



# **Xilinx Video Mixer IP (v5.0)**

## **Feature Implementation on Zynq FPGA**



**Vitis Unified Software Platform 2020.1**  
August 25, 2021

## Overview

The main objective of this design is to implement the video mixer design and its major features. The design is implemented on the Digilent Zybo (Zynq-7000) development board. This is the updated version of the previous video mixer reference tutorial [[Xilinx\\_Zynq-Video-Mixer-Tutorial\\_LogicTronix](#)].

***Unlike previous versions, this reference tutorial version uses the AXI GPIO to implement the video mixer features, such as enabling/disabling of overlay layers, setting the alpha value, scaling and moving the overlay layers through the onboard Slide Switches and Push Buttons.***

This tutorial design is created with VITIS IDE 2020.1. In 2020 Vitis/Vivado 2020.1 software was released and this software was available with features such as 500+ FPGA-accelerated functions spread across 11 open-source Vitis libraries along with the Improved RTL kernel integration & enhancements for easier custom Vitis target platform creation for embedded platforms and many other features which are listed in the Xilinx website ([Ref. 1](#)). Vivado is the Xilinx software that is used for creating block diagrams for working in hardware such as the Zynq Board, Pynq Board and other boards. Vitis IDE is used for the software development for the project in the Vivado and the Vitis HLS is mainly used for developing the IP packages for the Vivado.

The ZYNq BOard (ZYBO) is a feature-rich, ready-to-use, entry-level embedded software and digital circuit development platform that was built around the Xilinx Zynq-7000 family. The Z-7010 is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture that tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. It is combined with a set of various multimedia and connectivity peripherals that can be used for hosting a whole system design. The Zynq Board comes with the peripherals such as onboard memory, video and audio I/O, dual-role USB, Ethernet, and SD slot, which are designed up-and-ready with no additional hardware needed. Additionally, six Pmod ports are available to put any design on an easy growth path. ([Ref. 2](#))

## Board Overview

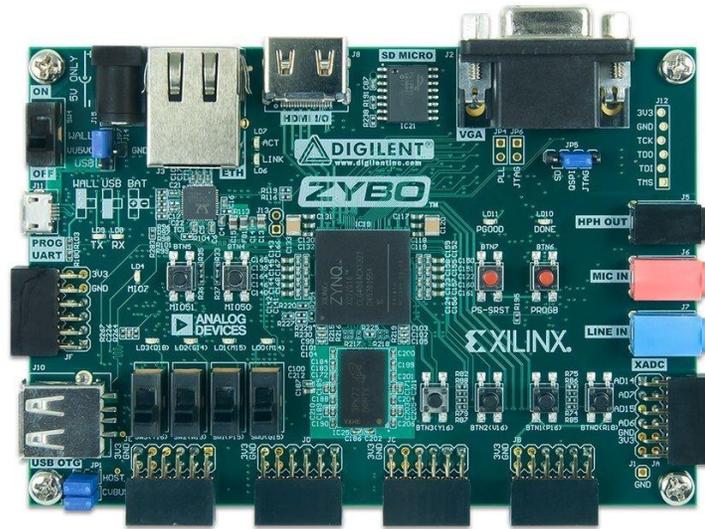


Figure 1 ZYNq BOard (ZYBO)

Its key features are given below:

- 650MHz dual-core Cortex-A9 processor
- DDR3 memory controller with 8 DMA channels
- High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
- Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
- Reprogrammable logic equivalent to Artix-7 FPGA
- ZYNQ XC7Z010-1CLG400C
- 512MB x32 DDR3 w/ 1050Mbps bandwidth
- Dual-role (Source/Sink) HDMI port
- 16-bits per pixel VGA source port
- Trimode (1Gbit/100Mbit/10Mbit) Ethernet PHY
- MicroSD slot (supports Linux file system)
- OTG USB 2.0 PHY (supports host and device)
- External EEPROM (programmed with 48-bit globally unique EUJ-48/64™ compatible identifier)
- Audio codec with headphone out, microphone and line-in jacks
- 128Mb Serial Flash w/ QSPI interface
- On-board JTAG programming and UART to USB converter

- GPIO: 6 pushbuttons, 4 slide switches, 5 LEDs
- Six Pmod ports (1 processor-dedicated, 1 dual analog/digital, 3 high-speed differentials, 1 logic-dedicated). ([Ref. 2](#))

## IP overview

### Video Mixer IP v5.0



Figure 2 Video Mixer

Video mixer is the Logical IP core of the Xilinx Vivado software. The latest video mixer is version 5.0. Video mixer is used by users to mix the video layers from the stream or memory-mapped layers onto the master layer which is the main video stream. This IP core's main objective is to provide a system which is called a video processing system in which the users can overlay layers onto the master layer. The video mixer can support up to 16 layers of the stream or memory-mapped layers for overlay with a Logo layer as an optional layer ([Ref. 3](#)). This IP is programmable to control frame size, background color, and layer position. The main features of the Video Mixer are given below:

- Supports (per pixel) alpha-blending of nine video/graphics and logo layers video/graphics
- Optional logo (in block RAM (BRAM)) layer with color transparency support
- Layers can either be memory-mapped AXI4 interface or AXI4-Stream
- Provides programmable background color
- Provides programmable layer position and size
- Provides scaling of layers by 1x, 2x, or 4x
- Optional built-in color space conversion
- Supports RGB, YUV 444, YUV 422, YUV 420
- Supports 8, 10, 12, and 16 bits per color component input and output on stream interface, 8-bit and 10-bit per color component on the memory interface
- Supports semi-planar memory formats next to packed memory formats
- Supports spatial resolutions from  $64 \times 64$  up to  $4096 \times 2160$
- Supports 4K60 in all supported device families. ([Ref. 3](#))

## Design Flow

The design flow step for the project is in two steps. They are Hardware Design and Software Design. The Hardware design is done in Vivado and the software design is Vitis IDE.

### Hardware Design

#### Creating New Project

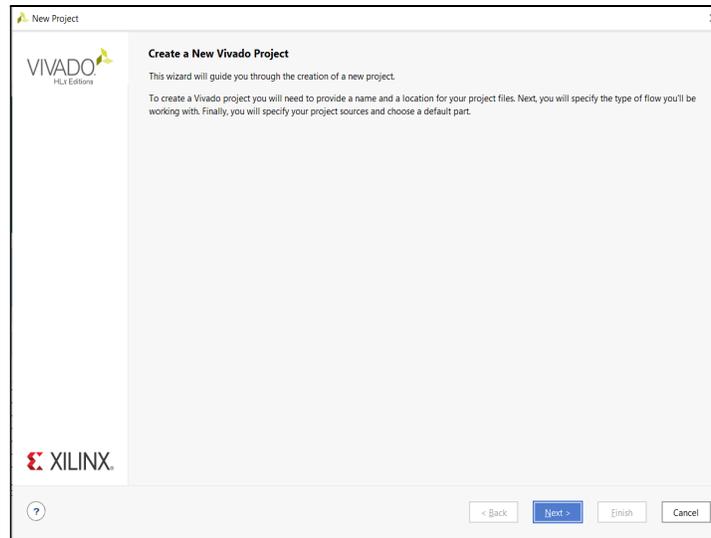


Figure 3 Creating New Project

The new project is created by creating a project and by clicking the **Next** button, we can select the project name, project type, sources and board selection.

#### Board selection

Display Name	Preview	Vendor	File Version	Part	I/O Pin Count	Boards
Zedboard		digilentinc.com	1.0	xc7z020clg484-1	484	D:
Zybo Z7-10		digilentinc.com	1.0	xc7z010clg400-1	400	B:
Zybo Z7-20		digilentinc.com	1.0	xc7z020clg400-1	400	B:
Zybo		digilentinc.com	2.0	xc7z010clg400-1	400	B:

Figure 4 Board Selection

Here the Zybo board is selected. The Zybo board is not initially present in the board selection menu. It had to be added manually by adding the board files to the vivado. The board files can be downloaded from the Digilent website.

## Creating Block Design

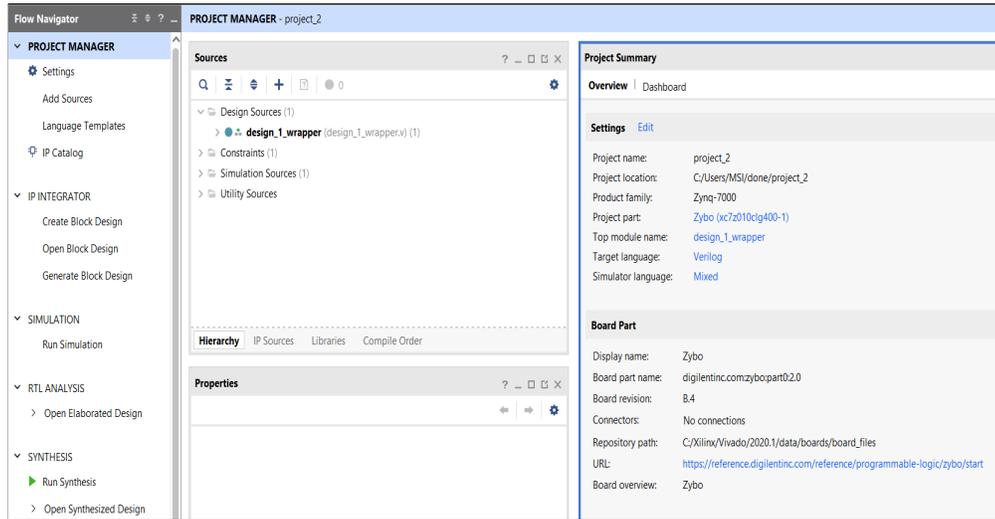


Figure 5 Creating Block Design

A block design is created by navigating to the **Project Manager** and then to the **IP Integrator > Create Block Design**. This will open a window in which the design name is selected and then the IP integrator window is opened.

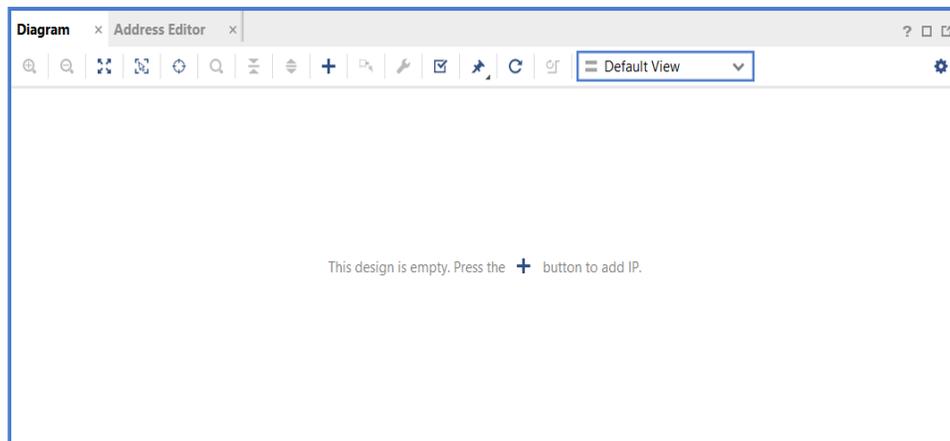


Figure 6 Adding IP and designing the block diagram

The above figure is the **Diagram Window** in which the IP is added and connected to create a respective block diagram of the project. Here we can also add the extra IP packages by going to the **Project Manager** and selecting the **Settings** and below the IP selecting the **Repository** tab. In the repository tab, by clicking the add button (+ sign button) we can add the IP package from the respective location and then select the **OK** button.

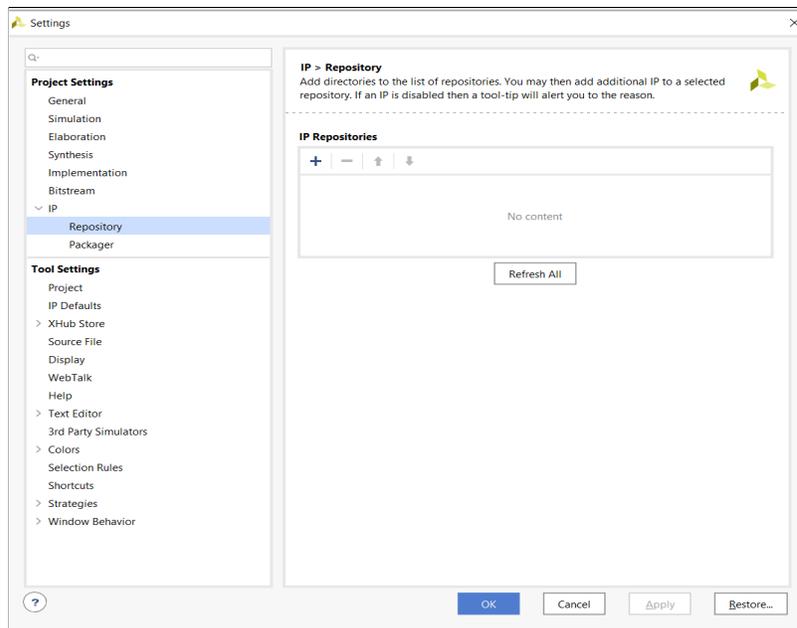


Figure 7 Adding the IP package repository

After completing the block design we have to assign the address to the IP of the AXI IP cores. It can be automatically done by clicking **Auto Assign Address** (downside arrow) as shown in figure 8.

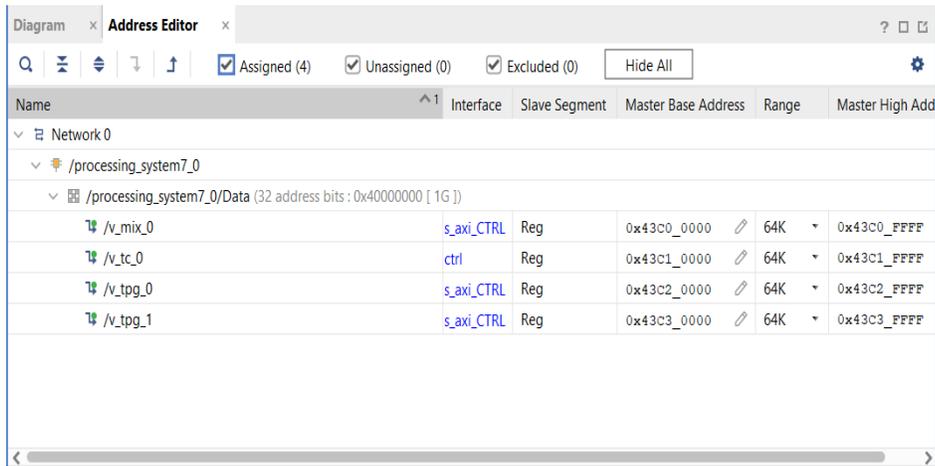


Figure 8 Assigning the Address

## IP Connection and Details

### 1. Zynq Processing System IP



Figure 9 Zynq Processing System

The Zynq Processing System is the main processing unit of the Zynq Board. Its main function is to provide the configuration and control of all IP drivers and the video processing IP blocks ([Ref. 4](#)). **Block automation** is run for the Zynq processing system and then the **DDR** and **FIXED\_IO** are made using external terminals. The DDR of the processing system is used as a frame buffer. The **FCLK\_CLK0** is the clock configuration port that provides the clock frequency to the other IP. Here the value of the clock frequency used is 100 MHz and it is connected to the input of the Clocking Wizard IP. The **FCLK\_RESET0\_N** pin is connected to the external\_reset pin of the Processor system reset. The **M\_AXI\_GP0** is connected to the AXI INTERCONNECT master axis pin that is used to configure the video processing chain.

### 2. Clocking Wizard IP



Figure 10 Clocking Wizard IP Core

The Clocking wizard IP is used for providing the clocking frequency to the IP cores in the vivado design blocks ([Ref. 5](#)).

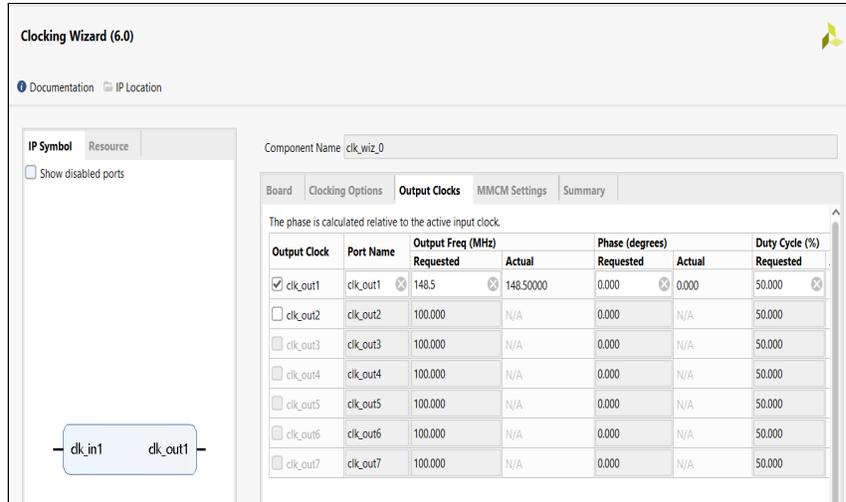


Figure 11: Clocking Wizard Customization window

Here in the above figure, 148.5 MHz frequency clock is set which is provided to the remaining IP.

### 3. DVI2RGB IP

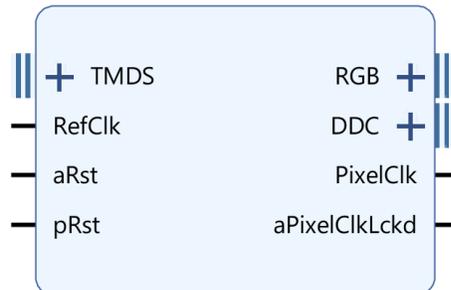


Figure 12 DVI2RGB IP Core

The DVI2RGB IP block is used for inputting the HDMI data from the HDMI port of the Zybo. This IP block is designed by Digilent. This IP gives the RGB output which is 24-bit data.

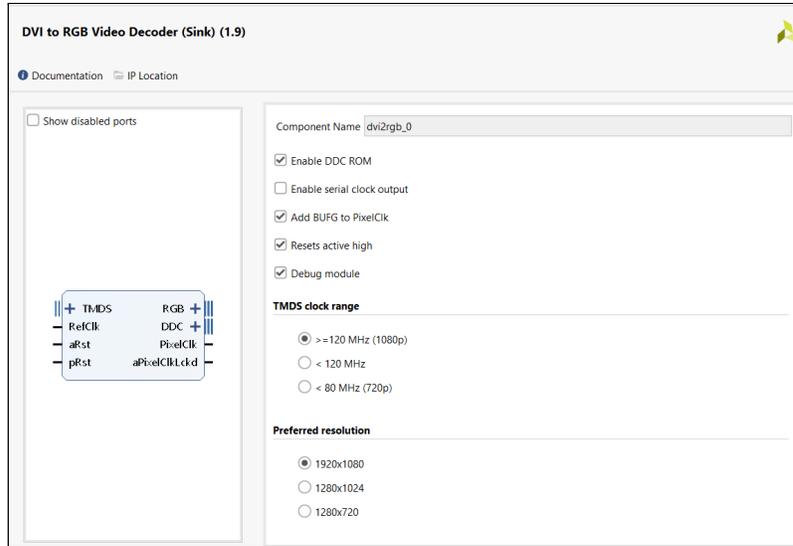


Figure 13 DVI2RGB Customization Window

The customization window of the DVI2RGB IP block is shown in the above figure. Here we take the clock range greater or equal to 120 MHz for the 1080p and the resolution selected was 1920x1080p with the DDC ROM enabled. In this design, HDMI works as a master layer.

#### 4. Video In to AXI4-Stream IP

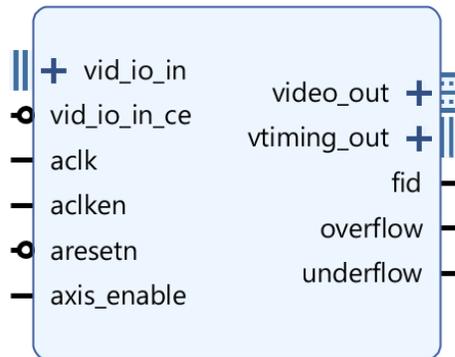


Figure 14 Video in to AXI4-Stream IP core

This IP block is mainly used for inputting the streaming video input from the HDMI or VGA IP which gives the RGB output and this RGB output is connected to the **vid\_io\_in** pin which then converts the RGB video data to the AXI4 stream format. Also, the video timing is given output separately which is then connected to the **v\_timingin** pin of the Video Timing Controller. ([Ref. 7](#))

## 5. Video Test Pattern Generator IP

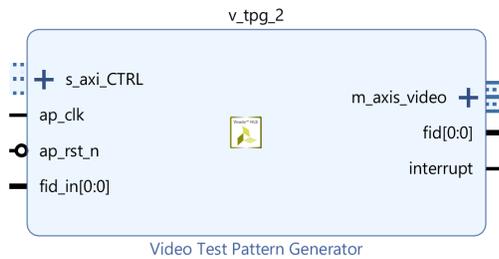


Figure 12 Video Test Pattern Generator IP Core

The Video Test Pattern Generator provides a video stream for testing which can be of different patterns such as color bars, color ramp, solid color and other patterns ([Ref. 8](#)). Here in this design, TPG is used to connect the stream layer interface of a video mixer IP.

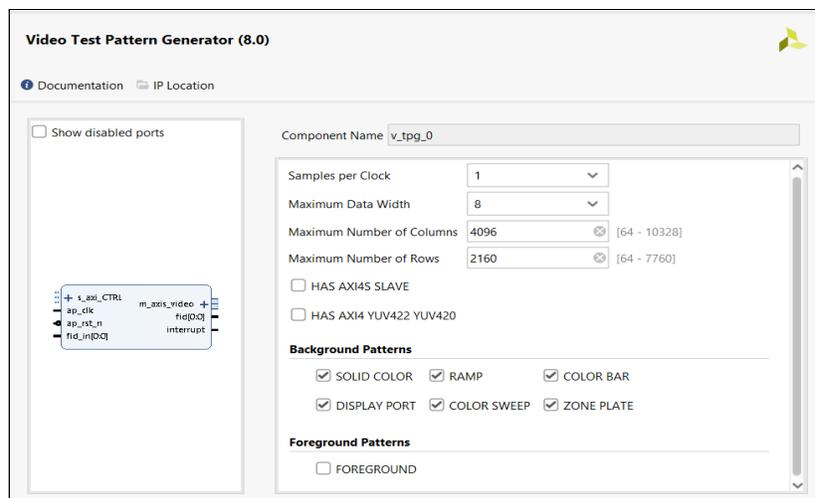


Figure 13 Video Test Pattern Generator Customization Window

The TPG is programmable to generate background patterns, foreground patterns, to set the resolution of the video etc. Here all the background patterns are selected and the sample per clock is taken as 1 and the maximum data width is selected as 8.

## 6. Video Mixer IP

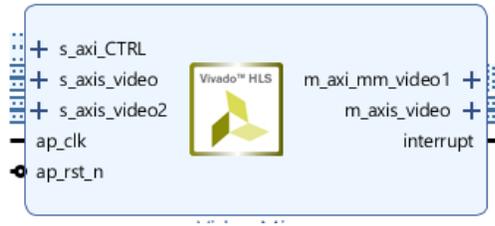


Figure 14 Video Mixer IP Core

Video mixer is the Logical IP core of the Xilinx Vivado software. The latest video mixer is version 5.0. It enables the user to mix the video layers from the stream or memory-mapped layers. The IP core's main objective is to provide a video processing system in which the users can overlay video onto one another. The main feature of the video mixer is that it can be used to overlay up to 16 videos on the main video.

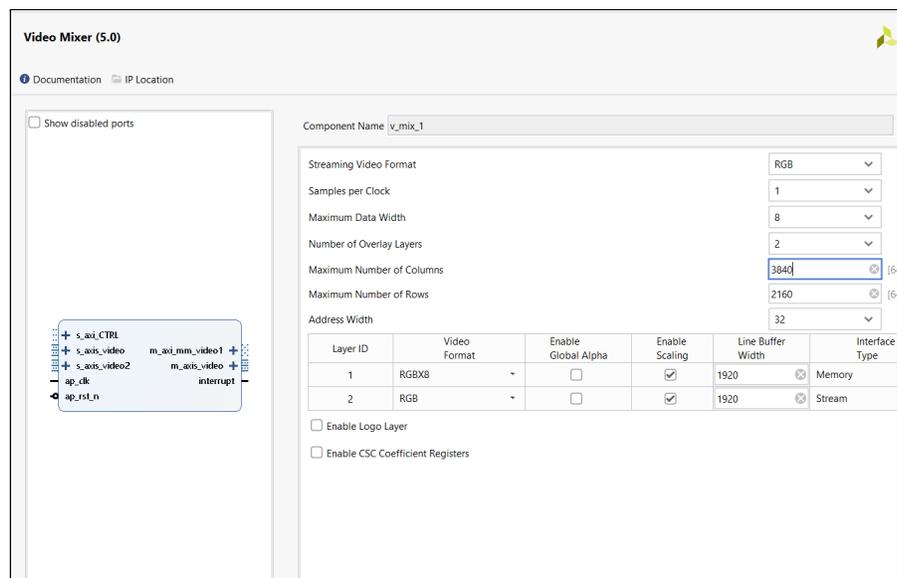


Figure 15 Video Mixer configuration window

In this design, the video mixer is configured such that 2 overlay layers were enabled with interface type memory interface and stream interface type and video format were RGBX8, RGB with alpha blending disabled and scaling enabled respectively. The stream TPG video out pin is connected to the **s\_axis\_video2** pin of the video mixer. HDMI video is fed to the master layer in which the other TPG stream layer and memory-mapped layer are overlaid. The main configuration of the video mixer is done from the software application with the logo layer is disabled.

## 7. AXI4-Stream to Video Out IP

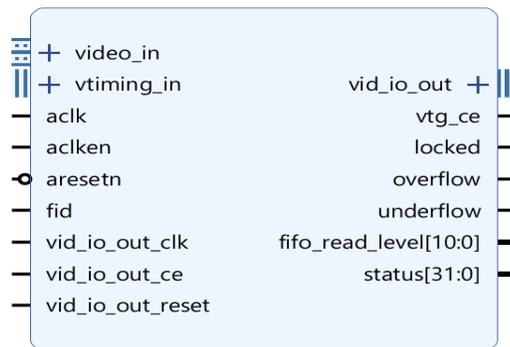


Figure 16 AXI4-Stream to Video out IP Core

The above figure shows the AXI4-Stream To Video out IP core. This IP core takes the AXI4-Stream and its corresponding video timing and then converts it into native video output. ([Ref. 9](#))

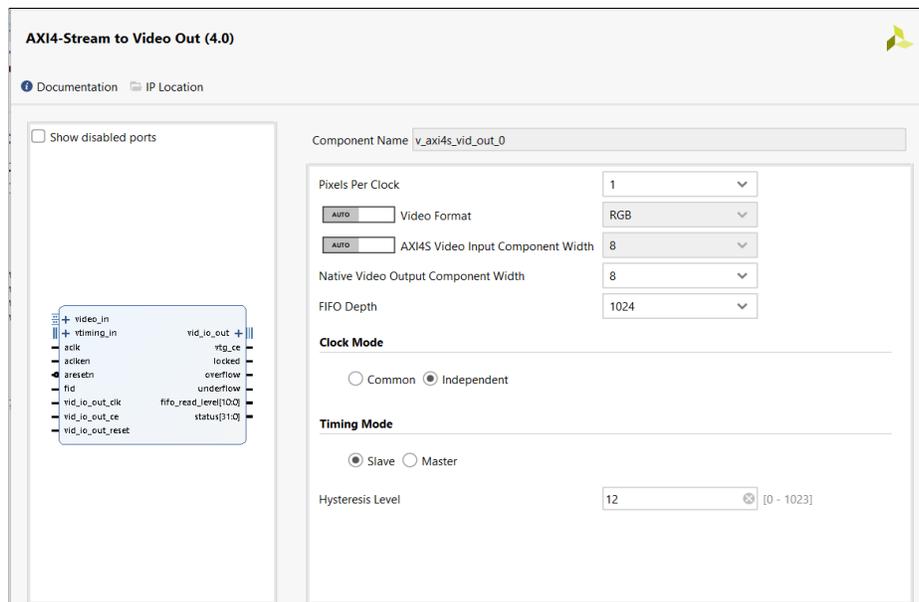


Figure 17 AXI4-Stream To Video Out Customization Window

The above figure shows the AXI4-Stream To Video Out IP core customization window. This IP core takes the video input from the video mixer output. The video stream is the AXI4-Stream which is converted into RGB. The output is sent from the `vid_io_out` pin to the RGB2VGA IP block as input. The video timing is

obtained from the Video Timing Controller **vtiming\_out** pin. The **vtg\_ce** pin is connected to the VTC **gen\_clk** pin.

## 8. Video Timing Controller IP

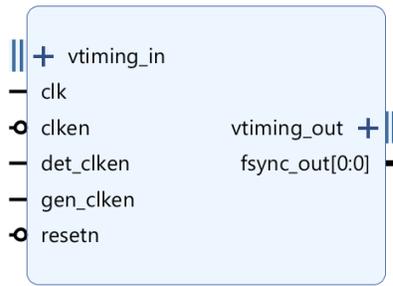


Figure 18 Video Timing Controller IP Core

The above figure shows the Video Timing Controller (VTC) IP core. Its main feature is to give timing parameters for different video modes such as 720, 1080 etc. There are mainly two ways the VTC is used: one is the generation and the other is detection ([Ref. 10](#)). The generation is used for generating the vertical blank, horizontal blank, vertical sync, horizontal sync, active video and the detection is used for detecting the video timing from the other IP.

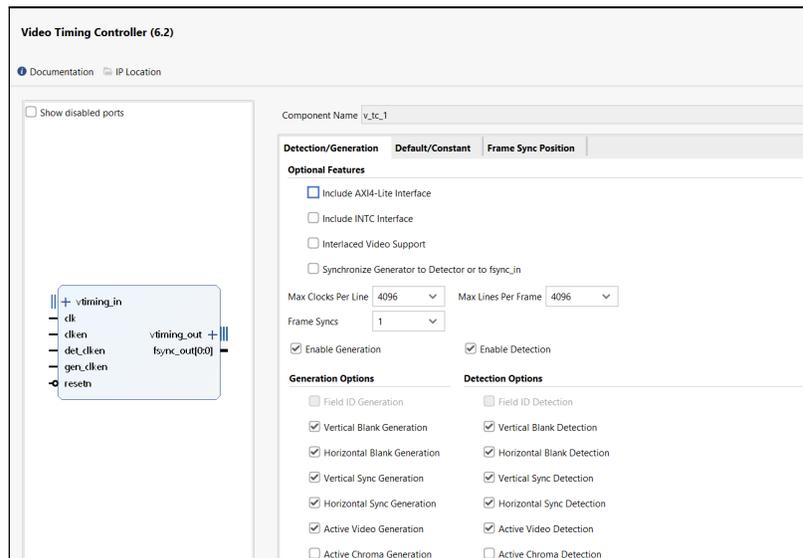


Figure 19 Video Timing Controller Customization Window

Here both generation and detection are enabled on which for the generation of the video timing and also for the detection of the video timing necessary for the project. The video mode is selected to be 1080p and since the AXI4-Lite interface is not selected there is no need for coding in the software application. The **clken** and **s\_axi\_aclken** pins are connected to the constant with value 1. And the **gen\_clken** is connected to the **vtg\_ce** pin of the AXI4-Stream To Video\_Out IP.

## 9. AXI GPIO IP



Figure 20 AXI GPIO IP Block

The AXI GPIO IP block is one of the Xilinx IP cores which is used for the general-purpose input/output interface ([Ref. 6](#)). This block has two channels that can be customized to use either single channel and dual channel. Here in this design, it is used as a dual-channel for push buttons and slide switches.

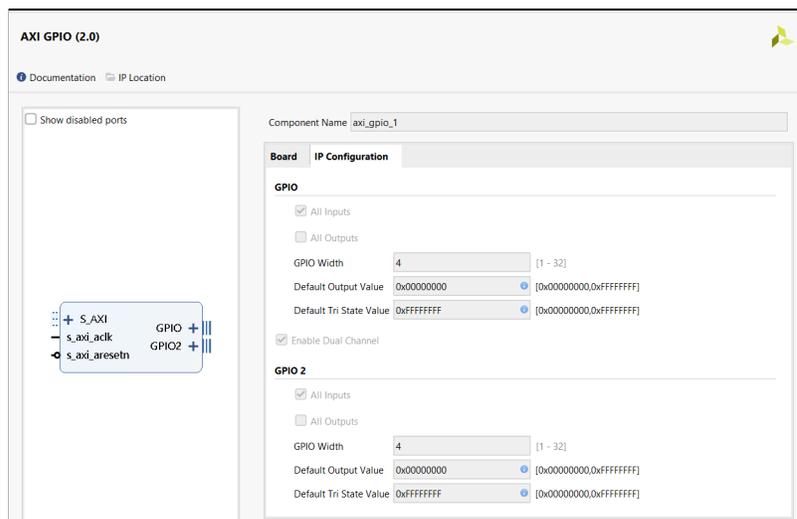


Figure 21 AXI GPIO customization window

The above figure shows the customization window of the AXI GPIO IP. Here the GPIO is used in dual channels and channel 1 is for push buttons and channel 2 is for the slide switch. The width for both GPIO

channels is 4 because there are 4 slide switches and 4 push buttons used and both channels are used for all inputs.

## 10. RGB2VGA IP

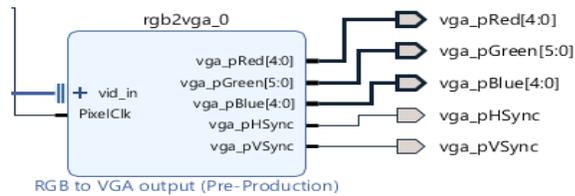


Figure 22 RGB2VGA IP

The above figure shows the RGB2VGA IP core. It is also one of the Digilent IPs which is added from the IP repository. This IP is used to obtain the output through the VGA port in the ZYBO Board. The **vid\_in** pin is connected to the **vid\_io\_out** pin of the AXI4-Stream To Video Out.

## 11. Other IPs

Except for the above IPs, the block design also consists of the following IPs.

### a. Processor System Reset IP

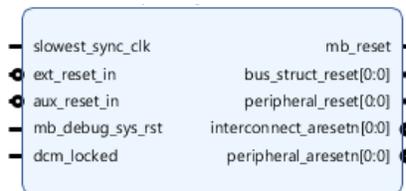


Figure 23 Processor System Reset

The Processing System Reset IP provides a mechanism to handle the synchronous reset conditions for a given designed system in the vivado. This IP core handles numerous reset conditions at the input and generates appropriate resets at the output. This core generates the reset based upon external or internal reset conditions. Here the **ext\_reset\_in** pin is connected to the **FCLK\_RESET0\_N** pin of the Zynq processing system. The **slowest\_sync\_clk** pin is connected to the clocking wizard output clock.

## b. Constant IP

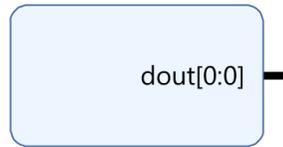


Figure 24 Constant IP Core

Constant IP is used to enable the clk pin in the Video Timing Controller and AXI-4 Stream To Video Out IP blocks. Here the constant value is chosen as 1 and is used to connect the pin **clken**, **s\_axi\_clken**, **aclken**, and **vid\_io\_out\_ce** pins. One was used for the **hdmi\_hpd** and the other was used for the **hdmi\_out\_en**. The hpd stands for Hot Plug Detect. **hdmi\_hpd** is the feature of the HDMI which on single contact to the connector initiates the process. It helps to determine whether the HDMI data is streamed or not in the ZYBO board. **hdmi\_out\_en** determines whether the HDMI port is used for input or output. If the value 0 is given to the **hdmi\_out\_en** pin then the HDMI port works as input and if value 1 is fed to the **hdmi\_out\_en** then the HDMI port works as output.

## c. AXI Interconnect IP

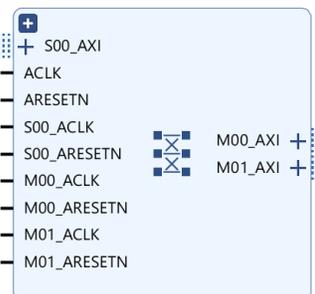


Figure 25 AXI Interconnect IP core

The AXI interconnect is used for connecting the slave and master of IP blocks such as Test Pattern Generator, Video Mixer and Video Timing Controller to the Zynq processor system.

## d. AXI Smart Connect IP

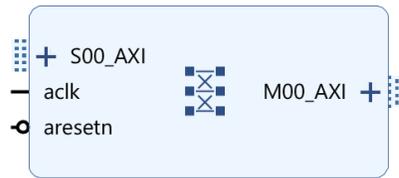


Figure 25 AXI Smart Connect

The above figure shows the AXI SmartConnect IP core which is used to connect the memory-mapped layer pin of the video mixer to the Zynq processor system. The master pin of this IP is connected to the high-performance slave pin of the Zynq processor system and the slave pin is connected to the memory-mapped layer of the video mixer.

## 12. Final Block Design

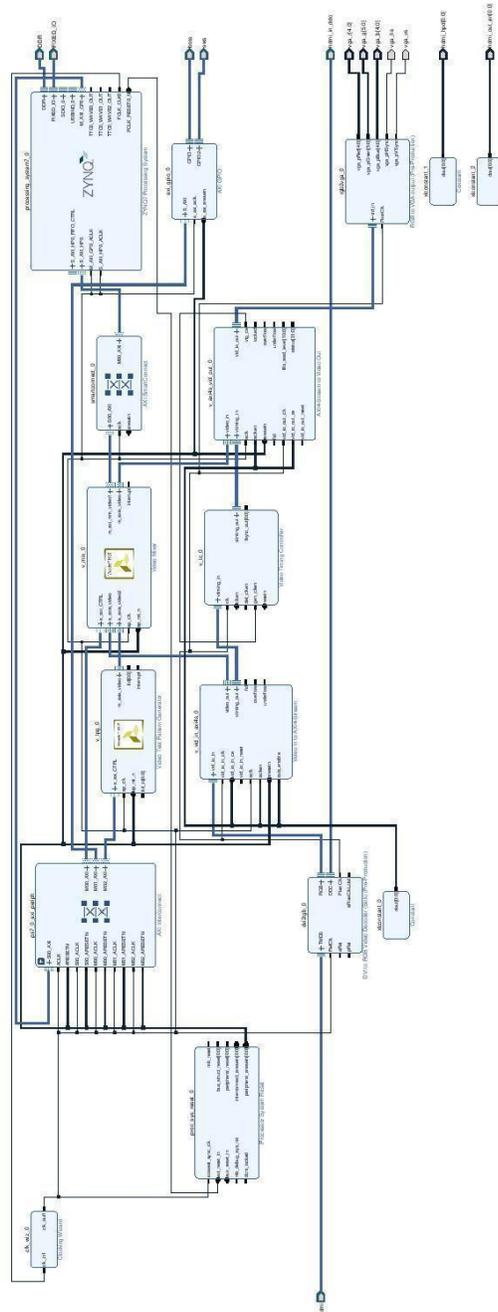


Figure 26 Complete Block Design.

The above figure is the complete block design of using push buttons and slide switches for using video mixer design. Here the slide switch and push buttons are used for input to enable and disable the features of the video mixer and the output is shown from the VGA port. Here the clocking wizard provides the 148.5 MHz clocking frequency which is supplied to all AXI clocks of the IP. The pixel clock from the **dvi2rgb** is fed to the **vid\_io\_clk** pin of **Video In to AXI4-Stream** and **AXI4-Stream To Video Out**, to the clock frequency of the video timing controller and also to the pixel clock of the **RGB2VGA** IP block. The **Video Timing Controller** IP is set to both generation and detection so that it can detect the timing of the inputted video and generate the timing same as the inputted video. Here both the **DVI2RGB** and **VTC IP** are configured to work with 1080p resolution. Then one **V\_TPG\_0** is used as stream TPG and one memory-mapped layer is used as an overlay layer for the HDMI video which is fed to the **s\_axis\_video** pin of the Video Mixer. In the video mixer, there was the use of the memory interface type as Layer 1 and stream type interface for the video stream which is Layer 2 with both scalings enabled. Then the **mm\_axis\_video** pin of the video mixer is passed to the **AXI4-Stream To Video Out** IP **video\_in** pin. In which **VTC** IP timing is also passed and the output is then passed to the **RGB2VGA** IP from which the output is passed through the monitor. The **AXI GPIO** is used for the input and in which a dual-channel is used, one for the push button and the other for the slide switch.

### 13. Utilization Report

Resource	Utilization	Available	Utilization %
<b>LUT</b>	11468	17600	65.16
<b>LUTBRAM</b>	858	6000	14.30
<b>FF</b>	15135	35200	43.00
<b>BRAM</b>	11.	60	18.33
<b>DSP</b>	12	80	15.00
<b>IO</b>	38	100	38.00
<b>BUFG</b>	5	32	15.63
<b>MMCM</b>	2	2	100.00

The above table shows the utilization of board resources by the current video mixer design.

## 14. IP Connection

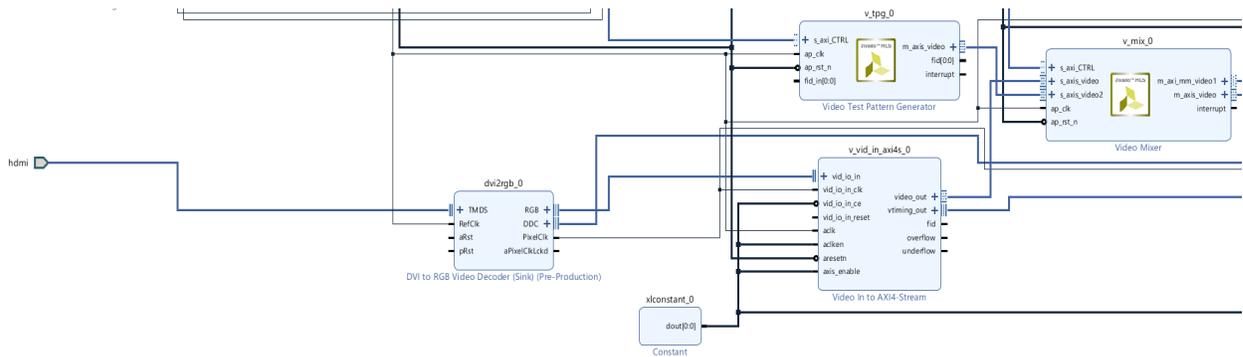


Figure 27 IP connection of Video Input

The above figure depicts the IP connection for the inputting video through the HDMI port. Here the **dvi2rgb** IP is used for inputting the HDMI data. The **TMDS** pin is made external, which is configured in the constraint and then the **RGB** pin is connected to the **vid\_io\_in** pin of the **Video In to AXI4-Stream** IP. The **Video In to AXI4-Stream** is selected with independent mode and the **vid\_io\_in\_clk** clock is connected to the **PixelClk** of the **dvi2rgb** IP. Constant IP is used for enabling the clocks. The Video out from this IP is connected to the **s\_axi\_video** pin of the video mixer and it is fed into a master layer. Also, the **vtiming\_out** pin is connected to the **VTC** which detects the timing from the IP. The **TPG** is connected to the stream layer which is overlaid on the master layer. The TPG **s\_axis\_CTRL** is connected to the AXI interconnect master axis. The **m\_axis\_video** pin of the **v\_tpg\_0** is connected to the **s\_axis\_video2** pin of the video mixer. The video mixer **s\_axis\_CTRL** is also connected to the master axis of the **AXI interconnect**. The **AXI interconnect** is connected to the Zynq processing system. The **m\_axi\_video** pin of the video mixer is connected to the **video\_in** pin of the **AXI Stream to Video Out** IP block.

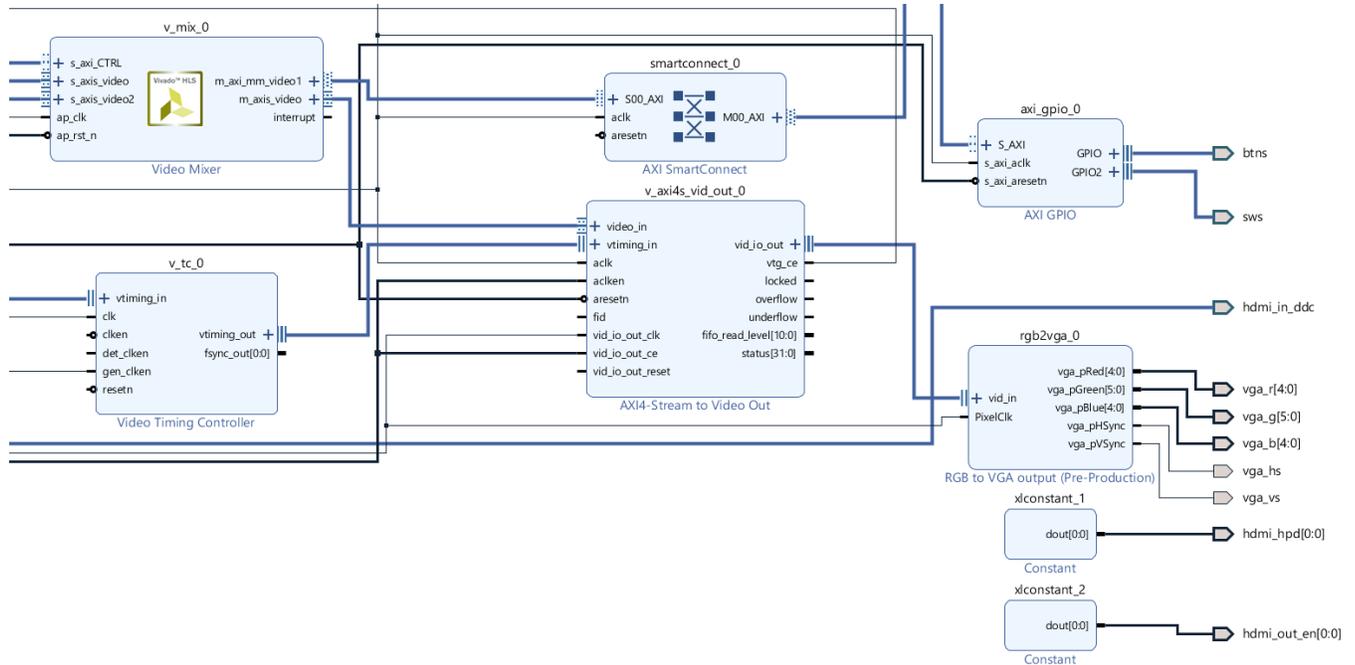


Figure 28 Video Output IP connection

The above figure shows the video mixer IP connection. Here the video out from the video mixer is connected to the **video\_in** pin of the **AXI-4 Stream to Video Out** IP block. The memory-mapped layer pin **m\_axis\_mm\_video1** is connected to the **AXI Smart connect** slave pin and the AXI Smart connect master pin is connected to the high-performance slave PIN of the Zynq processor system. The Video Timing Controller **v\_timings\_in** pin is connected to the **Video In To AXI4-Stream** IP’s **v\_timing\_out** pin and **v\_timing\_out** pin is connected to the **AXI4-Stream To Video Out** IP **v\_timings\_in** pin. The **m\_axis\_video** is connected to the **video\_in** pin which gives the video mixer video output and then the **vid\_io\_out** pin is connected to the **vid\_in** pin of the **RGB2VGA** IP. Output is shown from the **RGB2VGA** IP by configuring the external pin in the constraint file. The constants are used for **hdmi\_hpd** and **hdmi\_out\_en**. The constant IP is used for the enable of the clock such as **clken**, **s\_axis\_clken** pin of the **VTC** and **aclken**, **vid\_io\_out\_ce** pin of the **AXI4-Stream to Video Out** IP block. The **vtg\_ce** pin of the **AXI4-Stream to Video Out** IP is connected to the **gen\_clken** pin of the **Video Timing Controller**. The **axi\_gpio\_0** IP block enables dual channels and they are made external, such as **btns** for buttons and **sws** for switch.

## 15. Bitstream Generation

After completing the block design and auto assigning the address, the validation of the block design is done. After successful validation, the bitstream is generated. Before the generation of the bitstream, the synthesis and the implementation must be completed without any errors. Otherwise, the design has to be rechecked and corrected for errors.

Steps to proceed generation of the bitstream,

**Flow Navigator>Program and Debug>Generate Bitstream**

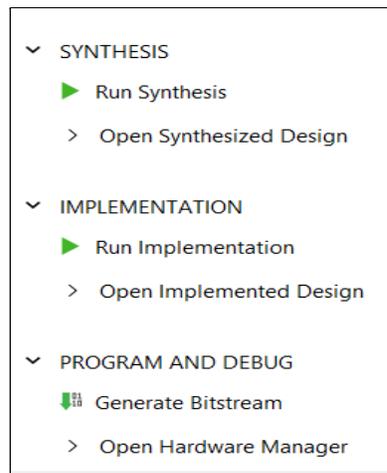


Figure 29 Bitstream Generation

## 16. Exporting Hardware

After the completion of the generation of the bitstream, the hardware is exported from the Vivado in .xsa file.

Steps to export the hardware,

**File > Export Menu >Export Hardware.**

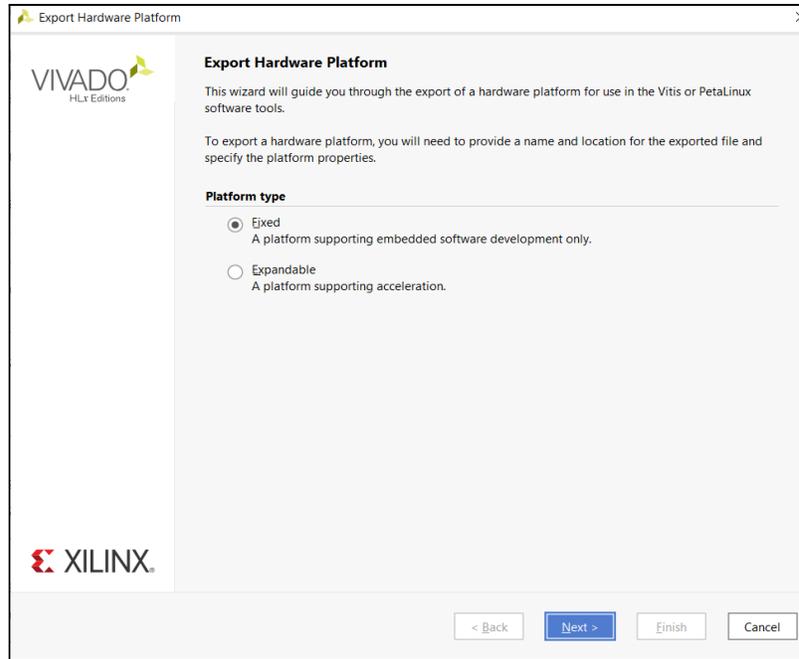


Figure 30 Exporting the Hardware

After that, the Vitis IDE is launched either by opening the **Vitis IDE** from the desktop or from Vivado menu **Tools > launch Vitis IDE**.

## Software Design

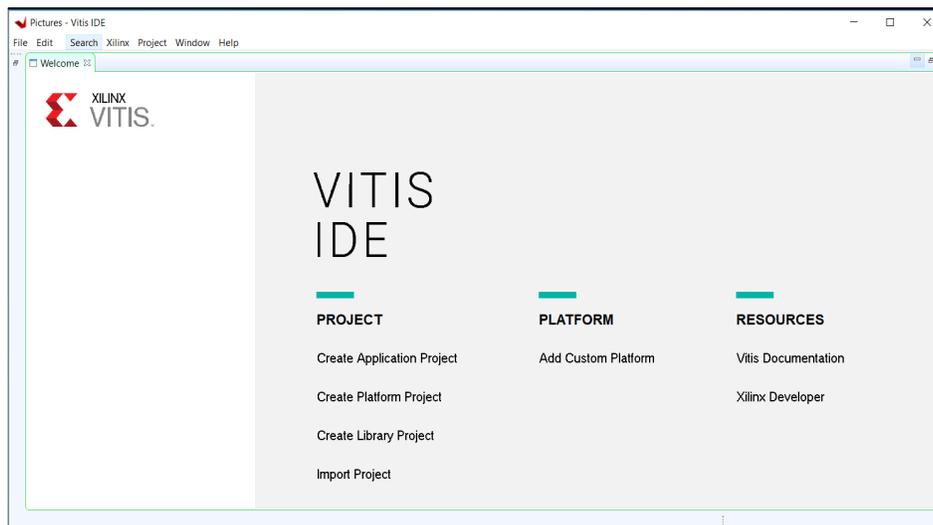


Figure 31 Vitis IDE

The above figure shows the Vitis IDE main window. In Vitis IDE, to create a software application, a platform project and application project have to be created. The application can be created by either creating a **platform project** first and then creating an **application project** one after another or, creating an **application project** first, where Vitis IDE automatically creates a **platform project** by allowing a user to specify a **.xsa** file.

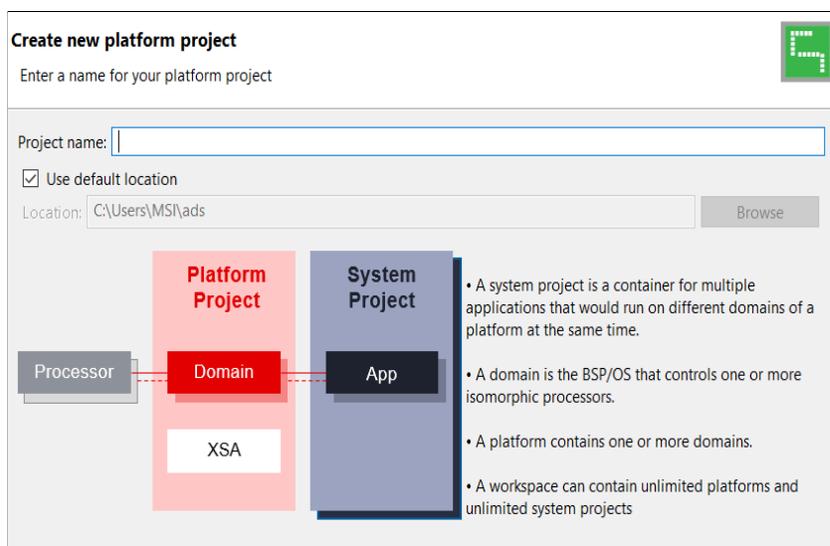


Figure 32: Platform Project

The above figure shows creating a **platform project**.

Steps to follow

### File > New > Platform Project

The creation of a platform project begins with giving the **project name** and going **Next** to select the platform.

Here, go to **Create a new platform from hardware (XSA)** tab. Go to **Browse** to locate the vivado exported **.xsa** file and click **Finish**. Vitis will proceed to create a platform project.

After the platform project, head for creating an application project.

Steps to follow

### File > New > Application Project

After following the steps **Create a New Application Project** window is opened, which can be ignored by clicking **Next**. This will open the **Platform** window. Out of two tabs, go to **Select a platform from repository**, where select previously created platform project and go to **Next**. The following figure shown in the dialog window will appear after that.

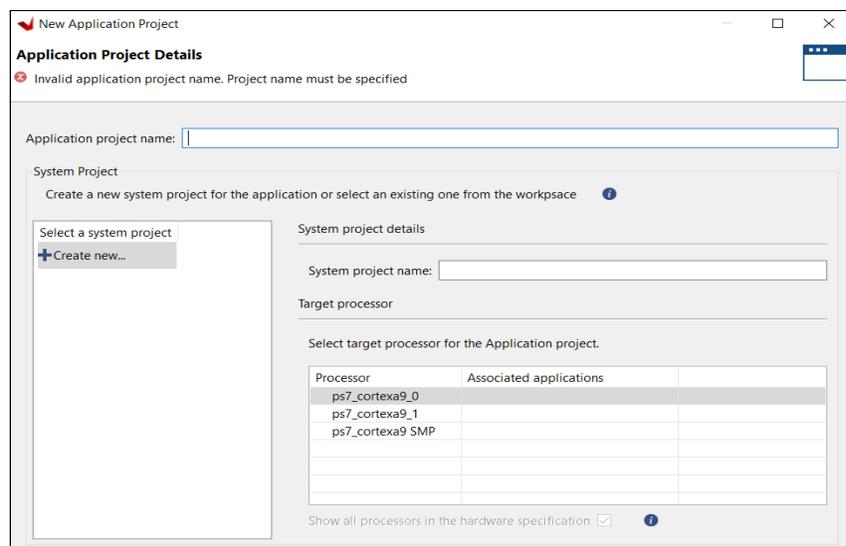


Figure 33 Application Project

Give **Application Project Name** and click **Next**.

**Domain** dialog window is opened, which can be ignored and go to **Next**.

A **Template** selection window is opened.

Select **hello world** template this will now open an application system in the **Explorer** pane on left of Vitis IDE.

We create the following **main.c** and **red.c** file to start coding. The **main.c** will have the main source code to run the video mixer design and **red.c** will have red GUI hex code for overlaying the memory-mapped layer in the video mixer.

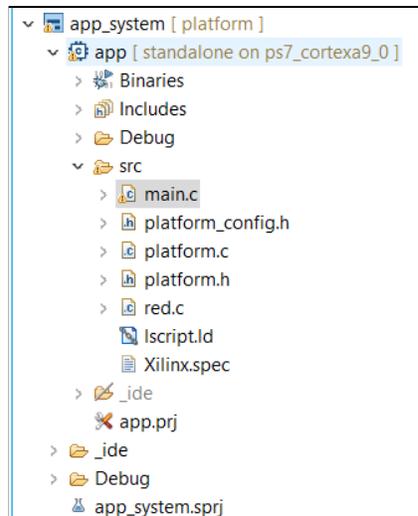


Figure 34 Application system and sources

Below is the snippet code of the Vitis IDE application project.

```

#include "platform.h"
#include "sleep.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xv_tpg.h"
#include "xv_mix_l2.h"
#include "xvidc.h"
#include "xvtc.h"
#include "xgpio.h"
    
```

Figure 35 Header File

The above figure is the header file for the use of the devices in the block diagram.

Here the *xv\_tpg.h* is the header file of the video test pattern generator device that is used for the use of the TPG function.

*xv\_mixl2.h* is a layer 2 header file of the video mixer. This header file provides a higher level of the implementation of features and additional functions such as more 8 layers configuration, scaling per layer, alpha per layer.

*xvidc.h* is the header file that is used for the function for the video related application and driver.

*xvvc.h* is the header file for the video timing controller. VTC detects/generates the video timing signal. It provides timing features such as hsync, vsync, vblank, hblank and active video. And this header file enables the use of these features.

```
extern u32          red_gui[];
unsigned int srcBuffer = (XPAR_PS7_DDR_0_S_AXI_BASEADDR + 0x100000);
#define XVMIX_LAYER1_BASEADDR      (XPAR_PS7_DDR_0_S_AXI_BASEADDR + (0x10000000))
#define XVMIX_LAYER_ADDR_OFFSET   (0x01000000U)
#define XVMIX_CHROMA_ADDR_OFFSET  (0x01000000U)
```

*Figure 36 Memory Address Assigned*

The above picture depicts the assignment or definition of the address for the memory layers. Here a base address (XVIMX\_LAYER1\_BASEADDR) is assigned for layer 1 and the additional layer will be addressed by adding the base address with the *LAYER\_ADDR\_OFFSET* address. The chroma address is also assigned when the color format is selected for the YUV color format for memory address. The *srcBuffer* is defined after 0x100000 offset because the PS7\_DDR memory locations up to 0x00100000 are allocated for the configuration files and if these locations are used for other memory usages, it can cause issues to run the design. So, we add a 0x100000 offset value to the PS7\_DDR base address. For the layer base address, the value 0x10000000 value is added to the base address of the PS7\_DDR. And the value to be added is chosen such that it does not exceed the high memory value of the board.

```
/* Definitions for peripheral PS7_DDR_0 */  
#define XPAR_PS7_DDR_0_S_AXI_BASEADDR 0x00100000  
#define XPAR_PS7_DDR_0_S_AXI_HIGHADDR 0x1FFFFFFF
```

Figure 37 ZYBO base and high memory address

The above Figure shows the base address and the highest address for the PS7\_DDR. Here the `XPAR_PS7_DDR_0_S_AXI_HIGHADD 0x1FFFFFFF` defines the highest memory for the Zybo board and if the memory address exceeds this value then the error is generated. It is very important for knowing the memory address of the board for memory-mapped coding

```
XV_Mix_l2 mix;  
XV_tpg strmtpg;  
XVidC_VideoStream VidStream;  
XVidC_ColorFormat Cfmt;  
XVidC_VideoTiming const *TimingPtr;  
XGpio gpio;
```

Figure 38 Declaration of Variable

Above figure shows the declaration of the instance variables that such as `XV_tpg strmtpg` which is used for the configuration of the stream TPG which is defined in the `xv_tpg.h` header file, `XV_Mix_l2 mix`, `XVtc vtc` are used to access the functions of the video mixer and VTC header files respectively.

`Vidstream` is the instance variable for `XVidC_VideoStream`. The color format of the video is stored in the `Cfmt` variable. Pointer is used for video timing because the pointer stores the address of another variable or data rather than the data itself. Timing pointer is used because in the timing struct, there are variables such as `vblank`, `hblank`, `vsync`, `hsync`, `active video` and the other that provides data at a constant rate so the pointer doesn't need large chunks of the memory. `XGpio` provides the function for the `gpio` usages.

```
XVidC_VideoMode videoModes[2] = { XVIDC_VM_720_60_P, XVIDC_VM_1080_60_P };
```

The above code snippet is the use of the features in the video stream which is the video modes structure. These are recently added functions that provide the ease of use of the different modes of the video format which is 480, 720, 1080, 1440 and higher resolution formats. But here there is the use of the two modes which are 720 and 1080 with the frame rate of 60 Hz. These resolutions are selected because of the limitation of the hardware resources.

```
static const XVidC_VideoWindow MixLayerConfig[4] = {
    { 12, 10, 128, 128 },
    { 600, 10, 128, 128 },
    { 900, 10, 128, 128 },
    { 1500, 10, 128, 128 }
};
```

Figure 39 Layers Window

The above figure shows the use of another struct of the *XVidC\_VideoStream*. Here we use the *XVidC\_VideoWindow* struct. This struct has X, Y, Width and height of the video frame. This struct was used to create different windows for the video mixer layers. Here this struct was used to make a four-layer window. X and Y position is used to place the video window in the monitor frame and the height and width give the size of the window.

```
static int DriverInit(void);
static void Configstrmtpg(XVidC_VideoStream *StreamPtr);
static void ConfigMixer(XVidC_VideoStream *StreamPtr);
static int RunMixer(XVidC_VideoStream *StreamPtr);
```

Figure 40 Function Declaration

The above figure shows the declaration of the function definitions. For each configuration of the devices in the design, each function was created; the *DriveInit (void)* function was used for the initialization and status of the device in the design. It helps in knowing the success of the initialization of the device in the hardware. These functions are called from the *main function*.

```

int main(void)
{
    int Status;
    init_platform();
    xil_printf("Start Mixer Design\r\n");
    Status = DriverInit();
    if (Status != XST_SUCCESS) {
        xil_printf("ERROR:: Driver Init. Failed\r\n");
        xil_printf("ERROR:: Test could not be completed\r\n");
        while (1);
    }
    XVMix_GetLayerColorFormat(&mix, XVMIX_LAYER_MASTER, &Cfmt);
    VidStream.ColorFormatId = Cfmt;
    VidStream.VmId = videoModes[1];
    TimingPtr = XVidC_GetTimingInfo(VidStream.VmId);
    VidStream.Timing = *TimingPtr;
    VidStream.FrameRate = XVidC_GetFrameRate(VidStream.VmId);

    xil_printf("\r\n*****\r\n");
    xil_printf("Test Input Stream: %s (%s)\r\n",
              XVidC_GetVideoModeStr(VidStream.VmId),
              XVidC_GetColorFormatStr(VidStream.ColorFormatId));
    xil_printf("*****\r\n");

    Configstrmtpg(&VidStream);
    ConfigMixer(&VidStream);
    RunMixer(&VidStream);
}
    
```

Figure 41 Main Function

The above figure shows the main function of the code. Here at first, the driver initialization function was run for initializing the device function in the initial state. And also the video mode is set and the framerate is determined through the video mode . At last, the functions are called to run each configuration and the mixer test.

```

static void Configstrmtpg(XVidC_VideoStream *StreamPtr)
{
    XV_tpg_DisableAutoRestart(&strmtpg);
    XV_tpg_Set_height(&strmtpg, 128);
    XV_tpg_Set_width(&strmtpg, 128);
    XV_tpg_Set_colorFormat(&strmtpg, StreamPtr->ColorFormatId);
    XV_tpg_Set_bckgndId(&strmtpg, XTPG_BKGND_COLOR_BARS);
    XV_tpg_Set_ovrlyId(&strmtpg, 0);
    XV_tpg_EnableAutoRestart(&strmtpg);
    XV_tpg_Start(&strmtpg);
    xil_printf("INFO: stream tpg configured\r\n");
}
    
```

Figure 42 TPG configuration

The above figure shows the configuration of the stream TPG. Here at first, *autorestart* was disabled for the configuration of the TPG. Then the *height* and *width* for stream TPG were set. For the *strmtpg*, the width

and height were set to 128 respectively, which were the same as the layer window. Then the *colorformat*, *background patterns* and the *overlay ID* were also set for the TPG.

```

static void ConfigMixer(XVidC_VideoStream *StreamPtr)
{
    u32 memaddr;
    XV_Mix_12 *MixerPtr = &mix;
    XVMix_LayerDisable(MixerPtr, XVMIX_LAYER_MASTER);
    XVMix_SetVidStream(MixerPtr, StreamPtr);
    memaddr = XVMIX_LAYER1_BASEADDR;
    XVMix_SetLayerBufferAddr(MixerPtr, XVMIX_LAYER_1, memaddr);
    Xil_DCacheDisable();
    memcpy((UINTPTR *)MixerPtr->Layer[1].BufAddr, &red_gui, 32768 * sizeof(u32));
    Xil_DCacheEnable();
    XVMix_LayerEnable(MixerPtr, XVMIX_LAYER_MASTER);
    XVMix_InterruptDisable(MixerPtr);
    XVMix_Start(MixerPtr);
    xil_printf("INFO: Mixer configured\r\n");
}
    
```

Figure 43 Mixer Configuration

The above figure shows the configuration of the Video Mixer. There is the use of the stream interface and memory-mapped interface types. Here in this function, the master layer was disabled first for the configuration of the mixer and then the parameters from the video stream are passed to the mixer pointer. Then the **memaddr** value is set for layer 1 buffer address.

Here the function *Xil\_DCacheDisable()* disables the cache memory and we need to first disable the cache memory because we are writing on the RAM and after the writing the cache is enabled because the cache is also used by other functions.

```

.....
memcpy((UINTPTR *)MixerPtr->Layer[1].BufAddr, &red_gui, 32768 * sizeof(u32));
    
```

The above snippet of code is used for the memory-mapped layer. Here *memcpy* copies the *red\_gui* value to the layer 1 buffer address and *red\_gui* is the array for the red color whose value is in RGBX8 color format. And 32768 is the total pixel value of the *red\_gui* width x height i.e. 128x256. The interrupt was also disabled so that the video mixer is restarted automatically. After these, the video mixer is configured and started.

```
void printMenu()
{
    xil_printf("+-----+\n\r");
    xil_printf("|           Video Mixer Demonstration           |\n\r");
    xil_printf("+-----+\n\r");
    xil_printf("| Slide Switch 1- Enable Layer 1                 |\n\r");
    xil_printf("| Slide Switch 2- Enable Layer 2                 |\n\r");
    xil_printf("| Slide Switch 3- Disable Layer 1                |\n\r");
    xil_printf("| Slide Switch 4- Disable Layer 2                |\n\r");
    xil_printf("| Button 1- Scale 2x Layer 1                     |\n\r");
    xil_printf("| Button 2- Scale 2x Layer 2                     |\n\r");
    xil_printf("| Slide Switch 1 and 2- Scale back Layer 1 and Layer 2 |\n\r");
    xil_printf("| Button 3- Move Layer 1                         |\n\r");
    xil_printf("| Button 4- Move Layer 2                         |\n\r");
    xil_printf("+-----+\n\r");
    xil_printf("+-----+\n\r");
}
```

Figure 44 Main Menu Print

The above figure shows the function for displaying the main menu in the Terminal message. By observing the message in the terminal, input is fed to the device through the push buttons and slide switches.

```

static int RunMixer(XVidC_VideoStream *StreamPtr)
{
    XVidC_ColorFormat Cfmt;
    XV_Mix_l2 *MixerPtr = &mix;
    int btns=0,sws=0;
    int step1,step2;
    XVMix_LayerEnable(MixerPtr, XVMIX_LAYER_MASTER);
    XGpio_SetDataDirection(&gpio, 1, 1);
    XGpio_SetDataDirection(&gpio, 2, 1);
    printMenu();
    while(1)
    {
        btns = XGpio_DiscreteRead(&gpio, 1);
        sws = XGpio_DiscreteRead(&gpio, 2);
        if (sws == 1)
        {
            if(!XVMix_IsLayerEnabled(MixerPtr,XVMIX_LAYER_1))
            {
                XVMix_GetLayerColorFormat(MixerPtr, XVMIX_LAYER_1, &Cfmt);
                step1 = XVMix_LayerEnable(MixerPtr, XVMIX_LAYER_1);
                step2 = XVMix_SetLayerWindow(MixerPtr, XVMIX_LAYER_1, &MixLayerConfig[0], 512);
                //step3 = XVMix_SetLayerAlpha(MixerPtr, XVMIX_LAYER_1, 100);
                if(step1!=XST_SUCCESS || step2!=XST_SUCCESS )
                {
                    XVMix_LayerDisable(MixerPtr, XVMIX_LAYER_1);
                }
            }
        }
        if (sws == 2)
        {
            if(!XVMix_IsLayerEnabled(MixerPtr,XVMIX_LAYER_2))
            {
                XVMix_GetLayerColorFormat(MixerPtr, XVMIX_LAYER_2, &Cfmt);
                step1 = XVMix_LayerEnable(MixerPtr, XVMIX_LAYER_2);
                step2 = XVMix_SetLayerWindow(MixerPtr, XVMIX_LAYER_2, &MixLayerConfig[1], 512);

                if(step1!=XST_SUCCESS || step2!=XST_SUCCESS )
                {
                    XVMix_LayerDisable(MixerPtr, XVMIX_LAYER_2);
                }
            }
        }
    }
}

```

Figure 45 First Part of the RunMixer Function

The above figure shows the first half of the runmixer function. Here the *btns* and *sws* are taken as integer and the function *XGpio\_SetDataDirection (&gpio, 1, 1)* set to channel 1 as input and *XGpio\_SetDataDirection (&gpio,1,1)* set to channel 2 as input for the GPIO IP block. The *XGpio\_DiscreteRead (&gpio, 1)* is used so that the *btns* and *sws* can read the data inputted from the push buttons and slide switches. Here, if switch 1 is ON then layer 1 is enabled and if switch 2 is ON then layer 2 is enabled.

```

    }
    if (sws == 4)
    {
        XVMix_LayerDisable(MixerPtr, XVMIX_LAYER_1);
    }
    if (sws == 8)
    {
        XVMix_LayerDisable(MixerPtr, XVMIX_LAYER_2);
    }
    if (btns == 1)
    {
        XVMix_SetLayerScaleFactor(MixerPtr, XVMIX_LAYER_1, XVMIX_SCALE_FACTOR_2X);
    }
    if (btns == 2)
    {
        XVMix_SetLayerScaleFactor(MixerPtr, XVMIX_LAYER_2, XVMIX_SCALE_FACTOR_2X);
    }
    if (sws == 3)
    {
        XVMix_SetLayerScaleFactor(MixerPtr, XVMIX_LAYER_1, XVMIX_SCALE_FACTOR_1X);
        usleep(200000);
        XVMix_SetLayerScaleFactor(MixerPtr, XVMIX_LAYER_2, XVMIX_SCALE_FACTOR_1X);
    }
    if (btns == 4)
    {
        XVMix_SetLayerWindow(MixerPtr, XVMIX_LAYER_1, &MixLayerConfig[2], 512);
    }
    if (btns == 8)
    {
        XVMix_SetLayerWindow(MixerPtr, XVMIX_LAYER_2, &MixLayerConfig[3], 512);
    }
}

```

*Figure 46 Second part of RunMixer Function*

The above figure is the code for the second part of the runmixer function . Here if switch 3 is ON then layer 1 is disabled and if switch 4 is ON then layer 2 is disabled. The button 1 when pushed scales the layer 1 to 2x and button 2 scales the layer 2 to 2x. If both switch 1 and 2 is ON then both layers are scaled to 1x with a delay between them. Button 3 and 4 move the layer 1 and 2 to the right position which is shown in the output.

## Final Output

```

Start Mixer Design
*****
Test Input Stream: 1920x1080@60Hz (RGB)
*****
INFO: VTC configured
INFO: Mixer configured
INFO: Master and stream tpg configured
+-----+
|               Video Mixer Demonstration               |
+-----+
| Slide Switch 1- Enable Layer 1                       |
| Slide Switch 2- Enable Layer 2                       |
| Slide Switch 3- Disable Layer 1                     |
| Slide Switch 4- Disable Layer 2                     |
| Button 1- Scale 2x Layer 1                          |
| Button 2- Scale 2x Layer 2                          |
| Slide Switch 1 and 2- Scale back Layer 1 and Layer 2 |
| Button 3- Move Layer 1                              |
| Button 4- Move Layer 2                              |
+-----+
    
```

Figure 47 Terminal Message Menu

The above figure shows the output shown in the terminal message of the Xilinx Vitis. This is the menu that is printed in the terminal message and is used for navigating the function of different cases.

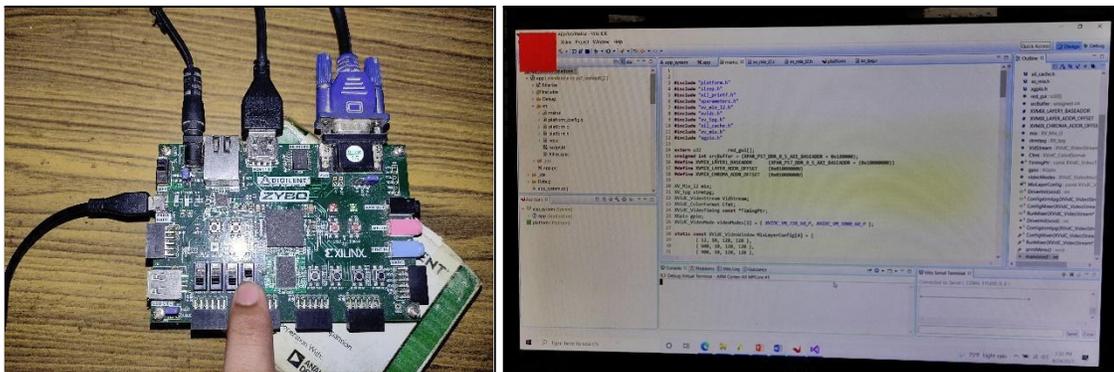


Figure 48 Switch 1 turned ON and output

The above figures show the output in which switch 1 is turned ON which enables layer 1.



Figure 49 Switch 2 turned ON and output

The above figures show the output in which switch 2 is turned ON which enables layer 2.

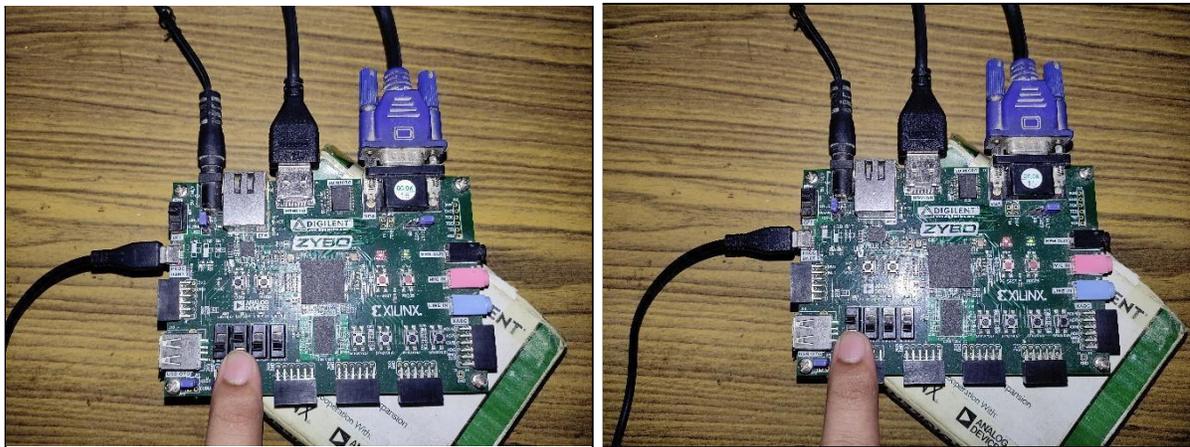


Figure 50 Turning switch 3 and switch 4

Turning switch 3 ON disables Layer 1 and turning switch 4 ON disables Layer 2 which is shown in the figure below.

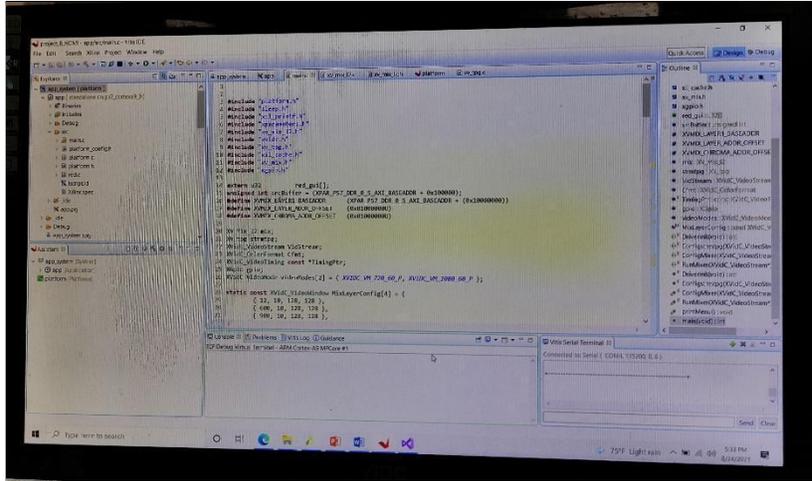


Figure 51 Both Layers disabled

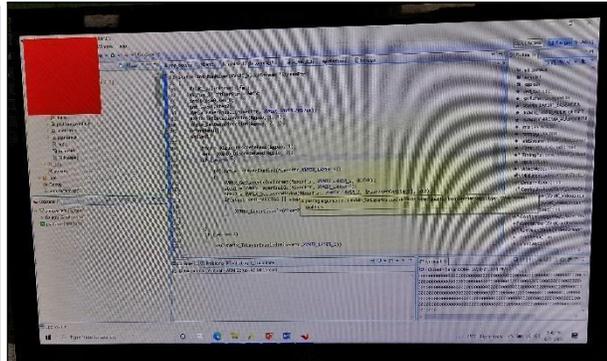


Figure 52 Scaled 2x Layer 1

The above figures show the output in which the layer 1 is scaled to 2x by pushing the button 1.

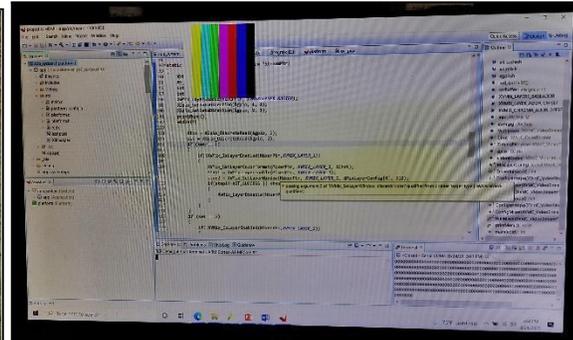


Figure 53 Scaled 2x Layer 2

Above figures show the output in which the layer 2 is scaled to 2x by pushing the button 2.



Figure 54 Scaled back to original size

The above figure is the output in which Layer 1 and Layer 2 are resized to their original size by turning on both slide switches 1 and 2 at once.

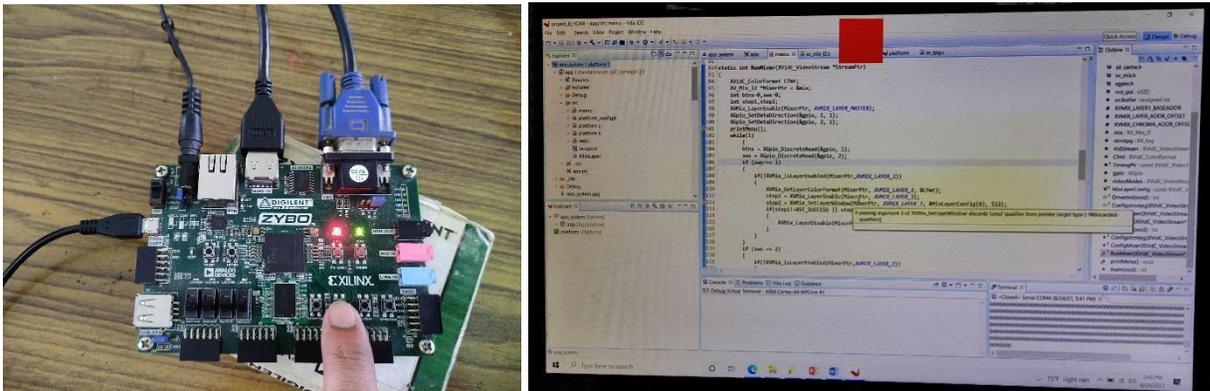


Figure 55 Moved Layer 1 to right position

The above figures show the output in which layer 1 is moved to the right position of the screen by pushing button 3.

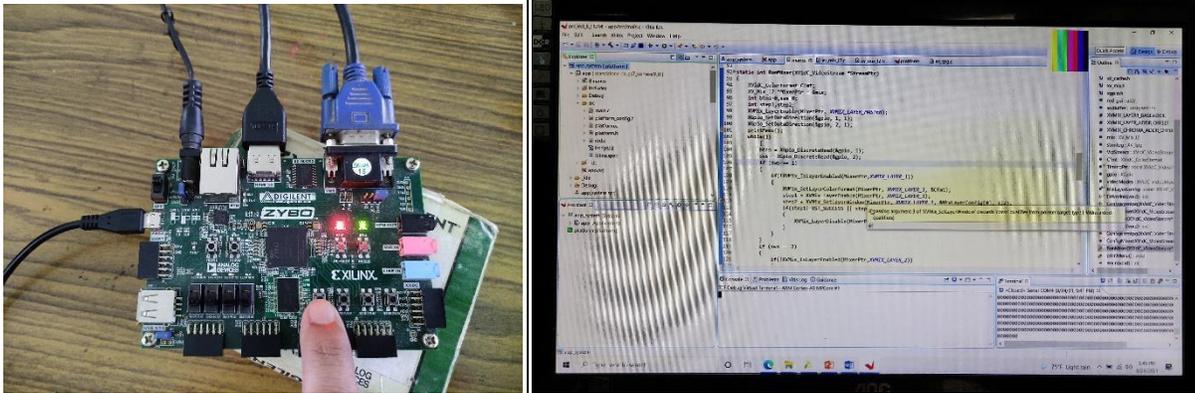


Figure 56 Moved Layer 2 to right position

The above figures show the output in which layer 2 is moved to the right position of the screen by pushing the button 4.

## References

1. Vivado Design Suite User Guide([ug973](#))
2. Zybo Reference Manual ([Link](#))
3. Video Mixer v3.0 LogiCORE IP Product Guide ([PG243](#))
4. Processing System 7 v5.5 LogiCORE IP Product Guide ([PG082](#))
5. Clocking Wizard v6.0 LogiCORE IP Product Guide ([PG065](#))
6. AXI GPIO v2.0 LogiCORE IP Product Guide ([PG144](#))
7. Video In to AXI4-Stream v4.0 LogiCORE IP Product Guide ([PG043](#))
8. Video Test Pattern Generator v8.0 LogiCORE IP Product Guide ([PG103](#))
9. AXI4-Stream to Video Out v4.0 LogiCORE IP Product Guide ([PG044](#))
10. Video Timing Controller v6.1 LogiCORE IP Product Guide ([PG016](#))

\*\*\*

Thank you for going through this “Reference Tutorial”!

**For any queries on “Video Mixture based Implementations” on Zynq UltraScale+ MPSoC or Zynq 7000, please contact us at: [info@logictronix.com](mailto:info@logictronix.com)**