UG1191 November 18, 2015

LibXil SKey for Zynq-7000 AP SoC Devices (v4.0)

Overview

The LibXil SKey library provides a programming mechanism for user-defined eFUSE bits and for programming the KEY into battery-backed RAM (BBRAM) of Zynq® SoC, provides programming mechanisms for eFUSE bits of UltraScale™ devices. The library also provides programming mechanisms for eFUSE bits and BBRAM key of the Zynq® UltraScale+™ MPSoC devices.

In Zynq:

- 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.
- BBRAM holds the AES key.

In UltraScale™:

PL eFuse holds the AES key, the user key, RSA key hash and some of the feature bits.

In Zynq UltraScale+ MPSoC:

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

The following user application (example) files are provided:

- `xilskey_bbram_example.c` file lets you write the key to BBRAM of Zynq.
- `xilskey_efuse_example.c` file lets you write into the PS/PL eFUSE of Zynq and Ultrascale.
- `xilskey_efuseps_zynqmp_example.c` file lets you write into eFUSE PS of Zynq UltraScale+ MPSoC.
- `xilskey_bbramps_zynqmp_example.c` file lets you write BBRAM key of Zynq UltraScale+ MPSoC.

Caution! Make sure to 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.

SDK Project File and Folders

Table 1 lists the eFUSE application SDK project files, folders, and macros.

Table 1: **eFUSE SDK Application Project Files**

File or Folder	Description
`xilskey_efuse_example.c`	Contains the main application code. Does the PS/PL structure initialization and writes/reads the PS/PL eFUSE based on the user settings provided in the `xilskey_input.h`.
`xilskey_input.h`	Contains 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. In this file, specify the 256 bit key to be programmed into BBRAM. In this file, specify the AES(256 bit) key, User (32 bit) key and RSA key hash(384 bit) key to be programmed into eFuse of UltraScale.
`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`	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 done, in that order.
`xilskey_efuseps_zynqmp_example.c`	Contains the example code to program the PS eFUSE and read back of eFUSE bits from the cache.
`xilskey_efuseps_zynqmp_input.h`	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`	Contains the example code to program and verify BBRAM key. Default is zero. You can modify this key on top of the file.

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` 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`. See Table 2.

Table 2: **User Configurable PS eFUSE Parameters**

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.

Table 2: **User Configurable PS eFUSE Parameters *(Cont'd)***

Macro Name	Description
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.
XSK_EFUSEPS_RSA_KEY_HASH_VALUE	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.

User-Configurable PL eFUSE Parameters

Table 3 shows the user-configurable PL eFUSE parameters.

Table 3: **User-Configurable PL eFUSE Parameters**

Macro Name	Definition
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.
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.

Table 3: **User-Configurable PL eFUSE Parameters *(Cont'd)***

Macro Name	Definition
`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 sets the Zynq ARM DAP controller in bypass mode. 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.

MIO Pins for PL JTAG Operations

You can change the listed pins at your discretion. See Table 4.

Table 4: **MIO Pins for PL JTAG**

Pin Name	Pin Number[1]
`XSK_EFUSEPL_MIO_JTAG_TDI`	(17)
`XSK_EFUSEPL_MIO_JTAG_TDO`	(18)
`XSK_EFUSEPL_MIO_JTAG_TCK`	(19)
`XSK_EFUSEPL_MIO_JTAG_TMS`	(20)

Notes:

1. The pin numbers listed are examples. You must assign appropriate pin numbers per your hardware design.

MUX

The following subsections describe MUX usage, the MUX selection pin, and the MUX parameter.

MUX Usage Requirements

To write the PL eFUSE 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 writing, you must define the following:

- The MIOs used for JTAG operations (TCK,TMS,TDI,TDO), shown in Table 4.
- The MIO used for the MUX Select Line, shown in Table 5.
- The Value on the MUX Select line, shown in Table 6, to select JTAG for PL eFUSE writing.

Figure 1 illustrates correct MUX usage.

X-Ref Target - Figure 1

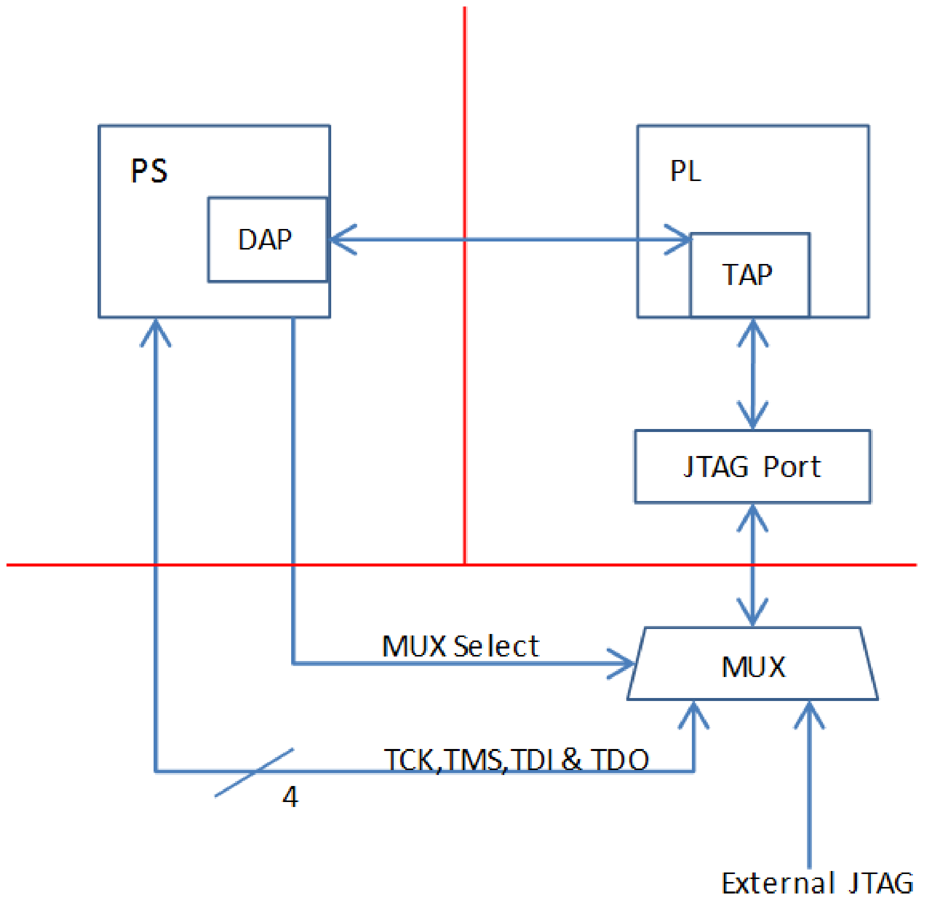

Figure 1: **MUX Usage**

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

Selection Pin

Table 5 shows the MUX selection pin.

Table 5: **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

Table 6 shows the MUX parameter.

Table 6: **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

Table 7 shows the AES and user key parameters.

Table 7: **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.
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.

User-Configurable BBRAM Parameters

Table 8 shows the user-configurable BBRAM parameters.

Table 8: **User-Configurable BBRAM Parameters**

Parameter	Default Value	Description
XSK_BBRAM_FORCE_PCYCLE_RECONFIG	FALSE	If TRUE, part has to be power cycled to be able to be reconfigured.
XSK_BBRAM_DISABLE_JTAG_CHAIN	FALSE	If TRUE, permanently sets the Zynq ARM DAP controller in bypass mode.

MIO Pins Used for PL JTAG Signals

The MIO pins shown in Table 9 are used for PL JTAG signals. These can be changed depending on your hardware

Table 9: **MIO Pins Used for PL JTAG Signals**

JTAG Signal	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

MUX Parameter

Table 10 shows the MUX parameter.

Table 10: **MUX Parameter**

Parameter	Default Value	Description
XSK_BBRAM_MIO_MUX_SEL_DEFAULT_VAL	LOW	Default value to enable the PL JTAG.

AES Key and Related Parameters

Table 11 shows the AES key and related parameters.

Table 11: **AES Key and Related Parameters**

Parameter Name	Default Value	Description
XSK_BBRAM_AES_KEY	XX	AES key (in HEX) that must be programmed into BBRAM.
XSK_BBRAM_AES_KEY_SIZE_IN_BITS	256	Size of AES key. Must be 256 bits.

User-Configurable eFuse Parameters of UltraScale

Table 12 shows the user-configurable eFuse parameters.

Table 12: **User-Configurable eFuse Parameters**

Parameter	Default Value	Description
`XSK_EFUSEPL_DISABLE_AES_KEY_READ`	`FALSE`	TRUE permanently disables AES CRC check and programming of the AES key. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_USER_KEY_READ`	`FALSE`	TRUE permanently disables reading and programming of User key. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_SECURE_READ`	`FALSE`	TRUE permanently disables reading and programming of Secure bits. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_FUSE_CNTRL_WRITE`	`FALSE`	Default = `FALSE`. TRUE permanently disables programming of Control bits `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_RSA_KEY_READ`	`FALSE`	Default = `FALSE`. TRUE permanently disables reading and programming of RSA hash key. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_KEY_WRITE`	`FALSE`	Default = `FALSE`. TRUE permanently disables programming of AES key `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_USER_KEY_WRITE`	`FALSE`	Default = `FALSE`. TRUE permanently disables programming of User key. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_SECURE_WRITE`	`FALSE`	Default = `FALSE`. TRUE permanently disables programming of Secure bits `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_RSA_HASH_WRITE`	`FALSE`	Default = `FALSE`. TRUE permanently disables programming of RSA key Hash. `FALSE` does not affect the eFUSE bit.
Secure bits		
`XSK_EFUSEPL_ALLOW_ENCRYPTED_ONLY`	`FALSE`	Default = `FALSE`. TRUE permanently forces to use only encrypted bitstreams. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_FORCE_USE_FUSE_AES_ONLY`	`FALSE`	Default = `FALSE`. TRUE permanently forces to use secure boot with eFUSE key only. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_ENABLE_RSA_AUTH`	`FALSE`	Default = **`FALSE`**. TRUE will permanently enable RSA authentication of bitstream. `FALSE` does not affect the eFUSE bit.
`XSK_EFUSEPL_DISABLE_JTAG_CHAIN`	`FALSE`	Default = `FALSE`. TRUE permanently sets the Ultrscale device in bypass mode. `FALSE` does not affect the eFUSE bit.

Table 12: **User-Configurable eFuse Parameters *(Cont'd)***

Parameter	Default Value	Description
XSK_EFUSEPL_DISABLE_TEST_ACCESS	FALSE	Default = FALSE. TRUE permanently disables test access for UltraScale. FALSE does not affect the eFUSE bit.
XSK_EFUSEPL_DISABLE_DECODER	FALSE	Default = FALSE. TRUE permanently disables decoder. FALSE does not affect the eFUSE bit.

GPIO Pins Used for PL Master JTAG Signal

The following GPIO pins are used for PL master JTAG signals. These can be changed depending on your hardware. Table 13 shows the MIO pins used for PL JTAG signals.

Table 13: **MIO Pins Used for PL JTAG Signals**

Master JTAG Signal	PIN Number (Default)
XSK_EFUSEPL_AXI_GPIO_JTAG_TDO	0
XSK_EFUSEPL_AXI_GPIO_JTAG_TDI	0
XSK_EFUSEPL_AXI_GPIO_JTAG_TMS	1
XSK_EFUSEPL_AXI_GPIO_JTAG_TCK	2

GPIO Channels

Table 14 shows theMUX parameter.

Table 14: **MUX Parameter**

Parameter	Channel Number (Default)
XSK_EFUSEPL_GPIO_INPUT_CH	2
XSK_EFUSEPL_GPIO_OUTPUT_CH	1

Note: GPIO input (TDO) and output (TDI, TMS and TCK) signals can belongs to same channel *or* inputs in one channel and outputs in the other channel. But some inputs in one channel and others in different channels are not accepted in this library.

Keys and Related Parameters

Table 15 shows AES key and related parameters.

Table 15: **AES Key and Related Parameters**

Parameter Name	Default Value	Description
XSK_EFUSEPL_PROGRAM_AES_KEY	FALSE	Default = FALSE If TRUE will program the AES key provided in the macro XSK_EFUSEPL_AES_KEY into eFUSE. FALSE ignores the provided values in XSK_EFUSEPL_AES_KEY
XSK_EFUSEPL_AES_KEY	000000000000 000000000000 000000000000 000000000000 000000000000 0000	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 the AES key. To write AES key, XSK_EFUSEPL_PROGRAM _AES_ must have a value of TRUE.
XSK_EFUSEPL_CHECK_AES_KEY_CRC	FALSE	Default = FALSE If TRUE will perform CRC check of AES key with provided CRC of estimated AES key in macro XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY FALSE ignores the provided values
XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY	0x621C42AA	Default is 0x621C42AA. This is the 32 bit hexadecimal CRC value of AES key. 0x621C42AA is CRC of AES key all zeros. In this place have to provide CRC of AES key. The result will be saved in PL instance parameter AESKeyMatched

Table 16 shows user key and related parameters.

Table 16: **User Key and Related Parameters**

Parameter Name	Default Value	Description
XSK_EFUSEPL_PROGRAM_USER_KEY	FALSE	Default = FALSE If TRUE will program the User key provided in the macro XSK_EFUSEPL_USER_KEY into eFUSE. FALSE ignores the provided values in XSK_EFUSEPL_USER_KEY
XSK_EFUSEPL_USER_KEY	00000000	This value converted to hex buffer and written into the PL eFUSE array when write API is used. This value should be the User Key, given in string format. It must be 8 characters long. Valid characters are 0-9, a-f, A-F. Any other character is considered an invalid string and will not burn User Key. To write **USER** Key, XSK_EFUSEPL_PROGRAM_USER_KEY_ must have a value of TRUE.
XSK_EFUSEPL_READ_USER_KEY	FALSE	Default = Read. TRUE will read the user key of eFUSE and store it in PL instance. **FALSE** will not read the user key.

Table 17 shows the RSA hash key and related parameters.

Table 17: **RSA Hash Key and Related Parameters**

Parameter Name	Default Value	Description

`XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH`	`FALSE`	Default = `FALSE`
		If `TRUE` will program the RSA key hash provided in the macro `XSK_EFUSEPL_RSA_KEY_HASH_VALUE` into eFUSE.
		`FALSE` ignores the provided values in `XSK_EFUSEPL_RSA_KEY_HASH_VALUE`

`XSK_EFUSEPL_RSA_KEY_HASH_VALUE`	`00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000000 00000000`	Default = `000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000` This value converted to hex buffer and written into the PL eFUSE array when write API is used. This value should be the RSA Key hash, given in string format. It must be 96 characters long. Valid characters are 0-9, a-f, A-F. Any other character is considered an invalid string and will not burn RSA Key hash.To write RSA key hash, `XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH` must have a value of TRUE.
`XSK_EFUSEPL_READ_RSA_KEY_HASH`	`FALSE`	Default = `Read`. `TRUE` will reads RSA key hash of eFUSE and stores in PL instance. `FALSE` will not read the user key.

User-Configurable eFuse PS Parameters of Zynq UltraScale+ MPSoC

Table 18 shows the user-configurable eFuse parameters of Zynq UltraScale+ MPSoC.

Table 18: **User-Configurable eFuse PS Parameters of Zynq UltraScale+ MPSoC**

Parameter	Default Value	Description
`XSK_EFUSEPS_AES_RD_LOCK`	`FALSE`	TRUE permanently disables the CRC check of FUSE_AES. FALSE does not modify this control bit of eFuse.
`XSK_EFUSEPS_AES_WR_LOCK`	`FALSE`	TRUE permanently disables the writing to FUSE_AES block. FALSE does not modify this control bit of eFuse.
`XSK_EFUSEPs_FORCE_USE_AES_ONLY`	`FALSE`	TRUE permanently disables encrypted booting only using the Fuse key. FALSE does not modify this control bit of eFuse.
`XSK_EFUSEPS_BBRAM_DISABLE`	`FALSE`	TRUE permanently disables the BBRAM key. FALSE does not modify this control bit of eFuse.
`XSK_EFUSEPS_ERR_OUTOF_PMU_DISABLE`	`FALSE`	TRUE permanently disables the error output from the PMU. FALSE does not modify this control bit of eFuse.

Table 18: **User-Configurable eFuse PS Parameters of Zynq UltraScale+ MPSoC** ***(Cont'd)***

