Xilinx Standalone Library Documentation

XilFPGA Library v5.2

UG1229 (v2020.1) June 3, 2020

XILINX®

Table of Contents

Chapter 1

Overview

The XilFPGA library provides an interface to the Linux or bare-metal users for configuring the programmable logic (PL) over PCAP from PS. The library is designed for Zynq UltraScale+ MPSoC to run on top of Xilinx standalone BSPs. It is tested for A53, R5 and MicroBlaze. In the most common use case, we expect users to run this library on the PMU MicroBlaze with PMUFW to serve requests from either Linux or Uboot for Bitstream programming.

***Note*:** XILFPGA does not support a DDR less system. DDR must be present for use of XilFPGA.

Supported Features

The following features are supported in Zynq UltraScale+ MPSoC platform.

- Full bitstream loading
- Partial bitstream loading
- Encrypted bitstream loading
- Authenticated bitstream loading
- Authenticated and encrypted bitstream loading
- Readback of configuration registers
- Readback of configuration data

XilFPGA library Interface modules

XilFPGA library uses the below major components to configure the PL through PS.

Processor Configuration Access Port (PCAP)

The processor configuration access port (PCAP) is used to configure the programmable logic (PL) through the PS.

CSU DMA driver

The CSU DMA driver is used to transfer the actual bitstream file for the PS to PL after PCAP initialization.

XilSecure Library

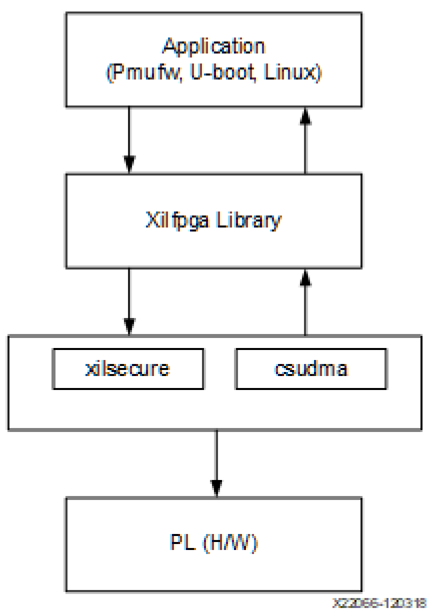
The XilSecure library provides APIs to access secure hardware on the Zynq UltraScale+ MPSoC devices.

***Note*:** The current version of library supports only Zynq UltraScale MPSoC devices.

Design Summary

XilFPGA library acts as a bridge between the user application and the PL device. It provides the required functionality to the user application for configuring the PL Device with the required bitstream. The following figure illustrates an implementation where the XilFPGA library needs the CSU DMA driver APIs to transfer the bitstream from the DDR to the PL region. The XilFPGA library also needs the XilSecure library APIs to support programming authenticated and encrypted bitstream files.

