

ECE 554 Minilab2 Report

Srivibhav Jonnalagadda, Sandeep Sankar, Nate Parmley

Implementation

We developed `image_proc.sv` as the top-level module, responsible for either performing convolution or displaying a grayscale image using the input stream buffer. A separate module, `grayscale_conv.sv`, converts the incoming pixel stream into grayscale by interpolating the Bayer color pattern.

The grayscale pixel stream is then passed to the `convolution.sv` module. Control of functionality is handled using onboard switches on the DE1-SoC: `SW[1]` enables grayscale passthrough mode when set high, bypassing convolution. When `SW[1]` is low, convolution is applied. In this mode, `SW[2]` selects the kernel orientation, where a high value corresponds to horizontal edge detection, and a low value selects vertical edge detection.

To support convolution, we instantiate a three-tap line buffer that stores pixel data across multiple rows, providing access to the required 3×3 neighborhood for filtering. Finally, the processed pixel stream is output and displayed via VGA.

Testbench Design

To test our design, we wrote a self-checking testbench to confirm the functionality of our convolution with the vertical and horizontal edge detector filters. The testbench for our top-level image processor tests the convolution by inputting a random matrix and confirming that we achieve the expected output. Knowing the input matrix values, we can determine the expected output of the grayscale conversion and then the expected output of convolution.

To make the expected math easier, the test data only has non-zero values in the first row of the input matrix. Despite this, the convolution results in several rows of non-zero output data that we can check. The image below shows the waveforms of the output values for the first row of convolution with the vertical edge detector. There is also an image showing that our self-checking testbench passed all tests.

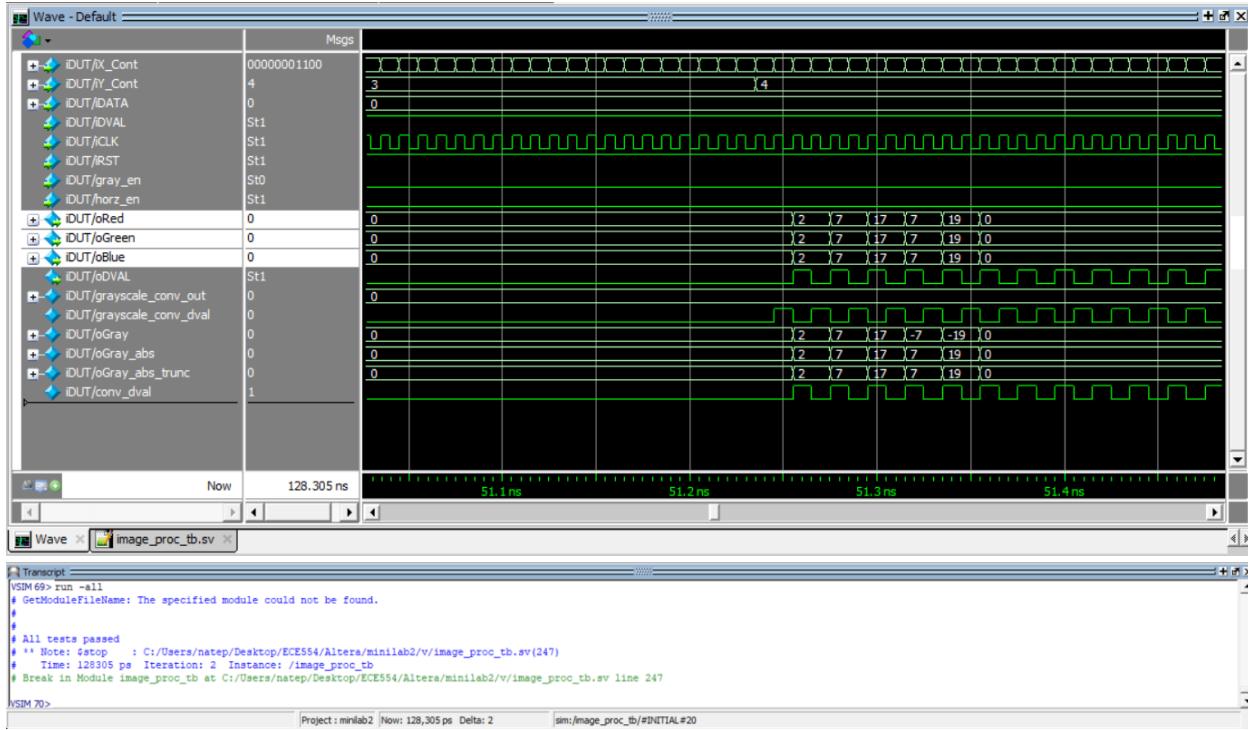


Figure 1. Snapshot showing the log output and waveforms.

Development Challenges and Resolutions

We faced our main challenge when implementing the convolution module. Specifically, we found it challenging to keep track of the age of the data output from the taps in the generated LineBuffer module. This led to errors where we were multiplying the values of the kernel by the wrong values in the matrix when doing our final addition for the convolution.

To solve this, we kept track of the age of the taps value with our naming, where mDATA_0 represented the oldest data from the tap2 port, while mDATA_2 represented the youngest data from the tap0 port. With this convention, we were able to accurately multiply the top row of the filter by the data from the tap2 port (mDATA_0) and the bottom row of the filter by the tap0 port (mDATA_2).

Another challenge we faced was determining whether to truncate the final 18-bit convoluted value down to 12 bits by removing the most-significant bits or the least-significant bits. Initially, we removed the least-significant bits because we thought that the extra noise represented by the smaller bit values could be discarded, as the final pixel information would be primarily encoded in the most-significant bits. However, when we did this, we saw a black screen in our final camera output. By changing the truncation to remove the final output's most-significant bits, we saw that the small fluctuations in pixel values, represented by the LSBs, were necessary to produce enough variation for a visualizable filter output.