Lab Manual

Laboratory

Synthesis of Digital Systems

Prof. Dr.-Ing. Ulf Schlichtmann

June 11, 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 D: Video processing on ZedBoard and Hardware Implementation of Gray Scale Filter

In this lab we will use Vivado HLS to synthesize C/C++ implementation of the video processing algorithm into RTL and integrate it into the Zynq PS design. We will use a synthesizable software implementation of the Gray scale conversion filter. Using Vivado HLS tool, we transform the C/C++ functions into RTL for desired clock frequency and hardware platform. Finally, the RTL implementation will be packaged into a standard IP format like IP-XACT or Pcore that will be used by other Xilinx tools for IP integration into hardware accelerated video processing.

Tasks:

1. Vivado project set up.
2. HDL generation from C-code and IP-generation.
3. IP-integration of filter IP to the system.
4. Software implementation for hardware accelerated system.
5. Application profiling.
6. Conclusions.

1.1. Vivado Project Set up

1. Open a new terminal, make a new directory LabD: *$mkdir LabD*. Create another directory *GrayScaleIP* within *LabD* directory, and go into *GrayScaleIP* directory.
2. Copy following project files from <project_documents>/LabD into the *<LabD>/GrayScaleIP* directory: *opencv_top.cpp, opencv_top.h, test_1080p.bmp, top.cpp and top.h.* These files give the software description for *Gray Scale filter*, for which we will be doing the HLS.
3. Inside *<LabD>/GrayScaleIP* directory, run $module load xilinx/vivado/2013.3 and then $vivado_hls to invoke Vivado-HLS.
4. Now we will create the Vivado project for Gray scale filter. Create a new project *"greyscale_prj"* with top function *"gray_scale"* in Vivado. In Design Files click *Add Files...* and browse to <project directory>/LabD/GrayScaleIP folder and add *top.cpp*.
5. In *Solution Configuration window*, set Solution Name: solution1; Clock Period: 150MHz (no space, also see note below); Uncertainty: <keep it blank>. Click *... (browse)* on *Part Selection*. Select *Zedboard (xc7z020clg484-1)* from the list of boards. **Note:** Clock Period/frequency should match the clock of the system bus to which it is attached.

Vivado_HLS window now should look like as in Figure 1.1. The new project 'grayscale_prj' will be created in the Vivado-HLS with all the files for Sources included. The project holds information on the design sources, and respective solutions.

Open the source code for review *top.cpp* under *Source* hierarchy in the *explorer pane*. In this design we will use the functions from *hls namespace* to perform graysacle color conversion. This function is implemented in the Vivado video library.

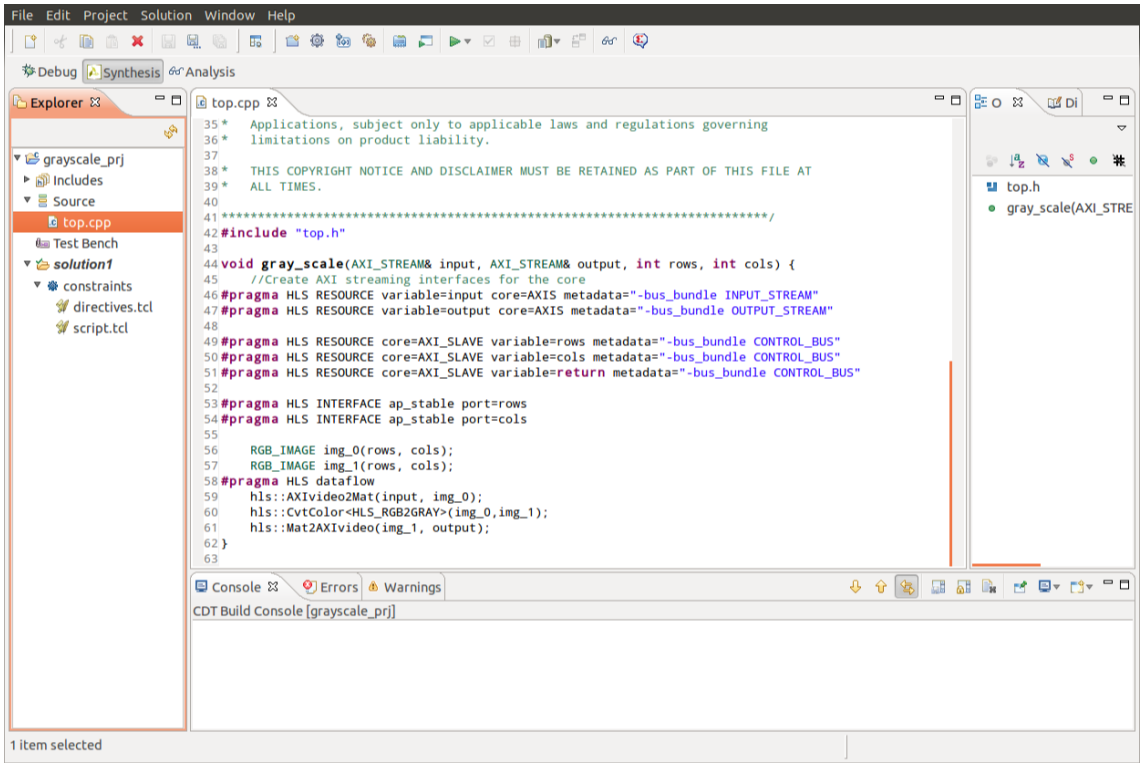

Figure 1.1.: Vivado_HLS window and top.cpp source code

As Xilinx design blocks use AXI4 streaming interface to transfer videos, the input and output images are represented as hls::stream format. Description about the usage of *#pragma* directives is given in the next section. hls::CvtColor <HLS_RGB2GRAY>(img_0, img_1) function provides a synthesizable C++ implementation of the grayscale filter. The AXIvideo2Mat function receives a sequence of images using the AXI4 streaming video and produces a hls::Mat representation. Conversely, the Mat2AXIvideo function receives a hls::Mat representation of a sequence of images and encodes it correctly into the AXI4 streaming video protocol.

1.2. HDL Generation from C-code and IP-generation

Now we will proceed with the synthesis of our design. The high level synthesis performs *Interface synthesis* as well as *algorithm synthesis* on the C implementation. *Interface synthesis* transforms the function parameters into RTL ports with specific protocol. Therefore, proper interface formats and IO protocols must be defined. This is done using the *#pragma* directives. Besides the video library functions, the synthesizable code shown in Figure 1.1 also contains a number of *#pragma* directives that directs interfacing protocol. First two directives (lines 46-47 Fig. 1.1) specify the input and output streams as AXI4 streaming interfaces (core=AXIS). They set the IP port name as INPUT_STREAM and OUTPUT_STREAM. The next three directives (lines 49-51) specify *rows, col and return* values as slave interface (core=AXI_SLAVE) and assign them to the same port *CONTROL_BUS*. In addition, *rows* and *cols* are specified to use *ap_stable* IO protocol in the next directives (lines 53-54).

After generating the RTL implementation, it will be packaged into a standard IP format like

IP-XACT or Pcore, that can be used by other Xilinx tools.

1. To synthesize the C-code, run *C Synthesis*. Synthesis should go through without errors and synthesis report will open automatically at the end of synthesis.

   A new directory named *syn* will be created under the project hierarchy that contains the RTL implementation file of our design.

2. Click on the *Export RTL* toolbar button to export the design as *PCore for EDK*.

   A new directory *impl* will be created in the project hierarchy. This directory contains the standard IP core of the RTL design as well as its software functions. We will use this IP in the next steps and attach it with the Zynq PS. Figure 1.2 shows the new generated directories *syn and impl*, and the corresponding pcore IP.

3. Close Vivado_HLS after Pcore generation.

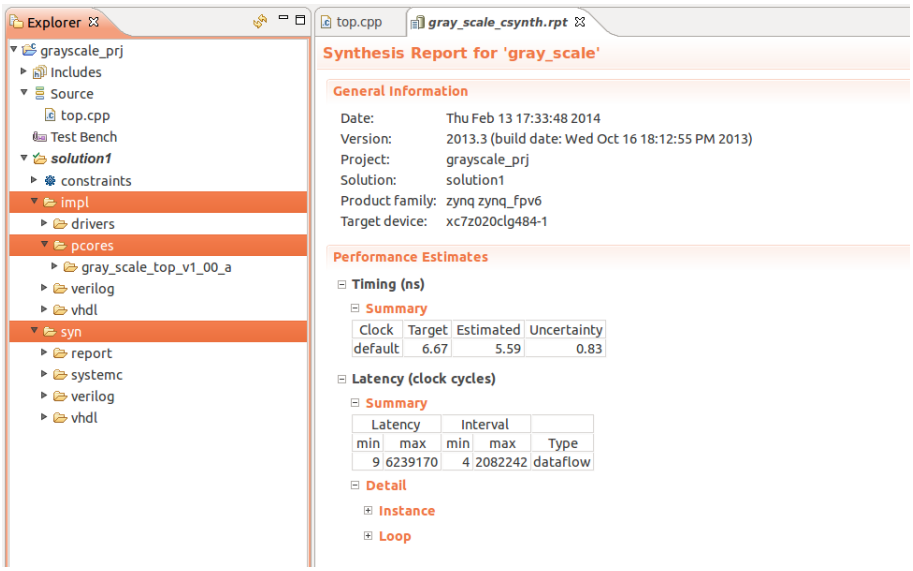

Figure 1.2.: Vivado_HLS window after IP creation.

1.3. IP-integration of Filter IP to the System

1. On terminal, go back to directory *<project directory >/LabD/*.
2. Copy the previously completed XPS project from *<LabC >/videoprocessing_prj* into *<work dir>/LabD/*. directory.
3. Also, copy the Pcore libraries for HDMI cores into the current LabD directory. *$cp -r <project directory >/LabC/cf_lib <project directory >/LabD/*.
4. Now copy the previously generated Pcore for gray scale filter, from path <LabD >/GrayScaleIP/ grayscale_prj/solution1/impl/pcores/* to <work dir>/LabD/ videoprocessing_prj/pcores/. Now the directory structure for LabD should look like in Figure 1.3.
5. Now, load the Xilinx ISE environment *$module load xilinx/ise/14.7* and start *xps*.
6. Open the project on XPS, select *File >Open Project....* Browse and select LabD/<project name >/system.xmp. Click *Open*

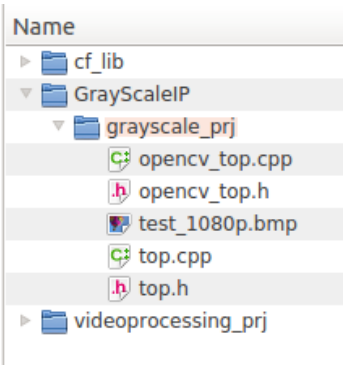

Figure 1.3.: Directory structure for LabD after copying necessary files.

7. On XPS to update the IP-catalog, do the rescan. *Project>Rescan User Repositories*. Now you should see the user IPs as in Figure 1.4 in the IP-catalog panel.
8. Now we will add a dedicated AXI bus-interconnect on which the Gray Scale hardware accelerator will be connected. Activate the *Bus Interfaces* tab on *System Assembly View*. From *IP Catalog* panel, select and drag *AXI Interconnect 1.06.a* under *Bus and Bridge* to *System Assembly View*. Click *Yes* to confirm. Set *Component Instance Name: axi_interconnect_3* and keep other settings as default and then click *OK*.
9. To define the clock for new *axi_interconnect_3*. In *zynq* tab, set *FCLK_CLK3: 150MHz* in *Clock Generation* (which is same as the clock frequency which we have used in Pcore creation for *GrayScaleIP filter*). Click *Validate Clocks* and then *OK*.
10. Now, select the *Ports* tab, expand *axi_interconnect_3* and make following connections,
    - *INTERCONNECT_ACLK* of *axi_interconnect_3* = *FCLK_CLK3* of *processing_system7_0*
    - *INTERCONNECT_ARESETN* of *axi_interconnect_3* = *FCLK_RESET3_N* of *processing_system7_0*
11. Similarly, add *AXI Video DMA* under *DMA and Timer* and *gray_scale_top* to the system. Name the new VDMA instance as *axi_vdma_1* and *gray_scale_top* instance as *gray_scale_top_0*. Keep them unconnected for the moment by selecting *User will make necessary connections and settings* in the configuration dialog.
12. We will need a bus-slave for the *VMDA* master in the *PS*. For this on *Zynq* tab, click on *High Performance AXI 32b/64b Slave Ports*. In the *XPS Core Config* expand *S_AXI*, select *Enable AXI PS Slave #1* (Figure 1.5), and click *OK*.
13. Make following Bus-Interfaces for *axi_vdma_1*,
    - *S_AXI_LITE* = *axi_interconnect_1*
    - *M_AXI_MM2S* & *M_AXI_S2MM* = *axi_interconnect_3*
    - *S_AXIS_S2MM* = *gray_scale_top_0_OUTPUT_STREAM*
14. For *gray_scale_top_0* make these connections,
    - *S_AXI_CONTROL_BUS* = *axi_interconnect_1*
    - *INPUT_STREAM* = *axi_vdma_1_M_AXIS_MM2S*
15. Make following Bus-Interfaces for *processing_system_7*,

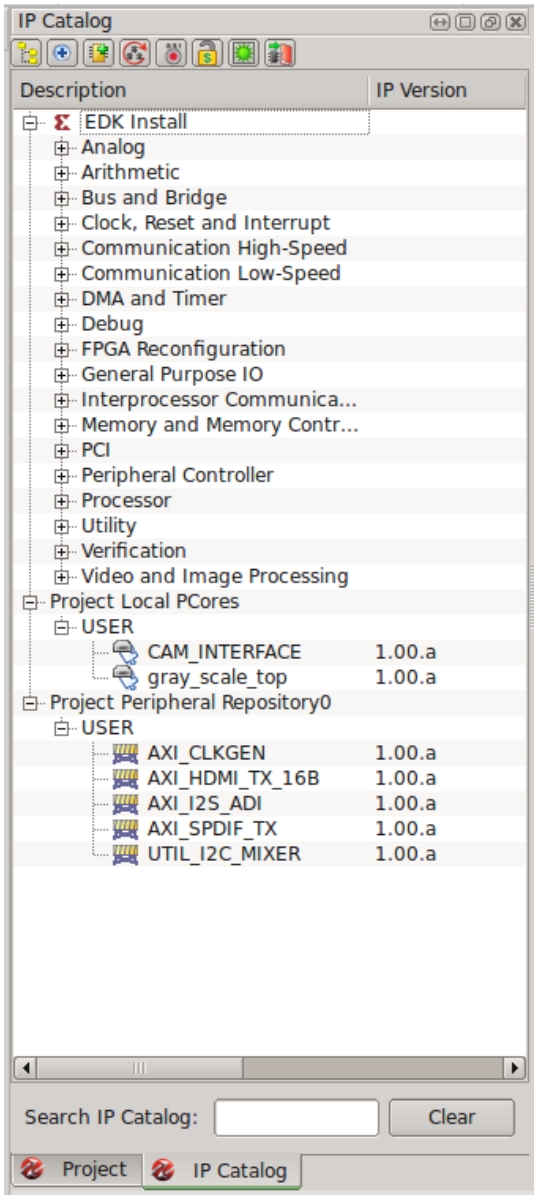

Figure 1.4.: User IPs in the IP Catalog panel.

- For *S_AXI_HP1*, click on *Bus Name*. In the dialog box, select *axi_interconnect_3* and check both *axi_vdma_1.M_AXI_MM2S and axi_vdma_1.M_AXI_S2MM* in *Select Master(s)*.

Figure 1.6 shows the bus-interconnections for the newly added IPs.

16. Now for the port connections, activate *Ports* tab. Expand *axi_vdma_1* ports and make the port connections as below,
    - m_axis_mm2s_aclk = processing_system7_0::FCLK_CLK3
    - s_axis_s2mm_aclk = processing_system7_0::FCLK_CLK3
    - s_axi_lite_aclk = processing_system7_0::FCLK_CLK0
    - m_axi_mm2s_aclk = processing_system7_0::FCLK_CLK3

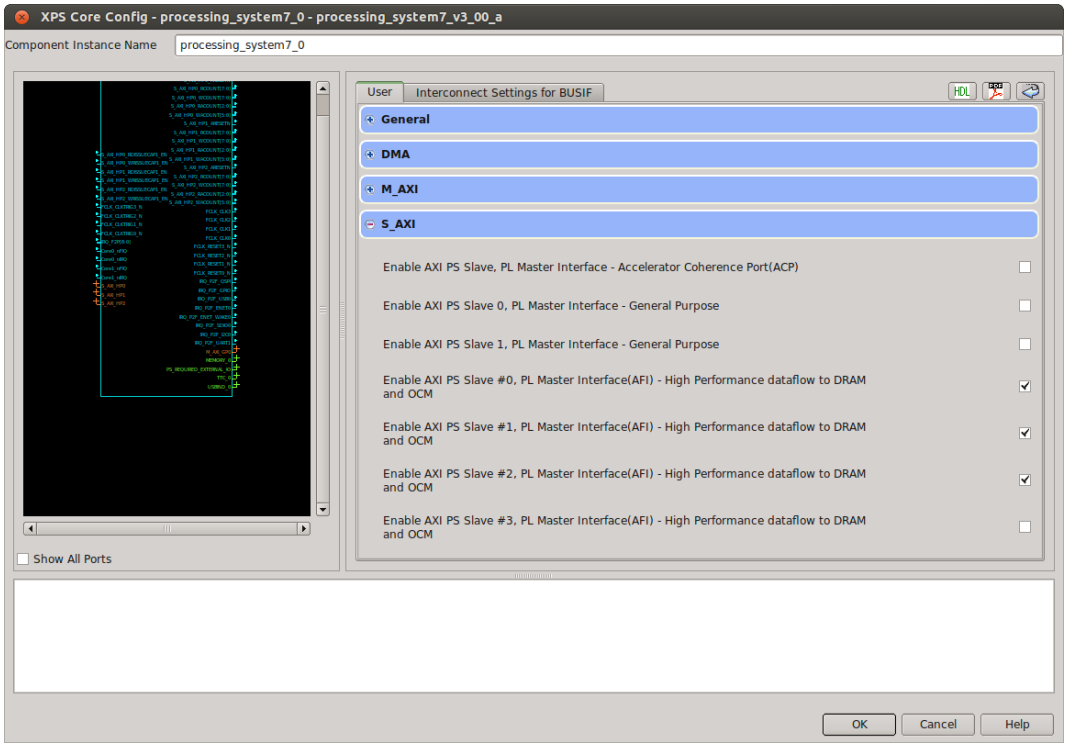

Figure 1.5.: HP Slave port for VDMA master instantiation.

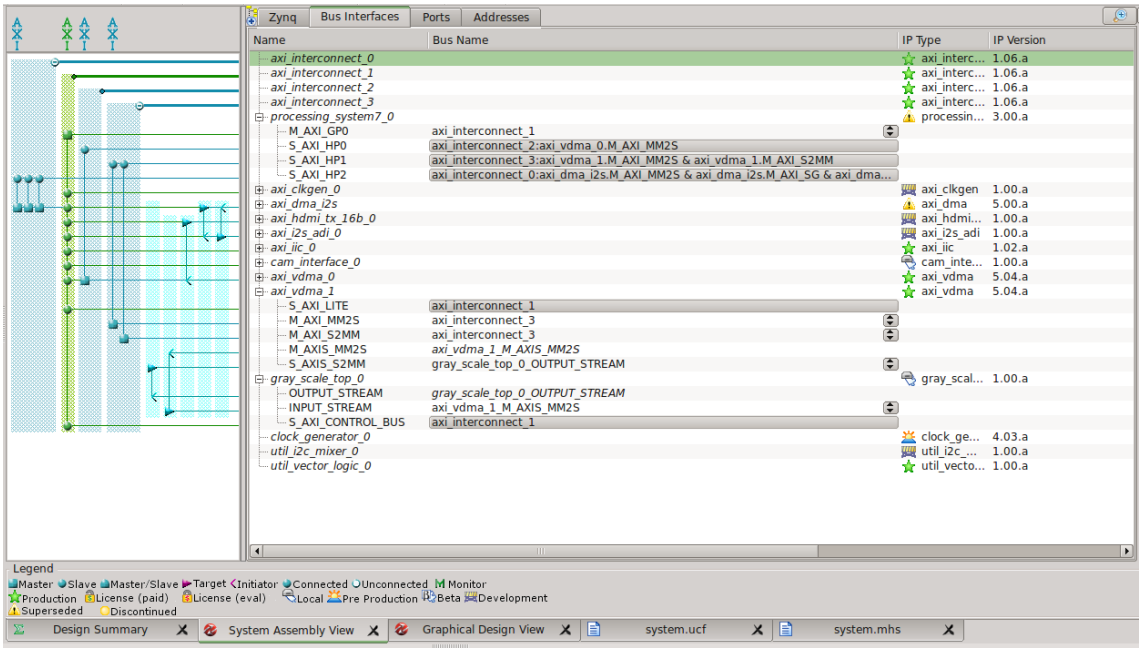

Figure 1.6.: Bus-interconnections for the newly added IPs.

- m_axi_s2mm_aclk = processing_system7_0::FCLK_CLK3
- mm2s_fsync = axi_vdma_0_mm2s_fsync_out

- s2mm_fsync = axi_vdma_0_mm2s_fsync_out

These connections include the clock signals and frame synchronization signals required for synchronizing with the input frame sequence.

17. Expand *gray_scale_top_0* ports, connect the *interrupt* port to *processing_system7_0* as earlier and make the following port connections,
    - *OUTPUT_STREAM >aclk = processing_system7_0_FCLK_CLK3*
    - *INPUT_STREAM >aclk = processing_system7_0_FCLK_CLK3*
    - *S_AXI_CONROL_BUS >aclk = processing_system7_0_FCLK_CLK3*
18. Expand *processing_system7_0* and make this connection,
    - *(BUS_IF) S_AXI_HP1>S_AXI_HP1_ACLK = processing_system7_0_FCLK_CLK3*

    Figure 1.7 shows the port connections for the new IPs.

Zynq | Bus Interfaces | Ports | Addresses

Name	Connected Port	Direction	Range	Class
axi_interconnect_3				
processing_system7_0				
axi_clkgen_0				
axi_dma_i2s				
axi_hdmi_tx_16b_0				
axi_i2s_adi_0				
axi_iic_0				
axi_vdma_0				
cam_interface_0				
clock_generator_0				
util_i2c_mixer_0				
util_vector_logic_0				
axi_vdma_1				
m_axis_mm2s_aclk	processing_system7_0::FCLK_CLK3	I		CLK
s_axis_s2mm_aclk	processing_system7_0::FCLK_CLK3	I		CLK
mm2s_prmry_reset_out_n		O		
s2mm_prmry_reset_out_n		O		
mm2s_fsync	axi_vdma_0::mm2s_fsync_out	I		
mm2s_frame_ptr_in		I	[5:0]	
mm2s_frame_ptr_out		O	[5:0]	
mm2s_fsync_out		O		
mm2s_prmtr_update		O		
mm2s_buffer_empty		O		
mm2s_buffer_almost_empty		O		
s2mm_fsync	axi_vdma_0::mm2s_fsync_out	I		
s2mm_frame_ptr_in		I	[5:0]	
s2mm_frame_ptr_out		O	[5:0]	
s2mm_fsync_out		O		
s2mm_buffer_full		O		
s2mm_buffer_almost_full		O		
s2mm_prmtr_update		O		
mm2s_introut		O		INTERRUPT
s2mm_introut		O		INTERRUPT
axi_vdma_tstvec		O	[63:0]	
(BUS_IF) S_AXI_LITE	Connected to BUS axi_interconnect_1			
s_axi_lite_aclk	processing_system7_0::FCLK_CLK0	I		CLK
(BUS_IF) M_AXI_MM2S	Connected to BUS axi_interconnect_3			
m_axi_mm2s_aclk	processing_system7_0::FCLK_CLK3	I		CLK
(BUS_IF) M_AXI_S2MM	Connected to BUS axi_interconnect_3			
m_axi_s2mm_aclk	processing_system7_0::FCLK_CLK3	I		CLK
gray_scale_top_0				
interrupt	processing_system7_0::IRQ_F2P	O		INTERRUPT
(BUS_IF) S_AXI_CONTROL_BUS	Connected to BUS axi_interconnect_1			
aclk	processing_system7_0::FCLK_CLK3	I		CLK
(BUS_IF) INPUT_STREAM	Connected to BUS axi_vdma_1_M_AXIS_MM2S			
aclk	processing_system7_0::FCLK_CLK3	I		CLK
(BUS_IF) OUTPUT_STREAM	Connected to BUS gray_scale_top_0_OUTPUT_STREAM			
aclk	processing_system7_0::FCLK_CLK3	I		CLK

Figure 1.7.: The port-connections for the newly added IPs.

19. Verify that the addresses for each peripheral has been auto-generated (*Addresses* tab). If there are some unmapped addresses, click the *Generate Addresses* button (on top right), to generate the addressses.

20. Add the following parameter values in the file *system.mhs* below the instance declaration of *axi_vdma_1*. This code sets the hardware configuration for the VDMA IP module (Data widths, line buffer, fsync, etc.).

```
BEGIN axi_vdma
PARAMETER INSTANCE = axi_vdma_1
PARAMETER HW_VER = 5.04.a

# Start copying from here
PARAMETER C_INCLUDE_SG = 0
PARAMETER C_INCLUDE_MM2S = 1
PARAMETER C_M_AXI_S2MM_DATA_WIDTH = 64
PARAMETER C_M_AXI_MM2S_DATA_WIDTH = 64
PARAMETER C_S_AXIS_S2MM_TDATA_WIDTH = 32
PARAMETER C_M_AXIS_MM2S_TDATA_WIDTH = 32
PARAMETER C_S2MM_LINEBUFFER_THRESH = 8
PARAMETER C_MM2S_LINEBUFFER_THRESH = 8
PARAMETER C_INTERCONNECT_M_AXI_S2MM_AW_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_S2MM_AR_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_S2MM_W_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_S2MM_R_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_S2MM_B_REGISTER = 8
PARAMETER C_USE_FSYNC = 1
PARAMETER C_FLUSH_ON_FSYNC = 1
PARAMETER C_INCLUDE_S2MM_SF = 0
PARAMETER C_INCLUDE_MM2S_SF = 0
PARAMETER C_PRMRY_IS_ACLK_ASYNC = 1
PARAMETER C_INCLUDE_S2MM_DRE = 1
PARAMETER C_INCLUDE_MM2S_DRE = 1
PARAMETER C_INTERCONNECT_S_AXI_LITE_AW_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_LITE_AR_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_LITE_W_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_LITE_R_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_LITE_B_REGISTER = 8
PARAMETER C_S2MM_LINEBUFFER_DEPTH = 4096
PARAMETER C_S2MM_MAX_BURST_LENGTH = 16
PARAMETER C_MM2S_LINEBUFFER_DEPTH = 4096
PARAMETER C_MM2S_MAX_BURST_LENGTH = 16
PARAMETER C_INTERCONNECT_M_AXI_S2MM_WRITE_FIFO_DEPTH = 512
PARAMETER C_INTERCONNECT_M_AXI_S2MM_WRITE_ISSUING = 8
PARAMETER C_INTERCONNECT_M_AXI_S2MM_WRITE_FIFO_DELAY = 1
PARAMETER C_INTERCONNECT_M_AXI_MM2S_AW_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_MM2S_AR_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_MM2S_W_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_MM2S_R_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_MM2S_B_REGISTER = 8
PARAMETER C_INTERCONNECT_M_AXI_MM2S_READ_FIFO_DEPTH = 512
PARAMETER C_INTERCONNECT_M_AXI_MM2S_READ_FIFO_DELAY = 1
```

```
PARAMETER C_INTERCONNECT_M_AXI_MM2S_READ_ISSUING = 8
# End copying here

PARAMETER C_BASEADDR = 0x43000000
PARAMETER C_HIGHADDR = 0x4300ffff
```

21. Click *Run DRCs*. *Run DRCs* will finish with *"Done!"* message. **NOTE:** There will be some bogus *error messages (EDK:4207, EDK:4208, and EDK:4209)*, please ignore those.
22. Generate the bit stream. It will take about 30mins, so you can go and take a short coffee break!
23. Bit generation will complete without errors and prints *"bitstream generation is complete"* message on the console.
24. Last step here is to export the design, click *Export Design*. Select *Include Bitstream and BMM* and click *Export & Launch SDK*.
25. In *Workspace Launcher* dialog, Browse to LabD's *SDK_Workspace*. Click *OK*.

Questions:

1. Describe the AXI4 Streaming interface? (ref. User manual, ug902)
2. Which peripherals are connected to *axi_interconnect_3* and what determines its operating frequency?
3. What are the *fsync* signals in the *port connections* for *axi_vdma_1*? (ref. to LogiCORE IP product guide, pg020)
4. Explain about the *bus interfaces* for *axi_vdma_1* and *gray_scale_top_0* and how is the video frame data handled and processed?
5. What is an openCV library? (ref. to www.opencv.org)

1.4. Software Implementation for Hardware Accelerated System

After the last step of the previous section, the SDK window for software implementation will open. The *Project Explorer* panel will already be populated with the project *hw_platform and BSP (standalone_bsp_0)*.

1. Copy the software driver files from the *GrayScaleIP* HLS project. $cp <LabD >/GrayScaleIP/grayscale_prj/solution1/impl/pcores/gray_scle_top_v1_00_a/include/* <LabD >/videoprocessing_prj/SDK/SDK_Workspace/video_filer_sw/inc/.
2. Import hw_config.c and hw_config.h from the <project docs>/LabD into *src* of *video_filter_sw* project.
3. Open the main.c file under *video_filter_sw*. Include these files in the header,

```
#include "xgray_scale.h"
#include "hw_config.h"

XGray_scale xGrayScaleFilter;
```

4. Add the following code to *main()* function, immediately after the hardware *platform initialization* step (Figure 1.8),

```
xGrayScaleFilter.Control_bus_BaseAddress =
        XPAR_GRAY_SCALE_TOP_0_S_AXI_CONTROL_BUS_BASEADDR;
xGrayScaleFilter.IsReady = XIL_COMPONENT_IS_READY;
config_grayScaleFilter();

resetVDMA();
config_filterVDMA(XPAR_AXI_VDMA_1_BASEADDR, DMA_MEM_TO_DEV ,
VIDEO_BASEADDR);
config_filterVDMA(XPAR_AXI_VDMA_1_BASEADDR, DMA_DEV_TO_MEM,
HWPROC_VIDEO_BASEADDR);
```

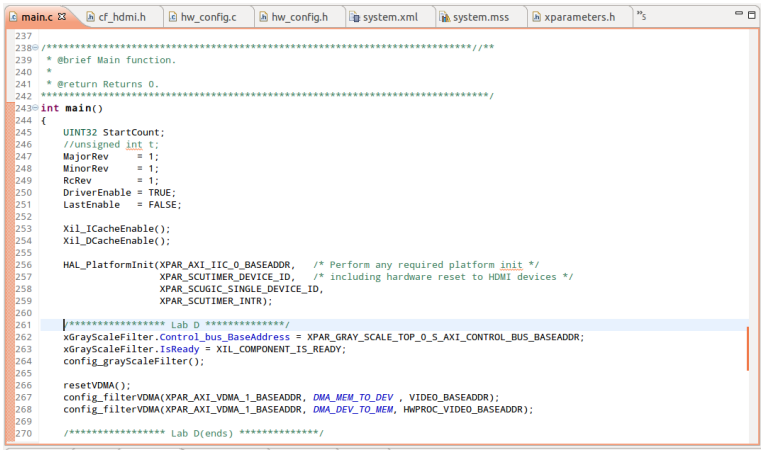

Figure 1.8.: *main()* function, immidiately after hardware *platform initialization* step.

5. Clean all the errors (if any) related to new peripherals in the SDK.
6. Change the *main.c*, such that it displays the hardware processed video stream on pressing key 'h' (similar to that in Lab C).
7. After making these changes, *program FPGA* with the current hardware and the software.

1.5. Application Profiling

Now we will profile the SW application to find the computational load on the ARM processor. The ARM Cortex-A9 processor provides a set of performance monitor registers, that can be used for benchmarking and cycle-accurate profiling of the software. The *Cycle Count Register* counts processor clock cycles. To profile a part of code in a software, we can read the *Cycle Count Register* value before and after that particular code.

1. In SDK, open the *main.c* file and include *"profile_cnt.h"* file (#include "profile_cnt.h"). The *profile_cnt.h* is already present in your project hierarchy in the *src* folder. The file *profile_cnt.h* contains following profiling functions,
    - *EnablePerfCounters*: enables the user access to performance monitor registers

- *get_cyclecount*: returns the current value of Cycle Count Register which indicates the processor clock cycles

2. *init_perfcounters*: resets the Cycle Count Register. Always set *enable_divider* to 0 when calling this function.
3. In *main.c*, enable the performance monitor registers for profiling (hint: EnablePerfCounters function).
4. Use the above mentioned functions in the main function to determine the number of processor clock cycles consumed by the software function for Gray scale conversion. Print the results on the minicom terminal using *xil_printf*.

Questions:

6. How many CPU clock cycles does the gray scale conversion function takes for execution?
7. How much time (in ms) is it required to process single frame by the CPU?
8. Why does the number of CPU cycles vary for each frame processing?

1.6. Conclusions

Now we have implemented a complete hardware accelerated video processing system on ZedBoard. This is the end of Lab D and the laboratory part of the SDS course. By the end of this laboratory you should be in position to understand these concepts,

- Basics about ZedBoard for system prototyping.
- High level synthesis using industry standard tools from Xilinx.
- Implementation of video processing systems on FPGA.
- HW-SW co-design for video processing applications.
- Acceleration of the application using hardware accelerators.
- Using auto-generated software driver files from Xilinx HLS to build application software code.
- Performance analysis of the System.

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.