Minilab 2 Report

Written by Prateek Tandon

Created / Altered Files:

- convolution_top.sv
    - This is a top level module that instantiates everything
- row_buffer.sv
    - This is a row buffer meant to hold 1280 12-bit pixels. It uses Buffer_IP_1280.v, which is the Quartus-provided row buffer for greater optimization.
    - NOTE: Buffer_IP_640.v is now unused (originally it was an alternative)
- bayer_generator.sv
    - This performs the actual grayscale logic by taking in two pixels as input (one meant to be from a row buffer and one that is the current pixel) and averages them to create a grayscale picture.
    - NOTE: This will reduce the image from 1280x960 to 640x480 (which is what the VGA input expects to begin with, so no further modification is needed)
- convolution.sv
    - This performs the actual convolution. It uses two row_buffers that are instantiated so that we can perform operations on a 3x3 grid. It expects the current pixel and the filter. The filter is meant to be either horizontal or vertical but can be changed in convolution_top.sv
    - NOTE: The output pixel is bigger than 12 bits to handle overflow - this is handled in absolute_value
- absolute_value.sv
    - This just takes the absolute value of the returned pixel and scales it back down to 12 bits.
- convolution_top.sv
    - This is the top level module that instantiates everything
- convolution_tb.sv
    - This is a testbench for the top-level module.
    - TO RUN THE TESTBENCH:
        - Pick an image to convert
        - Run ConvertImage.java - this converts the image to Bayer-style pixels that the module expects
        - Run the testbench - you can change parameters at the top to change the image and pixel size. The data is written to a hex file.
        - Run RemakeImage.java - this converts the hex file back into a PNG image so you can see what happened.
    - The testbench is not self-checking and doesn't handle a moving image but it is a good start to make sure convolution works correctly

- DE1_SoC_CAMERA.v
    - This file was altered to include the convolution module.
    - SW[1] changes between regular color mode and convolution mode
    - SW[2] changes the valid signal from the color valid to the convolution valid signal. This doesn't have much of an impact on the actual image.
    - SW[3] changes between the horizontal and vertical filter

My implementation is mainly based off of the diagram written in the directions. I separated each box to be its own separate module, then made a top-level module that instantiates everything. I then altered DE1_SoC_CAMERA.v to still have the original code as an option but also allowing for the convolution video instead. My testbench is meant to test convolution visually so you do not have to see waveforms or examine hex output as you will only be using this module visually, so checking for correct values doesn't imply that the code is working. However, the testbench cannot test a moving image or VGA output since it is not physically connected to a camera when simulating. It still is a good start to ensure the code is working. I wrote both java files to work with the testbench: CreateImage.java converts the image to a hex file that outputs similarly to how the camera provides pixels. RemakeImage.java then takes the created hex file generated by the testbench and converts it to a black and white PNG image so you can physically see the outcome.

I had all sorts of problems which mainly came down to the valid signal not being correct and initially trying to send a 1280x960 image to the VGA output. I originally just ran the testbench and was happy with the output, so I assumed everything would work (big mistake). When run on the FPGA, the output had the same image repeated over and over, indicating there was something wrong with the image size. I then examined Raw2RGB.v since we were writing something "equivalent", so the input and output sizes should be the same. That showed me I needed to scale the image back down. I followed how Raw2RGB made the valid signal as if it worked for their purposes it would likely work for mine as well. Ultimately when instantiating the signal I accidentally misspelled X_cont and Y_cont in the top-level module, resulting in the signal not working correctly which threw off both the convolution (as valid_in was part of the convolution logic) and the timing for displaying pixels on the screen. Once I fixed and corrected this error, the horizontal filter was working correctly. However, the vertical filter was showing a black screen instead of the correct data. I figured out I was updating convolution after every pixel rather than after every valid signal, so changing it to coincide with valid fixes all issues.

The utilization seems small as I only used 2% of the ALM - this is likely because the biggest hardware created was the row buffer, in which I used the IP to create a more optimized, smaller version. The next biggest part of hardware was likely the convolution module as it involved multiple adders and multipliers to work correctly. I used 9% of DSP and no BRAM.