Xilinx Standalone Library Documentation

Standalone Library v7.6

UG647 (v2021.2) October 13, 2021

XILINX®

Table of Contents

XILINX

Chapter 1

Xilinx Hardware Abstraction Layer APIs

Xilinx Hardware Abstraction Layer API

Assert APIs and Macros

Table 1: **Quick Function Reference**

Type	Name	Arguments
void	Xil_Assert	const char8 * File s32 Line
void	Xil_AssertSetCallback	Xil_AssertCallback Routine
void	XNullHandler	void * NullParameter

Functions

Xil_Assert

Note:

Prototype

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

Parameters

Xil_Assert

Table 2: **Xil_Assert Arguments**

Name	Description
File	filename of the source
Line	linenumber within File

Returns

Xil_AssertSetCallback

Note:

Prototype

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

Parameters

Xil_AssertSetCallback

Table 3: **Xil_AssertSetCallback Arguments**

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

Returns

XNullHandler

***Note*:**

Prototype

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

Parameters

XNullHandler

Table 4: **XNullHandler Arguments**

Name	Description
NullParameter	arbitrary void pointer and not used.

Returns

Definitions

#Define Xil_AssertVoid

Description

Parameters

```
Xil_AssertVoid
```

Table 5: **Xil_AssertVoid Arguments**

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

Returns

#Define Xil_AssertNonvoid

Description

Parameters

```
Xil_AssertNonvoid
```

Table 6: **Xil_AssertNonvoid Arguments**

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

Returns

#Define Xil_AssertVoidAlways

Description

Returns

#Define Xil_AssertNonvoidAlways

Description

Returns

Variables

u32 Xil_AssertStatus

s32 Xil_AssertWait

Table 7: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
INLINE void	Xil_Out16BE	UINTPTR Addr u16 Value
INLINE void	Xil_Out32BE	UINTPTR Addr u32 Value
INLINE u16	Xil_In16LE	UINTPTR Addr
INLINE u32	Xil_In32LE	UINTPTR Addr
INLINE void	Xil_Out16LE	UINTPTR Addr u16 Value
INLINE void	Xil_Out32LE	UINTPTR Addr u32 Value
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 int	Xil_SecureOut32	UINTPTR Addr u32 Value
u16	Xil_EndianSwap16	u16 Data

Table 7: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
u32	Xil_EndianSwap32	u32 Data

Functions

Xil_In16BE

Prototype

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

Parameters

Xil_In16BE

Table 8: **Xil_In16BE Arguments**

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

Returns

Xil_In32BE

Prototype

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

Parameters

Xil_In32BE

Table 9: **Xil_In32BE Arguments**

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

Returns

Xil_Out16BE

Prototype

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

Parameters

Xil_Out16BE

Table 10: **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

Prototype

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

Parameters

Xil_Out32BE

Table 11: **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

Prototype

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

Parameters

Xil_In16LE

Table 12: **Xil_In16LE Arguments**

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

Returns

Xil_In32LE

Prototype

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

Parameters

Xil_In32LE

Table 13: **Xil_In32LE Arguments**

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

Returns

Xil_Out16LE

Prototype

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

Parameters

Xil_Out16LE

Table 14: **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

Prototype

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

Parameters

Xil_Out32LE

Table 15: **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_In8

Prototype

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

Parameters

Xil_In8

Table 16: **Xil_In8 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

Xil_In16

Prototype

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

Parameters

Xil_In16

Table 17: **Xil_In16 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

Xil_In32

Prototype

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

Parameters

Xil_In32

Table 18: **Xil_In32 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

Xil_In64

Prototype

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

Parameters

Xil_In64

Table 19: **Xil_In64 Arguments**

Name	Description
Addr	contains the address to perform the input operation

Returns

Xil_Out8

Prototype

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

Parameters

Xil_Out8

Table 20: **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

Xil_Out16

Prototype

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

Parameters

Xil_Out16

Table 21: **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

Xil_Out32

Prototype

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

Parameters

Xil_Out32

Table 22: **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

Xil_Out64

Prototype

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

Parameters

Xil_Out64

Table 23: **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

Xil_SecureOut32

Prototype

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

Parameters

Xil_SecureOut32

Table 24: **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

Xil_EndianSwap16

Prototype

```
u16 Xil_EndianSwap16(u16 Data) INLINE __attribute__((always_inline));
```

Parameters

Xil_EndianSwap16

Table 25: **Xil_EndianSwap16 Arguments**

Name	Description
Data	16-bit value to be converted

Returns

Xil_EndianSwap32

Prototype

```
u32 Xil_EndianSwap32(u32 Data) INLINE __attribute__((always_inline));
```

