Xilinx Standalone Library Documentation

XilSKey Library v6.2

UG1191 (2017.1) April 5, 2017

Table of Contents

Appendix A Additional Resources and Legal Notices

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 All Programmable SoC, 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 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:
  - PS eFUSE holds the AES key, the user fuses, PPK0 and PPK1 hash, SPK ID and some user feature bits which can be used to enable or disable some Zynq UltraScale+ MPSoC features.
  - BBRAM holds the AES key.

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.

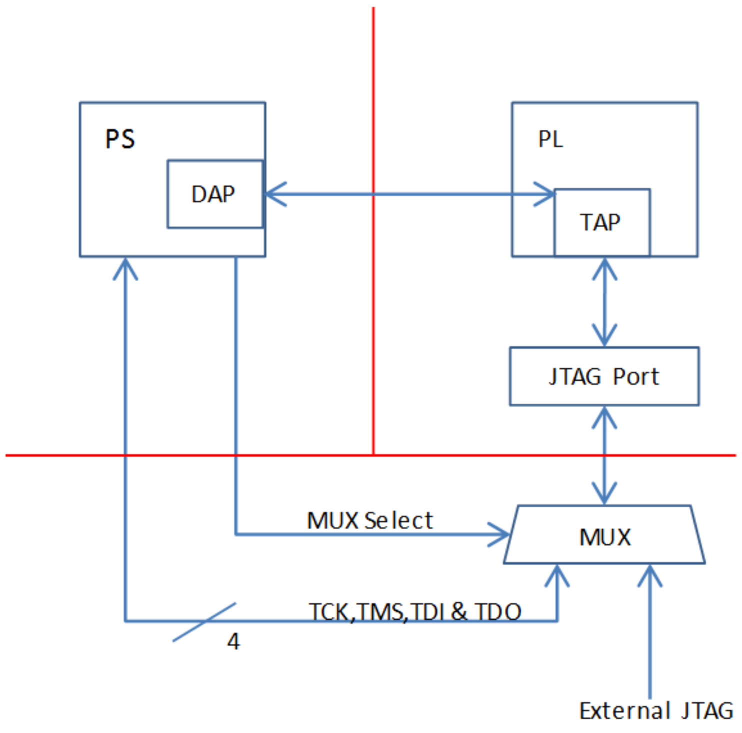

Figure 1.1: MUX Usage

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

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 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. 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.

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 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.

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)
- void XilSKey_ZynqMp_Bbram_Zeroise ()

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.

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.

void XilSKey_ZynqMp_Bbram_Zeroise ( )

This function zeroize's Bbram Key.

Parameters

None.	

Returns

None.

Note

BBRAM key will be zeroized.

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.
- Incase 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. 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 The system not in a position to write the requested PS eFUSE bits (because the bits are already written or not allowed to write) 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 occured.
- Incase 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 error if: The reference clock frequency is not in between 20 and 60MHz. Unable to unlock PS eFUSE controller or requested address corresponds to restricted bits. 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.

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 eFUSE other than 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 and programming PUF, the 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)

Function Documentation

u32 XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc (u32 *CrcValue*)

This function performs CRC check of AES key.

Parameters

CrcValue	32 bit CRC of expected AES key.

Returns

XST_SUCCESS - On success ErrorCode - on Failure

Note

For Calculating CRC of AES key use XilSKey_CrcCalculation() API or XilSkey_CrcCalculation_AesKey() API.

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

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

Parameters

UseFusePtr	Pointer to an array which holds the readback user fuse in.
UserFuse_Num	A variable which holds the user fuse number. Range is (User fuses: 0 to 7)
ReadOption	A variable which has to be provided by user based on this input reading is happend from cache or from efuse array. • 0 Reads from cache • 1 Reads from efuse array

Returns

XST_SUCCESS - On success ErrorCode - on Failure

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

This function is used to read PPK0 hash from efuse or cache based on user's read option.

Parameters

Ppk0Hash	A pointer to an array which holds the readback PPK0 hash in.
ReadOption	A variable which has to be provided by user based on this input reading is happend from cache or from efuse array. • 0 Reads from cache • 1 Reads from efuse array

Returns

XST_SUCCESS - On success ErrorCode - on Failure

Note

None.

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

This function is used to read PPK1 hash from efuse or cache based on user's read option.

Parameters

Ppk1Hash	Pointer to an array which holds the readback PPK1 hash in.
ReadOption	A variable which has to be provided by user based on this input reading is happend from cache or from efuse array. • 0 Reads from cache • 1 Reads from efuse array

Returns

XST_SUCCESS - On success 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	A variable which has to be provided by user based on this input reading is happend from cache or from efuse array. • 0 Reads from cache • 1 Reads from efuse array

Returns

XST_SUCCESS - On success ErrorCode - on Failure

void XilSKey_ZynqMp_EfusePs_ReadDna (u32 * *DnaRead*)

This function is used to read DNA from efuse.

Parameters

DnaRead	Pointer to 32 bit variable which holds the readback DNA in.

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	Variable which has to be provided by user based on this input reading is happend from cache or from efuse array. • 0 Reads from cache • 1 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

Cache reload is required for obtaining updated values through cache, to reload cache use XilSKey_ZynqMp_EfusePs_CacheLoad() API.

u32 XilSKey_ZynqMp_EfusePs_WritePufHelprData (XilSKey_Puf ∗ *InstancePtr*)

This function programs the PS efuse's with puf helper data of ZynqMp.

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 ZynqMp efuse.

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 API programs eFUSE with CHash value.

Parameters

InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

- XST_SUCCESS if chash is programmed successfully.
- Errorcode on failure

Note

To generate CHash value please use XilSKey_Puf_Registration API

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

This API reads efuse puf CHash Data from efuse array or cache based on the user read option.

Parameters

Address	Pointer which holds the read back value of chash
ReadOption	A u8 variable which has to be provided by user based on this input reading is happend from cache or from efuse array. • 0(XSK_EFUSEPS_READ_FROM_CACHE)Reads from cache • 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 ReadOption 0.

u32 XilSKey_ZynqMp_EfusePs_WritePufAux (XilSKey_Puf * *InstancePtr*)

This API programs efuse puf Auxilary Data.

Parameters

InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

- XST_SUCCESS if programs successfully.
- Errorcode on failure

Note

To generate Auxilary data please use the below API u32 XilSKey_Puf_Registration(XilSKey_Puf *InstancePtr)

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

This API reads efuse puf Auxilary Data from efuse array or cache based on user read option.

Parameters

Address	Pointer which holds the read back value of Auxilary
ReadOption	A u8 variable which has to be provided by user based on this input reading is happend from cache or from efuse array. • 0(XSK_EFUSEPS_READ_FROM_CACHE)Reads from cache • 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 ReadOption 0.

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 - On success 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 based on user selection.

Parameters

SecureBits	Pointer to the XilSKey_Puf_Secure which holds the read eFUSE secure bits of PUF.
ReadOption	A u8 variable which has to be provided by user based on this input reading is happened from cache or from efuse array. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from efuse array

Returns

XST_SUCCESS - On success ErrorCode - on Failure

u32 XilSKey_Puf_Debug2 (XilSKey_Puf $*$ *InstancePtr*)

PUF Debug 2 operation.

Parameters

InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

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

Note

Updates the Debug 2 mode result @ InstancePtr->Debug2Data

u32 XilSKey_Puf_Registration (XilSKey_Puf $*$ *InstancePtr*)

PUF Registration/Re-registration.

Parameters

InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

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

Note

Updates the syndrome data @ InstancePtr->SyndromeData

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 User-Configurable PL eFUSE Parameters.
- Requires hardware setup to program PL eFUSE of Zynq or UltraScale.

Functions

- u32 XilSKey_EfusePl_Program (XilSKey_EPl ∗PlInstancePtr)
- u32 XilSKey_EfusePl_ReadStatus (XilSKey_EPl ∗InstancePtr, u32 ∗StatusBits)
- u32 XilSKey_EfusePl_ReadKey (XilSKey_EPl ∗InstancePtr)
- u32 XilSKey_CrcCalculation (u8 ∗Key)

Function Documentation

u32 XilSKey_EfusePl_Program (XilSKey_EPl ∗ *InstancePtr*)

Programs PL eFUSE with input data given through InstancePtr.

