FINAL PROJECT REPORT

Course: Advanced System on Chip (SoC) Design Image Processing

Submitted to Dr. Lauren Christopher

By

Shivani Prasad Dusane

Shivaum Shashikant Heranjal

Department of Electrical and Computer Engineering

Purdue University, Indianapolis, Indiana

Spring Semester 2024

Table of Contents

I Project Plan:

Our project aims to perform edge detection using Sobel filter on images captured by camera in real time and displaying the processed images on a monitor.

Edge detection allows users to observe the features of an image for a significant change in the gray level. Edges are significant local changes of intensity in a digital image. An edge can be defined as a set of connected pixels that forms a boundary between two disjoint regions. Edges indicate the end of one region in the image and the beginning of another. There are three types of edges: Horizontal edges, Vertical edges, Diagonal edges. Edge detection reduces the amount of data in an image and preserves the structural properties of an image. Edge Detection Operators are of two types:

- Gradient – based operator which computes first-order derivations in a digital image like, Sobel operator, Prewitt operator, Robert operator.
- Gaussian – based operator which computes second-order derivations in a digital image like, Canny edge detector, Laplacian of Gaussian

Edge detection is required in numerous fields in the real world. It can be used in digital forensics, where information can be extracted from tampered videos or images; medical imaging wherein anatomical structures or abnormalities can be detected and studied from X-Rays, CT-Scans, etc. Similarly, robotics, satellite image analysis, autonomous vehicles, quality inspection are some of the many fields where edge detection plays a significant role.

When using Sobel Edge Detection, the image is processed in the X and Y directions separately first, and then combined to form a new image which represents the sum of the X and Y edges of the image. When using a Sobel Edge Detector, it is first best to convert the image from an RGB scale to a Grayscale image. Then from there, we will use what is called kernel convolution. A kernel is a 3 x 3 matrix consisting of differently (or symmetrically) weighted indexes. This will represent the filter that we will be implementing for edge detection. When we want to scan across the X direction of an image for example, we will want to use the following X Direction Kernel (Eqn. (1)) to scan for large changes in the gradient. Similarly, when we want to scan across the Y direction of an image, we could also use the following Y Direction Kernel (Eqn. (2)) to scan for large gradient changes as well.

$$Sobel_x = \begin{bmatrix}1\\2\\1\end{bmatrix}\begin{bmatrix}1 & 0 & -1\end{bmatrix} = \begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix} \quad \text{----Eqn. (1)}$$

$$Sobel_y = \begin{bmatrix}1\\0\\-1\end{bmatrix}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix} \quad \text{----Eqn. (2)}$$

$$Gradient = \sqrt{{Sobel_x}^2 + {Sobel_y}^2} \quad \text{----Eqn. (3)}$$

A. Work Statement

Hardware/ Software (HW/SW) Requirements:

- Pynq-Z2 board SD card and SD card for bootup with access to Jupyter Notebook.
- Webcam for receiving input images via USB from the board (dimension=640x480).
- Ethernet cable to access the Jupyter notebook from SD card on board from the PC.
- Monitor to display outputs via HDMI cable.
- Vivado 2018 SW to design and build project and to generate HW handoff and bitstream file to upload to the SD card on the board.
- Key python libraries used are Numpy -for reshape operations and more numeric operations (squaring, etc.), Scipy- for SW Sobel filter implementation, PIL-for image processing, Matplot library- to plot the graphs and images on the notebook.

Hardware Tasks:

- Connecting Webcam, Monitor, SD card and Ethernet cable to respective slots on the board
- Ensure bootup JTAG is setup to use SD card.
- Starting from Lab 5 VHDL (Sobel x filter), add a Sobel y filter block for parallel processing.
- Access input image data using DMAs and send it to the Sobel Filter IPs
- Receive processed outputs from the Sobel Filters (x and y) and store it back in the memory.

Software Tasks:

- Use the Jupyter notebook to write the Python code.
- Capture webcam images based on the frame availability, using the OpenCV library.
- Instantiate custom overlay (‘sob.bit’) created using Xilinx Vivado SW.
- Allocate input and output buffers with size = dimension of input image.
- Reshape 2D array obtained from webcam to 1D and store in input buffer
- Send the input buffer to PL from PS using DMAs containing input image data.
- Receive the output buffer from PL to PS using DMAs containing processed outputs from the Sobel Filters (x and y).
- The output fetched from the DMAs would be used to calculate the gradient using Eqn. (3) in Jupyter notebook.
- Display output images using Matplot library.
- Use the base overlay (‘base.bit’) for HDMI output.

- Display output images on the monitor.

Software - Hardware Integration:

- The input webcam data received in the Pynq-Z2 board is sent by the SW to be processed using the FIR filter IP blocks in the HW.
- These filtered images from the HW are later processed to calculate the gradient using the Jupyter notebook in SW.

Testing:

- To carry out the testing a set of real time images were captured and added into a list to execute the edge detection function.
- Towards the end we also compared the SW filtered output with the HW filtered output on each image captured to check for the efficacy of the HW filtered images.
- Additionally, time calculations are made for the execution of tasks via HW and SW respectively in the Jupyter notebook for comparisons.

Documentation Phases:

- The initial phase consisted of planning of the project, which is included in the proposal.
- As and when we moved ahead with the execution and development of the project, we added comments at each step of VHDL as well as python codes.
- After execution of each task of the project, like building the IP, subsequently the block designing, testing with SD card images, webcam integration, implementing real time image processing, we captured the screenshots at each stage for documentation purpose.

