Lab Manual

Synthesis of Digital Systems SS19

Prof. Dr.-Ing. Ulf Schlichtmann

July 1, 2019

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

Contents

1 Video Processing on ZedBoard using Vivado

In this lab-module, we will develop a simple video-processing system on the ZedBoard. The HW design will employ *Camera* and *VGA* interface IPs in PL in order to connect to external OV7670-camera and any VGA-compatible display respectively. Once the hardware setup is available, we will write application SW that performs simple grayscale-filtering on the captured camera data. Finally, we will profile the developed SW for quantifying execution performance of the whole system.

1.1 System Description

Fig. 1.1 shows the HW-level overview of the video-processing system (baring the power and USB connections). As shown, we will make use of the external *PMod* and *VGA* ports of ZedBoard in order to interface with the OV7670 camera module (via jumpers) and the VGA display (via VGA cable) respectively. This would allow us to get the image-data from the camera into the Zynq chip (PL memory). From there we will stream the data onto the DDR memory. The PS will then process this data in DDR memory. The processed data would then finally be streamed onto the VGA interface to be shown on the VGA-display. Fig. 1.2 shows the block-design of Zynq-chip within this system.

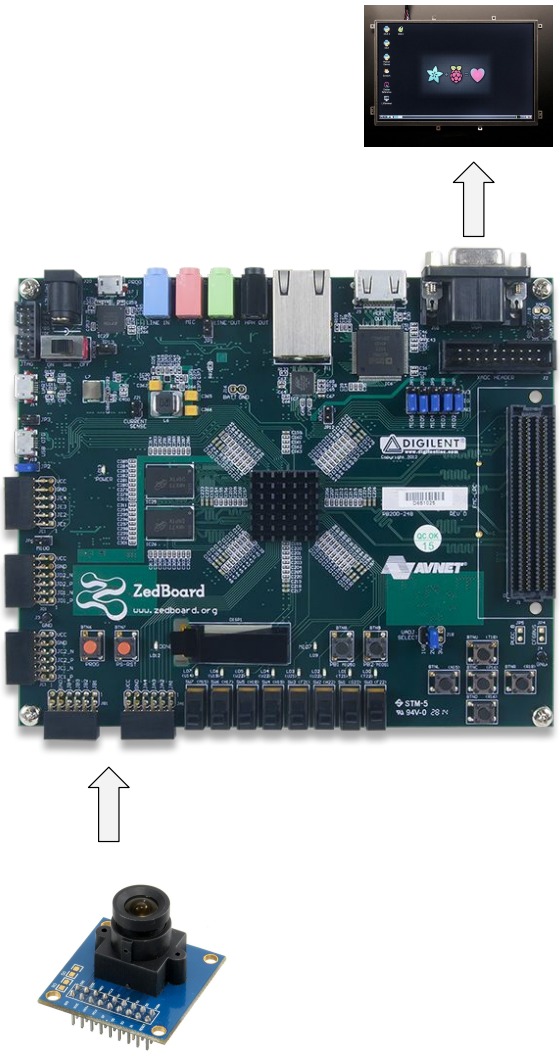

Fig. 1.1: HW Setup for Video-Processing

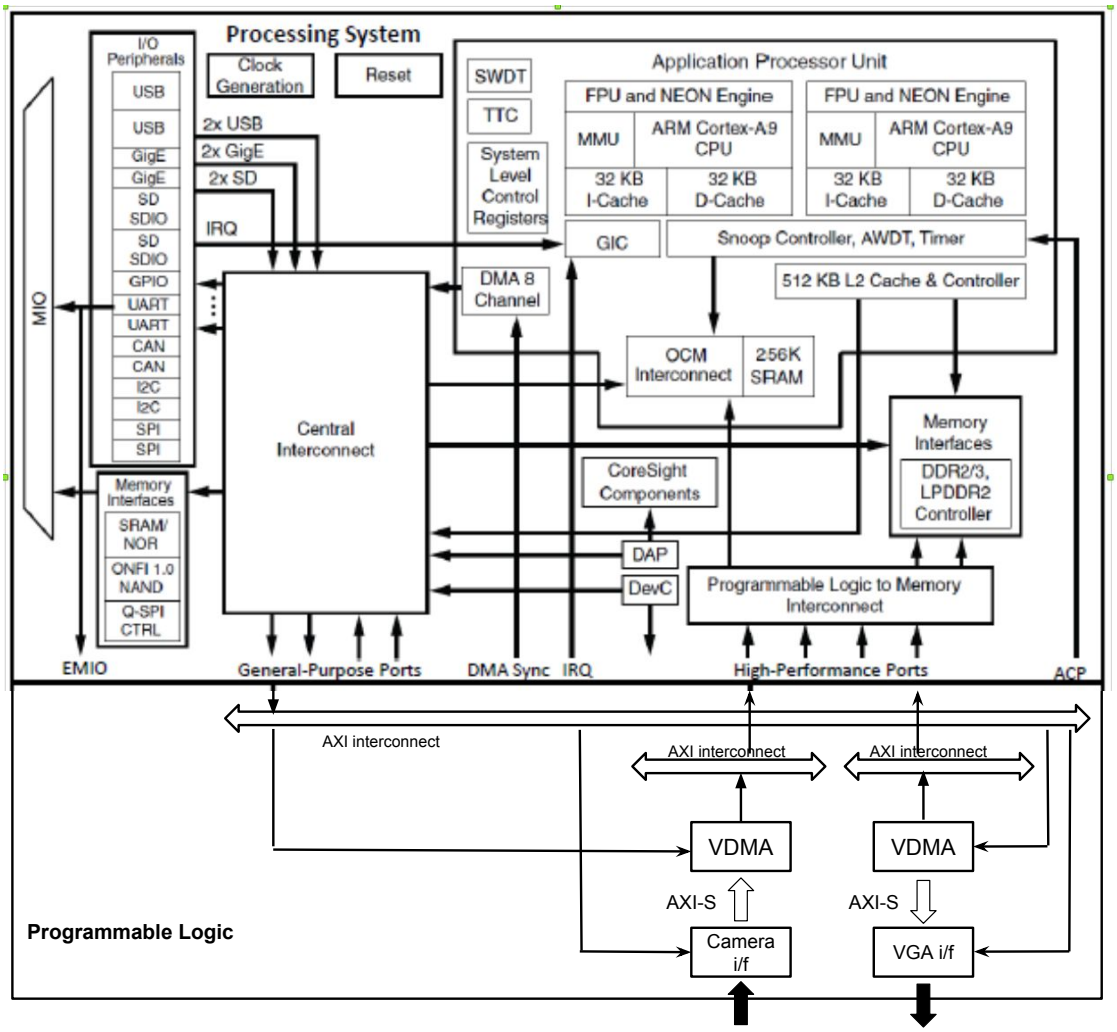

Fig. 1.2: Zynq Design for Video-Processing

1.1.1 Camera Interface

We use a custom-IP in PL for interfacing with the external camera-signals. The IP functionality is described at RTL-level using VHDL. The RTL design is then packaged as an IP and added to Vivado's IP-Catalog so that it can be used in IP-Integrator's block-designs. The block-diagram for the IP is shown in Fig. 1.3.

The camera-interface IP is composed of two main functionaly blocks. The *Controller* is responsible for programming the OV7670-Camera's registers via I2C interface for a specific image format and camera-mode. These registers' config data are currently hard-coded in the Controller block. Once the camera is configured properly, it starts transmitting image frames to the IP which are captured by the *Capture* module within the

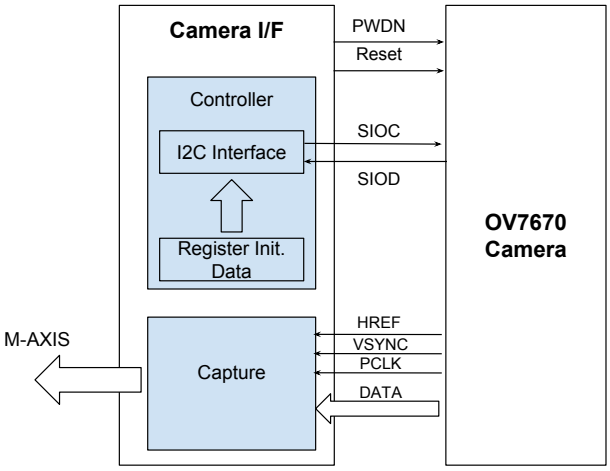

Fig. 1.3: Block-diagram of Camera-I/F IP

IP. The capture interface also includes synchronization signals (PCLK, HREF, VSYNC) along with the 8-bit data interface. For a detailed description of these signals, please refer to OV7670 data-sheet [2]. The capture data is then ready to be streamed via an *AXI-S* interface. The first chapter of *AXI Reference Guide* [3] covers various AXI-based interfaces available for Zynq based designs for curious readers.

1.1.2 Display Interface

Like the camera-interface IP, we design a custom module (in Verilog) for interfacing with the VGA-compatible Display via the VGA port on the ZedBoard. The corresponding block diagram is shown in 1.4. Here again, in addition to the data (red,green,blue) signals, we have additional signals for synchronization purposes. The VGA-interface IP gets the data via the AXI-S interface and transforms into VGA frames that are then transmitted via VGA signals.

1.1.3 Video Dataflow

In order to access the video frames within PS, we need to store the frame-data to the DDR memory (provided as a separate chip on ZedBoard that is accessible to PS). We make use of *Video Direct Memory Access (VDMA)* [5] IPs provided from Xilinx (within Vivado's IP-Catalog) to seamlessly transfer the video frames. The VDMAs allow offloading the CPU for data-transfer tasks and hence the CPU can focus on other important tasks. As shown earlier in 1.2, we connect the VDMAs via dedicated *AXI interconnects* to the *64-bit High Performance Ports (HPP)* of PS. VDMAs access the DDR memory using the PS's *Memory Controller* via the HPP interface. On the other end, each VDMA is interfaced with the Camera/VGA-interface IPs via AXI-S interfaces. The use of

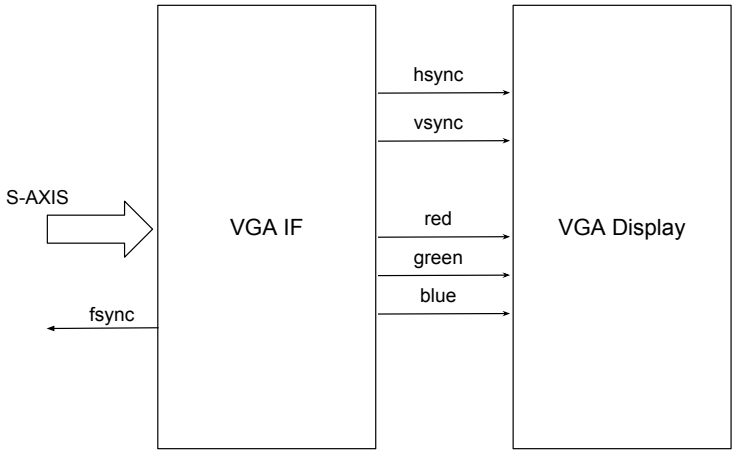

Fig. 1.4: Block-diagram of VGA-I/F IP

streaming protocol is again attributed to ensuring faster video-data throughput within the design.

In order to program these various PL-IPs within the application-SW, we use a separate AXI-interconnect. The PS communicates with this interconnnect via the *32-bit General Purpose Ports (GPP)* as high data-transfer is not desired while programming the IPs' registers. When VDMAs are properly programmed, the video data-flow within the entire design happens as following:

- The camera-interface IP programs the OV7670 camera to start transmitting 640x480 frames with each pixel encoded in *RGB565* format.
- The camera-interface writes the individual lines/rows of the captured frame to its corresponding VDMA's (called the Streaming-to-MemoryMap or S2MM VDMA) FIFO via AXI-S interface.
- The S2MM VDMA transfers the frame lines into DDR memory (the location in DDR memory is programmed while setting up the VDMA in SW).
- The application SW can now process the stored frame data.
- The *MemoryMap-to-Streaming MM2S* VDMA then picks up the processed-frame lines and writes to its FIFO.
- The VGA-interface IP then reads these lines from the FIFO; transforms them into VGA packets that are then transmitted out of ZedBoard.
- The Display monitors then shows these video frames.

1.2 Tasks

1. Using Xilinx IP libraries (axi-interconnects, clk / reset generators, vdma etc.) and custom Camera / VGA interface IPs, we prototyped the hardware design for

the video-processing system using IP-Integrator feature of Vivado. The design is available in *common* directory under *videoProcessing_base*. Copy this sub-directory over into your working directory and then change into it.

2. The directory contents are:
   - **createSystem.tcl**: Tcl script that automates the creation of the project, board selection, block-design creation etc. Vivado's Tcl support makes an already created project easily portable e.g. for version-control, only keeping this Tcl script should suffice.
   - **srcs**: contains various sources that will be used by the created project. The HDL sources (*.vhd, *.v) files and *ip-repo* (packaged IPs to be used in IP-Integrator) will be used by Tcl script while we provide the appication-SW code in *app_sw* sub-directory.
3. Launch Vivado. In the bottom-pane, you will see *Tcl Console*. This allows using Tcl features of Vivado. Within this console, source the *createSystem.tcl* to create the project.

   ```
   > source createSystem.tcl
   ```

4. Once the project is setup, click *Open Block Design* within IP-INTEGRATOR pane. This would show the entire block design of the video-processing system to realize Fig. 1.2 design. Study the block-design, its various components and their interconnections. You are also encouraged to explore the Zynq-PS7 configurations like PL-Fabric clocks configuration, AXI-interfaces etc.
5. As seen in the block, design there are several external interface connections (external to Zynq chip). In order to map these external interfaces to physical connections on the board (e.g. to map the VGA signals to physical VGA port), we use a *Xilinx Design Constraint (XDC)* file. Using the Sources tab, open *Constraints->constrs_1->zedboard_v1.xdc* file which we have prepared for this design. Here you can see how each of camera's and vga's signals are mapped from design signals to physical pins. (ZedBoard datasheet is useful in finding the names of all the external physical pins of Zynq chip)
6. Validate, Synthesize and Implement the design. Afterwards generate the bitstream in order to export it to SDK tool, so as to develop application-SW to run on this HW.
7. Launch SDK and create a *standalone* BSP and *Empty* application project. Instead of writing software from scratch, we will import the source-files. To do this, left-click the *src* direcotry of your created project and choose *Import*. In the Import dialog-box, choose *General -> File System*. Then browse for the *app_sw* directory within the *srcs* sub-directory in *videoProcessing_base*. Hit `OK`. Select both *main.c* and *profile_cnt.h* files and click `Finish`. You should see these two files imported within your project's src folder.
8. Study the code within *main.c* file.

- Before *main()*, you can see various `#defines` for configurations. Then we have a `typedef` for handling individual VDMA driver instances in spirit of Xilinx driver based programming. Then we provide the driver functions to program the VDMA IPs. These are:
  - `vdma_setup()`: this function populates the passed VDMA handle with passed configurations. The last parameter is the location in memory where the data will be written (in case of S2MM) or read from (in case of MM2S).
  - `vdma_set/get()`: read/write VDMA registers.
  - `vdma_start_s2mm/mm2s()`: this function programs the VDMA according to the configurations in the passed VDMA-handle. After calling this function, the VDMAs starts transferring their respective data from Stream-to-MMap (S2MM) or vice-versa (MM2S).

9. In the *main()* function, we first initialize the VDMA drivers' handles and start both of VDMA operations. Then we signal to the OV7670 camera module to start transmitting its frames by programming a `1` to `OV7670_STREAM` register. The camera transmits the pixels in `RGB565` format.
10. Our custom VGA interface is compatible with `RGB888` format, so we need to convert between these two formats in order for the VGA interface to work properly. This is done in `captureRaw()` function. We do this within the infinite `while()` loop, thus every new captured frame is transformed into VGA compatible format and placed separately in another section of DDR Memory. We had used this section while setting up the VGA VDMA (MM2S VDMA).
11. Make sure that the your ZedBoard is properly connected with the camera module and display monitor via the VGA-cable. Now build the project. Program the FGPA with the generated bitstream and download the compiled SW binary into the PS. You should see the video feed from the camera streamed live onto the display panel.
12. Now, you will write the grayscale filtering code within the `convToGray()` function. Make use of the code within `captureRaw` to accomplish this. Afterwards, program the MM2S VDMA to show the video frames from `PROC_VIDEO_BASEADDR`. The successful SW running on PS should result in a grayscale filtered stream on the display.
13. As the last step, you would profile your `convToGray()` implementation. For these have a look at the imported *profile_cnt.h* header file. Here, you are provided with:
    - **EnablePerfCounters()**: this enables accessing the performance-counters within the ARM cores. You have to set this up once in your application-SW in order to profile your code
    - **get_cyclecount()**: returns the current cycles as counted by the performance-counter

- **init_perfcounters()**: resets the performance-counter. Please use the second parameter `enable_divider` always 0 in your function calls

14. Print out the calculated cycles for every grayscale conversion to stdout to see them on the SDK-Terminal while the SW is running.

1.3 Deliverable Submission

Put the Vivado project's **.sdk* folder into the `<your_working_dir>/submissions//` directory. Make sure that the FPGA bitstream file (*.bit) is available in this SDK folder (normally this file is exported at `*.sdk/*_platform_0/*.bit`).

1.4 Questions

Please submit the answers to following questions in a pdf-report *ReportDa.pdf* in `<your_working_dir>/submissions/` within the deadline:

1. Briefly mention what are the different interfaces available on ZedBoard to connect external devices?
2. Please refer to the OV7670-camera timiing diagrams on page-7 of the datasheet [2]. What is the maximum frame-rate (number of frames per second) possible with the input pixel-clock of 16 MHz? The pixel format is RGB565 meaning it requires 2 Bytes to contain each pixel.
3. How is OV7670 camera configured / programmed for desired video operation?
4. Briefly describe AXI-Lite, AXI-S and AXI interfaces within the context of Zynq-based designs.
5. Describe briefly the VDMA IP core functionality and how VDMAs are used to stream data from the camera to the display monitor in our design.
6. How are RGB values extracted from the pixel-data within the application-SW?
7. How much time (in ms) is it required for grayscale-filtering a captured 640×480 frame?
8. Why do the number of CPU cycles vary for each frame-processing?