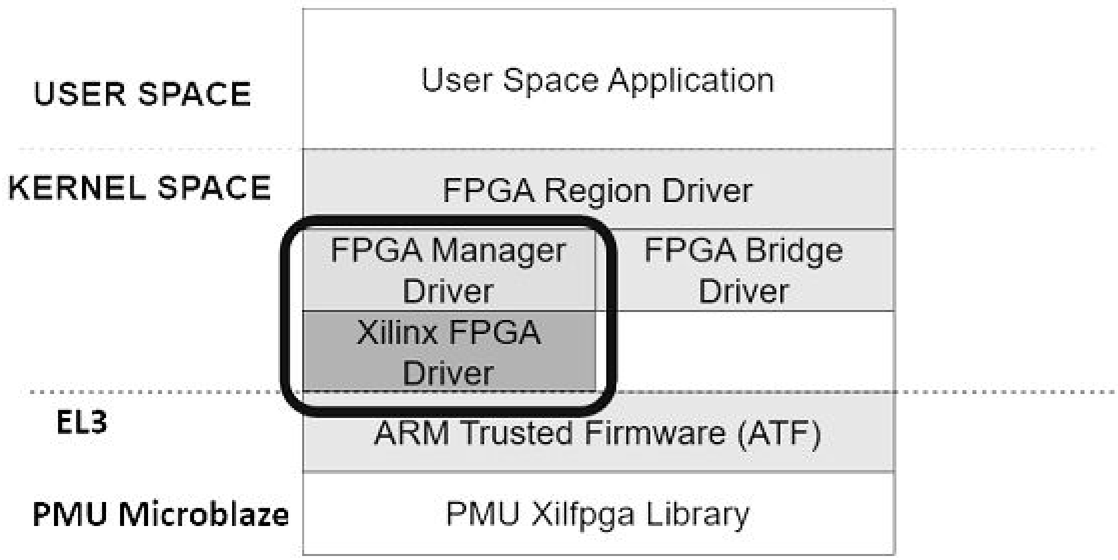
Figure 1: **XilFPGA Design Summary**



Flow Diagram

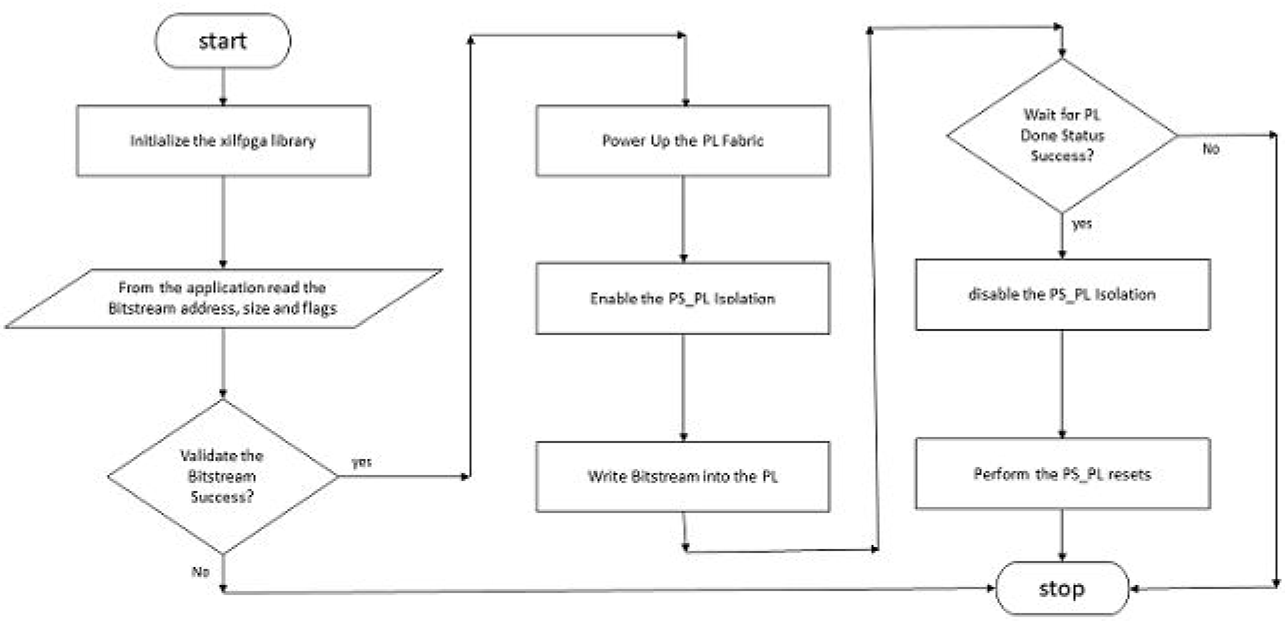
The following figure illustrates the Bitstream loading flow on the Linux operating system.

