Xilinx Standalone Library Documentation

OS and Libraries Document Collection

UG643 (v2020.2) November 24, 2020

XILINX®

Table of Contents

XILINX

Chapter 1

Xilinx OS and Libraries Overview

The Vitis™ Unified Software Development Environment provides a variety of Xilinx software packages, including drivers, libraries, board support packages, and complete operating systems to help you develop a software platform. This document collection provides information on these. Complete documentation for other operating systems can be found in the their respective reference guides. Device drivers are documented along with the corresponding peripheral documentation. The documentation is listed in the following table; click the name to open the document.

Library Name	Summary
Chapter 2: Xilinx Standard C Libraries	Describes the software libraries available for the embedded processors.
Chapter 3: Standalone Library v7.3	Describes the Standalone platform, a single-threaded, simple operating system (OS) platform that provides the lowest layer of software modules used to access processor-specific functions. Some typical functions offered by the Standalone platform include setting up the interrupts and exceptions systems, configuring caches, and other hardware specific functions. The Hardware Abstraction Layer (HAL) is described in this document.
Chapter 4: LwIP 2.1.1 Library	Describes the SDK port of the third party networking library, Light Weight IP (lwIP) for embedded processors.
Chapter 5: XilIsf Library v5.15	Describes the In System Flash hardware library, which enables higher-layer software (such as an application) to communicate with the Isf. XilIsf supports the Xilinx In-System Flash and external Serial Flash memories from Atmel (AT45XXXD), Spansion(S25FLXX), Winbond W25QXX, and Micron N25QXX.
Chapter 6: XilFFS Library v4.4	Xilffs is a generic FAT file system that is primarily added for use with SD/eMMC driver. The file system is open source and a glue layer is implemented to link it to the SD/eMMC driver.
Chapter 7: XilSecure Library v4.3	The XilSecure library provides APIs to access secure hardware on the Zynq UltraScale+ MPSoC devices.
Chapter 8: XilSkey Library v7.0	The XilSKey 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.
Chapter 9: XilPM Library v3.2	The Zynq UltraScale+ MPSoC and Versal ACAP power management framework is a set of power management options, based upon an implementation of the extensible energy management interface (EEMI). The power management framework allows software components running across different processing units (PUs) on a chip or device to issue or respond to requests for power management.
Chapter 10: XilFPGA Library v5.3	The XilFPGA library provides an interface to the Linux or bare-metal users for configuring the programmable logic (PL) over PCAP from PS. The library is designed for Zynq UltraScale+ MPSoC and Versal ACAP to run on top of Xilinx standalone BSPs.
Chapter 11: XilMailbox v1.2	The XilMailbox library provides the top-level hooks for sending or receiving an inter-processor interrupt (IPI) message using the Zynq UltraScale+ MPSoC and Versal ACAP IPI hardware.

Library Name	Summary
Chapter 12: XilSEM	The Xilinx Soft Error Mitigation (XilSEM) library is a pre-configured, pre-verified solution to detect and optionally correct soft errors in Configuration Memory of Versal ACAPs.

About the Libraries

The Standard C support library consists of the `newlib`, `libc`, which contains the standard C functions such as `stdio`, `stdlib`, and `string` routines. The math library is an enhancement over the `newlib` math library, `libm`, and provides the standard math routines. The LibXil libraries consist of the following:

- LibXil Driver (Xilinx device drivers)
- XilMFS (Xilinx memory file system)
- XilFlash (a parallel flash programming library)
- XilIsf (a serial flash programming library)

The Hardware Abstraction Layer (HAL) provides common functions related to register I/O, exception, and cache. These common functions are uniform across MicroBlaze™ and Cortex A9 processors. The Standalone platform document provides some processor specific functions and macros for accessing the processor-specific features. Most routines in the library are written in C and can be ported to any platform. User applications must include appropriate headers and link with required libraries for proper compilation and inclusion of required functionality. These libraries and their corresponding include files are created in the processor `\lib` and `\include` directories, under the current project, respectively. The `-I` and `-L` options of the compiler being used should be leveraged to add these directories to the search paths.

Library Organization

Your application can interface with the components in a variety of ways. The libraries are independent of each other, with the exception of some interactions. The LibXil drivers and the Standalone form the lowermost hardware abstraction layer. The library and OS components rely on standard C library components. The math library, `libm.a` is also available for linking with the user applications.

***Note*:** "LibXil Drivers" are the device drivers included in the software platform to provide an interface to the peripherals in the system. These drivers are provided along with the Vitis Unified Software Development Environment and are configured by Libgen. This document collection contains a section that briefly discusses the concept of device drivers and the way they integrate with the board support package in Vitis.

Taking into account some restrictions and implications, which are described in the reference guides for each component, you can mix and match the component libraries.

Xilinx Standard C Libraries

The Vitis™ Unified Software Development Environment libraries and device drivers provide standard C library functions, as well as functions to access peripherals. The SDK libraries are automatically configured based on the Microprocessor Software Specification (MSS) file. These libraries and include files are saved in the current project lib and include directories, respectively. The -I and -L options of mb-gcc are used to add these directories to its library search paths.

Standard C Library (libc.a)

The standard C library, `libc.a`, contains the standard C functions compiled for the MicroBlaze™ processor or the Cortex A9 processor. You can find the header files corresponding to these C standard functions in the `<XILINX_SDK>/gnu/<processor>/<platform>/<processor-lib>/include` folder, where:

- <Vitis> is the Vitis Unified Software Development Environment installation path
- <processor> is either ARM or MicroBlaze
- <platform> is either Solaris (sol), or Windows (nt), or Linux (lin)
- <processor-lib> is either arm-xilinx-eabi or microblaze-xilinx-elf

The `lib.c` directories and functions are:

```
_ansi.h    fastmath.h  machine/    reent.h     stdlib.h  utime.h    _syslist.h
fcntl.h    malloc.h    regdef.h    string.h    utmp.h    ar.h       float.h
math.h     setjmp.h    sys/        assert.h    grp.h     paths.h    signal.h
termios.h  ctype.h     ieeefp.h    process.h   stdarg.h  time.h     dirent.h
imits.h    pthread.h   stddef.h    nctrl.h     errno.h   locale.h   pwd.h
stdio.h    unistd.h
```

Programs accessing standard C library functions must be compiled as follows:

- For MicroBlaze processors:

  ```
  mb-gcc <C files>
  ```

- For Cortex A9 processors:

  ```
  arm-xilinx-eabi-gcc <C files>
  ```

The `libc` library is included automatically. For programs that access `libm` math functions, specify the `lm` option. For more information on the C runtime library, see *MicroBlaze Processor Reference Guide* (UG081).

Xilinx C Library (libxil.a)

The Xilinx C library, `libxil.a`, contains the following object files for the MicroBlaze processor embedded processor:

- _exception_handler.o
- _interrupt_handler.o
- _program_clean.o
- _program_init.o

Default exception and interrupt handlers are provided. The `libxil.a` library is included automatically. Programs accessing Xilinx C library functions must be compiled as follows:

```
mb-gcc <C files>
```

Memory Management Functions

The MicroBlaze processor and Cortex A9 processor C libraries support the standard memory management functions such as `malloc()`, `calloc()`, and `free()`. Dynamic memory allocation provides memory from the program heap. The heap pointer starts at low memory and grows toward high memory. The size of the heap cannot be increased at runtime. Therefore an appropriate value must be provided for the heap size at compile time. The `malloc()` function requires the heap to be at least 128 bytes in size to be able to allocate memory dynamically (even if the dynamic requirement is less than 128 bytes).

***Note*:** The return value of malloc must always be checked to ensure that it could actually allocate the memory requested.

Arithmetic Operations

Software implementations of integer and floating point arithmetic is available as library routines in libgcc.a for both processors. The compiler for both the processors inserts calls to these routines in the code produced, in case the hardware does not support the arithmetic primitive with an instruction.

Details of the software implementations of integer and floating point arithmetic for MicroBlaze processors are listed below:

Integer Arithmetic

By default, integer multiplication is done in software using the library function `__mulsi3`. Integer multiplication is done in hardware if the `-mno-xl-soft-mul` mb-gcc option is specified. Integer divide and mod operations are done in software using the library functions `__divsi3` and `__modsi3`. The MicroBlaze processor can also be customized to use a hard divider, in which case the `div` instruction is used in place of the `__divsi3` library routine. Double precision multiplication, division and mod functions are carried out by the library functions `__muldi3`, `__divdi3`, and `__moddi3` respectively. The unsigned version of these operations correspond to the signed versions described above, but are prefixed with an `__u` instead of `__`.

Floating Point Arithmetic

All floating point addition, subtraction, multiplication, division, and conversions are implemented using software functions in the C library.

Thread Safety

The standard C library provided with SDK is not built for a multi-threaded environment. STDIO functions like `printf()`, `scanf()` and memory management functions like `malloc()` and `free()` are common examples of functions that are not thread-safe. When using the C library in a multi-threaded environment, proper mutual exclusion techniques must be used to protect thread unsafe functions.

Input/Output Functions

The SDK libraries contains standard C functions for I/O, such as printf and scanf. These functions are large and might not be suitable for embedded processors. The prototypes for these functions are available in the stdio.h file.

These Input/Output routines require that a newline is terminated with both a CR and LF. Ensure that your terminal CR/LF behavior corresponds to this requirement.

***Note*:** The C standard I/O routines such as printf, scanf, vfprintf are, by default, line buffered. To change the buffering scheme to no buffering, you must call setvbuf appropriately. For example:

```
setvbuf (stdout, NULL, _IONBF, 0);
```

These Input/Output routines require that a newline is terminated with both a CR and LF. Ensure that your terminal CR/LF behavior corresponds to this requirement.

For more information on setting the standard input and standard output devices for a system, see *Embedded System Tools Reference Manual* (UG1043). In addition to the standard C functions, the SDK processors library provides the following smaller I/O functions:

Table 1: **Quick Function Reference**

Type	Name	Arguments
void	print	void
void	putnum	void
void	xil_printf	void

Functions

print

This function prints a string to the peripheral designated as standard output in the Microprocessor Software Specification (MSS) file. This function outputs the passed string as is and there is no interpretation of the string passed. For example, a \n passed is interpreted as a new line character and not as a carriage return and a new line as is the case with ANSI C printf function.

Prototype

```
void print(char *);
```

putnum

This function converts an integer to a hexadecimal string and prints it to the peripheral designated as standard output in the MSS file.

Prototype

```
void putnum(int);
```

xil_printf

`xil_printf()` is a light-weight implementation of printf. It is much smaller in size (only 1 Kb). It does not have support for floating point numbers. `xil_printf()` also does not support printing of long (such as 64-bit) numbers.

About format string support:

The format string is composed of zero or more directives: ordinary characters (not %), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character %, and ends with a conversion specifier.

In between there can be (in order) zero or more flags, an optional minimum field width and an optional precision. Supported flag characters are:

The character % is followed by zero or more of the following flags:

- 0 The value should be zero padded. For d, x conversions, the converted value is padded on the left with zeros rather than blanks. If the 0 and - flags both appear, the 0 flag is ignored.
- - The converted value is to be left adjusted on the field boundary. (The default is right justification.) Except for n conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A - overrides a 0 if both are given.

About supported field widths

Field widths are represented with an optional decimal digit string (with a nonzero in the first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it is padded with spaces on the left (or right, if the left-adjustment flag has been given). The supported conversion specifiers are:

- d The int argument is converted to signed decimal notation.
- l The int argument is converted to a signed long notation.
- x The unsigned int argument is converted to unsigned hexadecimal notation. The letters abcdef are used for x conversions.
- c The int argument is converted to an unsigned char, and the resulting character is written.
- s The const char* argument is expected to be a pointer to an array of character type (pointer to a string).

Characters from the array are written up to (but not including) a terminating NULL character; if a precision is specified, no more than the number specified are written. If a precision s given, no null character need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating NULL character.

Prototype

```
void xil_printf(const *char ctrl1,...);
```

Standalone Library v7.3

Xilinx Hardware Abstraction Layer API

Xilinx Hardware Abstraction Layer API

This section describes the Xilinx Hardware Abstraction Layer API, These APIs are applicable for all processors supported by Xilinx.

Assert APIs and Macros

This file contains basic assert related functions for Xilinx software IP.

The xil_assert.h file contains assert related functions and macros.

Assert APIs/Macros specifies that a application program satisfies certain conditions at particular points in its execution. These function can be used by application programs to ensure that, application code is satisfying certain conditions.

Table 2: **Quick Function Reference**

Type	Name	Arguments
void	Xil_Assert	file line
void	Xil_AssertSetCallback	routine
void	XNullHandler	void * NullParameter

Functions

Xil_Assert

Implement assert.

Currently, it calls a user-defined callback function if one has been set. Then, it potentially enters an infinite loop depending on the value of the Xil_AssertWait variable.

Note: None.

Prototype

```
void Xil_Assert(const char8 *File, s32 Line);
```

Parameters

The following table lists the `Xil_Assert` function arguments.

Table 3: **Xil_Assert Arguments**

Name	Description
file	filename of the source
line	linenumber within File

Returns

None.

Xil_AssertSetCallback

Set up a callback function to be invoked when an assert occurs.

If a callback is already installed, then it will be replaced.

Note: This function has no effect if NDEBUG is set

Prototype

```
void Xil_AssertSetCallback(Xil_AssertCallback Routine);
```

Parameters

The following table lists the `Xil_AssertSetCallback` function arguments.

Table 4: **Xil_AssertSetCallback Arguments**

Name	Description
routine	callback to be invoked when an assert is taken

Returns

None.

XNullHandler

Null handler function.

This follows the XInterruptHandler signature for interrupt handlers. It can be used to assign a null handler (a stub) to an interrupt controller vector table.

***Note*:** None.

Prototype

```
void XNullHandler(void *NullParameter);
```

Parameters

The following table lists the `XNullHandler` function arguments.

Table 5: **XNullHandler Arguments**

Name	Description
NullParameter	arbitrary void pointer and not used.

Returns

None.

Definitions

#Define Xil_AssertVoid

Description

This assert macro is to be used for void functions.

This in conjunction with the Xil_AssertWait boolean can be used to accommodate tests so that asserts which fail allow execution to continue.

Parameters

The following table lists the `Xil_AssertVoid` function arguments.

Table 6: **Xil_AssertVoid Arguments**

Name	Description
Expression	expression to be evaluated. If it evaluates to false, the assert occurs.

Returns

Returns void unless the Xil_AssertWait variable is true, in which case no return is made and an infinite loop is entered.

#Define Xil_AssertNonvoid

Description

This assert macro is to be used for functions that do return a value.

This in conjunction with the Xil_AssertWait boolean can be used to accommodate tests so that asserts which fail allow execution to continue.

Parameters

The following table lists the `Xil_AssertNonvoid` function arguments.

Table 7: **Xil_AssertNonvoid Arguments**

Name	Description
Expression	expression to be evaluated. If it evaluates to false, the assert occurs.

Returns

Returns 0 unless the Xil_AssertWait variable is true, in which case no return is made and an infinite loop is entered.

#Define Xil_AssertVoidAlways

Description

Always assert.

This assert macro is to be used for void functions. Use for instances where an assert should always occur.

Returns

Returns void unless the Xil_AssertWait variable is true, in which case no return is made and an infinite loop is entered.

#Define Xil_AssertNonvoidAlways

Description

Always assert.

This assert macro is to be used for functions that do return a value. Use for instances where an assert should always occur.

Returns

Returns void unless the Xil_AssertWait variable is true, in which case no return is made and an infinite loop is entered.

Variables

u32 Xil_AssertStatus

This variable allows testing to be done easier with asserts. An assert sets this variable such that a driver can evaluate this variable to determine if an assert occurred.

s32 Xil_AssertWait

This variable allows the assert functionality to be changed for testing such that it does not wait infinitely. Use the debugger to disable the waiting during testing of asserts.

IO Interfacing APIs

Contains I/O functions for memory-mapped or non-memory-mapped I/O architectures.

The xil_io.h file contains the interface for the general I/O component, which encapsulates the Input/Output functions for the processors that do not require any special I/O handling.

This file contains architecture-dependent code.

Note:

Table 8: **Quick Function Reference**

Type	Name	Arguments
INLINE u16	Xil_In16BE	UINTPTR Addr
INLINE u32	Xil_In32BE	UINTPTR Addr
INLINE void	Xil_Out16BE	UINTPTR Addr u16 Value
INLINE void	Xil_Out32BE	UINTPTR Addr u32 Value
INLINE u16	Xil_In16LE	UINTPTR Addr

Table 8: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
INLINE u32	Xil_In32LE	UINTPTR Addr
INLINE void	Xil_Out16LE	UINTPTR Addr u16 Value
INLINE void	Xil_Out32LE	UINTPTR Addr u32 Value
u16	Xil_EndianSwap16	u16 Data
u32	Xil_EndianSwap32	u32 Data
INLINE u8	Xil_In8	UINTPTR Addr
INLINE u16	Xil_In16	UINTPTR Addr
INLINE u32	Xil_In32	UINTPTR Addr
INLINE u64	Xil_In64	UINTPTR Addr
INLINE void	Xil_Out8	UINTPTR Addr u8 Value
INLINE void	Xil_Out16	UINTPTR Addr u16 Value
INLINE void	Xil_Out32	UINTPTR Addr u32 Value
INLINE void	Xil_Out64	UINTPTR Addr u64 Value
INLINE u32	Xil_SecureOut32	UINTPTR Addr u32 Value

Functions

Xil_In16BE

Perform an big-endian input operation for a 16-bit memory location by reading from the specified address and returning the value read from that address.

Prototype

```
INLINE u16 Xil_In16BE(UINTPTR Addr);
```

Parameters

The following table lists the `Xil_In16BE` function arguments.

Table 9: **Xil_In16BE Arguments**

Name	Description
Addr	contains the address at which to perform the input operation.

Returns

The value read from the specified input address with the proper endianness. The return value has the same endianness as that of the processor. For example, if the processor is little-endian, the return value is the byte-swapped value read from the address.

Xil_In32BE

Perform a big-endian input operation for a 32-bit memory location by reading from the specified address and returning the value read from that address.

Prototype

```
INLINE u32 Xil_In32BE(UINTPTR Addr);
```

Parameters

The following table lists the `Xil_In32BE` function arguments.

Table 10: **Xil_In32BE Arguments**

Name	Description
Addr	contains the address at which to perform the input operation.

Returns

The value read from the specified input address with the proper endianness. The return value has the same endianness as that of the processor. For example, if the processor is little-endian, the return value is the byte-swapped value read from the address.

Xil_Out16BE

Perform a big-endian output operation for a 16-bit memory location by writing the specified value to the specified address.

Prototype

```
INLINE void Xil_Out16BE(UINTPTR Addr, u16 Value);
```

Parameters

The following table lists the `Xil_Out16BE` function arguments.

Table 11: **Xil_Out16BE Arguments**

Name	Description
Addr	contains the address at which to perform the output operation.
Value	contains the value to be output at the specified address. The value has the same endianness as that of the processor. For example, if the processor is little-endian, the byteswapped value is written to the address.

Xil_Out32BE

Perform a big-endian output operation for a 32-bit memory location by writing the specified value to the specified address.

Prototype

```
INLINE void Xil_Out32BE(UINTPTR Addr, u32 Value);
```

Parameters

The following table lists the `Xil_Out32BE` function arguments.

Table 12: **Xil_Out32BE Arguments**

Name	Description
Addr	contains the address at which to perform the output operation.
Value	contains the value to be output at the specified address. The value has the same endianness as that of the processor. For example, if the processor is little-endian, the byteswapped value is written to the address.

Xil_In16LE

Perform a little-endian input operation for a 16-bit memory location by reading from the specified address and returning the value read from that address.

Prototype

```
INLINE u16 Xil_In16LE(UINTPTR Addr)[static];
```

Parameters

The following table lists the `Xil_In16LE` function arguments.

Table 13: **Xil_In16LE Arguments**

Name	Description
Addr	contains the address at which to perform the input operation.

Returns

The value read from the specified input address with the proper endianness. The return value has the same endianness as that of the processor. For example, if the processor is big-endian, the return value is the byte-swapped value read from the address.

Xil_In32LE

Perform a little-endian input operation for a 32-bit memory location by reading from the specified address and returning the value read from that address.

Prototype

```
INLINE u32 Xil_In32LE(UINTPTR Addr)[static];
```

Parameters

The following table lists the `Xil_In32LE` function arguments.

Table 14: **Xil_In32LE Arguments**

Name	Description
Addr	contains the address at which to perform the input operation.

Returns

The value read from the specified input address with the proper endianness. The return value has the same endianness as that of the processor. For example, if the processor is big-endian, the return value is the byte-swapped value read from the address.

Xil_Out16LE

Perform a little-endian output operation for a 16-bit memory location by writing the specified value to the specified address.

Prototype

```
INLINE void Xil_Out16LE(UINTPTR Addr, u16 Value)[static];
```

Parameters

The following table lists the `Xil_Out16LE` function arguments.

Table 15: **Xil_Out16LE Arguments**

Name	Description
Addr	contains the address at which to perform the input operation.
Value	contains the value to be output at the specified address. The value has the same endianness as that of the processor. For example, if the processor is big-endian, the byteswapped value is written to the address.

Xil_Out32LE

Perform a little-endian output operation for a 32-bit memory location by writing the specified value to the specified address.

Prototype

```
INLINE void Xil_Out32LE(UINTPTR Addr, u32 Value)[static];
```

Parameters

The following table lists the `Xil_Out32LE` function arguments.

Table 16: **Xil_Out32LE Arguments**

Name	Description
Addr	contains the address at which to perform the input operation.
Value	contains the value to be output at the specified address. The value has the same endianness as that of the processor. For example, if the processor is big-endian, the byteswapped value is written to the address

Xil_EndianSwap16

Perform a 16-bit endian conversion.

Prototype

```
u16 Xil_EndianSwap16(u16 Data);
```

Parameters

The following table lists the `Xil_EndianSwap16` function arguments.

Table 17: **Xil_EndianSwap16 Arguments**

Name	Description
Data	16 bit value to be converted

Returns

16 bit Data with converted endianness

Xil_EndianSwap32

Perform a 32-bit endian conversion.

Prototype

```
u32 Xil_EndianSwap32(u32 Data);
```

Parameters

The following table lists the `Xil_EndianSwap32` function arguments.

Table 18: **Xil_EndianSwap32 Arguments**

Name	Description
Data	32 bit value to be converted

Returns

32 bit data with converted endianness

Xil_In8

Performs an input operation for a memory location by reading from the specified address and returning the 8 bit Value read from that address.

Prototype

```
INLINE u8 Xil_In8(UINTPTR Addr);
```

Parameters

The following table lists the `Xil_In8` function arguments.

Table 19: **Xil_In8 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

The 8 bit Value read from the specified input address.

Xil_In16

Performs an input operation for a memory location by reading from the specified address and returning the 16 bit Value read from that address.

Prototype

```
INLINE u16 Xil_In16(UINTPTR Addr);
```

Parameters

The following table lists the `Xil_In16` function arguments.

Table 20: **Xil_In16 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

The 16 bit Value read from the specified input address.

Xil_In32

Performs an input operation for a memory location by reading from the specified address and returning the 32 bit Value read from that address.

Prototype

```
INLINE u32 Xil_In32(UINTPTR Addr);
```

Parameters

The following table lists the `Xil_In32` function arguments.

Table 21: **Xil_In32 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

The 32 bit Value read from the specified input address.

Xil_In64

Performs an input operation for a memory location by reading the 64 bit Value read from that address.

Prototype

```
INLINE u64 Xil_In64(UINTPTR Addr);
```

Parameters

The following table lists the `Xil_In64` function arguments.

Table 22: **Xil_In64 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

The 64 bit Value read from the specified input address.

Xil_Out8

Performs an output operation for an memory location by writing the 8 bit Value to the the specified address.

Prototype

```
INLINE void Xil_Out8(UINTPTR Addr, u8 Value);
```

Parameters

The following table lists the `Xil_Out8` function arguments.

Table 23: **Xil_Out8 Arguments**

Name	Description
Addr	contains the address to perform the output operation
Value	contains the 8 bit Value to be written at the specified address.

Returns

None.

Xil_Out16

Performs an output operation for a memory location by writing the 16 bit Value to the the specified address.

Prototype

```
INLINE void Xil_Out16(UINTPTR Addr, u16 Value);
```

Parameters

The following table lists the `Xil_Out16` function arguments.

Table 24: **Xil_Out16 Arguments**

Name	Description
Addr	contains the address to perform the output operation
Value	contains the Value to be written at the specified address.

Returns

None.

Xil_Out32

Performs an output operation for a memory location by writing the 32 bit Value to the the specified address.

Prototype

```
INLINE void Xil_Out32(UINTPTR Addr, u32 Value);
```

Parameters

The following table lists the `Xil_Out32` function arguments.

Table 25: **Xil_Out32 Arguments**

Name	Description
Addr	contains the address to perform the output operation
Value	contains the 32 bit Value to be written at the specified address.

Returns

None.

Xil_Out64

Performs an output operation for a memory location by writing the 64 bit Value to the the specified address.

Prototype

```
INLINE void Xil_Out64(UINTPTR Addr, u64 Value);
```

Parameters

The following table lists the `Xil_Out64` function arguments.

Table 26: **Xil_Out64 Arguments**

Name	Description
Addr	contains the address to perform the output operation
Value	contains 64 bit Value to be written at the specified address.

Returns

None.

Xil_SecureOut32

Performs an output operation for a memory location by writing the 32 bit Value to the the specified address and then reading it back to verify the value written in the register.

Prototype

```
INLINE u32 Xil_SecureOut32(UINTPTR Addr, u32 Value);
```

Parameters

The following table lists the `Xil_SecureOut32` function arguments.

Table 27: **Xil_SecureOut32 Arguments**

Name	Description
Addr	contains the address to perform the output operation
Value	contains 32 bit Value to be written at the specified address

Returns

Returns Status

- XST_SUCCESS on success
- XST_FAILURE on failure

Hardware Platform Information

This file contains information about hardware for which the code is built.

The xplatform_info.h file contains definitions for various available Xilinx platforms.

Also, it contains prototype of APIs, which can be used to get the platform information.

Table 28: **Quick Function Reference**

Type	Name	Arguments
u32	XGetPlatform_Info	void
u32	XGet_Zynq_UltraMp_Platform_info	void
u32	XGetPSVersion_Info	void

Functions

XGetPlatform_Info

This API is used to provide information about platform.

Prototype

```
u32 XGetPlatform_Info();
```

Returns

The information about platform defined in xplatform_info.h

XGet_Zynq_UltraMp_Platform_info

This API is used to provide information about zynq ultrascale MP platform.

Prototype

```
u32 XGet_Zynq_UltraMp_Platform_info();
```

Returns

The information about zynq ultrascale MP platform defined in xplatform_info.h

XGetPSVersion_Info

This API is used to provide information about PS Silicon version.

Prototype

```
u32 XGetPSVersion_Info();
```

Returns

The information about PS Silicon version.

Basic Data types for Xilinx Software IP

The xil_types.h file contains basic types for Xilinx software IP.

These data types are applicable for all processors supported by Xilinx.

Definitions

#Define XIL_COMPONENT_IS_READY

Description

In device drivers, This macro will be assigend to "IsReady" member of driver instance to indicate that driver instance is initialized and ready to use.

#Define XIL_COMPONENT_IS_STARTED

Description

In device drivers, This macro will be assigend to "IsStarted" member of driver instance to indicate that driver instance is started and it can be enabled.

#Define XUINT64_MSW

Description

Return the most significant half of the 64 bit data type.

Parameters

The following table lists the `XUINT64_MSW` function arguments.

Table 29: **XUINT64_MSW Arguments**

Name	Description
x	is the 64 bit word.

Returns

The upper 32 bits of the 64 bit word.

#Define XUINT64_LSW

Description

Return the least significant half of the 64 bit data type.

Parameters

The following table lists the `XUINT64_LSW` function arguments.

Table 30: **XUINT64_LSW Arguments**

Name	Description
x	is the 64 bit word.

Returns

The lower 32 bits of the 64 bit word.

#Define UPPER_32_BITS

Description

Returns 32-63 bits of a number.

Note: A basic shift-right of a 64- or 32-bit quantity. Use this to suppress the "right shift count >= width of type" warning when that quantity is 32-bits.

Parameters

The following table lists the `UPPER_32_BITS` function arguments.

Table 31: **UPPER_32_BITS Arguments**

Name	Description
n	: Number being accessed.

Returns

Bits 32-63 of number.

#Define LOWER_32_BITS

Description

Returns 0-31 bits of a number.

Parameters

The following table lists the `LOWER_32_BITS` function arguments.

Table 32: **LOWER_32_BITS Arguments**

Name	Description
n	: Number being accessed.

Returns

Bits 0-31 of number

Customized APIs for Memory Operations

The xil_mem.h file contains prototype for functions related to memory operations.

These APIs are applicable for all processors supported by Xilinx.

Table 33: **Quick Function Reference**

Type	Name	Arguments
void	Xil_MemCpy	void * dst const void * src u32 cnt

Functions

Xil_MemCpy

This function copies memory from once location to other.

Prototype

```
void Xil_MemCpy(void *dst, const void *src, u32 cnt);
```

Parameters

The following table lists the `Xil_MemCpy` function arguments.

Table 34: **Xil_MemCpy Arguments**

Name	Description
dst	pointer pointing to destination memory
src	pointer pointing to source memory
cnt	32 bit length of bytes to be copied

Xilinx software status codes

The xstatus.h file contains the Xilinx software status codes.These codes are used throughout the Xilinx device drivers.

Test Utilities for Memory and Caches

The xil_testcache.h, xil_testio.h and the xil_testmem.h files contain utility functions to test cache and memory.

Details of supported tests and subtests are listed below.

- Cache test: xil_testcache.h contains utility functions to test cache.
- I/O test: The Xil_testio.h file contains endian related memory IO functions. A subset of the memory tests can be selected or all of the tests can be run in order. If there is an error detected by a subtest, the test stops and the failure code is returned. Further tests are not run even if all of the tests are selected.
- Memory test: The xil_testmem.h file contains utility functions to test memory. A subset of the memory tests can be selected or all of the tests can be run in order. If there is an error detected by a subtest, the test stops and the failure code is returned. Further tests are not run even if all of the tests are selected.

Following list describes the supported memory tests:

- XIL_TESTMEM_ALLMEMTESTS: This test runs all of the subtests.
- XIL_TESTMEM_INCREMENT: This test starts at 'XIL_TESTMEM_INIT_VALUE' and uses the incrementing value as the test value for memory.
- XIL_TESTMEM_WALKONES: Also known as the Walking ones test. This test uses a walking '1' as the test value for memory.

```
location 1 = 0x00000001
location 2 = 0x00000002
...
```

- XIL_TESTMEM_WALKZEROS: Also known as the Walking zero's test. This test uses the inverse value of the walking ones test as the test value for memory.

```
location 1 = 0xFFFFFFFE
location 2 = 0xFFFFFFFD
...
```

- XIL_TESTMEM_INVERSEADDR: Also known as the inverse address test. This test uses the inverse of the address of the location under test as the test value for memory.
- XIL_TESTMEM_FIXEDPATTERN: Also known as the fixed pattern test. This test uses the provided patters as the test value for memory. If zero is provided as the pattern the test uses '0xDEADBEEF".

CAUTION! *The tests are **DESTRUCTIVE**. Run before any initialized memory spaces have been set up. The address provided to the memory tests is not checked for validity except for the NULL case. It is possible to provide a code-space pointer for this test to start with and ultimately destroy executable code causing random failures.*

***Note*:** Used for spaces where the address range of the region is smaller than the data width. If the memory range is greater than 2 ** width, the patterns used in XIL_TESTMEM_WALKONES and XIL_TESTMEM_WALKZEROS will repeat on a boundary of a power of two making it more difficult to detect addressing errors. The XIL_TESTMEM_INCREMENT and XIL_TESTMEM_INVERSEADDR tests suffer the same problem. Ideally, if large blocks of memory are to be tested, break them up into smaller regions of memory to allow the test patterns used not to repeat over the region tested.

Table 35: **Quick Function Reference**

Type	Name	Arguments
s32	Xil_TestMem32	u32 * Addr u32 Words u32 Pattern u8 Subtest
s32	Xil_TestMem16	u16 * Addr u32 Words u16 Pattern u8 Subtest

Table 35: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
s32	Xil_TestMem8	u8 * Addr u32 Words u8 Pattern u8 Subtest
u32	RotateLeft	u32 Input u8 Width
u32	RotateRight	u32 Input u8 Width
s32	Xil_TestDCacheRange	void
s32	Xil_TestDCacheAll	void
s32	Xil_TestICacheRange	void
s32	Xil_TestICacheAll	void
s32	Xil_TestIO8	u8 * Addr s32 Length u8 Value
s32	Xil_TestIO16	u16 * Addr s32 Length u16 Value s32 Kind s32 Swap
s32	Xil_TestIO32	u32 * Addr s32 Length u32 Value s32 Kind s32 Swap

Functions

Xil_TestMem32

Perform a destructive 32-bit wide memory test.

***Note*:** Used for spaces where the address range of the region is smaller than the data width. If the memory range is greater than 2 ** Width, the patterns used in XIL_TESTMEM_WALKONES and XIL_TESTMEM_WALKZEROS will repeat on a boundary of a power of two making it more difficult to detect addressing errors. The XIL_TESTMEM_INCREMENT and XIL_TESTMEM_INVERSEADDR tests suffer the same problem. Ideally, if large blocks of memory are to be tested, break them up into smaller regions of memory to allow the test patterns used not to repeat over the region tested.

Prototype

```
s32 Xil_TestMem32(u32 *Addr, u32 Words, u32 Pattern, u8 Subtest);
```

Parameters

The following table lists the `Xil_TestMem32` function arguments.

Table 36: **Xil_TestMem32 Arguments**

Name	Description
Addr	pointer to the region of memory to be tested.
Words	length of the block.
Pattern	constant used for the constant pattern test, if 0, 0xDEADBEEF is used.
Subtest	test type selected. See xil_testmem.h for possible values.

Returns

- 0 is returned for a pass
- 1 is returned for a failure

Xil_TestMem16

Perform a destructive 16-bit wide memory test.

***Note*:** Used for spaces where the address range of the region is smaller than the data width. If the memory range is greater than 2 ** Width, the patterns used in XIL_TESTMEM_WALKONES and XIL_TESTMEM_WALKZEROS will repeat on a boundary of a power of two making it more difficult to detect addressing errors. The XIL_TESTMEM_INCREMENT and XIL_TESTMEM_INVERSEADDR tests suffer the same problem. Ideally, if large blocks of memory are to be tested, break them up into smaller regions of memory to allow the test patterns used not to repeat over the region tested.

Prototype

```
s32 Xil_TestMem16(u16 *Addr, u32 Words, u16 Pattern, u8 Subtest);
```

Parameters

The following table lists the `Xil_TestMem16` function arguments.

Table 37: **Xil_TestMem16 Arguments**

Name	Description
Addr	pointer to the region of memory to be tested.
Words	length of the block.
Pattern	constant used for the constant Pattern test, if 0, 0xDEADBEEF is used.
Subtest	type of test selected. See xil_testmem.h for possible values.

Returns

Xil_TestMem8

Perform a destructive 8-bit wide memory test.

***Note*:** Used for spaces where the address range of the region is smaller than the data width. If the memory range is greater than 2 ** Width, the patterns used in XIL_TESTMEM_WALKONES and XIL_TESTMEM_WALKZEROS will repeat on a boundary of a power of two making it more difficult to detect addressing errors. The XIL_TESTMEM_INCREMENT and XIL_TESTMEM_INVERSEADDR tests suffer the same problem. Ideally, if large blocks of memory are to be tested, break them up into smaller regions of memory to allow the test patterns used not to repeat over the region tested.

Prototype

```
s32 Xil_TestMem8(u8 *Addr, u32 Words, u8 Pattern, u8 Subtest);
```

Parameters

The following table lists the `Xil_TestMem8` function arguments.

Table 38: **Xil_TestMem8 Arguments**

Name	Description
Addr	pointer to the region of memory to be tested.
Words	length of the block.
Pattern	constant used for the constant pattern test, if 0, 0xDEADBEEF is used.
Subtest	type of test selected. See xil_testmem.h for possible values.

Returns

- -1 is returned for a failure
- 0 is returned for a pass

RotateLeft

Rotates the provided value to the left one bit position.

Prototype

```
u32 RotateLeft(u32 Input, u8 Width);
```

Parameters

The following table lists the `RotateLeft` function arguments.

Table 39: **RotateLeft Arguments**

Name	Description
Input	is value to be rotated to the left
Width	is the number of bits in the input data

Returns

The resulting unsigned long value of the rotate left

RotateRight

Rotates the provided value to the right one bit position.

Prototype

```
u32 RotateRight(u32 Input, u8 Width);
```

Parameters

The following table lists the `RotateRight` function arguments.

Table 40: **RotateRight Arguments**

Name	Description
Input	value to be rotated to the right
Width	number of bits in the input data

Returns

The resulting u32 value of the rotate right

Xil_TestDCacheRange

Perform DCache range related API test such as Xil_DCacheFlushRange and Xil_DCacheInvalidateRange.

This test function writes a constant value to the Data array, flushes the range, writes a new value, then invalidates the corresponding range.

Prototype

```
s32 Xil_TestDCacheRange(void);
```

Returns

- -1 is returned for a failure
- 0 is returned for a pass

Xil_TestDCacheAll

Perform DCache all related API test such as Xil_DCacheFlush and Xil_DCacheInvalidate.

This test function writes a constant value to the Data array, flushes the DCache, writes a new value, then invalidates the DCache.

Prototype

```
s32 Xil_TestDCacheAll(void);
```

Returns

- 0 is returned for a pass
- -1 is returned for a failure

Xil_TestICacheRange

Perform Xil_ICacheInvalidateRange() on a few function pointers.

***Note*:** The function will hang if it fails.

Prototype

```
s32 Xil_TestICacheRange(void);
```

Returns

- 0 is returned for a pass

Xil_TestICacheAll

Perform Xil_ICacheInvalidate() on a few function pointers.

***Note*:** The function will hang if it fails.

Prototype

```
s32 Xil_TestICacheAll(void);
```

Returns

- 0 is returned for a pass

Xil_TestIO8

Perform a destructive 8-bit wide register IO test where the register is accessed using Xil_Out8 and Xil_In8, and comparing the written values by reading them back.

Prototype

```
s32 Xil_TestIO8(u8 *Addr, s32 Length, u8 Value);
```

Parameters

The following table lists the `Xil_TestIO8` function arguments.

Table 41: **Xil_TestIO8 Arguments**

Name	Description
Addr	a pointer to the region of memory to be tested.
Length	Length of the block.
Value	constant used for writing the memory.

Returns

- -1 is returned for a failure
- 0 is returned for a pass

Xil_TestIO16

Perform a destructive 16-bit wide register IO test.

Each location is tested by sequentially writing a 16-bit wide register, reading the register, and comparing value. This function tests three kinds of register IO functions, normal register IO, little-endian register IO, and big-endian register IO. When testing little/big-endian IO, the function performs the following sequence, Xil_Out16LE/Xil_Out16BE, Xil_In16, Compare In-Out values, Xil_Out16, Xil_In16LE/Xil_In16BE, Compare In-Out values. Whether to swap the read-in value before comparing is controlled by the 5th argument.

Prototype

```
s32 Xil_TestIO16(u16 *Addr, s32 Length, u16 Value, s32 Kind, s32 Swap);
```

Parameters

The following table lists the `Xil_TestIO16` function arguments.

Table 42: **Xil_TestIO16 Arguments**

Name	Description
Addr	a pointer to the region of memory to be tested.
Length	Length of the block.
Value	constant used for writing the memory.
Kind	Type of test. Acceptable values are: XIL_TESTIO_DEFAULT, XIL_TESTIO_LE, XIL_TESTIO_BE.
Swap	indicates whether to byte swap the read-in value.

Returns

- -1 is returned for a failure
- 0 is returned for a pass

Xil_TestIO32

Perform a destructive 32-bit wide register IO test.

Each location is tested by sequentially writing a 32-bit wide register, reading the register, and comparing value. This function tests three kinds of register IO functions, normal register IO, little-endian register IO, and big-endian register IO. When testing little/big-endian IO, the function perform the following sequence, Xil_Out32LE/ Xil_Out32BE, Xil_In32, Compare, Xil_Out32, Xil_In32LE/Xil_In32BE, Compare. Whether to swap the read-in value *before comparing is controlled by the 5th argument.

Prototype

```
s32 Xil_TestIO32(u32 *Addr, s32 Length, u32 Value, s32 Kind, s32 Swap);
```

Parameters

The following table lists the `Xil_TestIO32` function arguments.

Table 43: **Xil_TestIO32 Arguments**

Name	Description
Addr	a pointer to the region of memory to be tested.
Length	Length of the block.
Value	constant used for writing the memory.
Kind	type of test. Acceptable values are: XIL_TESTIO_DEFAULT, XIL_TESTIO_LE, XIL_TESTIO_BE.
Swap	indicates whether to byte swap the read-in value.

Returns

- -1 is returned for a failure
- 0 is returned for a pass

Microblaze Processor API

Microblaze Processor API

This section provides a linked summary and detailed descriptions of the Microblaze Processor APIs.

Microblaze Pseudo-asm Macros and Interrupt Handling APIs

Microblaze BSP includes macros to provide convenient access to various registers in the MicroBlaze processor.

Some of these macros are very useful within exception handlers for retrieving information about the exception.Also, the interrupt handling functions help manage interrupt handling on MicroBlaze processor devices.To use these functions, include the header file mb_interface.h in your source code

Table 44: **Quick Function Reference**

Type	Name	Arguments
void	microblaze_register_handler	XInterruptHandler Handler void * DataPtr
void	microblaze_register_exception_handler	u32 ExceptionId Top void * DataPtr

Functions

microblaze_register_handler

Registers a top-level interrupt handler for the MicroBlaze.

The argument provided in this call as the DataPtr is used as the argument for the handler when it is called.

Prototype

```
void microblaze_register_handler(XInterruptHandler Handler, void *DataPtr);
```

Parameters

The following table lists the `microblaze_register_handler` function arguments.

Table 45: **microblaze_register_handler Arguments**

Name	Description
Handler	Top level handler.
DataPtr	a reference to data that will be passed to the handler when it gets called.

Returns

None.

microblaze_register_exception_handler

Registers an exception handler for the MicroBlaze.

The argument provided in this call as the DataPtr is used as the argument for the handler when it is called.

Prototype

```
void microblaze_register_exception_handler(u32 ExceptionId,
Xil_ExceptionHandler Handler, void *DataPtr);
```

Parameters

The following table lists the `microblaze_register_exception_handler` function arguments.

Table 46: **microblaze_register_exception_handler Arguments**

Name	Description
ExceptionId	is the id of the exception to register this handler for.
Top	level handler.
DataPtr	is a reference to data that will be passed to the handler when it gets called.

Returns

None.

Definitions

#Define mfgpr

Description

Return value from the general purpose register (GPR) rn.

Parameters

The following table lists the `mfgpr` function arguments.

Table 47: **mfgpr Arguments**

Name	Description
rn	General purpose register to be read.

#Define mfmsr

Description

Return the current value of the MSR.

Parameters

The following table lists the `mfmsr` function arguments.

Table 48: **mfmsr Arguments**

Name	Description
None	

#Define mfear

Description

Return the current value of the Exception Address Register (EAR).

Parameters

The following table lists the `mfear` function arguments.

Table 49: **mfear Arguments**

Name	Description
None	

#Define mfeare

Description

#Define mfesr

Description

Return the current value of the Exception Status Register (ESR).

Parameters

The following table lists the `mfesr` function arguments.

Table 50: **mfesr Arguments**

Name	Description
None	

#Define mffsr

Description

Return the current value of the Floating Point Status (FPS).

Parameters

The following table lists the `mffsr` function arguments.

Table 51: **mffsr Arguments**

Name	Description
None	

MicroBlaze Exception APIs

The xil_exception.h file, available in the <install-directory>/src/microblaze folder, contains Microblaze specific exception related APIs and macros.

Application programs can use these APIs for various exception related operations. For example, enable exception, disable exception, register exception hander.

***Note*:** To use exception related functions, xil_exception.h must be added in source code

Table 52: **Quick Function Reference**

Type	Name	Arguments
void	Xil_ExceptionNullHandler	void * Data
void	Xil_ExceptionInit	void
void	Xil_ExceptionEnable	void
void	Xil_ExceptionDisable	void
void	Xil_ExceptionRegisterHandler	u32 Id Xil_ExceptionHandler Handler void * Data
void	Xil_ExceptionRemoveHandler	u32 Id

Functions

Xil_ExceptionNullHandler

This function is a stub handler that is the default handler that gets called if the application has not setup a handler for a specific exception.

The function interface has to match the interface specified for a handler even though none of the arguments are used.

Prototype

```
void Xil_ExceptionNullHandler(void *Data);
```

Parameters

The following table lists the `Xil_ExceptionNullHandler` function arguments.

Table 53: **Xil_ExceptionNullHandler Arguments**

Name	Description
Data	unused by this function.

Xil_ExceptionInit

Initialize exception handling for the processor.

The exception vector table is setup with the stub handler for all exceptions.

Prototype

```
void Xil_ExceptionInit(void);
```

Returns

None.

Xil_ExceptionEnable

Enable Exceptions.

Prototype

```
void Xil_ExceptionEnable(void);
```

Returns

None.

Xil_ExceptionDisable

Disable Exceptions.

Prototype

```
void Xil_ExceptionDisable(void);
```

Returns

None.

Xil_ExceptionRegisterHandler

Makes the connection between the Id of the exception source and the associated handler that is to run when the exception is recognized.

The argument provided in this call as the DataPtr is used as the argument for the handler when it is called.

Prototype

```
void Xil_ExceptionRegisterHandler(u32 Id, Xil_ExceptionHandler Handler,
void *Data);
```

Parameters

The following table lists the `Xil_ExceptionRegisterHandler` function arguments.

Table 54: **Xil_ExceptionRegisterHandler Arguments**

Name	Description
Id	contains the 32 bit ID of the exception source and should be XIL_EXCEPTION_INT or be in the range of 0 to XIL_EXCEPTION_LAST. See xil_mach_exception.h for further information.
Handler	handler function to be registered for exception
Data	a reference to data that will be passed to the handler when it gets called.

Xil_ExceptionRemoveHandler

Removes the handler for a specific exception Id.

The stub handler is then registered for this exception Id.

Prototype

```
void Xil_ExceptionRemoveHandler(u32 Id);
```

Parameters

The following table lists the `Xil_ExceptionRemoveHandler` function arguments.

Table 55: **Xil_ExceptionRemoveHandler Arguments**

Name	Description
Id	contains the 32 bit ID of the exception source and should be XIL_EXCEPTION_INT or in the range of 0 to XIL_EXCEPTION_LAST. See xexception_l.h for further information.

MicroBlaze Cache APIs

This contains implementation of cache related driver functions.

The xil_cache.h file contains cache related driver functions (or macros) that can be used to access the device.

The user should refer to the hardware device specification for more details of the device operation. The functions in this header file can be used across all Xilinx supported processors.

Table 56: **Quick Function Reference**

Type	Name	Arguments
void	Xil_DCacheDisable	void
void	Xil_ICacheDisable	void

Functions

Xil_DCacheDisable

Disable the data cache.

Prototype

```
void Xil_DCacheDisable(void);
```

Returns

None.

Xil_ICacheDisable

Disable the instruction cache.

Prototype

```
void Xil_ICacheDisable(void);
```

Returns

None.

Definitions

#Define Xil_L1DCacheInvalidate

Description

Invalidate the entire L1 data cache.

If the cacheline is modified (dirty), the modified contents are lost.

Note: Processor must be in real mode.

#Define Xil_L2CacheInvalidate

Description

Invalidate the entire L2 data cache.

If the cacheline is modified (dirty),the modified contents are lost.

Note: Processor must be in real mode.

#Define Xil_L1DCacheInvalidateRange

Description

Invalidate the L1 data cache for the given address range.

If the bytes specified by the address (Addr) are cached by the L1 data cache, the cacheline containing that byte is invalidated.If the cacheline is modified (dirty), the modified contents are lost.

***Note*:** Processor must be in real mode.

Parameters

The following table lists the `Xil_L1DCacheInvalidateRange` function arguments.

Table 57: **Xil_L1DCacheInvalidateRange Arguments**

Name	Description
Addr	is address of range to be invalidated.
Len	is the length in bytes to be invalidated.

#Define Xil_L2CacheInvalidateRange

Description

Invalidate the L1 data cache for the given address range.

If the bytes specified by the address (Addr) are cached by the L1 data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the modified contents are lost.

***Note*:** Processor must be in real mode.

Parameters

The following table lists the `Xil_L2CacheInvalidateRange` function arguments.

Table 58: **Xil_L2CacheInvalidateRange Arguments**

Name	Description
Addr	address of range to be invalidated.
Len	length in bytes to be invalidated.

#Define Xil_L1DCacheFlushRange

Description

Flush the L1 data cache for the given address range.

If the bytes specified by the address (Addr) are cached by the data cache, and is modified (dirty), the cacheline will be written to system memory.The cacheline will also be invalidated.

Parameters

The following table lists the `Xil_L1DCacheFlushRange` function arguments.

Table 59: **Xil_L1DCacheFlushRange Arguments**

Name	Description
Addr	the starting address of the range to be flushed.
Len	length in byte to be flushed.

#Define Xil_L2CacheFlushRange

Description

Flush the L2 data cache for the given address range.

If the bytes specified by the address (Addr) are cached by the data cache, and is modified (dirty), the cacheline will be written to system memory. The cacheline will also be invalidated.

Parameters

The following table lists the `Xil_L2CacheFlushRange` function arguments.

Table 60: **Xil_L2CacheFlushRange Arguments**

Name	Description
Addr	the starting address of the range to be flushed.
Len	length in byte to be flushed.

#Define Xil_L1DCacheFlush

Description

Flush the entire L1 data cache.

If any cacheline is dirty, the cacheline will be written to system memory. The entire data cache will be invalidated.

#Define Xil_L2CacheFlush

Description

Flush the entire L2 data cache.

If any cacheline is dirty, the cacheline will be written to system memory. The entire data cache will be invalidated.

#Define Xil_L1ICacheInvalidateRange

Description

Invalidate the instruction cache for the given address range.

Parameters

The following table lists the `Xil_L1ICacheInvalidateRange` function arguments.

Table 61: **Xil_L1ICacheInvalidateRange Arguments**

Name	Description
Addr	is address of ragne to be invalidated.
Len	is the length in bytes to be invalidated.

#Define Xil_L1ICacheInvalidate

Description

Invalidate the entire instruction cache.

#Define Xil_L1DCacheEnable

Description

Enable the L1 data cache.

Note: This is processor specific.

#Define Xil_L1DCacheDisable

Description

Disable the L1 data cache.

Note: This is processor specific.

#Define Xil_L1ICacheEnable

Description

Enable the instruction cache.

Note: This is processor specific.

#Define Xil_L1ICacheDisable

Description

Disable the L1 Instruction cache.

Note: This is processor specific.

#Define Xil_DCacheEnable

Description

Enable the data cache.

#Define Xil_ICacheEnable

Description

Enable the instruction cache.

#Define Xil_DCacheInvalidate

Description

Invalidate the entire Data cache.

#Define Xil_DCacheInvalidateRange

Description

Invalidate the Data cache for the given address range.

If the bytes specified by the address (adr) are cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the modified contents are lost and are NOT written to system memory before the line is invalidated.

Parameters

The following table lists the `Xil_DCacheInvalidateRange` function arguments.

Table 62: **Xil_DCacheInvalidateRange Arguments**

Name	Description
Addr	Start address of range to be invalidated.
Len	Length of range to be invalidated in bytes.

#Define Xil_DCacheFlush

Description

Flush the entire Data cache.

#Define Xil_DCacheFlushRange

Description

Flush the Data cache for the given address range.

If the bytes specified by the address (adr) are cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the written to system memory first before the before the line is invalidated.

Parameters

The following table lists the `Xil_DCacheFlushRange` function arguments.

Table 63: **Xil_DCacheFlushRange Arguments**

Name	Description
Addr	Start address of range to be flushed.
Len	Length of range to be flushed in bytes.

#Define Xil_ICacheInvalidate

Description

Invalidate the entire instruction cache.

MicroBlaze Processor FSL Macros

Microblaze BSP includes macros to provide convenient access to accelerators connected to the MicroBlaze Fast Simplex Link (FSL) Interfaces.To use these functions, include the header file fsl.h in your source code.

Definitions

#Define getfslx

Description

Performs a get function on an input FSL of the MicroBlaze processor.

Parameters

The following table lists the `getfslx` function arguments.

Table 64: **getfslx Arguments**

Name	Description
val	variable to sink data from get function
id	literal in the range of 0 to 7 (0 to 15 for MicroBlaze v7.00.a and later)
flags	valid FSL macro flags

#Define putfslx

Description

Performs a put function on an input FSL of the MicroBlaze processor.

Parameters

The following table lists the `putfslx` function arguments.

Table 65: **putfslx Arguments**

Name	Description
val	variable to source data to put function
id	literal in the range of 0 to 7 (0 to 15 for MicroBlaze v7.00.a and later)
flags	valid FSL macro flags

#Define tgetfslx

Description

Performs a test get function on an input FSL of the MicroBlaze processor.

Parameters

The following table lists the `tgetfslx` function arguments.

Table 66: **tgetfslx Arguments**

Name	Description
val	variable to sink data from get function
id	literal in the range of 0 to 7 (0 to 15 for MicroBlaze v7.00.a and later)
flags	valid FSL macro flags

#Define tputfslx

Description

Performs a put function on an input FSL of the MicroBlaze processor.

Parameters

The following table lists the `tputfslx` function arguments.

Table 67: **tputfslx Arguments**

Name	Description
id	FSL identifier
flags	valid FSL macro flags

#Define getdfslx

Description

Performs a getd function on an input FSL of the MicroBlaze processor.

Parameters

The following table lists the `getdfslx` function arguments.

Table 68: **getdfslx Arguments**

Name	Description
val	variable to sink data from getd function
var	literal in the range of 0 to 7 (0 to 15 for MicroBlaze v7.00.a and later)
flags	valid FSL macro flags

#Define putdfslx

Description

Performs a putd function on an input FSL of the MicroBlaze processor.

Parameters

The following table lists the `putdfslx` function arguments.

Table 69: **putdfslx Arguments**

Name	Description
val	variable to source data to putd function
var	literal in the range of 0 to 7 (0 to 15 for MicroBlaze v7.00.a and later)
flags	valid FSL macro flags

#Define tgetdfslx

Description

Performs a test getd function on an input FSL of the MicroBlaze processor;.

Parameters

The following table lists the `tgetdfslx` function arguments.

Table 70: **tgetdfslx Arguments**

Name	Description
val	variable to sink data from getd function
var	literal in the range of 0 to 7 (0 to 15 for MicroBlaze v7.00.a and later)
flags	valid FSL macro flags

#Define tputdfslx

Description

Performs a put function on an input FSL of the MicroBlaze processor.

Parameters

The following table lists the `tputdfslx` function arguments.

Table 71: **tputdfslx Arguments**

Name	Description
var	FSL identifier
flags	valid FSL macro flags

Microblaze PVR access routines and macros

MicroBlaze processor v5.00.a and later versions have configurable Processor Version Registers (PVRs).

The contents of the PVR are captured using the pvr_t data structure, which is defined as an array of 32-bit words, with each word corresponding to a PVR register on hardware. The number of PVR words is determined by the number of PVRs configured in the hardware. You should not attempt to access PVR registers that are not present in hardware, as the pvr_t data structure is resized to hold only as many PVRs as are present in hardware. To access information in the PVR:

1. Use the microblaze_get_pvr() function to populate the PVR data into a pvr_t data structure.
2. In subsequent steps, you can use any one of the PVR access macros list to get individual data stored in the PVR.
3. pvr.h header file must be included to source to use PVR macros.

Definitions

#Define MICROBLAZE_PVR_IS_FULL

Description

Return non-zero integer if PVR is of type FULL, 0 if basic.

Parameters

The following table lists the `MICROBLAZE_PVR_IS_FULL` function arguments.

Table 72: **MICROBLAZE_PVR_IS_FULL Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_BARREL

Description

Return non-zero integer if hardware barrel shifter present.

Parameters

The following table lists the `MICROBLAZE_PVR_USE_BARREL` function arguments.

Table 73: **MICROBLAZE_PVR_USE_BARREL Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_DIV

Description

Return non-zero integer if hardware divider present.

Parameters

The following table lists the `MICROBLAZE_PVR_USE_DIV` function arguments.

Table 74: **MICROBLAZE_PVR_USE_DIV Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_HW_MUL

Description

Return non-zero integer if hardware multiplier present.

Parameters

The following table lists the `MICROBLAZE_PVR_USE_HW_MUL` function arguments.

Table 75: **MICROBLAZE_PVR_USE_HW_MUL Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_FPU

Description

Return non-zero integer if hardware floating point unit (FPU) present.

Parameters

The following table lists the `MICROBLAZE_PVR_USE_FPU` function arguments.

Table 76: **MICROBLAZE_PVR_USE_FPU Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_ICACHE

Description

Return non-zero integer if I-cache present.

Parameters

The following table lists the `MICROBLAZE_PVR_USE_ICACHE` function arguments.

Table 77: **MICROBLAZE_PVR_USE_ICACHE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_DCACHE

Description

Return non-zero integer if D-cache present.

Parameters

The following table lists the `MICROBLAZE_PVR_USE_DCACHE` function arguments.

Table 78: **MICROBLAZE_PVR_USE_DCACHE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_MICROBLAZE_VERSION

Description

Return MicroBlaze processor version encoding.

Refer to the MicroBlaze Processor Reference Guide (UG081) for mappings from encodings to actual hardware versions.

Parameters

The following table lists the `MICROBLAZE_PVR_MICROBLAZE_VERSION` function arguments.

Table 79: **MICROBLAZE_PVR_MICROBLAZE_VERSION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USER1

Description

Return the USER1 field stored in the PVR.

Parameters

The following table lists the `MICROBLAZE_PVR_USER1` function arguments.

Table 80: **MICROBLAZE_PVR_USER1 Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USER2

Description

Return the USER2 field stored in the PVR.

Parameters

The following table lists the `MICROBLAZE_PVR_USER2` function arguments.

Table 81: **MICROBLAZE_PVR_USER2 Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_D_AXI

Description

#Define MICROBLAZE_PVR_D_LMB

Description

Return non-zero integer if Data Side PLB interface is present.

Parameters

The following table lists the `MICROBLAZE_PVR_D_LMB` function arguments.

Table 82: **MICROBLAZE_PVR_D_LMB Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_D_PLB

Description

Return non-zero integer if Data Side PLB interface is present.

Parameters

The following table lists the `MICROBLAZE_PVR_D_PLB` function arguments.

Table 83: **MICROBLAZE_PVR_D_PLB Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_I_LMB

Description

Return non-zero integer if Instruction Side Local Memory Bus (LMB) interface present.

Parameters

The following table lists the `MICROBLAZE_PVR_I_LMB` function arguments.

Table 84: **MICROBLAZE_PVR_I_LMB Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_I_PLB

Description

Return non-zero integer if Instruction Side PLB interface present.

Parameters

The following table lists the `MICROBLAZE_PVR_I_PLB` function arguments.

Table 85: **MICROBLAZE_PVR_I_PLB Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_INTERRUPT_IS_EDGE

Description

Return non-zero integer if interrupts are configured as edge-triggered.

Parameters

The following table lists the `MICROBLAZE_PVR_INTERRUPT_IS_EDGE` function arguments.

Table 86: **MICROBLAZE_PVR_INTERRUPT_IS_EDGE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_EDGE_IS_POSITIVE

Description

Return non-zero integer if interrupts are configured as positive edge triggered.

Parameters

The following table lists the `MICROBLAZE_PVR_EDGE_IS_POSITIVE` function arguments.

Table 87: **MICROBLAZE_PVR_EDGE_IS_POSITIVE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_INTERCONNECT

Description

Return non-zero if MicroBlaze processor has PLB interconnect; otherwise return zero.

Parameters

The following table lists the `MICROBLAZE_PVR_INTERCONNECT` function arguments.

Table 88: **MICROBLAZE_PVR_INTERCONNECT Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_MUL64

Description

Return non-zero integer if MicroBlaze processor supports 64-bit products for multiplies.

Parameters

The following table lists the `MICROBLAZE_PVR_USE_MUL64` function arguments.

Table 89: **MICROBLAZE_PVR_USE_MUL64 Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_OPCODE_0x0_ILLEGAL

Description

Return non-zero integer if opcode 0x0 is treated as an illegal opcode.

multiplies.

Parameters

The following table lists the `MICROBLAZE_PVR_OPCODE_0x0_ILLEGAL` function arguments.

Table 90: **MICROBLAZE_PVR_OPCODE_0x0_ILLEGAL Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_UNALIGNED_EXCEPTION

Description

Return non-zero integer if unaligned exceptions are supported.

Parameters

The following table lists the `MICROBLAZE_PVR_UNALIGNED_EXCEPTION` function arguments.

Table 91: **MICROBLAZE_PVR_UNALIGNED_EXCEPTION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_ILL_OPCODE_EXCEPTION

Description

Return non-zero integer if illegal opcode exceptions are supported.

Parameters

The following table lists the `MICROBLAZE_PVR_ILL_OPCODE_EXCEPTION` function arguments.

Table 92: **MICROBLAZE_PVR_ILL_OPCODE_EXCEPTION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_IPLB_BUS_EXCEPTION

Description

Return non-zero integer if I-PLB exceptions are supported.

Parameters

The following table lists the `MICROBLAZE_PVR_IPLB_BUS_EXCEPTION` function arguments.

Table 93: **MICROBLAZE_PVR_IPLB_BUS_EXCEPTION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DPLB_BUS_EXCEPTION

Description

Return non-zero integer if I-PLB exceptions are supported.

Parameters

The following table lists the `MICROBLAZE_PVR_DPLB_BUS_EXCEPTION` function arguments.

Table 94: **MICROBLAZE_PVR_DPLB_BUS_EXCEPTION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DIV_ZERO_EXCEPTION

Description

Return non-zero integer if divide by zero exceptions are supported.

Parameters

The following table lists the `MICROBLAZE_PVR_DIV_ZERO_EXCEPTION` function arguments.

Table 95: **MICROBLAZE_PVR_DIV_ZERO_EXCEPTION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_FPU_EXCEPTION

Description

Return non-zero integer if FPU exceptions are supported.

Parameters

The following table lists the `MICROBLAZE_PVR_FPU_EXCEPTION` function arguments.

Table 96: **MICROBLAZE_PVR_FPU_EXCEPTION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_FSL_EXCEPTION

Description

Return non-zero integer if FSL exceptions are present.

Parameters

The following table lists the `MICROBLAZE_PVR_FSL_EXCEPTION` function arguments.

Table 97: **MICROBLAZE_PVR_FSL_EXCEPTION Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DEBUG_ENABLED

Description

Return non-zero integer if debug is enabled.

Parameters

The following table lists the `MICROBLAZE_PVR_DEBUG_ENABLED` function arguments.

Table 98: **MICROBLAZE_PVR_DEBUG_ENABLED Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_NUMBER_OF_PC_BRK

Description

Return the number of hardware PC breakpoints available.

Parameters

The following table lists the `MICROBLAZE_PVR_NUMBER_OF_PC_BRK` function arguments.

Table 99: **MICROBLAZE_PVR_NUMBER_OF_PC_BRK Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_NUMBER_OF_RD_ADDR_BRK

Description

Return the number of read address hardware watchpoints supported.

Parameters

The following table lists the `MICROBLAZE_PVR_NUMBER_OF_RD_ADDR_BRK` function arguments.

Table 100: **MICROBLAZE_PVR_NUMBER_OF_RD_ADDR_BRK Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_NUMBER_OF_WR_ADDR_BRK

Description

Return the number of write address hardware watchpoints supported.

Parameters

The following table lists the `MICROBLAZE_PVR_NUMBER_OF_WR_ADDR_BRK` function arguments.

Table 101: **MICROBLAZE_PVR_NUMBER_OF_WR_ADDR_BRK Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_FSL_LINKS

Description

Return the number of FSL links present.

Parameters

The following table lists the `MICROBLAZE_PVR_FSL_LINKS` function arguments.

Table 102: **MICROBLAZE_PVR_FSL_LINKS Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_ICACHE_ADDR_TAG_BITS

Description

Return the number of address tag bits for the I-cache.

Parameters

The following table lists the `MICROBLAZE_PVR_ICACHE_ADDR_TAG_BITS` function arguments.

Table 103: **MICROBLAZE_PVR_ICACHE_ADDR_TAG_BITS Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_ICACHE_ALLOW_WR

Description

Return non-zero if writes to I-caches are allowed.

Parameters

The following table lists the `MICROBLAZE_PVR_ICACHE_ALLOW_WR` function arguments.

Table 104: **MICROBLAZE_PVR_ICACHE_ALLOW_WR Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_ICACHE_LINE_LEN

Description

Return the length of each I-cache line in bytes.

Parameters

The following table lists the `MICROBLAZE_PVR_ICACHE_LINE_LEN` function arguments.

Table 105: **MICROBLAZE_PVR_ICACHE_LINE_LEN Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_ICACHE_BYTE_SIZE

Description

Return the size of the D-cache in bytes.

Parameters

The following table lists the `MICROBLAZE_PVR_ICACHE_BYTE_SIZE` function arguments.

Table 106: **MICROBLAZE_PVR_ICACHE_BYTE_SIZE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DCACHE_ADDR_TAG_BITS

Description

Return the number of address tag bits for the D-cache.

Parameters

The following table lists the `MICROBLAZE_PVR_DCACHE_ADDR_TAG_BITS` function arguments.

Table 107: **MICROBLAZE_PVR_DCACHE_ADDR_TAG_BITS Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DCACHE_ALLOW_WR

Description

Return non-zero if writes to D-cache are allowed.

Parameters

The following table lists the `MICROBLAZE_PVR_DCACHE_ALLOW_WR` function arguments.

Table 108: **MICROBLAZE_PVR_DCACHE_ALLOW_WR Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DCACHE_LINE_LEN

Description

Return the length of each line in the D-cache in bytes.

Parameters

The following table lists the `MICROBLAZE_PVR_DCACHE_LINE_LEN` function arguments.

Table 109: **MICROBLAZE_PVR_DCACHE_LINE_LEN Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DCACHE_BYTE_SIZE

Description

Return the size of the D-cache in bytes.

Parameters

The following table lists the `MICROBLAZE_PVR_DCACHE_BYTE_SIZE` function arguments.

Table 110: **MICROBLAZE_PVR_DCACHE_BYTE_SIZE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_ICACHE_BASEADDR

Description

Return the base address of the I-cache.

Parameters

The following table lists the `MICROBLAZE_PVR_ICACHE_BASEADDR` function arguments.

Table 111: **MICROBLAZE_PVR_ICACHE_BASEADDR Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_ICACHE_HIGHADDR

Description

Return the high address of the I-cache.

Parameters

The following table lists the `MICROBLAZE_PVR_ICACHE_HIGHADDR` function arguments.

Table 112: **MICROBLAZE_PVR_ICACHE_HIGHADDR Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DCACHE_BASEADDR

Description

Return the base address of the D-cache.

Parameters

The following table lists the `MICROBLAZE_PVR_DCACHE_BASEADDR` function arguments.

Table 113: **MICROBLAZE_PVR_DCACHE_BASEADDR Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_DCACHE_HIGHADDR

Description

Return the high address of the D-cache.

Parameters

The following table lists the `MICROBLAZE_PVR_DCACHE_HIGHADDR` function arguments.

Table 114: **MICROBLAZE_PVR_DCACHE_HIGHADDR Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_TARGET_FAMILY

Description

Return the encoded target family identifier.

Parameters

The following table lists the `MICROBLAZE_PVR_TARGET_FAMILY` function arguments.

Table 115: **MICROBLAZE_PVR_TARGET_FAMILY Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_MSR_RESET_VALUE

Description

Refer to the MicroBlaze Processor Reference Guide (UG081) for mappings from encodings to target family name strings.

Parameters

The following table lists the `MICROBLAZE_PVR_MSR_RESET_VALUE` function arguments.

Table 116: **MICROBLAZE_PVR_MSR_RESET_VALUE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_MMU_TYPE

Description

Returns the value of C_USE_MMU.

Refer to the MicroBlaze Processor Reference Guide (UG081) for mappings from MMU type values to MMU function.

Parameters

The following table lists the `MICROBLAZE_PVR_MMU_TYPE` function arguments.

Table 117: **MICROBLAZE_PVR_MMU_TYPE Arguments**

Name	Description
_pvr	pvr data structure

Sleep Routines for Microblaze

The microblaze_sleep.h file contains microblaze sleep APIs.

These APIs provides delay for requested duration.

***Note*:** The microblaze_sleep.h file may contain architecture-dependent items.

Table 118: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SleepAxiTimer	void
void	XTime_StartAxiTimer	void
void	MB_Sleep	MilliSeconds-

Functions

Xil_SleepAxiTimer

Prototype

```
void Xil_SleepAxiTimer(u32 delay, u64 frequency);
```

XTime_StartAxiTimer

Prototype

```
void XTime_StartAxiTimer();
```

MB_Sleep

Provides delay for requested duration.

***Note*:** Instruction cache should be enabled for this to work.

Prototype

```
void MB_Sleep(u32 MilliSeconds) __attribute__((__deprecated__));
```

Parameters

The following table lists the `MB_Sleep` function arguments.

Table 119: **MB_Sleep Arguments**

Name	Description
MilliSeconds-	Delay time in milliseconds.

Returns

None.

ARM Processor Common API

This section provides a linked summary and detailed descriptions of the ARM Processor Common APIs.

Note: These APIs are applicable for the Cortex-A72 processor as well.

ARM Processor Exception Handling

ARM processors specific exception related APIs for Cortex Armv8-A, A9 and R5 can utilized for enabling/disabling IRQ, registering/removing handler for exceptions or initializing exception vector table with null handler.

Table 120: **Quick Function Reference**

Type	Name	Arguments
void	Xil_ExceptionRegisterHandler	exception_id Xil_ExceptionHandler Handler void * Data
void	Xil_ExceptionRemoveHandler	exception_id
void	Xil_GetExceptionRegisterHandler	exception_id Xil_ExceptionHandler * Handler void ** Data
void	Xil_ExceptionInit	None.
void	Xil_DataAbortHandler	None
void	Xil_PrefetchAbortHandler	None

Table 120: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_UndefinedExceptionHandler	None

Functions

Xil_ExceptionRegisterHandler

Register a handler for a specific exception.

This handler is being called when the processor encounters the specified exception.

Note: None.

Prototype

```
void Xil_ExceptionRegisterHandler(u32 Exception_id, Xil_ExceptionHandler
Handler, void *Data);
```

Parameters

The following table lists the `Xil_ExceptionRegisterHandler` function arguments.

Table 121: **Xil_ExceptionRegisterHandler Arguments**

Name	Description
exception_id	contains the ID of the exception source and should be in the range of 0 to XIL_EXCEPTION_ID_LAST. See xil_exception.h for further information.
Handler	to the Handler for that exception.
Data	is a reference to Data that will be passed to the Handler when it gets called.

Returns

None.

Xil_ExceptionRemoveHandler

Removes the Handler for a specific exception Id.

The stub Handler is then registered for this exception Id.