Parameter	Default Value	Description
XSK_EFUSEPS_JTAG_DISABLE	FALSE	TRUE permanently disables JTAG controller. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_DFT_DISABLE	FALSE	TRUE permanently disables DFT boot mode. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_PROG_GATE_0_DISABLE	FALSE	TRUE permanently disables PROG_GATE 0 feature in PPD. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_PROG_GATE_1_DISABLE	FALSE	TRUE permanently disables PROG_GATE 1 feature in PPD. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_PROG_GATE_2_DISABLE	FALSE	TRUE permanently disables PROG_GATE 2 feature in PPD. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_SECURE_LOCK	FALSE	TRUE permanently disables reboot into JTAG mode when doing a secure lockdown. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_RSA_ENABLE	FALSE	TRUE permanently disables RSA authentication during boot. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_PPK0_WR_LOCK	FALSE	TRUE permanently disables writing to PPK0 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_PPK0_REVOKE	FALSE	TRUE permanently revokes PPK0. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_PPK1_WR_LOCK	FALSE	TRUE permanently disables writing PPK1 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_PPK1_REVOKE	FALSE	TRUE permanently revokes PPK1. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_0	FALSE	TRUE permanently disables writing to USER_0 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_1	FALSE	TRUE permanently disables writing to USER_1 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_2	FALSE	TRUE permanently disables writing to USER_2 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_3	FALSE	TRUE permanently disables writing to USER_3 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_4	FALSE	TRUE permanently disables writing to USER_4 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_5	FALSE	TRUE permanently disables writing to USER_5 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_6	FALSE	TRUE permanently disables writing to USER_6 efuses. FALSE does not modify this control bit of eFuse.
XSK_EFUSEPS_USER_WRLK_7	FALSE	TRUE permanently disables writing to USER_7 efuses. FALSE does not modify this control bit of eFuse.

Keys and Related Parameters

Table 19 shows AES key and related parameters.

Table 19: **AES Key and Related Parameters**

Parameter Name	Default Value	Description
XSK_EFUSEPS_WRITE_AES_KEY	FALSE	Default = FALSE If TRUE, programs the AES key provided in the macro XSK_EFUSEPS_AES_KEY into eFUSE. FALSE ignores the provided values in XSK_EFUSEPL_AES_KEY
XSK_EFUSEPS_AES_KEY	000000000000 000000000000 000000000000 000000000000 000000000000 0000	Default = 0000000000000000000000000000000000000000000000000000000000000000 This value converted to hex buffer and written into the PS 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 the AES key. To write AES key, XSK_EFUSEPS_WRITE_AES_KEY must have a value of TRUE

Table 20 shows user key and related parameters.

Table 20: **User Key and Related Parameters**

Parameter Name	Default Value	Description
XSK_EFUSEPS_WRITE_USER_KEY	FALSE	Default = FALSE If TRUE, programs the USER key provided in the macro XSK_EFUSEPS_USER_KEY into eFUSE. FALSE ignores the provided values in XSK_EFUSEPS_USER_KEY
XSK_EFUSEPS_USER_KEY	000000000000 000000000000 000000000000 000000000000 000000000000 0000	Default = 0000000000000000000000000000000000000000000000000000000000000000 This value converted to hex buffer and written into the PS eFUSE array when write API is used. This value should be the USER 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 the USER key. To write USER key, XSK_EFUSEPS_WRITE_USER_KEY must have a value of TRUE

Table 21 shows the PPK0 hash key and related parameters.

Table 21: **PPK0 Hash Key and Related Parameters**

Parameter Name	Default Value	Description

`XSK_EFUSEPS_WRITE_PPK0_HASH`	`FALSE`	Default = `FALSE`
		If `TRUE,` programs the PPK0 hash provided into the macro `XSK_EFUSEPS_PPK0_HASH` into eFUSE.
		`FALSE` ignores the provided values in `XSK_EFUSEPS_PPK0_HASH`

`XSK_EFUSEPS_PPK0_IS_SHA3`	`TRUE`	Default = `TRUE` If `TRUE`, `XSK_EFUSEPS_PPK0_HASH` is considered as SHA3 hash and hexa-decimal string should be of length 96 characters If `FALSE`, `XSK_EFUSEPS_PPK0_HASH` is considered as SHA2 hash and hexa-decimal string should be of length 64 characters

`XSK_EFUSEPS_PPK0_HASH`	`00000000000`
	`00000000000`
	`00000000000`
	`00000000000`
	`00000000000`
	`00000000000`
	`00000000000`
	`00000000000`
	`00000000`
		Default =
		`0000000000000000000000000000000000000000000`
		`0000000000000000000000000000000000000000000`
		`000000000`
		This value is converted to hex buffer and is written into the PS eFUSE array when write API is used. This value should be the USER Key, given in string format. It must be 96/64 characters long. Valid characters are 0-9, a-f, A-F. Any other character is considered an invalid string and will not burn the PPK0 hash. To write PPK0 hash, `XSK_EFUSEPS_WRITE_PPK0_HASH` must have a value of `TRUE`.
		By default PPK0 hash is of length 96 characters length, as `XSK_EFUSEPS_PPK0_IS_SHA3` is `TRUE` One can also program PPK0 hash with SHA2 hash, for programming PPK0 hash with SHA2 hash, `XSK_EFUSEPS_PPK0_IS_SHA3` should be `FALSE` and hash in `XSK_EFUSEPS_PPK0_HASH` should be length of 64 characters long.

Table 22 shows the PPK1 hash key and related parameters.

Table 22: **PPK1 Hash Key and Related Parameters**

Parameter Name	Default Value	Description

`XSK_EFUSEPS_WRITE_PPK1_HASH`	`FALSE`	Default = `FALSE`
		If `TRUE,` programs the PPK1 hash provided into the macro `XSK_EFUSEPS_PPK1_HASH` into eFUSE.
		`FALSE` ignores the provided values in `XSK_EFUSEPS_PPK1_HASH`

Table 22: **PPK1 Hash Key and Related Parameters *(Cont'd)***

Parameter Name	Default Value	Description
XSK_EFUSEPS_PPK1_IS_SHA3	FALSE	Default = FALSE If FALSE, XSK_EFUSEPS_PPK1_HASH is considered as SHA2 hash and hexa-decimal string should be of length 64 characters If TRUE, XSK_EFUSEPS_PPK1_HASH is considered as SHA3 hash and hexa-decimal string should be of length 96 characters
XSK_EFUSEPS_PPK1_HASH	00000000000 00000000000 00000000000 00000000000 00000000000 000000000	Default = 0000000000000000000000000000000000000000000000000000000000000000 This value converted to hex buffer and written into the PS eFUSE array when write API is used. This value should be the USER Key, given in string format. It must be 96/64 characters long. Valid characters are 0-9, a-f, A-F. Any other character is considered an invalid string and will not burn the PPK1 hash. To write PPK1 hash, XSK_EFUSEPS_WRITE_PPK1_HASH must have a value of TRUE. By default PPK1 hash is of length 64 characters length, as XSK_EFUSEPS_PPK1_IS_SHA3 is FALSE One can also program PPK1 hash with SHA3 hash, for programming PPK1 hash with SHA3 hash, XSK_EFUSEPS_PPK1_IS_SHA3 should be TRUE and hash in XSK_EFUSEPS_PPK1_HASH macro should be length of 96 characters long.

Table 23 shows the SPK ID and related parameters.

Table 23: **SPK ID and Related Parameters**

Parameter Name	Default Value	Description
XSK_EFUSEPS_WRITE_SPKID	FALSE	Default = FALSE If TRUE, programs the SPK ID provided in the macro XSK_EFUSEPS_SPK_ID into eFUSE. FALSE ignores the provided values in XSK_EFUSEPS_SPK_ID
XSK_EFUSEPS_SPK_ID	00000000	Default = 00000000 This value converted to hex buffer and written into the PS eFUSE array when write API is used. This value should be the SPK ID, given in string format. It must be 8 characters long. Valid characters are 0-9, a-f, A-F. Any other character is considered an invalid string and will not burn the SPK ID. To write SPK ID, XSK_EFUSEPS_WRITE_SPKID must have a value of TRUE

Table 24 shows the JTAG user code and related parameters.

Table 24: **JTAG User Code and Related Parameters**

Parameter Name	Default Value	Description
XSK_EFUSEPS_WRITE_JTAG_USERCODE	FALSE	Default = FALSE If TRUE, programs the JTAG USER CODE provided in the macro XSK_EFUSEPS_JTAG_USERCODE into eFUSE. FALSE ignores the values provided in XSK_EFUSEPS_JTAG_USERCODE
XSK_EFUSEPS_JTAG_USERCODE	00000000	This value converted to hex buffer and written into the PS eFUSE array when write API is used. This value should be the JTAG User code, given in string format. It must be 8 characters long. Valid characters are 0-9, a-f, A-F. Any other character is considered an invalid string and will not burn the JTAG USer code. To write JTAG user code, XSK_EFUSEPS_WRITE_JTAG_USERCODE must have a value of TRUE

Error Codes

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

PL eFUSE Error Codes

Table 25 shows the PL eFUSE error codes.PS eFUSE Error Codes

Table 25: **PL eFUSE Error Codes**

Error Code	Value	Description
XSK_EFUSEPL_ERROR_NONE	0	No error
EFUSE Read Error Codes		
XSK_EFUSEPL_ERROR_ROW_NOT_ZERO	0x10	Row is not zero
XSK_EFUSEPL_ERROR_READ_ROW_OUT_OF_RANGE	0x11	Row is out of range
XSK_EFUSEPL_ERROR_READ_MARGIN_OUT_OF_RANGE	0x12	Margin is out of range
XSK_EFUSEPL_ERROR_READ_BUFFER_NULL	0x13	No buffer
XSK_EFUSEPL_ERROR_READ_BIT_VALUE_NOT_SET	0x14	Bit not set
XSK_EFUSEPL_ERROR_READ_BIT_OUT_OF_RANGE	0x15	Bit is out of range
XSK_EFUSEPL_ERROR_READ_TEMPERATURE_OUT_OF_RANGE	0x16	Temperature obtained from XADC is out of range

Table 25: **PL eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
XSK_EFUSEPL_ERROR_READ_VCCAUX_VOLTAGE_OUT_OF_RANGE	0x17	VCCAUX obtained from XADC is out of range
XSK_EFUSEPL_ERROR_READ_VCCINT_VOLTAGE_OUT_OF_RANGE	PL	VCCINT obtained from XADC is out of range
EFUSE Write Error Codes		
XSK_EFUSEPL_ERROR_WRITE_ROW_OUT_OF_RANGE	0x19	Row is out of range
XSK_EFUSEPL_ERROR_WRITE_BIT_OUT_OF_RANGE	0x1A	Bit is out of range
XSK_EFUSEPL_ERROR_WRITE_TEMPERATURE_OUT_OF_RANGE	0x1B	Temperature obtained from XADC is out of range
XSK_EFUSEPL_ERROR_WRITE_VCCAUX_VOLTAGE_OUT_OF_RANGE	0x1C	VCCAUX obtained from XADC is out of range
XSK_EFUSEPL_ERROR_WRITE_VCCINT_VOLTAGE_OUT_OF_RANGE	0x1D	VCCINT obtained from XADC is out of range
XSK_EFUSEPL_ERROR_IN_PROGRAMMING_ROW	0x29	Error occured when programming row of eFUSE
XSK_EFUSEPL_ERROR_PRGRMG_ROWS_NOT_EMPTY	0x2A	Error when tried to program non Zero rows of eFUSE.
EFUSE CNTRL Error Codes		
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
EFUSE KEY Error Codes		
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
XSKEfusepl_Program_Efuse() Error Codes		
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

Table 25: **PL eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
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_RSA_HASH	0x8000	Error in programming RSA hash
XSK_EFUSEPL_ERROR_PRGRMG_USER_KEY	0x4000	Error in programming user key

Table 26 shows the PS eFUSE error codes. These error codes are applicable for both Zynq and Zynq UltraScale+ MPSoC eFUSE PS.

Table 26: **PS eFUSE Error Codes**

Error Code	Value	Description
XSK_EFUSEPL_ERROR_NONE	0	No error
EFUSE Read Error Codes		
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_VCCAUX_VOLTAGE_OUT_OF_RANGE	0x03	VCCAUX obtained from XADC is out of range
XSK_EFUSEPS_ERROR_READ_VCCINT_VOLTAGE_OUT_OF_RANGE	0x04	VCCINT obtained from XADC is out of range
XSK_EFUSEPS_ERROR_READ	0x00B0	Error in reading rows
EFUSE Write Error Codes		
XSK_EFUSEPS_ERROR_WRITE_TEMPERATURE_OUT_OF_RANGE	0x05	Temperature obtained from XADC is out of range
XSK_EFUSEPS_ERROR_WRITE_VCCAUX_VOLTAGE_OUT_OF_RANGE	0x06	VCCAUX obtained from XADC is out of range
XSK_EFUSEPS_ERROR_WRITE_VCCINT_VOLTAGE_OUT_OF_RANGE	0x07	VCCINT obtained from XADC is out of range

Table 26: **PS eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
XSK_EFUSEPS_ERROR_VERIFICATION	0x08	Verification error
XSK_EFUSEPS_ERROR_RSA_HASH_ALREADY_PROGRAMMED	0x09	RSA hash was already programmed
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_JTAG_USER_CODE_ALREADY_PROGRAMED	0x14	JTAG user code is already programmed
XSK_EFUSEPS_ERROR_USER_KEY_ALREADY_PROGRAMMED	0x15	User key is already programmed
EFUSE CNTRL Error Codes		
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
XADC Error Codes		
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
Utils Error Codes		
XSK_EFUSEPS_ERROR_PARAMETER_NULL	0x10	Passed parameter null
XSK_EFUSEPS_ERROR_STRING_INVALID	0x20	Passed string is invalid
XSKEfuse_Write/Read()common Error Codes		
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
XSKEfusePs_Write() Error Codes		
XSK_EFUSEPS_ERROR_WRITE_128K_CRC_BIT	0x9100	Error in enabling 128K CRC
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_WRONG_TBIT_PATTERN	0xA200	Error in programming TBIT pattern
XSK_EFUSEPS_ERROR_WRITE_AES_KEY	0xA300	Error in programming AES key

Table 26: **PS eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
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 PPK 0 hash
XSK_EFUSEPS_ERROR_WRITE_PPK1_HASH	0xA700	Error in programming PPK 1 hash
XSK_EFUSEPS_ERROR_WRITE_JTAG_USERCODE	0xA800	Error in programming JTAG user code
XSK_EFUSEPS_ERROR_BEFORE_PROGRAMMING	0xA900	Error occurred before programming
XSK_EFUSEPS_ERROR_PROGRAMMING_TBIT_PATTERN	0XB100	Error in programming TBITS
XSK_EFUSEPS_ERROR_CACHE_LOAD	0xB000	Error in re-loading CACHE
XSKEfusePs_Read() Error Codes		
XSK_EFUSEPS_ERROR_READ_RSA_HASH	0xA100	Error in reading RSA key

Table 27 shows the BBRAM error codes for Zynq UltraScale+ MPSoC.

Table 27: **BBRAM Error Codes for Zynq UltraScale+ MPSoC**

Error Code	Value	Description
XSK_ZYNQMP_BBRAMPS_ERROR_IN_PRGRMG_ENABLE	0x01	Error in programming enable
XSK_ZYNQMP_BBRAMPS_ERROR_IN_CRC_CHECK	0xB000	Error in CRC check after programming AES key
XSK_ZYNQMP_BBRAMPS_ERROR_IN_PRGRMG	0xC000	Error in programming AES key

Status Code

For Zynq and UltraScale in `xilskey_efuse_example.c` the status 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.

Error codes are as described in Table 25, and Table 26. Table 28 shows the status codes.

Table 28: **Status Codes**

Status Code Value	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 in `xilskey_bbramps_zynqmp_example.c` and `xilskey_efuseps_zynqmp_example.c` files, the status 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.

Procedures

Creating an SVF File using XMD

Procedures

Use the following XMD code to create an SVF from the generated ELF file.

Note: The path to the ELF file is provided in the OPT file.

```
“xmd -tcl efuse.tcl -opt efuse.opt”
```

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.

eFUSE Driver Compilation Procedure for OCM

1. Open the linker script (`lscript.ld`) in the SDK project.
2. Map all the sections to point to `ps7_ram_0_S_AXI_BASEADDR` instead of `ps7_ddr_0_S_AXI_BASEADDR`.

   Example: Click the **Memory Region** tab for the `.text` section 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.

Status/Error codes are as described in Error Codes.

UltraScale eFUSE Access Procedure

Accessing UltraScale MicroBlaze eFuse is done by using block RAM initialization.

- 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.
- All inputs (TDO) and all outputs (TDI, TMS, TCK) of MASTER_JTAG 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 Ctrl memory mapped (1MB). The system management wizard should be operated in DRP interface.

Note: MASTER_JTAG will disable all other JTAGs

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.
6. For calculating CRC of AES key reverse polynomial is `0x82F63B78` or you can use the API u32 Xilskey_CrcCalculation(u8 *Key)

LibXil SKey Library APIs

This section provides linked summary and detailed descriptions of the LibXil SKey library APIs.

API Summary

The following is a summary list of APIs provided by the LibXil SKey library. Descriptions of the APIs follow the list.

u32 XilSKey_EfusePs_Write (XilSKey_EPs *InstancePtr)

u32 XilSKey_EfusePs_Read(XilSKey_EPs *InstancePtr)

u32 XilSKey_EfusePl_Program (XilSKey_EPl *InstancePtr)

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

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

u32 XilSKey_EfusePl_ReadKey(XilSKey_EPl *InstancePtr);

`u32 XilSKey_EfusePs_Write (XilSKey_EPs *InstancePtr)`

Parameters	`InstancePtr`: The pointer to the PS eFUSE handler that describes which PS eFUSE bit should be burned.
Returns	`XST_SUCCESS` on success. In case of error, value is as defined in `xilskey_utils.h`. The error value is a combination of an upper 8-bit value and a lower 8-bit value. For example, `0x8A03` should be checked in `xilskey_utils.h` as `0x8A00` and `0x03`. The upper 8-bit value signifies the major error, and the lower 8-bit value provides more detail about the error.

Description	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
Includes	`xilskey_eps.h, xilskey_epshw.h, xilskey_utils.h`

`u32 XilSKey_EfusePs_Read(XilSKey_EPs *InstancePtr)`

Parameters	`InstancePtr`: The pointer to the PS eFUSE handler that describes which PS eFUSE bit should be burned.
Returns	`XST_SUCCESS on success.` In case of error, the value is as defined in `xilskey_utils.h`. The error value is a combination of an upper 8-bit value and a lower 8-bit value. For example, 0x8A03 should be checked in `xilskey_utils.h` as 0x8A00 and 0x03. The upper 8-bit value signifies the major error and the lower 8-bit values provides more detail about the error.
Description	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
Includes	`xilskey_eps.h, xilskey_epshw.h, xilskey_utils.h`

`u32 XilSKey_EfusePl_Program (XilSKey_EPl *InstancePtr)`

Parameters	InstancePtr is input data to be written to PL eFUSE
Returns	`XST_SUCCESS` on success. In case of error, the value is defined in `xilskey_utils.h`. The error value is a combination of the upper 8-bit value and lower 8-bit value. For example, 0x8A03 should be checked in `xilskey_utils.h` as 0x8A00 and 0x03. The upper 8-bit value signifies the major error, and the lower 8-bit value provides more detail.

Description	When called, this API: • Initializes the timer, XADC and JTAG server subsystems. • Writes the AES & User Keys if requested. • Writes the Control Bits if requested. • In UltraScale, it also programs the RSA key Hash Returns an error if: • 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.
Includes	`xilskey_utils.h, xilskey_epl.h`

`u32 XilSKey_EfusePs_ReadStatus(XilSKey_EPs *InstancePtr, u32 *StatusBits);`

Parameters	• InstancePtr - Pointer to PS eFUSE instance • StatusBits - Buffer to store status register value
Returns	`XST_SUCCESS` on success. On failure, returns error codes as described in "Error Codes," page 16.
Description	This API unlocks the controller and reads the PS eFUSE status register.
Includes	`xilskey_eps.h, xilskey_utils.h`

`u32 XilSKey_EfusePl_ReadStatus(XilSKey_EPl *InstancePtr, u32 *StatusBits);`

Parameters	• InstancePtr - Pointer to PL eFUSE instance • StatusBits - Buffer to store status bits
Returns	`XST_SUCCESS` on success. On failure, returns error codes as described in "Error Codes," page 16.
Description	This API reads the status bits from row 0. It initializes the timer, XADC and JTAG server subsystems, if not already done so. In UltraScale it reads the Status register and gets all the secure and control bits.
Includes	`xilskey_epl.h, xilskey_utils.h`

`u32 XilSKey_EfusePl_ReadKey(XilSKey_EPl *InstancePtr);`

Parameters	InstancePtr - Pointer to PL eFUSE instance
Returns	`XST_SUCCESS` on success. On failure, returns error codes as described in "Error Codes," page 16.
Description	This API reads the AES and user key 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 UltraScale eFuse, this API performs same as the above but reads extra key RSA key hash.
Includes	`xilskey_epl.h, xilskey_utils.h`

BBRAM API Description

This section provides a linked summary and detailed descriptions of the battery-backed RAM (BBRAM) APIs.

API Summary

`int XilSKey_Bbram_Program(XilSKey_Bbram *InstancePtr)`

Parameters	BBRAM instance pointer
Returns	`XST_SUCCESS` on success, or `XST_FAILURE` on failure.
Description	API to program and verify the key.
Includes	`xilskey_utils.h`, `xilskey_bbram.h`

Important! This API performs BBRAM program and verify together. This is how the BBRAM algorithm works and it is not possible to do program/verify operations independently.

Zynq UltraScale+ MPSoC API Description

This section provides linked summary and detailed descriptions of the Zynq UltraScale+ MPSoC eFUSE and battery-backed RAM (BBRAM) APIs.

BBRAM PS API Summary

The following is a summary list of BBRAM PS APIs for Zynq UltraScale+ MPSoC. Descriptions of the APIs follow the list.

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

`u32 XilSKey_ZynqMp_Bbram_Program(u32 *AesKey)`

Parameters	Aes Key is a pointer to an array which holds AES key to be programmed.
Returns	XST_SUCCESS if programming and verification is done successfully. ErrorCode if it fails to program.
Description	This API programs the Zynq UltraScale+ MPSoc's BBRAM key with the provided key and also performs CRC check of programmed key.
Includes	`xilskey_utils.h, xilskey_bbram.h`

`void XilSKey_ZynqMp_Bbram_Zeroise()`

Parameters	None.
Returns	XST_SUCCESS if programming and verification is done successfully. ErrorCode if it fails to program.
Description	This API zeroises the key programmed in BBRAM.
Includes	`xilskey_utils.h, xilskey_bbram.h`

eFUSE PS API Summary

The following is a summary list of eFUSE APIs for Zynq UltraScale+ MPSoC. Descriptions of the APIs follow the list.