B. Projected Schedule with milestones

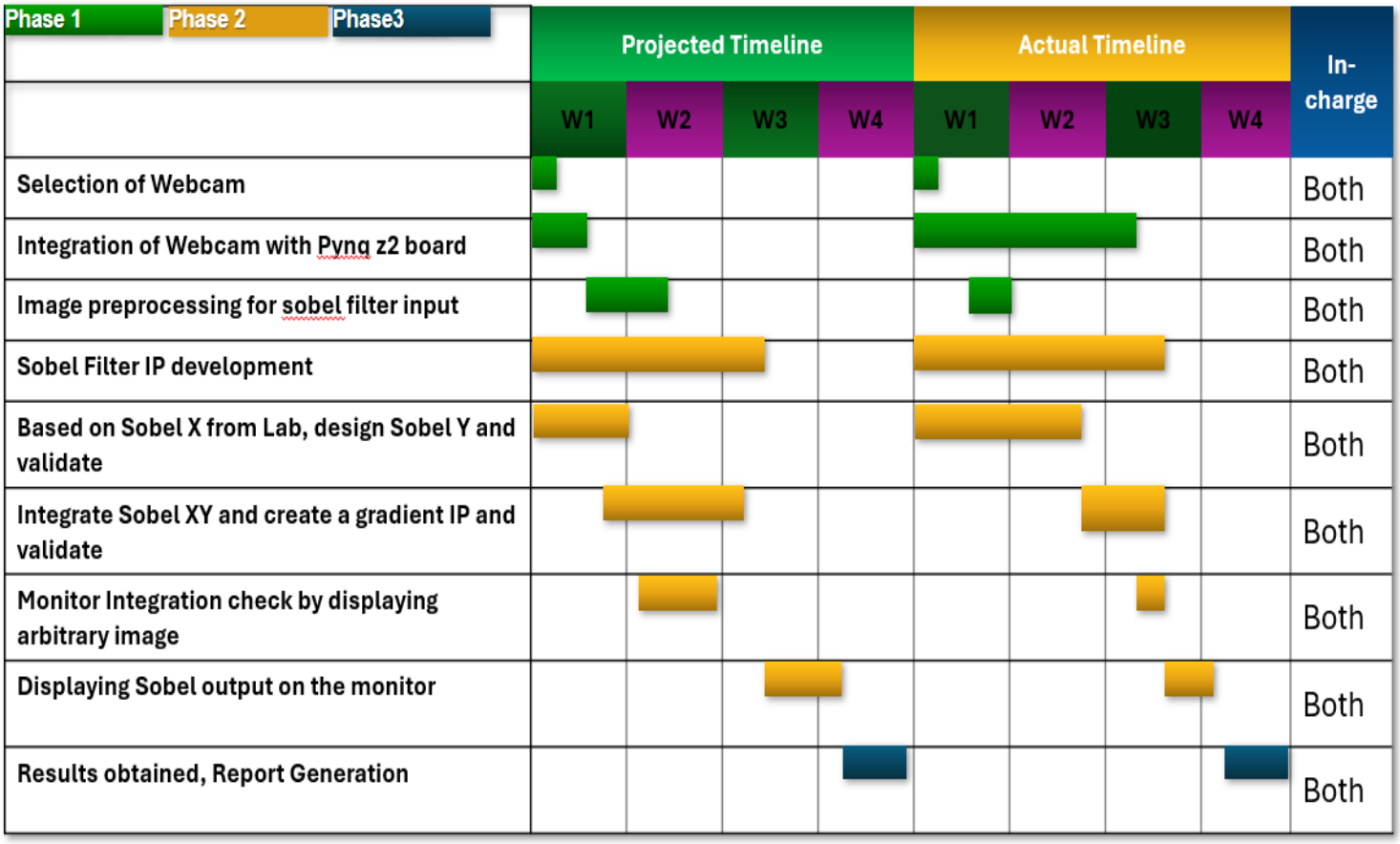

Figure 1: Projected and Actual Timeline and corresponding milestones of the project

Overall, slight extensions/delays were observed in the actual execution of the tasks. A major issue faced here was the webcam integration where OpenCV was integral. The code structure and the frame availability were figured out by the start of 3rd week which was expected to be done in the 1st week itself. Sobel Y was difficult to implement, has been described in further section.

C. Resource requirements

- Level of effort – 4 out of 5
- Number of people – 2
- Time required with Xilinx board – 50 hours (cumulative for both members)
- Additional hardware – Webcam, Monitor.

D. Previous work done in the area

1. Webcam as input image source: https://community.element14.com/products/roadtest/b/blog/posts/pynqz2-dev-kit---cifar-10-convolutional-neural-network

This project had worked on integrating the webcam with the Pynq-Z2 board to get its images classified using the quantized neural network trained on the CIFAR-10 dataset implemented on the board. We will be using the webcam part of the project and in addition to that do the preprocessing (grayscale conversion) of the images to feed to the Sobel filters (x, y).

2. Sobel filter implementation on webcam input and HDMI output streaming video: https://github.com/aclich/PYNQ-Z2_sobel_filter_HDMI

   For the Sobel filter HDMI output, they have used the Vivado HLS to design the Sobel filter and send a stream of frames to display on the monitor in the form of a video. We on the other hand will process images using VHDL in Vivado instead of HLS and start from the Sobel x filter designed in HW5 and work our way up to design Sobel y and then retrieve the outputs of both to calculate gradient for the final output.

II Work Accomplished:

1. Hardware:

In this project we have used the Xilinx Pynq Z2 board. The HW design is inspired by Lab 5 design with additional Sobel y filter. We have implemented a Sobel y filter in parallel to Sobel x filter. These would generate filtered output data from incoming image data from the input buffer. This data is then passed through the AXI buses to DMA and is received at the SW side through the output buffer for further processing (gradient calculation and reshaping) and final display at the monitor. We have used the DRAM memory for the storage of the data.

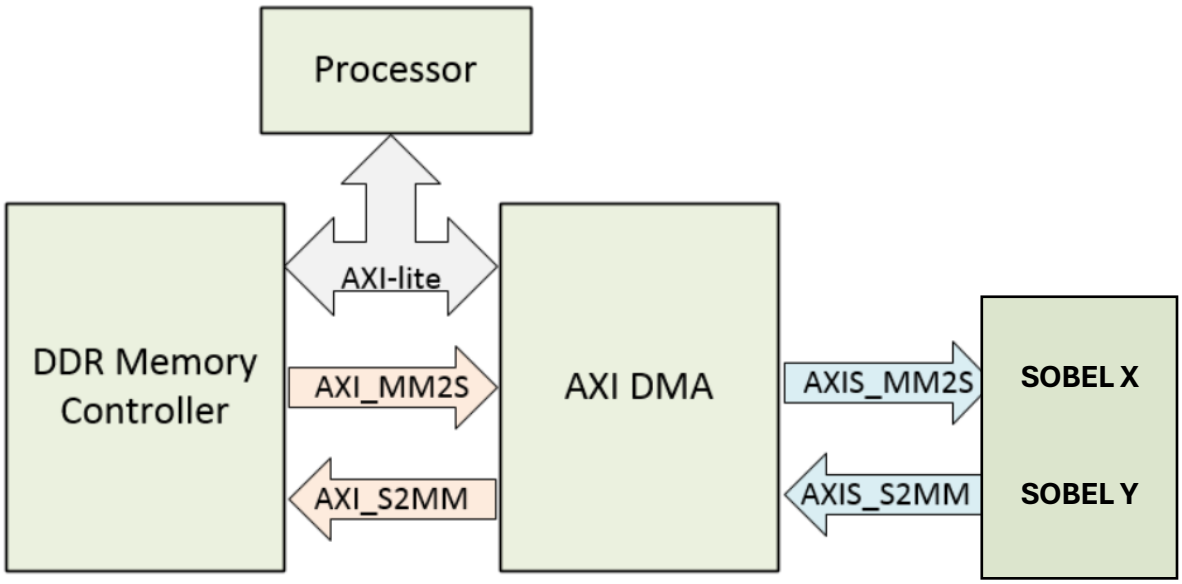

Figure 2: Reference Logic Block Diagram

2. Block Diagram:

The Block diagram in Figure 3 shows all the modules/IPs used in our project.

- Zynq Processing system Block
- AXI Direct Memory Access (DMA)
- AXI Smartconnect
- Sobel x and Sobel y custom IPs
- Processor System Reset Block

The master port from Zynq processing system is (M_AXI_GP0) connected to an AXI smartconnect to get 2 parallel master connections to connect to both the DMAs (S_AXI_LITE, one for Sobel x and other for Sobel y). Data will be read from memory through the M_AXI_MM2S port on the DMAs and sent to the M_AXIS_MM2S (read channel) port which is connected to its corresponding custom Sobel IPs (S00_AXIS). The output of the Sobel IPs (M00_AXIS) is sent to the S_AXIS_S2MM (write channel) port in the DMAs which sends the processed data back to memory through the M_AXI_S2MM port.

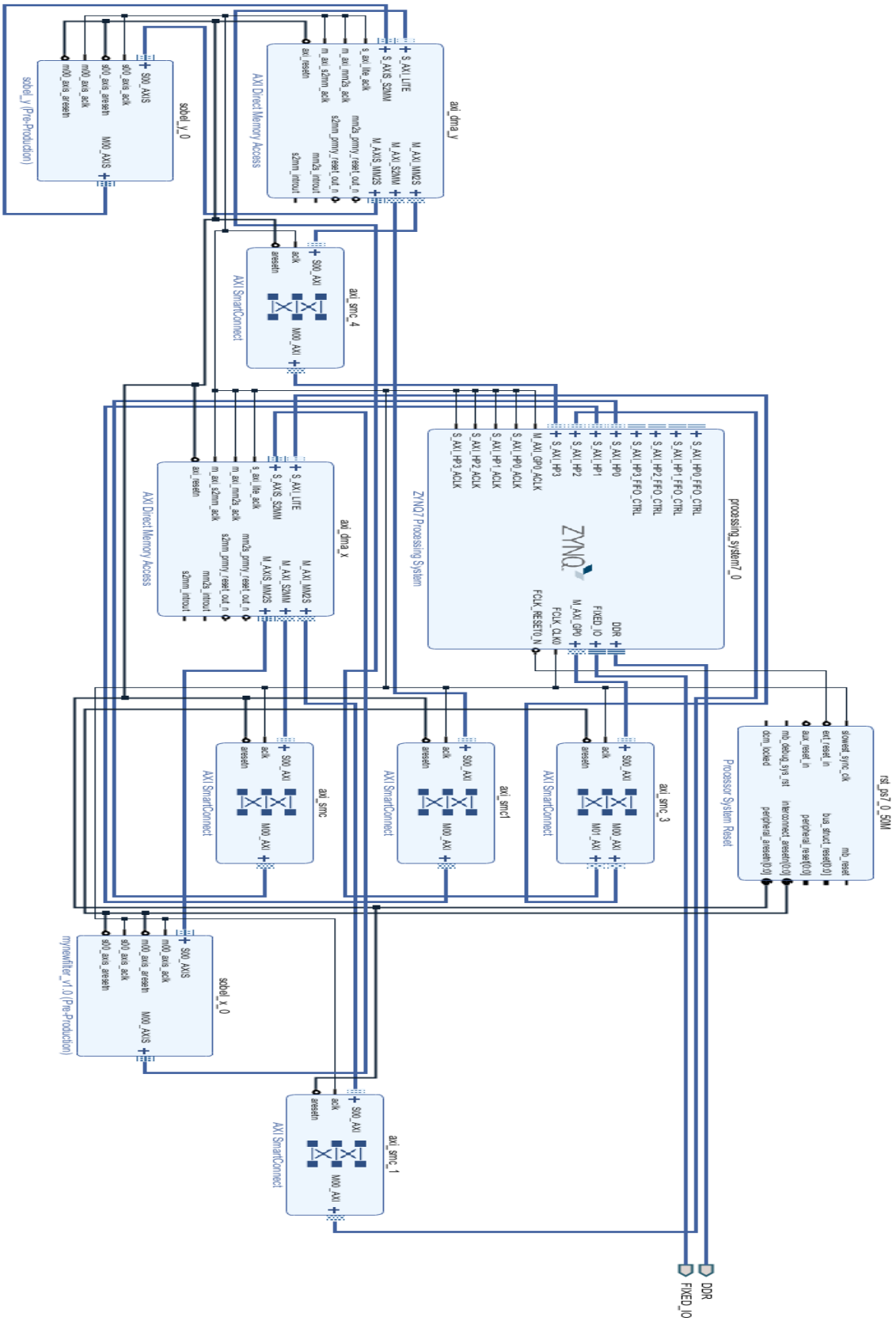

Figure 3: Block Diagram of the project

3. Software:

We implemented the SW section of the project using python in the Jupyter notebook. We captured frames/images from the webcam using the OpenCV library. This captured data is processed using several other libraries in Python. We generated a custom overlay from the Vivado Xilinx, integrated it with the Jupyter notebook program to process the images. The SW part in the Jupyter notebook performs the gradient using the $\sqrt{Sobel_x{}^2 + Sobel_y{}^2}$ formula on the data processed (Sobel x, y filter) by the HW. This gradient data is then passed through the HDMI to be displayed on the monitor. For this display we used another 'base' overlay. In the SW, we have also implemented filters to compare the HW and the SW outputs. We also calculated the total execution time using HW and the SW.

4. Flowchart:

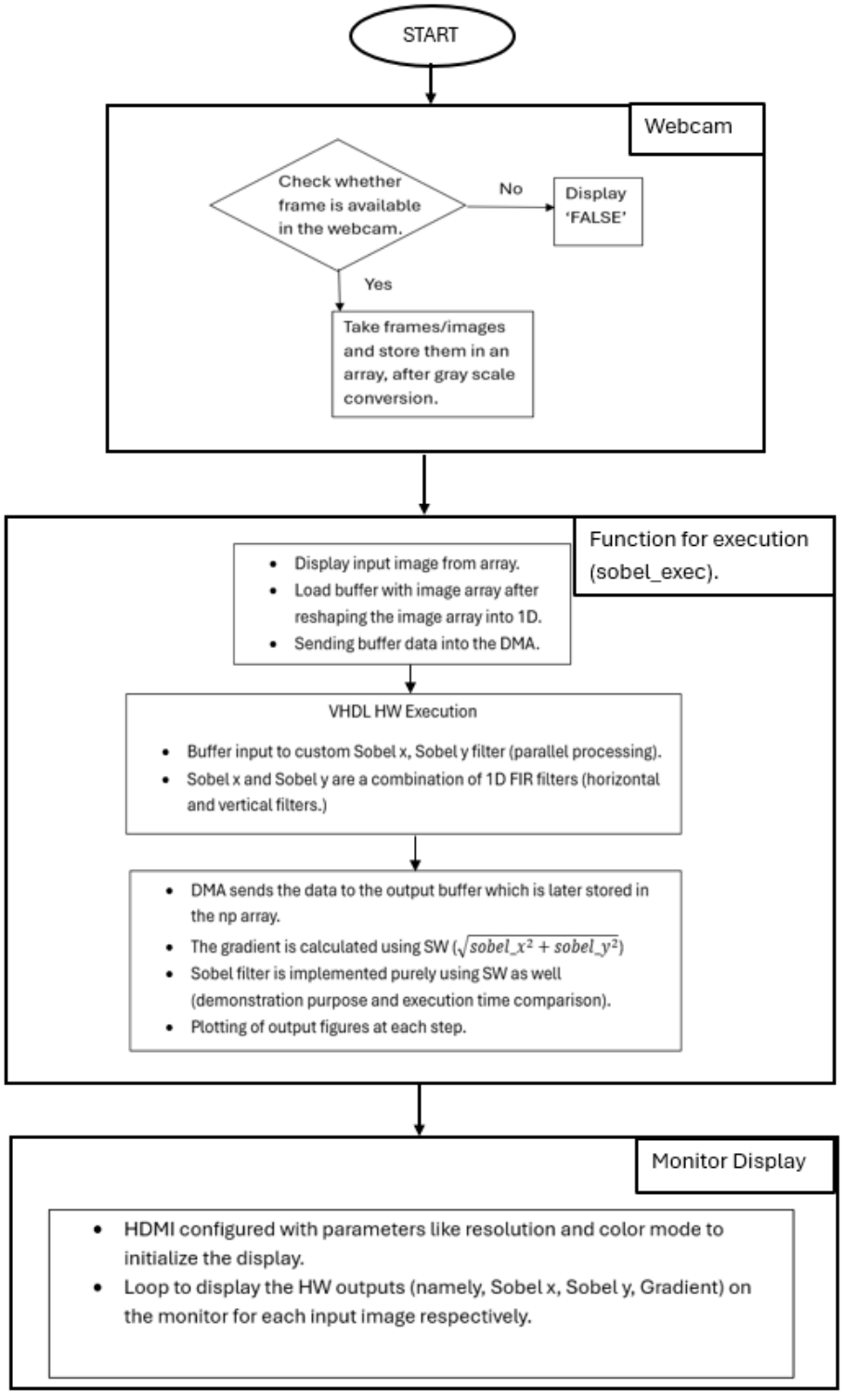

Figure 4: Execution Flowchart

The execution of the whole project- from initialization to displaying output images on the monitor is in accordance with the flowchart shown in Figure 4. Please refer code file for further information.

III Results:

1. Time Comparison:

```
Time stats for image: 1
time to process sobel x,y in hardware: 0.00081
Total Time taken in hardware: 1.266
Time in software for sobel x,y: 0.17809
Total time in software : 1.408
HW speedup factor: 221
```

```
Time stats for image: 2
time to process sobel x,y in hardware: 0.00084
Total Time taken in hardware: 1.266
Time in software for sobel x,y: 0.17863
Total time in software : 1.397
HW speedup factor: 214
```

Figure 4: a) Time stats for image 1; b) Time stats for image 2

As seen in Figure 4, implementing Sobel filter in hardware is over 200 times faster than software. However, due to the gradient still being calculated on the SW end, the total time in HW is 1.266s which acts as a bottleneck and drastically reduces the speed of the HW implementation, but it is still faster than total SW by at least 0.13s.

2. Project Summary:

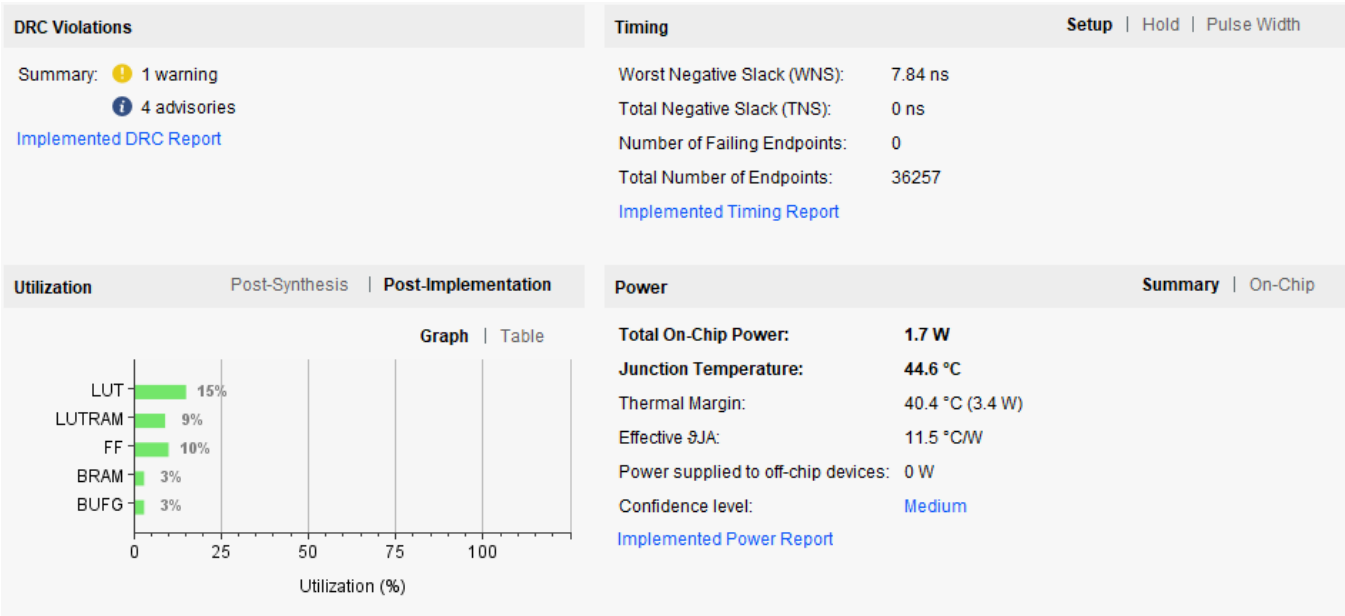

Figure 5: Project summary report after synthesis, implementation and bitstream generation

From the project summary (Figure 5), we observe that less than 20% of the board is utilized, 15% of the lookup tables (LUTs), 9% of the LUT RAMs and 10% of the Flipflops (FFs) are being used. This leads to the conclusion that there is still potential for integrating more custom IPs in the block design, even adding the gradient calculation part in the HW instead of SW. The on-chip power consumption was 1.7 W, while the worst negative slack was 7.84 ns.

3. **Hardware Implemented Design:**

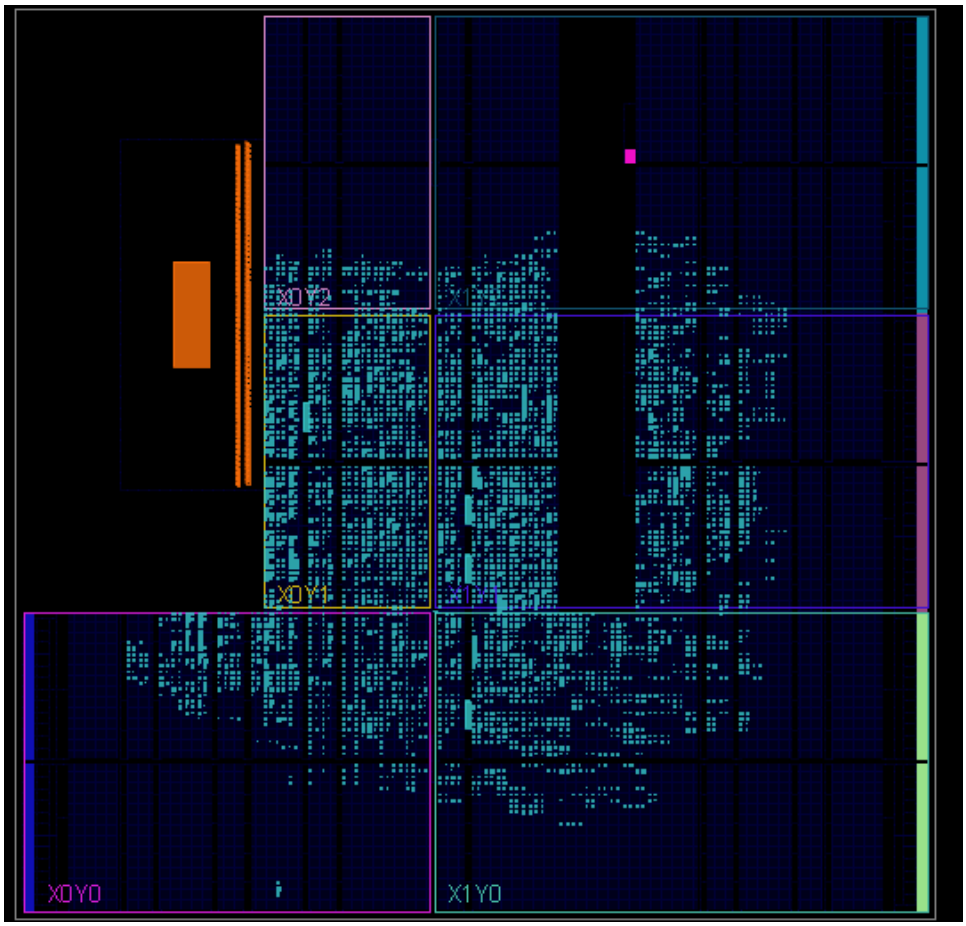

Figure 6: Implemented Design on the board

The above image (Figure 6) from Xilinx Vivado after implementation depicts the HW deployed for this project.

4. Output Results:

- Input Image:

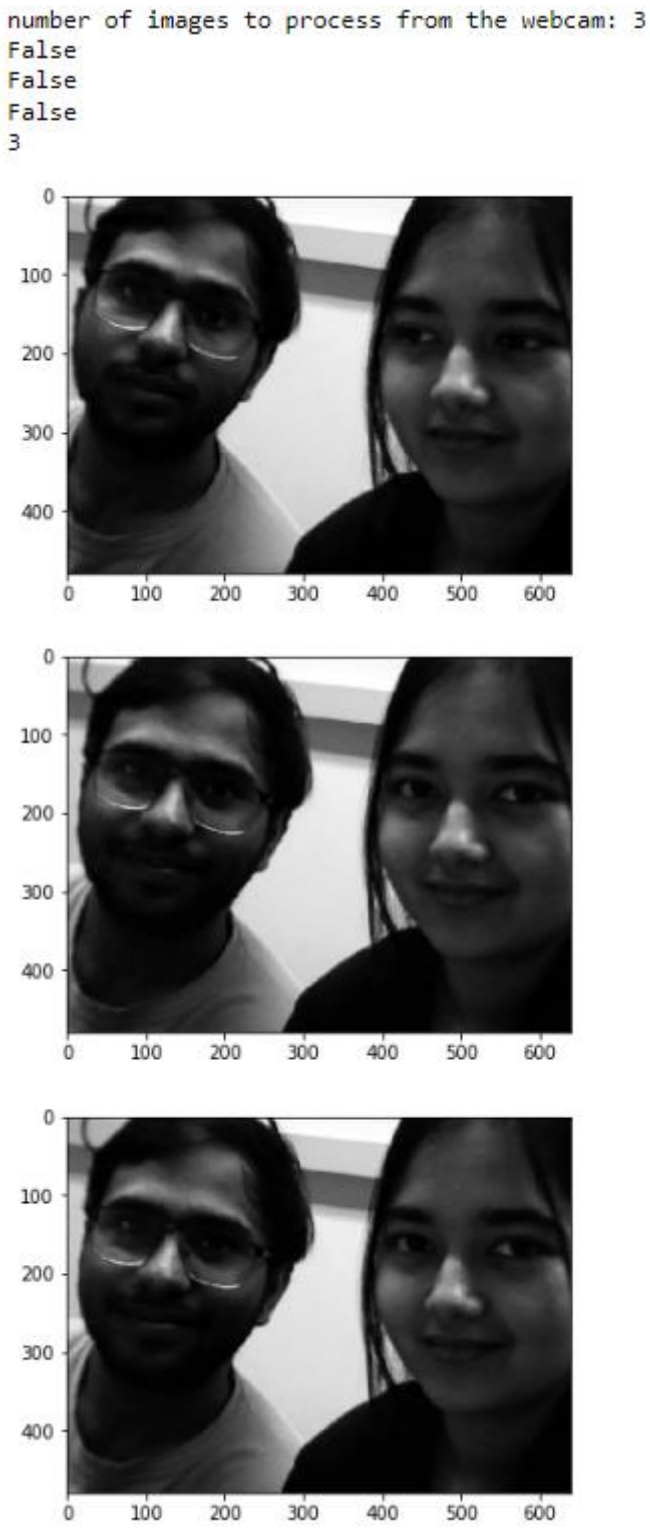

Figure 7: Input images from the webcam (user input n=3)

- HW Outputs

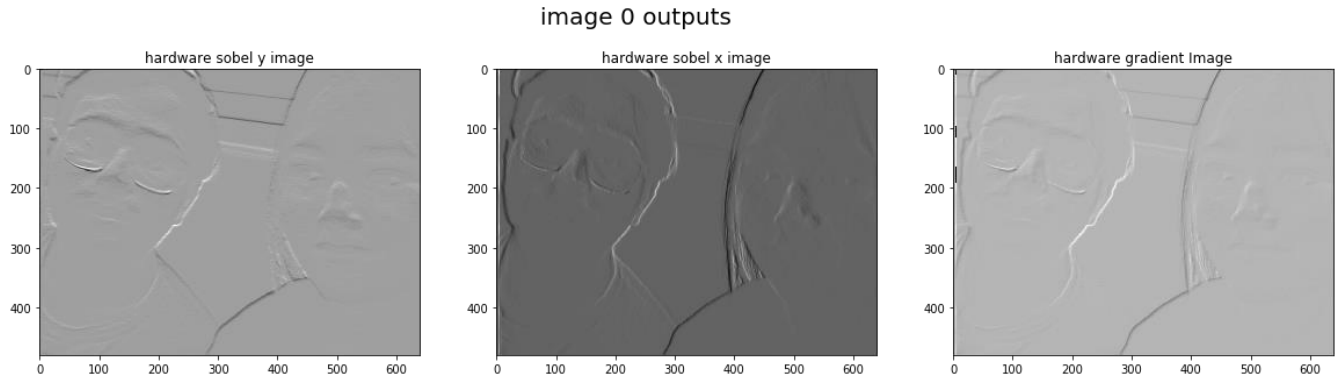

Figure 8: Sobel y, Sobel x and Gradient outputs from the HW

- SW Outputs

Figure 9: Sobel y, Sobel x and Gradient outputs from the SW

The Sobel x and Sobel y HW outputs (Figure 8) are comparable to the SW outputs (Figure 9). Notice how the gradient image in HW is grayscale while the gradient image in SW is black and white. We initially chose 640x480 resolution as the mode for the HDMI outframe, but faced issues where the image would shift to the right and warp on the left. To address this, we increased the display resolution to 1280x720 and zero padded our output images as seen in Figures 10, 11 and 12.

- Monitor Outputs

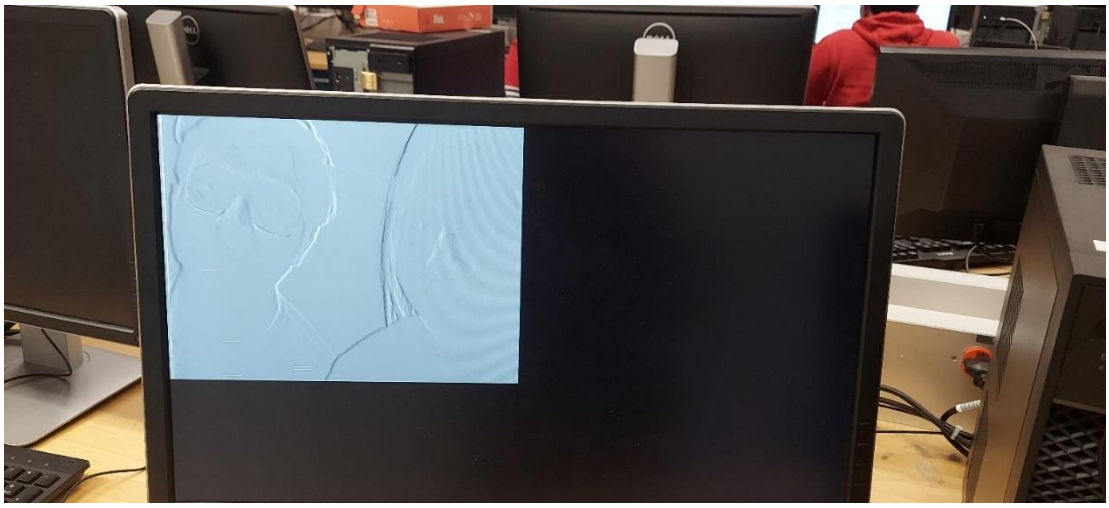

Figure 10: Sobel x HW output displayed on the monitor

Figure 11: Sobel y HW output displayed on the monitor

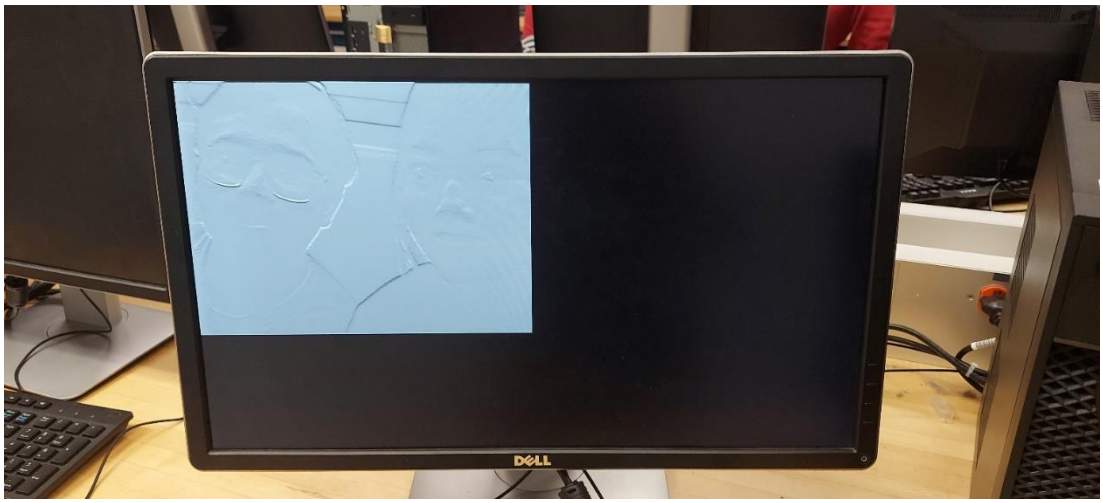

Figure 12: Gradient HW output displayed on the monitor

5. Conclusion:

The proposed tasks of the project were:

1. Integrating Webcam with the board to get input images
2. Implementing Sobel Filter (both x and y) and combining the outputs to obtain gradient for edge detection in HW
3. Implementing the same in SW for output and time comparisons
4. Integrating a monitor to display outputs through HDMI out of the board.

We were able to accomplish all the proposed objectives stated at the start of the project in the proposal. We got familiar with the Vivado software across all the lab projects and homework throughout the semester. We leveraged this familiarity to first simplify the block design from Lab 5 by using one DMA for both read/write operations and later adding a Sobel y filter IP to process input data parallelly and send it back to the memory using the DMA.

6. Discussion:

One major setback was the design of the Sobel y filter- The Sobel x filter IP design had decomposed the 3x3 kernel into 1x3 horizontal and 3x1 vertical kernels and implemented them using separate 1D FIR filters, the input data went to the horizontal filter first, whose outputs went to the vertical filter and then scaled (dividing by 8) and shifted to 0-255 range (addition by 128) for final output. For the Sobel y filter, we took a copy of the Sobel X and changed the coefficients. However, we were getting minor "glitches" where few pixels around the edge were completely black or white, we had to face this issue till the last week where we thought of changing the order of the 1D filters, we first processed the data with the vertical filter and then with the horizontal filter and it resolved our "glitch" issue.

Another issue we were facing was not getting comparable outputs in HW and SW visually. While processing SD card images, we first got Black and White outputs in SW (Figure 13) while grayscale outputs in HW, this made us doubt the HW implementation and we went ahead with connecting a custom gaussian filter IP to the sobel outputs in the block diagram and then subtracting gaussian outputs from the sobel outputs to obtain the final x,y outputs to further process the gradient. This blur and subtraction process acted as a sharpening which got us the Black and White outputs similar to SW. However, on integration of webcam and further discussion with professor, we realized that including a gaussian filter to the design meant, we weren't comparing Sobel SW (which was grayscale now) to Sobel HW anymore (Sobel + Gaussian HW now) and it wasn't a fair comparison. Therefore, we removed the custom Gaussian IP block from our block design and reverted to pure Sobel HW implementation. The only setback we have now is that our HW gradient is grayscale while the SW gradient is Black and White.

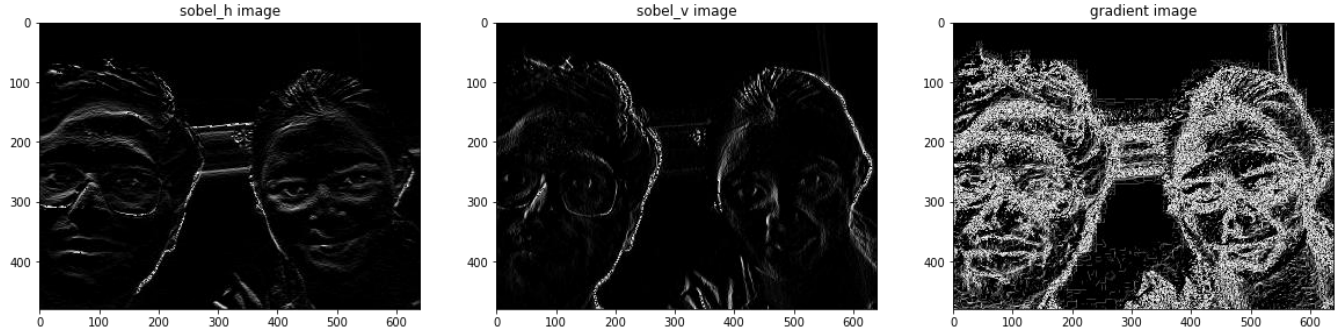

Figure 13: Initial SW outputs which were inconsistent with the HW outputs.

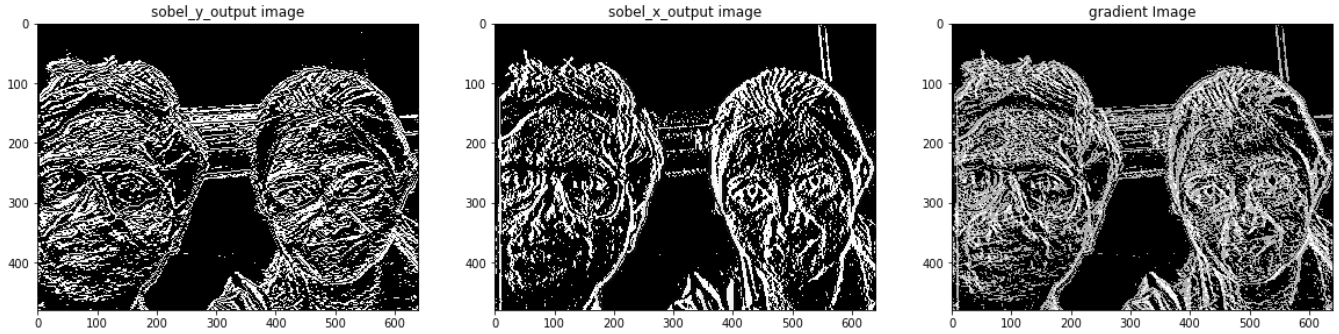

Figure 14: HW outputs after "sharpening" using integrated custom Gaussian Filter in HW

IV Recommendations for future work:

In the current work, 2 overlays are being used, our own custom overlay for image processing, and the base overlay for output display on monitor. Work can be done to integrate the elements required for HDMI output from the base overlay into our custom designed overlay to avoid calling multiple overlay instances. Secondly, while the Sobel x,y speedup factor is over 200, the total time in hardware is still comparable to software since the gradient calculation is done in SW, which takes 1.2 seconds. To retain the HW advantage, gradient calculation needs to be implemented in the HW too.

This work can be expanded to implement any 3x3 separable filters for image processing. Finally, HDMI in port could be leveraged to get real time video from the webcam and apply image processing to all incoming frames and then display on the monitor.