Note: None.

Prototype

```
void Xil_ExceptionRemoveHandler(u32 Exception_id);
```

Parameters

The following table lists the `Xil_ExceptionRemoveHandler` function arguments.

Table 122: **Xil_ExceptionRemoveHandler Arguments**

Name	Description
exception_id	contains the ID of the exception source and should be in the range of 0 to XIL_EXCEPTION_ID_LAST. See xil_exception.h for further information.

Returns

None.

Xil_GetExceptionRegisterHandler

Get a handler for a specific exception.

This handler is being called when the processor encounters the specified exception.

Note: None.

Prototype

```
void Xil_GetExceptionRegisterHandler(u32 Exception_id, Xil_ExceptionHandler
*Handler, void **Data);
```

Parameters

The following table lists the `Xil_GetExceptionRegisterHandler` function arguments.

Table 123: **Xil_GetExceptionRegisterHandler Arguments**

Name	Description
exception_id	contains the ID of the exception source and should be in the range of 0 to XIL_EXCEPTION_ID_LAST. See xil_exception.h for further information.
Handler	to the Handler for that exception.
Data	is a reference to Data that will be passed to the Handler when it gets called.

Returns

None.

Xil_ExceptionInit

The function is a common API used to initialize exception handlers across all supported arm processors.

For ARM v8-A, Cortex-R5, and Cortex-A9, the exception handlers are being initialized statically and this function does not do anything. However, it is still present to take care of backward compatibility issues (in earlier versions of BSPs, this API was being used to initialize exception handlers).

***Note*:** None.

Prototype

```
void Xil_ExceptionInit(void);
```

Parameters

The following table lists the `Xil_ExceptionInit` function arguments.

Table 124: **Xil_ExceptionInit Arguments**

Name	Description
None.	

Returns

None.

Xil_DataAbortHandler

Default Data abort handler which prints data fault status register through which information about data fault can be acquired.

***Note*:** None.

Prototype

```
void Xil_DataAbortHandler(void *CallBackRef);
```

Parameters

The following table lists the `Xil_DataAbortHandler` function arguments.

Table 125: **Xil_DataAbortHandler Arguments**

Name	Description
None	

Returns

None.

Xil_PrefetchAbortHandler

Default Prefetch abort handler which prints prefetch fault status register through which information about instruction prefetch fault can be acquired.

***Note*:** None.

Prototype

```
void Xil_PrefetchAbortHandler(void *CallBackRef);
```

Parameters

The following table lists the `Xil_PrefetchAbortHandler` function arguments.

Table 126: **Xil_PrefetchAbortHandler Arguments**

Name	Description
None	

Returns

None.

Xil_UndefinedExceptionHandler

Default undefined exception handler which prints address of the undefined instruction if debug prints are enabled.

***Note*:** None.

Prototype

```
void Xil_UndefinedExceptionHandler(void *CallBackRef);
```

Parameters

The following table lists the `Xil_UndefinedExceptionHandler` function arguments.

Table 127: **Xil_UndefinedExceptionHandler Arguments**

Name	Description
None	

Returns

None.

Definitions

Define Xil_ExceptionEnableMask

Description

Enable Exceptions.

***Note*:** If bit is 0, exception is enabled. C-Style signature: void `Xil_ExceptionEnableMask(Mask)`

Define Xil_ExceptionEnable

Description

Enable the IRQ exception.

***Note*:** None.

Define Xil_ExceptionDisableMask

Description

Disable Exceptions.

***Note*:** If bit is 1, exception is disabled. C-Style signature: `Xil_ExceptionDisableMask(Mask)`

Define Xil_ExceptionDisable

Description

Disable the IRQ exception.

***Note*:** None.

Define Xil_EnableNestedInterrupts

Description

Enable nested interrupts by clearing the I and F bits in CPSR.

This API is defined for cortex-a9 and cortex-r5.

***Note*:** This macro is supposed to be used from interrupt handlers. In the interrupt handler the interrupts are disabled by default (I and F are 1). To allow nesting of interrupts, this macro should be used. It clears the I and F bits by changing the ARM mode to system mode. Once these bits are cleared and provided the preemption of interrupt conditions are met in the GIC, nesting of interrupts will start happening. Caution: This macro must be used with caution. Before calling this macro, the user must ensure that the source of the current IRQ is appropriately cleared. Otherwise, as soon as we clear the I and F bits, there can be an infinite loop of interrupts with an eventual crash (all the stack space getting consumed).

Define Xil_DisableNestedInterrupts

Description

Disable the nested interrupts by setting the I and F bits.

This API is defined for cortex-a9 and cortex-r5.

Note: This macro is meant to be called in the interrupt service routines. This macro cannot be used independently. It can only be used when nesting of interrupts have been enabled by using the macro `Xil_EnableNestedInterrupts()` . In a typical flow, the user first calls the Xil_EnableNestedInterrupts in the ISR at the appropriate point. The user then must call this macro before exiting the interrupt service routine. This macro puts the ARM back in IRQ/FIQ mode and hence sets back the I and F bits.

Cortex R5 Processor API

Cortex R5 Processor API

Standalone BSP contains boot code, cache, exception handling, file and memory management, configuration, time and processor-specific include functions.

It supports gcc compiler. This section provides a linked summary and detailed descriptions of the Cortex R5 processor APIs.

Cortex R5 Processor Boot Code

The boot.S file contains a minimal set of code for transferring control from the processor reset location of the processor to the start of the application. The boot code performs minimum configuration which is required for an application to run starting from reset state of the processor. Below is a sequence illustrating what all configuration is performed before control reaches to main function.

1. Program vector table base for exception handling
2. Program stack pointer for various modes (IRQ, FIQ, supervisor, undefine, abort, system)
3. Disable instruction cache, data cache and MPU
4. Invalidate instruction and data cache
5. Configure MPU with short descriptor translation table format and program base address of translation table
6. Enable data cache, instruction cache and MPU
7. Enable Floating point unit

8. Transfer control to _start which clears BSS sections and jumping to main application

Cortex R5 Processor MPU specific APIs

MPU functions provides access to MPU operations such as enable MPU, disable MPU and set attribute for section of memory.

Boot code invokes Init_MPU function to configure the MPU. A total of 10 MPU regions are allocated with another 6 being free for users. Overview of the memory attributes for different MPU regions is as given below,

	Memory Range	Attributes of MPURegion
DDR	0x00000000 - 0x7FFFFFFF	Normal write-back Cacheable
PL	0x80000000 - 0xBFFFFFFF	Strongly Ordered
QSPI	0xC0000000 - 0xDFFFFFFF	Device Memory
PCIe	0xE0000000 - 0xEFFFFFFF	Device Memory
STM_CORESIGHT	0xF8000000 - 0xF8FFFFFF	Device Memory
RPU_R5_GIC	0xF9000000 - 0xF90FFFFF	Device memory
FPS	0xFD000000 - 0xFDFFFFFF	Device Memory
LPS	0xFE000000 - 0xFFFFFFFF	Device Memory
OCM	0xFFFC0000 - 0xFFFFFFFF	Normal write-back Cacheable

***Note*:** For a system where DDR is less than 2GB, region after DDR and before PL is marked as undefined in translation table. Memory range 0xFE000000-0xFEFFFFFF is allocated for upper LPS slaves, where as memory region 0xFF000000-0xFFFFFFFF is allocated for lower LPS slaves.

Table 128: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	INTPTR Addr u32 attrib
void	Xil_EnableMPU	None.
void	Xil_DisableMPU	None.
u32	Xil_SetMPURegion	Addr u64 size u32 attrib

Table 128: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
u32	Xil_UpdateMPUConfig	u32 reg_num INTPTR address u32 size u32 attrib
void	Xil_GetMPUConfig	XMpu_Config mpuconfig
u32	Xil_GetNumOfFreeRegions	none
u32	Xil_GetNextMPURegion	none
u32	Xil_DisableMPURegionByRegNum	u32 reg_num
u16	Xil_GetMPUFreeRegMask	none
u32	Xil_SetMPURegionByRegNum	u32 reg_num address u64 size u32 attrib
void *	Xil_MemMap	void

Functions

Xil_SetTlbAttributes

This function sets the memory attributes for a section covering 1MB, of memory in the translation table.

Prototype

```
void Xil_SetTlbAttributes(INTPTR Addr, u32 attrib);
```

Parameters

The following table lists the `Xil_SetTlbAttributes` function arguments.

Table 129: **Xil_SetTlbAttributes Arguments**

Name	Description
Addr	32-bit address for which memory attributes need to be set.

Table 129: **Xil_SetTlbAttributes Arguments** *(cont'd)*

Name	Description
attrib	Attribute for the given memory region.

Returns

None.

Xil_EnableMPU

Enable MPU for Cortex R5 processor.

This function invalidates I cache and flush the D Caches, and then enables the MPU.

Prototype

```
void Xil_EnableMPU(void);
```

Parameters

The following table lists the `Xil_EnableMPU` function arguments.

Table 130: **Xil_EnableMPU Arguments**

Name	Description
None.	

Returns

None.

Xil_DisableMPU

Disable MPU for Cortex R5 processors.

This function invalidates I cache and flush the D Caches, and then disabes the MPU.

Prototype

```
void Xil_DisableMPU(void);
```

Parameters

The following table lists the `Xil_DisableMPU` function arguments.

Table 131: **Xil_DisableMPU Arguments**

Name	Description
None.	

Returns

None.

Xil_SetMPURegion

Set the memory attributes for a section of memory in the translation table.

Prototype

```
u32 Xil_SetMPURegion(INTPTR addr, u64 size, u32 attrib);
```

Parameters

The following table lists the `Xil_SetMPURegion` function arguments.

Table 132: **Xil_SetMPURegion Arguments**

Name	Description
Addr	32-bit address for which memory attributes need to be set..
size	size is the size of the region.
attrib	Attribute for the given memory region.

Returns

None.

Xil_UpdateMPUConfig

Update the MPU configuration for the requested region number in the global MPU configuration table.

Prototype

```
u32 Xil_UpdateMPUConfig(u32 reg_num, INTPTR address, u32 size, u32 attrib);
```

Parameters

The following table lists the `Xil_UpdateMPUConfig` function arguments.

Table 133: **Xil_UpdateMPUConfig Arguments**

Name	Description
reg_num	The requested region number to be updated information for.
address	32 bit address for start of the region.
size	Requested size of the region.
attrib	Attribute for the corresponding region.

Returns

XST_FAILURE: When the requested region number if 16 or more. XST_SUCCESS: When the MPU configuration table is updated.

Xil_GetMPUConfig

The MPU configuration table is passed to the caller.

Prototype

```
void Xil_GetMPUConfig(XMpu_Config mpuconfig);
```

Parameters

The following table lists the `Xil_GetMPUConfig` function arguments.

Table 134: **Xil_GetMPUConfig Arguments**

Name	Description
mpuconfig	This is of type XMpu_Config which is an array of 16 entries of type structure representing the MPU config table

Returns

none

Xil_GetNumOfFreeRegions

Returns the total number of free MPU regions available.

Prototype

```
u32 Xil_GetNumOfFreeRegions(void);
```

Parameters

The following table lists the `Xil_GetNumOfFreeRegions` function arguments.

Table 135: **Xil_GetNumOfFreeRegions Arguments**

Name	Description
none	

Returns

Number of free regions available to users

Xil_GetNextMPURegion

Returns the next available free MPU region.

Prototype

```
u32 Xil_GetNextMPURegion(void);
```

Parameters

The following table lists the `Xil_GetNextMPURegion` function arguments.

Table 136: **Xil_GetNextMPURegion Arguments**

Name	Description
none	

Returns

The free MPU region available

Xil_DisableMPURegionByRegNum

Disables the corresponding region number as passed by the user.

Prototype

```
u32 Xil_DisableMPURegionByRegNum(u32 reg_num);
```

Parameters

The following table lists the `Xil_DisableMPURegionByRegNum` function arguments.

Table 137: **Xil_DisableMPURegionByRegNum Arguments**

Name	Description
reg_num	The region number to be disabled

Returns

XST_SUCCESS: If the region could be disabled successfully XST_FAILURE: If the requested region number is 16 or more.

Xil_GetMPUFreeRegMask

Returns the total number of free MPU regions available in the form of a mask.

A bit of 1 in the returned 16 bit value represents the corresponding region number to be available. For example, if this function returns 0xC0000, this would mean, the regions 14 and 15 are available to users.

Prototype

```
u16 Xil_GetMPUFreeRegMask(void);
```

Parameters

The following table lists the `Xil_GetMPUFreeRegMask` function arguments.

Table 138: **Xil_GetMPUFreeRegMask Arguments**

Name	Description
none	

Returns

The free region mask as a 16 bit value

Xil_SetMPURegionByRegNum

Enables the corresponding region number as passed by the user.

Prototype

```
u32 Xil_SetMPURegionByRegNum(u32 reg_num, INTPTR addr, u64 size, u32
attrib);
```

Parameters

The following table lists the `Xil_SetMPURegionByRegNum` function arguments.

Table 139: **Xil_SetMPURegionByRegNum Arguments**

Name	Description
reg_num	The region number to be enabled
address	32 bit address for start of the region.

Table 139: **Xil_SetMPURegionByRegNum Arguments** *(cont'd)*

Name	Description
size	Requested size of the region.
attrib	Attribute for the corresponding region.

Returns

XST_SUCCESS: If the region could be created successfully XST_FAILURE: If the requested region number is 16 or more.

Xil_MemMap

Prototype

```
void * Xil_MemMap(UINTPTR Physaddr, size_t size, u32 flags);
```

Cortex R5 Processor Cache Functions

Cache functions provide access to cache related operations such as flush and invalidate for instruction and data caches.

It gives option to perform the cache operations on a single cacheline, a range of memory and an entire cache.

Table 140: **Quick Function Reference**

Type	Name	Arguments
void	Xil_DCacheEnable	None.
void	Xil_DCacheDisable	None.
void	Xil_DCacheInvalidate	None.
void	Xil_DCacheInvalidateRange	INTPTR adr u32 len
void	Xil_DCacheFlush	None.
void	Xil_DCacheFlushRange	INTPTR adr u32 len
void	Xil_DCacheInvalidateLine	INTPTR adr

Table 140: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_DCacheFlushLine	INTPTR adr
void	Xil_DCacheStoreLine	INTPTR adr
void	Xil_ICacheEnable	None.
void	Xil_ICacheDisable	None.
void	Xil_ICacheInvalidate	None.
void	Xil_ICacheInvalidateRange	INTPTR adr u32 len
void	Xil_ICacheInvalidateLine	INTPTR adr

Functions

Xil_DCacheEnable

Enable the Data cache.

***Note*:** None.

Prototype

```
void Xil_DCacheEnable(void);
```

Parameters

The following table lists the `Xil_DCacheEnable` function arguments.

Table 141: **Xil_DCacheEnable Arguments**

Name	Description
None.	

Returns

None.

Xil_DCacheDisable

Disable the Data cache.

***Note*:** None.

Prototype

```
void Xil_DCacheDisable(void);
```

Parameters

The following table lists the `Xil_DCacheDisable` function arguments.

Table 142: **Xil_DCacheDisable Arguments**

Name	Description
None.	

Returns

None.

Xil_DCacheInvalidate

Invalidate the entire Data cache.

Prototype

```
void Xil_DCacheInvalidate(void);
```

Parameters

The following table lists the `Xil_DCacheInvalidate` function arguments.

Table 143: **Xil_DCacheInvalidate Arguments**

Name	Description
None.	

Returns

None.

Xil_DCacheInvalidateRange

Invalidate the Data cache for the given address range.

If the bytes specified by the address (adr) are cached by the Data cache,the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the modified contents are lost and are NOT written to system memory before the line is invalidated.

Prototype

```
void Xil_DCacheInvalidateRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_DCacheInvalidateRange` function arguments.

Table 144: **Xil_DCacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of range to be invalidated in bytes.

Returns

None.

Xil_DCacheFlush

Flush the entire Data cache.

Prototype

```
void Xil_DCacheFlush(void);
```

Parameters

The following table lists the `Xil_DCacheFlush` function arguments.

Table 145: **Xil_DCacheFlush Arguments**

Name	Description
None.	

Returns

None.

Xil_DCacheFlushRange

Flush the Data cache for the given address range.

If the bytes specified by the address (adr) are cached by the Data cache, the cacheline containing those bytes is invalidated.If the cacheline is modified (dirty), the written to system memory before the lines are invalidated.

Prototype

```
void Xil_DCacheFlushRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_DCacheFlushRange` function arguments.

Table 146: **Xil_DCacheFlushRange Arguments**

Name	Description
adr	32bit start address of the range to be flushed.
len	Length of the range to be flushed in bytes

Returns

None.

Xil_DCacheInvalidateLine

Invalidate a Data cache line.

If the byte specified by the address (adr) is cached by the data cache, the cacheline containing that byte is invalidated.If the cacheline is modified (dirty), the modified contents are lost and are NOT written to system memory before the line is invalidated.

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheInvalidateLine(INTPTR adr);
```

Parameters

The following table lists the `Xil_DCacheInvalidateLine` function arguments.

Table 147: **Xil_DCacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the data to be flushed.

Returns

None.

Xil_DCacheFlushLine

Flush a Data cache line.

If the byte specified by the address (adr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory before the line is invalidated.

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheFlushLine(INTPTR adr);
```

Parameters

The following table lists the `Xil_DCacheFlushLine` function arguments.

Table 148: **Xil_DCacheFlushLine Arguments**

Name	Description
adr	32bit address of the data to be flushed.

Returns

None.

Xil_DCacheStoreLine

Store a Data cache line.

If the byte specified by the address (adr) is cached by the Data cache and the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory.After the store completes, the cacheline is marked as unmodified (not dirty).

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheStoreLine(INTPTR adr);
```

Parameters

The following table lists the `Xil_DCacheStoreLine` function arguments.

Table 149: **Xil_DCacheStoreLine Arguments**

Name	Description
adr	32bit address of the data to be stored

Returns

None.

Xil_ICacheEnable

Enable the instruction cache.

Prototype

```
void Xil_ICacheEnable(void);
```

Parameters

The following table lists the `Xil_ICacheEnable` function arguments.

Table 150: **Xil_ICacheEnable Arguments**

Name	Description
None.	

Returns

None.

Xil_ICacheDisable

Disable the instruction cache.

Prototype

```
void Xil_ICacheDisable(void);
```

Parameters

The following table lists the `Xil_ICacheDisable` function arguments.

Table 151: **Xil_ICacheDisable Arguments**

Name	Description
None.	

Returns

None.

Xil_ICacheInvalidate

Invalidate the entire instruction cache.

Prototype

```
void Xil_ICacheInvalidate(void);
```

Parameters

The following table lists the `Xil_ICacheInvalidate` function arguments.

Table 152: **Xil_ICacheInvalidate Arguments**

Name	Description
None.	

Returns

None.

Xil_ICacheInvalidateRange

Invalidate the instruction cache for the given address range.

If the bytes specified by the address (adr) are cached by the Data cache, the cacheline containing that byte is invalidated. If the cachelineis modified (dirty), the modified contents are lost and are NOT written to system memory before the line is invalidated.

Prototype

```
void Xil_ICacheInvalidateRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_ICacheInvalidateRange` function arguments.

Table 153: **Xil_ICacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_ICacheInvalidateLine

Invalidate an instruction cache line.If the instruction specified by the address is cached by the instruction cache, the cacheline containing that instruction is invalidated.

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_ICacheInvalidateLine(INTPTR adr);
```

Parameters

The following table lists the `Xil_ICacheInvalidateLine` function arguments.

Table 154: **Xil_ICacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the instruction to be invalidated.

Returns

None.

Cortex R5 Time Functions

The xtime_l.h provides access to 32-bit TTC timer counter.

These functions can be used by applications to track the time.

Table 155: **Quick Function Reference**

Type	Name	Arguments
void	XTime_SetTime	XTime Xtime_Global
void	XTime_GetTime	XTime * Xtime_Global

Functions

XTime_SetTime

TTC Timer runs continuously and the time can not be set as desired.

This API doesn't contain anything. It is defined to have uniformity across platforms.

***Note*:** In multiprocessor environment reference time will reset/lost for all processors, when this function called by any one processor.

Prototype

```
void XTime_SetTime(XTime Xtime_Global);
```

Parameters

The following table lists the `XTime_SetTime` function arguments.

Table 156: **XTime_SetTime Arguments**

Name	Description
Xtime_Global	32 bit value to be written to the timer counter register.

Returns

None.

XTime_GetTime

Get the time from the timer counter register.

Prototype

```
void XTime_GetTime(XTime *Xtime_Global);
```

Parameters

The following table lists the `XTime_GetTime` function arguments.

Table 157: **XTime_GetTime Arguments**

Name	Description
Xtime_Global	Pointer to the 32 bit location to be updated with the time current value of timer counter register.

Returns

None.

Cortex R5 Event Counters Functions

Cortex R5 event counter functions can be utilized to configure and control the Cortex-R5 performance monitor events.

Cortex-R5 Performance Monitor has 3 event counters which can be used to count a variety of events described in Coretx-R5 TRM. The xpm_counter.h file defines configurations XPM_CNTRCFGx which can be used to program the event counters to count a set of events.

Table 158: **Quick Function Reference**

Type	Name	Arguments
void	Xpm_SetEvents	s32 PmcrCfg
void	Xpm_GetEventCounters	u32 * PmCtrValue
u32	Xpm_DisableEvent	Event
u32	Xpm_SetUpAnEvent	Event
u32	Xpm_GetEventCounter	Event Pointer
void	Xpm_DisableEventCounters	None.
void	Xpm_EnableEventCounters	None.
void	Xpm_ResetEventCounters	None.
void	Xpm_SleepPerfCounter	u32 delay u64 frequency

Functions

Xpm_SetEvents

This function configures the Cortex R5 event counters controller, with the event codes, in a configuration selected by the user and enables the counters.

Prototype

```
void Xpm_SetEvents(s32 PmcrCfg);
```

Parameters

The following table lists the `Xpm_SetEvents` function arguments.

Table 159: **Xpm_SetEvents Arguments**

Name	Description
PmcrCfg	Configuration value based on which the event counters are configured.XPM_CNTRCFG* values defined in xpm_counter.h can be utilized for setting configuration

Returns

None.

Xpm_GetEventCounters

This function disables the event counters and returns the counter values.

Prototype

```
void Xpm_GetEventCounters(u32 *PmCtrValue);
```

Parameters

The following table lists the `Xpm_GetEventCounters` function arguments.

Table 160: **Xpm_GetEventCounters Arguments**

Name	Description
PmCtrValue	Pointer to an array of type u32 PmCtrValue[6]. It is an output parameter which is used to return the PM counter values.

Returns

None.

Xpm_DisableEvent

Disables the requested event counter.

***Note*:** None.

Prototype

```
u32 Xpm_DisableEvent(u32 EventHandlerId);
```

Parameters

The following table lists the `Xpm_DisableEvent` function arguments.

Table 161: **Xpm_DisableEvent Arguments**

Name	Description
Event	Counter ID. The counter ID is the same that was earlier returned through a call to Xpm_SetUpAnEvent. Cortex-R5 supports only 3 counters. The valid values are 0, 1, or 2.

Returns

- XST_SUCCESS if successful.
- XST_FAILURE if the passed Counter ID is invalid (i.e. greater than 2).

Xpm_SetUpAnEvent

Sets up one of the event counters to count events based on the Event ID passed.

For supported Event IDs please refer xpm_counter.h. Upon invoked, the API searches for an available counter. After finding one, it sets up the counter to count events for the requested event.

Note: None.

Prototype

```
u32 Xpm_SetUpAnEvent(u32 EventID);
```

Parameters

The following table lists the `Xpm_SetUpAnEvent` function arguments.

Table 162: **Xpm_SetUpAnEvent Arguments**

Name	Description
Event	ID. For valid values, please refer xpm_counter.h.

Returns

- Counter Number if successful. For Cortex-R5, valid return values are 0, 1, or 2.
- XPM_NO_COUNTERS_AVAILABLE (0xFF) if all counters are being used

Xpm_GetEventCounter

Reads the counter value for the requested counter ID.

This is used to read the number of events that has been counted for the requsted event ID. This can only be called after a call to Xpm_SetUpAnEvent.

Note: None.

Prototype

```
u32 Xpm_GetEventCounter(u32 EventHandlerId, u32 *CntVal);
```

Parameters

The following table lists the `Xpm_GetEventCounter` function arguments.

Table 163: **Xpm_GetEventCounter Arguments**

Name	Description
Event	Counter ID. The counter ID is the same that was earlier returned through a call to Xpm_SetUpAnEvent. Cortex-R5 supports only 3 counters. The valid values are 0, 1, or 2.
Pointer	to a 32 bit unsigned int type. This is used to return the event counter value.

Returns

- XST_SUCCESS if successful.
- XST_FAILURE if the passed Counter ID is invalid (i.e. greater than 2).

Xpm_DisableEventCounters

This function disables the Cortex R5 event counters.

Prototype

```
void Xpm_DisableEventCounters(void);
```

Parameters

The following table lists the `Xpm_DisableEventCounters` function arguments.

Table 164: **Xpm_DisableEventCounters Arguments**

Name	Description
None.	

Returns

None.

Xpm_EnableEventCounters

This function enables the Cortex R5 event counters.

Prototype

```
void Xpm_EnableEventCounters(void);
```

Parameters

The following table lists the `Xpm_EnableEventCounters` function arguments.

Table 165: **Xpm_EnableEventCounters Arguments**

Name	Description
None.	

Returns

None.

Xpm_ResetEventCounters

This function resets the Cortex R5 event counters.

Prototype

```
void Xpm_ResetEventCounters(void);
```

Parameters

The following table lists the `Xpm_ResetEventCounters` function arguments.

Table 166: **Xpm_ResetEventCounters Arguments**

Name	Description
None.	

Returns

None.

Xpm_SleepPerfCounter

This is helper function used by sleep/usleep APIs to generate delay in sec/usec.

Prototype

```
void Xpm_SleepPerfCounter(u32 delay, u64 frequency);
```

Parameters

The following table lists the `Xpm_SleepPerfCounter` function arguments.

Table 167: **Xpm_SleepPerfCounter Arguments**

Name	Description
delay	- delay time in sec/usec
frequency	- Number of countes in second/micro second

Returns

None.

Cortex R5 Processor Specific Include Files

The xpseudo_asm.h file includes xreg_cortexr5.h and xpseudo_asm_gcc.h.

The xreg_cortexr5.h include file contains the register numbers and the register bits for the Arm Cortex-R5 processor.

The xpseudo_asm_gcc.h file contains the definitions for the most often used inline assembler instructions, available as macros. These can be very useful for tasks such as setting or getting special purpose registers, synchronization,or cache manipulation. These inline assembler instructions can be used from drivers and user applications written in C.

Cortex R5 peripheral definitions

The xparameters_ps.h file contains the canonical definitions and constant declarations for peripherals within hardblock, attached to the ARM Cortex R5 core.

These definitions can be used by drivers or applications to access the peripherals.

Cortex A9 Processor API

Cortex A9 Processor API

Standalone BSP contains boot code, cache, exception handling, file and memory management, configuration, time and processor-specific include functions.

It supports gcc compilers.

Cortex A9 Processor Boot Code

The boot code performs minimum configuration which is required for an application to run starting from processor reset state of the processor. Below is a sequence illustrating what all configuration is performed before control reaches to main function.

1. Program vector table base for exception handling
2. Invalidate instruction cache, data cache and TLBs
3. Program stack pointer for various modes (IRQ, FIQ, supervisor, undefine, abort, system)
4. Configure MMU with short descriptor translation table format and program base address of translation table
5. Enable data cache, instruction cache and MMU
6. Enable Floating point unit
7. Transfer control to _start which clears BSS sections, initializes global timer and runs global constructor before jumping to main application

The translation_table.S contains a static page table required by MMU for cortex-A9. This translation table is flat mapped (input address = output address) with default memory attributes defined for zynq architecture. It utilizes short descriptor translation table format with each section defining 1 MB of memory.

The overview of translation table memory attributes is described below.

For region 0x00000000 - 0x3FFFFFFF, a system where DDR is less than 1 GB, region after DDR and before PL is marked as undefined/reserved in translation table. In 0xF8000000 - 0xF8FFFFFF, 0xF8000C00 - 0xF8000FFF, 0xF8010000 - 0xF88FFFFF and 0xF8F03000 to 0xF8FFFFFF are reserved but due to granual size of 1 MB, it is not possible to define separate regions for them. For region 0xFFF00000 - 0xFFFFFFFF, 0xFFF00000 to 0xFFFB0000 is reserved but due to 1MB granual size, it is not possible to define separate region for it.

	Memory Range	Definition in Translation Table
DDR	0x00000000 - 0x3FFFFFFF	Normal write-back Cacheable
PL	0x40000000 - 0xBFFFFFFF	Strongly Ordered
Reserved	0xC0000000 - 0xDFFFFFFF	Unassigned
Memory mapped devices	0xE0000000 - 0xE02FFFFF	Device Memory
Reserved	0xE0300000 - 0xE0FFFFFF	Unassigned
NAND, NOR	0xE1000000 - 0xE3FFFFFF	Device memory
SRAM	0xE4000000 - 0xE5FFFFFF	Normal write-back Cacheable
Reserved	0xE6000000 - 0xF7FFFFFF	Unassigned
AMBA APB Peripherals	0xF8000000 - 0xF8FFFFFF	Device Memory

	Memory Range	Definition in Translation Table
Reserved	0xF9000000 - 0xFBFFFFFF	Unassigned
Linear QSPI - XIP	0xFC000000 - 0xFDFFFFFF	Normal write-through cacheable
Reserved	0xFE000000 - 0xFFEFFFFF	Unassigned
OCM	0xFFF00000 - 0xFFFFFFFF	Normal inner write-back cacheable

Cortex A9 Processor Cache Functions

Cache functions provide access to cache related operations such as flush and invalidate for instruction and data caches.

It gives option to perform the cache operations on a single cacheline, a range of memory and an entire cache.

Table 168: **Quick Function Reference**

Type	Name	Arguments
void	Xil_DCacheEnable	void
void	Xil_DCacheDisable	void
void	Xil_DCacheInvalidate	void
void	Xil_DCacheInvalidateRange	INTPTR adr u32 len
void	Xil_DCacheFlush	void
void	Xil_DCacheFlushRange	INTPTR adr u32 len
void	Xil_ICacheEnable	void
void	Xil_ICacheDisable	void
void	Xil_ICacheInvalidate	void
void	Xil_ICacheInvalidateRange	INTPTR adr u32 len
void	Xil_DCacheInvalidateLine	u32 adr

Table 168: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_DCacheFlushLine	u32 adr
void	Xil_DCacheStoreLine	u32 adr
void	Xil_ICacheInvalidateLine	u32 adr
void	Xil_L1DCacheEnable	void
void	Xil_L1DCacheDisable	void
void	Xil_L1DCacheInvalidate	void
void	Xil_L1DCacheInvalidateLine	u32 adr
void	Xil_L1DCacheInvalidateRange	u32 adr u32 len
void	Xil_L1DCacheFlush	void
void	Xil_L1DCacheFlushLine	u32 adr
void	Xil_L1DCacheFlushRange	u32 adr u32 len
void	Xil_L1DCacheStoreLine	Address
void	Xil_L1ICacheEnable	void
void	Xil_L1ICacheDisable	void
void	Xil_L1ICacheInvalidate	void
void	Xil_L1ICacheInvalidateLine	u32 adr
void	Xil_L1ICacheInvalidateRange	u32 adr u32 len
void	Xil_L2CacheEnable	void

Table 168: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_L2CacheDisable	void
void	Xil_L2CacheInvalidate	void
void	Xil_L2CacheInvalidateLine	u32 adr
void	Xil_L2CacheInvalidateRange	u32 adr u32 len
void	Xil_L2CacheFlush	void
void	Xil_L2CacheFlushLine	u32 adr
void	Xil_L2CacheFlushRange	u32 adr u32 len
void	Xil_L2CacheStoreLine	u32 adr

Functions

Xil_DCacheEnable

Enable the Data cache.

Note: None.

Prototype

```
void Xil_DCacheEnable(void);
```

Returns

None.

Xil_DCacheDisable

Disable the Data cache.

Note: None.

Prototype

```
void Xil_DCacheDisable(void);
```

Returns

None.

Xil_DCacheInvalidate

Invalidate the entire Data cache.

***Note*:** None.

Prototype

```
void Xil_DCacheInvalidate(void);
```

Returns

None.

Xil_DCacheInvalidateRange

Invalidate the Data cache for the given address range.

If the bytes specified by the address range are cached by the Data cache, the cachelines containing those bytes are invalidated. If the cachelines are modified (dirty), the modified contents are lost and NOT written to the system memory before the lines are invalidated.

In this function, if start address or end address is not aligned to cache-line, particular cache-line containing unaligned start or end address is flush first and then invalidated the others as invalidating the same unaligned cache line may result into loss of data. This issue raises few possibilities.

If the address to be invalidated is not cache-line aligned, the following choices are available:

1. Invalidate the cache line when required and do not bother much for the side effects. Though it sounds good, it can result in hard-to-debug issues. The problem is, if some other variable are allocated in the same cache line and had been recently updated (in cache), the invalidation would result in loss of data.
2. Flush the cache line first. This will ensure that if any other variable present in the same cache line and updated recently are flushed out to memory. Then it can safely be invalidated. Again it sounds good, but this can result in issues. For example, when the invalidation happens in a typical ISR (after a DMA transfer has updated the memory), then flushing the cache line means, losing data that were updated recently before the ISR got invoked.

Linux prefers the second one. To have uniform implementation (across standalone and Linux), the second option is implemented. This being the case, following needs to be taken care of:

1. Whenever possible, the addresses must be cache line aligned. Please nore that, not just start address, even the end address must be cache line aligned. If that is taken care of, this will always work.
2. Avoid situations where invalidation has to be done after the data is updated by peripheral/DMA directly into the memory. It is not tough to achieve (may be a bit risky). The common use case to do invalidation is when a DMA happens. Generally for such use cases, buffers can be allocated first and then start the DMA. The practice that needs to be followed here is, immediately after buffer allocation and before starting the DMA, do the invalidation. With this approach, invalidation need not to be done after the DMA transfer is over.

This is going to always work if done carefully. However, the concern is, there is no guarantee that invalidate has not needed to be done after DMA is complete. For example, because of some reasons if the first cache line or last cache line (assuming the buffer in question comprises of multiple cache lines) are brought into cache (between the time it is invalidated and DMA completes) because of some speculative prefetching or reading data for a variable present in the same cache line, then we will have to invalidate the cache after DMA is complete.

***Note*:** None.

Prototype

```
void Xil_DCacheInvalidateRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_DCacheInvalidateRange` function arguments.

Table 169: **Xil_DCacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_DCacheFlush

Flush the entire Data cache.

***Note*:** None.

Prototype

```
void Xil_DCacheFlush(void);
```

Returns

None.

Xil_DCacheFlushRange

Flush the Data cache for the given address range.

If the bytes specified by the address range are cached by the data cache, the cachelines containing those bytes are invalidated. If the cachelines are modified (dirty), they are written to the system memory before the lines are invalidated.

Prototype

```
void Xil_DCacheFlushRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_DCacheFlushRange` function arguments.

Table 170: **Xil_DCacheFlushRange Arguments**

Name	Description
adr	32bit start address of the range to be flushed.
len	Length of the range to be flushed in bytes.

Returns

None.

Xil_ICacheEnable

Enable the instruction cache.

Prototype

```
void Xil_ICacheEnable(void);
```

Returns

None.

Xil_ICacheDisable

Disable the instruction cache.

Prototype

```
void Xil_ICacheDisable(void);
```

Returns

None.

Xil_ICacheInvalidate

Invalidate the entire instruction cache.

Prototype

```
void Xil_ICacheInvalidate(void);
```

Returns

None.

Xil_ICacheInvalidateRange

Invalidate the instruction cache for the given address range.

If the instructions specified by the address range are cached by the instrunction cache, the cachelines containing those instructions are invalidated.

Prototype

```
void Xil_ICacheInvalidateRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_ICacheInvalidateRange` function arguments.

Table 171: **Xil_ICacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_DCacheInvalidateLine

Invalidate a Data cache line.

If the byte specified by the address (adr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the modified contents are lost and are NOT written to the system memory before the line is invalidated.

Note: The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheInvalidateLine(u32 adr);
```

Parameters

The following table lists the `Xil_DCacheInvalidateLine` function arguments.

Table 172: **Xil_DCacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the data to be flushed.

Returns

None.

Xil_DCacheFlushLine

Flush a Data cache line.

If the byte specified by the address (adr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory before the line is invalidated.

Note: The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheFlushLine(u32 adr);
```

Parameters

The following table lists the `Xil_DCacheFlushLine` function arguments.

Table 173: **Xil_DCacheFlushLine Arguments**

Name	Description
adr	32bit address of the data to be flushed.

Returns

None.

Xil_DCacheStoreLine

Store a Data cache line.

If the byte specified by the address (adr) is cached by the Data cache and the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory. After the store completes, the cacheline is marked as unmodified (not dirty).

Note: The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheStoreLine(u32 adr);
```

Parameters

The following table lists the `Xil_DCacheStoreLine` function arguments.

Table 174: **Xil_DCacheStoreLine Arguments**

Name	Description
adr	32bit address of the data to be stored.

Returns

None.

Xil_ICacheInvalidateLine

Invalidate an instruction cache line.

If the instruction specified by the address is cached by the instruction cache, the cacheline containing that instruction is invalidated.

Note: The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_ICacheInvalidateLine(u32 adr);
```

Parameters

The following table lists the `Xil_ICacheInvalidateLine` function arguments.

Table 175: **Xil_ICacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the instruction to be invalidated.

Returns

None.

Xil_L1DCacheEnable

Enable the level 1 Data cache.

Prototype

```
void Xil_L1DCacheEnable(void);
```

Returns

None.

Xil_L1DCacheDisable

Disable the level 1 Data cache.

Prototype

```
void Xil_L1DCacheDisable(void);
```

Returns

None.

Xil_L1DCacheInvalidate

Invalidate the level 1 Data cache.

Note: In Cortex A9, there is no cp instruction for invalidating the whole D-cache. This function invalidates each line by set/way.

Prototype

```
void Xil_L1DCacheInvalidate(void);
```

Returns

None.

Xil_L1DCacheInvalidateLine

Invalidate a level 1 Data cache line.

If the byte specified by the address (Addr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the modified contents are lost and are NOT written to system memory before the line is invalidated.

***Note*:** The bottom 5 bits are set to 0, forced by architecture.

Prototype

```
void Xil_L1DCacheInvalidateLine(u32 adr);
```

Parameters

The following table lists the `Xil_L1DCacheInvalidateLine` function arguments.

Table 176: **Xil_L1DCacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the data to be invalidated.

Returns

None.

Xil_L1DCacheInvalidateRange

Invalidate the level 1 Data cache for the given address range.

If the bytes specified by the address range are cached by the Data cache, the cachelines containing those bytes are invalidated. If the cachelines are modified (dirty), the modified contents are lost and NOT written to the system memory before the lines are invalidated.

Prototype

```
void Xil_L1DCacheInvalidateRange(u32 adr, u32 len);
```

Parameters

The following table lists the `Xil_L1DCacheInvalidateRange` function arguments.

Table 177: **Xil_L1DCacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_L1DCacheFlush

Flush the level 1 Data cache.

***Note*:** In Cortex A9, there is no cp instruction for flushing the whole D-cache. Need to flush each line.

Prototype

```
void Xil_L1DCacheFlush(void);
```

Returns

None.

Xil_L1DCacheFlushLine

Flush a level 1 Data cache line.

If the byte specified by the address (adr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory before the line is invalidated.

***Note*:** The bottom 5 bits are set to 0, forced by architecture.

Prototype

```
void Xil_L1DCacheFlushLine(u32 adr);
```

Parameters

The following table lists the `Xil_L1DCacheFlushLine` function arguments.

Table 178: **Xil_L1DCacheFlushLine Arguments**

Name	Description
adr	32bit address of the data to be flushed.

Returns

None.

Xil_L1DCacheFlushRange

Flush the level 1 Data cache for the given address range.

If the bytes specified by the address range are cached by the Data cache, the cacheline containing those bytes are invalidated. If the cachelines are modified (dirty), they are written to system memory before the lines are invalidated.

Prototype

```
void Xil_L1DCacheFlushRange(u32 adr, u32 len);
```

Parameters

The following table lists the `Xil_L1DCacheFlushRange` function arguments.

Table 179: **Xil_L1DCacheFlushRange Arguments**

Name	Description
adr	32bit start address of the range to be flushed.
len	Length of the range to be flushed in bytes.

Returns

None.

Xil_L1DCacheStoreLine

Store a level 1 Data cache line.

If the byte specified by the address (adr) is cached by the Data cache and the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory. After the store completes, the cacheline is marked as unmodified (not dirty).

***Note*:** The bottom 5 bits are set to 0, forced by architecture.

Prototype

```
void Xil_L1DCacheStoreLine(u32 adr);
```

Parameters

The following table lists the `Xil_L1DCacheStoreLine` function arguments.

Table 180: **Xil_L1DCacheStoreLine Arguments**

Name	Description
Address	to be stored.

Returns

None.

Xil_L1ICacheEnable

Enable the level 1 instruction cache.

Prototype

```
void Xil_L1ICacheEnable(void);
```

Returns

None.

Xil_L1ICacheDisable

Disable level 1 the instruction cache.

Prototype

```
void Xil_L1ICacheDisable(void);
```

Returns

None.

Xil_L1ICacheInvalidate

Invalidate the entire level 1 instruction cache.

Prototype

```
void Xil_L1ICacheInvalidate(void);
```

Returns

None.

Xil_L1ICacheInvalidateLine

Invalidate a level 1 instruction cache line.

If the instruction specified by the address is cached by the instruction cache, the cacheline containing that instruction is invalidated.

Note: The bottom 5 bits are set to 0, forced by architecture.

Prototype

```
void Xil_L1ICacheInvalidateLine(u32 adr);
```

Parameters

The following table lists the `Xil_L1ICacheInvalidateLine` function arguments.

Table 181: **Xil_L1ICacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the instruction to be invalidated.

Returns

None.

Xil_L1ICacheInvalidateRange

Invalidate the level 1 instruction cache for the given address range.

If the instrucions specified by the address range are cached by the instruction cache, the cacheline containing those bytes are invalidated.

Prototype

```
void Xil_L1ICacheInvalidateRange(u32 adr, u32 len);
```

Parameters

The following table lists the `Xil_L1ICacheInvalidateRange` function arguments.

Table 182: **Xil_L1ICacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_L2CacheEnable

Enable the L2 cache.

Prototype

```
void Xil_L2CacheEnable(void);
```

Returns

None.

Xil_L2CacheDisable

Disable the L2 cache.

Prototype

```
void Xil_L2CacheDisable(void);
```

Returns

None.

Xil_L2CacheInvalidate

Invalidate the entire level 2 cache.

Prototype

```
void Xil_L2CacheInvalidate(void);
```

Returns

None.

Xil_L2CacheInvalidateLine

Invalidate a level 2 cache line.

If the byte specified by the address (adr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the modified contents are lost and are NOT written to system memory before the line is invalidated.

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_L2CacheInvalidateLine(u32 adr);
```

Parameters

The following table lists the `Xil_L2CacheInvalidateLine` function arguments.

Table 183: **Xil_L2CacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the data/instruction to be invalidated.

Returns

None.

Xil_L2CacheInvalidateRange

Invalidate the level 2 cache for the given address range.

If the bytes specified by the address range are cached by the L2 cache, the cacheline containing those bytes are invalidated. If the cachelines are modified (dirty), the modified contents are lost and are NOT written to system memory before the lines are invalidated.

Prototype

```
void Xil_L2CacheInvalidateRange(u32 adr, u32 len);
```

Parameters

The following table lists the `Xil_L2CacheInvalidateRange` function arguments.

Table 184: **Xil_L2CacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_L2CacheFlush

Flush the entire level 2 cache.

Prototype

```
void Xil_L2CacheFlush(void);
```

Returns

None.

Xil_L2CacheFlushLine

Flush a level 2 cache line.

If the byte specified by the address (adr) is cached by the L2 cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory before the line is invalidated.

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_L2CacheFlushLine(u32 adr);
```

Parameters

The following table lists the `Xil_L2CacheFlushLine` function arguments.

Table 185: **Xil_L2CacheFlushLine Arguments**

Name	Description
adr	32bit address of the data/instruction to be flushed.

Returns

None.

Xil_L2CacheFlushRange

Flush the level 2 cache for the given address range.

If the bytes specified by the address range are cached by the L2 cache, the cacheline containing those bytes are invalidated. If the cachelines are modified (dirty), they are written to the system memory before the lines are invalidated.

Prototype

```
void Xil_L2CacheFlushRange(u32 adr, u32 len);
```

Parameters

The following table lists the `Xil_L2CacheFlushRange` function arguments.

Table 186: **Xil_L2CacheFlushRange Arguments**

Name	Description
adr	32bit start address of the range to be flushed.
len	Length of the range to be flushed in bytes.

Returns

None.

Xil_L2CacheStoreLine

Store a level 2 cache line.

If the byte specified by the address (adr) is cached by the L2 cache and the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory. After the store completes, the cacheline is marked as unmodified (not dirty).

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_L2CacheStoreLine(u32 adr);
```

Parameters

The following table lists the `Xil_L2CacheStoreLine` function arguments.

Table 187: **Xil_L2CacheStoreLine Arguments**

Name	Description
adr	32bit address of the data/instruction to be stored.

Returns

None.

Cortex A9 Processor MMU Functions

MMU functions equip users to enable MMU, disable MMU and modify default memory attributes of MMU table as per the need.

Table 188: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	INTPTR Addr u32 attrib
void	Xil_EnableMMU	None.
void	Xil_DisableMMU	None.
void *	Xil_MemMap	UINTPTR PhysAddr size_t size u32 flags

Functions

Xil_SetTlbAttributes

This function sets the memory attributes for a section covering 1MB of memory in the translation table.

***Note*:** The MMU or D-cache does not need to be disabled before changing a translation table entry.

Prototype

```
void Xil_SetTlbAttributes(INTPTR Addr, u32 attrib);
```

Parameters

The following table lists the `Xil_SetTlbAttributes` function arguments.

Table 189: **Xil_SetTlbAttributes Arguments**

Name	Description
Addr	32-bit address for which memory attributes need to be set.
attrib	Attribute for the given memory region. xil_mmu.h contains definitions of commonly used memory attributes which can be utilized for this function.

Returns

None.

Xil_EnableMMU

Enable MMU for cortex A9 processor.

This function invalidates the instruction and data caches, and then enables MMU.

Prototype

```
void Xil_EnableMMU(void);
```

Parameters

The following table lists the `Xil_EnableMMU` function arguments.

Table 190: **Xil_EnableMMU Arguments**

Name	Description
None.	

Returns

None.

Xil_DisableMMU

Disable MMU for Cortex A9 processors.

This function invalidates the TLBs, Branch Predictor Array and flushed the D Caches before disabling the MMU.

Note: When the MMU is disabled, all the memory accesses are treated as strongly ordered.

Prototype

```
void Xil_DisableMMU(void);
```

Parameters

The following table lists the `Xil_DisableMMU` function arguments.

Table 191: **Xil_DisableMMU Arguments**

Name	Description
None.	

Returns

None.

Xil_MemMap

Memory mapping for Cortex A9 processor.

Note: : Previously this was implemented in libmetal. Move to embeddedsw as this functionality is specific to A9 processor.

Prototype

```
void * Xil_MemMap(UINTPTR PhysAddr, size_t size, u32 flags);
```

Parameters

The following table lists the `Xil_MemMap` function arguments.

Table 192: **Xil_MemMap Arguments**

Name	Description
PhysAddr	is physical address.
size	is size of region.
flags	is flags used to set translation table.

Returns

Pointer to virtual address.

Cortex A9 Time Functions

xtime_l.h provides access to the 64-bit Global Counter in the PMU.

This counter increases by one at every two processor cycles. These functions can be used to get/set time in the global timer.

Table 193: **Quick Function Reference**

Type	Name	Arguments
void	XTime_SetTime	XTime Xtime_Global
void	XTime_GetTime	XTime * Xtime_Global

Functions

XTime_SetTime

Set the time in the Global Timer Counter Register.

Note: When this function is called by any one processor in a multi- processor environment, reference time will reset/lost for all processors.

Prototype

```
void XTime_SetTime(XTime Xtime_Global);
```

Parameters

The following table lists the `XTime_SetTime` function arguments.

Table 194: **XTime_SetTime Arguments**

Name	Description
Xtime_Global	64-bit Value to be written to the Global Timer Counter Register.

Returns

None.

XTime_GetTime

Get the time from the Global Timer Counter Register.

Note: None.

Prototype

```
void XTime_GetTime(XTime *Xtime_Global);
```

Parameters

The following table lists the `XTime_GetTime` function arguments.

Table 195: **XTime_GetTime Arguments**

Name	Description
Xtime_Global	Pointer to the 64-bit location which will be updated with the current timer value.

Returns

None.

Cortex A9 Event Counter Function

Cortex A9 event counter functions can be utilized to configure and control the Cortex-A9 performance monitor events.

Cortex-A9 performance monitor has six event counters which can be used to count a variety of events described in Coretx-A9 TRM. xpm_counter.h defines configurations XPM_CNTRCFGx which can be used to program the event counters to count a set of events.

***Note*:** It doesn't handle the Cortex-A9 cycle counter, as the cycle counter is being used for time keeping.

Table 196: **Quick Function Reference**

Type	Name	Arguments
void	Xpm_SetEvents	s32 PmcrCfg
void	Xpm_GetEventCounters	u32 * PmCtrValue

Functions

Xpm_SetEvents

This function configures the Cortex A9 event counters controller, with the event codes, in a configuration selected by the user and enables the counters.

***Note*:** None.

Prototype

```
void Xpm_SetEvents(s32 PmcrCfg);
```

Parameters

The following table lists the `Xpm_SetEvents` function arguments.

Table 197: **Xpm_SetEvents Arguments**

Name	Description
PmcrCfg	Configuration value based on which the event counters are configured. XPM_CNTRCFG* values defined in xpm_counter.h can be utilized for setting configuration.

Returns

None.

Xpm_GetEventCounters

This function disables the event counters and returns the counter values.

***Note*:** None.

Prototype

```
void Xpm_GetEventCounters(u32 *PmCtrValue);
```

Parameters

The following table lists the `Xpm_GetEventCounters` function arguments.

Table 198: **Xpm_GetEventCounters Arguments**

Name	Description
PmCtrValue	Pointer to an array of type u32 PmCtrValue[6]. It is an output parameter which is used to return the PM counter values.

Returns

None.

PL310 L2 Event Counters Functions

xl2cc_counter.h contains APIs for configuring and controlling the event counters in PL310 L2 cache controller.

PL310 has two event counters which can be used to count variety of events like DRHIT, DRREQ, DWHIT, DWREQ, etc. xl2cc_counter.h contains definitions for different configurations which can be used for the event counters to count a set of events.

Table 199: **Quick Function Reference**

Type	Name	Arguments
void	XL2cc_EventCtrInit	s32 Event0 s32 Event1
void	XL2cc_EventCtrStart	None.
void	XL2cc_EventCtrStop	u32 * EveCtr0

Functions

XL2cc_EventCtrInit

This function initializes the event counters in L2 Cache controller with a set of event codes specified by the user.

Note: The definitions for event codes XL2CC_* can be found in xl2cc_counter.h.

Prototype

```
void XL2cc_EventCtrInit(s32 Event0, s32 Event1);
```

Parameters

The following table lists the `XL2cc_EventCtrInit` function arguments.

Table 200: **XL2cc_EventCtrInit Arguments**

Name	Description
Event0	Event code for counter 0.
Event1	Event code for counter 1.

Returns

None.

XL2cc_EventCtrStart

This function starts the event counters in L2 Cache controller.

Note: None.

Prototype

```
void XL2cc_EventCtrStart(void);
```

Parameters

The following table lists the `XL2cc_EventCtrStart` function arguments.

Table 201: **XL2cc_EventCtrStart Arguments**

Name	Description
None.	

Returns

None.

XL2cc_EventCtrStop

This function disables the event counters in L2 Cache controller, saves the counter values and resets the counters.

Note: None.

Prototype

```
void XL2cc_EventCtrStop(u32 *EveCtr0, u32 *EveCtr1);
```

Parameters

The following table lists the `XL2cc_EventCtrStop` function arguments.

Table 202: **XL2cc_EventCtrStop Arguments**

Name	Description
EveCtr0	Output parameter which is used to return the value in event counter 0. EveCtr1: Output parameter which is used to return the value in event counter 1.

Returns

None.

Cortex A9 Processor and pl310 Errata Support

Various ARM errata are handled in the standalone BSP.

The implementation for errata handling follows ARM guidelines and is based on the open source Linux support for these errata.

***Note*:** The errata handling is enabled by default. To disable handling of all the errata globally, un-define the macro ENABLE_ARM_ERRATA in xil_errata.h. To disable errata on a per-erratum basis, un-define relevant macros in xil_errata.h.

Definitions

Define CONFIG_ARM_ERRATA_742230

Definition

```
#define CONFIG_ARM_ERRATA_7422301
```

Description

Errata No: 742230 Description: DMB operation may be faulty.

Define CONFIG_ARM_ERRATA_743622

Definition

```
#define CONFIG_ARM_ERRATA_7436221
```

Description

Errata No: 743622 Description: Faulty hazard checking in the Store Buffer may lead to data corruption.

Define CONFIG_ARM_ERRATA_775420

Definition

```
#define CONFIG_ARM_ERRATA_7754201
```

Description

Errata No: 775420 Description: A data cache maintenance operation which aborts, might lead to deadlock.

Define CONFIG_ARM_ERRATA_794073

Definition

```
#define CONFIG_ARM_ERRATA_7940731
```

Description

Errata No: 794073 Description: Speculative instruction fetches with MMU disabled might not comply with architectural requirements.

Define CONFIG_PL310_ERRATA_588369

Definition

```
#define CONFIG_PL310_ERRATA_5883691
```

Description

PL310 L2 Cache Errata.

Errata No: 588369 Description: Clean & Invalidate maintenance operations do not invalidate clean lines.

Define CONFIG_PL310_ERRATA_727915

Definition

```
#define CONFIG_PL310_ERRATA_7279151
```

Description

Errata No: 727915 Description: Background Clean and Invalidate by Way operation can cause data corruption.

Cortex A9 Processor Specific Include Files

The xpseudo_asm.h includes xreg_cortexa9.h and xpseudo_asm_gcc.h.

The xreg_cortexa9.h file contains definitions for inline assembler code. It provides inline definitions for Cortex A9 GPRs, SPRs, MPE registers, co-processor registers and Debug registers.

The xpseudo_asm_gcc.h contains the definitions for the most often used inline assembler instructions, available as macros. These can be very useful for tasks such as setting or getting special purpose registers, synchronization, or cache manipulation etc. These inline assembler instructions can be used from drivers and user applications written in C.

Cortex A53 32-bit Processor API

Cortex A53 32-bit Processor API

Cortex-A53 standalone BSP contains two separate BSPs for 32-bit mode and 64-bit mode.

The 32-bit mode of cortex-A53 is compatible with ARMv7-A architecture.

Cortex A53 32-bit Processor Boot Code

The boot.S file contains a minimal set of code for transferring control from the processor reset location to the start of the application. The boot code performs minimum configuration which is required for an application to run starting from processor reset state of the processor. Below is a sequence illustrating what all configuration is performed before control reaches to main function.

1. Program vector table base for exception handling
2. Invalidate instruction cache, data cache and TLBs
3. Program stack pointer for various modes (IRQ, FIQ, supervisor, undefine, abort, system)
4. Program counter frequency
5. Configure MMU with short descriptor translation table format and program base address of translation table
6. Enable data cache, instruction cache and MMU

7. Transfer control to _start which clears BSS sections and runs global constructor before jumping to main application

The translation_table.S contains a static page table required by MMU for cortex-A53. This translation table is flat mapped (input address = output address) with default memory attributes defined for zynq ultrascale+ architecture. It utilizes short descriptor translation table format with each section defining 1MB of memory.

For DDR in region 0x00000000 - 0x7FFFFFFF, a system where DDR is less than 2GB, region after DDR and before PL is marked as undefined/reserved in translation table. In region 0xFFC00000 - 0xFFDFFFFF, it contains CSU and PMU memory which are marked as Device since it is less than 1MB and falls in a region with device memory.

The overview of translation table memory attributes is described below.

	Memory Range	Definition in Translation Table
DDR	0x00000000 - 0x7FFFFFFF	Normal write-back Cacheable
PL	0x80000000 - 0xBFFFFFFF	Strongly Ordered
QSPI, lower PCIe	0xC0000000 - 0xEFFFFFFF	Device Memory
Reserved	0xF0000000 - 0xF7FFFFFF	Unassigned
STM Coresight	0xF8000000 - 0xF8FFFFFF	Device Memory
GIC	0xF9000000 - 0xF90FFFFF	Device memory
Reserved	0xF9100000 - 0xFCFFFFFF	Unassigned
FPS, LPS slaves	0xFD000000 - 0xFFBFFFFF	Device memory
CSU, PMU	0xFFC00000 - 0xFFDFFFFF	Device Memory
TCM, OCM	0xFFE00000 - 0xFFFFFFFF	Normal write-back cacheable

Cortex A53 32-bit Processor Cache Functions

Cache functions provide access to cache related operations such as flush and invalidate for instruction and data caches.

It gives option to perform the cache operations on a single cacheline, a range of memory and an entire cache.

Table 203: **Quick Function Reference**

Type	Name	Arguments
void	Xil_DCacheEnable	void
void	Xil_DCacheDisable	void
void	Xil_DCacheInvalidate	void

Table 203: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_DCacheInvalidateRange	INTPTR adr u32 len
void	Xil_DCacheFlush	void
void	Xil_DCacheFlushRange	INTPTR adr u32 len
void	Xil_DCacheInvalidateLine	u32 adr
void	Xil_DCacheFlushLine	u32 adr
void	Xil_ICacheInvalidateLine	u32 adr
void	Xil_ICacheEnable	void
void	Xil_ICacheDisable	void
void	Xil_ICacheInvalidate	void
void	Xil_ICacheInvalidateRange	INTPTR adr u32 len

Functions

Xil_DCacheEnable

Enable the Data cache.

Prototype

```
void Xil_DCacheEnable(void);
```

Returns

None.

Xil_DCacheDisable

Disable the Data cache.

Prototype

```
void Xil_DCacheDisable(void);
```

Returns

None.

Xil_DCacheInvalidate

Invalidate the Data cache.

The contents present in the data cache are cleaned and invalidated.

***Note*:** In Cortex-A53, functionality to simply invalid the cachelines is not present. Such operations are a problem for an environment that supports virtualisation. It would allow one OS to invalidate a line belonging to another OS. This could lead to the other OS crashing because of the loss of essential data. Hence, such operations are promoted to clean and invalidate to avoid such corruption.

Prototype

```
void Xil_DCacheInvalidate(void);
```

Returns

None.

Xil_DCacheInvalidateRange

Invalidate the Data cache for the given address range.

The cachelines present in the adderss range are cleaned and invalidated

***Note*:** In Cortex-A53, functionality to simply invalid the cachelines is not present. Such operations are a problem for an environment that supports virtualisation. It would allow one OS to invalidate a line belonging to another OS. This could lead to the other OS crashing because of the loss of essential data. Hence, such operations are promoted to clean and invalidate to avoid such corruption.

Prototype

```
void Xil_DCacheInvalidateRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_DCacheInvalidateRange` function arguments.

Table 204: **Xil_DCacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_DCacheFlush

Flush the Data cache.

Prototype

```
void Xil_DCacheFlush(void);
```

Returns

None.

Xil_DCacheFlushRange

Flush the Data cache for the given address range.

If the bytes specified by the address range are cached by the Data cache, the cachelines containing those bytes are invalidated. If the cachelines are modified (dirty), they are written to system memory before the lines are invalidated.

Prototype

```
void Xil_DCacheFlushRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_DCacheFlushRange` function arguments.

Table 205: **Xil_DCacheFlushRange Arguments**

Name	Description
adr	32bit start address of the range to be flushed.
len	Length of range to be flushed in bytes.

Returns

None.

Xil_DCacheInvalidateLine

Invalidate a Data cache line.

The cacheline is cleaned and invalidated.

***Note*:** In Cortex-A53, functionality to simply invalid the cachelines is not present. Such operations are a problem for an environment that supports virtualisation. It would allow one OS to invalidate a line belonging to another OS. This could lead to the other OS crashing because of the loss of essential data. Hence, such operations are promoted to clean and invalidate to avoid such corruption.

Prototype

```
void Xil_DCacheInvalidateLine(u32 adr);
```

Parameters

The following table lists the `Xil_DCacheInvalidateLine` function arguments.

Table 206: **Xil_DCacheInvalidateLine Arguments**

Name	Description
adr	32 bit address of the data to be invalidated.

Returns

None.

Xil_DCacheFlushLine

Flush a Data cache line.

If the byte specified by the address (adr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory before the line is invalidated.

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheFlushLine(u32 adr);
```

Parameters

The following table lists the `Xil_DCacheFlushLine` function arguments.

Table 207: **Xil_DCacheFlushLine Arguments**

Name	Description
adr	32bit address of the data to be flushed.

Returns

None.

Xil_ICacheInvalidateLine

Invalidate an instruction cache line.

If the instruction specified by the address is cached by the instruction cache, the cachecline containing that instruction is invalidated.

***Note*:** The bottom 4 bits are set to 0, forced by architecture.

Prototype

```
void Xil_ICacheInvalidateLine(u32 adr);
```

Parameters

The following table lists the `Xil_ICacheInvalidateLine` function arguments.

Table 208: **Xil_ICacheInvalidateLine Arguments**

Name	Description
adr	32bit address of the instruction to be invalidated..

Returns

None.

Xil_ICacheEnable

Enable the instruction cache.

Prototype

```
void Xil_ICacheEnable(void);
```

Returns

None.

Xil_ICacheDisable

Disable the instruction cache.

Prototype

```
void Xil_ICacheDisable(void);
```

Returns

None.

Xil_ICacheInvalidate

Invalidate the entire instruction cache.

Prototype

```
void Xil_ICacheInvalidate(void);
```

Returns

None.

Xil_ICacheInvalidateRange

Invalidate the instruction cache for the given address range.

If the instructions specified by the address range are cached by the instrunction cache, the cachelines containing those instructions are invalidated.

Prototype

```
void Xil_ICacheInvalidateRange(INTPTR adr, u32 len);
```

Parameters

The following table lists the `Xil_ICacheInvalidateRange` function arguments.

Table 209: **Xil_ICacheInvalidateRange Arguments**

Name	Description
adr	32bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Cortex A53 32-bit Processor MMU Handling

MMU functions equip users to enable MMU, disable MMU and modify default memory attributes of MMU table as per the need.

None.

***Note*:**

Table 210: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	UINTPTR Addr u32 attrib
void	Xil_EnableMMU	None.
void	Xil_DisableMMU	None.

Functions

Xil_SetTlbAttributes

This function sets the memory attributes for a section covering 1MB of memory in the translation table.

***Note*:** The MMU or D-cache does not need to be disabled before changing a translation table entry.

Prototype

```
void Xil_SetTlbAttributes(UINTPTR Addr, u32 attrib);
```

Parameters

The following table lists the `Xil_SetTlbAttributes` function arguments.

Table 211: **Xil_SetTlbAttributes Arguments**

Name	Description
Addr	32-bit address for which the attributes need to be set.
attrib	Attributes for the specified memory region. xil_mmu.h contains commonly used memory attributes definitions which can be utilized for this function.

Returns

None.

Xil_EnableMMU

Enable MMU for Cortex-A53 processor in 32bit mode.

This function invalidates the instruction and data caches before enabling MMU.

Prototype

```
void Xil_EnableMMU(void);
```

Parameters

The following table lists the `Xil_EnableMMU` function arguments.

Table 212: **Xil_EnableMMU Arguments**

Name	Description
None.	

Returns

None.

Xil_DisableMMU

Disable MMU for Cortex A53 processors in 32bit mode.

This function invalidates the TLBs, Branch Predictor Array and flushed the data cache before disabling the MMU.

Note: When the MMU is disabled, all the memory accesses are treated as strongly ordered.

Prototype

```
void Xil_DisableMMU(void);
```

Parameters

The following table lists the `Xil_DisableMMU` function arguments.

Table 213: **Xil_DisableMMU Arguments**

Name	Description
None.	

Returns

None.

Cortex A53 32-bit Mode Time Functions

xtime_l.h provides access to the 64-bit physical timer counter.

Table 214: **Quick Function Reference**

Type	Name	Arguments
u64	arch_counter_get_cntvct	void
void	XTime_StartTimer	None.
void	XTime_SetTime	XTime Xtime_Global
void	XTime_GetTime	XTime * Xtime_Global

Functions

arch_counter_get_cntvct

Prototype

```
u64 arch_counter_get_cntvct(void);
```

XTime_StartTimer

Start the 64-bit physical timer counter.

Note: The timer is initialized only if it is disabled. If the timer is already running this function does not perform any operation.

Prototype

```
void XTime_StartTimer(void);
```

Parameters

The following table lists the `XTime_StartTimer` function arguments.

Table 215: **XTime_StartTimer Arguments**

Name	Description
None.	

Returns

None.

XTime_SetTime

Timer of A53 runs continuously and the time can not be set as desired.

This API doesn't contain anything. It is defined to have uniformity across platforms.

Note: None.

Prototype

```
void XTime_SetTime(XTime Xtime_Global);
```

Parameters

The following table lists the `XTime_SetTime` function arguments.

Table 216: **XTime_SetTime Arguments**

Name	Description
Xtime_Global	64bit Value to be written to the Global Timer Counter Register. But since the function does not contain anything, the value is not used for anything.

Returns

None.

XTime_GetTime

Get the time from the physical timer counter register.

Note: None.

Prototype

```
void XTime_GetTime(XTime *Xtime_Global);
```

Parameters

The following table lists the `XTime_GetTime` function arguments.

Table 217: **XTime_GetTime Arguments**

Name	Description
Xtime_Global	Pointer to the 64-bit location to be updated with the current value in physical timer counter.

Returns

None.

Cortex A53 32-bit Processor Specific Include Files

The xpseudo_asm.h includes xreg_cortexa53.h and xpseudo_asm_gcc.h.

The xreg_cortexa53.h file contains definitions for inline assembler code. It provides inline definitions for Cortex A53 GPRs, SPRs, co-processor registers and floating point registers.

The xpseudo_asm_gcc.h contains the definitions for the most often used inline assembler instructions, available as macros. These can be very useful for tasks such as setting or getting special purpose registers, synchronization, or cache manipulation etc. These inline assembler instructions can be used from drivers and user applications written in C.

Cortex A53 64-bit Processor API

Cortex A53 64-bit Processor API

Cortex-A53 standalone BSP contains two separate BSPs for 32-bit mode and 64-bit mode. The 64-bit mode of cortex-A53 contains ARMv8-A architecture. This section provides a linked summary and detailed descriptions of the Cortex A53 64-bit Processor APIs.

Note: These APIs are applicable for the Cortex-A72 processor as well.

Cortex A53 64-bit Processor Boot Code

The boot code performs minimum configuration which is required for an application. Cortex-A53 starts by checking current exception level. If the current exception level is EL3 and BSP is built for EL3, it will do initialization required for application execution at EL3. Below is a sequence illustrating what all configuration is performed before control reaches to main function for EL3 execution.

1. Program vector table base for exception handling
2. Set reset vector table base address
3. Program stack pointer for EL3
4. Routing of interrupts to EL3
5. Enable ECC protection
6. Program generic counter frequency

7. Invalidate instruction cache, data cache and TLBs
8. Configure MMU registers and program base address of translation table
9. Transfer control to _start which clears BSS sections and runs global constructor before jumping to main application

If the current exception level is EL1 and BSP is also built for EL1_NONSECURE it will perform initialization required for application execution at EL1 non-secure. For all other combination, the execution will go into infinite loop. Below is a sequence illustrating what all configuration is performed before control reaches to main function for EL1 execution.

1. Program vector table base for exception handling
2. Program stack pointer for EL1
3. Invalidate instruction cache, data cache and TLBs
4. Configure MMU registers and program base address of translation table
5. Transfer control to _start which clears BSS sections and runs global constructor before jumping to main application

Cortex A53 64-bit Processor Cache Functions

Cache functions provide access to cache related operations such as flush and invalidate for instruction and data caches.

It gives option to perform the cache operations on a single cacheline, a range of memory and an entire cache.

Table 218: **Quick Function Reference**

Type	Name	Arguments
void	Xil_DCacheEnable	void
void	Xil_DCacheDisable	void
void	Xil_DCacheInvalidate	void
void	Xil_DCacheInvalidateRange	INTPTR adr INTPTR len
void	Xil_DCacheInvalidateLine	INTPTR adr
void	Xil_DCacheFlush	void

Table 218: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_DCacheFlushLine	INTPTR adr
void	Xil_ICacheEnable	void
void	Xil_ICacheDisable	void
void	Xil_ICacheInvalidate	void
void	Xil_ICacheInvalidateRange	INTPTR adr INTPTR len
void	Xil_ICacheInvalidateLine	INTPTR adr
void	Xil_ConfigureL1Prefetch	u8 num

Functions

Xil_DCacheEnable

Enable the Data cache.

Prototype

```
void Xil_DCacheEnable(void);
```

Returns

None.

Xil_DCacheDisable

Disable the Data cache.

Prototype

```
void Xil_DCacheDisable(void);
```

Returns

None.

Xil_DCacheInvalidate

Invalidate the Data cache.

The contents present in the cache are cleaned and invalidated.

***Note*:** In Cortex-A53, functionality to simply invalid the cachelines is not present. Such operations are a problem for an environment that supports virtualisation. It would allow one OS to invalidate a line belonging to another OS. This could lead to the other OS crashing because of the loss of essential data. Hence, such operations are promoted to clean and invalidate which avoids such corruption.

Prototype

```
void Xil_DCacheInvalidate(void);
```

Returns

None.

Xil_DCacheInvalidateRange

Invalidate the Data cache for the given address range.

The cachelines present in the adderss range are cleaned and invalidated

***Note*:** In Cortex-A53, functionality to simply invalid the cachelines is not present. Such operations are a problem for an environment that supports virtualisation. It would allow one OS to invalidate a line belonging to another OS. This could lead to the other OS crashing because of the loss of essential data. Hence, such operations are promoted to clean and invalidate which avoids such corruption.

Prototype

```
void Xil_DCacheInvalidateRange(INTPTR adr, INTPTR len);
```

Parameters

The following table lists the `Xil_DCacheInvalidateRange` function arguments.

Table 219: **Xil_DCacheInvalidateRange Arguments**

Name	Description
adr	64bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_DCacheInvalidateLine

Invalidate a Data cache line.

The cacheline is cleaned and invalidated.

***Note*:** In Cortex-A53, functionality to simply invalid the cachelines is not present. Such operations are a problem for an environment that supports virtualisation. It would allow one OS to invalidate a line belonging to another OS. This could lead to the other OS crashing because of the loss of essential data. Hence, such operations are promoted to clean and invalidate which avoids such corruption.

Prototype

```
void Xil_DCacheInvalidateLine(INTPTR adr);
```

Parameters

The following table lists the `Xil_DCacheInvalidateLine` function arguments.

