1 Simulation results

1.1 Clock cycle count

Answer:

The simulation results demonstrating the system performance and functional correctness are shown in Fig. 1.

```
# Loading sv_std.std
# Loading work.edge_detect_tb(fast)
# Loading work.edge_detect_top(fast)
# Loading work.fifo(fast)
# Loading work.grayscale(fast)
# Loading work.fifo(fast__1)
# Loading work.sobel(fast)
# Simulation Started...
# Finished writing input data.
# Finished reading output data. Total Errors: 0
# Total Clock Cycles: 1555216
# Simulation Finished.
# ** Note: $finish    : ../sv/edge_detect_tb.sv(175)
#    Time: 15552260 ns  Iteration: 1  Instance: /edge_detect_tb
# End time: 20:20:30 on Feb 02,2026, Elapsed time: 0:00:10
# Errors: 0, Warnings: 1
```

(a) Traditional Testbench output showing Total Clock Cycles.

```
# UVM_INFO ../uvm/my_uvm_sequence.sv(30) @ 0: uvm_test_top.env.agent.seqr@@seq [SEQ_RUN] Loading file ../images/copper_720_540.bm
p...
# UVM_INFO ../uvm/my_uvm_sequence.sv(53) @ 15455685: uvm_test_top.env.agent.seqr@@seq [SEQ_RUN] Closing file ../images/copper_720
_540.bmp...
# UVM_INFO /vol/mentor/questa_sim-2019.3_2/questasim/verilog_src/uvm-1.2/src/base/uvm_objection.svh(1270) @ 15552275: reporter [T
EST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ../uvm/my_uvm_monitor.sv(141) @ 15552275: uvm_test_top.env.agent.mon_cmp [MON_CMP_FINAL] Closing file ../ref_images/co
pper_stage2_sobel.bmp...
# UVM_INFO ../uvm/my_uvm_monitor.sv(69) @ 15552275: uvm_test_top.env.agent.mon_out [MON_OUT_FINAL] Closing file ../out/output.bmp
...
# UVM_INFO /vol/mentor/questa_sim-2019.3_2/questasim/verilog_src/uvm-1.2/src/base/uvm_report_server.svh(847) @ 15552275: reporter
 [UVM/REPORT/SERVER]
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   10
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [MON_CMP_FINAL]     1
# [MON_OUT_FINAL]     1
# [Questa UVM]     2
# [RNTST]     1
# [SEQ_RUN]     2
# [TEST_DONE]     1
# [UVM/RELNOTES]     1
# [UVMTOP]     1
#
# ** Note: $finish    : /vol/mentor/questa_sim-2019.3_2/questasim/verilog_src/uvm-1.2/src/base/uvm_root.svh(517)
#    Time: 15552275 ns  Iteration: 70  Instance: /my_uvm_tb
# End time: 20:14:39 on Feb 02,2026, Elapsed time: 0:00:29
# Errors: 0, Warnings: 0
```

(b) UVM Report Summary confirming 0 errors.

Figure 1: Simulation verification results.

Fig. 1a shows the final simulation output from the top-level testbench. The simulation was configured with a clock period of 10 ns (100 MHz). The design processed the entire image in **1,555,216 cycles**. Based on the image resolution of 720×540 pixels (total $388,800$ pixels), the performance metric is

calculated as:

$$\text{Cycles per Pixel} = \frac{1,555,216}{388,800} \approx 4.00 \tag{1}$$

This performance of 4 cycles per pixel is a direct result of the FSM design in the `sobel.sv` module. Unlike simple point-operations (like grayscale), the Sobel filter requires a 3×3 spatial window, necessitating a more complex data flow:

1. **Data Caching (Line Buffers):** To avoid re-reading the image multiple times, we implemented two Line Buffers (`lb0`, `lb1`) to cache the previous two rows of the image. This allows the convolution to occur in a single pass by only reading each pixel once.

2. **3-State FSM Operation:** The architecture uses three states to manage the window update and calculation:

   - **s0 (Compute/Decision):** Determines if enough data is present in the window.
   - **s2 (Read Input):** Reads a new pixel from the input FIFO, shifts the 3×3 window register, and updates the Line Buffers.
   - **s1 (Write Output):** If a valid window center is formed, the Sobel gradient is computed combinatorially in one cycle, and the result is written to the output FIFO.

The sequence of reading a pixel (s2), updating the window logic (s0), and writing the result (s1) results in a steady-state throughput of approximately 4 clock cycles per pixel.

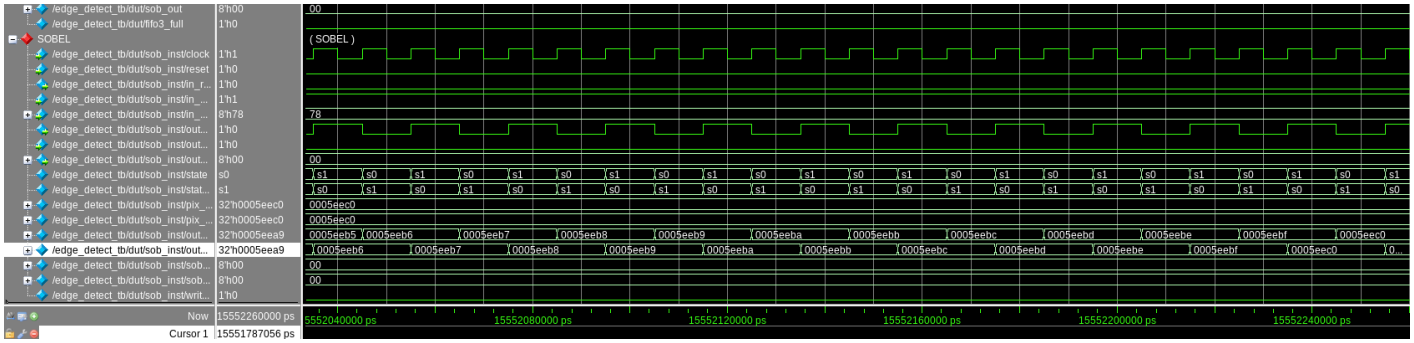


Figure 2: Detailed waveform of the Sobel module in steady-state operation.

Fig. 2 provides a zoomed-in view of the Sobel module's FSM during the active processing phase. The waveform reveals the efficient toggling between states:

- **State s1 (Write Output):** The module writes the calculated Sobel value to the output FIFO and increments the output counter (`out_cnt`).

- **State s0 (Compute/Decision):** Immediately following a write, the FSM returns to s0 to check if the next window is valid and ready for computation.

As observed in the waveform, when the internal line buffers are fully populated, the module can achieve a peak throughput of **2 cycles per pixel** (toggling between s0 and s1). However, the average throughput settles at roughly 4 cycles per pixel due to the periodic need to enter state **s2** to fetch new input data and shift the window.

1.2 Errors reported

Answer:

Functional verification was performed using two methods:

1. **Bit-True Comparison:** The traditional testbench compares the hardware output pixel-by-pixel against a software-generated golden model ("copper_stage2_sobel.bmp"). As shown in Fig. 1a, the **total error count is 0**.

2. **UVM Verification:** To ensure robustness, we also verified the design using a UVM environment. The UVM Scoreboard automatically compared the DUT output transaction stream against the reference model. Fig. 1b confirms that the design passed with **0 UVM Errors** and **0 UVM Fatals**.

2 Synthesis results

2.1 Maximum frequency

Answer:

The estimated maximum frequency of the design is **74.4 MHz**.

As shown in Fig. 3, the synthesis tool reports an estimated frequency of 74.4 MHz. This performance represents a significant improvement over the initial baseline, achieved through targeted datapath optimizations in the `sobel` module.

Timing Summary

Clock Name (clock_name)	Req Freq (req_freq)	Est Freq (est_freq)	Slack (slack)
edge_detect_top|clock	87.5 MHz	74.4 MHz	-2.016
Detailed report		Timing Report View	

Figure 3: Timing summary synthesis report showing an estimated frequency of 74.4 MHz.

The primary frequency limitation was the single-cycle combinational logic required for the 3×3 Sobel convolution. To improve timing, we restructured the arithmetic operations:

1. **Adder Tree Optimization:** Instead of a direct sum of products, we separated the convolution kernel into positive and negative term groups (e.g., `gx_pos` and `gx_neg`). This allowed for parallel accumulation before the final subtraction, reducing the depth of the adder tree.

2. **Efficient Absolute Value Calculation:** By comparing the positive and negative sums directly (e.g., `if gx_pos > gx_neg`), we streamlined the absolute value logic, eliminating the need for intermediate signed arithmetic and 2's complement inversion on the critical path.

These optimizations flattened the logic depth of the Sobel calculation stage, enabling the design to achieve 74.4 MHz while maintaining single-cycle throughput. This frequency supports a processing rate of over 100 frames per second for 720×540 video.

2.2 Registers/LUTs/Logic Elements

Answer:

The resource utilization for the Edge Detection System on the Cyclone IV-E FPGA is summarized below, based on the synthesis area report shown in Figure 4:

1. **Total Registers (Non I/O):** 400
2. **Total Combinational Functions (LUTs):** 921
3. **Total Memory Bits:** 53,504

These resources are utilized to implement the pipelined FSM control logic, the 3×3 convolution arithmetic in the Sobel module, and the address generation for the line buffers. The significant memory usage (53,504 bits) reflects the efficient implementation of the input/output FIFOs and the two 720-pixel line buffers required for the streaming architecture.

Area Summary			
LUTs for combinational functions (total_luts)	921	Non I/O Registers (non_io_reg)	400
I/O Pins	39	I/O registers (total_io_reg)	0
DSP Blocks (dsp_used)	0 (266)	Memory Bits	53504
Detailed report		Hierarchical Area report	

Figure 4: Area summary report detailing the resource usage: 921 LUTs, 400 Registers, and 53,504 Memory Bits.

2.3 Memory utilization

Answer:

The design utilizes **53,504 bits** of memory, as indicated in Figure 4. This significant memory footprint is primarily driven by:

- **Line Buffers:** Two 720-byte line buffers in the Sobel module, essential for the streaming 3×3 convolution window ($720 \times 8 \times 2 = 11,520$ bits).
- **I/O FIFOs:** Three deep FIFOs (Input, Grayscale-to-Sobel, Output), each configured with a depth of 1024 words to robustly handle data rate variations and prevent pipeline stalls.

2.4 Multipliers (DSPs)

Answer:

The DSP block utilization is **0**, as shown in Figure 4.

- **Sobel Convolution:** The kernel coefficients are simple powers of two or small integers (1, 2, -1, -2). These multiplications were optimized into shift operations (e.g., `<< 1`) and additions, which are implemented efficiently using LUTs rather than dedicated DSP multipliers.
- **Grayscale Conversion:** The averaging operation (/3) is implemented as a combinational divider logic using LUTs to ensure bit-true accuracy with the software model, avoiding the need for DSP blocks.

2.5 Worst path(timing analysis)

Answer:

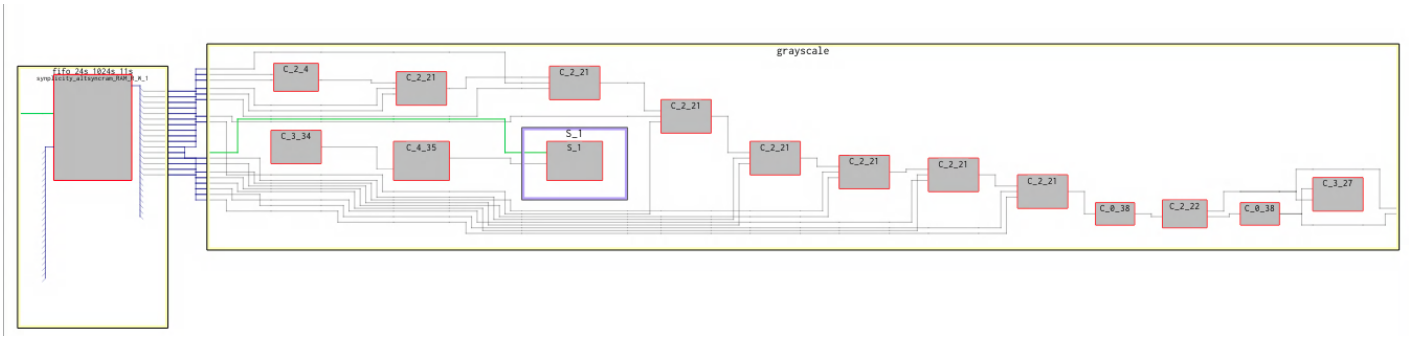
The final critical path analysis reveals that the timing bottleneck has shifted after our optimizations.

