Xilinx Standalone Library Documentation

Xillsf Library v5.15

UG652 (v2020.1) June 3, 2020

XILINX®

Table of Contents

Overview

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 Xillsf library APIs.

Table 1: **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

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

Type	Name	Arguments
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
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 2: **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 3: **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 4: **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 5: **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 6: **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 7: **XIsf_Write Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
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 8: **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

Table 8: **XIsf_Read Arguments** *(cont'd)*

Type	Name	Description
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 9: **XIsf_Erase Arguments**

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

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

Type	Name	Description
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 10: **XIsf_SectorProtect Arguments**

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

Table 10: **XIsf_SectorProtect Arguments** *(cont'd)*

Type	Name	Description
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 11: **XIsf_Ioctl Arguments**

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

Table 11: **XIsf_Ioctl Arguments** *(cont'd)*

Type	Name	Description
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 12: **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 13: **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 14: **XIsf_SetSpiConfiguration Arguments**

Type	Name	Description
XIsf *	InstancePtr	Pointer to the XIsf instance.
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 15: **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 16: **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
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.

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