Configurable PL BBRAM Parameters.
The procedure for programming BBRAM of eFUSE of UltraScale or UltraScale+ can be referred at UltraScale BBRAM Access Procedure and UltraScale eFUSE Access Procedure.

# Source Files

The following is a list of eFUSE and BBRAM application project files, folders and macros.

- `xilskey_efuse_example.c`: This file contains the main application code. The file helps in the PS/PL structure initialization and writes/reads the PS/PL eFUSE based on the user settings provided in the xilskey_input.h file.
- `xilskey_input.h`: This file ontains all the actions that are supported by the eFUSE library. Using the preprocessor directives given in the file, you can read/write the bits in the PS/PL eFUSE. More explanation of each directive is provided in the following sections. Burning or reading the PS/PL eFUSE bits is based on the values set in the xilskey_input.h file. Also contains GPIO pins and channels connected to MASTER JTAG primitive and hardware module to access Ultrascale eFUSE.
  In this file:
  - specify the 256 bit key to be programmed into BBRAM.
  - specify the AES(256 bit) key, User (32 bit and 128 bit) keys and RSA key hash(384 bit) key to be programmed into UltraScale eFUSE.
  - `XSK_EFUSEPS_DRIVER`: Define to enable the writing and reading of PS eFUSE.
  - `XSK_EFUSEPL_DRIVER`: Define to enable the writing of PL eFUSE.
- `xilskey_bbram_example.c`: This file contains the example to program a key into BBRAM and verify the key.

  **Note**

  This algorithm only works when programming and verifying key are both executed in the recommended order.

- `xilskey_efuseps_zynqmp_example.c`: This file contains the example code to program the PS eFUSE and read back of eFUSE bits from the cache.

- `xilskey_efuseps_zynqmp_input.h`: This file contains all the inputs supported for eFUSE PS of Zynq UltraScale+ MPSoC. eFUSE bits are programmed based on the inputs from the xilskey_efuseps_zynqmp_input.h file.
- `xilskey_bbramps_zynqmp_example.c`: This file contains the example code to program and verify BBRAM key of Zynq UltraScale+ MPSoC. Default is zero. You can modify this key on top of the file.
- `xilskey_bbram_ultrascale_example.c`: This file contains example code to program and verify BBRAM key of UltraScale.

  **Note**

  Programming and verification of BBRAM key cannot be done separately.

- `xilskey_bbram_ultrascale_input.h`: This file contains all the preprocessor directives you need to provide. In this file, specify BBRAM AES key or Obfuscated AES key to be programmed, DPA protection enable and, GPIO pins and channels connected to MASTER JTAG primitive.
- `xilskey_puf_registration.c`: This file contains all the PUF related code. This example illustrates PUF registration and generating black key and programming eFUSE with PUF helper data, CHash and Auxilary data along with the Black key.
- `xilskey_puf_registration.h`: This file contains all the preprocessor directives based on which read/write the eFUSE bits and Syndrome data generation. More explanation of each directive is provided in the following sections.

**WARNING:** *Ensure that you enter the correct information before writing or 'burning' eFUSE bits. Once burned, they cannot be changed. The BBRAM key can be programmed any number of times.*

**Note**

POR reset is required for the eFUSE values to be recognized.

XILINX

Chapter 2

# BBRAM PL API

## Overview

This chapter provides a linked summary and detailed descriptions of the battery-backed RAM (BBRAM) APIs of Zynq® PL and UltraScale™ devices.

## Example Usage

- Zynq BBRAM PL example usage:
  - The Zynq BBRAM PL example application should contain the `xilskey_bbram_example.c` and `xilskey_input.h` files.
  - You should provide user configurable parameters in the `xilskey_input.h` file. For more information, refer Zynq User-Configurable PL BBRAM Parameters.
- UltraScale BBRAM example usage:
  - The UltraScale BBRAM example application should contain the `xilskey_bbram_ultrascale_input.h` and `xilskey_bbram_ultrascale_example.c` files.
  - You should provide user configurable parameters in the `xilskey_bbram_ultrascale_input.h` file. For more information, refer UltraScale or UltraScale+ User-Configurable BBRAM PL Parameters.

**Note**

It is assumed that you have set up your hardware prior to working on the example application. For more information, refer Hardware Setup.

# Functions

- int XilSKey_Bbram_Program (XilSKey_Bbram *InstancePtr)

# Function Documentation

## int XilSKey_Bbram_Program ( XilSKey_Bbram * *InstancePtr* )

This function implements the BBRAM algorithm for programming and verifying key.
The program and verify will only work together in and in that order.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to XilSKey_Bbram |

**Returns**

- XST_FAILURE - In case of failure
- XST_SUCCESS - In case of Success

**Note**

This function will program BBRAM of Ultrascale and Zynq as well.

XILINX

Chapter 3

# Zynq UltraScale+ MPSoC BBRAM PS API

## Overview

This chapter provides a linked summary and detailed descriptions of the battery-backed RAM (BBRAM) APIs for Zynq® UltraScale+™ MPSoC devices.

## Example Usage

- The Zynq UltraScale+ MPSoc example application should contain the `xilskey_bbramps_zynqmp_example.c` file.
- User configurable key can be modified in the same file (`xilskey_bbramps_zynqmp_example.c`), at the `XSK_ZYNQMP_BBRAMPS_AES_KEY` macro.

## Functions

- u32 XilSKey_ZynqMp_Bbram_Program (u32 *AesKey)
- u32 XilSKey_ZynqMp_Bbram_Zeroise (void)

## Function Documentation

### u32 XilSKey_ZynqMp_Bbram_Program ( u32 * *AesKey* )

This function implements the BBRAM programming and verifying the key written.
Program and verification of AES will work only together. CRC of the provided key will be calculated internally and verified after programming.

**Parameters**

| | |
|---|---|
| *AesKey* | Pointer to the key which has to be programmed. |

**Returns**

- Error code from XskZynqMp_Ps_Bbram_ErrorCodes enum if it fails
- XST_SUCCESS if programming is done.

# u32 XilSKey_ZynqMp_Bbram_Zeroise ( void )

This function zeroize's Bbram Key.

### Parameters

| | |
|---|---|
| *None.* | |

### Returns

None.

### Note

BBRAM key will be zeroized.

XILINX

Chapter 4

# Zynq eFUSE PS API

## Overview

This chapter provides a linked summary and detailed descriptions of the Zynq eFUSE PS APIs.

## Example Usage

- The Zynq eFUSE PS example application should contain the `xilskey_efuse_example.c` and the `xilskey_input.h` files.
- There is no need of any hardware setup. By default, both the eFUSE PS and PL are enabled in the application. You can comment 'XSK_EFUSEPL_DRIVER' to execute only the PS. For more details, refer Zynq User-Configurable PS eFUSE Parameters.

## Functions

- u32 XilSKey_EfusePs_Write (XilSKey_EPs *PsInstancePtr)
- u32 XilSKey_EfusePs_Read (XilSKey_EPs *PsInstancePtr)
- u32 XilSKey_EfusePs_ReadStatus (XilSKey_EPs *InstancePtr, u32 *StatusBits)

## Function Documentation

### u32 XilSKey_EfusePs_Write ( XilSKey_EPs * *InstancePtr* )

PS eFUSE interface functions.
PS eFUSE interface functions.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the PsEfuseHandle which describes which PS eFUSE bit should be burned. |

**Returns**

- XST_SUCCESS.
- In case of error, value is as defined in xilskey_utils.h Error value is a combination of Upper 8 bit value and Lower 8 bit value. For example, 0x8A03 should be checked in error.h as 0x8A00 and 0x03. Upper 8 bit value signifies the major error and lower 8 bit values tells more precisely.

**Note**

When called, this Initializes the timer, XADC subsystems. Unlocks the PS eFUSE controller.Configures the PS eFUSE controller. Writes the hash and control bits if requested. Programs the PS eFUSE to enable the RSA authentication if requested. Locks the PS eFUSE controller. Returns an error, if the reference clock frequency is not in between 20 and 60 MHz or if the system not in a position to write the requested PS eFUSE bits (because the bits are already written or not allowed to write) or if the temperature and voltage are not within range

## u32 XilSKey_EfusePs_Read ( XilSKey_EPs * *InstancePtr* )

This function is used to read the PS eFUSE.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the PsEfuseHandle which describes which PS eFUSE should be burned. |

**Returns**

- XST_SUCCESS no errors occurred.
- In case of error, value is as defined in xilskey_utils.h. Error value is a combination of Upper 8 bit value and Lower 8 bit value. For example, 0x8A03 should be checked in error.h as 0x8A00 and 0x03. Upper 8 bit value signifies the major error and lower 8 bit values tells more precisely.

**Note**

When called: This API initializes the timer, XADC subsystems. Unlocks the PS eFUSE Controller. Configures the PS eFUSE Controller and enables read-only mode. Reads the PS eFUSE (Hash Value), and enables read-only mode. Locks the PS eFUSE Controller. Returns an error, if the reference clock frequency is not in between 20 and 60MHz. or if unable to unlock PS eFUSE controller or requested address corresponds to restricted bits. or if the temperature and voltage are not within range

## u32 XilSKey_EfusePs_ReadStatus ( XilSKey_EPs * *InstancePtr,* u32 * *StatusBits* )

This function is used to read the PS efuse status register.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the PS eFUSE instance. |
| *StatusBits* | Buffer to store the status register read. |

**Returns**

- XST_SUCCESS.
- XST_FAILURE

**Note**

This API unlocks the controller and reads the Zynq PS eFUSE status register.

XILINX

Chapter 5

# Zynq UltraScale+ MPSoC eFUSE PS API

## Overview

This chapter provides a linked summary and detailed descriptions of the Zynq MPSoC UltraScale+ eFUSE PS APIs.

## Example Usage

- For programming eFUSEs other than the PUF, the Zynq UltraScale+ MPSoC example application should contain the `xilskey_efuseps_zynqmp_example.c` and the `xilskey_efuseps_zynqmp_input.h` files.
- For PUF registration, programming PUF helper data, AUX, chash, and black key, the Zynq UltraScale+ MPSoC example application should contain the `xilskey_puf_registration.c` and the `xilskey_puf_registration.h` files.
- For more details on the user configurable parameters, refer Zynq UltraScale+ MPSoC User-Configurable PS eFUSE Parameters and Zynq UltraScale+ MPSoC User-Configurable PS PUF Parameters.

## Functions

- u32 XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc (u32 CrcValue)
- u32 XilSKey_ZynqMp_EfusePs_ReadUserFuse (u32 *UseFusePtr, u8 UserFuse_Num, u8 ReadOption)
- u32 XilSKey_ZynqMp_EfusePs_ReadPpk0Hash (u32 *Ppk0Hash, u8 ReadOption)
- u32 XilSKey_ZynqMp_EfusePs_ReadPpk1Hash (u32 *Ppk1Hash, u8 ReadOption)
- u32 XilSKey_ZynqMp_EfusePs_ReadSpkId (u32 *SpkId, u8 ReadOption)
- void XilSKey_ZynqMp_EfusePs_ReadDna (u32 *DnaRead)
- u32 XilSKey_ZynqMp_EfusePs_ReadSecCtrlBits (XilSKey_SecCtrlBits *ReadBackSecCtrlBits, u8 ReadOption)
- u32 XilSKey_ZynqMp_EfusePs_Write (XilSKey_ZynqMpEPs *InstancePtr)
- u32 XilSKey_ZynqMp_EfusePs_WritePufHelprData (XilSKey_Puf *InstancePtr)
- u32 XilSKey_ZynqMp_EfusePs_ReadPufHelprData (u32 *Address)
- u32 XilSKey_ZynqMp_EfusePs_WritePufChash (XilSKey_Puf *InstancePtr)
- u32 XilSKey_ZynqMp_EfusePs_ReadPufChash (u32 *Address, u8 ReadOption)
- u32 XilSKey_ZynqMp_EfusePs_WritePufAux (XilSKey_Puf *InstancePtr)
- u32 XilSKey_ZynqMp_EfusePs_ReadPufAux (u32 *Address, u8 ReadOption)

- u32 XilSKey_Write_Puf_EfusePs_SecureBits (XilSKey_Puf_Secure *WriteSecureBits)
- u32 XilSKey_Read_Puf_EfusePs_SecureBits (XilSKey_Puf_Secure *SecureBitsRead, u8 ReadOption)
- u32 XilSKey_Puf_Debug2 (XilSKey_Puf *InstancePtr)
- u32 XilSKey_Puf_Registration (XilSKey_Puf *InstancePtr)
- u32 XilSKey_Puf_Regeneration (XilSKey_Puf *InstancePtr)

# Function Documentation

## u32 XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc ( u32 *CrcValue* )

This function performs the CRC check of AES key.