Parameters

Xil_EndianSwap32

Table 26: **Xil_EndianSwap32 Arguments**

Name	Description
Data	32-bit value to be converted

Returns

Hardware Platform Information

Table 27: **Quick Function Reference**

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

Functions

XGetPlatform_Info

Prototype

```
u32 XGetPlatform_Info();
```

Returns

XGet_Zynq_UltraMp_Platform_info

Prototype

```
u32 XGet_Zynq_UltraMp_Platform_info();
```

Returns

XGetPSVersion_Info

Prototype

```
u32 XGetPSVersion_Info();
```

Returns

Basic Data types for Xilinx Software IP

Customized APIs for Memory Operations

Table 28: **Quick Function Reference**

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

Functions

Xil_MemCpy

Prototype

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

Parameters

Xil_MemCpy

Table 29: **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

Definitions

#Define XIL_MEM_H

Description

Xilinx Software Status Codes

Test Utilities for Memory and Caches

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

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

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*:**

Table 30: **Quick Function Reference**

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

Table 30: **Quick Function Reference** *(cont'd)*

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

***Note*:**

Prototype

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

Parameters

Xil_TestMem32

Table 31: **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

Xil_TestMem16

***Note*:**

Prototype

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

Parameters

Xil_TestMem16

Table 32: **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

***Note*:**

Prototype

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

Parameters

Xil_TestMem8

Table 33: **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

RotateLeft

Prototype

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

Parameters

RotateLeft

Table 34: **RotateLeft Arguments**

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

Returns

RotateRight

Prototype

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

Parameters

RotateRight

Table 35: **RotateRight Arguments**

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

Returns

Xil_TestDCacheRange

Prototype

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

Returns

Xil_TestDCacheAll

Prototype

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

Returns

Xil_TestICacheRange

***Note*:**

Prototype

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

Returns

Xil_TestICacheAll

Note:

Prototype

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

Returns

Xil_TestIO8

Prototype

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

Parameters

Xil_TestIO8

Table 36: **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

Xil_TestIO16

Prototype

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

Parameters

Xil_TestIO16

Table 37: **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

XTtO

Prototype

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

Parameters

```
Xil_TestIO32
```

Table 38: **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

MicroBlaze Processor APIs

MicroBlaze Processor API

MicroBlaze Pseudo-asm Macros and Interrupt Handling APIs

mb_interface.h

Table 39: **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

Prototype

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

Parameters

microblaze_register_handler

Table 40: **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

microblaze_register_exception_handler

Prototype

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

Parameters

microblaze_register_exception_handler

Table 41: **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

MicroBlaze Exception APIs

***Note*:**

Table 42: **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

Prototype

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

Parameters

Xil_ExceptionNullHandler

Table 43: **Xil_ExceptionNullHandler Arguments**

Name	Description
Data	unused by this function.

Xil_ExceptionInit

Prototype

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

Xil_ExceptionEnable

Prototype

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

Xil_ExceptionDisable

Prototype

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

Xil_ExceptionRegisterHandler

Prototype

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

Parameters

Xil_ExceptionRegisterHandler

Table 46: **Quick Function Reference**

Type	Name	Arguments
void	Xil_DCacheDisable	void
void	Xil_ICacheDisable	void

Functions

Xil_DCacheDisable

Prototype

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

Returns

Xil_ICacheDisable

Prototype

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

Returns

Definitions

#Define Xil_L1DCacheInvalidate

Description

***Note*:**

#Define Xil_L2CacheInvalidate

Description

***Note*:**

#Define Xil_L1DCacheInvalidateRange

Description

***Note*:**

Parameters

`Xil_L1DCacheInvalidateRange`

Table 47: **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

***Note*:**

Parameters

`Xil_L2CacheInvalidateRange`

Table 48: **Xil_L2CacheInvalidateRange Arguments**

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

#Define Xil_L1DCacheFlushRange

Description

Parameters

`Xil_L1DCacheFlushRange`

Table 49: **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

Parameters

`Xil_L2CacheFlushRange`

Table 50: **Xil_L2CacheFlushRange Arguments**

Name	Description
Addr	the starting address of the range to be flushed.

Table 50: **Xil_L2CacheFlushRange Arguments** *(cont'd)*

Name	Description
Len	length in byte to be flushed.

#Define Xil_L1DCacheFlush

Description

#Define Xil_L2CacheFlush

Description

#Define Xil_L1ICacheInvalidateRange

Description

Parameters

`Xil_L1ICacheInvalidateRange`

Table 51: **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

#Define Xil_L1DCacheEnable

Description

Note:

#Define Xil_L1DCacheDisable

Description

Note:

#Define Xil_L1ICacheEnable

Description

Note:

#Define Xil_L1ICacheDisable

Description

Note:

#Define Xil_DCacheEnable

Description

#Define Xil_ICacheEnable

Description

#Define Xil_DCacheInvalidate

Description

#Define Xil_DCacheInvalidateRange

Description

Table 53: **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

Parameters

```
putfslx
```

Table 55: **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

Parameters

```
tgetfslx
```

Table 56: **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

Parameters

```
tputfslx
```

Table 57: **tputfslx Arguments**

Name	Description
id	FSL identifier
flags	valid FSL macro flags

#Define getdfslx

Description

Parameters

```
getdfslx
```

Table 58: **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

Parameters

```
putdfslx
```

Table 59: **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

Parameters

```
tgetdfslx
```

Table 60: **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

Parameters

```
tputdfslx
```

Table 61: **tputdfslx Arguments**

Name	Description
var	FSL identifier
flags	valid FSL macro flags

MicroBlaze PVR Access Routines and Macros

```
microblaze_get_pvr()
```

Note: `pvr.h`

Table 62: **Quick Function Reference**

Type	Name	Arguments
int	microblaze_get_pvr	pvr-

Functions

microblaze_get_pvr

Prototype

```
int microblaze_get_pvr(pvr_t *pvr);
```

Parameters

microblaze_get_pvr

Table 63: **microblaze_get_pvr Arguments**

Name	Description
pvr-	address of PVR data structure to be populated

Returns

Definitions

#Define MICROBLAZE_PVR_IS_FULL

Description

Parameters

MICROBLAZE_PVR_IS_FULL

Table 64: **MICROBLAZE_PVR_IS_FULL Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_BARREL

Description

Parameters

```
MICROBLAZE_PVR_USE_BARREL
```

Table 65: **MICROBLAZE_PVR_USE_BARREL Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_DIV

Description

Parameters

```
MICROBLAZE_PVR_USE_DIV
```

Table 66: **MICROBLAZE_PVR_USE_DIV Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_HW_MUL

Description

Parameters

```
MICROBLAZE_PVR_USE_HW_MUL
```

Table 67: **MICROBLAZE_PVR_USE_HW_MUL Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_FPU

Description

Parameters

```
MICROBLAZE_PVR_USE_FPU
```

Table 68: **MICROBLAZE_PVR_USE_FPU Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_ICACHE

Description

Parameters

```
MICROBLAZE_PVR_USE_ICACHE
```

Table 69: **MICROBLAZE_PVR_USE_ICACHE Arguments**

Name	Description
_pvr	pvr data structure

#Define MICROBLAZE_PVR_USE_DCACHE

Description

Parameters

```
MICROBLAZE_PVR_USE_DCACHE
```

Table 70: **MICROBLAZE_PVR_USE_DCACHE Arguments**

Name	Description
_pvr	pvr data structure

Sleep Routines for MicroBlaze Processor

`microblaze_sleep.h`

Note: `microblaze_sleep.h`

Table 71: **Quick Function Reference**

Type	Name	Arguments
u32	Xil_SetMBFrequency	u32 Val
u32	Xil_GetMBFrequency	void
void	MB_Sleep	MilliSeconds-

Functions

Xil_SetMBFrequency

Note:

Prototype

```
u32 Xil_SetMBFrequency(u32 Val);
```

Parameters

`Xil_SetMBFrequency`

Table 72: **Xil_SetMBFrequency Arguments**

Name	Description
Val	- Frequency value to be set

Returns

Xil_GetMBFrequency

Prototype

```
u32 Xil_GetMBFrequency();
```

Returns

MB_Sleep

Note:

Prototype

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

Parameters

MB_Sleep

Table 73: **MB_Sleep Arguments**

Name	Description
MilliSeconds-	Delay time in milliseconds.

Returns

Arm Processor Common APIs

Arm Processor Exception Handling

Table 74: **Quick Function Reference**

Type	Name	Arguments
void	Xil_ExceptionRegisterHandler	u32 Exception_id Xil_ExceptionHandler Handler void * Data
void	Xil_ExceptionRemoveHandler	u32 Exception_id
void	Xil_GetExceptionRegisterHandler	u32 Exception_id Xil_ExceptionHandler * Handler void ** Data
void	Xil_ExceptionInit	void
void	Xil_DataAbortHandler	void
void	Xil_PrefetchAbortHandler	void
void	Xil_UndefinedExceptionHandler	void

Functions

Xil_ExceptionRegisterHandler

Prototype

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

Parameters

Xil_ExceptionRegisterHandler

Table 75: **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

Xil_ExceptionRemoveHandler

Prototype

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

Parameters

Xil_ExceptionRemoveHandler

Table 76: **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

Xil_GetExceptionRegisterHandler

Prototype

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

Parameters

Xil_GetExceptionRegisterHandler

Table 77: **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

Xil_ExceptionInit

Prototype

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

Returns

Xil_DataAbortHandler

Prototype

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

Returns

Xil_PrefetchAbortHandler

Prototype

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

Returns

Xil_UndefinedExceptionHandler

Prototype

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

Returns

Definitions

Define Xil_ExceptionEnableMask

Definition

```
#define Xil_ExceptionEnableMask        {                                    \
          register u32 Reg __asm("cpsr"); \
          mtcpsr((Reg) & (~((Mask) & XIL_EXCEPTION_ALL))); \
        }
```

Description

Note: Xil_ExceptionEnableMask(Mask)

Define Xil_ExceptionEnable

Definition

```
#define Xil_ExceptionEnable

              Xil_ExceptionEnableMask
            (XIL_EXCEPTION_IRQ)
```

Description

Note:

Define Xil_ExceptionDisableMask

Definition

```
#define Xil_ExceptionDisableMask
{                                       \
          register u32 Reg __asm("cpsr"); \
          mtcpsr((Reg) | ((Mask) & XIL_EXCEPTION_ALL)); \
        }
```

Description

Note: Xil_ExceptionDisableMask(Mask)

Define Xil_ExceptionDisable

Definition

```
#define Xil_ExceptionDisable

                Xil_ExceptionDisableMask
            (XIL_EXCEPTION_IRQ)
```

Description

Note:

Define Xil_EnableNestedInterrupts

Definition

```
#define Xil_EnableNestedInterrupts          __asm__ __volatile__ ("stmfd
sp!, {lr}"); \
        __asm__ __volatile__ ("mrs     lr, spsr");  \
        __asm__ __volatile__ ("stmfd   sp!, {lr}"); \
        __asm__ __volatile__ ("msr     cpsr_c, #0x1F"); \
        __asm__ __volatile__ ("stmfd   sp!, {lr}");
```

Description

Note:

Define Xil_DisableNestedInterrupts

Definition

```
#define Xil_DisableNestedInterrupts          __asm__ __volatile__ ("ldmfd
sp!, {lr}");   \
        __asm__ __volatile__ ("msr     cpsr_c, #0x92"); \
        __asm__ __volatile__ ("ldmfd   sp!, {lr}"); \
        __asm__ __volatile__ ("msr     spsr_cxsf, lr"); \
        __asm__ __volatile__ ("ldmfd   sp!, {lr}"); \
```

Description

Note:

`Xil_EnableNestedInterrupts()`

Chapter 4

Arm Cortex-R5F Processor APIs

Arm Cortex-R5F Processor API

Arm Cortex-R5F Processor Boot Code

Arm Cortex-R5F Processor MPU specific APIs

	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*:**

Table 78: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	addr u32 attrib
void	Xil_EnableMPU	void
void	Xil_DisableMPU	void
u32	Xil_SetMPURegion	INTPTR addr u64 size u32 attrib
u32	Xil_UpdateMPUConfig	u32 reg_num INTPTR address u32 size u32 attrib

Table 78: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_GetMPUConfig	XMpu_Config mpuconfig
u32	Xil_GetNumOfFreeRegions	void
u32	Xil_GetNextMPURegion	void
u32	Xil_DisableMPURegionByRegNum	u32 reg_num
u16	Xil_GetMPUFreeRegMask	void
u32	Xil_SetMPURegionByRegNum	u32 reg_num INTPTR addr u64 size u32 attrib
void *	Xil_MemMap	UINTPTR Physaddr size_t size u32 flags

Functions

Xil_SetTlbAttributes

Prototype

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

Parameters

Xil_SetTlbAttributes

Table 79: **Xil_SetTlbAttributes Arguments**

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

Returns

Xil_EnableMPU

Prototype

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

Returns

Xil_DisableMPU

Prototype

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

Returns

Xil_SetMPURegion

Prototype

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

Parameters

Xil_SetMPURegion

Table 80: **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

Xil_UpdateMPUConfig

Prototype

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

Parameters

Xil_UpdateMPUConfig

Table 81: **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

Xil_GetMPUConfig

Prototype

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

Parameters

Xil_GetMPUConfig

Table 82: **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

Xil_GetNumOfFreeRegions

Prototype

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

Returns

Xil_GetNextMPURegion

Prototype

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

Returns

Xil_DisableMPURegionByRegNum

Prototype

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

Parameters

Xil_DisableMPURegionByRegNum

Table 83: **Xil_DisableMPURegionByRegNum Arguments**

Name	Description
reg_num	The region number to be disabled

Returns

Xil_GetMPUFreeRegMask

Prototype

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

Returns

Xil_SetMPURegionByRegNum

Prototype

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

Parameters

Xil_SetMPURegionByRegNum

Table 84: **Xil_SetMPURegionByRegNum Arguments**

Name	Description
reg_num	The region number to be enabled

Table 84: **Xil_SetMPURegionByRegNum Arguments** *(cont'd)*

Name	Description
addr	32 bit address for start of the region.
size	Requested size of the region.
attrib	Attribute for the corresponding region.

Returns

Xil_MemMap

Prototype

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

Parameters

Xil_MemMap

Table 85: **Xil_MemMap Arguments**

Name	Description
Physaddr	is base physical address at which to start mapping. NULL in Physaddr masks possible mapping errors.
size	of region to be mapped.
flags	used to set translation table.

Returns

Arm Cortex-R5F Processor Cache Functions

Table 86: **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_DCacheInvalidateLine	INTPTR adr
void	Xil_DCacheFlushLine	INTPTR adr
void	Xil_DCacheStoreLine	INTPTR adr
void	Xil_ICacheEnable	void
void	Xil_ICacheDisable	void
void	Xil_ICacheInvalidate	void
void	Xil_ICacheInvalidateRange	INTPTR adr u32 len
void	Xil_ICacheInvalidateLine	INTPTR adr

Prototype

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

Returns

Xil_DCacheDisable

Prototype

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

Returns

Xil_DCacheInvalidate

Prototype

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

Returns

Xil_DCacheInvalidateRange

Prototype

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

Parameters

Xil_DCacheInvalidateRange	

Table 87: **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

Xil_DCacheFlush

Prototype

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

Returns

Xil_DCacheFlushRange

Prototype

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

Parameters

Xil_DCacheFlushRange

Table 88: **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

Xil_DCacheInvalidateLine

Note:

Prototype

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

Parameters

Xil_DCacheInvalidateLine

Table 89: **Xil_DCacheInvalidateLine Arguments**

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

Returns

Xil_DCacheFlushLine

Note:

Prototype

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

Parameters

Xil_DCacheFlushLine

Table 90: **Xil_DCacheFlushLine Arguments**

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

Returns

Xil_DCacheStoreLine

***Note*:**

Prototype

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

Parameters

Xil_DCacheStoreLine

Table 91: **Xil_DCacheStoreLine Arguments**

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

Returns

Xil_ICacheEnable

Prototype

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

Returns

Xil_ICacheDisable

Prototype

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

Returns

Xil_ICacheInvalidate

Prototype

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

Returns

Xil_ICacheInvalidateRange

Prototype

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

Parameters

Xil_ICacheInvalidateRange

Returns

Xil_ICacheInvalidateLine

***Note*:**

Prototype

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

Parameters

Xil_ICacheInvalidateLine

Table 93: **Xil_ICacheInvalidateLine Arguments**

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

Returns

Arm Cortex-R5F Time Functions

Table 94: **Quick Function Reference**

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

Functions

XTime_SetTime

***Note*:**

Prototype

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

Parameters

XTime_SetTime

Table 95: **XTime_SetTime Arguments**

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

Returns

XTime_GetTime

Prototype

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

Parameters

XTime_GetTime

Table 96: **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

Arm Cortex-R5F Event Counters Functions

Table 97: **Quick Function Reference**

Type	Name	Arguments
void	Xpm_SetEvents	s32 PmcrCfg
void	Xpm_GetEventCounters	u32 * PmCtrValue
u32	Xpm_DisableEvent	EventCntrId
u32	Xpm_SetUpAnEvent	u32 EventID
u32	Xpm_GetEventCounter	EventCntrId u32 * CntVal
void	Xpm_DisableEventCounters	void
void	Xpm_EnableEventCounters	void
void	Xpm_ResetEventCounters	void
void	Xpm_SleepPerfCounter	u32 delay u64 frequency

Functions

Xpm_SetEvents

Prototype

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

Parameters

Xpm_SetEvents

Table 98: **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

Xpm_GetEventCounters

Prototype

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

Parameters

Xpm_GetEventCounters

Table 99: **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

Xpm_DisableEvent

Prototype

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

Parameters

Xpm_DisableEvent

Table 100: **Xpm_DisableEvent Arguments**

Name	Description
EventCntrId	Event Counter ID. The counter ID is the same that was earlier returned through a call to Xpm_SetUpAnEvent. Cortex-R5F supports only 3 counters. The valid values are 0, 1, or 2.

Returns

Xpm_SetUpAnEvent

Prototype

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

Parameters

Xpm_SetUpAnEvent

Table 101: **Xpm_SetUpAnEvent Arguments**

Name	Description
EventID	For valid values, please refer xpm_counter.h.

Returns

Send Feedback

Xpm_EnableEventCounters

Prototype

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

Returns

Xpm_ResetEventCounters

Prototype

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

Returns

Xpm_SleepPerfCounter

Prototype

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

Parameters

Xpm_SleepPerfCounter

Table 103: **Xpm_SleepPerfCounter Arguments**

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

Returns

Arm Cortex-R5F Processor Specific Include Files

Arm Cortex-R5F Peripheral Definitions

XILINX

Chapter 5

Arm Cortex-A9 Processor APIs

Arm Cortex-A9 Processor API

Arm Cortex-A9 Processor Boot Code

	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
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

Note:

Arm Cortex-A9 Processor Cache Functions

Table 104: **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

Table 104: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
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
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

Table 104: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	Xil_L1DCacheStoreLine	u32 adr
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
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