Figure 2: **Bitstream loading on Linux:**



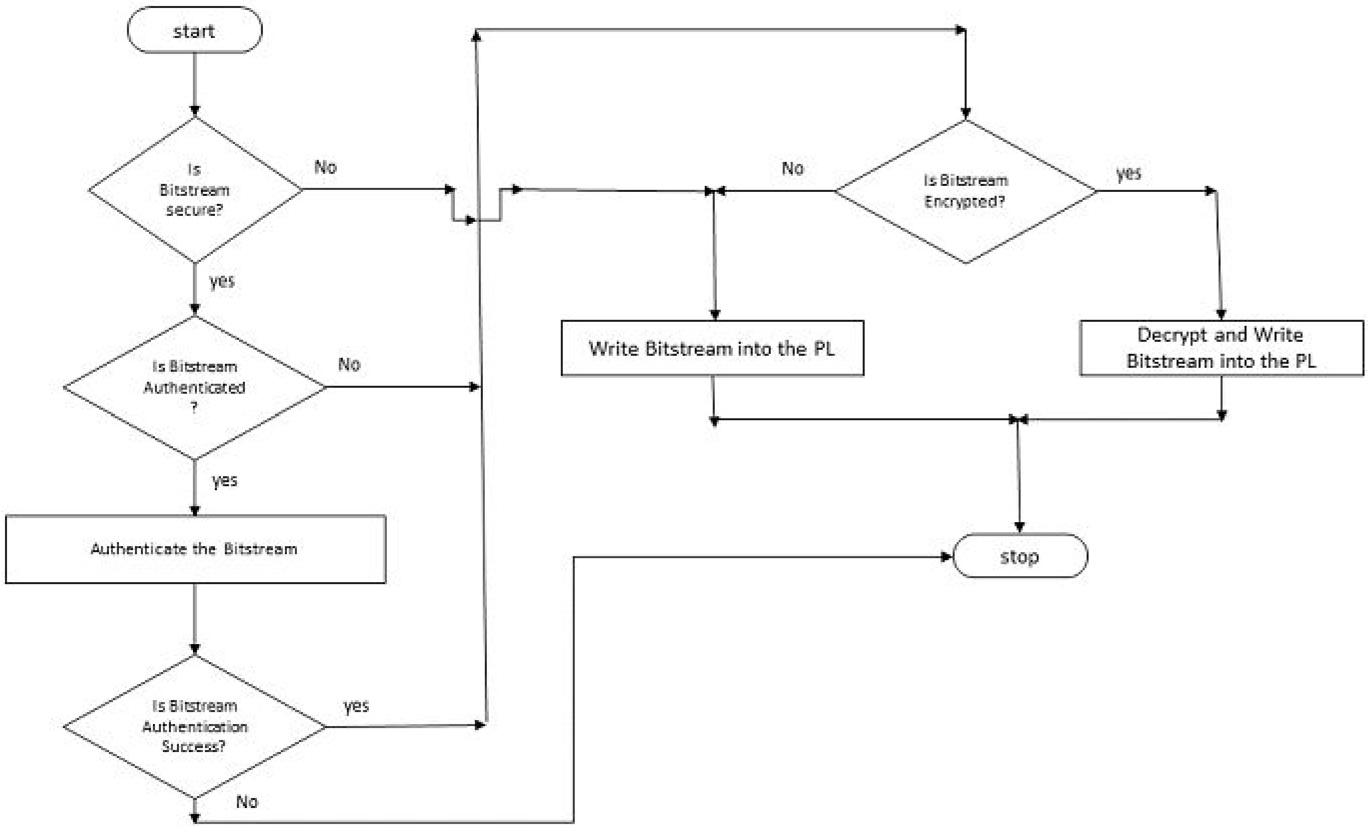
The following figure illustrates the XilFPGA PL configuration sequence.

Figure 3: **XilFPGA PL Configuration Sequence**



The following figure illustrates the Bitstream write sequence.

Figure 4: **Bitstream write Sequence**



XilFPGA BSP Configuration Settings

The XilFPGA library provides user configuration BSP settings. The following table describes the parameters and their default value:

Parameter Name	Type	Default Value	Description
secure_mode	bool	TRUE	Enables secure Bitstream loading support.
debug_mode	bool	FALSE	Enables the Debug messages in the library.
ocm_address	int	0xfffc0000	Address used for the Bitstream authentication.
base_address	int	0x80000	Holds the Bitstream Image address. This flag is valid only for the Cortex-A53 or the Cortex-R5 processors.

Parameter Name	Type	Default Value	Description
secure_readback	bool	FALSE	Should be set to TRUE to allow the secure Bitstream configuration data read back. The application environment should be secure and trusted to enable this flag.
secure_environment	bool	FALSE	Enable the secure PL configuration using the IPI. This flag is valid only for the Cortex-A53 or the Cortex-R5 processors.

Setting up the Software System

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

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

13. Using the Board Support Package Settings page, you can select the OS Version and which of the Supported Libraries are to be enabled in this domain/BSP.
14. Select the **xilfpga** library from the list of **Supported Libraries**.
15. Expand the **Overview** tree and select **xilfpga**. The configuration options for xilfpga are listed.
16. Configure the xilfpga by providing the base address of the Bit-stream file (DDR address) and the size (in bytes).
17. Click **OK**. The board support package automatically builds with XilFPGA library included in it.
18. Double-click the **system.mss** file to open it in the **Editor** view.
19. Scroll-down and locate the **Libraries** section.
20. Click **Import Examples** adjacent to the XilFPGA entry.

Enabling Security

To support encrypted and/or authenticated bitstream loading, you must enable security in PMUFW.

1. Click **File** > **New** > **Platform Project**.
2. Click **Specify** to create a new Hardware Platform Specification.
3. Provide a new name for the domain in the **Project name** field if you wish to override the default value.
4. Select the location for the board support project files. To use the default location, as displayed in the **Location** field, leave the **Use default location** check box selected. Otherwise, deselect the checkbox and then type or browse to the directory location.
5. From the **Hardware Platform** drop-down choose the appropriate platform for your application or click the **New** button to browse to an existing Hardware Platform.
6. Select the target CPU from the drop-down list.
7. From the **Board Support Package OS** list box, select the type of board support package to create. A description of the platform types displays in the box below the drop-down list.
8. Click **Finish**. The wizard creates a new software platform and displays it in the Vitis Navigator pane.
9. Select **Project** > **Build Automatically** to automatically build the board support package. The Board Support Package Settings dialog box opens. Here you can customize the settings for the domain.
10. Click **OK** to accept the settings, build the platform, and close the dialog box.
11. From the Explorer, double-click platform.spr file and select the appropriate domain/board support package. The overview page opens.

12. In the overview page, click **Modify BSP Settings**.
13. Using the Board Support Package Settings page, you can select the OS Version and which of the Supported Libraries are to be enabled in this domain/BSP.
14. Expand the **Overview** tree and select **Standalone**.
15. Select a supported hardware platform.
16. Select **psu_pmu_0** from the **Processor** drop-down list.
17. Click Next. The **Templates** page appears.
18. Select **ZynqMP PMU Firmware** from the **Available Templates** list.
19. Click **Finish**. A PMUFW application project is created with the required BSPs.
20. Double-click the **system.mss** file to open it in the **Editor** view.
21. Click the **Modify this BSP's Settings** button. The **Board Support Package Settings** dialog box appears.
22. Select **xilfpga**. Various settings related to the library appears.
23. Select **secure_mode** and modify its value to **true** .
24. Click **OK** to save the configuration.

Note: By default the secure mode is enabled. To disable modify the secure_mode value to false.

Bitstream Authentication Using External Memory

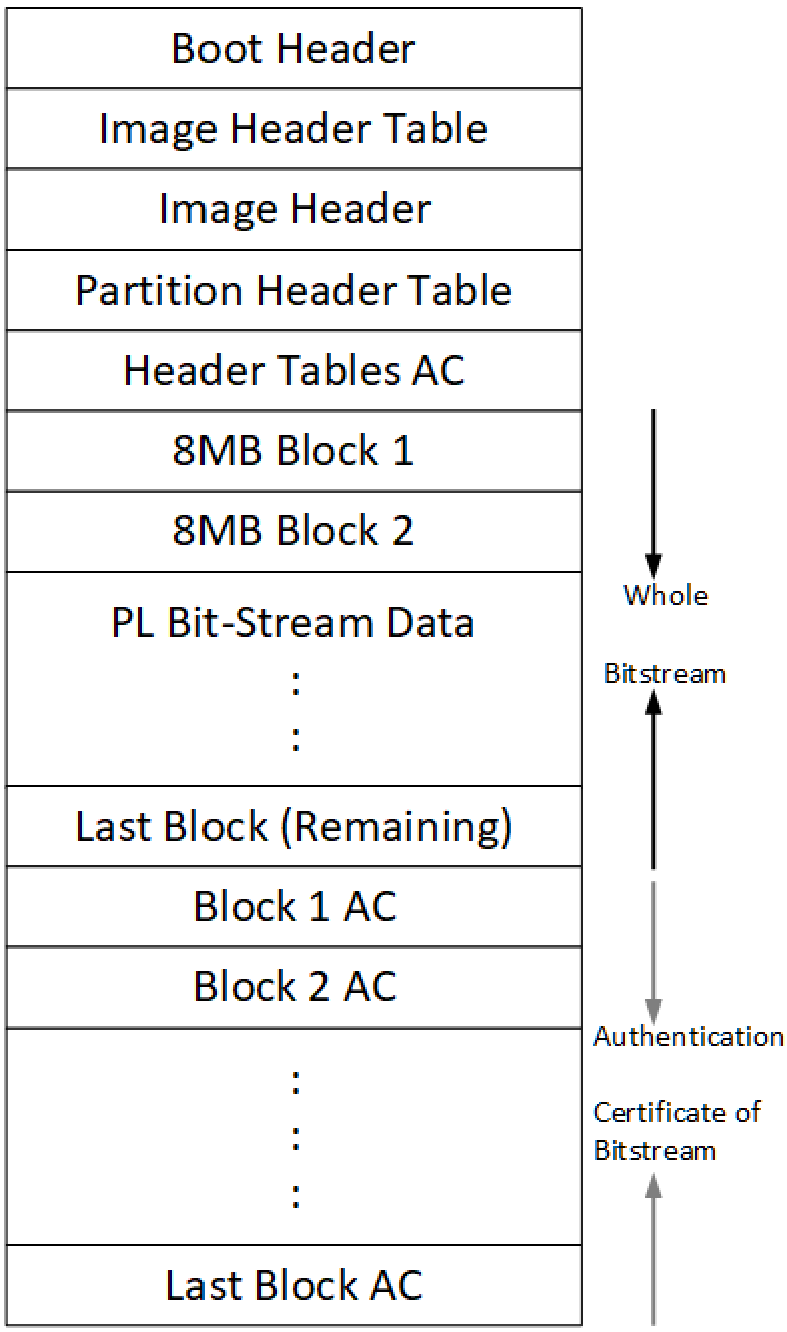
The size of the Bitstream is too large to be contained inside the device, therefore external memory must be used. The use of external memory could create a security risk. Therefore, two methods are provided to authenticate and decrypt a Bitstream.

- The first method uses the internal OCM as temporary buffer for all cryptographic operations. For details, see `Authenticated and Encrypted Bitstream Loading Using OCM`. This method does not require trust in external DDR.
- The second method uses external DDR for authentication prior to sending the data to the decryptor, there by requiring trust in the external DDR. For details, see `Authenticated and Encrypted Bitstream Loading Using DDR`.

Bootgen

When a Bitstream is requested for authentication, Bootgen divides the Bitstream into blocks of 8MB each and assigns an authentication certificate for each block. If the size of a Bitstream is not in multiples of 8 MB, the last block contains the remaining Bitstream data.

Figure 5: **Bitstream Blocks**



When both authentication and encryption are enabled, encryption is first done on the Bitstream. Bootgen then divides the encrypted data into blocks and assigns an Authentication certificate for each block.

Authenticated and Encrypted Bitstream Loading Using OCM

To authenticate the Bitstream partition securely, XilFPGA uses the FSBL section's OCM memory to copy the bitstream in chunks from DDR. This method does not require trust in the external DDR to securely authenticate and decrypt a Bitstream.

The software workflow for authenticating Bitstream is as follows:

1. XilFPGA identifies DDR secure Bitstream image base address. XilFPGA has two buffers in OCM, the Read Buffer is of size 56KB and hash of chunks to store intermediate hashes calculated for each 56 KB of every 8MB block.
2. XilFPGA copies a 56KB chunk from the first 8MB block to Read Buffer.
3. XilFPGA calculates hash on 56 KB and stores in HashsOfChunks.
4. XilFPGA repeats steps 1 to 3 until the entire 8MB of block is completed.