**Parameters**

| | |
|---|---|
| *CrcValue* | A 32 bit CRC value of an expected AES key. |

**Returns**

- XST_SUCCESS on successful CRC check.
- ErrorCode on failure

**Note**

For Calculating the CRC of the AES key use the XilSKey_CrcCalculation() function or XilSkey_CrcCalculation_AesKey() function

## u32 XilSKey_ZynqMp_EfusePs_ReadUserFuse ( u32 * *UseFusePtr*, u8 *UserFuse_Num*, u8 *ReadOption* )

This function is used to read a user fuse from the eFUSE or cache.

**Parameters**

| | |
|---|---|
| *UseFusePtr* | Pointer to an array which holds the readback user fuse. |
| *UserFuse_Num* | A variable which holds the user fuse number. Range is (User fuses: 0 to 7) |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS on successful read
- ErrorCode on failure

## u32 XilSKey_ZynqMp_EfusePs_ReadPpk0Hash ( u32 * *Ppk0Hash*, u8 *ReadOption* )

This function is used to read the PPK0 hash from an eFUSE or eFUSE cache.

**Parameters**

| | |
|---|---|
| *Ppk0Hash* | A pointer to an array which holds the readback PPK0 hash. |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS on successful read
- ErrorCode on failure

## u32 XilSKey_ZynqMp_EfusePs_ReadPpk1Hash ( u32 * *Ppk1Hash*, u8 *ReadOption* )

This function is used to read the PPK1 hash from eFUSE or cache.

**Parameters**

| | |
|---|---|
| *Ppk1Hash* | Pointer to an array which holds the readback PPK1 hash. |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS on successful read
- ErrorCode on failure

## u32 XilSKey_ZynqMp_EfusePs_ReadSpkId ( u32 * *SpkId*, u8 *ReadOption* )

This function is used to read SPKID from eFUSE or cache based on user's read option.

**Parameters**

| | |
|---|---|
| *SpkId* | Pointer to a 32 bit variable which holds SPK ID. |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS on successful read
- ErrorCode on failure

## void XilSKey_ZynqMp_EfusePs_ReadDna ( u32 * *DnaRead* )

This function is used to read DNA from eFUSE.

**Parameters**

| | |
|---|---|
| *DnaRead* | Pointer to an array of 3 x u32 words which holds the readback DNA. |

**Returns**

None.

## u32 XilSKey_ZynqMp_EfusePs_ReadSecCtrlBits ( XilSKey_SecCtrlBits * *ReadBackSecCtrlBits*, u8 *ReadOption* )

This function is used to read the PS eFUSE secure control bits from cache or eFUSE based on user input provided.

**Parameters**

| | |
|---|---|
| *ReadBackSecCtrlBits* | Pointer to the XilSKey_SecCtrlBits which holds the read secure control bits. |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS if reads successfully
- XST_FAILURE if reading is failed

**Note**

Cache reload is required for obtaining updated values for ReadOption 0.

## u32 XilSKey_ZynqMp_EfusePs_Write ( XilSKey_ZynqMpEPs * *InstancePtr* )

This function is used to program the PS eFUSE of ZynqMP, based on user inputs.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the XilSKey_ZynqMpEPs. |

**Returns**

- XST_SUCCESS if programs successfully.
- Errorcode on failure

**Note**

After eFUSE programming is complete, the cache is automatically reloaded so all programmed eFUSE bits can be directly read from cache.

## u32 XilSKey_ZynqMp_EfusePs_WritePufHelprData ( XilSKey_Puf * *InstancePtr* )

This function programs the PS eFUSEs with the PUF helper data.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the XilSKey_Puf instance. |

**Returns**

- XST_SUCCESS if programs successfully.
- Errorcode on failure

**Note**

To generate PufSyndromeData please use XilSKey_Puf_Registration API

## u32 XilSKey_ZynqMp_EfusePs_ReadPufHelprData ( u32 * *Address* )

This function reads the PUF helper data from eFUSE.

**Parameters**

| | |
|---|---|
| *Address* | Pointer to data array which holds the PUF helper data read from eFUSEs. |

**Returns**

- XST_SUCCESS if reads successfully.
- Errorcode on failure.

**Note**

This function only reads from eFUSE non-volatile memory. There is no option to read from Cache.

## u32 XilSKey_ZynqMp_EfusePs_WritePufChash ( XilSKey_Puf * *InstancePtr* )

This function programs eFUSE with CHash value.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the XilSKey_Puf instance. |

**Returns**

- XST_SUCCESS if chash is programmed successfully.
- An Error code on failure

**Note**

To generate the CHash value, please use XilSKey_Puf_Registration function.

## u32 XilSKey_ZynqMp_EfusePs_ReadPufChash ( u32 * *Address*, u8 *ReadOption* )

This function reads eFUSE PUF CHash data from the eFUSE array or cache based on the user read option.

**Parameters**

| | |
|---|---|
| *Address* | Pointer which holds the read back value of the chash. |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS if programs successfully.
- Errorcode on failure

**Note**

Cache reload is required for obtaining updated values for reading from cache..

## u32 XilSKey_ZynqMp_EfusePs_WritePufAux ( XilSKey_Puf * *InstancePtr* )

This function programs eFUSE PUF auxiliary data.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the XilSKey_Puf instance. |

**Returns**

- XST_SUCCESS if the eFUSE is programmed successfully.
- Errorcode on failure

**Note**

To generate auxiliary data, please use XilSKey_Puf_Registration function.

## u32 XilSKey_ZynqMp_EfusePs_ReadPufAux ( u32 * *Address*, u8 *ReadOption* )

This function reads eFUSE PUF auxiliary data from eFUSE array or cache based on user read option.

**Parameters**

| | |
|---|---|
| *Address* | Pointer which holds the read back value of PUF's auxiliary data. |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS if PUF auxiliary data is read successfully.
- Errorcode on failure

**Note**

Cache reload is required for obtaining updated values for reading from cache.

## u32 XilSKey_Write_Puf_EfusePs_SecureBits ( XilSKey_Puf_Secure * *WriteSecureBits* )

This function programs the eFUSE PUF secure bits.

**Parameters**

| | |
|---|---|
| *WriteSecureBits* | Pointer to the XilSKey_Puf_Secure structure |

**Returns**

- XST_SUCCESS if eFUSE PUF secure bits are programmed successfully.
- Errorcode on failure.

## u32 XilSKey_Read_Puf_EfusePs_SecureBits ( XilSKey_Puf_Secure * *SecureBitsRead,* u8 *ReadOption* )

This function is used to read the PS eFUSE PUF secure bits from cache or from eFUSE array.

**Parameters**

| | |
|---|---|
| *SecureBits* | Pointer to the XilSKey_Puf_Secure structure which holds the read eFUSE secure bits from the PUF. |
| *ReadOption* | Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache.<br>• 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from cache<br>• 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array |

**Returns**

- XST_SUCCESS if reads successfully.
- Errorcode on failure.

## u32 XilSKey_Puf_Debug2 ( XilSKey_Puf * *InstancePtr* )

This function Outputs distance metric that may be useful for software to determine impending key generation failures.
Distance metric also is useful to obtain a more stable provisioning syndrome value.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the XilSKey_Puf instance. |

**Returns**

- XST_SUCCESS if debug 2 mode was successful.
- ERROR if registration was unsuccessful.

## u32 XilSKey_Puf_Registration ( XilSKey_Puf * *InstancePtr* )

This function performs registration of PUF which generates a new KEK and associated CHash, Auxiliary and PUF-syndrome data which are unique for each silicon.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to the XilSKey_Puf instance. |

**Returns**

- XST_SUCCESS if registration/re-registration was successful.
- ERROR if registration was unsuccessful

**Note**

With the help of generated PUF syndrome data, it will be possible to re-generate same PUF KEK.

# u32 XilSKey_Puf_Regeneration ( XilSKey_Puf * *InstancePtr* )

This function regenerates the PUF data so that the PUF's output can be used as the key source to the AES-GCM hardware cryptographic engine.

**Parameters**

| | |
|---|---|
| *InstancePtr* | is a pointer to the XilSKey_Puf instance. |

**Returns**

- XST_SUCCESS if regeneration was successful.
- ERROR if regeneration was unsuccessful

XILINX

Chapter 6

# eFUSE PL API

## Overview

This chapter provides a linked summary and detailed descriptions of the eFUSE APIs of Zynq eFUSE PL and UltraScale eFUSE.

## Example Usage

- The Zynq eFUSE PL and UltraScale example application should contain the `xilskey_efuse_example.c` and the `xilskey_input.h` files.
- By default, both the eFUSE PS and PL are enabled in the application. You can comment 'XSK_EFUSEPL_DRIVER' to execute only the PS.
- For UltraScale, it is mandatory to comment 'XSK_EFUSEPS_DRIVER else the example will generate an error.
- For more details on the user configurable parameters, refer Zynq User-Configurable PL eFUSE Parameters and UltraScale or UltraScale+ User-Configurable PL eFUSE Parameters.
- Requires hardware setup to program PL eFUSE of Zynq or UltraScale.

## Functions

- u32 XilSKey_EfusePl_SystemInit (XilSKey_EPl *InstancePtr)
- u32 XilSKey_EfusePl_Program (XilSKey_EPl *PlInstancePtr)
- u32 XilSKey_EfusePl_ReadStatus (XilSKey_EPl *InstancePtr, u32 *StatusBits)
- u32 XilSKey_EfusePl_ReadKey (XilSKey_EPl *InstancePtr)

## Function Documentation

### u32 XilSKey_EfusePl_SystemInit ( XilSKey_EPl * *InstancePtr* )

Initializes PL eFUSE with input data given.

**Parameters**

| | |
|---|---|
| *InstancePtr* | - Input data to be written to PL eFUSE |

**Returns**

- XST_FAILURE - In case of failure
- XST_SUCCESS - In case of Success

**Note**

Updates the global variable ErrorCode with error code(if any).

## u32 XilSKey_EfusePl_Program ( XilSKey_EPl * *InstancePtr* )

Programs PL eFUSE with input data given through InstancePtr.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to PL eFUSE instance which holds the input data to be written to PL eFUSE. |

**Returns**

- XST_FAILURE - In case of failure
- XST_SUCCESS - In case of Success

**Note**

When this API is called: Initializes the timer, XADC/xsysmon and JTAG server subsystems. Returns an error in the following cases, if the reference clock frequency is not in the range or if the PL DAP ID is not identified, if the system is not in a position to write the requested PL eFUSE bits (because the bits are already written or not allowed to write) if the temperature and voltage are not within range.

## u32 XilSKey_EfusePl_ReadStatus ( XilSKey_EPl * *InstancePtr*, u32 * *StatusBits* )

Reads the PL efuse status bits and gets all secure and control bits.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to PL eFUSE instance. |
| *StatusBits* | Buffer to store the status bits read. |

**Returns**

- XST_FAILURE - In case of failure
- XST_SUCCESS - In case of Success

## u32 XilSKey_EfusePl_ReadKey ( XilSKey_EPl * *InstancePtr* )

Reads the PL efuse keys and stores them in the corresponding arrays in instance structure.

**Parameters**

| | |
|---|---|
| *InstancePtr* | Pointer to PL eFUSE instance. |

**Returns**

- XST_FAILURE - In case of failure
- XST_SUCCESS - In case of Success

**Note**

This function initializes the timer, XADC and JTAG server subsystems, if not already done so. In Zynq - Reads AES key and User keys. In Ultrascale - Reads 32 bit and 128 bit User keys and RSA hash But AES key cannot be read directly it can be verified with CRC check (for that we need to update the instance with 32 bit CRC value, API updates whether provided CRC value is matched with actuals or not). To calculate the CRC of expected AES key one can use any of the following APIs XilSKey_CrcCalculation() or XilSkey_CrcCalculation_AesKey()

XILINX

Chapter 7

# CRC Calculation API

## Overview

This chapter provides a linked summary and detailed descriptions of the CRC calculation APIs.
For UltraScale and Zynq UltraScale+ MPSoC devices, the programmed AES cannot be read back. The programmed AES key can only be verified by reading the CRC value of AES key.

## Functions

- u32 XilSKey_CrcCalculation (u8 *Key)
- u32 XilSkey_CrcCalculation_AesKey (u8 *Key)

## Function Documentation

### u32 XilSKey_CrcCalculation ( u8 * *Key* )

This function Calculates CRC value based on hexadecimal string passed.

**Parameters**

| | |
|---|---|
| *Key* | Pointer to the string contains AES key in hexadecimal of length less than or equal to 64. |

**Returns**

- On Success returns the Crc of AES key value.
- On failure returns the error code when string length is greater than 64

**Note**

If the length of the string provided is less than 64, this function appends the string with zeros. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8 *Key) API or reverse polynomial 0x82F63B78.

