Eugene Ngo
EE 469
4/7/2023
Lab 1 Report

Procedure:

This lab contained 3 tasks. Task 1 was to refresh my digital design memory, task 2 was to design and simulate a register file, and task 3 was to design and simulate an ALU. Once the designs were implemented, they were simulated and tested in Modelsim to demonstrate their functionality.

Task #2:

The second task was to implement a register file, the rapid memory unit on a CPU. This was done by first utilizing the register file given in the lab pdf.

```
module reg_file_ex(input  logic     clk, wr_en,
                   input  logic [31:0] write_data,
                   input  logic [3:0]  write_addr,
                    input  logic [3:0]  read_addr,
                    output logic [31:0] read_data);

     logic [15:0][31:0] memory;

     always_ff @(posedge clk) begin
          if (wr_en) begin
               memory[write_addr] <= write_data;
          end

          read_data <= memory[read_addr];
     end
endmodule
```

Figure 2: The base-level register file provided by the lab pdf

This register file implementation was for a 16x32 register file that was synchronous and had one read port and one write port. The task specified a register file to be made that was also 16x32 with one write port, but with two read ports and asynchronous. To achieve this, the given register file had one more read signal added and the always_ff block was changed to always_comb for reading from the file, but the always_ff block was retained for writing to the file since the specification demanded thus.

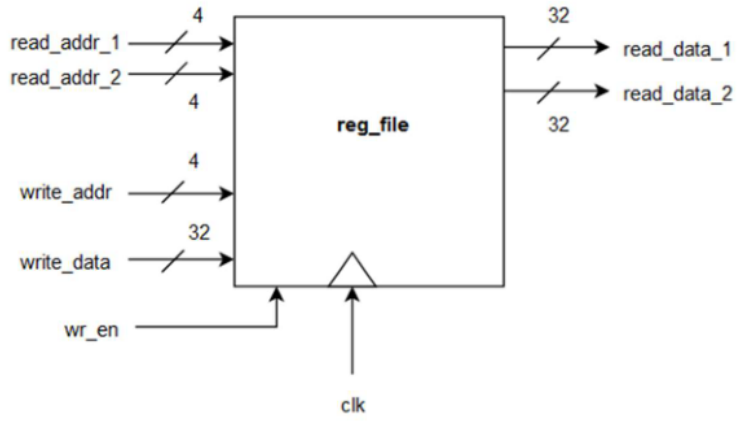

Figure 3: This is a block diagram for the overall register file taken from the lab specification

Task #3:

The third task was to implement an ALU unit from a CPU. This was done by first taking the specifications required by the lab and mapping them out to create a block diagram.

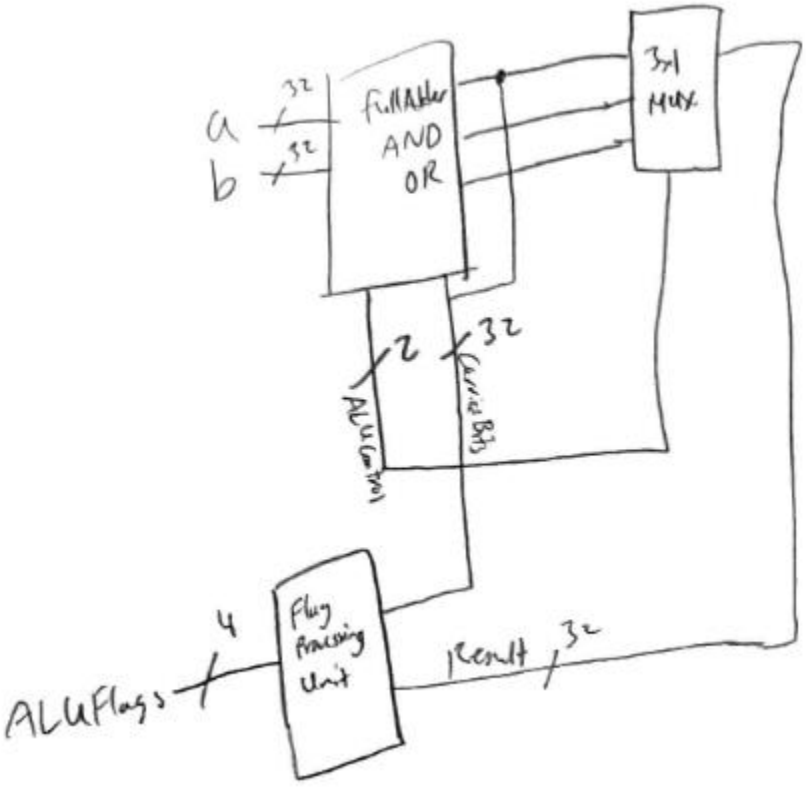

Figure 4: The overall block diagram for the ALU

A and B are fed into fulladders, and and or modules bit-by-bit. The outputs from each are then passed through to a 3x1 MUX and processed to output result. Outputs from the fullAdder are used to determine the carrier inputs in a feedback loop. The ALUControl signal is used as a mux

control signal and then the Result combined with the carrier inputs is used to determine the ALUFlags signal.

Following the block diagram, the circuit was implemented in Quartus using System Verilog. It was then tested in Modelsim by varying a ,b, ALUControl, to see the change in Result and ALUFlags. Then a, b and ALUControl were varied using testvectors, see attached alu.tv screenshot in Appendix for details.

Test	ALUControl[1:0]	A	B	Y	ALUFlags
ADD 0+0	0	00000000	00000000	00000000	4
ADD 0+(-1)	0	00000000	FFFFFFFF	FFFFFFFF	8
ADD 1+(-1)	0	00000001	FFFFFFFF	00000000	6
ADD FF+1	0	000000FF	00000001		0
SUB 0-0	1	00000000	00000000	00000000	6
SUB 0-(-1)	1	00000000	FFFFFFFF	00000001	0
SUB 1-1	1	00000001	0000001	00000000	6
SUB 100-1	1	00000100	00000001	000000FF	2
AND FFFFFFFF, FFFFFFFF	2	FFFFFFFF	FFFFFFFF	FFFFFFFF	10
AND FFFFFFFF, 12345678	2	FFFFFFFF	12345678	12345678	2
AND 12345678, 87654321	2	12345678	87654321	02244220	0
AND 00000000, FFFFFFFF	2	00000000	FFFFFFFF	00000000	4
OR FFFFFFFF, FFFFFFFF	3	FFFFFFFF	FFFFFFFF	FFFFFFFF	10
OR 12345678, 87654321	3	12345678	87654321	97755779	9
OR 00000000, FFFFFFFF	3	00000000	FFFFFFFF	FFFFFFFF	8
OR 00000000, 00000000	3	00000000	00000000	00000000	6

Figure 5: The values that were varied for the test vector used and the resulting values

I used this table to verify that all four ALU operations work as intended and filled in the missing values.

Results

Task #2: Register File

After implementing the register file in Quartus, I ran ModelSim to test it.

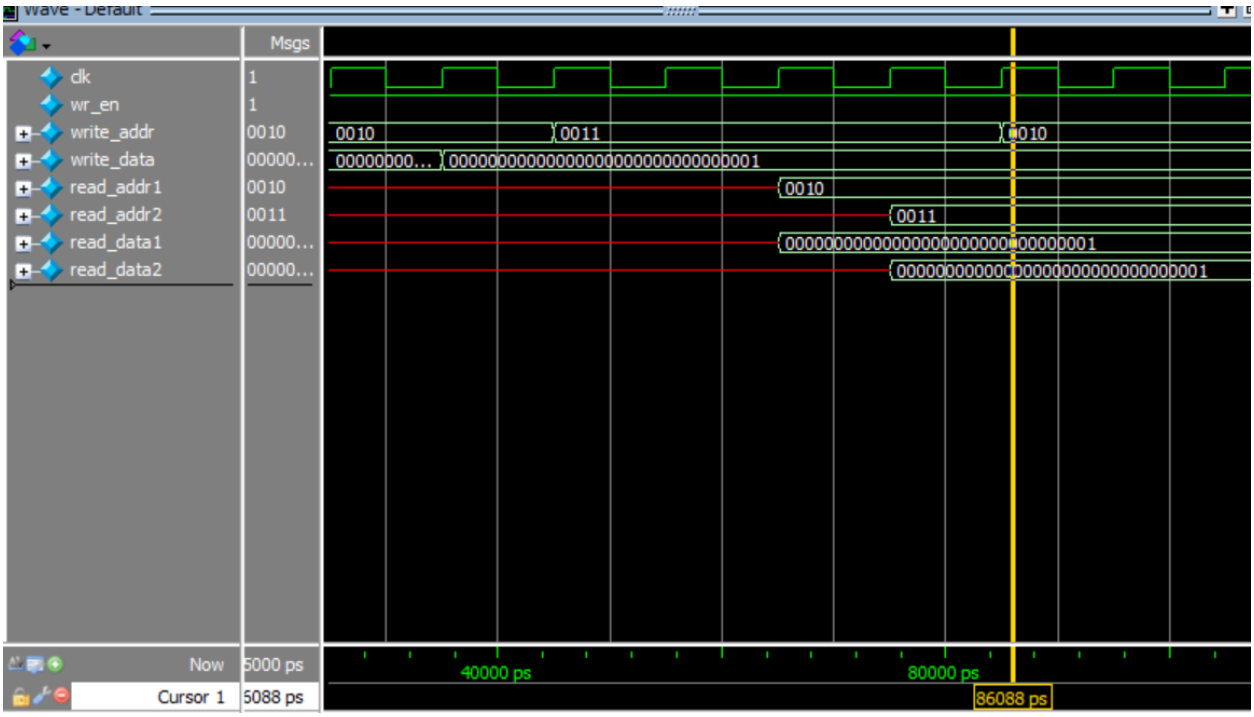

Figure 6: ModelSim waves for testing writes delays (1 cycle)

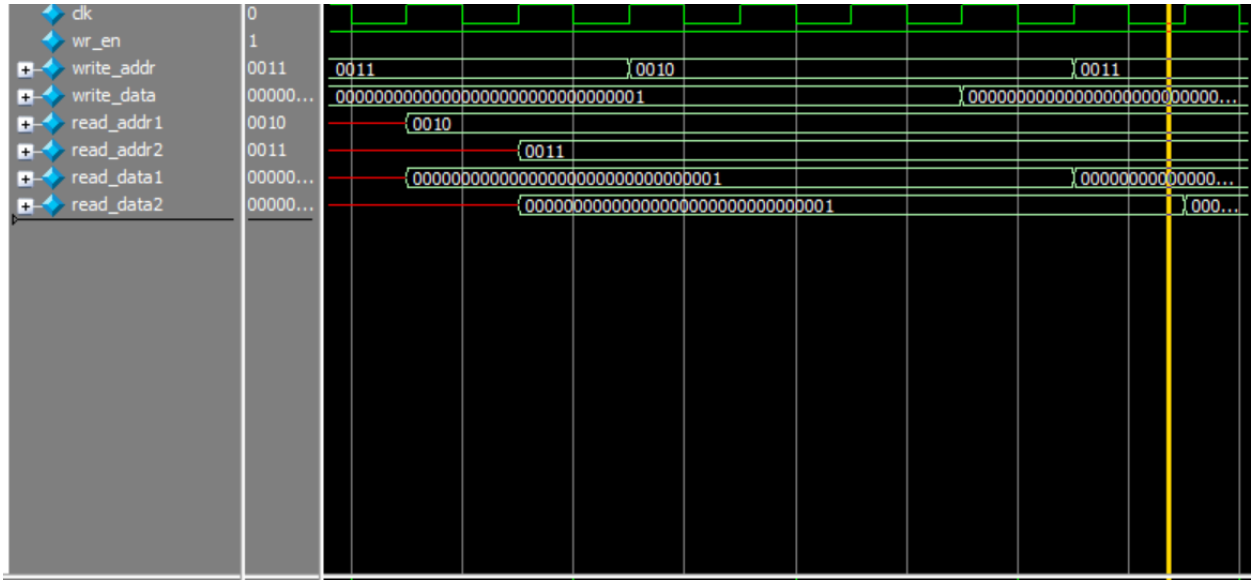

Figure 7: ModelSim waves for testing read delays (0 cycles)

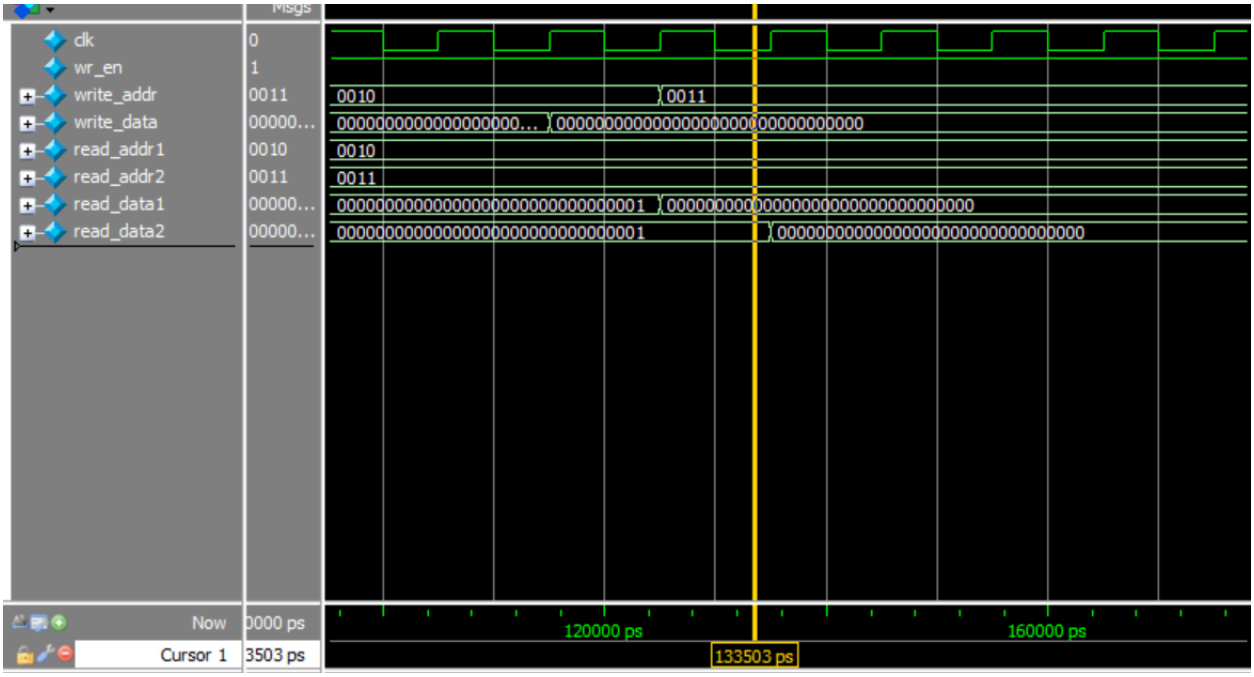

Figure 8: ModelSim waves for testing read update delays after a write (1 cycle after the write)

The three functionalities that are being tested are given by the lab specification:

1. Write data is written into the register file the clock cycle **after** wr_en is asserted.
2. Read data is updated to the register data at an address the **same** cycle the address was provided. Do this for both read addresses and data outputs.
3. Read data is updated to write data at an address the cycle **after** the address was provided if the write address is the same and wr_en asserted. Do this for both read addresses and data outputs.

As seen in the waveforms above, data is written only 1 cycle after ‘wr_en’ is equal to 1. Read data pushes through in the very same cycle since it is done using combinational logic. Read data is updated 1 cycle after if a write is performed at the same address as the read.

Task #3: ALU

After implementing the ALU in Quartus, I ran ModelSim to test it.

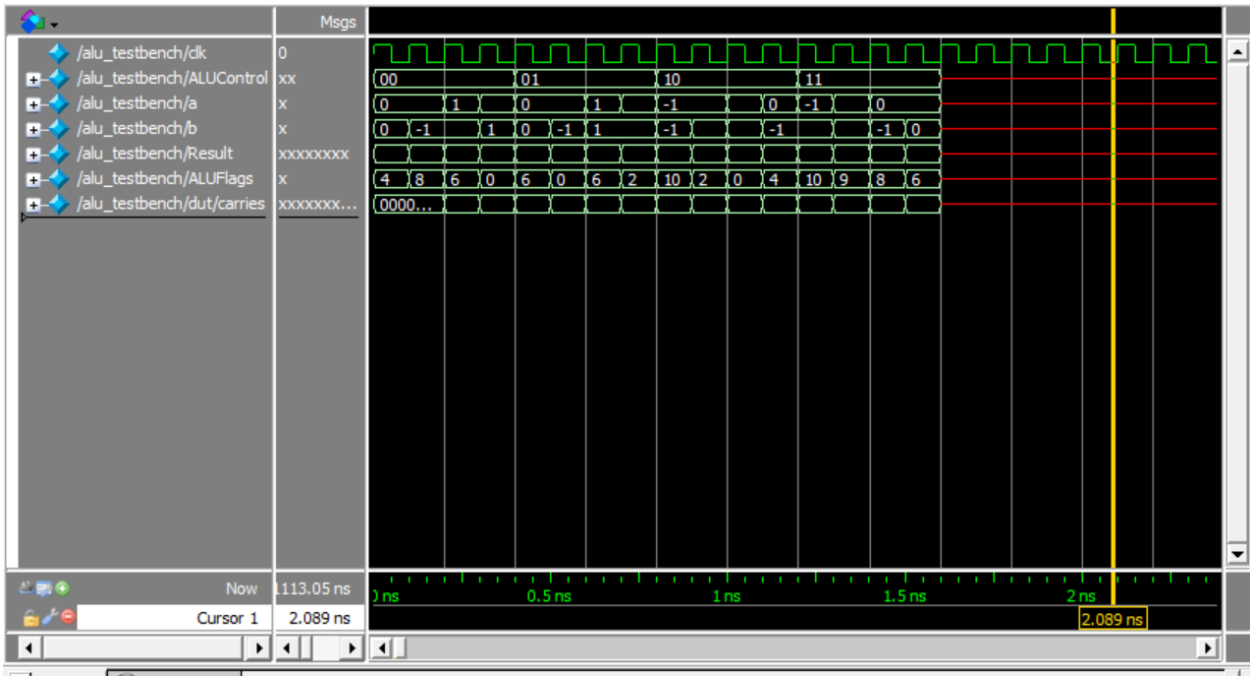

Figure 9: Waveform generated by running a test vector through the ALU

As seen from the figure, Result is the output of performing the function defined by ALUControl and the ALUFlags are updated based on the carriers within the ALU and the Result value.

Appendix

See the following list for the order:

Task 2: reg_file.sv

reg_file_testbench.sv

Task 3: alu.sv

alu_testbench.sv

singleALU.sv

fullAdder.sv

alu.tv

See the attached documents for the code: