Minilab1A and Minilab1B Report

Prateek Tandon

Minilab 1A

Github Information

GitHub Repo Link: https://github.com/IWinHa/ECE554_Tandon

I made the repo in Minilab 0 so some of the steps I took were applicable to the earlier lab and not this one. Some steps I took are below:

Madi first created the repository as she had all the Vivado files on her personal PC. She added everyone to the group so we had access to the screenshots and v/sv/IP files needed.

I then followed instructions at https://stackoverflow.com/questions/6613166/how-to-duplicate-a-git-repository-without-forking to duplicate the repo separately so I can make personal changes without affecting Madi's original repo. I then downloaded git on my CAE machine and cloned the repo to my PC as normal.

For Minilab 1A and 1B, I used the same repo and added new folders for the later minilabs.

Why was resource utilization different?

The Quartus IP catalog has highly optimized versions of certain blocks, which take advantage of infrastructure on the board and/or utilizes them in a way that is hard to recreate by an engineer. Due to this, the utilization on the board will be different in terms of signal routes (where to place signals due to the new IP), area usage, and timing (shown in ALM and Register count difference)

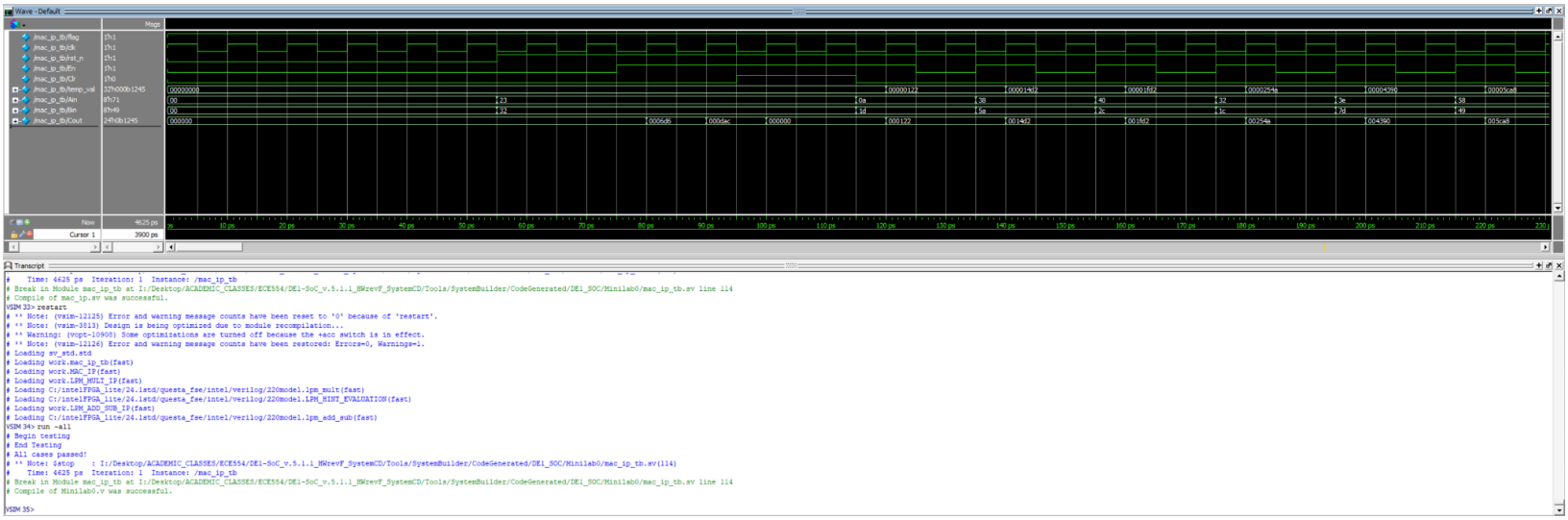
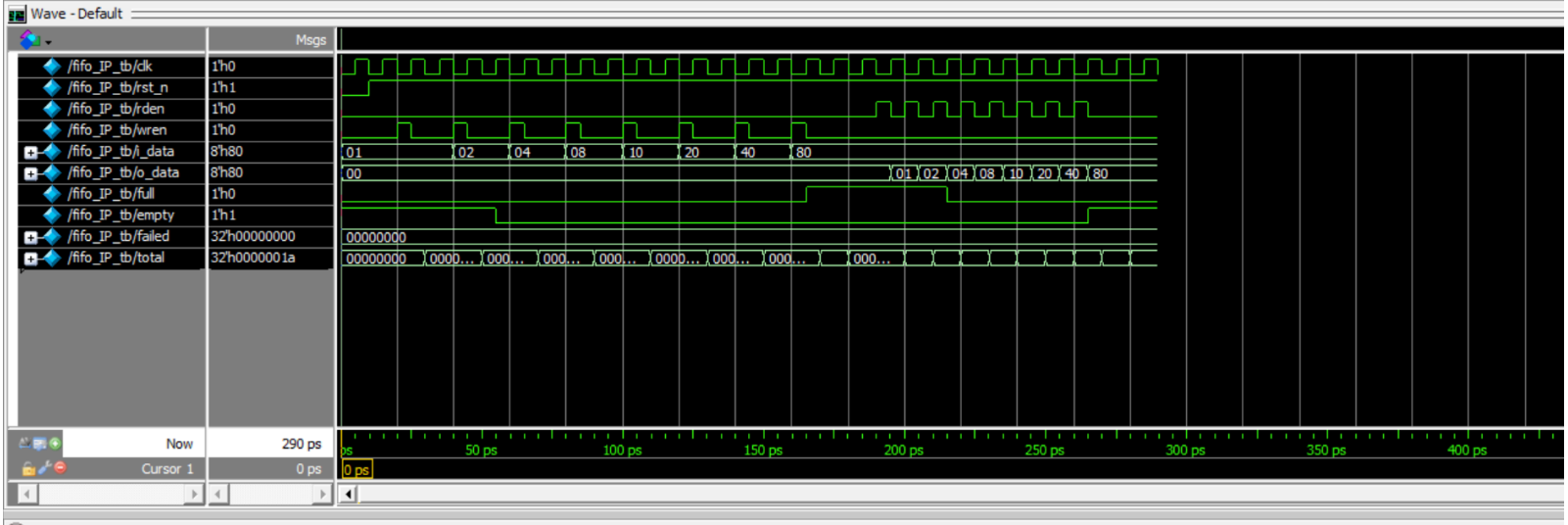


Figure: MAC works correctly



Transcript

```
# ** Note: (vsim-8009) Loading existing optimized design _opt8
# ** Note: (vsim-12126) Error and warning message counts have been restored: Errors=0, Warnings=6.
# Loading work.fifo_IP_tb(fast)
# Loading work.FIFO_IP(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_mixed_widths(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.ALTERA_DEVICE_FAMILIES(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_async(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.ALTERA_DEVICE_FAMILIES(fast__1)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_dffpipe(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_dffpipe(fast__1)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_dffpipe(fast__2)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_fefifo(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_fefifo(fast__1)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_low_latency(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.ALTERA_DEVICE_FAMILIES(fast__2)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.ALTERA_MF_HINT_EVALUATION(fast)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_dffpipe(fast__3)
# Loading C:/intelFPGA_lite/24.1std/questa_fse/intel/verilog/altera_mf.dcfifo_dffpipe(fast__4)
VSIM 39> run -all
# Yahoo!! All tests passed! 26/26
# ** Note: $stop    : I:/Desktop/ACADEMIC_CLASSES/ECE554/DE1-SoC_v.5.1.1_HWrevF_SystemCD/Tools/SystemBuilder/CodeGenerated/DE1_SOC/Minilab0/fifo_IP_tb.sv(113)
#    Time: 290 ps  Iteration: 1  Instance: /fifo_IP_tb
# Break in Module fifo_IP_tb at I:/Desktop/ACADEMIC_CLASSES/ECE554/DE1-SoC_v.5.1.1_HWrevF_SystemCD/Tools/SystemBuilder/CodeGenerated/DE1_SOC/Minilab0/fifo_IP_tb.sv line 113

VSIM 40>
```

Figure: FIFO Works Correctly

All figures are provided as png images in the zip file if the provided picture(s) are too small.

Minilab1B

How did we test the design?

Our group split up tasks where one person wrote the implementation for feed_from_mem, one person wrote the MAC array logic, and one person wrote a testbench. To debug, we wrote a smaller testbench that quickly checked the end result rather than the entire ALU process that the later testbench would end up checking. Both feed_from_mem and the MAC array had their own dedicated testbenches that we used for debugging. These testbenches were vital in ensuring the Verilog code we made would actually work correctly and as intended.

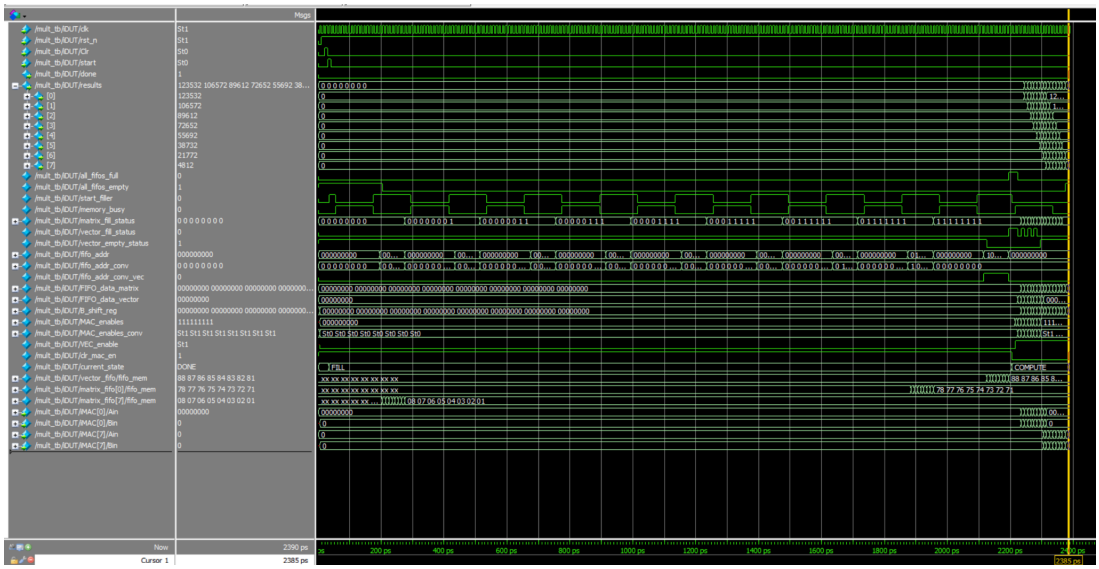


Figure: “Quick” Testbench

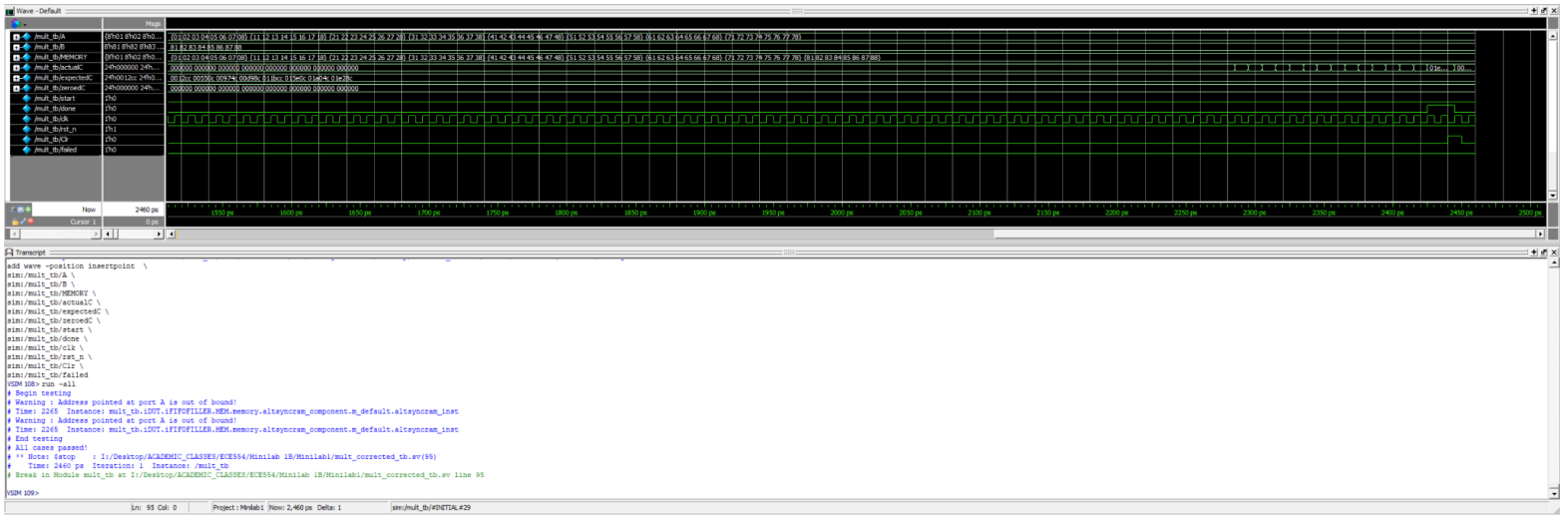


Figure: feed_from_mem testbench passing

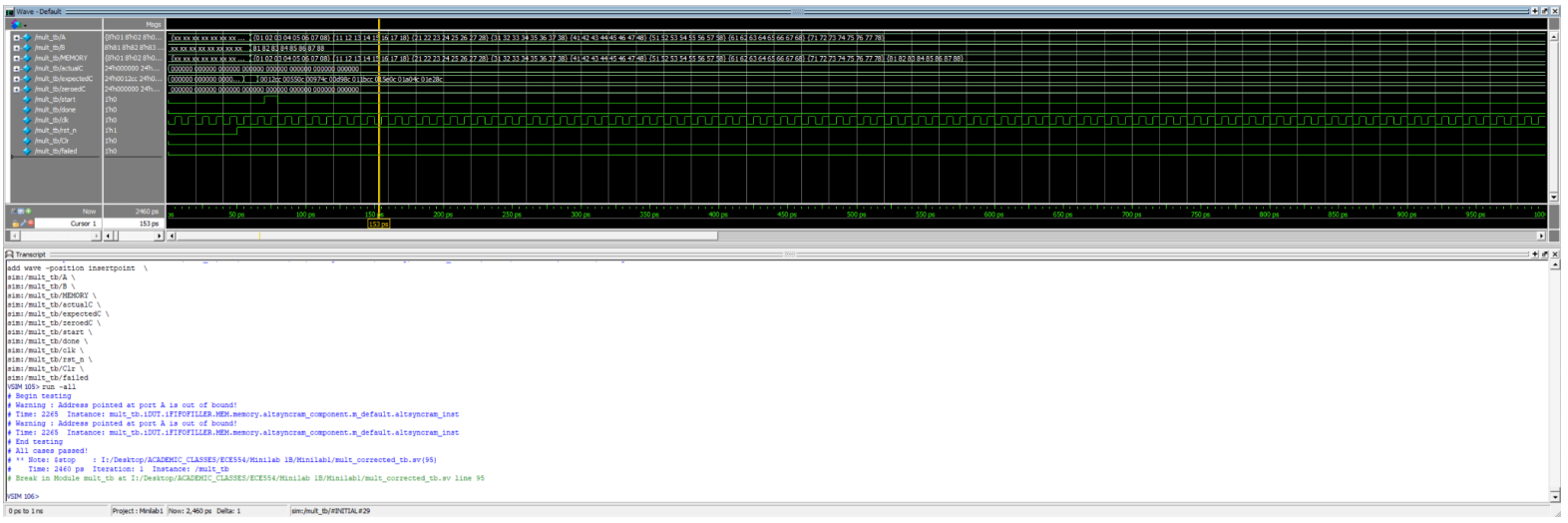
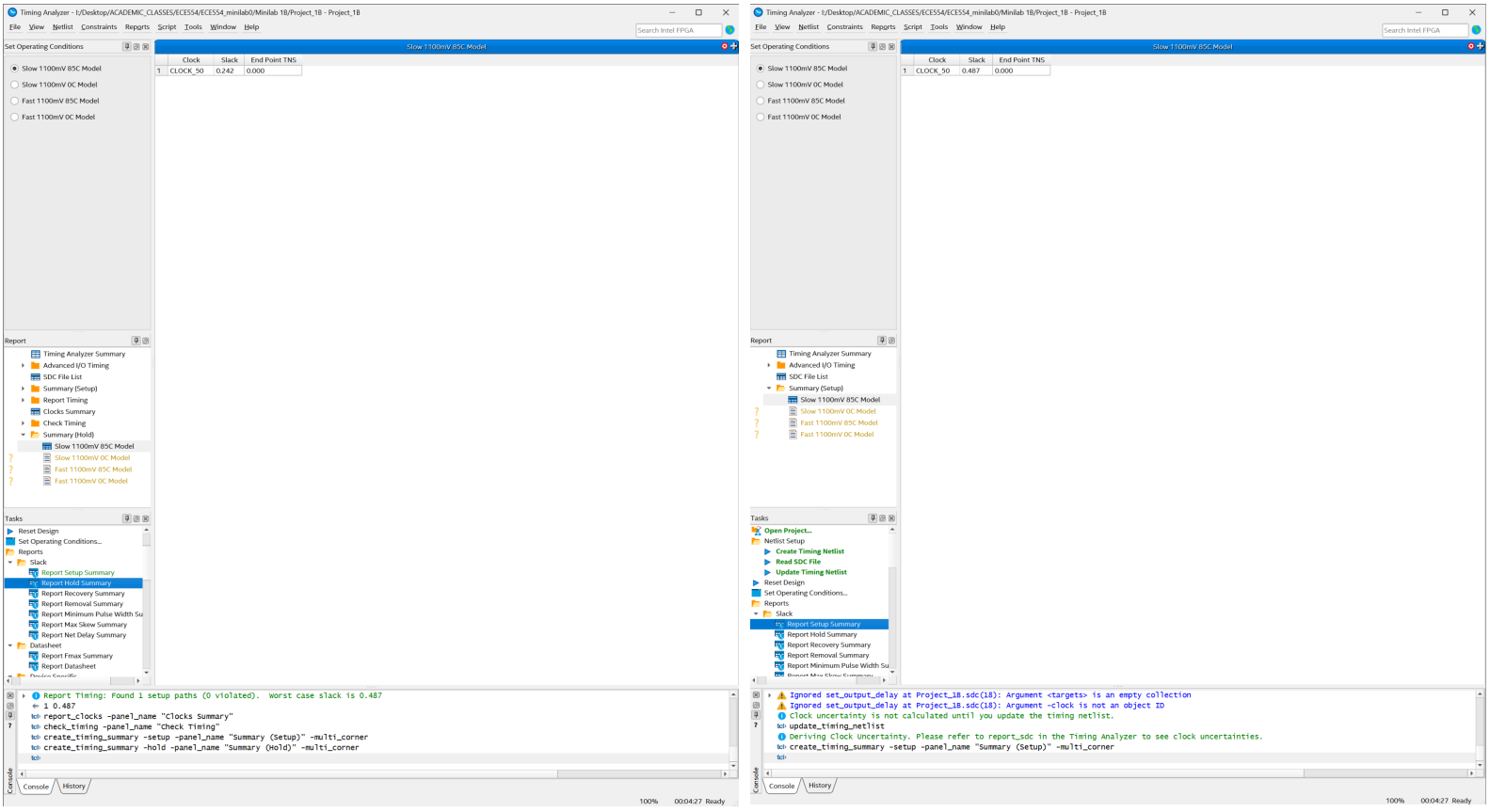


Figure: mult testbench passing

All figures are provided as png images in the zip file if the provided picture(s) are too small.

Timing Constraints

We first ran timing and met the hold time requirement immediately. However, the slack requirement wasn't met as our worst-case path was 1.1 ns too slow. The critical path ended up being the MAC accumulator according to the Timing Analyzer's timing report. To fix this, we changed the multiplier to use the Quartus IP's version instead of the default multiplier that Verilog instantiates. This saved about .1ns worth of time. We tried pipelining the result so that there would be an additional clock cycle to obtain the result, but it didn't improve clock timing at all. We then split the multiplier and accumulator between both cycles to save time in the initial cycle, which brought our slack time down to -0.004 ns. To fix this, we changed the clear and enable signals to be on the second clock phase instead of the first one, which solved the slack time issue. The hold time was still met even with all of these changes.



Figures: Left shows hold time slack, right shows setup time slack

All figures are provided as png images in the zip file if the provided picture(s) are too small.

Design Tested on Board

To translate the design to the FPGA, we had to map the signals correctly and incorporate switches into our design. This included using available clocks, rst_n, and start buttons instead of using our own internal signal. We also added state and data information to be displayed on the 7-LCD display while the program was running instead of being internally tracked. Finally, we compiled the design and allowed the placement of routing to occur.

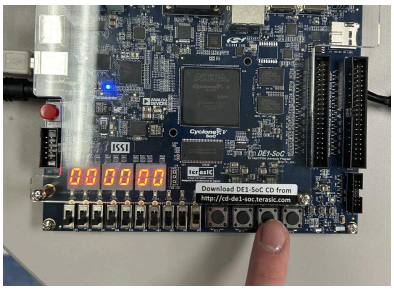
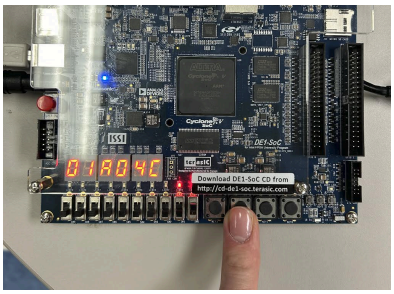
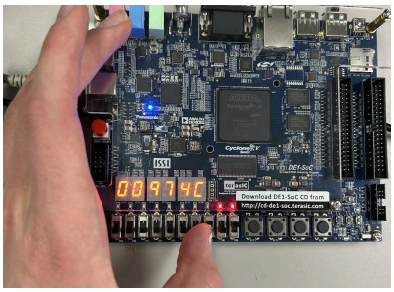
		

Figure: A reset/cleared board	Figure: Start has been asserted (key[2])	Figure: Done State

All figures are provided as png images in the zip file if the provided picture(s) are too small.

Testing Through Signal Tap

To test Signal Tap, we provided the clock and used memRead as our trigger for the “logic analyzer”. We grabbed some FIFO data, the current address we were reading from, and if the memory was reported as “busy”, or if the hardware was in the process of fetching data. The signals we were able to see all seemed to work correctly based on the provided diagrams.

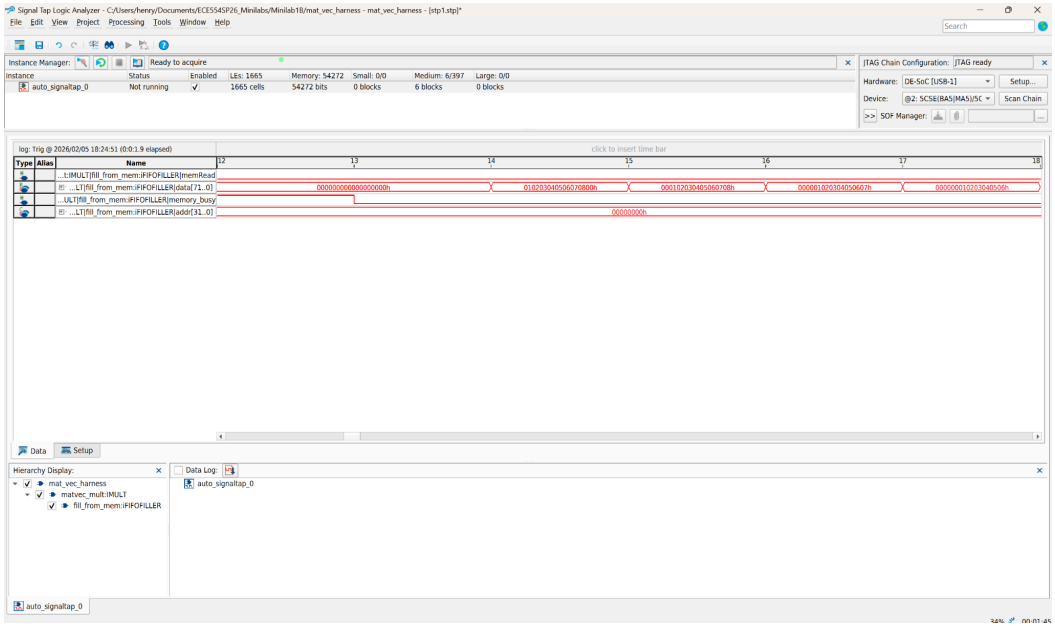


Figure: Signal tap waveforms

All figures are provided as png images in the zip file if the provided picture(s) are too small.

Difficulties During the Process

The two main problems we had were figuring out how to get the FIFO/MAC accumulate logic to work and how to get the final design within setup time constraints. The main logic took some time to understand and lots of discussions and troubleshooting before we were able to figure out how we could get the module to work. We used our testbenches as much as we could, but eventually we were confused as to what the problem may be. However, we realized our checks assumed the numbers were in decimal when in fact they were in hex. This error meant the interface we had was actually working correctly, so we were able to move on.

Our setup time was too slow with a 200 MHz clock by around -1.1ns. The critical path reported was in the MAC, specifically during the clock cycle where the multiply and accumulate were taking place. To remedy this, we first tried using a multiplier IP, but this ended up only reducing slack down to -1ns. We then tried pipelining the stage to allow more time but this simply moved the critical path to before the pipeline, thereby not doing anything. We then tried splitting the multiply and accumulate across multiple clock cycles, thus having fewer operations in one clock cycle. This brought slack to -0.004ns. Finally, we moved the clear and enable signals away from the critical path to save time and flop logic, which did the trick and allowed us to pass both setup and hold time constraints.