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 validate our design, we developed a self-checking testbench to verify the functionality of the convolution module using both vertical and horizontal edge detection filters. The testbench drives the top-level image processor with a randomized input matrix and automatically checks that the produced outputs match the expected results.

Because the input values are known, we can analytically compute the expected outputs after both grayscale conversion and convolution. To simplify verification, the test data is constructed such that only the first row of the input matrix contains non-zero values. Despite this simplification, the convolution operation produces multiple rows of non-zero outputs, which provides a robust basis for correctness checks.

The accompanying waveform demonstrates the output values for the first row when applying the vertical edge detection filter. Additionally, a summary image confirms that the self-checking testbench successfully passed all test cases.

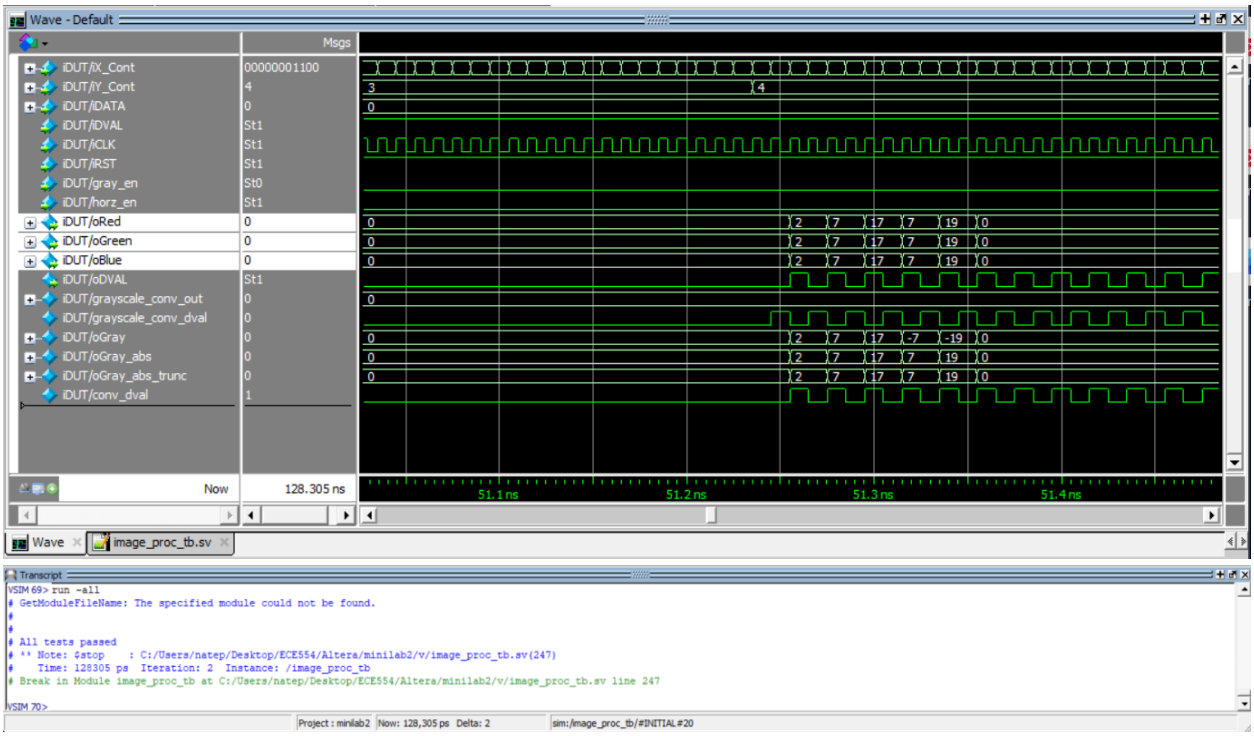

Figure 1. Snapshot showing the log output and waveforms.

Development Challenges and Resolutions

Our primary challenge arose during the implementation of the convolution module. Specifically, we found it difficult to correctly track the “age” of the data output from the taps in the generated LineBuffer module. This resulted in errors where kernel coefficients were multiplied with incorrect pixel values during the final accumulation step of the convolution.

To address this, we introduced a consistent naming convention to reflect data age. In this scheme, mDATA_0 represents the oldest data (from the tap2 port), while mDATA_2 represents the most recent data (from the tap0 port). Using this convention, we were able to correctly align the filter with the pixel window—mapping the top row of the kernel to tap2 (mDATA_0) and the bottom row to tap0 (mDATA_2).

Another challenge involved determining how to reduce the 18-bit convolution result to a 12-bit output. Initially, we truncated the least significant bits (LSBs), assuming they primarily represented noise and that the most significant bits (MSBs) contained the essential pixel information. However, this approach resulted in a completely black output image.

After switching to truncating the most significant bits instead, we obtained a valid image. This indicated that preserving the LSBs was important, as they captured the subtle variations in pixel intensity needed for a visible result. In contrast, keeping only the MSBs caused the output values to become improperly scaled, leading to a loss of visible detail.