When called, this API: Initializes the timer, XADC/xsysmon and JTAG server subsystems. Writes the AES & User Keys if requested. In UltraScale, if requested it also programs the RSA Hash Writes the Control Bits if requested. Returns an error if: In Zynq the reference clock frequency is not in between 20 and 60 MHz. The PL DAP ID is not identified. 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) Temperature and voltage are not within range.

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

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

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

Note

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

u32 XilSKey_EfusePl_ReadKey (XilSKey_EPl ∗ *InstancePtr*)

Reads the PL efuse keys and stores them in the corresponding arrays in instance structure, it initializes the timer, XADC and JTAG server subsystems, if not already done so.
In Zynq - Reads AES key and User keys In Ultrascale - 32 bit and 128 bit User keys and RSA hash But in Ultrascale 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()

Parameters

InstancePtr	Pointer to PL eFUSE instance.

Returns

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

Note

In Zynq updates the global variable ErrorCode with error code(if any).

u32 XilSKey_CrcCalculation (u8 ∗ *Key*)

Calculates CRC value of provided key, this API expects key in string format.

Parameters

Key	is the string contains AES key in hexa decimal 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

This API calculates CRC of AES key for Ultrascale Microblaze's PL eFuse and ZynqMp UltraScale PS eFuse. If length of the string provided is lesser than 64, API appends the string with zeros.

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, programmed AES cannot be read back. The programmed AES key can only be confirmed by doing CRC check of AES key.

Functions

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

Function Documentation

u32 XilSKey_CrcCalculation (u8 ∗ *Key*)

Calculates CRC value of provided key, this API expects key in string format.

Parameters

Key	is the string contains AES key in hexa decimal 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

This API calculates CRC of AES key for Ultrascale Microblaze's PL eFuse and ZynqMp UltraScale PS eFuse. If length of the string provided is lesser than 64, API appends the string with zeros.

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 size 32 which contains AES key in hexa decimal.

Returns

Crc of provided AES key value.

Note

This API calculates CRC of AES key for Ultrascale Microblaze's PL eFuse and ZynqMp Ultrascale's PS eFuse. This API calculates CRC on AES key provided in hexa format. To calculate CRC on the AES key in string format please use XilSKey_CrcCalculation. To call this API one can directly pass array of AES key which exists in an instance. Example for storing key into Buffer: If Key is "123456" buffer should be {0x12 0x34 0x56}

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 User-Configurable BBRAM PL Parameters
- 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. 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. 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. FALSE does not modify the write-protect bits.
XSK_EFUSEPS_ENABLE_RSA_AUTH	Default = FALSE. 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. FALSE does not modify the RSA enable bit.
XSK_EFUSEPS_ENABLE_ROM_128K_CRC	Default = FALSE. TRUE burns the ROM 128K CRC bit. In every successive boot, BootROM calculates 128k CRC. FALSE does not modify the ROM CRC 128K bit.
XSK_EFUSEPS_ENABLE_RSA_KEY_HASH	Default = FALSE. 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. FALSE ignores the provided value.

Macro Name	Description
XSK_EFUSEPS_RSA_KEY_HASH_VALUE	Default = 0000000000000000000000000000000000000000000000000000000000000000 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 TRUE disables DFT JTAG permanently. FALSE will not modify the eFuse PS DFT JTAG disable bit.
XSK_EFUSEPS_DISABLE_DFT_MODE	Default = FALSE 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 If the value is set to TRUE, then the part has to be power-cycled to be reconfigured. FALSE does not set the eFUSE control bit.
XSK_EFUSEPL_DISABLE_KEY_WRITE	Default = FALSE TRUE disables the eFUSE write to FUSE_AES and FUSE_USER blocks. FALSE does not affect the EFUSE bit.

