Xilinx Standalone Library Documentation

XilFPGA Library v5.1

UG1229 (2019.2) October 30, 2019

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.

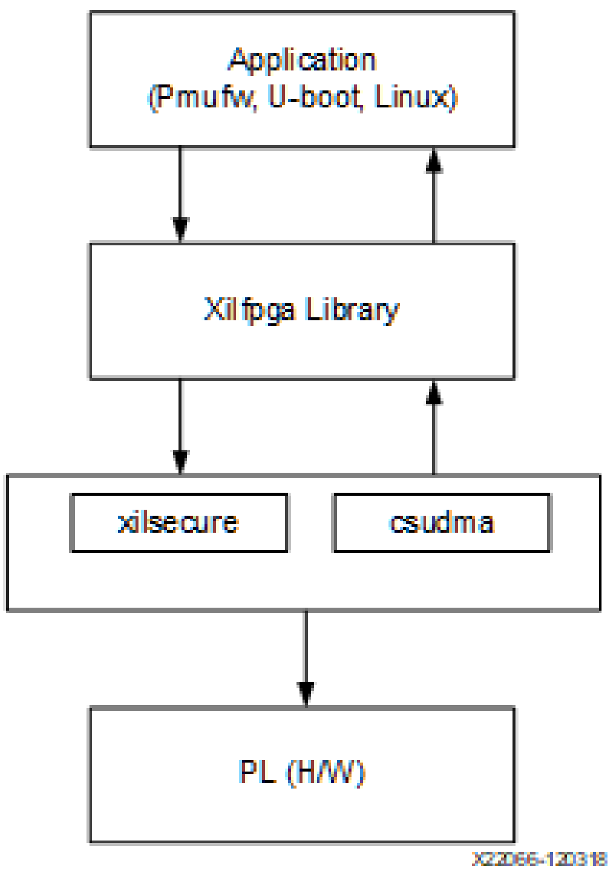

Figure 1.1: XilFPGA Design Summary

Flow Diagram

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

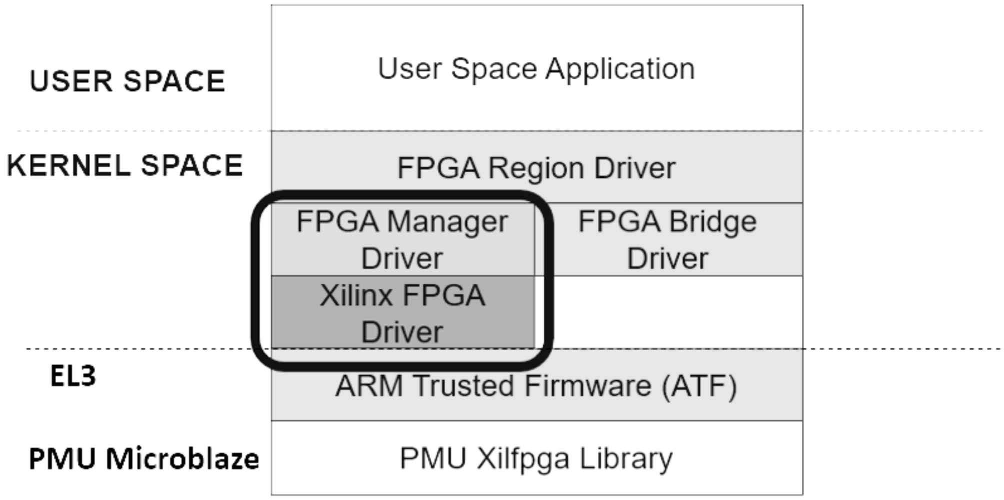

Figure 1.2: Bitstream loading on Linux:

The following figure illustrates the XilFPGA PL configuration sequence.

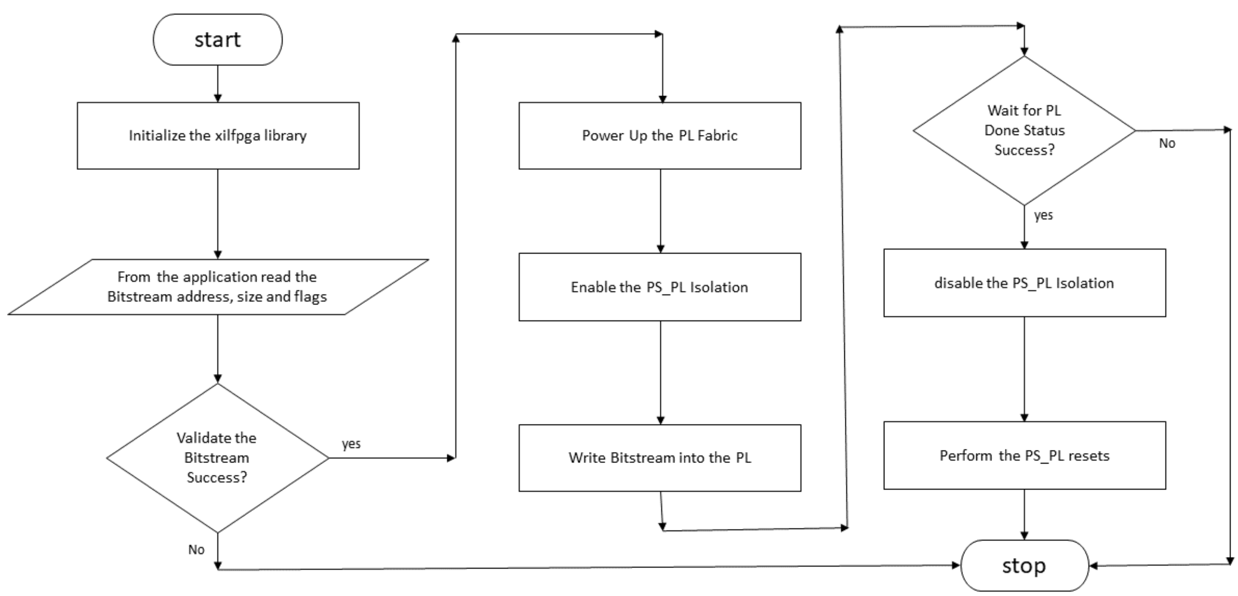

Figure 1.3: XilFPGA PL Configuration Sequence

The following figure illustrates the Bitstream write sequence.

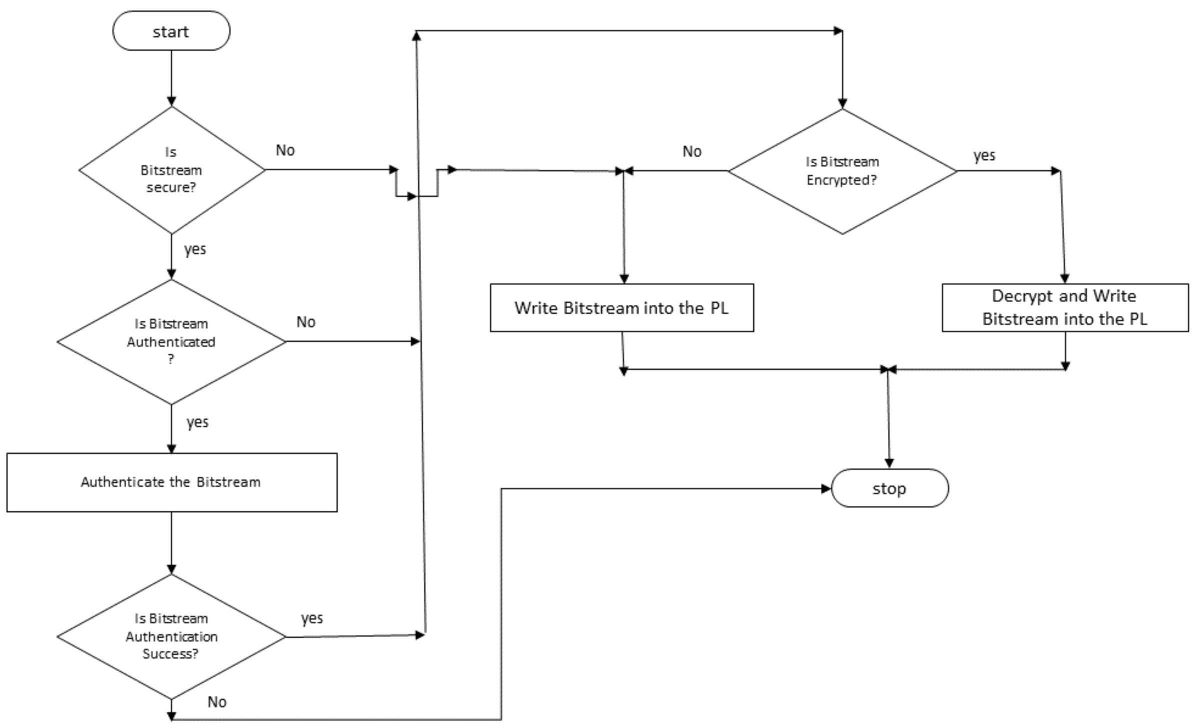

Figure 1.4: Bitstream write Sequence

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. Launch Xilinx SDK. Xilinx SDK prompts you to create a workspace.
2. Select **File** > **New** > **Xilinx Board Support Package**. The **New Board Support Package** wizard appears.
3. Specify a project name.
4. Select **Standalone** from the **Board Support Package OS** drop-down list. The **Board Support Package Settings** wizard appears.
5. Select the **xilfpga** library from the list of **Supported Libraries**.
6. Expand the **Overview** tree and select **xilfpga**. The configuration options for xilfpga are listed.
7. Configure the xilfpga by providing the base address of the Bit-stream file (DDR address) and the size (in bytes).

8. Click **OK**. The board support package automatically builds with XilFPGA library included in it.
9. Double-click the **system.mss** file to open it in the **Editor** view.
10. Scroll-down and locate the **Libraries** chapter.
11. Click **Import Examples** adjacent to the XilFPGA 5.1 entry.

Enabling Security

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

1. Launch Xilinx SDK. Xilinx SDK prompts you to create a workspace.
2. Select **File** > **New** > **Application Project**. The **New Application Project** wizard appears.
3. Specify a project name.
4. Select **Standalone** from the **OS Platform** drop-down list.
5. Select a supported hardware platform.
6. Select **psu_pmu_0** from the **Processor** drop-down list.
7. Click Next. The **Templates** page appears.
8. Select **ZynqMP PMU Firmware** from the **Available Templates** list.
9. Click **Finish**. A PMUFW application project is created with the required BSPs.
10. Double-click the **system.mss** file to open it in the **Editor** view.
11. Click the **Modify this BSP's Settings** button. The **Board Support Package Settings** dialog box appears.
12. Select **xilfpga**. Various settings related to the library appears.
13. Select **secure_mode** and modify its value to **true** .
14. 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.

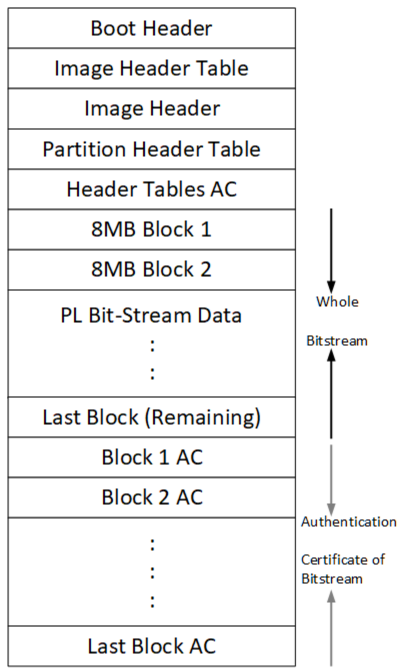

Figure 1.5: Bitstream Blocks

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 chapter'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

Overview

This chapter provides detailed descriptions of the XilFPGA library APIs.

Functions

- u32 XFpga_PL_BitStream_Load (XFpga *InstancePtr, UINTPTR BitstreamImageAddr, UINTPTR AddrPtr_Size, u32 Flags)
- u32 XFpga_PL_PostConfig (XFpga *InstancePtr)
- u32 XFpga_PL_ValidateImage (XFpga *InstancePtr, UINTPTR BitstreamImageAddr, UINTPTR AddrPtr_Size, u32 Flags)
- u32 XFpga_GetPlConfigData (XFpga *InstancePtr, UINTPTR ReadbackAddr, u32 ConfigReg_NumFrames)
- u32 XFpga_GetPlConfigReg (XFpga *InstancePtr, UINTPTR ReadbackAddr, u32 ConfigReg_NumFrames)
- u32 XFpga_InterfaceStatus (XFpga *InstancePtr)

Function Documentation

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

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.

Parameters

InstancePtr	Pointer to the XFgpa structure.
BitstreamImageAddr	Linear memory Bitstream image base address
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.
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.

u32 XFpga_PL_PostConfig (XFpga * *InstancePtr*)

This function set FPGA to operating state after writing.

Parameters

InstancePtr	Pointer to the XFgpa structure

Returns

Codes as mentioned in xilfpga.h

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

This function is used to validate the Bitstream Image.

Parameters

InstancePtr	Pointer to the XFgpa structure
BitstreamImageAddr	Linear memory Bitstream image base address
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.

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

u32 XFpga_GetPlConfigData (XFpga ∗ *InstancePtr,* UINTPTR *ReadbackAddr,* u32 *ConfigReg_NumFrames*)

This function provides functionality to read back the PL configuration data.

Parameters

InstancePtr	Pointer to the XFgpa structure

Address which is used to store the PL readback data.
Configuration register value to be returned (or) 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.

u32 XFpga_GetPlConfigReg (XFpga ∗ *InstancePtr,* UINTPTR *ReadbackAddr,* u32 *ConfigReg_NumFrames*)

This function provides PL specific configuration register values.

Parameters

InstancePtr	Pointer to the XFgpa structure
ConfigReg	Constant which represents the configuration register value to be returned.
Address	DMA linear buffer address.

Returns

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

u32 XFpga_InterfaceStatus (XFpga ∗ *InstancePtr*)

This function provides the STATUS of PL programming interface.

Parameters

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 .

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Please Read: Important Legal Notices