

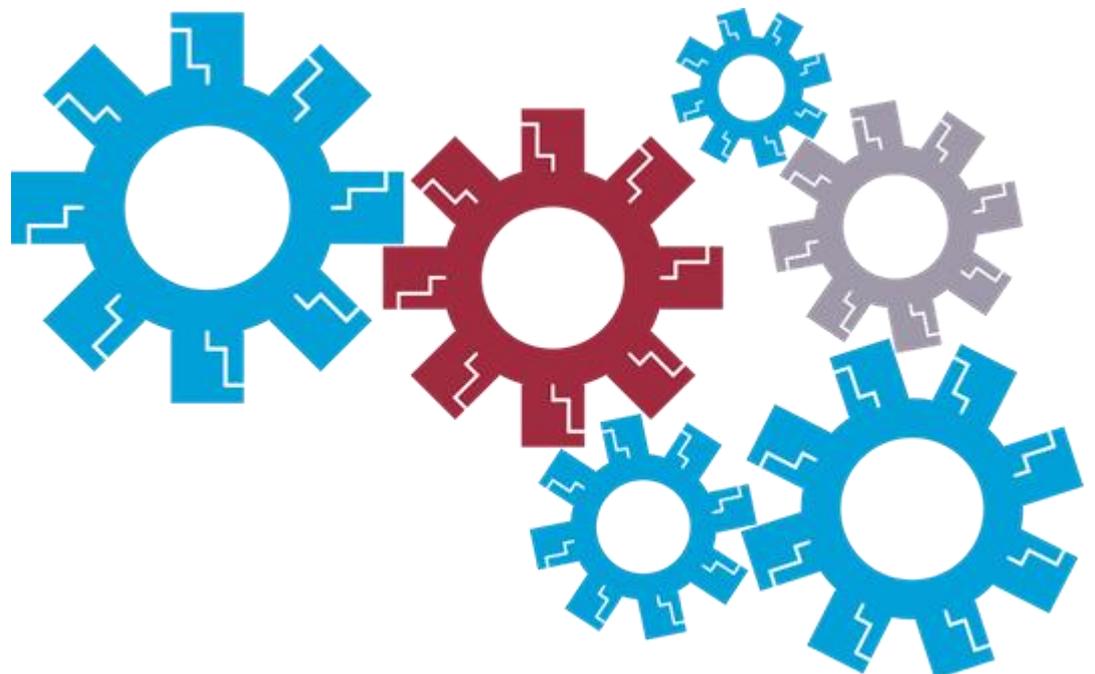
# *logiREF-DFX-IDF*

*logiREF-DFX-IDF Dynamic Function eXchange  
Design Framework with Isolation Design Flow*

## **User's Manual**

**Version: 1.0.2**

logiREF-DFX-IDF\_v1.0.2





All rights reserved. This manual may not be reproduced or utilized without the prior written permission issued by Xylon.

Copyright © Xylon d.o.o. logicBRICKS® is a registered Xylon trademark.

All other trademarks and registered trademarks are the property of their respective owners.

This publication has been carefully checked for accuracy. However, Xylon does not assume any responsibility for the contents or use of any product described herein. Xylon reserves the right to make any changes to product without further notice. Our customers should ensure to take appropriate action so that their use of our products does not infringe upon any patents.

<b>1</b>	<b>ABOUT THE FRAMEWORK</b>	<b>5</b>
1.1	PROGRAMMABLE LOGIC UTILIZATION	6
1.2	HARDWARE REQUIREMENTS	8
1.2.1	GMSL2 Deserializer FMC Module	8
1.2.2	Xylon GMSL2 Camera	9
1.3	SOFTWARE REQUIREMENTS	10
1.3.1	About Xilinx Vitis™ Development Environment	10
1.4	DESIGN DELIVERABLES	11
1.4.1	Hardware Design Files	11
1.4.2	Software	11
1.4.3	Binaries	11
1.5	REFERENCE DESIGN	12
<b>2</b>	<b>LOGICBRICKS IP CORES</b>	<b>13</b>
2.1	ABOUT LOGICBRICKS IP LIBRARY	13
2.2	EVALUATION LOGICBRICKS IP CORES	14
2.3	LOGICBRICKS IP CORES USED IN THIS DESIGN	15
2.3.1	logiCVC-ML Compact Multilayer Video Controller	15
2.3.2	logiWIN Versatile Video Input	16
2.4	LOGICBRICKS IP CORES FOR VIDEO PROCESSING	17
<b>3</b>	<b>GET AND INSTALL THE DESIGN FRAMEWORK</b>	<b>18</b>
3.1	INSTALLATION PROCESS	18
3.1.1	Filesystem Permissions of the Installed Folder (Microsoft® Windows® OS)	19
	FOLDER STRUCTURE	20
<b>4</b>	<b>GETTING THE IP LICENSES</b>	<b>22</b>
4.1	GETTING LOGICBRICKS IP LICENCES	22
4.2	GETTING THE XILINX HDMI 1.4/2.0 TRANSMITTER SUBSYSTEM LICENSE	23
<b>5</b>	<b>LOGIREF-DFX-IDF REFERENCE DESIGN</b>	<b>24</b>
5.1	LOGIREF-DFX-IDF SoC DESIGN AND MEMORY LAYOUT	24
5.2	VIDEO INPUT/OUTPUT SYNCHRONIZATION	27
5.2.1	logiWIN Hardware Buffering Implementation	27
5.2.2	logiCVC-ML Hardware Buffering Implementation	27
5.3	XYLON LOGICBRICKS IP CORE CONFIGURATION	29
5.4	SIHA IP MANAGER	30
5.5	SEM IP CONTROLLER	31
5.6	RESTORING FULL MPSOC DESIGN FROM XYLON DELIVERABLES	31
5.7	SOFTWARE DESCRIPTION	41
5.7.1	Demo application	41
5.7.2	DFX functionality description	44
5.7.3	Error injection using SEM	47
5.7.4	Configuration file description	47
5.7.5	Input resolution and the frame rate	48
5.7.6	Output resolution and the frame rate	48
<b>6</b>	<b>QUICK START</b>	<b>49</b>
6.1	RUN THE PRECOMPILED LINUX DEMO EXAMPLES	50
6.2	DEMO CONTROLS	50
6.3	CHANGE THE DELIVERED SOFTWARE	50
6.3.1	Xilinx Development Software	50

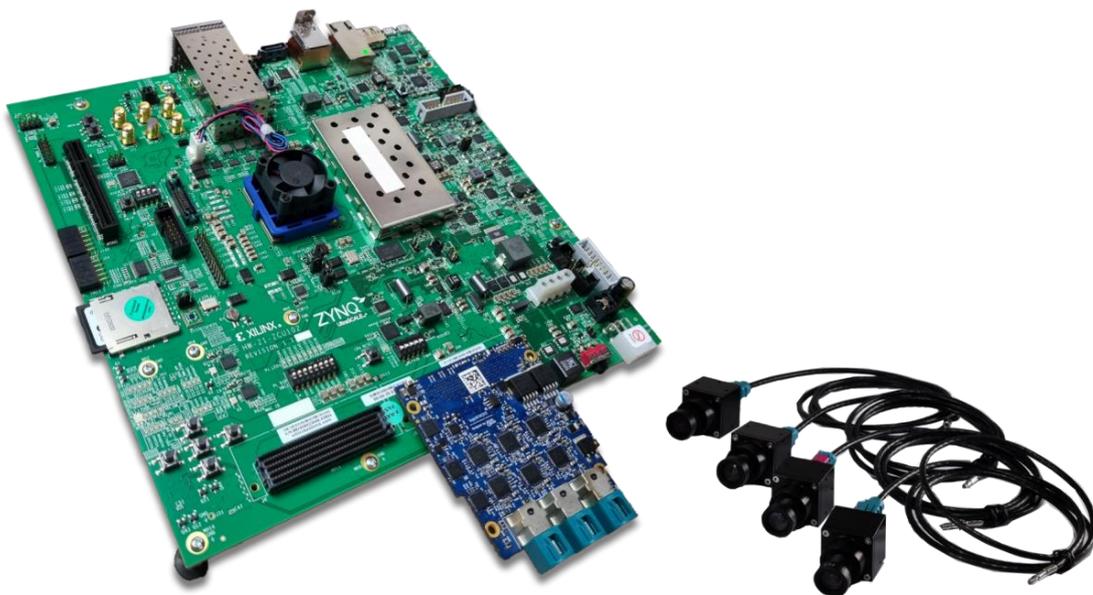
6.3.2	Set Up Linux System Software Development Tools .....	51
6.3.3	Set Up git Tools .....	51
6.3.4	Setting up and building the Petalinux project .....	51
6.3.5	Setting up the Vitis workspace .....	58
6.4	DEBUGGING SOFTWARE APPLICATION WITH THE TCF AGENT .....	63
6.5	SEM UART EXAMPLE COMMANDS .....	65
<b>7</b>	<b>REFERENCES</b> .....	<b>69</b>
<b>8</b>	<b>REVISION HISTORY</b> .....	<b>70</b>

## 1 ABOUT THE FRAMEWORK

The logiREF-DFX-IDF Dynamic Function eXchange Design Framework with Isolation Design Flow enables users to quickly utilize the provided hardware platform for their own development of the Xilinx® All Programmable Zynq® UltraScale+™ MPSoC based embedded multi-camera vision systems. The flexibility of the existing design is taken one step further by utilizing Dynamic Function eXchange (DFX) and Isolation Design Flow (IDF) solution that allows modifications of operating FPGA design by loading partial BIT files into dedicated and predefined reconfigurable partitions (RP) on FPGA. That partial design functionality will contain predefined and fully built Vitis Vision Library HLS functions ([REF \[2\]](#)) for provisional filtering of video inputs on all four cameras. Design comes with implemented Xilinx's Soft Mitigation IP (SEM) which is automatically configured, pre-verified solution for detection and correction of errors in Configuration Memory of Xilinx® FPGAs ([REF \[4\]](#)).

Also, the reference design for the Partial reconfiguration is done by following the new Vivado ML Edition 2021.1 modular design approach using Block Design Containers (BDC). This is a new Vivado feature that allows users to segment design into multiple block designs, enabling modular and team-based design flows, including the DFX. More information on these new features and Isolation Design Flow can be found in official Xilinx documentation ([REF \[1\]](#) and [REF \[3\]](#)).

The framework includes pre-verified logicBRICKS reference designs for video capture and HDMI display output under the Linux operating system. Video capture is performed using Xylon video cameras with the Maxim Integrated **GMSL2** high-speed digital video interface (GMSL2 version; see Figure 1). Reference design is prepared for hardware-centric Vivado® Design Suite 2021.1.



**Figure 1: Xylon logiREF-DFX-IDF-ZU Development Kit – GMSL2 Version**

This reference design is modular, and it defines static region and four reconfigurable partitions. Instead of starting from scratch and having to spend months designing and building a new design framework, the logiADAK-VDF-ZU design framework is used in the static block of the design. Four reconfigurable partitions are then added using the BDC design flow, and those partitions are fully controlled from the static block as shown in the Figure 2. Control of reconfigurable regions is performed using Xilinx new SIHA Manager for clock control, reset control, and decoupling logic for all four reconfigurable regions. The SIHA Manager IP is also part of the static block design. Following the IDF requirements, both static and reconfigurable partitions are isolated from each other.

The logiREF-DFX-IDF reference design includes Xylon logicBRICKS IP cores and hardware design files prepared for Xilinx Vivado ML Edition 2021.1. The design also includes IP cores built using Vitis Vision Library HLS and serving as HW accelerated filters: No filter, CCM Green filter and Sobel filter.

Hardware designers can customize design and add their own IP cores through the Vivado IP Integrator (IPI). The Linux OS and software drivers for logicBRICKS IP cores enable software developers to efficiently work with the framework, without knowing the hardware implementation details. This modular approach enables developing and adding additional logic for reconfigurable regions without any impact on the static portion of design. User can easily add for example some other filter from the Vitis Vision Library and prepare it for use in specific reconfigurable region. In this reference design, as mentioned before, three Vitis Vision Library filters (actually two filters and one pass through design) were prepared and built following DFX rules for use in partial reprogramming flow.

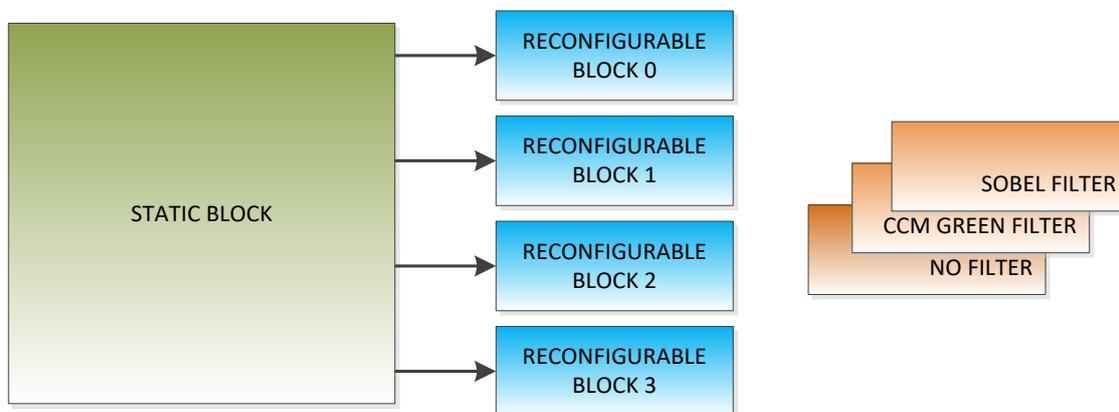


Figure 2: Xylon logiREF-DFX-IDF Simplified Block Diagram

## 1.1 Programmable Logic Utilization

The logiREF-DFX-IDF reference design utilizes just a small fraction of available programmable logic resources in the Xilinx Zynq UltraScale+ MPSoC XCZU9EG device. Free resources can be utilized by users altering pre-defined logicBRICKS configurations and changing programmable logic utilization. Please note that due to the nature of Dynamic Function eXchange workflow, multiple design runs have to be executed to get the complete design. The total number of runs is defined by number of

February 21 <sup>st</sup> , 2022		Version: v1.0.2
----------------------------------	--	-----------------

different DFX functions to be used. In this reference design, three reprogrammable DFX functions are used, so the total number of three Vivado runs had to be done. Utilization for each of these runs is shown in the tables below:

**Table 1: logiREF-DFX-IDF Reference Design Programmable Logic Utilization (No filter)**

Family (Device)	F (MHz)			LUT <sup>1</sup>	FF <sup>1</sup>	IOB <sup>2</sup>	BRAM	MULT/ DSP48/E	PLL / MMCM	BUFG	GTx	Design Tools
	mclk <sup>4</sup>	vclk <sup>4</sup>	rclk									
Zynq UltraScale+ <sup>3</sup> (XCZU9EG-2)	(200/200)	(150/200)	100	98048	123892	48	265	177	1/2	27	0	Vivado ML 2021.1

**Table 2: logiREF-DFX-IDF Reference Design Programmable Logic Utilization (CCM Green filter)**

Family (Device)	F (MHz)			LUT <sup>1</sup>	FF <sup>1</sup>	IOB <sup>2</sup>	BRAM	MULT/ DSP48/E	PLL / MMCM	BUFG	GTx	Design Tools
	mclk <sup>4</sup>	vclk <sup>4</sup>	rclk									
Zynq UltraScale+ <sup>3</sup> (XCZU9EG-2)	(200/200)	(150/200)	100	99472	123980	48	265	177	1/2	27	0	Vivado ML 2021.1

**Table 3: logiREF-DFX-IDF Reference Design Programmable Logic Utilization (Sobel filter)**

Family (Device)	F (MHz)			LUT <sup>1</sup>	FF <sup>1</sup>	IOB <sup>2</sup>	BRAM	MULT/ DSP48/E	PLL / MMCM	BUFG	GTx	Design Tools
	mclk <sup>4</sup>	vclk <sup>4</sup>	rclk									
Zynq UltraScale+ <sup>3</sup> (XCZU9EG-2)	(200/200)	(150/200)	100	103244	127084	48	283	177	1/2	27	0	Vivado ML 2021.1

Notes:

- 1) Assuming the following configuration: AXI Stream, RGB output, 32-bit AXI4-Lite register interface, 64-bit AXI4 memory interface with max. burst size of 64 words, scaling in both directions with multipliers (DSP48s), output stride set to 2048 pixels
- 2) Assuming only video inputs are routed off-chip, register and memory interfaces are connected internally
- 3) Only burst size of 16 words is supported on HP ports in the Xilinx Zynq UltraScale+ MPSoC
- 4) logiCVC/logiWIN clock frequencies

## 1.2 Hardware Requirements

The logiREF-DFX-IDF Vision Development Kit (Figure 1) includes the following hardware utilized by reference designs provided in the logiREF-DFX-IDF design framework:

- 1x Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit (Manufacturer part No.: EK-U1-ZCU102-G)
  - Kit package contains:
    - ZCU102 Evaluation Board (Manufacturer part No.: HW-Z1-ZCU102 Rev. 1.1)
    - Power supply
    - Xilinx license slip
- 1x Xylon GMSL2 FMC daughter card (part no.: logiFMC-GMSL2-9296A-12C)
- 4x Xylon GMSL2 video camera (part no.: logiCAM-GMSL2-AR0231-05525FM)
  - Camera package contains:
    - Xylon video camera
    - FAKRA cable assembly (5m)
- 3x Xylon 4 HFM Rosenberger connector adapter (part no: logiFMC-CBL-4HFM)
- 1x SD card
  - High/Extreme Capacity, High/Ultra Speed

**Table 4: Included Hardware**

Reference Design Hardware
1x Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit <sup>1</sup> (ver 1.1) with the XCZU9EG-FFVB1156-2 device
1x Xylon GMSL2 Deserializer FMC Module (Product code: logiFMC-GMSL2-9296)
4x 2.3-Mpix Xylon GMSL2 Cameras
1x SD card
4x FAKRA cable assemblies
Power supply



<sup>1</sup> – OEM kit version without the Xilinx Vivado Design Suite seat

<sup>2</sup> – logiREF-DFX-IDF is delivered in GMSL2 version

### 1.2.1 GMSL2 Deserializer FMC Module

The logiREF-DFX-IDF reference design delivered requires Xylon GMSL2 Deserializer FMC module (Figure 3), which comes as a part of the hardware kit deliverables.



**Figure 3: Xylon GMSL2 Deserializer FMC Module**

### 1.2.2 Xylon GMSL2 Camera

For transmissions of high-definition uncompressed video and camera control data the logiREF-DFX-IDF development kit includes Xylon cameras compatible with the Maxim Integrated GMSL2 high-speed digital video interface. Each camera includes the ON Semiconductor AR0231 2.3-megapixel camera sensor that combines high-definition (HD) 1928x1208p30 Full HD video with the color high dynamic range (HDR) functionality, GMSL2 serializer (transmitter) board, short cable lead with a connector and FIFO Optics 05525FM narrow-angle lens.

All camera parts are enclosed in the aluminium housing designed by Xylon. Its rugged metal construction provides excellent lens and imager module protection and enables safe and easy test vehicle installations.



**Figure 1.4: Xylon Video Camera – logiCAM-GMSL2-AR0231**

More information about Xylon video camera, its features and customization abilities can be obtained from Xylon Sales Team ([sales@logicbricks.com](mailto:sales@logicbricks.com)) and Xylon Support Team ([support@logicbricks.com](mailto:support@logicbricks.com)).

Video cameras are connected to FMC daughter card using Rosenberger Quadpod cable assemblies – L02-027-1000-Z-ZZZZ\_V2. These cable assemblies can be obtained from Avnet Electronics Marketing. The assembly is presented in **Figure 1.5**, with Rosenberger Quadpod connector.



**Figure 1.5: Rosenberger Quad HFM to 4 FAKRA Cable Assembly – L02-027-1000-Z-ZZZZ\_V2**

## 1.3 Software Requirements

The logiREF-DFX-IDF reference design and Xylon logicBRICKS IP cores are fully compatible with Xilinx development tools – Vivado ML 2021.1, Vitis 2021.1 and PetaLinux 2021.1. Future design releases shall be synchronized with the newest Xilinx development tools.

Demo software application is a Linux application developed for Petalinux 2021.1. For building Petalinux 2021.1 from the delivered Board Support Package (BSP), as well as rebuilding after making any modifications to the OS setup, such as adding/removing kernel drivers, changing kernel driver settings, changing rootfs settings, or simply making modifications to the user DTS files, a Petalinux 2021.1 build environment is required. This environment can be downloaded from the Xilinx website (<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html>). The environment can only be installed on Linux machines.

### 1.3.1 About Xilinx Vitis™ Development Environment

The Xilinx Vitis™ Development Environment is a specialized development environment for accelerating AI inference on Xilinx embedded platforms, Alveo accelerator cards, or on the FPGA-instances in the cloud. The Vitis™ Development Environment is Xilinx's development platform for AI inference on Xilinx hardware platforms, including both edge devices and Alveo™ cards. It consists of optimized IP, tools, libraries, models, and example designs. It is designed with high efficiency and ease-of-use in mind, unleashing the full potential of AI acceleration on Xilinx FPGA and ACAP.

Complete with the industry's first C/C++ full-system optimizing compiler, Xilinx Vitis™ Development Environment delivers system level profiling, automated software acceleration in programmable logic, automated system connectivity generation, and libraries to speed up programming.

More info about Xilinx Vitis™ can be found at Xilinx website (<https://www.xilinx.com/products/design-tools/vitis.html>).

***Xylon is the Vitis™ Development Environment development environment-qualified Xilinx Alliance Member and offers logicBRICKS IP cores, complete Xilinx All Programmable based solutions and design services.***

## 1.4 Design Deliverables

### 1.4.1 Hardware Design Files

- Configuration bitstreams files for the programmable logic and the Vitis export of the reference design that allows for instant design check-up and software changes
- Reference design prepared for Vivado ML 2021.1
- Xylon evaluation logicBRICKS IP cores:
  - logiCVC-ML Compact Multilayer Video Controller
  - logiWIN Versatile Video Input
- Xylon IP Cores that are not sold separately but only serve to augment this specific reference design:
  - TUSER-Trimmer

### 1.4.2 Software

- Linux user space drivers with driver examples
- Bare-metal software drivers for logicBRICKS IP cores
- logiVIOF VideoIn-VideoOut Library
- DFX-IDF Demo application sources

### 1.4.3 Binaries

- FPGA bitstreams
- Linux binaries:
  - boot.bin
    - First Stage Boot Loader
    - Universal Boot Loader
    - FPGA
    - Platform Management Unit Firmware
  - image.ub
    - kernel image
    - device tree blob
    - minimal Root File System
  - Four Camera DFX-IDF demo

## 1.5 Reference Design

The logiREF-DFX-IDF reference design implements four parallel video inputs from Xylon cameras, and the display output with the RGB graphic overlay. The design features a total of two MIPI CSI-2 input interfaces (ports) multiplexing video signal from four Xylon GMSSL2 video cameras and the Full HD display monitor output using the on board HDMI. All video inputs are stored in the video memory, and user can use on-board push buttons or console terminal attached to select any video input for the single camera full screen view, or all inputs can be viewed simultaneously in a 4x4 grid display.

In each case, different filters can be applied on any or all cameras independently. User applies partial reconfiguration on the running design using simple commands which load new bitstreams into the reconfigurable regions. The result is different filter effect applied on the chosen video input, running as the HW accelerator and shown on the screen in real time without restarting the board.

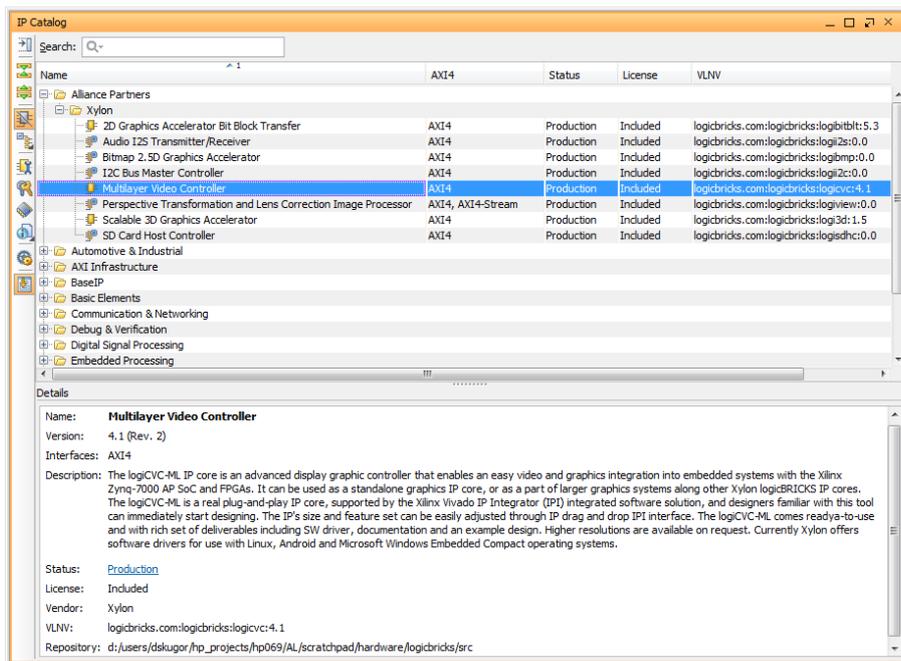
## 2 LOGICBRICKS IP CORES

### 2.1 About logicBRICKS IP Library

Xylon's logicBRICKS IP core library provides IP cores optimized for Xilinx All Programmable FPGA and SoC devices. The logicBRICKS IP cores shorten development time and enable fast design of complex embedded systems based on Xilinx All Programmable devices.

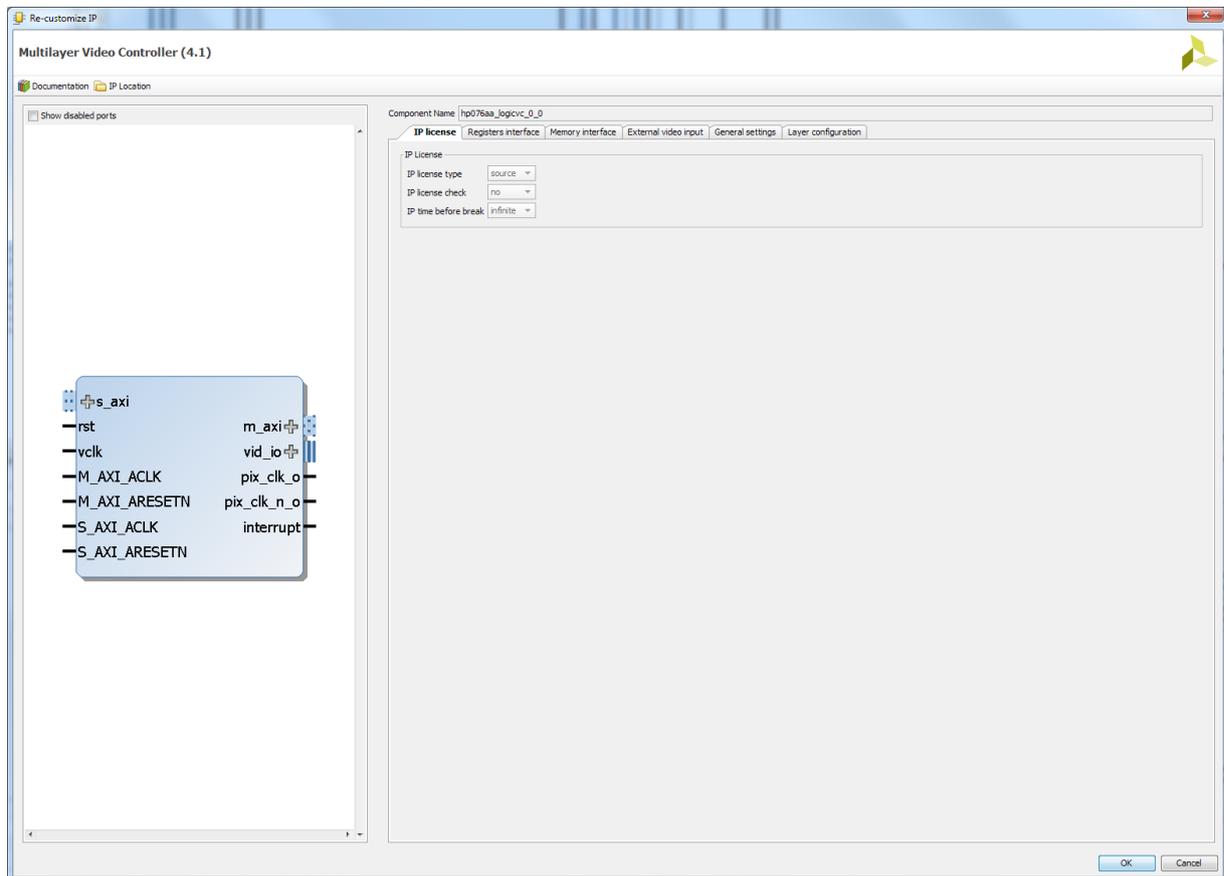
The key features of the logicBRICKS IP cores are:

- logicBRICKS can be used in the same way as Xilinx IP cores within the Xilinx Vivado Design Suite, and require no skills beyond general tools knowledge. IP users setup feature sets and programmable logic utilization through implementation tools' Graphical User Interface (GUI).
- Each logicBRICKS IP core comes with the extensive documentation, reference design examples and can be evaluated on reference hardware platforms. Xylon provides evaluation logicBRICKS IP cores to enable risk-free evaluation prior to purchase.
- Broad software support – from bare-metal software drivers to standard software drivers for different operating systems (OS). Standard software support allows graphics designers and software developers to use logicBRICKS in a familiar and comfortable way.
- Xylon assures skilled technical support.



**Figure 6: logicBRICKS IP Cores Imported into the Vivado IP Catalog**

The Figure 6 shows logicBRICKS IP cores imported into Vivado Design Suite, while the Figure 7 shows a typical logicBRICKS IP core's configuration GUI.



**Figure 7: Example of logicBRICKS IP Configuration GUI**

*To access logicBRICKS IP cores' User's Manuals, double-click on the specific IP core's icon, and then the Documentation icon in the opened IP configuration GUI. Choose either the Product guide to open the manual, or the Change Log to open IP core's change log.*

*logicBRICKS User's Manuals contain all necessary information about the IP cores' features, architecture, registers, modes of operation, etc.*

## 2.2 Evaluation logicBRICKS IP Cores

Xylon offers free evaluation logicBRICKS IP cores which enable full hardware evaluation:

- Import into the Xilinx Vivado tools (IP Integration)
- IP parameterization through the GUI interface
- Simulation (if Xilinx tools support it)
- Bitstream generation

The logicBRICKS evaluation IP cores are run-time limited and cease to function after some time. Proper operation can be restored by reloading the bitstream. Besides this run-time limitation, there are no other functional differences between the evaluation and fully licensed logicBRICKS IP cores.

Evaluation logicBRICKS IP cores are distributed as parts of the Xylon reference designs:  
<http://www.logicbricks.com/logicBRICKS/Reference-logicBRICKS-Design.aspx>.

Specific IP cores can be downloaded from Xylon's web shop:  
<http://www.logicbricks.com/Products/IP-Cores.aspx>.

## 2.3 logicBRICKS IP Cores Used in This Design

### 2.3.1 logiCVC-ML Compact Multilayer Video Controller



The logiCVC-ML IP core is an advanced display graphics controller for LCD and CRT displays, which enables an easy video and graphics integration into embedded systems with Xilinx Zynq-7000 All Programmable SoC and FPGAs.

This IP core is the cornerstone of all 2D and 3D GPUs. Though its main function is to provide flexible display control, it also includes hardware acceleration functions: three types of alpha blending, panning, buffering of multiple frames, etc.

- Supports all Xilinx FPGA families
- Supports LCD and CRT displays (easily tailored for special display types)
- Display resolutions up to 8192x8129 (including 4K2K resolution)
- Available SW drivers for: Linux and Microsoft Windows Embedded Compact OS
- Support for higher display resolutions available on request
- Supports up to 5 layers; the last one configurable as a background layer
- Configurable layers' size, position and offset
- Alpha blending and Color keyed transparency
- Pixel, layer, or Color Lookup Table (CLUT) alpha blending mode can be independently set for each layer
- Packed pixel layer memory organization:
  - RGB – 8bpp, 8bpp using CLUT, 16bpp 5-6-5, 24bpp 8-8-8 and 30bpp 10-10-10
  - YCbCr – 16bpp (4:2:2), 20bpp (4:2:2), 24bpp (4:4:4), 30bpp (4:4:4)
- Configurable ARM<sup>®</sup> AMBA<sup>®</sup> AXI4 memory interface data width (32, 64, 128 or 256 bits)
- Programmable layer memory base address and stride
- Simple programming due to small number of control registers
- Support for multiple output formats:
  - Parallel display data bus (RGB or YCrCb) with 1, 2 or 4 pix per clock: 12x2-bit, 15, 16, 18, 20, 24, 30 bit
  - Digital Video ITU-656: PAL and NTSC
  - LVDS output format: 3 or 4 data pairs plus clock
  - Camera link output format: 4 data pairs plus clock
  - DVI output format (currently not supported in US and US+ devices)
- Supports synchronization to external parallel input

- Versatile and programmable sync signals timing
- Double/triple buffering enables flicker-free reproduction
- Display power-on sequencing control signals
- Parametrical VHDL design that allows tuning of slice consumption and features set
- Prepared for Xilinx Vivado tools

More info: <http://www.logicbricks.com/Products/logiCVC-ML.aspx>

Datasheet: [http://www.logicbricks.com/Documentation/Datasheets/IP/logiCVC-ML\\_hds.pdf](http://www.logicbricks.com/Documentation/Datasheets/IP/logiCVC-ML_hds.pdf)

### 2.3.2 logiWIN Versatile Video Input



The logiWIN IP core enables easy implementation of video frame grabbers. Input video can be decoded, real-time scaled, de-interlaced, cropped, anti-aliased, positioned on the screen... Multiple logiWIN instances enable processing of multiple video inputs by a single Xilinx device.

- Supports versatile digital video input formats:
  - ITU656 and ITU1120 (PAL and NTSC)
  - RGB
  - YUV 4:2:2
- Maximum input and output resolutions are 2048 x 2048 pixels
- Built-in YcrCb to RGB converter, YUV to RGB converter and RGB to YcrCb converter
- Embedded image color enhancements: contrast, saturation, brightness and hue for ITU and YUV separately
- Real-time video scale-up (zoom in) up to 64x
- Real-time video scale-down (zoom out) down to 16 times
  - Lossless scaling down to 2x, or 4x in cascade scaling mode
- Supports video input cropping and smooth image positioning
- Configurable register interface; ARM® AMBA® AXI4-Lite
- ARM® AMBA® AXI4 and AXI4-Lite bus compliant
- Compressed stencil buffer in BRAM (mask over output buffer)
- Supports pixel alpha blending
- Provides “Bob” and “Weave” de-interlacing algorithms
- Supported big and little Endianness memory layout
- Double or triple buffering for flicker-free video
- Prepared for Xilinx Vivado tools

More info: <https://www.logicbricks.com/Products/logiWIN.aspx>

Datasheet: [https://www.logicbricks.com/Documentation/Datasheets/IP/logiWIN\\_hds.pdf](https://www.logicbricks.com/Documentation/Datasheets/IP/logiWIN_hds.pdf)

## 2.4 logicBRICKS IP Cores for Video Processing

Xylon offers several logicBRICKS IP cores for video processing on Xilinx All Programmable FPGA, SoC and MPSoC devices:

### logiVIEW Perspective Transformation and Lens Correction Image Processor



Removes fish-eye lens distortions and executes programmable transformations on multiple video inputs in real time. Programmable homographic transformation enables: cropping, resizing, rotating, transiting and arbitrary combinations. Arbitrary non-homographic transformations are supported by programmable Memory Look-Up Tables (MLUT).

More info: <http://www.logicbricks.com/Products/logiVIEW.aspx>

Datasheet: [http://www.logicbricks.com/Documentation/Datasheets/IP/logiVIEW\\_hds.pdf](http://www.logicbricks.com/Documentation/Datasheets/IP/logiVIEW_hds.pdf)

### logiISP Image Signal Processing (ISP) Pipeline



The logiISP Image Signal Processing Pipeline IP core is a full high-definition ISP pipeline designed for digital processing and image quality enhancements of an input video stream in Smarter Vision embedded designs based on Xilinx All Programmable devices.

More info: <http://www.logicbricks.com/Products/logiISP.aspx>

Datasheet: [http://www.logicbricks.com/Documentation/Datasheets/IP/logiISP\\_hds.pdf](http://www.logicbricks.com/Documentation/Datasheets/IP/logiISP_hds.pdf)

### logiHDR High Dynamic Range (HDR) Pipeline



Ultra-High Definition (UHD, including 4K2Kp60) HDR pipeline for camera image quality enhancements. Enables extraction of the maximum detail from high-contrast scenes, i.e. scenes with objects highlighted by a direct sunlight and objects placed in extreme shades.

More info: <http://www.logicbricks.com/Products/logiHDR.aspx>

Datasheet: [http://www.logicbricks.com/Documentation/Datasheets/IP/logiHDR\\_hds.pdf](http://www.logicbricks.com/Documentation/Datasheets/IP/logiHDR_hds.pdf)

### 3 GET AND INSTALL THE DESIGN FRAMEWORK



Customers entitled for the logiREF-DFX-IDF installation package delivery get the unique FTP or web download account from Xylon. To purchase the design framework, please visit our online catalogue: <http://www.logicbricks.com/Products/logiREF-DFX-IDF-ZU.aspx>

#### 3.1 Installation Process



Installation process is quick and easy. The logiREF-DFX-IDF framework can be downloaded as a cross-platform Java JAR self-extracting installer. Please make sure that you have a copy of the JRE (Java Runtime Environment) version 6 or higher on your system to run Java applications and applets. Double-click on the installer's icon to run the installation.

At the beginning, you will be requested to accept the design framework license – Figure 8. For installation in Linux OS, please follow instructions:

<http://www.logicbricks.com/logicBRICKS/Reference-logicBRICKS-Design/Xylon-Reference-Designs-Linux-Installation.aspx>.

If you agree with the conditions from the Xylon license, click NEXT and select the installation path for your logicBRICKS reference design (Figure 9). The installation process takes several minutes. It generates the folder structure described in the paragraph 3.1.1 Folder Structure.



Figure 8: Installation Process – Step 1

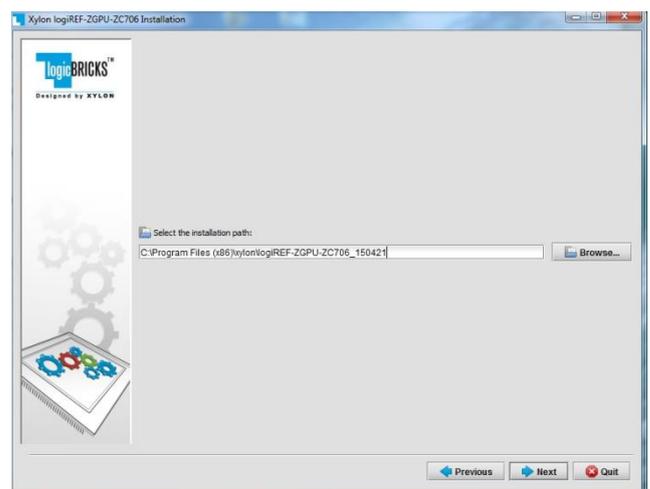


Figure 9: Installation Process – Step 2

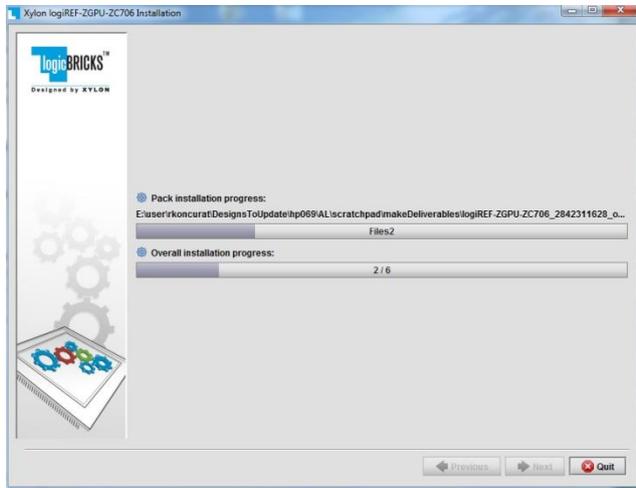


Figure 10: Installation Process – Step 3

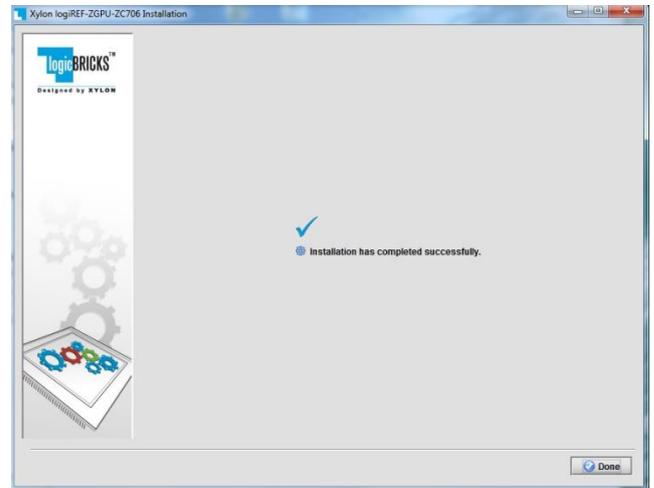


Figure 11: Installation Process – Step 4

### 3.1.1 Filesystem Permissions of the Installed Folder (Microsoft® Windows® OS)

The reference design installed in the default path `C:\Program Files\xylon` may inherit read-only filesystem permissions from the parent folder. This will block you in opening the hardware project file in Xilinx Vivado tools. Therefore it is necessary to change the filesystem permissions for the current user to “Full control” preferably.

To change the user permissions for `C:\Program Files\xylon` folder and all of its subdirectories, right click on the `C:\Program Files\xylon` folder and select “Properties”. Under “Security” tab select “Edit”. Select “Users” group in the list and check “Full control” checkbox in the “Allow” column.

## Folder Structure

Figure 12 gives a top level view of the directories and files included with the logiREF-DFX-IDF video design framework. Table 5 explains the purpose of directories.

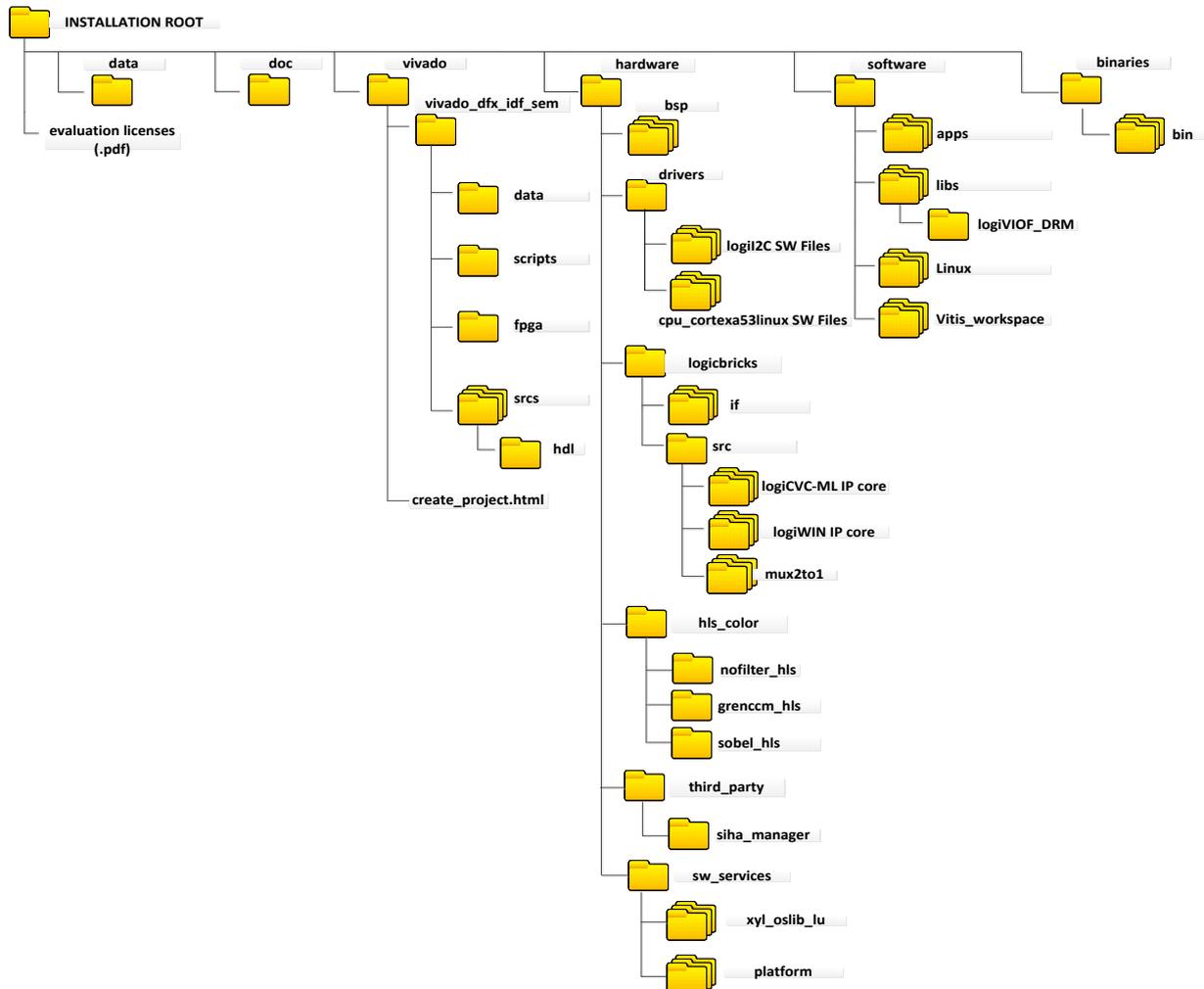


Figure 12: The Folder Structure

Folder	Purpose
INSTALLATION ROOT	This folder contains the <i>start.html</i> page – the jump-start navigation page through the reference design.
data	Additional hardware documentation/datasheets/manuals.
doc	Project documentation.
vivado	

vivado_dfx_idf_sem		This folder contains the complete Vivado project and files necessary for regenerating project.
	data	Design constraints files (XDC).
	srcs	Block design GUI script and HDL wrappers.
	scripts	TCL scripts to create block design from scratch.
	fpga	ZCU102 reference design bitstreams.
hardware		
	bsp	Xylon Linux user space Board Support Package (BSP); custom Xylon BSP compatible with the Xilinx Vitis. It enables users to quickly build Linux User space applications within the Vitis workspace.
	drivers	Standalone (bare-metal) drivers for logicBRICKS IP cores with documentation and examples.
	logicbricks/if	Xylon custom IP core interfaces (bus definitions).
	logicbricks/eval	Evaluation logicBRICKS IP cores. IP cores' User's Manuals are stored in <code>doc</code> subdirectories.
	hls_color	Builded Vitis Vision Library HLS IP cores
	third_party	Sources for the SIHA Manager IP. This IP is used for the clock/reset control of reconfigurable partitions in design.
	sw_services	<code>xyl_oslib_lu</code> – Xylon Linux User Space OS abstraction library for Xilinx– use in Linux User Space applications.
software		
	apps	DFX Demo application source files
	libs	Libraries source files used by DFX Demo Application
	Linux	Petalinux 2021.1 project files
	Vitis_workspace	Vitis workspace ready for importing to Vitis 2021.1
binaries		
	bin	Precompiled Linux DFX Demo files

**Table 5: Explanation of the logiREF-DFX-IDF folder structure**

## 4 GETTING THE IP LICENSES

### 4.1 Getting logicBRICKS IP Licences

The logiREF-DFX-IDF installation comes with the evaluation versions of the logicBRICKS IP cores, and in order to be able to change the provided reference designs, you need to request the proper licenses from Xylon.

Please contact Xylon Technical Support Service [support@logicbricks.com](mailto:support@logicbricks.com) and immediately provide your Ethernet MAC ID number or Sun Host ID.



For instructions how to find your Ethernet MAC or host ID, please visit:  
<http://www.logicbricks.com/Documentation/Article.aspx?articleID=KBA-01186-M0JXKD..>

For each logicBRICKS IP core used in the logiADAK-VDF-ZU reference designs Xylon will generate and send to you separated e-mails with the license keys (file) and full instructions for setting up the license key and downloading the logicBRICKS IP core. Please follow the provided instructions.

If you experience any troubles during the registration process, please contact Xylon Technical Support Service – [support@logicbricks.com](mailto:support@logicbricks.com).

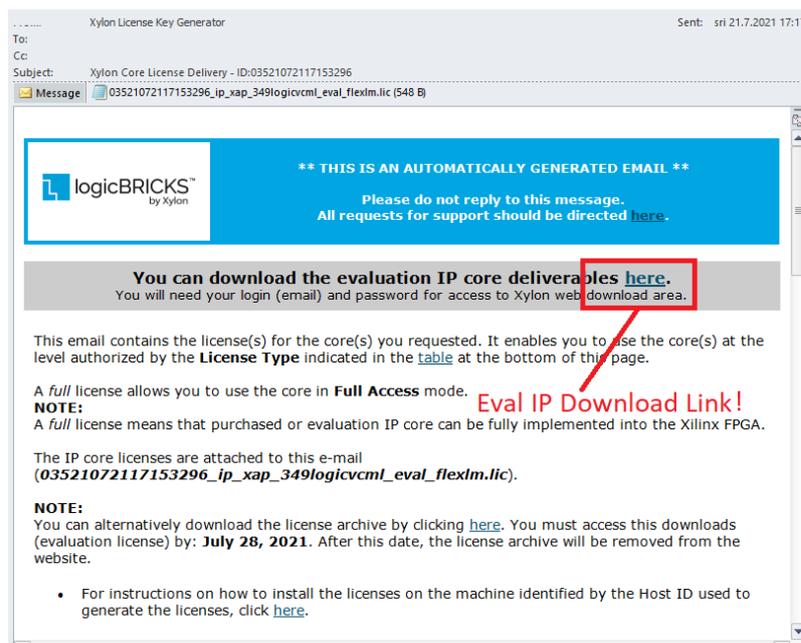


Figure 13: E-mail with logicBRICKS License and Download Instructions

## 4.2 Getting the Xilinx HDMI 1.4/2.0 Transmitter Subsystem License

The logiADAK-VDF-ZU reference design comes with the the Xilinx HDMI 1.4/2.0 Transmitter Subsystem hierarchical IP, and in order to be able to change the provided reference designs, a valid license for the IP Core Bundle is needed. The HDMI 1.4/2.0 Transmitter Subsystem is a hierarchical IP that bundles a collection of HDMI IP sub-cores and outputs them as a single IP.

Digital code vouchers for the Xilinx HDMI 1.4/2.0 Transmitter Subsystem IP are delivered by Xylon to all buyers of the logiVID-ZU Evaluation Kit for use with the logiADAK-VDF-ZU reference design. Xylon does not offer an alternative direct method of acquiring the license. Voucher codes can only be redeemed once.

To redeem the voucher code for the license given to you by Xylon, please perform the following steps:

1. Go to the following link [www.xilinx.com/getproduct](http://www.xilinx.com/getproduct) and login with your Xilinx account.
2. After logging in you should see the page shown in Figure 14.
3. Input the voucher code in the section where it says “Have a Voucher to Redeem?” and then hit Redeem Now.
4. You will then get asked if you want to redeem your voucher for “LogiCORE, HDMI, Site License” and after saying yes it will populate the Certificated Bases Licenses section with the newly added license option. You can now select it there and generate the license which is tied to your desired Ethernet MAC ID number.

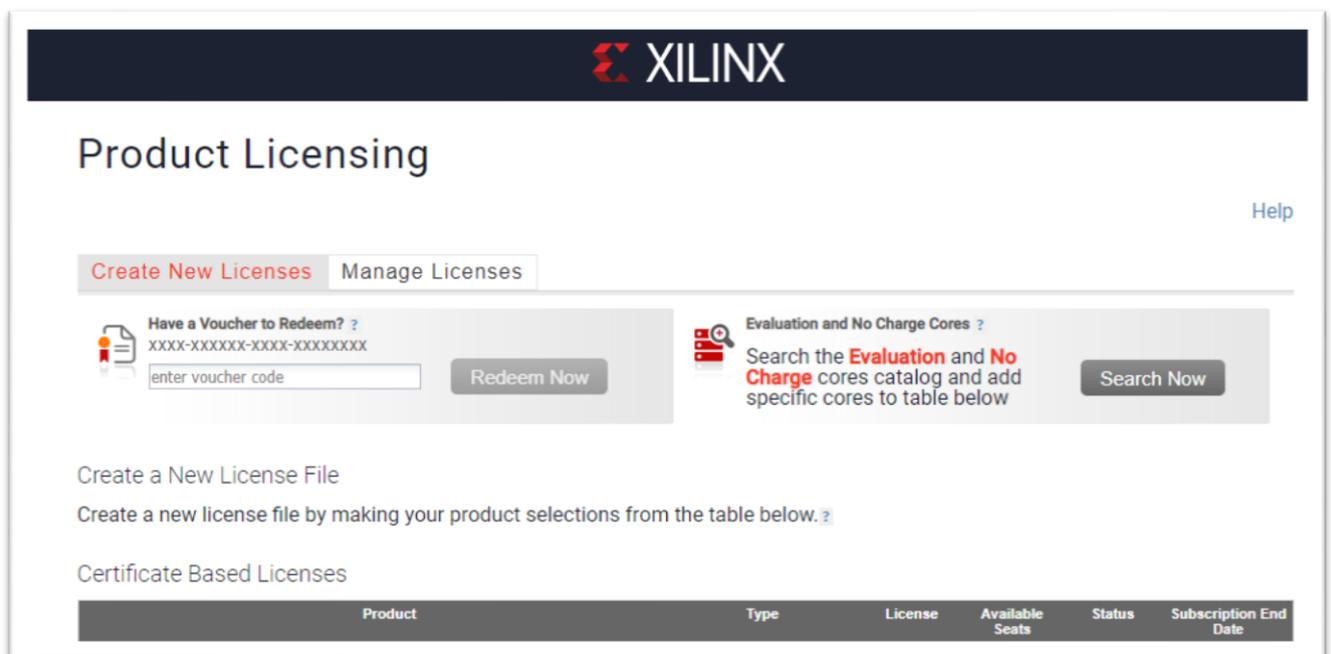


Figure 14: Xilinx License Page

If you experience any trouble during the redeeming process, please contact Xylon Technical Support Service – [support@logicbricks.com](mailto:support@logicbricks.com).

## 5 LOGIREF-DFX-IDF REFERENCE DESIGN

### 5.1 logiREF-DFX-IDF SoC Design and Memory Layout

logiREF-DFX-IDF contains four video camera Vitis platform design utilizing ZCU102 board and four Xylon GMSL2 video cameras with Xylon's GMSL2 FMC daughter card. Xylon's platform for the targeted hardware kit is designed following Xilinx's Vitis Unified Software Platform Documentation, UG1400 (v2021.1). Platform design is based on the Xylon's VDF Reference Design functionality with addition of the Dynamic Function eXchange functionality (DFX) that is used for partial reconfiguration of dedicated FPGA regions with HLS C-based filters. Filter implementations provide simple examples how to implement C-code based hardware accelerators in the Programmable Logic with usage of Vivado HLS tool and Vivado IP packager.

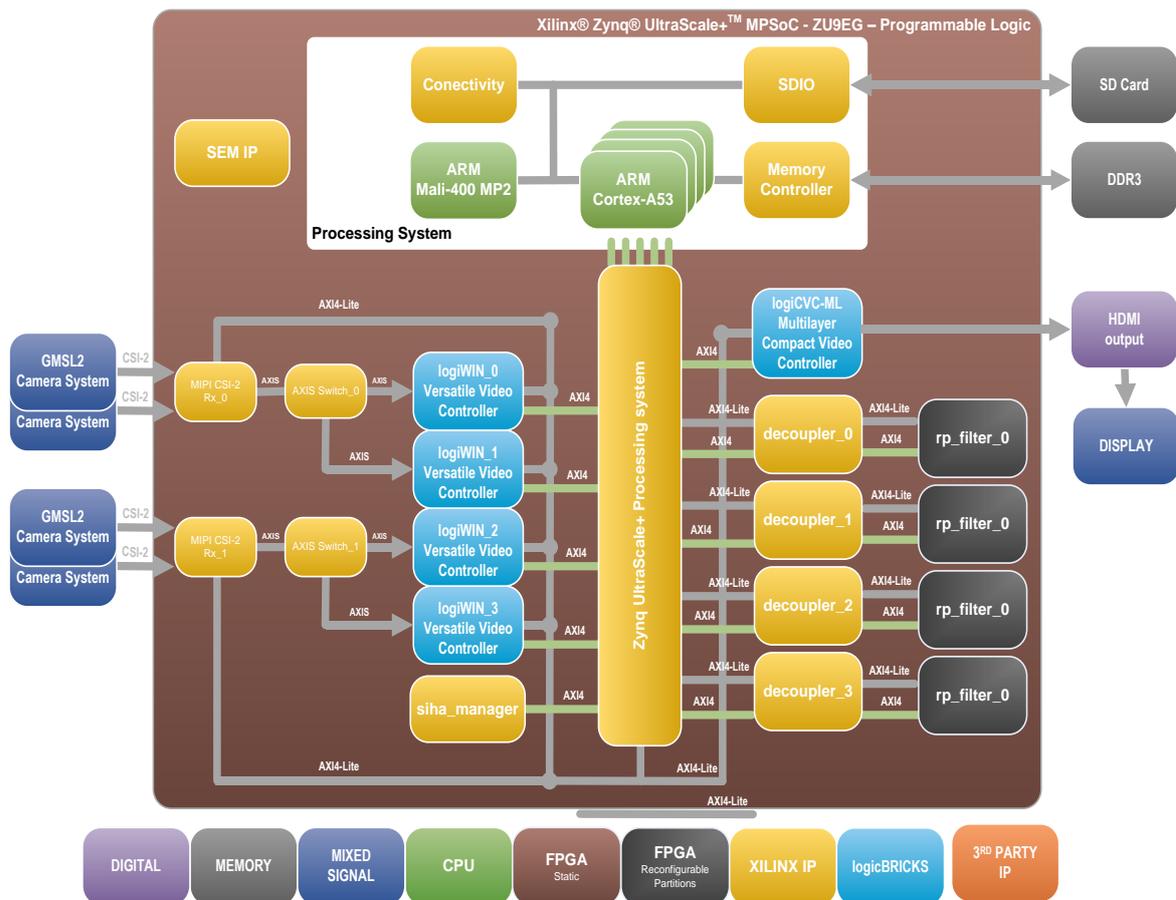
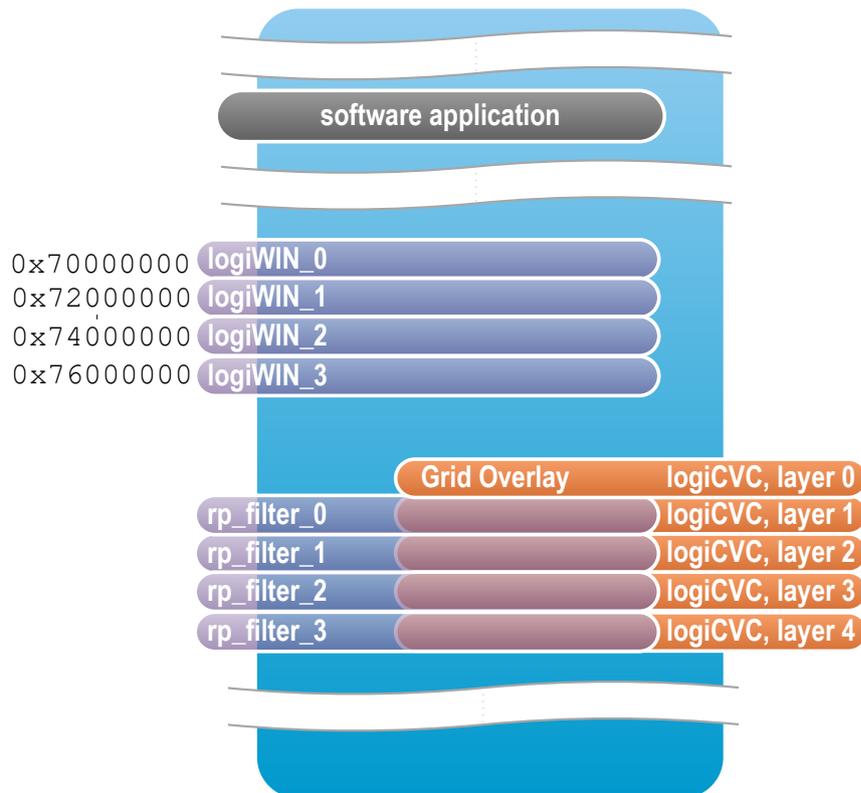


Figure 15: logiREF-DFX-IDF MPSoC Design Block Diagram

(Clock Generator Module and other utility IP cores are not shown)

The Figure 16 shows the memory layout of video buffers. Each logiCVC layer has its own memory space reserved and multiple video buffers (not shown). Four filters in reconfigurable partition process inputs from logiWINs and results are transferred to be used by four logiCVC layers. The Grid Overlay logiCVC layer is used by the application software to display filter, IP statistics and other information.



**Figure 16: logiREF-DFX-IDF Design Memory Layout**

Four video cameras are grouped into two pairs: video cameras #1 and #2 and video cameras #3 and #4. Each pair is connected to a dedicated dual link GMSL2 deserializer on the FMC daughter card. One camera in the pair is connected through the link A, and the other through the link B. Since these are 2.3 Mpix video cameras outputting video signal at 30 fps rate, a single 4-lane MIPI interface has enough bandwidth to transfer video signals from both video cameras. Deserializers are configured for transfer rate of 1200 Mb/s per lane to cater for such video signals on a single MIPI port. Each MIPI port in the Receiver Subsystem is connected to one Xilinx MIPI CSI-2 RX IP core decoding MIPI to AXI4-Stream with accompanied virtual channel information: VC0 for video signal coming from one video camera (GMSL2 link A), and VC1 for video signal coming from the other video camera (GMSL2 link B) on the same GMSL2 deserializer device. Xilinx MIPI CSI-2 RX IP core is a MIPI CSI Controller Subsystem IP that requires licensing. There are no licenses available as part of Vivado Design Suite installation, regardless of the installed version and/or package. License, either purchased or evaluation can be obtained from Xilinx (<https://www.xilinx.com/products/intellectual-property/ef-di-mipi-csi-rx.html#overview>).

More information about the IP core can be found in the Xilinx product guide:

[https://www.xilinx.com/support/documentation/ip\\_documentation/mipi\\_csi2\\_rx\\_subsystem/v4\\_1/pg232-mipi-csi2-rx.pdf](https://www.xilinx.com/support/documentation/ip_documentation/mipi_csi2_rx_subsystem/v4_1/pg232-mipi-csi2-rx.pdf).

AXI4-Stream Switch IP core separates incoming AXI4-Stream into two streams according to the virtual channel information, and feeds two logiWIN IP core instances simultaneously. Each logiWIN IP core instance stores the video image in the assigned video buffer matching the source image address for the filters inside the reconfigurable region for that channel. Further, logiCVC-ML IP core layers are mapped so that they show video images from four reconfigurable regions filter outputs, and display it on the HDMI output. When these video images are to be displayed in a tiled fashion, each logiWIN IP core instance performs scaling down and adequate positioning of the video image on the monitor.

Video cameras provide video signal in 8-bit YUV format (YUV422). This format is preserved through the input video chain, but is then converted to 32-bit RGB format (ARGB888) in the logiWIN IP core, and as such stored into video buffer. This means that the corresponding layer of the logiCVC-ML IP core instance is configured for RGB888 pixel format. As explained before, there is additional conversion to and from RGBA format implemented within HLS filters, and not shown here.

**Table 5.1** shows how logiCVC layers are configured and organised with respect to design features. The first layer is used as the overlay grid layer, and the other four layers are used for outputs from the reconfigurable partitions filters.

**Table 5.1: logiCVC-ML IP Core Layer Configuration and Assignment**

logiCVC-ML IP core layer	Layer format	Alpha blending type	Buffer offset	Design feature
layer 0	32bpp – RGB	Pixel	1080 lines	Overlay grid
layer 1	32bpp – RGB	Pixel	1080 lines	RP Filter 0 Result
layer 2	32bpp – RGB	Pixel	1080 lines	RP Filter 1 Result
layer 3	32bpp – RGB	Pixel	1080 lines	RP Filter 2 Result
layer 4	32bpp – RGB	Pixel	1080 lines	RP Filter 3 Result

### 5.1.1.1 Registers Address Map

All IP cores in the reference design that have their AXI4-Lite interface connected to AXI4 infrastructure for register access have a physical base address and address range assigned. These assignment present registers address map accessible by the MPSoC's APU and all other PL and PS modules that are masters in this AXI4 infrastructure. In the logiREF-DFX-IDF Reference Design, the APU is the only master through the M\_AXI\_HPM0\_FPD PS-PL interface port. The registers address map is presented in **Table 5.2**.

**Table 5.2: logiREF-DFX-IDF Reference Design Registers Address Map**

IP core instance	Base address	High address	Range [kB]
rp_filter_0	0x00_8002_0000	0x00_8002_FFFF	64
rp_filter_0_r	0x00_8007_0000	0x00_8007_FFFF	64
rp_filter_1	0x00_8009_0000	0x00_8009_FFFF	64
rp_filter_1_r	0x00_800B_0000	0x00_800B_FFFF	64
rp_filter_2	0x00_800D_0000	0x00_800D_FFFF	64
rp_filter_2_r	0x00_800E_0000	0x00_800E_FFFF	64
rp_filter_3	0x00_800F_0000	0x00_800F_FFFF	64
rp_filter_3_r	0x00_8012_0000	0x00_8012_FFFF	64
logicvc_0	0x00_8001_0000	0x00_8001_FFFF	64

IP core instance	Base address	High address	Range [kB]
logiwin_0	0x00_8003_0000	0x00_8003_FFFF	64
logiwin_1	0x00_8004_0000	0x00_8004_FFFF	64
logiwin_2	0x00_8005_0000	0x00_8005_FFFF	64
logiwin_3	0x00_8006_0000	0x00_8006_FFFF	64
v_hdmi_tx	0x00_8010_0000	0x00_8011_FFFF	128
vid_phy_controller	0x00_800C_0000	0x00_800C_1FFF	64
axi_iic_0	0x00_8000_0000	0x00_8000_0FFF	4
mipi_csi2_rx_subsystem_0	0x00_8008_0000	0x00_8008_1FFF	8
mipi_csi2_rx_subsystem_1	0x00_800A_0000	0x00_800A_1FFF	8
SIHA_Manager	0x00_8013_0000	0x00_8013_1FFF	64

## 5.2 Video Input/Output Synchronization

Hardware synchronization is implemented between the logiWIN and logiCVC-ML IP cores. SoC systems implementing video input units have video input frame rates and video output frame rates rarely equal, and need to implement frame rate conversions from lower to higher frame rate, or vice versa.

### 5.2.1 logiWIN Hardware Buffering Implementation

Double/triple buffering state machine is placed outside the logiWIN. The logiWIN output pin *curr\_vbuff[1:0]* sends to the double/triple buffer external controller information to which buffer the logiWIN currently writes. On the input pin *next\_vbuff[1:0]* the double/triple buffer external controller sends to the logiWIN information to which buffer the next frame should be written. For double buffering *next\_vbuff* value changes between 0 and 1, and for triple buffering between 0, 1 and 2. If double/triple buffering is not in use, the *next\_vbuff* must be set to 0. Output signal *sw\_vbuff\_req* signals to the external controller that the current buffer *curr\_vbuff[1:0]* is written and requests buffer switching. External controller grants buffer switching over *sw\_vbuff\_grant* together with the pointer to the next buffer *next\_vbuff[1:0]*.

### 5.2.2 logiCVC-ML Hardware Buffering Implementation

External video synchronization requires three separate frame buffers (*buffer\_0*, *buffer\_1* and *buffer\_2* implemented in the video memory). In SoC designs with the logiCVC-ML display controller IP core, three frame buffers must be setup for every logiCVC-ML graphic layer. The triple buffering method provides an advantage over the double buffering synchronization method, since the video input units do not have to wait on buffers swapping as there is always a spare frame buffer for new frame data writing.

To support this feature, the logiCVC-ML uses video input synchronization control port which consists of *e\_curr\_vbuff[C\_NUM\_OF\_LAYERS\*2-1:0]* and *e\_switch\_vbuff[C\_NUM\_OF\_LAYERS-1:0]* input signals, and *e\_next\_vbuff[C\_NUM\_OF\_LAYERS\*2-1:0]* and *e\_switch\_grant[C\_NUM\_OF\_LAYERS-1:0]* output signals.

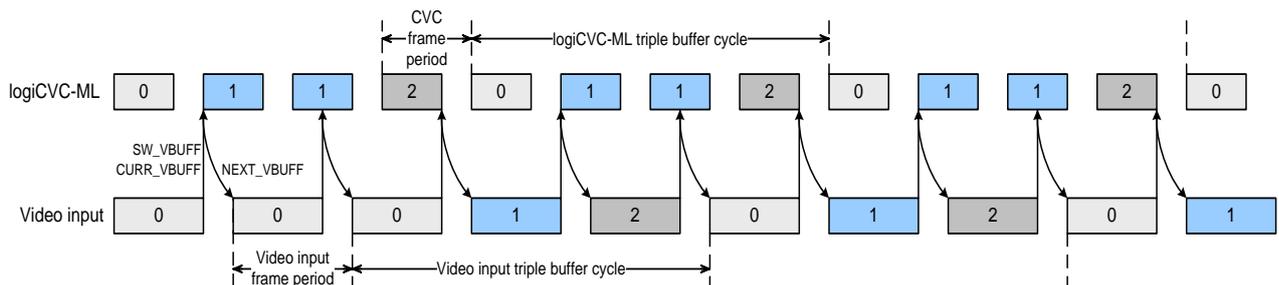
With the input signals *e\_current\_vbuff[n\*2+1:n\*2]* and *e\_switch\_vbuff[n]* external video source signals to logiCVC-ML layer *n* the currently written buffer and when to switch buffers (typically on the end of its active frame of external video source). With the output signal

$e\_switch\_grant[C\_NUM\_OF\_LAYERS-1:0]$  the logiCVC-ML grants the video source to start writing its next frame to  $e\_next\_vbuff[n*2+1:n*2]$  buffer.

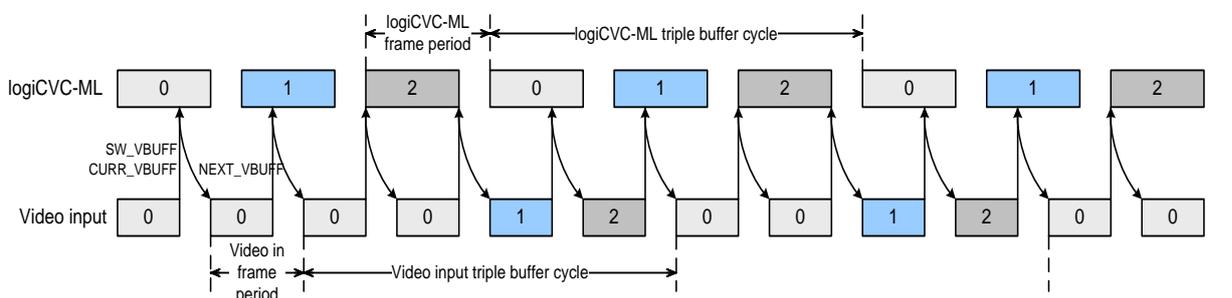
The logiCVC-ML IP core is constantly sampling  $e\_current\_vbuff$  and  $e\_switch\_vbuff$  inputs with the memory clock. When  $e\_switch\_vbuff$  high state is detected, the logiCVC samples  $e\_current\_vbuff$  and asserts  $e\_switch\_grant$  along with the associated  $e\_next\_vbuff$ . External logic should constantly sample  $e\_switch\_grant$  signal, and when it detects that  $e\_switch\_grant$  is high, it should sample  $e\_next\_vbuff$  and de-assert  $e\_switch\_vbuff$ . When logiCVC detects  $e\_switch\_vbuff$  low, it de-asserts  $e\_switch\_grant$  signal on the next memory clock cycle.  $e\_switch\_vbuff$  and  $e\_switch\_grant$  signals are used as handshake signals between logiCVC and external logic. This kind of implementation supports buffers switching between logiCVC and external logic running on synchronous and on asynchronous clocks.

To enable external frame buffer synchronization for a specific graphic layer, user has to enable it by setting the EN\_EXT\_VBUFF\_SW bit to 1 in the corresponding layer control register.

If external video input signals are connected to the logiCVC-ML's video input synchronization control port and synchronization are turned off (EN\_EXT\_VBUFF\_SW=0), logiCVC-ML will always signal the external video input to write data to buffer 0, i.e.  $e\_next\_vbuff[n*2+1:n*2]=0$ . At the same time, logiCVC-ML will work in the CPU synchronization mode so it will read memory buffer, which is defined with layer address register.



**Figure 17: Triple Buffering Example when logiCVC-ML Refresh Rate is Higher than Video Input**



**Figure 18: Triple Buffering Example when logiCVC-ML Refresh Rate is Lower than Video Input**

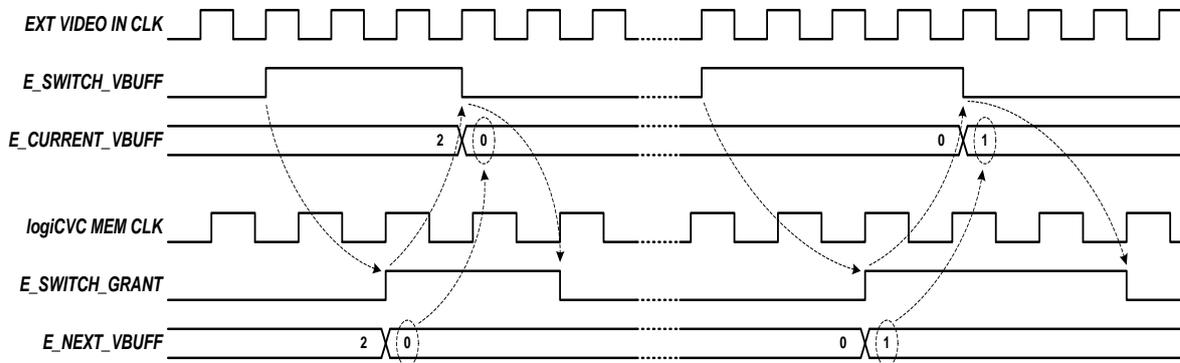


Figure 19: External buffer control signals timing diagram

### 5.3 Xylon logicBRICKS IP Core Configuration

In the reference design there are several Xylon IP cores used. There are logicBRICKS IP cores, which are delivered as evaluation IP cores and other auxiliary Xylon IP cores which are license-free and in source code. More information on Xylon logicBRICKS library of IP cores and its concept, as well as evaluation editions can be found in chapter **LOGICBRICKS IP CORES**

Auxiliary Xylon IP cores used in the reference design is TUSER Trimmer IP. TUSER Trimmer IP core is used to adjust TUSER control signal on AXI4-Stream channels driven by Xilinx MIPI CSI2-RX IP cores.

The logicBRICKS IP cores are:

- logiCVC-ML – Compact Video Controller Multilayer Alpha Blending (see section 2.3.1 **logiCVC-ML Compact Multilayer Video Controller** for details).
- logiWIN – Versatile Video Input (see section 2.3.2 **logiWIN Versatile Video Input** for details).

Configuration of logiCVC-ML IP core instance in the VDF Reference Design is:

- Register interface: AXI4-Lite slave
- Register interface data width: 32 bits
- Readable registers: yes
- Memory interface: AXI4 master
- Memory interface data width: 128 bits
- Memory interface burst size: 64 words
- Number of layers: 5
- Background layer: no
- External video input: no
- Layer configuration: see Table 5.1
- Layer size position: yes
- Triple buffering interface: yes
- Use DSP resources: yes
- Output video type: parallel video interface with separate synchronization signals
- Output video interface format: 24-bit RGB

Configuration of logiWIN IP core instances 0 to 3 in the VDF Reference Design is:

- Register interface: AXI4-Lite slave
- Register interface data width: 32 bits
- Readable registers: yes
- Memory interface: AXI4 master
- Memory interface data width: 64 bits
- Memory interface burst size: 64 words
- Row stride: 2048 pixels
- Buffer switch offset: 1080 lines
- Input video type: 16-bit AXI4-Stream
- Input video format: 8-bit YUV422
- Maximum input resolution: 2048 (pixels/lines)
- Output video image storage format: ARGB888
- Output video image storing method: burst by burst
- Maximum output resolution: 2048 (pixels/lines)
- Horizontal scaling: yes
- Vertical scaling: yes
- Scaler uses: DSP48 components
- Cascade scaling: no
- Parallel interpolation: no

## 5.4 SIHA IP Manager

The SIHA Manager IP is provided directly from Xilinx and it is used for control of four dedicated configurable regions. This IP manages clocks, resets and AXI\$ cache/port signalling between static and reconfigurable regions. Also, the SIHA Manager IP is directly in control of decoupling logic between static and reconfigurable partitions by using dedicated decouple signals for all four Xilinx's DFX decoupler IP.

More about DFX Decoupler can be found at <https://www.xilinx.com/products/intellectual-property/dfx-decoupler.html>). In this design, the SIHA IP Manager is configured for total of nine reconfigurable slots, but only four are used in design.

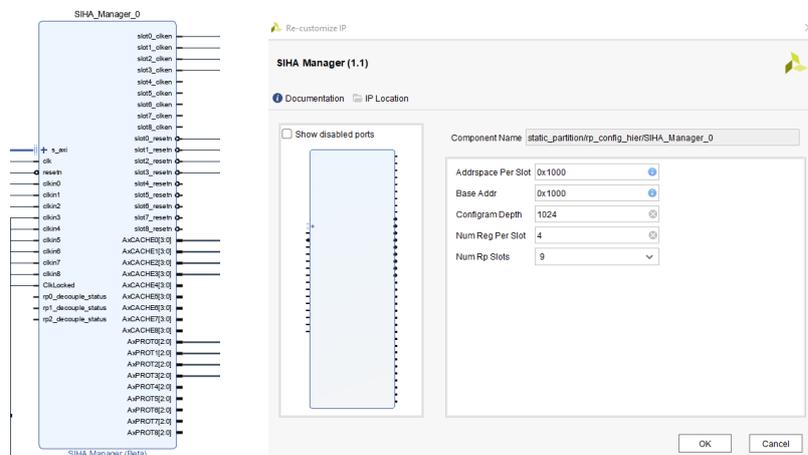


Figure 20: SIHA Manager IP Configuration

## 5.5 SEM IP Controller

The IP Soft Error Mitigation (SEM) Controller is an automatically configured, pre-verified solution to detect and correct soft errors in Configuration Memory of Xilinx FPGAs. Soft errors are unintended changes to the values stored in state elements caused by ionizing radiation. The SEM Controller does not prevent soft errors; however, it provides a method to better manage the system-level effects of soft errors. Proper management of these events can increase reliability and availability, and reduce system maintenance and downtime costs. More info about the IP can be found here ([REF \[4\]](#)).

In logiREF-DFX-IDF design SEM IP is integrated inside the static partition and it is used in Mitigation and Testing mode, with Error injection and Error correction functionality enabled.

Basic **Advanced Mitigation** Summary

General

Mode: Mitigation and Testing

Controller Clock Period (ps): 10000 [5000 - 125000]

Structural Options

ICAP and FRAME\_ECC placement: Core

Mode description

Features	Modes					
	Mitigation + testing	Mitigation only	Detect + testing	Detect only	Emulation	Monitoring
IP state after initialization	OBSV	OBSV	DETECT	DETECT	IDLE	IDLE
Correction (Repair)	✓	✓	NA	NA	NA	NA
Classification	optional	optional	NA	NA	NA	NA
Error injection	✓	NA	✓	NA	✓	NA
Debugging features: -Transition to IDLE state -Configuration frames and register reads -External memory reads (if classification is enabled) -Address translations	✓	✓	✓	✓	✓	✓
On-demand detect features: -Detect-only -Diagnostic Scan	✓	✓	✓	✓	✓	✓

Figure 21: SEM IP Configuration

## 5.6 Restoring Full MPSoC Design from Xylon Deliverables

Xylon provides all necessary design files to enable full project restore in the Xilinx Vivado ML 2021.1.

If the logiREF-DFX-IDF Vivado project was never opened before in current user environment, the script for generating the project file should be used first as it should properly setup the project and generate the Vivado project file (.xpr). The project creation script can be found under the project root installation folder:

***logiREF-DFX-IDF/deliverables/vivado/vivado\_dfx\_idf\_sem/scripts/project.tcl***

The script should either be used from the command line (running Vivado in batch mode) or it can be sourced from within Vivado tcl console if Vivado was started in project mode. Linux users should switch to the folder where script is located before sourcing the script to avoid possible permission issues. Windows batch file **create\_project.bat** is provided for convenience for Windows users.



The following IDF related properties setting and the IDF patch application instructions had to be applied for Vivado 2021.1 release used. The IDF patch may not be necessary or different properties might have to be applied for later Vivado versions. Please read the Xilinx IDF related documentation and set the project properties accordingly.

Before opening and building design, additional properties for the successful DFX and IDF solution needs to be set trough the Vivado tcl console, or those can be easily set in *Vivado\_init.tcl* file located at {Vivado installation folder}\scripts folder. If file doesn't exist, create the file and add commands below:

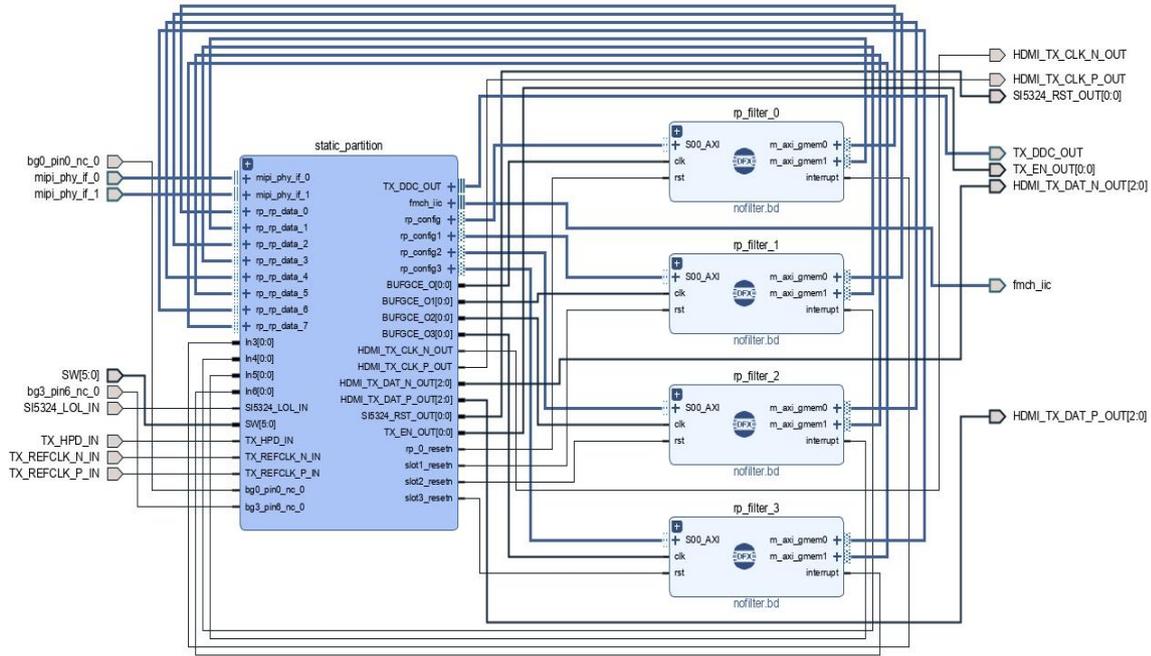
```
set_param project.enableVersallPIPRFlow 1
set_param hd.enableIDFPR 1
set_param hd.enableIDFDRC 1
set_param hd.enableIDFPRFanoutSupport false
set_param hd.enableUnifiedRoutingFootprint false
```

After setting those parameters, additional Vivado patch for DFX/IDF solution has to be applied for proper design to be built. This Vivado patch is located inside the compressed AR000032523\_vivado\_2021\_1\_preliminary\_rev1.zip file obtained from the Xilinx support. Just extract the whole content to the {VIVADO\_INSTALL\_DIRECTORY/patches}. If patches directory did not exist, create it manually. After editing the Vivado initialization script and applying Vivado patch for DFX/IDF solution, the project can be opened using the project file located at

***logiREF-DFX-IDF/vivado/vivado\_dfx\_idf\_sem/hp130ba/hp130.xpr***

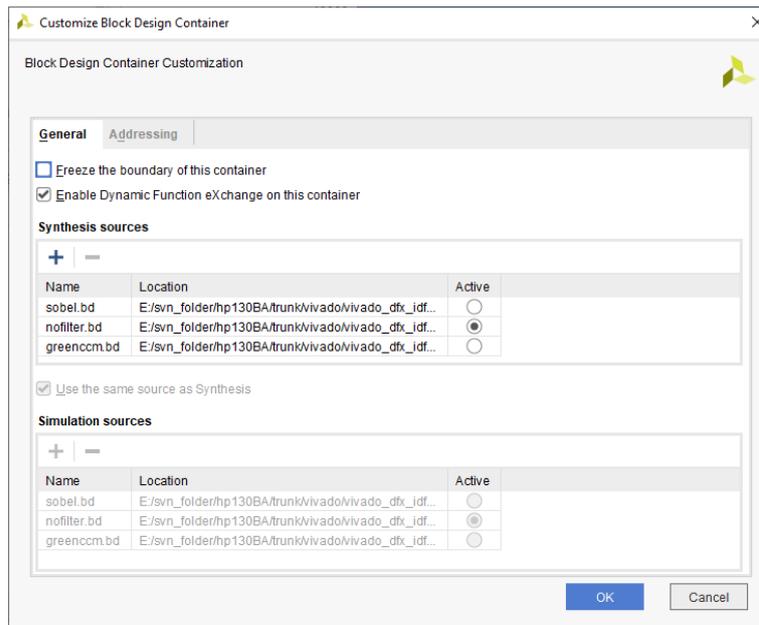
Finally, the complete design can be built from the scratch.

Before running the build, designer can select which of those three options will be default. In this case, the 'No filter' option is selected for all four BDCs.



**Figure 22: logiREF-DFX-IDF Vivado IP Integrator Block Diagram**

This is done by double clicking DFX blocks and selecting active synthesis source. Please, make sure the option “Enable Dynamic Function eXchange on this container” is properly selected as shown in the **Figure 23**.



**Figure 23: BDC Customization Dialog Box**

Finally, the design opened should be validated to make sure everything was properly initialized, and the bitstream can be generated. It is a complex design, and it will take a while before all design runs are finished and bitstreams generated. Of course, if no design changes were needed, design rebuild is not necessary since all bitstreams and the hardware export file (XSA file) are already generated.

Please note that DFX workflow requires manual floorplanning for all DFX blocks (reconfigurable partitions) and this is already defined in design constraints files. There are three runs defined, so this can be time consuming, but the result should be as shown on Design Runs tab upon completion.

Name	Configuration	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1 (active)		constrs_1	synth_design Complete!								63	54	0.0	0	0
✓ impl_1 (active)	config_1	constrs_1	write_bitstream Complete!	0.115	0.000	0.010	0.000	0.000	6.481	0	98048	123892	265.0	0	177
✓ child_0_impl_1	config_2	constrs_1	write_bitstream Complete!	0.115	0.000	0.010	0.000	0.000	6.509	0	99472	123980	265.0	0	177
✓ child_1_impl_1	config_3	constrs_1	write_bitstream Complete!	0.115	0.000	0.010	0.000	0.000	6.615	0	103244	127084	283.0	0	177

Figure 24: Design Runs Statistics

Single configuration run produces the complete bitstream for that configuration (static + reconfigurable combined) and four additional partial bitstreams. Those partial bitstreams from the first run are partial bitstreams for every reconfigurable partition. So, for the first run, four 'No filter' partials are created. Same goes on for the second and the third run, where four CCM Green filter partials and four Sobel partials are created respectively. Total number of partial bitstreams that will be used in demonstrating DFX functionality is twelve, as there are three bitstreams for each of four reconfigurable partitions. The hardware (.xsa) file exported contains the default configuration setup that is used for building the Petalinux image and application. This hardware image can be used as is by the SW developer who can start developing application using default filters inside reconfigurable partitions. In order to change the filter loaded into reconfigurable partition from default to some other selected and already preloaded into the PSU RAM, some interaction is still needed with the FPGA manager utility within the application to reprogram partitions. The example how it could be done is given within the logiREF-DFX-IDF application demo.

A closer look at implemented FPGA design as shown in the **Figure 26** illustrates which portions of FPGA device were used and defined as reconfigurable regions. The area outside the configurable partitions is available to the static portion of the design.

The placement of reconfigurable regions is purely provisional and is totally up to designer to achieve it correctly by following the DFX workflow rules. The same floorplan can be seen when opening the second and third implementation run. It is visible that the placement of reconfigurable partitions is exactly the same for each run, only the utilization of used logic inside partitions depends on complexity of the filter selected. The area reserved must be large enough to provide enough space for the most complex filter to be placed inside of that particular area.

The static partition pblock and the reconfigurable pblocks floorplan are defined in the **physical\_constraints.xdc** file:

```
create_pblock rp_pblock_0
add_cells_to_pblock [get_pblocks rp_pblock_0] [get_cells -quiet [list hp130ba_i/rp_filter_0]]
resize_pblock [get_pblocks rp_pblock_0] -add {CLOCKREGION_X0Y6:CLOCKREGION_X0Y6}
set_property CONTAIN_ROUTING 1 [get_pblocks rp_pblock_0]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks rp_pblock_0]
```

```
set_property SNAPPING_MODE ON [get_pblocks rp_pblock_0]
```

```
create_pblock rp_pblock_1
add_cells_to_pblock [get_pblocks rp_pblock_1] [get_cells -quiet [list hp130ba_i/rp_filter_1]]
resize_pblock [get_pblocks rp_pblock_1] -add {CLOCKREGION_X2Y6:CLOCKREGION_X2Y6}
set_property CONTAIN_ROUTING 1 [get_pblocks rp_pblock_1]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks rp_pblock_1]
set_property SNAPPING_MODE ON [get_pblocks rp_pblock_1]
```

```
create_pblock rp_pblock_2
add_cells_to_pblock [get_pblocks rp_pblock_2] [get_cells -quiet [list hp130ba_i/rp_filter_2]]
resize_pblock [get_pblocks rp_pblock_2] -add {CLOCKREGION_X0Y3:CLOCKREGION_X0Y3}
set_property CONTAIN_ROUTING 1 [get_pblocks rp_pblock_2]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks rp_pblock_2]
set_property SNAPPING_MODE ON [get_pblocks rp_pblock_2]
```

```
create_pblock rp_pblock_3
add_cells_to_pblock [get_pblocks rp_pblock_3] [get_cells -quiet [list hp130ba_i/rp_filter_3]]
resize_pblock [get_pblocks rp_pblock_3] -add {CLOCKREGION_X2Y4:CLOCKREGION_X2Y4}
set_property CONTAIN_ROUTING 1 [get_pblocks rp_pblock_3]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks rp_pblock_3]
set_property SNAPPING_MODE ON [get_pblocks rp_pblock_3]
```

```
create_pblock static_pblock
add_cells_to_pblock [get_pblocks static_pblock] [get_cells -quiet [list SEM_UART_inst
hp130ba_i/static_partition]]
resize_pblock [get_pblocks static_pblock] -add {CLOCKREGION_X3Y6:CLOCKREGION_X3Y6
CLOCKREGION_X1Y6:CLOCKREGION_X1Y6          CLOCKREGION_X0Y5:CLOCKREGION_X3Y5
CLOCKREGION_X3Y4:CLOCKREGION_X3Y4          CLOCKREGION_X0Y4:CLOCKREGION_X1Y4
CLOCKREGION_X1Y3:CLOCKREGION_X3Y3          CLOCKREGION_X0Y0:CLOCKREGION_X3Y2}
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks static_pblock]
```

These constraints will place our four pblocks on different sections in actual FPGA (Figure 26). Everything that is not part of these four reconfigurable partitions is placed inside the defined static pblock section (gray color).

Additional set of constraints needs to be set to enable isolation between these pblocks. Those isolation constraints were set following the Isolation Design Flow described in ([REF \[3\]](#)). This flow defines proper setting for isolation modules. I Isolation constraints were defined in **im\_pblocks\_constraints.xdc** file:

```
##### Constraints for isolation of IMs inside RPs and Static Block isolation
set_property HD.ISOLATED true [get_cells hp130ba_i/static_partition]
set_property HD.ISOLATED true [get_cells hp130ba_i/rp_filter_0/im_wrapper]
set_property HD.ISOLATED true [get_cells hp130ba_i/rp_filter_1/im_wrapper]
set_property HD.ISOLATED true [get_cells hp130ba_i/rp_filter_2/im_wrapper]
set_property HD.ISOLATED true [get_cells hp130ba_i/rp_filter_3/im_wrapper]
```

```
##### Constraints for RMs inside Reconfigurable blocks
create_pblock im_0
```

```
add_cells_to_pblock [get_pblocks im_0] [get_cells -quiet [list hp130ba_i/rp_filter_0/im_wrapper]]
resize_pblock [get_pblocks im_0] -add {SLICE_X1Y362:SLICE_X27Y417}
resize_pblock [get_pblocks im_0] -add {BUFCE_LEAF_X8Y24:BUFCE_LEAF_X151Y27}
resize_pblock [get_pblocks im_0] -add {BUFCE_ROW_FSR_X0Y6:BUFCE_ROW_FSR_X38Y6}
resize_pblock [get_pblocks im_0] -add {DSP48E2_X0Y146:DSP48E2_X4Y165}
resize_pblock [get_pblocks im_0] -add {HARD_SYNC_X0Y12:HARD_SYNC_X7Y13}
resize_pblock [get_pblocks im_0] -add {RAMB18_X0Y146:RAMB18_X3Y165}
resize_pblock [get_pblocks im_0] -add {RAMB36_X0Y73:RAMB36_X3Y82}
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks im_0]
```

```
create_pblock im_1
```

```
add_cells_to_pblock [get_pblocks im_1] [get_cells -quiet [list hp130ba_i/rp_filter_1/im_wrapper]]
resize_pblock [get_pblocks im_1] -add {SLICE_X58Y362:SLICE_X75Y418}
resize_pblock [get_pblocks im_1] -add {BUFCE_LEAF_X312Y24:BUFCE_LEAF_X407Y27}
resize_pblock [get_pblocks im_1] -add {BUFCE_ROW_FSR_X77Y6:BUFCE_ROW_FSR_X102Y6}
resize_pblock [get_pblocks im_1] -add {DSP48E2_X12Y146:DSP48E2_X14Y165}
resize_pblock [get_pblocks im_1] -add {HARD_SYNC_X14Y12:HARD_SYNC_X19Y13}
resize_pblock [get_pblocks im_1] -add {RAMB18_X7Y146:RAMB18_X9Y165}
resize_pblock [get_pblocks im_1] -add {RAMB36_X7Y73:RAMB36_X9Y82}
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks im_1]
```

```
create_pblock im_2
```

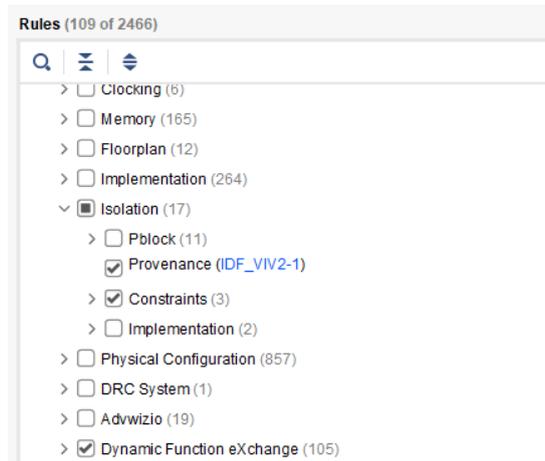
```
add_cells_to_pblock [get_pblocks im_2] [get_cells -quiet [list hp130ba_i/rp_filter_2/im_wrapper]]
resize_pblock [get_pblocks im_2] -add {SLICE_X1Y182:SLICE_X27Y236}
resize_pblock [get_pblocks im_2] -add {BUFCE_LEAF_X8Y12:BUFCE_LEAF_X151Y15}
resize_pblock [get_pblocks im_2] -add {BUFCE_ROW_FSR_X0Y3:BUFCE_ROW_FSR_X38Y3}
resize_pblock [get_pblocks im_2] -add {DSP48E2_X0Y74:DSP48E2_X4Y93}
resize_pblock [get_pblocks im_2] -add {HARD_SYNC_X0Y6:HARD_SYNC_X7Y7}
resize_pblock [get_pblocks im_2] -add {RAMB18_X0Y74:RAMB18_X3Y93}
resize_pblock [get_pblocks im_2] -add {RAMB36_X0Y37:RAMB36_X3Y46}
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks im_2]
```

```
create_pblock im_3
```

```
add_cells_to_pblock [get_pblocks im_3] [get_cells -quiet [list hp130ba_i/rp_filter_3/im_wrapper]]
resize_pblock [get_pblocks im_3] -add {SLICE_X58Y242:SLICE_X75Y297}
resize_pblock [get_pblocks im_3] -add {BUFCE_LEAF_X312Y16:BUFCE_LEAF_X407Y19}
resize_pblock [get_pblocks im_3] -add {BUFCE_ROW_FSR_X77Y4:BUFCE_ROW_FSR_X102Y4}
resize_pblock [get_pblocks im_3] -add {DSP48E2_X12Y98:DSP48E2_X14Y117}
resize_pblock [get_pblocks im_3] -add {HARD_SYNC_X14Y8:HARD_SYNC_X19Y9}
resize_pblock [get_pblocks im_3] -add {RAMB18_X7Y98:RAMB18_X9Y117}
resize_pblock [get_pblocks im_3] -add {RAMB36_X7Y49:RAMB36_X9Y58}
set_property SNAPPING_MODE FINE_GRAINED [get_pblocks im_3]
```

```
set_property HD.ISOLATED_EXEMPT true [get_cells
hp130ba_i/static_partition/display_hier/vid_phy_controller_0/inst/gt_usrclk_source_inst/tx_mmcm.GT
0_TX_MMCM_CLKOUT1_OBUFTDS_INST]
set_property HD.ISOLATED_EXEMPT true [get_cells -hierarchical -filter {PRIMITIVE_TYPE ==
CLOCK.BUFFER.BUFGCE}]
set_property HD.ISOLATED_EXEMPT true [get_cells -hierarchical -filter {PRIMITIVE_TYPE ==
CLOCK.BUFFER.BUFG_GT}]
```

Before continuing design runs and implementation, user can use Vivado rule checks just to confirm that everything regarding floorplan for DFX and IDF is set correctly. At opened synthesized run, just run the DRC check for the DFX and Isolation flow. For Isolation flow, select Provenance and Constraints checks.



**Figure 25: DFX IDF after synthesis DRC Check**

If everything is ok with design, some advisory may be presented, but no violations should be noticed in result file:

In this reference design, IDF constraints (HD.ISOLATED) main implementation is set directly through Vivado GUI. For other two child runs, those constrains are added as post synthesis script commands.

```
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_0_greenccm.tcl -of [get_fileset
utils_1] ] [get_runs greenccm_inst_0_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_1_greenccm.tcl -of [get_fileset
utils_1] ] [get_runs greenccm_inst_1_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_2_greenccm.tcl -of [get_fileset
utils_1] ] [get_runs greenccm_inst_2_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_3_greenccm.tcl -of [get_fileset
utils_1] ] [get_runs greenccm_inst_3_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_0_sobel.tcl -of [get_fileset
utils_1] ] [get_runs sobel_inst_0_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_1_sobel.tcl -of [get_fileset
utils_1] ] [get_runs sobel_inst_1_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_2_sobel.tcl -of [get_fileset
utils_1] ] [get_runs sobel_inst_2_synth_1]
set_property STEPS.SYNTH_DESIGN.TCL.POST [ get_files config_3_sobel.tcl -of [get_fileset
utils_1] ] [get_runs sobel_inst_3_synth_1]
```

For example, if we want to set that isolation property on GREEN CCM filter at reconfigurable partition 0, we can just set that property directly to design checkpoint.

```
set_property HD.ISOLATED true [get_cells im_wrapper] -quiet
```

```
write_checkpoint -force -noxdef greenccm_inst_0.dcp
```

For easy reference design build, these steps are already incorporated in **project.tcl** file, where all isolation property checkpoints are properly set and configured.

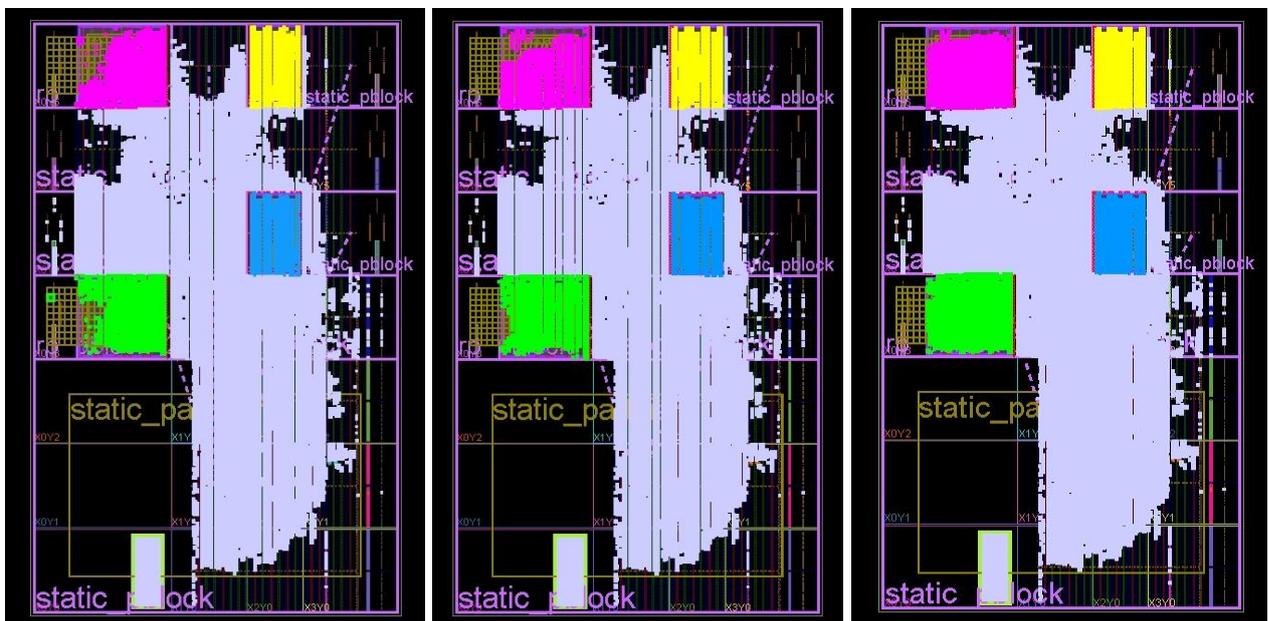
Final step before starting implementation runs is setting the pre opt design scripts for child runs. This is also already set in **project.tcl** file.

```
set_property STEPS.OPT_DESIGN.TCL.PRE [ get_files child_0_impl_1_pre_opt.tcl -of [get_fileset
utils_1] ] [get_runs child_0_impl_1]
set_property STEPS.OPT_DESIGN.TCL.PRE [ get_files child_1_impl_1_pre_opt.tcl -of [get_fileset
utils_1] ] [get_runs child_1_impl_1]
```

Those tcl files are just applying exactly the same isolation pblocks constraints for the child runs:

```
read_xdc ../../data/im_pblocks_constraints.xdc
```

As we noted before, user doesn't have to worry about these commands and scripts in this reference design, because they are already set and prepared at design build from scratch by running `create_project.bat` command or sourcing the `project.tcl` script. After running complete implementation runs, the result implemented design is show at Figure 26.

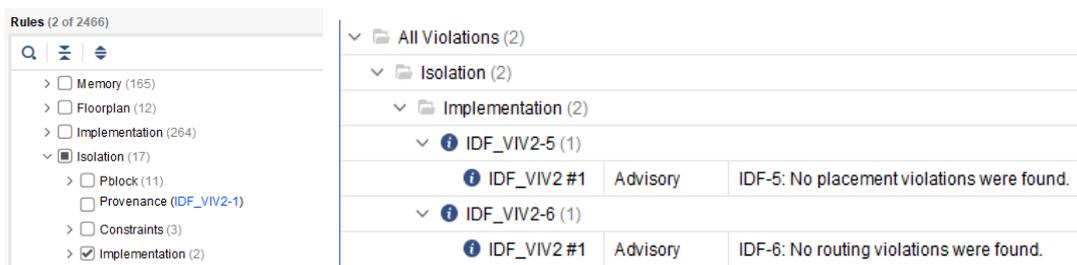


**Figure 26: Implementation of three config runs (No Filter, CCM Green Filter, Sobel Filter)**

As shown in Figure 26, utilization inside configurable partitions varies depending on the filter selected. It's important that designer take this level of congestion into consideration while creating a floorplan of design. The bigger the block, the more resources are available for the filter, and the Vivado tool will have easier task and complete implementation and routing sooner.

Resulting twelve partial bitstreams will be generated inside the same Vivado project runs folder. For the default implementation, those bitstream files are stored in the *impl\_1* directory, for the second implementation, files are located in the *child\_0\_impl\_1* directory, and for the last implementation run files are located inside the *child\_1\_impl\_1* directory.

Just a side note, for user to check that everything is correctly implemented following the Isolation design flow, opening implemented design and running DRC Isolation (Implementation) check should result without any violations.



**Figure 27: Implemented DRC Isolation check**

This reference design generates a total of 12 partial bitstreams for use in DFX flow. Those partial bitstreams are named automatically in Vivado Dynamic Function eXchange Wizard. For every reconfigurable partition, there are three available partials bitstreams. The number of available partial bitstreams for reconfigurable partitions depends on how many functions (in our case filters) are available. Resulting partial bitstreams for each reconfigurable partition are shown below:

**RP0:**

1. hp130ba\_i\_rp\_filter\_0\_nofilter\_inst\_0\_partial.bin
2. hp130ba\_i\_rp\_filter\_0\_greenccm\_inst\_0\_partial.bin
3. hp130ba\_i\_rp\_filter\_0\_sobel\_inst\_0\_partial.bin

**RP1:**

1. hp130ba\_i\_rp\_filter\_1\_nofilter\_inst\_1\_partial.bin
2. hp130ba\_i\_rp\_filter\_1\_greenccm\_inst\_1\_partial.bin
3. hp130ba\_i\_rp\_filter\_1\_sobel\_inst\_1\_partial.bin

**RP2:**

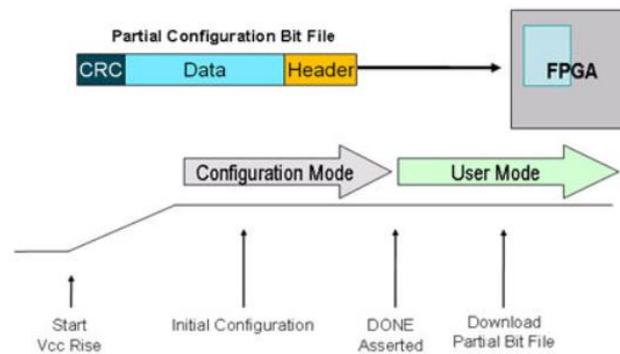
1. hp130ba\_i\_rp\_filter\_2\_nofilter\_inst\_2\_partial.bin
2. hp130ba\_i\_rp\_filter\_2\_greenccm\_inst\_2\_partial.bin
3. hp130ba\_i\_rp\_filter\_2\_sobel\_inst\_2\_partial.bin

**RP3:**

1. hp130ba\_i\_rp\_filter\_3\_nofilter\_inst\_3\_partial.bin
2. hp130ba\_i\_rp\_filter\_3\_greenccm\_inst\_3\_partial.bin
3. hp130ba\_i\_rp\_filter\_3\_sobel\_inst\_3\_partial.bin

The size of a partial bitstream is directly proportional to the size of the region it is reconfiguring. For example, if the RP is composed of 20% of the device resources, the partial bitstream is roughly 20% the size of the full design bitstream. Partial bitstreams are fully self-contained, so they are delivered to

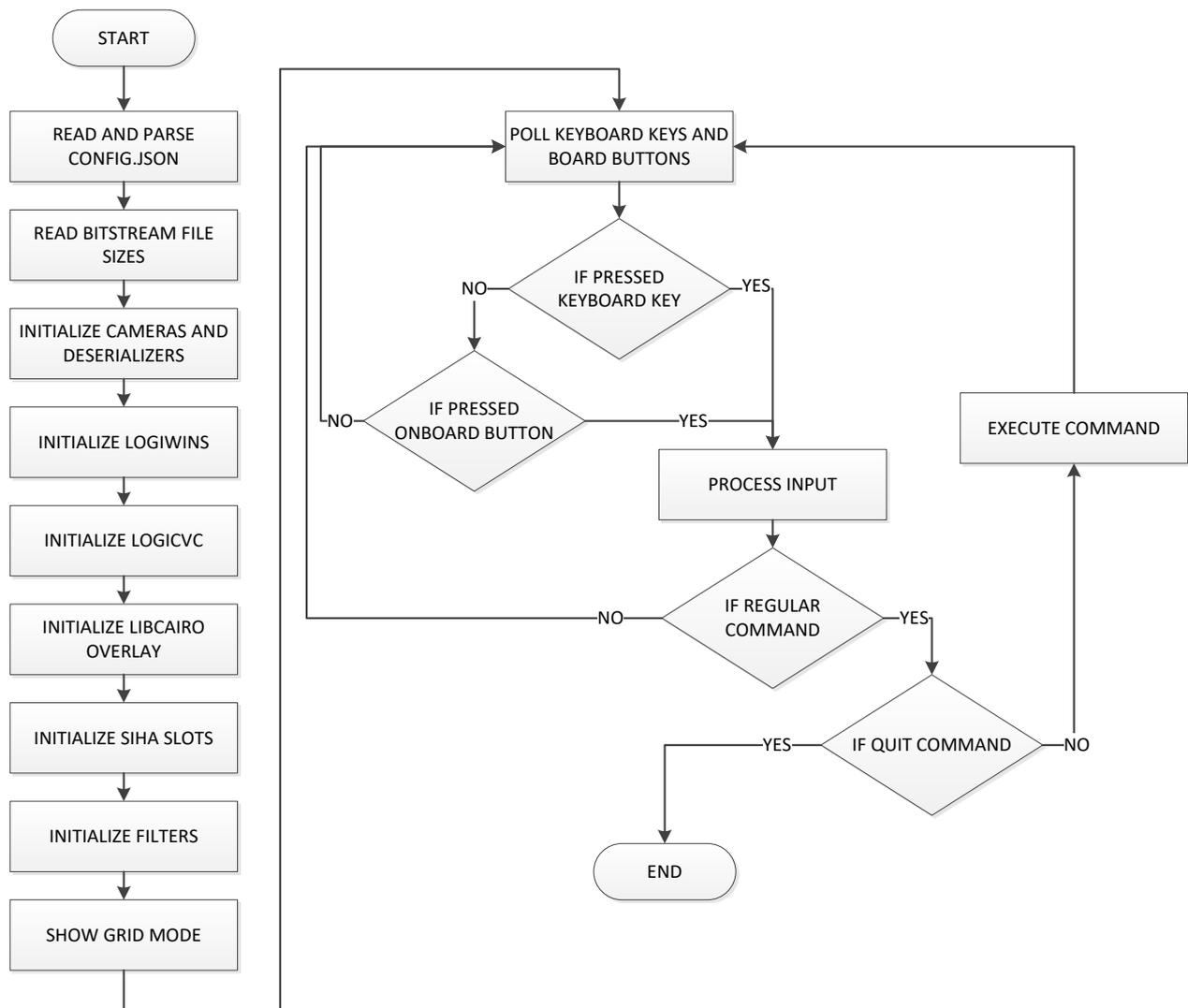
an appropriate configuration port. All addressing, header, and footer details are contained within these bitstreams, just as they would be for full configuration bitstreams. You deliver partial bitstreams to the FPGA through any external non-master configuration mode, such as JTAG, Slave Serial, or Slave SelectMap. Internal configuration access includes the ICAP (all devices), PCAP (Zynq-7000 SoC devices), and MCAP (UltraScale and UltraScale+ devices through PCIe). A partially reconfigured FPGA is in user mode while the partial file is loaded. This allows the portion of the FPGA logic not being reconfigured to continue functioning while the reconfigurable partition is modified. More info on this can be found here ([REF \[1\]](#)).



**Figure 28: Configuration with partial bitstream**

## 5.7 Software Description

### 5.7.1 Demo application



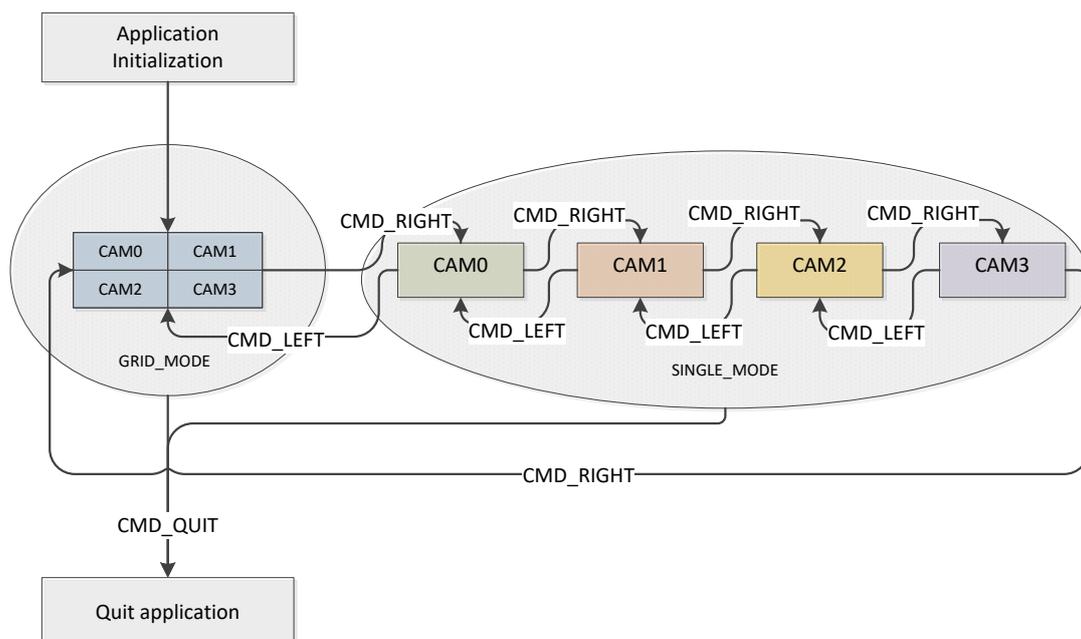
**Figure 29: Demo application software block diagram**

DFX Demo application consists of the initialization part and the main loop part as shown in the Figure 29.

Following functions are used for application initialization:

- **bool read\_config\_file(const char \*name, Config &conf)**
  - Read and parse `config.json` file using rapidjson library
- **bool read\_bitstream\_sizes()**
  - Read bitstream file sizes using `<sys/stat.h> stat()` function
- **int plf\_configuration()**
  - Initialize cameras and deserializers using plf+ library
- **void logiWIN\_init\_start(unsigned idx)**
  - Initialize logiWIN IP indexed with `idx` and scale input from cameras
- **pVideoOutInstT vout\_init(char \*devpath)**
  - Initialize logiCVC IP using logiVIOF\_DRM library and position and scale CVC layers to form GRID\_MODE, and get CVC layer addresses
- **Overlay (struct bo &buffObj, unsigned width, unsigned height)**
  - Class Overlay constructor is used to initialize overlay using libcairo
- **int enable\_SIHA\_slot(unsigned slot)**
  - Initialize SIHA slot indexed with `slot`
- **void enable\_filter(unsigned slot, unsigned outAddr)**
  - Initialize filter indexed with `slot`

After required initializations are done, the application starts in the GRID\_MODE by default. The main loop polls keyboard keys via UART serial console and on board buttons (SW14-18), checks if input corresponded to regular commands and executes regular command issued.



**Figure 30: Demo application modes description**

Demo application consists of two modes as shown in the Figure 30:

1. GRID\_MODE - shows 4 cameras in 2x2 grid
2. SINGLE\_MODE - shows 1 camera in full screen

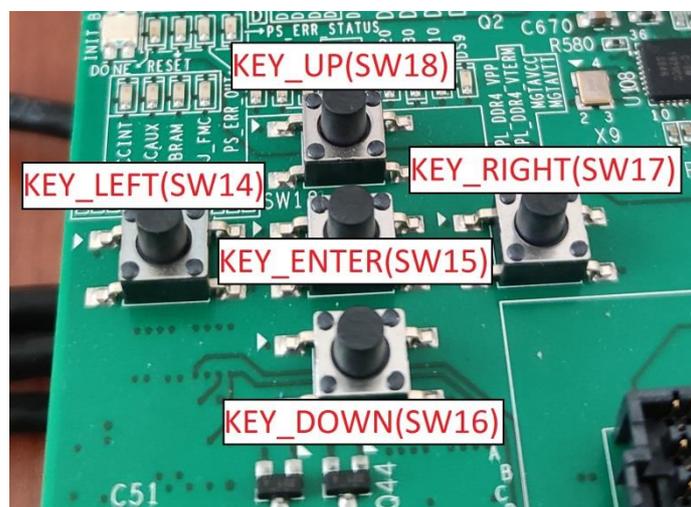
In total 5 different screens can be displayed:

1. GRID\_MODE
2. SINGLE\_MODE - CAM0
3. SINGLE\_MODE - CAM1
4. SINGLE\_MODE - CAM2
5. SINGLE\_MODE - CAM3

The Figure 30 shows how various commands trigger transitions from GRID\_MODE to SINGLE\_MODE and vice versa. User can circle through modes using commands `CMD_RIGHT` and `CMD_LEFT`. Commands `CMD_KYBD_TOGGLE` and `CMD_BTN_TOGGLE` are used to initiate filter toggling through the DFX FPGA Manager partial reconfiguration. The `CMD_QUIT` command is used to exit the main loop, disable SIHA slots and filters, release the logiWIN and the logiCVC and quit the application. See how the commands are mapped to user inputs in the Table 5.3 and the Figure 31.

**Table 5.3: Application input and corresponding commands**

Command name	Keyboard	Onboard button	Command description
<code>CMD_RIGHT</code>	'd'	KEY_RIGHT	circular next mode (see Figure 36)
<code>CMD_LEFT</code>	'a'	KEY_LEFT	circular previous mode (see Figure 36)
<code>CMD_KYBD_TOGGLE</code>	't'		toggle filter using keyboard
<code>CMD_BTN_TOGGLE</code>		KEY_ENTER	toggle filter using on board button
<code>CMD_IDLE</code>			do nothing
<code>CMD_QUIT</code>	'q'		exit main loop and quit demo application
<code>CMD_LOAD_CORRUPTED</code>	'l'	KEY_UP	load corrupted bitstream on CAM0 (PR0) while no_filter active
<code>CMD_START_SEM</code>	'm'		start SEM



**Figure 31: ZCU102 on board buttons (SW14-18) used in Demo application**

This demo features one extra bitstream which is intentionally corrupted and it could be displayed to demonstrate fixing FPGA by DFX partial reconfiguration. It is initiated by:

- pressing 'l' on keyboard
- pressing KEY\_UP button

Only condition is that Single mode CAM0 no\_filter bitstream is active while corresponding key is pressed. Main loop is stopped while corrupted bitstream is displayed and user is prompted to press corresponding key to initiate DFX partial reconfiguration of programmable region that corresponds to CAM0. After finishing, main loop resumes to default demo control commands.

## 5.7.2 DFX functionality description

Filter toggling implemented in the demo application demonstrates the DFX functionality using FPGA manager partial reconfiguration.

Function `void toggle_filter(unsigned PRIdx)` is used to toggle filter on programmable region index `PRIdx` and it consists of the following algorithm:

- Disable filter
- Disable SIHA slot
- Load partial bitstream through FPGA manager
- Enable SIHA slot
- Enable filter

Function `void dfx_load_bitstream(unsigned bitIdx, unsigned PRIdx)` is used to load bitstream through FPGA manager driver using partial reconfiguration.

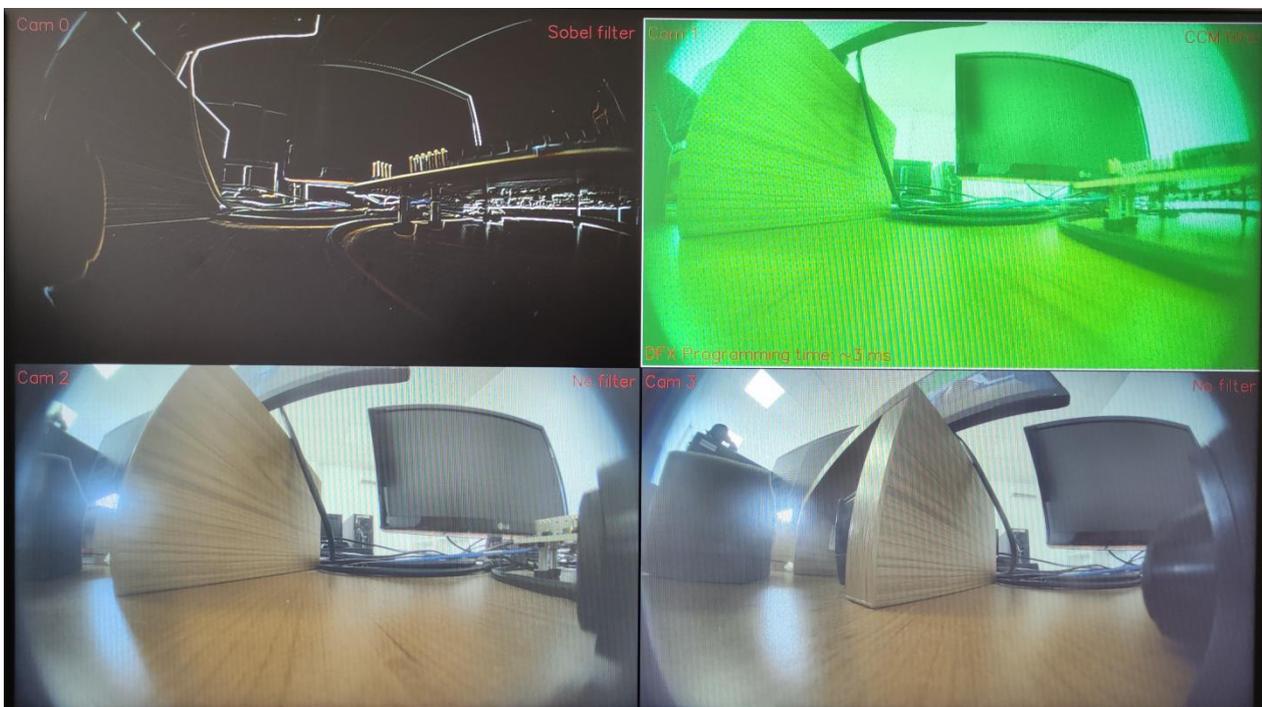
Commands `CMD_KYBD_TOGGLE` and `CMD_BTN_TOGGLE` are used in `SINGLE_MODE` mode of the application to initiate filter toggling by cycling through available filters indexed in the following manner: 0 - No filter, 1 – Green CCM filter and 2 - Sobel filter.



**Figure 32: SINGLE\_MODE camera 0 (programmable region 0) Sobel filter example**

In the GRID\_MODE there are two different ways to initiate bitstream loading through FPGA manager driver using partial reconfiguration, based on the user input received:

1. Using command `CMD_KYBD_TOGGLE`, application prompts user to input integer number (0 – 3) for desired camera number (programmable region) where filter should toggle.
2. Using command `CMD_BTN_TOGGLE`, application waits for user to press the on board button:  
`KEY_UP` (SW18) - toggle filter on camera 0  
`KEY_RIGHT` (SW17) - toggle filter on camera 1  
`KEY_DOWN` (SW16) - toggle filter on camera 2  
`KEY_LEFT` (SW14) - toggle filter on camera 3



**Figure 33: GRID\_MODE camera 0 (programmable region 0) Sobel filter example**

### 5.7.3 Error injection using SEM

SEM registers are initialized by pressing 'm' (`CMD_START_SEM`) on keyboard. Second UART (SEM UART) is enabled and user is prompted to reset SEM by typing `R 04` command in SEM UART. This has to be done every time SEM is started in order to SEM function properly.

SEM UART commands that are used in demo are described in section **6.6 SEM UART Example Commands**. After resetting SEM to inject and detect errors, user has to follow instructions displayed on main application UART. The instructions are the following:

1. Put SEM in Idle mode
2. Inject errors
3. Put SEM in Observation Mode

If 2-bit error is injected to FPGA, SEM detects uncorrectable error (`FC 60` is displayed on SEM UART) and uncorrectable GPIO status is set. This GPIO is polled in main application loop and if detected it initiates disabling SEM and fixing errors by partially DFX reconfiguring all 4 programmable regions (PRs). After it is finished, application resumes to default demo control commands and main loop.

### 5.7.4 Configuration file description

Demo application configuration file is `config.json` and it is loaded into the application from the main thread. It contains all file and directory paths necessary to run the application. Description of file contents is showed in Table 6.x.

**Table 5.4: Contents of the application configuration file `config.json`**

members in .json file	member description
<code>plfConfPath</code>	platform configuration file path
<code>filterBitPaths</code>	filter bitstream file paths used by FPGA manager



To run the application all file paths and names have to be correctly stated in the `config.json` or application will terminate.

If user needed to change the path to the `config.json` to some other path on the SD card or in the file system, new path has to be specified as an application argument. The default `config.json` path is `/mnt/sd-mmcbk0p1/config.json`. For example, if the file path is to be changed to the `/mnt/sd-mmcbk0p1/configs/config.json`, the application has to be run as follows:

```
./app_zcu102_4cam_linux.elf /mnt/sd-mmcbk0p1/configs/config.json
```



If `config.json` file path was changed, make sure to add new path as application argument also in the `init.sh` startup script. This will start application with correct `config.json` file path after every board reboot.

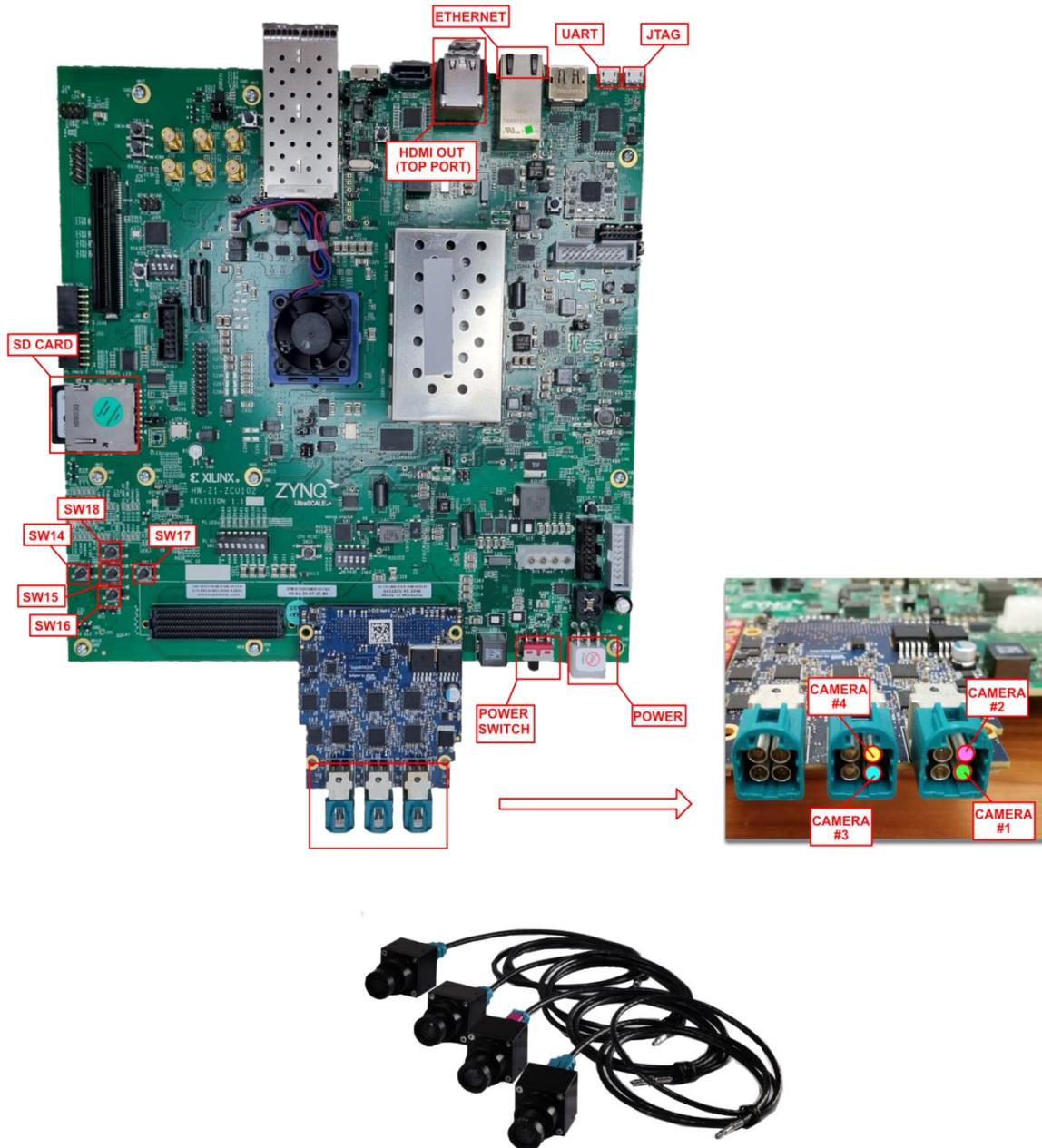
### 5.7.5 Input resolution and the frame rate

The logiREF-DFX-IDF reference design uses the 1928x1208@30 input video resolution.

### 5.7.6 Output resolution and the frame rate

The logiREF-DFX-IDF reference design uses 1928x1208@60 output resolution.

## 6 QUICK START



**Figure 34: logiREF-DFX-IDF HW Setup and the GMSL2 card connection**

## 6.1 Run the Precompiled Linux Demo Examples

To enable rapid testing of the hardware setup, Xylon provides application demo binaries in the *binaries* folder of the deliverables.

To run the DFX Demo application, copy the content of the *binaries/bin* folder to the root folder of the formatted SD card. The SD card should be formatted as FAT32.

Optionally, you can use a serial terminal program (baud rate 115200 8N1) and the USB UART connection to the ZCU102 board to monitor the system's operation during the boot and application execution.

For full explanation of the ZCU102's features and settings, please check the documentation Xilinx [XTP426](#).

## 6.2 Demo controls

Start the DFX Demo application and the display will show the video output from 4 attached cameras in 2x2 grid. Press following keys to control application:

- 'd' / **KEY\_RIGHT(SW17)** = next mode
- 'a' / **KEY\_LEFT(SW14)** = previous mode
- 'q' = quit application

In SINGLE\_MODE press 't' / **KEY\_ENTER(SW15)** to toggle filter on current camera

In GRID\_MODE press:

- 't' = application prompts user to input number from 0 to 3 to toggle filter on that camera number
- **KEY\_ENTER(SW15)** = application prompts user to input one of 4 other on board keys that correspond to camera numbers in the following manner:
  - **KEY\_UP(SW18)** - toggle filter on camera 0
  - **KEY\_RIGHT(SW17)** - toggle filter on camera 1
  - **KEY\_DOWN(SW16)** - toggle filter on camera 2
  - **KEY\_LEFT(SW14)** - toggle filter on camera 3

## 6.3 Change the Delivered Software

### 6.3.1 Xilinx Development Software

The logiREF-DFX-IDF reference design and Xylon logicBRICKS IP cores are fully compatible with Xilinx development tools – Vivado Design Suite 2021.1. Future design releases shall be synchronized with the newest Xilinx development tools.

Licensed users of Xilinx tools can use their existing software installation for the logiREF-DFX-IDF evaluation and modifications.

## 6.3.2 Set Up Linux System Software Development Tools

Set of ARM GNU tools are required to build the Linux software and applications. The complete tool chain for the Zynq UltraScale+ MPSoC can be obtained from the Xilinx ARM GNU Tools wiki page: <http://wiki.xilinx.com/Install+Xilinx+Tools>. Access to tools requires a valid, registered Xilinx user login name and password.

## 6.3.3 Set Up git Tools

Git is a free Source Code Management (SCM) tool for managing distributed version control and collaborative development of software. It provides the developer a local copy of the entire development project files and the very latest changes to the software.

Visit for example <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841694/Using+Git> to get some basic instructions how to use git.

To get the latest version of Xylon logicBRICKS software drivers for Linux operating system, please visit Xylon's git: <https://github.com/logicbricks>.

## 6.3.4 Setting up and building the Petalinux project

Development for the Linux target is tightly coupled with Linux kernel configuration, understanding of boot mechanisms, configuration of device tree handling hardware components dependency, kernel and user space driver development and other Linux specific topics outside of the scope of this manual. As Xilinx environment uses the Petalinux to make the Linux based development easier, quick steps required to build the Petalinux project from the hardware specification (XSA file) and get results needed are given here.

The file *logiREF-DFX-IDF\_YYMMDD/software/Linux/petalinux-setup\_2021.1.txt* file delivered with the product deliverables describes the same steps. If there were differences in procedure described, the text file mentioned should take the precedence.



Some application dependencies are built in Petalinux project (libdrm, libcairo and libplf+). If any changes are needed in this components, make sure to rebuild the Petalinux project as described in *logiREF-DFX-IDF\_YYMMDD\software\Linux\petalinux-setup\_2021.1.txt*. Even if petalinux project rebuild was not necessary due to HW or kernel changes, the SYSROOT file structure is going to be needed for application development.

The Petalinux project should be created and built even before attempt of change the application software because the Linux cross development environment depends on the SYSROOT and other file file structures holding headers and libraries to build against.

### 6.3.4.1 Tools needed

PetaLinux 2021.1 Installer is needed, it can only be used within the Linux environment and it can be freely downloaded from the Xilinx support page:

<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2021-1.html>

### 6.3.4.2 Installation on Ubuntu 16.04/18.04/20.04

Follow the instructions for Petalinux 2021.1 installation found in the User's Guide for relevant release, 2021.1 in our case: UG1144 (v2021.1) June 16, 2021 Petalinux Tools Documentation Reference Guide

### 6.3.4.3 Setup and basic configuration

There are important file paths to be used and referenced at some point, so get familiar with those locations in your file system.

<PATH\_TO\_PETALINUX> -> path to Petalinux 2021.1 installation directory

In the following examples the default installation is in the user home directory:

```
~/tools/petalinux
```

<PLNX\_PROJECT\_NAME> -> path to Petalinux project

In the following examples the default installation is in the delivery directory as extracted by the jar installation process:

```
~/xylon/logiREF-DFX-IDF_211208/software/Linux/zcu102_4cam_idf_dfx/
```

The Petalinux project directory structure is best copied from delivery folder to some convenient location on the Linux build machine. That way the configuration changes done by the user if needed do not overwrite the default delivered and working Petalinux project setup.

```
cp -r ~/xylon/logiREF-DFX-IDF_211208/software/Linux/zcu102_4cam_idf_dfx/ ~/dfxidf_build_test/
```

If we now took a look at the `~/dfxidf_build_test/zcu102_4cam_idf_dfx` directory now we should find the petalinux project specification with all necessary meta information to be used by Petalinux and Yocto.

### 6.3.4.4 Petalinux configuration and building

Finally we should source the petalinux environment script and start building the project.

```
source ~/tools/petalinux/settings.sh
```

The result should be something like this (warnings are to be considered, but in this case did not cause issues):

```
PetaLinux environment set to '/home/dcika/tools/petalinux'  
WARNING: /bin/sh is not bash!  
bash is PetaLinux recommended shell. Please set your default shell to bash.  
WARNING: This is not a supported OS  
INFO: Checking free disk space  
INFO: Checking installed tools  
INFO: Checking installed development libraries  
INFO: Checking network and other services
```

Now we can do the petalinux main configuration. It is mandatory for the first build, or after any hardware design changes impacting the linux environment.

The command for this is picking the XSA files from the location specified:

```
petalinux-config --get-hw-description=<PATH-TO-XSA>
```

<PATH-TO-XSA> location depends on the software delivery structure created after executing the delivery jar script. This is the location of the XSA file, and in this example command run it is the following:

```
~/xylon/logiREF-DFX-IDF_211208/vivado/vivado_dfx_idf_sem/fpga/hp130ba_wrapper.xsa
```

So the petalinux-config command ran is the following (please note that actual command does not brake the XSA file path argument into the next line):

```
petalinux-config --get-hw-description ~/xylon/logiREF-DFX-  
IDF_211208/vivado/vivado_dfx_idf_sem/fpga/hp130ba_wrapper.xsa
```

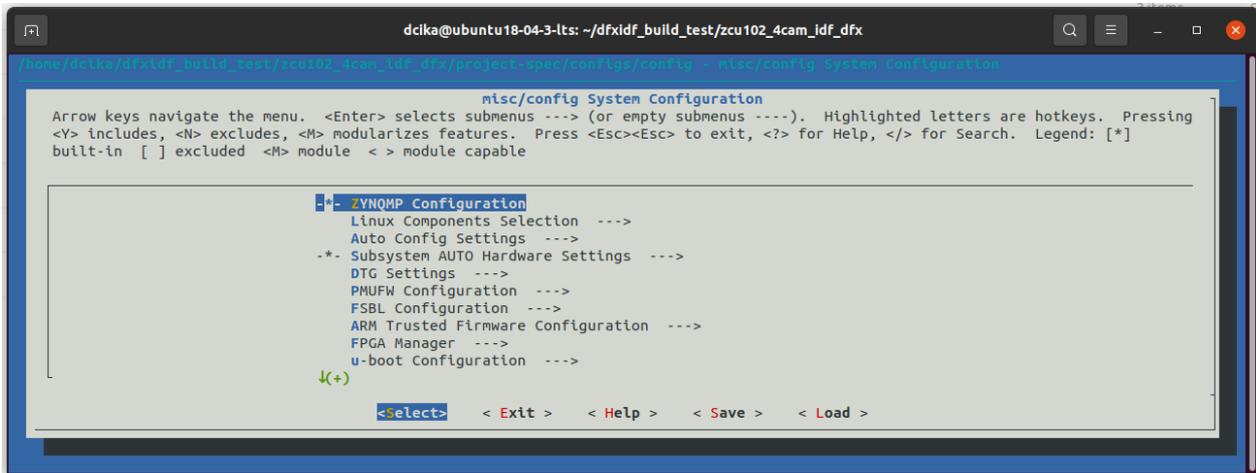
The result should be something like this, please note how the Petalinux has renamed the XSA file into something more generic:

```
[INFO] Sourcing buildtools  
INFO: Getting hardware description...  
INFO: Renaming hp130ba_wrapper.xsa to system.xsa  
[INFO] Generating Kconfig for project  
[INFO] Menuconfig project
```

```
*** End of the configuration.  
*** Execute 'make' to start the build or try 'make help'.
```

```
[INFO] Sourcing buildtools extended  
[INFO] Extracting yocto SDK to components/yocto. This may take time!  
[INFO] Sourcing build environment  
[INFO] Generating kconfig for Rootfs  
[INFO] Silentconfig rootfs  
[INFO] Generating plnxtool conf  
[INFO] Adding user layers  
[INFO] Generating workspace directory
```

The screen shown in the Figure 35 is the standard Linux Kconfig screen normally seen with buildroot and other Linux configurations. This is the place to make any changes to the kernel needed, simply by going through menus and selecting options. In our example project no changes are really needed, but we could select some less critical changes just to make sure the kernel was built as specified. For example, the petalinux-config command could be issued again without any arguments to do some more changes. On the main screen, use down arrow to select the Firmware Version Configuration and enter different host name and product name for example, save the configuration and exit.



**Figure 35: petalinux-config commands bring up the standard Kconfig screen**

The petalinux-config command can be rerun also without XSA file as an argument, and it will pick up from the configuration done before, because the system.xsa file is copied and preserved for any further configurations. If the HW was changed, the steps above should be done again.

In any case result should be something like this, please note that we do not actually run the make command after configuration, petalinux-build command following will take care of everything:

```
[INFO] Sourcing buildtools
[INFO] Menuconfig project
configuration written to /home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/project-spec/configs/config
```

```
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
```

```
[INFO] Sourcing buildtools extended
[INFO] Sourcing build environment
[INFO] Generating kconfig for Rootfs
[INFO] Silentconfig rootfs
[INFO] Generating plnxtool conf
[INFO] Generating workspace directory
[INFO] Successfully configured project
```

Finally we can run the build command to build the kernel and other components selected (ramdisk (also fsbl, uboot, pmu...))

### *petalinux-build*

The successful build should result in build and images folders created and populated with build results.

```
Initialising tasks: 100% |#####| Time: 0:00:04
Checking sstate mirror object availability: 100% |#####| Time: 0:00:02
Sstate summary: Wanted 1478 Found 0 Missed 1478 Current 0 (0% match, 0% complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 4416 tasks of which 5 didn't need to be rerun and all succeeded.
INFO: copy to TFTP-boot directory is not enabled !!
[INFO] Successfully built project
dcika@ubuntu18-04-3-lts:~/dfxidf_build_test/zcu102_4cam_idf_dfx$ ls
build components config.project images project-spec
dcika@ubuntu18-04-3-lts:~/dfxidf_build_test/zcu102_4cam_idf_dfx$
```

### 6.3.4.5 Boot and Linux image files preparation

All necessary files are already present after successful petalinux-build command, now it is time to prepare and package boot, fsbl, and other image files necessary by running the petalinux-package command with following arguments:

```
petalinux-package --boot --format BIN --fsbl images/linux/zynqmp_fsbl.elf --u-boot images/linux/u-boot.elf --pmufw images/linux/pmufw.elf --fpga images/linux/system.bit --force
```

The result should be something like the following, please note which files has been generated by this step and where there were stored.

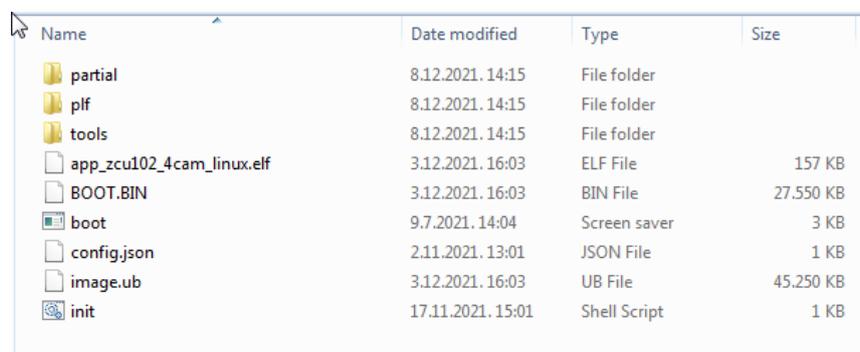
```
[INFO] Sourcing buildtools
INFO: Getting system flash information...
INFO: File in BOOT BIN: "/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/pmufw.elf"
INFO: File in BOOT BIN: "/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/system.bit"
INFO: File in BOOT BIN: "/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/bl31.elf"
INFO: File in BOOT BIN: "/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/system.dtb"
INFO: File in BOOT BIN: "/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...
```

```
***** Xilinx Bootgen v2021.1
**** Build date : May 28 2021-21:36:22
** Copyright 1986-2021 Xilinx, Inc. All Rights Reserved.
```

[INFO] : Bootimage generated successfully

INFO: Binary is ready.

There are more files generated and stored in the images/linux folder, select what is necessary depending on the boot method. For booting from the SD card on this project, only few files are necessary and copied to the root of the FAT32 formatted SD card, specifically the BOOT.BIN, boot.scr, and image.ub from those created in this step. The rest of the files shown in the Figure 36 are partial bitstreams, camera initialization and configuration files, some tools, the application executable, the application configuration file and the init startup script.



Name	Date modified	Type	Size
partial	8.12.2021. 14:15	File folder	
plf	8.12.2021. 14:15	File folder	
tools	8.12.2021. 14:15	File folder	
app_zcu102_4cam_linux.elf	3.12.2021. 16:03	ELF File	157 KB
BOOT.BIN	3.12.2021. 16:03	BIN File	27.550 KB
boot	9.7.2021. 14:04	Screen saver	3 KB
config.json	2.11.2021. 13:01	JSON File	1 KB
image.ub	3.12.2021. 16:03	UB File	45.250 KB
init	17.11.2021. 15:01	Shell Script	1 KB

**Figure 36: Content of the FAT32 formatted SD card prepared for the demo application**

### 6.3.4.6 Application development environment preparation

In order to be able to do the embedded cross development, some files have to be prepared and present on the host development computer for each target HW architecture. Those files will be later accessed by application development environment through the SYSROOT variable set to the specific folder. Building the Yocto SDK will result in building the sysroot file structure, and it requires a single argument to the petalinux-build command:

```
petalinux-build --sdk
```

After considerable time (at least for the first build) the sdk build process should end with success. If any error was logged, try rerun the command. Petalinux is able to continue from where it was interrupted and identify missing tasks that did not complete for some reason. The end result (after getting some error and running again) may look something like this:

NOTE: Tasks Summary: Attempted 3831 tasks of which 2855 didn't need to be rerun and 1 failed.

Summary: 1 task failed:

```
/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/components/yocto/layers/core/meta/recipes-devtools/gdb/gdb-cross-canadian_9.2.bb:do_install
```

Summary: There were 2 ERROR messages shown, returning a non-zero exit code.

ERROR: Failed to build project. Check the /home/dcika/dfxidf\_build\_test/zcu102\_4cam\_idf\_dfx/build/build.log file for more details...

```
dcika@ubuntu18-04-3-lts:~/dfxidf_build_test/zcu102_4cam_idf_dfx$ gedit
```

```
/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/build/build.log
```

```
dcika@ubuntu18-04-3-lts:~/dfxidf_build_test/zcu102_4cam_idf_dfx$ petalinux-build --sdk
```

```
[INFO] Sourcing buildtools
```

```
[INFO] Building project
```

```
[INFO] Sourcing buildtools extended
```

```
[INFO] Sourcing build environment
```

```
[INFO] Generating workspace directory
```

```
INFO: bitbake petalinux-image-minimal -c do_populate_sdk
```

```
NOTE: Started PRServer with DBfile: /home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/build/cache/prserv.sqlite3, IP: 127.0.0.1, PORT: 39747, PID: 2991346
```

```
Loading cache: 100%
```

```
#####
```

```
#####| Time: 0:00:01
```

```
Loaded 5100 entries from dependency cache.
```

```
Parsing recipes: 100%
```

```
#####
```

```
#####| Time: 0:00:01
```

```
Parsing of 3473 .bb files complete (3465 cached, 8 parsed). 5108 targets, 224 skipped, 0 masked, 0 errors.
```

```
NOTE: Resolving any missing task queue dependencies
```

```
Initialising tasks: 100%
```

```
#####
```

```
#####| Time: 0:00:02
```

```
Sstate summary: Wanted 166 Found 151 Missed 15 Current 1134 (90% match, 98% complete)
```

```
NOTE: Executing Tasks
```

```
NOTE: Tasks Summary: Attempted 3850 tasks of which 3830 didn't need to be rerun and all succeeded.
```

```
[INFO] Copying SDK Installer...
```

```
[INFO] Successfully built project
```

If we checked the /build/tmp/deploy directory now we would find the folder generated there containing the script for building the sdk. This script can be executed directly as it contains all necessary files for sdk and sysroot generation, but the Petalinux way of preparing packages is again by using the package command, only with different argument this time compared to earlier boot files preparation:

## *petalinux-package --sysroot*

```
dcika@ubuntu18-04-3-lts:~/dfxidf_build_test/zcu102_4cam_idf_dfx$ petalinux-package --sysroot
PetaLinux SDK installer version 2021.1
=====
You are about to install the SDK to "/home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/sdk". Proceed [Y/n]? Y
Extracting SDK.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
$ . /home/dcika/dfxidf_build_test/zcu102_4cam_idf_dfx/images/linux/sdk/environment-setup-cortexa72-cortexa53-xilinx-linux
dcika@ubuntu18-04-3-lts:~/dfxidf_build_test/zcu102_4cam_idf_dfx$
```

It can be seen from the picture above where the sdk installation folder is. There may be different architectures supported so the specific sysroot location in our case is

<PLNX\_PROJECT\_NAME>/images/linux/sdk/sysroots/cortexa72-cortexa53-xilinx-linux

<PLNX\_PROJECT\_NAME> is /home/dcika/dfxidf\_build\_test/zcu102\_4cam\_idf\_dfx in our example case the, so the complete path to the sysroot folder location when asked in our case is the

/home/dcika/dfxidf\_build\_test/zcu102\_4cam\_idf\_dfx/images/linux/sdk/sysroots/cortexa72-cortexa53-xilinx-linux/

The usual Linux file structure should be found there:

```
dcika@ubuntu18-04-3-lts:~/dfxidf_build_test/zcu102_4cam_idf_dfx$ ls -all images/linux/sdk/sysroots/cortexa72-cortexa53-xilinx-linux/
total 68
drwxrwxr-x 17 dcika dcika 4096 Dec  9 12:11 .
drwxr-xr-x  4 dcika dcika 4096 Dec  9 12:11 ..
drwxr-xr-x  3 dcika dcika 4096 Dec  9 12:11 bin
drwxr-xr-x  2 dcika dcika 4096 Dec  9 12:11 boot
drwxr-xr-x  2 dcika dcika 4096 Dec  9 12:11 dev
drwxr-xr-x 33 dcika dcika 4096 Dec  9 12:11 etc
drwxr-xr-x  3 dcika dcika 4096 Dec  9 12:11 home
drwxr-xr-x  8 dcika dcika 4096 Dec  9 12:11 lib
drwxr-xr-x  2 dcika dcika 4096 Dec  9 12:11 media
drwxr-xr-x  2 dcika dcika 4096 Dec  9 12:11 mnt
dr-xr-xr-x  2 dcika dcika 4096 Dec  9 12:11 proc
drwxr-xr-x  2 dcika dcika 4096 Dec  9 12:11 run
drwxr-xr-x  3 dcika dcika 4096 Dec  9 12:11 sbin
dr-xr-xr-x  2 dcika dcika 4096 Dec  9 12:11 sys
drwxrwxr-x  2 dcika dcika 4096 Dec  9 12:11 tmp
drwxr-xr-x 11 dcika dcika 4096 Dec  9 12:11 usr
drwxr-xr-x  9 dcika dcika 4096 Dec  9 12:11 var
```

### 6.3.5 Setting up the Vitis workspace

DFX/IDF demo software application is delivered also in the source code to enable users to do software customizations wherever necessary. This paragraph explains how to setup the Xilinx Vitis environment for software customizations referencing the Petalinux project build results described earlier where necessary.

Source files delivered can be found in the location selected when running the `java --jar` command giving the jar file with deliverables as an argument. The default name of the deliverables folder should be something like `logiREF-DFX-IDF_YYMMDD`, and the underlying folders found there would be like shown in the **Figure 37**. Feel free to explore it. We have already seen parts of the software/Linux content, now our focus is on the `software/Vitis_workspace` path.

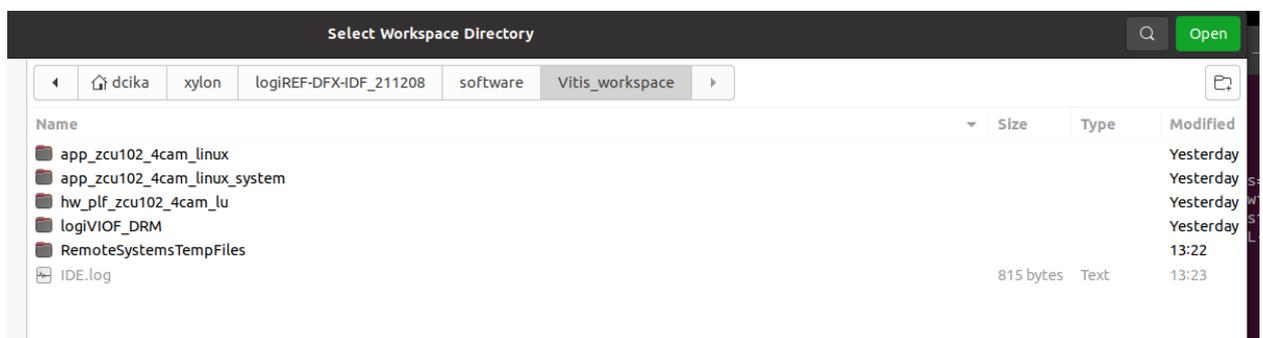


**Figure 37: Directory structure of project deliverables**

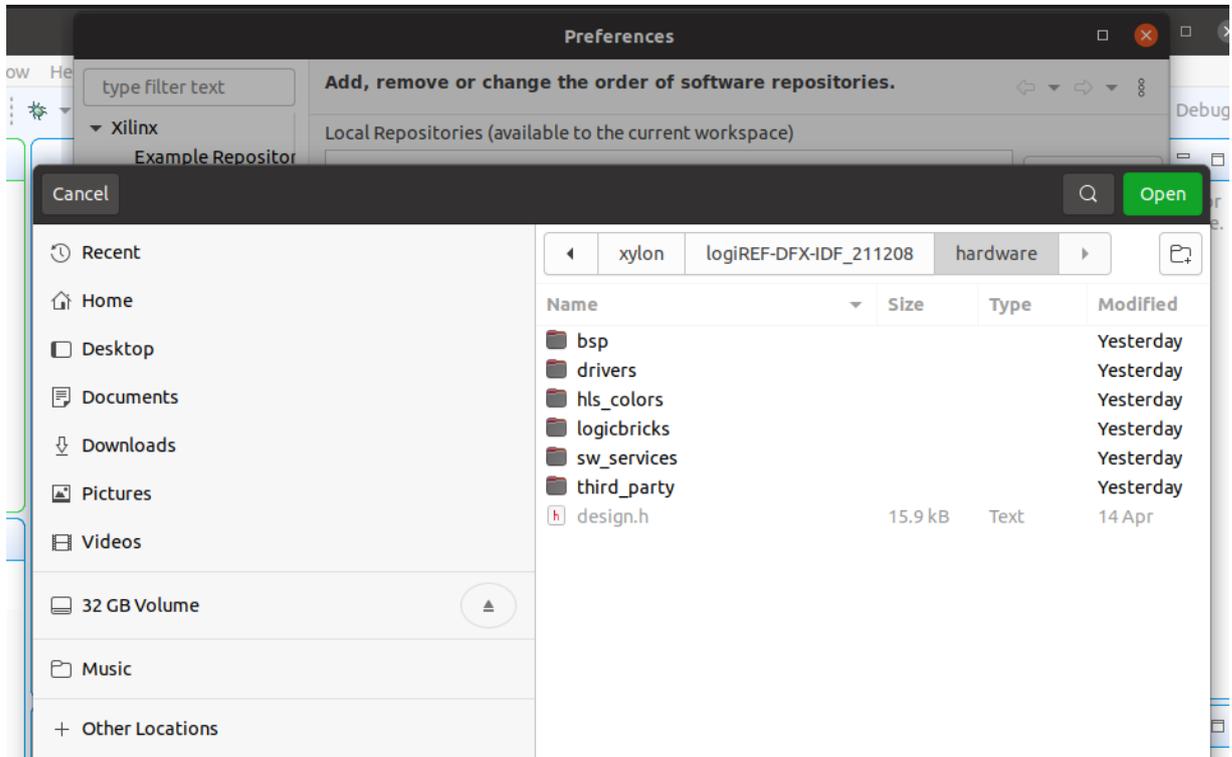
Quick steps required to set up the Vitis workspace and become ready for application development are the following:

1. Start Vitis, select the workspace: `logiREF-DFX-IDF_YYMMDD/software/Vitis_workspace`

```
dcika@ubuntu18-04-3-lts:~$ source ~/tools/Vitis/2021.1/settings64.sh
dcika@ubuntu18-04-3-lts:~$ vitis
```

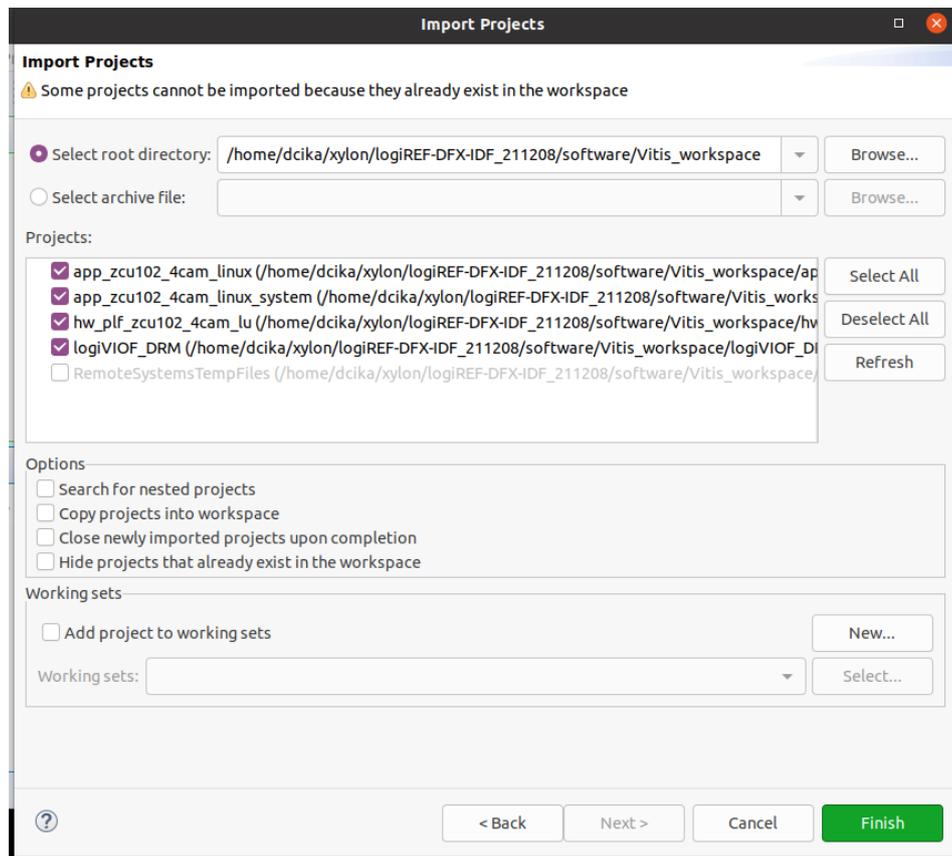


- In the Vitis go to: *Xilinx*→*Software Repositories*→*New*, add **logiREF-DFX-IDF\_YYMMDD/hardware** directory

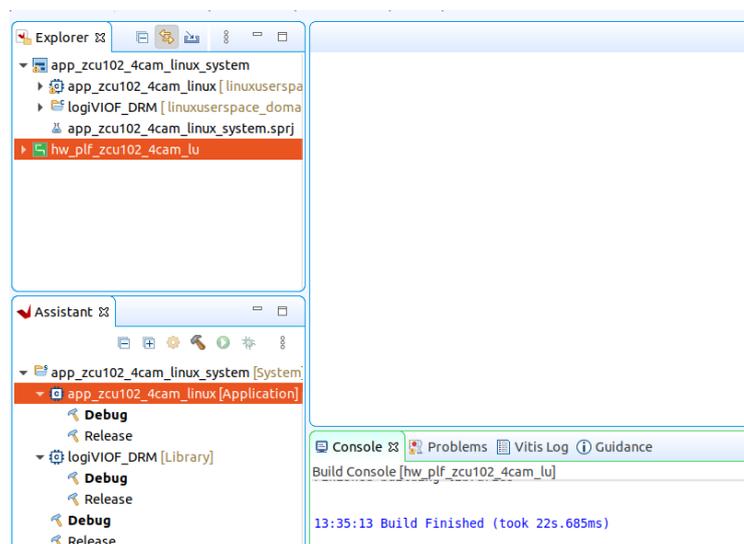


Please note that adding the custom HW information to the Vitis Software repositories is very important for using and generating the custom HW platform of the type linuxuserspace that could not be created by default otherwise. This platform is referred by applications developed in this demo.

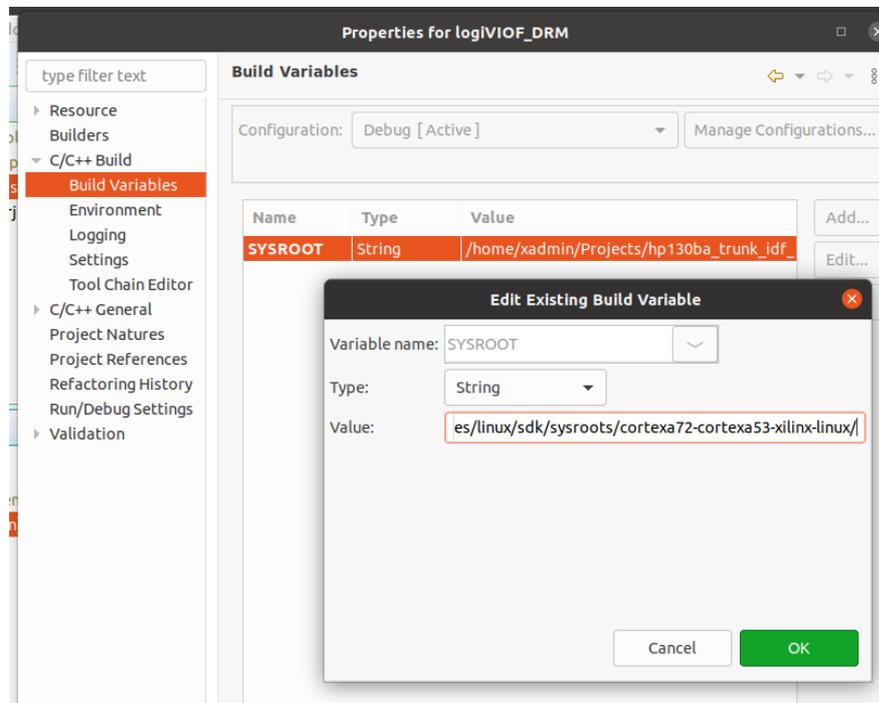
- In the Vitis go to: *File*→*Import*→*Eclipse workspace or zip file*→*Next*, in *Select a root directory* choose **logiREF-DFX-IDF\_YYMMDD/software/Vitis\_workspace**, deselect option “Copy projects into workspace” and select all projects and click *Finish*



- Right click the platform project **hw\_plf\_zcu102\_4cam\_lu** and build it



- Right click **logiVIOF\_DRM** → *C/C++ Build Settings* → *C/C++ Build* → *Build variables* → select **SYSROOT** → *Edit...* and set *Value* to < path\_to\_petalinux\_sysroot\_directory >

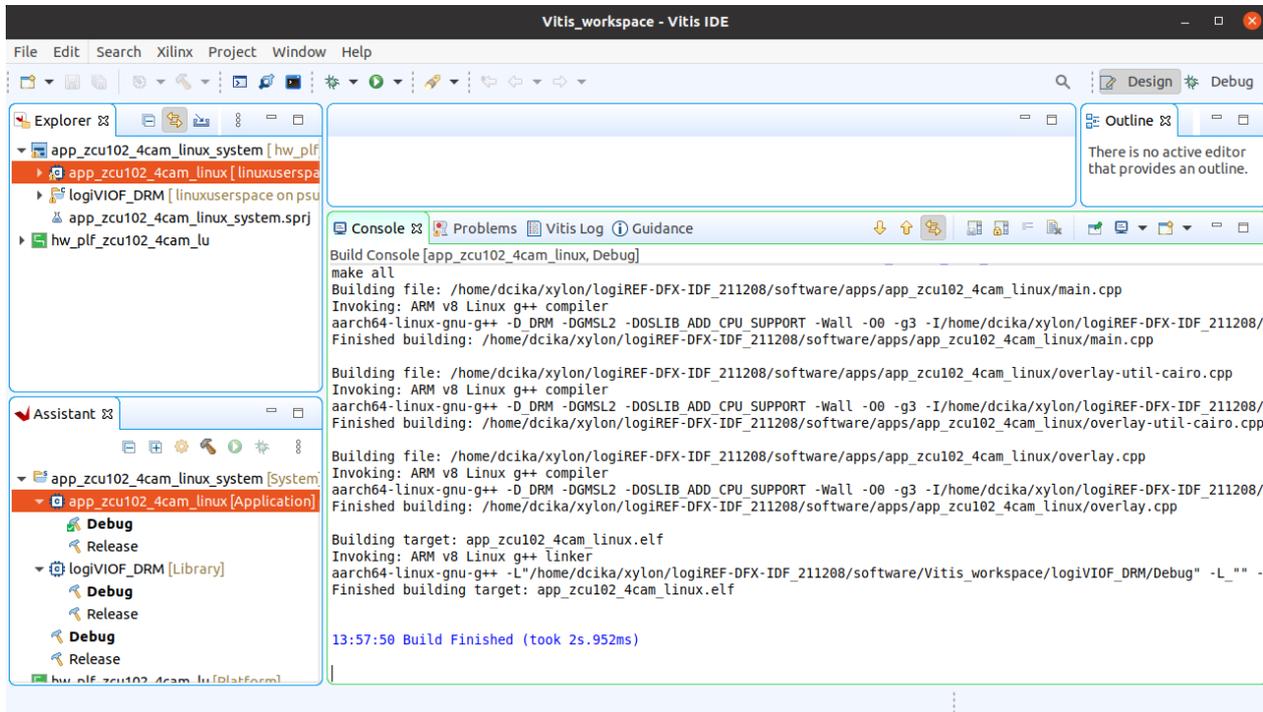


The **SYSROOT** variable may not be set, or more likely it may point to the value not relevant for your system as it was set by developer running a different environment. Either the existing value should be deleted and set to correct value, or it could be edited. In any case if not set properly, the development environment may not have some fundamental information to build the system and build process will terminate.

- Right click and build Library project **logiVIOF\_DRM**
- Right click **app\_zcu102\_4cam\_linux** → *C/C++ Build Settings* → *C/C++ Build* → *Build variables* → select **SYSROOT** → *Edit...* and set *Value* to the < path\_to\_petalinux\_sysroot\_directory >
- Build the **app\_zcu102\_4cam\_linux** application, check for build errors, and copy the application executable (i.e. **app\_zcu102\_4cam\_linux.elf**) to the SD card along with other necessary files. **Figure 38** shows the example of successfully built application for **DEBUG** configuration.



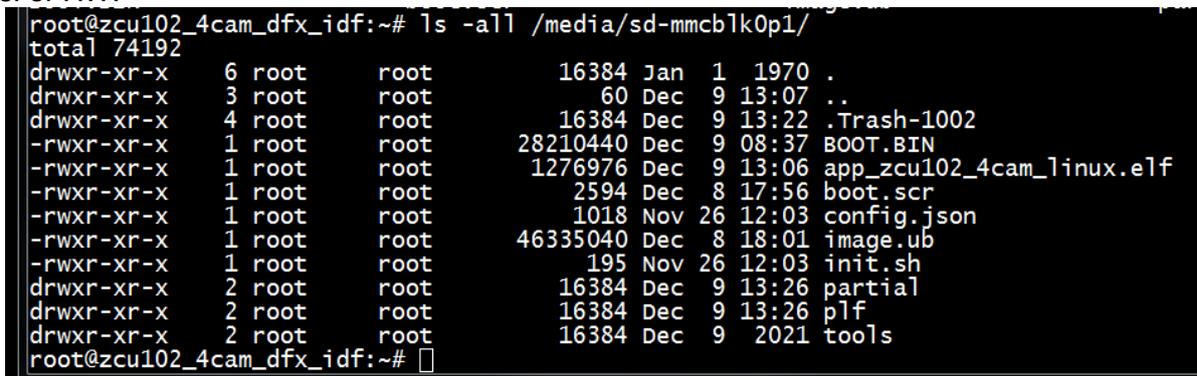
Due to dependencies between the application and the logiVIOF library, please make sure to use the same configuration options (Release/Debug) for both the application and the logiVIOF library.



**Figure 38: Vitis application development environment in action**

Now the board can be restarted, and init script will run the application automatically. Of course, it is possible to start the application manually by finding where the SD card was mounted in the Linux file system, and running this or different executable directly like with any Linux system.

A partial screenshot of the Linux console connection with files on the SD card is shown in the Other executables found on the SD card does not have to be rebuilt if there were no changes in kernel or HW.



**Figure 39: ZCU102 Linux console connection with the SD card mounted**

## 6.4 Debugging Software Application with the TCF Agent

1. Launch Vitis with imported Vitis workspace
2. Make sure that ZCU102 board is correctly connected to PC via Ethernet connection
3. In Target Connections Window (enable it in *Window* → *Show view...* → *Target Connections* → *Open*) right click on **Linux TCF Agent** and click *New Target*
4. Set **Target Name** and under **Host** fill with IP address of ZCU102 and Test Connection
5. Open **Debug Configurations** window
6. To create a new Debug Configuration, right click on the **Target Communication Framework** section on the left hand side of the Debug Configuration GUI and click on **New Configuration**
7. Uncheck Use local host as the target
8. In Available targets section the Host IP address of target has to match IP set on ZCU102 board, then select that target (see Figure 41)

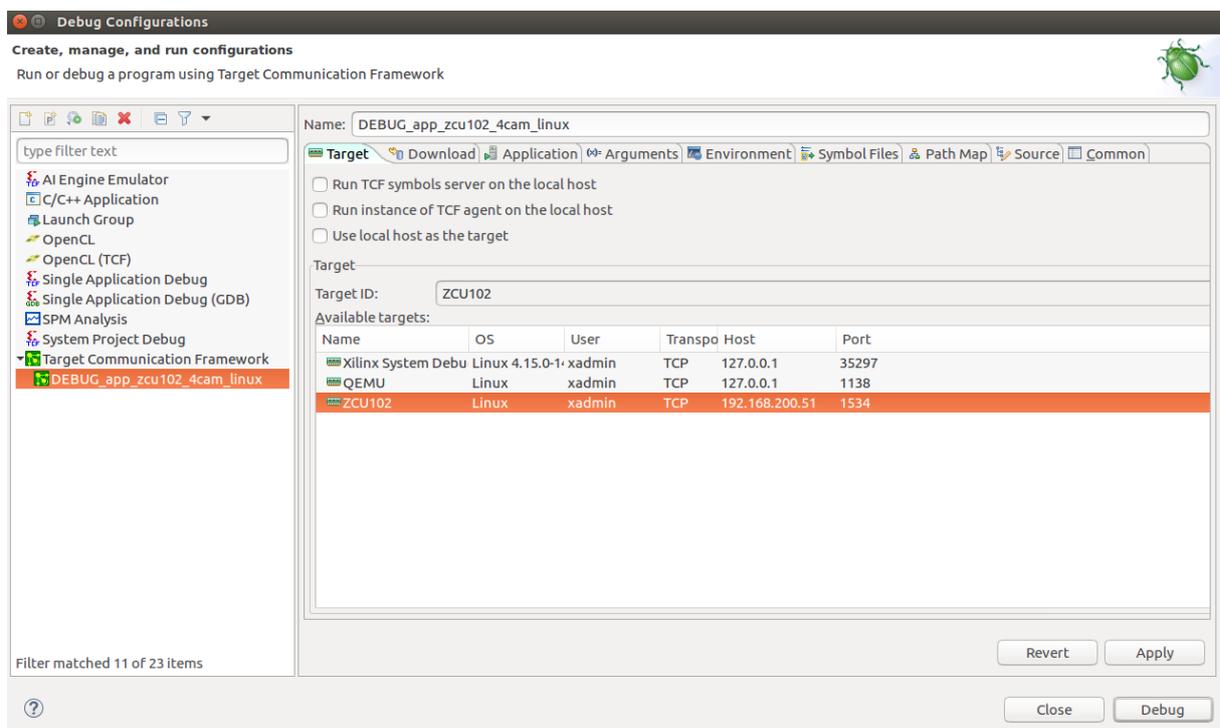
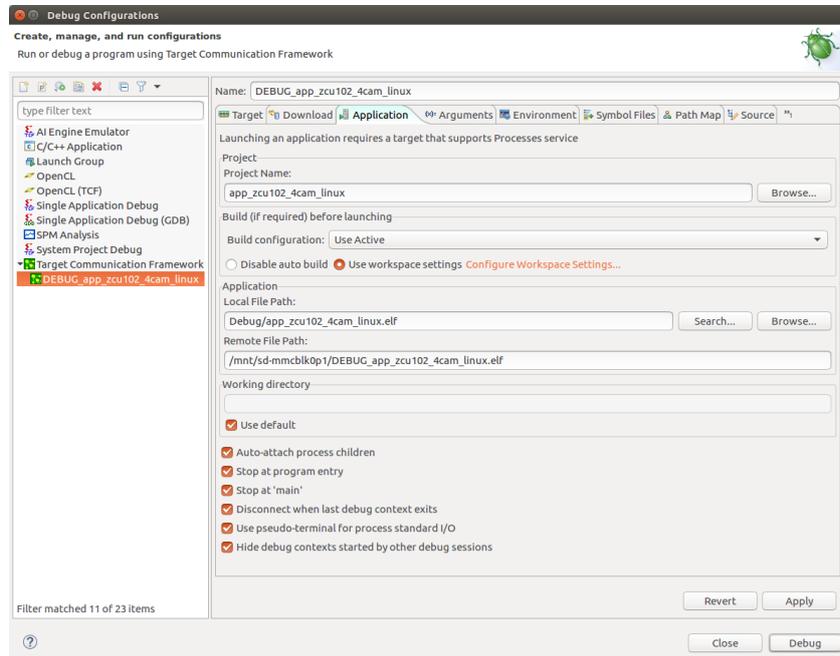


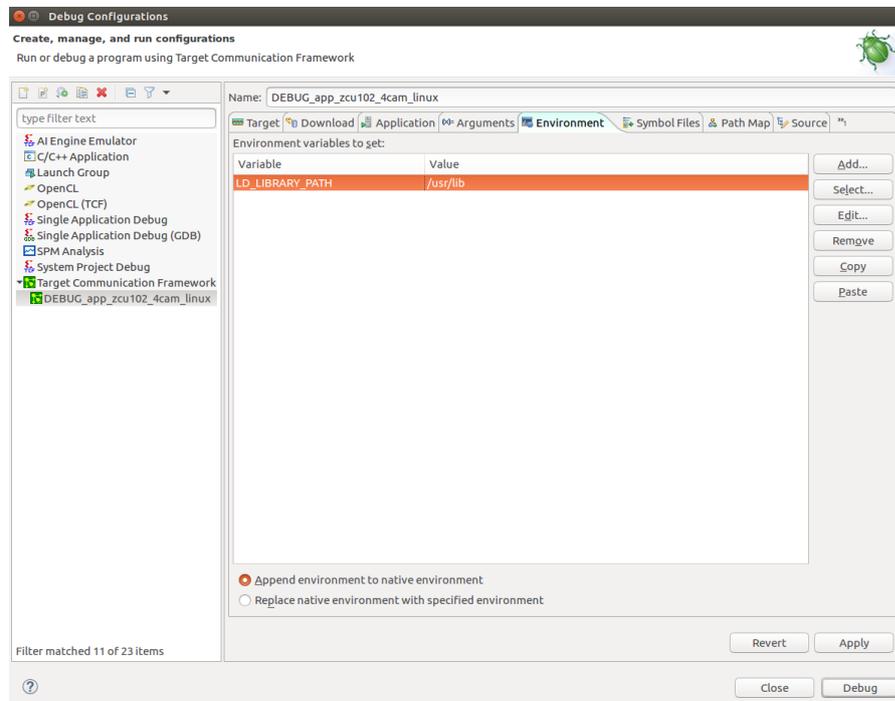
Figure 40: Vitis Workspace – Debug Configurations

9. Switch to the **Application** Tab
10. Enter **Project Name**, **Local File Path** and **Remote File Path** (see Figure 42)



**Figure 41: Vitis Workspace – Application Tab**

11. If shared libraries are used set paths to in the **Environment** tab (see Figure 43)



**Figure 42: Vitis Workspace – Environment Tab**

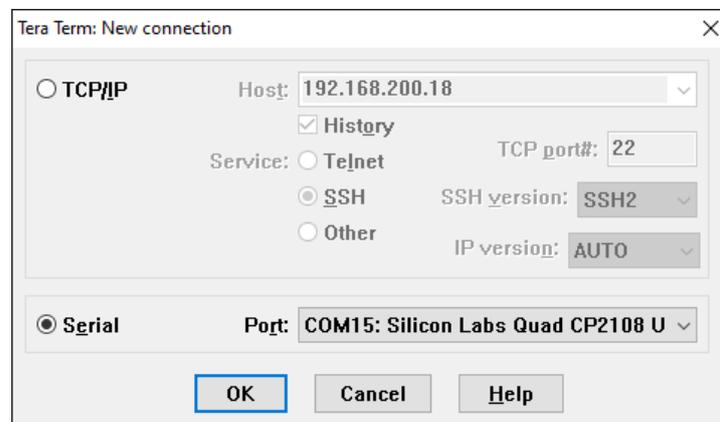
## 12. Start Debug

## 6.5 SEM UART Example commands

SEM IP Controller can inject errors at specific locations through the dedicated SEM UART connection. The error injections performed below are by Linear Frame Address (LFA). This process is a simple read-modify-write to invert one configuration memory bit at an address specified as part of the error injection command. SEM IP Controller must stop scanning for errors before injecting an error, so it is commanded to transition to the Idle state. While in the Idle state, errors can be injected into the configuration memory one bit at a time. After injecting errors, SEM IP is commanded to transition to the Observation state. In the Observation state, SEM IP Controller begins scanning for errors in the configuration memory. When an error is detected, SEM IP corrects the error and generates a report through the SEM UART interface.

The report contains the address in which the error occurred. For more information on SEM IP commands, reports, and behaviour, see the ([REF \[4\]](#)) documentation.

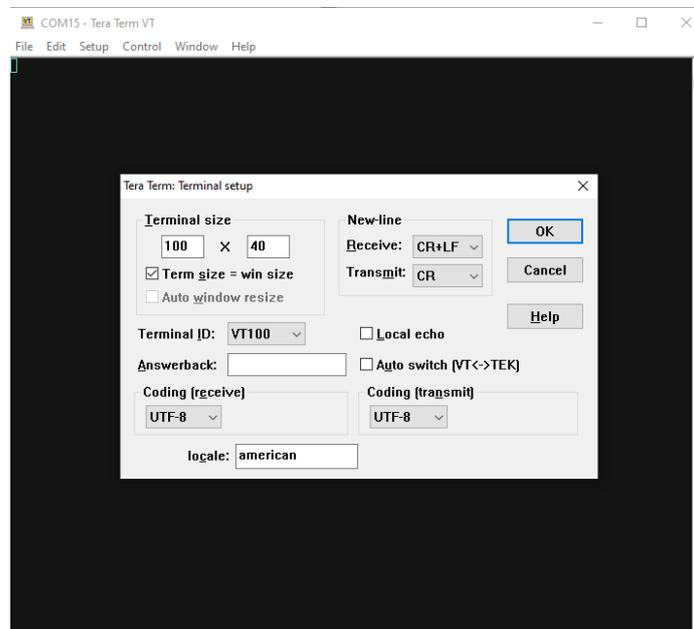
Some basic example commands and logs that are based on UltraScale+™ FPGAs are run below. These short examples from SEM IP UART logs are a sample of observable activity when the SEM IP is configured in Mitigation and Testing mode with the Error classification feature disabled, except where noted. First thing that user needs to do to use SEM UART and interact with the SEM IP controller is open another terminal (TeraTerm etc...), select the correct port on Silicon Labs CP210x USB to UART Bridge. In our case, COM15 is used.



**Figure 43: Connection for the SEM UART dialog box**

After connection is a success, some additional setting needs to be made in connection settings window for the proper and correct UART functionality. Those setting are:

*Baud: 115200*  
*Settings: Data 8, Parity None, Stop 1*  
*Flow Control None*  
*Terminal ID VT100*  
*New-line transmit CR*  
*New-line receive CR+LF*  
*Local Echo No*



**Figure 44: Connections settings for SEM UART**

After everything is set up correctly and the design is running, on SEM UART there is a initialization log that indicates correctly configured and started SEM IP controller. More on this how to work with SEM IP controller can be found ([REF \[4\]](#)) documentation.

```
SEM_ULTRA_V3_1
SC 01
FS 04
AF 01
ICAP OK
RDBK OK
INIT OK
SC 02
O>
```

There, we can see that SEM IP Controller starts in Observation mode (O>) right after initialization. In our reference design, we will test some single bit and double bit error injection into one of our four reconfigurable regions by running appropriate SEM IP Controller Injection commands. User can inject single bit error that will be corrected by SEM IP Controller and the appropriate flag will be set. Prior to error injection, user must set SEM IP controller into Idle mode (running I command at SEM UART). This will return correct Status code (SC 00 – Idle mode)

```
O> I
SC 00
I>
```

In Idle mode, user can then perform single bit injection errors, one bit at the time by running appropriate command. For the ECC error in reconfigurable partition 0, injection command is:

```
N C0005D2E5A0
```

```
SC 10
SC 00
```

Reading status codes after error injection, we can see that the SC 01 is stated first, which indicates that the Injection was performed and after that, SEM IP Controller is again in Idle mode. To set SEM IP controller into error detection mode, SEM IP Controller needs to be configured in Observation mode (running O at SEM UART).

```
O>
RI 00
SC 04
ECC
TS 000019F9
PA 000C6000
LA 00005D2E
COR
WD 2D BT 00
END
FC 00
SC 08
FC 40
SC 02
O>
```

This logs shows us the type of error that is detected and address at which error occurred. Also, by reading status codes and flags, we can see that SC 04 is set. This status codes indicate that Correction is performed by SEM IP Controller. Finally, setting the flag FC 04 indicates type of error detected, so we can see that Correctable and Essential error is detected. After detection and correction, SEM IP Controller goes again into Observation mode. (SC 02).

For injection of two-bit error, that will cause SEM IP Controller to trigger the Uncorrectable event, we are simply injecting two error commands one after one when SEM IP Controller is in Idle mode:

```
I> N C0005D2E5A0
SC 10
SC 00
I> N C0005D2E5A4
SC 10
SC 00
I>
```

Now, if we put SEM IP Controller into Observation mode, next log is shown:

```
I> O
SC 02
O>
RI 00
SC 04
ECC
TS 0000589A
PA 000C6000
LA 00005D2E
COR
```

```
END  
FC 60  
SC 08  
FC 60  
SC 00  
I>
```

We can see now that SEM IP Controller flag is set to FC 60 which tells us that Uncorrectable and Essential Error is detected. From this point, user must perform additional actions on corrupted configuration memory because SEM is not able to correct these types of errors. In our DFX solution, when there is an uncorrectable error in one of reconfigurable partitions, we can just simply perform partial reconfiguration at affected partition with correct and available partial bitstreams.

## 7 REFERENCES

Table 7.1: List of references

Reference	Description
REF [1]	Vivado Design Suite User Guide: Dynamic Function eXchange ( <a href="#">UG909</a> )
REF [2]	<a href="#">Vitis Vision Library Functions</a>
REF [3]	<a href="#">Isolation Design Flow for UltraScale+ and FPGAs and Zynq UltraScale+ MPSoCs</a>
REF [4]	<a href="#">UltraScale Architecture Soft Error Mitigation Controller v3.1</a>

## 8 REVISION HISTORY

<b>Version</b>	<b>Date</b>	<b>Author</b>	<b>Approved by</b>
1.00.a	November 24 <sup>th</sup> , 2021	A. Popovic, T. Cvoriscec	D. Cika
	Initial Xylon release		
1.01	December 8 <sup>th</sup> , 2021	D. Cika	G. Galic
	Update of chapter 6.3 - Change the Delivered Software		
1.02	February 21 <sup>st</sup> , 2022	R.Soldat	
	Added Chapter 4.2 - Getting the Xilinx HDMI 1.4/2.0 Transmitter Subsystem License. Updated the full name of the reference design.		