Xilinx Standalone Library Documentation

XilFlash Library v4.9

UG651 (v2022.1) April 20, 2022

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.

AMD
XILINX

Table of Contents

Chapter 1: Overview **3**

Chapter 2: Device Geometry **4**

Chapter 3: XilFlash Library API **7**

Functions 8

Chapter 4: Library Parameters in MSS File **15**

Appendix A: Additional Resources and Legal Notices **16**

Xilinx Resources 16

Documentation Navigator and Design Hubs 16

Please Read: Important Legal Notices 17

Overview

The XilFlash library provides read/write/erase/lock/unlock features to access a parallel flash device.

This library implements the functionality for flash memory devices that conform to the "Common Flash Interface" (CFI) standard. CFI allows a single flash library to be used for an entire family of parts and helps us determine the algorithm to utilize during runtime.

Note: All the calls in the library are blocking in nature in that the control is returned back to user only after the current operation is completed successfully or an error is reported.

Library Initialization

The `XFlash_Initialize()` function should be called by the application before any other function in the library. The initialization function checks for the device family and initializes the XFlash instance with the family specific data. The VT table (contains the function pointers to family specific APIs) is setup and family specific initialization routine is called.

Note: The XilFlash library is not supported by Zynq Ultrascale+ MPSoC and Versal platforms.

Device Geometry

The device geometry varies for different flash device families.

Following sections describes the geometry of different flash device families:

Intel Flash Device Geometry

Flash memory space is segmented into areas called blocks. The size of each block is based on a power of 2. A region is defined as a contiguous set of blocks of the same size. Some parts have several regions while others have one. The arrangement of blocks and regions is referred to by this module as the part's geometry. Some Intel flash supports multiple banks on the same device. This library supports single and multiple bank flash devices.

AMD Flash Device Geometry

Flash memory space is segmented into areas called banks and further in to regions and blocks. The size of each block is based on a power of 2. A region is defined as a contiguous set of blocks of the same size. Some parts have several regions while others have one. A bank is defined as a contiguous set of blocks. The bank may contain blocks of different size. The arrangement of blocks, regions and banks is referred to by this module as the part's geometry. The cells within the part can be programmed from a logic 1 to a logic 0 and not the other way around. To change a cell back to a logic 1, the entire block containing that cell must be erased. When a block is erased all bytes contain the value 0xFF. The number of times a block can be erased is finite. Eventually the block will wear out and will no longer be capable of erasure. As of this writing, the typical flash block can be erased 100,000 or more times.

Write Operation

The write call can be used to write a minimum of zero bytes and a maximum entire flash. If the Offset Address specified to write is out of flash or if the number of bytes specified from the Offset address exceed flash boundaries an error is reported back to the user. The write is blocking in nature in that the control is returned back to user only after the write operation is completed successfully or an error is reported.

Read Operation

The read call can be used to read a minimum of zero bytes and maximum of entire flash. If the Offset Address specified to write is out of flash boundary an error is reported back to the user. The read function reads memory locations beyond Flash boundary. Care should be taken by the user to make sure that the Number of Bytes + Offset address is within the Flash address boundaries. The read is blocking in nature in that the control is returned back to user only after the read operation is completed successfully or an error is reported.

Erase Operation

The erase operations are provided to erase a Block in the Flash memory. The erase call is blocking in nature in that the control is returned back to user only after the erase operation is completed successfully or an error is reported.

Sector Protection

The Flash Device is divided into Blocks. Each Block can be protected individually from unwarranted writing/erasing. The Block locking can be achieved using `XFlash_Lock()` lock. All the memory locations from the Offset address specified will be locked. The block can be unlocked using XFlash_UnLock() call. All the Blocks which are previously locked will be unlocked. The Lock and Unlock calls are blocking in nature in that the control is returned back to user only after the operation is completed successfully or an error is reported. The AMD flash device requires high voltage on Reset pin to perform lock and unlock operation. User must provide this high voltage (As defined in datasheet) to reset pin before calling lock and unlock API for AMD flash devices. Lock and Unlock features are not tested for AMD flash device.

Device Control

Functionalities specific to a Flash Device Family are implemented as Device Control. The following are the Intel specific device control:

- Retrieve the last error data.
- Get Device geometry.
- Get Device properties.
- Set RYBY pin mode.
- Set the Configuration register (Platform Flash only).

The following are the AMD specific device control:

- Get Device geometry.
- Get Device properties.
- Erase Resume.

- Erase Suspend.
- Enter Extended Mode.
- Exit Extended Mode.
- Get Protection Status of Block Group.
- Erase Chip.

***Note*:** This library needs to know the type of EMC core (AXI or XPS) used to access the cfi flash, to map the correct APIs. This library should be used with the emc driver, v3_01_a and above, so that this information can be automatically obtained from the emc driver.

This library is intended to be RTOS and processor independent. It works with physical addresses only. Any needs for dynamic memory management, threads, mutual exclusion, virtual memory, cache control, or HW write protection management must be satisfied by the layer above this library. All writes to flash occur in units of bus-width bytes. If more than one part exists on the data bus, then the parts are written in parallel. Reads from flash are performed in any width up to the width of the data bus. It is assumed that the flash bus controller or local bus supports these types of accesses.

XilFlash Library API

This section provides a linked summary and detailed descriptions of the XilFlash library APIs.

Table 1: **Quick Function Reference**

Type	Name	Arguments
int	XFlash_Initialize	XFlash * InstancePtr u32 BaseAddress u8 BusWidth int IsPlatformFlash
int	XFlash_Reset	XFlash * InstancePtr
int	XFlash_DeviceControl	XFlash * InstancePtr u32 Command DeviceCtrlParam * Parameters
int	XFlash_Read	XFlash * InstancePtr u32 Offset u32 Bytes void * DestPtr
int	XFlash_Write	XFlash * InstancePtr u32 Offset u32 Bytes void * SrcPtr
int	XFlash_Erase	XFlash * InstancePtr u32 Offset u32 Bytes
int	XFlash_Lock	XFlash * InstancePtr u32 Offset u32 Bytes
int	XFlash_Unlock	XFlash * InstancePtr u32 Offset u32 Bytes

Table 1: **Quick Function Reference** *(cont'd)*

Type	Name	Arguments
int	XFlash_IsReady	XFlash * InstancePtr

Functions

XFlash_Initialize

This function initializes a specific XFlash instance.

The initialization entails:

- Check the Device family type.
- Issuing the CFI query command.
- Get and translate relevant CFI query information.
- Set default options for the instance.
- Setup the VTable.
- Call the family initialize function of the instance.

Initialize the Xilinx Platform Flash XL to Async mode if the user selects to use the Platform Flash XL in the MLD. The Platform Flash XL is an Intel CFI complaint device.

- XFLASH_PART_NOT_SUPPORTED if the command set algorithm or Layout is not supported by any flash family compiled into the system.
- XFLASH_CFI_QUERY_ERROR if the device would not enter CFI query mode. Either the device(s) do not support CFI, the wrong BaseAddress param was used, an unsupported part layout exists, or a hardware problem exists with the part.

***Note*:** BusWidth is not the width of an individual part. Its the total operating width. For example, if there are two 16-bit parts, with one tied to data lines D0-D15 and other tied to D15-D31, BusWidth would be (32 / 8) = 4. If a single 16-bit flash is in 8-bit mode, then BusWidth should be (8 / 8) = 1.

Prototype

```
int XFlash_Initialize(XFlash *InstancePtr, u32 BaseAddress, u8 BusWidth,
int IsPlatformFlash);
```

Parameters

The following table lists the `XFlash_Initialize` function arguments.

Table 2: **XFlash_Initialize Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.
u32	BaseAddress	Base address of the flash memory.
u8	BusWidth	Total width of the flash memory, in bytes
int	IsPlatformFlash	Used to specify if the flash is a platform flash.

Returns

- XST_SUCCESS if successful.

XFlash_Reset

This function resets the flash device and places it in read mode.

***Note*:** None.

Prototype

```
int XFlash_Reset(XFlash *InstancePtr);
```

Parameters

The following table lists the `XFlash_Reset` function arguments.

Table 3: **XFlash_Reset Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.

Returns

- XST_SUCCESS if successful.
- XFLASH_BUSY if the flash devices were in the middle of an operation and could not be reset.
- XFLASH_ERROR if the device(s) have experienced an internal error during the operation. `XFlash_DeviceControl()` must be used to access the cause of the device specific error. condition.

XFlash_DeviceControl

This function is used to execute device specific commands.

For a list of device specific commands, see the xilflash.h.

***Note*:** None.

Prototype

```
int XFlash_DeviceControl(XFlash *InstancePtr, u32 Command, DeviceCtrlParam
*Parameters);
```

Parameters

The following table lists the `XFlash_DeviceControl` function arguments.

Table 4: **XFlash_DeviceControl Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.
u32	Command	Device specific command to issue.
DeviceCtrlParam *	Parameters	Specifies the arguments passed to the device control function.

Returns

- XST_SUCCESS if successful.
- XFLASH_NOT_SUPPORTED if the command is not recognized/supported by the device(s).

XFlash_Read

This function reads the data from the Flash device and copies it into the specified user buffer.

The source and destination addresses can be on any alignment supported by the processor.

The device is polled until an error or the operation completes successfully.

***Note*:** This function allows the transfer of data past the end of the device's address space. If this occurs, then results are undefined.

Prototype

```
int XFlash_Read(XFlash *InstancePtr, u32 Offset, u32 Bytes, void *DestPtr);
```

Parameters

The following table lists the `XFlash_Read` function arguments.

Table 5: **XFlash_Read Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.
u32	Offset	Offset into the device(s) address space from which to read.

Table 5: **XFlash_Read Arguments** *(cont'd)*

Type	Name	Description
u32	Bytes	Number of bytes to copy.
void *	DestPtr	Destination address to copy data to.

Returns

- XST_SUCCESS if successful.
- XFLASH_ADDRESS_ERROR if the source address does not start within the addressable areas of the device(s).

XFlash_Write

This function programs the flash device(s) with data specified in the user buffer.

The source and destination address must be aligned to the width of the flash's data bus.

The device is polled until an error or the operation completes successfully.

***Note*:** None.

Prototype

```
int XFlash_Write(XFlash *InstancePtr, u32 Offset, u32 Bytes, void *SrcPtr);
```

Parameters

The following table lists the `XFlash_Write` function arguments.

Table 6: **XFlash_Write Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.
u32	Offset	Offset into the device(s) address space from which to begin programming. Must be aligned to the width of the flash's data bus.
u32	Bytes	Number of bytes to program.
void *	SrcPtr	Source address containing data to be programmed. Must be aligned to the width of the flash's data bus. The SrcPtr doesn't have to be aligned to the flash width if the processor supports unaligned access. But, since this library is generic, and some processors(eg. Microblaze) do not support unaligned access; this API requires the SrcPtr to be aligned.

Returns

- XST_SUCCESS if successful.

- XFLASH_ERROR if a write error occurred. This error is usually device specific. Use `XFlash_DeviceControl()` to retrieve specific error conditions. When this error is returned, it is possible that the target address range was only partially programmed.

XFlash_Erase

This function erases the specified address range in the flash device.

The number of bytes to erase can be any number as long as it is within the bounds of the device(s).

The device is polled until an error or the operation completes successfully.

Note: Due to flash memory design, the range actually erased may be larger than what was specified by the Offset & Bytes parameters. This will occur if the parameters do not align to block boundaries.

Prototype

```
int XFlash_Erase(XFlash *InstancePtr, u32 Offset, u32 Bytes);
```

Parameters

The following table lists the `XFlash_Erase` function arguments.

Table 7: **XFlash_Erase Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.
u32	Offset	Offset into the device(s) address space from which to begin erasure.
u32	Bytes	Number of bytes to erase.

Returns

- XST_SUCCESS if successful.
- XFLASH_ADDRESS_ERROR if the destination address range is not completely within the addressable areas of the device(s).

XFlash_Lock

This function Locks the blocks in the specified range of the flash device(s).

The device is polled until an error or the operation completes successfully.

- XFLASH_ADDRESS_ERROR if the destination address range is not completely within the addressable areas of the device(s).

***Note*:** Due to flash memory design, the range actually locked may be larger than what was specified by the Offset & Bytes parameters. This will occur if the parameters do not align to block boundaries.

Prototype

```
int XFlash_Lock(XFlash *InstancePtr, u32 Offset, u32 Bytes);
```

Parameters

The following table lists the `XFlash_Lock` function arguments.

Table 8: **XFlash_Lock Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.
u32	Offset	Offset into the device(s) address space from which to begin block locking. The first three bytes of every block is reserved for special purpose. The offset should be at least three bytes from start of the block.
u32	Bytes	Number of bytes to Lock in the Block starting from Offset.

Returns

- XST_SUCCESS if successful.

XFlash_Unlock

This function Unlocks the blocks in the specified range of the flash device(s).

The device is polled until an error or the operation completes successfully.

***Note*:** None.

Prototype

```
int XFlash_Unlock(XFlash *InstancePtr, u32 Offset, u32 Bytes);
```

Parameters

The following table lists the `XFlash_Unlock` function arguments.

Table 9: **XFlash_Unlock Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.
u32	Offset	Offset into the device(s) address space from which to begin block UnLocking. The first three bytes of every block is reserved for special purpose. The offset should be at least three bytes from start of the block.

Table 9: **XFlash_Unlock Arguments** *(cont'd)*

Type	Name	Description
u32	Bytes	Number of bytes to UnLock in the Block starting from Offset.

Returns

- XST_SUCCESS if successful.
- XFLASH_ADDRESS_ERROR if the destination address range is not completely within the addressable areas of the device(s).

XFlash_IsReady

This function checks the readiness of the device, which means it has been successfully initialized.

***Note*:** None.

Prototype

```
int XFlash_IsReady(XFlash *InstancePtr);
```

Parameters

The following table lists the `XFlash_IsReady` function arguments.

Table 10: **XFlash_IsReady Arguments**

Type	Name	Description
XFlash *	InstancePtr	Pointer to the XFlash instance.

Returns

TRUE if the device has been initialized (but not necessarily started), and FALSE otherwise.

Library Parameters in MSS File

XilFlash Library can be integrated with a system using the following snippet in the Microprocessor Software Specification (MSS) file:

```
BEGIN LIBRARY
PARAMETER LIBRARY_NAME = xilflash
PARAMETER LIBRARY_VER = 4.9
PARAMETER PROC_INSTANCE = microblaze_0
PARAMETER ENABLE_INTEL = true
PARAMETER ENABLE_AMD = false
END
```

The table below describes the libgen customization parameters.

Parameter	Default Value	Description
LIBRARY_NAME	xilflash	Specifies the library name.
LIBRARY_VER	4.9	Specifies the library version.
PROC_INSTANCE	microblaze_0	Specifies the processor name.
ENABLE_INTEL	true/false	Enables or disables the Intel flash device family.
ENABLE_AMD	true/false	Enables or disables the AMD flash device family.

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