Table 220: **Xil_DCacheInvalidateLine Arguments**

Name	Description
adr	64bit address of the data to be flushed.

Returns

None.

Xil_DCacheFlush

Flush the Data cache.

Prototype

```
void Xil_DCacheFlush(void);
```

Returns

None.

Xil_DCacheFlushLine

Flush a Data cache line.

If the byte specified by the address (adr) is cached by the Data cache, the cacheline containing that byte is invalidated. If the cacheline is modified (dirty), the entire contents of the cacheline are written to system memory before the line is invalidated.

***Note*:** The bottom 6 bits are set to 0, forced by architecture.

Prototype

```
void Xil_DCacheFlushLine(INTPTR adr);
```

Parameters

The following table lists the `Xil_DCacheFlushLine` function arguments.

Table 221: **Xil_DCacheFlushLine Arguments**

Name	Description
adr	64bit address of the data to be flushed.

Returns

None.

Xil_ICacheEnable

Enable the instruction cache.

Prototype

```
void Xil_ICacheEnable(void);
```

Returns

None.

Xil_ICacheDisable

Disable the instruction cache.

Prototype

```
void Xil_ICacheDisable(void);
```

Returns

None.

Xil_ICacheInvalidate

Invalidate the entire instruction cache.

Prototype

```
void Xil_ICacheInvalidate(void);
```

Returns

None.

Xil_ICacheInvalidateRange

Invalidate the instruction cache for the given address range.

If the instructions specified by the address range are cached by the instrunction cache, the cachelines containing those instructions are invalidated.

Prototype

```
void Xil_ICacheInvalidateRange(INTPTR adr, INTPTR len);
```

Parameters

The following table lists the `Xil_ICacheInvalidateRange` function arguments.

Table 222: **Xil_ICacheInvalidateRange Arguments**

Name	Description
adr	64bit start address of the range to be invalidated.
len	Length of the range to be invalidated in bytes.

Returns

None.

Xil_ICacheInvalidateLine

Invalidate an instruction cache line.

If the instruction specified by the parameter adr is cached by the instruction cache, the cacheline containing that instruction is invalidated.

***Note*:** The bottom 6 bits are set to 0, forced by architecture.

Prototype

```
void Xil_ICacheInvalidateLine(INTPTR adr);
```

Parameters

The following table lists the `Xil_ICacheInvalidateLine` function arguments.

Table 223: **Xil_ICacheInvalidateLine Arguments**

Name	Description
adr	64bit address of the instruction to be invalidated.

Returns

None.

Xil_ConfigureL1Prefetch

Configure the maximum number of outstanding data prefetches allowed in L1 cache.

Note: This function is implemented only for EL3 privilege level.

Prototype

```
void Xil_ConfigureL1Prefetch(u8 num);
```

Parameters

The following table lists the `Xil_ConfigureL1Prefetch` function arguments.

Table 224: **Xil_ConfigureL1Prefetch Arguments**

Name	Description
num	maximum number of outstanding data prefetches allowed, valid values are 0-7.

Returns

None.

Cortex A53 64-bit Processor MMU Handling

MMU function equip users to modify default memory attributes of MMU table as per the need.

None.

Note:

Table 225: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	UINTPTR Addr u64 attrib

Functions

Xil_SetTlbAttributes

It sets the memory attributes for a section, in the translation table.

If the address (defined by Addr) is less than 4GB, the memory attribute(attrib) is set for a section of 2MB memory. If the address (defined by Addr) is greater than 4GB, the memory attribute (attrib) is set for a section of 1GB memory.

Note: The MMU and D-cache need not be disabled before changing an translation table attribute.

Prototype

```
void Xil_SetTlbAttributes(UINTPTR Addr, u64 attrib);
```

Parameters

The following table lists the `Xil_SetTlbAttributes` function arguments.

Table 226: **Xil_SetTlbAttributes Arguments**

Name	Description
Addr	64-bit address for which attributes are to be set.
attrib	Attribute for the specified memory region. xil_mmu.h contains commonly used memory attributes definitions which can be utilized for this function.

Returns

None.

Cortex A53 64-bit Mode Time Functions

xtime_l.h provides access to the 64-bit physical timer counter.

Table 227: **Quick Function Reference**

Type	Name	Arguments
void	XTime_StartTimer	None.
void	XTime_SetTime	XTime Xtime_Global
void	XTime_GetTime	XTime * Xtime_Global

Functions

XTime_StartTimer

Start the 64-bit physical timer counter.

***Note*:** The timer is initialized only if it is disabled. If the timer is already running this function does not perform any operation. This API is effective only if BSP is built for EL3. For EL1 Non-secure, it simply exits.

Prototype

```
void XTime_StartTimer(void);
```

Parameters

The following table lists the `XTime_StartTimer` function arguments.

Table 228: **XTime_StartTimer Arguments**

Name	Description
None.	

Returns

None.

XTime_SetTime

Timer of A53 runs continuously and the time can not be set as desired.

This API doesn't contain anything. It is defined to have uniformity across platforms.

***Note*:** None.

Prototype

```
void XTime_SetTime(XTime Xtime_Global);
```

Parameters

The following table lists the `XTime_SetTime` function arguments.

Table 229: **XTime_SetTime Arguments**

Name	Description
Xtime_Global	64bit value to be written to the physical timer counter register. Since API does not do anything, the value is not utilized.

Returns

None.

XTime_GetTime

Get the time from the physical timer counter register.

Note: None.

Prototype

```
void XTime_GetTime(XTime *Xtime_Global);
```

Parameters

The following table lists the `XTime_GetTime` function arguments.

Table 230: **XTime_GetTime Arguments**

Name	Description
Xtime_Global	Pointer to the 64-bit location to be updated with the current value of physical timer counter register.

Returns

None.

Cortex A53 64-bit Processor Specific Include Files

The xpseudo_asm.h includes xreg_cortexa53.h and xpseudo_asm_gcc.h.

The xreg_cortexa53.h file contains definitions for inline assembler code. It provides inline definitions for Cortex A53 GPRs, SPRs and floating point registers.

The xpseudo_asm_gcc.h contains the definitions for the most often used inline assembler instructions, available as macros. These can be very useful for tasks such as setting or getting special purpose registers, synchronization, or cache manipulation etc. These inline assembler instructions can be used from drivers and user applications written in C.

XILINX

Chapter 4

LwIP 2.1.1 Library

Introduction

The lwIP is an open source TCP/IP protocol suite available under the BSD license. The lwIP is a standalone stack; there are no operating systems dependencies, although it can be used along with operating systems. The lwIP provides two A05PIs for use by applications:

- RAW API: Provides access to the core lwIP stack.
- Socket API: Provides a BSD sockets style interface to the stack.

The lwip211_v1.3 is built on the open source lwIP library version 2.1.1. The lwip211 library provides adapters for the Ethernetlite (axi_ethernetlite), the TEMAC (axi_ethernet), and the Gigabit Ethernet controller and MAC (GigE) cores. The library can run on MicroBlaze™, Arm® Cortex-A9, Arm® Cortex-A53, Arm® Cortex-A72, and Arm® Cortex-R5F processors. The Ethernetlite and TEMAC cores apply for MicroBlaze™ systems. The Gigabit Ethernet controller and MAC (GigE) core is applicable only for Arm Cortex-A9 system (Zynq-7000 processor devices) and Arm Cortex-A53 & Arm Cortex-R5F system (Zynq® UltraScale+™ MPSoC), and Arm Cortex-A72 and Arm Cortex-R5F system (Versal™ ACAP).

Features

The lwIP provides support for the following protocols:

- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- User Datagram Protocol (UDP)
- TCP (Transmission Control Protocol (TCP)
- Address Resolution Protocol (ARP)
- Dynamic Host Configuration Protocol (DHCP)
- Internet Group Message Protocol (IGMP)

References

- FreeRTOS: http://www.freertos.org/Interactive_Frames/Open_Frames.html?http://interactive.freertos.org/forums
- lwIP wiki: http://lwip.scribblewiki.com
- Xilinx® lwIP designs and application examples: http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf
- lwIP examples using RAW and Socket APIs: http://savannah.nongnu.org/projects/lwip/
- FreeRTOS Port for Zynq is available for download from the [FreeRTOS][freertos] website

Using lwIP

Overview

The following are the key steps to use lwIP for networking:

1. Creating a hardware system containing the processor, ethernet core, and a timer. The timer and ethernet interrupts must be connected to the processor using an interrupt controller.
2. Configuring lwip211_v1.3 to be a part of the software platform. For operating with lwIP socket API, the Xilkernel library or FreeRTOS BSP is a prerequisite. See the Note below.

***Note*:** The Xilkernel library is available only for MicroBlaze systems. For Cortex-A9 based systems (Zynq devices), Cortex-A53 or Cortex-R5F based systems (Zynq UltraScale+ MPSoC), and Arm Cortex-A72 and Arm Cortex-R5F system (Versal ACAP). There is no support for Xilkernel. Instead, use FreeRTOS. A FreeRTOS BSP is available for Zynq, Zynq UltraScale+ MPSoC, and Versal systems and must be included for using lwIP socket API. The FreeRTOS BSP for Zynq devices, Zynq UltraScale+ MPSoC, and Versal ACAP is available for download from the the FreeRTOS website.

Setting up the Hardware System

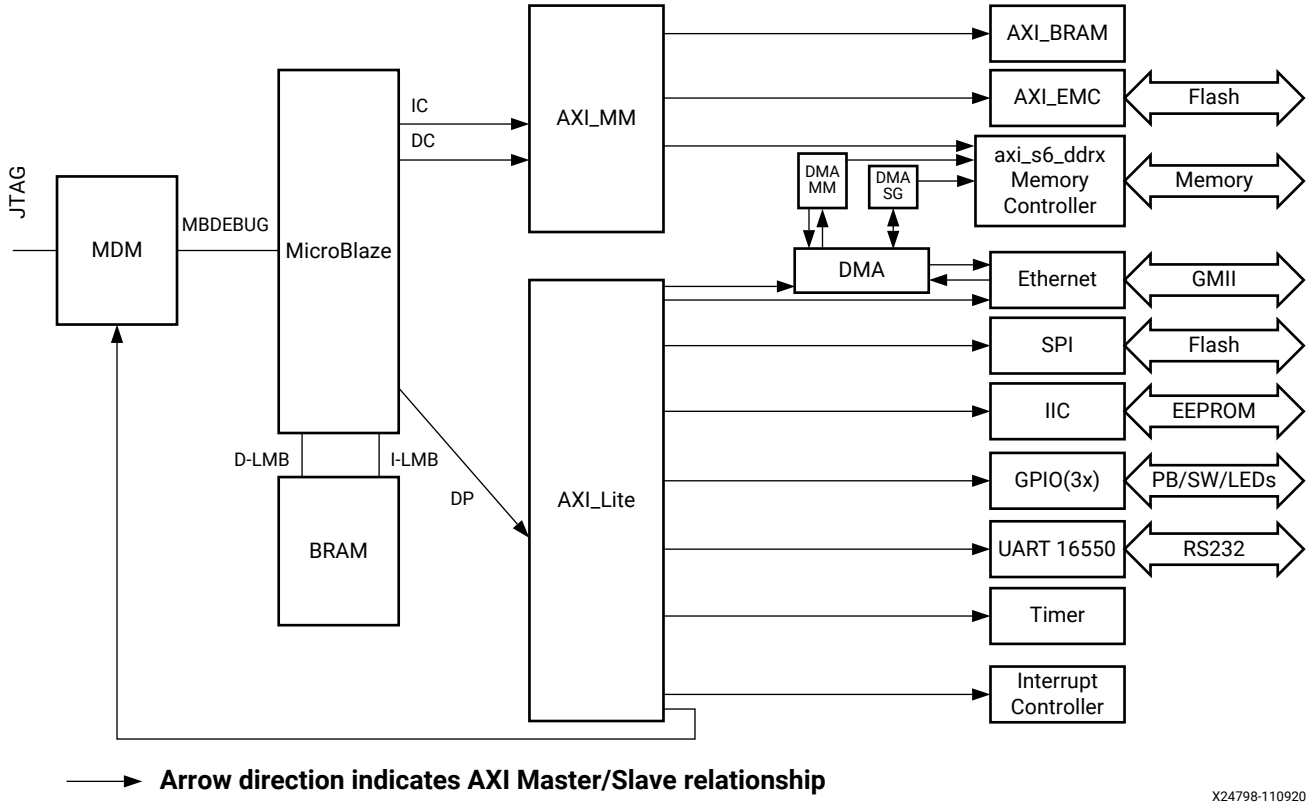
This section describes the hardware configurations supported by lwIP. The key components of the hardware system include:

- **Processor:** Either a MicroBlaze or a Cortex-A9 or a Cortex-A53 or a Cortex-R5F processor or a Cortex-A72. The Cortex-A9 processor applies to Zynq systems. The Cortex-A53 and Cortex-R5F processors apply to Zynq UltraScale+ MPSoC systems. The Cortex-A72 and Cortex-R5F processors apply to Versal ACAP systems.
- **MAC:** LwIP supports axi_ethernetlite, axi_ethernet, and Gigabit Ethernet controller and MAC (GigE) cores.

- **Timer:** To maintain TCP timers, lwIP raw API based applications require that certain functions are called at periodic intervals by the application. An application can do this by registering an interrupt handler with a timer.
- **DMA:** For axi_ethernet based systems, the axi_ethernet cores can be configured with a soft DMA engine (AXI DMA and MCDMA) or a FIFO interface. For GigE-based Zynq devices, Zynq UltraScale+ MPSoC, and Versal ACAP systems, there is a built-in DMA and so no extra configuration is needed. Same applies to axi_ethernetlite based systems, which have their built-in buffer management provisions.

The following figure shows a sample system architecture with a Kintex-6 device using the axi_ethernet core with DMA.

Figure 1: **System Architecture using axi_ethernet core with DMA**



Setting up the Software System

To use lwIP in a software application, you must first compile the lwIP library as a part of the software application.

1. Click File > New > Platform Project.
2. Click Specify to create a new Hardware Platform Specification.

3. Provide a new name for the domain in the Project name field if you wish to override the default value.
4. Select the location for the board support project files. To use the default location, as displayed in the Location field, leave the Use default location check box selected. Otherwise, deselect the checkbox and then type or browse to the directory location.
5. From the Hardware Platform drop-down choose the appropriate platform for your application or click the New button to browse to an existing Hardware Platform.
6. Select the target CPU from the drop-down list.
7. From the Board Support Package OS list box, select the type of board support package to create. A description of the platform types displays in the box below the drop-down list.
8. Click Finish. The wizard creates a new software platform and displays it in the Vitis Navigator pane.
9. Select Project > Build Automatically to automatically build the board support package. The Board Support Package Settings dialog box opens. Here you can customize the settings for the domain.
10. Click OK to accept the settings, build the platform, and close the dialog box.
11. From the Explorer, double-click platform.spr file and select the appropriate domain/board support package. The overview page opens.
12. In the overview page, click Modify BSP Settings.
13. Using the Board Support Package Settings page, you can select the OS Version and which of the Supported Libraries are to be enabled in this domain/BSP.
14. Select the lwip211 library from the list of Supported Libraries.
15. Expand the Overview tree and select lwip211. The configuration options for the lwip211 library are listed.
16. Configure the lwip211 library and click OK.

Configuring lwIP Options

The lwIP library provides configurable parameters. There are two major categories of configurable options:

- Xilinx Adapter to lwIP options: These control the settings used by Xilinx adapters for the ethernet cores.
- Base lwIP options: These options are part of lwIP library itself, and include parameters for TCP, UDP, IP and other protocols supported by lwIP. The following sections describe the available lwIP configurable options.

Customizing lwIP API Mode

The lwip211_v1.3 supports both raw API and socket API:

- The raw API is customized for high performance and lower memory overhead. The limitation of raw API is that it is callback-based, and consequently does not provide portability to other TCP stacks.
- The socket API provides a BSD socket-style interface and is very portable; however, this mode is not as efficient as raw API mode in performance and memory requirements. The lwip211_v1.3 also provides the ability to set the priority on TCP/IP and other lwIP application threads.

The following table describes the lwIP library API mode options.

Attribute	Description	Type	Default
api_mode {RAW_API | SOCKET_API}	The lwIP library mode of operation	enum	RAW_API
socket_mode_thread_prio	Priority of lwIP TCP/IP thread and all lwIP application threads. This setting applies only when Xilkernel is used in priority mode. It is recommended that all threads using lwIP run at the same priority level. For GigE based Zynq-7000, Zynq UltraScale+ MPSoC, and Versal systems using FreeRTOS, appropriate priority should be set. The default priority of 1 will not give the expected behaviour. For FreeRTOS (Zynq-7000, Zynq UltraScale+ MPSoC, and Versal systems), all internal lwIP tasks (except the main TCP/IP task) are created with the priority level set for this attribute. The TCP/IP task is given a higher priority than other tasks for improved performance. The typical TCP/IP task priority is 1 more than the priority set for this attribute for FreeRTOS.	integer	1
use_axieth_on_zynq	In the event that the AxiEthernet soft IP is used on a Zynq-7000 device or a Zynq UltraScale+ MPSoC device. This option ensures that the GigE on the Zynq-7000 PS (EmacPs) is not enabled and the device uses the AxiEthernet soft IP for Ethernet traffic. The existing Xilinx-provided lwIP adapters are not tested for multiple MACs. Multiple Axi Ethernet's are not supported on Zynq UltraScale+ MPSOC devices.	integer	0 = Use Zynq-7000 PS-based or ZynMP PS-based GigE controller 1= User AxiEthernet

Configuring Xilinx Adapter Options

The Xilinx adapters for EMAC/GigE cores are configurable.

Ethernetlite Adapter Options

The following table describes the configuration parameters for the axi_ethernetlite adapter.

Attribute	Description	Type	Default
sw_rx_fifo_size	Software Buffer Size in bytes of the receive data between EMAC and processor	integer	8192
sw_tx_fifo_size	Software Buffer Size in bytes of the transmit data between processor and EMAC	integer	8192

TEMAC Adapter Options

The following table describes the configuration parameters for the axi_ethernet and GigE adapters.

Attribute	Type	Description
n_tx_descriptors	integer	Number of TX descriptors to be used. For high performance systems there might be a need to use a higher value. Default is 64.
n_rx_descriptors	integer	Number of RX descriptors to be used. For high performance systems there might be a need to use a higher value. Typical values are 128 and 256. Default is 64.
n_tx_coalesce	integer	Setting for TX interrupt coalescing. Default is 1.
n_rx_coalesce	integer	Setting for RX interrupt coalescing. Default is 1.
tcp_rx_checksum_offload	boolean	Offload TCP Receive checksum calculation (hardware support required). For GigE in Zynq devices, Zynq UltraScale+ MPSoC, and Versal ACAP, the TCP receive checksum offloading is always present, so this attribute does not apply. Default is false.
tcp_tx_checksum_offload	boolean	Offload TCP Transmit checksum calculation (hardware support required). For GigE cores (Zynq devices, Zynq UltraScale+ MPSoC, and Versal ACAP), the TCP transmit checksum offloading is always present, so this attribute does not apply. Default is false.

Attribute	Type	Description
tcp_ip_rx_checksum_ofload	boolean	Offload TCP and IP Receive checksum calculation (hardware support required). Applicable only for AXI systems. For GigE in Zynq devices, Zynq UltraScale+ MPSoC, and Versal ACAP, the TCP and IP receive checksum offloading is always present, so this attribute does not apply. Default is false.
tcp_ip_tx_checksum_ofload	boolean	Offload TCP and IP Transmit checksum calculation (hardware support required). Applicable only for AXI systems. For GigE in Zynq, Zynq UltraScale+ MPSoC, and Versal ACAP, the TCP and IP transmit checksum offloading is always present, so this attribute does not apply. Default is false.
phy_link_speed	CONFIG_LINKSPEED_ AUTODETECT	Link speed as auto-negotiated by the PHY. lwIP configures the TEMAC/GigE for this speed setting. This setting must be correct for the TEMAC/GigE to transmit or receive packets. The CONFIG_LINKSPEED_ AUTODETECT setting attempts to detect the correct linkspeed by reading the PHY registers; however, this is PHY dependent, and has been tested with the Marvell and TI PHYs present on Xilinx development boards. For other PHYs, select the correct speed. Default is enum.
temac_use_jumbo_ frames_experimental	boolean	Use TEMAC jumbo frames (with a size up to 9k bytes). If this option is selected, jumbo frames are allowed to be transmitted and received by the TEMAC. For GigE in Zynq there is no support for jumbo frames, so this attribute does not apply. Default is false.

Configuring Memory Options

The lwIP stack provides different kinds of memories. Similarly, when the application uses socket mode, different memory options are used. All the configurable memory options are provided as a separate category. Default values work well unless application tuning is required. The following table describes the memory parameter options.

Attribute	Default	Type	Description
mem_size	131072	Integer	Total size of the heap memory available, measured in bytes. For applications which use a lot of memory from heap (using C library malloc or lwIP routine mem_malloc or pbuf_alloc with PBUF_RAM option), this number should be made higher as per the requirements.
memp_n_pbuf	16	Integer	The number of memp struct pbufs. If the application sends a lot of data out of ROM (or other static memory), this should be set high.
memp_n_udp_pcb	4	Integer	The number of UDP protocol control blocks. One per active UDP connection.
memp_n_tcp_pcb	32	Integer	The number of simultaneously active TCP connections.
memp_n_tcp_pcb _listen	8	Integer	The number of listening TC connections.
memp_n_tcp_seg	256	Integer	The number of simultaneously queued TCP segments.
memp_n_sys_timeout	8	Integer	Number of simultaneously active timeouts.
memp_num_netbuf	8	Integer	Number of allowed structure instances of type netbufs. Applicable only in socket mode.
memp_num_netconn	16	Integer	Number of allowed structure instances of type netconns. Applicable only in socket mode.
memp_num_api_msg	16	Integer	Number of allowed structure instances of type api_msg. Applicable only in socket mode.
memp_num_tcpip_msg	64	Integer	Number of TCPIP msg structures (socket mode only).

***Note*:** Because Sockets Mode support uses Xilkernel services, the number of semaphores chosen in the Xilkernel configuration must take the value set for the memp_num_netbuf parameter into account. For FreeRTOS BSP there is no setting for the maximum number of semaphores. For FreeRTOS, you can create semaphores as long as memory is available.

Configuring Packet Buffer (Pbuf) Memory Options

Packet buffers (Pbufs) carry packets across various layers of the TCP/IP stack. The following are the pbuf memory options provided by the lwIP stack. Default values work well unless application tuning is required. The following table describes the parameters for the Pbuf memory options.

Attribute	Default	Type	Description
pbuf_pool_size	256	Integer	Number of buffers in pbuf pool. For high performance systems, you might consider increasing the pbuf pool size to a higher value, such as 512.
pbuf_pool_bufsize	1700	Integer	Size of each pbuf in pbuf pool. For systems that support jumbo frames, you might consider using a pbuf pool buffer size that is more than the maximum jumbo frame size.
pbuf_link_hlen	16	Integer	Number of bytes that should be allocated for a link level header.

Configuring ARP Options

The following table describes the parameters for the ARP options. Default values work well unless application tuning is required.

Attribute	Default	Type	Description
arp_table_size	10	Integer	Number of active hardware address IP address pairs cached.
arp_queueing	1	Integer	If enabled outgoing packets are queued during hardware address resolution. This attribute can have two values: 0 or 1.

Configuring IP Options

The following table describes the IP parameter options. Default values work well unless application tuning is required.

Attribute	Default	Type	Description
ip_forward	0	Integer	Set to 1 for enabling ability to forward IP packets across network interfaces. If running lwIP on a single network interface, set to 0. This attribute can have two values: 0 or 1.

Attribute	Default	Type	Description
ip_options	0	Integer	When set to 1, IP options are allowed (but not parsed). When set to 0, all packets with IP options are dropped. This attribute can have two values: 0 or 1.
ip_reassembly	1	Integer	Reassemble incoming fragmented IP packets.
ip_frag	1	Integer	Fragment outgoing IP packets if their size exceeds MTU.
ip_reass_max_pbufs	128	Integer	Reassembly pbuf queue length.
ip_frag_max_mtu	1500	Integer	Assumed max MTU on any interface for IP fragmented buffer.
ip_default_ttl	255	Integer	Global default TTL used by transport layers.

Configuring ICMP Options

The following table describes the parameter for ICMP protocol option. Default values work well unless application tuning is required.

For GigE cores (for Zynq and Zynq MPSoC) there is no support for ICMP in the hardware.

Attribute	Default	Type	Description
icmp_ttl	255	Integer	ICMP TTL value.

Configuring IGMP Options

The IGMP protocol is supported by lwIP stack. When set true, the following option enables the IGMP protocol.

Attribute	Default	Type	Description
imgp_options	false	Boolean	Specify whether IGMP is required.

Configuring UDP Options

The following table describes UDP protocol options. Default values work well unless application tuning is required.

Attribute	Default	Type	Description
lwip_udp	true	Boolean	Specify whether UDP is required.
udp_ttl	255	Integer	UDP TTL value.

Configuring TCP Options

The following table describes the TCP protocol options. Default values work well unless application tuning is required.

Attribute	Default	Type	Description
lwip_tcp	true	Boolean	Require TCP.
tcp_ttl	255	Integer	TCP TTL value.
tcp_wnd	2048	Integer	TCP Window size in bytes.
tcp_maxrtx	12	Integer	TCP Maximum retransmission value.
tcp_synmaxrtx	4	Integer	TCP Maximum SYN retransmission value.
tcp_queue_ooseq	1	Integer	Accept TCP queue segments out of order. Set to 0 if your device is low on memory.
tcp_mss	1460	Integer	TCP Maximum segment size.
tcp_snd_buf	8192	Integer	TCP sender buffer space in bytes.

Configuring DHCP Options

The DHCP protocol is supported by lwIP stack. The following table describes DHCP protocol options. Default values work well unless application tuning is required.

Attribute	Default	Type	Description
lwip_dhcp	false	Boolean	Specify whether DHCP is required.
dhcp_does_arp_check	false	Boolean	Specify whether ARP checks on offered addresses.

Configuring the Stats Option

lwIP stack has been written to collect statistics, such as the number of connections used; amount of memory used; and number of semaphores used, for the application. The library provides the stats_display() API to dump out the statistics relevant to the context in which the call is used. The stats option can be turned on to enable the statistics information to be collected and displayed when the stats_display API is called from user code. Use the following option to enable collecting the stats information for the application.

Attribute	Description	Type	Default
lwip_stats	Turn on lwIP Statistics	int	0

Configuring the Debug Option

lwIP provides debug information. The following table lists all the available options.

Attribute	Default	Type	Description
lwip_debug	false	Boolean	Turn on/off lwIP debugging.
ip_debug	false	Boolean	Turn on/off IP layer debugging.
tcp_debug	false	Boolean	Turn on/off TCP layer debugging.
udp_debug	false	Boolean	Turn on/off UDP layer debugging.
icmp_debug	false	Boolean	Turn on/off ICMP protocol debugging.
igmp_debug	false	Boolean	Turn on/off IGMP protocol debugging.
netif_debug	false	Boolean	Turn on/off network interface layer debugging.
sys_debug	false	Boolean	Turn on/off sys arch layer debugging.
pbuf_debug	false	Boolean	Turn on/off pbuf layer debugging

LwIP Library APIs

The lwIP library provides two different APIs: RAW API and Socket API.

Raw API

The Raw API is callback based. Applications obtain access directly into the TCP stack and vice-versa. As a result, there is no extra socket layer, and using the Raw API provides excellent performance at the price of compatibility with other TCP stacks.

Xilinx Adapter Requirements when using the RAW API

In addition to the lwIP RAW API, the Xilinx adapters provide the `xemacif_input` utility function for receiving packets. This function must be called at frequent intervals to move the received packets from the interrupt handlers to the lwIP stack. Depending on the type of packet received, lwIP then calls registered application callbacks. The `<Vitis_install_path>/sw/ThirdParty/sw_services/lwip211/src/lwip-2.1.1/doc/rawapi.txt` file describes the lwIP Raw API.

LwIP Performance

The following table provides the maximum TCP throughput achievable by FPGA, CPU, EMAC, and system frequency in RAW modes. Applications requiring high performance should use the RAW API.

FPGA	CPU	EMAC	System Frequency	Max TCP Throughput in RAW Mode (Mbps)
Virtex	MicroBlaze	axi-ethernet	100 MHz	RX Side: 182 TX Side: 100
Virtex	MicroBlaze	xps-ll-temac	100 MHz	RX Side: 178 TX Side: 100
Virtex	MicroBlaze	xps-ethernetlite	100 MHz	RX Side: 50 TX Side: 38

RAW API Example

Applications using the RAW API are single threaded. The following pseudo-code illustrates a typical RAW mode program structure.

```
int main()
{
        struct netif *netif, server_netif;
        ip_addr_t ipaddr, netmask, gw;


        unsigned char mac_ethernet_address[] =
                {0x00, 0x0a, 0x35, 0x00, 0x01, 0x02};

        lwip_init();


        if (!xemac_add(netif, &ipaddr, &netmask,
                &gw, mac_ethernet_address,
                EMAC_BASEADDR)) {
                printf("Error adding N/W interface\n\r");
                return -1;
        }
        netif_set_default(netif);


        platform_enable_interrupts();


        netif_set_up(netif);


        start_application();


        while (1) {
                xemacif_input(netif);

                transfer_data();
        }
}
```

Socket API

The lwIP socket API provides a BSD socket-style API to programs. This API provides an execution model that is a blocking, open-read-write-close paradigm.

Xilinx Adapter Requirements when using the Socket API

Applications using the Socket API with Xilinx adapters need to spawn a separate thread called `xemacif_input_thread`. This thread takes care of moving received packets from the interrupt handlers to the `tcpip_thread` of the lwIP. Application threads that use lwIP must be created using the lwIP `sys_thread_new` API. Internally, this function makes use of the appropriate thread or task creation routines provided by XilKernel or FreeRTOS.

Xilkernel/FreeRTOS scheduling policy when using the Socket API

lwIP in socket mode requires the use of the Xilkernel or FreeRTOS, which provides two policies for thread scheduling: round-robin and priority based. There are no special requirements when round-robin scheduling policy is used because all threads or tasks with same priority receive the same time quanta. This quanta is fixed by the RTOS (Xilkernel or FreeRTOS) being used. With priority scheduling, care must be taken to ensure that lwIP threads or tasks are not starved. For Xilkernel, lwIP internally launches all threads at the priority level specified in `socket_mode_thread_prio`. For FreeRTOS, lwIP internally launches all tasks except the main TCP/IP task at the priority specified in `socket_mode_thread_prio`. The TCP/IP task in FreeRTOS is launched with a higher priority (one more than priority set in `socket_mode_thread_prio`). In addition, application threads must launch `xemacif_input_thread`. The priorities of both `xemacif_input_thread`, and the lwIP internal threads (`socket_mode_thread_prio`) must be high enough in relation to the other application threads so that they are not starved.

Socket API Example

XilKernel-based applications in socket mode can specify a static list of threads that Xilkernel spawns on startup in the Xilkernel Software Platform Settings dialog box. Assuming that `main_thread()` is a thread specified to be launched by XIlkernel, control reaches this first thread from application `main` after the Xilkernel schedule is started. In `main_thread`, one more thread (network_thread) is created to initialize the MAC layer. For FreeRTOS (Zynq, Zynq Ultrascale+, and Versal processor systems) based applications, once the control reaches

application `main` routine, a task (can be termed as main_thread) with an entry point function as main_thread() is created before starting the scheduler. After the FreeRTOS scheduler starts, the control reaches `main_thread()`, where the lwIP internal initialization happens. The application then creates one more thread (network_thread) to initialize the MAC layer. The following pseudo-code illustrates a typical socket mode program structure.

```
void network_thread(void *p)
{
        struct netif *netif;
        ip_addr_t ipaddr, netmask, gw;


        unsigned char mac_ethernet_address[] =
                {0x00, 0x0a, 0x35, 0x00, 0x01, 0x02};

        netif = &server_netif;


        IP4_ADDR(&ipaddr,192,168,1,10);
        IP4_ADDR(&netmask,255,255,255,0);
        IP4_ADDR(&gw,192,168,1,1);


        if (!xemac_add(netif, &ipaddr, &netmask,
                        &gw, mac_ethernet_address,
                        EMAC_BASEADDR)) {
                printf("Error adding N/W interface\n\r");
                return;
        }
        netif_set_default(netif);


        netif_set_up(netif);


        sys_thread_new("xemacif_input_thread", xemacif_input_thread,
                netif,
                THREAD_STACKSIZE, DEFAULT_THREAD_PRIO);


        sys_thread_new("httpd" web_application_thread, 0,
                        THREAD_STACKSIZE DEFAULT_THREAD_PRIO);
}

int main_thread()
{

        lwip_init();


        sys_thread_new("network_thread" network_thread, NULL,
                        THREAD_STACKSIZE DEFAULT_THREAD_PRIO);

        return 0;
}
```

Using the Xilinx Adapter Helper Functions

The Xilinx adapters provide the following helper functions to simplify the use of the lwIP APIs.

Table 231: **Quick Function Reference**

Type	Name	Arguments
void	xemacif_input_thread	void
struct netif *	xemac_add	void
void	lwip_init	void
int	xemacif_input	void
void	xemacpsif_resetrx_on_no_rxdata	void

Functions

xemacif_input_thread

In the socket mode, the application thread must launch a separate thread to receive the input packets. This performs the same work as the RAW mode function, `xemacif_input()` , except that it resides in its own separate thread; consequently, any lwIP socket mode application is required to have code similar to the following in its main thread:

***Note*:** For Socket mode only.

```
sys_thread_new("xemacif_input_thread",

               xemacif_input_thread
            , netif, THREAD_STACK_SIZE, DEFAULT_THREAD_PRIO);
```

The application can then continue launching separate threads for doing application specific tasks. The `xemacif_input_thread()` receives data processed by the interrupt handlers, and passes them to the lwIP tcpip_thread.

Prototype

```
void xemacif_input_thread(struct netif *netif);
```

Returns

xemac_add

The `xemac_add()` function provides a unified interface to add any Xilinx EMAC IP as well as GigE core. This function is a wrapper around the lwIP netif_add function that initializes the network interface 'netif' given its IP address ipaddr, netmask, the IP address of the gateway, gw, the 6 byte ethernet address mac_ethernet_address, and the base address, mac_baseaddr, of the axi_ethernetlite or axi_ethernet MAC core.

Prototype

```
struct netif * xemac_add(struct netif *netif, ip_addr_t *ipaddr, ip_addr_t
*netmask, ip_addr_t *gw, unsigned char *mac_ethernet_address, unsigned
mac_baseaddr);
```

lwip_init

Initialize all modules. Use this in NO_SYS mode. Use tcpip_init() otherwise.

This function provides a single initialization function for the lwIP data structures. This replaces specific calls to initialize stats, system, memory, pbufs, ARP, IP, UDP, and TCP layers.

Prototype

```
void lwip_init(void);
```

xemacif_input

The Xilinx lwIP adapters work in interrupt mode. The receive interrupt handlers move the packet data from the EMAC/GigE and store them in a queue. The `xemacif_input()` function takes those packets from the queue, and passes them to lwIP; consequently, this function is required for lwIP operation in RAW mode. The following is a sample lwIP application in RAW mode.

Note: For RAW mode only.

```
            while (1) {


               xemacif_input
            (netif);


}
```

Note: The program is notified of the received data through callbacks.

Prototype

```
int xemacif_input(struct netif *netif);
```

Returns

xemacpsif_resetrx_on_no_rxdata

There is an errata on the GigE controller that is related to the Rx path. The errata describes conditions whereby the Rx path of GigE becomes completely unresponsive with heavy Rx traffic of small sized packets. The condition occurrence is rare; however a software reset of the Rx logic in the controller is required when such a condition occurs. This API must be called periodically (approximately every 100 milliseconds using a timer or thread) from user applications to ensure that the Rx path never becomes unresponsive for more than 100 milliseconds.

Note: Used in both Raw and Socket mode and applicable only for the Zynq-7000 devices, Zynq UltraScale+ MPSoC, and Versal processors and the GigE controller

Prototype

```
void xemacpsif_resetrx_on_no_rxdata(struct netif *netif);
```

Returns

XilIsf Library v5.15

Overview

***Note*:** This library is deprecated from the 2020.2 release onward.

The LibXil Isf library:

- Allows you to Write, Read, and Erase the Serial Flash.
- Allows protection of the data stored in the Serial Flash from unwarranted modification by enabling the Sector Protection feature.
- Supports multiple instances of Serial Flash at a time, provided they are of the same device family (Atmel, Intel, STM, Winbond, SST, or Spansion) as the device family is selected at compile time.
- Allows your application to perform Control operations on Intel, STM, Winbond, SST, and Spansion Serial Flash.
- Requires the underlying hardware platform to contain the axi_quad_spi, ps7_spi, ps7_qspi, psu_qspi, psv_ospi, or psu_spi device for accessing the Serial Flash.
- Uses the Xilinx SPI interface drivers in interrupt-driven mode or polled mode for communicating with the Serial Flash. In interrupt mode, the user application must acknowledge any associated interrupts from the Interrupt Controller.

Additional information

- In interrupt mode, the application is required to register a callback to the library and the library registers an internal status handler to the selected interface driver.
- When your application requests a library operation, it is initiated and control is given back to the application. The library tracks the status of the interface transfers, and notifies the user application upon completion of the selected library operation.
- Added support in the library for SPI PS and QSPI PS. You must select one of the interfaces at compile time.
- Added support for QSPIPSU and SPIPS flash interface on Zynq UltraScale+ MPSoC.
- Added support for OSPIPSV flash interface

- When your application requests selection of QSPIPS interface during compilation, the QSPI PS or QSPI PSU interface, based on the hardware platform, are selected.
- When the SPIPS interface is selected during compilation, the SPI PS or the SPI PSU interface is selected.
- When the OSPI interface is selected during compilation, the OSPIPSV interface is selected.

Supported Devices

The table below lists the supported Xilinx in-system and external serial flash memories.

Device Series	Manufacturer
AT45DB011D AT45DB021D AT45DB041D AT45DB081D AT45DB161D AT45DB321D AT45DB642D	Atmel
W25Q16 W25Q32 W25Q64 W25Q80 W25Q128 W25X10 W25X20 W25X40 W25X80 W25X16 W25X32 W25X64	Winbond
S25FL004 S25FL008 S25FL016 S25FL032 S25FL064 S25FL128 S25FL129 S25FL256 S25FL512 S70FL01G	Spansion
SST25WF080	SST
N25Q032 N25Q064 N25Q128 N25Q256 N25Q512 N25Q00AA MT25Q01 MT25Q02 MT25Q512 MT25QL02G MT25QU02G MT35XU512ABA	Micron
MX66L1G45G MX66U1G45G	Macronix
IS25WP256D IS25LP256D IS25LWP512M IS25LP512M IS25WP064A IS25LP064A IS25WP032D IS25LP032D IS25WP016D IS25LP016D IS25WP080D IS25LP080D IS25LP128F IS25WP128F	ISSI

***Note*:** Intel, STM, and Numonyx serial flash devices are now a part of Serial Flash devices provided by Micron.

References

- Spartan-3AN FPGA In-System Flash User Guide (UG333):http://www.xilinx.com/support/documentation/user_guides/ug333.pdf
- Winbond Serial Flash Page:http://www.winbond.com/hq/product/code-storage-flash-memory/ serial-nor-flash/?__locale=en
- Intel (Numonyx) S33 Serial Flash Memory, SST SST25WF080, Micron N25Q flash family : https://www.micron.com/products/nor-flash/serial-nor-flash

XilIsf Library API

This section provides a linked summary and detailed descriptions of the XilIsf library APIs.

Table 232: **Quick Function Reference**

Type	Name	Arguments
int	XIsf_Initialize	XIsf * InstancePtr XIsf_Iface * SpiInstPtr u8 SlaveSelect u8 * WritePtr
int	XIsf_GetStatus	XIsf * InstancePtr u8 * ReadPtr
int	XIsf_GetStatusReg2	XIsf * InstancePtr u8 * ReadPtr
int	XIsf_GetDeviceInfo	XIsf * InstancePtr u8 * ReadPtr
int	XIsf_Transfer	void
u32	GetRealAddr	XIsf_Iface * QspiPtr u32 Address
int	XIsf_Write	XIsf * InstancePtr XIsf_WriteOperation Operation void * OpParamPtr
int	XIsf_Read	XIsf * InstancePtr XIsf_ReadOperation Operation void * OpParamPtr
int	XIsf_Erase	XIsf * InstancePtr XIsf_EraseOperation Operation u32 Address
int	XIsf_SectorProtect	XIsf * InstancePtr XIsf_SpOperation Operation u8 * BufferPtr
int	XIsf_Ioctl	XIsf * InstancePtr XIsf_IoctlOperation Operation
int	XIsf_WriteEnable	XIsf * InstancePtr u8 WriteEnable

Table 232: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	XIsf_RegisterInterface	XIsf * InstancePtr
int	XIsf_SetSpiConfiguration	XIsf * InstancePtr XIsf_Iface * SpiInstPtr u32 Options u8 PreScaler
void	XIsf_SetStatusHandler	XIsf * InstancePtr XIsf_Iface * XIfaceInstancePtr XIsf_StatusHandler XilIsf_Handler
void	XIsf_IfaceHandler	void * CallBackRef u32 StatusEvent unsigned int ByteCount

Functions

XIsf_Initialize

This API when called initializes the SPI interface with default settings.

With custom settings, user should call `XIsf_SetSpiConfiguration()` and then call this API. The geometry of the underlying Serial Flash is determined by reading the Joint Electron Device Engineering Council (JEDEC) Device Information and the Status Register of the Serial Flash.

***Note*:**

- The `XIsf_Initialize()` API is a blocking call (for both polled and interrupt modes of the Spi driver). It reads the JEDEC information of the device and waits till the transfer is complete before checking if the information is valid.
- This library can support multiple instances of Serial Flash at a time, provided they are of the same device family (either Atmel, Intel or STM, Winbond or Spansion) as the device family is selected at compile time.

Prototype

```
int XIsf_Initialize(XIsf *InstancePtr, XIsf_Iface *SpiInstPtr, u8
SlaveSelect, u8 *WritePtr);
```

Parameters

The following table lists the `XIsf_Initialize` function arguments.

Table 233: **XIsf_Initialize Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
XIsf_Iface *	SpiInstPtr	Pointer to XIsf_Iface instance to be worked on.
u8	SlaveSelect	It is a 32-bit mask with a 1 in the bit position of slave being selected. Only one slave can be selected at a time.
u8 *	WritePtr	Pointer to the buffer allocated by the user to be used by the In-system and Serial Flash Library to perform any read/write operations on the Serial Flash device. User applications must pass the address of this buffer for the Library to work. • Write operations : ◦ The size of this buffer should be equal to the Number of bytes to be written to the Serial Flash + XISF_CMD_MAX_EXTRA_BYTES. ◦ The size of this buffer should be large enough for usage across all the applications that use a common instance of the Serial Flash. ◦ A minimum of one byte and a maximum of ISF_PAGE_SIZE bytes can be written to the Serial Flash, through a single Write operation. • Read operations : ◦ The size of this buffer should be equal to XISF_CMD_MAX_EXTRA_BYTES, if the application only reads from the Serial Flash (no write operations).

Returns

- XST_SUCCESS if successful.
- XST_DEVICE_IS_STOPPED if the device must be started before transferring data.
- XST_FAILURE, otherwise.

XIsf_GetStatus

This API reads the Serial Flash Status Register.

***Note*:** The contents of the Status Register is stored at second byte pointed by the ReadPtr.

Prototype

```
int XIsf_GetStatus(XIsf *InstancePtr, u8 *ReadPtr);
```

Parameters

The following table lists the `XIsf_GetStatus` function arguments.

Table 234: **XIsf_GetStatus Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
u8 *	ReadPtr	Pointer to the memory where the Status Register content is copied.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_GetStatusReg2

This API reads the Serial Flash Status Register 2.

Note: The contents of the Status Register 2 is stored at the second byte pointed by the ReadPtr. This operation is available only in Winbond Serial Flash.

Prototype

```
int XIsf_GetStatusReg2(XIsf *InstancePtr, u8 *ReadPtr);
```

Parameters

The following table lists the `XIsf_GetStatusReg2` function arguments.

Table 235: **XIsf_GetStatusReg2 Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
u8 *	ReadPtr	Pointer to the memory where the Status Register content is copied.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_GetDeviceInfo

This API reads the Joint Electron Device Engineering Council (JEDEC) information of the Serial Flash.

Note: The Device information is stored at the second byte pointed by the ReadPtr.

Prototype

```
int XIsf_GetDeviceInfo(XIsf *InstancePtr, u8 *ReadPtr);
```

Parameters

The following table lists the `XIsf_GetDeviceInfo` function arguments.

Table 236: **XIsf_GetDeviceInfo Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
u8 *	ReadPtr	Pointer to the buffer where the Device information is copied.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_Transfer

Prototype

```
int XIsf_Transfer(XIsf *InstancePtr, u8 *WritePtr, u8 *ReadPtr, u32
ByteCount);
```

GetRealAddr

Function to get the real address of flash in case dual parallel and stacked configuration.

Function to get the real address of flash in case dual parallel and stacked configuration.

This functions translates the address based on the type of interconnection. In case of stacked, this function asserts the corresponding slave select.

***Note*:** None.

Prototype

```
u32 GetRealAddr(XIsf_Iface *QspiPtr, u32 Address);
```

Parameters

The following table lists the `GetRealAddr` function arguments.

Table 237: **GetRealAddr Arguments**

Type	Name	Description
XIsf_Iface *	QspiPtr	is a pointer to XIsf_Iface instance to be worked on.
u32	Address	which is to be accessed (for erase, write or read)

Returns

RealAddr is the translated address - for single it is unchanged for stacked, the lower flash size is subtracted for parallel the address is divided by 2.

XIsf_Write

This API writes the data to the Serial Flash.

Operations

- Normal Write(XISF_WRITE), Dual Input Fast Program (XISF_DUAL_IP_PAGE_WRITE), Dual Input Extended Fast Program(XISF_DUAL_IP_EXT_PAGE_WRITE), Quad Input Fast Program(XISF_QUAD_IP_PAGE_WRITE), Quad Input Extended Fast Program (XISF_QUAD_IP_EXT_PAGE_WRITE):
  - The OpParamPtr must be of type struct XIsf_WriteParam.
    - OpParamPtr->Address is the start address in the Serial Flash.
    - OpParamPtr->WritePtr is a pointer to the data to be written to the Serial Flash.
    - OpParamPtr->NumBytes is the number of bytes to be written to Serial Flash.
    - This operation is supported for Atmel, Intel, STM, Winbond and Spansion Serial Flash.
- Auto Page Write (XISF_AUTO_PAGE_WRITE):
  - The OpParamPtr must be of 32 bit unsigned integer variable.
  - This is the address of page number in the Serial Flash which is to be refreshed.
  - This operation is only supported for Atmel Serial Flash.
- Buffer Write (XISF_BUFFER_WRITE):
  - The OpParamPtr must be of type struct XIsf_BufferToFlashWriteParam.
  - OpParamPtr->BufferNum specifies the internal SRAM Buffer of the Serial Flash. The valid values are XISF_PAGE_BUFFER1 or XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 is not valid in case of AT45DB011D Flash as it contains a single buffer.
  - OpParamPtr->WritePtr is a pointer to the data to be written to the Serial Flash SRAM Buffer.
  - OpParamPtr->ByteOffset is byte offset in the buffer from where the data is to be written.
  - OpParamPtr->NumBytes is number of bytes to be written to the Buffer. This operation is supported only for Atmel Serial Flash.
- Buffer To Memory Write With Erase (XISF_BUF_TO_PAGE_WRITE_WITH_ERASE)/ Buffer To Memory Write Without Erase (XISF_BUF_TO_PAGE_WRITE_WITHOUT_ERASE):
  - The OpParamPtr must be of type struct XIsf_BufferToFlashWriteParam.

    - OpParamPtr->BufferNum specifies the internal SRAM Buffer of the Serial Flash. The valid values are XISF_PAGE_BUFFER1 or XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 is not valid in case of AT45DB011D Flash as it contains a single buffer.
    - OpParamPtr->Address is starting address in the Serial Flash memory from where the data is to be written. These operations are only supported for Atmel Serial Flash.
- Write Status Register (XISF_WRITE_STATUS_REG):
    - The OpParamPtr must be of type of 8 bit unsigned integer variable. This is the value to be written to the Status Register.
    - This operation is only supported for Intel, STM Winbond and Spansion Serial Flash.
- Write Status Register2 (XISF_WRITE_STATUS_REG2):
    - The OpParamPtr must be of type (u8 *) and should point to two 8 bit unsigned integer values. This is the value to be written to the 16 bit Status Register. This operation is only supported in Winbond (W25Q) Serial Flash.
- One Time Programmable Area Write(XISF_OTP_WRITE):
    - The OpParamPtr must be of type struct XIsf_WriteParam.
    - OpParamPtr->Address is the address in the SRAM Buffer of the Serial Flash to which the data is to be written.
    - OpParamPtr->WritePtr is a pointer to the data to be written to the Serial Flash.
    - OpParamPtr->NumBytes should be set to 1 when performing OTPWrite operation. This operation is only supported for Intel Serial Flash.

Note:

- Application must fill the structure elements of the third argument and pass its pointer by type casting it with void pointer.
- For Intel, STM, Winbond and Spansion Serial Flash, the user application must call the `XIsf_WriteEnable()` API by passing XISF_WRITE_ENABLE as an argument, before calling the `XIsf_Write()` API.

Prototype

```
int XIsf_Write(XIsf *InstancePtr, XIsf_WriteOperation Operation, void
*OpParamPtr);
```

Parameters

The following table lists the `XIsf_Write` function arguments.

Table 238: **XIsf_Write Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.

Table 238: **XIsf_Write Arguments** *(cont'd)*

Type	Name	Description
XIsf_WriteOperation	Operation	Type of write operation to be performed on the Serial Flash. The different operations are • XISF_WRITE: Normal Write • XISF_DUAL_IP_PAGE_WRITE: Dual Input Fast Program • XISF_DUAL_IP_EXT_PAGE_WRITE: Dual Input Extended Fast Program • XISF_QUAD_IP_PAGE_WRITE: Quad Input Fast Program • XISF_QUAD_IP_EXT_PAGE_WRITE: Quad Input Extended Fast Program • XISF_AUTO_PAGE_WRITE: Auto Page Write • XISF_BUFFER_WRITE: Buffer Write • XISF_BUF_TO_PAGE_WRITE_WITH_ERASE: Buffer to Page Transfer with Erase • XISF_BUF_TO_PAGE_WRITE_WITHOUT_ERASE: Buffer to Page Transfer without Erase • XISF_WRITE_STATUS_REG: Status Register Write • XISF_WRITE_STATUS_REG2: 2 byte Status Register Write • XISF_OTP_WRITE: OTP Write.
void *	OpParamPtr	Pointer to a structure variable which contains operational parameters of the specified operation. This parameter type is dependant on value of first argument(Operation). For more details, refer `Operations`.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_Read

This API reads the data from the Serial Flash.

Operations

- Normal Read (XISF_READ), Fast Read (XISF_FAST_READ), One Time Programmable Area Read(XISF_OTP_READ), Dual Output Fast Read (XISF_CMD_DUAL_OP_FAST_READ), Dual Input/Output Fast Read (XISF_CMD_DUAL_IO_FAST_READ), Quad Output Fast Read (XISF_CMD_QUAD_OP_FAST_READ) and Quad Input/Output Fast Read (XISF_CMD_QUAD_IO_FAST_READ):
  - The OpParamPtr must be of type struct XIsf_ReadParam.
  - OpParamPtr->Address is start address in the Serial Flash.

- OpParamPtr->ReadPtr is a pointer to the memory where the data read from the Serial Flash is stored.
- OpParamPtr->NumBytes is number of bytes to read.
- OpParamPtr->NumDummyBytes is the number of dummy bytes to be transmitted for the Read command. This parameter is only used in case of Dual and Quad reads.
- Normal Read and Fast Read operations are supported for Atmel, Intel, STM, Winbond and Spansion Serial Flash.
- Dual and quad reads are supported for Winbond (W25QXX), Numonyx(N25QXX) and Spansion (S25FL129) quad flash.
- OTP Read operation is only supported in Intel Serial Flash.

- Page To Buffer Transfer (XISF_PAGE_TO_BUF_TRANS):
  - The OpParamPtr must be of type struct XIsf_FlashToBufTransferParam .
  - OpParamPtr->BufferNum specifies the internal SRAM Buffer of the Serial Flash. The valid values are XISF_PAGE_BUFFER1 or XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 is not valid in case of AT45DB011D Flash as it contains a single buffer.
  - OpParamPtr->Address is start address in the Serial Flash. This operation is only supported in Atmel Serial Flash.
- Buffer Read (XISF_BUFFER_READ) and Fast Buffer Read(XISF_FAST_BUFFER_READ):
  - The OpParamPtr must be of type struct XIsf_BufferReadParam.
  - OpParamPtr->BufferNum specifies the internal SRAM Buffer of the Serial Flash. The valid values are XISF_PAGE_BUFFER1 or XISF_PAGE_BUFFER2. XISF_PAGE_BUFFER2 is not valid in case of AT45DB011D Flash as it contains a single buffer.
  - OpParamPtr->ReadPtr is pointer to the memory where data read from the SRAM buffer is to be stored.
  - OpParamPtr->ByteOffset is byte offset in the SRAM buffer from where the first byte is read.
  - OpParamPtr->NumBytes is the number of bytes to be read from the Buffer. These operations are supported only in Atmel Serial Flash.

***Note*:**

- Application must fill the structure elements of the third argument and pass its pointer by type casting it with void pointer.
- The valid data is available from the fourth location pointed to by the ReadPtr for Normal Read and Buffer Read operations.
- The valid data is available from fifth location pointed to by the ReadPtr for Fast Read, Fast Buffer Read and OTP Read operations.
- The valid data is available from the (4 + NumDummyBytes)th location pointed to by ReadPtr for Dual/Quad Read operations.

Prototype

```
int XIsf_Read(XIsf *InstancePtr, XIsf_ReadOperation Operation, void
*OpParamPtr);
```

Parameters

The following table lists the `XIsf_Read` function arguments.

Table 239: **XIsf_Read Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
XIsf_ReadOperation	Operation	Type of the read operation to be performed on the Serial Flash. The different operations are • XISF_READ: Normal Read • XISF_FAST_READ: Fast Read • XISF_PAGE_TO_BUF_TRANS: Page to Buffer Transfer • XISF_BUFFER_READ: Buffer Read • XISF_FAST_BUFFER_READ: Fast Buffer Read • XISF_OTP_READ: One Time Programmable Area (OTP) Read • XISF_DUAL_OP_FAST_READ: Dual Output Fast Read • XISF_DUAL_IO_FAST_READ: Dual Input/Output Fast Read • XISF_QUAD_OP_FAST_READ: Quad Output Fast Read • XISF_QUAD_IO_FAST_READ: Quad Input/Output Fast Read
void *	OpParamPtr	Pointer to structure variable which contains operational parameter of specified Operation. This parameter type is dependant on the type of Operation to be performed. For more details, refer `Operations`.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_Erase

This API erases the contents of the specified memory in the Serial Flash.

***Note*:**

- The erased bytes will read as 0xFF.
- For Intel, STM, Winbond or Spansion Serial Flash the user application must call `XIsf_WriteEnable()` API by passing XISF_WRITE_ENABLE as an argument before calling `XIsf_Erase()` API.

- Atmel Serial Flash support Page/Block/Sector Erase operations.
- Intel, Winbond, Numonyx (N25QXX) and Spansion Serial Flash support Sector/Block/Bulk Erase operations.
- STM (M25PXX) Serial Flash support Sector/Bulk Erase operations.

Prototype

```
int XIsf_Erase(XIsf *InstancePtr, XIsf_EraseOperation Operation, u32
Address);
```

Parameters

The following table lists the `XIsf_Erase` function arguments.

Table 240: **XIsf_Erase Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
XIsf_EraseOperation	Operation	Type of Erase operation to be performed on the Serial Flash. The different operations are • XISF_PAGE_ERASE: Page Erase • XISF_BLOCK_ERASE: Block Erase • XISF_SECTOR_ERASE: Sector Erase • XISF_BULK_ERASE: Bulk Erase
u32	Address	Address of the Page/Block/Sector to be erased. The address can be either Page address, Block address or Sector address based on the Erase operation to be performed.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_SectorProtect

This API is used for performing Sector Protect related operations.

***Note*:**

- The SPR content is stored at the fourth location pointed by the BufferPtr when performing XISF_SPR_READ operation.
- For Intel, STM, Winbond and Spansion Serial Flash, the user application must call the `XIsf_WriteEnable()` API by passing XISF_WRITE_ENABLE as an argument, before calling the `XIsf_SectorProtect()` API, for Sector Protect Register Write (XISF_SPR_WRITE) operation.
- Atmel Flash supports all these Sector Protect operations.

- Intel, STM, Winbond and Spansion Flash support only Sector Protect Read and Sector Protect Write operations.

Prototype

```
int XIsf_SectorProtect(XIsf *InstancePtr, XIsf_SpOperation Operation, u8
*BufferPtr);
```

Parameters

The following table lists the `XIsf_SectorProtect` function arguments.

Table 241: **XIsf_SectorProtect Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
XIsf_SpOperation	Operation	Type of Sector Protect operation to be performed on the Serial Flash. The different operations are • XISF_SPR_READ: Read Sector Protection Register • XISF_SPR_WRITE: Write Sector Protection Register • XISF_SPR_ERASE: Erase Sector Protection Register • XISF_SP_ENABLE: Enable Sector Protection • XISF_SP_DISABLE: Disable Sector Protection
u8 *	BufferPtr	Pointer to the memory where the SPR content is read to/written from. This argument can be NULL if the Operation is SprErase, SpEnable and SpDisable.

Returns

- XST_SUCCESS if successful.
- XST_FAILURE if it fails.

XIsf_Ioctl

This API configures and controls the Intel, STM, Winbond and Spansion Serial Flash.

***Note*:**

- Atmel Serial Flash does not support any of these operations.
- Intel Serial Flash support Enter/Release from DPD Mode and Clear Status Register Fail Flags.
- STM, Winbond and Spansion Serial Flash support Enter/Release from DPD Mode.
- Winbond (W25QXX) Serial Flash support Enable High Performance mode.

Prototype

```
int XIsf_Ioctl(XIsf *InstancePtr, XIsf_IoctlOperation Operation);
```

Parameters

The following table lists the `XIsf_Ioctl` function arguments.

Table 242: **XIsf_Ioctl Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
XIsf_IoctlOperation	Operation	Type of Control operation to be performed on the Serial Flash. The different control operations are • XISF_RELEASE_DPD: Release from Deep Power Down (DPD) Mode • XISF_ENTER_DPD: Enter DPD Mode • XISF_CLEAR_SR_FAIL_FLAGS: Clear Status Register Fail Flags

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_WriteEnable

This API Enables/Disables writes to the Intel, STM, Winbond and Spansion Serial Flash.

***Note*:** This API works only for Intel, STM, Winbond and Spansion Serial Flash. If this API is called for Atmel Flash, XST_FAILURE is returned.

Prototype

```
int XIsf_WriteEnable(XIsf *InstancePtr, u8 WriteEnable);
```

Parameters

The following table lists the `XIsf_WriteEnable` function arguments.

Table 243: **XIsf_WriteEnable Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
u8	WriteEnable	Specifies whether to Enable (XISF_CMD_ENABLE_WRITE) or Disable (XISF_CMD_DISABLE_WRITE) the writes to the Serial Flash.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_RegisterInterface

This API registers the interface SPI/SPI PS/QSPI PS.

Prototype

```
void XIsf_RegisterInterface(XIsf *InstancePtr);
```

Parameters

The following table lists the `XIsf_RegisterInterface` function arguments.

Table 244: **XIsf_RegisterInterface Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.

Returns

None

XIsf_SetSpiConfiguration

This API sets the configuration of SPI.

This will set the options and clock prescaler (if applicable).

***Note*:** This API can be called before calling `XIsf_Initialize()` to initialize the SPI interface in other than default options mode. PreScaler is only applicable to PS SPI/QSPI.

Prototype

```
int XIsf_SetSpiConfiguration(XIsf *InstancePtr, XIsf_Iface *SpiInstPtr, u32
Options, u8 PreScaler);
```

Parameters

The following table lists the `XIsf_SetSpiConfiguration` function arguments.

Table 245: **XIsf_SetSpiConfiguration Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.

Table 245: **XIsf_SetSpiConfiguration Arguments** *(cont'd)*

Type	Name	Description
XIsf_Iface *	SpiInstPtr	Pointer to XIsf_Iface instance to be worked on.
u32	Options	Specified options to be set.
u8	PreScaler	Value of the clock prescaler to set.

Returns

XST_SUCCESS if successful else XST_FAILURE.

XIsf_SetStatusHandler

This API is to set the Status Handler when an interrupt is registered.

Note: None.

Prototype

```
void XIsf_SetStatusHandler(XIsf *InstancePtr, XIsf_Iface
*XIfaceInstancePtr, XIsf_StatusHandler XilIsf_Handler);
```

Parameters

The following table lists the `XIsf_SetStatusHandler` function arguments.

Table 246: **XIsf_SetStatusHandler Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf Instance.
XIsf_Iface *	XIfaceInstancePtr	Pointer to the XIsf_Iface instance to be worked on.
XIsf_StatusHandler	XilIsf_Handler	Status handler for the application.

Returns

None

XIsf_IfaceHandler

This API is the handler which performs processing for the QSPI driver.

It is called from an interrupt context such that the amount of processing performed should be minimized. It is called when a transfer of QSPI data completes or an error occurs.

This handler provides an example of how to handle QSPI interrupts but is application specific.

Note: None.

Prototype

```
void XIsf_IfaceHandler(void *CallBackRef, u32 StatusEvent, unsigned int
ByteCount);
```

Parameters

The following table lists the `XIsf_IfaceHandler` function arguments.

Table 247: **XIsf_IfaceHandler Arguments**

Type	Name	Description
void *	CallBackRef	Reference passed to the handler.
u32	StatusEvent	Status of the QSPI .
unsigned int	ByteCount	Number of bytes transferred.

Returns

None

Library Parameters in MSS File

Xillsf Library can be integrated with a system using the following snippet in the Microprocessor Software Specification (MSS) file:

```
BEGIN LIBRARY`
PARAMETER LIBRARY_NAME = xilisf
PARAMETER LIBRARY_VER = 5.15
PARAMETER serial_flash_family = 1
PARAMETER serial_flash_interface = 1
END
```

The table below describes the libgen customization parameters.

Parameter	Default Value	Description
LIBRARY_NAME	xilisf	Specifies the library name.
LIBRARY_VER	5.15	Specifies the library version.
serial_flash_family	1	Specifies the serial flash family. Supported numerical values are: 1 = Xilinx In-system Flash or Atmel Serial Flash 2 = Intel (Numonyx) S33 Serial Flash 3 = STM (Numonyx) M25PXX/ N25QXX Serial Flash 4 = Winbond Serial Flash 5 = Spansion Serial Flash/Micron Serial Flash/Cypress Serial Flash 6 = SST Serial Flash

Parameter	Default Value	Description
Serial_flash_interface	1	Specifies the serial flash interface. Supported numerical values are: 1 = AXI QSPI Interface 2 = SPI PS Interface 3 = QSPI PS Interface or QSPI PSU Interface 4 = OSPIPSV Interface for OSPI

Note: Intel, STM, and Numonyx serial flash devices are now a part of Serial Flash devices provided by Micron.

XilFFS Library v4.4

XilFFS Library API Reference

The Xilinx fat file system (FFS) library consists of a file system and a glue layer. This FAT file system can be used with an interface supported in the glue layer. The file system code is open source and is used as it is. Currently, the Glue layer implementation supports the SD/eMMC interface and a RAM based file system. Application should make use of APIs provided in ff.h. These file system APIs access the driver functions through the glue layer.

The file system supports FAT16, FAT32, and exFAT (optional). The APIs are standard file system APIs. For more information, see the http://elm-chan.org/fsw/ff/00index_e.html.

***Note*:** The XilFFS library uses Revision R0.13b of the generic FAT filesystem module.

Library Files

The table below lists the file system files.

File	Description
ff.c	Implements all the file system APIs
ff.h	File system header
ffconf.h	File system configuration header – File system configurations such as READ_ONLY, MINIMAL, can be set here. This library uses FF_FS_MINIMIZE and FF_FS_TINY and Read/Write (NOT read only)

The table below lists the glue layer files.

File	Description
diskio.c	Glue layer – implements the function used by file system to call the driver APIs
ff.h	File system header
diskio.h	Glue layer header

Selecting a File System with an SD Interface

To select a file system with an SD interface:

1. Click **File → New → Platform Project**.
2. Click **Specify** to create a new hardware platform specification.
3. Provide a new name for the domain in the **Project name** field if you wish to override the default value.
4. Select the location for the board support project files. To use the default location, as displayed in the **Location** field, leave the **Use default location** check box selected. Otherwise, deselect the checkbox and then type or browse to the directory location.
5. From the **Hardware Platform** drop-down, choose the appropriate platform for your application or click the **New** button to browse to an existing hardware platform.
6. Select the target CPU from the drop-down list.
7. From the **Board Support Package OS** list box, select the type of board support package to create. A description of the platform types displays in the box below the drop-down list.
8. Click **Finish**. The wizard creates a new software platform and displays it in the Vitis Navigator pane.
9. Select **Project → Build Automatically** to automatically build the board support package. The **Board Support Package Settings** dialog box opens. Here you can customize the settings for the domain.
10. Click **OK** to accept the settings, build the platform, and close the dialog box.
11. From the Explorer, double-click `platform.spr` file and select the appropriate domain/board support package. The **Overview** page opens.
12. In the overview page, click **Modify BSP Settings**.
13. Using the Board Support Package Settings page, you can select the OS version and which of the supported libraries are to be enabled in this domain/BSP.
14. Select the **xilffs** library from the list of **Supported Libraries**.
15. Expand the **Overview** tree and select **xilffs**. The configuration options for xilffs are listed.
16. Configure the xilffs by setting the `fs_interface = 1` to select the `SD/eMMC`. This is the default value. Ensure that the SD/eMMC interface is available, prior to selecting the `fs_interface = 1` option.
17. Build the bsp and the application to use the file system with SD/eMMC. SD or eMMC will be recognized by the low level driver.

Selecting a RAM Based File System

To select a RAM based file system:

1. Click **File → New → Platform Project**.
2. Click **Specify** to create a new hardware platform specification.
3. Provide a new name for the domain in the **Project name** field if you wish to override the default value.
4. Select the location for the board support project files. To use the default location, as displayed in the **Location** field, leave the **Use default location** check box selected. Otherwise, deselect the checkbox and then type or browse to the directory location.
5. From the **Hardware Platform** drop-down, choose the appropriate platform for your application or click the **New** button to browse to an existing hardware platform.
6. Select the target CPU from the drop-down list.
7. From the **Board Support Package OS** list box, select the type of board support package to create. A description of the platform types displays in the box below the drop-down list.
8. Click **Finish**. The wizard creates a new software platform and displays it in the Vitis Navigator pane.
9. Select **Project → Build Automatically** to automatically build the board support package. The **Board Support Package Settings** dialog box opens. Here you can customize the settings for the domain.
10. Click **OK** to accept the settings, build the platform, and close the dialog box.
11. From the Explorer, double-click `platform.spr` file and select the appropriate domain/board support package. The **Overview** page opens.
12. In the **Overview** page, click **Modify BSP Settings**.
13. Using the Board Support Package Settings page, you can select the OS version and which of the supported libraries are to be enabled in this domain/BSP.
14. Select the **xilffs** library from the list of **Supported Libraries**.
15. Expand the **Overview** tree and select **xilffs**. The configuration options for xilffs are listed.
16. Configure the xilffs by setting the `fs_interface = 2` to select the `RAM`.
17. As this project is used by LWIP based application, select **lwip library** and configure according to your requirements. For more information, see the LwIP Library API Reference documentation.
18. Use any lwip application that requires a RAM based file system - TCP/UDP performance test apps or tftp or webserver examples.
19. Build the bsp and the application to use the RAM based file system.

Library Parameters in MSS File

XilFFS Library can be integrated with a system using the following code snippet in the Microprocessor Software Specification (MSS) file:

```
BEGIN LIBRARY
    PARAMETER LIBRARY_NAME = xilffs
    PARAMETER LIBRARY_VER = 4.4
    PARAMETER fs_interface = 1
    PARAMETER read_only = false
    PARAMETER use_lfn = 0
    PARAMETER enable_multi_partition = false
    PARAMETER num_logical_vol = 2
    PARAMETER use_mkfs = true
    PARAMETER use_strfunc = 0
    PARAMETER set_fs_rpath = 0
    PARAMETER enable_exfat = false
    PARAMETER word_access = true
    PARAMETER use_chmod = false
END
```

The table below describes the libgen customization parameters.

Parameter	Default Value	Description
LIBRARY_NAME	xilffs	Specifies the library name.
LIBRARY_VER	4.4	Specifies the library version.
fs_interface	1 for SD/eMMC 2 for RAM	File system interface. SD/eMMC and RAM based file system are supported.
read_only	False	Enables the file system in Read Only mode, if true. Default is false. For Zynq UltraScale+ MPSoC devices, sets this option as true.
use_lfn	0	Enables the Long File Name(LFN) support if non-zero. 0: Disabled (Default) 1: LFN with static working buffer 2 (on stack) or 3 (on heap): Dynamic working buffer
enable_multi_partitio	False	Enables the multi partition support, if true.
num_logical_vol	2	Number of volumes (logical drives, from 1 to 10) to be used.
use_mkfs	True	Enables the mkfs support, if true. For Zynq UltraScale+ MPSoC devices, set this option as false.
use_strfunc	0	Enables the string functions (valid values 0 to 2). Default is 0.
set_fs_rpath	0	Configures relative path feature (valid values 0 to 2). Default is 0.
ramfs_size	3145728	Ram FS size is applicable only when RAM based file system is selected.
ramfs_start_addr	0x10000000	RAM FS start address is applicable only when RAM based file system is selected.

Parameter	Default Value	Description
enable_exfat	false	Enables support for exFAT file system. 0: Disable exFAT 1: Enable exFAT(Also Enables LFN)
word_access	True	Enables word access for misaligned memory access platform.
use_chmod	false	Enables use of CHMOD functionality for changing attributes (valid only with read_only set to false).

XilSecure Library v4.3

Overview

The XilSecure library provides APIs to access cryptographic accelerators on the Zynq UltraScale+ MPSoC devices. The library is designed to run on top of Xilinx standalone BSPs. It is tested for A53, R5 and MicroBlaze. XilSecure is used during the secure boot process. The primary post-boot use case is to run this library on the PMU MicroBlaze with PMUFW to service requests from Uboot or Linux for cryptographic acceleration.

The XilSecure library includes:

- SHA-3/384 engine for 384 bit hash calculation.
- AES-GCM engine for symmetric key encryption and decryption using a 256-bit key.
- RSA engine for signature generation, signature verification, encryption and decryption. Key sizes supported include 2048, 3072, and 4096.

CAUTION! *SDK defaults to using a software stack in DDR and any variables used by XilSecure will be placed in DDR. For better security, change the linker settings to make sure the stack used by XilSecure is either in the OCM or the TCM.*

XilSecure provides an user configuration under BSP settings to enable or disable secure environment, this bsp parameter is valid only when BSP is build for the PMU MicroBlaze for post boot use cases and XilSecure is been accessed using the IPI response calls to PMUFW from Linux or U-boot or baremetal applications. When the application environment is secure and trusted this variable should be set to TRUE.

By default, PMUFW will not allow device key for any decryption operation requested through IPI response unless authentication is enabled. If the user space is secure and trusted PMUFW can be build by setting the secure_environment variable. Only then the PMUFW allows usage of the device key for encrypting or decrypting the data blobs, decryption of bitstream or image.

Parameter	Description
secure_environment	Default = FALSE. Set the value to TRUE to allow usage of device key through the IPI response calls.

The source files for the library can be found at:

- https://github.com/Xilinx/embeddedsw/tree/master/lib/sw_services/xilsecure/src/zynqmp
- https://github.com/Xilinx/embeddedsw/tree/master/lib/sw_services/xilsecure/src/common

AES-GCM

This software uses AES-GCM hardened cryptographic accelerator to encrypt or decrypt the provided data and requires a key of size 256 bits and initialization vector(IV) of size 96 bits.

XilSecure library supports the following features:

- Encryption of data with provided key and IV
- Decryption of data with provided key and IV
- Authentication using a GCM tag.
- Key loading based on key selection, the key can be either the user provided key loaded into the KUP key or the device key used during boot.

For either encryption or decryption the AES-GCM engine should be initialized first using the XSecure_AesInitialize function.

When all the data to be encrypted is available, the `XSecure_AesEncryptData()` can be used. When all the data is not available, use the following functions in the suggested order:

1. `XSecure_AesEncryptInit()`
2. `XSecure_AesEncryptUpdate()` - This function can be called multiple times till input data is completed.

When all the data to be decrypted is available, the `XSecure_AesDecryptData()` can be used. When all the data is not available, use the following functions in the suggested order:

1. `XSecure_AesDecryptInit()`
2. `XSecure_AesDecryptUpdate()` - This function can be called multiple times till input data is completed.

During decryption, the provided GCM tag is compared to the GCM tag calculated by the engine. The two tags are then compared in the software and returned to the user as to whether or not the tags matched.

CAUTION! *when using the KUP key for encryption/decryption of the data, where the key is stored should be carefully considered. Key should be placed in an internal memory region that has access controls. Not doing so may result in security vulnerability.*

Table 248: **Quick Function Reference**

Type	Name	Arguments
s32	XSecure_AesInitialize	XSecure_Aes * InstancePtr XCsuDma * CsuDmaPtr u32 KeySel u32 * IvPtr u32 * KeyPtr
u32	XSecure_AesDecryptInit	XSecure_Aes * InstancePtr u8 * DecData u32 Size u8 * GcmTagAddr
s32	XSecure_AesDecryptUpdate	XSecure_Aes * InstancePtr u8 * EncData u32 Size
s32	XSecure_AesDecryptData	XSecure_Aes * InstancePtr u8 * DecData u8 * EncData u32 Size u8 * GcmTagAddr
s32	XSecure_AesDecrypt	XSecure_Aes * InstancePtr const u8 * Src u8 * Dst u32 Length
u32	XSecure_AesEncryptInit	XSecure_Aes * InstancePtr u8 * EncData u32 Size
u32	XSecure_AesEncryptUpdate	XSecure_Aes * InstancePtr const u8 * Data u32 Size
u32	XSecure_AesEncryptData	XSecure_Aes * InstancePtr u8 * Dst const u8 * Src u32 Len
void	XSecure_AesReset	XSecure_Aes * InstancePtr

Functions

XSecure_AesInitialize

This function initializes the instance pointer.

Note: All the inputs are accepted in little endian format but the AES engine accepts the data in big endian format, The decryption and encryption functions in xsecure_aes handle the little endian to big endian conversion using few API's, Xil_Htonl (provided by Xilinx xil_io library) and XSecure_AesCsuDmaConfigureEndiannes for handling data endianness conversions. If higher performance is needed, users can strictly use data in big endian format and modify the xsecure_aes functions to remove the use of the Xil_Htonl and XSecure_AesCsuDmaConfigureEndiannes functions as required.

Prototype

```
s32 XSecure_AesInitialize(XSecure_Aes *InstancePtr, XCsuDma *CsuDmaPtr, u32
KeySel, u32 *IvPtr, u32 *KeyPtr);
```

Parameters

The following table lists the `XSecure_AesInitialize` function arguments.

Table 249: **XSecure_AesInitialize Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.
CsuDmaPtr	Pointer to the XCsuDma instance.
KeySel	Key source for decryption, can be KUP/device key • XSECURE_CSU_AES_KEY_SRC_KUP :For KUP key • XSECURE_CSU_AES_KEY_SRC_DEV :For Device Key
IvPtr	Pointer to the Initialization Vector for decryption
KeyPtr	Pointer to Aes key in case KUP key is used. Pass Null if the device key is to be used.

Returns

XST_SUCCESS if initialization was successful.

XSecure_AesDecryptInit

This function initializes the AES engine for decryption and is required to be called before calling XSecure_AesDecryptUpdate.

Note: If all of the data to be decrypted is available, the XSecure_AesDecryptData function can be used instead.

Prototype

```
u32 XSecure_AesDecryptInit(XSecure_Aes *InstancePtr, u8 *DecData, u32 Size,
u8 *GcmTagAddr);
```

Parameters

The following table lists the `XSecure_AesDecryptInit` function arguments.

Table 250: **XSecure_AesDecryptInit Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.
DecData	Pointer in which decrypted data will be stored.
Size	Expected size of the data in bytes whereas the number of bytes provided should be multiples of 4.
GcmTagAddr	Pointer to the GCM tag which needs to be verified during decryption of the data.

Returns

None

XSecure_AesDecryptUpdate

This function decrypts the encrypted data passed in and updates the GCM tag from any previous calls. The size from XSecure_AesDecryptInit is decremented from the size passed into this function to determine when the GCM tag passed to XSecure_AesDecryptInit needs to be compared to the GCM tag calculated in the AES engine.

Note: When Size of the data equals to size of the remaining data that data will be treated as final data. This API can be called multpile times but sum of all Sizes should be equal to Size mention in init. Return of the final call of this API tells whether GCM tag is matching or not.

Prototype

```
s32 XSecure_AesDecryptUpdate(XSecure_Aes *InstancePtr, u8 *EncData, u32
Size);
```

Parameters

The following table lists the `XSecure_AesDecryptUpdate` function arguments.

Table 251: **XSecure_AesDecryptUpdate Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.
EncData	Pointer to the encrypted data which needs to be decrypted.

Table 251: **XSecure_AesDecryptUpdate Arguments** *(cont'd)*

Name	Description
Size	Expected size of data to be decrypted in bytes, whereas the number of bytes should be multiples of 4.

Returns

Final call of this API returns the status of GCM tag matching.

- XSECURE_CSU_AES_GCM_TAG_MISMATCH: If GCM tag is mismatched
- XSECURE_CSU_AES_ZEROIZATION_ERROR: If GCM tag is mismatched, zeroize the decrypted data and send the status of zeroization.
- XST_SUCCESS: If GCM tag is matching.

XSecure_AesDecryptData

This function decrypts the encrypted data provided and updates the DecData buffer with decrypted data.

Note: When using this function to decrypt data that was encrypted with XSecure_AesEncryptData, the GCM tag will be stored as the last sixteen (16) bytes of data in XSecure_AesEncryptData's Dst (destination) buffer and should be used as the GcmTagAddr's pointer.

Prototype

```
s32 XSecure_AesDecryptData(XSecure_Aes *InstancePtr, u8 *DecData, u8
*EncData, u32 Size, u8 *GcmTagAddr);
```

Parameters

The following table lists the `XSecure_AesDecryptData` function arguments.

Table 252: **XSecure_AesDecryptData Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.
DecData	Pointer to a buffer in which decrypted data will be stored.
EncData	Pointer to the encrypted data which needs to be decrypted.
Size	Size of data to be decrypted in bytes, whereas the number of bytes should be multiples of 4.
GcmTagAddr	Pointer to the GCM tag which needs to be verified during decryption of the data.

Returns

This API returns the status of GCM tag matching.

- XSECURE_CSU_AES_GCM_TAG_MISMATCH: If GCM tag was mismatched
- XST_SUCCESS: If GCM tag was matched.

XSecure_AesDecrypt

This function will handle the AES-GCM Decryption.

***Note*:** This function is used for decrypting the Image's partition encrypted by Bootgen

Prototype

```
s32 XSecure_AesDecrypt(XSecure_Aes *InstancePtr, u8 *Dst, const u8 *Src,
u32 Length);
```

Parameters

The following table lists the `XSecure_AesDecrypt` function arguments.

Table 253: **XSecure_AesDecrypt Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.
Src	Pointer to encrypted data source location
Dst	Pointer to location where decrypted data will be written.
Length	Expected total length of decrypted image expected.

Returns

returns XST_SUCCESS if successful, or the relevant errorcode.

XSecure_AesEncryptInit

This function is used to initialize the AES engine for encryption.

***Note*:** If all of the data to be encrypted is available, the XSecure_AesEncryptData function can be used instead.

Prototype

```
u32 XSecure_AesEncryptInit(XSecure_Aes *InstancePtr, u8 *EncData, u32 Size);
```

Parameters

The following table lists the `XSecure_AesEncryptInit` function arguments.

Table 254: **XSecure_AesEncryptInit Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.
EncData	Pointer of a buffer in which encrypted data along with GCM TAG will be stored. Buffer size should be Size of data plus 16 bytes.
Size	A 32 bit variable, which holds the size of the input data to be encrypted in bytes, whereas number of bytes provided should be multiples of 4.

Returns

None

XSecure_AesEncryptUpdate

This function encrypts the clear-text data passed in and updates the GCM tag from any previous calls. The size from XSecure_AesEncryptInit is decremented from the size passed into this function to determine when the final CSU DMA transfer of data to the AES-GCM cryptographic core.

Note: If all of the data to be encrypted is available, the XSecure_AesEncryptData function can be used instead.

Prototype

```
u32 XSecure_AesEncryptUpdate(XSecure_Aes *InstancePtr, const u8 *Data, u32
Size);
```

Parameters

The following table lists the `XSecure_AesEncryptUpdate` function arguments.

Table 255: **XSecure_AesEncryptUpdate Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.
Data	Pointer to the data for which encryption should be performed.
Size	A 32 bit variable, which holds the size of the input data in bytes, whereas the number of bytes provided should be multiples of 4.

Returns

None

XSecure_AesEncryptData

This function encrypts Len (length) number of bytes of the passed in Src (source) buffer and stores the encrypted data along with its associated 16 byte tag in the Dst (destination) buffer.

***Note*:** If data to be encrypted is not available in one buffer one can call `XSecure_AesEncryptInit()` and update the AES engine with data to be encrypted by calling `XSecure_AesEncryptUpdate()` API multiple times as required.

Prototype

```
u32 XSecure_AesEncryptData(XSecure_Aes *InstancePtr, u8 *Dst, const u8
*Src, u32 Len);
```

Parameters

The following table lists the `XSecure_AesEncryptData` function arguments.

Table 256: **XSecure_AesEncryptData Arguments**

Name	Description
InstancePtr	A pointer to the XSecure_Aes instance.
Dst	A pointer to a buffer where encrypted data along with GCM tag will be stored. The Size of buffer provided should be Size of the data plus 16 bytes
Src	A pointer to input data for encryption.
Len	Size of input data in bytes, whereas the number of bytes provided should be multiples of 4.

Returns

None

XSecure_AesReset

This function sets and then clears the AES-GCM's reset line.

Prototype

```
void XSecure_AesReset(XSecure_Aes *InstancePtr);
```

Parameters

The following table lists the `XSecure_AesReset` function arguments.

Table 257: **XSecure_AesReset Arguments**

Name	Description
InstancePtr	is a pointer to the XSecure_Aes instance.

Returns

None

Definitions

XSecure_AesWaitForDone

This macro waits for AES engine completes configured operation.

Definition

```
#define XSecure_AesWaitForDoneXil_WaitForEvent((InstancePtr)->BaseAddress +
XSECURE_CSU_AES_STS_OFFSET,\
                    XSECURE_CSU_AES_STS_AES_BUSY,    \
                    0U,    \
                    XSECURE_AES_TIMEOUT_MAX)
```

Parameters

The following table lists the `XSecure_AesWaitForDone` definition values.

Table 258: **XSecure_AesWaitForDone Values**

Name	Description
InstancePtr	Pointer to the XSecure_Aes instance.

Returns

XST_SUCCESS if the AES engine completes configured operation. XST_FAILURE if a timeout has occurred.

AES-GCM Error Codes

The table below lists the AES-GCM error codes.

Error Code	Error Value	Description
XSECURE_CSU_AES_GCM_TAG_MISMATCH	0x1	User provided GCM tag does not match with GCM calculated on data

Error Code	Error Value	Description
XSECURE_CSU_AES_IMAGE_LEN_MISMATCH	0x2	When there is a Image length mismatch
XSECURE_CSU_AES_DEVICE_COPY_ERROR	0x3	When there is device copy error.
XSECURE_CSU_AES_ZEROIZATION_ERROR	0x4	When there is an error with Zeroization. ***Note***: In case of any error during Aes decryption, we perform zeroization of the decrypted data.
XSECURE_CSU_AES_KEY_CLEAR_ERROR	0x20	Error when clearing key storage registers after Aes operation.

AES-GCM API Example Usage

The following example illustrates the usage of AES encryption and decryption APIs.

```
static s32 SecureAesExample(void)
{
        XCsuDma_Config *Config;
        s32 Status;
        u32 Index;
        XCsuDma CsuDmaInstance;
        XSecure_Aes Secure_Aes;

        /* Initialize CSU DMA driver */
        Config = XCsuDma_LookupConfig(XSECURE_CSUDMA_DEVICEID);
        if (NULL == Config) {
                return XST_FAILURE;
        }

        Status = XCsuDma_CfgInitialize(&CsuDmaInstance, Config,
                                       Config->BaseAddress);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        /* Initialize the Aes driver so that it's ready to use */
        XSecure_AesInitialize(&Secure_Aes, &CsuDmaInstance,
                              XSECURE_CSU_AES_KEY_SRC_KUP,
                              (u32 *)Iv, (u32 *)Key);

        xil_printf("Data to be encrypted: \n\r");
        for (Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
                xil_printf("%02x", Data[Index]);
        }
        xil_printf( "\r\n\n");

        /* Encryption of Data */
        /*
         * If all the data to be encrypted is contiguous one can call
         * XSecure_AesEncryptData API directly.
         */
        XSecure_AesEncryptInit(&Secure_Aes, EncData, XSECURE_DATA_SIZE);
```

```
        XSecure_AesEncryptUpdate(&Secure_Aes, Data, XSECURE_DATA_SIZE);

        xil_printf("Encrypted data: \n\r");
        for (Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
                xil_printf("%02x", EncData[Index]);
        }
        xil_printf( "\r\n");

        xil_printf("GCM tag: \n\r");
        for (Index = 0; Index < XSECURE_SECURE_GCM_TAG_SIZE; Index++) {
                xil_printf("%02x", EncData[XSECURE_DATA_SIZE + Index]);
        }
        xil_printf( "\r\n\n");

        /* Decrypt's the encrypted data */
        /*
         * If data to be decrypted is contiguous one can also call
         * single API XSecure_AesDecryptData
         */
        XSecure_AesDecryptInit(&Secure_Aes, DecData, XSECURE_DATA_SIZE,
                                        EncData + XSECURE_DATA_SIZE);
        /* Only the last update will return the GCM TAG matching status */
        Status = XSecure_AesDecryptUpdate(&Secure_Aes, EncData,
                                                XSECURE_DATA_SIZE);
        if (Status != XST_SUCCESS) {
                xil_printf("Decryption failure- GCM tag was not matched\n
\r");
                return Status;
        }

        xil_printf("Decrypted data\n\r");
        for (Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
                xil_printf("%02x", DecData[Index]);
        }
        xil_printf( "\r\n");

        /* Comparison of Decrypted Data with original data */
        for(Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
                if (Data[Index] != DecData[Index]) {
                        xil_printf("Failure during comparison of the data\n
\r");
                        return XST_FAILURE;
                }
        }

        return XST_SUCCESS;
}
```

***Note*:** Relevant examples are available in the <library-install-path>\examples folder. Where <library-install-path> is the XilSecure library installation path.

AES-GCM Usage to decrypt Boot Image

The Multiple key(Key Rolling) or Single key encrypted images will have the same format. The images include:

- Secure header - This includes the dummy AES key of 32byte + Block 0 IV of 12byte + DLC for Block 0 of 4byte + GCM tag of 16byte(Un-Enc).
- Block N - This includes the boot image data for the block N of n size + Block N+1 AES key of 32byte + Block N+1 IV of 12byte + GCM tag for Block N of 16byte(Un-Enc).

The Secure header and Block 0 will be decrypted using the device key or user provided key. If more than one block is found then the key and the IV obtained from previous block will be used for decryption.

Following are the instructions to decrypt an image:

1. Read the first 64 bytes and decrypt 48 bytes using the selected Device key.
2. Decrypt Block 0 using the IV + Size and the selected Device key.
3. After decryption, you will get the decrypted data+KEY+IV+Block Size. Store the KEY/IV into KUP/IV registers.
4. Using Block size, IV and the next Block key information, start decrypting the next block.
5. If the current image size is greater than the total image length, perform the next step. Else, go back to the previous step.
6. If there are failures, an error code is returned. Else, the decryption is successful.

RSA

The xsecure_rsa.h file contains hardware interface related information for the RSA hardware accelerator. This hardened cryptographic accelerator, within the CSU, performs the modulus math based on the Rivest-Shamir-Adelman (RSA) algorithm. It is an asymmetric algorithm.

The RSA driver instance can be initialized by using the `XSecure_RsaInitialize()` function.

The method used for RSA implementation can take a pre-calculated value of R^2 mod N. If you do not have the pre-calculated exponential value pass NULL, the controller will take care of the exponential value.

***Note*:**

- From the RSA key modulus, the exponent should be extracted.
- For verification, PKCS v1.5 padding scheme has to be applied for comparing the data hash with decrypted hash.

Table 259: **Quick Function Reference**

Type	Name	Arguments
s32	XSecure_RsaInitialize	XSecure_Rsa * InstancePtr u8 * Mod u8 * ModExt u8 * ModExpo
u32	XSecure_RsaSignVerification	u8 * Signature u8 * Hash u32 HashLen
s32	XSecure_RsaPublicEncrypt	XSecure_Rsa * InstancePtr u8 * Input u32 Size u8 * Result
s32	XSecure_RsaPrivateDecrypt	XSecure_Rsa * InstancePtr u8 * Input u32 Size u8 * Result

Functions

XSecure_RsaInitialize

This function initializes a a XSecure_Rsa structure with the default values required for operating the RSA cryptographic engine.

***Note*:** Modulus, ModExt and ModExpo are part of prtition signature when authenticated boot image is generated by bootgen, else the all of them should be extracted from the key.

Prototype

```
s32 XSecure_RsaInitialize(XSecure_Rsa *InstancePtr, u8 *Mod, u8 *ModExt, u8
*ModExpo);
```

Parameters

The following table lists the `XSecure_RsaInitialize` function arguments.

Table 260: **XSecure_RsaInitialize Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Rsa instance.

Table 260: **XSecure_RsaInitialize Arguments** *(cont'd)*

Name	Description
Mod	A character Pointer which contains the key Modulus of key size.
ModExt	A Pointer to the pre-calculated exponential (R^2 Mod N) value. • NULL - if user doesn't have pre-calculated R^2 Mod N value, control will take care of this calculation internally.
ModExpo	Pointer to the buffer which contains key exponent.

Returns

XST_SUCCESS if initialization was successful.

XSecure_RsaSignVerification

This function verifies the RSA decrypted data provided is either matching with the provided expected hash by taking care of PKCS padding.

Prototype

```
u32 XSecure_RsaSignVerification(u8 *Signature, u8 *Hash, u32 HashLen);
```

Parameters

The following table lists the `XSecure_RsaSignVerification` function arguments.

Table 261: **XSecure_RsaSignVerification Arguments**

Name	Description
Signature	Pointer to the buffer which holds the decrypted RSA signature
Hash	Pointer to the buffer which has the hash calculated on the data to be authenticated.
HashLen	Length of Hash used. • For SHA3 it should be 48 bytes

Returns

- XST_SUCCESS if decryption was successful.
- XST_FAILURE in case of mismatch.

XSecure_RsaPublicEncrypt

This function handles the RSA encryption with the public key components provided when initializing the RSA cryptographic core with the XSecure_RsaInitialize function.

***Note*:** The Size passed here needs to match the key size used in the XSecure_RsaInitialize function.

Prototype

```
s32 XSecure_RsaPublicEncrypt(XSecure_Rsa *InstancePtr, u8 *Input, u32 Size,
u8 *Result);
```

Parameters

The following table lists the `XSecure_RsaPublicEncrypt` function arguments.

Table 262: **XSecure_RsaPublicEncrypt Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Rsa instance.
Input	Pointer to the buffer which contains the input data to be encrypted.
Size	Key size in bytes, Input size also should be same as Key size mentioned.Inputs supported are • XSECURE_RSA_4096_KEY_SIZE • XSECURE_RSA_2048_KEY_SIZE • XSECURE_RSA_3072_KEY_SIZE
Result	Pointer to the buffer where resultant decrypted data to be stored .

Returns

- XST_SUCCESS if encryption was successful.
- Error code on failure

XSecure_RsaPrivateDecrypt

This function handles the RSA decryption with the private key components provided when initializing the RSA cryptographic core with the XSecure_RsaInitialize function.

***Note*:** The Size passed in needs to match the key size used in the XSecure_RsaInitialize function..

Prototype

```
s32 XSecure_RsaPrivateDecrypt(XSecure_Rsa *InstancePtr, u8 *Input, u32
Size, u8 *Result);
```

Parameters

The following table lists the `XSecure_RsaPrivateDecrypt` function arguments.

Table 263: **XSecure_RsaPrivateDecrypt Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Rsa instance.
Input	Pointer to the buffer which contains the input data to be decrypted.
Size	Key size in bytes, Input size also should be same as Key size mentioned. Inputs supported are • XSECURE_RSA_4096_KEY_SIZE, • XSECURE_RSA_2048_KEY_SIZE • XSECURE_RSA_3072_KEY_SIZE
Result	Pointer to the buffer where resultant decrypted data to be stored .

Returns

- XST_SUCCESS if decryption was successful.
- XSECURE_RSA_DATA_VALUE_ERROR - if input data is greater than modulus.
- XST_FAILURE - on RSA operation failure.

Definitions

XSECURE_RSA_BYTE_PAD_LENGTH

PKCS Byte Padding

Definition

```
#define XSECURE_RSA_BYTE_PAD_LENGTH(3U)
```

XSECURE_RSA_T_PAD_LENGTH

PKCS T Padding

Definition

```
#define XSECURE_RSA_T_PAD_LENGTH(19U)
```

XSECURE_RSA_BYTE_PAD1

PKCS T Padding Byte

Definition

```
#define XSECURE_RSA_BYTE_PAD1(0X00U)
```

XSECURE_RSA_BYTE_PAD2

PKCS T Padding Byte

Definition

```
#define XSECURE_RSA_BYTE_PAD2(0X01U)
```

XSECURE_RSA_BYTE_PAD3

PKCS T Padding Byte

Definition

```
#define XSECURE_RSA_BYTE_PAD3(0XFFU)
```

RSA API Example Usage

The following example illustrates the usage of the RSA library to encrypt data using the public key and to decrypt the data using private key.

***Note*:** Application should take care of the padding.

```
u32 SecureRsaExample(void)
{
        u32 Index;

        /* RSA signature decrypt with private key */
        /*
         * Initialize the Rsa driver with private key components
         * so that it's ready to use
         */
        XSecure_RsaInitialize(&Secure_Rsa, Modulus, NULL, PrivateExp);


        if(XST_SUCCESS != XSecure_RsaPrivateDecrypt(&Secure_Rsa, Data,
                                                Size, Signature))       {
                xil_printf("Failed at RSA signature decryption\n\r");
                return XST_FAILURE;
        }

        xil_printf("\r\n Decrypted Signature with private key\r\n ");

        for(Index = 0; Index < Size; Index++) {
                xil_printf(" %02x ", Signature[Index]);
        }
```

```
        xil_printf(" \r\n ");

        /* Verification if Data is expected */
        for(Index = 0; Index < Size; Index++) {
                if (Signature[Index] != ExpectedSign[Index]) {
                        xil_printf("\r\nError at verification of RSA
signature"
                                        " Decryption\n\r");
                        return XST_FAILURE;
                }
        }

        /* RSA signature encrypt with Public key components */

        /*
         * Initialize the Rsa driver with public key components
         * so that it's ready to use
         */

        XSecure_RsaInitialize(&Secure_Rsa, Modulus, NULL, (u8 *)&PublicExp);

        if(XST_SUCCESS != XSecure_RsaPublicEncrypt(&Secure_Rsa, Signature,
                                                Size,
EncryptSignatureOut))     {
                xil_printf("\r\nFailed at RSA signature encryption\n\r");
                return XST_FAILURE;
        }
        xil_printf("\r\n Encrypted Signature with public key\r\n ");

        for(Index = 0; Index < Size; Index++) {
                xil_printf(" %02x ", EncryptSignatureOut[Index]);
        }

        /* Verification if Data is expected */
        for(Index = 0; Index < Size; Index++) {
                if (EncryptSignatureOut[Index] != Data[Index]) {
                        xil_printf("\r\nError at verification of RSA
signature"
                                        " encryption\n\r");
                        return XST_FAILURE;
                }
        }


        return XST_SUCCESS;
}
```

***Note*:** Relevant examples are available in the <library-install-path>\examples folder. Where <library-install-path> is the XilSecure library installation path.

SHA-3

This block uses the NIST-approved SHA-3 algorithm to generate a 384-bit hash on the input data. Because the SHA-3 hardware only accepts 104 byte blocks as the minimum input size, the input data will be padded with user selectable Keccak or NIST SHA-3 padding and is handled internally in the SHA-3 library.

The SHA-3 driver instance can be initialized using the `XSecure_Sha3Initialize()` function.

A pointer to CsuDma instance has to be passed during initialization as the CSU DMA will be used for data transfers to the SHA module.

When all the data is available on which the SHA3 hash must be calculated, the `XSecure_Sha3Digest()` can be used with the appropriate parameters as described. When all the data is not available, use the SHA3 functions in the following order:

1. `XSecure_Sha3Start()`
2. `XSecure_Sha3Update()` - This function can be called multiple times until all input data has been passed to the SHA-3 cryptographic core.
3. `XSecure_Sha3Finish()` - Provides the final hash of the data. To get intermediate hash values after each `XSecure_Sha3Update()` , you can call `XSecure_Sha3_ReadHash()` after the `XSecure_Sha3Update()` call.

Table 264: **Quick Function Reference**

Type	Name	Arguments
s32	XSecure_Sha3Initialize	XSecure_Sha3 * InstancePtr XCsuDma * CsuDmaPtr
void	XSecure_Sha3Start	XSecure_Sha3 * InstancePtr
u32	XSecure_Sha3Update	XSecure_Sha3 * InstancePtr const u8 * Data const u32 Size
u32	XSecure_Sha3Finish	XSecure_Sha3 * InstancePtr u8 * Hash
u32	XSecure_Sha3Digest	XSecure_Sha3 * InstancePtr const u8 * In const u32 Size u8 * Out
void	XSecure_Sha3_ReadHash	XSecure_Sha3 * InstancePtr u8 * Hash
s32	XSecure_Sha3PadSelection	XSecure_Sha3 * InstancePtr XSecure_Sha3PadType Sha3Type
s32	XSecure_Sha3LastUpdate	XSecure_Sha3 * InstancePtr

Table 264: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
u32	XSecure_Sha3WaitForDone	XSecure_Sha3 * InstancePtr

Functions

XSecure_Sha3Initialize

This function initializes a XSecure_Sha3 structure with the default values required for operating the SHA3 cryptographic engine.

***Note*:** The base address is initialized directly with value from xsecure_hw.h The default is NIST SHA3 padding, to change to KECCAK padding call `XSecure_Sha3PadSelection()` after `XSecure_Sha3Initialize()`.

Prototype

```
s32 XSecure_Sha3Initialize(XSecure_Sha3 *InstancePtr, XCsuDma *CsuDmaPtr);
```

Parameters

The following table lists the `XSecure_Sha3Initialize` function arguments.

Table 265: **XSecure_Sha3Initialize Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.
CsuDmaPtr	Pointer to the XCsuDma instance.

Returns

XST_SUCCESS if initialization was successful

XSecure_Sha3Start

This function configures Secure Stream Switch and starts the SHA-3 engine.

Prototype

```
void XSecure_Sha3Start(XSecure_Sha3 *InstancePtr);
```

Parameters

The following table lists the `XSecure_Sha3Start` function arguments.

Table 266: **XSecure_Sha3Start Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.

Returns

None

XSecure_Sha3Update

This function updates the SHA3 engine with the input data.

Prototype

```
u32 XSecure_Sha3Update(XSecure_Sha3 *InstancePtr, const u8 *Data, const u32
Size);
```

Parameters

The following table lists the `XSecure_Sha3Update` function arguments.

Table 267: **XSecure_Sha3Update Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.
Data	Pointer to the input data for hashing.
Size	Size of the input data in bytes.

Returns

XST_SUCCESS if the update is successful XST_FAILURE if there is a failure in SSS config

XSecure_Sha3Finish

This function updates SHA3 engine with final data which includes SHA3 padding and reads final hash on complete data.

Prototype

```
u32 XSecure_Sha3Finish(XSecure_Sha3 *InstancePtr, u8 *Hash);
```

Parameters

The following table lists the `XSecure_Sha3Finish` function arguments.

Table 268: **XSecure_Sha3Finish Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.
Hash	Pointer to location where resulting hash will be written

Returns

XST_SUCCESS if finished without any errors XST_FAILURE if Sha3PadType is other than KECCAK or NIST

XSecure_Sha3Digest

This function calculates the SHA-3 digest on the given input data.

Prototype

```
u32 XSecure_Sha3Digest(XSecure_Sha3 *InstancePtr, const u8 *In, const u32
Size, u8 *Out);
```

Parameters

The following table lists the `XSecure_Sha3Digest` function arguments.

Table 269: **XSecure_Sha3Digest Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.
In	Pointer to the input data for hashing
Size	Size of the input data
Out	Pointer to location where resulting hash will be written.

Returns

XST_SUCCESS if digest calculation done successfully XST_FAILURE if any error from Sha3Update or Sha3Finish.

XSecure_Sha3_ReadHash

This function reads the SHA3 hash of the data and it can be called between calls to XSecure_Sha3Update.

Prototype

```
void XSecure_Sha3_ReadHash(XSecure_Sha3 *InstancePtr, u8 *Hash);
```

Parameters

The following table lists the `XSecure_Sha3_ReadHash` function arguments.

Table 270: **XSecure_Sha3_ReadHash Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.
Hash	Pointer to a buffer in which read hash will be stored.

Returns

None

XSecure_Sha3PadSelection

This function provides an option to select the SHA-3 padding type to be used while calculating the hash.

***Note*:** The default provides support for NIST SHA-3. If a user wants to change the padding to Keccak SHA-3, this function should be called after `XSecure_Sha3Initialize()`

Prototype

```
s32 XSecure_Sha3PadSelection(XSecure_Sha3 *InstancePtr, XSecure_Sha3PadType
Sha3PadType);
```

Parameters

The following table lists the `XSecure_Sha3PadSelection` function arguments.

Table 271: **XSecure_Sha3PadSelection Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.
Sha3Type	Type of SHA3 padding to be used. • For NIST SHA-3 padding - XSECURE_CSU_NIST_SHA3 • For KECCAK SHA-3 padding - XSECURE_CSU_KECCAK_SHA3

Returns

XST_SUCCESS if pad selection is successful. XST_FAILURE if pad selecction is failed.

XSecure_Sha3LastUpdate

This function is to notify this is the last update of data where sha padding is also been included along with the data in the next update call.

Prototype

```
s32 XSecure_Sha3LastUpdate(XSecure_Sha3 *InstancePtr);
```

Parameters

The following table lists the `XSecure_Sha3LastUpdate` function arguments.

Table 272: **XSecure_Sha3LastUpdate Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.
XSecure_Sha3PadType	

Returns

XST_SUCCESS if last update can be accepted

XSecure_Sha3WaitForDone

This inline function waits till SHA3 completes its operation.

Prototype

```
u32 XSecure_Sha3WaitForDone(XSecure_Sha3 *InstancePtr);
```

Parameters

The following table lists the `XSecure_Sha3WaitForDone` function arguments.

Table 273: **XSecure_Sha3WaitForDone Arguments**

Name	Description
InstancePtr	Pointer to the XSecure_Sha3 instance.

Returns

XST_SUCCESS if the SHA3 completes its operation. XST_FAILURE if a timeout has occurred.

SHA-3 API Example Usage

The xilsecure_sha_example.c file is a simple example application that demonstrates the usage of SHA-3 accelerator to calculate a 384-bit hash on the Hello World string. A typical use case for the SHA3 accelerator is for calcuation of the boot image hash as part of the autentication operation. This is illustrated in the xilsecure_rsa_example.c.

The contents of the xilsecure_sha_example.c file are shown below:

```
u32 SecureSha3Example()
{
        XSecure_Sha3 Secure_Sha3;
        XCsuDma CsuDma;
        XCsuDma_Config *Config;
        u8 Out[SHA3_HASH_LEN_IN_BYTES];
        u32 Status = XST_FAILURE;
        u32 Size = 0U;

        Size = Xil_Strnlen(Data, SHA3_INPUT_DATA_LEN);
        if (Size != SHA3_INPUT_DATA_LEN) {
                xil_printf("Provided data length is Invalid\n\r");
                Status = XST_FAILURE;
                goto END;
        }

        Config = XCsuDma_LookupConfig(0);
        if (NULL == Config) {
                xil_printf("config failed\n\r");
                Status = XST_FAILURE;
                goto END;
        }

        Status = XCsuDma_CfgInitialize(&CsuDma, Config, Config-
>BaseAddress);
        if (Status != XST_SUCCESS) {
                Status = XST_FAILURE;
                goto END;
        }

        /*
         * Initialize the SHA-3 driver so that it's ready to use
         */
        XSecure_Sha3Initialize(&Secure_Sha3, &CsuDma);

        XSecure_Sha3Digest(&Secure_Sha3, (const u8*)Data, Size, Out);

        xil_printf(" Calculated Hash \r\n ");
        SecureSha3PrintHash(Out);

        Status = SecureSha3CompareHash(Out, ExpHash);
END:
        return Status;
}

/
*****************************************************************************
/
u32 SecureSha3CompareHash(u8 *Hash, u8 *ExpectedHash)
```

```
{
        u32 Index;
        u32 Status = XST_FAILURE;

        for (Index = 0U; Index < SHA3_HASH_LEN_IN_BYTES; Index++) {
                if (Hash[Index] != ExpectedHash[Index]) {
                        xil_printf("Expected Hash \r\n");
                        SecureSha3PrintHash(ExpectedHash);
                        xil_printf("SHA Example Failed at Hash Comparison \r
\n");
                        break;
                }
        }
        if (Index == SHA3_HASH_LEN_IN_BYTES) {
                Status = XST_SUCCESS;
        }

        return Status;
}

/
*****************************************************************************
/
void SecureSha3PrintHash(u8 *Hash)
{
        u32 Index;

        for (Index = 0U; Index < SHA3_HASH_LEN_IN_BYTES; Index++) {
                xil_printf(" %0x ", Hash[Index]);
        }
        xil_printf(" \r\n ");
 }
```

***Note*:** The xilsecure_sha_example.c and xilsecure_rsa_example.c example files are available in the <library-install-path>\examples folder. Where <library-install-path> is the XilSecure library installation path.

XilSkey Library v7.0

Overview

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

- In Zynq-7000 devices:
  - PS eFUSE holds the RSA primary key hash bits and user feature bits, which can enable or disable some Zynq-7000 processor features.
  - PL eFUSE holds the AES key, the user key and some of the feature bits.
  - PL BBRAM holds the AES key.
- In Kintex/Virtex UltraScale or UltraScale+:
  - PL eFUSE holds the AES key, 32 bit and 128 bit user key, RSA hash and some of the feature bits.
  - PL BBRAM holds AES key with or without DPA protection enable or obfuscated key programming.
- In Zynq UltraScale+ MPSoC:
  - PUF registration and Regeneration.
  - PS eFUSE holds:

Programming AES key and can perform CRC verification of AES key

- Programming/Reading User fuses
- Programming/Reading PPK0/PPK1 sha3 hash
- Programming/Reading SPKID
- Programming/Reading secure control bits
  - PS BBRAM holds the AES key.
  - PL eFUSE holds the AES key, 32 bit and 128 bit user key, RSA hash and some of the feature bits.

- PL BBRAM holds AES key with or without DPA protection enable or obfuscated key programming.

BOARD Support Package Settings

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

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

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

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

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

Hardware Setup

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

Hardware setup for Zynq PL

This section 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:

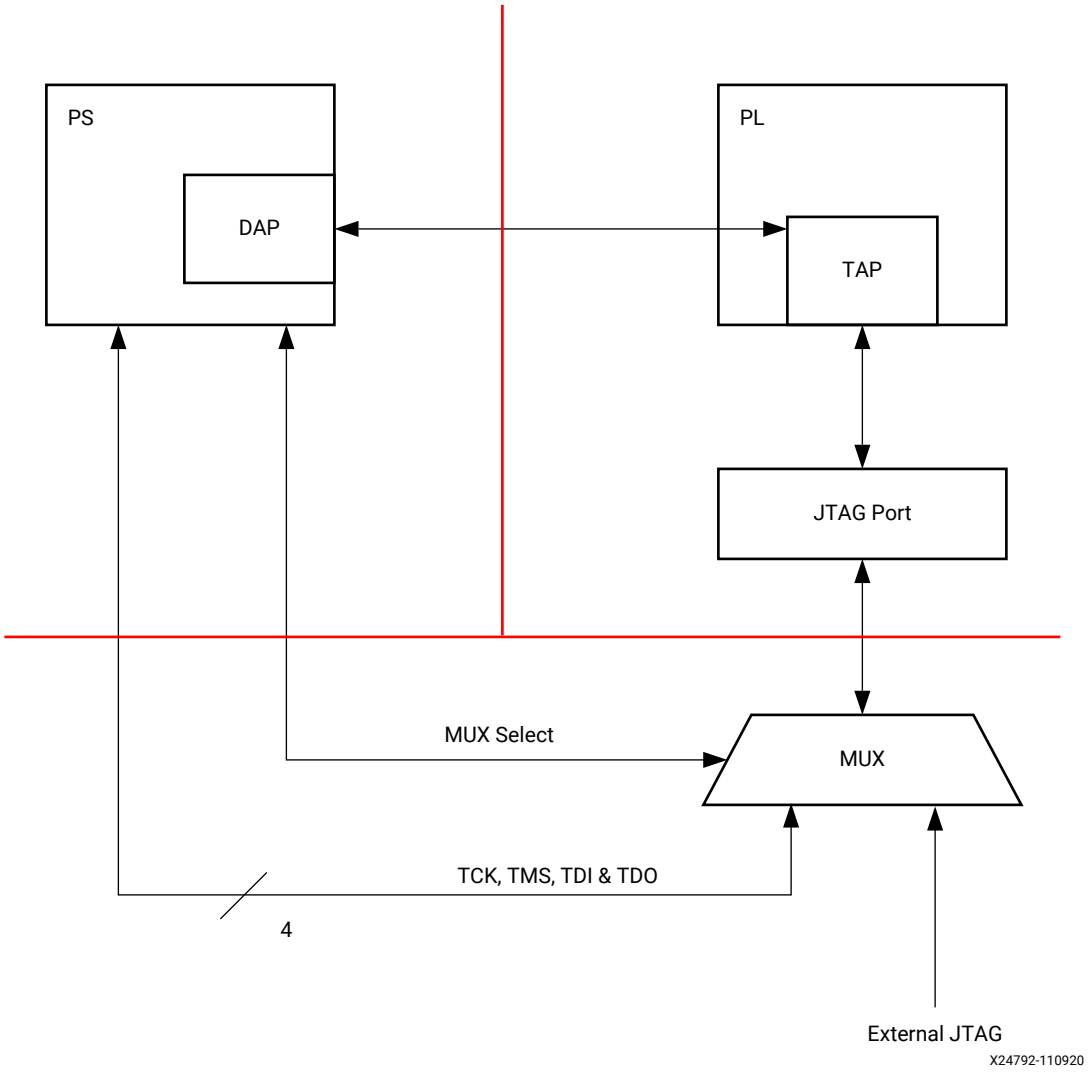
- Include a MUX between the external JTAG and the JTAG operation driven by the MIOs
- Assign a MUX selection PIN

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 2: **MUX Usage**



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

Hardware setup for UltraScale or UltraScale+

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

- You need to add the Master JTAG primitive to design, that is, the MASTER_JTAG_inst instantiation has to be performed and AXI GPIO pins have to be connected to TDO, TDI, TMS and TCK signals of the MASTER_JTAG primitive.
- For programming eFUSE, along with master JTAG, hardware module(HWM) has to be added in design and it's signals XSK_EFUSEPL_AXI_GPIO_HWM_READY , XSK_EFUSEPL_AXI_GPIO_HWM_END and XSK_EFUSEPL_AXI_GPIO_HWM_START, needs to be connected to AXI GPIO pins to communicate with HWM. Hardware module is not mandatory for programming BBRAM. If your design has a HWM, it is not harmful for accessing BBRAM.
- All inputs (Master JTAG's TDO and HWM's HWM_READY, HWM_END) and all outputs (Master JTAG TDI, TMS, TCK and HWM's HWM_START) can be connected in one channel (or) inputs in one channel and outputs in other channel.
- Some of the outputs of GPIO in one channel and some others in different channels are not supported.
- The design should contain AXI BRAM control memory mapped (1MB).

***Note*:** MASTER_JTAG will disable all other JTAGs.

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

Source Files

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

- xilskey_efuse_example.c: This file contains the main application code. The file helps in the PS/PL structure initialization and writes/reads the PS/PL eFUSE based on the user settings provided in the xilskey_input.h file.
- xilskey_input.h: This file ontains all the actions that are supported by the eFUSE library. Using the preprocessor directives given in the file, you can read/write the bits in the PS/PL eFUSE. More explanation of each directive is provided in the following sections. Burning or reading the PS/PL eFUSE bits is based on the values set in the xilskey_input.h file. Also contains GPIO pins and channels connected to MASTER JTAG primitive and hardware module to access Ultrascale eFUSE.

  In this file:

  - specify the 256 bit key to be programmed into BBRAM.
  - specify the AES(256 bit) key, User (32 bit and 128 bit) keys and RSA key hash(384 bit) key to be programmed into UltraScale eFUSE.

    - XSK_EFUSEPS_DRIVER: Define to enable the writing and reading of PS eFUSE.
    - XSK_EFUSEPL_DRIVER: Define to enable the writing of PL eFUSE.

- xilskey_bbram_example.c: This file contains the example to program a key into BBRAM and verify the key.

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

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

  Note: Programming and verification of BBRAM key cannot be done separately.

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

> **CAUTION!** *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.

BBRAM PL API

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

Example Usage

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

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

Table 274: **Quick Function Reference**

Type	Name	Arguments
int	XilSKey_Bbram_Program	XilSKey_Bbram * InstancePtr
int	XilSKey_Bbram_JTAGServerInit	XilSKey_Bbram * InstancePtr

Functions

XilSKey_Bbram_Program

This function implements the BBRAM algorithm for programming and verifying key.

The program and verify will only work together in and in that order.

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

Prototype

```
int XilSKey_Bbram_Program(XilSKey_Bbram *InstancePtr);
```

Parameters

The following table lists the `XilSKey_Bbram_Program` function arguments.

Table 275: **XilSKey_Bbram_Program Arguments**

Type	Name	Description
XilSKey_Bbram *	InstancePtr	Pointer to XilSKey_Bbram

Returns

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

XilSKey_Bbram_JTAGServerInit

This function initializes JTAG server.

Prototype

```
int XilSKey_Bbram_JTAGServerInit(XilSKey_Bbram *InstancePtr);
```

Parameters

The following table lists the `XilSKey_Bbram_JTAGServerInit` function arguments.

Table 276: **XilSKey_Bbram_JTAGServerInit Arguments**

Type	Name	Description
XilSKey_Bbram *	InstancePtr	Pointer to XilSKey_Bbram

Returns

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

Zynq UltraScale+ MPSoC BBRAM PS API

This section 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.

Table 277: **Quick Function Reference**

Type	Name	Arguments
u32	XilSKey_ZynqMp_Bbram_Program	u32 * AesKey
u32	XilSKey_ZynqMp_Bbram_Zeroise	void

Functions

XilSKey_ZynqMp_Bbram_Program

This function implements the BBRAM programming and verifying the key written.

Program and verification of AES will work only together. CRC of the provided key will be calculated internally and verified after programming.

Prototype

```
u32 XilSKey_ZynqMp_Bbram_Program(u32 *AesKey);
```

Parameters

The following table lists the `XilSKey_ZynqMp_Bbram_Program` function arguments.

Table 278: **XilSKey_ZynqMp_Bbram_Program Arguments**

Type	Name	Description
u32 *	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.

XilSKey_ZynqMp_Bbram_Zeroise

This function zeroizes Bbram Key.

Note: BBRAM key will be zeroized.

Prototype

```
u32 XilSKey_ZynqMp_Bbram_Zeroise(void);
```

Returns

Zynq eFUSE PS API

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`.

Table 279: **Quick Function Reference**

Type	Name	Arguments
u32	XilSKey_EfusePs_Write	InstancePtr
u32	XilSKey_EfusePs_Read	InstancePtr
u32	XilSKey_EfusePs_ReadStatus	XilSKey_EPs * InstancePtr u32 * StatusBits

Functions

XilSKey_EfusePs_Write

This function is used to write to the PS eFUSE.

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

Prototype

```
u32 XilSKey_EfusePs_Write(XilSKey_EPs *PsInstancePtr);
```

Parameters

The following table lists the `XilSKey_EfusePs_Write` function arguments.

Table 280: **XilSKey_EfusePs_Write Arguments**

Type	Name	Description
Commented parameter InstancePtr does not exist in function XilSKey_EfusePs_Write.	InstancePtr	Pointer to the PsEfuseHandle which describes which PS eFUSE bit should be burned.

Returns

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

XilSKey_EfusePs_Read

This function is used to read the PS eFUSE.

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

Prototype

```
u32 XilSKey_EfusePs_Read(XilSKey_EPs *PsInstancePtr);
```

Parameters

The following table lists the `XilSKey_EfusePs_Read` function arguments.

Table 281: **XilSKey_EfusePs_Read Arguments**

Type	Name	Description
Commented parameter InstancePtr does not exist in function XilSKey_EfusePs_Read.	InstancePtr	Pointer to the PsEfuseHandle which describes which PS eFUSE should be burned.

Returns

- XST_SUCCESS no errors occurred.

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

XilSKey_EfusePs_ReadStatus

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

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

Prototype

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

Parameters

The following table lists the `XilSKey_EfusePs_ReadStatus` function arguments.

Table 282: **XilSKey_EfusePs_ReadStatus Arguments**

Type	Name	Description
XilSKey_EPs *	InstancePtr	Pointer to the PS eFUSE instance.
u32 *	StatusBits	Buffer to store the status register read.

Returns

- XST_SUCCESS.
- XST_FAILURE

Zynq UltraScale+ MPSoC eFUSE PS API

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

Example Usage

- For programming eFUSEs other than the PUF, the Zynq UltraScale+ MPSoC example application should contain the xilskey_efuseps_zynqmp_example.c and the xilskey_efuseps_zynqmp_input.h files.
- For PUF registration, programming PUF helper data, AUX, chash, and black key, the Zynq UltraScale+ MPSoC example application should contain the xilskey_puf_registration.c and the xilskey_puf_registration.h files.

- For more details on the user configurable parameters, refer Zynq UltraScale+ MPSoC User-Configurable PS eFUSE Parameters and Zynq UltraScale+ MPSoC User-Configurable PS PUF Parameters.

Table 283: **Quick Function Reference**

Type	Name	Arguments
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_CacheLoad	void
u32	XilSKey_ZynqMp_EfusePs_Write	XilSKey_ZynqMpEPs * InstancePtr
u32	XilSkey_ZynqMpEfuseAccess	const u32 AddrHigh const u32 AddrLow
void	XilSKey_ZynqMp_EfusePs_SetTimerValues	void
u32	XilSKey_ZynqMp_EfusePs_ReadRow	u8 Row XskEfusePs_Type EfuseType u32 * RowData
u32	XilSKey_ZynqMp_EfusePs_SetWriteConditions	void

Table 283: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
u32	XilSKey_ZynqMp_EfusePs_WriteAndVerifyBit	u8 Row u8 Column XskEfusePs_Type EfuseType
u32	XilSKey_ZynqMp_EfusePs_Init	void
u32	XilSKey_ZynqMp_EfusePs_CheckForZeros	u8 RowStart u8 RowEnd XskEfusePs_Type EfuseType
u32	XilSKey_ZynqMp_EfusePs_WritePufHelprData	const XilSKey_Puf * InstancePtr
u32	XilSKey_ZynqMp_EfusePs_ReadPufHelprData	u32 * Address
u32	XilSKey_ZynqMp_EfusePs_WritePufChash	const XilSKey_Puf * InstancePtr
u32	XilSKey_ZynqMp_EfusePs_ReadPufChash	u32 * Address u8 ReadOption
u32	XilSKey_ZynqMp_EfusePs_WritePufAux	const XilSKey_Puf * InstancePtr
u32	XilSKey_ZynqMp_EfusePs_ReadPufAux	u32 * Address u8 ReadOption
u32	XilSKey_Write_Puf_EfusePs_SecureBits	const XilSKey_Puf_Secure * WriteSecureBits
u32	XilSKey_Read_Puf_EfusePs_SecureBits	SecureBits u8 ReadOption
u32	XilSKey_Puf_Registration	XilSKey_Puf * InstancePtr
u32	XilSKey_Puf_Regeneration	const XilSKey_Puf * InstancePtr

Functions

XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc

This function performs the CRC check of AES key.

***Note*:** For Calculating the CRC of the AES key use the `XilSKey_CrcCalculation()` function or `XilSkey_CrcCalculation_AesKey()` function

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc(u32 CrcValue);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc` function arguments.

Table 284: **XilSKey_ZynqMp_EfusePs_CheckAesKeyCrc Arguments**

Type	Name	Description
u32	CrcValue	A 32 bit CRC value of an expected AES key.

Returns

- XST_SUCCESS on successful CRC check.
- ErrorCode on failure

XilSKey_ZynqMp_EfusePs_ReadUserFuse

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

***Note*:** It is highly recommended to read from eFuse cache. Because reading from efuse may reduce the life of the efuse. And Cache reload is required for obtaining updated values for ReadOption 0.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_ReadUserFuse(u32 *UseFusePtr, u8 UserFuse_Num,
u8 ReadOption);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadUserFuse` function arguments.

Table 285: **XilSKey_ZynqMp_EfusePs_ReadUserFuse Arguments**

Type	Name	Description
u32 *	UseFusePtr	Pointer to an array which holds the readback user fuse.
u8	UserFuse_Num	A variable which holds the user fuse number. Range is (User fuses: 0 to 7)

Table 285: **XilSKey_ZynqMp_EfusePs_ReadUserFuse Arguments** *(cont'd)*

Type	Name	Description
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array

Returns

- XST_SUCCESS on successful read
- ErrorCode on failure

XilSKey_ZynqMp_EfusePs_ReadPpk0Hash

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

***Note*:** It is highly recommended to read from eFuse cache. Because reading from efuse may reduce the life of the efuse. And Cache reload is required for obtaining updated values for ReadOption 0.

Prototype

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

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadPpk0Hash` function arguments.

Table 286: **XilSKey_ZynqMp_EfusePs_ReadPpk0Hash Arguments**

Type	Name	Description
u32 *	Ppk0Hash	A pointer to an array which holds the readback PPK0 hash.
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array

Returns

- XST_SUCCESS on successful read
- ErrorCode on failure

XilSKey_ZynqMp_EfusePs_ReadPpk1Hash

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

***Note*:** It is highly recommended to read from eFuse cache. Because reading from efuse may reduce the life of the efuse. And Cache reload is required for obtaining updated values for ReadOption 0.

Prototype

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

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadPpk1Hash` function arguments.

Table 287: **XilSKey_ZynqMp_EfusePs_ReadPpk1Hash Arguments**

Type	Name	Description
u32 *	Ppk1Hash	Pointer to an array which holds the readback PPK1 hash.
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array

Returns

- XST_SUCCESS on successful read
- ErrorCode on failure

XilSKey_ZynqMp_EfusePs_ReadSpkId

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

***Note*:** It is highly recommended to read from eFuse cache. Because reading from efuse may reduce the life of the efuse. And Cache reload is required for obtaining updated values for ReadOption 0.

Prototype

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

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadSpkId` function arguments.

Table 288: **XilSKey_ZynqMp_EfusePs_ReadSpkId Arguments**

Type	Name	Description
u32 *	SpkId	Pointer to a 32 bit variable which holds SPK ID.
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array

Returns

- XST_SUCCESS on successful read
- ErrorCode on failure

XilSKey_ZynqMp_EfusePs_ReadDna

This function is used to read DNA from eFUSE.

Prototype

```
void XilSKey_ZynqMp_EfusePs_ReadDna(u32 *DnaRead);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadDna` function arguments.

Table 289: **XilSKey_ZynqMp_EfusePs_ReadDna Arguments**

Type	Name	Description
u32 *	DnaRead	Pointer to an array of 3 x u32 words which holds the readback DNA.

XilSKey_ZynqMp_EfusePs_ReadSecCtrlBits

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

***Note*:** It is highly recommended to read from eFuse cache. Because reading from efuse may reduce the life of the efuse. And Cache reload is required for obtaining updated values for ReadOption 0.

Prototype

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

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadSecCtrlBits` function arguments.

Table 290: **XilSKey_ZynqMp_EfusePs_ReadSecCtrlBits Arguments**

Type	Name	Description
XilSKey_SecCtrlBits *	ReadBackSecCtrlBits	Pointer to the XilSKey_SecCtrlBits which holds the read secure control bits.
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from eFUSE cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array

Returns

- XST_SUCCESS if reads successfully
- XST_FAILURE if reading is failed

XilSKey_ZynqMp_EfusePs_CacheLoad

This function reloads the cache of eFUSE so that can be directly read from cache.

***Note*:** Not recommended to call this API frequently, if this API is called all the cache memory is reloded by reading eFUSE array, reading eFUSE bit multiple times may diminish the life time.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_CacheLoad(void);
```

Returns

- XST_SUCCESS on successful cache reload
- ErrorCode on failure

XilSKey_ZynqMp_EfusePs_Write

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

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

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_Write(XilSKey_ZynqMpEPs *InstancePtr);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_Write` function arguments.

Table 291: **XilSKey_ZynqMp_EfusePs_Write Arguments**

Type	Name	Description
XilSKey_ZynqMpEPs *	InstancePtr	Pointer to the XilSKey_ZynqMpEPs.

Returns

- XST_SUCCESS if programs successfully.
- Errorcode on failure

XilSkey_ZynqMpEfuseAccess

This function is used by PMUFW IPI call handler for programming eFUSE.

Prototype

```
u32 XilSkey_ZynqMpEfuseAccess(const u32 AddrHigh, const u32 AddrLow);
```

Parameters

The following table lists the `XilSkey_ZynqMpEfuseAccess` function arguments.

Table 292: **XilSkey_ZynqMpEfuseAccess Arguments**

Type	Name	Description
const u32	AddrHigh	Higher 32-bit address of the XilSKey_Efuse structure.
const u32	AddrLow	Lower 32-bit address of the XilSKey_Efuse structure.

Returns

XST_SUCCESS - On success ErrorCode - on Failure

***Note*:** eFUSEs accessible from Linux are as follows:

Table 293: **eFUSEs Accessible from Linux**

Register	Read	Write	Size in bytes	Offset
DNA	YES	NO	0xc	0xC

Table 293: **eFUSEs Accessible from Linux** *(cont'd)*

Register	Read	Write	Size in bytes	Offset
User0	YES	YES	0x4	0x20
User1	YES	YES	0x4	0x24
User2	YES	YES	0x4	0x28
User3	YES	YES	0x4	0x2c
User4	YES	YES	0x4	0x30
User5	YES	YES	0x4	0x34
User6	YES	YES	0x4	0x38
User7	YES	YES	0x4	0x3C
Miscellaneous User	YES	YES	0x4	0x40
Secure control	YES	YES	0x4	0x58
SPK ID	YES	0x4	0x4	0x5C
AES Key	YES	0x20	0x20	0x60
PPK0 Hash	YES	YES	0x30	0xA0
PPK1 Hash	YES	YES	0x30	0xD0

XilSKey_ZynqMp_EfusePs_SetTimerValues

This function sets timers for programming and reading from eFUSE.

Prototype

```
void XilSKey_ZynqMp_EfusePs_SetTimerValues(void);
```

XilSKey_ZynqMp_EfusePs_ReadRow

This function returns particular row data directly from eFUSE array.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_ReadRow(u8 Row, XskEfusePs_Type EfuseType, u32
*RowData);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadRow` function arguments.

Table 294: **XilSKey_ZynqMp_EfusePs_ReadRow Arguments**

Type	Name	Description
u8	Row	specifies the row number to read.
XskEfusePs_Type	EfuseType	specifies the eFUSE type.

Table 294: **XilSKey_ZynqMp_EfusePs_ReadRow Arguments** *(cont'd)*

Type	Name	Description
u32 *	RowData	is a pointer to 32 bit variable to hold the data read from provided data

Returns

XST_SUCCESS - On success ErrorCode - on Failure

XilSKey_ZynqMp_EfusePs_SetWriteConditions

This function sets all the required parameters to program eFUSE array.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_SetWriteConditions(void);
```

Returns

XST_SUCCESS - On success ErrorCode - on Failure

XilSKey_ZynqMp_EfusePs_WriteAndVerifyBit

This function programs and verifies the particular bit of eFUSE array.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_WriteAndVerifyBit(u8 Row, u8 Column,
XskEfusePs_Type EfuseType);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_WriteAndVerifyBit` function arguments.

Table 295: **XilSKey_ZynqMp_EfusePs_WriteAndVerifyBit Arguments**

Type	Name	Description
u8	Row	specifies the row number.
u8	Column	specifies the column number.
XskEfusePs_Type	EfuseType	specifies the eFUSE type.

Returns

XST_SUCCESS - On success ErrorCode - on Failure

XilSKey_ZynqMp_EfusePs_Init

This function initializes sysmonpsu driver.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_Init(void);
```

Returns

XST_SUCCESS - On success ErrorCode - on Failure

XilSKey_ZynqMp_EfusePs_CheckForZeros

This function is used verify eFUSE keys for Zeros.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_CheckForZeros(u8 RowStart, u8 RowEnd,
XskEfusePs_Type EfuseType);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_CheckForZeros` function arguments.

Table 296: **XilSKey_ZynqMp_EfusePs_CheckForZeros Arguments**

Type	Name	Description
u8	RowStart	is row number from which verification has to be started.
u8	RowEnd	is row number till which verification has to be ended.
XskEfusePs_Type	EfuseType	is the type of the eFUSE in which these rows reside.

Returns

XST_SUCCESS if keys are not programmed. Errorcode on failure.

XilSKey_ZynqMp_EfusePs_WritePufHelprData

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

***Note*:** To generate PufSyndromeData please use XilSKey_Puf_Registration API

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_WritePufHelprData(const XilSKey_Puf
*InstancePtr);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_WritePufHelprData` function arguments.

Table 297: **XilSKey_ZynqMp_EfusePs_WritePufHelprData Arguments**

Type	Name	Description
const XilSKey_Puf *	InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

- XST_SUCCESS if programs successfully.
- Errorcode on failure

XilSKey_ZynqMp_EfusePs_ReadPufHelprData

This function reads the PUF helper data from eFUSE.

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

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_ReadPufHelprData(u32 *Address);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadPufHelprData` function arguments.

Table 298: **XilSKey_ZynqMp_EfusePs_ReadPufHelprData Arguments**

Type	Name	Description
u32 *	Address	Pointer to data array which holds the PUF helper data read from eFUSEs.

Returns

- XST_SUCCESS if reads successfully.
- Errorcode on failure.

XilSKey_ZynqMp_EfusePs_WritePufChash

This function programs eFUSE with CHash value.

***Note*:** To generate the CHash value, please use XilSKey_Puf_Registration function.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_WritePufChash(const XilSKey_Puf *InstancePtr);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_WritePufChash` function arguments.

Table 299: **XilSKey_ZynqMp_EfusePs_WritePufChash Arguments**

Type	Name	Description
const XilSKey_Puf *	InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

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

XilSKey_ZynqMp_EfusePs_ReadPufChash

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

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

Prototype

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

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadPufChash` function arguments.

Table 300: **XilSKey_ZynqMp_EfusePs_ReadPufChash Arguments**

Type	Name	Description
u32 *	Address	Pointer which holds the read back value of the chash.
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 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

XilSKey_ZynqMp_EfusePs_WritePufAux

This function programs eFUSE PUF auxiliary data.

***Note*:** To generate auxiliary data, please use XilSKey_Puf_Registration function.

Prototype

```
u32 XilSKey_ZynqMp_EfusePs_WritePufAux(const XilSKey_Puf *InstancePtr);
```

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_WritePufAux` function arguments.

Table 301: **XilSKey_ZynqMp_EfusePs_WritePufAux Arguments**

Type	Name	Description
const XilSKey_Puf *	InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

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

XilSKey_ZynqMp_EfusePs_ReadPufAux

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

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

Prototype

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

Parameters

The following table lists the `XilSKey_ZynqMp_EfusePs_ReadPufAux` function arguments.

Table 302: **XilSKey_ZynqMp_EfusePs_ReadPufAux Arguments**

Type	Name	Description
u32 *	Address	Pointer which holds the read back value of PUF's auxiliary data.
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array

Returns

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

XilSKey_Write_Puf_EfusePs_SecureBits

This function programs the eFUSE PUF secure bits.

Prototype

```
u32 XilSKey_Write_Puf_EfusePs_SecureBits(const XilSKey_Puf_Secure
*WriteSecureBits);
```

Parameters

The following table lists the `XilSKey_Write_Puf_EfusePs_SecureBits` function arguments.

Table 303: **XilSKey_Write_Puf_EfusePs_SecureBits Arguments**

Type	Name	Description
const XilSKey_Puf_Secure *	WriteSecureBits	Pointer to the XilSKey_Puf_Secure structure

Returns

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

XilSKey_Read_Puf_EfusePs_SecureBits

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

Prototype

```
u32 XilSKey_Read_Puf_EfusePs_SecureBits(XilSKey_Puf_Secure *SecureBitsRead,
u8 ReadOption);
```

Parameters

The following table lists the `XilSKey_Read_Puf_EfusePs_SecureBits` function arguments.

Table 304: **XilSKey_Read_Puf_EfusePs_SecureBits Arguments**

Type	Name	Description
Commented parameter SecureBits does not exist in function XilSKey_Read_Puf_EfusePs_SecureBits.	SecureBits	Pointer to the XilSKey_Puf_Secure structure which holds the read eFUSE secure bits from the PUF.
u8	ReadOption	Indicates whether or not to read from the actual eFUSE array or from the eFUSE cache. • 0(XSK_EFUSEPS_READ_FROM_CACHE) Reads from cache • 1(XSK_EFUSEPS_READ_FROM_EFUSE) Reads from eFUSE array

Returns

- XST_SUCCESS if reads successfully.
- Errorcode on failure.

XilSKey_Puf_Registration

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

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

Prototype

```
u32 XilSKey_Puf_Registration(XilSKey_Puf *InstancePtr);
```

Parameters

The following table lists the `XilSKey_Puf_Registration` function arguments.

Table 305: **XilSKey_Puf_Registration Arguments**

Type	Name	Description
XilSKey_Puf *	InstancePtr	Pointer to the XilSKey_Puf instance.

Returns

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

XilSKey_Puf_Regeneration

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

Prototype

```
u32 XilSKey_Puf_Regeneration(const XilSKey_Puf *InstancePtr);
```

Parameters

The following table lists the `XilSKey_Puf_Regeneration` function arguments.

Table 306: **XilSKey_Puf_Regeneration Arguments**

Type	Name	Description
const XilSKey_Puf *	InstancePtr	is a pointer to the XilSKey_Puf instance.

Returns

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

eFUSE PL API

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

Example Usage

- The Zynq eFUSE PL and UltraScale example application should contain the xilskey_efuse_example.c and the xilskey_input.h files.
- By default, both the eFUSE PS and PL are enabled in the application. You can comment 'XSK_EFUSEPL_DRIVER' to execute only the PS.
- For UltraScale, it is mandatory to comment `XSK_EFUSEPS_DRIVER else the example will generate an error.

- For more details on the user configurable parameters, refer `Zynq User-Configurable PL eFUSE Parameters` and `UltraScale or UltraScale+ User-Configurable PL eFUSE Parameters`.
- Requires hardware setup to program PL eFUSE of Zynq or UltraScale.

***Note*:** Contains the function prototypes, defines and macros for the PL eFUSE functionality.

MODIFICATION HISTORY: Ver Who Date Changes 1.00a rpoolla 04/26/13 First release 1.02a hk 10/28/13 Added API's to read status bits and key : u32 `XilSKey_EfusePl_ReadKey(XilSKey_EPl *InstancePtr)` u32 `XilSKey_EfusePl_ReadKey(XilSKey_EPl *InstancePtr)` 2.00 hk 22/01/14 Corrected PL voltage checks to VCCINT and VCCAUX. CR#768077 3.00 vns 31/07/15 Added efuse functionality for Ultrascale. 6.0 vns 07/07/16 Added Hardware module pins in eFUSE PL instance. 6.4 vns 02/27/18 Added support for programming secure bit 6 - enable obfuscation feature for eFUSE AES key vns 03/09/18 Added correct status bit positions to Ultrascale plus 6.6 vns 06/06/18 Added doxygen tags 6.7 arc 01/05/19 Fixed MISRA-C violations. psl 03/20/19 Added eFuse key write support for SSIT devices. psl 03/29/19 Added Support for user configurable GPIO for jtag control

Table 307: **Quick Function Reference**

Type	Name	Arguments
u32	XilSKey_EfusePl_SystemInit	`XilSKey_EPl` * InstancePtr
u32	XilSKey_EfusePl_Program	InstancePtr
u32	XilSKey_EfusePl_ReadStatus	`XilSKey_EPl` * InstancePtr u32 * StatusBits
u32	XilSKey_EfusePl_ReadKey	`XilSKey_EPl` * InstancePtr

Functions

XilSKey_EfusePl_SystemInit

Initializes PL eFUSE with input data given.

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

Prototype

```
u32 XilSKey_EfusePl_SystemInit(XilSKey_EPl *InstancePtr);
```

Parameters

The following table lists the `XilSKey_EfusePl_SystemInit` function arguments.

Table 308: **XilSKey_EfusePl_SystemInit Arguments**

Type	Name	Description
`XilSKey_EPl` *	InstancePtr	- Input data to be written to PL eFUSE

Returns

XilSKey_EfusePl_Program

Programs PL eFUSE with input data given through InstancePtr.

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

Prototype

```
u32 XilSKey_EfusePl_Program(XilSKey_EPl *PlInstancePtr);
```

Parameters

The following table lists the `XilSKey_EfusePl_Program` function arguments.

Table 309: **XilSKey_EfusePl_Program Arguments**

Type	Name	Description
Commented parameter InstancePtr does not exist in function XilSKey_EfusePl_Program.	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

XilSKey_EfusePl_ReadStatus

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

Prototype

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

Parameters

The following table lists the `XilSKey_EfusePl_ReadStatus` function arguments.

Table 310: **XilSKey_EfusePl_ReadStatus Arguments**

Type	Name	Description
`XilSKey_EPl` *	InstancePtr	Pointer to PL eFUSE instance.
u32 *	StatusBits	Buffer to store the status bits read.

Returns

XilSKey_EfusePl_ReadKey

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

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

Prototype

```
u32 XilSKey_EfusePl_ReadKey(XilSKey_EPl *InstancePtr);
```

Parameters

The following table lists the `XilSKey_EfusePl_ReadKey` function arguments.

Table 311: **XilSKey_EfusePl_ReadKey Arguments**

Type	Name	Description
`XilSKey_EPl` *	InstancePtr	Pointer to PL eFUSE instance.

Returns

Enumerations

Enumeration XSKEfusePl_FuseStatusBits_F8Series

Table 312: **Enumeration XSKEfusePl_FuseStatusBits_F8Series Values**

Value	Description
XSK_EFUSEPL_STATUS_DISABLE_KEY_READ_ULTRA	Bit 0 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_USER_KEY_READ_ULTRA	Bit 1 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_SECURE_READ_ULTRA	Bit 2 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_CNTRL_WRITE_ULTRA	Bit 5 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_RSA_KEY_READ_ULTRA	Bit 6 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_KEY_WRITE_ULTRA	Bit 7 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_USER_KEY_WRITE_ULTRA	Bit 8 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_SECURE_WRITE_ULTRA	Bit 9 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_RSA_KEY_WRITE_ULTRA	Bit 15 of Status reg.
XSK_EFUSEPL_STATUS_DIABLE_128BIT_USER_KEY_WRITE_ULTRA	
XSK_EFUSEPL_STATUS_AES_ONLY_ENABLED_ULTRA	Bit 25 of Status reg.
XSK_EFUSEPL_STATUS_RSA_AUTH_ENABLED_ULTRA	Bit 27 of Status reg.
XSK_EFUSEPL_STATUS_SECURITY_ENABLE_ULTRA	Bit 29 of Status reg.
XSK_EFUSEPL_STATUS_DISABLE_DCRPTR_ULTRA	Bit 30 of Status reg.
XSK_EFUSEPL_STATUS_ENABLE_OBFUSCATED_EFUSE_KEY	Bit 31 of Status reg.

Definitions

Define XSK_EFUSEPL_AES_KEY_SIZE_IN_BYTES

Definition

```
#define XSK_EFUSEPL_AES_KEY_SIZE_IN_BYTES(32U)
```

Description

AES Key size in Bytes.

Define XSK_EFUSEPL_USER_KEY_SIZE_IN_BYTES

Definition

```
#define XSK_EFUSEPL_USER_KEY_SIZE_IN_BYTES(4U)
```

Description

User Key size in Bytes.

Define XSK_EFUSEPL_CRC_FOR_AES_ZEROS

Definition

```
#define XSK_EFUSEPL_CRC_FOR_AES_ZEROS(0x621C42AAU)
```

Description

CRC of AES key with all zeros.

Define XSK_EFUSEPL_CRC_FOR_AES_ZEROS_ULTRA_PLUS

Definition

```
#define XSK_EFUSEPL_CRC_FOR_AES_ZEROS_ULTRA_PLUS(0x3117503AU)
```

Description

Define XSK_EFUSEPL_AES_KEY_STRING_LEN

Definition

```
#define XSK_EFUSEPL_AES_KEY_STRING_LEN(64U)
```

Description

AES key String length.

CRC Calculation API

This chapter provides a linked summary and detailed descriptions of the CRC calculation APIs.

For UltraScale and Zynq UltraScale+ MPSoC devices, the programmed AES cannot be read back. The programmed AES key can only be verified by reading the CRC value of AES key.

Table 313: **Quick Function Reference**

Type	Name	Arguments
u32	XilSKey_CrcCalculation	const u8 * Key
u32	XilSKey_CrcCalculation_AesKey	const u8 * Key

Functions

XilSKey_CrcCalculation

This function Calculates CRC value based on hexadecimal string passed.

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

Prototype

```
u32 XilSKey_CrcCalculation(const u8 *Key);
```

Parameters

The following table lists the `XilSKey_CrcCalculation` function arguments.

Table 314: **XilSKey_CrcCalculation Arguments**

Type	Name	Description
const u8 *	Key	Pointer to the string contains AES key in hexadecimal of length less than or equal to 64.

Returns

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

XilSkey_CrcCalculation_AesKey

Calculates CRC value of the provided key.

Key should be provided in hexa buffer.

Prototype

```
u32 XilSkey_CrcCalculation_AesKey(const u8 *Key);
```

Parameters

The following table lists the `XilSkey_CrcCalculation_AesKey` function arguments.

Table 315: **XilSkey_CrcCalculation_AesKey Arguments**

Type	Name	Description
const u8 *	Key	Pointer to an array of 32 bytes, which holds an AES key.

Returns

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

User-Configurable Parameters

This section provides detailed descriptions of the various user configurable 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.
XSK_EFUSEPS_RSA_KEY_HASH_VALUE	Default = 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

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.
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.

MIO Pins for Zynq PL eFUSE JTAG Operations

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

You can change the listed pins at your discretion.

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

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

MUX Selection Pin for Zynq PL eFUSE JTAG Operations

The table below lists the MUX selection pin.

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

MUX Parameter for Zynq PL eFUSE JTAG Operations

The table below lists the MUX parameter.

Parameter Name	Description
XSK_EFUSEPL_MIO_MUX_SEL_DEFAULT_VAL	Default = LOW. 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.

Parameter Name	Description
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.

Zynq User-Configurable PL BBRAM Parameters

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)

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 or UltraScale+ User-Configurable BBRAM PL Parameters

Following parameters need to be configured.

Based on your inputs, BBRAM is programmed with the provided AES key.

AES Keys and Related Parameters

The following table shows AES key related parameters.

Parameter Name	Description
XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_ CONFIG_ORDER_0	Default = FALSE By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_ CONFIG_ORDER_0 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_0 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ ORDER_0 and DPA protection cannot be enabled.

Parameter Name	Description
XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1	Default = FALSE By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_ CONFIG_ORDER_1 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_1 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ ORDER_1 and DPA protection cannot be enabled.
XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_2	Default = FALSE By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_ CONFIG_ORDER_2 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_2 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ ORDER_2 and DPA protection cannot be enabled.
XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_3	Default = FALSE By default, XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR_ CONFIG_ORDER_3 is FALSE. BBRAM is programmed with a non-obfuscated key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_3 and DPA protection can be either in enabled/disabled state. TRUE programs the BBRAM with key provided in XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ ORDER_3 and DPA protection cannot be enabled.
XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_0	Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1 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_ SLR_CONFIG_ORDER_0 should have TRUE value.
XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ORDER_1	Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1 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_ SLR_CONFIG_ORDER_1 should have TRUE value.

Parameter Name	Description
XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ ORDER_2	Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1 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_ SLR_CONFIG_ORDER_2 should have TRUE value.
XSK_BBRAM_OBFUSCATED_KEY_SLR_CONFIG_ ORDER_3	Default = b1c276899d71fb4cdd4a0a7905ea46c2e11f9574d09c7ea23b70b67de713ccd1 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_ SLR_CONFIG_ORDER_3 should have TRUE value.
XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ ORDER_0	Default = FALSE TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_0
XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ ORDER_1	Default = FALSE TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_1
XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ ORDER_2	Default = FALSE TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_2
XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG_ ORDER_3	Default = FALSE TRUE will program BBRAM with AES key provided in XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_3

Parameter Name	Description
XSK_BBRAM_AES_KEY_SLR_CONFIG_ ORDER_0	Default = 0000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa 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 AES key, XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG _ORDER_0 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR _CONFIG_ORDER_0 should have FALSE value.
XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_1	Default = 0000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa 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 AES key, XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG _ORDER_1 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR _CONFIG_ORDER_1 should have FALSE value
XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_2	Default = 0000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa 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 AES key, XSK_BBRAM_PGM_AES_KEY_SLR_CONFIG _ORDER_2 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR _CONFIG_ORDER_2 should have FALSE value

Parameter Name	Description
XSK_BBRAM_AES_KEY_SLR_CONFIG_ORDER_3	Default = 00000000000000000524156a63950bcedafeadcdeabaadee34216615aaaabbaaa 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 AES key, XSK_BBRAM_PGM_AES_KEY_SLR _CONFIG_ORDER_3 should have TRUE value , and XSK_BBRAM_PGM_OBFUSCATED_KEY_SLR _CONFIG_ORDER_3 should have FALSE value
XSK_BBRAM_AES_KEY_SIZE_IN_BITS	Default= 256 Size of AES key must be 256 bits.

DPA Protection for BBRAM key

The following table shows DPA protection configurable parameter.

Parameter Name	Description
XSK_BBRAM_DPA_PROTECT_ENABLE	Default = FALSE 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 Device Used for Connecting PL Master JTAG Signals

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

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

GPIO Pins Used for PL Master JTAG Signals

In Ultrascale the following GPIO pins are used for connecting MASTER_JTAG pins to access BBRAM.

These can be changed depending on your hardware.The table below shows the GPIO pins used for PL MASTER JTAG signals.

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

GPIO Channels

The following table shows GPIO channel number.

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

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

UltraScale or UltraScale+ User-Configurable PL eFUSE Parameters

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.

Macro Name	Description
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.
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.
XSK_EFUSEPL_ENABLE_OBFUSCATION_ EFUSEAES	Default = FALSE. TRUE will enable obfuscation feature for eFUSE AES key.

GPIO Device Used for Connecting PL Master JTAG Signals

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

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

GPIO Pins Used for PL Master JTAG and HWM Signals

In Ultrascale the following GPIO pins are used for connecting MASTER_JTAG pins to access eFUSE.

These can be changed depending on your hardware.The table below shows the GPIO pins used for PL MASTER JTAG signals.

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

GPIO Channels

The following table shows GPIO channel number.

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

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

SLR Selection to Program eFUSE on MONO/SSIT Devices

The following table shows parameters for programming different SLRs.

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

eFUSE PL Read Parameters

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

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

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

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

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

AES Keys and Related Parameters

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

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

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

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

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 should have the TRUE value.
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 should have the TRUE value.
2. Enable AES key programming:
   - XSK_EFUSEPL_PROGRAM_AES_KEY should have the TRUE value.
3. Provide key to be programmed on SLR:
   - XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_0 should have key to be programmed on SLR config order 0's eFUSE in the string format.
   - XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_3 should have key to be programmed on SLR config order 3's eFUSE in the string format.

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

Parameter Name	Description
XSK_EFUSEPL_PROGRAM_AES_KEY	Default = FALSE TRUE will burn the AES key provided in: XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_ INDEX if corresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_ INDEX is TRUE and FALSE will ignore the values given.
XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_0	Default = 0000000000000000000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 0 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key. ***Note***: For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _0 are enabled with the TRUE value.
XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_1	Default = 0000000000000000000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 1 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key. ***Note***: For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _1 are enabled with the TRUE value.

Parameter Name	Description
XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_2	Default = 0000000000000000000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 2 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key. ***Note*:** For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _2 are enabled with TRUE value.
XSK_EFUSEPL_AES_KEY_CONFIG_ORDER_3	Default = 0000000000000000000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR 3 when write API is used. This value should be the AES key given in string format. It should be 64 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn AES Key. ***Note*:** For writing the AES Key, make sure XSK_EFUSEPL_PROGRAM_AES_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _3 are enabled with TRUE value.

USER Keys (32-bit) and Related Parameters

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

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

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 should have the TRUE value.
2. Enable USER key programming:
   - XSK_EFUSEPL_PROGRAM_USER_KEY should have the TRUE value.
3. Provide key to be programmed on SLR:
   - XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_2 should have key to be programmed in the string format.

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

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

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

Parameter Name	Description
XSK_EFUSEPL_PROGRAM_USER_KEY	Default = FALSE TRUE will burn 32 bit User key given in XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_ INDEX if orresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_ INDEX is TRUE, FALSE will ignore the values given.
XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_0	Default = 00000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note***: For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_ 0 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_1	Default = 00000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array when the write API is used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note***: For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _1 are enabled with TRUE value.

Parameter Name	Description
XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_2	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*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _2 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_CONFIG_ORDER_3	Default = 00000000 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*:** For writing the User Key, XSK_EFUSEPL_PROGRAM_USER_KEY and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _3 are enabled with TRUE value.

RSA Hash and Related Parameters

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

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

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

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

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_0 should have the TRUE value.
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_3 should have the TRUE value.

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

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

Parameter Name	Description
XSK_EFUSEPL_PROGRAM_RSA_KEY_ HASH	Default = FALSE TRUE will burn RSA hash given in XSK_EFUSEPL_RSA_KEY_HASH_VALUE_CONFIG _ORDER_INDEX if corresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_ INDEX is TRUE, FALSE will ignore the values given.
XSK_EFUSEPL_RSA_KEY_HASH_VALUE_ CONFIG_ORDER_0	Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1/MONO when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value. ***Note***: For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _0 are enabled with TRUE value.
XSK_EFUSEPL_RSA_KEY_HASH_VALUE_ CONFIG_ORDER_1	Default = 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value. ***Note***: For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _1 are enabled with TRUE value.

Parameter Name	Description
XSK_EFUSEPL_RSA_KEY_HASH_VALUE_ CONFIG_ORDER_2	Default = 000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value. ***Note*:** For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _2 are enabled with TRUE value.
XSK_EFUSEPL_RSA_KEY_HASH_VALUE_ CONFIG_ORDER_3	Default = 000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000 The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR4 when write API used. This value should be the RSA Key hash given in string format. It should be 96 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn RSA hash value. ***Note*:** For writing the RSA hash, make sure XSK_EFUSEPL_PROGRAM_RSA_KEY_HASH and XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER _3 are enabled with TRUE value.

USER Keys (128-bit) and Related Parameters

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

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

1. Enable programming for SLR:
   - XSK_EFUSEPL_PGM_SLR_CONFIG_ORDER_2 should have the TRUE value.
2. Enable USER key programming:
   - XSK_EFUSEPL_PROGRAM_USER_KEY_128BIT should have the TRUE value.

3. Provide key to be programmed on SLR:
    - XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_2 , XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_2, XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_2, XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_2 should have value to be programmed in the string format. The key should be provided as below
        - XSK_EFUSEPL_USER_KEY_128BIT_0_CONFIG_ORDER_2 holds 31:0 bits,

XSK_EFUSEPL_USER_KEY_128BIT_1_CONFIG_ORDER_2 holds 63:32 bits,

- XSK_EFUSEPL_USER_KEY_128BIT_2_CONFIG_ORDER_2 holds 95:64 bits and
- XSK_EFUSEPL_USER_KEY_128BIT_3_CONFIG_ORDER_2 holds 127:96 bits of whole 128 bit User key. The following table shows USER key 128 bit and related parameters.

Parameter Name	Description
XSK_EFUSEPL_PROGRAM_USER_KEY_ 128BIT	Default = FALSE TRUE will burn 128 bit User key given in: XSK_EFUSEPL_USER_KEY_128BIT_0_ CONFIG_ORDER_INDEX, XSK_EFUSEPL_USER_KEY_128BIT_1 _CONFIG_ORDER_INDEX, XSK_EFUSEPL_USER_KEY_128BIT_2 _CONFIG_ORDER_INDEX, XSK_EFUSEPL_USER_KEY_128BIT_3 _CONFIG_ORDER_INDEX if corresponding XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_INDEX is TRUE, FALSE will ignore the values given.
XSK_EFUSEPL_USER_KEY_128BIT_0 _CONFIG_ORDER_0	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_ CONFIG_ORDER_0 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_0 are enabled with TRUE value.

Parameter Name	Description
XSK_EFUSEPL_USER_KEY_128BIT_1 _CONFIG_ORDER_0	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_ CONFIG_ORDER_0 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_0 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_2 _CONFIG_ORDER_0	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_ CONFIG_ORDER_0 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_0 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_3 _CONFIG_ORDER_0	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_ CONFIG_ORDER_0 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR0/MONO when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_0 are enabled with TRUE value.

Parameter Name	Description
XSK_EFUSEPL_USER_KEY_128BIT_0 _CONFIG_ORDER_1	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_ CONFIG_ORDER_1 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_1 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_1 _CONFIG_ORDER_1	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_ CONFIG_ORDER_1 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_1 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_2 _CONFIG_ORDER_1	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_ CONFIG_ORDER_1 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_1 are enabled with TRUE value.

Parameter Name	Description
XSK_EFUSEPL_USER_KEY_128BIT_3 _CONFIG_ORDER_1	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_ CONFIG_ORDER_1 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR1 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_1 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_0 _CONFIG_ORDER_2	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_ CONFIG_ORDER_2 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_2 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_1 _CONFIG_ORDER_2	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_ CONFIG_ORDER_2 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_2 are enabled with TRUE value.

Parameter Name	Description
XSK_EFUSEPL_USER_KEY_128BIT_2 _CONFIG_ORDER_2	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_ CONFIG_ORDER_2 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_2 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_3 _CONFIG_ORDER_2	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_ CONFIG_ORDER_2 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR2 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_2 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_0 _CONFIG_ORDER_3	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_0_ CONFIG_ORDER_3 holds 31:0 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_3 are enabled with TRUE value.

Parameter Name	Description
XSK_EFUSEPL_USER_KEY_128BIT_1 _CONFIG_ORDER_3	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_1_ CONFIG_ORDER_3 holds 63:32 bits, of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_3 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_2 _CONFIG_ORDER_3	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_2_ CONFIG_ORDER_3 holds 95:64 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_3 are enabled with TRUE value.
XSK_EFUSEPL_USER_KEY_128BIT_3 _CONFIG_ORDER_3	Default = 00000000 Provides 128-bit User key for XSK_EFUSEPL_USER_KEY_128BIT_3_ CONFIG_ORDER_3 holds 127:96 bits of whole 128 bit User key. The value mentioned in this will be converted to hex buffer and written into the PL eFUSE array of SLR3 when write API used. This value should be the User Key given in string format. It should be 8 characters long, valid characters are 0-9,a-f,A-F. Any other character is considered as invalid string and will not burn User Key. ***Note*:** For writing the User Key, make sure XSK_EFUSEPL_PROGRAM_USER_ KEY_128BIT and XSK_EFUSEPL_PGM_SLR_CONFIG_ ORDER_3 are enabled with TRUE value.

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

Enabling XSK_EFUSEPL_PROGRAM_USER_KEY will enable programming of USER key for both SLR1 And SLR2 as programming is enabled for both the SLR.

- Enabling XSK_EFUSEPL_PROGRAM_AES_KEY will enable programming of AES key for both SLR1 And SLR2 as programming is enabled for both the SLR.
- If you want to program USER key only for SLR1, then provided USER key will be programmed for SLR1 and Default key (all zeroes) will be programmed for SLR2.
- If you want to program AES key only for SLR2, then provided AES key will be programmed for SLR2 and Default key will be programmed for SLR1.

  To avoid all the above mentioned scenarios, if programming is required for different key on different SLR, separate runs should be done.

AES key CRC verification

You cannot read the AES key.

You can verify only by providing the CRC of the expected AES key. The following lists the parameters that may help you in verifying the AES key:

Parameter Name	Description
XSK_EFUSEPL_CHECK_AES_KEY_ CRC	Default = FALSE 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_EP1` instance parameter AESKeyMatched with either TRUE or FALSE. FALSE CRC check of FUSE_AES will not be performed.

Parameter Name	Description
XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_ CONFIG_ORDER_0	Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 0 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API. For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_ FOR_AES_ZEROS). For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_ AES_ZEROS_ULTRA_PLUS)
XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_ CONFIG_ORDER_1	Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 1 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API. For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_ FOR_AES_ZEROS). For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_ AES_ZEROS_ULTRA_PLUS)

Parameter Name	Description
XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_ CONFIG_ORDER_2	Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 2 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API. For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_ FOR_AES_ZEROS). For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_ AES_ZEROS_ULTRA_PLUS)
XSK_EFUSEPL_CRC_OF_EXPECTED_AES_KEY_ CONFIG_ORDER_3	Default = XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS CRC value of FUSE_AES with all Zeros. Expected FUSE_AES key's CRC value of SLR config order 3 has to be updated in place of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS. For Checking CRC of FUSE_AES XSK_EFUSEPL_CHECK_AES_KEY_ULTRA macro should be TRUE otherwise CRC check will not be performed. For calculation of AES key's CRC one can use u32 XilSKey_CrcCalculation(u8_Key) API. For UltraScale, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ ZEROS is 0x621C42AA(XSK_EFUSEPL_CRC_ FOR_AES_ZEROS). For UltraScale+, the value of XSK_EFUSEPL_AES_CRC_OF_ALL_ZEROS is 0x3117503A(XSK_EFUSEPL_CRC_FOR_ AES_ZEROS_ULTRA_PLUS)

Zynq UltraScale+ MPSoC User-Configurable PS eFUSE Parameters

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.

Macro Name	Description
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.
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.

Macro Name	Description
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.
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.

AES Keys and Related Parameters

The following table shows AES key related parameters.

Parameter Name	Description
XSK_EFUSEPS_WRITE_AES_KEY	Default = FALSE TRUE will burn the AES key provided in XSK_EFUSEPS_AES_KEY. FALSE will ignore the key provide XSK_EFUSEPS_AES_KEY.

Parameter Name	Description
XSK_EFUSEPS_AES_KEY	Default = 0000000000000000000000000000000000000000000000000000000000000000 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.
XSK_EFUSEPS_CHECK_AES_KEY_CRC	Default value is FALSE. TRUE will check the CRC provided in XSK_EFUSEPS_AES_KEY. CRC verification is done after programming AES key to verify the key is programmed properly or not, if not library error outs the same. So While programming AES key it is not necessary to verify the AES key again. ***Note*:** Please make sure if intention is to check only CRC of the provided key and not programming AES key then do not modify XSK_EFUSEPS_WRITE_AES_ KEY (TRUE will Program key).

User Keys and Related Parameters

Single bit programming is allowed for all the user eFUSEs.

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

Parameter Name	Description
XSK_EFUSEPS_WRITE_USER0_FUSE	Default = FALSE TRUE will burn User0 Fuse provided in XSK_EFUSEPS_USER0_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER0_FUSES
XSK_EFUSEPS_WRITE_USER1_FUSE	Default = FALSE TRUE will burn User1 Fuse provided in XSK_EFUSEPS_USER1_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER1_FUSES
XSK_EFUSEPS_WRITE_USER2_FUSE	Default = FALSE TRUE will burn User2 Fuse provided in XSK_EFUSEPS_USER2_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER2_FUSES
XSK_EFUSEPS_WRITE_USER3_FUSE	Default = FALSE TRUE will burn User3 Fuse provided in XSK_EFUSEPS_USER3_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER3_FUSES

Parameter Name	Description
XSK_EFUSEPS_WRITE_USER4_FUSE	Default = FALSE TRUE will burn User4 Fuse provided in XSK_EFUSEPS_USER4_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER4_FUSES
XSK_EFUSEPS_WRITE_USER5_FUSE	Default = FALSE TRUE will burn User5 Fuse provided in XSK_EFUSEPS_USER5_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER5_FUSES
XSK_EFUSEPS_WRITE_USER6_FUSE	Default = FALSE TRUE will burn User6 Fuse provided in XSK_EFUSEPS_USER6_FUSES. FALSE will ignore the value provided in XSK_EFUSEPS_USER6_FUSES
XSK_EFUSEPS_WRITE_USER7_FUSE	Default = FALSE 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
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

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

PPK1 Keys and Related Parameters

The following table shows the PPK1 keys and related parameters.

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

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

SPK ID and Related Parameters

The following table shows the SPK ID and related parameters.

Parameter Name	Description
XSK_EFUSEPS_WRITE_SPKID	Default = FALSE TRUE will burn SPKID provided in XSK_EFUSEPS_SPK_ID. FALSE will ignore the hash provided in XSK_EFUSEPS_SPK_ID.
XSK_EFUSEPS_SPK_ID	Default = 00000000 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 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000 AES key (in HEX) that must be programmed into BBRAM.

Parameter Name	Description
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_MANUFACTURER	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.
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

Macro Name	Description
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 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000 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.
XSK_PUF_BLACK_KEY_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.

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

Enumerations

Enumeration XSKEfusePl_ErrorCodes

Table 316: **Enumeration XSKEfusePl_ErrorCodes Values**

Value	Description
XSK_EFUSEPL_ERROR_NONE	0 No error.
XSK_EFUSEPL_ERROR_ROW_NOT_ZERO	0x10 Row is not zero.
XSK_EFUSEPL_ERROR_READ_ROW_OUT_OF_RANGE	0x11 Read Row is out of range.
XSK_EFUSEPL_ERROR_READ_MARGIN_OUT_OF_RANGE	0x12 Read Margin is out of range.
XSK_EFUSEPL_ERROR_READ_BUFFER_NULL	0x13 No buffer for read.
XSK_EFUSEPL_ERROR_READ_BIT_VALUE_NOT_SET	0x14 Read bit not set.
XSK_EFUSEPL_ERROR_READ_BIT_OUT_OF_RANGE	0x15 Read bit is out of range.
XSK_EFUSEPL_ERROR_READ_TMEPERATURE_OUT_OF_RANGE	0x16 Temperature obtained from XADC is out of range to read.
XSK_EFUSEPL_ERROR_READ_VCCAUX_VOLTAGE_OUT_OF_RANGE	0x17 VCCAUX obtained from XADC is out of range to read.
XSK_EFUSEPL_ERROR_READ_VCCINT_VOLTAGE_OUT_OF_RANGE	0x18 VCCINT obtained from XADC is out of range to read.
XSK_EFUSEPL_ERROR_WRITE_ROW_OUT_OF_RANGE	0x19 To write row is out of range.
XSK_EFUSEPL_ERROR_WRITE_BIT_OUT_OF_RANGE	0x1A To read bit is out of range.
XSK_EFUSEPL_ERROR_WRITE_TMEPERATURE_OUT_OF_RANGE	0x1B To eFUSE write Temperature obtained from XADC is outof range.
XSK_EFUSEPL_ERROR_WRITE_VCCAUX_VOLTAGE_OUT_OF_RANGE	0x1C To write eFUSE VCCAUX obtained from XADC is out of range.
XSK_EFUSEPL_ERROR_WRITE_VCCINT_VOLTAGE_OUT_OF_RANGE	0x1D To write into eFUSE VCCINT obtained from XADC is out of range.
XSK_EFUSEPL_ERROR_FUSE_CNTRL_WRITE_DISABLED	0x1E Fuse control write is disabled.
XSK_EFUSEPL_ERROR_CNTRL_WRITE_BUFFER_NULL	0x1F Buffer pointer that is supposed to contain control data is null.

Table 316: **Enumeration XSKEfusePl_ErrorCodes Values** *(cont'd)*

Value	Description
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.

Table 316: **Enumeration XSKEfusePl_ErrorCodes Values** *(cont'd)*

Value	Description
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

Enumerations

Enumeration XSKEfusePs_ErrorCodes

Table 317: **Enumeration XSKEfusePs_ErrorCodes Values**

Value	Description
XSK_EFUSEPS_ERROR_NONE	0 No error.
XSK_EFUSEPS_ERROR_ADDRESS_XIL_RESTRICTED	0x01 Address is restricted.
XSK_EFUSEPS_ERROR_READ_TMEPERATURE_OUT_OF_RANGE	0x02 Temperature obtained from XADC is out of range.
XSK_EFUSEPS_ERROR_READ_VCCPAUX_VOLTAGE_OUT_OF_RANGE	0x03 VCCAUX obtained from XADC is out of range.
XSK_EFUSEPS_ERROR_READ_VCCPINT_VOLTAGE_OUT_OF_RANGE	0x04 VCCINT obtained from XADC is out of range.
XSK_EFUSEPS_ERROR_WRITE_TEMPERATURE_OUT_OF_RANGE	0x05 Temperature obtained from XADC is out of range.
XSK_EFUSEPS_ERROR_WRITE_VCCPAUX_VOLTAGE_OUT_OF_RANGE	0x06 VCCAUX obtained from XADC is out of range.
XSK_EFUSEPS_ERROR_WRITE_VCCPINT_VOLTAGE_OUT_OF_RANGE	0x07 VCCINT obtained from XADC is out of range.
XSK_EFUSEPS_ERROR_VERIFICATION	0x08 Verification error.
XSK_EFUSEPS_ERROR_RSA_HASH_ALREADY_PROGRAMMED	0x09 RSA hash was already programmed.
XSK_EFUSEPS_ERROR_CONTROLLER_MODE	0x0A Controller mode error
XSK_EFUSEPS_ERROR_REF_CLOCK	0x0B Reference clock not between 20 to 60MHz
XSK_EFUSEPS_ERROR_READ_MODE	0x0C Not supported read mode
XSK_EFUSEPS_ERROR_XADC_CONFIG	0x0D XADC configuration error.
XSK_EFUSEPS_ERROR_XADC_INITIALIZE	0x0E XADC initialization error.
XSK_EFUSEPS_ERROR_XADC_SELF_TEST	0x0F XADC self-test failed.
XSK_EFUSEPS_ERROR_PARAMETER_NULL	0x10 Passed parameter null.
XSK_EFUSEPS_ERROR_STRING_INVALID	0x20 Passed string is invalid.

Table 317: **Enumeration XSKEfusePs_ErrorCodes Values** *(cont'd)*

Value	Description
XSK_EFUSEPS_ERROR_AES_ALREADY_PROGRAMMED	0x12 AES key is already programmed.
XSK_EFUSEPS_ERROR_SPKID_ALREADY_PROGRAMMED	0x13 SPK ID is already programmed.
XSK_EFUSEPS_ERROR_PPK0_HASH_ALREADY_PROGRAMMED	0x14 PPK0 hash is already programmed.
XSK_EFUSEPS_ERROR_PPK1_HASH_ALREADY_PROGRAMMED	0x15 PPK1 hash is already programmed.
XSK_EFUSEPS_ERROR_IN_TBIT_PATTERN	0x16 Error in TBITS pattern .
XSK_EFUSEPS_ERROR_PROGRAMMING	0x00A0 Error in programming eFUSE.
XSK_EFUSEPS_ERROR_PGM_NOT_DONE	0x00A1 Program not done
XSK_EFUSEPS_ERROR_READ	0x00B0 Error in reading.
XSK_EFUSEPS_ERROR_BYTES_REQUEST	0x00C0 Error in requested byte count.
XSK_EFUSEPS_ERROR_RESRVD_BITS_PRGRMG	0x00D0 Error in programming reserved bits.
XSK_EFUSEPS_ERROR_ADDR_ACCESS	0x00E0 Error in accessing requested address.
XSK_EFUSEPS_ERROR_READ_NOT_DONE	0x00F0 Read not done
XSK_EFUSEPS_ERROR_PS_STRUCT_NULL	0x8100 PS structure pointer is null.
XSK_EFUSEPS_ERROR_XADC_INIT	0x8200 XADC initialization error.
XSK_EFUSEPS_ERROR_CONTROLLER_LOCK	0x8300 PS eFUSE controller is locked.
XSK_EFUSEPS_ERROR_EFUSE_WRITE_PROTECTED	0x8400 PS eFUSE is write protected.
XSK_EFUSEPS_ERROR_CONTROLLER_CONFIG	0x8500 Controller configuration error.
XSK_EFUSEPS_ERROR_PS_PARAMETER_WRONG	0x8600 PS eFUSE parameter is not TRUE/FALSE.
XSK_EFUSEPS_ERROR_WRITE_128K_CRC_BIT	0x9100 Error in enabling 128K CRC.
XSK_EFUSEPS_ERROR_WRITE_NONSECURE_INITB_BIT	0x9200 Error in programming NON secure bit.

Table 317: **Enumeration XSKEfusePs_ErrorCodes Values** *(cont'd)*

Value	Description
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.

Table 317: **Enumeration XSKEfusePs_ErrorCodes Values** *(cont'd)*

Value	Description
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.

Table 317: **Enumeration XSKEfusePs_ErrorCodes Values** *(cont'd)*

Value	Description
XSK_EFUSEPS_ERROR_WRTIE_PPK0_WR_LK_BIT	0xD400 Error in programming PPK0 write lock bit.
XSK_EFUSEPS_ERROR_WRTIE_PPK0_RVK_BIT	0xD500 Error in programming PPK0 revoke bit.
XSK_EFUSEPS_ERROR_WRTIE_PPK1_WR_LK_BIT	0xD600 Error in programming PPK1 write lock bit.
XSK_EFUSEPS_ERROR_WRTIE_PPK1_RVK_BIT	0xD700 Error in programming PPK0 revoke bit.
XSK_EFUSEPS_ERROR_WRITE_PUF_SYN_INVLD	0xD800 Error while programming the PUF syndrome invalidate bit.
XSK_EFUSEPS_ERROR_WRITE_PUF_SYN_WRLK	0xD900 Error while programming Syndrome write lock bit.
XSK_EFUSEPS_ERROR_WRITE_PUF_SYN_REG_DIS	0xDA00 Error while programming PUF syndrome register disable bit.
XSK_EFUSEPS_ERROR_WRITE_PUF_RESERVED_BIT	0xDB00 Error while programming PUF reserved bit.
XSK_EFUSEPS_ERROR_WRITE_LBIST_EN_BIT	0xDC00 Error while programming LBIST enable bit.
XSK_EFUSEPS_ERROR_WRITE_LPD_SC_EN_BIT	0xDD00 Error while programming LPD SC enable bit.
XSK_EFUSEPS_ERROR_WRITE_FPD_SC_EN_BIT	0xDE00 Error while programming FPD SC enable bit.
XSK_EFUSEPS_ERROR_WRITE_PBR_BOOT_ERR_BIT	0xDF00 Error while programming PBR boot error bit.
XSK_EFUSEPS_ERROR_PUF_INVALID_REG_MODE	0xE000 Error when PUF registration is requested with invalid registration mode.
XSK_EFUSEPS_ERROR_PUF_REG_WO_AUTH	0xE100 Error when write not allowed without authentication enabled.
XSK_EFUSEPS_ERROR_PUF_REG_DISABLED	0xE200 Error when trying to do PUF registration and when PUF registration is disabled.
XSK_EFUSEPS_ERROR_PUF_INVALID_REQUEST	0xE300 Error when an invalid mode is requested.
XSK_EFUSEPS_ERROR_PUF_DATA_ALREADY_PROGRAMMED	0xE400 Error when PUF is already programmed in eFUSE.
XSK_EFUSEPS_ERROR_PUF_DATA_OVERFLOW	0xE500 Error when an over flow occurs.
XSK_EFUSEPS_ERROR_SPKID_BIT_CANT_REVERT	0xE600 Already programmed SPKID bit cannot be reverted
XSK_EFUSEPS_ERROR_PUF_DATA_UNDERFLOW	0xE700 Error when an under flow occurs.

Table 317: **Enumeration XSKEfusePs_ErrorCodes Values** *(cont'd)*

Value	Description
XSK_EFUSEPS_ERROR_PUF_TIMEOUT	0xE800 Error when an PUF generation timedout.
XSK_EFUSEPS_ERROR_PUF_ACCESS	0xE900 Error when an PUF Access violation.
XSK_EFUSEPS_ERROR_PUF_CHASH_ALREADY_PROGRAMMED	0xEA00 Error When PUF Chash already programmed in eFuse.
XSK_EFUSEPS_ERROR_PUF_AUX_ALREADY_PROGRAMMED	0xEB00 Error When PUF AUX already programmed in eFuse.
XSK_EFUSEPS_ERROR_CMPLTD_EFUSE_PRGRM_WITH_ERR	0x10000 eFUSE programming is completed with temp and vol read errors.
XSK_EFUSEPS_ERROR_CACHE_LOAD	0x20000U Error in re-loading CACHE.
XSK_EFUSEPS_RD_FROM_EFUSE_NOT_ALLOWED	0x30000U Read from eFuse is not allowed.
XSK_EFUSEPS_ERROR_FUSE_PROTECTED	0x00080000 Requested eFUSE is write protected.
XSK_EFUSEPS_ERROR_USER_BIT_CANT_REVERT	0x00800000 Already programmed user FUSE bit cannot be reverted.
XSK_EFUSEPS_ERROR_BEFORE_PROGRAMMING	0x08000000U Error occurred before programming.

Zynq UltraScale+ MPSoC BBRAM PS Error Codes

Enumerations

Enumeration XskZynqMp_Ps_Bbram_ErrorCodes

Table 318: **Enumeration XskZynqMp_Ps_Bbram_ErrorCodes Values**

Value	Description
XSK_ZYNQMP_BBRAMPS_ERROR_NONE	0 No error.
XSK_ZYNQMP_BBRAMPS_ERROR_IN_PRGRMG_ENABLE	0x010 If this error is occurred programming is not possible.
XSK_ZYNQMP_BBRAMPS_ERROR_IN_ZEROISE	0x20 zeroize bbram is failed.
XSK_ZYNQMP_BBRAMPS_ERROR_IN_CRC_CHECK	0xB000 If this error is occurred programming is done but CRC check is failed.
XSK_ZYNQMP_BBRAMPS_ERROR_IN_PRGRMG	0xC000 programming of key is failed.
XSK_ZYNQMP_BBRAMPS_ERROR_IN_WRITE_CRC	0xE800 error write CRC value.

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.

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.

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.

Procedures

This section 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 platform 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 platform project.
2. Map all the sections 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 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.

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
3. Update memory has to be done using the tcl command updatemem.
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
3. Update memory has to be done using the tcl command updatemem:
4. Program the board using `$Final.bit bitstream`.
5. Output can be seen in UART terminal.

Data Structure Index

The following is a list of data structures:

XilSKey_EPl

XSK_EfusePl is the PL eFUSE driver instance.

Using this structure, user can define the eFUSE bits to be blown.

Declaration

```
typedef struct
{
  u32 ForcePowerCycle,
  u32 KeyWrite,
  u32 AESKeyRead,
  u32 UserKeyRead,
  u32 CtrlWrite,
  u32 RSARead,
  u32 UserKeyWrite,
  u32 SecureWrite,
  u32 RSAWrite,
  u32 User128BitWrite,
  u32 SecureRead,
  u32 AESKeyExclusive,
  u32 JtagDisable,
  u32 UseAESOnly,
  u32 EncryptOnly,
  u32 IntTestAccessDisable,
  u32 DecoderDisable,
  u32 RSAEnable,
  u32 FuseObfusEn,
  u32 ProgAESandUserLowKey,
  u32 ProgUserHighKey,
  u32 ProgAESKeyUltra,
  u32 ProgUserKeyUltra,
  u32 ProgRSAKeyUltra,
  u32 ProgUser128BitUltra,
  u32 CheckAESKeyUltra,
  u32 ReadUserKeyUltra,
  u32 ReadRSAKeyUltra,
  u32 ReadUser128BitUltra,
  u8 AESKey[XSK_EFUSEPL_AES_KEY_SIZE_IN_BYTES],
  u8 UserKey[XSK_EFUSEPL_USER_KEY_SIZE_IN_BYTES],
  u8 RSAKeyHash[XSK_EFUSEPL_RSA_KEY_HASH_SIZE_IN_BYTES],
  u8 User128Bit[XSK_EFUSEPL_128BIT_USERKEY_SIZE_IN_BYTES],
  u32 JtagMioTDI,
  u32 JtagMioTDO,
  u32 JtagMioTCK,
  u32 JtagMioTMS,
  u32 JtagMioMuxSel,
  u32 JtagMuxSelLineDefVal,
  u32 JtagGpioID,
  u32 HwmGpioStart,
  u32 HwmGpioReady,
  u32 HwmGpioEnd,
  u32 JtagGpioTDI,
  u32 JtagGpioTDO,
  u32 JtagGpioTMS,
  u32 JtagGpioTCK,
  u32 GpioInputCh,
```

```
  u32 GpioOutPutCh,
  u8 AESKeyReadback[XSK_EFUSEPL_AES_KEY_SIZE_IN_BYTES],
  u8 UserKeyReadback[XSK_EFUSEPL_USER_KEY_SIZE_IN_BYTES],
  u32 CrcOfAESKey,
  u8 AESKeyMatched,
  u8 RSAHashReadback[XSK_EFUSEPL_RSA_KEY_HASH_SIZE_IN_BYTES],
  u8 User128BitReadBack[XSK_EFUSEPL_128BIT_USERKEY_SIZE_IN_BYTES],
  u32 SystemInitDone,
  XSKEfusePl_Fpga FpgaFlag,
  u32 CrcToVerify,
  u32 NumSlr,
  u32 MasterSlr,
  u32 SlrConfigOrderIndex
} XilSKey_EPl;
```

Table 319: **Structure XilSKey_EPl member description**

Member	Description
ForcePowerCycle	Following are the FUSE CNTRL bits[1:5, 8-10]. If XTRUE then part has to be power cycled to be able to be reconfigured only for zynq
KeyWrite	If XTRUE will disable eFUSE write to FUSE_AES and FUSE_USER blocks valid only for zynq but in ultrascale If XTRUE will disable eFUSE write to FUSE_AESKEY block in Ultrascale.
AESKeyRead	If XTRUE will disable eFUSE read to FUSE_AES block and also disables eFUSE write to FUSE_AES and FUSE_USER blocks in Zynq Pl.but in Ultrascale if XTRUE will disable eFUSE read to FUSE_KEY block and also disables eFUSE write to FUSE_KEY blocks.
UserKeyRead	If XTRUE will disable eFUSE read to FUSE_USER block and also disables eFUSE write to FUSE_AES and FUSE_USER blocks in zynq but in ultrascale if XTRUE will disable eFUSE read to FUSE_USER block and also disables eFUSE write to FUSE_USER blocks.
CtrlWrite	If XTRUE will disable eFUSE write to FUSE_CNTRL block in both Zynq and Ultrascale.
RSARead	If XTRUE will disable eFuse read to FUSE_RSA block and also disables eFuse write to FUSE_RSA block in Ultrascale.
UserKeyWrite	If XTRUE will disable eFUSE write to FUSE_USER block in Ultrascale.
SecureWrite	If XTRUE will disable eFUSE write to FUSE_SEC block in Ultrascale.
RSAWrite	If XTRUE will disable eFUSE write to FUSE_RSA block in Ultrascale.
User128BitWrite	If TRUE will disable eFUSE write to 128BIT FUSE_USER block in Ultrascale.
SecureRead	IF XTRUE will disable eFuse read to FUSE_SEC block and also disables eFuse write to FUSE_SEC block in Ultrascale.
AESKeyExclusive	If XTRUE will force eFUSE key to be used if booting Secure Image In Zynq.
JtagDisable	If XTRUE then permanently sets the Zynq ARM DAP controller in bypass mode in both zynq and ultrascale.
UseAESOnly	If XTRUE will force to use Secure boot with eFUSE key only for both Zynq and Ultrascale.
EncryptOnly	If XTRUE will only allow encrypted bitstreams only.
IntTestAccessDisable	If XTRUE then sets the disable's Xilinx internal test access in Ultrascale.
DecoderDisable	If XTRUE then permanently disables the decryptor in Ultrascale.
RSAEnable	Enable RSA authentication in ultrascale.
FuseObfusEn	Enable Obfuscated feature for decryption of eFUSE AES.

Table 319: **Structure XilSKey_EPl member description** *(cont'd)*

Member	Description
ProgAESandUserLowKey	Following is the define to select if the user wants to select AES key and User Low Key for Zynq.
ProgUserHighKey	Following is the define to select if the user wants to select User Low Key for Zynq.
ProgAESKeyUltra	Following is the define to select if the user wants to select User key for Ultrascale.
ProgUserKeyUltra	Following is the define to select if the user wants to select User key for Ultrascale.
ProgRSAKeyUltra	Following is the define to select if the user wants to select RSA key for Ultrascale.
ProgUser128BitUltra	Following is the define to select if the user wants to program 128 bit User key for Ultrascale.
CheckAESKeyUltra	Following is the define to select if the user wants to read AES key for Ultrascale.
ReadUserKeyUltra	Following is the define to select if the user wants to read User key for Ultrascale.
ReadRSAKeyUltra	Following is the define to select if the user wants to read RSA key for Ultrascale.
ReadUser128BitUltra	Following is the define to select if the user wants to read 128 bit User key for Ultrascale.
AESKey	This is the REF_CLK value in Hz. This is for the aes_key value
UserKey	This is for the user_key value.
RSAKeyHash	This is for the rsa_key value for Ultrascale.
User128Bit	This is for the User 128 bit key value for Ultrascale.
JtagMioTDI	TDI MIO Pin Number for ZYNQ.
JtagMioTDO	TDO MIO Pin Number for ZYNQ.
JtagMioTCK	TCK MIO Pin Number for ZYNQ.
JtagMioTMS	TMS MIO Pin Number for ZYNQ.
JtagMioMuxSel	MUX Selection MIO Pin Number for ZYNQ.
JtagMuxSelLineDefVal	Value on the MUX Selection line for ZYNQ.
JtagGpioID	GPIO device ID.
HwmGpioStart	Hardware module Start signal's GPIO pin number.
HwmGpioReady	Hardware module Ready signal's GPIO pin number.
HwmGpioEnd	Hardware module End signal's GPIO pin number.
JtagGpioTDI	TDI AXI GPIO pin number for Ultrascale.
JtagGpioTDO	TDO AXI GPIO pin number for Ultrascale.
JtagGpioTMS	TMS AXI GPIO pin number for Ultrascale.
JtagGpioTCK	TCK AXI GPIO pin number for Ultrascale.
GpioInputCh	AXI GPIO Channel number of all Inputs TDO.
GpioOutPutCh	AXI GPIO Channel number for all Outputs TDI/TMS/TCK.
AESKeyReadback	AES key read only for Zynq.
UserKeyReadback	User key read in Ultrascale and Zynq.

Table 319: **Structure XilSKey_EPl member description** *(cont'd)*

Member	Description
CrcOfAESKey	Expected AES key's CRC for Ultrascale here we can't read AES key directly.
AESKeyMatched	Flag is True is AES's CRC is matched, otherwise False.
RSAHashReadback	RSA key read back for Ultrascale.
User128BitReadBack	User 128 bit key read back for Ultrascale.
SystemInitDone	Internal variable to check if timer, XADC and JTAG are initialized.
FpgaFlag	Stores Fpga series of Efuse.
CrcToVerify	CRC of AES key to verify programmed AES key.
NumSlr	Number of SLRs to iterate through.
MasterSlr	Current SLR to iterate through.
SlrConfigOrderIndex	Master SLR.

XilPM Library v3.2

XilPM Zynq UltraScale+ MPSoC APIs

Xilinx Power Management (XilPM) provides Embedded Energy Management Interface (EEMI) APIs for power management on Zynq UltraScale+ MPSoC. For more details about EEMI, see the Embedded Energy Management Interface (EEMI) API User Guide (UG1200).

Table 320: **Quick Function Reference**

Type	Name	Arguments
XStatus	XPm_InitXilpm	XIpiPsu * IpiInst
enum XPmBootStatus	XPm_GetBootStatus	void
void	XPm_SuspendFinalize	void
XStatus	pm_ipi_send	struct XPm_Master *const master u32 payload
XStatus	pm_ipi_buff_read32	struct XPm_Master *const master u32 * value1 u32 * value2 u32 * value3
XStatus	XPm_SelfSuspend	const enum XPmNodeId nid const u32 latency const u8 state const u64 address
XStatus	XPm_SetConfiguration	const u32 address
XStatus	XPm_InitFinalize	void

Table 320: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
XStatus	XPm_RequestSuspend	const enum XPmNodeId target const enum XPmRequestAck ack const u32 latency const u8 state
XStatus	XPm_RequestWakeUp	const enum XPmNodeId target const bool setAddress const u64 address const enum XPmRequestAck ack
XStatus	XPm_ForcePowerDown	const enum XPmNodeId target const enum XPmRequestAck ack
XStatus	XPm_AbortSuspend	const enum XPmAbortReason reason
XStatus	XPm_SetWakeUpSource	const enum XPmNodeId target const enum XPmNodeId wkup_node const u8 enable
XStatus	XPm_SystemShutdown	restart
XStatus	XPm_RequestNode	const enum XPmNodeId node const u32 capabilities const u32 qos const enum XPmRequestAck ack
XStatus	XPm_SetRequirement	const enum XPmNodeId nid const u32 capabilities const u32 qos const enum XPmRequestAck ack
XStatus	XPm_ReleaseNode	const enum XPmNodeId node
XStatus	XPm_SetMaxLatency	const enum XPmNodeId node const u32 latency
void	XPm_InitSuspendCb	const enum XPmSuspendReason reason const u32 latency const u32 state const u32 timeout

Table 320: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	XPm_AcknowledgeCb	const enum XPmNodeId node const XStatus status const u32 oppoint
void	XPm_NotifyCb	const enum XPmNodeId node const enum XPmNotifyEvent event const u32 oppoint
XStatus	XPm_GetApiVersion	u32 * version
XStatus	XPm_GetNodeStatus	const enum XPmNodeId node XPm_NodeStatus *const nodestatus
XStatus	XPm_GetOpCharacteristic	const enum XPmNodeId node const enum XPmOpCharType type u32 *const result
XStatus	XPm_ResetAssert	const enum XPmReset reset assert
XStatus	XPm_ResetGetStatus	const enum XPmReset reset u32 * status
XStatus	XPm_RegisterNotifier	XPm_Notifier *const notifier
XStatus	XPm_UnregisterNotifier	XPm_Notifier *const notifier
XStatus	XPm_MmioWrite	const u32 address const u32 mask const u32 value
XStatus	XPm_MmioRead	const u32 address u32 *const value
XStatus	XPm_ClockEnable	const enum XPmClock clk
XStatus	XPm_ClockDisable	const enum XPmClock clk
XStatus	XPm_ClockGetStatus	const enum XPmClock clk u32 *const status

Table 320: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
XStatus	XPm_ClockSetOneDivider	const enum XPmClock clk const u32 divider const u32 divId
XStatus	XPm_ClockSetDivider	const enum XPmClock clk const u32 divider
XStatus	XPm_ClockGetOneDivider	const enum XPmClock clk u32 *const divider
XStatus	XPm_ClockGetDivider	const enum XPmClock clk u32 *const divider
XStatus	XPm_ClockSetParent	const enum XPmClock clk const enum XPmClock parent
XStatus	XPm_ClockGetParent	const enum XPmClock clk enum XPmClock *const parent
XStatus	XPm_ClockSetRate	const enum XPmClock clk const u32 rate
XStatus	XPm_ClockGetRate	const enum XPmClock clk u32 *const rate
XStatus	XPm_PllSetParameter	const enum XPmNodeId node const enum XPmPllParam parameter const u32 value
XStatus	XPm_PllGetParameter	const enum XPmNodeId node const enum XPmPllParam parameter u32 *const value
XStatus	XPm_PllSetMode	const enum XPmNodeId node const enum XPmPllMode mode
XStatus	XPm_PllGetMode	const enum XPmNodeId node enum XPmPllMode *const mode
XStatus	XPm_PinCtrlAction	const u32 pin

Table 320: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
XStatus	XPm_PinCtrlRequest	const u32 pin
XStatus	XPm_PinCtrlRelease	const u32 pin
XStatus	XPm_PinCtrlSetFunction	const u32 pin const enum XPmPinFn fn
XStatus	XPm_PinCtrlGetFunction	const u32 pin enum XPmPinFn *const fn
XStatus	XPm_PinCtrlSetParameter	const u32 pin const enum XPmPinParam param const u32 value
XStatus	XPm_PinCtrlGetParameter	const u32 pin const enum XPmPinParam param u32 *const value

Functions

XPm_InitXilpm

Initialize xilpm library.

Note: None

Prototype

```
XStatus XPm_InitXilpm(XIpiPsu *IpiInst);
```

Parameters

The following table lists the `XPm_InitXilpm` function arguments.

Table 321: **XPm_InitXilpm Arguments**

Type	Name	Description
XIpiPsu *	IpiInst	Pointer to IPI driver instance

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_GetBootStatus

This Function returns information about the boot reason. If the boot is not a system startup but a resume, power down request bitfield for this processor will be cleared.

Note: None

Prototype

```
enum
                XPmBootStatus
              XPm_GetBootStatus(void);
```

Returns

Returns processor boot status

- PM_RESUME : If the boot reason is because of system resume.
- PM_INITIAL_BOOT : If this boot is the initial system startup.

XPm_SuspendFinalize

This Function waits for PMU to finish all previous API requests sent by the PU and performs client specific actions to finish suspend procedure (e.g. execution of wfi instruction on A53 and R5 processors).

Note: This function should not return if the suspend procedure is successful.

Prototype

```
void XPm_SuspendFinalize(void);
```

Returns

pm_ipi_send

Sends IPI request to the PMU.

Note: None

Prototype

```
XStatus pm_ipi_send(struct XPm_Master *const master, u32
payload[PAYLOAD_ARG_CNT]);
```

Parameters

The following table lists the `pm_ipi_send` function arguments.

Table 322: **pm_ipi_send Arguments**

Type	Name	Description
struct `XPm_Master` *const	master	Pointer to the master who is initiating request
u32	payload	API id and call arguments to be written in IPI buffer

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

pm_ipi_buff_read32

Reads IPI response after PMU has handled interrupt.

***Note*:** None

Prototype

```
XStatus pm_ipi_buff_read32(struct XPm_Master *const master, u32 *value1,
u32 *value2, u32 *value3);
```

Parameters

The following table lists the `pm_ipi_buff_read32` function arguments.

Table 323: **pm_ipi_buff_read32 Arguments**

Type	Name	Description
struct `XPm_Master` *const	master	Pointer to the master who is waiting and reading response
u32 *	value1	Used to return value from 2nd IPI buffer element (optional)
u32 *	value2	Used to return value from 3rd IPI buffer element (optional)
u32 *	value3	Used to return value from 4th IPI buffer element (optional)

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SelfSuspend

This function is used by a CPU to declare that it is about to suspend itself. After the PMU processes this call it will wait for the requesting CPU to complete the suspend procedure and become ready to be put into a sleep state.

Note: This is a blocking call, it will return only once PMU has responded

Prototype

```
XStatus XPm_SelfSuspend(const enum XPmNodeId nid, const u32 latency, const
u8 state, const u64 address);
```

Parameters

The following table lists the `XPm_SelfSuspend` function arguments.

Table 324: **XPm_SelfSuspend Arguments**

Type	Name	Description
const enum `XPmNodeId`	nid	Node ID of the CPU node to be suspended.
const u32	latency	Maximum wake-up latency requirement in us(microsecs)
const u8	state	Instead of specifying a maximum latency, a CPU can also explicitly request a certain power state.
const u64	address	Address from which to resume when woken up.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SetConfiguration

This function is called to configure the power management framework. The call triggers power management controller to load the configuration object and configure itself according to the content of the object.

Note: The provided address must be in 32-bit address space which is accessible by the PMU.

Prototype

```
XStatus XPm_SetConfiguration(const u32 address);
```

Parameters

The following table lists the `XPm_SetConfiguration` function arguments.

Table 325: **XPm_SetConfiguration Arguments**

Type	Name	Description
const u32	address	Start address of the configuration object

Returns

XST_SUCCESS if successful, otherwise an error code

XPm_InitFinalize

This function is called to notify the power management controller about the completed power management initialization.

Note: It is assumed that all used nodes are requested when this call is made. The power management controller may power down the nodes which are not requested after this call is processed.

Prototype

```
XStatus XPm_InitFinalize(void);
```

Returns

XST_SUCCESS if successful, otherwise an error code

XPm_RequestSuspend

This function is used by a PU to request suspend of another PU. This call triggers the power management controller to notify the PU identified by 'nodeID' that a suspend has been requested. This will allow said PU to gracefully suspend itself by calling XPm_SelfSuspend for each of its CPU nodes, or else call XPm_AbortSuspend with its PU node as argument and specify the reason.

Note: If 'ack' is set to PM_ACK_NON_BLOCKING, the requesting PU will be notified upon completion of suspend or if an error occurred, such as an abort. REQUEST_ACK_BLOCKING is not supported for this command.

Prototype

```
XStatus XPm_RequestSuspend(const enum XPmNodeId target, const enum
XPmRequestAck ack, const u32 latency, const u8 state);
```

Parameters

The following table lists the `XPm_RequestSuspend` function arguments.

Table 326: **XPm_RequestSuspend Arguments**

Type	Name	Description
const enum XPmNodeId	target	Node ID of the PU node to be suspended
const enum XPmRequestAck	ack	Requested acknowledge type
const u32	latency	Maximum wake-up latency requirement in us(micro sec)
const u8	state	Instead of specifying a maximum latency, a PU can also explicitly request a certain power state.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_RequestWakeUp

This function can be used to request power up of a CPU node within the same PU, or to power up another PU.

***Note*:** If acknowledge is requested, the calling PU will be notified by the power management controller once the wake-up is completed.

Prototype

```
XStatus XPm_RequestWakeUp(const enum XPmNodeId target, const bool
setAddress, const u64 address, const enum XPmRequestAck ack);
```

Parameters

The following table lists the `XPm_RequestWakeUp` function arguments.

Table 327: **XPm_RequestWakeUp Arguments**

Type	Name	Description
const enum XPmNodeId	target	Node ID of the CPU or PU to be powered/woken up.
const bool	setAddress	Specifies whether the start address argument is being passed. • 0 : do not set start address • 1 : set start address
const u64	address	Address from which to resume when woken up. Will only be used if set_address is 1.
const enum XPmRequestAck	ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ForcePowerDown

One PU can request a forced poweroff of another PU or its power island or power domain. This can be used for killing an unresponsive PU, in which case all resources of that PU will be automatically released.

***Note*:** Force power down may not be requested by a PU for itself.

Prototype

```
XStatus XPm_ForcePowerDown(const enum XPmNodeId target, const enum
XPmRequestAck ack);
```

Parameters

The following table lists the `XPm_ForcePowerDown` function arguments.

Table 328: **XPm_ForcePowerDown Arguments**

Type	Name	Description
const enum `XPmNodeId`	target	Node ID of the PU node or power island/domain to be powered down.
const enum `XPmRequestAck`	ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_AbortSuspend

This function is called by a CPU after a XPm_SelfSuspend call to notify the power management controller that CPU has aborted suspend or in response to an init suspend request when the PU refuses to suspend.

***Note*:** Calling PU expects the PMU to abort the initiated suspend procedure. This is a non-blocking call without any acknowledge.

Prototype

```
XStatus XPm_AbortSuspend(const enum XPmAbortReason reason);
```

Parameters

The following table lists the `XPm_AbortSuspend` function arguments.

Table 329: **XPm_AbortSuspend Arguments**

Type	Name	Description
const enum XPmAbortReason	reason	Reason code why the suspend can not be performed or completed • ABORT_REASON_WKUP_EVENT : local wakeup-event received • ABORT_REASON_PU_BUSY : PU is busy • ABORT_REASON_NO_PWRDN : no external powerdown supported • ABORT_REASON_UNKNOWN : unknown error during suspend procedure

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SetWakeUpSource

This function is called by a PU to add or remove a wake-up source prior to going to suspend. The list of wake sources for a PU is automatically cleared whenever the PU is woken up or when one of its CPUs aborts the suspend procedure.

***Note*:** Declaring a node as a wakeup source will ensure that the node will not be powered off. It also will cause the PMU to configure the GIC Proxy accordingly if the FPD is powered off.

Prototype

```
XStatus XPm_SetWakeUpSource(const enum XPmNodeId target, const enum
XPmNodeId wkup_node, const u8 enable);
```

Parameters

The following table lists the `XPm_SetWakeUpSource` function arguments.

Table 330: **XPm_SetWakeUpSource Arguments**

Type	Name	Description
const enum XPmNodeId	target	Node ID of the target to be woken up.
const enum XPmNodeId	wkup_node	Node ID of the wakeup device.
const u8	enable	Enable flag: • 1 : the wakeup source is added to the list • 0 : the wakeup source is removed from the list

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SystemShutdown

This function can be used by a privileged PU to shut down or restart the complete device.

Note: In either case the PMU will call XPm_InitSuspendCb for each of the other PUs, allowing them to gracefully shut down. If a PU is asleep it will be woken up by the PMU. The PU making the XPm_SystemShutdown should perform its own suspend procedure after calling this API. It will not receive an init suspend callback.

Prototype

```
XStatus XPm_SystemShutdown(u32 type, u32 subtype);
```

Parameters

The following table lists the `XPm_SystemShutdown` function arguments.

Table 331: **XPm_SystemShutdown Arguments**

Type	Name	Description
Commented parameter restart does not exist in function XPm_SystemShutdown.	restart	Should the system be restarted automatically? • PM_SHUTDOWN : no restart requested, system will be powered off permanently • PM_RESTART : restart is requested, system will go through a full reset

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_RequestNode

Used to request the usage of a PM-slave. Using this API call a PU requests access to a slave device and asserts its requirements on that device. Provided the PU is sufficiently privileged, the PMU will enable access to the memory mapped region containing the control registers of that device. For devices that can only be serving a single PU, any other privileged PU will now be blocked from accessing this device until the node is released.

Note: None

Prototype

```
XStatus XPm_RequestNode(const enum XPmNodeId node, const u32 capabilities,
const u32 qos, const enum XPmRequestAck ack);
```

Parameters

The following table lists the `XPm_RequestNode` function arguments.

Table 332: **XPm_RequestNode Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	Node ID of the PM slave requested
const u32	capabilities	Slave-specific capabilities required, can be combined • PM_CAP_ACCESS : full access / functionality • PM_CAP_CONTEXT : preserve context • PM_CAP_WAKEUP : emit wake interrupts
const u32	qos	Quality of Service (0-100) required
const enum `XPmRequestAck`	ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SetRequirement

This function is used by a PU to announce a change in requirements for a specific slave node which is currently in use.

***Note*:** If this function is called after the last awake CPU within the PU calls SelfSuspend, the requirement change shall be performed after the CPU signals the end of suspend to the power management controller, (e.g. WFI interrupt).

Prototype

```
XStatus XPm_SetRequirement(const enum XPmNodeId nid, const u32
capabilities, const u32 qos, const enum XPmRequestAck ack);
```

Parameters

The following table lists the `XPm_SetRequirement` function arguments.

Table 333: **XPm_SetRequirement Arguments**

Type	Name	Description
const enum XPmNodeId	nid	Node ID of the PM slave.
const u32	capabilities	Slave-specific capabilities required.
const u32	qos	Quality of Service (0-100) required.
const enum XPmRequestAck	ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ReleaseNode

This function is used by a PU to release the usage of a PM slave. This will tell the power management controller that the node is no longer needed by that PU, potentially allowing the node to be placed into an inactive state.

Note: None

Prototype

```
XStatus XPm_ReleaseNode(const enum XPmNodeId node);
```

Parameters

The following table lists the `XPm_ReleaseNode` function arguments.

Table 334: **XPm_ReleaseNode Arguments**

Type	Name	Description
const enum XPmNodeId	node	Node ID of the PM slave.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SetMaxLatency

This function is used by a PU to announce a change in the maximum wake-up latency requirements for a specific slave node currently used by that PU.

Note: Setting maximum wake-up latency can constrain the set of possible power states a resource can be put into.

Prototype

```
XStatus XPm_SetMaxLatency(const enum XPmNodeId node, const u32 latency);
```

Parameters

The following table lists the `XPm_SetMaxLatency` function arguments.

Table 335: **XPm_SetMaxLatency Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	Node ID of the PM slave.
const u32	latency	Maximum wake-up latency required.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_InitSuspendCb

Callback function to be implemented in each PU, allowing the power management controller to request that the PU suspend itself.

Note: If the PU fails to act on this request the power management controller or the requesting PU may choose to employ the forceful power down option.

Prototype

```
void XPm_InitSuspendCb(const enum XPmSuspendReason reason, const u32
latency, const u32 state, const u32 timeout);
```

Parameters

The following table lists the `XPm_InitSuspendCb` function arguments.

Table 336: **XPm_InitSuspendCb Arguments**

Type	Name	Description
const enum `XPmSuspendReason`	reason	Suspend reason: • SUSPEND_REASON_PU_REQ : Request by another PU • SUSPEND_REASON_ALERT : Unrecoverable SysMon alert • SUSPEND_REASON_SHUTDOWN : System shutdown • SUSPEND_REASON_RESTART : System restart
const u32	latency	Maximum wake-up latency in us(micro secs). This information can be used by the PU to decide what level of context saving may be required.

Table 336: **XPm_InitSuspendCb Arguments** *(cont'd)*

Type	Name	Description
const u32	state	Targeted sleep/suspend state.
const u32	timeout	Timeout in ms, specifying how much time a PU has to initiate its suspend procedure before it's being considered unresponsive.

Returns

None

XPm_AcknowledgeCb

This function is called by the power management controller in response to any request where an acknowledge callback was requested, i.e. where the 'ack' argument passed by the PU was REQUEST_ACK_NON_BLOCKING.

Note: None

Prototype

```
void XPm_AcknowledgeCb(const enum XPmNodeId node, const XStatus status,
const u32 oppoint);
```

Parameters

The following table lists the `XPm_AcknowledgeCb` function arguments.

Table 337: **XPm_AcknowledgeCb Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	ID of the component or sub-system in question.
const XStatus	status	Status of the operation: • OK: the operation completed successfully • ERR: the requested operation failed
const u32	oppoint	Operating point of the node in question

Returns

None

XPm_NotifyCb

This function is called by the power management controller if an event the PU was registered for has occurred. It will populate the notifier data structure passed when calling XPm_RegisterNotifier.

Note: None

Prototype

```
void XPm_NotifyCb(const enum XPmNodeId node, const enum XPmNotifyEvent
event, const u32 oppoint);
```

Parameters

The following table lists the `XPm_NotifyCb` function arguments.

Table 338: **XPm_NotifyCb Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	ID of the node the event notification is related to.
const enum `XPmNotifyEvent`	event	ID of the event
const u32	oppoint	Current operating state of the node.

Returns

None

XPm_GetApiVersion

This function is used to request the version number of the API running on the power management controller.

Note: None

Prototype

```
XStatus XPm_GetApiVersion(u32 *version);
```

Parameters

The following table lists the `XPm_GetApiVersion` function arguments.

Table 339: **XPm_GetApiVersion Arguments**

Type	Name	Description
u32 *	version	Returns the API 32-bit version number. Returns 0 if no PM firmware present.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_GetNodeStatus

This function is used to obtain information about the current state of a component. The caller must pass a pointer to an `XPm_NodeStatus` structure, which must be pre-allocated by the caller.

- status - The current power state of the requested node.
  - For CPU nodes:
    - 0 : if CPU is off (powered down),
    - 1 : if CPU is active (powered up),
    - 2 : if CPU is in sleep (powered down),
    - 3 : if CPU is suspending (powered up)
  - For power islands and power domains:
    - 0 : if island is powered down,
    - 1 : if island is powered up
  - For PM slaves:
    - 0 : if slave is powered down,
    - 1 : if slave is powered up,
    - 2 : if slave is in retention
- requirement - Slave nodes only: Returns current requirements the requesting PU has requested of the node.
- usage - Slave nodes only: Returns current usage status of the node:
  - 0 : node is not used by any PU,
  - 1 : node is used by caller exclusively,
  - 2 : node is used by other PU(s) only,
  - 3 : node is used by caller and by other PU(s)

Note: None

Prototype

```
XStatus XPm_GetNodeStatus(const enum XPmNodeId node, XPm_NodeStatus *const
nodestatus);
```

Parameters

The following table lists the `XPm_GetNodeStatus` function arguments.

Table 340: **XPm_GetNodeStatus Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	ID of the component or sub-system in question.
`XPm_NodeStatus` *const	nodestatus	Used to return the complete status of the node.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_GetOpCharacteristic

Call this function to request the power management controller to return information about an operating characteristic of a component.

***Note*:** Power value is not actual power consumption of device. It is default dummy power value which is fixed in PMUFW. Temperature type is not supported for ZynqMP.

Prototype

```
XStatus XPm_GetOpCharacteristic(const enum XPmNodeId node, const enum
XPmOpCharType type, u32 *const result);
```

Parameters

The following table lists the `XPm_GetOpCharacteristic` function arguments.

Table 341: **XPm_GetOpCharacteristic Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	ID of the component or sub-system in question.
const enum `XPmOpCharType`	type	Type of operating characteristic requested: • power (current power consumption), • latency (current latency in micro seconds to return to active state), • temperature (current temperature),
u32 *const	result	Used to return the requested operating characteristic.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ResetAssert

This function is used to assert or release reset for a particular reset line. Alternatively a reset pulse can be requested as well.

Note: None

Prototype

```
XStatus XPm_ResetAssert(const enum XPmReset reset, const enum
XPmResetAction resetaction);
```

Parameters

The following table lists the `XPm_ResetAssert` function arguments.

Table 342: **XPm_ResetAssert Arguments**

Type	Name	Description
const enum `XPmReset`	reset	ID of the reset line
Commented parameter assert does not exist in function XPm_ResetAssert.	assert	Identifies action: • PM_RESET_ACTION_RELEASE : release reset, • PM_RESET_ACTION_ASSERT : assert reset, • PM_RESET_ACTION_PULSE : pulse reset,

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ResetGetStatus

Call this function to get the current status of the selected reset line.

Note: None

Prototype

```
XStatus XPm_ResetGetStatus(const enum XPmReset reset, u32 *status);
```

Parameters

The following table lists the `XPm_ResetGetStatus` function arguments.

Table 343: **XPm_ResetGetStatus Arguments**

Type	Name	Description
const enum XPmReset	reset	Reset line
u32 *	status	Status of specified reset (true - asserted, false - released)

Returns

Returns 1/XST_FAILURE for 'asserted' or 0/XST_SUCCESS for 'released'.

XPm_RegisterNotifier

A PU can call this function to request that the power management controller call its notify callback whenever a qualifying event occurs. One can request to be notified for a specific or any event related to a specific node.

- nodeID : ID of the node to be notified about,
- eventID : ID of the event in question, '-1' denotes all events (- EVENT_STATE_CHANGE, EVENT_ZERO_USERS),
- wake : true: wake up on event, false: do not wake up (only notify if awake), no buffering/ queueing
- callback : Pointer to the custom callback function to be called when the notification is available. The callback executes from interrupt context, so the user must take special care when implementing the callback. Callback is optional, may be set to NULL.
- received : Variable indicating how many times the notification has been received since the notifier is registered.

***Note*:** The caller shall initialize the notifier object before invoking the XPm_RegisteredNotifier function. While notifier is registered, the notifier object shall not be modified by the caller.

Prototype

```
XStatus XPm_RegisterNotifier(XPm_Notifier *const notifier);
```

Parameters

The following table lists the `XPm_RegisterNotifier` function arguments.

Table 344: **XPm_RegisterNotifier Arguments**

Type	Name	Description
XPm_Notifier *const	notifier	Pointer to the notifier object to be associated with the requested notification. The notifier object contains the following data related to the notification:

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_UnregisterNotifier

A PU calls this function to unregister for the previously requested notifications.

***Note*:** None

Prototype

```
XStatus XPm_UnregisterNotifier(XPm_Notifier *const notifier);
```

Parameters

The following table lists the `XPm_UnregisterNotifier` function arguments.

Table 345: **XPm_UnregisterNotifier Arguments**

Type	Name	Description
`XPm_Notifier` *const	notifier	Pointer to the notifier object associated with the previously requested notification

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_MmioWrite

Call this function to write a value directly into a register that isn't accessible directly, such as registers in the clock control unit. This call is bypassing the power management logic. The permitted addresses are subject to restrictions as defined in the PCW configuration.

***Note*:** If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_MmioWrite(const u32 address, const u32 mask, const u32 value);
```

Parameters

The following table lists the `XPm_MmioWrite` function arguments.

Table 346: **XPm_MmioWrite Arguments**

Type	Name	Description
const u32	address	Physical 32-bit address of memory mapped register to write to.
const u32	mask	32-bit value used to limit write to specific bits in the register.
const u32	value	Value to write to the register bits specified by the mask.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_MmioRead

Call this function to read a value from a register that isn't accessible directly. The permitted addresses are subject to restrictions as defined in the PCW configuration.

Note: If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_MmioRead(const u32 address, u32 *const value);
```

Parameters

The following table lists the `XPm_MmioRead` function arguments.

Table 347: **XPm_MmioRead Arguments**

Type	Name	Description
const u32	address	Physical 32-bit address of memory mapped register to read from.
u32 *const	value	Returns the 32-bit value read from the register

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockEnable

Call this function to enable (activate) a clock.

Note: If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_ClockEnable(const enum XPmClock clk);
```

Parameters

The following table lists the `XPm_ClockEnable` function arguments.

Table 348: **XPm_ClockEnable Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock to be enabled

Returns

Status of performing the operation as returned by the PMU-FW

XPm_ClockDisable

Call this function to disable (gate) a clock.

Note: If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_ClockDisable(const enum XPmClock clk);
```

Parameters

The following table lists the `XPm_ClockDisable` function arguments.

Table 349: **XPm_ClockDisable Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock to be disabled

Returns

Status of performing the operation as returned by the PMU-FW

XPm_ClockGetStatus

Call this function to get status of a clock gate state.

Prototype

```
XStatus XPm_ClockGetStatus(const enum XPmClock clk, u32 *const status);
```

Parameters

The following table lists the `XPm_ClockGetStatus` function arguments.

Table 350: **XPm_ClockGetStatus Arguments**

Type	Name	Description
const enum XPmClock	clk	Identifier of the target clock
u32 *const	status	Location to store clock gate state (1=enabled, 0=disabled)

Returns

Status of performing the operation as returned by the PMU-FW

XPm_ClockSetOneDivider

Call this function to set divider for a clock.

Note: If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_ClockSetOneDivider(const enum XPmClock clk, const u32 divider,
const u32 divId);
```

Parameters

The following table lists the `XPm_ClockSetOneDivider` function arguments.

Table 351: **XPm_ClockSetOneDivider Arguments**

Type	Name	Description
const enum XPmClock	clk	Identifier of the target clock
const u32	divider	Divider value to be set
const u32	divId	ID of the divider to be set

Returns

Status of performing the operation as returned by the PMU-FW

XPm_ClockSetDivider

Call this function to set divider for a clock.

Note: If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_ClockSetDivider(const enum XPmClock clk, const u32 divider);
```

Parameters

The following table lists the `XPm_ClockSetDivider` function arguments.

Table 352: **XPm_ClockSetDivider Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock
const u32	divider	Divider value to be set

Returns

XST_INVALID_PARAM or status of performing the operation as returned by the PMU-FW

XPm_ClockGetOneDivider

Local function to get one divider (DIV0 or DIV1) of a clock.

Prototype

```
XStatus XPm_ClockGetOneDivider(const enum XPmClock clk, u32 *const divider,
const u32 divId);
```

Parameters

The following table lists the `XPm_ClockGetOneDivider` function arguments.

Table 353: **XPm_ClockGetOneDivider Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock
u32 *const	divider	Location to store the divider value

Returns

Status of performing the operation as returned by the PMU-FW

XPm_ClockGetDivider

Call this function to get divider of a clock.

Prototype

```
XStatus XPm_ClockGetDivider(const enum XPmClock clk, u32 *const divider);
```

Parameters

The following table lists the `XPm_ClockGetDivider` function arguments.

Table 354: **XPm_ClockGetDivider Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock
u32 *const	divider	Location to store the divider value

Returns

XST_INVALID_PARAM or status of performing the operation as returned by the PMU-FW

XPm_ClockSetParent

Call this function to set parent for a clock.

Note: If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_ClockSetParent(const enum XPmClock clk, const enum XPmClock
parent);
```

Parameters

The following table lists the `XPm_ClockSetParent` function arguments.

Table 355: **XPm_ClockSetParent Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock
const enum `XPmClock`	parent	Identifier of the target parent clock

Returns

XST_INVALID_PARAM or status of performing the operation as returned by the PMU-FW.

XPm_ClockGetParent

Call this function to get parent of a clock.

Prototype

```
XStatus XPm_ClockGetParent(const enum XPmClock clk, enum XPmClock *const
parent);
```

Parameters

The following table lists the `XPm_ClockGetParent` function arguments.

Table 356: **XPm_ClockGetParent Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock
enum `XPmClock` *const	parent	Location to store clock parent ID

Returns

XST_INVALID_PARAM or status of performing the operation as returned by the PMU-FW.

XPm_ClockSetRate

Call this function to set rate of a clock.

Note: If the action isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_ClockSetRate(const enum XPmClock clk, const u32 rate);
```

Parameters

The following table lists the `XPm_ClockSetRate` function arguments.

Table 357: **XPm_ClockSetRate Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock
const u32	rate	Clock frequency (rate) to be set

Returns

Status of performing the operation as returned by the PMU-FW

XPm_ClockGetRate

Call this function to get rate of a clock.

Prototype

```
XStatus XPm_ClockGetRate(const enum XPmClock clk, u32 *const rate);
```

Parameters

The following table lists the `XPm_ClockGetRate` function arguments.

Table 358: **XPm_ClockGetRate Arguments**

Type	Name	Description
const enum `XPmClock`	clk	Identifier of the target clock
u32 *const	rate	Location where the rate should be stored

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PllSetParameter

Call this function to set a PLL parameter.

Note: If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_PllSetParameter(const enum XPmNodeId node, const enum
XPmPllParam parameter, const u32 value);
```

Parameters

The following table lists the `XPm_PllSetParameter` function arguments.

Table 359: **XPm_PllSetParameter Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	PLL node identifier
const enum XPmPllParam	parameter	PLL parameter identifier
const u32	value	Value of the PLL parameter

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PllGetParameter

Call this function to get a PLL parameter.

Prototype

```
XStatus XPm_PllGetParameter(const enum XPmNodeId node, const enum
XPmPllParam parameter, u32 *const value);
```

Parameters

The following table lists the `XPm_PllGetParameter` function arguments.

Table 360: **XPm_PllGetParameter Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	PLL node identifier
const enum XPmPllParam	parameter	PLL parameter identifier
u32 *const	value	Location to store value of the PLL parameter

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PllSetMode

Call this function to set a PLL mode.

***Note*:** If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_PllSetMode(const enum XPmNodeId node, const enum XPmPllMode
mode);
```

Parameters

The following table lists the `XPm_PllSetMode` function arguments.

Table 361: **XPm_PllSetMode Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	PLL node identifier
const enum XPmPllMode	mode	PLL mode to be set

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PllGetMode

Call this function to get a PLL mode.

Prototype

```
XStatus XPm_PllGetMode(const enum XPmNodeId node, enum XPmPllMode *const
mode);
```

Parameters

The following table lists the `XPm_PllGetMode` function arguments.

Table 362: **XPm_PllGetMode Arguments**

Type	Name	Description
const enum `XPmNodeId`	node	PLL node identifier
enum XPmPllMode *const	mode	Location to store the PLL mode

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PinCtrlAction

Locally used function to request or release a pin control.

Prototype

```
XStatus XPm_PinCtrlAction(const u32 pin, const enum XPmApiId api);
```

Parameters

The following table lists the `XPm_PinCtrlAction` function arguments.

Table 363: **XPm_PinCtrlAction Arguments**

Type	Name	Description
const u32	pin	PIN identifier (index from range 0-77) API identifier (request or release pin control)

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PinCtrlRequest

Call this function to request a pin control.

Prototype

```
XStatus XPm_PinCtrlRequest(const u32 pin);
```

Parameters

The following table lists the `XPm_PinCtrlRequest` function arguments.

Table 364: **XPm_PinCtrlRequest Arguments**

Type	Name	Description
const u32	pin	PIN identifier (index from range 0-77)

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PinCtrlRelease

Call this function to release a pin control.

Prototype

```
XStatus XPm_PinCtrlRelease(const u32 pin);
```

Parameters

The following table lists the `XPm_PinCtrlRelease` function arguments.

Table 365: **XPm_PinCtrlRelease Arguments**

Type	Name	Description
const u32	pin	PIN identifier (index from range 0-77)

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PinCtrlSetFunction

Call this function to set a pin function.

***Note*:** If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_PinCtrlSetFunction(const u32 pin, const enum XPmPinFn fn);
```

Parameters

The following table lists the `XPm_PinCtrlSetFunction` function arguments.

Table 366: **XPm_PinCtrlSetFunction Arguments**

Type	Name	Description
const u32	pin	Pin identifier
const enum XPmPinFn	fn	Pin function to be set

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PinCtrlGetFunction

Call this function to get currently configured pin function.

Prototype

```
XStatus XPm_PinCtrlGetFunction(const u32 pin, enum XPmPinFn *const fn);
```

Parameters

The following table lists the `XPm_PinCtrlGetFunction` function arguments.

Table 367: **XPm_PinCtrlGetFunction Arguments**

Type	Name	Description
const u32	pin	PLL node identifier
enum XPmPinFn *const	fn	Location to store the pin function

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PinCtrlSetParameter

Call this function to set a pin parameter.

***Note*:** If the access isn't permitted this function returns an error code.

Prototype

```
XStatus XPm_PinCtrlSetParameter(const u32 pin, const enum XPmPinParam
param, const u32 value);
```

Parameters

The following table lists the `XPm_PinCtrlSetParameter` function arguments.

Table 368: **XPm_PinCtrlSetParameter Arguments**

Type	Name	Description
const u32	pin	Pin identifier
const enum XPmPinParam	param	Pin parameter identifier
const u32	value	Value of the pin parameter to set

Returns

Status of performing the operation as returned by the PMU-FW

XPm_PinCtrlGetParameter

Call this function to get currently configured value of pin parameter.

Prototype

```
XStatus XPm_PinCtrlGetParameter(const u32 pin, const enum XPmPinParam
param, u32 *const value);
```

Parameters

The following table lists the `XPm_PinCtrlGetParameter` function arguments.

Table 369: **XPm_PinCtrlGetParameter Arguments**

Type	Name	Description
const u32	pin	Pin identifier
const enum XPmPinParam	param	Pin parameter identifier
u32 *const	value	Location to store value of the pin parameter

Returns

Status of performing the operation as returned by the PMU-FW

Enumerations

Enumeration XPmApiId

APIs for Miscellaneous functions, suspending of PUs, managing PM slaves and Direct control.

Table 370: **Enumeration XPmApiId Values**

Value	Description
PM_GET_API_VERSION	
PM_SET_CONFIGURATION	
PM_GET_NODE_STATUS	
PM_GET_OP_CHARACTERISTIC	
PM_REGISTER_NOTIFIER	
PM_REQUEST_SUSPEND	
PM_SELF_SUSPEND	
PM_FORCE_POWERDOWN	
PM_ABORT_SUSPEND	
PM_REQUEST_WAKEUP	
PM_SET_WAKEUP_SOURCE	
PM_SYSTEM_SHUTDOWN	
PM_REQUEST_NODE	
PM_RELEASE_NODE	
PM_SET_REQUIREMENT	
PM_SET_MAX_LATENCY	
PM_RESET_ASSERT	
PM_RESET_GET_STATUS	
PM_MMIO_WRITE	
PM_MMIO_READ	
PM_INIT_FINALIZE	
PM_FPGA_LOAD	
PM_FPGA_GET_STATUS	
PM_GET_CHIPID	
PM_SECURE_SHA	
PM_SECURE_RSA	
PM_PINCTRL_REQUEST	
PM_PINCTRL_RELEASE	
PM_PINCTRL_GET_FUNCTION	
PM_PINCTRL_SET_FUNCTION	
PM_PINCTRL_CONFIG_PARAM_GET	
PM_PINCTRL_CONFIG_PARAM_SET	
PM_IOCTL	

Table 370: **Enumeration XPmApiId Values** *(cont'd)*

Value	Description
PM_QUERY_DATA	
PM_CLOCK_ENABLE	
PM_CLOCK_DISABLE	
PM_CLOCK_GETSTATE	
PM_CLOCK_SETDIVIDER	
PM_CLOCK_GETDIVIDER	
PM_CLOCK_SETRATE	
PM_CLOCK_GETRATE	
PM_CLOCK_SETPARENT	
PM_CLOCK_GETPARENT	
PM_SECURE_IMAGE	
PM_FPGA_READ	
PM_SECURE_AES	
PM_PLL_SET_PARAMETER	
PM_PLL_GET_PARAMETER	
PM_PLL_SET_MODE	
PM_PLL_GET_MODE	
PM_REGISTER_ACCESS	
PM_EFUSE_ACCESS	
PM_API_MAX	

Enumeration XPmApiCbId

PM API Callback Id Enum

Table 371: **Enumeration XPmApiCbId Values**

Value	Description
PM_INIT_SUSPEND_CB	
PM_ACKNOWLEDGE_CB	
PM_NOTIFY_CB	
PM_NOTIFY_STL_NO_OP	

Enumeration XPmNodeId

PM Node ID Enum

Table 372: **Enumeration XPmNodeId Values**

Value	Description
NODE_UNKNOWN	
NODE_APU	
NODE_APU_0	
NODE_APU_1	
NODE_APU_2	
NODE_APU_3	
NODE_RPU	
NODE_RPU_0	
NODE_RPU_1	
NODE_PLD	
NODE_FPD	
NODE_OCM_BANK_0	
NODE_OCM_BANK_1	
NODE_OCM_BANK_2	
NODE_OCM_BANK_3	
NODE_TCM_0_A	
NODE_TCM_0_B	
NODE_TCM_1_A	
NODE_TCM_1_B	
NODE_L2	
NODE_GPU_PP_0	
NODE_GPU_PP_1	
NODE_USB_0	
NODE_USB_1	
NODE_TTC_0	
NODE_TTC_1	
NODE_TTC_2	
NODE_TTC_3	
NODE_SATA	
NODE_ETH_0	
NODE_ETH_1	
NODE_ETH_2	
NODE_ETH_3	
NODE_UART_0	
NODE_UART_1	
NODE_SPI_0	
NODE_SPI_1	
NODE_I2C_0	
NODE_I2C_1	

Table 372: **Enumeration XPmNodeId Values** *(cont'd)*

Value	Description
NODE_SD_0	
NODE_SD_1	
NODE_DP	
NODE_GDMA	
NODE_ADMA	
NODE_NAND	
NODE_QSPI	
NODE_GPIO	
NODE_CAN_0	
NODE_CAN_1	
NODE_EXTERN	
NODE_APLL	
NODE_VPLL	
NODE_DPLL	
NODE_RPLL	
NODE_IOPLL	
NODE_DDR	
NODE_IPI_APU	
NODE_IPI_RPU_0	
NODE_GPU	
NODE_PCIE	
NODE_PCAP	
NODE_RTC	
NODE_LPD	
NODE_VCU	
NODE_IPI_RPU_1	
NODE_IPI_PL_0	
NODE_IPI_PL_1	
NODE_IPI_PL_2	
NODE_IPI_PL_3	
NODE_PL	
NODE_ID_MAX	

Enumeration XPmRequestAck

PM Acknowledge Request Types

Table 373: **Enumeration XPmRequestAck Values**

Value	Description
REQUEST_ACK_NO	
REQUEST_ACK_BLOCKING	
REQUEST_ACK_NON_BLOCKING	
REQUEST_ACK_CB_CERROR	

Enumeration XPmAbortReason

PM Abort Reasons Enum

Table 374: **Enumeration XPmAbortReason Values**

Value	Description
ABORT_REASON_WKUP_EVENT	
ABORT_REASON_PU_BUSY	
ABORT_REASON_NO_PWRDN	
ABORT_REASON_UNKNOWN	
ABORT_REASON_WKUP_EVENT	
ABORT_REASON_PU_BUSY	
ABORT_REASON_NO_PWRDN	
ABORT_REASON_UNKNOWN	

Enumeration XPmSuspendReason

PM Suspend Reasons Enum

Table 375: **Enumeration XPmSuspendReason Values**

Value	Description
SUSPEND_REASON_PU_REQ	
SUSPEND_REASON_ALERT	
SUSPEND_REASON_SYS_SHUTDOWN	
SUSPEND_REASON_PU_REQ	
SUSPEND_REASON_ALERT	
SUSPEND_REASON_SYS_SHUTDOWN	

Enumeration XPmRamState

PM RAM States Enum

Table 376: **Enumeration XPmRamState Values**

Value	Description
PM_RAM_STATE_OFF	
PM_RAM_STATE_RETENTION	
PM_RAM_STATE_ON	

Enumeration XPmOpCharType

PM Operating Characteristic types Enum

Table 377: **Enumeration XPmOpCharType Values**

Value	Description
PM_OPCHAR_TYPE_POWER	
PM_OPCHAR_TYPE_TEMP	
PM_OPCHAR_TYPE_LATENCY	
PM_OPCHAR_TYPE_POWER	
PM_OPCHAR_TYPE_TEMP	
PM_OPCHAR_TYPE_LATENCY	

Enumeration XPmBootStatus

Boot Status Enum

Table 378: **Enumeration XPmBootStatus Values**

Value	Description
PM_INITIAL_BOOT	boot is a fresh system startup
PM_RESUME	boot is a resume
PM_BOOT_ERROR	error, boot cause cannot be identified
PM_INITIAL_BOOT	
PM_RESUME	
PM_BOOT_ERROR	

Enumeration XPmResetAction

PM Reset Action types

Table 379: **Enumeration XPmResetAction Values**

Value	Description
XILPM_RESET_ACTION_RELEASE	
XILPM_RESET_ACTION_ASSERT	

Table 379: **Enumeration XPmResetAction Values** *(cont'd)*

Value	Description
XILPM_RESET_ACTION_PULSE	

Enumeration XPmReset

PM Reset Line IDs

Table 380: **Enumeration XPmReset Values**

Value	Description
XILPM_RESET_PCIE_CFG	
XILPM_RESET_PCIE_BRIDGE	
XILPM_RESET_PCIE_CTRL	
XILPM_RESET_DP	
XILPM_RESET_SWDT_CRF	
XILPM_RESET_AFI_FM5	
XILPM_RESET_AFI_FM4	
XILPM_RESET_AFI_FM3	
XILPM_RESET_AFI_FM2	
XILPM_RESET_AFI_FM1	
XILPM_RESET_AFI_FM0	
XILPM_RESET_GDMA	
XILPM_RESET_GPU_PP1	
XILPM_RESET_GPU_PP0	
XILPM_RESET_GPU	
XILPM_RESET_GT	
XILPM_RESET_SATA	
XILPM_RESET_ACPU3_PWRON	
XILPM_RESET_ACPU2_PWRON	
XILPM_RESET_ACPU1_PWRON	
XILPM_RESET_ACPU0_PWRON	
XILPM_RESET_APU_L2	
XILPM_RESET_ACPU3	
XILPM_RESET_ACPU2	
XILPM_RESET_ACPU1	
XILPM_RESET_ACPU0	
XILPM_RESET_DDR	
XILPM_RESET_APM_FPD	
XILPM_RESET_SOFT	
XILPM_RESET_GEM0	

Table 380: **Enumeration XPmReset Values** *(cont'd)*

Value	Description
XILPM_RESET_GEM1	
XILPM_RESET_GEM2	
XILPM_RESET_GEM3	
XILPM_RESET_QSPI	
XILPM_RESET_UART0	
XILPM_RESET_UART1	
XILPM_RESET_SPI0	
XILPM_RESET_SPI1	
XILPM_RESET_SDIO0	
XILPM_RESET_SDIO1	
XILPM_RESET_CAN0	
XILPM_RESET_CAN1	
XILPM_RESET_I2C0	
XILPM_RESET_I2C1	
XILPM_RESET_TTC0	
XILPM_RESET_TTC1	
XILPM_RESET_TTC2	
XILPM_RESET_TTC3	
XILPM_RESET_SWDT_CRL	
XILPM_RESET_NAND	
XILPM_RESET_ADMA	
XILPM_RESET_GPIO	
XILPM_RESET_IOU_CC	
XILPM_RESET_TIMESTAMP	
XILPM_RESET_RPU_R50	
XILPM_RESET_RPU_R51	
XILPM_RESET_RPU_AMBA	
XILPM_RESET_OCM	
XILPM_RESET_RPU_PGE	
XILPM_RESET_USB0_CORERESET	
XILPM_RESET_USB1_CORERESET	
XILPM_RESET_USB0_HIBERRESET	
XILPM_RESET_USB1_HIBERRESET	
XILPM_RESET_USB0_APB	
XILPM_RESET_USB1_APB	
XILPM_RESET_IPI	
XILPM_RESET_APM_LPD	
XILPM_RESET_RTC	
XILPM_RESET_SYSMON	

Table 380: **Enumeration XPmReset Values** *(cont'd)*

Value	Description
XILPM_RESET_AFI_FM6	
XILPM_RESET_LPD_SWDT	
XILPM_RESET_FPD	
XILPM_RESET_RPU_DBG1	
XILPM_RESET_RPU_DBG0	
XILPM_RESET_DBG_LPD	
XILPM_RESET_DBG_FPD	
XILPM_RESET_APLL	
XILPM_RESET_DPLL	
XILPM_RESET_VPLL	
XILPM_RESET_IOPLL	
XILPM_RESET_RPLL	
XILPM_RESET_GPO3_PL_0	
XILPM_RESET_GPO3_PL_1	
XILPM_RESET_GPO3_PL_2	
XILPM_RESET_GPO3_PL_3	
XILPM_RESET_GPO3_PL_4	
XILPM_RESET_GPO3_PL_5	
XILPM_RESET_GPO3_PL_6	
XILPM_RESET_GPO3_PL_7	
XILPM_RESET_GPO3_PL_8	
XILPM_RESET_GPO3_PL_9	
XILPM_RESET_GPO3_PL_10	
XILPM_RESET_GPO3_PL_11	
XILPM_RESET_GPO3_PL_12	
XILPM_RESET_GPO3_PL_13	
XILPM_RESET_GPO3_PL_14	
XILPM_RESET_GPO3_PL_15	
XILPM_RESET_GPO3_PL_16	
XILPM_RESET_GPO3_PL_17	
XILPM_RESET_GPO3_PL_18	
XILPM_RESET_GPO3_PL_19	
XILPM_RESET_GPO3_PL_20	
XILPM_RESET_GPO3_PL_21	
XILPM_RESET_GPO3_PL_22	
XILPM_RESET_GPO3_PL_23	
XILPM_RESET_GPO3_PL_24	
XILPM_RESET_GPO3_PL_25	
XILPM_RESET_GPO3_PL_26	

Table 380: **Enumeration XPmReset Values** *(cont'd)*

Value	Description
XILPM_RESET_GPO3_PL_27	
XILPM_RESET_GPO3_PL_28	
XILPM_RESET_GPO3_PL_29	
XILPM_RESET_GPO3_PL_30	
XILPM_RESET_GPO3_PL_31	
XILPM_RESET_RPU_LS	
XILPM_RESET_PS_ONLY	
XILPM_RESET_PL	
XILPM_RESET_GPIO5_EMIO_92	
XILPM_RESET_GPIO5_EMIO_93	
XILPM_RESET_GPIO5_EMIO_94	
XILPM_RESET_GPIO5_EMIO_95	

Enumeration XPmNotifyEvent

PM Notify Events Enum

Table 381: **Enumeration XPmNotifyEvent Values**

Value	Description
EVENT_STATE_CHANGE	
EVENT_ZERO_USERS	
EVENT_STATE_CHANGE	
EVENT_ZERO_USERS	

Enumeration XPmClock

PM Clock IDs

Table 382: **Enumeration XPmClock Values**

Value	Description
PM_CLOCK_IOPLL	
PM_CLOCK_RPLL	
PM_CLOCK_APLL	
PM_CLOCK_DPLL	
PM_CLOCK_VPLL	
PM_CLOCK_IOPLL_TO_FPD	
PM_CLOCK_RPLL_TO_FPD	
PM_CLOCK_APLL_TO_LPD	

Table 382: **Enumeration XPmClock Values** *(cont'd)*

Value	Description
PM_CLOCK_DPLL_TO_LPD	
PM_CLOCK_VPLL_TO_LPD	
PM_CLOCK_ACPU	
PM_CLOCK_ACPU_HALF	
PM_CLOCK_DBG_FPD	
PM_CLOCK_DBG_LPD	
PM_CLOCK_DBG_TRACE	
PM_CLOCK_DBG_TSTMP	
PM_CLOCK_DP_VIDEO_REF	
PM_CLOCK_DP_AUDIO_REF	
PM_CLOCK_DP_STC_REF	
PM_CLOCK_GDMA_REF	
PM_CLOCK_DPDMA_REF	
PM_CLOCK_DDR_REF	
PM_CLOCK_SATA_REF	
PM_CLOCK_PCIE_REF	
PM_CLOCK_GPU_REF	
PM_CLOCK_GPU_PP0_REF	
PM_CLOCK_GPU_PP1_REF	
PM_CLOCK_TOPSW_MAIN	
PM_CLOCK_TOPSW_LSBUS	
PM_CLOCK_GTGREF0_REF	
PM_CLOCK_LPD_SWITCH	
PM_CLOCK_LPD_LSBUS	
PM_CLOCK_USB0_BUS_REF	
PM_CLOCK_USB1_BUS_REF	
PM_CLOCK_USB3_DUAL_REF	
PM_CLOCK_USB0	
PM_CLOCK_USB1	
PM_CLOCK_CPU_R5	
PM_CLOCK_CPU_R5_CORE	
PM_CLOCK_CSU_SPB	
PM_CLOCK_CSU_PLL	
PM_CLOCK_PCAP	
PM_CLOCK_IOU_SWITCH	
PM_CLOCK_GEM_TSU_REF	
PM_CLOCK_GEM_TSU	
PM_CLOCK_GEM0_TX	
PM_CLOCK_GEM1_TX	

Table 382: **Enumeration XPmClock Values** *(cont'd)*

Value	Description
PM_CLOCK_GEM2_TX	
PM_CLOCK_GEM3_TX	
PM_CLOCK_GEM0_RX	
PM_CLOCK_GEM1_RX	
PM_CLOCK_GEM2_RX	
PM_CLOCK_GEM3_RX	
PM_CLOCK_QSPI_REF	
PM_CLOCK_SDIO0_REF	
PM_CLOCK_SDIO1_REF	
PM_CLOCK_UART0_REF	
PM_CLOCK_UART1_REF	
PM_CLOCK_SPI0_REF	
PM_CLOCK_SPI1_REF	
PM_CLOCK_NAND_REF	
PM_CLOCK_I2C0_REF	
PM_CLOCK_I2C1_REF	
PM_CLOCK_CAN0_REF	
PM_CLOCK_CAN1_REF	
PM_CLOCK_CAN0	
PM_CLOCK_CAN1	
PM_CLOCK_DLL_REF	
PM_CLOCK_ADMA_REF	
PM_CLOCK_TIMESTAMP_REF	
PM_CLOCK_AMS_REF	
PM_CLOCK_PL0_REF	
PM_CLOCK_PL1_REF	
PM_CLOCK_PL2_REF	
PM_CLOCK_PL3_REF	
PM_CLOCK_WDT	
PM_CLOCK_IOPLL_INT	
PM_CLOCK_IOPLL_PRE_SRC	
PM_CLOCK_IOPLL_HALF	
PM_CLOCK_IOPLL_INT_MUX	
PM_CLOCK_IOPLL_POST_SRC	
PM_CLOCK_RPLL_INT	
PM_CLOCK_RPLL_PRE_SRC	
PM_CLOCK_RPLL_HALF	
PM_CLOCK_RPLL_INT_MUX	
PM_CLOCK_RPLL_POST_SRC	

Table 382: **Enumeration XPmClock Values** *(cont'd)*

Value	Description
PM_CLOCK_APLL_INT	
PM_CLOCK_APLL_PRE_SRC	
PM_CLOCK_APLL_HALF	
PM_CLOCK_APLL_INT_MUX	
PM_CLOCK_APLL_POST_SRC	
PM_CLOCK_DPLL_INT	
PM_CLOCK_DPLL_PRE_SRC	
PM_CLOCK_DPLL_HALF	
PM_CLOCK_DPLL_INT_MUX	
PM_CLOCK_DPLL_POST_SRC	
PM_CLOCK_VPLL_INT	
PM_CLOCK_VPLL_PRE_SRC	
PM_CLOCK_VPLL_HALF	
PM_CLOCK_VPLL_INT_MUX	
PM_CLOCK_VPLL_POST_SRC	
PM_CLOCK_CAN0_MIO	
PM_CLOCK_CAN1_MIO	
PM_CLOCK_ACPU_FULL	
PM_CLOCK_GEM0_REF	
PM_CLOCK_GEM1_REF	
PM_CLOCK_GEM2_REF	
PM_CLOCK_GEM3_REF	
PM_CLOCK_GEM0_REF_UNGATED	
PM_CLOCK_GEM1_REF_UNGATED	
PM_CLOCK_GEM2_REF_UNGATED	
PM_CLOCK_GEM3_REF_UNGATED	
PM_CLOCK_EXT_PSS_REF	
PM_CLOCK_EXT_VIDEO	
PM_CLOCK_EXT_PSS_ALT_REF	
PM_CLOCK_EXT_AUX_REF	
PM_CLOCK_EXT_GT_CRX_REF	
PM_CLOCK_EXT_SWDT0	
PM_CLOCK_EXT_SWDT1	
PM_CLOCK_EXT_GEM0_TX_EMIO	
PM_CLOCK_EXT_GEM1_TX_EMIO	
PM_CLOCK_EXT_GEM2_TX_EMIO	
PM_CLOCK_EXT_GEM3_TX_EMIO	
PM_CLOCK_EXT_GEM0_RX_EMIO	
PM_CLOCK_EXT_GEM1_RX_EMIO	

Table 382: **Enumeration XPmClock Values** *(cont'd)*

Value	Description
PM_CLOCK_EXT_GEM2_RX_EMIO	
PM_CLOCK_EXT_GEM3_RX_EMIO	
PM_CLOCK_EXT_MIO50_OR_MIO51	
PM_CLOCK_EXT_MIO0	
PM_CLOCK_EXT_MIO1	
PM_CLOCK_EXT_MIO2	
PM_CLOCK_EXT_MIO3	
PM_CLOCK_EXT_MIO4	
PM_CLOCK_EXT_MIO5	
PM_CLOCK_EXT_MIO6	
PM_CLOCK_EXT_MIO7	
PM_CLOCK_EXT_MIO8	
PM_CLOCK_EXT_MIO9	
PM_CLOCK_EXT_MIO10	
PM_CLOCK_EXT_MIO11	
PM_CLOCK_EXT_MIO12	
PM_CLOCK_EXT_MIO13	
PM_CLOCK_EXT_MIO14	
PM_CLOCK_EXT_MIO15	
PM_CLOCK_EXT_MIO16	
PM_CLOCK_EXT_MIO17	
PM_CLOCK_EXT_MIO18	
PM_CLOCK_EXT_MIO19	
PM_CLOCK_EXT_MIO20	
PM_CLOCK_EXT_MIO21	
PM_CLOCK_EXT_MIO22	
PM_CLOCK_EXT_MIO23	
PM_CLOCK_EXT_MIO24	
PM_CLOCK_EXT_MIO25	
PM_CLOCK_EXT_MIO26	
PM_CLOCK_EXT_MIO27	
PM_CLOCK_EXT_MIO28	
PM_CLOCK_EXT_MIO29	
PM_CLOCK_EXT_MIO30	
PM_CLOCK_EXT_MIO31	
PM_CLOCK_EXT_MIO32	
PM_CLOCK_EXT_MIO33	
PM_CLOCK_EXT_MIO34	
PM_CLOCK_EXT_MIO35	

Table 382: **Enumeration XPmClock Values** *(cont'd)*

Value	Description
PM_CLOCK_EXT_MIO36	
PM_CLOCK_EXT_MIO37	
PM_CLOCK_EXT_MIO38	
PM_CLOCK_EXT_MIO39	
PM_CLOCK_EXT_MIO40	
PM_CLOCK_EXT_MIO41	
PM_CLOCK_EXT_MIO42	
PM_CLOCK_EXT_MIO43	
PM_CLOCK_EXT_MIO44	
PM_CLOCK_EXT_MIO45	
PM_CLOCK_EXT_MIO46	
PM_CLOCK_EXT_MIO47	
PM_CLOCK_EXT_MIO48	
PM_CLOCK_EXT_MIO49	
PM_CLOCK_EXT_MIO50	
PM_CLOCK_EXT_MIO51	
PM_CLOCK_EXT_MIO52	
PM_CLOCK_EXT_MIO53	
PM_CLOCK_EXT_MIO54	
PM_CLOCK_EXT_MIO55	
PM_CLOCK_EXT_MIO56	
PM_CLOCK_EXT_MIO57	
PM_CLOCK_EXT_MIO58	
PM_CLOCK_EXT_MIO59	
PM_CLOCK_EXT_MIO60	
PM_CLOCK_EXT_MIO61	
PM_CLOCK_EXT_MIO62	
PM_CLOCK_EXT_MIO63	
PM_CLOCK_EXT_MIO64	
PM_CLOCK_EXT_MIO65	
PM_CLOCK_EXT_MIO66	
PM_CLOCK_EXT_MIO67	
PM_CLOCK_EXT_MIO68	
PM_CLOCK_EXT_MIO69	
PM_CLOCK_EXT_MIO70	
PM_CLOCK_EXT_MIO71	
PM_CLOCK_EXT_MIO72	
PM_CLOCK_EXT_MIO73	
PM_CLOCK_EXT_MIO74	

Table 382: **Enumeration XPmClock Values** *(cont'd)*

Value	Description
PM_CLOCK_EXT_MIO75	
PM_CLOCK_EXT_MIO76	
PM_CLOCK_EXT_MIO77	

Enumeration XPmPllParam

Table 383: **Enumeration XPmPllParam Values**

Value	Description
PM_PLL_PARAM_ID_DIV2	
PM_PLL_PARAM_ID_FBDIV	
PM_PLL_PARAM_ID_DATA	
PM_PLL_PARAM_ID_PRE_SRC	
PM_PLL_PARAM_ID_POST_SRC	
PM_PLL_PARAM_ID_LOCK_DLY	
PM_PLL_PARAM_ID_LOCK_CNT	
PM_PLL_PARAM_ID_LFHF	
PM_PLL_PARAM_ID_CP	
PM_PLL_PARAM_ID_RES	

Enumeration XPmPllMode

Table 384: **Enumeration XPmPllMode Values**

Value	Description
PM_PLL_MODE_INTEGER	
PM_PLL_MODE_FRACTIONAL	
PM_PLL_MODE_RESET	
PM_PLL_MODE_RESET	
PM_PLL_MODE_INTEGER	
PM_PLL_MODE_FRACTIONAL	

Enumeration XPmPinFn

Table 385: **Enumeration XPmPinFn Values**

Value	Description
PINCTRL_FUNC_CAN0	
PINCTRL_FUNC_CAN1	
PINCTRL_FUNC_ETHERNET0	

Table 385: **Enumeration XPmPinFn Values** *(cont'd)*

Value	Description
PINCTRL_FUNC_ETHERNET1	
PINCTRL_FUNC_ETHERNET2	
PINCTRL_FUNC_ETHERNET3	
PINCTRL_FUNC_GEMTSU0	
PINCTRL_FUNC_GPIO0	
PINCTRL_FUNC_I2C0	
PINCTRL_FUNC_I2C1	
PINCTRL_FUNC_MDIO0	
PINCTRL_FUNC_MDIO1	
PINCTRL_FUNC_MDIO2	
PINCTRL_FUNC_MDIO3	
PINCTRL_FUNC_QSPI0	
PINCTRL_FUNC_QSPI_FBCLK	
PINCTRL_FUNC_QSPI_SS	
PINCTRL_FUNC_SPI0	
PINCTRL_FUNC_SPI1	
PINCTRL_FUNC_SPI0_SS	
PINCTRL_FUNC_SPI1_SS	
PINCTRL_FUNC_SDIO0	
PINCTRL_FUNC_SDIO0_PC	
PINCTRL_FUNC_SDIO0_CD	
PINCTRL_FUNC_SDIO0_WP	
PINCTRL_FUNC_SDIO1	
PINCTRL_FUNC_SDIO1_PC	
PINCTRL_FUNC_SDIO1_CD	
PINCTRL_FUNC_SDIO1_WP	
PINCTRL_FUNC_NAND0	
PINCTRL_FUNC_NAND0_CE	
PINCTRL_FUNC_NAND0_RB	
PINCTRL_FUNC_NAND0_DQS	
PINCTRL_FUNC_TTC0_CLK	
PINCTRL_FUNC_TTC0_WAV	
PINCTRL_FUNC_TTC1_CLK	
PINCTRL_FUNC_TTC1_WAV	
PINCTRL_FUNC_TTC2_CLK	
PINCTRL_FUNC_TTC2_WAV	
PINCTRL_FUNC_TTC3_CLK	
PINCTRL_FUNC_TTC3_WAV	
PINCTRL_FUNC_UART0	

Table 385: **Enumeration XPmPinFn Values** *(cont'd)*

Value	Description
PINCTRL_FUNC_UART1	
PINCTRL_FUNC_USB0	
PINCTRL_FUNC_USB1	
PINCTRL_FUNC_SWDT0_CLK	
PINCTRL_FUNC_SWDT0_RST	
PINCTRL_FUNC_SWDT1_CLK	
PINCTRL_FUNC_SWDT1_RST	
PINCTRL_FUNC_PMU0	
PINCTRL_FUNC_PCIE0	
PINCTRL_FUNC_CSU0	
PINCTRL_FUNC_DPAUX0	
PINCTRL_FUNC_PJTAG0	
PINCTRL_FUNC_TRACE0	
PINCTRL_FUNC_TRACE0_CLK	
PINCTRL_FUNC_TESTSCAN0	

Enumeration XPmPinParam

Table 386: **Enumeration XPmPinParam Values**

Value	Description
PINCTRL_CONFIG_SLEW_RATE	
PINCTRL_CONFIG_BIAS_STATUS	
PINCTRL_CONFIG_PULL_CTRL	
PINCTRL_CONFIG_SCHMITT_CMOS	
PINCTRL_CONFIG_DRIVE_STRENGTH	
PINCTRL_CONFIG_VOLTAGE_STATUS	

Definitions

Define PACK_PAYLOAD

Definition

```
#define PACK_PAYLOADpl[0] = (u32)(arg0);                          \
    pl[1] = (u32)(arg1);                          \
    pl[2] = (u32)(arg2);                          \
    pl[3] = (u32)(arg3);                          \
    pl[4] = (u32)(arg4);                          \
    pl[5] = (u32)(arg5);                          \
    pl[6] = (u32)(rsvd);                          \
    pm_dbg("%s(%x, %x, %x, %x, %x, %x)\n", (__func__), (arg1), (arg2),
(arg3), (arg4), (arg5), (rsvd));
```

Description

Define PACK_PAYLOAD0

Definition

```
#define PACK_PAYLOAD0PACK_PAYLOAD(pl, (api_id), 0U, 0U, 0U, 0U, 0U, 0U)
```

Description

Define PACK_PAYLOAD1

Definition

```
#define PACK_PAYLOAD1PACK_PAYLOAD(pl, (api_id), (arg1), 0U, 0U, 0U, 0U, 0U)
```

Description

Define PACK_PAYLOAD2

Definition

```
#define PACK_PAYLOAD2PACK_PAYLOAD(pl, (api_id), (arg1), (arg2), 0U, 0U, 0U,
0U)
```

Description

Define PACK_PAYLOAD3

Definition

```
#define PACK_PAYLOAD3PACK_PAYLOAD(pl, (api_id), (arg1), (arg2), (arg3), 0U,
0U, 0U)
```

Description

Define PACK_PAYLOAD4

Definition

```
#define PACK_PAYLOAD4PACK_PAYLOAD(pl, (api_id), (arg1), (arg2), (arg3),
(arg4), 0U, 0U)
```

Description

Define PACK_PAYLOAD5

Definition

```
#define PACK_PAYLOAD5PACK_PAYLOAD(pl, (api_id), (arg1), (arg2), (arg3),
(arg4), (arg5), 0U)
```

Description

Define PM_VERSION_MAJOR

Definition

```
#define PM_VERSION_MAJOR1
```

Description

Define PM_VERSION_MINOR

Definition

```
#define PM_VERSION_MINOR1
```

Description

Define PM_VERSION

Definition

```
#define PM_VERSION((PM_VERSION_MAJOR << 16) | PM_VERSION_MINOR)
```

Description

Define PM_CAP_ACCESS

Definition

```
#define PM_CAP_ACCESS0x1U
```

Description

Define PM_CAP_CONTEXT

Definition

```
#define PM_CAP_CONTEXT0x2U
```

Description

Define PM_CAP_WAKEUP

Definition

```
#define PM_CAP_WAKEUP0x4U
```

Description

Define NODE_STATE_OFF

Definition

```
#define NODE_STATE_OFF0
```

Description

Define NODE_STATE_ON

Definition

```
#define NODE_STATE_ON1
```

Description

Define PROC_STATE_FORCEDOFF

Definition

```
#define PROC_STATE_FORCEDOFF0
```

Description

Define PROC_STATE_ACTIVE

Definition

```
#define PROC_STATE_ACTIVE1
```

Description

Define PROC_STATE_SLEEP

Definition

```
#define PROC_STATE_SLEEP2
```

Description

Define PROC_STATE_SUSPENDING

Definition

```
#define PROC_STATE_SUSPENDING3
```

Description

Define MAX_LATENCY

Definition

```
#define MAX_LATENCY(~0U)
```

Description

Define MAX_QOS

Definition

```
#define MAX_QOS100U
```

Description

Define PMF_SHUTDOWN_TYPE_SHUTDOWN

Definition

```
#define PMF_SHUTDOWN_TYPE_SHUTDOWN0U
```

Description

Define PMF_SHUTDOWN_TYPE_RESET

Definition

```
#define PMF_SHUTDOWN_TYPE_RESET1U
```

Description

Define PMF_SHUTDOWN_SUBTYPE_SUBSYSTEM

Definition

```
#define PMF_SHUTDOWN_SUBTYPE_SUBSYSTEM0U
```

Description

Define PMF_SHUTDOWN_SUBTYPE_PS_ONLY

Definition

```
#define PMF_SHUTDOWN_SUBTYPE_PS_ONLY1U
```

Description

Define PMF_SHUTDOWN_SUBTYPE_SYSTEM

Definition

```
#define PMF_SHUTDOWN_SUBTYPE_SYSTEM2U
```

Description

Define PM_API_MIN

Definition

```
#define PM_API_MINPM_GET_API_VERSION
```

Description

Define PM_CLOCK_DIV0_ID

Definition

```
#define PM_CLOCK_DIV0_ID0U
```

Description

Define PM_CLOCK_DIV1_ID

Definition

```
#define PM_CLOCK_DIV1_ID1U
```

Description

XilPM Versal ACAP APIs

Xilinx Power Management (XilPM) provides Embedded Energy Management Interface (EEMI) APIs for power management on Versal ACAP devices. For more details about EEMI, see the Embedded Energy Management Interface (EEMI) API User Guide (UG1200).

The platform and power management functionality used by the APU/RPU applications is provided by the files in the 'XilPM<version>/versal/client' folder, where '<version>' is the version of the XilPM library.

Table 387: **Quick Function Reference**

Type	Name	Arguments
XStatus	XPm_IpiSend	struct XPm_Proc *const Proc u32 * Payload
XStatus	Xpm_IpiReadBuff32	struct XPm_Proc *const Proc u32 * Val1 u32 * Val2 u32 * Val3
XStatus	XPm_InitXilpm	XIpiPsu * IpiInst
enum XPmBootStatus	XPm_GetBootStatus	void
XStatus	XPm_GetChipID	u32 * IDCode u32 * Version
XStatus	XPm_GetApiVersion	version
XStatus	XPm_RequestNode	const u32 DeviceId const u32 Capabilities const u32 QoS const u32 Ack
XStatus	XPm_ReleaseNode	const u32 DeviceId

Table 387: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
XStatus	XPm_SetRequirement	const u32 DeviceId const u32 Capabilities const u32 QoS const u32 Ack
XStatus	XPm_GetNodeStatus	const u32 DeviceId XPm_NodeStatus *const NodeStatus
XStatus	XPm_ResetAssert	const u32 ResetId const u32 Action
XStatus	XPm_ResetGetStatus	const u32 ResetId u32 *const State
XStatus	XPm_PinCtrlRequest	const u32 PinId
XStatus	XPm_PinCtrlRelease	const u32 PinId
XStatus	XPm_PinCtrlSetFunction	const u32 PinId const u32 FunctionId
XStatus	XPm_PinCtrlGetFunction	const u32 PinId u32 *const FunctionId
XStatus	XPm_PinCtrlSetParameter	const u32 PinId const u32 ParamId const u32 ParamVal
XStatus	XPm_PinCtrlGetParameter	const u32 PinId const u32 ParamId u32 *const ParamVal
XStatus	XPm_DevIoctl	const u32 DeviceId const pm_ioctl_id IoctlId const u32 Arg1 const u32 Arg2 u32 *const Response
XStatus	XPm_ClockEnable	const u32 ClockId
XStatus	XPm_ClockDisable	const u32 ClockId

Table 387: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
XStatus	XPm_ClockGetStatus	const u32 ClockId u32 *const State
XStatus	XPm_ClockSetDivider	const u32 ClockId const u32 Divider
XStatus	XPm_ClockGetDivider	const u32 ClockId u32 *const Divider
XStatus	XPm_ClockSetParent	const u32 ClockId const u32 ParentIdx
XStatus	XPm_ClockGetParent	const u32 ClockId u32 *const ParentIdx
int	XPm_ClockGetRate	const u32 ClockId u32 *const Rate
int	XPm_ClockSetRate	const u32 ClockId const u32 Rate
XStatus	XPm_PllSetParameter	const u32 ClockId const enum XPm_PllConfigParams ParamId const u32 Value
XStatus	XPm_PllGetParameter	const u32 ClockId const enum XPm_PllConfigParams ParamId u32 *const Value
XStatus	XPm_PllSetMode	const u32 ClockId const u32 Value
XStatus	XPm_PllGetMode	const u32 ClockId u32 *const Value
XStatus	XPm_SelfSuspend	const u32 DeviceId const u32 Latency const u8 State const u64 Address

Table 387: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
XStatus	XPm_RequestWakeUp	const u32 TargetDevId const u8 SetAddress const u64 Address const u32 Ack
void	XPm_SuspendFinalize	void
XStatus	XPm_RequestSuspend	const u32 TargetSubsystemId const u32 Ack const u32 Latency const u32 State
XStatus	XPm_AbortSuspend	reason
XStatus	XPm_ForcePowerDown	const u32 TargetDevId const u32 Ack
XStatus	XPm_SystemShutdown	const u32 Type const u32 SubType
XStatus	XPm_SetWakeUpSource	const u32 TargetDeviceId const u32 DeviceId const u32 Enable
XStatus	XPm_Query	Qid const u32 Arg1 const u32 Arg2 const u32 Arg3 u32 *const Data
int	XPm_SetMaxLatency	const u32 DeviceId const u32 Latency
XStatus	XPm_GetOpCharacteristic	const u32 DeviceId const enum `XPmOpCharType` Type u32 *const Result
int	XPm_InitFinalize	void
int	XPm_RegisterNotifier	`XPm_Notifier` *const Notifier

Table 387: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
int	XPm_UnregisterNotifier	`XPm_Notifier` *const Notifier
void	XPm_InitSuspendCb	const enum `XPmSuspendReason` Reason const u32 Latency const u32 State const u32 Timeout
void	XPm_AcknowledgeCb	const u32 Node const XStatus Status const u32 Oppoint
void	XPm_NotifyCb	const u32 Node const enum `XPmNotifyEvent` Event const u32 Oppoint
int	XPm_SetConfiguration	void
int	XPm_MmioWrite	void
int	XPm_MmioRead	void
XStatus	XPm_FeatureCheck	const u32 FeatureId u32 * Version

Functions

XPm_IpiSend

Sends IPI request to the target module.

Prototype

```
XStatus XPm_IpiSend(struct XPm_Proc *const Proc, u32 *Payload);
```

Parameters

The following table lists the `XPm_IpiSend` function arguments.

Table 388: **XPm_IpiSend Arguments**

Type	Name	Description
struct XPm_Proc *const	Proc	Pointer to the processor who is initiating request
u32 *	Payload	API id and call arguments to be written in IPI buffer

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Xpm_IpiReadBuff32

Reads IPI Response after target module has handled interrupt.

Prototype

```
XStatus Xpm_IpiReadBuff32(struct XPm_Proc *const Proc, u32 *Val1, u32
*Val2, u32 *Val3);
```

Parameters

The following table lists the `Xpm_IpiReadBuff32` function arguments.

Table 389: **Xpm_IpiReadBuff32 Arguments**

Type	Name	Description
struct XPm_Proc *const	Proc	Pointer to the processor who is waiting and reading Response
u32 *	Val1	Used to return value from 2nd IPI buffer element (optional)
u32 *	Val2	Used to return value from 3rd IPI buffer element (optional)
u32 *	Val3	Used to return value from 4th IPI buffer element (optional)

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_InitXilpm

Initialize xilpm library.

Prototype

```
XStatus XPm_InitXilpm(XIpiPsu *IpiInst);
```

Parameters

The following table lists the `XPm_InitXilpm` function arguments.

Table 390: **XPm_InitXilpm Arguments**

Type	Name	Description
XIpiPsu *	IpiInst	Pointer to IPI driver instance

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_GetBootStatus

This Function returns information about the boot reason. If the boot is not a system startup but a resume, power down request bitfield for this processor will be cleared.

Prototype

```
enum
                XPmBootStatus
              XPm_GetBootStatus(void);
```

Returns

Returns processor boot status

- PM_RESUME : If the boot reason is because of system resume.
- PM_INITIAL_BOOT : If this boot is the initial system startup.

XPm_GetChipID

This function is used to request the version and ID code of a chip.

Prototype

```
XStatus XPm_GetChipID(u32 *IDCode, u32 *Version);
```

Parameters

The following table lists the `XPm_GetChipID` function arguments.

Table 391: **XPm_GetChipID Arguments**

Type	Name	Description
u32 *	IDCode	Returns the chip ID code.
u32 *	Version	Returns the chip version.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_GetApiVersion

This function is used to request the version number of the API running on the platform management controller.

Prototype

```
XStatus XPm_GetApiVersion(u32 *Version);
```

Parameters

The following table lists the `XPm_GetApiVersion` function arguments.

Table 392: **XPm_GetApiVersion Arguments**

Type	Name	Description
Commented parameter version does not exist in function XPm_GetApiVersion.	version	Returns the API 32-bit version number.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_RequestNode

This function is used to request the device.

Prototype

```
XStatus XPm_RequestNode(const u32 DeviceId, const u32 Capabilities, const
u32 QoS, const u32 Ack);
```

Parameters

The following table lists the `XPm_RequestNode` function arguments.

Table 393: **XPm_RequestNode Arguments**

Type	Name	Description
const u32	DeviceId	Device which needs to be requested

Table 393: **XPm_RequestNode Arguments** *(cont'd)*

Type	Name	Description
const u32	Capabilities	Device Capabilities, can be combined • PM_CAP_ACCESS : full access / functionality • PM_CAP_CONTEXT : preserve context • PM_CAP_WAKEUP : emit wake interrupts
const u32	QoS	Quality of Service (0-100) required
const u32	Ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ReleaseNode

This function is used to release the requested device.

Prototype

```
XStatus XPm_ReleaseNode(const u32 DeviceId);
```

Parameters

The following table lists the `XPm_ReleaseNode` function arguments.

Table 394: **XPm_ReleaseNode Arguments**

Type	Name	Description
const u32	DeviceId	Device which needs to be released

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SetRequirement

This function is used to set the requirement for specified device.

Prototype

```
XStatus XPm_SetRequirement(const u32 DeviceId, const u32 Capabilities,
const u32 QoS, const u32 Ack);
```

Parameters

The following table lists the `XPm_SetRequirement` function arguments.

Table 395: **XPm_SetRequirement Arguments**

Type	Name	Description
const u32	DeviceId	Device for which requirement needs to be set
const u32	Capabilities	Device Capabilities, can be combined • PM_CAP_ACCESS : full access / functionality • PM_CAP_CONTEXT : preserve context • PM_CAP_WAKEUP : emit wake interrupts
const u32	QoS	Quality of Service (0-100) required
const u32	Ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_GetNodeStatus

This function is used to get the device status.

Prototype

```
XStatus XPm_GetNodeStatus(const u32 DeviceId, XPm_NodeStatus *const
NodeStatus);
```

Parameters

The following table lists the `XPm_GetNodeStatus` function arguments.

Table 396: **XPm_GetNodeStatus Arguments**

Type	Name	Description
const u32	DeviceId	Device for which status is requested

Table 396: **XPm_GetNodeStatus Arguments** *(cont'd)*

Type	Name	Description
XPm_NodeStatus *const	NodeStatus	Structure pointer to store device status • Status - The current power state of the device ◦ For CPU nodes: - 0 : if CPU is powered down, - 1 : if CPU is active (powered up), - 8 : if CPU is suspending (powered up) ◦ For power islands and power domains: - 0 : if island is powered down, - 2 : if island is powered up ◦ For slaves: - 0 : if slave is powered down, - 1 : if slave is powered up, - 9 : if slave is in retention • Requirement - Requirements placed on the device by the caller • Usage ◦ 0 : node is not used by any PU, ◦ 1 : node is used by caller exclusively, ◦ 2 : node is used by other PU(s) only, ◦ 3 : node is used by caller and by other PU(s)

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ResetAssert

This function is used to assert or release reset for a particular reset line. Alternatively a reset pulse can be requested as well.

Prototype

```
XStatus XPm_ResetAssert(const u32 ResetId, const u32 Action);
```

Parameters

The following table lists the `XPm_ResetAssert` function arguments.

Table 397: **XPm_ResetAssert Arguments**

Type	Name	Description
const u32	ResetId	Reset ID
const u32	Action	Reset action to be taken • PM_RESET_ACTION_RELEASE for Release Reset • PM_RESET_ACTION_ASSERT for Assert Reset • PM_RESET_ACTION_PULSE for Pulse Reset

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ResetGetStatus

This function is used to get the status of reset.

Prototype

```
XStatus XPm_ResetGetStatus(const u32 ResetId, u32 *const State);
```

Parameters

The following table lists the `XPm_ResetGetStatus` function arguments.

Table 398: **XPm_ResetGetStatus Arguments**

Type	Name	Description
const u32	ResetId	Reset ID
u32 *const	State	Pointer to store the status of specified reset • 0 for reset released • 1 for reset asserted

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PinCtrlRequest

This function is used to request the pin.

Prototype

```
XStatus XPm_PinCtrlRequest(const u32 PinId);
```

Parameters

The following table lists the `XPm_PinCtrlRequest` function arguments.

Table 399: **XPm_PinCtrlRequest Arguments**

Type	Name	Description
const u32	PinId	Pin ID

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PinCtrlRelease

This function is used to release the pin.

Prototype

```
XStatus XPm_PinCtrlRelease(const u32 PinId);
```

Parameters

The following table lists the `XPm_PinCtrlRelease` function arguments.

Table 400: **XPm_PinCtrlRelease Arguments**

Type	Name	Description
const u32	PinId	Pin ID

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PinCtrlSetFunction

This function is used to set the function on specified pin.

Prototype

```
XStatus XPm_PinCtrlSetFunction(const u32 PinId, const u32 FunctionId);
```

Parameters

The following table lists the `XPm_PinCtrlSetFunction` function arguments.

Table 401: **XPm_PinCtrlSetFunction Arguments**

Type	Name	Description
const u32	PinId	Pin ID
const u32	FunctionId	Function ID which needs to be set

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PinCtrlGetFunction

This function is used to get the function on specified pin.

Prototype

```
XStatus XPm_PinCtrlGetFunction(const u32 PinId, u32 *const FunctionId);
```

Parameters

The following table lists the `XPm_PinCtrlGetFunction` function arguments.

Table 402: **XPm_PinCtrlGetFunction Arguments**

Type	Name	Description
const u32	PinId	Pin ID
u32 *const	FunctionId	Pointer to Function ID

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PinCtrlSetParameter

This function is used to set the pin parameter of specified pin.

The following table lists the parameter ID and their respective values:

ParamId	ParamVal
PINCTRL_CONFIG_SLEW_RATE	PINCTRL_SLEW_RATE_SLOW, PINCTRL_SLEW_RATE_FAST
PINCTRL_CONFIG_BIAS_STATUS	PINCTRL_BIAS_DISABLE, PINCTRL_BIAS_ENABLE
PINCTRL_CONFIG_PULL_CTRL	PINCTRL_BIAS_PULL_DOWN, PINCTRL_BIAS_PULL_UP

ParamId	ParamVal
PINCTRL_CONFIG_SCHMITT_CMOS	PINCTRL_INPUT_TYPE_CMOS, PINCTRL_INPUT_TYPE_SCHMITT
PINCTRL_CONFIG_DRIVE_STRENGTH	PINCTRL_DRIVE_STRENGTH_TRISTATE, PINCTRL_DRIVE_STRENGTH_4MA, PINCTRL_DRIVE_STRENGTH_8MA, PINCTRL_DRIVE_STRENGTH_12MA
PINCTRL_CONFIG_TRI_STATE	PINCTRL_TRI_STATE_DISABLE, PINCTRL_TRI_STATE_ENABLE

Prototype

```
XStatus XPm_PinCtrlSetParameter(const u32 PinId, const u32 ParamId, const
u32 ParamVal);
```

Parameters

The following table lists the `XPm_PinCtrlSetParameter` function arguments.

Table 403: **XPm_PinCtrlSetParameter Arguments**

Type	Name	Description
const u32	PinId	Pin ID
const u32	ParamId	Parameter ID
const u32	ParamVal	Value of the parameter

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PinCtrlGetParameter

This function is used to get the pin parameter of specified pin.

The following table lists the parameter ID and their respective values:

ParamId	ParamVal
PINCTRL_CONFIG_SLEW_RATE	PINCTRL_SLEW_RATE_SLOW, PINCTRL_SLEW_RATE_FAST
PINCTRL_CONFIG_BIAS_STATUS	PINCTRL_BIAS_DISABLE, PINCTRL_BIAS_ENABLE
PINCTRL_CONFIG_PULL_CTRL	PINCTRL_BIAS_PULL_DOWN, PINCTRL_BIAS_PULL_UP
PINCTRL_CONFIG_SCHMITT_CMOS	PINCTRL_INPUT_TYPE_CMOS, PINCTRL_INPUT_TYPE_SCHMITT
PINCTRL_CONFIG_DRIVE_STRENGTH	PINCTRL_DRIVE_STRENGTH_TRISTATE, PINCTRL_DRIVE_STRENGTH_4MA, PINCTRL_DRIVE_STRENGTH_8MA, PINCTRL_DRIVE_STRENGTH_12MA
PINCTRL_CONFIG_VOLTAGE_STATUS	1 for 1.8v mode, 0 for 3.3v mode
PINCTRL_CONFIG_TRI_STATE	PINCTRL_TRI_STATE_DISABLE, PINCTRL_TRI_STATE_ENABLE

Prototype

```
XStatus XPm_PinCtrlGetParameter(const u32 PinId, const u32 ParamId, u32
*const ParamVal);
```

Parameters

The following table lists the `XPm_PinCtrlGetParameter` function arguments.

Table 404: **XPm_PinCtrlGetParameter Arguments**

Type	Name	Description
const u32	PinId	Pin ID
const u32	ParamId	Parameter ID
u32 *const	ParamVal	Pointer to the value of the parameter

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_DevIoctl

This function performs driver-like IOCTL functions on shared system devices.

Prototype

```
XStatus XPm_DevIoctl(const u32 DeviceId, const pm_ioctl_id IoctlId, const
u32 Arg1, const u32 Arg2, u32 *const Response);
```

Parameters

The following table lists the `XPm_DevIoctl` function arguments.

Table 405: **XPm_DevIoctl Arguments**

Type	Name	Description
const u32	DeviceId	ID of the device
const pm_ioctl_id	IoctlId	IOCTL function ID
const u32	Arg1	Argument 1
const u32	Arg2	Argument 2
u32 *const	Response	Ioctl response

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockEnable

This function is used to enable the specified clock.

Prototype

```
XStatus XPm_ClockEnable(const u32 ClockId);
```

Parameters

The following table lists the `XPm_ClockEnable` function arguments.

Table 406: **XPm_ClockEnable Arguments**

Type	Name	Description
const u32	ClockId	Clock ID

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockDisable

This function is used to disable the specified clock.

Prototype

```
XStatus XPm_ClockDisable(const u32 ClockId);
```

Parameters

The following table lists the `XPm_ClockDisable` function arguments.

Table 407: **XPm_ClockDisable Arguments**

Type	Name	Description
const u32	ClockId	Clock ID

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockGetStatus

This function is used to get the state of specified clock.

Prototype

```
XStatus XPm_ClockGetStatus(const u32 ClockId, u32 *const State);
```

Parameters

The following table lists the `XPm_ClockGetStatus` function arguments.

Table 408: **XPm_ClockGetStatus Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
u32 *const	State	Pointer to store the clock state • 1 for enable and 0 for disable

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockSetDivider

This function is used to set the divider value for specified clock.

Prototype

```
XStatus XPm_ClockSetDivider(const u32 ClockId, const u32 Divider);
```

Parameters

The following table lists the `XPm_ClockSetDivider` function arguments.

Table 409: **XPm_ClockSetDivider Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
const u32	Divider	Value of the divider

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockGetDivider

This function is used to get divider value for specified clock.

Prototype

```
XStatus XPm_ClockGetDivider(const u32 ClockId, u32 *const Divider);
```

Parameters

The following table lists the `XPm_ClockGetDivider` function arguments.

Table 410: **XPm_ClockGetDivider Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
u32 *const	Divider	Pointer to store divider value

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockSetParent

This function is used to set the parent for specified clock.

Prototype

```
XStatus XPm_ClockSetParent(const u32 ClockId, const u32 ParentIdx);
```

Parameters

The following table lists the `XPm_ClockSetParent` function arguments.

Table 411: **XPm_ClockSetParent Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
const u32	ParentIdx	Parent clock index

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockGetParent

This function is used to get the parent of specified clock.

Prototype

```
XStatus XPm_ClockGetParent(const u32 ClockId, u32 *const ParentIdx);
```

Parameters

The following table lists the `XPm_ClockGetParent` function arguments.

Table 412: **XPm_ClockGetParent Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
u32 *const	ParentIdx	Pointer to store the parent clock index

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockGetRate

This function is used to get rate of specified clock.

Prototype

```
int XPm_ClockGetRate(const u32 ClockId, u32 *const Rate);
```

Parameters

The following table lists the `XPm_ClockGetRate` function arguments.

Table 413: **XPm_ClockGetRate Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
u32 *const	Rate	Pointer to store the rate clock

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ClockSetRate

This function is used to set the rate of specified clock.

Prototype

```
int XPm_ClockSetRate(const u32 ClockId, const u32 Rate);
```

Parameters

The following table lists the `XPm_ClockSetRate` function arguments.

Table 414: **XPm_ClockSetRate Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
const u32	Rate	Clock rate

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PllSetParameter

This function is used to set the parameters for specified PLL clock.

Prototype

```
XStatus XPm_PllSetParameter(const u32 ClockId, const enum
XPm_PllConfigParams ParamId, const u32 Value);
```

Parameters

The following table lists the `XPm_PllSetParameter` function arguments.

Table 415: **XPm_PllSetParameter Arguments**

Type	Name	Description
const u32	ClockId	Clock ID

Table 415: **XPm_PllSetParameter Arguments** *(cont'd)*

Type	Name	Description
const enum XPm_PllConfigParams	ParamId	Parameter ID • PM_PLL_PARAM_ID_DIV2 • PM_PLL_PARAM_ID_FBDIV • PM_PLL_PARAM_ID_DATA • PM_PLL_PARAM_ID_PRE_SRC • PM_PLL_PARAM_ID_POST_SRC • PM_PLL_PARAM_ID_LOCK_DLY • PM_PLL_PARAM_ID_LOCK_CNT • PM_PLL_PARAM_ID_LFHF • PM_PLL_PARAM_ID_CP • PM_PLL_PARAM_ID_RES
const u32	Value	Value of parameter (See register description for possible values)

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PllGetParameter

This function is used to get the parameter of specified PLL clock.

Prototype

```
XStatus XPm_PllGetParameter(const u32 ClockId, const enum
XPm_PllConfigParams ParamId, u32 *const Value);
```

Parameters

The following table lists the `XPm_PllGetParameter` function arguments.

Table 416: **XPm_PllGetParameter Arguments**

Type	Name	Description
const u32	ClockId	Clock ID

Table 416: **XPm_PllGetParameter Arguments** *(cont'd)*

Type	Name	Description
const enum XPm_PllConfigParams	ParamId	Parameter ID • PM_PLL_PARAM_ID_DIV2 • PM_PLL_PARAM_ID_FBDIV • PM_PLL_PARAM_ID_DATA • PM_PLL_PARAM_ID_PRE_SRC • PM_PLL_PARAM_ID_POST_SRC • PM_PLL_PARAM_ID_LOCK_DLY • PM_PLL_PARAM_ID_LOCK_CNT • PM_PLL_PARAM_ID_LFHF • PM_PLL_PARAM_ID_CP • PM_PLL_PARAM_ID_RES
u32 *const	Value	Pointer to store parameter value (See register description for possible values)

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PllSetMode

This function is used to set the mode of specified PLL clock.

Prototype

```
XStatus XPm_PllSetMode(const u32 ClockId, const u32 Value);
```

Parameters

The following table lists the `XPm_PllSetMode` function arguments.

Table 417: **XPm_PllSetMode Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
const u32	Value	Mode which need to be set • 0 for Reset mode • 1 for Integer mode • 2 for Fractional mode

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_PllGetMode

This function is used to get the mode of specified PLL clock.

Prototype

```
XStatus XPm_PllGetMode(const u32 ClockId, u32 *const Value);
```

Parameters

The following table lists the `XPm_PllGetMode` function arguments.

Table 418: **XPm_PllGetMode Arguments**

Type	Name	Description
const u32	ClockId	Clock ID
u32 *const	Value	Pointer to store the value of mode • 0 for Reset mode • 1 for Integer mode • 2 for Fractional mode

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SelfSuspend

This function is used by a CPU to declare that it is about to suspend itself.

Prototype

```
XStatus XPm_SelfSuspend(const u32 DeviceId, const u32 Latency, const u8
State, const u64 Address);
```

Parameters

The following table lists the `XPm_SelfSuspend` function arguments.

Table 419: **XPm_SelfSuspend Arguments**

Type	Name	Description
const u32	DeviceId	Device ID of the CPU
const u32	Latency	Maximum wake-up latency requirement in us(microsecs)
const u8	State	Instead of specifying a maximum latency, a CPU can also explicitly request a certain power state.
const u64	Address	Address from which to resume when woken up.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_RequestWakeUp

This function can be used to request power up of a CPU node within the same PU, or to power up another PU.

Prototype

```
XStatus XPm_RequestWakeUp(const u32 TargetDevId, const u8 SetAddress, const u64 Address, const u32 Ack);
```

Parameters

The following table lists the `XPm_RequestWakeUp` function arguments.

Table 420: **XPm_RequestWakeUp Arguments**

Type	Name	Description
const u32	TargetDevId	Device ID of the CPU or PU to be powered/woken up.
const u8	SetAddress	Specifies whether the start address argument is being passed. • 0 : do not set start address • 1 : set start address
const u64	Address	Address from which to resume when woken up. Will only be used if set_address is 1.
const u32	Ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SuspendFinalize

This Function waits for firmware to finish all previous API requests sent by the PU and performs client specific actions to finish suspend procedure (e.g. execution of wfi instruction on A53 and R5 processors).

***Note*:** This function should not return if the suspend procedure is successful.

Prototype

```
void XPm_SuspendFinalize(void);
```

Returns

XPm_RequestSuspend

This function is used by a CPU to request suspend to another CPU.

Prototype

```
XStatus XPm_RequestSuspend(const u32 TargetSubsystemId, const u32 Ack,
const u32 Latency, const u32 State);
```

Parameters

The following table lists the `XPm_RequestSuspend` function arguments.

Table 421: **XPm_RequestSuspend Arguments**

Type	Name	Description
const u32	TargetSubsystemId	Subsystem ID of the target
const u32	Ack	Requested acknowledge type
const u32	Latency	Maximum wake-up latency requirement in us(microsecs)
const u32	State	Power State

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_AbortSuspend

This function is called by a CPU after a SelfSuspend call to notify the platform management controller that CPU has aborted suspend or in response to an init suspend request when the PU refuses to suspend.

Prototype

```
XStatus XPm_AbortSuspend(const enum XPmAbortReason Reason);
```

Parameters

The following table lists the `XPm_AbortSuspend` function arguments.

Table 422: **XPm_AbortSuspend Arguments**

Type	Name	Description
Commented parameter reason does not exist in function XPm_AbortSuspend.	reason	Reason code why the suspend can not be performed or completed • ABORT_REASON_WKUP_EVENT : local wakeup-event received • ABORT_REASON_PU_BUSY : PU is busy • ABORT_REASON_NO_PWRDN : no external powerdown supported • ABORT_REASON_UNKNOWN : unknown error during suspend procedure

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_ForcePowerDown

This function is used by PU to request a forced poweroff of another PU or its power island or power domain. This can be used for killing an unresponsive PU, in which case all resources of that PU will be automatically released.

***Note*:** Force power down may not be requested by a PU for itself.

Prototype

```
XStatus XPm_ForcePowerDown(const u32 TargetDevId, const u32 Ack);
```

Parameters

The following table lists the `XPm_ForcePowerDown` function arguments.

Table 423: **XPm_ForcePowerDown Arguments**

Type	Name	Description
const u32	TargetDevId	Device ID of the PU node to be forced powered down.
const u32	Ack	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SystemShutdown

This function can be used by a privileged PU to shut down or restart the complete device.

Prototype

```
XStatus XPm_SystemShutdown(const u32 Type, const u32 SubType);
```

Parameters

The following table lists the `XPm_SystemShutdown` function arguments.

Table 424: **XPm_SystemShutdown Arguments**

Type	Name	Description
const u32	Type	Shutdown type (shutdown/restart)
const u32	SubType	Shutdown subtype (subsystem-only/PU-only/system)

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SetWakeUpSource

This function is used by a CPU to set wakeup source.

Prototype

```
XStatus XPm_SetWakeUpSource(const u32 TargetDeviceId, const u32 DeviceId,
const u32 Enable);
```

Parameters

The following table lists the `XPm_SetWakeUpSource` function arguments.

Table 425: **XPm_SetWakeUpSource Arguments**

Type	Name	Description
const u32	TargetDeviceId	Device ID of the target
const u32	DeviceId	Device ID used as wakeup source
const u32	Enable	1 - Enable, 0 - Disable

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_Query

This function queries information about the platform resources.

Prototype

```
XStatus XPm_Query(const u32 QueryId, const u32 Arg1, const u32 Arg2, const
u32 Arg3, u32 *const Data);
```

Parameters

The following table lists the `XPm_Query` function arguments.

Table 426: **XPm_Query Arguments**

Type	Name	Description
Commented parameter Qid does not exist in function XPm_Query.	Qid	The type of data to query
const u32	Arg1	Query argument 1
const u32	Arg2	Query argument 2
const u32	Arg3	Query argument 3
u32 *const	Data	Pointer to the output data

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_SetMaxLatency

This function is used by a CPU to announce a change in the maximum wake-up latency requirements for a specific device currently used by that CPU.

***Note*:** Setting maximum wake-up latency can constrain the set of possible power states a resource can be put into.

Prototype

```
int XPm_SetMaxLatency(const u32 DeviceId, const u32 Latency);
```

Parameters

The following table lists the `XPm_SetMaxLatency` function arguments.

Table 427: **XPm_SetMaxLatency Arguments**

Type	Name	Description
const u32	DeviceId	Device ID.
const u32	Latency	Maximum wake-up latency required.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_GetOpCharacteristic

Call this function to request the power management controller to return information about an operating characteristic of a component.

***Note*:** Currently power type is not supported for Versal.

Prototype

```
XStatus XPm_GetOpCharacteristic(const u32 DeviceId, const enum
XPmOpCharType Type, u32 *const Result);
```

Parameters

The following table lists the `XPm_GetOpCharacteristic` function arguments.

Table 428: **XPm_GetOpCharacteristic Arguments**

Type	Name	Description
const u32	DeviceId	Device ID.
const enum `XPmOpCharType`	Type	Type of operating characteristic requested: • power (current power consumption), • latency (current latency in micro seconds to return to active state), • temperature (current temperature in Celsius (Q8.7 format)),
u32 *const	Result	Used to return the requested operating characteristic.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_InitFinalize

This function is called to notify the power management controller about the completed power management initialization.

Prototype

```
int XPm_InitFinalize(void);
```

Returns

XST_SUCCESS if successful, otherwise an error code

XPm_RegisterNotifier

A PU can call this function to request that the power management controller call its notify callback whenever a qualifying event occurs. One can request to be notified for a specific or any event related to a specific node.

***Note*:** The caller shall initialize the notifier object before invoking the XPm_RegisteredNotifier function. While notifier is registered, the notifier object shall not be modified by the caller.

Prototype

```
int XPm_RegisterNotifier(XPm_Notifier *const Notifier);
```

Parameters

The following table lists the `XPm_RegisterNotifier` function arguments.

Table 429: **XPm_RegisterNotifier Arguments**

Type	Name	Description
`XPm_Notifier` *const	Notifier	Pointer to the notifier object to be associated with the requested notification.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_UnregisterNotifier

A PU calls this function to unregister for the previously requested notifications.

Prototype

```
int XPm_UnregisterNotifier(XPm_Notifier *const Notifier);
```

Parameters

The following table lists the `XPm_UnregisterNotifier` function arguments.

Table 430: **XPm_UnregisterNotifier Arguments**

Type	Name	Description
`XPm_Notifier` *const	Notifier	Pointer to the notifier object associated with the previously requested notification

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

XPm_InitSuspendCb

Callback function to be implemented in each PU, allowing the power management controller to request that the PU suspend itself.

Note: If the PU fails to act on this request the power management controller or the requesting PU may choose to employ the forceful power down option.

Prototype

```
void XPm_InitSuspendCb(const enum XPmSuspendReason Reason, const u32
Latency, const u32 State, const u32 Timeout);
```

Parameters

The following table lists the `XPm_InitSuspendCb` function arguments.

Table 431: **XPm_InitSuspendCb Arguments**

Type	Name	Description
const enum `XPmSuspendReason`	Reason	Suspend reason: • SUSPEND_REASON_PU_REQ : Request by another PU • SUSPEND_REASON_ALERT : Unrecoverable SysMon alert • SUSPEND_REASON_SHUTDOWN : System shutdown • SUSPEND_REASON_RESTART : System restart
const u32	Latency	Maximum wake-up latency in us(micro secs). This information can be used by the PU to decide what level of context saving may be required.
const u32	State	Targeted sleep/suspend state.
const u32	Timeout	Timeout in ms, specifying how much time a PU has to initiate its suspend procedure before it's being considered unresponsive.

Returns

None

XPm_AcknowledgeCb

This function is called by the power management controller in response to any request where an acknowledge callback was requested, i.e. where the 'ack' argument passed by the PU was REQUEST_ACK_NON_BLOCKING.

Prototype

```
void XPm_AcknowledgeCb(const u32 Node, const XStatus Status, const u32
Oppoint);
```

Parameters

The following table lists the `XPm_AcknowledgeCb` function arguments.

Table 432: **XPm_AcknowledgeCb Arguments**

Type	Name	Description
const u32	Node	ID of the component or sub-system in question.
const XStatus	Status	Status of the operation: • OK: the operation completed successfully • ERR: the requested operation failed
const u32	Oppoint	Operating point of the node in question

Returns

None

XPm_NotifyCb

This function is called by the power management controller if an event the PU was registered for has occurred. It will populate the notifier data structure passed when calling XPm_RegisterNotifier.

Prototype

```
void XPm_NotifyCb(const u32 Node, const enum XPmNotifyEvent Event, const
u32 Oppoint);
```

Parameters

The following table lists the `XPm_NotifyCb` function arguments.

Table 433: **XPm_NotifyCb Arguments**

Type	Name	Description
const u32	Node	ID of the device the event notification is related to.
const enum XPmNotifyEvent	Event	ID of the event
const u32	Oppoint	Current operating state of the device.

Returns

None

XPm_SetConfiguration

Prototype

```
int XPm_SetConfiguration(const u32 Address);
```

XPm_MmioWrite

Prototype

```
int XPm_MmioWrite(const u32 Address, const u32 Mask, const u32 Value);
```

XPm_MmioRead

Prototype

```
int XPm_MmioRead(const u32 Address, u32 *const Value);
```

XPm_FeatureCheck

This function queries information about the feature version.

Prototype

```
XStatus XPm_FeatureCheck(const u32 FeatureId, u32 *Version);
```

Parameters

The following table lists the `XPm_FeatureCheck` function arguments.

Table 434: **XPm_FeatureCheck Arguments**

Type	Name	Description
const u32	FeatureId	The feature ID (API-ID)
u32 *	Version	Pointer to the output data where version of feature store. For the supported feature get non zero value in version, But if version is 0U that means feature not supported.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Enumerations

Enumeration XPmAbortReason

PM abort reasons enumeration.

Table 435: **Enumeration XPmAbortReason Values**

Value	Description
ABORT_REASON_WKUP_EVENT	
ABORT_REASON_PU_BUSY	
ABORT_REASON_NO_PWRDN	
ABORT_REASON_UNKNOWN	

Enumeration XPmBootStatus

Boot status enumeration.

Table 436: **Enumeration XPmBootStatus Values**

Value	Description
PM_INITIAL_BOOT	boot is a fresh system startup
PM_RESUME	boot is a resume
PM_BOOT_ERROR	error, boot cause cannot be identified

Enumeration XPmCapability

Device capability requirements enumeration.

Table 437: **Enumeration XPmCapability Values**

Value	Description
PM_CAP_ACCESS	Full access

Table 437: **Enumeration XPmCapability Values** *(cont'd)*

Value	Description
PM_CAP_CONTEXT	Configuration and contents retained
PM_CAP_WAKEUP	Enabled as a wake-up source
PM_CAP_UNUSABLE	Not usable

Enumeration XPmDeviceUsage

Table 438: **Enumeration XPmDeviceUsage Values**

Value	Description
PM_USAGE_CURRENT_SUBSYSTEM	
PM_USAGE_OTHER_SUBSYSTEM	

Enumeration XPmResetActions

Table 439: **Enumeration XPmResetActions Values**

Value	Description
PM_RESET_ACTION_RELEASE	
PM_RESET_ACTION_ASSERT	
PM_RESET_ACTION_PULSE	

Enumeration XPmSuspendReason

Table 440: **Enumeration XPmSuspendReason Values**

Value	Description
SUSPEND_REASON_PU_REQ	
SUSPEND_REASON_ALERT	
SUSPEND_REASON_SYS_SHUTDOWN	

Enumeration XPmApiCbId_t

Table 441: **Enumeration XPmApiCbId_t Values**

Value	Description
PM_INIT_SUSPEND_CB	
PM_ACKNOWLEDGE_CB	
PM_NOTIFY_CB	

Enumeration pm_query_id

Table 442: **Enumeration pm_query_id Values**

Value	Description
XPM_QID_INVALID	
XPM_QID_CLOCK_GET_NAME	
XPM_QID_CLOCK_GET_TOPOLOGY	
XPM_QID_CLOCK_GET_FIXEDFACTOR_PARAMS	
XPM_QID_CLOCK_GET_MUXSOURCES	
XPM_QID_CLOCK_GET_ATTRIBUTES	
XPM_QID_PINCTRL_GET_NUM_PINS	
XPM_QID_PINCTRL_GET_NUM_FUNCTIONS	
XPM_QID_PINCTRL_GET_NUM_FUNCTION_GROUPS	
XPM_QID_PINCTRL_GET_FUNCTION_NAME	
XPM_QID_PINCTRL_GET_FUNCTION_GROUPS	
XPM_QID_PINCTRL_GET_PIN_GROUPS	
XPM_QID_CLOCK_GET_NUM_CLOCKS	
XPM_QID_CLOCK_GET_MAX_DIVISOR	

Enumeration PmPinFunIds

Table 443: **Enumeration PmPinFunIds Values**

Value	Description
PIN_FUNC_SPI0	
PIN_FUNC_SPI0_SS	
PIN_FUNC_SPI1	
PIN_FUNC_SPI1_SS	
PIN_FUNC_CAN0	
PIN_FUNC_CAN1	
PIN_FUNC_I2C0	
PIN_FUNC_I2C1	
PIN_FUNC_I2C_PMC	
PIN_FUNC_TTC0_CLK	
PIN_FUNC_TTC0_WAV	
PIN_FUNC_TTC1_CLK	
PIN_FUNC_TTC1_WAV	
PIN_FUNC_TTC2_CLK	
PIN_FUNC_TTC2_WAV	
PIN_FUNC_TTC3_CLK	
PIN_FUNC_TTC3_WAV	

Table 443: **Enumeration PmPinFunIds Values** *(cont'd)*

Value	Description
PIN_FUNC_WWDT0	
PIN_FUNC_WWDT1	
PIN_FUNC_SYSMON_I2C0	
PIN_FUNC_SYSMON_I2C0_ALERT	
PIN_FUNC_UART0	
PIN_FUNC_UART0_CTRL	
PIN_FUNC_UART1	
PIN_FUNC_UART1_CTRL	
PIN_FUNC_GPIO0	
PIN_FUNC_GPIO1	
PIN_FUNC_GPIO2	
PIN_FUNC_EMIO0	
PIN_FUNC_GEM0	
PIN_FUNC_GEM1	
PIN_FUNC_TRACE0	
PIN_FUNC_TRACE0_CLK	
PIN_FUNC_MDIO0	
PIN_FUNC_MDIO1	
PIN_FUNC_GEM_TSU0	
PIN_FUNC_PCIE0	
PIN_FUNC_SMAP0	
PIN_FUNC_USB0	
PIN_FUNC_SD0	
PIN_FUNC_SD0_PC	
PIN_FUNC_SD0_CD	
PIN_FUNC_SD0_WP	
PIN_FUNC_SD1	
PIN_FUNC_SD1_PC	
PIN_FUNC_SD1_CD	
PIN_FUNC_SD1_WP	
PIN_FUNC_OSPI0	
PIN_FUNC_OSPI0_SS	
PIN_FUNC_QSPI0	
PIN_FUNC_QSPI0_FBCLK	
PIN_FUNC_QSPI0_SS	
PIN_FUNC_TEST_CLK	
PIN_FUNC_TEST_SCAN	
PIN_FUNC_TAMPER_TRIGGER	
MAX_FUNCTION	

Enumeration pm_pinctrl_config_param

Table 444: **Enumeration pm_pinctrl_config_param Values**

Value	Description
PINCTRL_CONFIG_SLEW_RATE	
PINCTRL_CONFIG_BIAS_STATUS	
PINCTRL_CONFIG_PULL_CTRL	
PINCTRL_CONFIG_SCHMITT_CMOS	
PINCTRL_CONFIG_DRIVE_STRENGTH	
PINCTRL_CONFIG_VOLTAGE_STATUS	
PINCTRL_CONFIG_TRI_STATE	
PINCTRL_CONFIG_MAX	

Enumeration pm_pinctrl_slew_rate

Table 445: **Enumeration pm_pinctrl_slew_rate Values**

Value	Description
PINCTRL_SLEW_RATE_FAST	
PINCTRL_SLEW_RATE_SLOW	

Enumeration pm_pinctrl_bias_status

Table 446: **Enumeration pm_pinctrl_bias_status Values**

Value	Description
PINCTRL_BIAS_DISABLE	
PINCTRL_BIAS_ENABLE	

Enumeration pm_pinctrl_pull_ctrl

Table 447: **Enumeration pm_pinctrl_pull_ctrl Values**

Value	Description
PINCTRL_BIAS_PULL_DOWN	
PINCTRL_BIAS_PULL_UP	

Enumeration pm_pinctrl_schmitt_cmos

Table 448: **Enumeration pm_pinctrl_schmitt_cmos Values**

Value	Description
PINCTRL_INPUT_TYPE_CMOS	
PINCTRL_INPUT_TYPE_SCHMITT	

Enumeration pm_pinctrl_drive_strength

Table 449: **Enumeration pm_pinctrl_drive_strength Values**

Value	Description
PINCTRL_DRIVE_STRENGTH_TRISTATE	
PINCTRL_DRIVE_STRENGTH_4MA	
PINCTRL_DRIVE_STRENGTH_8MA	
PINCTRL_DRIVE_STRENGTH_12MA	
PINCTRL_DRIVE_STRENGTH_MAX	

Enumeration pm_pinctrl_tri_state

Table 450: **Enumeration pm_pinctrl_tri_state Values**

Value	Description
PINCTRL_TRI_STATE_DISABLE	
PINCTRL_TRI_STATE_ENABLE	

Enumeration pm_ioctl_id

Table 451: **Enumeration pm_ioctl_id Values**

Value	Description
IOCTL_GET_RPU_OPER_MODE	
IOCTL_SET_RPU_OPER_MODE	
IOCTL_RPU_BOOT_ADDR_CONFIG	
IOCTL_TCM_COMB_CONFIG	
IOCTL_SET_TAPDELAY_BYPASS	
IOCTL_SET_SGMII_MODE	
IOCTL_SD_DLL_RESET	
IOCTL_SET_SD_TAPDELAY	
IOCTL_SET_PLL_FRAC_MODE	
IOCTL_GET_PLL_FRAC_MODE	
IOCTL_SET_PLL_FRAC_DATA	

Table 451: **Enumeration pm_ioctl_id Values** *(cont'd)*

Value	Description
IOCTL_GET_PLL_FRAC_DATA	
IOCTL_WRITE_GGS	
IOCTL_READ_GGS	
IOCTL_WRITE_PGGS	
IOCTL_READ_PGGS	
IOCTL_ULPI_RESET	
IOCTL_SET_BOOT_HEALTH_STATUS	
IOCTL_AFI	
IOCTL_PROBE_COUNTER_READ	
IOCTL_PROBE_COUNTER_WRITE	
IOCTL_OSPI_MUX_SELECT	
IOCTL_USB_SET_STATE	

Enumeration XPm_PllConfigParams

Table 452: **Enumeration XPm_PllConfigParams Values**

Value	Description
PM_PLL_PARAM_ID_DIV2	
PM_PLL_PARAM_ID_FBDIV	
PM_PLL_PARAM_ID_DATA	
PM_PLL_PARAM_ID_PRE_SRC	
PM_PLL_PARAM_ID_POST_SRC	
PM_PLL_PARAM_ID_LOCK_DLY	
PM_PLL_PARAM_ID_LOCK_CNT	
PM_PLL_PARAM_ID_LFHF	
PM_PLL_PARAM_ID_CP	
PM_PLL_PARAM_ID_RES	
PM_PLL_PARAM_MAX	

Enumeration XPmPllMode

Table 453: **Enumeration XPmPllMode Values**

Value	Description
PM_PLL_MODE_INTEGER	
PM_PLL_MODE_FRACTIONAL	
PM_PLL_MODE_RESET	

Enumeration XPmInitFunctions

PM init node functions

Table 454: **Enumeration XPmInitFunctions Values**

Value	Description
FUNC_INIT_START	
FUNC_INIT_FINISH	
FUNC_SCAN_CLEAR	
FUNC_BISR	
FUNC_LBIST	
FUNC_MEM_INIT	
FUNC_MBIST_CLEAR	
FUNC_HOUSECLEAN_PL	
FUNC_HOUSECLEAN_COMPLETE	
FUNC_XPPU_CTRL	
FUNC_XMPU_CTRL	

Enumeration XPmOpCharType

PM operating characteristic types enumeration.

Table 455: **Enumeration XPmOpCharType Values**

Value	Description
PM_OPCHAR_TYPE_POWER	
PM_OPCHAR_TYPE_TEMP	
PM_OPCHAR_TYPE_LATENCY	

Enumeration XPmNotifyEvent

PM notify events enumeration.

Table 456: **Enumeration XPmNotifyEvent Values**

Value	Description
EVENT_STATE_CHANGE	
EVENT_ZERO_USERS	

Definitions

Define PM_VERSION_MAJOR

Definition

```
#define PM_VERSION_MAJOR1UL
```

Description

Define PM_VERSION_MINOR

Definition

```
#define PM_VERSION_MINOR0UL
```

Description

Define PM_VERSION

Definition

```
#define PM_VERSION((PM_VERSION_MAJOR << 16) | PM_VERSION_MINOR)
```

Description

Define XPM_MAX_CAPABILITY

Definition

```
#define XPM_MAX_CAPABILITY((u32)
               PM_CAP_ACCESS
             | (u32)
               PM_CAP_CONTEXT
             | (u32)
               PM_CAP_WAKEUP
            )
```

Description

Define XPM_MAX_LATENCY

Definition

```
#define XPM_MAX_LATENCY(0xFFFFU)
```

Description

Define XPM_MAX_QOS

Definition

```
#define XPM_MAX_QOS(100)
```

Description

Define XPM_MIN_CAPABILITY

Definition

```
#define XPM_MIN_CAPABILITY(0)
```

Description

Define XPM_MIN_LATENCY

Definition

```
#define XPM_MIN_LATENCY(0)
```

Description

Define XPM_MIN_QOS

Definition

```
#define XPM_MIN_QOS(0)
```

Description

Define XPM_DEF_CAPABILITY

Definition

```
#define XPM_DEF_CAPABILITYXPM_MAX_CAPABILITY
```

Description

Define XPM_DEF_LATENCY

Definition

```
#define XPM_DEF_LATENCYXPM_MAX_LATENCY
```

Description

Define XPM_DEF_QOS

Definition

```
#define XPM_DEF_QOSXPM_MAX_QOS
```

Description

Define NODE_STATE_OFF

Definition

```
#define NODE_STATE_OFF(0U)
```

Description

Define NODE_STATE_ON

Definition

```
#define NODE_STATE_ON(1U)
```

Description

Define PROC_STATE_SLEEP

Definition

```
#define PROC_STATE_SLEEPNODE_STATE_OFF
```

Description

Define PROC_STATE_ACTIVE

Definition

```
#define PROC_STATE_ACTIVENODE_STATE_ON
```

Description

Define PROC_STATE_FORCEDOFF

Definition

```
#define PROC_STATE_FORCEDOFF(7U)
```

Description

Define PROC_STATE_SUSPENDING

Definition

```
#define PROC_STATE_SUSPENDING(8U)
```

Description

Define PM_SHUTDOWN_TYPE_SHUTDOWN

Definition

```
#define PM_SHUTDOWN_TYPE_SHUTDOWN(0U)
```

Description

Define PM_SHUTDOWN_TYPE_RESET

Definition

```
#define PM_SHUTDOWN_TYPE_RESET(1U)
```

Description

Define PM_SHUTDOWN_SUBTYPE_RST_SUBSYSTEM

Definition

```
#define PM_SHUTDOWN_SUBTYPE_RST_SUBSYSTEM(0U)
```

Description

Define PM_SHUTDOWN_SUBTYPE_RST_PS_ONLY

Definition

```
#define PM_SHUTDOWN_SUBTYPE_RST_PS_ONLY(1U)
```

Description

Define PM_SHUTDOWN_SUBTYPE_RST_SYSTEM

Definition

```
#define PM_SHUTDOWN_SUBTYPE_RST_SYSTEM(2U)
```

Description

Define PM_SUSPEND_STATE_CPU_IDLE

Definition

```
#define PM_SUSPEND_STATE_CPU_IDLE0x0U
```

Description

Define PM_SUSPEND_STATE_SUSPEND_TO_RAM

Definition

```
#define PM_SUSPEND_STATE_SUSPEND_TO_RAM0xFU
```

Description

Define XPM_RPU_MODE_LOCKSTEP

Definition

```
#define XPM_RPU_MODE_LOCKSTEP0U
```

Description

Define XPM_RPU_MODE_SPLIT

Definition

```
#define XPM_RPU_MODE_SPLIT1U
```

Description

Define XPM_RPU_BOOTMEM_LOVEC

Definition

```
#define XPM_RPU_BOOTMEM_LOVEC(0U)
```

Description

Define XPM_RPU_BOOTMEM_HIVEC

Definition

```
#define XPM_RPU_BOOTMEM_HIVEC(1U)
```

Description

Define XPM_RPU_TCM_SPLIT

Definition

```
#define XPM_RPU_TCM_SPLIT0U
```

Description

Define XPM_RPU_TCM_COMB

Definition

```
#define XPM_RPU_TCM_COMB1U
```

Description

Define XPM_BOOT_HEALTH_STATUS_MASK

Definition

```
#define XPM_BOOT_HEALTH_STATUS_MASK(0x1U)
```

Description

Define XPM_TAPDELAY_QSPI

Definition

```
#define XPM_TAPDELAY_QSPI(2U)
```

Description

Define XPM_TAPDELAY_BYPASS_DISABLE

Definition

```
#define XPM_TAPDELAY_BYPASS_DISABLE(0U)
```

Description

Define XPM_TAPDELAY_BYPASS_ENABLE

Definition

```
#define XPM_TAPDELAY_BYPASS_ENABLE(1U)
```

Description

Define XPM_OSPI_MUX_SEL_DMA

Definition

```
#define XPM_OSPI_MUX_SEL_DMA(0U)
```

Description

Define XPM_OSPI_MUX_SEL_LINEAR

Definition

```
#define XPM_OSPI_MUX_SEL_LINEAR(1U)
```

Description

Define XPM_OSPI_MUX_GET_MODE

Definition

```
#define XPM_OSPI_MUX_GET_MODE(2U)
```

Description

Define XPM_TAPDELAY_INPUT

Definition

```
#define XPM_TAPDELAY_INPUT(0U)
```

Description

Define XPM_TAPDELAY_OUTPUT

Definition

```
#define XPM_TAPDELAY_OUTPUT(1U)
```

Description

Define XPM_DLL_RESET_ASSERT

Definition

```
#define XPM_DLL_RESET_ASSERT(0U)
```

Description

Define XPM_DLL_RESET_RELEASE

Definition

```
#define XPM_DLL_RESET_RELEASE(1U)
```

Description

Define XPM_DLL_RESET_PULSE

Definition

```
#define XPM_DLL_RESET_PULSE(2U)
```

Description

Define XPM_RESET_REASON_EXT_POR

Definition

```
#define XPM_RESET_REASON_EXT_POR(0U)
```

Description

Define XPM_RESET_REASON_SW_POR

Definition

```
#define XPM_RESET_REASON_SW_POR(1U)
```

Description

Define XPM_RESET_REASON_SLR_POR

Definition

```
#define XPM_RESET_REASON_SLR_POR(2U)
```

Description

Define XPM_RESET_REASON_ERR_POR

Definition

```
#define XPM_RESET_REASON_ERR_POR(3U)
```

Description

Define XPM_RESET_REASON_DAP_SRST

Definition

```
#define XPM_RESET_REASON_DAP_SRST(7U)
```

Description

Define XPM_RESET_REASON_ERR_SRST

Definition

```
#define XPM_RESET_REASON_ERR_SRST(8U)
```

Description

Define XPM_RESET_REASON_SW_SRST

Definition

```
#define XPM_RESET_REASON_SW_SRST(9U)
```

Description

Define XPM_RESET_REASON_SLR_SRST

Definition

```
#define XPM_RESET_REASON_SLR_SRST(10U)
```

Description

Define XPM_RESET_REASON_INVALID

Definition

```
#define XPM_RESET_REASON_INVALID(0xFFU)
```

Description

Define XPM_PROBE_COUNTER_TYPE_LAR_LSR

Definition

```
#define XPM_PROBE_COUNTER_TYPE_LAR_LSR(0U)
```

Description

Define XPM_PROBE_COUNTER_TYPE_MAIN_CTL

Definition

```
#define XPM_PROBE_COUNTER_TYPE_MAIN_CTL(1U)
```

Description

Define XPM_PROBE_COUNTER_TYPE_CFG_CTL

Definition

```
#define XPM_PROBE_COUNTER_TYPE_CFG_CTL(2U)
```

Description

Define XPM_PROBE_COUNTER_TYPE_STATE_PERIOD

Definition

```
#define XPM_PROBE_COUNTER_TYPE_STATE_PERIOD(3U)
```

Description

Define XPM_PROBE_COUNTER_TYPE_PORT_SEL

Definition

```
#define XPM_PROBE_COUNTER_TYPE_PORT_SEL(4U)
```

Description

Define XPM_PROBE_COUNTER_TYPE_SRC

Definition

```
#define XPM_PROBE_COUNTER_TYPE_SRC(5U)
```

Description

Define XPM_PROBE_COUNTER_TYPE_VAL

Definition

```
#define XPM_PROBE_COUNTER_TYPE_VAL(6U)
```

Description

Define XST_API_BASE_VERSION

Definition

```
#define XST_API_BASE_VERSION(1U)
```

Description

Define XST_API_QUERY_DATA_VERSION

Definition

```
#define XST_API_QUERY_DATA_VERSION(2U)
```

Description

Define PM_GET_API_VERSION

Definition

```
#define PM_GET_API_VERSION1U
```

Description

Define PM_SET_CONFIGURATION

Definition

```
#define PM_SET_CONFIGURATION2U
```

Description

Define PM_GET_NODE_STATUS

Definition

```
#define PM_GET_NODE_STATUS3U
```

Description

Define PM_GET_OP_CHARACTERISTIC

Definition

```
#define PM_GET_OP_CHARACTERISTIC4U
```

Description

Define PM_REGISTER_NOTIFIER

Definition

```
#define PM_REGISTER_NOTIFIER5U
```

Description

Define PM_REQUEST_SUSPEND

Definition

```
#define PM_REQUEST_SUSPEND6U
```

Description

Define PM_SELF_SUSPEND

Definition

```
#define PM_SELF_SUSPEND7U
```

Description

Define PM_FORCE_POWERDOWN

Definition

```
#define PM_FORCE_POWERDOWN8U
```

Description

Define PM_ABORT_SUSPEND

Definition

```
#define PM_ABORT_SUSPEND9U
```

Description

Define PM_REQUEST_WAKEUP

Definition

```
#define PM_REQUEST_WAKEUP10U
```

Description

Define PM_SET_WAKEUP_SOURCE

Definition

```
#define PM_SET_WAKEUP_SOURCE11U
```

Description

Define PM_SYSTEM_SHUTDOWN

Definition

```
#define PM_SYSTEM_SHUTDOWN12U
```

Description

Define PM_REQUEST_NODE

Definition

```
#define PM_REQUEST_NODE13U
```

Description

Define PM_RELEASE_NODE

Definition

```
#define PM_RELEASE_NODE14U
```

Description

Define PM_SET_REQUIREMENT

Definition

```
#define PM_SET_REQUIREMENT15U
```

Description

Define PM_SET_MAX_LATENCY

Definition

```
#define PM_SET_MAX_LATENCY16U
```

Description

Define PM_RESET_ASSERT

Definition

```
#define PM_RESET_ASSERT17U
```

Description

Define PM_RESET_GET_STATUS

Definition

```
#define PM_RESET_GET_STATUS18U
```

Description

Define PM_MMIO_WRITE

Definition

```
#define PM_MMIO_WRITE19U
```

Description

Define PM_MMIO_READ

Definition

```
#define PM_MMIO_READ20U
```

Description

Define PM_INIT_FINALIZE

Definition

```
#define PM_INIT_FINALIZE21U
```

Description

Define PM_FPGA_LOAD

Definition

```
#define PM_FPGA_LOAD22U
```

Description

Define PM_FPGA_GET_STATUS

Definition

```
#define PM_FPGA_GET_STATUS23U
```

Description

Define PM_GET_CHIPID

Definition

```
#define PM_GET_CHIPID24U
```

Description

Define PM_SECURE_RSA_AES

Definition

```
#define PM_SECURE_RSA_AES25U
```

Description

Define PM_SECURE_SHA

Definition

```
#define PM_SECURE_SHA26U
```

Description

Define PM_SECURE_RSA

Definition

```
#define PM_SECURE_RSA27U
```

Description

Define PM_PINCTRL_REQUEST

Definition

```
#define PM_PINCTRL_REQUEST28U
```

Description

Define PM_PINCTRL_RELEASE

Definition

```
#define PM_PINCTRL_RELEASE29U
```

Description

Define PM_PINCTRL_GET_FUNCTION

Definition

```
#define PM_PINCTRL_GET_FUNCTION30U
```

Description

Define PM_PINCTRL_SET_FUNCTION

Definition

```
#define PM_PINCTRL_SET_FUNCTION31U
```

Description

Define PM_PINCTRL_CONFIG_PARAM_GET

Definition

```
#define PM_PINCTRL_CONFIG_PARAM_GET32U
```

Description

Define PM_PINCTRL_CONFIG_PARAM_SET

Definition

```
#define PM_PINCTRL_CONFIG_PARAM_SET33U
```

Description

Define PM_IOCTL

Definition

```
#define PM_IOCTL34U
```

Description

Define PM_QUERY_DATA

Definition

```
#define PM_QUERY_DATA35U
```

Description

Define PM_CLOCK_ENABLE

Definition

```
#define PM_CLOCK_ENABLE36U
```

Description

Define PM_CLOCK_DISABLE

Definition

```
#define PM_CLOCK_DISABLE37U
```

Description

Define PM_CLOCK_GETSTATE

Definition

```
#define PM_CLOCK_GETSTATE38U
```

Description

Define PM_CLOCK_SETDIVIDER

Definition

```
#define PM_CLOCK_SETDIVIDER39U
```

Description

Define PM_CLOCK_GETDIVIDER

Definition

```
#define PM_CLOCK_GETDIVIDER40U
```

Description

Define PM_CLOCK_SETRATE

Definition

```
#define PM_CLOCK_SETRATE41U
```

Description

Define PM_CLOCK_GETRATE

Definition

```
#define PM_CLOCK_GETRATE42U
```

Description

Define PM_CLOCK_SETPARENT

Definition

```
#define PM_CLOCK_SETPARENT43U
```

Description

Define PM_CLOCK_GETPARENT

Definition

```
#define PM_CLOCK_GETPARENT44U
```

Description

Define PM_SECURE_IMAGE

Definition

```
#define PM_SECURE_IMAGE45U
```

Description

Define PM_FPGA_READ

Definition

```
#define PM_FPGA_READ46U
```

Description

Define PM_PLL_SET_PARAMETER

Definition

```
#define PM_PLL_SET_PARAMETER48U
```

Description

Define PM_PLL_GET_PARAMETER

Definition

```
#define PM_PLL_GET_PARAMETER49U
```

Description

Define PM_PLL_SET_MODE

Definition

```
#define PM_PLL_SET_MODE50U
```

Description

Define PM_PLL_GET_MODE

Definition

```
#define PM_PLL_GET_MODE51U
```

Description

Define PM_REGISTER_ACCESS

Definition

```
#define PM_REGISTER_ACCESS52U
```

Description

Define PM_EFUSE_ACCESS

Definition

```
#define PM_EFUSE_ACCESS53U
```

Description

Define PM_ADD_SUBSYSTEM

Definition

```
#define PM_ADD_SUBSYSTEM54U
```

Description

Define PM_DESTROY_SUBSYSTEM

Definition

```
#define PM_DESTROY_SUBSYSTEM55U
```

Description

Define PM_DESCRIBE_NODES

Definition

```
#define PM_DESCRIBE_NODES56U
```

Description

Define PM_ADD_NODE

Definition

```
#define PM_ADD_NODE57U
```

Description

Define PM_ADD_NODE_PARENT

Definition

```
#define PM_ADD_NODE_PARENT58U
```

Description

Define PM_ADD_NODE_NAME

Definition

```
#define PM_ADD_NODE_NAME59U
```

Description

Define PM_ADD_REQUIREMENT

Definition

```
#define PM_ADD_REQUIREMENT60U
```

Description

Define PM_SET_CURRENT_SUBSYSTEM

Definition

```
#define PM_SET_CURRENT_SUBSYSTEM61U
```

Description

Define PM_INIT_NODE

Definition

```
#define PM_INIT_NODE62U
```

Description

Define PM_FEATURE_CHECK

Definition

```
#define PM_FEATURE_CHECK63U
```

Description

Define PM_ISO_CONTROL

Definition

```
#define PM_ISO_CONTROL64U
```

Description

Define PM_ACTIVATE_SUBSYSTEM

Definition

```
#define PM_ACTIVATE_SUBSYSTEM65U
```

Description

Define PM_API_MIN

Definition

```
#define PM_API_MINPM_GET_API_VERSION
```

Description

Define PM_API_MAX

Definition

```
#define PM_API_MAXPM_ISO_CONTROL
```

Description

Define PACK_PAYLOAD

Definition

```
#define PACK_PAYLOADPayload[0] = (u32)Arg0;                              \
    Payload[1] = (u32)Arg1;                                 \
    Payload[2] = (u32)Arg2;                                 \
    Payload[3] = (u32)Arg3;                                 \
    Payload[4] = (u32)Arg4;                                 \
    Payload[5] = (u32)Arg5;                                 \
    XPm_Dbg("%s(%x, %x, %x, %x, %x)\r\n", __func__, Arg1, Arg2, Arg3, Arg4,
Arg5);
```

Description

Define LIBPM_MODULE_ID

Definition

```
#define LIBPM_MODULE_ID(0x02UL)
```

Description

Define HEADER

Definition

```
#define HEADER((len << 16U) | (LIBPM_MODULE_ID << 8U) | ((u32)ApiId))
```

Description

Define PACK_PAYLOAD0

Definition

```
#define PACK_PAYLOAD0PACK_PAYLOAD(Payload, HEADER(0UL, ApiId), 0, 0, 0, 0,
0)
```

Description

Define PACK_PAYLOAD1

Definition

```
#define PACK_PAYLOAD1PACK_PAYLOAD(Payload, HEADER(1UL, ApiId), Arg1, 0, 0,
0, 0)
```

Description

Define PACK_PAYLOAD2

Definition

```
#define PACK_PAYLOAD2PACK_PAYLOAD(Payload, HEADER(2UL, ApiId), Arg1, Arg2,
0, 0, 0)
```

Description

Define PACK_PAYLOAD3

Definition

```
#define PACK_PAYLOAD3PACK_PAYLOAD(Payload, HEADER(3UL, ApiId), Arg1, Arg2,
Arg3, 0, 0)
```

Description

Define PACK_PAYLOAD4

Definition

```
#define PACK_PAYLOAD4PACK_PAYLOAD(Payload, HEADER(4UL, ApiId), Arg1, Arg2,
Arg3, Arg4, 0)
```

Description

Define PACK_PAYLOAD5

Definition

```
#define PACK_PAYLOAD5PACK_PAYLOAD(Payload, HEADER(5UL, ApiId), Arg1, Arg2,
Arg3, Arg4, Arg5)
```

Description

Power Nodes

Definitions

Define PM_POWER_PMC

Definition

```
#define PM_POWER_PMC(0x4208001U)
```

Description

Define PM_POWER_LPD

Definition

```
#define PM_POWER_LPD(0x4210002U)
```

Description

Define PM_POWER_FPD

Definition

```
#define PM_POWER_FPD(0x420c003U)
```

Description

Define PM_POWER_NOC

Definition

```
#define PM_POWER_NOC(0x4214004U)
```

Description

Define PM_POWER_ME

Definition

```
#define PM_POWER_ME(0x421c005U)
```

Description

Define PM_POWER_PLD

Definition

```
#define PM_POWER_PLD(0x4220006U)
```

Description

Define PM_POWER_CPM

Definition

```
#define PM_POWER_CPM(0x4218007U)
```

Description

Define PM_POWER_PL_SYSMON

Definition

```
#define PM_POWER_PL_SYSMON(0x4208008U)
```

Description

Define PM_POWER_RPU0_0

Definition

```
#define PM_POWER_RPU0_0(0x4104009U)
```

Description

Define PM_POWER_GEM0

Definition

```
#define PM_POWER_GEM0(0x410400aU)
```

Description

Define PM_POWER_GEM1

Definition

```
#define PM_POWER_GEM1(0x410400bU)
```

Description

Define PM_POWER_OCM_0

Definition

```
#define PM_POWER_OCM_0(0x410400cU)
```

Description

Define PM_POWER_OCM_1

Definition

```
#define PM_POWER_OCM_1(0x410400dU)
```

Description

Define PM_POWER_OCM_2

Definition

```
#define PM_POWER_OCM_2(0x410400eU)
```

Description

Define PM_POWER_OCM_3

Definition

```
#define PM_POWER_OCM_3(0x410400fU)
```

Description

Define PM_POWER_TCM_0_A

Definition

```
#define PM_POWER_TCM_0_A(0x4104010U)
```

Description

Define PM_POWER_TCM_0_B

Definition

```
#define PM_POWER_TCM_0_B(0x4104011U)
```

Description

Define PM_POWER_TCM_1_A

Definition

```
#define PM_POWER_TCM_1_A(0x4104012U)
```

Description

Define PM_POWER_TCM_1_B

Definition

```
#define PM_POWER_TCM_1_B(0x4104013U)
```

Description

Define PM_POWER_ACPU_0

Definition

```
#define PM_POWER_ACPU_0(0x4104014U)
```

Description

Define PM_POWER_ACPU_1

Definition

```
#define PM_POWER_ACPU_1(0x4104015U)
```

Description

Define PM_POWER_L2_BANK_0

Definition

```
#define PM_POWER_L2_BANK_0(0x4104016U)
```

Description

Clock nodes

Definitions

Define PM_CLK_PMC_PLL

Definition

```
#define PM_CLK_PMC_PLL(0x8104001U)
```

Description

Define PM_CLK_APU_PLL

Definition

```
#define PM_CLK_APU_PLL(0x8104002U)
```

Description

Define PM_CLK_RPU_PLL

Definition

```
#define PM_CLK_RPU_PLL(0x8104003U)
```

Description

Define PM_CLK_CPM_PLL

Definition

```
#define PM_CLK_CPM_PLL(0x8104004U)
```

Description

Define PM_CLK_NOC_PLL

Definition

```
#define PM_CLK_NOC_PLL(0x8104005U)
```

Description

Define PM_CLK_PMC_PRESRC

Definition

```
#define PM_CLK_PMC_PRESRC(0x8208007U)
```

Description

Define PM_CLK_PMC_POSTCLK

Definition

```
#define PM_CLK_PMC_POSTCLK(0x8208008U)
```

Description

Define PM_CLK_PMC_PLL_OUT

Definition

```
#define PM_CLK_PMC_PLL_OUT(0x8208009U)
```

Description

Define PM_CLK_PPLL

Definition

```
#define PM_CLK_PPLL(0x820800aU)
```

Description

Define PM_CLK_NOC_PRESRC

Definition

```
#define PM_CLK_NOC_PRESRC(0x820800bU)
```

Description

Define PM_CLK_NOC_POSTCLK

Definition

```
#define PM_CLK_NOC_POSTCLK(0x820800cU)
```

Description

Define PM_CLK_NOC_PLL_OUT

Definition

```
#define PM_CLK_NOC_PLL_OUT(0x820800dU)
```

Description

Define PM_CLK_NPLL

Definition

```
#define PM_CLK_NPLL(0x820800eU)
```

Description

Define PM_CLK_APU_PRESRC

Definition

```
#define PM_CLK_APU_PRESRC(0x820800fU)
```

Description

Define PM_CLK_APU_POSTCLK

Definition

```
#define PM_CLK_APU_POSTCLK(0x8208010U)
```

Description

Define PM_CLK_APU_PLL_OUT

Definition

```
#define PM_CLK_APU_PLL_OUT(0x8208011U)
```

Description

Define PM_CLK_APLL

Definition

```
#define PM_CLK_APLL(0x8208012U)
```

Description

Define PM_CLK_RPU_PRESRC

Definition

```
#define PM_CLK_RPU_PRESRC(0x8208013U)
```

Description

Define PM_CLK_RPU_POSTCLK

Definition

```
#define PM_CLK_RPU_POSTCLK(0x8208014U)
```

Description

Define PM_CLK_RPU_PLL_OUT

Definition

```
#define PM_CLK_RPU_PLL_OUT(0x8208015U)
```

Description

Define PM_CLK_RPLL

Definition

```
#define PM_CLK_RPLL(0x8208016U)
```

Description

Define PM_CLK_CPM_PRESRC

Definition

```
#define PM_CLK_CPM_PRESRC(0x8208017U)
```

Description

Define PM_CLK_CPM_POSTCLK

Definition

```
#define PM_CLK_CPM_POSTCLK(0x8208018U)
```

Description

Define PM_CLK_CPM_PLL_OUT

Definition

```
#define PM_CLK_CPM_PLL_OUT(0x8208019U)
```

Description

Define PM_CLK_CPLL

Definition

```
#define PM_CLK_CPLL(0x820801aU)
```

Description

Define PM_CLK_PPLL_TO_XPD

Definition

```
#define PM_CLK_PPLL_TO_XPD(0x820801bU)
```

Description

Define PM_CLK_NPLL_TO_XPD

Definition

```
#define PM_CLK_NPLL_TO_XPD(0x820801cU)
```

Description

Define PM_CLK_APLL_TO_XPD

Definition

```
#define PM_CLK_APLL_TO_XPD(0x820801dU)
```

Description

Define PM_CLK_RPLL_TO_XPD

Definition

```
#define PM_CLK_RPLL_TO_XPD(0x820801eU)
```

Description

Define PM_CLK_EFUSE_REF

Definition

```
#define PM_CLK_EFUSE_REF(0x820801fU)
```

Description

Define PM_CLK_SYSMON_REF

Definition

```
#define PM_CLK_SYSMON_REF(0x8208020U)
```

Description

Define PM_CLK_IRO_SUSPEND_REF

Definition

```
#define PM_CLK_IRO_SUSPEND_REF(0x8208021U)
```

Description

Define PM_CLK_USB_SUSPEND

Definition

```
#define PM_CLK_USB_SUSPEND(0x8208022U)
```

Description

Define PM_CLK_SWITCH_TIMEOUT

Definition

```
#define PM_CLK_SWITCH_TIMEOUT(0x8208023U)
```

Description

Define PM_CLK_RCLK_PMC

Definition

```
#define PM_CLK_RCLK_PMC(0x8208024U)
```

Description

Define PM_CLK_RCLK_LPD

Definition

```
#define PM_CLK_RCLK_LPD(0x8208025U)
```

Description

Define PM_CLK_WDT

Definition

```
#define PM_CLK_WDT(0x8208026U)
```

Description

Define PM_CLK_TTC0

Definition

```
#define PM_CLK_TTC0(0x8208027U)
```

Description

Define PM_CLK_TTC1

Definition

```
#define PM_CLK_TTC1(0x8208028U)
```

Description

Define PM_CLK_TTC2

Definition

```
#define PM_CLK_TTC2(0x8208029U)
```

Description

Define PM_CLK_TTC3

Definition

```
#define PM_CLK_TTC3(0x820802aU)
```

Description

Define PM_CLK_GEM_TSU

Definition

```
#define PM_CLK_GEM_TSU(0x820802bU)
```

Description

Define PM_CLK_GEM_TSU_LB

Definition

```
#define PM_CLK_GEM_TSU_LB(0x820802cU)
```

Description

Define PM_CLK_MUXED_IRO_DIV2

Definition

```
#define PM_CLK_MUXED_IRO_DIV2(0x820802dU)
```

Description

Define PM_CLK_MUXED_IRO_DIV4

Definition

```
#define PM_CLK_MUXED_IRO_DIV4(0x820802eU)
```

Description

Define PM_CLK_PSM_REF

Definition

```
#define PM_CLK_PSM_REF(0x820802fU)
```

Description

Define PM_CLK_GEM0_RX

Definition

```
#define PM_CLK_GEM0_RX(0x8208030U)
```

Description

Define PM_CLK_GEM0_TX

Definition

```
#define PM_CLK_GEM0_TX(0x8208031U)
```

Description

Define PM_CLK_GEM1_RX

Definition

```
#define PM_CLK_GEM1_RX(0x8208032U)
```

Description

Define PM_CLK_GEM1_TX

Definition

```
#define PM_CLK_GEM1_TX(0x8208033U)
```

Description

Define PM_CLK_CPM_CORE_REF

Definition

```
#define PM_CLK_CPM_CORE_REF(0x8208034U)
```

Description

Define PM_CLK_CPM_LSBUS_REF

Definition

```
#define PM_CLK_CPM_LSBUS_REF(0x8208035U)
```

Description

Define PM_CLK_CPM_DBG_REF

Definition

```
#define PM_CLK_CPM_DBG_REF(0x8208036U)
```

Description

Define PM_CLK_CPM_AUX0_REF

Definition

```
#define PM_CLK_CPM_AUX0_REF(0x8208037U)
```

Description

Define PM_CLK_CPM_AUX1_REF

Definition

```
#define PM_CLK_CPM_AUX1_REF(0x8208038U)
```

Description

Define PM_CLK_QSPI_REF

Definition

```
#define PM_CLK_QSPI_REF(0x8208039U)
```

Description

Define PM_CLK_OSPI_REF

Definition

```
#define PM_CLK_OSPI_REF(0x820803aU)
```

Description

Define PM_CLK_SDIO0_REF

Definition

```
#define PM_CLK_SDIO0_REF(0x820803bU)
```

Description

Define PM_CLK_SDIO1_REF

Definition

```
#define PM_CLK_SDIO1_REF(0x820803cU)
```

Description

Define PM_CLK_PMC_LSBUS_REF

Definition

```
#define PM_CLK_PMC_LSBUS_REF(0x820803dU)
```

Description

Define PM_CLK_I2C_REF

Definition

```
#define PM_CLK_I2C_REF(0x820803eU)
```

Description

Define PM_CLK_TEST_PATTERN_REF

Definition

```
#define PM_CLK_TEST_PATTERN_REF(0x820803fU)
```

Description

Define PM_CLK_DFT_OSC_REF

Definition

```
#define PM_CLK_DFT_OSC_REF(0x8208040U)
```

Description

Define PM_CLK_PMC_PL0_REF

Definition

```
#define PM_CLK_PMC_PL0_REF(0x8208041U)
```

Description

Define PM_CLK_PMC_PL1_REF

Definition

```
#define PM_CLK_PMC_PL1_REF(0x8208042U)
```

Description

Define PM_CLK_PMC_PL2_REF

Definition

```
#define PM_CLK_PMC_PL2_REF(0x8208043U)
```

Description

Define PM_CLK_PMC_PL3_REF

Definition

```
#define PM_CLK_PMC_PL3_REF(0x8208044U)
```

Description

Define PM_CLK_CFU_REF

Definition

```
#define PM_CLK_CFU_REF(0x8208045U)
```

Description

Define PM_CLK_SPARE_REF

Definition

```
#define PM_CLK_SPARE_REF(0x8208046U)
```

Description

Define PM_CLK_NPI_REF

Definition

```
#define PM_CLK_NPI_REF(0x8208047U)
```

Description

Define PM_CLK_HSM0_REF

Definition

```
#define PM_CLK_HSM0_REF(0x8208048U)
```

Description

Define PM_CLK_HSM1_REF

Definition

```
#define PM_CLK_HSM1_REF(0x8208049U)
```

Description

Define PM_CLK_SD_DLL_REF

Definition

```
#define PM_CLK_SD_DLL_REF(0x820804aU)
```

Description

Define PM_CLK_FPD_TOP_SWITCH

Definition

```
#define PM_CLK_FPD_TOP_SWITCH(0x820804bU)
```

Description

Define PM_CLK_FPD_LSBUS

Definition

```
#define PM_CLK_FPD_LSBUS(0x820804cU)
```

Description

Define PM_CLK_ACPU

Definition

```
#define PM_CLK_ACPU(0x820804dU)
```

Description

Define PM_CLK_DBG_TRACE

Definition

```
#define PM_CLK_DBG_TRACE(0x820804eU)
```

Description

Define PM_CLK_DBG_FPD

Definition

```
#define PM_CLK_DBG_FPD(0x820804fU)
```

Description

Define PM_CLK_LPD_TOP_SWITCH

Definition

```
#define PM_CLK_LPD_TOP_SWITCH(0x8208050U)
```

Description

Define PM_CLK_ADMA

Definition

```
#define PM_CLK_ADMA(0x8208051U)
```

Description

Define PM_CLK_LPD_LSBUS

Definition

```
#define PM_CLK_LPD_LSBUS(0x8208052U)
```

Description

Define PM_CLK_CPU_R5

Definition

```
#define PM_CLK_CPU_R5(0x8208053U)
```

Description

Define PM_CLK_CPU_R5_CORE

Definition

```
#define PM_CLK_CPU_R5_CORE(0x8208054U)
```

Description

Define PM_CLK_CPU_R5_OCM

Definition

```
#define PM_CLK_CPU_R5_OCM(0x8208055U)
```

Description

Define PM_CLK_CPU_R5_OCM2

Definition

```
#define PM_CLK_CPU_R5_OCM2(0x8208056U)
```

Description

Define PM_CLK_IOU_SWITCH

Definition

```
#define PM_CLK_IOU_SWITCH(0x8208057U)
```

Description

Define PM_CLK_GEM0_REF

Definition

```
#define PM_CLK_GEM0_REF(0x8208058U)
```

Description

Define PM_CLK_GEM1_REF

Definition

```
#define PM_CLK_GEM1_REF(0x8208059U)
```

Description

Define PM_CLK_GEM_TSU_REF

Definition

```
#define PM_CLK_GEM_TSU_REF(0x820805aU)
```

Description

Define PM_CLK_USB0_BUS_REF

Definition

```
#define PM_CLK_USB0_BUS_REF(0x820805bU)
```

Description

Define PM_CLK_UART0_REF

Definition

```
#define PM_CLK_UART0_REF(0x820805cU)
```

Description

Define PM_CLK_UART1_REF

Definition

```
#define PM_CLK_UART1_REF(0x820805dU)
```

Description

Define PM_CLK_SPI0_REF

Definition

```
#define PM_CLK_SPI0_REF(0x820805eU)
```

Description

Define PM_CLK_SPI1_REF

Definition

```
#define PM_CLK_SPI1_REF(0x820805fU)
```

Description

Define PM_CLK_CAN0_REF

Definition

```
#define PM_CLK_CAN0_REF(0x8208060U)
```

Description

Define PM_CLK_CAN1_REF

Definition

```
#define PM_CLK_CAN1_REF(0x8208061U)
```

Description

Define PM_CLK_I2C0_REF

Definition

```
#define PM_CLK_I2C0_REF(0x8208062U)
```

Description

Define PM_CLK_I2C1_REF

Definition

```
#define PM_CLK_I2C1_REF(0x8208063U)
```

Description

Define PM_CLK_DBG_LPD

Definition

```
#define PM_CLK_DBG_LPD(0x8208064U)
```

Description

Define PM_CLK_TIMESTAMP_REF

Definition

```
#define PM_CLK_TIMESTAMP_REF(0x8208065U)
```

Description

Define PM_CLK_DBG_TSTMP

Definition

```
#define PM_CLK_DBG_TSTMP(0x8208066U)
```

Description

Define PM_CLK_CPM_TOPSW_REF

Definition

```
#define PM_CLK_CPM_TOPSW_REF(0x8208067U)
```

Description

Define PM_CLK_USB3_DUAL_REF

Definition

```
#define PM_CLK_USB3_DUAL_REF(0x8208068U)
```

Description

Define PM_CLK_REF_CLK

Definition

```
#define PM_CLK_REF_CLK(0x830c06aU)
```

Description

Define PM_CLK_PL_ALT_REF_CLK

Definition

```
#define PM_CLK_PL_ALT_REF_CLK(0x830c06bU)
```

Description

Define PM_CLK_MUXED_IRO

Definition

```
#define PM_CLK_MUXED_IRO(0x830c06cU)
```

Description

Define PM_CLK_PL_EXT

Definition

```
#define PM_CLK_PL_EXT(0x830c06dU)
```

Description

Define PM_CLK_PL_LB

Definition

```
#define PM_CLK_PL_LB(0x830c06eU)
```

Description

Define PM_CLK_MIO_50_OR_51

Definition

```
#define PM_CLK_MIO_50_OR_51(0x830c06fU)
```

Description

Define PM_CLK_MIO_24_OR_25

Definition

```
#define PM_CLK_MIO_24_OR_25(0x830c070U)
```

Description

Define PM_CLK_EMIO

Definition

```
#define PM_CLK_EMIO(0x830c071U)
```

Description

Define PM_CLK_MIO

Definition

```
#define PM_CLK_MIO(0x830c072U)
```

Description

Define PM_CLK_PL_PMC_ALT_REF_CLK

Definition

```
#define PM_CLK_PL_PMC_ALT_REF_CLK(0x830c076U)
```

Description

Define PM_CLK_PL_LPD_ALT_REF_CLK

Definition

```
#define PM_CLK_PL_LPD_ALT_REF_CLK(0x830c077U)
```

Description

Define PM_CLK_PL_FPD_ALT_REF_CLK

Definition

```
#define PM_CLK_PL_FPD_ALT_REF_CLK(0x830c078U)
```

Description

MIO nodes

Definitions

Define PM_STMIC_LMIO_0

Definition

```
#define PM_STMIC_LMIO_0(0x14104001U)
```

Description

Define PM_STMIC_LMIO_1

Definition

```
#define PM_STMIC_LMIO_1(0x14104002U)
```

Description

Define PM_STMIC_LMIO_2

Definition

```
#define PM_STMIC_LMIO_2(0x14104003U)
```

Description

Define PM_STMIC_LMIO_3

Definition

```
#define PM_STMIC_LMIO_3(0x14104004U)
```

Description

Define PM_STMIC_LMIO_4

Definition

```
#define PM_STMIC_LMIO_4(0x14104005U)
```

Description

Define PM_STMIC_LMIO_5

Definition

```
#define PM_STMIC_LMIO_5(0x14104006U)
```

Description

Define PM_STMIC_LMIO_6

Definition

```
#define PM_STMIC_LMIO_6(0x14104007U)
```

Description

Define PM_STMIC_LMIO_7

Definition

```
#define PM_STMIC_LMIO_7(0x14104008U)
```

Description

Define PM_STMIC_LMIO_8

Definition

```
#define PM_STMIC_LMIO_8(0x14104009U)
```

Description

Define PM_STMIC_LMIO_9

Definition

```
#define PM_STMIC_LMIO_9(0x1410400aU)
```

Description

Define PM_STMIC_LMIO_10

Definition

```
#define PM_STMIC_LMIO_10(0x1410400bU)
```

Description

Define PM_STMIC_LMIO_11

Definition

```
#define PM_STMIC_LMIO_11(0x1410400cU)
```

Description

Define PM_STMIC_LMIO_12

Definition

```
#define PM_STMIC_LMIO_12(0x1410400dU)
```

Description

Define PM_STMIC_LMIO_13

Definition

```
#define PM_STMIC_LMIO_13(0x1410400eU)
```

Description

Define PM_STMIC_LMIO_14

Definition

```
#define PM_STMIC_LMIO_14(0x1410400fU)
```

Description

Define PM_STMIC_LMIO_15

Definition

```
#define PM_STMIC_LMIO_15(0x14104010U)
```

Description

Define PM_STMIC_LMIO_16

Definition

```
#define PM_STMIC_LMIO_16(0x14104011U)
```

Description

Define PM_STMIC_LMIO_17

Definition

```
#define PM_STMIC_LMIO_17(0x14104012U)
```

Description

Define PM_STMIC_LMIO_18

Definition

```
#define PM_STMIC_LMIO_18(0x14104013U)
```

Description

Define PM_STMIC_LMIO_19

Definition

```
#define PM_STMIC_LMIO_19(0x14104014U)
```

Description

Define PM_STMIC_LMIO_20

Definition

```
#define PM_STMIC_LMIO_20(0x14104015U)
```

Description

Define PM_STMIC_LMIO_21

Definition

```
#define PM_STMIC_LMIO_21(0x14104016U)
```

Description

Define PM_STMIC_LMIO_22

Definition

```
#define PM_STMIC_LMIO_22(0x14104017U)
```

Description

Define PM_STMIC_LMIO_23

Definition

```
#define PM_STMIC_LMIO_23(0x14104018U)
```

Description

Define PM_STMIC_LMIO_24

Definition

```
#define PM_STMIC_LMIO_24(0x14104019U)
```

Description

Define PM_STMIC_LMIO_25

Definition

```
#define PM_STMIC_LMIO_25(0x1410401aU)
```

Description

Define PM_STMIC_PMIO_0

Definition

```
#define PM_STMIC_PMIO_0(0x1410801bU)
```

Description

Define PM_STMIC_PMIO_1

Definition

```
#define PM_STMIC_PMIO_1(0x1410801cU)
```

Description

Define PM_STMIC_PMIO_2

Definition

```
#define PM_STMIC_PMIO_2(0x1410801dU)
```

Description

Define PM_STMIC_PMIO_3

Definition

```
#define PM_STMIC_PMIO_3(0x1410801eU)
```

Description

Define PM_STMIC_PMIO_4

Definition

```
#define PM_STMIC_PMIO_4(0x1410801fU)
```

Description

Define PM_STMIC_PMIO_5

Definition

```
#define PM_STMIC_PMIO_5(0x14108020U)
```

Description

Define PM_STMIC_PMIO_6

Definition

```
#define PM_STMIC_PMIO_6(0x14108021U)
```

Description

Define PM_STMIC_PMIO_7

Definition

```
#define PM_STMIC_PMIO_7(0x14108022U)
```

Description

Define PM_STMIC_PMIO_8

Definition

```
#define PM_STMIC_PMIO_8(0x14108023U)
```

Description

Define PM_STMIC_PMIO_9

Definition

```
#define PM_STMIC_PMIO_9(0x14108024U)
```

Description

Define PM_STMIC_PMIO_10

Definition

```
#define PM_STMIC_PMIO_10(0x14108025U)
```

Description

Define PM_STMIC_PMIO_11

Definition

```
#define PM_STMIC_PMIO_11(0x14108026U)
```

Description

Define PM_STMIC_PMIO_12

Definition

```
#define PM_STMIC_PMIO_12(0x14108027U)
```

Description

Define PM_STMIC_PMIO_13

Definition

```
#define PM_STMIC_PMIO_13(0x14108028U)
```

Description

Define PM_STMIC_PMIO_14

Definition

```
#define PM_STMIC_PMIO_14(0x14108029U)
```

Description

Define PM_STMIC_PMIO_15

Definition

```
#define PM_STMIC_PMIO_15(0x1410802aU)
```

Description

Define PM_STMIC_PMIO_16

Definition

```
#define PM_STMIC_PMIO_16(0x1410802bU)
```

Description

Define PM_STMIC_PMIO_17

Definition

```
#define PM_STMIC_PMIO_17(0x1410802cU)
```

Description

Define PM_STMIC_PMIO_18

Definition

```
#define PM_STMIC_PMIO_18(0x1410802dU)
```

Description

Define PM_STMIC_PMIO_19

Definition

```
#define PM_STMIC_PMIO_19(0x1410802eU)
```

Description

Define PM_STMIC_PMIO_20

Definition

```
#define PM_STMIC_PMIO_20(0x1410802fU)
```

Description

Define PM_STMIC_PMIO_21

Definition

```
#define PM_STMIC_PMIO_21(0x14108030U)
```

Description

Define PM_STMIC_PMIO_22

Definition

```
#define PM_STMIC_PMIO_22(0x14108031U)
```

Description

Define PM_STMIC_PMIO_23

Definition

```
#define PM_STMIC_PMIO_23(0x14108032U)
```

Description

Define PM_STMIC_PMIO_24

Definition

```
#define PM_STMIC_PMIO_24(0x14108033U)
```

Description

Define PM_STMIC_PMIO_25

Definition

```
#define PM_STMIC_PMIO_25(0x14108034U)
```

Description

Define PM_STMIC_PMIO_26

Definition

```
#define PM_STMIC_PMIO_26(0x14108035U)
```

Description

Define PM_STMIC_PMIO_27

Definition

```
#define PM_STMIC_PMIO_27(0x14108036U)
```

Description

Define PM_STMIC_PMIO_28

Definition

```
#define PM_STMIC_PMIO_28(0x14108037U)
```

Description

Define PM_STMIC_PMIO_29

Definition

```
#define PM_STMIC_PMIO_29(0x14108038U)
```

Description

Define PM_STMIC_PMIO_30

Definition

```
#define PM_STMIC_PMIO_30(0x14108039U)
```

Description

Define PM_STMIC_PMIO_31

Definition

```
#define PM_STMIC_PMIO_31(0x1410803aU)
```

Description

Define PM_STMIC_PMIO_32

Definition

```
#define PM_STMIC_PMIO_32(0x1410803bU)
```

Description

Define PM_STMIC_PMIO_33

Definition

```
#define PM_STMIC_PMIO_33(0x1410803cU)
```

Description

Define PM_STMIC_PMIO_34

Definition

```
#define PM_STMIC_PMIO_34(0x1410803dU)
```

Description

Define PM_STMIC_PMIO_35

Definition

```
#define PM_STMIC_PMIO_35(0x1410803eU)
```

Description

Define PM_STMIC_PMIO_36

Definition

```
#define PM_STMIC_PMIO_36(0x1410803fU)
```

Description

Define PM_STMIC_PMIO_37

Definition

```
#define PM_STMIC_PMIO_37(0x14108040U)
```

Description

Define PM_STMIC_PMIO_38

Definition

```
#define PM_STMIC_PMIO_38(0x14108041U)
```

Description

Define PM_STMIC_PMIO_39

Definition

```
#define PM_STMIC_PMIO_39(0x14108042U)
```

Description

Define PM_STMIC_PMIO_40

Definition

```
#define PM_STMIC_PMIO_40(0x14108043U)
```

Description

Define PM_STMIC_PMIO_41

Definition

```
#define PM_STMIC_PMIO_41(0x14108044U)
```

Description

Define PM_STMIC_PMIO_42

Definition

```
#define PM_STMIC_PMIO_42(0x14108045U)
```

Description

Define PM_STMIC_PMIO_43

Definition

```
#define PM_STMIC_PMIO_43(0x14108046U)
```

Description

Define PM_STMIC_PMIO_44

Definition

```
#define PM_STMIC_PMIO_44(0x14108047U)
```

Description

Define PM_STMIC_PMIO_45

Definition

```
#define PM_STMIC_PMIO_45(0x14108048U)
```

Description

Define PM_STMIC_PMIO_46

Definition

```
#define PM_STMIC_PMIO_46(0x14108049U)
```

Description

Define PM_STMIC_PMIO_47

Definition

```
#define PM_STMIC_PMIO_47(0x1410804aU)
```

Description

Define PM_STMIC_PMIO_48

Definition

```
#define PM_STMIC_PMIO_48(0x1410804bU)
```

Description

Define PM_STMIC_PMIO_49

Definition

```
#define PM_STMIC_PMIO_49(0x1410804cU)
```

Description

Define PM_STMIC_PMIO_50

Definition

```
#define PM_STMIC_PMIO_50(0x1410804dU)
```

Description

Define PM_STMIC_PMIO_51

Definition

```
#define PM_STMIC_PMIO_51(0x1410804eU)
```

Description

Device nodes

Definitions

Define PM_DEV_PLD_0

Definition

```
#define PM_DEV_PLD_0(0x18700000U)
```

Description

Define PM_DEV_PMC_PROC

Definition

```
#define PM_DEV_PMC_PROC(0x18104001U)
```

Description

Define PM_DEV_PSM_PROC

Definition

```
#define PM_DEV_PSM_PROC(0x18108002U)
```

Description

Define PM_DEV_ACPU_0

Definition

```
#define PM_DEV_ACPU_0(0x1810c003U)
```

Description

Define PM_DEV_ACPU_1

Definition

```
#define PM_DEV_ACPU_1(0x1810c004U)
```

Description

Define PM_DEV_RPU0_0

Definition

```
#define PM_DEV_RPU0_0(0x18110005U)
```

Description

Define PM_DEV_RPU0_1

Definition

```
#define PM_DEV_RPU0_1(0x18110006U)
```

Description

Define PM_DEV_OCM_0

Definition

```
#define PM_DEV_OCM_0(0x18314007U)
```

Description

Define PM_DEV_OCM_1

Definition

```
#define PM_DEV_OCM_1(0x18314008U)
```

Description

Define PM_DEV_OCM_2

Definition

```
#define PM_DEV_OCM_2(0x18314009U)
```

Description

Define PM_DEV_OCM_3

Definition

```
#define PM_DEV_OCM_3(0x1831400aU)
```

Description

Define PM_DEV_TCM_0_A

Definition

```
#define PM_DEV_TCM_0_A(0x1831800bU)
```

Description

Define PM_DEV_TCM_0_B

Definition

```
#define PM_DEV_TCM_0_B(0x1831800cU)
```

Description

Define PM_DEV_TCM_1_A

Definition

```
#define PM_DEV_TCM_1_A(0x1831800dU)
```

Description

Define PM_DEV_TCM_1_B

Definition

```
#define PM_DEV_TCM_1_B(0x1831800eU)
```

Description

Define PM_DEV_L2_BANK_0

Definition

```
#define PM_DEV_L2_BANK_0(0x1831c00fU)
```

Description

Define PM_DEV_DDR_0

Definition

```
#define PM_DEV_DDR_0(0x18320010U)
```

Description

Define PM_DEV_USB_0

Definition

```
#define PM_DEV_USB_0(0x18224018U)
```

Description

Define PM_DEV_GEM_0

Definition

```
#define PM_DEV_GEM_0(0x18224019U)
```

Description

Define PM_DEV_GEM_1

Definition

```
#define PM_DEV_GEM_1(0x1822401aU)
```

Description

Define PM_DEV_SPI_0

Definition

```
#define PM_DEV_SPI_0(0x1822401bU)
```

Description

Define PM_DEV_SPI_1

Definition

```
#define PM_DEV_SPI_1(0x1822401cU)
```

Description

Define PM_DEV_I2C_0

Definition

```
#define PM_DEV_I2C_0(0x1822401dU)
```

Description

Define PM_DEV_I2C_1

Definition

```
#define PM_DEV_I2C_1(0x1822401eU)
```

Description

Define PM_DEV_CAN_FD_0

Definition

```
#define PM_DEV_CAN_FD_0(0x1822401fU)
```

Description

Define PM_DEV_CAN_FD_1

Definition

```
#define PM_DEV_CAN_FD_1(0x18224020U)
```

Description

Define PM_DEV_UART_0

Definition

```
#define PM_DEV_UART_0(0x18224021U)
```

Description

Define PM_DEV_UART_1

Definition

```
#define PM_DEV_UART_1(0x18224022U)
```

Description

Define PM_DEV_GPIO

Definition

```
#define PM_DEV_GPIO(0x18224023U)
```

Description

Define PM_DEV_TTC_0

Definition

```
#define PM_DEV_TTC_0(0x18224024U)
```

Description

Define PM_DEV_TTC_1

Definition

```
#define PM_DEV_TTC_1(0x18224025U)
```

Description

Define PM_DEV_TTC_2

Definition

```
#define PM_DEV_TTC_2(0x18224026U)
```

Description

Define PM_DEV_TTC_3

Definition

```
#define PM_DEV_TTC_3(0x18224027U)
```

Description

Define PM_DEV_SWDT_LPD

Definition

```
#define PM_DEV_SWDT_LPD(0x18224028U)
```

Description

Define PM_DEV_SWDT_FPD

Definition

```
#define PM_DEV_SWDT_FPD(0x18224029U)
```

Description

Define PM_DEV_OSPI

Definition

```
#define PM_DEV_OSPI(0x1822402aU)
```

Description

Define PM_DEV_QSPI

Definition

```
#define PM_DEV_QSPI(0x1822402bU)
```

Description

Define PM_DEV_GPIO_PMC

Definition

```
#define PM_DEV_GPIO_PMC(0x1822402cU)
```

Description

Define PM_DEV_I2C_PMC

Definition

```
#define PM_DEV_I2C_PMC(0x1822402dU)
```

Description

Define PM_DEV_SDIO_0

Definition

```
#define PM_DEV_SDIO_0(0x1822402eU)
```

Description

Define PM_DEV_SDIO_1

Definition

```
#define PM_DEV_SDIO_1(0x1822402fU)
```

Description

Define PM_DEV_RTC

Definition

```
#define PM_DEV_RTC(0x18224034U)
```

Description

Define PM_DEV_ADMA_0

Definition

```
#define PM_DEV_ADMA_0(0x18224035U)
```

Description

Define PM_DEV_ADMA_1

Definition

```
#define PM_DEV_ADMA_1(0x18224036U)
```

Description

Define PM_DEV_ADMA_2

Definition

```
#define PM_DEV_ADMA_2(0x18224037U)
```

Description

Define PM_DEV_ADMA_3

Definition

```
#define PM_DEV_ADMA_3(0x18224038U)
```

Description

Define PM_DEV_ADMA_4

Definition

```
#define PM_DEV_ADMA_4(0x18224039U)
```

Description

Define PM_DEV_ADMA_5

Definition

```
#define PM_DEV_ADMA_5(0x1822403aU)
```

Description

Define PM_DEV_ADMA_6

Definition

```
#define PM_DEV_ADMA_6(0x1822403bU)
```

Description

Define PM_DEV_ADMA_7

Definition

```
#define PM_DEV_ADMA_7(0x1822403cU)
```

Description

Define PM_DEV_IPI_0

Definition

```
#define PM_DEV_IPI_0(0x1822403dU)
```

Description

Define PM_DEV_IPI_1

Definition

```
#define PM_DEV_IPI_1(0x1822403eU)
```

Description

Define PM_DEV_IPI_2

Definition

```
#define PM_DEV_IPI_2(0x1822403fU)
```

Description

Define PM_DEV_IPI_3

Definition

```
#define PM_DEV_IPI_3(0x18224040U)
```

Description

Define PM_DEV_IPI_4

Definition

```
#define PM_DEV_IPI_4(0x18224041U)
```

Description

Define PM_DEV_IPI_5

Definition

```
#define PM_DEV_IPI_5(0x18224042U)
```

Description

Define PM_DEV_IPI_6

Definition

```
#define PM_DEV_IPI_6(0x18224043U)
```

Description

Define PM_DEV_SOC

Definition

```
#define PM_DEV_SOC(0x18428044U)
```

Description

Define PM_DEV_DDRMC_0

Definition

```
#define PM_DEV_DDRMC_0(0x18520045U)
```

Description

Define PM_DEV_DDRMC_1

Definition

```
#define PM_DEV_DDRMC_1(0x18520046U)
```

Description

Define PM_DEV_DDRMC_2

Definition

```
#define PM_DEV_DDRMC_2(0x18520047U)
```

Description

Define PM_DEV_DDRMC_3

Definition

```
#define PM_DEV_DDRMC_3(0x18520048U)
```

Description

Define PM_DEV_GT_0

Definition

```
#define PM_DEV_GT_0(0x1862c049U)
```

Description

Define PM_DEV_GT_1

Definition

```
#define PM_DEV_GT_1(0x1862c04aU)
```

Description

Define PM_DEV_GT_2

Definition

```
#define PM_DEV_GT_2(0x1862c04bU)
```

Description

Define PM_DEV_GT_3

Definition

```
#define PM_DEV_GT_3(0x1862c04cU)
```

Description

Define PM_DEV_GT_4

Definition

```
#define PM_DEV_GT_4(0x1862c04dU)
```

Description

Define PM_DEV_GT_5

Definition

```
#define PM_DEV_GT_5(0x1862c04eU)
```

Description

Define PM_DEV_GT_6

Definition

```
#define PM_DEV_GT_6(0x1862c04fU)
```

Description

Define PM_DEV_GT_7

Definition

```
#define PM_DEV_GT_7(0x1862c050U)
```

Description

Define PM_DEV_GT_8

Definition

```
#define PM_DEV_GT_8(0x1862c051U)
```

Description

Define PM_DEV_GT_9

Definition

```
#define PM_DEV_GT_9(0x1862c052U)
```

Description

Define PM_DEV_GT_10

Definition

```
#define PM_DEV_GT_10(0x1862c053U)
```

Description

Define PM_DEV_EFUSE_CACHE

Definition

```
#define PM_DEV_EFUSE_CACHE(0x18330054U)
```

Description

Define PM_DEV_AMS_ROOT

Definition

```
#define PM_DEV_AMS_ROOT(0x18224055U)
```

Description

Define PM_DEV_AIE

Definition

```
#define PM_DEV_AIE(0x18224072U)
```

Description

Define PM_DEV_IPI_PMC

Definition

```
#define PM_DEV_IPI_PMC(0x18224073U)
```

Description

Subsystem nodes

Definitions

Define PM_SUBSYS_DEFAULT

Definition

```
#define PM_SUBSYS_DEFAULT(0x1c000000U)
```

Description

Define PM_SUBSYS_PMC

Definition

```
#define PM_SUBSYS_PMC(0x1c000001U)
```

Description

Data Structure Index

The following is a list of data structures:

pm_acknowledge

Declaration

```
typedef struct
{
  u8 received,
  u32 node,
  XStatus status,
  u32 opp,
  bool received,
  enum XPmNodeId node
} pm_acknowledge;
```

Table 457: **Structure pm_acknowledge member description**

Member	Description
received	Has acknowledge argument been received?
node	Node argument about which the acknowledge is
status	Acknowledged status
opp	Operating point of node in question
received	Has acknowledge argument been received?
node	Node argument about which the acknowledge is

pm_init_suspend

Declaration

```
typedef struct
{
  u8 received,
  enum XPmSuspendReason reason,
  u32 latency,
  u32 state,
  u32 timeout,
  bool received
} pm_init_suspend;
```

Table 458: **Structure pm_init_suspend member description**

Member	Description
received	Has init suspend callback been received/handled
reason	Reason of initializing suspend
latency	Maximum allowed latency
state	Targeted sleep/suspend state
timeout	Period of time the client has to response
received	Has init suspend callback been received/handled

XPm_DeviceStatus

Contains the device status information.

Declaration

```
typedef struct
{
  u32 Status,
  u32 Requirement,
  u32 Usage
} XPm_DeviceStatus;
```

Table 459: **Structure XPm_DeviceStatus member description**

Member	Description
Status	Device power state
Requirement	Requirements placed on the device by the caller
Usage	Usage info (which subsystem is using the device)

XPm_Master

`XPm_Master` - Master structure

Declaration

```
typedef struct
{
  enum XPmNodeId node_id,
  const u32 pwrctl,
  const u32 pwrdn_mask,
  XIpiPsu * ipi
} XPm_Master;
```

Table 460: **Structure XPm_Master member description**

Member	Description
node_id	Node ID
pwrctl	
pwrdn_mask	< Power Control Register Address Power Down Mask
ipi	IPI Instance

XPm_NodeStatus

`XPm_NodeStatus` - struct containing node status information

Declaration

```
typedef struct
{
  u32 status,
  u32 requirements,
  u32 usage
} XPm_NodeStatus;
```

Table 461: **Structure XPm_NodeStatus member description**

Member	Description
status	Node power state
requirements	Current requirements asserted on the node (slaves only)
usage	Usage information (which master is currently using the slave)

XPm_Notifier

`XPm_Notifier` - Notifier structure registered with a callback by app

Declaration

```
typedef struct
{
  void(*const callback)(struct XPm_Ntfier *const notifier),
  const u32 node,
  enum XPmNotifyEvent event,
  u32 flags,
```

```
  u32 oppoint,
  u32 received,
  struct XPm_Ntfier * next,
  enum XPmNodeId node
} XPm_Notifier;
```

Table 462: **Structure XPm_Notifier member description**

Member	Description
callback	Custom callback handler to be called when the notification is received. The custom handler would execute from interrupt context, it shall return quickly and must not block! (enables event-driven notifications)
node	Node argument (the node to receive notifications about)
event	Event argument (the event type to receive notifications about)
flags	Flags
oppoint	Operating point of node in question. Contains the value updated when the last event notification is received. User shall not modify this value while the notifier is registered.
received	How many times the notification has been received - to be used by application (enables polling). User shall not modify this value while the notifier is registered.
next	Pointer to next notifier in linked list. Must not be modified while the notifier is registered. User shall not ever modify this value.
node	Node argument (the node to receive notifications about)

XPm_Proc

`XPm_Proc` - Processor structure

Declaration

```
typedef struct
{
  const u32 DevId,
  const u32 PwrCtrl,
  const u32 PwrDwnMask,
  XIpiPsu * Ipi
} XPm_Proc;
```

Table 463: **Structure XPm_Proc member description**

Member	Description
DevId	Device ID
PwrCtrl	Power Control Register Address
PwrDwnMask	Power Down Mask
Ipi	IPI Instance

Error Status

This section lists the Power management specific return error statuses.

Definitions

Define XST_PM_INTERNAL

Definition

```
#define XST_PM_INTERNAL2000L
```

Description

An internal error occurred while performing the requested operation

Define XST_PM_CONFLICT

Definition

```
#define XST_PM_CONFLICT2001L
```

Description

Conflicting requirements have been asserted when more than one processing cluster is using the same PM slave

Define XST_PM_NO_ACCESS

Definition

```
#define XST_PM_NO_ACCESS2002L
```

Description

The processing cluster does not have access to the requested node or operation

Define XST_PM_INVALID_NODE

Definition

```
#define XST_PM_INVALID_NODE2003L
```

Description

The API function does not apply to the node passed as argument

Define XST_PM_DOUBLE_REQ

Definition

```
#define XST_PM_DOUBLE_REQ2004L
```

Description

A processing cluster has already been assigned access to a PM slave and has issued a duplicate request for that PM slave

Define XST_PM_ABORT_SUSPEND

Definition

```
#define XST_PM_ABORT_SUSPEND2005L
```

Description

The target processing cluster has aborted suspend

Define XST_PM_TIMEOUT

Definition

```
#define XST_PM_TIMEOUT2006L
```

Description

A timeout occurred while performing the requested operation

Define XST_PM_NODE_USED

Definition

```
#define XST_PM_NODE_USED2007L
```

Description

Slave request cannot be granted since node is non-shareable and used

Reset Nodes

Definitions

Define PM_RST_PMC_POR

Definition

```
#define PM_RST_PMC_POR(0xc30c001U)
```

Description

Define PM_RST_PMC

Definition

```
#define PM_RST_PMC(0xc410002U)
```

Description

Define PM_RST_PS_POR

Definition

```
#define PM_RST_PS_POR(0xc30c003U)
```

Description

Define PM_RST_PL_POR

Definition

```
#define PM_RST_PL_POR(0xc30c004U)
```

Description

Define PM_RST_NOC_POR

Definition

```
#define PM_RST_NOC_POR(0xc30c005U)
```

Description

Define PM_RST_FPD_POR

Definition

```
#define PM_RST_FPD_POR(0xc30c006U)
```

Description

Define PM_RST_ACPU_0_POR

Definition

```
#define PM_RST_ACPU_0_POR(0xc30c007U)
```

Description

Define PM_RST_ACPU_1_POR

Definition

```
#define PM_RST_ACPU_1_POR(0xc30c008U)
```

Description

Define PM_RST_OCM2_POR

Definition

```
#define PM_RST_OCM2_POR(0xc30c009U)
```

Description

Define PM_RST_PS_SRST

Definition

```
#define PM_RST_PS_SRST(0xc41000aU)
```

Description

Define PM_RST_PL_SRST

Definition

```
#define PM_RST_PL_SRST(0xc41000bU)
```

Description

Define PM_RST_NOC

Definition

```
#define PM_RST_NOC(0xc41000cU)
```

Description

Define PM_RST_NPI

Definition

```
#define PM_RST_NPI(0xc41000dU)
```

Description

Define PM_RST_SYS_RST_1

Definition

```
#define PM_RST_SYS_RST_1(0xc41000eU)
```

Description

Define PM_RST_SYS_RST_2

Definition

```
#define PM_RST_SYS_RST_2(0xc41000fU)
```

Description

Define PM_RST_SYS_RST_3

Definition

```
#define PM_RST_SYS_RST_3(0xc410010U)
```

Description

Define PM_RST_FPD

Definition

```
#define PM_RST_FPD(0xc410011U)
```

Description

Define PM_RST_PL0

Definition

```
#define PM_RST_PL0(0xc410012U)
```

Description

Define PM_RST_PL1

Definition

```
#define PM_RST_PL1(0xc410013U)
```

Description

Define PM_RST_PL2

Definition

```
#define PM_RST_PL2(0xc410014U)
```

Description

Define PM_RST_PL3

Definition

```
#define PM_RST_PL3(0xc410015U)
```

Description

Define PM_RST_APU

Definition

```
#define PM_RST_APU(0xc410016U)
```

Description

Define PM_RST_ACPU_0

Definition

```
#define PM_RST_ACPU_0(0xc410017U)
```

Description

Define PM_RST_ACPU_1

Definition

```
#define PM_RST_ACPU_1(0xc410018U)
```

Description

Define PM_RST_ACPU_L2

Definition

```
#define PM_RST_ACPU_L2(0xc410019U)
```

Description

Define PM_RST_ACPU_GIC

Definition

```
#define PM_RST_ACPU_GIC(0xc41001aU)
```

Description

Define PM_RST_RPU_ISLAND

Definition

```
#define PM_RST_RPU_ISLAND(0xc41001bU)
```

Description

Define PM_RST_RPU_AMBA

Definition

```
#define PM_RST_RPU_AMBA(0xc41001cU)
```

Description

Define PM_RST_R5_0

Definition

```
#define PM_RST_R5_0(0xc41001dU)
```

Description

Define PM_RST_R5_1

Definition

```
#define PM_RST_R5_1(0xc41001eU)
```

Description

Define PM_RST_SYSMON_PMC_SEQ_RST

Definition

```
#define PM_RST_SYSMON_PMC_SEQ_RST(0xc41001fU)
```

Description

Define PM_RST_SYSMON_PMC_CFG_RST

Definition

```
#define PM_RST_SYSMON_PMC_CFG_RST(0xc410020U)
```

Description

Define PM_RST_SYSMON_FPD_CFG_RST

Definition

```
#define PM_RST_SYSMON_FPD_CFG_RST(0xc410021U)
```

Description

Define PM_RST_SYSMON_FPD_SEQ_RST

Definition

```
#define PM_RST_SYSMON_FPD_SEQ_RST(0xc410022U)
```

Description

Define PM_RST_SYSMON_LPD

Definition

```
#define PM_RST_SYSMON_LPD(0xc410023U)
```

Description

Define PM_RST_PDMA_RST1

Definition

```
#define PM_RST_PDMA_RST1(0xc410024U)
```

Description

Define PM_RST_PDMA_RST0

Definition

```
#define PM_RST_PDMA_RST0(0xc410025U)
```

Description

Define PM_RST_ADMA

Definition

```
#define PM_RST_ADMA(0xc410026U)
```

Description

Define PM_RST_TIMESTAMP

Definition

```
#define PM_RST_TIMESTAMP(0xc410027U)
```

Description

Define PM_RST_OCM

Definition

```
#define PM_RST_OCM(0xc410028U)
```

Description

Define PM_RST_OCM2_RST

Definition

```
#define PM_RST_OCM2_RST(0xc410029U)
```

Description

Define PM_RST_IPI

Definition

```
#define PM_RST_IPI(0xc41002aU)
```

Description

Define PM_RST_SBI

Definition

```
#define PM_RST_SBI(0xc41002bU)
```

Description

Define PM_RST_LPD

Definition

```
#define PM_RST_LPD(0xc41002cU)
```

Description

Define PM_RST_QSPI

Definition

```
#define PM_RST_QSPI(0xc10402dU)
```

Description

Define PM_RST_OSPI

Definition

```
#define PM_RST_OSPI(0xc10402eU)
```

Description

Define PM_RST_SDIO_0

Definition

```
#define PM_RST_SDIO_0(0xc10402fU)
```

Description

Define PM_RST_SDIO_1

Definition

```
#define PM_RST_SDIO_1(0xc104030U)
```

Description

Define PM_RST_I2C_PMC

Definition

```
#define PM_RST_I2C_PMC(0xc104031U)
```

Description

Define PM_RST_GPIO_PMC

Definition

```
#define PM_RST_GPIO_PMC(0xc104032U)
```

Description

Define PM_RST_GEM_0

Definition

```
#define PM_RST_GEM_0(0xc104033U)
```

Description

Define PM_RST_GEM_1

Definition

```
#define PM_RST_GEM_1(0xc104034U)
```

Description

Define PM_RST_SPARE

Definition

```
#define PM_RST_SPARE(0xc104035U)
```

Description

Define PM_RST_USB_0

Definition

```
#define PM_RST_USB_0(0xc104036U)
```

Description

Define PM_RST_UART_0

Definition

```
#define PM_RST_UART_0(0xc104037U)
```

Description

Define PM_RST_UART_1

Definition

```
#define PM_RST_UART_1(0xc104038U)
```

Description

Define PM_RST_SPI_0

Definition

```
#define PM_RST_SPI_0(0xc104039U)
```

Description

Define PM_RST_SPI_1

Definition

```
#define PM_RST_SPI_1(0xc10403aU)
```

Description

Define PM_RST_CAN_FD_0

Definition

```
#define PM_RST_CAN_FD_0(0xc10403bU)
```

Description

Define PM_RST_CAN_FD_1

Definition

```
#define PM_RST_CAN_FD_1(0xc10403cU)
```

Description

Define PM_RST_I2C_0

Definition

```
#define PM_RST_I2C_0(0xc10403dU)
```

Description

Define PM_RST_I2C_1

Definition

```
#define PM_RST_I2C_1(0xc10403eU)
```

Description

Define PM_RST_GPIO_LPD

Definition

```
#define PM_RST_GPIO_LPD(0xc10403fU)
```

Description

Define PM_RST_TTC_0

Definition

```
#define PM_RST_TTC_0(0xc104040U)
```

Description

Define PM_RST_TTC_1

Definition

```
#define PM_RST_TTC_1(0xc104041U)
```

Description

Define PM_RST_TTC_2

Definition

```
#define PM_RST_TTC_2(0xc104042U)
```

Description

Define PM_RST_TTC_3

Definition

```
#define PM_RST_TTC_3(0xc104043U)
```

Description

Define PM_RST_SWDT_FPD

Definition

```
#define PM_RST_SWDT_FPD(0xc104044U)
```

Description

Define PM_RST_SWDT_LPD

Definition

```
#define PM_RST_SWDT_LPD(0xc104045U)
```

Description

Define PM_RST_USB

Definition

```
#define PM_RST_USB(0xc104046U)
```

Description

Define PM_RST_DPC

Definition

```
#define PM_RST_DPC(0xc208047U)
```

Description

Define PM_RST_PMCDBG

Definition

```
#define PM_RST_PMCDBG(0xc208048U)
```

Description

Define PM_RST_DBG_TRACE

Definition

```
#define PM_RST_DBG_TRACE(0xc208049U)
```

Description

Define PM_RST_DBG_FPD

Definition

```
#define PM_RST_DBG_FPD(0xc20804aU)
```

Description

Define PM_RST_DBG_TSTMP

Definition

```
#define PM_RST_DBG_TSTMP(0xc20804bU)
```

Description

Define PM_RST_RPU0_DBG

Definition

```
#define PM_RST_RPU0_DBG(0xc20804cU)
```

Description

Define PM_RST_RPU1_DBG

Definition

```
#define PM_RST_RPU1_DBG(0xc20804dU)
```

Description

Define PM_RST_HSDP

Definition

```
#define PM_RST_HSDP(0xc20804eU)
```

Description

Define PM_RST_DBG_LPD

Definition

```
#define PM_RST_DBG_LPD(0xc20804fU)
```

Description

Define PM_RST_CPM_POR

Definition

```
#define PM_RST_CPM_POR(0xc30c050U)
```

Description

Define PM_RST_CPM

Definition

```
#define PM_RST_CPM(0xc410051U)
```

Description

Define PM_RST_CPMDBG

Definition

```
#define PM_RST_CPMDBG(0xc208052U)
```

Description

Define PM_RST_PCIE_CFG

Definition

```
#define PM_RST_PCIE_CFG(0xc410053U)
```

Description

Define PM_RST_PCIE_CORE0

Definition

```
#define PM_RST_PCIE_CORE0(0xc410054U)
```

Description

Define PM_RST_PCIE_CORE1

Definition

```
#define PM_RST_PCIE_CORE1(0xc410055U)
```

Description

Define PM_RST_PCIE_DMA

Definition

```
#define PM_RST_PCIE_DMA(0xc410056U)
```

Description

Define PM_RST_CMN

Definition

```
#define PM_RST_CMN(0xc410057U)
```

Description

Define PM_RST_L2_0

Definition

```
#define PM_RST_L2_0(0xc410058U)
```

Description

Define PM_RST_L2_1

Definition

```
#define PM_RST_L2_1(0xc410059U)
```

Description

Define PM_RST_ADDR_REMAP

Definition

```
#define PM_RST_ADDR_REMAP(0xc41005aU)
```

Description

Define PM_RST_CPI0

Definition

```
#define PM_RST_CPI0(0xc41005bU)
```

Description

Define PM_RST_CPI1

Definition

```
#define PM_RST_CPI1(0xc41005cU)
```

Description

Define PM_RST_AIE_ARRAY

Definition

```
#define PM_RST_AIE_ARRAY(0xc10405eU)
```

Description

Define PM_RST_AIE_SHIM

Definition

```
#define PM_RST_AIE_SHIM(0xc10405fU)
```

Description

XilFPGA Library v5.3

Overview

The XilFPGA library provides an interface for the users to configure the programmable logic (PL) from PS. The library is designed to run on top of Xilinx® standalone BSPs. It acts as a bridge between the user application and the PL device. It provides the required functionality to the user application for configuring the PL Device with the required bitstream.

Note: XILFPGA does not support a DDR less system. DDR must be present for use of XilFPGA.

Supported Features

Zynq® UltraScale+™ MPSoC platform

The following features are supported in Zynq UltraScale+ MPSoC platform:

- Full bitstream loading
- Partial bitstream loading
- Encrypted bitstream loading
- Authenticated bitstream loading
- Authenticated and encrypted bitstream loading
- Readback of configuration registers
- Readback of configuration data

Versal™ ACAP

The following features are supported in Versal platform:

- Full/Partial bitstream loading
- Device Key Encrypted bitstream loading
- Authenticated bitstream loading

- Authenticated and Device-key encrypted bitstream loading

Zynq UltraScale+ MPSoC XilFPGA Library

The library when used for Zynq UltraScale+ MPSoC runs on top of Xilinx standalone BSPs. It is tested for Arm Cortex A53, Arm Cortex R5Fand MicroBlaze. In the most common use case, you should run this library on the PMU MicroBlaze with PMUFW to serve requests from either Linux or U-Boot for bitstream programming.

XilFPGA library Interface modules

XilFPGA library uses the below major components to configure the PL through PS.

Processor Configuration Access Port (PCAP)

The processor configuration access port (PCAP) is used to configure the programmable logic (PL) through the PS.

CSU DMA driver

The CSU DMA driver is used to transfer the actual bitstream file for the PS to PL after PCAP initialization.

XilSecure Library

The XilSecure library provides APIs to access secure hardware on the Zynq UltraScale+ MPSoC devices.

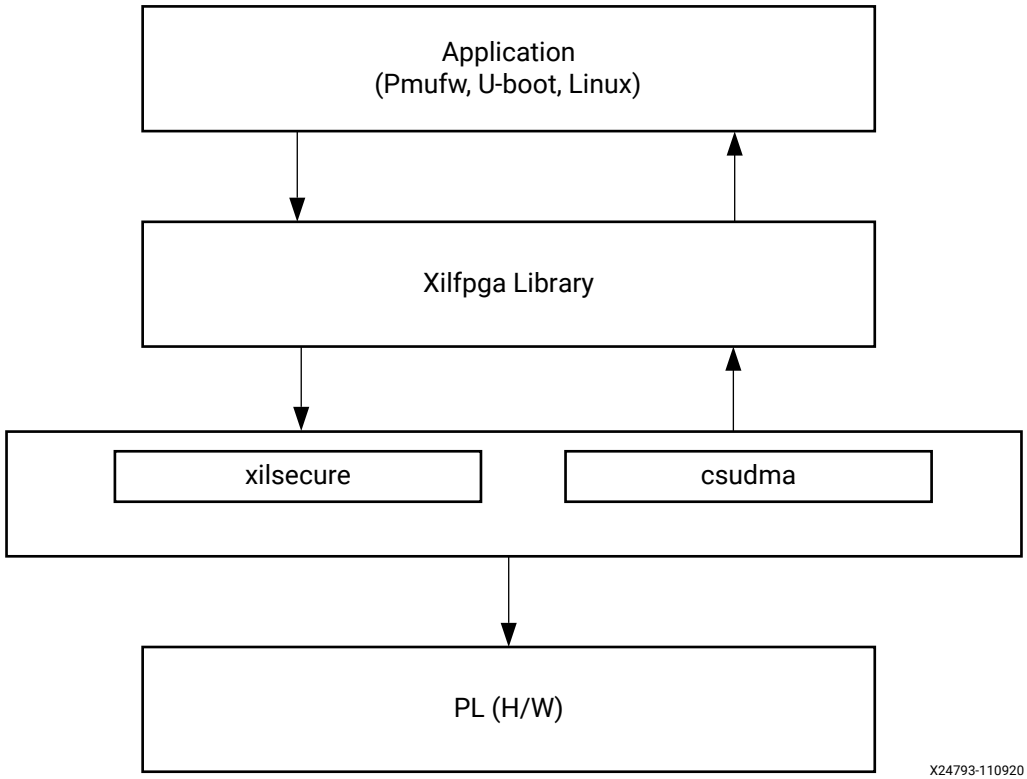
***Note*:** The current version of library supports only Zynq UltraScale MPSoC devices.

Design Summary

XilFPGA library acts as a bridge between the user application and the PL device.

It provides the required functionality to the user application for configuring the PL Device with the required bitstream. The following figure illustrates an implementation where the XilFPGA library needs the CSU DMA driver APIs to transfer the bitstream from the DDR to the PL region. The XilFPGA library also needs the XilSecure library APIs to support programming authenticated and encrypted bitstream files.

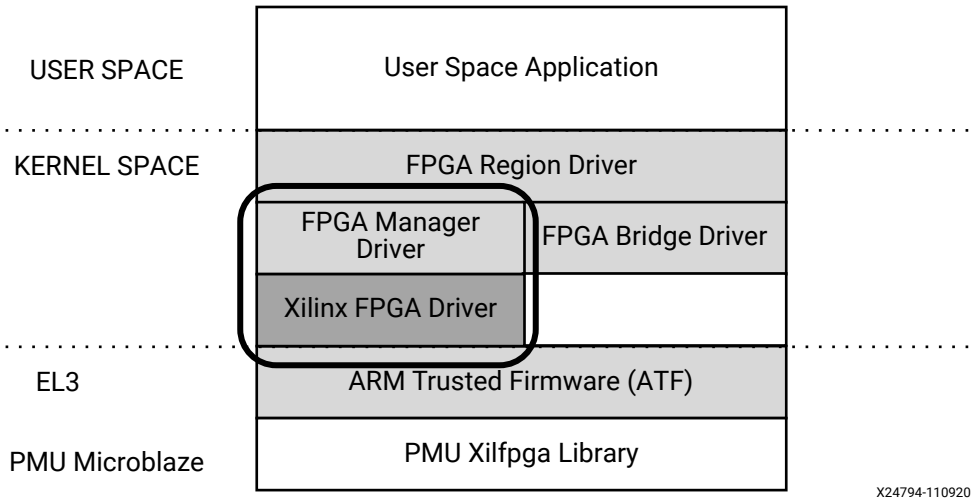
Figure 3: **XilFPGA Design Summary**



Flow Diagram

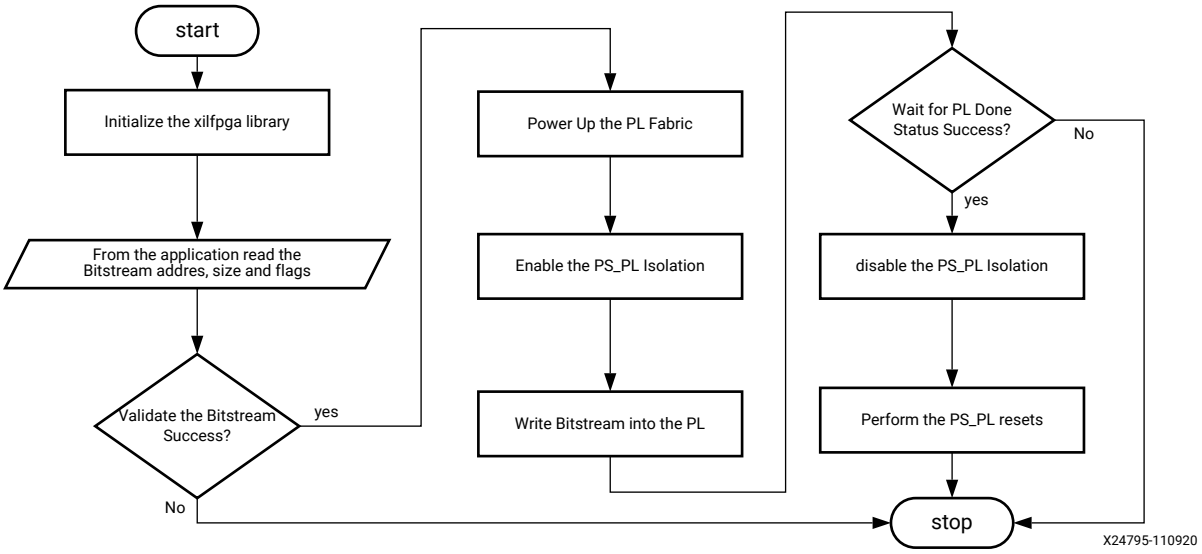
The following figure illustrates the Bitstream loading flow on the Linux operating system.

Figure 4: **Bitstream loading on Linux:**



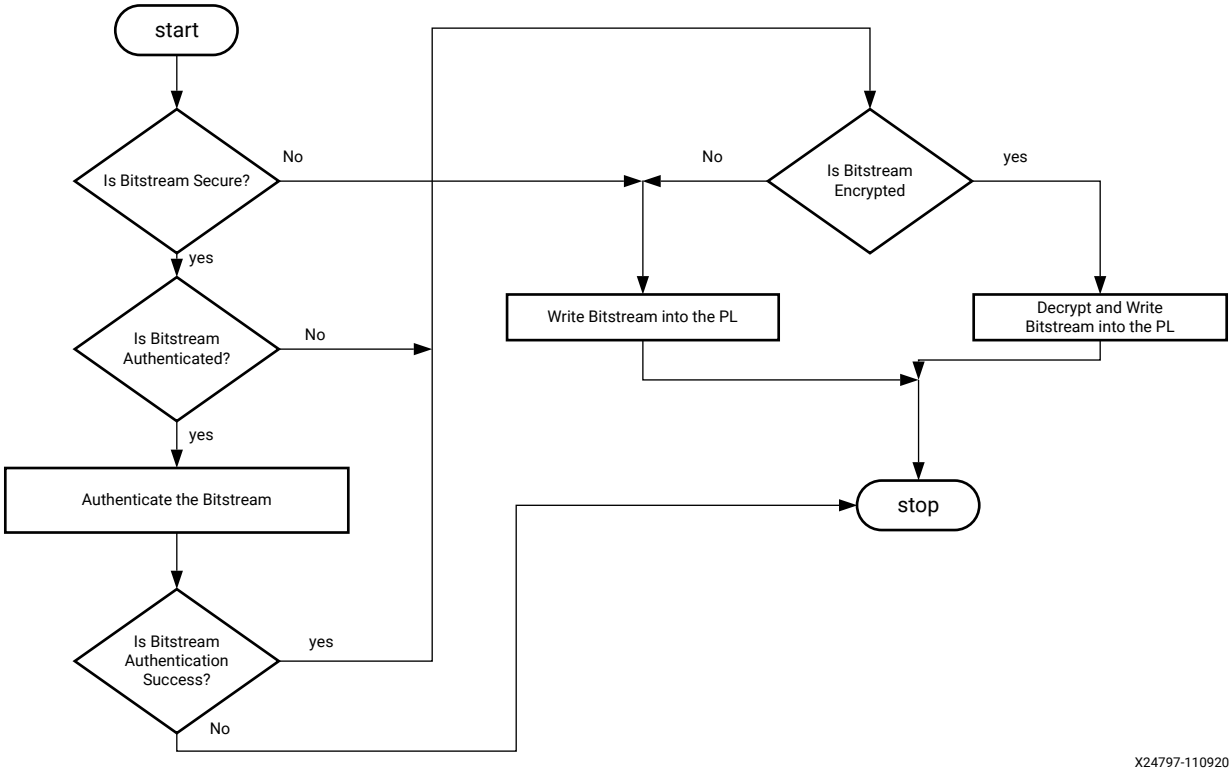
The following figure illustrates the XilFPGA PL configuration sequence.

Figure 5: **XilFPGA PL Configuration Sequence**



The following figure illustrates the Bitstream write sequence.

Figure 6: **Bitstream write Sequence**



XilFPGA BSP Configuration Settings

Xilfpga provides the following user configuration BSP settings.

Parameter Name	Type	Default Value	Description
secure_mode	bool	TRUE	Enables secure Bitstream loading support.
debug_mode	bool	FALSE	Enables the Debug messages in the library.
ocm_address	int	0xfffc0000	Address used for the Bitstream authentication.
base_address	int	0x80000	Holds the Bitstream Image address. This flag is valid only for the Cortex-A53 or the Cortex-R5 processors.
secure_readback	bool	FALSE	Should be set to TRUE to allow the secure Bitstream configuration data read back. The application environment should be secure and trusted to enable this flag.
secure_environment	bool	FALSE	Enable the secure PL configuration using the IPI. This flag is valid only for the Cortex-A53 or the Cortex-R5 processors.

Setting up the Software System

To use XilFPGA in a software application, you must first compile the XilFPGA library as part of software application.

1. Click **File** > **New** > **Platform Project**.
2. Click **Specify** to create a new Hardware Platform Specification.
3. Provide a new name for the domain in the **Project name** field if you wish to override the default value.
4. Select the location for the board support project files. To use the default location, as displayed in the **Location** field, leave the **Use default location** check box selected. Otherwise, deselect the checkbox and then type or browse to the directory location.
5. From the **Hardware Platform** drop-down choose the appropriate platform for your application or click the **New** button to browse to an existing Hardware Platform.
6. Select the target CPU from the drop-down list.
7. From the **Board Support Package OS** list box, select the type of board support package to create. A description of the platform types displays in the box below the drop-down list.
8. Click **Finish**. The wizard creates a new software platform and displays it in the Vitis Navigator pane.

9. Select **Project** > **Build Automatically** to automatically build the board support package. The Board Support Package Settings dialog box opens. Here you can customize the settings for the domain.
10. Click **OK** to accept the settings, build the platform, and close the dialog box.
11. From the Explorer, double-click platform.spr file and select the appropriate domain/board support package. The overview page opens.
12. In the overview page, click **Modify BSP Settings**.
13. Using the Board Support Package Settings page, you can select the OS Version and which of the Supported Libraries are to be enabled in this domain/BSP.
14. Select the **xilfpga** library from the list of **Supported Libraries**.
15. Expand the **Overview** tree and select **xilfpga**. The configuration options for xilfpga are listed.
16. Configure the xilfpga by providing the base address of the Bit-stream file (DDR address) and the size (in bytes).
17. Click **OK**. The board support package automatically builds with XilFPGA library included in it.
18. Double-click the **system.mss** file to open it in the **Editor** view.
19. Scroll-down and locate the **Libraries** section.
20. Click **Import Examples** adjacent to the XilFPGA entry.

Enabling Security

To support encrypted and/or authenticated bitstream loading, you must enable security in PMUFW.

1. Click **File** > **New** > **Platform Project**.
2. Click **Specify** to create a new Hardware Platform Specification.
3. Provide a new name for the domain in the **Project name** field if you wish to override the default value.
4. Select the location for the board support project files. To use the default location, as displayed in the **Location** field, leave the **Use default location** check box selected. Otherwise, deselect the checkbox and then type or browse to the directory location.
5. From the **Hardware Platform** drop-down choose the appropriate platform for your application or click the **New** button to browse to an existing Hardware Platform.
6. Select the target CPU from the drop-down list.
7. From the **Board Support Package OS** list box, select the type of board support package to create. A description of the platform types displays in the box below the drop-down list.
8. Click **Finish**. The wizard creates a new software platform and displays it in the Vitis Navigator pane.

9. Select **Project** > **Build Automatically** to automatically build the board support package. The Board Support Package Settings dialog box opens. Here you can customize the settings for the domain.
10. Click **OK** to accept the settings, build the platform, and close the dialog box.
11. From the Explorer, double-click platform.spr file and select the appropriate domain/board support package. The overview page opens.
12. In the overview page, click **Modify BSP Settings**.
13. Using the Board Support Package Settings page, you can select the OS Version and which of the Supported Libraries are to be enabled in this domain/BSP.
14. Expand the **Overview** tree and select **Standalone**.
15. Select a supported hardware platform.
16. Select **psu_pmu_0** from the **Processor** drop-down list.
17. Click Next. The **Templates** page appears.
18. Select **ZynqMP PMU Firmware** from the **Available Templates** list.
19. Click **Finish**. A PMUFW application project is created with the required BSPs.
20. Double-click the **system.mss** file to open it in the **Editor** view.
21. Click the **Modify this BSP's Settings** button. The **Board Support Package Settings** dialog box appears.
22. Select **xilfpga**. Various settings related to the library appears.
23. Select **secure_mode** and modify its value to **true** .
24. Click **OK** to save the configuration.

***Note*:** By default the secure mode is enabled. To disable modify the secure_mode value to false.

Bitstream Authentication Using External Memory

The size of the Bitstream is too large to be contained inside the device, therefore external memory must be used.

The use of external memory could create a security risk. Therefore, two methods are provided to authenticate and decrypt a Bitstream.

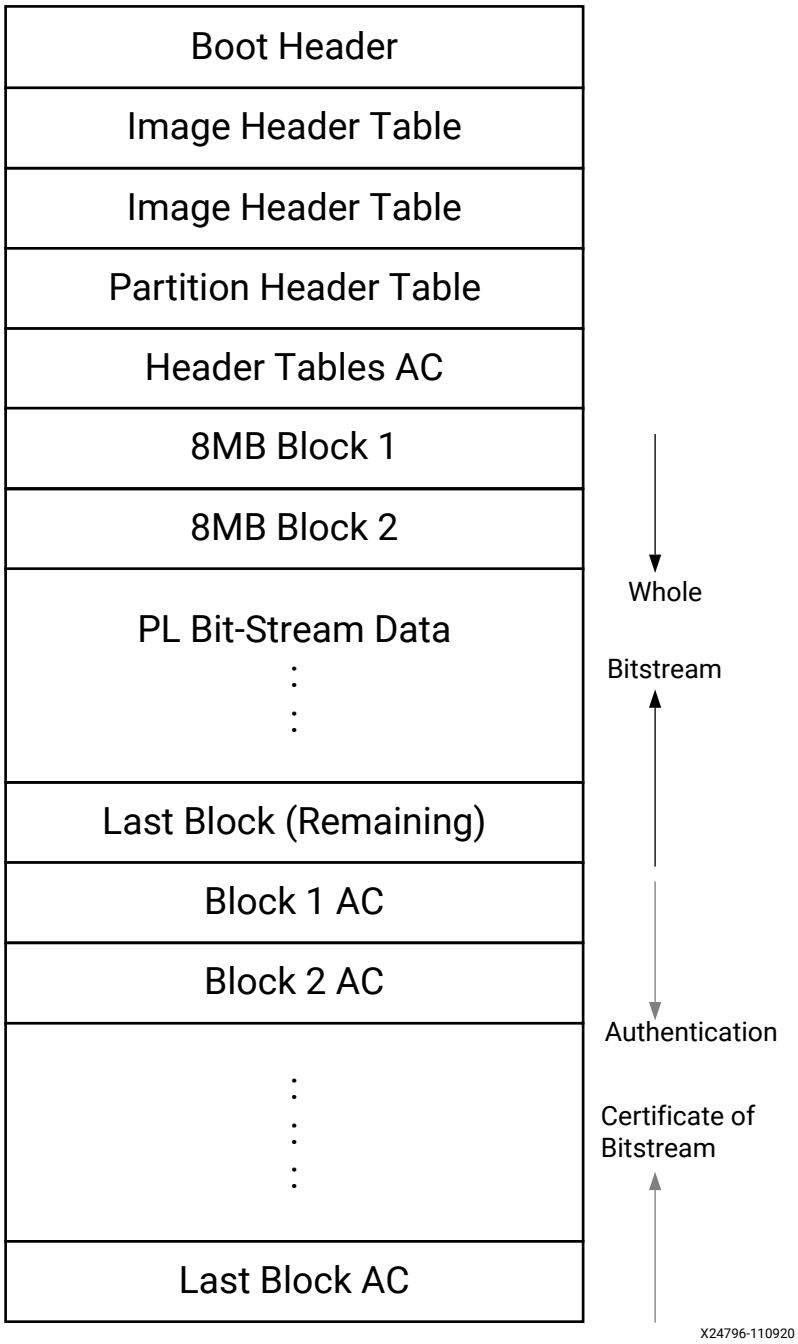
- The first method uses the internal OCM as temporary buffer for all cryptographic operations. For details, see `Authenticated and Encrypted Bitstream Loading Using OCM`. This method does not require trust in external DDR.
- The second method uses external DDR for authentication prior to sending the data to the decryptor, there by requiring trust in the external DDR. For details, see `Authenticated and Encrypted Bitstream Loading Using DDR`.

Bootgen

When a Bitstream is requested for authentication, Bootgen divides the Bitstream into blocks of 8MB each and assigns an authentication certificate for each block.

If the size of a Bitstream is not in multiples of 8 MB, the last block contains the remaining Bitstream data.

Figure 7: **Bitstream Blocks**



When both authentication and encryption are enabled, encryption is first done on the Bitstream. Bootgen then divides the encrypted data into blocks and assigns an Authentication certificate for each block.

Authenticated and Encrypted Bitstream Loading Using OCM

To authenticate the Bitstream partition securely, XilFPGA uses the FSBL section's OCM memory to copy the bitstream in chunks from DDR.

This method does not require trust in the external DDR to securely authenticate and decrypt a Bitstream.

The software workflow for authenticating Bitstream is as follows:

1. XilFPGA identifies DDR secure Bitstream image base address. XilFPGA has two buffers in OCM, the Read Buffer is of size 56KB and hash of chunks to store intermediate hashes calculated for each 56 KB of every 8MB block.
2. XilFPGA copies a 56KB chunk from the first 8MB block to Read Buffer.
3. XilFPGA calculates hash on 56 KB and stores in HashsOfChunks.
4. XilFPGA repeats steps 1 to 3 until the entire 8MB of block is completed.

   Note: The chunk that XilFPGA copies can be of any size. A 56KB chunk is taken for better performance.
5. XilFPGA authenticates the 8MB Bitstream chunk.
6. Once the authentication is successful, XilFPGA starts copying information in batches of 56KB starting from the first block which is located in DDR to Read Buffer, calculates the hash, and then compares it with the hash stored at HashsOfChunks.
7. If the hash comparison is successful, FSBL transmits data to PCAP using DMA (for un-encrypted Bitstream) or AES (if encryption is enabled).
8. XilFPGA repeats steps 6 and 7 until the entire 8MB block is completed.
9. Repeats steps 1 through 8 for all the blocks of Bitstream.

Note: You can perform warm restart even when the FSBL OCM memory is used to authenticate the Bitstream. PMU stores the FSBL image in the PMU reserved DDR memory which is visible and accessible only to the PMU and restores back to the OCM when APU-only restart needs to be performed. PMU uses the SHA3 hash to validate the FSBL image integrity before restoring the image to OCM (PMU takes care of only image integrity and not confidentiality).

Authenticated and Encrypted Bitstream Loading Using DDR

The software workflow for authenticating Bitstream is as follows:

1. XilFPGA identifies DDR secure Bitstream image base address.
2. XilFPGA calculates hash for the first 8MB block.

3. XilFPGA authenticates the 8MB block while stored in the external DDR.
4. If Authentication is successful, XilFPGA transmits data to PCAP via DMA (for unencrypted Bitstream) or AES (if encryption is enabled).
5. Repeats steps 1 through 4 for all the blocks of Bitstream.

XilFPGA APIs

This section provides detailed descriptions of the XilFPGA library APIs.

Xilfpga Error = Lower level Errors + Interface specific Errors + Xilfpga top layer Errors

Table 464: **XilFPGA Error**

Lower Level Errors (other libraries or drivers used by XilFPGA)	Interface Specific Errors (PCAP Interface)	Xilfpga Top layer Errors
31 - 16 bits	15 - 8 bits	7 - 0 bits

- **XilFPGA Top Layer:** The functionality exist in this layers is completely Interface agnostic. It provides a unique interface to load the Bitstream across multiple platforms.
- **Interface Specific Layer:** This layer is responsible for providing the interface specific related errors. In case of Zynq UltraScale+ MPSoC, it provides the errors related to PCAP Interface.
- **XilFPGA Lower Layer:** This layer is responsible for providing the Error related to the lower level drivers used by interface layer.

Table 465: **Quick Function Reference**

Type	Name	Arguments
u32	XFpga_Initialize	XFpga * InstancePtr
u32	XFpga_PL_BitStream_Load	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags
u32	XFpga_PL_Preconfig	XFpga * InstancePtr
u32	XFpga_PL_Write	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags

Table 465: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
u32	XFpga_PL_PostConfig	XFpga * InstancePtr
u32	XFpga_PL_ValidateImage	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags
u32	XFpga_GetPlConfigData	XFpga * InstancePtr UINTPTR ReadbackAddr u32 NumFrames
u32	XFpga_GetPlConfigReg	XFpga * InstancePtr UINTPTR ReadbackAddr u32 ConfigRegAddr
u32	XFpga_InterfaceStatus	XFpga * InstancePtr

Functions

XFpga_Initialize

This API when called initializes the XFPGA interface with default settings.

Prototype

```
u32 XFpga_Initialize(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_Initialize` function arguments.

Table 466: **XFpga_Initialize Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure.

Returns

Returns Status

- XFPGA_SUCCESS on success
- Error code on failure

XFpga_PL_BitStream_Load

The API is used to load the bitstream file into the PL region.

It supports vivado generated Bitstream(*.bit, *.bin) and bootgen generated Bitstream(*.bin) loading, Passing valid Bitstream size (AddrPtr_Size) info is mandatory for vivado * generated Bitstream, For bootgen generated Bitstreams it will take Bitstream size from the Bitstream Header.

Prototype

```
u32 XFpga_PL_BitStream_Load(XFpga *InstancePtr, UINTPTR BitstreamImageAddr,
UINTPTR AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_BitStream_Load` function arguments.

Table 467: **XFpga_PL_BitStream_Load Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure.
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Aes key address which is used for Decryption (or) In non-secure Bistream use cases it is used to store the size of Bitstream Image.
u32	Flags	Flags are used to specify the type of Bitstream file. • BIT(0) - Bitstream type ◦ 0 - Full Bitstream ◦ 1 - Partial Bitstream • BIT(1) - Authentication using DDR ◦ 1 - Enable ◦ 0 - Disable • BIT(2) - Authentication using OCM ◦ 1 - Enable ◦ 0 - Disable • BIT(3) - User-key Encryption ◦ 1 - Enable ◦ 0 - Disable • BIT(4) - Device-key Encryption ◦ 1 - Enable ◦ 0 - Disable

Returns

- XFPGA_SUCCESS on success
- Error code on failure.
- XFPGA_VALIDATE_ERROR.
- XFPGA_PRE_CONFIG_ERROR.
- XFPGA_WRITE_BITSTREAM_ERROR.
- XFPGA_POST_CONFIG_ERROR.

XFpga_PL_Preconfig

This function prepare the FPGA to receive configuration data.

Prototype

```
u32 XFpga_PL_Preconfig(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_PL_Preconfig` function arguments.

Table 468: **XFpga_PL_Preconfig Arguments**

Type	Name	Description
XFpga *	InstancePtr	is the pointer to the XFgpa.

Returns

Codes as mentioned in xilfpga.h

XFpga_PL_Write

This function write count bytes of configuration data into the PL.

Prototype

```
u32 XFpga_PL_Write(XFpga *InstancePtr, UINTPTR BitstreamImageAddr, UINTPTR
AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_Write` function arguments.

Table 469: **XFpga_PL_Write Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Aes key address which is used for Decryption (or) In non-secure Bistream use cases it is used to store the size of Bitstream Image.
u32	Flags	Flags are used to specify the type of Bitstream file. • BIT(0) - Bitstream type ◦ 0 - Full Bitstream ◦ 1 - Partial Bitstream • BIT(1) - Authentication using DDR ◦ 1 - Enable ◦ 0 - Disable • BIT(2) - Authentication using OCM ◦ 1 - Enable ◦ 0 - Disable • BIT(3) - User-key Encryption ◦ 1 - Enable ◦ 0 - Disable • BIT(4) - Device-key Encryption ◦ 1 - Enable ◦ 0 - Disable

Returns

Codes as mentioned in xilfpga.h

XFpga_PL_PostConfig

This function set FPGA to operating state after writing.

Prototype

```
u32 XFpga_PL_PostConfig(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_PL_PostConfig` function arguments.

Table 470: **XFpga_PL_PostConfig Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure

Returns

Codes as mentioned in xilfpga.h

XFpga_PL_ValidateImage

This function is used to validate the Bitstream Image.

Prototype

```
u32 XFpga_PL_ValidateImage(XFpga *InstancePtr, UINTPTR BitstreamImageAddr,
UINTPTR AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_ValidateImage` function arguments.

Table 471: **XFpga_PL_ValidateImage Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Aes key address which is used for Decryption (or) In non-secure Bistream use cases it is used to store the size of Bitstream Image.

Table 471: **XFpga_PL_ValidateImage Arguments** *(cont'd)*

Type	Name	Description
u32	Flags	Flags are used to specify the type of Bitstream file. • BIT(0) - Bitstream type ◦ 0 - Full Bitstream ◦ 1 - Partial Bitstream • BIT(1) - Authentication using DDR ◦ 1 - Enable ◦ 0 - Disable • BIT(2) - Authentication using OCM ◦ 1 - Enable ◦ 0 - Disable • BIT(3) - User-key Encryption ◦ 1 - Enable ◦ 0 - Disable • BIT(4) - Device-key Encryption ◦ 1 - Enable ◦ 0 - Disable

Returns

Codes as mentioned in xilfpga.h

XFpga_GetPlConfigData

This function provides functionality to read back the PL configuration data.

Prototype

```
u32 XFpga_GetPlConfigData(XFpga *InstancePtr, UINTPTR ReadbackAddr, u32
NumFrames);
```

Parameters

The following table lists the `XFpga_GetPlConfigData` function arguments.

Table 472: **XFpga_GetPlConfigData Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure

Table 472: **XFpga_GetPlConfigData Arguments** *(cont'd)*

Type	Name	Description
UINTPTR	ReadbackAddr	Address which is used to store the PL readback data.
u32	NumFrames	The number of Fpga configuration frames to read.

Returns

- XFPGA_SUCCESS if successful
- XFPGA_FAILURE if unsuccessful
- XFPGA_OPS_NOT_IMPLEMENTED if implementation not exists.

XFpga_GetPlConfigReg

This function provides PL specific configuration register values.

Prototype

```
u32 XFpga_GetPlConfigReg(XFpga *InstancePtr, UINTPTR ReadbackAddr, u32
ConfigRegAddr);
```

Parameters

The following table lists the `XFpga_GetPlConfigReg` function arguments.

Table 473: **XFpga_GetPlConfigReg Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure
UINTPTR	ReadbackAddr	Address which is used to store the PL Configuration register data.
u32	ConfigRegAddr	Configuration register address as mentioned in the ug570.

Returns

- XFPGA_SUCCESS if successful
- XFPGA_FAILURE if unsuccessful
- XFPGA_OPS_NOT_IMPLEMENTED if implementation not exists.

XFpga_InterfaceStatus

This function provides the STATUS of PL programming interface.

Prototype

```
u32 XFpga_InterfaceStatus(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_InterfaceStatus` function arguments.

Table 474: **XFpga_InterfaceStatus Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure

Returns

Status of the PL programming interface

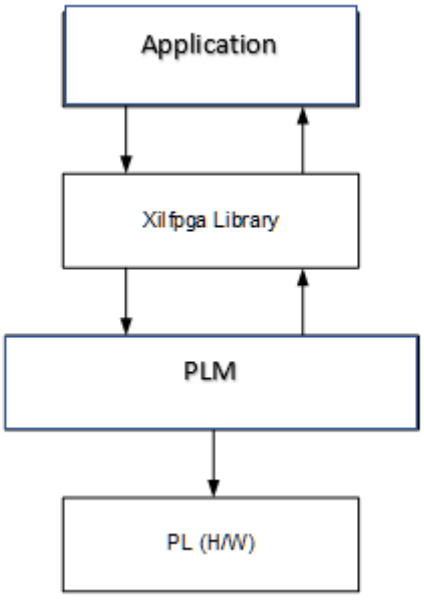
Versal ACAP XilFPGA Library

The library, when used for Versal ACAP, runs on top of Xilinx standalone BSPs. It is tested for Arm Cortex A72 and Arm Cortex R5F. The most common usecase is that the user can run this on either Arm Cortex A72 or Arm Cortex R5F and requests PLM to load the bitstream (PDI) on to the PL. In versal the bitstream always comes in the format of PDI file.

Design Summary

The following figure shows the flow diagram of how an user application interacts with XilFPGA interacts and other SW components for loading the bitstream from DDR to the PL region.

Figure 8: **XilFPGA Design Summary**



BSP Configuration Settings

XilFPGA provides the following user configuration BSP settings.

Table 475: **BSP Configuration Settings**

Parameter Name	Type	Default Value	Description
base_address	int	0x80000	Holds the bitstream Image address. This flag is valid only for the Cortex-A72 or the Cortex-R5 processors.

Setting up the Software System

To use XilFPGA in a software application, you must first compile the XilFPGA library as part of software application.

1. Click **File→ New→ Platform Project**.
2. Click **Specify** to create a new Hardware Platform Specification.
3. Provide a new name for the domain in the Project name field if you wish to override the default value.
4. Select the location for the board support project files. To use the default location, as displayed in the Location field, leave the Use default location check box selected. Otherwise, deselect the checkbox and then type or browse to the directory location.
5. From the Hardware Platform drop-down choose the appropriate platform for your application or click the New button to browse to an existing Hardware Platform.
6. Select the target CPU from the drop-down list.
7. From the Board Support Package OS list box, select the type of board support package to create. A description of the platform types displays in the box below the drop-down list.
8. Click Finish. The wizard creates a new software platform and displays it in the Vitis Navigator pane.
9. Select **Project→Build Automatically** to automatically build the board support package. The Board Support Package Settings dialog box opens. Here you can customize the settings for the domain.
10. Click OK to accept the settings, build the platform, and close the dialog box.
11. From the Explorer, double-click **platform.spr** file and select the appropriate domain/board support package. The overview page opens.
12. In the overview page, click **Modify BSP Settings**.
13. Using the Board Support Package Settings page, you can select the OS Version and which of the Supported Libraries are to be enabled in this domain/BSP.
14. Select the xilfpga and xilmailbox library from the list of Supported Libraries.

15. Expand the Overview tree and select xilfpga. The configuration options for xilfpga are listed.
16. Configure the xilfpga by providing the base address of the bitstream file (DDR address) and the size (in bytes).
17. Click **OK**. The board support package automatically builds with XilFPGA library included in it.
18. Double-click the `system.mss` file to open it in the Editor view.
19. Scroll-down and locate the Libraries section.
20. Click **Import Examples** adjacent to the XilFPGA entry.

XilFPGA APIs

This section provides detailed descriptions of the XilFPGA library APIs.

Table 476: **Quick Function Reference**

Type	Name	Arguments
u32	XFpga_Initialize	XFpga * InstancePtr
u32	XFpga_PL_BitStream_Load	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags
u32	XFpga_PL_Write	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags
U32	XFpga_PL_Preconfig	XFpga * InstancePtr
U32	XFpga_PL_PostConfig	XFpga * InstancePtr
U32	XFpga_PL_ValidateImage	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags

XFpga_Initialize

This API when called initializes the XFPGA interface with default settings.

Prototype

```
u32 XFpga_Initialize(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_Initialize` function arguments.

Table 477: **XFpga_Initialize Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure.

Returns

Returns Status

- XFPGA_SUCCESS on success
- Error code on failure

XFpga_PL_ValidateImage

This function is used to validate the input params and the Bitstream Image.

Prototype

```
u32 XFpga_PL_ValidateImage(XFpga *InstancePtr, UINTPTR BitstreamImageAddr,
u32 INTPTR AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_ValidateImage` function arguments.

Table 478: **XFpga_PL_ValidateImage Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Used to store the size of Bitstream Image
u32	Flags	Optional for Versal

Returns

- XFPGA_VALIDATE_ERROR
- XFPGA_OPS_NOT_IMPLEMENTED

***Note*:** This API doesn't contain any bitstream image validation functionality with the current version of XilFPGA and it returns xFPGA_VALIDATE_ERROR incase of input parameters validation failure and XFPGA_OPS_NOT_IMPLEMENTED in all other cases.

XFpga_PL_BitStream_Load

The API is used to load the bitstream/PDI file into the PL region.

Prototype

```
u32 XFpga_PL_BitStream_Load(XFpga *InstancePtr, UINTPTR BitstreamImageAddr,
UINTPTR AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_BitStream_Load` function arguments.

Table 479: **XFpga_PL_BitStream_Load Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure.
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Used to store the size of Bitstream Image
u32	Flags	Optional for Versal

Returns

- XFPGA_SUCCESS on success
- Error code on failure.

XFpga_PL_Write

This function write count bytes of configuration data into the PL.

Prototype

```
u32 XFpga_PL_Write(XFpga *InstancePtr, UINTPTR BitstreamImageAddr, UINTPTR
AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_Write` function arguments.

Table 480: **XFpga_PL_Write Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Used to store the size of Bitstream Image
u32	Flags	Optional for Versal

Returns

- XFPGA_SUCCESS on success

- Error code on failure.

XFpga_PL_Preconfig

This function prepare the FPGA to receive configuration data.

Prototype

```
u32 XFpga_PL_Preconfig(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_PL_Preconfig` function arguments.

Table 481: **XFpga_PL_Preconfig Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure.

Returns

- XFPGA_OPS_NOT_IMPLEMENTED

***Note*:** This API doesn't contain any functionality with the current version of XilFPGA and it always returns XFPGA_OPS_NOT_IMPLEMENTED.

XFpga_PL_PostConfig

This function set FPGA to operating state after writing.

Prototype

```
u32 XFpga_PL_PostConfig(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_PL_PostConfig` function arguments.

Table 482: **XFpga_PL_PostConfig Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure

Returns

- XFPGA_OPS_NOT_IMPLEMENTED

Note: This API doesn't contain any functionality with the current version of XilFPGA and it always returns XFPGA_OPS_NOT_IMPLEMENTED.

XilMailbox v1.2

XilMailbox Library API Reference

The XilMailbox library provides the top-level hooks for sending or receiving an inter-processor interrupt (IPI) message using the Zynq® UltraScale+™ MPSoC and Versal ACAP IPI hardware.This library supports Zynq Ultrascale+ MPSoC and Versal platforms. For more details on the IPI interrupts, see the Zynq UltraScale+ MPSoC Technical Reference Manual (UG1085).

The XilMailbox library supports the following features:

- Triggering an IPI to a remote agent.
- Sending an IPI message to a remote agent.
- Callbacks for error and recv IPI events.
- Reading an IPI message.

The following is a list of software initalization events for a given IPI channel:

- IPI Initialization using XMailbox_Initalize() function. This step initializes a library instance for the given IPI channel.
- XMailbox_Send() function triggers an IPI to a remote agent.
- XMailbox_SendData() function sends an IPI message to a remote agent. Message type should be either XILMBOX_MSG_TYPE_REQ (OR) XILMBOX_MSG_TYPE_RESP.
- XMailbox_Recv() function reads an IPI message from a specified source agent. Message type should be either XILMBOX_MSG_TYPE_REQ (OR) XILMBOX_MSG_TYPE_RESP.
- XMailbox_SetCallBack() using this function user can register call backs for recv and error events.

Table 483: **Quick Function Reference**

Type	Name	Arguments
u32	XMailbox_Send	XMailbox * InstancePtr u32 RemoteId u8 Is_Blocking

Table 483: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
u32	XMailbox_SendData	XMailbox * InstancePtr u32 RemoteId void * BufferPtr u32 MsgLen u8 BufferType u8 Is_Blocking
u32	XMailbox_Recv	XMailbox * InstancePtr u32 SourceId void * BufferPtr u32 MsgLen u8 BufferType
s32	XMailbox_SetCallBack	XMailbox * InstancePtr XMailbox_Handler HandlerType CallBackFunc CallBackRef
u32	XMailbox_Initialize	XMailbox * InstancePtr u8 DeviceId
u32	XIpiPs_Init	XMailbox * InstancePtr u8 DeviceId
u32	XIpiPs_Send	XMailbox * InstancePtr u8 Is_Blocking
u32	XIpiPs_SendData	XMailbox * InstancePtr void * MsgBufferPtr u32 MsgLen u8 BufferType u8 Is_Blocking
u32	XIpiPs_PollforDone	XMailbox * InstancePtr
u32	XIpiPs_RecvData	XMailbox * InstancePtr void * MsgBufferPtr u32 MsgLen u8 BufferType
XStatus	XIpiPs_RegisterIrq	void

Table 483: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	XIpiPs_ErrorIntrHandler	void
void	XIpiPs_IntrHandler	void

Functions

XMailbox_Send

This function triggers an IPI to a destination CPU.

Prototype

```
u32 XMailbox_Send(XMailbox *InstancePtr, u32 RemoteId, u8 Is_Blocking);
```

Parameters

The following table lists the `XMailbox_Send` function arguments.

Table 484: **XMailbox_Send Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
u32	RemoteId	Mask of the CPU to which IPI is to be triggered
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

- XST_SUCCESS if successful
- XST_FAILURE if unsuccessful

XMailbox_SendData

This function sends an IPI message to a destination CPU.

Prototype

```
u32 XMailbox_SendData(XMailbox *InstancePtr, u32 RemoteId, void *BufferPtr,
u32 MsgLen, u8 BufferType, u8 Is_Blocking);
```

Parameters

The following table lists the `XMailbox_SendData` function arguments.

Table 485: **XMailbox_SendData Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
u32	RemoteId	Mask of the CPU to which IPI is to be triggered
void *	BufferPtr	Pointer to buffer which contains the message to be sent
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer (XILMBOX_MSG_TYPE_REQ (OR) XILMBOX_MSG_TYPE_RESP)
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

- XST_SUCCESS if successful
- XST_FAILURE if unsuccessful

XMailbox_Recv

This function reads an IPI message.

Prototype

```
u32 XMailbox_Recv(XMailbox *InstancePtr, u32 SourceId, void *BufferPtr, u32
MsgLen, u8 BufferType);
```

Parameters

The following table lists the `XMailbox_Recv` function arguments.

Table 486: **XMailbox_Recv Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
u32	SourceId	Mask for the CPU which has sent the message
void *	BufferPtr	Pointer to buffer to which the read message needs to be stored
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer (XILMBOX_MSG_TYPE_REQ or XILMBOX_MSG_TYPE_RESP)

Returns

- XST_SUCCESS if successful

- XST_FAILURE if unsuccessful

XMailbox_SetCallBack

This routine installs an asynchronous callback function for the given HandlerType.

Prototype

```
s32 XMailbox_SetCallBack(XMailbox *InstancePtr, XMailbox_Handler
HandlerType, void *CallBackFuncPtr, void *CallBackRefPtr);
```

Parameters

The following table lists the `XMailbox_SetCallBack` function arguments.

Table 487: **XMailbox_SetCallBack Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance.
`XMailbox_Handler`	HandlerType	Specifies which callback is to be attached.
Commented parameter CallBackFunc does not exist in function XMailbox_SetCallBack.	CallBackFunc	Address of the callback function.
Commented parameter CallBackRef does not exist in function XMailbox_SetCallBack.	CallBackRef	User data item that is passed to the callback function when it is invoked.

Returns

- XST_SUCCESS when handler is installed.
- XST_INVALID_PARAM when HandlerType is invalid.

XMailbox_Initialize

This function initializes the `XMailbox` instance.

Prototype

```
u32 XMailbox_Initialize(XMailbox *InstancePtr, u8 DeviceId);
```

Parameters

The following table lists the `XMailbox_Initialize` function arguments.

Table 488: **XMailbox_Initialize Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the instance to be worked on
u8	DeviceId	IPI instance to be worked on

Returns

XST_SUCCESS if initialization was successful XST_FAILURE in case of failure

XIpiPs_Init

This function initializes the Zynq UltraScale+ MPSoC Mailbox instance.

Prototype

```
u32 XIpiPs_Init(XMailbox *InstancePtr, u8 DeviceId);
```

Parameters

The following table lists the `XIpiPs_Init` function arguments.

Table 489: **XIpiPs_Init Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the instance to be worked on
u8	DeviceId	IPI instance to be worked on

Returns

XST_SUCCESS if initialization was successful XST_FAILURE in case of failure

XIpiPs_Send

This function triggers an IPI to a destination CPU.

Prototype

```
u32 XIpiPs_Send(XMailbox *InstancePtr, u8 Is_Blocking);
```

Parameters

The following table lists the `XIpiPs_Send` function arguments.

Table 490: **XIpiPs_Send Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the XMailbox instance.
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

XST_SUCCESS in case of success XST_FAILURE in case of failure

XIpiPs_SendData

This function sends an IPI message to a destnation CPU.

Prototype

```
u32 XIpiPs_SendData(XMailbox *InstancePtr, void *MsgBufferPtr, u32 MsgLen,
u8 BufferType, u8 Is_Blocking);
```

Parameters

The following table lists the `XIpiPs_SendData` function arguments.

Table 491: **XIpiPs_SendData Arguments**

Type	Name	Description
XMailbox *	InstancePtr	Pointer to the XMailbox instance
void *	MsgBufferPtr	Pointer to buffer which contains the message to be sent
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer
u8	Is_Blocking	If set, triggers the notification in blocking mode

Returns

XST_SUCCESS in case of success XST_FAILURE in case of failure

XIpiPs_PollforDone

This function polls for an acknowledgement using the Observation Register.

Prototype

```
u32 XIpiPs_PollforDone(XMailbox *InstancePtr);
```

Parameters

The following table lists the `XIpiPs_PollforDone` function arguments.

Table 492: **XIpiPs_PollforDone Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance

Returns

XST_SUCCESS in case of success XST_FAILURE in case of failure

XIpiPs_RecvData

This function reads an IPI message.

Prototype

```
u32 XIpiPs_RecvData(XMailbox *InstancePtr, void *MsgBufferPtr, u32 MsgLen,
u8 BufferType);
```

Parameters

The following table lists the `XIpiPs_RecvData` function arguments.

Table 493: **XIpiPs_RecvData Arguments**

Type	Name	Description
`XMailbox` *	InstancePtr	Pointer to the `XMailbox` instance
void *	MsgBufferPtr	Pointer to buffer to which the read message needs to be stored
u32	MsgLen	Length of the buffer/message
u8	BufferType	Type of buffer

Returns

- XST_SUCCESS if successful
- XST_FAILURE if unsuccessful

XIpiPs_RegisterIrq

Prototype

```
XStatus XIpiPs_RegisterIrq(XScuGic *IntcInstancePtr, XMailbox *InstancePtr,
u32 IpiIntrId);
```

XIpiPs_ErrorIntrHandler

Prototype

```
void XIpiPs_ErrorIntrHandler(void *XMailboxPtr);
```

XIpiPs_IntrHandler

Prototype

```
void XIpiPs_IntrHandler(void *XMailboxPtr);
```

Enumerations

Enumeration XMailbox_Handler

Contains XMAILBOX Handler Types.

Table 494: **Enumeration XMailbox_Handler Values**

Value	Description
XMAILBOX_RECV_HANDLER	For Recv Handler.
XMAILBOX_ERROR_HANDLER	For Error Handler.

Data Structure Index

The following is a list of data structures:

XMailbox

Holds the function pointers for the operations that can be performed.

Declaration

```
typedef struct
{
  u32(* XMbox_IPI_Send)(struct XMboxTag *InstancePtr, u8 Is_Blocking),
  u32(* XMbox_IPI_SendData)(struct XMboxTag *InstancePtr, void *BufferPtr,
u32 MsgLen, u8 BufferType, u8 Is_Blocking),
  u32(* XMbox_IPI_Recv)(struct XMboxTag *InstancePtr, void *BufferPtr, u32
MsgLen, u8 BufferType),
```

```
  XMailbox_RecvHandler RecvHandler,
  XMailbox_ErrorHandler ErrorHandler,
  void * ErrorRefPtr,
  void * RecvRefPtr,
  XMailbox_Agent Agent
} XMailbox;
```

Table 495: **Structure XMailbox member description**

Member	Description
XMbox_IPI_Send	Triggers an IPI to a destination CPU.
XMbox_IPI_SendData	Sends an IPI message to a destination CPU.
XMbox_IPI_Recv	Reads an IPI message.
RecvHandler	Callback for rx IPI event.
ErrorHandler	Callback for error event.
ErrorRefPtr	To be passed to the error interrupt callback.
RecvRefPtr	To be passed to the receive interrupt callback.
Agent	Used to store IPI Channel information.

XilSEM

XilSEM Library API Reference

The Xilinx Soft Error Mitigation (XilSEM) library is a pre-configured, pre-verified solution to detect and optionally correct soft errors in Configuration Memory of Versal ACAPs. A soft error is caused by ionizing radiation and is extremely uncommon in commercial terrestrial operating environments. While a soft error does not damage the device, it carries a small statistical possibility of transiently altering the device behavior.

The XilSEM library does not prevent soft errors; however, it provides a method to better manage the possible system-level effect. Proper management of a soft error can increase reliability and availability, and reduce system maintenance and downtime. In most applications, soft errors can be ignored. In applications where a soft error cannot be ignored, this library documentation provides information to configure the XilSEM library through the CIPS IP core and interact with the XilSEM library at runtime.

The XilSEM library is part of the Platform Loader and Manager (PLM) which is loaded into and runs on the Platform Management Controller (PMC).

When a soft error occurs, one or more bits of information are corrupted. The information affected might be in the device Configuration Memory (which determines the intended behavior of the design) or might be in design memory elements (which determine the state of the design). In the Versal architecture, Configuration Memory includes Configuration RAM and NPI Registers.

Configuration RAM is associated with the Programmable Logic (PL) and is used to configure the function of the design loaded into the PL. This includes function block behavior and function block connectivity. This memory is physically distributed across the PL and is the larger category of Configuration Memory by bit count. Only a fraction of these bits are essential to the correct operation of the associated resources for the currently loaded design.

NPI Registers are associated with other function blocks which can be configured independently from the PL. The Versal Architecture integrated shell is a notable application of function blocks configured in this manner, making it possible for the integrated shell to be operational with the PL in an unconfigured state. This memory is physically distributed throughout the device and is the smaller category of Configuration Memory by bit count. Only a fraction of these bits are essential to the correct operation of the associated resources for the currently loaded design.

Prior to configuring the XilSEM library for use through the CIPS IP core, assess whether the XilSEM library helps meet the soft error mitigation goals of your system's deployment and what mitigation approaches should be included, if any. There are two main considerations:

- Understanding the soft error mitigation requirements for your system
- What design and system actions need to be taken if a soft error occurs

If the effects of soft errors are a concern for your system's deployment, each component in the system design will usually need a soft error rate (SER) budget distributed from a system-level SER budget. To make estimates for the contribution from a Versal ACAP, use the Xilinx SEU Estimator tool, which is tentatively planned for availability after the UG116 publication of production qualification data for Xilinx 7 nm technology. To estimate SER for a deployment, at the minimum you need:

- Target device selection(s) to be deployed
- Estimated number of devices in deployment

In addition to providing a device level estimate, the Xilinx SEU Estimator also estimates the number of soft errors statistically likely to occur across the deployment for a selected operation time interval. After an estimate has been generated, it must be used to evaluate if the SER requirement for the deployment is met. Other design approaches might need to be implemented to reduce the system-level SER. These approaches include addition of mitigation solutions in the system, which might be suitable for implementation in any devices in the system which can offer such features. Versal ACAPs offer many supporting features, including the XilSEM library.

The XilSEM library can be configured to detect or detect and correct soft errors. Therefore, consideration must be given to the system-level response, if any, that must be taken in response to a soft error in the Versal ACAP. If no action is taken when a soft error is detected in the Versal ACAP, the benefits of using the XilSEM library should be assessed and understood. While this is a valid use case, the effects of soft errors in the Versal ACAP should be anticipated and this approach should be reviewed to ensure it supports the system-level soft error mitigation strategy.

If there is no SER target at the system level, then it is unclear what system changes (and trade-offs that come with them) need to be made to mitigate soft errors, including whether a deployment benefits from use of the XilSEM library.

XilSEM Features

The XilSEM library features include:

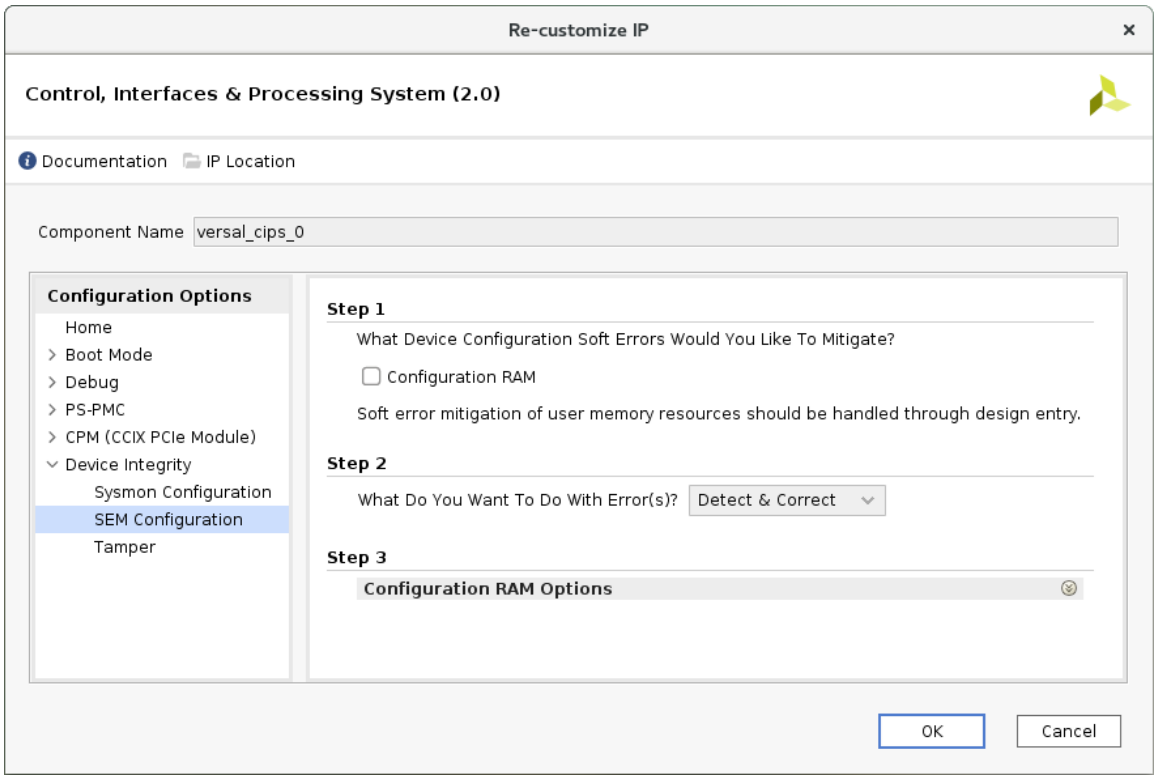
- Integration of silicon features to fully leverage and improve upon the inherent protections for Configuration Memory.
- Scan-based detection of soft errors in Configuration RAM using ECC and CRC techniques.

- Algorithmic correction of soft errors in Configuration RAM using ECC techniques.

XilSEM Configuration

The XilSEM library configuration options available through the CIPS IP core are shown in the following figure. One configuration option is available, to enable the mitigation of soft errors in Configuration RAM.

Figure 9: **XilSEM Library Configuration Options**



XilSEM APIs

API information is tentatively planned for publication after verification and validation of the XilSEM library has completed.

XILINX

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

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

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

Please Read: Important Legal Notices

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

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

Copyright