Prototype

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

Returns

Xil_DCacheDisable

Prototype

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

Returns

Xil_DCacheInvalidate

Prototype

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

Returns

Xil_DCacheInvalidateRange

Prototype

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

Parameters

Xil_DCacheInvalidateRange

Table 105: **Xil_DCacheInvalidateRange Arguments**

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

Returns

Xil_DCacheFlush

Prototype

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

Returns

Xil_DCacheFlushRange

Prototype

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

Parameters

Xil_DCacheFlushRange

Table 106: **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

Xil_ICacheEnable

Prototype

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

Returns

Xil_ICacheDisable

Prototype

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

Returns

Xil_ICacheInvalidate

Prototype

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

Returns

Xil_ICacheInvalidateRange

Prototype

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

Parameters

Xil_ICacheInvalidateRange

Table 107: **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

Xil_DCacheInvalidateLine

Note:

Prototype

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

Parameters

Xil_DCacheInvalidateLine

Table 108: **Xil_DCacheInvalidateLine Arguments**

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

Returns

Xil_DCacheFlushLine

Note:

Prototype

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

Parameters

Xil_DCacheFlushLine

Table 109: **Xil_DCacheFlushLine Arguments**

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

Returns

Xil_DCacheStoreLine

Note:

Prototype

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

Parameters

Xil_DCacheStoreLine

Table 110: **Xil_DCacheStoreLine Arguments**

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

Returns

Xil_ICacheInvalidateLine

***Note*:**

Prototype

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

Parameters

Xil_ICacheInvalidateLine

Table 111: **Xil_ICacheInvalidateLine Arguments**

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

Returns

Xil_L1DCacheEnable

Prototype

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

Returns

Xil_L1DCacheDisable

Prototype

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

Returns

Xil_L1DCacheInvalidate

Note:

Prototype

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

Returns

Xil_L1DCacheInvalidateLine

Note:

Prototype

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

Parameters

Xil_L1DCacheInvalidateLine

Table 112: **Xil_L1DCacheInvalidateLine Arguments**

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

Returns

Xil_L1DCacheInvalidateRange

Prototype

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

Parameters

Xil_L1DCacheInvalidateRange

Table 113: **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

Xil_L1DCacheFlush

***Note*:**

Prototype

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

Returns

Xil_L1DCacheFlushLine

***Note*:**

Prototype

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

Parameters

Xil_L1DCacheFlushLine

Table 114: **Xil_L1DCacheFlushLine Arguments**

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

Returns

Xil_L1DCacheFlushRange

Prototype

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

Parameters

Xil_L1DCacheFlushRange

Table 115: **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

Xil_L1DCacheStoreLine

***Note*:**

Prototype

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

Parameters

Xil_L1DCacheStoreLine

Table 116: **Xil_L1DCacheStoreLine Arguments**

Name	Description
adr	Address to be stored.

Returns

Xil_L1ICacheEnable

Prototype

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

Returns

Xil_L1ICacheDisable

Prototype

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

Returns

Xil_L1ICacheInvalidate

Prototype

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

Returns

Xil_L1ICacheInvalidateLine

***Note*:**

Prototype

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

Parameters

Xil_L1ICacheInvalidateLine

Table 117: **Xil_L1ICacheInvalidateLine Arguments**

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

Returns

Xil_L1ICacheInvalidateRange

Prototype

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

Parameters

Xil_L1ICacheInvalidateRange

Table 118: **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

Xil_L2CacheEnable

Prototype

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

Returns

Xil_L2CacheDisable

Prototype

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

Returns

Xil_L2CacheInvalidate

Prototype

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

Returns

Xil_L2CacheInvalidateLine

***Note*:**

Prototype

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

Parameters

Xil_L2CacheInvalidateLine

Table 119: **Xil_L2CacheInvalidateLine Arguments**

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

Returns

Xil_L2CacheInvalidateRange

Prototype

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

Parameters

Xil_L2CacheInvalidateRange

Table 120: **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

Xil_L2CacheFlush

Prototype

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

Returns

Xil_L2CacheFlushLine

***Note*:**

Prototype

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

Parameters

Xil_L2CacheFlushLine

Table 121: **Xil_L2CacheFlushLine Arguments**

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

Returns

Xil_L2CacheFlushRange

Prototype

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

Parameters

Xil_L2CacheFlushRange

Table 122: **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

Xil_L2CacheStoreLine

***Note*:**

Prototype

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

Parameters

Xil_L2CacheStoreLine

Table 123: **Xil_L2CacheStoreLine Arguments**

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

Returns

Arm Cortex-A9 Processor MMU Functions

Table 124: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	INTPTR Addr u32 attrib
void	Xil_EnableMMU	void
void	Xil_DisableMMU	void
void *	Xil_MemMap	UINTPTR PhysAddr size_t size u32 flags

Functions

Xil_SetTlbAttributes

Note:

Prototype

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

Parameters

Xil_SetTlbAttributes

Table 125: **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

Xil_EnableMMU

Prototype

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

Returns

Xil_DisableMMU

Note:

Prototype

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

Returns

Xil_MemMap

***Note*:**

Prototype

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

Parameters

Xil_MemMap

Table 126: **Xil_MemMap Arguments**

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

Returns

Arm Cortex-A9 Time Functions

Table 127: **Quick Function Reference**

Type	Name	Arguments
void	XTime_SetTime	XTime Xtime_Global

Table 127: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
void	XTime_GetTime	XTime * Xtime_Global

Functions

XTime_SetTime

***Note*:**

Prototype

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

Parameters

XTime_SetTime

Table 128: **XTime_SetTime Arguments**

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

Returns

XTime_GetTime

***Note*:**

Prototype

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

Parameters

XTime_GetTime

Table 129: **XTime_GetTime Arguments**

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

Returns

Arm Cortex-A9 Event Counter Function

***Note*:**

Table 130: **Quick Function Reference**

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

Functions

Xpm_SetEvents

Prototype

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

Parameters

Xpm_SetEvents

Table 131: **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

Xpm_GetEventCounters

Prototype

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

Parameters

Xpm_GetEventCounters

Table 132: **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

PL310 L2 Event Counters Functions

Table 133: **Quick Function Reference**

Type	Name	Arguments
void	XL2cc_EventCtrInit	s32 Event0 s32 Event1
void	XL2cc_EventCtrStart	void
void	XL2cc_EventCtrStop	u32 * EveCtr0 u32 * EveCtr1

Functions

XL2cc_EventCtrInit

***Note*:**

Prototype

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

Parameters

XL2cc_EventCtrInit

Table 134: **XL2cc_EventCtrInit Arguments**

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

Returns

XL2cc_EventCtrStart

Prototype

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

Returns

XL2cc_EventCtrStop

Prototype

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

Parameters

XL2cc_EventCtrStop

Table 135: **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

Arm Cortex-A9 Processor and pl310 Errata Support

***Note*:**

Definitions

Define CONFIG_ARM_ERRATA_742230

Definition

```
#define CONFIG_ARM_ERRATA_7422301
```

Description

Define CONFIG_ARM_ERRATA_743622

Definition

```
#define CONFIG_ARM_ERRATA_7436221
```

Description

Define CONFIG_ARM_ERRATA_775420

Definition

```
#define CONFIG_ARM_ERRATA_7754201
```

Description

Define CONFIG_ARM_ERRATA_794073

Definition

```
#define CONFIG_ARM_ERRATA_7940731
```

Description

Define CONFIG_PL310_ERRATA_588369

Definition

```
#define CONFIG_PL310_ERRATA_5883691
```

Description

Define CONFIG_PL310_ERRATA_727915

Definition

```
#define CONFIG_PL310_ERRATA_7279151
```

Description

Arm Cortex-A9 Processor Specific Include Files

Chapter 6

Arm Cortex-A53 32-bit Processor APIs

Arm Cortex-A53 32-bit Processor API

Arm Cortex-A53 32-bit Processor Boot Code

	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

***Note*:**

Arm Cortex-A53 32-bit Processor Cache Functions

Table 136: **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

Table 136: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
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

Prototype

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

Returns

Xil_DCacheDisable

Prototype

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

Returns

Xil_DCacheInvalidate

Note:

Prototype

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

Returns

Xil_DCacheInvalidateRange

Note:

Prototype

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

Parameters

Xil_DCacheInvalidateRange	

Table 137: **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

Xil_DCacheFlush

Prototype

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

Returns

Xil_DCacheFlushRange

Prototype

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

Parameters

Xil_DCacheFlushRange

Table 138: **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

Xil_DCacheInvalidateLine

***Note*:**

Prototype

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

Parameters

Xil_DCacheInvalidateLine

Table 139: **Xil_DCacheInvalidateLine Arguments**

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

Returns

Xil_DCacheFlushLine

***Note*:**

Prototype

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

Parameters

Xil_DCacheFlushLine

Table 140: **Xil_DCacheFlushLine Arguments**

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

Returns

Xil_ICacheInvalidateLine

***Note*:**

Prototype

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

Parameters

Xil_ICacheInvalidateLine

Table 141: **Xil_ICacheInvalidateLine Arguments**

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

Returns

Xil_ICacheEnable

Prototype

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

Returns

Xil_ICacheDisable

Prototype

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

Returns

Xil_ICacheInvalidate

Prototype

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

Returns

Xil_ICacheInvalidateRange

Prototype

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

Parameters

Xil_ICacheInvalidateRange

Table 142: **Xil_ICacheInvalidateRange Arguments**

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

Table 142: **Xil_ICacheInvalidateRange Arguments** *(cont'd)*

Name	Description
len	Length of the range to be invalidated in bytes.

