1 Simulation results

1.1 Clock cycle count

Answer:

The simulation results demonstrating the performance improvement and functional correctness are shown in Fig. 1 and Fig. 2.

```
# Loading sv_std.std
# Loading work.matmul_tb(fast)
# Loading work.matmul_top(fast)
# Loading work.bram(fast)
# Loading work.matmul(fast)
# @ 20: Loading file y.txt into BRAM B...
# @ 20: Loading file x.txt into BRAM A...
# @ 675: Beginning simulation...
# @ 7095: Simulation completed.
# @ 7100: Verifying results against z.txt...
# Total simulation cycle count: 642
# Total error count: 0
# SUCCESS: Simulation passed!
```

(a) Initial implementation: 642 cycles.

```
# Loading sv_std.std
# Loading work.matmul_tb(fast)
# Loading work.matmul_top(fast)
# Loading work.bram(fast)
# Loading work.matmul(fast)
# @ 20: Loading file y.txt into BRAM B...
# @ 20: Loading file x.txt into BRAM A...
# @ 675: Beginning simulation...
# @ 5825: Simulation completed.
# @ 5830: Verifying results against z.txt...
# Total simulation cycle count: 515
# Total error count: 0
# SUCCESS: Simulation passed!
```

(b) Optimized implementation: 515 cycles.

Figure 1: Comparison of simulation cycle counts. The initial design (a) consumed 642 cycles, while the optimized pipelined design (b) achieved near-theoretical performance with 515 cycles.

Fig. 1a shows the initial simulation result of 642 cycles. This overhead originated from the Finite State Machine spending extra cycles in separate states for data preloading ('S_PRELOAD') and writing back results ('S_WRITE'). For an 8×8 matrix multiplication, this resulted in 64 cycles wasted on preloading and another 64 cycles on write-back operations ($512 + 64 + 64 \approx 640$).

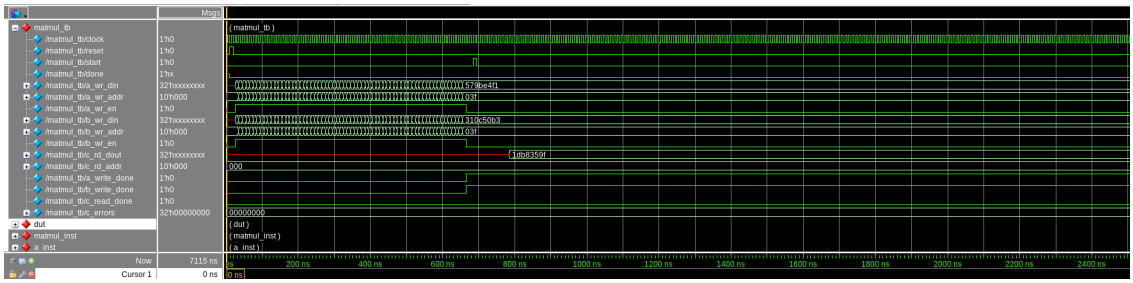
To optimize this, I merged the memory operations into the computation pipeline, as shown in Fig. 1b. By performing the write back of the current result (C_{ij}) and the prefetch of the next data (A_{next}, B_{next}) simultaneously within the last cycle of the inner loop, we eliminated the idle states. This reduced the total execution time to **515 cycles**, which is very close to the theoretical minimum of $N^3 = 512$ cycles, with only minimal overhead for pipeline startup and termination.

The functional correctness is verified by the waveforms in Fig. 2. Fig. 2a captures the initialization phase where the testbench writes input data to Block RAMs A and B before triggering the 'start' signal. Fig. 2b shows the completion of the matrix multiplication. Upon the assertion of the done signal, the testbench reads the results from Block RAM C ('c_rd_addr') to compare against the golden model. The error counter 'c_errors' remains zero throughout the simulation, confirming bit-true accuracy.

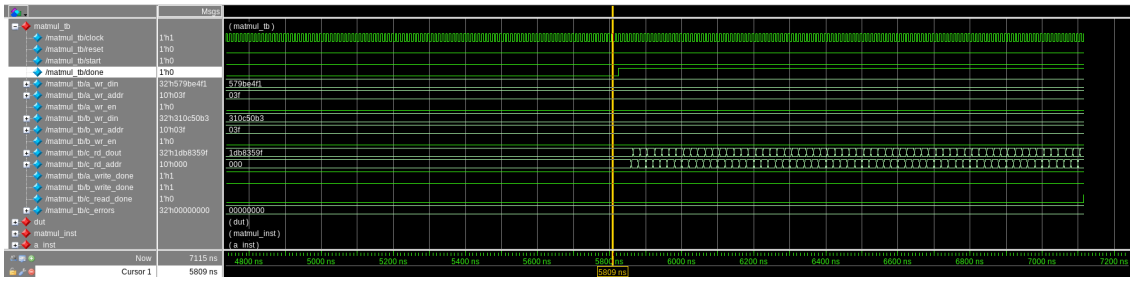
1.2 Errors reported

Answer:

The simulation reported **0 errors**. As demonstrated in the testbench output (Fig. 2b), the error counter `c_errors` remained at zero throughout the verification phase, the hardware implementation is accurate compared to the C-model golden reference.



(a) Loading and Computation Start Phase.



(b) Computation Done and Verification Phase.

Figure 2: Simulation waveforms verifying functional correctness. (a) shows the testbench loading input matrices A and B into Block RAMs, followed by the assertion of the `start` signal. (b) shows the completion of the operation (`done` signal) and the verification phase where results are read from Block RAM C.

2 Synthesis results

2.1 Maximum frequency

Answer:

The estimated maximum frequency of the design is **70.8 MHz**.

As shown in Fig. 3, the synthesis tool reports an estimated frequency of 70.8 MHz against a requested frequency of 83.3 MHz, resulting in a negative slack of -2.119 ns. This frequency limitation is a direct consequence of the single-cycle pipeline optimization strategy. To achieve the minimal cycle count (515 cycles), the critical path includes memory read access, a 32-bit multiplication, and a 32-bit accumulation within a single clock cycle, which naturally increases the path delay and limits the maximum operating frequency.

Timing Summary

Clock Name (clock_name)	Req Freq (req_freq)	Est Freq (est_freq)	Slack (slack)
matmul_top|clock	83.3 MHz	70.8 MHz	-2.119
Detailed report		Timing Report View	

Figure 3: Timing summary synthesis report. The estimated frequency is 70.8 MHz, constrained by the long combinational path required for single-cycle MAC operations.

2.2 Registers/LUTs/Logic Elements

Answer:

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

1. Total Registers: 175
2. Total Combinational Functions (LUTs): 325
3. Logic Elements: 325 (Based on total combinational functions utilization)

These resources are used to implement the 32-bit datapath (including loop counters i, j, k and the accumulator) and the pipelined finite state machine logic required for the matrix operations.

Area Summary			
LUTs for combinational functions (total_luts)	325	Non I/O Registers (non_io_reg)	175
I/O Pins	132	I/O registers (total_io_reg)	0
DSP Blocks (dsp_used)	3 (266)	Memory Bits	98304
Detailed report		Hierarchical Area report	

Figure 4: Area summary report detailing the resource usage, including 325 LUTs and 175 registers.

2.3 Memory utilization

Answer:

The design utilizes **98,304 bits** of memory, as indicated in the "Memory Bits" field of Figure 4. This significant memory usage corresponds to the three Block RAM instances (matrices A, B, and C), each configured as a 1024×32-bit memory block, which are necessary to store the input matrices and the computation results.

2.4 Multipliers (DSPs)

Answer:

The DSP block utilization is **3 (1.13%)**, as shown in Figure 4. Unlike simple addition-based algorithms, matrix multiplication requires intensive multiplication operations. The synthesis tool inferred 3 DSP blocks to implement the 32-bit hardware multiplier within the datapath, optimizing the performance of the `acc = acc + A * B` operation.

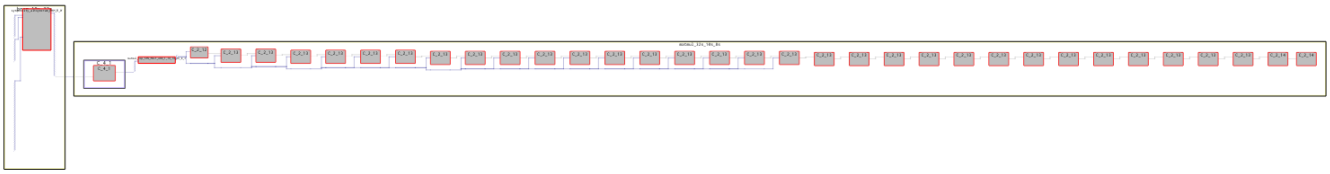
2.5 Worst path(timing analysis)

Answer:

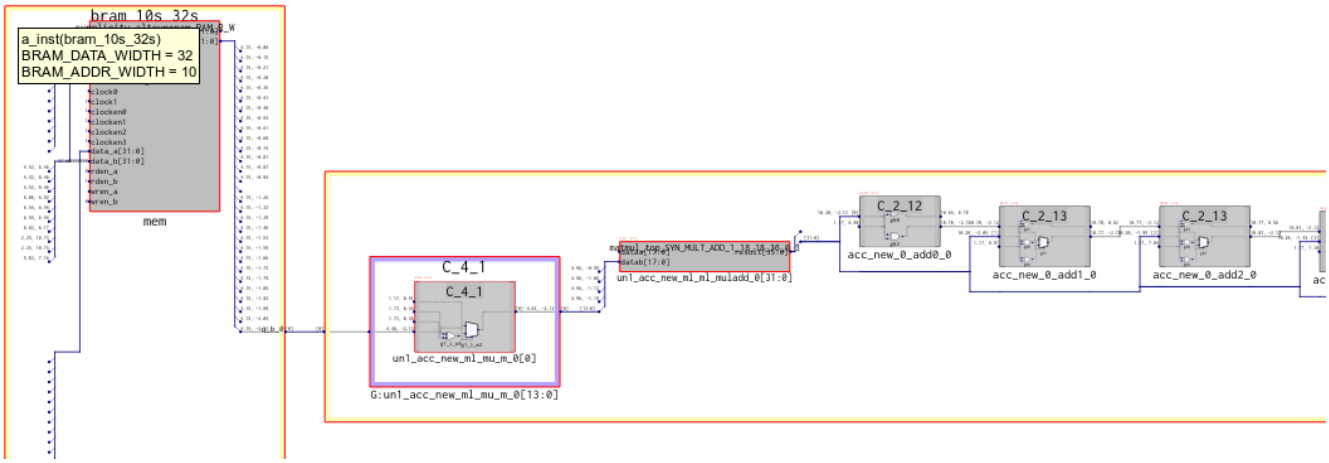
1. **Critical Path Slack:** -2.119 ns (at 83.3 MHz target)
2. **Path Delay:** 14.653 ns
3. **Start Point:** `a_inst.mem / q_b[0]` (Block RAM A output)
4. **End Point:** `c_inst.mem.datareg[31] / d` (Block RAM C input)
5. **Logic Levels:** 36

The critical path analysis reveals a structural bottleneck in the optimized single-cycle pipeline architecture. Figure 5 illustrates this path, originating from Block RAM A (Fig. 5b), traversing the hardware multiplier and a deep 32-bit accumulator carry chain (Fig. 5a), and terminating at Block RAM C. The 36 logic levels limit the maximum frequency to 70.8 MHz.

Furthermore, Figure 6 confirms this is a parallel datapath issue. The synthesis report lists multiple starting (Fig. 6a) and ending points (Fig. 6b) with nearly identical negative slack (-2.119 ns). This uniformity across data bits indicates the delay is inherent to the single-cycle "Read-Modify-Write" architecture rather than a specific routing anomaly.



(a) Critical path logic schematic showing the long adder chain.



(b) Start of the critical path at Block RAM A.

Figure 5: RTL Schematic of the worst timing path. The highlighted path shows the significant combinational delay from memory read (b) through the accumulator (a) to memory write.

Starting Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Arrival Time	Slack
a_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W	q_b[0]	q_b_0	4.154	-2.119
b_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_0	q_b[0]	q_b_0_0	4.154	-2.119
b_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_0	q_b[1]	q_b_0_1	4.154	-2.053
a_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W	q_b[1]	q_b_1	4.154	-2.053
b_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_0	q_b[2]	q_b_0_2	4.154	-1.987
a_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W	q_b[2]	q_b_2	4.154	-1.987
b_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_0	q_b[3]	q_b_0_3	4.154	-1.921
a_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W	q_b[3]	q_b_3	4.154	-1.921
b_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_0	q_b[4]	q_b_0_4	4.154	-1.855
a_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W	q_b[4]	q_b_4	4.154	-1.855

(a) Starting points with worst slack.

Ending Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Required Time	Slack
c_inst.mem.datareg[31]	matmul_top|clock	dffeas	d	c_din_combout_13	12.534	-2.119
c_inst.mem.datareg[30]	matmul_top|clock	dffeas	d	c_din_combout_12	12.534	-2.053
c_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_1	data_a[31]	c_din_31	12.985	-1.994
c_inst.mem.datareg[29]	matmul_top|clock	dffeas	d	c_din_combout_11	12.534	-1.987
c_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_1	data_a[30]	c_din_30	12.985	-1.928
c_inst.mem.datareg[28]	matmul_top|clock	dffeas	d	c_din_combout_10	12.534	-1.921
c_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_1	data_a[29]	c_din_29	12.985	-1.862
c_inst.mem.datareg[27]	matmul_top|clock	dffeas	d	c_din_combout_9	12.534	-1.855
c_inst.mem	matmul_top|clock	synplicity_altsyncram_RAM_R_W_1	data_a[28]	c_din_28	12.985	-1.796
c_inst.mem.datareg[26]	matmul_top|clock	dffeas	d	c_din_combout_8	12.534	-1.789

(b) Ending points with worst slack.

Figure 6: Synthesis timing report showing uniform slack violations across multiple bits, confirming an architectural bottleneck.

2.6 Schematic architecture(RTL)

Answer:

The RTL schematic mainly consists of:

a. A 3-state Finite State Machine (`S_IDLE`, `S_PRELOAD`, `S_RUN`) that orchestrates the pipeline execution.

b. Three 32-bit Loop Counters: i, j, k for iterating through the matrix rows and columns.

c. One 32-bit Accumulator Register (Internal to `matmul` core)

d. One 32-bit Multiplier: For computing $A \times B$.

e. One 32-bit Adder (Part of the MAC unit within `matmul`)

f. Address Generation Logic: Combinational logic that calculates read/write addresses ($i \times N + k$, etc.) for the three Block RAMs based on current counter values.

The RTL architecture of the matrix multiplication design is visualized in the following figures, generated from the Synplify Premier RTL Viewer.

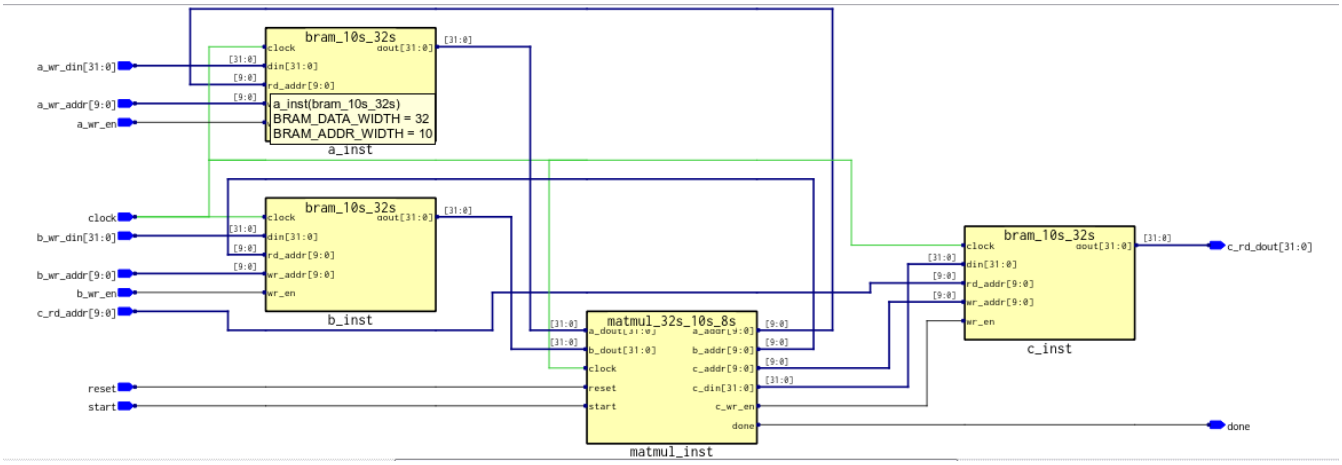


Figure 7: Top-level schematic view showing the `matmul_top` module. It depicts the `matmul` core instantiated in the center, interfacing with three Block RAM instances: `a_inst` and `b_inst` for reading input matrices, and `c_inst` for writing the result matrix.

Figure 7 illustrates the system integration. The central `matmul` module generates addresses (`a_addr`, `b_addr`, `c_addr`) to access the memory blocks and receives data from `a_inst` and `b_inst` to perform computations, finally writing the output to `c_inst`.

The control logic is governed by the optimized FSM shown in Figure 8. This design utilizes a simplified 3 state machine. The `S_RUN` state handles the continuous pipeline execution, including simultaneous calculation, memory write-back, and next-address prefetching, which is the key to achieving the high throughput of 515 cycles.

Figure 9 provides a glimpse into the internal datapath logic controlled by the FSM. The schematic reveals the complexity of the combinational logic required to manage the loop counters and the accumulator within the single-cycle pipeline.

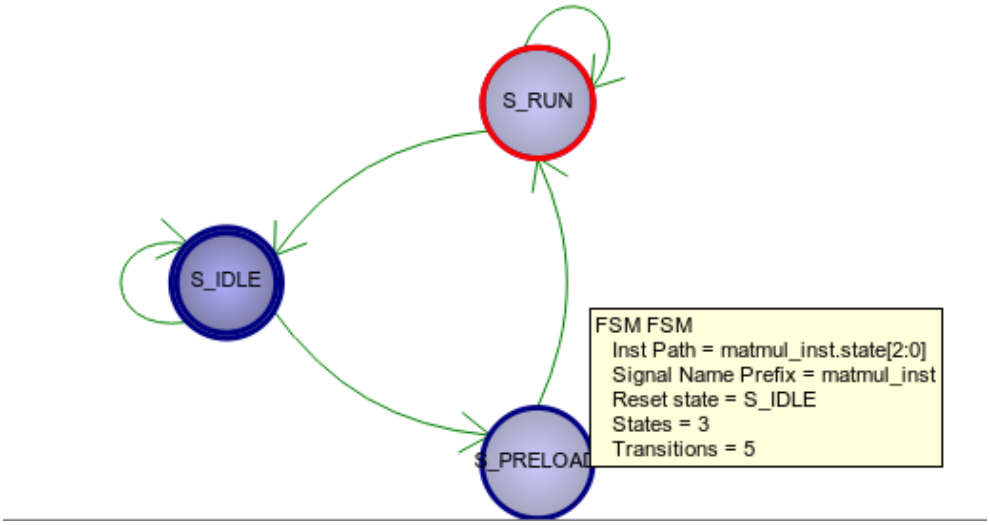


Figure 8: State Transition Diagram of the optimized Finite State Machine (FSM). The FSM consists of only three states: `S_IDLE`, `S_PRELOAD`, and the main pipeline state `S_RUN`, minimizing control overhead.

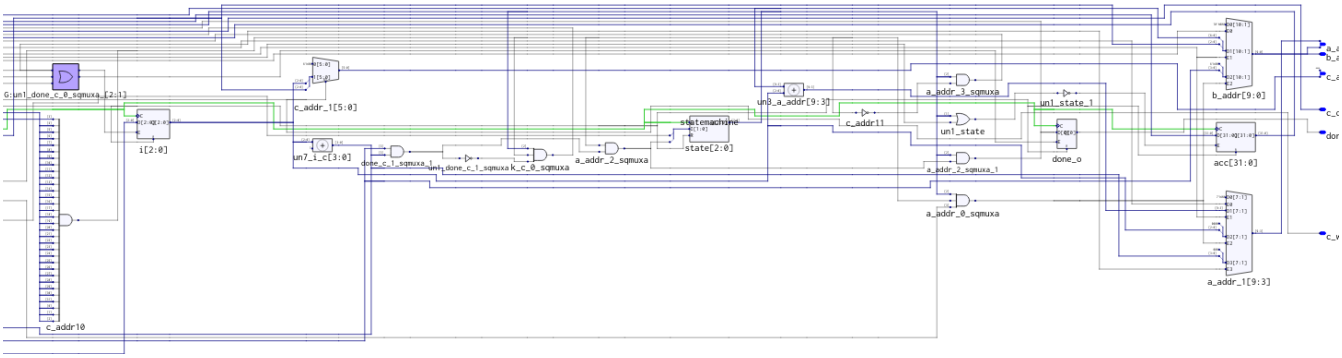


Figure 9: Detailed RTL schematic of the `matmul` core datapath. It shows the `state_machine` block (center) generating control signals for the datapath components, including the accumulator adders and multiplexers for loop counters (i, j, k).