1. **Critical Path Slack:** -2.016 ns (at 87.5 MHz target)
2. **Path Delay:** 13.965 ns (Logic: 64.4%, Route: 35.6%)
3. **Start Point:** `fifo_in.fifo_buf / q_b[16]` (Input FIFO Block RAM output)
4. **End Point:** `gs_inst.gs[0] / d` (Grayscale module register input)
5. **Logic Levels:** 23

Initially, the design's maximum frequency was limited to approximately 50.3 MHz by the complex 3×3 convolution logic in the Sobel module. However, after successfully restructuring the Sobel arithmetic (as detailed in the previous section), the timing bottleneck has shifted to the `Grayscale` module.

As shown in the RTL schematic (Fig. 5), the current worst path originates from the read port of the Input FIFO, propagates through a combinational logic cloud, and terminates at the `gs_inst` internal register. This path implements the pixel averaging operation: $Gray = (R + G + B)/3$. To ensure bit-true accuracy with the software model, we retained the exact integer division logic rather than using an approximation. This division logic, synthesized into a deep adder/subtractor tree (23 logic levels), now dictates the maximum frequency of 74.4 MHz.

Figure 6 confirms this systematic limitation. The uniform negative slack across the FIFO output bits indicates that the RGB-to-Grayscale conversion is the new critical constraint. Despite this, the achieved 74.4 MHz is a significant improvement over the initial Sobel-limited performance.



(a) RTL Schematic of the worst timing path: FIFO (left) to Grayscale logic (center).

Figure 5: Visual analysis of the critical path showing the combinational delay through the Grayscale arithmetic logic.

2.6 Schematic architecture(RTL)

Answer:

The RTL architecture implements a modular, distributed control scheme where each processing stage operates independently, decoupled by FIFO buffers to maximize throughput and isolate clock domains.

The architecture consists of the following key components:

Starting Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Arrival Time	Slack
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[16]	fifo1_dout_16	4.154	-2.016
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[8]	fifo1_dout_8	4.154	-2.005
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[17]	fifo1_dout_17	4.154	-1.950
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[9]	fifo1_dout_9	4.154	-1.944
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[18]	fifo1_dout_18	4.154	-1.884
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[10]	fifo1_dout_10	4.154	-1.878
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[19]	fifo1_dout_19	4.154	-1.818
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[11]	fifo1_dout_11	4.154	-1.812
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[20]	fifo1_dout_20	4.154	-1.752
fifo_in.fifo_buf	edge_detect_top|clock	synplicity_altsyncram_RAM_R_W_1	q_b[12]	fifo1_dout_12	4.154	-1.746

(a) Starting points (FIFO outputs).

Ending Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Required Time	Slack
gs_inst.gs[0]	edge_detect_top|clock	dffeas	d	gs_1_0_0__g0_i_x4	11.949	-2.016
gs_inst.gs[1]	edge_detect_top|clock	dffeas	d	multl_un68_sum_add3	11.949	-1.233
gs_inst.gs[2]	edge_detect_top|clock	dffeas	d	multl_un61_sum_add3	11.949	-0.237
gs_inst.gs[3]	edge_detect_top|clock	dffeas	d	multl_un54_sum_add3	11.949	0.791
sob_inst.sob_res[0]	edge_detect_top|clock	dffeas	d	sob_res_1_0_0__g0_0	11.949	0.842
sob_inst.sob_res[1]	edge_detect_top|clock	dffeas	d	sob_res_1_0_1__g0_0	11.949	0.842
sob_inst.sob_res[2]	edge_detect_top|clock	dffeas	d	sob_res_1_0_2__g0_0	11.949	0.842
sob_inst.sob_res[3]	edge_detect_top|clock	dffeas	d	sob_res_1_0_3__g0_0	11.949	0.842
sob_inst.sob_res[4]	edge_detect_top|clock	dffeas	d	sob_res_1_0_4__g0_0	11.949	0.842
sob_inst.sob_res[5]	edge_detect_top|clock	dffeas	d	sob_res_1_0_5__g0_0	11.949	0.842

(b) Ending points (Grayscale registers).

Figure 6: Synthesis timing report showing uniform slack violations, confirming the Grayscale bottleneck.

a. **Top-Level Pipeline**: A linear cascade of `Grayscale` and `Sobel` modules connected by deep FIFOs (1024-word depth).

b. **Grayscale Unit**: Converts 24-bit RGB inputs to 8-bit luminance using arithmetic datapath.

c. **Sobel Processing Unit**: The core of the design, featuring a 3-state FSM that manages a 3×3 sliding window using internal line buffers and performs single-cycle convolution.

Figure 7 illustrates the system-level integration. The distinct yellow blocks represent the FIFOs (`fifo_in`, `fifo_gs_sob`, `fifo_out`) that isolate the processing stages. The central blocks show the `gs_inst` (Grayscale) feeding into the `sob_inst` (Sobel), depicting the streaming data flow.

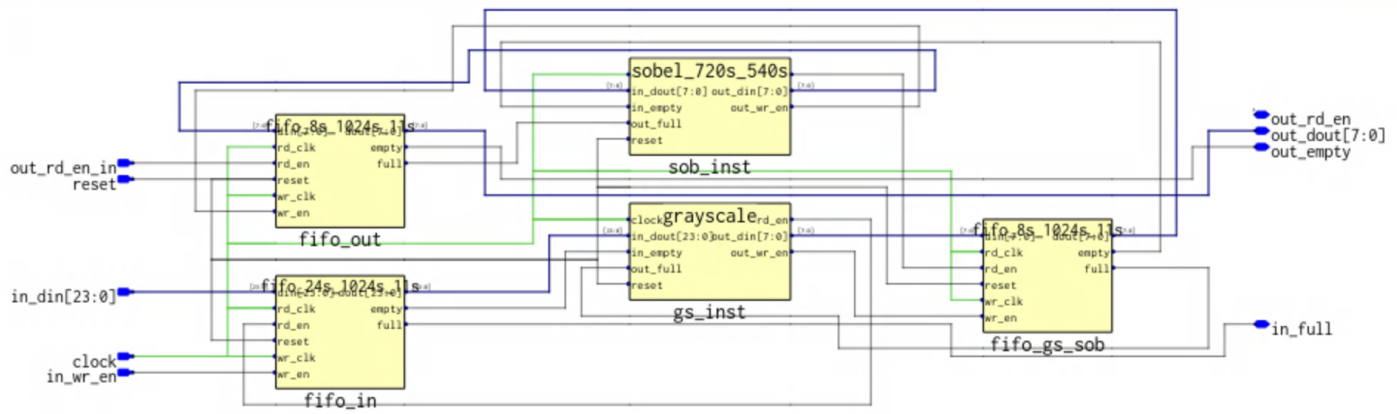


Figure 7: Top-level RTL schematic of `edge_detect_top`, showing the pipeline: Input FIFO → Grayscale → FIFO → Sobel → Output FIFO.

Figure 8 reveals the internal datapath of the Grayscale module. The logic implements the equation $Gray = (R + G + B)/3$. The visible adder chain sums the color channels before the division logic generates the final 8-bit output.

Unlike simple point-processing modules, the Sobel filter requires a sophisticated controller. Figure 9 shows the extracted 3-state FSM. This controller is responsible for managing the read pointers for the internal line buffers, shifting the 3×3 window registers, and coordinating the single-cycle arithmetic convolution.

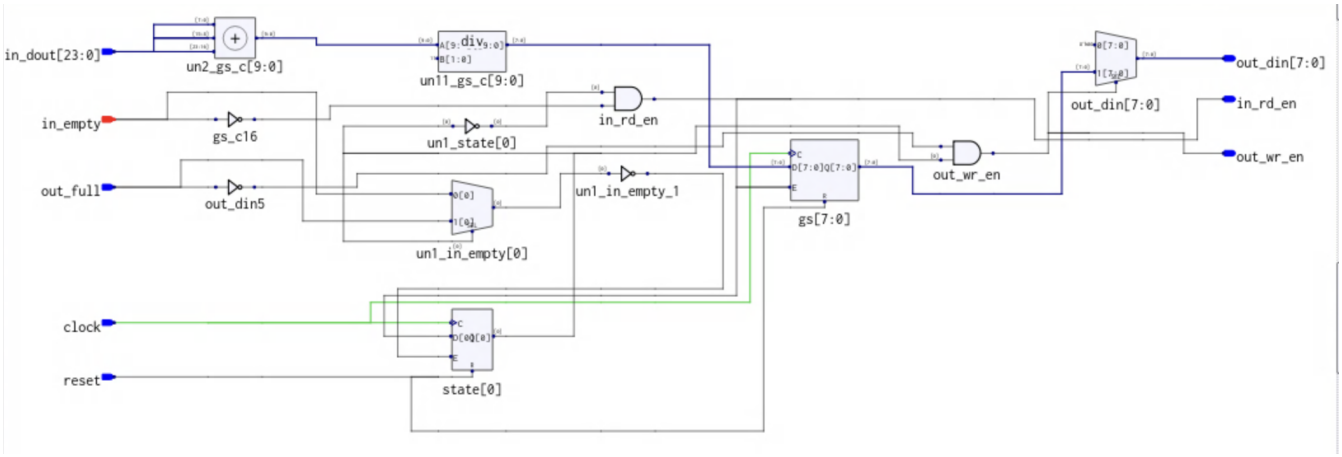


Figure 8: RTL schematic of the `grayscale` module. The synthesis tool has inferred an adder tree (top left) to sum the RGB channels, followed by the division logic to compute the average luminance.

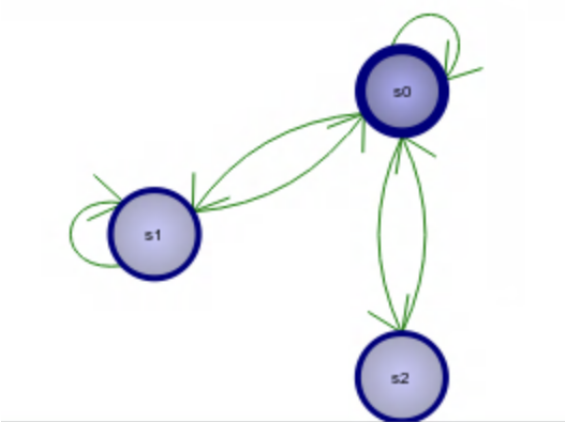


Figure 9: FSM of the Sobel module. The 3-state controller manages the complex sequence of Reading Input (s2), Window Update/Calculation (s0), and Writing Output (s1).

Figure 10 provides a close-up view of the optimized Sobel computation logic. The synthesis tool has mapped the SystemVerilog arithmetic descriptions into a highly parallel structure:

- The logic at the top right implements the optimized `abs_gx + abs_gy` calculation using parallel adders and multiplexers. This structure reflects our design choice to separate positive and negative convolution terms before subtraction, reducing logic depth.
- The registers at the bottom left (`x_cnt`, `y_cnt`) implement the coordinate tracking counters. These counters replaced the original modulo/division logic for boundary checking, significantly reducing the critical path delay and allowing the design to achieve higher operating frequencies.

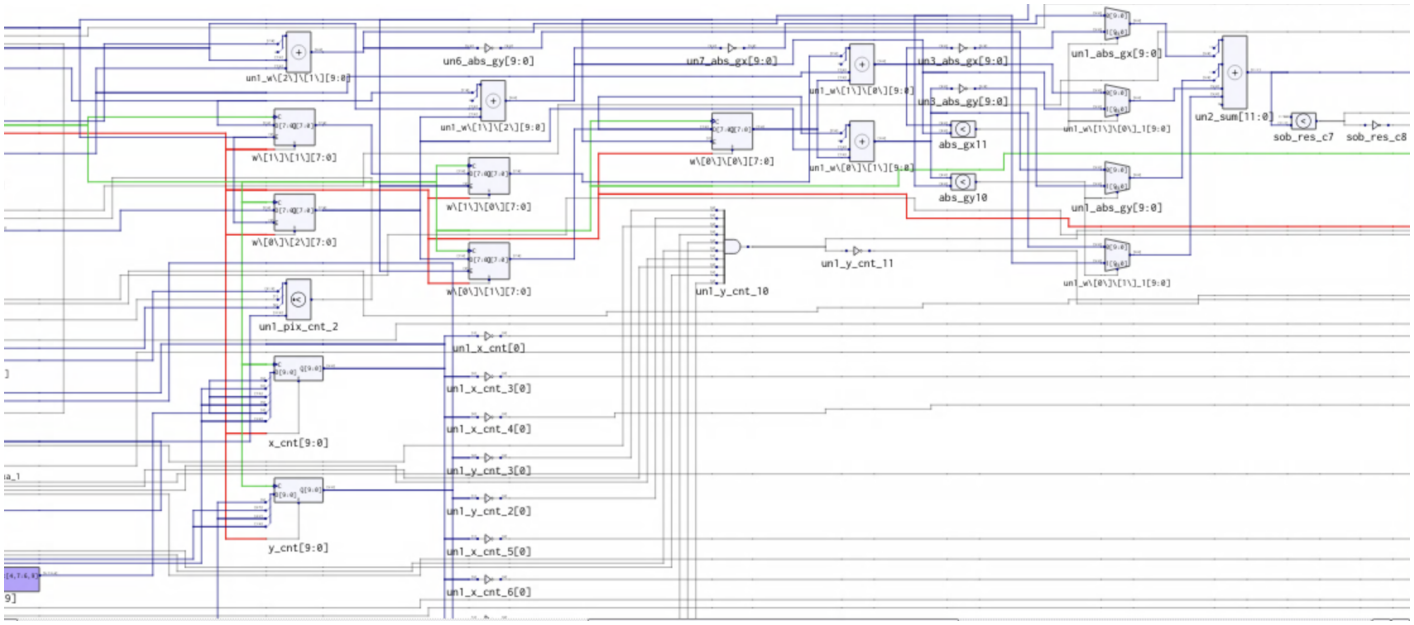


Figure 10: Detailed RTL view of the Sobel arithmetic datapath. The schematic shows the optimized adder tree and absolute value logic (top right) operating in parallel with the coordinate counters (bottom left).

2.7 Performance / Speedup

Answer:

We compared the performance of the hardware design against the software reference implementation.

- **Software Baseline:** The C program was executed on the lab server. Figure 11 shows the execution time for the Grayscale and Sobel stages on a 720×540 image was **0.030391 seconds**.

$$\text{Software FPS} = \frac{1}{0.030391} \approx 32.9 \text{ FPS} \tag{2}$$

- **Hardware Acceleration:** The synthesized design operates at **74.4 MHz** and requires **1,555,216 cycles** per frame.

$$\text{Hardware Time} = \frac{1,555,216}{74.4 \times 10^6} \approx 0.0209 \text{ seconds} \tag{3}$$

- **Speedup Factor:**

$$\text{Speedup} = \frac{0.030391}{0.0209} \approx \mathbf{1.45}\times \tag{4}$$

The FPGA achieves a 1.45x speedup over the CPU despite running at a significantly lower clock frequency (74 MHz vs 3 GHz). This is due to the pipeline architecture processing data in parallel, whereas the CPU executes instructions sequentially.

```
[gel8580@moore edge_detect]$ ./source/edgedetect_test ./images/copper_720_540.bmp
reading file...
Software execution time: 0.030391 seconds
```

Figure 11: Software execution time measurement.

2.8 Throughput (Frames-per-second)

Answer:

Based on the synthesis result of 74.4 MHz and the simulation result of 1,555,216 cycles per frame, the maximum achievable throughput is calculated as:

$$\text{FPS} = \frac{f_{max}}{\text{Cycles per Frame}} = \frac{74,400,000}{1,555,216} \approx \mathbf{47.8\ FPS} \tag{5}$$

For a standard video requirement of 30 FPS, the required frequency would be:

$$f_{req} = 30 \times 1,555,216 \approx 46.6\ \text{MHz} \tag{6}$$

Our achieved frequency of 74.4 MHz provides a **59% timing margin** over the 30 FPS requirement, making it suitable for real-time video processing of 720×540 streams.