Macro Name	Description
XSK_EFUSEPL_DISABLE_AES_KEY_READ	Default = FALSE TRUE disables the write to FUSE_AES and FUSE_USER key and disables the read of FUSE_AES. FALSE does not affect the eFUSE bit.
XSK_EFUSEPL_DISABLE_USER_KEY_READ	Default = FALSE. TRUE disables the write to FUSE_AES and FUSE_USER key and disables the read of FUSE_USER. FALSE does not affect the eFUSE bit.
XSK_EFUSEPL_DISABLE_FUSE_CNTRL_WRITE	Default = FALSE. TRUE disables the eFUSE write to FUSE_CTRL block. FALSE does not affect the eFUSE bit.
XSK_EFUSEPL_FORCE_USE_AES_ONLY	Default = FALSE. TRUE forces the use of secure boot with eFUSE AES key only. FALSE does not affect the eFUSE bit.
XSK_EFUSEPL_DISABLE_JTAG_CHAIN	Default = FALSE. TRUE permanently disables the Zynq ARM DAP and PL TAP. FALSE does not affect the eFUSE bit.
XSK_EFUSEPL_BBRAM_KEY_DISABLE	Default = FALSE. TRUE forces the eFUSE key to be used if booting Secure Image. 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	(18)
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	(21)	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. LOW writes zero on the MUX select line before PL_eFUSE writing. 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. 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. FALSE ignores the provided values. You cannot write the AES Key and the User Low Key separately.
XSK_EFUSEPL_PROGRAM_USER_HIGH_KEY	Default =FALSE. TRUE burns the User High hash key, given in XSK_EFUSEPL_PROGRAM_USER_HIGH_KEY. FALSE ignores the provided values.
XSK_EFUSEPL_AES_KEY	Default = 0000000000000000000000000000000000000000000000000000000000000000 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. 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 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. 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 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. 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. LOW writes zero on the MUX select line before PL_eFUSE writing. 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. AES key (in HEX) that must be programmed into BBRAM.
XSK_BBRAM_AES_KEY_SIZE_IN_BITS	Default = 256. Size of AES key. Must be 256 bits.

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 Pins Used for PL Master JTAG and HWM Signals
- GPIO Channels

AES Keys and Related Parameters

The following table shows AES key related parameters.

Parameter Name	Description
XSK_BBRAM_PGM_OBFUSCATED_KEY	Default = FALSE By default XSK_BBRAM_PGM_OBFUSCATED_KEY is FALSE, BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY and DPA protection cannot be enabled.
XSK_BBRAM_OBFUSCATED_KEY	Default = b1c276899d71fb4cdd4a0a7905ea46c2e1 1f9574d09c7ea23b70b67de713ccd1 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. **Note** For writing the OBFUSCATED Key, XSK_BBRAM_PGM_OBFUSCATED_KEY should have TRUE value.
XSK_BBRAM_AES_KEY	Default = 0000000000000000524156a63950bcedaf eadcdeabaadee34216615aaaabbaaa 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. **Note** For programming BBRAM with the key, XSK_BBRAM_PGM_OBFUSCATED_KEY should have a 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 By default, the DPA protection will be in disabled state. TRUE will enable DPA protection with provided DPA count and configuration in XSK_BBRAM_DPA_COUNT and XSK_BBRAM_DPA_MODE respectively. DPA protection cannot be enabled if BBRAM is been programmed with an obfuscated key.
XSK_BBRAM_DPA_COUNT	Default = 0 This input is valid only when DPA protection is enabled. Valid range of values are 1 - 255 when DPA protection is enabled else 0.
XSK_BBRAM_DPA_MODE	Default = XSK_BBRAM_INVALID_CONFIGURATIONS 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 Pins Used for PL Master JTAG and HWM 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
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 can 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 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 TRUE will permanently disable the write to FUSE_AES and check CRC for AES key by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_DISABLE_USER_KEY_READ	Default = FALSE TRUE will permanently disable the write to 32 bit FUSE_USER and read of FUSE_USER key by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_DISABLE_SECURE_READ	Default = FALSE TRUE will permanently disable the write to FUSE_Secure block and reading of secure block by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_DISABLE_FUSE_CNTRL_WRITE	Default = FALSE. TRUE will permanently disable the write to FUSE_CNTRL block by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.

Macro Name	Description
XSK_EFUSEPL_DISABLE_RSA_KEY_READ	Default = FALSE. 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. TRUE will permanently disable the write to FUSE_AES block by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_DISABLE_USER_KEY_WRITE	Default = FALSE. TRUE will permanently disable the write to FUSE_USER block by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_DISABLE_SECURE_WRITE	Default = FALSE. TRUE will permanently disable the write to FUSE_SECURE block by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_DISABLE_RSA_HASH_WRITE	Default = FALSE. TRUE will permanently disable the write to FUSE_RSA authentication key by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_DISABLE_128BIT_USER_KEY _WRITE	Default = FALSE. TRUE will permanently disable the write to 128 bit FUSE_USER by programming control bit of FUSE. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPL_ALLOW_ENCRYPTED_ONLY	Default = FALSE. TRUE will permanently allow encrypted bitstream only. FALSE will not modify this Secure bit of eFuse.
XSK_EFUSEPL_FORCE_USE_FUSE_AES_ONLY	Default = FALSE. 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. TRUE will enable RSA authentication of bitstream FALSE will not modify this secure bit of eFuse.

Macro Name	Description
XSK_EFUSEPL_DISABLE_JTAG_CHAIN	Default = FALSE. TRUE will disable JTAG permanently. FALSE will not modify this secure bit of eFuse.
XSK_EFUSEPL_DISABLE_TEST_ACCESS	Default = FALSE. TRUE will disables Xilinx test access. FALSE will not modify this secure bit of eFuse.
XSK_EFUSEPL_DISABLE_AES_DECRYPTOR	Default = FALSE. TRUE will disables decoder completely. FALSE will not modify this secure bit of eFuse.

Modules

- GPIO Pins Used for PL Master JTAG Signal
- GPIO Channels
- AES Keys and Related Parameters

GPIO Pins Used for PL Master JTAG Signal

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_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 can 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.

AES Keys and Related Parameters

The following table shows AES key related parameters.

Parameter Name	Description
XSK_EFUSEPL_PROGRAM_AES_KEY_ULTRA	Default = FALSE TRUE will burn the AES key given in XSK_EFUSEPL_AES_KEY. FALSE will ignore the values given.
XSK_EFUSEPL_PROGRAM_USER_KEY_ULTRA	Default = FALSE TRUE will burn 32 bit User key given in XSK_EFUSEPL_USER_KEY. FALSE will ignore the values given.
XSK_EFUSEPL_PROGRAM_RSA_HASH_ULTRA	Default = FALSE TRUE will burn RSA hash given in XSK_EFUSEPL_RSA_KEY_HASH_VALUE. FALSE will ignore the values given.
XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT	Default = FALSE TRUE will burn 128 bit User key given in XSK_EFUSEPL_USER_KEY_128BIT_0, XSK_EFUSEPL_USER_KEY_128BIT_1, XSK_EFUSEPL_USER_KEY_128BIT_2, XSK_EFUSEPL_USER_KEY_128BIT_3 FALSE will ignore the values given.

Parameter Name	Description
XSK_EFUSEPL_CHECK_AES_KEY_ULTRA	Default = FALSE 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_EPI instance parameter AESKeyMatched with either TRUE or FALSE. FALSE CRC check of FUSE_AES will not be performed.
XSK_EFUSEPL_READ_USER_KEY_ULTRA	Default = FALSE TRUE will read 32 bit FUSE_USER from UltraScale eFUSE and updates in XilSKey_EPI instance parameter UserKeyReadback. FALSE 32-bit FUSE_USER key read will not be performed.
XSK_EFUSEPL_READ_RSA_HASH_ULTRA	Default = FALSE TRUE will read FUSE_USER from UltraScale eFUSE and updates in XilSKey_EPI instance parameter RSAHashReadback. FALSE FUSE_RSA_HASH read will not be performed.
XSK_EFUSEPL_READ_USER_KEY128_BIT	Default = FALSE TRUE will read 128 bit USER key from UltraScale eFUSE and updates in XilSKey_EPI instance parameter User128BitReadBack. FALSE 128 bit USER key read will not be performed.
XSK_EFUSEPL_AES_KEY	Default = 00000000000000000000000000000000 0000000000000000000000000000000 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 PPK(Primary Public Key) hash 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 that for writing the AES Key, XSK_EFUSEPL_PROGRAM_AES_KEY_ULTRA should have TRUE value.

Parameter Name	Description
XSK_EFUSEPL_USER_KEY	Default = 00000000 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. Note that, for writing the User Key, XSK_EFUSEPL_PROGRAM_USER_KEY_ULTRA should have TRUE value.
XSK_EFUSEPL_RSA_KEY_HASH_VALUE	Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 he 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 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. Note that, for writing the RSA hash, XSK_EFUSEPL_PROGRAM_RSA_HASH_ULTRA should have TRUE value.
XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY	Default = 0x621C42AA 0x621C42AA is hexadecimal CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value has to be updated in place of 0x621C42AA. 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 or reverse polynomial 0x82F63B78.

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 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 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 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 TRUE will permanently disable the BBRAM key. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_ERR_DISABLE	Default = FALSE 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 TRUE will permanently disable JTAG controller. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_DFT_DISABLE	Default = FALSE TRUE will permanently disable DFT boot mode. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_PROG_GATE_DISABLE	Default = FALSE TRUE will permanently disable PROG_GATE feature in PPD. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_SECURE_LOCK	Default = FALSE 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 TRUE will permanently enable RSA authentication during boot. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_PPK0_WR_LOCK	Default = FALSE TRUE will permanently disable writing to PPK0 efuses. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_PPK0_INVLD	Default = FALSE TRUE will permanently revoke PPK0. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_PPK1_WR_LOCK	Default = FALSE TRUE will permanently disable writing PPK1 efuses. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_PPK1_INVLD	Default = FALSE TRUE will permanently revoke PPK1. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_0	Default = FALSE 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 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 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 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 TRUE will permanently disable writing to USER_4 efuses. FALSE will not modify this control bit of eFuse.

Macro Name	Description
XSK_EFUSEPS_USER_WRLK_5	Default = FALSE TRUE will permanently disable writing to USER_5 efuses. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_6	Default = FALSE 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 TRUE will permanently disable writing to USER_7 efuses. FALSE will not modify this control bit of eFuse.
XSK_EFUSEPS_LBIST_EN	Default = FALSE 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 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 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 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 = TRUE TRUE will burn the AES key provided in XSK_EFUSEPS_AES_KEY. FALSE will ignore the key provide XSK_EFUSEPS_AES_KEY.
XSK_EFUSEPS_AES_KEY	Default = 00000000000000000000000000000000 0000000000000000000000000000000 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. **Note** For writing the AES Key, XSK_EFUSEPS_WRITE_AES_KEY should have TRUE value.

User Keys and Related Parameters

Single bit programming is allowed for all the USER fuses.
If user requests to revert already programmed bit. Library throws an error. If user fuses is non-zero also 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 = TRUE 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 = TRUE 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 = TRUE TRUE will burn User2 Fuse provided in XSK_EFUSEPS_USER2_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER2_FUSES

Parameter Name	Description
XSK_EFUSEPS_WRITE_USER3_FUSE	Default = TRUE 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 = TRUE 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 = TRUE 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 = TRUE 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 = TRUE TRUE will burn User7 Fuse provided in XSK_EFUSEPS_USER7_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER7_FUSES
XSK_EFUSEPS_USER0_FUSES	Default = 00000000 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. **Note** For writing the User0 Fuse, XSK_EFUSEPS_WRITE_USER0_FUSE should have TRUE value

Parameter Name	Description
XSK_EFUSEPS_USER1_FUSES	Default = 00000000 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. **Note** For writing the User1 Fuse, XSK_EFUSEPS_WRITE_USER1_FUSE should have TRUE value
XSK_EFUSEPS_USER2_FUSES	Default = 00000000 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. **Note** For writing the User2 Fuse, XSK_EFUSEPS_WRITE_USER2_FUSE should have TRUE value
XSK_EFUSEPS_USER3_FUSES	Default = 00000000 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. **Note** For writing the User3 Fuse, XSK_EFUSEPS_WRITE_USER3_FUSE should have TRUE value

Parameter Name	Description
XSK_EFUSEPS_USER4_FUSES	Default = 00000000 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. **Note** For writing the User4 Fuse, XSK_EFUSEPS_WRITE_USER4_FUSE should have TRUE value
XSK_EFUSEPS_USER5_FUSES	Default = 00000000 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. **Note** For writing the User5 Fuse, XSK_EFUSEPS_WRITE_USER5_FUSE should have TRUE value
XSK_EFUSEPS_USER6_FUSES	Default = 00000000 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. **Note** For writing the User6 Fuse, XSK_EFUSEPS_WRITE_USER6_FUSE should have TRUE value

Parameter Name	Description
XSK_EFUSEPS_USER7_FUSES	Default = 00000000 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. **Note** 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 = TRUE 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.
XSK_EFUSEPS_PPK0_IS_SHA3	Default = TRUE 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.