   ***Note*:** The chunk that XilFPGA copies can be of any size. A 56KB chunk is taken for better performance.

5. XilFPGA authenticates the 8MB Bitstream chunk.
6. Once the authentication is successful, XilFPGA starts copying information in batches of 56KB starting from the first block which is located in DDR to Read Buffer, calculates the hash, and then compares it with the hash stored at HashsOfChunks.
7. If the hash comparison is successful, FSBL transmits data to PCAP using DMA (for un-encrypted Bitstream) or AES (if encryption is enabled).
8. XilFPGA repeats steps 6 and 7 until the entire 8MB block is completed.
9. Repeats steps 1 through 8 for all the blocks of Bitstream.

***Note*:** You can perform warm restart even when the FSBL OCM memory is used to authenticate the Bitstream. PMU stores the FSBL image in the PMU reserved DDR memory which is visible and accessible only to the PMU and restores back to the OCM when APU-only restart needs to be performed. PMU uses the SHA3 hash to validate the FSBL image integrity before restoring the image to OCM (PMU takes care of only image integrity and not confidentiality).

Authenticated and Encrypted Bitstream Loading Using DDR

The software workflow for authenticating Bitstream is as follows:

1. XilFPGA identifies DDR secure Bitstream image base address.
2. XilFPGA calculates hash for the first 8MB block.
3. XilFPGA authenticates the 8MB block while stored in the external DDR.
4. If Authentication is successful, XilFPGA transmits data to PCAP via DMA (for unencrypted Bitstream) or AES (if encryption is enabled).
5. Repeats steps 1 through 4 for all the blocks of Bitstream.

Chapter 2

XilFPGA APIs

This section provides detailed descriptions of the XilFPGA library APIs.

Table 1: **Quick Function Reference**

Type	Name	Arguments
u32	XFpga_Initialize	void
u32	XFpga_PL_BitStream_Load	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags
u32	XFpga_PL_Preconfig	void
u32	XFpga_PL_Write	void
u32	XFpga_PL_PostConfig	XFpga * InstancePtr
u32	XFpga_PL_ValidateImage	XFpga * InstancePtr UINTPTR BitstreamImageAddr UINTPTR AddrPtr_Size u32 Flags
u32	XFpga_GetPlConfigData	XFpga * InstancePtr
u32	XFpga_GetPlConfigReg	XFpga * InstancePtr ConfigReg Address
u32	XFpga_InterfaceStatus	XFpga * InstancePtr

Functions

XFpga_Initialize

Prototype

```
u32 XFpga_Initialize(XFpga *InstancePtr);
```

XFpga_PL_BitStream_Load

The API is used to load the bitstream file into the PL region.

It supports vivado generated Bitstream(*.bit, *.bin) and bootgen generated Bitstream(*.bin) loading, Passing valid Bitstream size (AddrPtr_Size) info is mandatory for vivado * generated Bitstream, For bootgen generated Bitstreams it will take Bitstream size from the Bitstream Header.

Prototype

```
u32 XFpga_PL_BitStream_Load(XFpga *InstancePtr, UINTPTR BitstreamImageAddr,
UINTPTR AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_BitStream_Load` function arguments.

Table 2: **XFpga_PL_BitStream_Load Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure.
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Aes key address which is used for Decryption (or) In none Secure Bitstream used it is used to store size of Bitstream Image.

Table 2: **XFpga_PL_BitStream_Load Arguments** *(cont'd)*

Type	Name	Description
u32	Flags	Flags are used to specify the type of Bitstream file. • BIT(0) - Bitstream type ◦ 0 - Full Bitstream ◦ 1 - Partial Bitstream • BIT(1) - Authentication using DDR ◦ 1 - Enable ◦ 0 - Disable • BIT(2) - Authentication using OCM ◦ 1 - Enable ◦ 0 - Disable • BIT(3) - User-key Encryption ◦ 1 - Enable ◦ 0 - Disable • BIT(4) - Device-key Encryption ◦ 1 - Enable ◦ 0 - Disable

Returns

- XFPGA_SUCCESS on success
- Error code on failure.
- XFPGA_VALIDATE_ERROR.
- XFPGA_PRE_CONFIG_ERROR.
- XFPGA_WRITE_BITSTREAM_ERROR.
- XFPGA_POST_CONFIG_ERROR.

XFpga_PL_Preconfig

Prototype

```
u32 XFpga_PL_Preconfig(XFpga *InstancePtr);
```

XFpga_PL_Write

Prototype

```
u32 XFpga_PL_Write(XFpga *InstancePtr, UINTPTR BitstreamImageAddr, UINTPTR
AddrPtr_Size, u32 Flags);
```

XFpga_PL_PostConfig

This function set FPGA to operating state after writing.

Prototype

```
u32 XFpga_PL_PostConfig(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_PL_PostConfig` function arguments.

Table 3: **XFpga_PL_PostConfig Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure

Returns

Codes as mentioned in xilfpga.h

XFpga_PL_ValidateImage

This function is used to validate the Bitstream Image.

Prototype

```
u32 XFpga_PL_ValidateImage(XFpga *InstancePtr, UINTPTR BitstreamImageAddr,
UINTPTR AddrPtr_Size, u32 Flags);
```

Parameters

The following table lists the `XFpga_PL_ValidateImage` function arguments.

Table 4: **XFpga_PL_ValidateImage Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure
UINTPTR	BitstreamImageAddr	Linear memory Bitstream image base address
UINTPTR	AddrPtr_Size	Aes key address which is used for Decryption (or) In none Secure Bitstream used it is used to store size of Bitstream Image.

Table 4: **XFpga_PL_ValidateImage Arguments** *(cont'd)*

Type	Name	Description

u32	Flags	Flags are used to specify the type of Bitstream file.
		• BIT(0) - Bitstream type
		  ◦ 0 - Full Bitstream
		  ◦ 1 - Partial Bitstream
		• BIT(1) - Authentication using DDR
		  ◦ 1 - Enable
		  ◦ 0 - Disable
		• BIT(2) - Authentication using OCM
		  ◦ 1 - Enable
		  ◦ 0 - Disable
		• BIT(3) - User-key Encryption
		  ◦ 1 - Enable
		  ◦ 0 - Disable
		• BIT(4) - Device-key Encryption
		  ◦ 1 - Enable
		  ◦ 0 - Disable

Returns

Codes as mentioned in xilfpga.h

XFpga_GetPlConfigData

Provides functionality to read back the PL configuration data.

Prototype

```
u32 XFpga_GetPlConfigData(XFpga *InstancePtr, UINTPTR ReadbackAddr, u32
NumFrames);
```

Parameters

The following table lists the `XFpga_GetPlConfigData` function arguments.

Table 5: **XFpga_GetPlConfigData Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure

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

Type	Name	Description
UINTPTR	ReadbackAddr	Address which is used to store the PL readback data.
u32	NumFrames	The number of Fpga configuration frames to read.

Returns

- XFPGA_SUCCESS if successful
- XFPGA_FAILURE if unsuccessful
- XFPGA_OPS_NOT_IMPLEMENTED if implementation not exists.

XFpga_GetPlConfigReg

Provides PL specific configuration register values.

Prototype

```
u32 XFpga_GetPlConfigReg(XFpga *InstancePtr, UINTPTR ReadbackAddr, u32
ConfigRegAddr);
```

Parameters

The following table lists the `XFpga_GetPlConfigReg` function arguments.

Table 6: **XFpga_GetPlConfigReg Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure
UINTPTR	ReadbackAddr	Address which is used to store the PL Configuration register data.
u32	ConfigRegAddr	Configuration register address. For more information, see,UG570 - UltraScale Architecture Configuration User Guide.

Returns

- XFPGA_SUCCESS if successful
- XFPGA_FAILURE if unsuccessful
- XFPGA_OPS_NOT_IMPLEMENTED if implementation not exists.

XFpga_InterfaceStatus

This function provides the STATUS of PL programming interface.

Prototype

```
u32 XFpga_InterfaceStatus(XFpga *InstancePtr);
```

Parameters

The following table lists the `XFpga_InterfaceStatus` function arguments.

Table 7: **XFpga_InterfaceStatus Arguments**

Type	Name	Description
XFpga *	InstancePtr	Pointer to the XFgpa structure

Returns

Status of the PL programming interface.

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

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright