Lab Manual

Laboratory

Synthesis of Digital Systems

Prof. Dr.-Ing. Ulf Schlichtmann

May 21, 2014

Institute for Electronic Design Automation
Technische Universität München
Arcisstr. 21, 80333 München
Tel.: (089) 289 23666

Synthesis of Digital Systems - Laboratory

Institute for Electronic Design Automation
Technische Universität München

2013-2014 – Munish Jassi

Contents

List of Figures

1. Lab C: Video Processing on ZedBoard and Software Implementation of Gray Scale Filter

In this Lab we will develop a simple video processing system on ZedBoard. At first, a display interface will be set through HDMI port, followed by a camera interface to the external camera device and finally the corresponding software application.
For the HDMI display, an Analog Devices ADV7511 HDMI Transmitter chip provides a digital video and audio interface to the ZedBoard. We will use the Pmod port to connect a camera module to read real time video into the FPGA. A camera interface IP module in the PL, handles the incoming data according to image format. The image data is continuously written to the DDR memory by the ARM processor and displayed on the monitor. Once the hardware set up is available, we will proceed with the development of the software to be run on the ARM processor to process video stream.

Here are the steps we will follow:

1. XPS Project creation and set up of the HDMI display.
2. Adding Camera peripheral core.
3. Software application project in SDK.
4. Hardware set up for Monitor and Camera.
5. FPGA programming and software application execution on ZedBoard.
6. Software application for Gray scale conversion.

1.1. Introduction - System Description

This section will describe the system to be implemented in this Lab. As mentioned earlier, the objective is to build a hardware system for processing the video input stream and display it on the display monitor. There are two ends of the system which we will be implementing. First is the *Camera Interface* end, and other is *Display Interface* end. Software application will enable the CPU (ARM Cortex-A9) to read the video inputs and write it to DDR memory. VDMA (Video Direct memory access) module will tranfer the video data on DDR memory (read by CPU) to HDMI interface module, which further displays it on the display monitor.

Camera Interface:

Camera interface is a custom core and will be used in the PL. The block-diagram for the module and its connections are as shown in Figure 1.1. This core captures the image frames from the camera device and synchronizes the input frame for further CPU data read. The camera module is a VHDL implementation, at one end it connects to Pmod ports of ZedBoard to capture the input frame into line buffer and at the other end via line buffer CPU reads the data through AXI interface. This camera interface module also programs the external OV7670 camera device through I2C interface as shown in the Figure. Pcore IP of this module is provided for this lab in the <project docs>/LabC/cam_interface_v1_00_a folder.

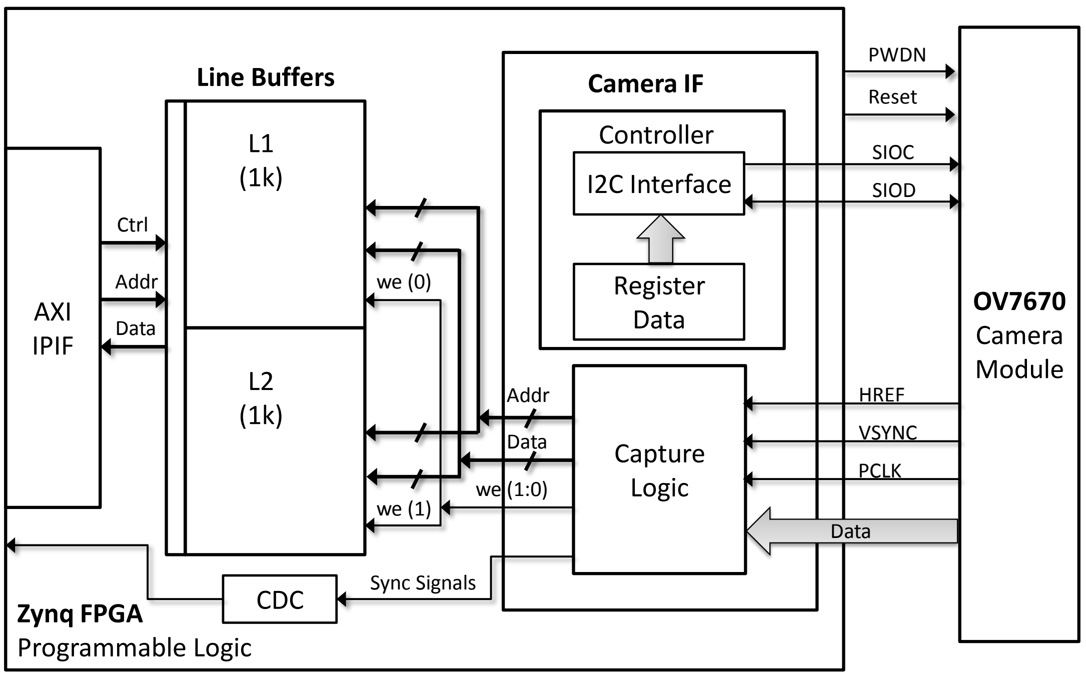


Figure 1.1.: Block diagram for camera interface module and corresponding connections to ZedBoard

It is important to understand the input signals from the camera module. A video is a succession of frames, a frame is a still image taken at an instant of time. A frame comprises of lines, and a line is comprised of pixels. In video processing the first concern is how to handle the synchronization signals or signal timings. Basically all video frames have three regions:

- Vertical blanking
- Horizontal blanking
- Active Image

The active region is the location of the frame where all the image pixels reside. All the processing algorithm act on the data in the active region and this is the data that is displayed on the monitor. The blanking regions are used to synchronize the video frames and provide space for signaling with video equipment to be able to detect and display the video properly. The video regions and synchronization signal are shown in Figure 1.2. These signals must be properly handled in order to process the video frames correctly. The values of front porch and back porch for both horizontal and vertical signals varies depending on the display resolution chosen.

The OV7670 camera module sends the video frame data along with synchronization signals as shown in Figure 1.3. These synchronization signals can be used to acquire the frame data properly. Vsync is utilized to synchronize frames whereas either of the HREF or HSYNC is used for line synchronization. This camera module can be configured for various video input formats and resolutions. This figure shows the timing for 640x480 RGB image transfer. For a detailed description of these signals please refer to OV7670 data-sheet [11].

Display Interface

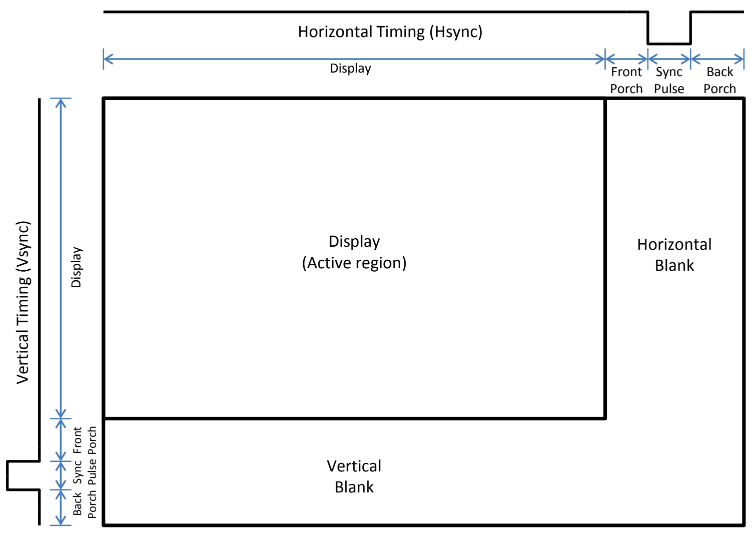


Figure 1.2.: Basic video frame format and synchronization signals

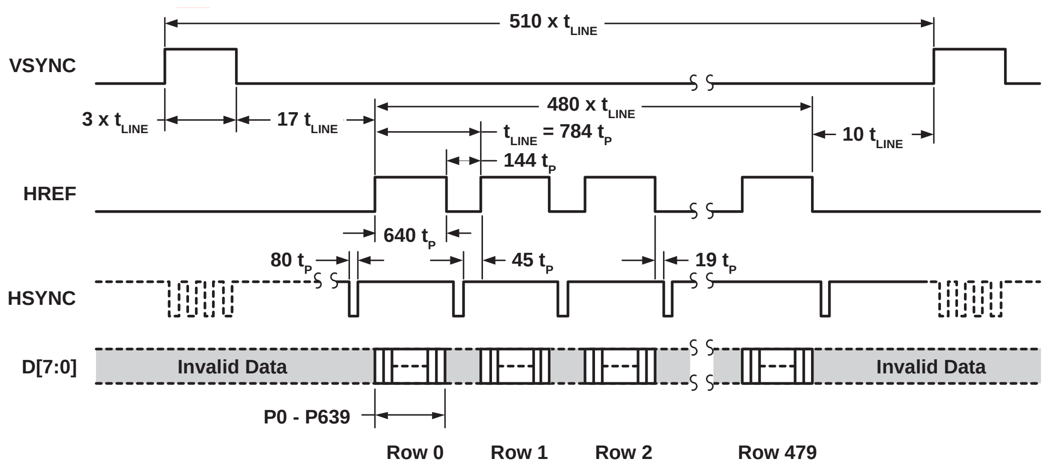


Figure 1.3.: Timing specification for video signal of OV7670 camera

For display we connect *display monitor* ZedBoard via HDMI connector. Block diagram for this interface is as shown in figure 1.4. During system initiallization process, CPU programs AXI-VDMA to transfer the data from DDR memory to HDMI interface core. VDMA module is a special DMA (Direct Memory Access) IP optimized for Video data from Xilinx. At one end it connects to AXI bus and at the other end it streams the data according to AXI Streaming interface protocol. VDMA allows two independent channels, one to read the data from a memory location (on AXI-bus) to *Streaming-Out* interface at other end, and other to write data from *Streaming-In* interface to some memory location (on AXI-bus). These two channels can be independently enabled and configured. For HDMI displace interface, we will only need read channel enabled, to read data from DDR to HDMI core. *CPU configures the control registers of the VDMA to set the source address as the location at which the frame from camera module is written.* This way once the CPU has written the frame on the DDR memory, VDMA reads that frame from the same address location and transfers it to HDMI core for displaying. For more details refer to ADV7511 programming guide [12] and VDMA reference [8]. In the figure, AXI_1 bus (AXI4-Lite interface) is connected to the peripheral modules for programming the control registers, while AXI_2 bus (AXI4 interface) is for the data transfer.

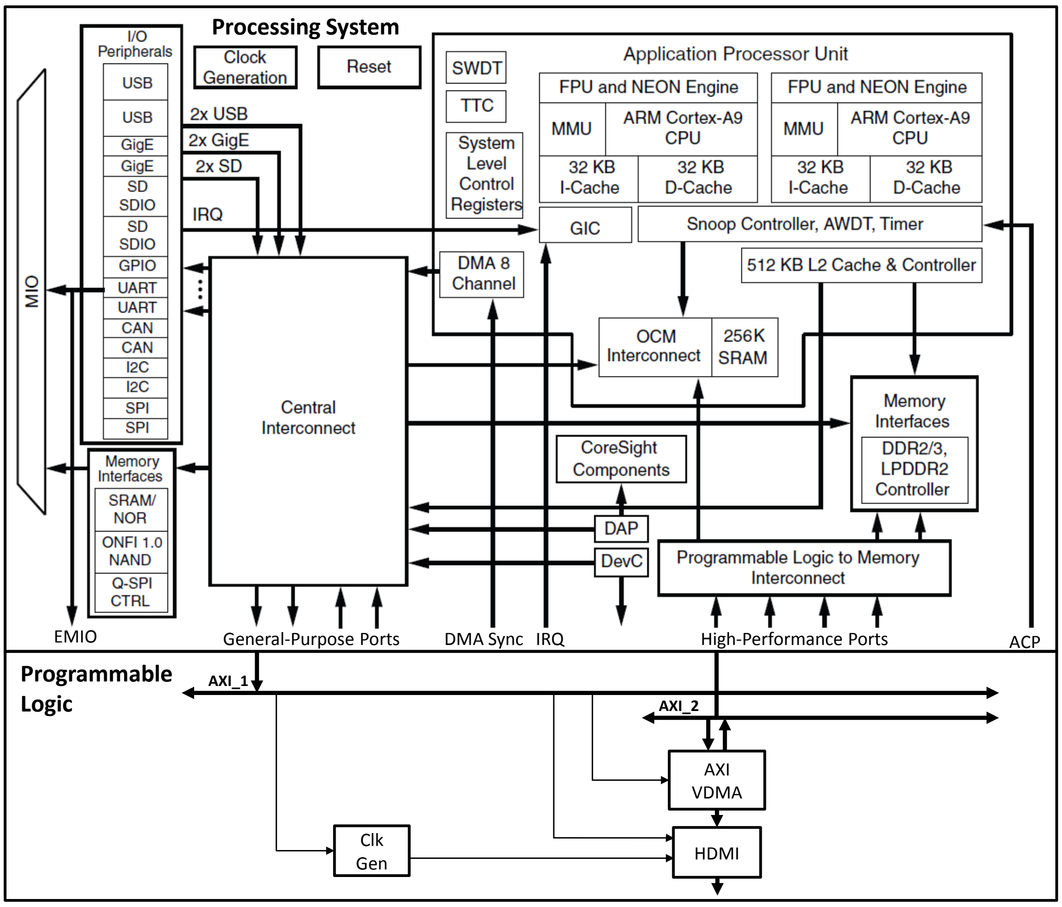


Figure 1.4.: Block diagram of hardware architecture for display Interface

Software application

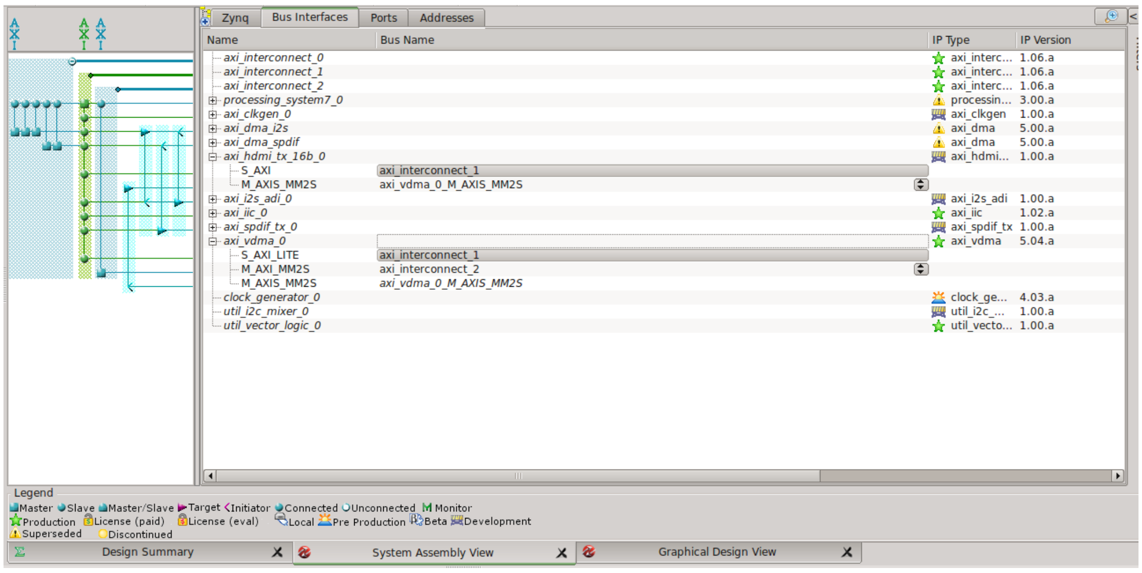
The target application for this lab will be a bare-metal application (no Operating System). The first step during execution will be to initiallize the PS and all the custom hardware modules in the PL. We will use the available driver libraries from Xilinx and other hardware vendors to initiallize and configure the hardware.

During initialization of PS, interrupt services are also initialized. *Handling of interrupts is important for the project, frame read operation from the camera module is a part of Interrupt Service Routine (ISR).* At the start of each new frame, camera module sends an interrupt to CPU, by which CPU jumps into corresponding ISR. In this ISR CPU reads the complete input frame onto a predefined memory location in the DDR memory. More detail on the software algorithm will be discussed later in Algorithm 1.

Tasks:

1.2. XPS Project Creation and set up of the HDMI Display

1. The first step is to set up the project directories and invoke the Xilinx ISE tool on the terminal. Open a new terminal, make a new directory LabC in the <work dir>, and change into this directory.
2. Once inside LabC directory, copy the provided reference project into LabC directory. ($cp -r <project docs>/LabC/videoprocessing/* .
3. Now run *$module load xilinx/ise/14.7* and then *$xps* to invoke Xilinx platform studio.
4. In the XPS environment, open *videoprocessing_prj/system.xmp* project. This will open an already created project for HDMI display. Select the *Bus Interfaces* tab. The system assembly view should look like Figure 1.5.
5. Current reference project provides the video interface between the Zynq FPGA and ADV7511 transmitter chip that controls the HDMI port.



Name	Bus Name	IP Type	IP Version
axi_interconnect_0		axi_interc...	1.06.a
axi_interconnect_1		axi_interc...	1.06.a
axi_interconnect_2		axi_interc...	1.06.a
processing_system7_0		processin...	3.00.a
axi_clkgen_0		axi_clkgen	1.00.a
axi_dma_i2s		axi_dma	5.00.a
axi_dma_spdif		axi_dma	5.00.a
axi_hdmi_tx_16b_0		axi_hdmi...	1.00.a
S_AXI	axi_interconnect_1		
M_AXIS_MM2S	axi_vdma_0_M_AXIS_MM2S		
axi_i2s_adi_0		axi_i2s_adi	1.00.a
axi_iic_0		axi_iic	1.02.a
axi_spdif_tx_0		axi_spdif_tx	1.00.a
axi_vdma_0		axi_vdma	5.04.a
S_AXI_LITE	axi_interconnect_1		
M_AXI_MM2S	axi_interconnect_2		
M_AXIS_MM2S	axi_vdma_0_M_AXIS_MM2S		
clock_generator_0		clock_ge...	4.03.a
util_i2c_mixer_0		util_i2c_...	1.00.a
util_vector_logic_0		util_vecto...	1.00.a

Legend
Master Slave Master/Slave Target Initiator Connected Unconnected Monitor
Production License (paid) License (eval) Local Pre Production Beta Development
Superseded Discontinued
Design Summary
System Assembly View
Graphical Design View

Figure 1.5.: HDMI project system assembly view.

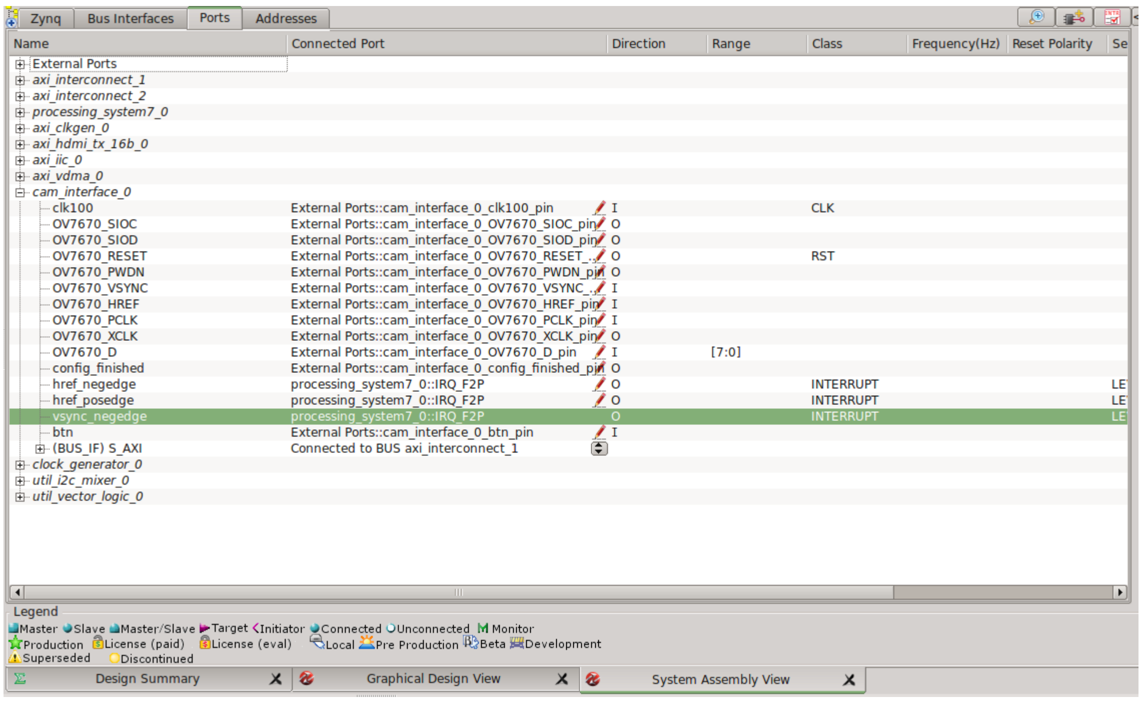
One important IP core in the video interface part is the HDMI Pcore i.e. axi_hdmi_tx_16b_0 (provided by Analog Devices for ADV7511 chip), that reads 24-bits of RGB data from DDR memory and prepares it for the ADV7511 transmitter chip. This core accesses the DDR memory via axi_vdma_0 module, which is an IP core from Xilinx for Video DMA (VDMA). In this design, the *VDMA* streams frame data to the *HDMI interface* core.

6. Expand 'processing_system7_0' to see how the system interconnects with these cores. The AXI4-Lite bus-interface 'axi_interconnect_1' is connected to the general purpose port 'GP0' of the PS. It is used by the CPU to configure parameters of the attached cores. Second AXI interface 'axi_interconnect_2' is connected to high performance port 'HP0' of the PS. The VDMA reads the video frame from the memory through this port. This frame is forwarded to HDMI core which prepares the frame for display and sends it to the ADV7511 transmitter.
7. Select *Ports* tab. Expand axi_hdmi_tx_16b_0 and axi_vdma_0 to see their port interconnections.

1.3. Adding Camera Peripheral Core

As this step, we will add custom hardware core for camera interface to the reference project.

1. We have provided the Pcore IP for camera interface for this project. Copy this Pcore IP into the project folder. $cp -r <project docs>/LabC/cam_interface_v1_00_a <LabC >/videoprocessing_prj/pcores/.
2. On XPS, select *Project>Rescan User Repositories*. Now, the newly added Pcore should get populated in the IP Catalog, under Project Local Pcores.
3. Select the CAM_INTERFACE module in *IP-Catalog* panel, drag it to *System Assembly View* panel and drop it there. Click *Yes* to confirm addition of new instance.
4. *Configuration window* for the CAM_INTERFACE IP will pop-up. Set the *Component Instance Name* as "cam_interface_0" and keeping other settings as default, click *OK* to confirm.
5. In the *Instantiate and Connect IP* dialog, check "Select processor instance to connect to" as processing_system7_0 and click *OK*. Now, the internal connections from cam_interface_0 to processing_system7_0 should be visible in the *Bus Interfaces* tab.
6. For configuration of external connections, go to the *Ports* tab on *System Assembly View*.
7. Expand the ports of cam_interface_0. Make clk100 port as External by right-clicking on clk100, and select *Make External*. Similarly, make all other ports of cam_interface_0 module as external, except the interrupt signals (href_negedge, href_posedge, and vsync_negedge) (Figure 1.6).



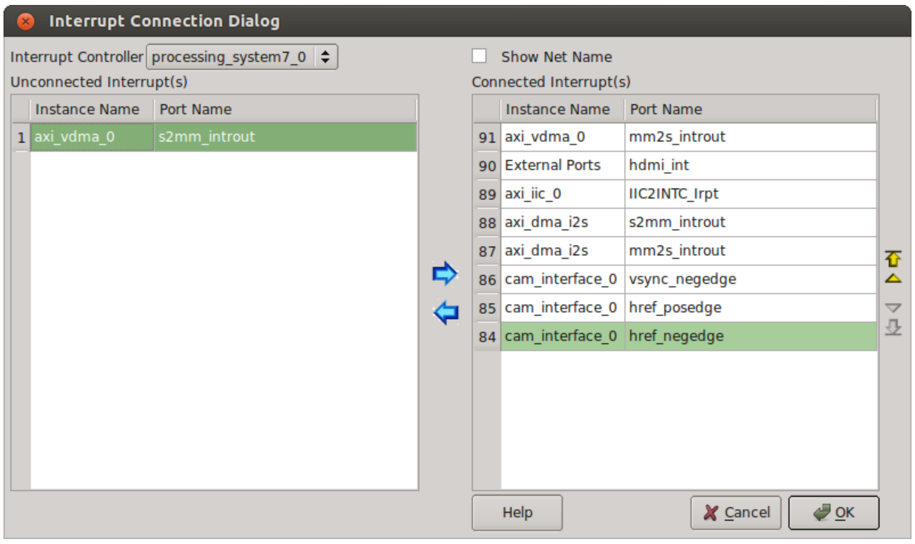
Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Reset Polarity	Se
External Ports							
axi_interconnect_1							
axi_interconnect_2							
processing_system7_0							
axi_clkgen_0							
axi_hdmi_tx_16b_0							
axi_iic_0							
axi_vdma_0							
cam_interface_0							
clk100	External Ports::cam_interface_0_clk100_pin	I		CLK			
OV7670_SIOC	External Ports::cam_interface_0_OV7670_SIOC_pin	O					
OV7670_SIOD	External Ports::cam_interface_0_OV7670_SIOD_pin	O					
OV7670_RESET	External Ports::cam_interface_0_OV7670_RESET_..	O		RST			
OV7670_PWDN	External Ports::cam_interface_0_OV7670_PWDN_pin	O					
OV7670_VSYNC	External Ports::cam_interface_0_OV7670_VSYNC_..	I					
OV7670_HREF	External Ports::cam_interface_0_OV7670_HREF_pin	I					
OV7670_PCLK	External Ports::cam_interface_0_OV7670_PCLK_pin	I					
OV7670_XCLK	External Ports::cam_interface_0_OV7670_XCLK_pin	O					
OV7670_D	External Ports::cam_interface_0_OV7670_D_pin	I	[7:0]				
config_finished	External Ports::cam_interface_0_config_finished_pin	O					
href_negedge	processing_system7_0::IRQ_F2P	O		INTERRUPT			LE
href_posedge	processing_system7_0::IRQ_F2P	O		INTERRUPT			LE
vsync_negedge	processing_system7_0::IRQ_F2P	O		INTERRUPT			LE
btn	External Ports::cam_interface_0_btn_pin	I					
(BUS_IF) S_AXI	Connected to BUS axi_interconnect_1						
clock_generator_0							
util_i2c_mixer_0							
util_vector_logic_0							

Legend
Master Slave Master/Slave Target Initiator Connected Unconnected Monitor
Production License (paid) License (eval) Local Pre Production Beta Development
Superseded Discontinued

Figure 1.6.: Port connections for cam_interface_0 instance.

8. To connect the interrupt ports, left-click on the *Connected Port* column for any interrupt port. In the Interrupt Connection dialog, make the connections for these interrupts to

CPU by moving them from the *Unconnected Interrupts* to *Connected Interrupts* column. Interrupt IDs for these interrupts are automatically assigned. Interrupt connections should look as in Figure 1.7.



Interrupt Controller processing_system7_0

Unconnected Interrupt(s)

	Instance Name	Port Name
1	axi_vdma_0	s2mm_introut

Connected Interrupt(s)

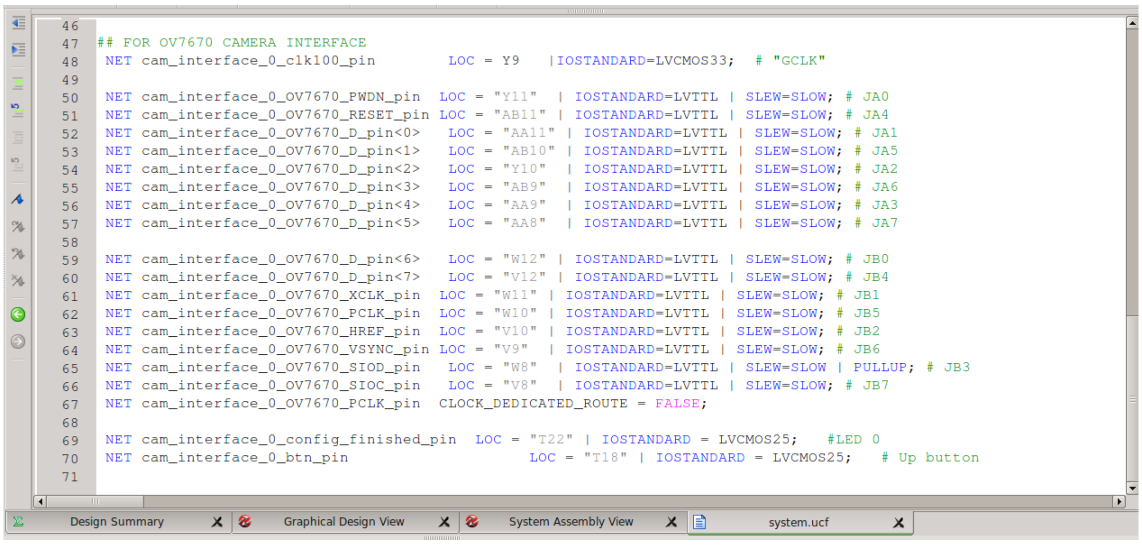
	Instance Name	Port Name
91	axi_vdma_0	mm2s_introut
90	External Ports	hdmi_int
89	axi_iic_0	IIC2INTC_Irpt
88	axi_dma_i2s	s2mm_introut
87	axi_dma_i2s	mm2s_introut
86	cam_interface_0	vsync_negedge
85	cam_interface_0	href_posedge
84	cam_interface_0	href_negedge

Figure 1.7.: Interrupt connections for CAM_INTERFACE to CPU interrupts.

9. For making the external connection of cam_interface_0 instance to Pmod connectors, external pin-mapping need to be defined in UCF file. Under Project tab on XPS (on left), double-click on the UCF File: *data/system.ucf*. At the end of *system.ucf* file, add these pin-mappings for Pmod connections (Figure 1.8),

```
## FOR OV7670 CAMERA INTERFACE
NET cam_interface_0_clk100_pin         LOC = Y9      |IOSTANDARD=LVCMOS33;
            # "GCLK"

NET cam_interface_0_OV7670_PWDN_pin  LOC = "Y11"   | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA1
NET cam_interface_0_OV7670_RESET_pin LOC = "AB11"  | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA7
NET cam_interface_0_OV7670_D_pin<0>  LOC = "AA11"  | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA2
NET cam_interface_0_OV7670_D_pin<1>  LOC = "AB10"  | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA8
NET cam_interface_0_OV7670_D_pin<2>  LOC = "Y10"   | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA3
NET cam_interface_0_OV7670_D_pin<3>  LOC = "AB9"   | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA9
NET cam_interface_0_OV7670_D_pin<4>  LOC = "AA9"   | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA4
NET cam_interface_0_OV7670_D_pin<5>  LOC = "AA8"   | IOSTANDARD=LVTTL
                    | SLEW=SLOW; # JA10

NET cam_interface_0_OV7670_D_pin<6>  LOC = "W12"  | IOSTANDARD=LVTTL
```

```
                        | SLEW=SLOW; # JB1
NET cam_interface_0_OV7670_D_pin<7>  LOC = "V12"  | IOSTANDARD=LVTTL
                        | SLEW=SLOW; # JB7
NET cam_interface_0_OV7670_XCLK_pin  LOC = "W11"  | IOSTANDARD=LVTTL
                        | SLEW=SLOW; # JB2
NET cam_interface_0_OV7670_PCLK_pin  LOC = "W10"  | IOSTANDARD=LVTTL
                        | SLEW=SLOW; # JB8
NET cam_interface_0_OV7670_HREF_pin  LOC = "V10"  | IOSTANDARD=LVTTL
                        | SLEW=SLOW; # JB3
NET cam_interface_0_OV7670_VSYNC_pin LOC = "V9"   | IOSTANDARD=LVTTL
                        | SLEW=SLOW; # JB9
NET cam_interface_0_OV7670_SIOD_pin  LOC = "W8"   | IOSTANDARD=LVTTL
                        | SLEW=SLOW | PULLUP; # JB4
NET cam_interface_0_OV7670_SIOC_pin  LOC = "V8"   | IOSTANDARD=LVTTL
                        | SLEW=SLOW; # JB10

NET cam_interface_0_OV7670_PCLK_pin  CLOCK_DEDICATED_ROUTE = FALSE;

NET cam_interface_0_config_finished_pin LOC = "T22"
                        | IOSTANDARD = LVCMOS25;     #LED 0
NET cam_interface_0_btn_pin             LOC = "T18"
                        | IOSTANDARD = LVCMOS25;     # Up button
```

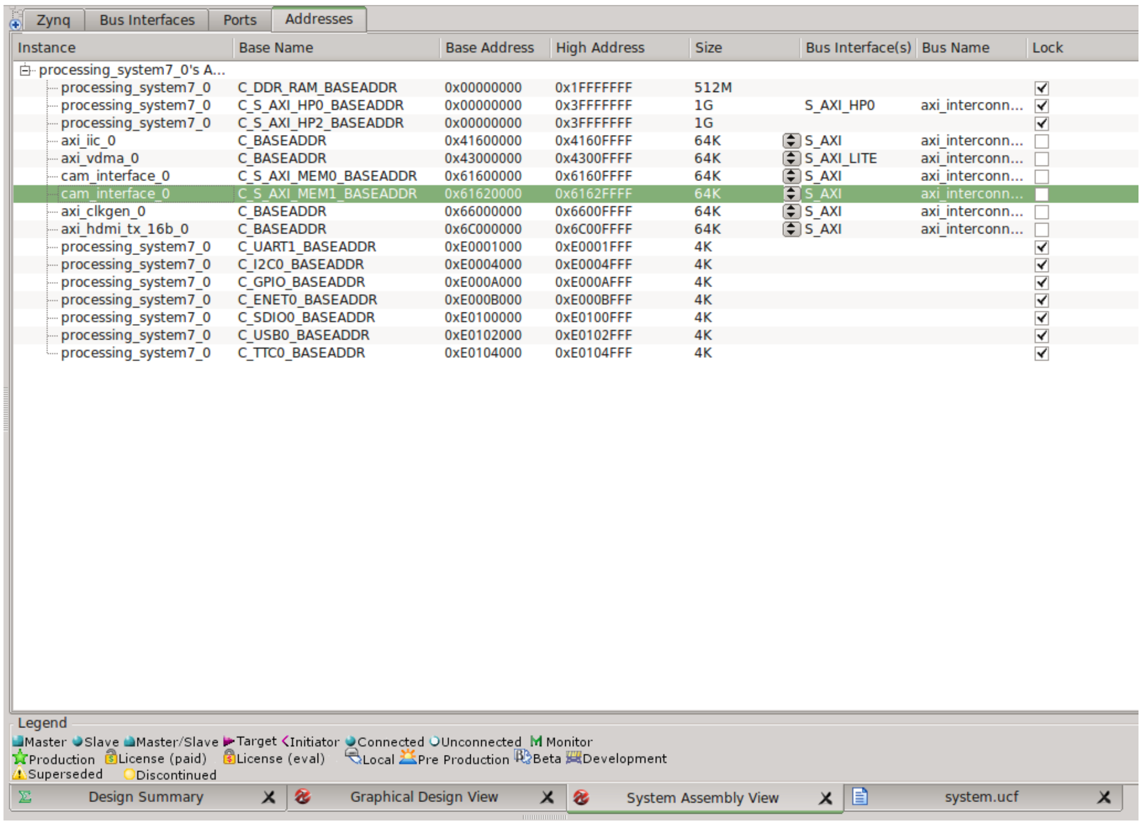


```
## FOR OV7670 CAMERA INTERFACE
 NET cam_interface_0_clk100_pin        LOC = Y9    |IOSTANDARD=LVCMOS33;  # "GCLK"

 NET cam_interface_0_OV7670_PWDN_pin  LOC = "Y11"  | IOSTANDARD=LVTTL | SLEW=SLOW; # JA0
 NET cam_interface_0_OV7670_RESET_pin LOC = "AB11" | IOSTANDARD=LVTTL | SLEW=SLOW; # JA4
 NET cam_interface_0_OV7670_D_pin<0>   LOC = "AA11" | IOSTANDARD=LVTTL | SLEW=SLOW; # JA1
 NET cam_interface_0_OV7670_D_pin<1>   LOC = "AB10" | IOSTANDARD=LVTTL | SLEW=SLOW; # JA5
 NET cam_interface_0_OV7670_D_pin<2>   LOC = "Y10"  | IOSTANDARD=LVTTL | SLEW=SLOW; # JA2
 NET cam_interface_0_OV7670_D_pin<3>   LOC = "AB9"  | IOSTANDARD=LVTTL | SLEW=SLOW; # JA6
 NET cam_interface_0_OV7670_D_pin<4>   LOC = "AA9"  | IOSTANDARD=LVTTL | SLEW=SLOW; # JA3
 NET cam_interface_0_OV7670_D_pin<5>   LOC = "AA8"  | IOSTANDARD=LVTTL | SLEW=SLOW; # JA7

 NET cam_interface_0_OV7670_D_pin<6>   LOC = "W12" | IOSTANDARD=LVTTL | SLEW=SLOW; # JB0
 NET cam_interface_0_OV7670_D_pin<7>   LOC = "V12" | IOSTANDARD=LVTTL | SLEW=SLOW; # JB4
 NET cam_interface_0_OV7670_XCLK_pin  LOC = "W11" | IOSTANDARD=LVTTL | SLEW=SLOW; # JB1
 NET cam_interface_0_OV7670_PCLK_pin  LOC = "W10" | IOSTANDARD=LVTTL | SLEW=SLOW; # JB5
 NET cam_interface_0_OV7670_HREF_pin  LOC = "V10" | IOSTANDARD=LVTTL | SLEW=SLOW; # JB2
 NET cam_interface_0_OV7670_VSYNC_pin LOC = "V9"  | IOSTANDARD=LVTTL | SLEW=SLOW; # JB6
 NET cam_interface_0_OV7670_SIOD_pin   LOC = "W8"  | IOSTANDARD=LVTTL | SLEW=SLOW | PULLUP; # JB3
 NET cam_interface_0_OV7670_SIOC_pin   LOC = "V8"  | IOSTANDARD=LVTTL | SLEW=SLOW; # JB7
 NET cam_interface_0_OV7670_PCLK_pin  CLOCK_DEDICATED_ROUTE = FALSE;

 NET cam_interface_0_config_finished_pin  LOC = "T22" | IOSTANDARD = LVCMOS25;   #LED 0
 NET cam_interface_0_btn_pin                    LOC = "T18" | IOSTANDARD = LVCMOS25;   # Up button
```

Figure 1.8.: Pin-mapping for Pmod connections for cam_interface_0 external ports.

At this stage, please confirm on *Addresses* tab in XPS that addresses for the new modules are automatically assigned, which can be seen in the *Base Address* column (as shown in Figure 1.9)

10. Click *Run DRCs* in the *Navigator panel* (on left) to verify the design connections. In this step there will be some warnings. **NOTE:** There could be some bogus *error messages EDK:4207, EDK:4208, and EDK:4209*, please ignore those.



Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
processing_system7_0's A...							
processing_system7_0	C_DDR_RAM_BASEADDR	0x00000000	0x1FFFFFFF	512M			✓
processing_system7_0	C_S_AXI_HP0_BASEADDR	0x00000000	0x3FFFFFFF	1G	S_AXI_HP0	axi_interconn...	✓
processing_system7_0	C_S_AXI_HP2_BASEADDR	0x00000000	0x3FFFFFFF	1G			✓
axi_iic_0	C_BASEADDR	0x41600000	0x4160FFFF	64K	S_AXI	axi_interconn...	☐
axi_vdma_0	C_BASEADDR	0x43000000	0x4300FFFF	64K	S_AXI_LITE	axi_interconn...	☐
cam_interface_0	C_S_AXI_MEM0_BASEADDR	0x61600000	0x6160FFFF	64K	S_AXI	axi_interconn...	☐
cam_interface_0	C_S_AXI_MEM1_BASEADDR	0x61620000	0x6162FFFF	64K	S_AXI	axi_interconn...	☐
axi_clkgen_0	C_BASEADDR	0x66000000	0x6600FFFF	64K	S_AXI	axi_interconn...	☐
axi_hdmi_tx_16b_0	C_BASEADDR	0x6C000000	0x6C00FFFF	64K	S_AXI	axi_interconn...	☐
processing_system7_0	C_UART1_BASEADDR	0xE0001000	0xE0001FFF	4K			✓
processing_system7_0	C_I2C0_BASEADDR	0xE0004000	0xE0004FFF	4K			✓
processing_system7_0	C_GPIO_BASEADDR	0xE000A000	0xE000AFFF	4K			✓
processing_system7_0	C_ENET0_BASEADDR	0xE000B000	0xE000BFFF	4K			✓
processing_system7_0	C_SDIO0_BASEADDR	0xE0100000	0xE0100FFF	4K			✓
processing_system7_0	C_USB0_BASEADDR	0xE0102000	0xE0102FFF	4K			✓
processing_system7_0	C_TTC0_BASEADDR	0xE0104000	0xE0104FFF	4K			✓

Figure 1.9.: Address mapping for newly added IP modules.

11. Next click *Generate Bitstream* in the Navigator panel, to generate the bit file.
12. Bitstream generation will take some time (approx. 10mins). It should complete successfully with "Done!" message on the console.

Questions:

1. What are the different interfaces available on ZedBoard to connect external devices?
2. For the frame timing information given in figure 1.3, what is the maximum frame rate (number of frames per second) possible with the input pixel clock of 16MHz? It is given that it requires 4 bytes to contain each pixel.
3. How is external camera OV7670 configured. Give the control register settings (ref. to section 5.3 and provided hardware files).
4. What is the role of *AXI_LITE* in the design (AXI_1)?
5. Describe briefly the VDMA IP core in the design and how is it used to display video stream on the external monitor?

1.4. Software Application Project in SDK

1. On XPS, click *Export Design* in the *Navigator tab* to export the hardware. Next, in the *Export to SDK* window check *Include bitstream and BMM file option* and click *Export &*

Launch SDK.

2. Here export to sdk process will start on XPS. At the end, SDK will start and will ask for the workspace location. Change the workspace path to <LabC>/videoProcessing_prj/SDK/SDK_Workspace.

3. On the SDK, ps7 configuration files will automatically be generated as we have seen in LabA.

4. To generate the BSP (Board Support Package) for this project follow the steps as in LabA. Select File>New>Board Support Package.

5. Create new application project. On *New Project* window, set Project Name: "video_filter_sw". Select BSP as (on bottom) Use-existing: standalone_bsp_0. Click *Next*, select *Empty Application* in *Available Templates*, then click *Finish*. New project video_filter_sw will be created under Project Explorer panel.

6. Now we will add the software application and driver files into the project, by right-click on *video_filter_sw>Import...* . In the *Import* window, expand General, select File System, click Next. On From directory, click Browse... and point to video_filter_sw_doc directory under project_documents. The directories under video_filter_sw_doc can be seen as in Figure 1.10. Select all three sub-directories (/src, /inc, and /lib). Click *Finish* to add those to the project.

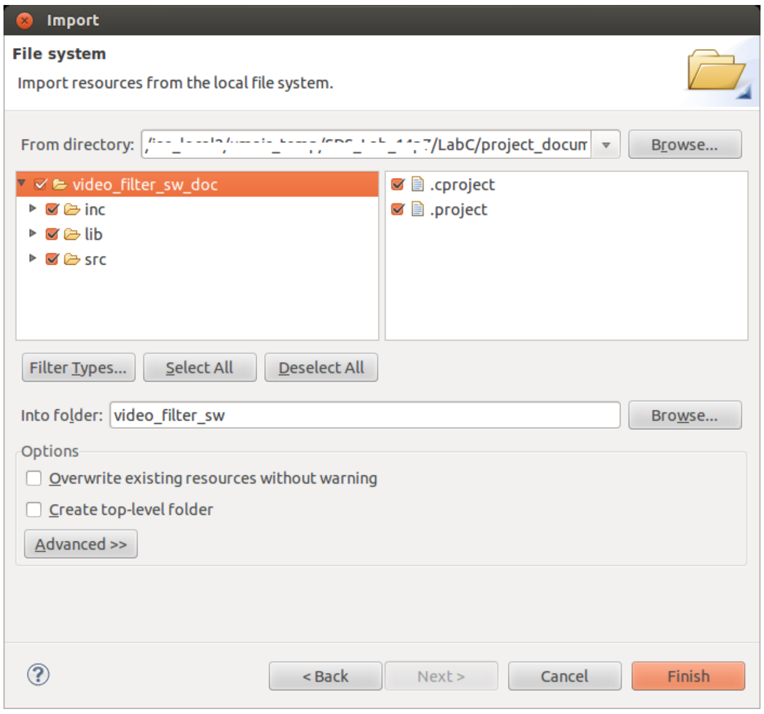
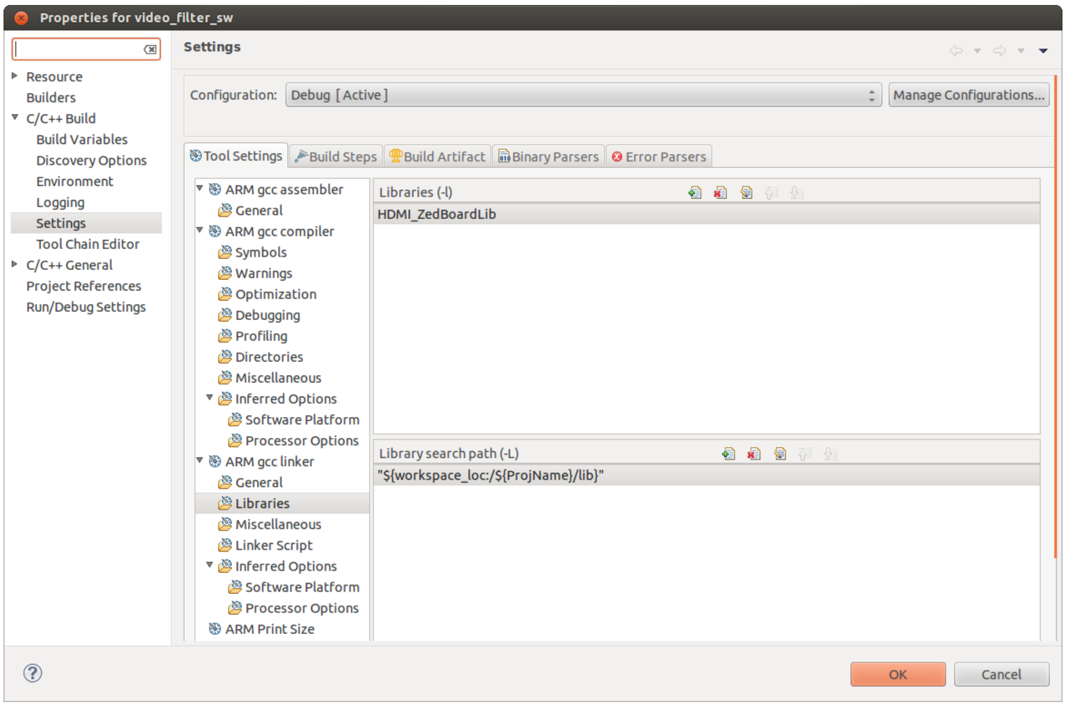

Import
File system
Import resources from the local file system.
From directory: [illegible]/LabC/project_docum
☑ video_filter_sw_doc
☑ inc
☑ lib
☑ src
☑ .cproject
☑ .project
Into folder: video_filter_sw
Options
☐ Overwrite existing resources without warning
☐ Create top-level folder

Figure 1.10.: Importing the source application and driver files for software application.

7. The next step is to change the project settings to include the newly added directories in the previous step (for libraries and header files). Right-click on *video_filter_sw >C/C++ Build Settings*. On *Properties for video_filter_sw* window, *Settings* under *C/C++ Build* (left-side) should be selected. In the *Settings* panel, enable *Tool Settings* tab. Select *Directories* under *ARM gcc compiler*. Click on '+' on *Include Paths* panel. On *Add directory path* dialog,

click *Workspace*. Select *inc* under video_filter_sw, and click *OK*. Then click *OK* on *Add directory path* dialog.

8. On *Properties* window, select *Libraries* under *ARM gcc linker*. On *Libraries (-l)* panel, click "+" , set the value to "HDMI_ZedBoardLib" and click *OK*. In the *Library search path (-L)* panel, click "+". On *Add directory path* dialog, click *Workspace...*, select lib under video_filter_sw and click *OK*. Click *OK* on *Add directory path dialog* (Figure 1.11). Then, click *OK* to accept the project settings.



Configuration: Debug [Active]

Libraries (-l)
HDMI_ZedBoardLib

Library search path (-L)
"${workspace_loc:/${ProjName}/lib}"

Figure 1.11.: Library settings for the project.

1.5. Hardware set up for Monitor and Camera

1. Set up the ZedBoard as we did it in LabA, including the PROG/UART connections to the USB ports of the PC, and the jumper settings.
2. Connect the provided HDMI cable to HDMI socket (J9) on ZedBoard and the other end of the cable (which is DVI port) to the monitor.
3. For Camera connections, connect 18 jumper cables from camera ports to Pmod ports (JA, JB) on ZedBoard according to table 1.1.
4. Power ON the ZedBoard (SW8), green power LED (LD13) will glow.

1.6. FPGA Programming and Software Application Execution on ZedBoard

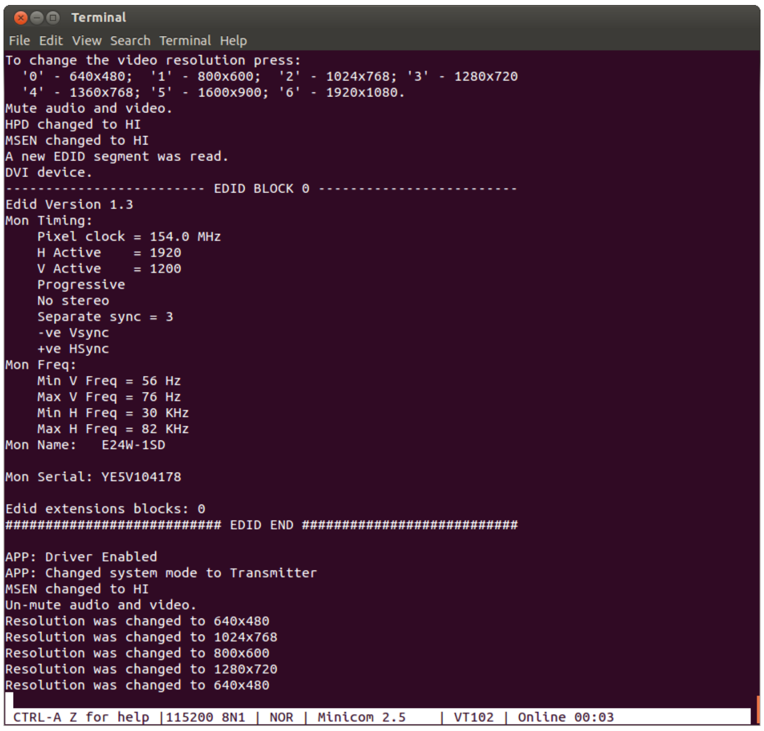
1. Open *minicom* on PC terminal by typing *$minicom -s*, make the settings as earlier.

Name	**Type**	**Description**	**Pmod connection**	**Zynq Pin connection**
SDIOC	Input	SCCB clock	JB10	V8
SDIOD	In/Out	SCCB data	JB4	W8
VSYNC	Output	Vertical synchronization	JB9	V9
HREF	Output	Horizontal synchronization	JB3	V10
PCLK	Output	Pixel clock	JB8	W10
XCLK	Input	System clock	JB2	W11
D_0	Output	Video data	JA2	AA11
D_1	Output	Video data	JA8	AB10
D_2	Output	Video data	JA3	Y10
D_3	Output	Video data	JA9	AB9
D_4	Output	Video data	JA4	AA9
D_5	Output	Video data	JA10	AA8
D_6	Output	Video data	JB1	W12
D_7	Output	Video data	JB7	V12
RESET	Input	Reset (Active low)	JA7	AB11
PWDN	Input	Power down (Active high)	JA1	Y11
GND	Ground	Camera ground port	GND (JA/JB)	
3V3	Power	Camera power port	VCC (JA/JB)	

Table 1.1.: Pin connections between camera device (OV7670) and ZedBoard

2. On SDK window, click *Program FPGA*, or select *XilinxTools>Program FPGA* with *Bitfile* as *<LabC>/videoprocessing_prj/SDK/SDK_Workspace/videoprocessing_prj_hw_platform/ system.bit.*
3. On successful programming of FPGA, the blue LED (LD12) will glow.
4. IMPORTANT: To reset the camera interface module, press Push Button (T18, BTNU) on ZedBoard long enough to make red LED (LD0) to blink.
5. To run the software on ZedBoard, select Run>Run Configurations... . On the Run Configurations window, double-click *Xilinx C/C++ application (GDB)* (on left) to create a new run configuration. Keep all other settings default and click *Run*.
6. On successful execution of the software application on ZedBoard, the monitor will start displaying the video from the camera and *minicom* will print the messages as shown in Figure 1.12.

   Algorithm 1 describes the software application algorithm,



```
To change the video resolution press:
  '0' - 640x480;  '1' - 800x600;  '2' - 1024x768; '3' - 1280x720
  '4' - 1360x768; '5' - 1600x900; '6' - 1920x1080.
Mute audio and video.
HPD changed to HI
MSEN changed to HI
A new EDID segment was read.
DVI device.
------------------------- EDID BLOCK 0 -------------------------
Edid Version 1.3
Mon Timing:
    Pixel clock = 154.0 MHz
    H Active    = 1920
    V Active    = 1200
    Progressive
    No stereo
    Separate sync = 3
    -ve Vsync
    +ve HSync
Mon Freq:
    Min V Freq = 56 Hz
    Max V Freq = 76 Hz
    Min H Freq = 30 KHz
    Max H Freq = 82 KHz
Mon Name:   E24W-1SD

Mon Serial: YE5V104178

Edid extensions blocks: 0
########################### EDID END ###########################

APP: Driver Enabled
APP: Changed system mode to Transmitter
MSEN changed to HI
Un-mute audio and video.
Resolution was changed to 640x480
Resolution was changed to 1024x768
Resolution was changed to 800x600
Resolution was changed to 1280x720
Resolution was changed to 640x480

CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.5    | VT102 | Online 00:03
```

Figure 1.12.: *minicom* terminal messages for the software application.

```
File: main.c
Data: Camera video stream
Result: Output stream on display, after processing
main() {
<HW Platform initialization >
<ADI peripherals initialization (from Analog devices) >
<HDMI configuration >
<CPU Interrupt controller initilization >
while true do
    if <Keyboard input to change resolution > then
        <Change HDMI configuration to new resolution >
    end
end
return;
}

File: axi_interrupt.c
AXI_INTERRUPT_VsyncIntr_Handler() {
        <DDRVideoWr: Read form Camera
                              Write to DDR memory>
}
```

Algorithm 1: Software for ZedBoard video application.

There are two main files in which the application code resides; *main.c* and *axi_interrupt.c*. *main.c* contains the main function, which does the device initiallization by calling the driver functions for different hardwares, initialization of the interrupt controller, and finally it goes into infinite loop to monitor user input for changing the desplay resolution. *axi_interrupt.c* contains the functions for interrupt service routines. *AXI_INTERRUPT_VsyncIntr_Handler()* is a function which handles the new frame interrupt from the camera module. At each new frame interrupt CPU reads the complete incoming video frame and writes it to the DDR memory location.

7. To stop the program execution, press 'q' on *minicom* terminal. Return to the SDK window.

1.7. Software Application for Gray Scale Conversion

Grayscale conversion of colored image is a single pixel operation. A colored image consists of R, G and B values, whereas a grayscale image only carries image intensity, composed of scales of gray. A straight forward way to convert a color pixel is to take an average value of three color channels. The *luminosity* method is a more sophisticated version of the average method. It calculates the weighted average to account for human perception. As human eye is more sensitive to green than other colors, so green is weighted most heavily. Equation 1.1 and 1.2 represent the formulas for *average* and *luminosity* methods respectively.

$$Y = (R + G + B)/3 \tag{1.1}$$
$$Y = .114R + .587G + .299B \tag{1.2}$$

Name	**Description**
axi_interrupt.h/.c	ISR for CPU interrupts
cf_hdmi.h/.c	DDRVideoWr(): Write video frames (size: 640x480) to DDR from Camera input. Set peripheral base addresses for software. Initialize HDMI Video Pcore.
main.c	Application code. Interrupt controller initialization.
profile_cnt.h	Processor profiling functions.
transmitter.ch	ADI peripheral initialization functions.

Table 1.2.: Description of important files in /*scr* directory.

1. Software functions required for video processing are provided to you at this path <project documents>/LabC/sw_functions/. To add these functions to the project, in *Project Explorer* panel, rightclick on *src*, select *Import...*. In the *Import* window, select *File System* under *General*, click *Next*. In *From directory* browse to <project docs>/LabC/sw_functions/, click *OK*. Now, select both the files (sw_function.c and sw_function.h) and click *Finish*.
2. Now, use these provided function (*convToGrayHLS*)in the *application code* appropriately, to perform the Gray scale conversion operation on the input video. (Hint: Consult Algorithm 1, Table 1.2, and project description in introduction section).

3. As a next step, modify the *application code* such that the monitor displays raw video input by default. On pressing 'p' it should display processed video and on pressing 'r' key display monitor should switch to raw video input.

After completion of above steps, you will have a working video processing system. This ends Lab C and in the next lab we will do the hardware acceleration of gray scale software function.

Questions:

6. Which peripherals are connected on *axi_interconnect_0, axi_interconnect_1, and axi_interconnect_2* at the end of complete hardware set up for this lab?
7. Refering to the software implementation, describe how the camera video data is read?
8. How are RGB values extracted from the pixel data? (ref. to SW implementation)

Bibliography

[1] ITRS-2011, *www.itrs.net*

[2] ANSI, IEEE Standards Board, IEEE Standard VHDL Language Reference Manual: IEEE Std 1076-1993, New York, 1988, ISBN 1559373768

[3] Peter J. Ashenden, Designer's Guide to Vhdl, Morgan Kaufmann Publishers, 1995, ISBN 1558602704

[4] Giovanni De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill Book Company, 1994, ISBN 0070163332

[5] Jayaram Bhasker, A VHDL Primer, Prentice Hall, Englewood Cliffs, 1998, ISBN 0130965758

[6] Zedboard, www.zedboard.org.

[7] Zedboard Hardware User's Guide v1.1, Aug 2012, Digilent.

[8] LogiCORE IP AXI Video Direct Memory Access v6.1, PG020, Dec 2013.

[9] Xilinx, www.xilinx.com.

[10] Vivado Design Suite Tutorial, UG871, June 2013, Xilinx.

[11] OV7670 (2006): OV7670/OV7171 CMOS VGA (640x480) CAMERACHIPT M with OmniPixel Technology, OmniVision Technologies Inc.

[12] PG, ADV7511 (2012): Low-Power HDMI Transmitter Programming Guide, Analog Devices, Inc.

[13] Zynq-7000 All Programmable SoC TRM v1.7, UG585, Feb 2014, Xilinx.

[14] Vivado Design Suite User Guide, High-Level Synthesis, UG902(v2013.2), June 19, 2013, Xilinx.

[15] Introduction to Zynq-7000^{TM} All Programmable SoC, Speedway.

[16] Umair Razzaq, Design Space Exploration for Embedded Vision Applications on Zynq FPGA using HLS, 2013, Master Thesis, EDA-TUM.