Video Mixer IP (v3.0) Feature Implementation on Zynq FPGA

V2.0, June 2020

1. Overview

This project documentation is based on the video mixer design under Vivado 2018.3, implemented on Digilent Zybo Z7-10 FPGA board.

The Zybo Z7 is an FPGA board made up with the Xilinx Zynq-7000 FPGA device. The main feature of Zybo Z7 is on its rich set of multimedia and connectivity peripherals, so it is one of low cost best-suited FPGA boards for computer vision implementations. It has interfaces like MIPI CSI-2 compatible Pcam connector, HDMI input, HDMI output, and high DDR3L bandwidth.

The video mixer design constitutes hardware design and software design. In hardware design, video mixer IP, version 3.0, is used to mix different sorts of video streams over the master stream and then into a single output video stream. This IP has to be controlled from the Processing System (PS). Therefore, the hardware design consists of Zynq Processing System, which then requires software design. In the software design part, it is necessary to initialize the video mixer driver and other supportive drivers as well. After that, essentially, there requires to set different control register values to start working.

All the hardware design and software design will be expounded in the following document.

2. Video Mixer IP (v3.0)

Figure 1. Video Mixer IP v3.0

Video mixer IP is one of the Xilinx® LogiCORE™ IPs. It provides a flexible video processing block for alpha blending and compositing multiple video and graphics layers. This IP supports

for up to nine layers, that is, one master layer and eight overlay layers, with an optional logo layer, using a combination of video inputs from either frame buffer or streaming video cores, through AXI4-Stream interfaces, is provided. The IP core is programmable through a comprehensive register interface to control frame size, background color, layer position and the AXI4-Lite interface. IP also consists of a comprehensive set of interrupt status bits for processor monitoring [1].

The other features of IP are as following;

- 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 memory interface
- Supports semi-planar memory formats next to packed memory formats
- Supports spatial resolutions from 64 × 64 up to 4,096 × 2,160
- Supports 4K60 in all supported device families

3. Design Flow

This design flow step expounds about key hardware design steps in Vivado IP integrator and software design steps in Xilinx SDK.

3.1.Hardware Design Flow

3.1.1. Create New Project

Let's create a new project.

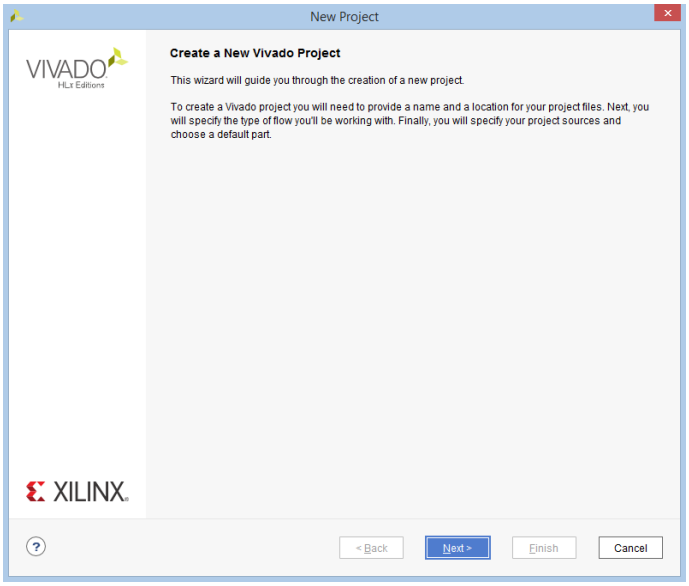


Figure 2. Project Create Dialog Box

After clicking next, we go through different series of dialog to give project name and its location directory and then adding block design and constraints files.

3.1.2. Board Selection

After following previous steps, we come to select the device to be implemented upon. User can choose either board parts or directly board. The best thing is to select the board other than part. Because, this allows us to access different board related presets while designing hardware block.

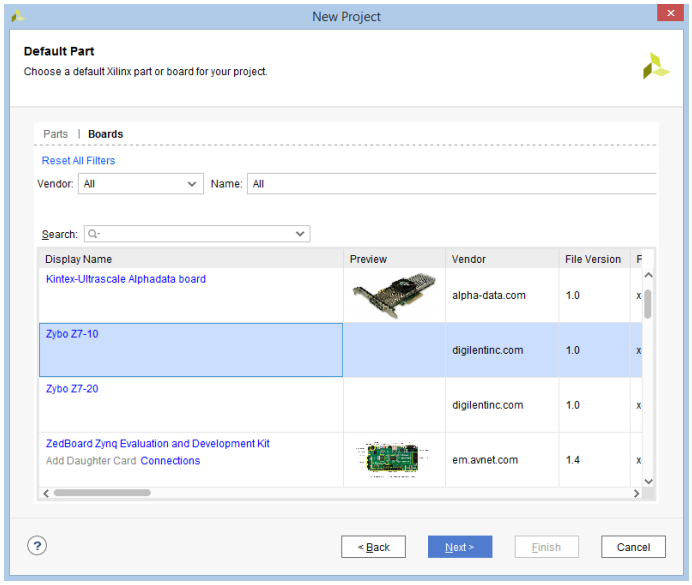


Figure 3. Board selection for project

User can select different boards to implement this project. However, this documentation is based on the zybo z7-10 board video mixer design, we currently choose Zybo Z7-10 board.

Clicking next shows another dialog that is related to new project summary. It displays about project name, block design, constraints file and finally, board related information such as type of board and its FPGA chip designation, product, family, package and speed grade.

3.1.3. Working with Vivado IP Integrator

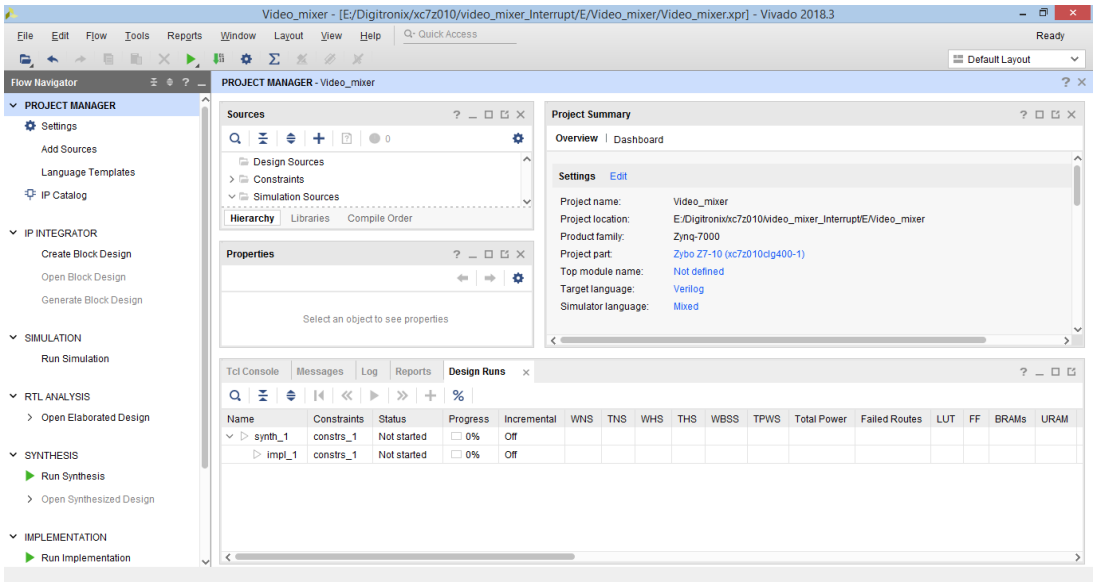


Figure 4. Vivado Window

After successful project creation, it opens new Vivado Window. Here, we start working with Vivado IP Integrator. First of all, user needs to create new block design.

User needs to follow the step:

Flow Navigator ⟩ IP Integrator ⟩ Create Block Design

This steps opens **Create Block Design** dialog as following

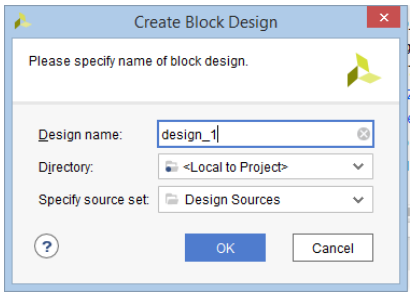


Figure 5. Create Block Design

We have to give the **Design Name**. We can give any name of it that designates the project. By default, the design name is set to **design_1**.

We can give **Directory** and **Specify source set**. However, these value are already set when we have specified the project directory in the beginning of project creation.

Click **OK** to open Vivado IP Integrator work space.

3.1.4. Adding IP Repository

In this step, we have to add IP repositories in our project. This is only required when we need to add IPs that are not available in Vivado IP catalog. That means to say that the Vivado has already some of Licensed Xilinx IPs. And if our block design has IPs which is not found in the Vivado IP catalog, then we need to add those missing IPs from adding **IP Repositories**.

Repositories can be added by following;

Flow Navigator ⟩ Project Manager ⟩ Setting

This opens Setting dialog.

Project Setting ⟩ IP ⟩ Repository

At right side sectioin, click "+" to go to add IP repo. You have to add the Digilent VIVADO Library IP repo in this step.

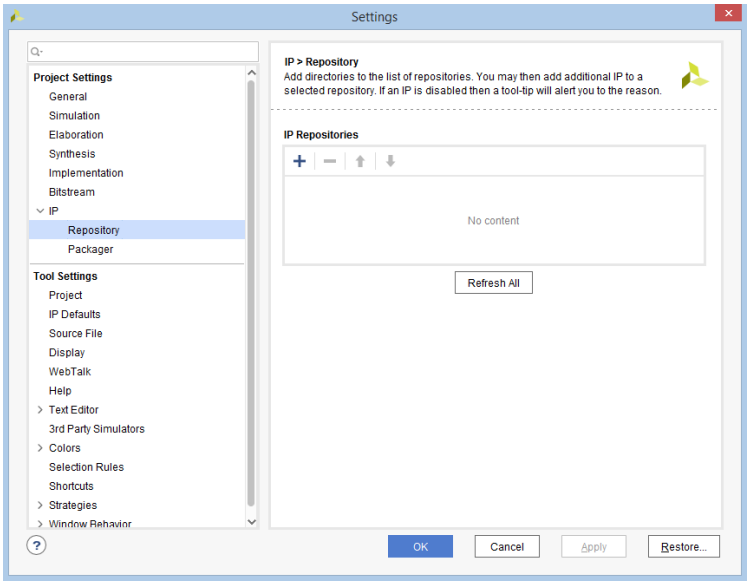


Figure 6. Add Repository

3.1.5. Creating block design

Now, we come to work with IP integrator. Here, we add necessary IPs and connect them as necessary to create logic circuit block design.

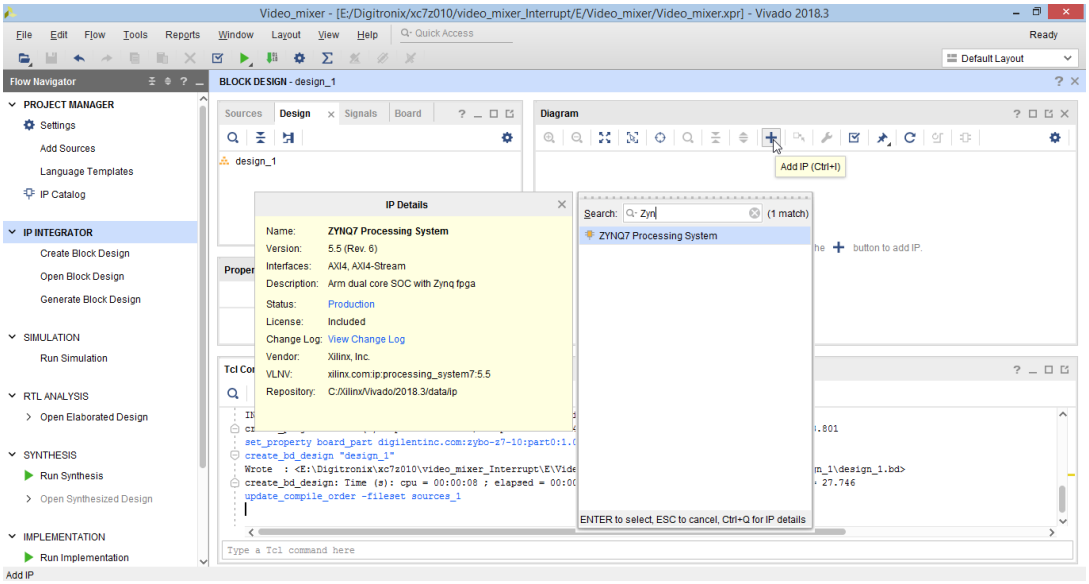


Figure 7. Adding IP to IP Integrator

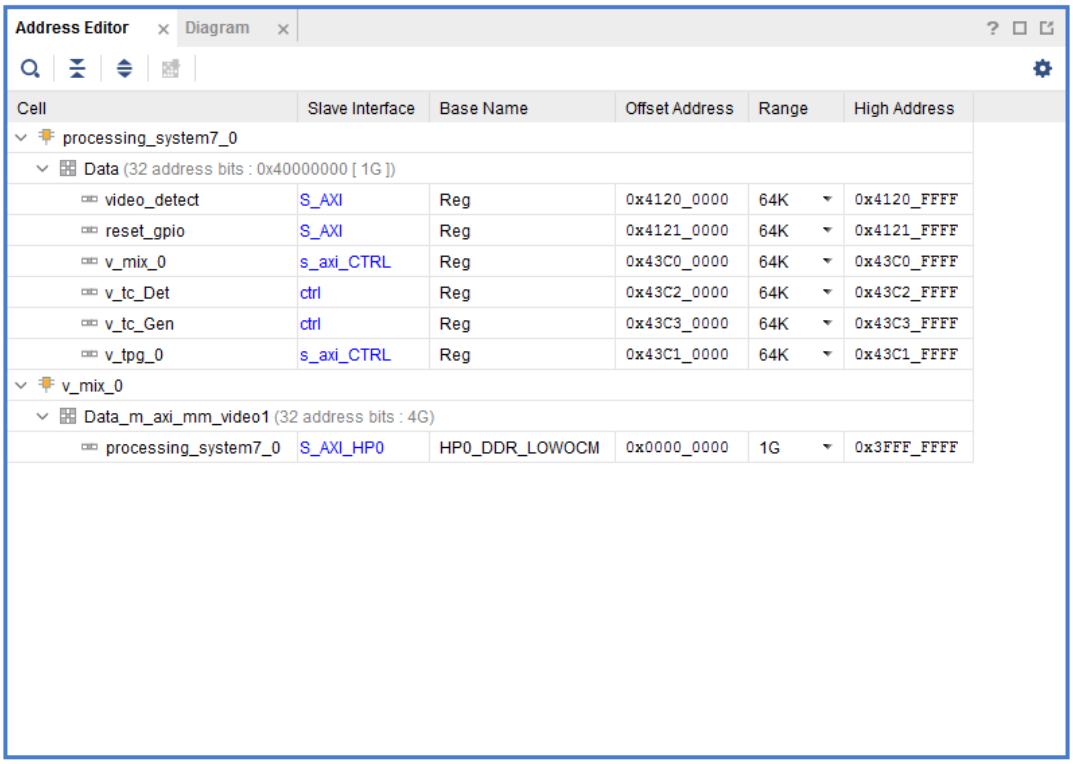
Next to **Flow Navigator** section, there is **BLOCK DESIGN** section with we given design name **design_1**. It has several sub sections. **Diagram** is one of its subsections. On diagram section top bar, there are several icons. Click ⍰ icon to **Add IP** in the block design work space. Or we can hit shortcut key **Ctrl+I** to add the IP.

It opens pop-up IP catalog box, where we can type and search the specific IP to add in our block design. This IP catalog displays IP only if it is present in the Vivado IP catalog. If it is not found, we can **Add Repository**. This part has been explained in step 4.

We can now add the all IPs that are required for our design. Note that all required IPs and their customizations and connections are explained in the hardware design section of this document.

3.1.6. Final Block Design

Following picture is of address editor which shows the addresses assigned for different AXI IPs.



Address Editor | Diagram

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
video_detect	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
reset_gpio	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
v_mix_0	s_axi_CTRL	Reg	0x43C0_0000	64K	0x43C0_FFFF
v_tc_Det	ctrl	Reg	0x43C2_0000	64K	0x43C2_FFFF
v_tc_Gen	ctrl	Reg	0x43C3_0000	64K	0x43C3_FFFF
v_tpg_0	s_axi_CTRL	Reg	0x43C1_0000	64K	0x43C1_FFFF
v_mix_0					
Data_m_axi_mm_video1 (32 address bits : 4G)					
processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	1G	0x3FFF_FFFF

Figure 8. Address Editor

The final block design is shown in following figure, while the details of each IP configuration is explained after the block design.

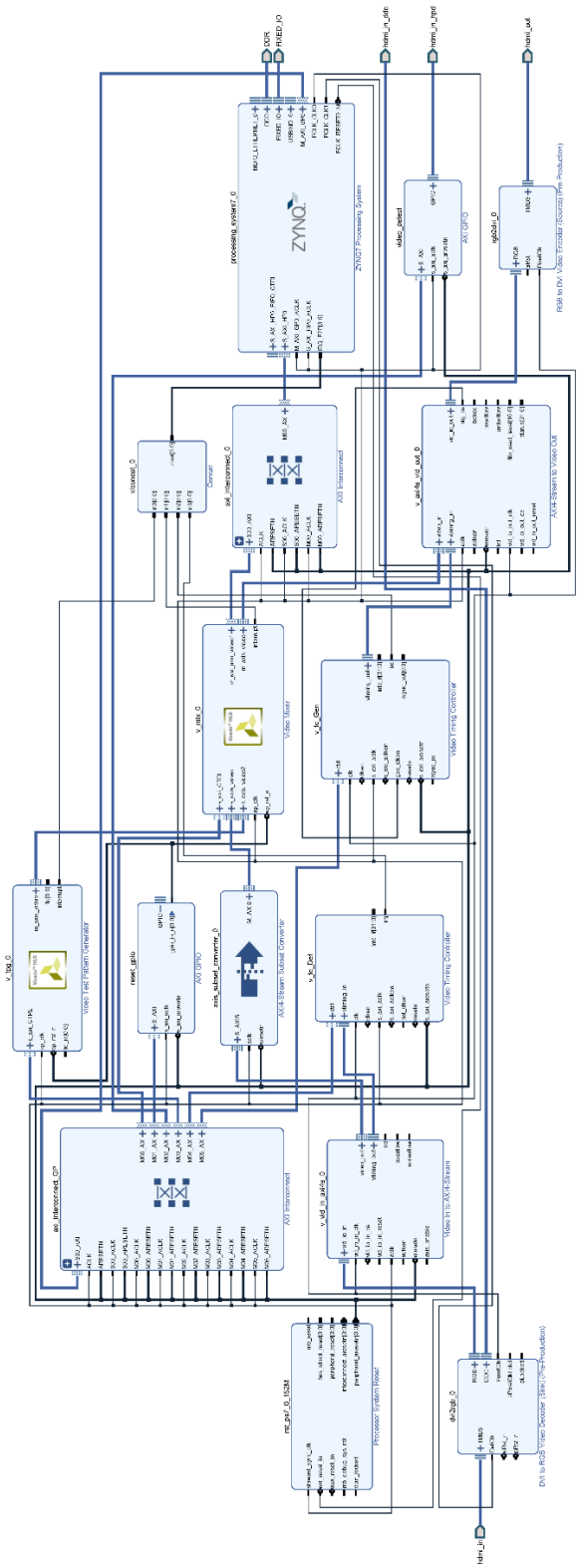


Figure 9. The Entire Block Design

3.1.7. Bitstream Generation

To proceed the bitstream generation, we can do following step;

Flow Navigator ⟩ PROGRAM & DEBUG ⟩ Generate Bitstream

This is the direct method of generating the bitstream, overriding the steps, for example, Synthesis and Implementation. However, Vivado takes care all of these.

3.1.8. Exporting the Hardware and Launching the SDK

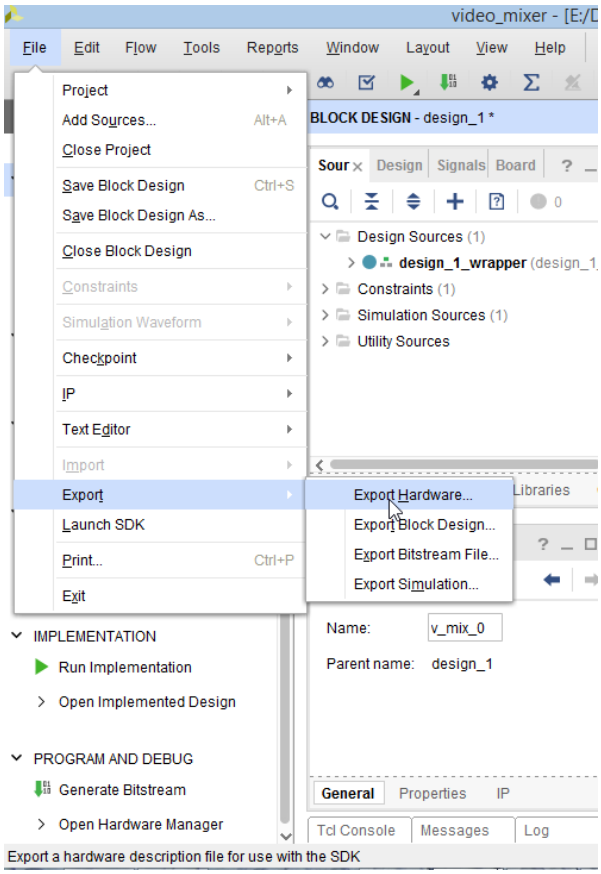


Figure 10. Hardware Design Export

After successful bitstream generation, we need to export our hardware design, including the corresponding bitstream, for the SDK part. If there is any already export hardware design, we can overwrite it.

Finally, it is time to access Xilinx SDK, by lunching it, by following.

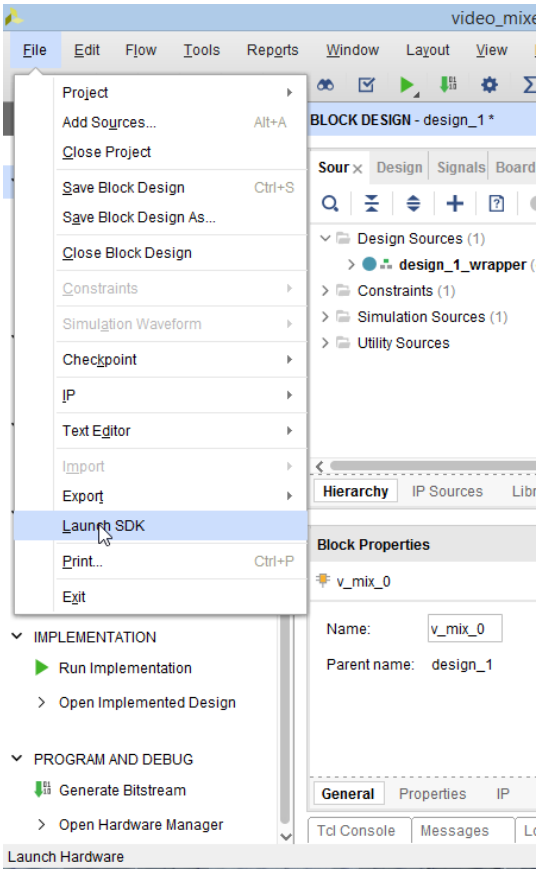


Figure 11. SDK Launching

3.2. IP details and connections

In the project, we have used different Xilinx Licensed Free IPs. The IPs that are used in the hardware design are detailed as follows.

3.2.1. Zynq Processing System(PS)

The PS in Zynq 7000 is dual-core ARM cortex A9 processor or CPU placed in the same FPGA chip along with the programmable logic(PL). This is the central processing system of the project. It provides the configuration and control of all IP drivers and hence the video processing. The DDR of processing system is used as frame buffer. This frame buffer is used by video mixer IP to overlay memory mapped layer. In processing system, HP 0 Slave is enabled, which provides video mixer to read data from DDR. GP0 Master is enabled, which is used to configure the video processing chain. There are two clock signals, viz. **FCLK_CLK0** and **FCLK_CLK1**. First clock is set to 148.5 MHz, which is used by video processing chain while second clock is set to 200 MHz, which is supplied to DVI2RGB IP as **RefClk**.

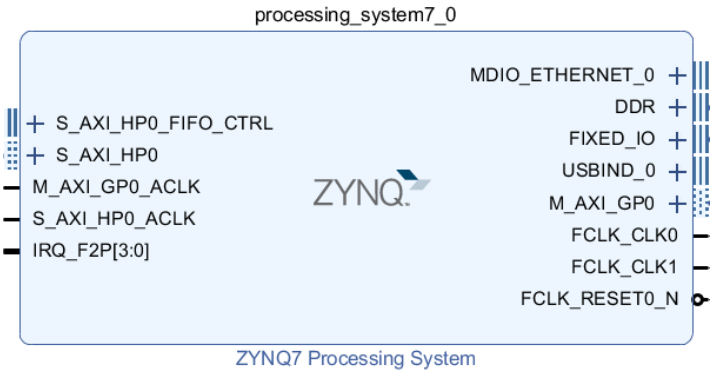


Figure 12. Zynq Processing System

When, initially, processing system is added, the Vivado IP Integrator allows us to **Run Block Automation.** By doing such automation, the Vivado IP Integrator automatically sets the board related presets on the processing.

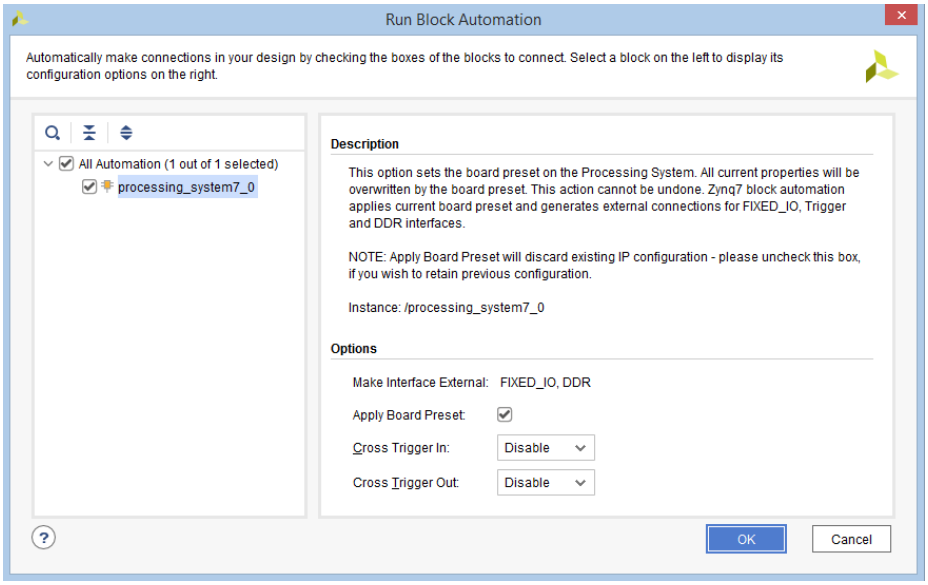


Figure 13. Run Block Automation

3.2.2. DVI to RGB Video Decoder

The real-time video stream is fed to board through **hdmi_in** port. This time the video signal is in the form of TMDS signal. This type of signal is converted into 24 bit RGB format with vertical and horizontal sync signals. It also allows to set the preferred resolution.

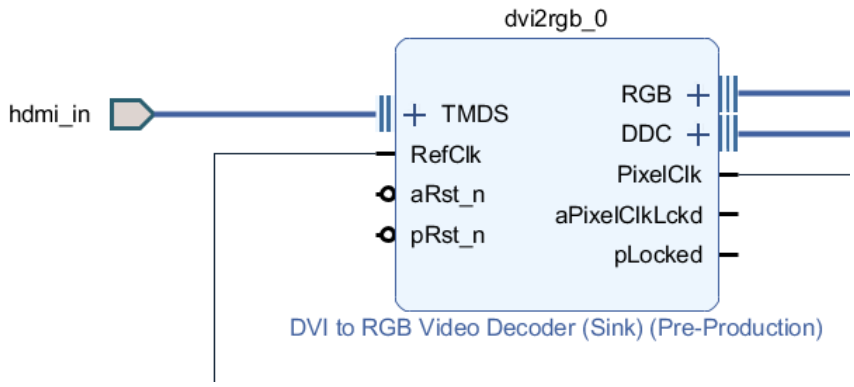


Figure 14. Dvi to Rgb Video Decoder

3.2.3. Video In to AXI Stream

This converts the parallel video and sync signals into AXI stream. Along with the image data on TDATA, the start of frame is identified by the TUSER signal while the end of line is identified by the TLAST signal.

Under the customization, this IP is configured in independent clock mode because, pixel clock and AXI stream clocks are different.

3.2.4. AXI Subset Converter

This component remaps the format of the 24 bit video output into the correct RGB format. This IP resides between **Video In to AXI Stream** IP and **Video Mixer** IP.

3.2.5. Video Mixer

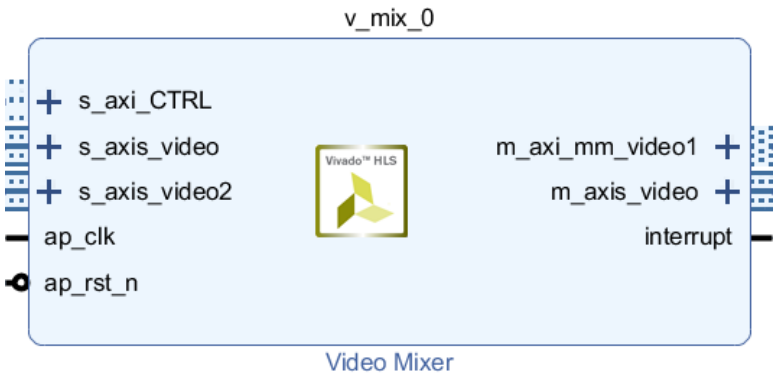


Figure 15. Video Mixer IP

This is the Vivado HLS generated IP, from one of the Xilinx LogiCORE IPs. It is a highly configurable IP core that supports blending of up to nine video and/or graphics layers plus an additional logo layer into one single output video stream. All layers except the master layer can either be a memory mapped AXI4 interface or AXI4-Stream based. Alpha-blending (global or per pixel) and scaling is supported per layer. Finally, built-in color space conversion between RGB and YUV 4:4:4 and chroma re-sampling between YUV 4:4:4, YUV 4:2:2, and YUV 4:2:0 is optionally available.

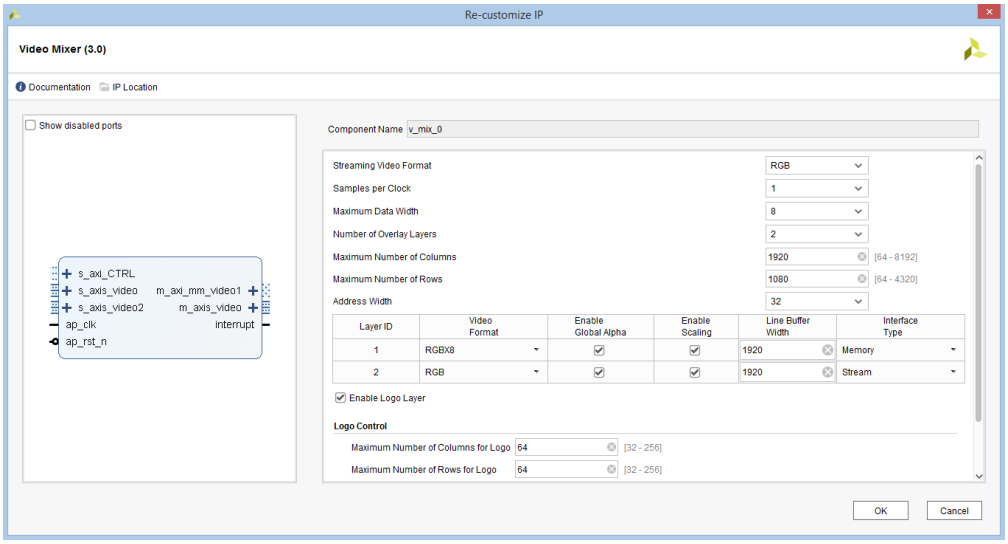


Figure 16. Video Mixer IP Customization

The above picture depicts about the customization of video mixer that is done in this project. This IP is customized in such a way that it has a master streaming layer with video format RGB. Its samples per clock is set to 1. Number of overlays are 2, in which one is memory mapped layer with video format is RGBX8 while another layer is stream layer with video format RGB. The stream layer has stream input from TPG IP, this will be discussed in the following section. Both layers are enabled with Global Alpha and Scaling Factor. Since this IP supports eight overlay layers, we can add or remove any other layer by setting the value of **Number of Overlay Layers.**

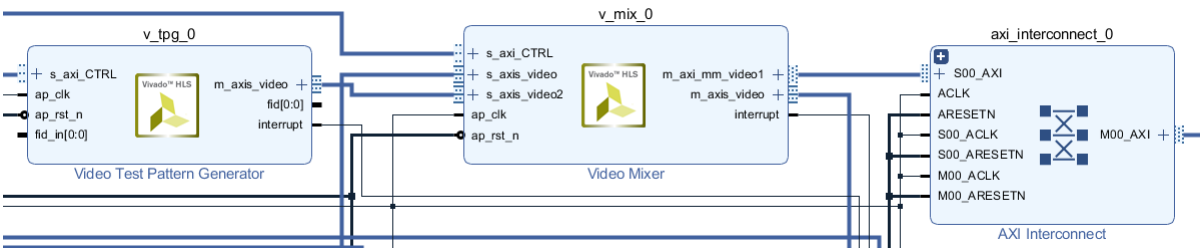
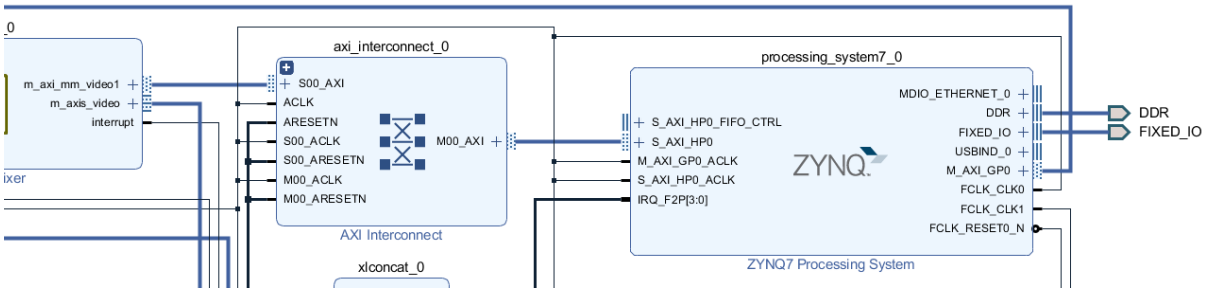


Figure 17. Video Mixer Overlay Layers Connection

From above picture, **s_axis_video** represents the port for master layer or video stream received from AXI4-Stream SubSet Converter IP. **s_axis_video2** is the port for the stream layer. The video stream from TPG IP is fed to this port. Similarly, **m_axi_mm_video1** is the port for memory mapped layer. Video mixer IP reads the memory mapped data from DDR memory via **AXI Interconnect** IP. This IP is connected to High Performance Slave Interface of Processing System. And then processing system allows access to DDR memory.



Finally, **m_axis_video** is the port of video mixer, from which the resulting mixed video stream are obtained. This resulting stream has stream and memory mapped layers overlaid to the master video stream and is connected to AXI4 Stream to Video Out IP later.

3.2.6. Video Test Pattern Generator (TPG)

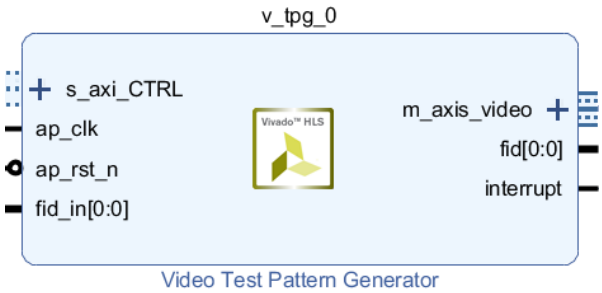


Figure 18. TPG IP

This is also one of the Vivado HLS generated IP from Xilinx. This IP has AXI4 Stream interface. It generates the video stream with different pattern. This pattern can be specified from PS through coding. But currently, this IP is configured to generate color bar test patterns, which is connected to stream overlay interface of video mixer IP.

3.2.7. AXI4 Stream to Video Out

This IP resides after Video mixer IP. This function is just opposite to that of Video In to AXI Stream IP. In other words, it converts the AXI Stream back into RGB form. This IP is also customized to work in independent clock mode.

3.2.8. RGB to DVI Video Encoder

This IP does encoding. It converts RGB form of video stream to TMDS signals to **hdmi_out** port. From this port, hdmi cable leads to output monitor to see the resulting video stream.

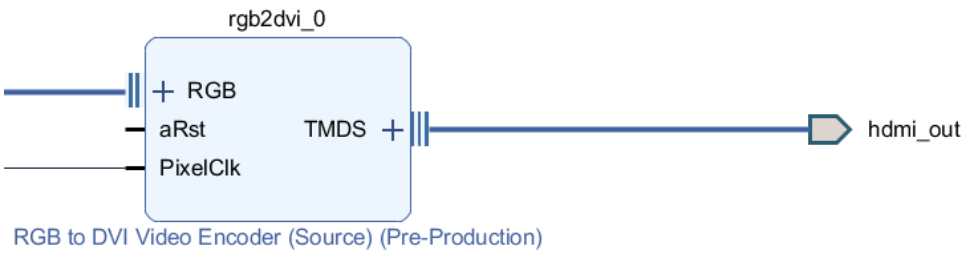


Figure 19. RGB2DVI IP

3.2.9. Video Timing Controller (VTC)

There are two VTC IPs are used. One is configured in detection mode and another is in generation mode. Both IPs are governed by PS. When VTC is in detection mode, it detects video timings and this is sent to PS. On contrary, VTC in generation mode generates the video timing for the output. This IP is configured from PS, such as, the timing information detected by detector VTC IP is now used to configure the generator VTC IP. It implies that based on the timing detected by detector VTC IP, the generator VTC IP is configured to support output resolution as input resolution. The video timing generation of generator VTC IP is controlled by AXI4 Stream to Video Out IP. This is so because, the video stream and the video timing are always locked state to produce output video.

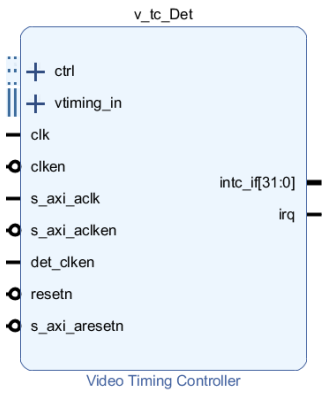


Figure 20. Detector VTC IP

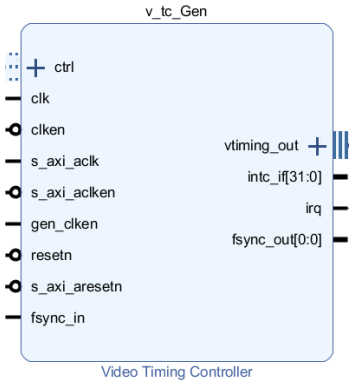


Figure 21. Generator VTC IP

3.2.10. AXI GPIO

There are two of this IP. One is used to assert the hot plug detect (HPD) on the HDMI source. Another is used for Video mixer and TPG IP reset.

Figure 22. AXI GPIO for Hdmi HPD

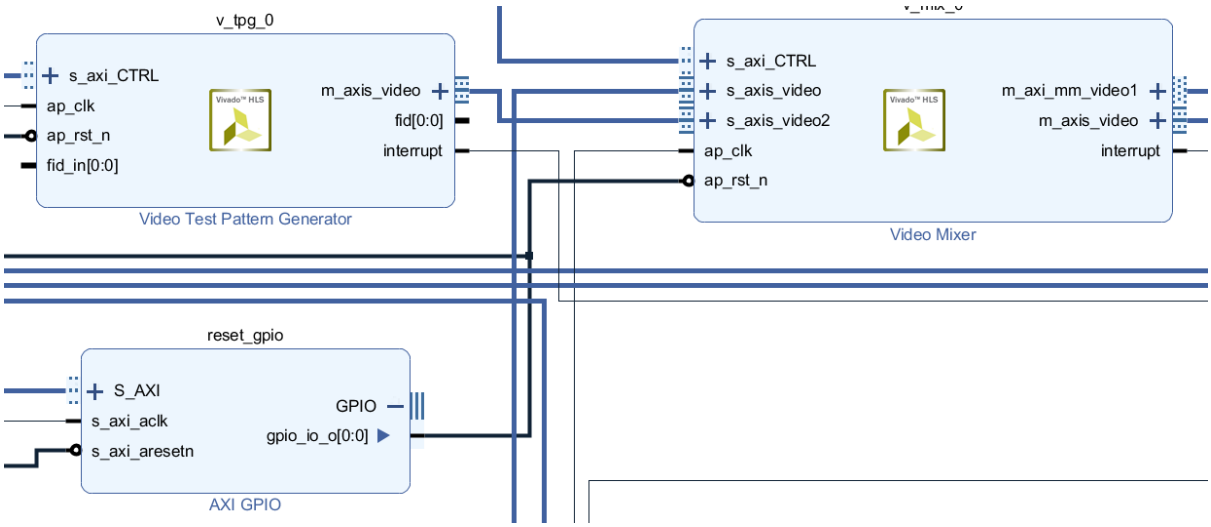


Figure 23. AXI GPIO for resetting Video Mixer and TPG IPs

3.2.11. Other IPs

Apart from above IPs, there are other IPs, which are usually automatically added and connected by Vivado while doing **Run Automatic Connection**. There are two **AXI Interconnect IP**s. One of which connects all AXI-LITE interface of IPs to General Purpose AXI Matster interface of PS whereas another AXI Interconnect IP connects the memory mapped interface to High Performance AXI Slave interface.

Similary, there is **Processor System IP**, which performs peripheral IP reset operation to all IPs that are connected to **FCLK_CLK0** clock source.

Finally, there is **Concatenate IP** used to connect multiple interrupt sources to PS fabric interrupt.

3.3.Software Design Flow

3.3.1. Creating New SDK Project

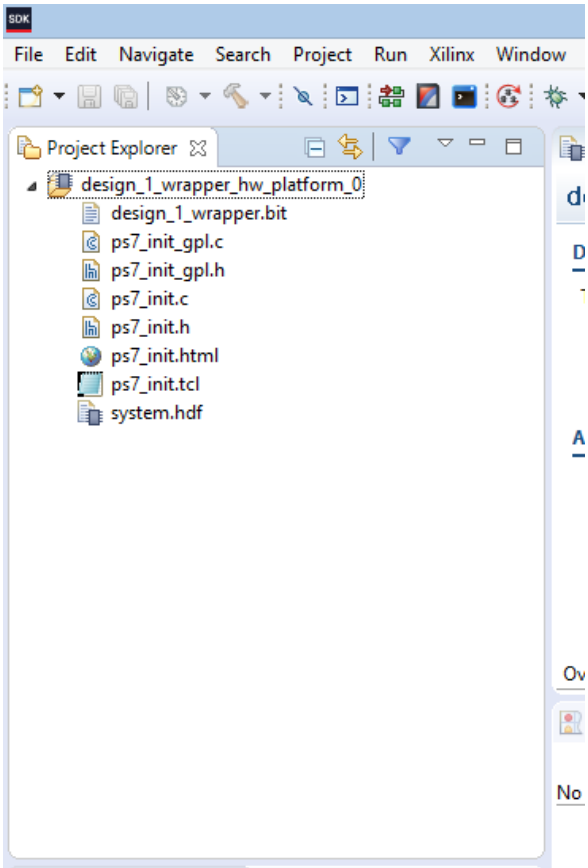


Figure 24. Hardware Design in Project Explorer Pane

After launching the Xilinx SDK, if the hardware export, including bitstream, was successful, then we would see our hardware platform and bit file in the **Project Explorer** pane.

Now, we have to create the new project to start coding and create software application to launch upon the board PS.

File 〉New 〉Application

Or shortcut key

Alt+Shift+N

Now, give the project name. Let all other options as it is. But one thing, we have to take care is, the **Hardware Platform** option selection must be same as that in the **Project Explorer** pane.

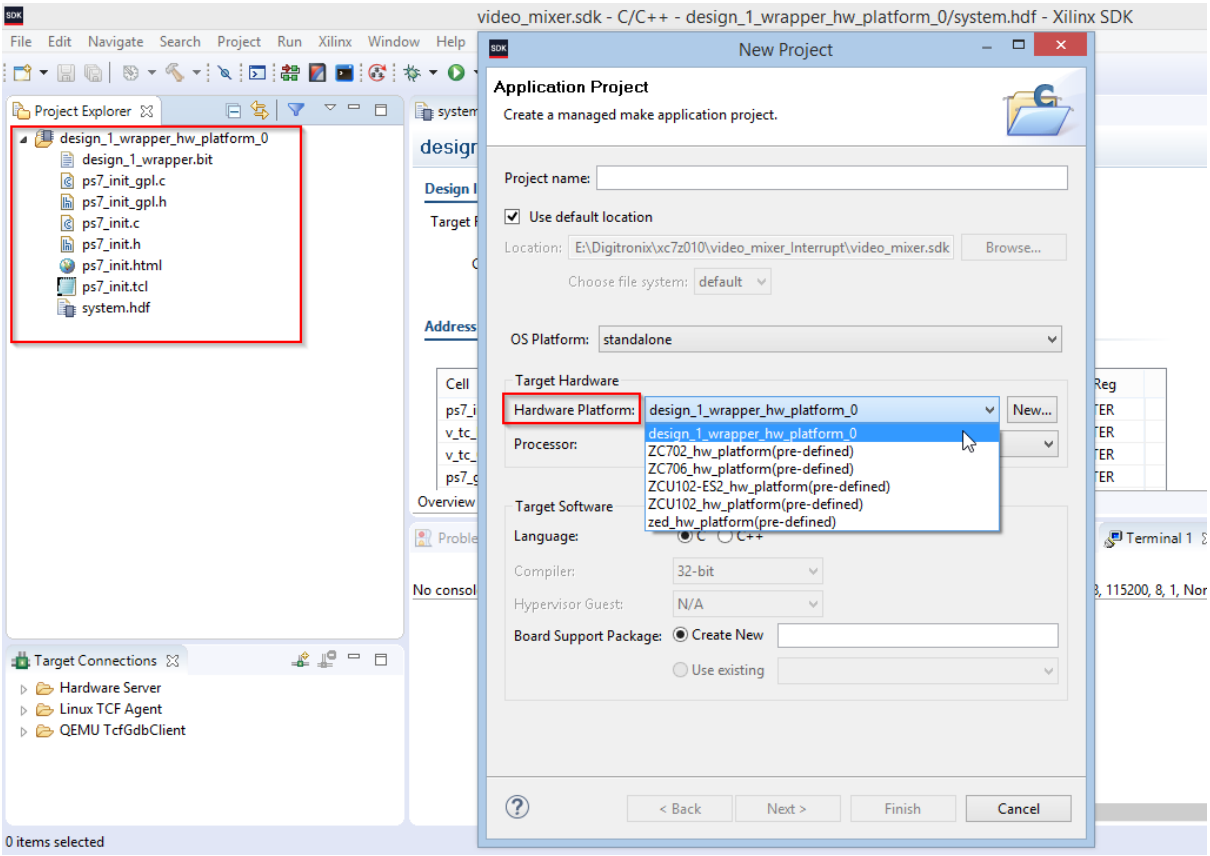


Figure 25. Creating New Application Project

Give the project name and click next to choose different application template.

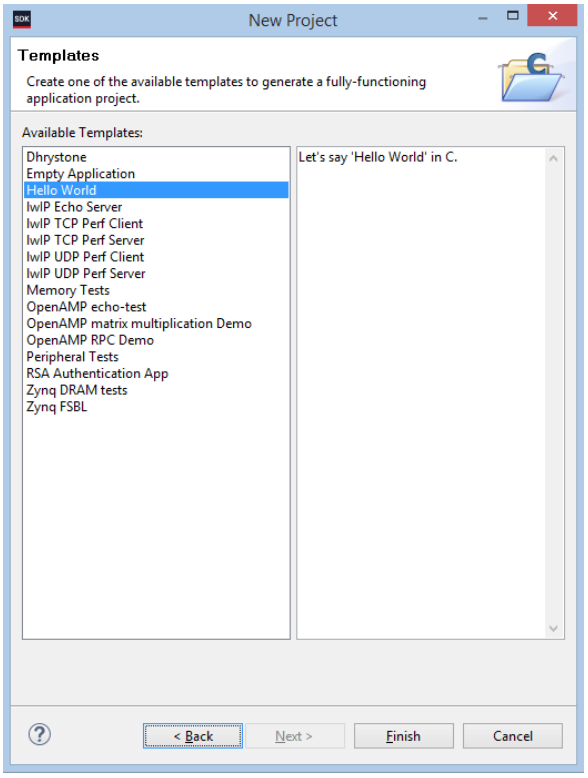


Figure 26. Application Templates

Best way to start is with **Hello World** application project for first time. However, one can even start with **Empty Application**. But difference is, in empty application, user has to include all the necessary files, including .c files and .h files, by himself/herself whereas hello world application contains all necessary files to begin with.

After selecting hello world project, click **Finish** to open project related necessary stuffs and lets start editing the main file, that is, **helloworld.c** file.

The **main** software application is explained here with in section by section format, while you can get complete code from the attached "VIVADO Project".

1. Including board specific, IP specific, operation specific header files, which allow to used library function in code.

```
#include    <stdio.h>
#include    "platform.h"
#include    "xil_printf.h"
#include    "xparameters.h"
#include    "xv_mix.h"
#include    "xv_mix_l2.h"
#include    "xgpio.h"
#include    "sleep.h"
#include    "xuartps.h"
#include    "xil_cache.h"
#include    "xv_tpg.h"
#include    "xscugic.h"
#include    "xvtc.h"
```

xv_mix.h and **xv_mix_l2.h** are the header files related to video mixer IP.

2. Definition for xparameter variables. This is done in order to access the different devices, their registers, memory, which are later used to initialize, configure the device and set their respective values while running application on Processing System.

```
#define UART_BASEADDR        XPAR_PS7_UART_1_BASEADDR
#define VTC_GEN_ID           XPAR_V_TC_GEN_DEVICE_ID
#define VTC_DET_ID           XPAR_V_TC_DET_DEVICE_ID
#define VIDE_MIXER_ID        XPAR_V_MIX_0_DEVICE_ID
#define TPG_ID               XPAR_V_TPG_0_DEVICE_ID
#define VIDEO_DETECT_ID      XPAR_V_TC_GEN_DEVICE_ID
```

This includes address of UART, Generator VTC IP, Detector VTC IP, video mixer IP, TPG IP, AXI GPIO.

3. Layer Buffer Address Definition

```
#define XVMIX_LAYER_BASEADDR       (XPAR_PS7_DDR_0_S_AXI_BASEADDR + (0x21000000))
#define XVMIX_LAYER_ADDR_OFFSET    (0x01000000U)
#define XVMIX_CHROMA_ADDR_OFFSET   (0x01000000U)
```

This is done to set buffer address in DDR for video mixer memory mapped overlay layer. Layer address offset determines the next starting buffer address for another memory mapped layer after current memory mapped layer. And chroma address offset determines the next starting chroma buffer address for another memory mapped layer after current memory mapped layer.

4. Declaration of global variables. So that, IP specific data are stored and are accessed globally from anywhere of the code.

```
XScuGic intc;
XV_mix mix;
XV_Mix_l2 mix12;
XV_Mix_l2 *MixerPtr = &mix12;
XGpio rstGpio;
XGpio hpdGpio;
XVidC_VideoStream VidStream;
XV_tpg tpg;
XVtc vtcDet, vtcGen;
XVtc_Config *vtcConfigDet, *vtcConfigGen;
XVtc_Timing TimingPtr;
XVtc_Timing vtc_timing;
```

This includes global variables of Generic Interrupt Controller, Video mixer, AXI Gpio, TPG, VTC, Video Stream.

5. Setting overlay layer window

```
static const XVidC_VideoWindow MixLayerConfig[2] =
            { // X    Y     W    H
               { 10, 10, 256, 256 }, //Layer 1
               { 300, 10, 256, 256 } //Layer 2

            };
```

This is used to set the overlay layer window, such as, layer position in x any y coordinate and dimension of layer in width and height. When overlay layers are enabled, then layers are overlaid according to these parameters.

Now, we need to write main function. It has following structure;

```
int main() {

    init_platform();

    /* Setup Reset line */
    gpio_hlsIpReset = (u32*) XPAR_RESET_GPIO_BASEADDR;
    //Release reset line
    *gpio_hlsIpReset = 1;

    SetupInterrupts();
    driverInit();
    detectHdmi();
    Xil_ExceptionEnable();
    resetIp();
    configVtcDet();
    configVtcGen();
    ConfigMixer(&VidStream);
    ConfigTpg(&VidStream);
    RunMixer();
    cleanup_platform();
    return 0;
}
```

3.3.2. Preparing the FPGA board and Launching the SDK

After successful completion of coding, it is time to launch on board.

First, make sure that the board is well configured to be programmed from the PC, and then connect it to PC by programming cable.

Now, to begin, we first do bitstream programming to board. And we need to sure that the right **Hardware Platform** and corresponding **BitStream** are selected. Then, do **Program**.

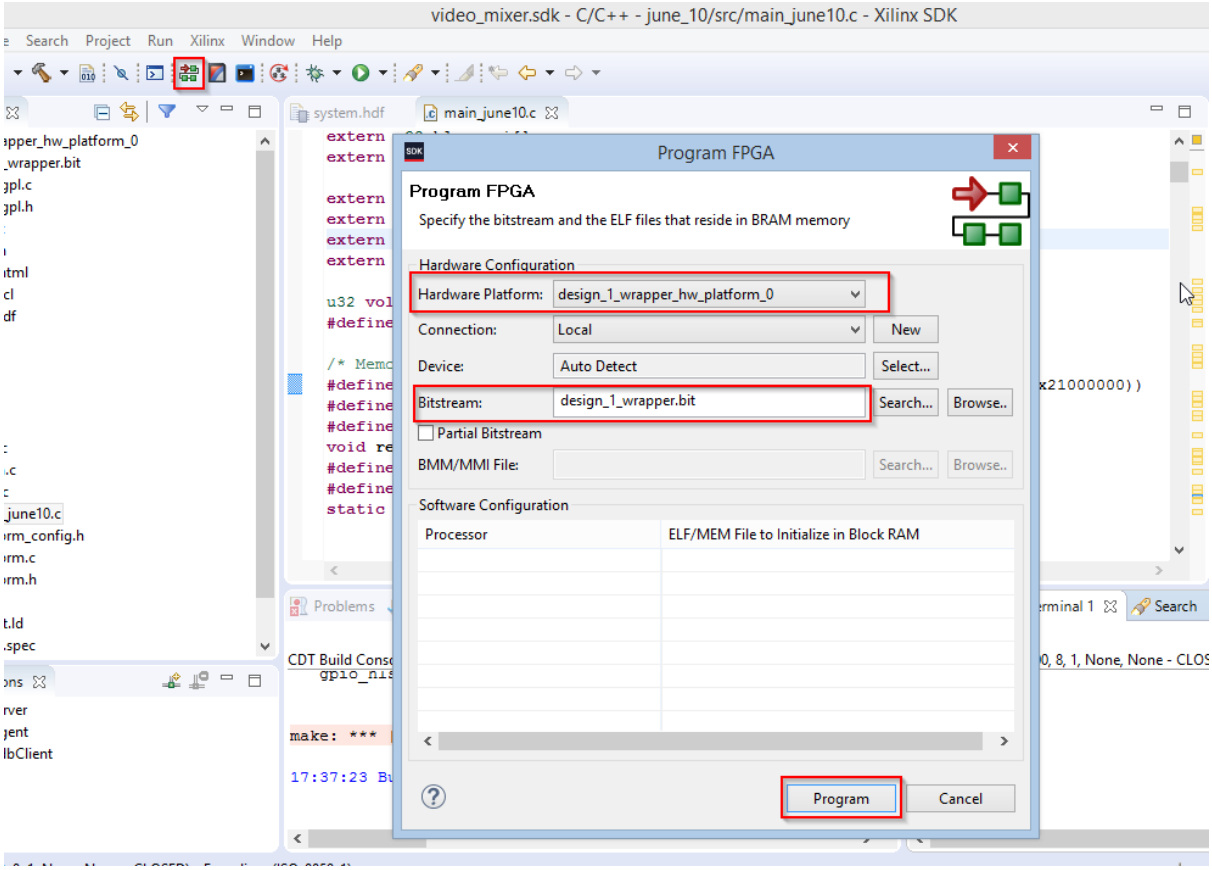


Figure 27. FPGA Programming

Successful bitstream programming can be monitored from **Green LED** on the target board.

We need to now launch the application on our board PS by following.

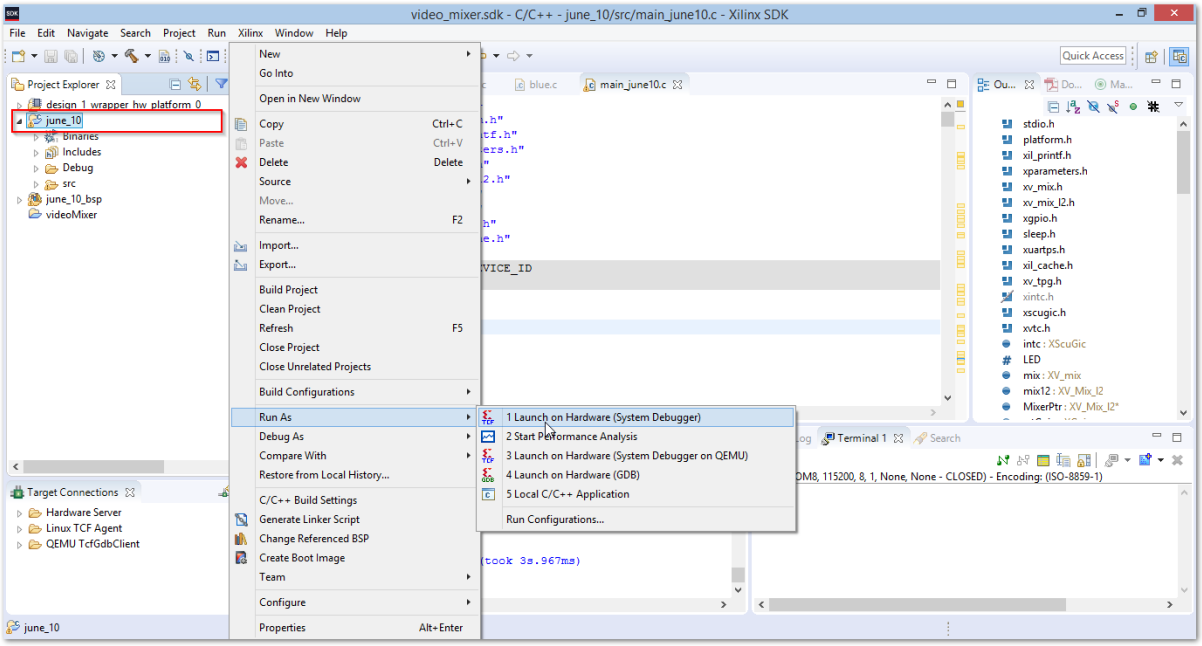


Figure 28. Launching the sdk

Make sure that the terminal is added and well connected to see the messages received from the board after successful running of application.

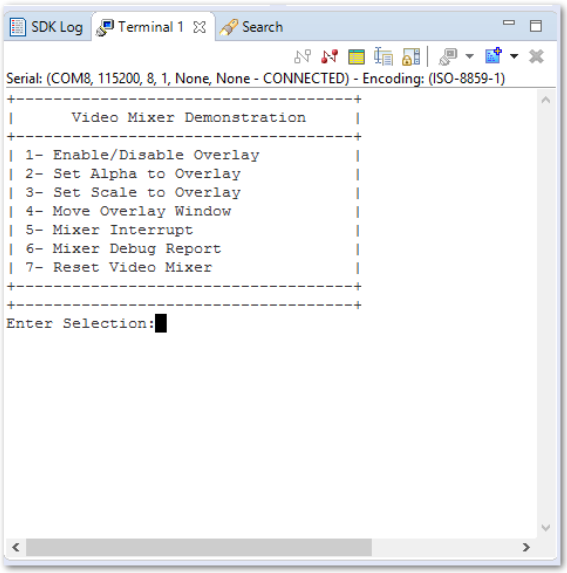


Figure 29. Terminal Message

In the terminal, we can see different options selection message to select different video mixer related action. As per selection of option, we obtain different video mixer output, output are detailed with different scenarios at the "final output section at last".

3.4. SDK Application details: functions

Initially, the platform is started. And then configuring Reset AXI GPIO for reset operation. The code under following functions are described individually as following;

1. Interrupt Setup

   This video mixer design runs under interrupt mode. Therefore, interrupt setup has to be done. This interrupt setup consists of configuration of Generic Interrupt Controller (GIC).

   GIC helps to connect the IP related interrupt to PS. Therefore, PS will be able to monitor the interrupt and will execute Interrupt Service Routine (ISR).

```
XScuGic *IntcPtr = &intc;
XScuGic_Config *IntcCfgPtr;
IntcCfgPtr = XScuGic_LookupConfig(XPAR_SCUGIC_0_DEVICE_ID);
if (IntcCfgPtr == NULL) {
    print("ERR:: Interrupt Controller not found");
    return (XST_DEVICE_NOT_FOUND);
}
int Status = XScuGic_CfgInitialize(IntcPtr, IntcCfgPtr,
        IntcCfgPtr->CpuBaseAddress);
if (Status != XST_SUCCESS) {
    xil_printf("Intc initialization failed!\r\n");
    return XST_FAILURE;
}
```

   These above line of codes initializes the Generic Interrupt Controller.

```
Status |= XScuGic_Connect(IntcPtr,
XPAR_FABRIC_V_MIX_0_INTERRUPT_INTR,
        (XInterruptHandler) XVMix_InterruptHandler, (void *) MixerPtr);
Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
        (Xil_ExceptionHandler) XScuGic_InterruptHandler, IntcPtr);
if (Status != XST_SUCCESS) {
    xil_printf("ERR:: Mixer interrupt connect failed!\r\n");
    return XST_FAILURE;
}
```

   This line of codes sets the Interrupt Service Routine, such as, it connects Video mixer interrupt to Processing System Fabric Interrupt. `XVMix_InterruptHandler` is the Interrupt handler of Video mixer.

```
XScuGic_Enable(IntcPtr, XPAR_FABRIC_V_MIX_0_INTERRUPT_INTR);
```

   This line of code now enables the GIC and starts the interrupt controller.

2. Initialization of devices

```
int Status = XV_mix_Initialize(&mix, VIDE_MIXER_ID);
if (Status != XST_SUCCESS) {
    xil_printf("video mixer initialization failed\n\r");
}
Status = XVMix_Initialize(&mix12, VIDE_MIXER_ID);
if (Status != XST_SUCCESS) {
    xil_printf("ERROR:: Mixer device not found\r\n");
    return (XST_FAILURE);
}
Status = XV_tpg_Initialize(&tpg, TPG_ID);
if (Status != XST_SUCCESS) {
    xil_printf("ERROR:: TPG device not found\r\n");
    return (XST_FAILURE);
}
```

These above lines of code initialize the Video mixer and TPG IPs.

```
vtcConfigDet = XVtc_LookupConfig(VTC_DET_ID);

Status = XVtc_CfgInitialize(&vtcDet, vtcConfigDet,
        vtcConfigDet->BaseAddress);
if (Status != (XST_SUCCESS))
    return;

vtcConfigGen = XVtc_LookupConfig(VTC_GEN_ID);

Status = XVtc_CfgInitialize(&vtcGen, vtcConfigGen,
        vtcConfigGen->BaseAddress);
if (Status != (XST_SUCCESS))
    return;
```

These lines of code initialize detector VTC IP and generator VTC IP.

```
Status = XGpio_Initialize(&hpdGpio, VIDEO_DETECT_ID);
if (Status != XST_SUCCESS) {
    xil_printf("hdmi hpd initialization failed\n\r");
}
```

This initializes the AXI GPIO IP for video detect.

3. Starting the video stream

```
XGpio_SetDataDirection(&hpdGpio, 1, ~LED);
XGpio_DiscreteWrite(&hpdGpio, 1, LED);
```

By these lines of code, the board asserts the **hdmi_in_hpd**, after which the video source starts video stream through HDMI.

4. Enabling the exception

```
Xil_ExceptionEnable();
```

This line of code enables the IRQ Exception.

5. Video mixer IP and TPG IP reset

```
*gpio_hlsIpReset = 0;
usleep(1000);          //hold reset line

*gpio_hlsIpReset = 1;
usleep(1000);          //wait
```

These lines of code are used to reset the video mixer IP and TPG IP, through Reset AXI GPIO IP. It resets the IPs for 1000us and then releases it for 1000us.

6. Configuration of detector VTC IP

```
XVtc_RegUpdateEnable(&vtcDet);
XVtc_IntrDisable(&vtcDet, 0x100);
XVtc_EnableDetector(&vtcDet);
usleep(3000000);
if (XVtc_GetDetectionStatus(&vtcDet)) {

    u16 vidMode = XVtc_GetDetectorVideoMode(&vtcDet);

    XVtc_ConvVideoMode2Timing(&vtcDet, vidMode, &TimingPtr);
}
```

It disables the interrupt features and then enables the VTC IP as detector mode. Then 3 seconds delay is used to get the detection status of VTC IP. If status is received, then VTC IP detects incoming video mode such as 720p or 1080p. And then, the video mode is converted into timing information. This information is stored in timing pointer variable `TimingPtr`.

7. Configuration of generator VTC IP

```
vtc_timing.HActiveVideo = TimingPtr.HActiveVideo;
vtc_timing.HFrontPorch = TimingPtr.HFrontPorch;
vtc_timing.HSyncWidth = TimingPtr.HSyncWidth;
vtc_timing.HBackPorch = TimingPtr.HBackPorch;
vtc_timing.HSyncPolarity = TimingPtr.HSyncPolarity;

vtc_timing.VActiveVideo = TimingPtr.VActiveVideo;
vtc_timing.V0FrontPorch = TimingPtr.V0FrontPorch;
vtc_timing.V0SyncWidth = TimingPtr.V0SyncWidth;
vtc_timing.V0BackPorch = TimingPtr.V0BackPorch;
vtc_timing.VSyncPolarity = TimingPtr.VSyncPolarity;

XVtc_SetGeneratorTiming(&vtcGen, &vtc_timing);
XVtc_Enable(&vtcGen);
XVtc_EnableGenerator(&vtcGen);
XVtc_RegUpdateEnable(&vtcGen);
```

The timing information that was stored in `TimingPtr` pointer variable, is now used to set the timing parameters for generator VTC IP to generate the video mode timing. And then Generator VTC IP is enabled.

8. Configuration of video mixer

In this section, video mixer is configured for its master layer and two other overlay layers.

```
/* Setup default config after reset */
XVMix_LayerDisable(MixerPtr, XVMIX_LAYER_MASTER);

/* set master layer resolution */
XV_mix_Set_HwReg_width(&mix, TimingPtr.HActiveVideo);
XV_mix_Set_HwReg_height(&mix, TimingPtr.VActiveVideo);
XV_mix_Start(&mix);
```

These lines of code, initially disables the master layer and then sets the master layer resolution according to video mode, such as, resolution of incoming video stream detected by detector VTC IP.

```
for (index = XVMIX_LAYER_1; index < NumLayers; index++) {
    XVMix_GetLayerColorFormat(MixerPtr, index, &Cfmt);
    if (!(XVMix_IsLayerInterfaceStream(MixerPtr, index))) {
        Status = XVMix_SetLayerBufferAddr(MixerPtr, index, MemAddr);
        if (Status != XST_SUCCESS) {
            xil_printf(
                    "MIXER ERROR:: Unable to set layer %d buffer addr to 0x%X\r\n",
                    index, MemAddr);
        } else {
            if ((Cfmt == XVIDC_CSF_MEM_Y_UV8)
                    || (Cfmt == XVIDC_CSF_MEM_Y_UV8_420)
                    || (Cfmt == XVIDC_CSF_MEM_Y_UV10)
                    || (Cfmt == XVIDC_CSF_MEM_Y_UV10_420)) {
                MemAddr += XVMIX_CHROMA_ADDR_OFFSET;
                Status = XVMix_SetLayerChromaBufferAddr(MixerPtr, index,
                        MemAddr);
                if (Status != XST_SUCCESS) {
                    xil_printf(
                            "MIXER ERROR:: Unable to set layer %d chroma buffer2 addr to 0x%X\r\n",
                            index, MemAddr);
                }
            }

            MemAddr += XVMIX_LAYER_ADDR_OFFSET;
        }
    }
}
```

These lines of code consists of for loop statement for assigning buffer address to memory mapped overlay layers one by one. For loop statement is so used because there are more than one overlay layers in video mixer.

Initially, layer interfaced is checked. If layer is memory mapped layer, then if statement is executed to set buffer address for this layer. While on the other hand, if layer is stream layer, then buffer address is not set for it. This is because, stream layer does not require buffer memory as it does not read from memory.

On the other hand, if overlay layer has different color format other than RGX8, then it is required to set chroma buffer address.

There are only two layers. Therefore, for loop only runs for twice. As the loop passes, the buffer address is incremented through `XVMIX_LAYER_ADDR_OFFSET` value.

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

This line of code is used to copy the memory mapped overlay data into its buffer address. The video mixer will read this data to display the overlay over the master layer. For writing data to buffer address, we have to disable the cache and after writing, we need to enable the cache.

```
XVMix_SetBackgndColor(MixerPtr, XVMIX_BKGND_BLUE, XVIDC_BPC_8);
XVMix_LayerEnable(MixerPtr, XVMIX_LAYER_MASTER);
XVMix_InterruptEnable(MixerPtr);
XVMix_SetCallback(MixerPtr, (void *) readFrame, MixerPtr);
XVMix_Start(MixerPtr);
```

These are the final coding for configuration of video mixer. First of all, the background color is set. We can specify different background color. Currently, it is set to *`XVMIX_BKGND_BLUE`*, that is, blue color with color depth value *`XVIDC_BPD_8`*, that is, 8 bit. The background color is only set for master layer. This is because, when master layer is absent or disabled, then there is seen background color, which we have set.

After this, interrupt is enabled, by which, the video mixer starts in interrupt mode. Therefore, it does not start automatically. Based on the interrupt handled, the IP works. In such interrupt mode, the video mixer only works, depending upon two types of interrupt. They are; **ap_start** and **ap_ready**. The first interrupt is generated, when IP starts and second interrupt is generated when IP completes the frame processing and then gets ready for processing another frame.

As each time IP generates the interrupt, this must be handled. For this, we need to set ISR. This is set by using `XVMix_SetCallback()` function. Here "readFrame" is the ISR. Each time interrupt is generated, the "readFrame" function is executed.

Finally, video mixer is started.

9. Configuration of TPG

```
//Stop TPG
XV_tpg_DisableAutoRestart(&tpg);

XV_tpg_Set_height(&tpg, 256);
XV_tpg_Set_width(&tpg, 256);
XV_tpg_Set_colorFormat(&tpg, StreamPtr->ColorFormatId);
XV_tpg_Set_bckgndId(&tpg, XTPG_BKGND_COLOR_BARS);
XV_tpg_Set_ovrlayId(&tpg, 0);

//Start TPG
XV_tpg_EnableAutoRestart(&tpg);
XV_tpg_Start(&tpg);
```

These lines of code are responsible for the configuration of TPG, as it sets the height and width of test pattern, color format and type of test patter. Currently, type of test pattern is set to `XTPG_BKGND_COLOR_BARS`, that is, vertical color bar. We can set different TPG type.

10. Running video mixer

Now after successful configuration, it is time to run the video mixer to work in real-time.

```
switch (userInput) {
case '1':
    optionIsSelected = FALSE;
    overlayEnableDisable();
    break;
case '2':
    optionIsSelected = FALSE;
    setAlphaBlending();
    break;
case '3':  //scale
    optionIsSelected = FALSE;
    setScaleFactor();
    break;
case '4':
    optionIsSelected = FALSE;
    moveLayerWindow();
    break;
case '5':
    optionIsSelected = TRUE;
    break;
case '6':
    optionIsSelected = FALSE;
    debugMenu();
    break;
case '7':
    main();
    break;
default:
    optionIsSelected = FALSE;
    printMainMenu();
    break;
}
XVMix_InterruptHandler(MixerPtr);
```

The video mixer works based on the selection of different option as seen in the terminal. For this, switching statement is used. According to selection value received from UART terminal, the selection is made and respective operation is performed. From above code, the last line of code, that is, `XVMix_InterruptHandler()`, is the interrupt handler of

video mixer. It monitors whether interrupt is occurred or not. If interrupt is occurred, it triggers the ISR, that is, readFrame() function is invoked.

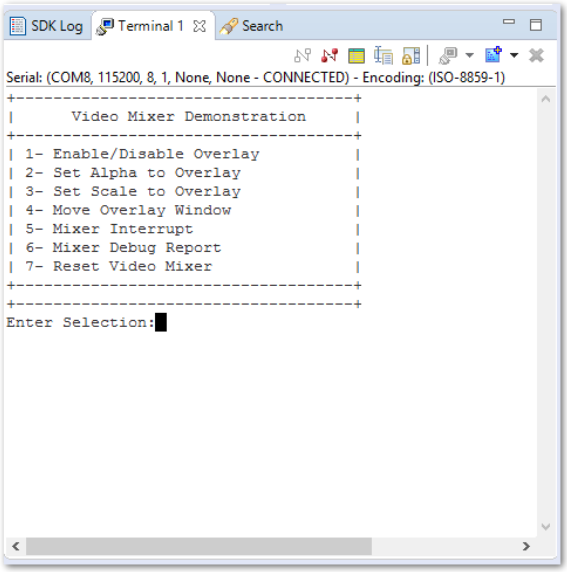


Figure 30. Terminal Message

4. Final Output

When all steps go right, we will be able to see the output on the output monitor.

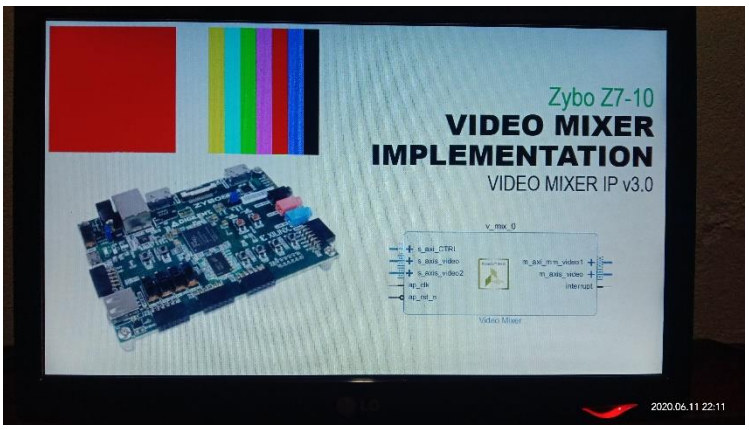
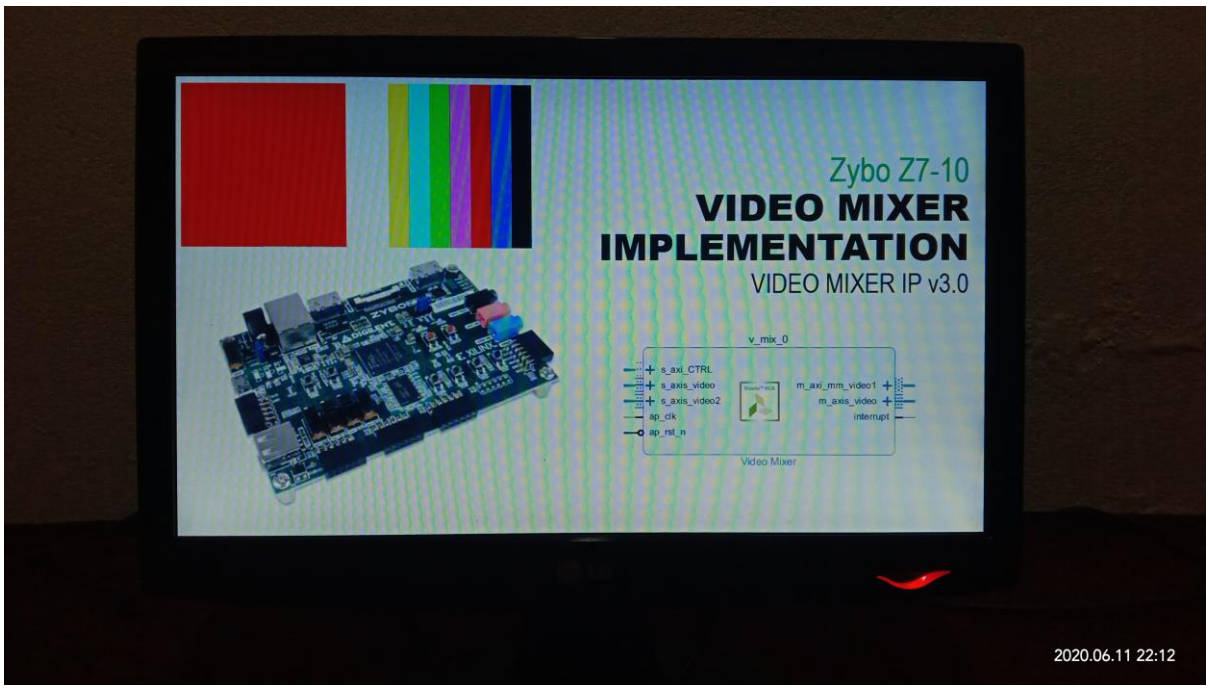


Figure 31. Video Mixer Design Output

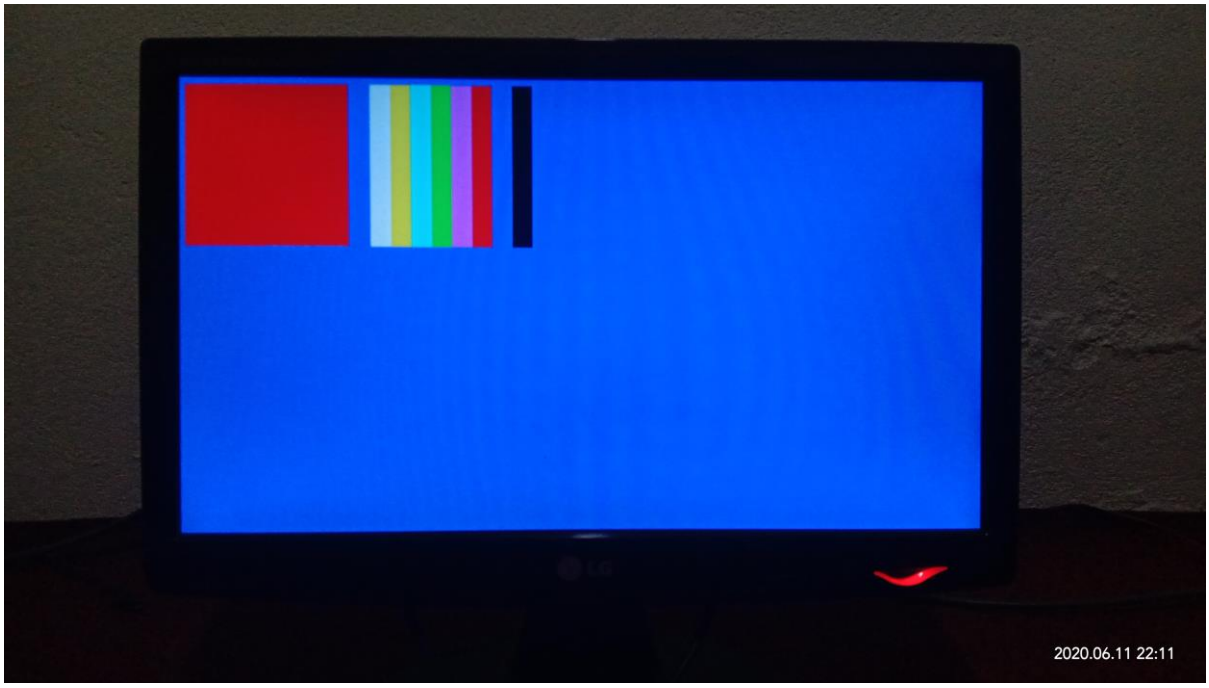
Different cases and outputs are included below.

4.1.Case 1:

When we make selection '1', then case 1 is executed, which then leads to execution of enable or disable of overlay layer including master layer. If master layer is disabled then background color is displayed.



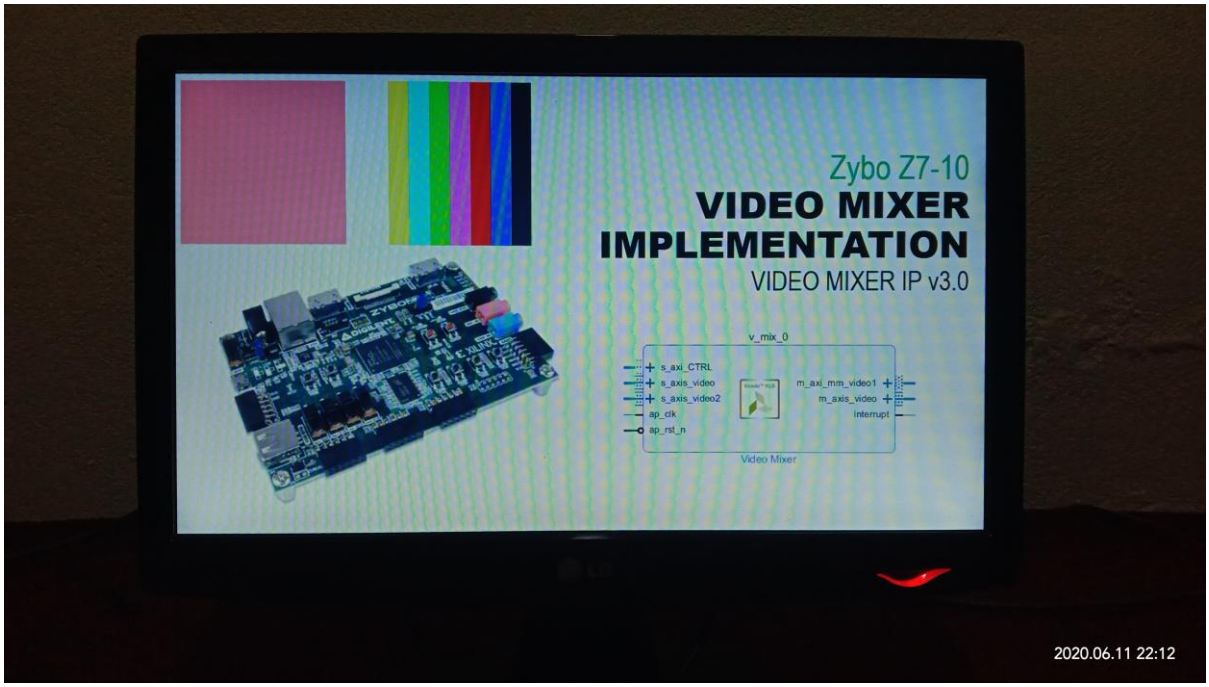
This is the output when both layers are enabled. Here, left red square is the memory mapped layer while right one is the stream layer from TPG IP.



This is the output when master layer is disabled.

4.2. Case 2:

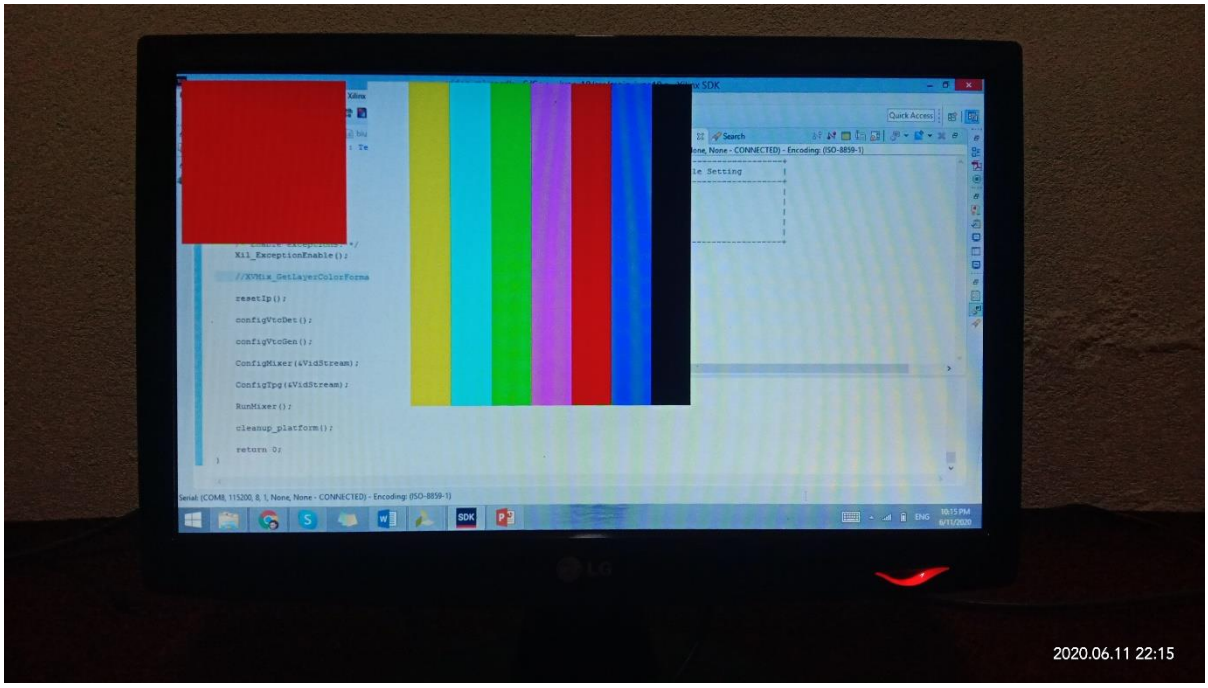
When we make selection '2', then case 2 is executed, which then leads to execution of code that is related to setting the alpha value of selected overlay layer.



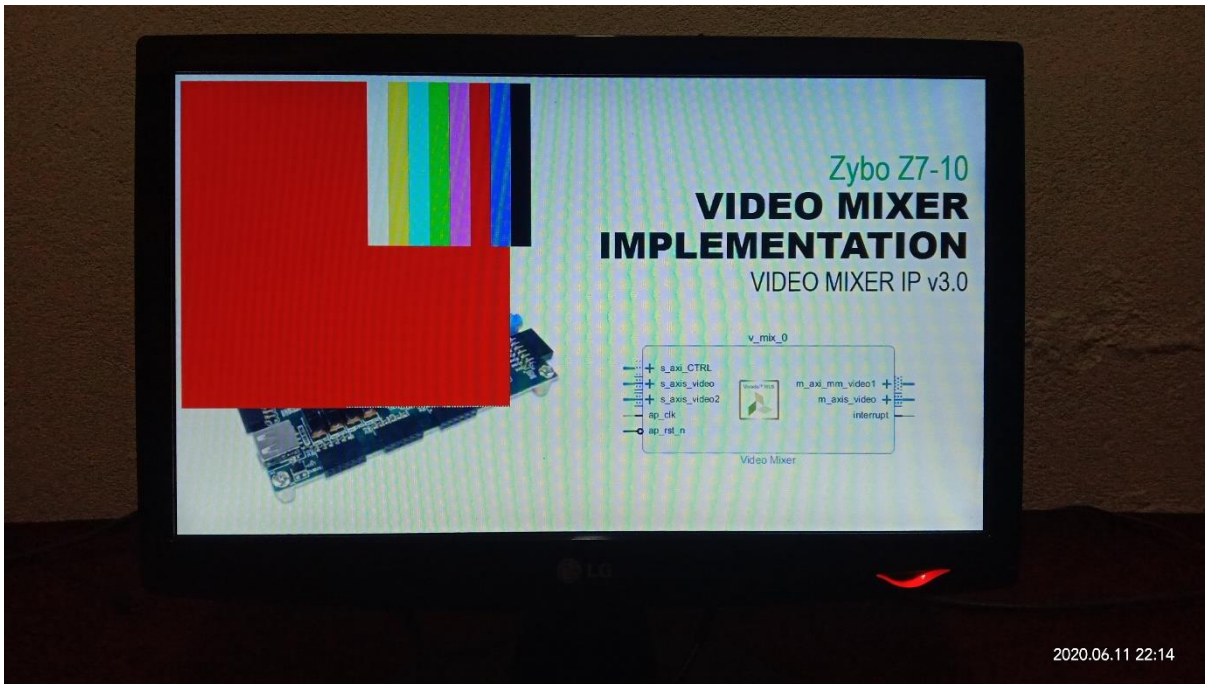
This is the output when setting alpha value to each layer.

4.3. Case 3:

When we make selection '3', then case 3 is executed, which then leads to execution of code that is related to setting the scale factor of selected overlay layer.



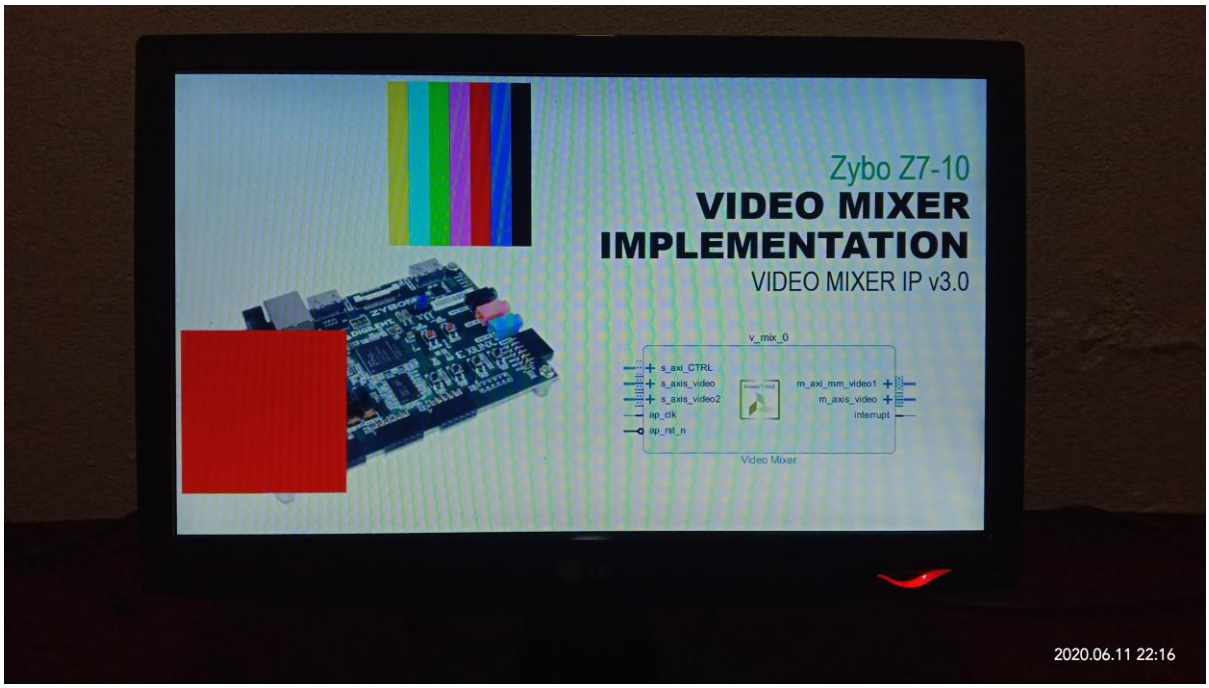
This is the output when stream is scaled by 2x.



This is the output when memory mapped layer is scaled by 2x.

4.4. Case 4:

When we make selection '4', then case 4 is executed, which then performs the changing the position of selected overlay layer.

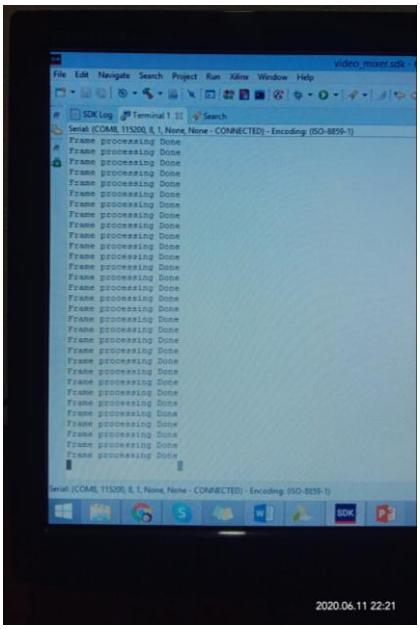


This is the output when memory mapped layer position is changed

This is the output when stream layer position is changed.

4.5.Case 5:

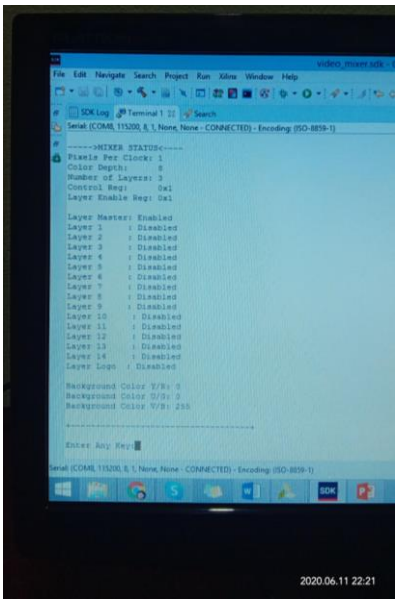
When we make selection '5', then case 5 is executed, which then displays the interrupt handled message in the terminal.



This output shows the status of video mixer interrupt handling. As the interrupt is occurred after processing the frame, the ISR is invoked which then prints this message in the terminal.

4.6.Case 6:

When we make selection '6', then case 6 is executed to display debug report status related to selected overlay layer.



This output shows the debug report of entire video mixer IP.

4.7.Case 7:

And finally, when we make selection '7', then case 7 is executed, which then performs the software resets to start the running of application from beginning.

5. References

[1] Xilinx, "Video Mixer v3.0 LogiCORE IP Product Guide," Xilinx, 5 December 2018. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/v_mix/v3_0/pg243-v-mix.pdf.

[2] Digilent, "Zybo Z7 Reference Manual," Digilent, [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual.

[3] Xilinx,Inc, "Video Series 32 - Visualizing the Video_Mixer example design using the ZC702 evaulation kit's On-Board HDMI," [Online]. Available: https://forums.xilinx.com/t5/Design-and-Debug-Techniques-Blog/Video-Series-32-Visualizing-the-Video-Mixer-example-design-using/ba-p/1026338.

***

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