# u32 XilSkey_CrcCalculation_AesKey ( u8 * *Key* )

Calculates CRC value of the provided key.
Key should be provided in hexa buffer.

### Parameters

| | |
|---|---|
| *Key* | Pointer to an array of 32 bytes, which holds an AES key. |

### Returns

Crc of provided AES key value. To calculate CRC on the AES key in string format please use XilSKey_CrcCalculation.

XILINX

Chapter 8

# User-Configurable Parameters

## Overview

This chapter provides detailed descriptions of the various user configurable parameters.

## Modules

- Zynq User-Configurable PS eFUSE Parameters
- Zynq User-Configurable PL eFUSE Parameters
- Zynq User-Configurable PL BBRAM Parameters
- UltraScale or UltraScale+ User-Configurable BBRAM PL Parameters
- UltraScale or UltraScale+ User-Configurable PL eFUSE Parameters
- Zynq UltraScale+ MPSoC User-Configurable PS eFUSE Parameters
- Zynq UltraScale+ MPSoC User-Configurable PS BBRAM Parameters
- Zynq UltraScale+ MPSoC User-Configurable PS PUF Parameters

## Zynq User-Configurable PS eFUSE Parameters

Define the XSK_EFUSEPS_DRIVER macro to use the PS eFUSE.
After defining the macro, provide the inputs defined with XSK_EFUSEPS_DRIVER to burn the bits in PS eFUSE. If the bit is to be burned, define the macro as TRUE; otherwise define the macro as FALSE. For details, refer the following table.

| Macro Name | Description |
|---|---|
| XSK_EFUSEPS_ENABLE_WRITE_PROTECT | Default = FALSE.<br>TRUE to burn the write-protect bits in eFUSE array. Write protect has two bits. When either of the bits is burned, it is considered write-protected. So, while burning the write-protected bits, even if one bit is blown, write API returns success. As previously mentioned, POR reset is required after burning for write protection of the eFUSE bits to go into effect. It is recommended to do the POR reset after write protection. Also note that, after write-protect bits are burned, no more eFUSE writes are possible.<br>If the write-protect macro is TRUE with other macros, write protect is burned in the last iteration, after burning all the defined values, so that for any error while burning other macros will not effect the total eFUSE array.<br>FALSE does not modify the write-protect bits. |
| XSK_EFUSEPS_ENABLE_RSA_AUTH | Default = FALSE.<br>Use TRUE to burn the RSA enable bit in the PS eFUSE array. After enabling the bit, every successive boot must be RSA-enabled apart from JTAG. Before burning (blowing) this bit, make sure that eFUSE array has the valid PPK hash. If the PPK hash burning is enabled, only after writing the hash successfully, RSA enable bit will be blown. For the RSA enable bit to take effect, POR reset is required.<br>FALSE does not modify the RSA enable bit. |
| XSK_EFUSEPS_ENABLE_ROM_128K_CRC | Default = FALSE.<br>TRUE burns the ROM 128K CRC bit. In every successive boot, BootROM calculates 128k CRC.<br>FALSE does not modify the ROM CRC 128K bit. |
| XSK_EFUSEPS_ENABLE_RSA_KEY_HASH | Default = FALSE.<br>TRUE burns (blows) the eFUSE hash, that is given in XSK_EFUSEPS_RSA_KEY_HASH_VALUE when write API is used. TRUE reads the eFUSE hash when the read API is used and is read into structure.<br>FALSE ignores the provided value. |

| Macro Name | Description |
|---|---|
| XSK_EFUSEPS_RSA_KEY_HASH_VALUE | Default = 00000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The specified value is converted to a hexadecimal buffer and written into the PS eFUSE array when the write API is used. This value should be the Primary Public Key (PPK) hash provided in string format. The buffer must be 64 characters long: valid characters are 0-9, a-f, and A-F. Any other character is considered an invalid string and will not burn RSA hash. When the `Xilskey_EfusePs_Write()` API is used, the RSA hash is written, and the XSK_EFUSEPS_ENABLE_RSA_KEY_HASH must have a value of TRUE. |
| XSK_EFUSEPS_DISABLE_DFT_JTAG | Default = FALSE<br>TRUE disables DFT JTAG permanently. FALSE will not modify the eFuse PS DFT JTAG disable bit. |
| XSK_EFUSEPS_DISABLE_DFT_MODE | Default = FALSE<br>TRUE disables DFT mode permanently. FALSE will not modify the eFuse PS DFT mode disable bit. |

# Zynq User-Configurable PL eFUSE Parameters

## Overview

Define the `XSK_EFUSEPL_DRIVER` macro to use the PL eFUSE.
After defining the macro, provide the inputs defined with `XSK_EFUSEPL_DRIVER` to burn the bits in PL eFUSE bits. If the bit is to be burned, define the macro as TRUE; otherwise define the macro as FALSE. The table below lists the user-configurable PL eFUSE parameters for Zynq® devices.

| Macro Name | Description |
|---|---|
| XSK_EFUSEPL_FORCE_PCYCLE_RECONFIG | Default = FALSE<br>If the value is set to TRUE, then the part has to be power-cycled to be reconfigured.<br>FALSE does not set the eFUSE control bit. |
| XSK_EFUSEPL_DISABLE_KEY_WRITE | Default = FALSE<br>TRUE disables the eFUSE write to FUSE_AES and FUSE_USER blocks.<br>FALSE does not affect the EFUSE bit. |

| Macro Name | Description |
|---|---|
| XSK_EFUSEPL_DISABLE_AES_KEY_READ | Default = FALSE<br>TRUE disables the write to FUSE_AES and FUSE_USER key and disables the read of FUSE_AES.<br>FALSE does not affect the eFUSE bit. |
| XSK_EFUSEPL_DISABLE_USER_KEY_READ | Default = FALSE.<br>TRUE disables the write to FUSE_AES and FUSE_USER key and disables the read of FUSE_USER.<br>FALSE does not affect the eFUSE bit. |
| XSK_EFUSEPL_DISABLE_FUSE_CNTRL_WRITE | Default = FALSE.<br>TRUE disables the eFUSE write to FUSE_CTRL block.<br>FALSE does not affect the eFUSE bit. |
| XSK_EFUSEPL_FORCE_USE_AES_ONLY | Default = FALSE.<br>TRUE forces the use of secure boot with eFUSE AES key only.<br>FALSE does not affect the eFUSE bit. |
| XSK_EFUSEPL_DISABLE_JTAG_CHAIN | Default = FALSE.<br>TRUE permanently disables the Zynq ARM DAP and PL TAP.<br>FALSE does not affect the eFUSE bit. |
| XSK_EFUSEPL_BBRAM_KEY_DISABLE | Default = FALSE.<br>TRUE forces the eFUSE key to be used if booting Secure Image.<br>FALSE does not affect the eFUSE bit. |

# Modules

- MIO Pins for Zynq PL eFUSE JTAG Operations
- MUX Selection Pin for Zynq PL eFUSE JTAG Operations
- MUX Parameter for Zynq PL eFUSE JTAG Operations
- AES and User Key Parameters

# MIO Pins for Zynq PL eFUSE JTAG Operations

The table below lists the MIO pins for Zynq PL eFUSE JTAG operations.
You can change the listed pins at your discretion.

**Note**

The pin numbers listed in the table below are examples. You must assign appropriate pin numbers as per your hardware design.

| Pin Name | Pin Number |
|---|---|
| XSK_EFUSEPL_MIO_JTAG_TDI | (17) |
| XSK_EFUSEPL_MIO_JTAG_TDO | (21) |
| XSK_EFUSEPL_MIO_JTAG_TCK | (19) |
| XSK_EFUSEPL_MIO_JTAG_TMS | (20) |

## MUX Selection Pin for Zynq PL eFUSE JTAG Operations

The table below lists the MUX selection pin.

| Pin Name | Pin Number | Description |
|---|---|---|
| XSK_EFUSEPL_MIO_JTAG_ MUX_SELECT | (11) | This pin toggles between the external JTAG or MIO driving JTAG operations. |

## MUX Parameter for Zynq PL eFUSE JTAG Operations

The table below lists the MUX parameter.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_MIO_MUX_SEL_DEFAULT_VAL | Default = LOW.<br>LOW writes zero on the MUX select line before PL_eFUSE writing.<br>HIGH writes one on the MUX select line before PL_eFUSE writing. |

# AES and User Key Parameters

The table below lists the AES and user key parameters.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_PROGRAM_AES_AND_USER_LOW_KEY | Default = FALSE.<br>TRUE burns the AES and User Low hash key, which are given in the XSK_EFUSEPL_AES_KEY and the XSK_EFUSEPL_USER_LOW_KEY respectively.<br>FALSE ignores the provided values.<br>You cannot write the AES Key and the User Low Key separately. |
| XSK_EFUSEPL_PROGRAM_USER_HIGH_KEY | Default =FALSE.<br>TRUE burns the User High hash key, given in XSK_EFUSEPL_PROGRAM_USER_HIGH_KEY.<br>FALSE ignores the provided values. |
| XSK_EFUSEPL_AES_KEY | Default = 00000000000000000000000000000000000000000000000000000000000000000000000000000000<br>This value converted to hex buffer and written into the PL eFUSE array when write API is used. This value should be the AES Key, given in string format. It must be 64 characters long. Valid characters are 0-9, a-f, A-F. Any other character is considered an invalid string and will not burn AES Key.<br>To write AES Key, XSK_EFUSEPL_PROGRAM_AES_AND_USER_LOW_KEY must have a value of TRUE. |
| XSK_EFUSEPL_USER_LOW_KEY | Default = 00<br>This value is converted to a hexadecimal buffer and written into the PL eFUSE array when the write API is used. This value is the User Low Key given in string format. It must be two characters long; valid characters are 0-9,a-f, and A-F. Any other character is considered as an invalid string and will not burn the User Low Key.<br>To write the User Low Key, XSK_EFUSEPL_PROGRAM_AES_AND_USER_LOW_KEY must have a value of TRUE. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_HIGH_KEY | Default = 000000<br>The default value is converted to a hexadecimal buffer and written into the PL eFUSE array when the write API is used. This value is the User High Key given in string format. The buffer must be six characters long: valid characters are 0-9, a-f, A-F. Any other character is considered to be an invalid string and does not burn User High Key.<br>To write the User High Key, the XSK_EFUSEPL_PROGRAM_USER_HIGH_KEY must have a value of TRUE. |

# Zynq User-Configurable PL BBRAM Parameters

## Overview

The table below lists the MIO pins for Zynq PL BBRAM JTAG operations.

**Note**

The pin numbers listed in the table below are examples. You must assign appropriate pin numbers as per your hardware design.

| Pin Name | Pin Number |
|---|---|
| XSK_BBRAM_MIO_JTAG_TDI | (17) |
| XSK_BBRAM_MIO_JTAG_TDO | (21) |
| XSK_BBRAM_MIO_JTAG_TCK | (19) |
| XSK_BBRAM_MIO_JTAG_TMS | (20) |

The table below lists the MUX selection pin for Zynq BBRAM PL JTAG operations.

| Pin Name | Pin Number |
|---|---|
| XSK_BBRAM_MIO_JTAG_MUX_SELECT | (11) |

## Modules

- MUX Parameter for Zynq BBRAM PL JTAG Operations
- AES and User Key Parameters

## MUX Parameter for Zynq BBRAM PL JTAG Operations

The table below lists the MUX parameter for Zynq BBRAM PL JTAG operations.

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_MIO_MUX_SEL_DEFAULT_VAL | Default = LOW.<br>LOW writes zero on the MUX select line before PL_eFUSE writing.<br>HIGH writes one on the MUX select line before PL_eFUSE writing. |

## AES and User Key Parameters

The table below lists the AES and user key parameters.

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_AES_KEY | Default = XX.<br>AES key (in HEX) that must be programmed into BBRAM. |
| XSK_BBRAM_AES_KEY_SIZE_IN_BITS | Default = 256.<br>Size of AES key. Must be 256 bits. |

# UltraScale or UltraScale+ User-Configurable BBRAM PL Parameters

## Overview

Following parameters need to be configured.
Based on your inputs, BBRAM is programmed with the provided AES key.

## Modules

- AES Keys and Related Parameters
- DPA Protection for BBRAM key
- GPIO Device Used for Connecting PL Master JTAG Signals
- GPIO Pins Used for PL Master JTAG Signals
- GPIO Channels

## AES Keys and Related Parameters

The following table shows AES key related parameters.

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_0 | Default = FALSE<br>By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_0 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_0 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_0 and DPA protection cannot be enabled. |
| XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1 | Default = FALSE<br>By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_1 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1 and DPA protection cannot be enabled. |
| XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_2 | Default = FALSE<br>By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_2 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_2 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_2 and DPA protection cannot be enabled. |
| XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_3 | Default = FALSE<br>By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_3 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_3 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_3 and DPA protection cannot be enabled. |

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_0 | Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM, when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing the OBFUSCATED Key, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_0 should have TRUE value. |
| XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1 | Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM, when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing the OBFUSCATED Key, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1 should have TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_2 | Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM, when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing the OBFUSCATED Key, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_2 should have TRUE value. |
| XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_3 | Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM, when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing the OBFUSCATED Key, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_3 should have TRUE value. |
| XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_0 | Default = FALSE<br>TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_0 |
| XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_1 | Default = FALSE<br>TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_1 |
| XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_2 | Default = FALSE<br>TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_2 |

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_3 | Default = FALSE<br>TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_3 |
| XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_0 | Default = 000000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM,when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing AES key, XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_0 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_0 should have FALSE value. |
| XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_1 | Default = 000000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM,when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing AES key, XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_1 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1 should have FALSE value |

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_2 | Default = 0000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM, when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing AES key, XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_2 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_2 should have FALSE value |
| XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_3 | Default = 0000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa<br>The value mentioned in this will be converted to hex buffer and the key is programmed into BBRAM, when program API is called. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not program BBRAM.<br><br>**Note**<br>For writing AES key, XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ORDER_3 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_3 should have FALSE value |
| XSK_BBRAM_AES_KEY_SIZE_IN_BITS | Default= 256 Size of AES key must be 256 bits. |

# DPA Protection for BBRAM key

The following table shows DPA protection configurable parameter.

| Parameter Name | Description |
|---|---|
| XSK_BBRAM_DPA_PROTECT_ENABLE | Default = FALSE<br>By default, the DPA protection will be in disabled state.<br>TRUE will enable DPA protection with provided DPA count and configuration in XSK_BBRAM_DPA_COUNT and XSK_BBRAM_DPA_MODE respectively.<br>DPA protection cannot be enabled if BBRAM is been programmed with an obfuscated key. |
| XSK_BBRAM_DPA_COUNT | Default = 0<br>This input is valid only when DPA protection is enabled.<br>Valid range of values are 1 - 255 when DPA protection is enabled else 0. |
| XSK_BBRAM_DPA_MODE | Default = XSK_BBRAM_INVALID_CONFIGURATIONS<br>When DPA protection is enabled it can be XSK_BBRAM_INVALID_CONFIGURATIONS or XSK_BBRAM_ALL_CONFIGURATIONS If DPA protection is disabled this input provided over here is ignored. |

## GPIO Device Used for Connecting PL Master JTAG Signals

In hardware design MASTER JTAG can be connected to any one of the available GPIO devices, based on the design the following parameter should be provided with corresponding device ID of selected GPIO device.

| Master JTAG Signal | Description |
|---|---|
| XSK_BBRAM_AXI_GPIO_DEVICE_ID | Default = XPAR_AXI_GPIO_0_DEVICE_ID<br>This is for providing exact GPIO device ID, based on the design configuration this parameter can be modified to provide GPIO device ID which is used for connecting master jtag pins. |

## GPIO Pins Used for PL Master JTAG Signals

In Ultrascale the following GPIO pins are used for connecting MASTER_JTAG pins to access BBRAM. These can be changed depending on your hardware.The table below shows the GPIO pins used for PL MASTER JTAG signals.

| Master JTAG Signal | Default PIN Number |
|---|---|
| XSK_BBRAM_AXI_GPIO_JTAG_TDO | 0 |

| Master JTAG Signal | Default PIN Number |
|---|---|
| XSK_BBRAM_AXI_GPIO_JTAG_TDI | 0 |
| XSK_BBRAM_AXI_GPIO_JTAG_TMS | 1 |
| XSK_BBRAM_AXI_GPIO_JTAG_TCK | 2 |

## GPIO Channels

The following table shows GPIO channel number.

| Parameter | Default Channel Number | Master JTAG Signal Connected |
|---|---|---|
| XSK_BBRAM_GPIO_INPUT_CH | 2 | TDO |
| XSK_BBRAM_GPIO_OUTPUT_CH | 1 | TDI, TMS, TCK |

**Note**

All inputs and outputs of GPIO should be configured in single channel. For example, XSK_BBRAM_GPIO_INPUT_CH = XSK_BBRAM_GPIO_OUTPUT_CH = 1 or 2. Among (TDI, TCK, TMS) Outputs of GPIO cannot be connected to different GPIO channels all the 3 signals should be in same channel. TDO can be a other channel of (TDI, TCK, TMS) or the same. DPA protection can be enabled only when programming non-obfuscated key.

# UltraScale or UltraScale+ User-Configurable PL eFUSE Parameters

## Overview

The table below lists the user-configurable PL eFUSE parameters for UltraScale™ devices.

| Macro Name | Description |
|---|---|
| XSK_EFUSEPL_DISABLE_AES_KEY_READ | Default = FALSE<br>TRUE will permanently disable the write to FUSE_AES and check CRC for AES key by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_USER_KEY_READ | Default = FALSE<br>TRUE will permanently disable the write to 32 bit FUSE_USER and read of FUSE_USER key by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |

| Macro Name | Description |
|---|---|
| XSK_EFUSEPL_DISABLE_SECURE_READ | Default = FALSE<br>TRUE will permanently disable the write to FUSE_Secure block and reading of secure block by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_FUSE_CNTRL_WRITE | Default = FALSE.<br>TRUE will permanently disable the write to FUSE_CNTRL block by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_RSA_KEY_READ | Default = FALSE.<br>TRUE will permanently disable the write to FUSE_RSA block and reading of FUSE_RSA Hash by programming control bit of FUSE. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_KEY_WRITE | Default = FALSE.<br>TRUE will permanently disable the write to FUSE_AES block by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_USER_KEY_WRITE | Default = FALSE.<br>TRUE will permanently disable the write to FUSE_USER block by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_SECURE_WRITE | Default = FALSE.<br>TRUE will permanently disable the write to FUSE_SECURE block by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_RSA_HASH_WRITE | Default = FALSE.<br>TRUE will permanently disable the write to FUSE_RSA authentication key by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_DISABLE_128BIT_USER_KEY_WRITE | Default = FALSE.<br>TRUE will permanently disable the write to 128 bit FUSE_USER by programming control bit of FUSE.<br>FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPL_ALLOW_ENCRYPTED_ONLY | Default = FALSE.<br>TRUE will permanently allow encrypted bitstream only. FALSE will not modify this Secure bit of eFuse. |

| Macro Name | Description |
|---|---|
| XSK_EFUSEPL_FORCE_USE_FUSE_AES_ONLY | Default = FALSE.<br>TRUE then allows only FUSE's AES key as source of encryption FALSE then allows FPGA to configure an unencrypted bitstream or bitstream encrypted using key stored BBRAM or eFuse. |
| XSK_EFUSEPL_ENABLE_RSA_AUTH | Default = FALSE.<br>TRUE will enable RSA authentication of bitstream FALSE will not modify this secure bit of eFuse. |
| XSK_EFUSEPL_DISABLE_JTAG_CHAIN | Default = FALSE.<br>TRUE will disable JTAG permanently. FALSE will not modify this secure bit of eFuse. |
| XSK_EFUSEPL_DISABLE_TEST_ACCESS | Default = FALSE.<br>TRUE will disables Xilinx test access. FALSE will not modify this secure bit of eFuse. |
| XSK_EFUSEPL_DISABLE_AES_DECRYPTOR | Default = FALSE.<br>TRUE will disables decoder completely. FALSE will not modify this secure bit of eFuse. |
| XSK_EFUSEPL_ENABLE_OBFUSCATION_EFUSEAES | Default = FALSE.<br>TRUE will enable obfuscation feature for eFUSE AES key. |

# Modules

- GPIO Device Used for Connecting PL Master JTAG Signals
- GPIO Pins Used for PL Master JTAG and HWM Signals
- GPIO Channels
- SLR Selection to Program eFUSE on MONO/SSIT Devices
- eFUSE PL Read Parameters
- AES Keys and Related Parameters
- USER Keys (32-bit) and Related Parameters
- RSA Hash and Related Parameters
- USER Keys (128-bit) and Related Parameters
- AES key CRC verification

# GPIO Device Used for Connecting PL Master JTAG Signals

In hardware design MASTER JTAG can be connected to any one of the available GPIO devices, based on the design the following parameter should be provided with corresponding device ID of selected GPIO device.

| Master JTAG Signal | Description |
|---|---|
| XSK_EFUSEPL_AXI_GPIO_DEVICE_ID | Default = XPAR_AXI_GPIO_0_DEVICE_ID<br>This is for providing exact GPIO device ID, based on the design configuration this parameter can be modified to provide GPIO device ID which is used for connecting master jtag pins. |

# GPIO Pins Used for PL Master JTAG and HWM Signals

In Ultrascale the following GPIO pins are used for connecting MASTER_JTAG pins to access eFUSE. These can be changed depending on your hardware.The table below shows the GPIO pins used for PL MASTER JTAG signals.

| Master JTAG Signal | Default PIN Number |
|---|---|
| XSK_EFUSEPL_AXI_GPIO_JTAG_TDO | 0 |
| XSK_EFUSEPL_AXI_GPIO_HWM_READY | 0 |
| XSK_EFUSEPL_AXI_GPIO_HWM_END | 1 |
| XSK_EFUSEPL_AXI_GPIO_JTAG_TDI | 2 |
| XSK_EFUSEPL_AXI_GPIO_JTAG_TMS | 1 |
| XSK_EFUSEPL_AXI_GPIO_JTAG_TCK | 2 |
| XSK_EFUSEPL_AXI_GPIO_HWM_START | 3 |

# GPIO Channels

The following table shows GPIO channel number.

| Parameter | Default Channel Number | Master JTAG Signal Connected |
|---|---|---|
| XSK_EFUSEPL_GPIO_INPUT_CH | 2 | TDO |
| XSK_EFUSEPL_GPIO_OUTPUT_CH | 1 | TDI, TMS, TCK |

**Note**

All inputs and outputs of GPIO should be configured in single channel. For example, XSK_EFUSEPL_GPIO_INPUT_CH = XSK_EFUSEPL_GPIO_OUTPUT_CH = 1 or 2. Among (TDI, TCK, TMS) Outputs of GPIO cannot be connected to different GPIO channels all the 3 signals should be in same channel. TDO can be a other channel of (TDI, TCK, TMS) or the same.

## SLR Selection to Program eFUSE on MONO/SSIT Devices

The following table shows parameters for programming different SLRs.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 | Default = FALSE<br>TRUE will enable programming SLR config order 0's eFUSE. FALSE will disable programming. |
| XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_1 | Default = FALSE<br>TRUE will enable programming SLR config order 1's eFUSE. FALSE will disable programming. |
| XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 | Default = FALSE<br>TRUE will enable programming SLR config order 2's eFUSE. FALSE will disable programming. |
| XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 | Default = FALSE<br>TRUE will enable programming SLR config order 3's eFUSE. FALSE will disable programming. |

## eFUSE PL Read Parameters

The following table shows parameters related to read USER 32/128bit keys and RSA hash.

**Note**

For only reading keys it is not required to enable XSK_EFUSEPL_PGM_SLR1, XSK_EFUSEPL_PGM_SLR2, XSK_EFUSEPL_PGM_SLR3, XSK_EFUSEPL_PGM_SLR4 macros, they can be in FALSE state.

By enabling any of the below parameters, by default will read corresponding hash/key associated with all the available SLRs. For example, if XSK_EFUSEPL_READ_USER_KEY is TRUE, USER key for all the available SLRs will be read.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_READ_USER_KEY | Default = FALSE<br>TRUE will read 32 bit FUSE_USER from eFUSE of all available SLRs and each time updates in XilSKey_EPI instance parameter UserKeyReadback, which will be displayed on UART by example before reading next SLR. FALSE 32-bit FUSE_USER key read will not be performed. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_READ_RSA_KEY_HASH | Default = FALSE<br>TRUE will read FUSE_USER from eFUSE of all available SLRs and each time updates in XilSKey_EPl instance parameter RSAHashReadback, which will be displayed on UART by example before reading next SLR. FALSE FUSE_RSA_HASH read will not be performed. |
| XSK_EFUSEPL_READ_USER_KEY128_BIT | Default = FALSE<br>TRUE will read 128 bit USER key eFUSE of all available SLRs and each time updates in XilSKey_EPl instance parameter User128BitReadBack, which will be displayed on UART by example before reading next SLR. FALSE 128 bit USER key read will not be performed. |

# AES Keys and Related Parameters

**Note**

For programming AES key for MONO/SSIT device, the corresponding SLR should be selected and AES key programming should be enabled.

**Example 1** Enable the following parameters if you want to program AES key for SLR config order 2:

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 should have the TRUE value.
2. Enable AES key programming:
   - XSK_EFUSEPL_PROGRAM_AES_KEY should have the TRUE value.
3. Provide key to be programmed on SLR:
   - XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_2 should have key to be programmed in the string format.

**Example 2** Enable the following parameters if you want to program AES key on both SLR config order 0 and 3:

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 should have the TRUE value.
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 should have the TRUE value.
2. Enable AES key programming:
   - XSK_EFUSEPL_PROGRAM_AES_KEY should have the TRUE value.
3. Provide key to be programmed on SLR:

- XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_0 should have key to be programmed on SLR config order 0's eFUSE in the string format.
- XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_3 should have key to be programmed on SLR config order 3's eFUSE in the string format.

The following table shows AES key and related parameters to be taken care while programming AES key.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_PROGRAM_AES_KEY | Default = FALSE<br>TRUE will burn the AES key provided in: XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_ INDEX if corresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_ INDEX is TRUE and FALSE will ignore the values given. |
| XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_0 | Default = 00000000000000000000000000000000 00000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 0 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key.<br>**Note**<br>For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _0 are enabled with the TRUE value. |
| XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_1 | Default = 00000000000000000000000000000000 00000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 1 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key.<br>**Note**<br>For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _1 are enabled with the TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_2 | Default = 00000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 2 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key.<br>**Note**<br>For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 are enabled with TRUE value. |
| XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_3 | Default = 00000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 3 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key.<br>**Note**<br>For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 are enabled with TRUE value. |

# USER Keys (32-bit) and Related Parameters

**Note**

For programming USER key for MONO/SSIT device, the corresponding SLR should be selected and USER key programming should be enabled.

**Example 1** Enable the following parameters if you want to program USER key for SLR2:

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 should have the TRUE value.

2. Enable USER key programming:
    - XSK_EFUSEPL_PROGRAM_USER_KEY should have the TRUE value.
3. Provide key to be programmed on SLR:
    - XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_2 should have key to be programmed in the string format.

**Example 2** Enable the following parameters if you want to program USER key on SLR0 and SLR3:

1. Enable programming for SLR:
    - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 should have the TRUE value.
    - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 should have the TRUE value.
2. Enable USER key programming:
    - XSK_EFUSEPL_PROGRAM_USER_KEY should have the TRUE value.
3. Provide key to be programmed on SLR:
    - XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_0 should have key to be programmed in the string format.
    - XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_3 should have key to be programmed in the string format.

The following table shows USER key and related parameters to be taken care while programming USER key.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_PROGRAM_USER_KEY | Default = FALSE<br>TRUE will burn 32 bit User key given in XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_ INDEX if orresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_ INDEX is TRUE, FALSE will ignore the values given. |
| XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_0 | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_ 0 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_1 | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array when the write API is used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_1 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_2 | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_3 | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|

# RSA Hash and Related Parameters

**Note**

For programming RSA hash for MONO/SSIT device, the corresponding SLR should be selected and RSA hash programming should be enabled.

**Example 1** Enable the following parameters if you want to program RSA hash for SLR2:

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 should have the TRUE value.
2. Enable RSA hash programming:
   - XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH should have the TRUE value.
3. Provide hash to be programmed on SLR:
   - XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_2 should have hash to be programmed in the string format.

**Example 2** Enable the following parameters if you want to program RSA hash on SLR0 and SLR3:

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 should have the TRUE value.
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 should have the TRUE value.
2. Enable RSA hash programming:
   - XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH should have the TRUE value.
3. Provide hash to be programmed on SLR:
   - XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_0 should have hash to be programmed in the string format.
   - XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_3 should have hash to be programmed in the string format.

The following table shows RSA hash and related parameters to be taken care while programming RSA hash.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH | Default = FALSE<br>TRUE will burn RSA hash given in XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_INDEX if corresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_INDEX is TRUE, FALSE will ignore the values given. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_0 | Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1/MONO when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value.<br>**Note**<br>For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 are enabled with TRUE value. |
| XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_1 | Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value.<br>**Note**<br>For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_1 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_2 | Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value.<br>**Note**<br>For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 are enabled with TRUE value. |
| XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG_ORDER_3 | Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR4 when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value.<br>**Note**<br>For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 are enabled with TRUE value. |

# USER Keys (128-bit) and Related Parameters

**Note**

For programming USER key 128 bit for MONO/SSIT device, the corresponding SLR and programming for USER key 128 bit should be enabled.

**Example 1** Enable the following parameters if you want to program USER key 128-bit for SLR2:

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 should have the TRUE value.
2. Enable USER key programming:
   - XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT should have the TRUE value.
3. Provide key to be programmed on SLR:
   - XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_2 , XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_2, XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_2, XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_2 should have value to be programmed in the string format. The key should be provided as below
     - XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_2 holds 31:0 bits,
     - XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_2 holds 63:32 bits,
     - XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_2 holds 95:64 bits and
     - XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_2 holds 127:96 bits of whole 128 bit User key. The following table shows USER key 128 bit and related parameters.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT | Default = FALSE<br>TRUE will burn 128 bit User key given in: XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_INDEX, XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_INDEX, XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_INDEX, XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_INDEX if corresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_INDEX is TRUE, FALSE will ignore the values given. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_0 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_0 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_0 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_0 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_0 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_0 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_0 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_0 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_1 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_1 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_1 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_1 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_1 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_1 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_1 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_1 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_1 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_1 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_1 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_1 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_2 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_2 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_2 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_2 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_2 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_2 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_2 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_2 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_3 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_3 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_3 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_3 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 are enabled with TRUE value. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_3 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_3 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 are enabled with TRUE value. |
| XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_3 | Default = 00000000<br>Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_3 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key.<br>**Note**<br>For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 are enabled with TRUE value. |

**WARNING:** *If you want to program USER key for SLR 1 and AES key for SLR2 then this should be done separately. For this you need to enable the XSK_EFUSEPL_PGM_SLR1, XSK_EFUSEPL_PGM_SLR2, XSK_EFUSEPL_PROGRAM_USER_KEY, and XSK_EFUSEPL_PROGRAM_AES_KEY parameters with the TRUE value. If you do all the settings in one single go and provide the USER key in XSK_EFUSEPL_USER_KEY and AES key in XSK_EFUSEPL_AES_KEY_SLR2 then:*

- Enabling XSK_EFUSEPL_PROGRAM_USER_KEY will enable programming of USER key for both SLR1 And SLR2 as programming is enabled for both the SLR.
- Enabling XSK_EFUSEPL_PROGRAM_AES_KEY will enable programming of AES key for both SLR1 And SLR2 as programming is enabled for both the SLR.
- If you want to program USER key only for SLR1, then provided USER key will be programmed for SLR1 and Default key (all zeroes) will be programmed for SLR2.
- If you want to program AES key only for SLR2, then provided AES key will be programmed for SLR2 and Default key will be programmed for SLR1.
  To avoid all the above mentioned scenarios, if programming is required for different key on different SLR, separate runs should be done.

# AES key CRC verification

You cannot read the AES key.
You can verify only by providing the CRC of the expected AES key. The following lists the parameters that may help you in verifying the AES key:

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_CHECK_AES_KEY_CRC | Default = FALSE<br>TRUE will perform CRC check of FUSE_AES with provided CRC value in macro XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY. And result of CRC check will be updated in XilSKey_EPl instance parameter AESKeyMatched with either TRUE or FALSE. FALSE CRC check of FUSE_AES will not be performed. |
| XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_CONFIG_ORDER_0 | Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS<br>CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 0 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API.<br>For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_FOR_AES_ZEROS).<br>For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_AES_ZEROS_ULTRA_PLUS) |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_CONFIG_ORDER_1 | Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS<br>CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 1 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API.<br>For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_FOR_AES_ZEROS).<br>For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_AES_ZEROS_ULTRA_PLUS) |
| XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_CONFIG_ORDER_2 | Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS<br>CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 2 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API.<br>For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_FOR_AES_ZEROS).<br>For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_AES_ZEROS_ULTRA_PLUS) |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_CONFIG_ORDER_3 | Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS<br>CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 3 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API.<br>For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_FOR_AES_ZEROS).<br>For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_AES_ZEROS_ULTRA_PLUS) |

# Zynq UltraScale+ MPSoC User-Configurable PS eFUSE Parameters

## Overview

The table below lists the user-configurable PS eFUSE parameters for Zynq UltraScale+ MPSoC devices.

| Macro Name | Description |
|---|---|
| XSK_EFUSEPS_AES_RD_LOCK | Default = FALSE<br>TRUE will permanently disable the CRC check of FUSE_AES. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_AES_WR_LOCK | Default = FALSE<br>TRUE will permanently disable the writing to FUSE_AES block. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_ENC_ONLY | Default = FALSE<br>TRUE will permanently enable encrypted booting only using the Fuse key. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_BBRAM_DISABLE | Default = FALSE<br>TRUE will permanently disable the BBRAM key. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_ERR_DISABLE | Default = FALSE<br>TRUE will permanently disables the error messages in JTAG status register. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_JTAG_DISABLE | Default = FALSE<br>TRUE will permanently disable JTAG controller. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_DFT_DISABLE | Default = FALSE<br>TRUE will permanently disable DFT boot mode. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_PROG_GATE_DISABLE | Default = FALSE<br>TRUE will permanently disable PROG_GATE feature in PPD. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_SECURE_LOCK | Default = FALSE<br>TRUE will permanently disable reboot into JTAG mode when doing a secure lockdown. FALSE will not modify thi s control bit of eFuse. |

| Macro Name | Description |
|---|---|
| XSK_EFUSEPS_RSA_ENABLE | Default = FALSE<br>TRUE will permanently enable RSA authentication during boot. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_PPK0_WR_LOCK | Default = FALSE<br>TRUE will permanently disable writing to PPK0 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_PPK0_INVLD | Default = FALSE<br>TRUE will permanently revoke PPK0. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_PPK1_WR_LOCK | Default = FALSE<br>TRUE will permanently disable writing PPK1 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_PPK1_INVLD | Default = FALSE<br>TRUE will permanently revoke PPK1. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_USER_WRLK_0 | Default = FALSE<br>TRUE will permanently disable writing to USER_0 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_USER_WRLK_1 | Default = FALSE<br>TRUE will permanently disable writing to USER_1 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_USER_WRLK_2 | Default = FALSE<br>TRUE will permanently disable writing to USER_2 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_USER_WRLK_3 | Default = FALSE<br>TRUE will permanently disable writing to USER_3 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_USER_WRLK_4 | Default = FALSE<br>TRUE will permanently disable writing to USER_4 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_USER_WRLK_5 | Default = FALSE<br>TRUE will permanently disable writing to USER_5 efuses. FALSE will not modify this control bit of eFuse. |

| Macro Name | Description |
|---|---|
| XSK_EFUSEPS_USER_WRLK_6 | Default = FALSE<br>TRUE will permanently disable writing to USER_6 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_USER_WRLK_7 | Default = FALSE<br>TRUE will permanently disable writing to USER_7 efuses. FALSE will not modify this control bit of eFuse. |
| XSK_EFUSEPS_LBIST_EN | Default = FALSE<br>TRUE will permanently enables logic BIST to be run during boot. FALSE will not modify this control bit of eFUSE. |
| XSK_EFUSEPS_LPD_SC_EN | Default = FALSE<br>TRUE will permanently enables zeroization of registers in Low Power Domain(LPD) during boot. FALSE will not modify this control bit of eFUSE. |
| XSK_EFUSEPS_FPD_SC_EN | Default = FALSE<br>TRUE will permanently enables zeroization of registers in Full Power Domain(FPD) during boot. FALSE will not modify this control bit of eFUSE. |
| XSK_EFUSEPS_PBR_BOOT_ERR | Default = FALSE<br>TRUE will permanently enables the boot halt when there is any PMU error. FALSE will not modify this control bit of eFUSE. |

# Modules

- AES Keys and Related Parameters
- User Keys and Related Parameters
- PPK0 Keys and Related Parameters
- PPK1 Keys and Related Parameters
- SPK ID and Related Parameters

# AES Keys and Related Parameters

The following table shows AES key related parameters.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_WRITE_AES_KEY | Default = FALSE<br>TRUE will burn the AES key provided in XSK_EFUSEPS_AES_KEY. FALSE will ignore the key provide XSK_EFUSEPS_AES_KEY. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_AES_KEY | Default = 0000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key.<br>**Note**<br>For writing the AES Key, XSK_EFUSEPS_WRITE_AES_KEY should have TRUE value. |
| XSK_EFUSEPS_CHECK_AES_KEY_CRC | Default value is FALSE. TRUE will check the CRC provided in XSK_EFUSEPS_AES_KEY. CRC verification is done after programming AES key to verify the key is programmed properly or not, if not library error outs the same. So While programming AES key it is not necessary to verify the AES key again.<br>**Note**<br>Please make sure if intention is to check only CRC of the provided key and not programming AES key then do not modify XSK_EFUSEPS_WRITE_AES_KEY (TRUE will Program key). |

# User Keys and Related Parameters

Single bit programming is allowed for all the user eFUSEs.
When you request to revert already programmed bit, the library will return an error. Also, if the user eFUSEs is non-zero, the library will not throw an error for valid requests. The following table shows the user keys and related parameters.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_WRITE_USER0_FUSE | Default = FALSE<br>TRUE will burn User0 Fuse provided in XSK_EFUSEPS_USER0_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER0_FUSES |
| XSK_EFUSEPS_WRITE_USER1_FUSE | Default = FALSE<br>TRUE will burn User1 Fuse provided in XSK_EFUSEPS_USER1_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER1_FUSES |
| XSK_EFUSEPS_WRITE_USER2_FUSE | Default = FALSE<br>TRUE will burn User2 Fuse provided in XSK_EFUSEPS_USER2_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER2_FUSES |
| XSK_EFUSEPS_WRITE_USER3_FUSE | Default = FALSE<br>TRUE will burn User3 Fuse provided in XSK_EFUSEPS_USER3_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER3_FUSES |
| XSK_EFUSEPS_WRITE_USER4_FUSE | Default = FALSE<br>TRUE will burn User4 Fuse provided in XSK_EFUSEPS_USER4_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER4_FUSES |
| XSK_EFUSEPS_WRITE_USER5_FUSE | Default = FALSE<br>TRUE will burn User5 Fuse provided in XSK_EFUSEPS_USER5_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER5_FUSES |
| XSK_EFUSEPS_WRITE_USER6_FUSE | Default = FALSE<br>TRUE will burn User6 Fuse provided in XSK_EFUSEPS_USER6_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER6_FUSES |
| XSK_EFUSEPS_WRITE_USER7_FUSE | Default = FALSE<br>TRUE will burn User7 Fuse provided in XSK_EFUSEPS_USER7_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER7_FUSES |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_USER0_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User0 Fuse, XSK_EFUSEPS_WRITE_USER0_FUSE should have TRUE value |
| XSK_EFUSEPS_USER1_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User1 Fuse, XSK_EFUSEPS_WRITE_USER1_FUSE should have TRUE value |
| XSK_EFUSEPS_USER2_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User2 Fuse, XSK_EFUSEPS_WRITE_USER2_FUSE should have TRUE value |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_USER3_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User3 Fuse, XSK_EFUSEPS_WRITE_USER3_FUSE should have TRUE value |
| XSK_EFUSEPS_USER4_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User4 Fuse, XSK_EFUSEPS_WRITE_USER4_FUSE should have TRUE value |
| XSK_EFUSEPS_USER5_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User5 Fuse, XSK_EFUSEPS_WRITE_USER5_FUSE should have TRUE value |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_USER6_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User6 Fuse, XSK_EFUSEPS_WRITE_USER6_FUSE should have TRUE value |
| XSK_EFUSEPS_USER7_FUSES | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the User7 Fuse, XSK_EFUSEPS_WRITE_USER7_FUSE should have TRUE value |

# PPK0 Keys and Related Parameters

The following table shows the PPK0 keys and related parameters.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_WRITE_PPK0_SHA3_HASH | Default = FALSE<br>TRUE will burn PPK0 sha3 hash provided in XSK_EFUSEPS_PPK0_SHA3_HASH. FALSE will ignore the hash provided in XSK_EFUSEPS_PPK0_SHA3_HASH. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_PPK0_IS_SHA3 | Default = TRUE<br>TRUE XSK_EFUSEPS_PPK0_SHA3_HASH should be of string length 96 it specifies that PPK0 is used to program SHA3 hash. FALSE XSK_EFUSEPS_PPK0_SHA3_HASH should be of string length 64 it specifies that PPK0 is used to program SHA2 hash. |
| XSK_EFUSEPS_PPK0_HASH | Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 96 or 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn PPK0 hash. Note that,for writing the PPK0 hash, XSK_EFUSEPS_WRITE_PPK0_SHA3_HASH should have TRUE value. While writing SHA2 hash, length should be 64 characters long XSK_EFUSEPS_PPK0_IS_SHA3 macro has to be made FALSE. While writing SHA3 hash, length should be 96 characters long and XSK_EFUSEPS_PPK0_IS_SHA3 macro should be made TRUE |

# PPK1 Keys and Related Parameters

The following table shows the PPK1 keys and related parameters.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_WRITE_PPK1_SHA3_HASH | Default = FALSE<br>TRUE will burn PPK1 sha3 hash provided in XSK_EFUSEPS_PPK1_SHA3_HASH. FALSE will ignore the hash provided in XSK_EFUSEPS_PPK1_SHA3_HASH. |
| XSK_EFUSEPS_PPK1_IS_SHA3 | Default = TRUE<br>TRUE XSK_EFUSEPS_PPK1_SHA3_HASH should be of string length 96 it specifies that PPK1 is used to program SHA3 hash. FALSE XSK_EFUSEPS_PPK1_SHA3_HASH should be of string length 64 it specifies that PPK1 is used to program SHA2 hash. |

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_PPK1_HASH | Default = 00000000000000000000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 64 or 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn PPK1 hash. Note that,for writing the PPK11 hash, XSK_EFUSEPS_WRITE_PPK1_SHA3_HASH should have TRUE value. By default, PPK1 hash will be provided with 64 character length to program PPK1 hash with sha2 hash so XSK_EFUSEPS_PPK1_IS_SHA3 also will be in FALSE state. But to program PPK1 hash with SHA3 hash make XSK_EFUSEPS_PPK1_IS_SHA3 to TRUE and provide sha3 hash of length 96 characters XSK_EFUSEPS_PPK1_HASH so that one can program sha3 hash. |

## SPK ID and Related Parameters

The following table shows the SPK ID and related parameters.

| Parameter Name | Description |
|---|---|
| XSK_EFUSEPS_WRITE_SPKID | Default = FALSE<br>TRUE will burn SPKID provided in XSK_EFUSEPS_SPK_ID. FALSE will ignore the hash provided in XSK_EFUSEPS_SPK_ID. |
| XSK_EFUSEPS_SPK_ID | Default = 00000000<br>The value mentioned in this will be converted to hex buffer and written into the Zynq UltraScale+ MPSoC PS eFUSE array when write API used. This value should be given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn SPK ID.<br>**Note**<br>For writing the SPK ID, XSK_EFUSEPS_WRITE_SPKID should have TRUE value. |

**Note**

PPK hash should be unmodified hash generated by bootgen. Single bit programming is allowed for User FUSEs (0 to 7), if you specify a value that tries to set a bit that was previously programmed to 1 back to 0, you will get an error. you have to provide already programmed bits also along with new requests.

# Zynq UltraScale+ MPSoC User-Configurable PS BBRAM Parameters

The table below lists the AES and user key parameters.

| Parameter Name | Description |
|---|---|
| XSK_ZYNQMP_BBRAMPS_AES_KEY | Default = 0000000000000000000000000000000000000000000000000000000000000000<br>AES key (in HEX) that must be programmed into BBRAM. |
| XSK_ZYNQMP_BBRAMPS_AES_KEY_LEN_IN_BYTES | Default = 32.<br>Length of AES key in bytes. |
| XSK_ZYNQMP_BBRAMPS_AES_KEY_LEN_IN_BITS | Default = 256.<br>Length of AES key in bits. |
| XSK_ZYNQMP_BBRAMPS_AES_KEY_STR_LEN | Default = 64.<br>String length of the AES key. |

# Zynq UltraScale+ MPSoC User-Configurable PS PUF Parameters

The table below lists the user-configurable PS PUF parameters for Zynq UltraScale+ MPSoC devices.

| Macro Name | Description |
|---|---|
| XSK_PUF_INFO_ON_UART | Default = FALSE<br>TRUE will display syndrome data on UART com port<br>FALSE will display any data on UART com port. |
| XSK_PUF_PROGRAM_EFUSE | Default = FALSE<br>TRUE will program the generated syndrome data, CHash and Auxilary values, Black key.<br>FALSE will not program data into eFUSE. |
| XSK_PUF_IF_CONTRACT_MANUFACTURER | Default = FALSE<br>This should be enabled when application is hand over to contract manufacturer.<br>TRUE will allow only authenticated application.<br>FALSE authentication is not mandatory. |
| XSK_PUF_REG_MODE | Default = XSK_PUF_MODE4K<br>PUF registration is performed in 4K mode. For only understanding it is provided in this file, but user is not supposed to modify this. |

| Macro Name | Description |
|---|---|
| XSK_PUF_READ_SECUREBITS | Default = FALSE<br>TRUE will read status of the puf secure bits from eFUSE and will be displayed on UART. FALSE will not read secure bits. |
| XSK_PUF_PROGRAM_SECUREBITS | Default = FALSE<br>TRUE will program PUF secure bits based on the user input provided at XSK_PUF_SYN_INVALID, XSK_PUF_SYN_WRLK and XSK_PUF_REGISTER_DISABLE.<br>FALSE will not program any PUF secure bits. |
| XSK_PUF_SYN_INVALID | Default = FALSE<br>TRUE will permanently invalidate the already programmed syndrome data.<br>FALSE will not modify anything |
| XSK_PUF_SYN_WRLK | Default = FALSE<br>TRUE will permanently disable programming syndrome data into eFUSE.<br>FALSE will not modify anything. |
| XSK_PUF_REGISTER_DISABLE | Default = FALSE<br>TRUE permanently does not allow PUF syndrome data registration.<br>FALSE will not modify anything. |
| XSK_PUF_RESERVED | Default = FALSE<br>TRUE programs this reserved eFUSE bit. FALSE will not modify anything. |
| XSK_PUF_AES_KEY | Default = 0000000000000000000000000000000000000000000000000000000000000000<br>The value mentioned in this will be converted to hex buffer and encrypts this with PUF helper data and generates a black key and written into the Zynq UltraScale+ MPSoC PS eFUSE array when XSK_PUF_PROGRAM_EFUSE macro is TRUE. This value should be given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key. Note Provided here should be red key and application calculates the black key and programs into eFUSE if XSK_PUF_PROGRAM_EFUSE macro is TRUE. To avoid programming eFUSE results can be displayed on UART com port by making XSK_PUF_INFO_ON_UART to TRUE. |

| Macro Name | Description |
|---|---|
| XSK_PUF_BLACK_KEY_IV | Default = 00000000000000000000000000<br>The value mentioned here will be converted to hex buffer. This is Initialization vector(IV) which is used to generated black key with provided AES key and generated PUF key.<br>This value should be given in string format. It should be 24 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string. |

XILINX

Chapter 9

# Error Codes

## Overview

The application error code is 32 bits long.
For example, if the error code for PS is `0x8A05`:

- `0x8A` indicates that a write error has occurred while writing RSA Authentication bit.
- `0x05` indicates that write error is due to the write temperature out of range.

Applications have the following options on how to show error status. Both of these methods of conveying the status are implemented by default. However, UART is required to be present and initialized for status to be displayed through UART.

- Send the error code through UART pins
- Write the error code in the reboot status register

## Modules


- PL eFUSE Error Codes
- PS eFUSE Error Codes
- Zynq UltraScale+ MPSoC BBRAM PS Error Codes


## PL eFUSE Error Codes

***XSK_EFUSEPL_ERROR_NONE*** 0
No error.

***XSK_EFUSEPL_ERROR_ROW_NOT_ZERO*** 0x10
Row is not zero.

***XSK_EFUSEPL_ERROR_READ_ROW_OUT_OF_RANGE*** 0x11
Read Row is out of range.

***XSK_EFUSEPL_ERROR_READ_MARGIN_OUT_OF_RANGE*** 0x12
Read Margin is out of range.

***XSK_EFUSEPL_ERROR_READ_BUFFER_NULL*** 0x13
No buffer for read.

***XSK_EFUSEPL_ERROR_READ_BIT_VALUE_NOT_SET*** 0x14
Read bit not set.

***XSK_EFUSEPL_ERROR_READ_BIT_OUT_OF_RANGE*** 0x15
Read bit is out of range.

***XSK_EFUSEPL_ERROR_READ_TMEPERATURE_OUT_OF_RANGE*** 0x16
Temperature obtained from XADC is out of range to read.

***XSK_EFUSEPL_ERROR_READ_VCCAUX_VOLTAGE_OUT_OF_RANGE*** 0x17
VCCAUX obtained from XADC is out of range to read.

***XSK_EFUSEPL_ERROR_READ_VCCINT_VOLTAGE_OUT_OF_RANGE*** 0x18
VCCINT obtained from XADC is out of range to read.

***XSK_EFUSEPL_ERROR_WRITE_ROW_OUT_OF_RANGE*** 0x19
To write row is out of range.

***XSK_EFUSEPL_ERROR_WRITE_BIT_OUT_OF_RANGE*** 0x1A
To read bit is out of range.

***XSK_EFUSEPL_ERROR_WRITE_TMEPERATURE_OUT_OF_RANGE*** 0x1B
To eFUSE write Temperature obtained from XADC is outof range.

***XSK_EFUSEPL_ERROR_WRITE_VCCAUX_VOLTAGE_OUT_OF_RANGE*** 0x1C
To write eFUSE VCCAUX obtained from XADC is out of range.

***XSK_EFUSEPL_ERROR_WRITE_VCCINT_VOLTAGE_OUT_OF_RANGE*** 0x1D
To write into eFUSE VCCINT obtained from XADC is out of range.

***XSK_EFUSEPL_ERROR_FUSE_CNTRL_WRITE_DISABLED*** 0x1E
Fuse control write is disabled.

***XSK_EFUSEPL_ERROR_CNTRL_WRITE_BUFFER_NULL*** 0x1F
Buffer pointer that is supposed to contain control data is null.

***XSK_EFUSEPL_ERROR_NOT_VALID_KEY_LENGTH*** 0x20
Key length invalid.

***XSK_EFUSEPL_ERROR_ZERO_KEY_LENGTH*** 0x21
Key length zero.

***XSK_EFUSEPL_ERROR_NOT_VALID_KEY_CHAR*** 0x22
Invalid key characters.

***XSK_EFUSEPL_ERROR_NULL_KEY*** 0x23
Null key.

***XSK_EFUSEPL_ERROR_FUSE_SEC_WRITE_DISABLED*** 0x24
Secure bits write is disabled.

***XSK_EFUSEPL_ERROR_FUSE_SEC_READ_DISABLED*** 0x25
Secure bits reading is disabled.

***XSK_EFUSEPL_ERROR_SEC_WRITE_BUFFER_NULL*** 0x26
Buffer to write into secure block is NULL.

***XSK_EFUSEPL_ERROR_READ_PAGE_OUT_OF_RANGE*** 0x27
Page is out of range.

***XSK_EFUSEPL_ERROR_FUSE_ROW_RANGE*** 0x28
Row is out of range.

***XSK_EFUSEPL_ERROR_IN_PROGRAMMING_ROW*** 0x29
Error programming fuse row.

***XSK_EFUSEPL_ERROR_PRGRMG_ROWS_NOT_EMPTY*** 0x2A
Error when tried to program non Zero rows of eFUSE.

***XSK_EFUSEPL_ERROR_HWM_TIMEOUT*** 0x80
Error when hardware module is exceeded the time for programming eFUSE.

***XSK_EFUSEPL_ERROR_USER_FUSE_REVERT*** 0x90
Error occurs when user requests to revert already programmed user eFUSE bit.

***XSK_EFUSEPL_ERROR_KEY_VALIDATION*** 0xF000
Invalid key.

***XSK_EFUSEPL_ERROR_PL_STRUCT_NULL*** 0x1000
Null PL structure.

***XSK_EFUSEPL_ERROR_JTAG_SERVER_INIT*** 0x1100
JTAG server initialization error.

***XSK_EFUSEPL_ERROR_READING_FUSE_CNTRL*** 0x1200
Error reading fuse control.

***XSK_EFUSEPL_ERROR_DATA_PROGRAMMING_NOT_ALLOWED*** 0x1300
Data programming not allowed.

***XSK_EFUSEPL_ERROR_FUSE_CTRL_WRITE_NOT_ALLOWED*** 0x1400
Fuse control write is disabled.

***XSK_EFUSEPL_ERROR_READING_FUSE_AES_ROW*** 0x1500
Error reading fuse AES row.

***XSK_EFUSEPL_ERROR_AES_ROW_NOT_EMPTY*** 0x1600
AES row is not empty.

***XSK_EFUSEPL_ERROR_PROGRAMMING_FUSE_AES_ROW*** 0x1700
Error programming fuse AES row.

***XSK_EFUSEPL_ERROR_READING_FUSE_USER_DATA_ROW*** 0x1800
Error reading fuse user row.

***XSK_EFUSEPL_ERROR_USER_DATA_ROW_NOT_EMPTY*** 0x1900
User row is not empty.

***XSK_EFUSEPL_ERROR_PROGRAMMING_FUSE_DATA_ROW*** 0x1A00
Error programming fuse user row.

***XSK_EFUSEPL_ERROR_PROGRAMMING_FUSE_CNTRL_ROW*** 0x1B00
Error programming fuse control row.

***XSK_EFUSEPL_ERROR_XADC*** 0x1C00
XADC error.

***XSK_EFUSEPL_ERROR_INVALID_REF_CLK*** 0x3000
Invalid reference clock.

***XSK_EFUSEPL_ERROR_FUSE_SEC_WRITE_NOT_ALLOWED*** 0x1D00
Error in programming secure block.

***XSK_EFUSEPL_ERROR_READING_FUSE_STATUS*** 0x1E00
Error in reading FUSE status.

***XSK_EFUSEPL_ERROR_FUSE_BUSY*** 0x1F00
Fuse busy.

***XSK_EFUSEPL_ERROR_READING_FUSE_RSA_ROW*** 0x2000
Error in reading FUSE RSA block.

***XSK_EFUSEPL_ERROR_TIMER_INTIALISE_ULTRA*** 0x2200
Error in initiating Timer.

***XSK_EFUSEPL_ERROR_READING_FUSE_SEC*** 0x2300
Error in reading FUSE secure bits.

***XSK_EFUSEPL_ERROR_PRGRMG_FUSE_SEC_ROW*** 0x2500
Error in programming Secure bits of efuse.

***XSK_EFUSEPL_ERROR_PRGRMG_USER_KEY*** 0x4000
Error in programming 32 bit user key.

***XSK_EFUSEPL_ERROR_PRGRMG_128BIT_USER_KEY*** 0x5000
Error in programming 128 bit User key.

***XSK_EFUSEPL_ERROR_PRGRMG_RSA_HASH*** 0x8000
Error in programming RSA hash.

# PS eFUSE Error Codes

***XSK_EFUSEPS_ERROR_NONE*** 0
No error.

***XSK_EFUSEPS_ERROR_ADDRESS_XIL_RESTRICTED*** 0x01
Address is restricted.

***XSK_EFUSEPS_ERROR_READ_TMEPERATURE_OUT_OF_RANGE*** 0x02
Temperature obtained from XADC is out of range.

***XSK_EFUSEPS_ERROR_READ_VCCPAUX_VOLTAGE_OUT_OF_RANGE*** 0x03
VCCAUX obtained from XADC is out of range.

***XSK_EFUSEPS_ERROR_READ_VCCPINT_VOLTAGE_OUT_OF_RANGE*** 0x04
VCCINT obtained from XADC is out of range.

***XSK_EFUSEPS_ERROR_WRITE_TEMPERATURE_OUT_OF_RANGE*** 0x05
Temperature obtained from XADC is out of range.

***XSK_EFUSEPS_ERROR_WRITE_VCCPAUX_VOLTAGE_OUT_OF_RANGE*** 0x06
VCCAUX obtained from XADC is out of range.

***XSK_EFUSEPS_ERROR_WRITE_VCCPINT_VOLTAGE_OUT_OF_RANGE*** 0x07
VCCINT obtained from XADC is out of range.

***XSK_EFUSEPS_ERROR_VERIFICATION*** 0x08
Verification error.

***XSK_EFUSEPS_ERROR_RSA_HASH_ALREADY_PROGRAMMED*** 0x09
RSA hash was already programmed.

***XSK_EFUSEPS_ERROR_CONTROLLER_MODE*** 0x0A
Controller mode error

***XSK_EFUSEPS_ERROR_REF_CLOCK*** 0x0B
Reference clock not between 20 to 60MHz

***XSK_EFUSEPS_ERROR_READ_MODE*** 0x0C
Not supported read mode

***XSK_EFUSEPS_ERROR_XADC_CONFIG*** 0x0D
XADC configuration error.

***XSK_EFUSEPS_ERROR_XADC_INITIALIZE*** 0x0E
XADC initialization error.

***XSK_EFUSEPS_ERROR_XADC_SELF_TEST*** 0x0F
XADC self-test failed.

***XSK_EFUSEPS_ERROR_PARAMETER_NULL*** 0x10
Passed parameter null.

***XSK_EFUSEPS_ERROR_STRING_INVALID*** 0x20
Passed string is invalid.

***XSK_EFUSEPS_ERROR_AES_ALREADY_PROGRAMMED*** 0x12
AES key is already programmed.

***XSK_EFUSEPS_ERROR_SPKID_ALREADY_PROGRAMMED*** 0x13
SPK ID is already programmed.

***XSK_EFUSEPS_ERROR_PPK0_HASH_ALREADY_PROGRAMMED*** 0x14
PPK0 hash is already programmed.

***XSK_EFUSEPS_ERROR_PPK1_HASH_ALREADY_PROGRAMMED*** 0x15
PPK1 hash is already programmed.

***XSK_EFUSEPS_ERROR_IN_TBIT_PATTERN*** 0x16
Error in TBITS pattern .

***XSK_EFUSEPS_ERROR_PROGRAMMING*** 0x00A0
Error in programming eFUSE.

***XSK_EFUSEPS_ERROR_READ*** 0x00B0
Error in reading.

***XSK_EFUSEPS_ERROR_BYTES_REQUEST*** 0x00C0
Error in requested byte count.

***XSK_EFUSEPS_ERROR_RESRVD_BITS_PRGRMG*** 0x00D0
Error in programming reserved bits.

***XSK_EFUSEPS_ERROR_ADDR_ACCESS*** 0x00E0
Error in accessing requested address.

***XSK_EFUSEPS_ERROR_READ_NOT_DONE*** 0x00F0
Read not done

***XSK_EFUSEPS_ERROR_PS_STRUCT_NULL*** 0x8100
PS structure pointer is null.

***XSK_EFUSEPS_ERROR_XADC_INIT*** 0x8200
XADC initialization error.

***XSK_EFUSEPS_ERROR_CONTROLLER_LOCK*** 0x8300
PS eFUSE controller is locked.

***XSK_EFUSEPS_ERROR_EFUSE_WRITE_PROTECTED*** 0x8400
PS eFUSE is write protected.

***XSK_EFUSEPS_ERROR_CONTROLLER_CONFIG*** 0x8500
Controller configuration error.

***XSK_EFUSEPS_ERROR_PS_PARAMETER_WRONG*** 0x8600
PS eFUSE parameter is not TRUE/FALSE.

***XSK_EFUSEPS_ERROR_WRITE_128K_CRC_BIT*** 0x9100
Error in enabling 128K CRC.

***XSK_EFUSEPS_ERROR_WRITE_NONSECURE_INITB_BIT*** 0x9200
Error in programming NON secure bit.

***XSK_EFUSEPS_ERROR_WRITE_UART_STATUS_BIT*** 0x9300
Error in writing UART status bit.

***XSK_EFUSEPS_ERROR_WRITE_RSA_HASH*** 0x9400
Error in writing RSA key.

***XSK_EFUSEPS_ERROR_WRITE_RSA_AUTH_BIT*** 0x9500
Error in enabling RSA authentication bit.

***XSK_EFUSEPS_ERROR_WRITE_WRITE_PROTECT_BIT*** 0x9600
Error in writing write-protect bit.

***XSK_EFUSEPS_ERROR_READ_HASH_BEFORE_PROGRAMMING*** 0x9700
Check RSA key before trying to program.

***XSK_EFUSEPS_ERROR_WRTIE_DFT_JTAG_DIS_BIT*** 0x9800
Error in programming DFT JTAG disable bit.

***XSK_EFUSEPS_ERROR_WRTIE_DFT_MODE_DIS_BIT*** 0x9900
Error in programming DFT MODE disable bit.

***XSK_EFUSEPS_ERROR_WRTIE_AES_CRC_LK_BIT*** 0x9A00
Error in enabling AES's CRC check lock.

***XSK_EFUSEPS_ERROR_WRTIE_AES_WR_LK_BIT*** 0x9B00
Error in programming AES write lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USE_AESONLY_EN_BIT*** 0x9C00
Error in programming use AES only bit.

***XSK_EFUSEPS_ERROR_WRTIE_BBRAM_DIS_BIT*** 0x9D00
Error in programming BBRAM disable bit.

***XSK_EFUSEPS_ERROR_WRTIE_PMU_ERR_DIS_BIT*** 0x9E00
Error in programming PMU error disable bit.

***XSK_EFUSEPS_ERROR_WRTIE_JTAG_DIS_BIT*** 0x9F00
Error in programming JTAG disable bit.

***XSK_EFUSEPS_ERROR_READ_RSA_HASH*** 0xA100
Error in reading RSA key.

***XSK_EFUSEPS_ERROR_WRONG_TBIT_PATTERN*** 0xA200
Error in programming TBIT pattern.

***XSK_EFUSEPS_ERROR_WRITE_AES_KEY*** 0xA300
Error in programming AES key.

***XSK_EFUSEPS_ERROR_WRITE_SPK_ID*** 0xA400
Error in programming SPK ID.

***XSK_EFUSEPS_ERROR_WRITE_USER_KEY*** 0xA500
Error in programming USER key.

***XSK_EFUSEPS_ERROR_WRITE_PPK0_HASH*** 0xA600
Error in programming PPK0 hash.

***XSK_EFUSEPS_ERROR_WRITE_PPK1_HASH*** 0xA700
Error in programming PPK1 hash.

***XSK_EFUSEPS_ERROR_WRITE_USER0_FUSE*** 0xC000
Error in programming USER 0 Fuses.

***XSK_EFUSEPS_ERROR_WRITE_USER1_FUSE*** 0xC100
Error in programming USER 1 Fuses.

***XSK_EFUSEPS_ERROR_WRITE_USER2_FUSE*** 0xC200
Error in programming USER 2 Fuses.

***XSK_EFUSEPS_ERROR_WRITE_USER3_FUSE*** 0xC300
Error in programming USER 3 Fuses.

***XSK_EFUSEPS_ERROR_WRITE_USER4_FUSE*** 0xC400
Error in programming USER 4 Fuses.

***XSK_EFUSEPS_ERROR_WRITE_USER5_FUSE*** 0xC500
Error in programming USER 5 Fuses.

***XSK_EFUSEPS_ERROR_WRITE_USER6_FUSE*** 0xC600
Error in programming USER 6 Fuses.

***XSK_EFUSEPS_ERROR_WRITE_USER7_FUSE*** 0xC700
Error in programming USER 7 Fuses.

***XSK_EFUSEPS_ERROR_WRTIE_USER0_LK_BIT*** 0xC800
Error in programming USER 0 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USER1_LK_BIT*** 0xC900
Error in programming USER 1 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USER2_LK_BIT*** 0xCA00
Error in programming USER 2 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USER3_LK_BIT*** 0xCB00
Error in programming USER 3 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USER4_LK_BIT*** 0xCC00
Error in programming USER 4 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USER5_LK_BIT*** 0xCD00
Error in programming USER 5 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USER6_LK_BIT*** 0xCE00
Error in programming USER 6 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_USER7_LK_BIT*** 0xCF00
Error in programming USER 7 fuses lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_PROG_GATE0_DIS_BIT*** 0xD000
Error in programming PROG_GATE0 disabling bit.

***XSK_EFUSEPS_ERROR_WRTIE_PROG_GATE1_DIS_BIT*** 0xD100
Error in programming PROG_GATE1 disabling bit.

***XSK_EFUSEPS_ERROR_WRTIE_PROG_GATE2_DIS_BIT*** 0xD200
Error in programming PROG_GATE2 disabling bit.

***XSK_EFUSEPS_ERROR_WRTIE_SEC_LOCK_BIT*** 0xD300
Error in programming SEC_LOCK bit.

***XSK_EFUSEPS_ERROR_WRTIE_PPK0_WR_LK_BIT*** 0xD400
Error in programming PPK0 write lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_PPK0_RVK_BIT*** 0xD500
Error in programming PPK0 revoke bit.

***XSK_EFUSEPS_ERROR_WRTIE_PPK1_WR_LK_BIT*** 0xD600
Error in programming PPK1 write lock bit.

***XSK_EFUSEPS_ERROR_WRTIE_PPK1_RVK_BIT*** 0xD700
Error in programming PPK0 revoke bit.

***XSK_EFUSEPS_ERROR_WRITE_PUF_SYN_INVLD*** 0xD800
Error while programming the PUF syndrome invalidate bit.

***XSK_EFUSEPS_ERROR_WRITE_PUF_SYN_WRLK*** 0xD900
Error while programming Syndrome write lock bit.

***XSK_EFUSEPS_ERROR_WRITE_PUF_SYN_REG_DIS*** 0xDA00
Error while programming PUF syndrome register disable bit.

***XSK_EFUSEPS_ERROR_WRITE_PUF_RESERVED_BIT*** 0xDB00
Error while programming PUF reserved bit.

***XSK_EFUSEPS_ERROR_WRITE_LBIST_EN_BIT*** 0xDC00
Error while programming LBIST enable bit.

***XSK_EFUSEPS_ERROR_WRITE_LPD_SC_EN_BIT*** 0xDD00
Error while programming LPD SC enable bit.

***XSK_EFUSEPS_ERROR_WRITE_FPD_SC_EN_BIT*** 0xDE00
Error while programming FPD SC enable bit.

***XSK_EFUSEPS_ERROR_WRITE_PBR_BOOT_ERR_BIT*** 0xDF00
Error while programming PBR boot error bit.

***XSK_EFUSEPS_ERROR_PUF_INVALID_REG_MODE*** 0xE000
Error when PUF registration is requested with invalid registration mode.

***XSK_EFUSEPS_ERROR_PUF_REG_WO_AUTH*** 0xE100
Error when write not allowed without authentication enabled.

***XSK_EFUSEPS_ERROR_PUF_REG_DISABLED*** 0xE200
Error when trying to do PUF registration and when PUF registration is disabled.

***XSK_EFUSEPS_ERROR_PUF_INVALID_REQUEST*** 0xE300
Error when an invalid mode is requested.

***XSK_EFUSEPS_ERROR_PUF_DATA_ALREADY_PROGRAMMED*** 0xE400
Error when PUF is already programmed in eFUSE.

***XSK_EFUSEPS_ERROR_PUF_DATA_OVERFLOW*** 0xE500
Error when an over flow occurs.

***XSK_EFUSEPS_ERROR_SPKID_BIT_CANT_REVERT*** 0xE600
Already programmed SPKID bit cannot be reverted

***XSK_EFUSEPS_ERROR_PUF_DATA_UNDERFLOW*** 0xE700
Error when an under flow occurs.

***XSK_EFUSEPS_ERROR_PUF_TIMEOUT*** 0xE800
Error when an PUF generation timedout.

***XSK_EFUSEPS_ERROR_PUF_ACCESS*** 0xE900
Error when an PUF Access violation.

***XSK_EFUSEPS_ERROR_CMPLTD_EFUSE_PRGRM_WITH_ERR*** 0x10000
eFUSE programming is completed with temp and vol read errors.

***XSK_EFUSEPS_ERROR_CACHE_LOAD*** 0x20000U
Error in re-loading CACHE.

***XSK_EFUSEPS_ERROR_FUSE_PROTECTED*** 0x00080000
Requested eFUSE is write protected.

***XSK_EFUSEPS_ERROR_USER_BIT_CANT_REVERT*** 0x00800000
Already programmed user FUSE bit cannot be reverted.

***XSK_EFUSEPS_ERROR_BEFORE_PROGRAMMING*** 0x08000000U
Error occurred before programming.

# Zynq UltraScale+ MPSoC BBRAM PS Error Codes

***XSK_ZYNQMP_BBRAMPS_ERROR_NONE*** 0
No error.

***XSK_ZYNQMP_BBRAMPS_ERROR_IN_PRGRMG_ENABLE*** 0x010
If this error is occurred programming is not possible.

***XSK_ZYNQMP_BBRAMPS_ERROR_IN_ZEROISE*** 0x20
zeroize bbram is failed.

***XSK_ZYNQMP_BBRAMPS_ERROR_IN_CRC_CHECK*** 0xB000
If this error is occurred programming is done but CRC check is failed.

***XSK_ZYNQMP_BBRAMPS_ERROR_IN_PRGRMG*** 0xC000
programming of key is failed.

***XSK_ZYNQMP_BBRAMPS_ERROR_IN_WRITE_CRC*** 0xE800
error write CRC value.

Chapter 10

# Status Codes

For Zynq® and UltraScale™, the status in the `xilskey_efuse_example.c` file is conveyed through a UART or reboot status register in the following format: `0xYYYYZZZZ`, where:

- YYYY represents the PS eFUSE Status.
- ZZZZ represents the PL eFUSE Status.

The table below lists the status codes.

| Status Code Values | Description |
|---|---|
| 0x0000ZZZZ | Represents PS eFUSE is successful and PL eFUSE process returned with error. |
| 0xYYYY0000 | Represents PL eFUSE is successful and PS eFUSE process returned with error. |
| 0xFFFF0000 | Represents PS eFUSE is not initiated and PL eFUSE is successful. |
| 0x0000FFFF | Represents PL eFUSE is not initiated and PS eFUSE is successful. |
| 0xFFFFZZZZ | Represents PS eFUSE is not initiated and PL eFUSE is process returned with error. |
| 0xYYYYFFFF | Represents PL eFUSE is not initiated and PS eFUSE is process returned with error. |

For Zynq UltraScale+ MPSoC, the status in the `xilskey_bbramps_zynqmp_example.c`, `xilskey_puf_registration.c` and `xilskey_efuseps_zynqmp_example.c` files is conveyed as 32 bit error code. Where Zero represents that no error has occurred and if the value is other than Zero, a 32 bit error code is returned.

XILINX

Chapter 11

# Procedures

This chapter provides detailed descriptions of the various procedures.

## Zynq eFUSE Writing Procedure Running from DDR as an Application

This sequence is same as the existing flow described below.

1. Provide the required inputs in `xilskey_input.h`, then compile the SDK project.
2. Take the latest FSBL (ELF), stitch the `<output>.elf` generated to it (using the bootgen utility), and generate a bootable image.
3. Write the generated binary image into the flash device (for example: QSPI, NAND).
4. To burn the eFUSE key bits, execute the image.

## Zynq eFUSE Driver Compilation Procedure for OCM

The procedure is as follows:

1. Open the linker script (`lscript.ld`) in the SDK project.
2. Map all the chapters to point to ps7_ram_0_S_AXI_BASEADDR instead of ps7_ddr_0_S_AXI_BASEADDR. For example, Click the Memory Region tab for the .text chapter and select ps7_ram_0_S_AXI_BASEADDR from the drop-down list.
3. Copy the ps7_init.c and ps7_init.h files from the hw_platform folder into the example folder.
4. In `xilskey_efuse_example.c`, un-comment the code that calls the `ps7_init()` routine.
5. Compile the project.
   The `<Project name>.elf` file is generated and is executed out of OCM.

When executed, this example displays the success/failure of the eFUSE application in a display message via UART (if UART is present and initialized) or the reboot status register.

# UltraScale eFUSE Access Procedure

The procedure is as follows:

1. After providing the required inputs in `xilskey_input.h`, compile the project.
2. Generate a memory mapped interface file using TCL command write_mem_info

   ```
   $Outfilename
   ```

3. Update memory has to be done using the tcl command updatemem.

   ```
   updatemem -meminfo $file.mmi -data $Outfilename.elf -bit $design.bit
   -proc design_1_i/microblaze_0 -out $Final.bit
   ```

4. Program the board using `$Final.bit bitstream`.
5. Output can be seen in UART terminal.

# UltraScale BBRAM Access Procedure

The procedure is as follows:

1. After providing the required inputs in the `xilskey_bbram_ultrascale_input.h`‘ file, compile the project.
2. Generate a memory mapped interface file using TCL command

   ```
   write_mem_info $Outfilename
   ```

3. Update memory has to be done using the tcl command updatemem:

   ```
   updatemem -meminfo $file.mmi -data $Outfilename.elf -bit $design.bit
   -proc design_1_i/microblaze_0 -out $Final.bit
   ```

4. Program the board using `$Final.bit bitstream`.
5. Output can be seen in UART terminal.

Appendix A

# Additional Resources and Legal Notices

## Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support .

## Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

## Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

**Automotive Applications Disclaimer**

AUTOMOTIVE PRODUCTS (IDENTIFIED AS ”XA” IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE (”SAFETY APPLICATION”) UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD (”SAFETY DESIGN”). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.