Minilab 2 Report

Nathan Woolf
Samuel Cooper
ECE 554 - UW Madison

GitHub Repository

Our team GitHub repository can be accessed through the following link.
https://github.com/nathanwoolf/ECE554SP25_Minilab2/tree/master

**NOTE: The most updated version of our code exists on the side branch “master”. We wanted to avoid a merge to avoid wrecking our project and needing to spend more time on it than what was needed. However, the link should navigate to the correct location in the repository.

Quartus Flow Summary Utilization Observations

The Flow Summary Utilization is included in the GitHub repository linked above. The utilization of the blocks detailed in the project description are listed below:

Implementation

Our implementation between the grayscale conversion and convolution of the image happens in the module *ImageProcessing*. The module takes in a stream of pixel values and performs the necessary transformations before outputting a stream of manipulated pixels.

Due to the nature of the transformations required, we made use of line buffers to stream the data through the module. Making use of the LSBs from iX_Cont and iY_Cont, we were able to gate when the line buffers responsible for convolution would take in a pixel, thus performing the necessary compression for VGA to display the frames from the camera. As a result, the grayscale pixel value was computed on every clock cycle, yet only “used” every other clock cycle. Similarly, when we were imputing an odd indexed row, the grayscale values were not taken.

As for the convolution, using the Sobel functions detailed in the writeup, we made use of two more line buffers, intentionally tying their inputs and taps together in a way that the data would propagate through the register local to *Image Processing* such that the vertical or horizontal edge detection would be performed on the frame. This decision is based on the switches of the FPGA, tied as inputs to the module.

Testbench

Due to the nature of the hardware input from the camera for this project, simulating the entire project was a daunting task. Thus, our team tailored our test bench strictly to the image processing module. Here, we generated random input values to the module with the appropriate control signals set and analyzed the outputs through a self-checking test bench. We tested the simple outputs by toggling the input signal responsible for controlling the grayscale or RGB image, ensuring the outputs were within reason.

We didn't see an obvious way to test the line detection with random input values and thus this was tested through observation by programming the project onto the FPGA and manually verifying its functionality.

```
# RESETTING
# waiting for oDVAL
# PASS: got out data valid signal
# Testing grayscale functionality
# turning on grayscale control signal
# PASS: grayscale is manipulating pixel values
# Testing turning off grayscale
# ALL TEST CASES PASS
# ** Note: $stop    : I:/ece554/ECE554SP25_Minilab2/Testbench/ImageProc_tb.sv(87)
#    Time: 51245 ps  Iteration: 2  Instance: /ImageProc_tb
# Break in Module ImageProc_tb at I:/ece554/ECE554SP25_Minilab2/Testbench/ImageProc_tb.sv line 87
```

Problems

One of the major problems our team faced throughout the implementation of Minilab2 was hurdling the understanding of how the fill buffers implemented in the RAW2RGB module behave. This was a huge block in our ability to proceed with the project. While asking questions in lab time and doing our best to make sense of it, we still struggled. Thus, in order to understand what was physically going on in these modules, our team spent time simulating the fill buffer. Once we invested this time, we had a much better understanding of this IP and were able to proceed with the remainder of the lab.