Parameter Name	Description
XSK_EFUSEPS_PPK0_HASH	Default = 0000000000000000000000000000000000 00000000000000000000000000000000000000000 000000000000000000000 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 = TRUE 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 = FALSE 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 = 0000000000000000000000000000000000000000000000000000000000000000 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 = TRUE TRUE will burn SPKID provided in XSK_EFUSEPS_SPK_ID. FALSE will ignore the hash provided in XSK_EFUSEPS_SPK_ID.

Parameter Name	Description
XSK_EFUSEPS_SPK_ID	Default = 00000000 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. **Note** 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 AES key (in HEX) that must be programmed into BBRAM.
XSK_ZYNQMP_BBRAMPS_AES_KEY_LEN_IN_BYTES	Default = 32. Length of AES key in bytes.
XSK_ZYNQMP_BBRAMPS_AES_KEY_LEN_IN_BITS	Default = 256. Length of AES key in bits.
XSK_ZYNQMP_BBRAMPS_AES_KEY_STR_LEN	Default = 64. 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 TRUE will display syndrome data on UART com port FALSE will display any data on UART com port.
XSK_PUF_PROGRAM_EFUSE	Default = FALSE TRUE will program the generated syndrome data, CHash and Auxilary values, Black key. FALSE will not program data into eFUSE.
XSK_PUF_IF_CONTRACT_MANUFATURER	Default = FALSE This should be enabled when application is hand over to contract manufacturer. TRUE will allow only authenticated application. FALSE authentication is not mandatory.
XSK_PUF_REG_MODE	Default = XSK_PUF_MODE4K 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 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 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. FALSE will not program any PUF secure bits.
XSK_PUF_SYN_INVALID	Default = FALSE TRUE will permanently invalidate the already programmed syndrome data. FALSE will not modify anything
XSK_PUF_SYN_WRLK	Default = FALSE TRUE will permanently disable programming syndrome data into eFUSE. FALSE will not modify anything.
XSK_PUF_REGISTER_DISABLE	Default = FALSE TRUE permanently does not allow PUF syndrome data registration. FALSE will not modify anything.
XSK_PUF_RESERVED	Default = FALSE TRUE programs this reserved eFUSE bit. FALSE will not modify anything.
XSK_PUF_AES_KEY	Default = 0000000000000000000000000000000000 000000000000000000000000000000 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_IV	Default = 000000000000000000000000 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. 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.

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 br>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 PL
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 60 MHz

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 Utils Error Codes. 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_PROGRAMMING_TBIT_PATTERN 0x16
Error in programming TBITS.

XSK_EFUSEPS_ERROR_BEFORE_PROGRAMMING 0x0080
Error occurred before programming.

XSK_EFUSEPS_ERROR_PROGRAMMING 0x00A0
Error in programming eFUSE.

XSK_EFUSEPS_ERROR_READ 0x00B0
Error in reading.

XSK_EFUSEPS_ERROR_PS_STRUCT_NULL XSKEfuse_Write/Read()common error codes. 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_CACHE_LOAD 0xB000
Error in re-loading CACHE.

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_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_CMPLTD_EFUSE_PRGRM_WITH_ERR 0x10000
eFUSE programming is completed with temp and vol read errors.

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.

Zynq UltraScale+ MPSoC BBRAM PS Error Codes

XSK_ZYNQMP_BBRAMPS_ERROR_NONE 0
No error.

XSK_ZYNQMP_BBRAMPS_ERROR_IN_PRGRMG_ENABLE 0x01
If this error is occurred programming is not possible.

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.

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.

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