2 Hardware Accelerated Video-Processing on ZedBoard using Vivado

In this lab-module, we will enhance current video-processing HW design with dedicated HW acceleration for grayscale filtering. We will leverage the Vivado-HLS to generate the RTL IP directly from the C description of grayscale algorithm. The synthesized IP will then be imported in Vivado's IP-Integrator to extend the HW design. On the SW side, Vivado-HLS provides useful drivers with the packaged-IP which makes programming/-controlling the IP from PS easily. Finally, we will profile the grayscale processing in the new setup to quantify the speedup compared with pure-SW based implementation.

2.1 System Description

Hardware/Software (HW/SW) Codesign is an integral part of modern *Electronic System Level* (ESL) design-flows that aims to exploit the trade-offs and synergy of HW and SW in order to tackle the complexity of modern systems. Typically, the application functionality is partitioned into SW components that are running on the processor-cores and HW components that are used to accelerate some parts of the application or to provide interfaces to the environment. Here, in module-D, we will accelerate the grayscale filtering which was found to be compute-intensive as per the profiling analysis.

Fig. 2.1 shows the Zynq design specific to grayscale processing that will be introduced in module-C base design. We will run this sub-system at a faster clock of 150MHz as compared with the rest of the design (<= 100MHz). As shown, this subsystem also comprise a dedicated VDMA-IP and AXI-interconnect to transfer data seamlessly between the system-memory and the GrayscaleIP without affecting the other data-streams or CPU processing. The captured data from system-memory is fetched by Grayscale-IP/Accelerator via VDMA's MM2S channel. On finishing the task, the processed data is stored back in main memory via VDMA's S2MM channel. As with other peripherals, we use a separate interconnect specifically for programming the VDMA and Grayscale registers.

2.2 High-Level Synthesis of Grayscale Filtering Algorithm

Please follow the following steps for this task:

1. Create a new directory *videoProcessing_HA* within `<your_working_dir>`. Create a new Vivado-HLS project *GrayscaleIP* within this directory and add the

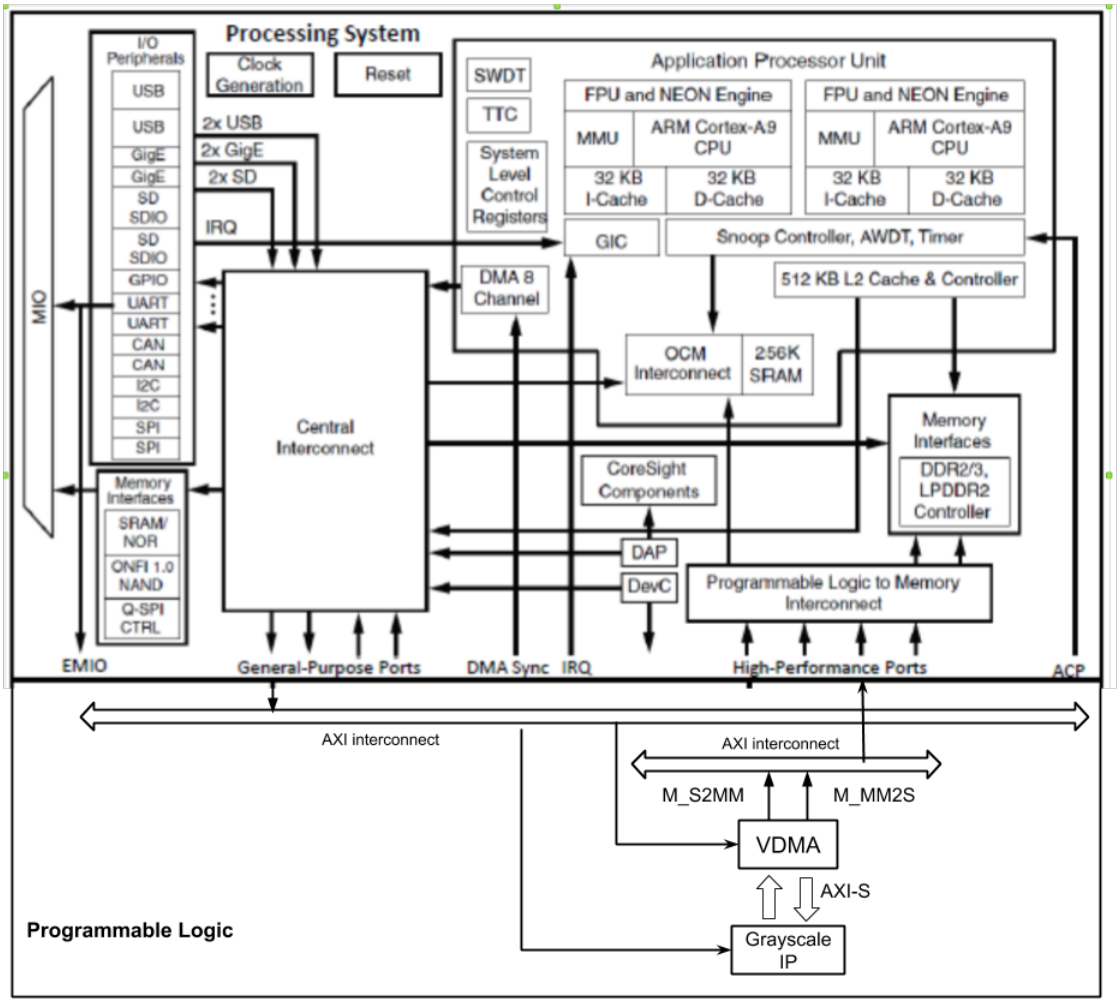

Fig. 2.1: Grayscale-IP Subsystem in Zynq-Design

GrayscaleIP-SW source files from `<your_working_dir>/../common/GrayscaleIP_SW/`. These files contain the SW description for GrayscaleIP that will be synthesized into RTL code by Vivado-HLS. Make sure that the *Top Function* name is *gray_scale*. We won't be simulating this design so avoid adding testbench files. In the *Solution Configuration*, set *Period* to 150MHz (without any spaces) and set the *Part* to be ZedBoard and hit `Finish`.

2. It is worthwhile to have a look at the code within the top *gray_scale()* function. To implement the grayscale function, we make use of the already available implementation within the *Xilinx HLS-Video library* (`hls_video.h` header). This library provides an OpenCV [1] compatible API for rapid development of video-processing algorithms. Using the `#pragma` constructs, we provide directives to perform AXI-S based interface synthesis of *input* and *output* function-parameters while the rest *rows, cols* are going to be accessible via AXI based protocol (*CONTROL_BUS*). Then we allocate memory-space for storing the received raw-frame as well as processed-frame that will be transmitted out to system-memory.
3. Within the computation part, we applied *dataflow* optimization-directive at the highest hierarchy level. The computation starts by transforming the received AXI-based video-frame to OpenCV's *Mat* type. Then we apply the `CvtColor` kernel to this captured frame. You can navigate to see the implementation of tihs kernel and you will find it to be quite similar to your own implemenation as carried out in module-C. Finally, we transform back the processed frame to AXI-based frame to transmit to VDMA.
4. Perform *C Synthesis* of the design. You can visualize the Synthesis report to find more the expected performance and resource-usage when this design will finally be implemented within Zynq-FPGA fabric. As the design meets our timing requirements, the design is now ready to be packaged and exported to Vivado as an IP.
5. From the menu-bar choose *Solution->Export RTL* option to launch the *Export RTL wizard*. Select *IP Catalog* in *Format Selection*. Optionally, you can choose to evaluate the generated packaged-IP (Verilog/VHDL) after synthesis and/or place&route to make sure all design constraints are still met. This is handy to avoid iterations between Vivado and Vivado-HLS flows. On successful evaluation, the HLS of GrayscaleIP is complete. You can browse the packaged IP contents in `solution*/impl/ip/` directory.

2.3 GrayscaleIP Integration in Vivado

Please follow the following steps for this task:

1. Create a new Vivado project *videoProcessing_HA_prj* within `<your_working_dir>/videoProcessing_HA/`.

2. We will start-off from module-C HW-design. In order to reuse the IPs, click *Settings* under *Project Manager* on the left-pane. In the opened window, navigate to *Project Settings->IP->Repository*. Then click `+` and browse to `<your_working_dir>/videoProcessing_base/srcs/ip_repo/` to add four packaged IPs that are needed for module-C block-design in IP-Integrator. Furthermore, add the generated GrayscaleIP (from its packaged location in Vivado-HLS project) using this interface as well.
3. There are also some raw-RTL IPs (non-packaged for IP-Catalog) that are needed in the block-design. To import their code, click `+` in Sources pane. Then choose *Add or create design sources* to add `vga640x480_top.v, vga640x480.v` from `<your_working_dir>/videoProcessing_base/srcs/` directory.
4. With all dependencies covered, we can now compile module-C block-design. Right-click on *IP INTEGRATOR* (in left pane) and choose *Import Block Design*. Then navigate to select `<your_working_dir>/videoProcessing_base/videoProcessing_base/videoProcessing_base.srcs/sources_1/bd/system/system.bd`. Hit `OK`. The block-design from module-C would be loaded which can be visualized by clicking *Open Block Design* under IP INTEGRATOR. Make sure the design validates successfully.
5. As mentioned earlier, we need a faster 150MHz clock-domain and a new dedicated high-performance interface for this grayscale based subsystem. To enable these, customize the *Zynq Processing System* IP. In the *Clock Configuration*, enable *FCLK_CLK3* within *PL Fabric Clocks* and set its frequency to desired 150MHz. Similarly enable *S AXI HP2/3* interface in *PS-PL Configuration->HP Slave AXI Interface*.
6. For profiling purposes later in application-SW, we will be needing interrupt-based communication between GrayscaleIP and the CPU. To enable interrupt support, check *Fabric Interrupts* in *Interrupts*. Then enable *PL-PS Interrupt Ports->IRQ_F2P[15:0]* interrupt-lines. Hit `Finish` to complete the customization.
7. Add VDMA, AXI-interconnect, Gray_scale IPs from IP-Catalog within this block-design. Customize the AXI-interconnect IP to have 2 Slave-Interfaces and 1 Master-interface. We will now setup the connection between these components as shown in Fig.2.1.
    - Connect `gray_scale::OUTPUT_STREAM` interface to `vdma::S_AXIS_S2MM`. Similarly connect `gray_scale::INPUT_STREAM` with `vdma::M_AXIS_MM2S`. Connect `gray_scale::interrupt` to `processing_system::IRQ_F2P[0:0]`.
    - Connect `vdma::M_AXI_MM2S` to `axi_interconnect::S00_AXI` and `vmda::M_AXI_S2MM` to `axi_interconnect::S01_AXI`. Connect `axi_interconnect::M00_AXI` to `processing_system::S_AXI_HP2`.
    - We can make use of *Run Connection Automation* for rest of the signals. On the launched wizard, first choose *axi_interconnect* IP. Here only clock signals are remaining. Check all these signals and select *FCLK_CLK3 (150MHz)* source for each of them.

- Then select *axi_vdma* IP. Here again all the clock signals should be selected to use *FCLK_CLK3 (150MHz)*. Make sure that the remaining interface `S_AXI_LITE` is connected to `processing_system::M_AXI_GP0` via the dedicated axi-interconnect *axi_periph* (for programming the peripheral registers).
- With *grayscale* IP, the `S_AXI_CONTROL_BUS` should also interface similarly with *axi_periph* interconnect.
- Finally on the *processing_system* IP, make sure that the clock for HP interface is set to *FCLK_CLK3* source as well.

8. After connecting fully the grayscaleIP subsystem to module-C design, go to the *Address Editor* tab. Here, you will see that the newly added VDMA still has no reference address assigned to it in order to access the slave HP interface on processing-system (forexample, compare it with earlier VDMAs of module-C). This can be done by simply right-clicking the new VDMA-IP and selecting *Auto Assign Address*. You should see a successful address assignment message. Now, revert back to *Diagram* tab and validate the design.
9. On successful validation, the design is now ready to be synthesized. In order to interface external connections to physical interfaces on ZedBoard, we will reuse the constraint file from module-C (We haven't added any new external interface connections in module-D). To do so, go to the *Sources* pane and hit `+` and afterwards choose *Add or create constraints* and browse to `<your_working_dir>/videoProcessing_base/srcs/zedboard_v1.xdc`. Hit `Finish`.
10. Create HDL Wrapper of the block-design and make sure that the generated *system_wrapper* module is set to Top module (by right-clicking it). Generate Bitstream for this design and afterwards export it to SDK.

2.4 Application-SW

Please follow the steps below to setup the application-SW for accelerated grayscale filtering:

1. Just like module-C, we will start off with the base application that displays the captured raw-video. Launch SDK and create a new application-project and import the base code from `<your_working_dir>/../common/videoProcessing_base/srcs/app_sw/`. Here we will remove the `convToGray(...)` function call wihtin the `while()` loop.
2. It is recommended to cover the Xilinx tutorial [4] on using drivers of generated IP from Vivado-HLS flow. You can find a copy in the `<your_working_dir>/../common/docs/` directory.
3. We use the generated driver code from Vivado-HLS as seen earlier. We notice that the BSP automatically contains the driver code for GrayscaleIP (`*_bsp/ps7_cortexa9_0/libsrc/gray_scale_top_v1_0/src/...`) and hence we don't need to import this explicitly. Use the following code to properly configure GrayscaleIP:

```
#include "xgray_scale.h"
...
// in main()
XGray_scale grayscale_filter;
int Status = XGray_scale_Initialize(&grayscale_filter,
    XPAR_GRAY_SCALE_0_DEVICE_ID);
if (Status != XST_SUCCESS) {
xil_printf("GrayscaleIP is not initialized properly\r\n");
return XST_FAILURE;
}
XGray_scale_SetRows(&grayscale_filter, HEIGHT);
XGray_scale_SetCols(&grayscale_filter, WIDTH);
XGray_scale_EnableAutoRestart(&grayscale_filter);
XGray_scale_Start(&grayscale_filter);
...
```

4. To setup the dataflow between the system-memory and grayscaleIP, the corresponding channels in the VMDA have to be configured and setup. Use the following code-snippets to do so:

```
// the baseaddress of GrayscaleIP's VDMA
#define VDMA_GRAYSCALE ...    // fill in from xparameters.h
...
// in main()
vdma_handle grayscale_vdma_handle_s2mm;
vdma_handle grayscale_vdma_handle_mm2s;
...
// fill in rest of params properly
vdma_setup(&grayscale_vdma_handle_s2mm, VDMA_GRAYSCALE, WIDTH, HEIGHT,
     4, PROC_VIDEO_BASEADDR); // grayscale to memory
vdma_setup(&grayscale_vdma_handle_mm2s, VDMA_GRAYSCALE, WIDTH, HEIGHT,
     4, VIDEO_RGB888_BASEADDR); // memory to grayscale
...
vdma_start_mm2s(&grayscale_vdma_handle_mm2s);
vdma_start_s2mm(&grayscale_vdma_handle_s2mm);
...
// make sure that the VGA's VDMA is handling the processed-video
    instead of the raw video
```

5. Compile the code. Program the FPGA and the ARM processor, and you should see a grayscale video stream being displayed.
6. In order to profile the grayscale filtering, we have to enable interrupt-support. As mentioned in [4], the provided driver with GrayscaleIP already has the functionality to raise its interrupt signal on successful completion of its task. In SW, we can write an *interrupt-handler/interrupt service routine (ISR)* that will be executed by the CPU in case the corresponding interrupt is raised. The ISR can then be used to profile the grayscale-filtering operation within the GrayscaleIP.
7. We have provided two main functions for enabling interrupt-support: `Grayscale_ISR(...)`, `setupInterrupt(...)` which are initially commented. Now uncomment them to use them. Enable interrupt support in both *GIC* as well as *GrayscaleIP* using:

```
#include "xscugic.h"
...
// after configuring GrayscaleIP
XGray_scale_InterruptGlobalEnable(&grayscale_filter);
XGray_scale_InterruptEnable(&grayscale_filter, 1);
...
XScuGic interrupt_controller;
Status = setupInterrupt(&interrupt_controller, &grayscale_filter);
...
```

8. Compile the code. Program the FPGA and the ARM processor, and you should now see *Interrupt Acknowledged* string on SDK-Terminal.
9. Extend the application-code to compute the number of CPU cycles that are passed while GrayscaleIP completes processing for one frame. Compare this with module-C to compute the speedup obtained by use of HW acceleration.

2.5 Deliverable Submission

Put the Vivado project's **.sdk* folder into the `<your_working_dir>/submissions/` directory. Make sure that the FPGA bitstream file (*.bit) is available in this SDK folder (normally this file is exported at `*.sdk/*_platform_0/*.bit`).

2.6 Questions

Please submit the answers to following questions in a pdf-report *ReportDb.pdf* in `<your_working_dir>/submissions/` within the deadline:

1. Briefly describe the HW/SW co-design methodology?
2. Briefly mention the role of hardware acceleration within the context of modern embedded-system design.
3. Describe the video data-flow from the camera to the final processed-video?
4. Describe the OpenCV library and Vivado-HLS' video-library. How is Vivado-HLS video-library using OpenCV concepts?
5. Briefly describe how did you profile the grayscale filtering operation?
6. How much speedup was obtained using HW accelerated grayscale filtering over pure-SW implementation?

Bibliography

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] OmniVision. *OV7670 CameraChip Sensor and OmniPixel Technology*, 2006.

[3] Xilinx. *AXI Reference Guide*, 2011.

[4] Xilinx. *Processor Control of Vivado-HLS Designs*, 2012.

[5] Xilinx. *AXI Video Direct Memory Access*, 2016.