- u32 XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc(u32 CrcValue)
- u32 XilSKey_ZynqMp_EfusePs_ReadUserKey(u32 *UseKeyPtr, 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)
- u32 XilSKey_ZynqMp_EfusePs_ReadJtagUsrCode(u32 *JtagUsrCode, u8 ReadOption)
- void XilSKey_ZynqMp_EfusePs_ReadDna(u32 *DnaRead)
- u32 XilSKey_ZynqMp_EfusePs_ReadSecCtrlBits(XilSKey_SecCtrlBits *ReadBackSecCtrlBits, u8 ReadOption)
- u32 XilSKey_ZynqMp_EfusePs_CacheLoad()
- u32 XilSKey_ZynqMp_EfusePs_Write(XilSKey_ZynqMpEPs *InstancePtr)
- u32 XilSKey_CrcCalculation(u8 *Key)

`u32 XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc(u32 CrcValue)`

Parameters	CrcValue is the CRC of expected AES key.
Returns	XST_SUCCESS if CRC check is passed. XST_FAILURE if CRC check is failed.
Description	This API performs the CRC check of eFUSE's AES key.
Includes	`xilskey_utils.h, xilskey_eps_zynqmp.h`

`u32 XilSKey_ZynqMp_EfusePs_ReadUserKey(u32 *UseKeyPtr, u8 ReadOption)`

Parameters	UserkeyPtr is a pointer to an array which holds the readback userkey in. ReadOption is a u8 variable which has to be provided by user based on which the input reading happens from cache or from efuse array. • 0 - Reads from cache • 1 - Reads from efuse array
Returns	XST_SUCCESS if key is read successfully. ErrorCode if it fails to read eFUSE user key.
Description	This API reads User key from eFUSE memory or Cache based on user input and stores in UseKeyPtr.
Includes	`xilskey_utils.h, xilskey_eps_zynqmp.h`

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

Parameters	Ppk0Hash is a pointer to an array which holds the readback PPK0 hash in. ReadOption is a u8 variable which has to be provided by user based on which the input reading happens from cache or from efuse array. • 0 - Reads from cache • 1 - Reads from efuse array
Returns	XST_SUCCESS if key is read successfully. ErrorCode if it fails to read PPK0 hash of eFUSE.

Description	This API reads PPK0 hash from eFUSE memory or Cache based on user input and stores in Ppk0Hash.
Includes	`xilskey_utils.h, xilskey_eps_zynqmp.h`

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

Parameters
: Ppk1Hash is a pointer to an array which holds the readback PPK1 hash in.
ReadOption is a u8 variable which has to be provided by user based on which the input reading happens from cache or from efuse array.

- 0 - Reads from cache
- 1 - Reads from efuse array

Returns
: XST_SUCCESS if key is read successfully.
ErrorCode if it fails to read PPK1 hash of eFUSE.

Description
: This API reads PPK1 hash from eFUSE memory or Cache based on user input and stores in Ppk1Hash.

Includes
: `xilskey_utils.h, xilskey_eps_zynqmp.h`

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

Parameters
: SpkId is a pointer which holds the readback SPK ID in.
ReadOption is a u8 variable which has to be provided by user based on which the input reading happens from cache or from efuse array.

- 0 - Reads from cache
- 1 - Reads from efuse array

Returns
: XST_SUCCESS if key is read successfully.
ErrorCode if it fails to read SPK ID of eFUSE.

Description
: This API reads SPK ID from eFUSE memory or Cache based on user input and stores in SpkId.

Includes
: `xilskey_utils.h, xilskey_eps_zynqmp.h`

`u32 XilSKey_ZynqMp_EfusePs_ReadJtagUsrCode(u32 *JtagUsrCode, u8 ReadOption)`

Parameters
: JtagUsrCode is a pointer which holds the readback JTAG user code in.
ReadOption is a u8 variable which has to be provided by user based on which the input reading happens from cache or from efuse array.

- 0 - Reads from cache
- 1 - Reads from efuse array

Returns
: XST_SUCCESS if key is read successfully.
ErrorCode if it fails to read JTAG user code of eFUSE.

Description
: This API reads JTAG user code from eFUSE memory or Cache based on user input and stores in JtagUsrCode.

Includes
: `xilskey_utils.h, xilskey_eps_zynqmp.h`

`void XilSKey_ZynqMp_EfusePs_ReadDna(u32 *DnaRead)`

Parameters	DnaRead is a pointer which holds the readback DNA in.
Returns	XST_SUCCESS if key is read successfully. ErrorCode if it fails to read DNA.
Description	This API reads DNA from Cache stores in DnaRead.
Includes	`xilskey_utils.h, xilskey_eps_zynqmp.h`

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

Parameters	ReadBackSecCtrlBits is a pointer to the XilSKey_SecCtrlBits structure which holds the secure control bits read back. ReadOption is a u8 variable which has to be provided by user based on which the input reading happens from cache or from efuse array. • 0 - Reads from cache • 1 - Reads from efuse array
Returns	XST_SUCCESS if key is read successfully. ErrorCode if it fails to read secure and control bits of eFUSE.
Description	This API reads secure control bits from eFUSE memory or Cache based on user input and stores in ReadBackSecCtrlBits.
Includes	`xilskey_utils.h, xilskey_eps_zynqmp.h`

`u32 XilSKey_ZynqMp_EfusePs_CacheLoad()`

Parameters	None.
Returns	XST_SUCCESS if cache reload is successfully. ErrorCode if it fails to reload cache of efuse.
Description	This API reloads caches of efuse, it updates cache with efuse memory.
Includes	`xilskey_utils.h, xilskey_eps_zynqmp.h`

`u32 XilSKey_ZynqMp_EfusePs_Write(XilSKey_ZynqMpEPs *InstancePtr)`

Parameters	InstancePtr is a pointer to efuse PS instance.
Returns	XST_SUCCESS if efuse programming is successfully. ErrorCode if it fails to program eFUSE.
Description	This API programs the efuse based on the user inputs.
Includes	`xilskey_utils.h, xilskey_eps_zynqmp.h`

`u32 XilSKey_CrcCalculation(u8 *Key)`

Parameters	Key is a hexa decimal character string for which CRC has to be calculated.

Returns	CRC of provided key.
Description	This API calculates the CRC of provided AES key of eFUSE. (This API calculates CRC for both Ultrascale and also Zynq MP platform eFUSE).
Includes	`xilskey_utils.h`