Returns

Arm Cortex-A53 32-bit Processor MMU Handling

***Note*:**

Table 143: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	UINTPTR Addr u32 attrib
void	Xil_EnableMMU	void
void	Xil_DisableMMU	void

Functions

Xil_SetTlbAttributes

***Note*:**

Prototype

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

Parameters

Xil_SetTlbAttributes

Table 144: **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

Xil_EnableMMU

Prototype

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

Returns

Xil_DisableMMU

***Note*:**

Prototype

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

Returns

Arm Cortex-A53 32-bit Mode Time Functions

Table 145: **Quick Function Reference**

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

Functions

XTime_SetTime

Prototype

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

Parameters

XTime_SetTime

Table 146: **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

XTime_GetTime

Prototype

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

Parameters

XTime_GetTime

Table 147: **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

Arm Cortex-A53 32-bit Processor Specific Include Files

Chapter 7

Arm Cortex-A53 64-bit Processor

	Memory Range	Definition in Translation Table
DDR	0x0000000000 - 0x007FFFFFFF	Normal write-back Cacheable
PL	0x0080000000 - 0x00BFFFFFFF	Strongly Ordered
QSPI, lower PCIe	0x00C0000000 - 0x00EFFFFFFF	Strongly Ordere
Reserved	0x00F0000000 - 0x00F7FFFFFF	Unassigned
STM Coresight	0x00F8000000 - 0x00F8FFFFFF	Strongly Ordered
GIC	0x00F9000000 - 0x00F91FFFFF	Strongly Ordered
Reserved	0x00F9200000 - 0x00FCFFFFFF	Unassigned
FPS, LPS slaves	0x00FD000000 - 0x00FFBFFFFF	Strongly Ordered
CSU, PMU	0x00FFC00000 - 0x00FFDFFFFF	Strongly Ordered
TCM, OCM	0x00FFE00000 - 0x00FFFFFFFF	Normal inner write-back cacheable
Reserved	0x0100000000 - 0x03FFFFFFFF	Unassigned
PL, PCIe	0x0400000000 - 0x07FFFFFFFF	Strongly Ordered
DDR	0x0800000000 - 0x0FFFFFFFFF	Normal inner write-back cacheable
PL, PCIe	0x1000000000 - 0xBFFFFFFFFF	Strongly Ordered
Reserved	0xC000000000 - 0xFFFFFFFFFF	Unassigned

***Note*:**

Arm Cortex-A53 64-bit Processor Cache Functions

Table 148: **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
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

Prototype

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

Returns

Xil_DCacheDisable

Prototype

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

Returns

Xil_DCacheInvalidate

Note:

Prototype

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

Returns

Xil_DCacheInvalidateRange

Note:

Prototype

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

Parameters

Xil_DCacheInvalidateRange

Table 149: **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

Xil_DCacheInvalidateLine

Note:

Prototype

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

Parameters

Xil_DCacheInvalidateLine

Table 150: **Xil_DCacheInvalidateLine Arguments**

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

Returns

Xil_DCacheFlush

Prototype

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

Returns

Xil_DCacheFlushLine

***Note*:**

Prototype

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

Parameters

`Xil_DCacheFlushLine`

Table 151: **Xil_DCacheFlushLine Arguments**

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

Returns

Xil_ICacheEnable

Prototype

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

Returns

Xil_ICacheDisable

Prototype

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

Returns

Xil_ICacheInvalidate

Prototype

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

Returns

Xil_ICacheInvalidateRange

Prototype

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

Parameters

Xil_ICacheInvalidateRange

Table 152: **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

Xil_ICacheInvalidateLine

***Note*:**

Prototype

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

Parameters

Xil_ICacheInvalidateLine

Table 153: **Xil_ICacheInvalidateLine Arguments**

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

Returns

Xil_ConfigureL1Prefetch

***Note*:**

Prototype

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

Parameters

Xil_ConfigureL1Prefetch

Table 154: **Xil_ConfigureL1Prefetch Arguments**

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

Returns

Arm Cortex-A53 64-bit Processor MMU Handling

***Note*:**

Table 155: **Quick Function Reference**

Type	Name	Arguments
void	Xil_SetTlbAttributes	UINTPTR Addr u64 attrib

Functions

Xil_SetTlbAttributes

***Note*:**

Prototype

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

Parameters

Xil_SetTlbAttributes

Table 156: **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.

Prototype

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

Parameters

XTime_SetTime

Table 158: **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

XTime_GetTime

Prototype

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

Parameters

XTime_GetTime

Table 159: **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

Arm Cortex-A53 64-bit Processor Specific Include Files

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

Documentation Navigator and Design Hubs

Help → Documentation and Tutorials

Start → All Programs → Xilinx Design Tools → DocNav

`docnav`

Design Hubs View

***Note*:**

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright