

Analysis & Synthesis Resource Usage Summary

 <<Filter>>

| | Resource | Usage |
|----|---|-------------------|
| 1 | Estimate of Logic utilization (ALMs needed) | 1352 |
| 2 | | |
| 3 | ▼ Combinational ALUT usage for logic | 1829 |
| 1 | -- 7 input functions | 15 |
| 2 | -- 6 input functions | 393 |
| 3 | -- 5 input functions | 195 |
| 4 | -- 4 input functions | 235 |
| 5 | -- <=3 input functions | 991 |
| 4 | | |
| 5 | Dedicated logic registers | 2153 |
| 6 | | |
| 7 | I/O pins | 226 |
| 8 | Total MLAB memory bits | 0 |
| 9 | Total block memory bits | 111936 |
| 10 | | |
| 11 | Total DSP Blocks | 0 |
| 12 | | |
| 13 | ▼ Total PLLs | 4 |
| 1 | -- PLLs | 4 |
| 14 | | |
| 15 | Maximum fan-out node | D5M_PIXLCLK~input |
| 16 | Maximum fan-out | 970 |
| 17 | Total fan-out | 17970 |
| 18 | Average fan-out | 3.84 |

Implementation Description:

DE1_SOC_CAMERA.v ← convolution_wrapper.v ← convolution.sv

(← indicates that the current module is instantiated in the module that it is pointing to)

Our implementation includes three modes of operation: greyscale, horizontal convolution filter, and vertical convolution filter. The convolution filters are implemented by using 3×3 Sobel-style edge filters for a raster-scanned pixel stream. Based on the switches implemented in the convolution and convolution wrapper, greyscale, horizontal convolution, or vertical convolution is outputted.

Greyscale Implementation:

The greyscale is simply implemented in convolution_wrapper.v by taking the RGB pixel values and using the following equation to assign new RGB values: $\text{value} = 2B + 3R + 3G/8$ (weighted greyscale approximation). We divide by 8 so as not to deal with the complicated division by 3 logic.

Convolution Filter Implementations:

Sobel

| | | |
|----|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Sobel

| | | |
|----|----|----|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

(sobel-x on left, sobel-y on right)

For every valid cycle, both gx (weighted sum using Sobel-x coefficients) and gy (weighted sum using Sobel-y) coefficients are calculated and converted to magnitudes. Based on the switches' orientation (`i_horizontal`) on each clock cycle, either the horizontal convolution (`i_horizontal == 1`) or vertical convolution (`i_horizontal != 1`) is outputted

We generated the ip Line_Buffer2: The design uses two RAM-based line buffers (Line_Buffer2) and a 3×3 register grid (`_internal_grid[3][3]`) that shifts every valid pixel. On each asserted `i_val_valid`, the new pixel enters `_internal_grid[0][0]`, and the top row shifts right across `[0][0] → [0][1] → [0][2]`. The middle row is sourced from the first line buffer output `_internal_fifo_out[0]`,

and the bottom row comes from the second line buffer, allowing the full 3×3 neighborhood to be formed

Testing:

To test, we simply →

Connect necessary ports to DUT →

Drive known pixel stream into DUT →

Compute gx, gy, abs_gx, abs_gy using the same Sobel coefficients →

Apply mode logic: ($i_{horizontal}=1 \rightarrow$ expect abs_gx) and ($i_{horizontal}=0 \rightarrow$ expect $\max(abs_gy - abs_gx, 0)$) →

Compare expected vs actual on each o_val_valid

Problems Encountered:

We ran into two main errors: a reset error and an off-by-one error

Reset error:

We mistakenly assumed the reset signal in the top-level was active high, and we negated it in our design. However, it was actually active low which took us a while to realize.

Off By One Error:

The FIFO we used was off by one, which was messing up the display for the vertical filter. We were able to fix this by redoing our Line_Buffer2 ip to match our implementation (640 pixels → 639 pixels)