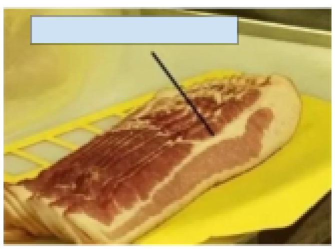
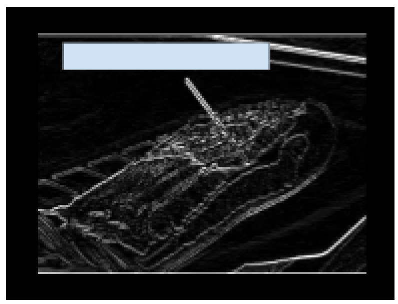
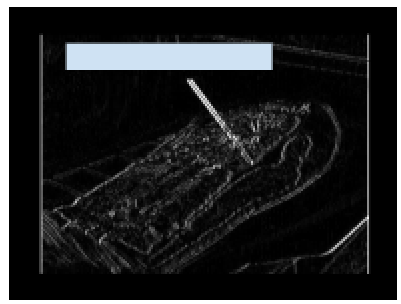
Henry Wysong-Grass
Minilab2
2026-02-11

Minilab2 Report

Testbenching

Below is the output of the testbench. I use a python script to convert an image into 12 bit hex values (this version uses a simplified version of the grayscale) and the testbench then streams this data into the module with several signals before capturing the output as another 12 bit hex value (expanded grayscale). This output is then reconverted back into a png and the outputs of such can be seen below. The full test bench can be found under my GitHub at the ECE554SP26_Minilabs>Minilab2>testbench path as `edge_detect_tb.sv`
The two python scripts are not provided in the repo but can be given upon request. They simply did png↔hex conversions.

Original Image	Vertical Detect	Horizontal Detect
		

Text redacted due to irrelevancy.

Henry Wysong-Grass
Minilab2
2026-02-11

Quartus Flow Summary and Reports

The report file can be found under ECE554SP26_Minilabs>DE1_SoC_CAMERA> Minilab2.flow.rpt

```
+---------------------------------------------------------------------------------+
; Flow Summary                                                                    ;
+---------------------------------+-----------------------------------------------+
; Flow Status                     ; Successful - Tue Feb 10 19:10:43 2026         ;
; Quartus Prime Version           ; 25.1std.0 Build 1129 10/21/2025 SC Lite Edition ;
; Revision Name                   ; DE1_SoC_CAMERA                                ;
; Top-level Entity Name           ; DE1_SoC_CAMERA                                ;
; Family                          ; Cyclone V                                     ;
; Device                          ; 5CSEMA5F31C6                                  ;
; Timing Models                   ; Final                                         ;
; Logic utilization (in ALMs)     ; 741 / 32,070 ( 2 % )                          ;
; Total registers                 ; 1364                                          ;
; Total pins                      ; 226 / 457 ( 49 % )                            ;
; Total virtual pins              ; 0                                             ;
; Total block memory bits         ; 72,144 / 4,065,280 ( 2 % )                    ;
; Total DSP Blocks                ; 0 / 87 ( 0 % )                                ;
; Total HSSI RX PCSs              ; 0                                             ;
; Total HSSI PMA RX Deserializers ; 0                                             ;
; Total HSSI TX PCSs              ; 0                                             ;
; Total HSSI PMA TX Serializers   ; 0                                             ;
; Total PLLs                      ; 1 / 6 ( 17 % )                                ;
; Total DLLs                      ; 0 / 4 ( 0 % )                                 ;
+---------------------------------+-----------------------------------------------+
```

Above is the resource utilization report from the rpt file. In total we use only 2% of available logic, NO dsp blocks and also only 2% of available BRAM. Probably we would have been able to use a dedicated DSP block and reduce our resource utilization more, but without experience using DSP blocks I do not know for certain.

Report of Implementation

Grayscale: Our implementation follows mostly what is described within the project spec. We started by designing the grayscale aspect of the pipeline. This module initially took the output of the RAW2RGB IP and would average the colors, but a final version would take the raw camera input and then do this interpolation and averaging to produce a more accurate grayscale.

Row buffer: I then implemented the row buffer which simply buffers one single line of the camera input (640 12 bit values) and is conditionally enabled. It is implemented as a FIFO with a read and write counter where the read counter leads the write counter by one. This essentially turns our FIFO into a shift register since we have a constant offset between the read and write pointers.

Edge detect: The final module we implemented was the edge detection module which performs the convolution with the provided kernels (which detect method is used is determined by a signal passed from the top level corresponding to an FPGA switch). To do this, we input the grayscale pixels, and begin buffering them. When we have enough values, ie we have at least 2

Henry Wysong-Grass
Minilab2
2026-02-11

lines + 3 pixels saved in our row buffers, we begin the convolution. Since we require a 3x3 matrix to operate on, we have 9 additional registers to store these values. The “top” row of the matrix is captured from the output of the last (oldest) row buffer. The “middle” values come from the output of the first row buffer, and the “bottom” comes from the 3 most recent inputs. This gives us the 9 values we need to operate on the center pixel. We then do signed arithmetic with both sobel kernels, and then select the appropriate one as our output. These values are then sent to the SDRAM before continuing on in the demo pipeline.

Problems

The actual grayscale, edge detect pipeline design and implementation was remarkably smooth. The grayscale and edge detect compiled and worked first try (I think that’s the first time my Verilog worked first try). The biggest issue came with integrating our module into the existing pipeline. We saw 8 strange, changing 1 pixel wide vertical lines on our output. While we were unable to confirm this, I believed that this was likely due to a timing issue with the SDRAM and the basic camera setup as, even when we completely removed our additions/ bypassed them, the lines were present showing the issue was not with our modules. To fix this issue we copied the base demo files and then inserted our module into it. This, after adding several wires, worked perfectly and the edge detect loaded and executed properly with the switch correctly switching the edge detect type.

Files I Made or modified:
row_buffer.sv
edge_detect.sv
RAW2GRAY.v
DE1_SoC_CAMERA.v
Note: (we encountered weird RAM buffering issues (vertical lines appeared) when using the files generated from the system builder so had to switch back to some of the demo files)

Files I Copied:
All SDRAM files, base camera IP and line_buffer1.v.