XILINX
ALL PROGRAMMABLE™

Appendix B

LibXil SKey for Zynq-7000 AP SoC Devices (v2.1)

Overview

The LibXil SKey Library provides a programming mechanism for user-defined eFUSE bits and for programming the KEY into battery-backed RAM (BBRAM).

- 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.

The following user application (example) files are provided:

- The efuse example file lets you write into the PS/PL eFUSE.
- The BBRAM example file let you write the key to BBRAM.

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

The following table lists the eFUSE application SDK project files, folders, and macros.

Table B-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.
`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.

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.

Table B-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.
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.

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

Macro Name	Description
XSK_EFUSEPS_RSA_KEY_HASH_VALUE	"00000000000000000000000000000000000000000000000000000000000000000000000000000000" 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 B-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.
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.

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

Macro Name	Definition
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.

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

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

a. 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 B-4.
- The MIO used for the MUX Select Line, shown in Table B-5.
- The Value on the MUX Select line, shown in Table B-6, to select JTAG for PL eFUSE writing.

The following figure illustrates correct MUX usage.

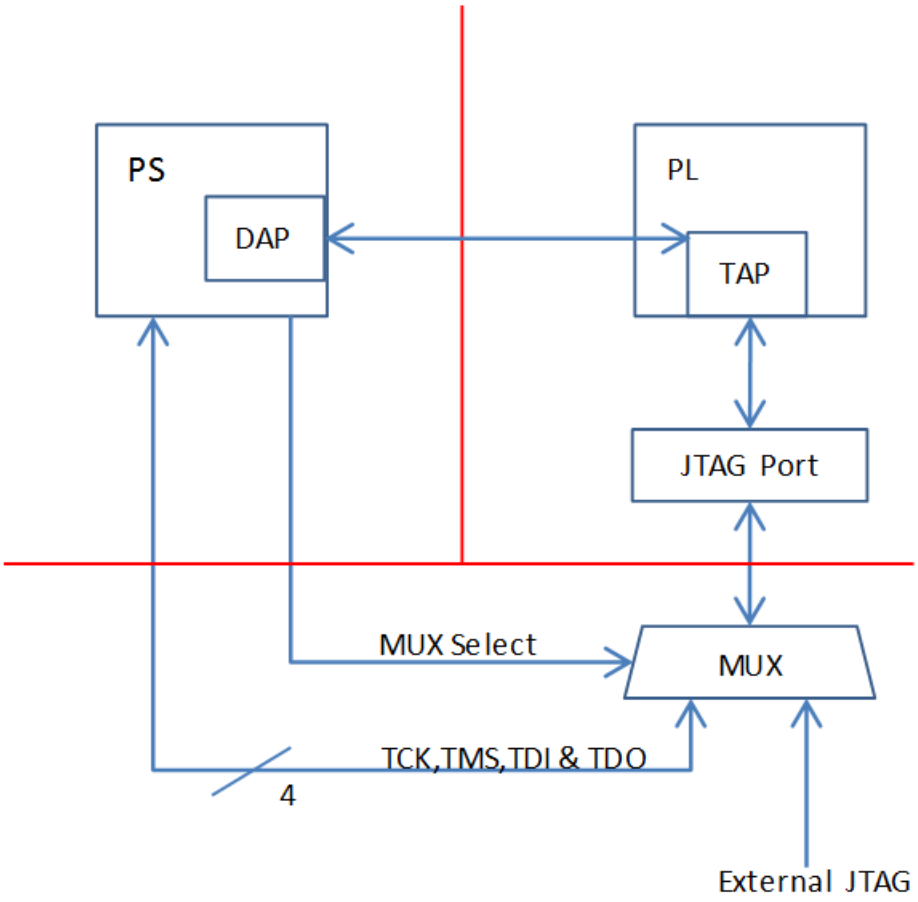

Figure B 1: **MUX Usage**

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 B-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 B-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 B-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.

Table B-7: **AES and User Key Parameters *(Cont'd)***

Parameter Name	Description
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 B-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 following MIO pins are used for PL JTAG signals. These can be changed depending on your hardware

Table B-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 B-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 B-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.

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 B-12: **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
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
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

Table B-12: **PL eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
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

Table B-12: **PL eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
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
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

PS eFUSE Error Codes

Table B-13: **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
EFUSE Write Error Codes		
XSK_EFUSEPS_ERROR_WRITE_TEMPERATURE_OUT_OF_RANGE	0x05	Temperature obtained from XADC is out of range

Table B-13: **PS eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
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
XSK_EFUSEPS_ERROR_VERIFICATION	0x08	Verification error
XSK_EFUSEPS_ERROR_RSA_HASH_ALREADY_PROGRAMMED	0x09	RSA hash was 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

Table B-13: **PS eFUSE Error Codes *(Cont'd)***

Error Code	Value	Description
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
XSKEfusePs_Read() Error Codes		
XSK_EFUSEPS_ERROR_READ_RSA_HASH	0xA100	Error in reading RSA key

Status Code

The status is conveyed through 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 12, page 77, and Table 13, page 79.

Table B-14: **Status Codes**

Status Code	Description
Value 0x0000ZZZZ	Represents PS eFUSE is successful & PL eFUSE process returned with error.
Value 0xYYYY0000	Represents PL eFUSE is successful & PS eFUSE process returned with error.
Value 0xFFFF0000	Represents PS eFUSE is not initiated & PL eFUSE is successful.

Table B-14: **Status Codes**

Status Code	Description
Value 0x0000FFFF	Represents PL eFUSE is not initiated & PS eFUSE is successful.
Value 0xFFFFZZZZ	Represents PS eFUSE is not initiated & PL eFUSE is process returned with error.
Value 0xYYYYFFFF	Represents PL eFUSE is not initiated & PS eFUSE is process returned with error.

Creating an SVF File using XMD

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.

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. • Read the PS eFUSE (Hash Value) • Locks the PS eFUSE Controller. Returns error if: • The reference clock frequency is not in between 20 and 60MHz. • The system is not in a position to write the requested PS eFUSE bits (because the bits are already written or not allowed to write) • 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. 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 76.
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 76.
Description	This API reads the status bits from row 0. It initializes the timer, XADC and JTAG server subsystems, if not already done so.
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 76.
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.
Includes	xilskey_epl.h, xilskey_utils.h

BBRAM API Description

This section provides 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.