Laboratory 04

CPE 166 Advanced Logic Design

Section 03

Vigomar Kim Algador

Fall 2022

TABLE OF CONTENT

INTRODUCTION

This laboratory is an extensive topic using Verilog and the other designs that were introduced previously in the past laboratories such as FSM and hierarchical design. This laboratory is divided into 2 main topics. The first part of the laboratory introduces the SRAM which stands for Static Random Access Memory. This part of the laboratory will be designing an FSM, RAM, and the final top design combining the two. The second part of the laboratory is writing a simplified microprocessor design. This part of the laboratory is divided into 2 sub parts: Microprocessor Data Path Design and Microprocessor Design.

PART 1: SRAM DESIGN

Design Purpose and Engineering Data

This part of the laboratory introduced to student about the concept of SRAM designing. The student is required to write a 4'b1010 into the 32 address locations of the SRAM and read data from RAM. The SRAM is based on the following function table given by the laboratory manual shown below.

WE	CS	OE	Function
X	L	X	High – z
L	H	L	High – z
L	H	H	Read Data
H	H	X	Write Data

RAM Design

At first, the student design the RAM for this project based on what information learned from the lecture. Using the instructions given, the student was able to format the design by changing the address and data.

```
1  module ram ( address, data, cs, we, oe);
2  input cs, we, oe;
3  input [4:0] address;
4  inout [3:0] data;
5
6  reg [3:0] data_out ;
7  reg [3:0] mem [0:31];      //32-bit address bus, 8-bit data bus.
8
9  assign data = (cs && oe && !we) ? data_out : 4'bzzzz;
10
11 always @ (cs or we or data or address)
12 begin
13     if (cs && we)
14         mem[address] = data;        // write
15 end
16
17 always @ (cs or we or oe or address or data)
18 begin
19     if (cs && !we && oe)
20         data_out = mem[address];    // read
21 end
22 endmodule
```

Figure 1. RAM Verilog design (ram.v)

FSM Design

For the second design, the student is required to design an FSM which is shown below.

```
module mem_fsm ( clk, reset, address, data, cs, we, oe);
input        clk, reset;
output [4:0] address;
inout  [3:0] data;
output       cs, we, oe;

reg          cs, we, oe;
reg    [4:0] address;
reg    [3:0] data_reg;
reg    [2:0] state;
parameter idle = 3'b000, s1=3'b001, s2=3'b010, s3=3'b011, s4=3'b100;
assign data = data_reg;

always@(posedge clk or posedge reset)
begin
    if (reset) begin
        state   <= idle;
        address <= 0;
    end
    else
        case (state)
            idle: begin
                state   <= s1;
                address <= 0;
            end
            s1: begin
                state   <= s2;
                address <= 0;
            end
            s2: begin
                address <= address + 1;
                if(address == 31)
                    state <= s3;
                else
                    state <= s2;
            end
            s3: begin
                state <= s4;
                address <= 0;
            end
            s4: begin
                address <= address + 1;
                if(address == 31)
                    state <= idle;
                else
                    state <= s4;
            end
            default: begin
                state <= idle;
                address   <= 0;
            end
        endcase
    end

always@(state)
begin
    case (state)
        idle: begin
            cs = 0; we = 0; oe = 0;
            data_reg = 4'bzzzz;
        end
        s1: begin   //writing
            cs = 1; we = 1; oe = 0;
            data_reg = 4'b1010;
        end
        s2: begin   //writing
            cs = 1; we = 1; oe = 0;
            data_reg = 4'b1010;
        end
        s3: begin   //reading
            cs = 1; we = 0; oe = 1;
            data_reg = 4'b1010;
        end
        s4: begin   //reading
            cs = 1; we = 0; oe = 1;
            data_reg = 4'b1010;
        end
        default: begin
            cs = 0; we = 0; oe = 0;
            data_reg = 4'bzzzz;
        end
    endcase
end
endmodule
```

Figure 2. FSM Verilog Design (mem_fsm.v)

Final SRAM design

Lastly, the student needs to combine the files by creating a top design.

```
module top(clk, reset, address, data, cs, we, oe);
input clk, reset;
output [4:0] address;
output [3:0] data;
output cs, we, oe;

wire [4:0] address_wire;
wire[3:0] data_wire;
wire cs_wire, we_wire, oe_wire;

assign address = address_wire;
assign data = data_wire;
assign cs = cs_wire;
assign we = we_wire;
assign oe = oe_wire;

mem_fsm FSM(.clk(clk), .reset(reset), .address(address_wire), .data(data_wire), .cs(cs_wire), .we(we_wire), .oe(oe_wire));
ram RAM(.address(address_wire), .data(data_wire), .cs(cs_wire), .we(we_wire), .oe(oe_wire));
endmodule
```

```
module top_tb;
reg clk, reset;
wire[4:0] address; //2^5 = 32
wire[3:0] data;   //4 bits
wire cs, we, oe;

top uut(clk, reset, address, data, cs, we, oe);

always begin
    #5 clk = ~clk;
end

initial begin
    clk=0;
    #10 reset = 1;
    #10 reset = 0;
    #100 $stop;
end
endmodule
```

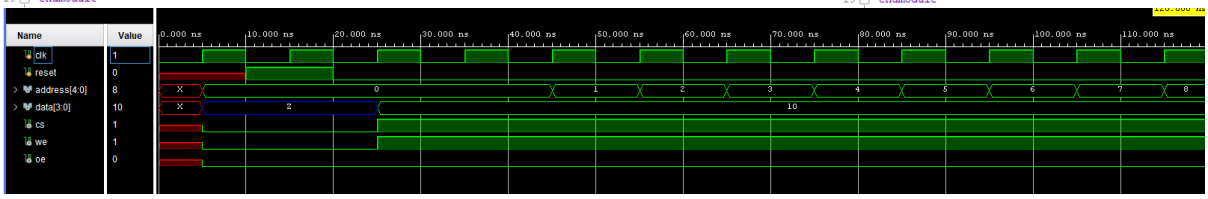

Figure 3. Final TOP Verilog design (top.v), testbench (top_tb.v), and simulation

Result Discussion

For this part of this laboratory, I was able to design the whole SRAM project with 3 files: RAM, FSM, and TOP. The only challenging part of this design is the FSM for this project, however, the guidelines and information taught in the class are really helpful to design the overall project.

PART 2: SIMPLIFIED MICROPROCESSOR DESIGN (1)

Design Purpose and Engineering Data

For this part of the laboratory, I will be designing the first component which is the microprocessor data path design of the simplified microprocessor block diagram shown below.

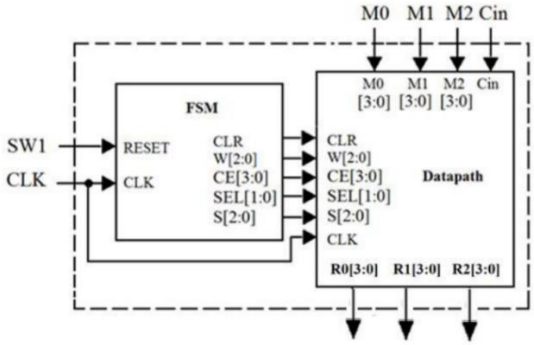

Figure 4. Simplified Microprocessor Block Diagram

Within the Datapath, below is the full design shown below.

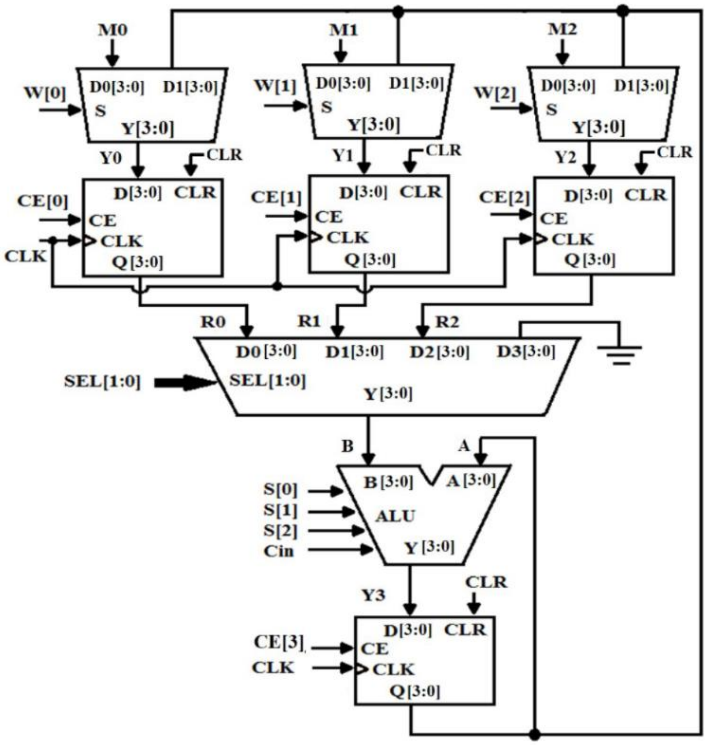

Figure 5. Simplified Microprocessor Datapath Circuit

With the given information, this part 2 requires using structural hierarchical design method to implement the data path circuit.

Multiplexer Design

For the first design, I was able to write the multiplexer using the knowledge I have from previous laboratories. There will be two different multiplexers for this project. The first multiplexer will be the mux2 which is the top part of the data path circuit shown below.

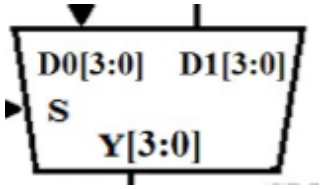

Figure 6. multiplexer for mux2 block

With this, I was able to write the Verilog and testbench for it shown below.

```
1  // Vigomar Kim Algador
2  // CpE166 Section 03
3  // Laboratory 04: Part 02-03
4  // 4-bit multiplexer (MUX2)
5
6  module mux2_tb;
7  reg [3:0] D0, D1;
8  reg S;
9  wire [3:0] Y;
10 integer k;
11
12 mux2 u1(D0, D1, S, Y);
13
14 initial begin
15     for (k = 0; k < 512; k = k+1)
16         #10 {D0, D1, S} = k;
17     #10 $stop;
18 end
19 endmodule
20
```

```
1  // Vigomar Kim Algador
2  // CpE166 Section 03
3  // Laboratory 04: Part 02-03
4  // 4-bit multiplexer (MUX2)
5
6  module mux2(D0, D1, S, Y);
7      input [3:0] D0, D1;
8      input S;
9      output [3:0] Y;
10     reg [3:0] Y;
11
12     always @ (D0 or D1 or S)
13     begin
14         if (S == 1)
15             Y = D1;
16         else
17             Y = D0;
18     end
19 endmodule
20
```

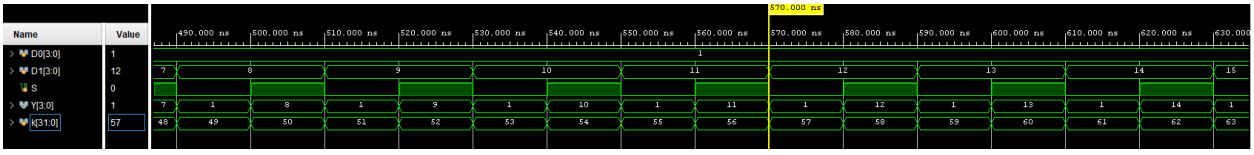

Figure 7. mux2 Verilog, testbench, and simulation

For another multiplexer design, below is the mux4 block diagram that I need to follow and write in Verilog. Using the same design from mux2, I was able to modify changes and write the Verilog as well as testbench shown in figure 9.

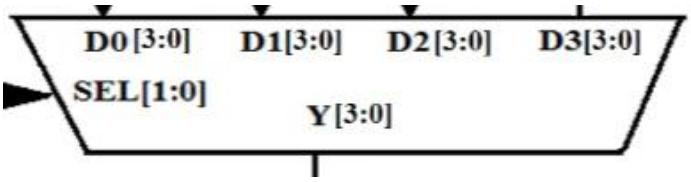

Figure 8. multiplexer mux4 block

```verilog
// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 04: Part 02-03
// 4-bit multiplexer (MUX4)

module mux4_tb;
reg [3:0] D0, D1, D2, D3;
reg [1:0] SEL;
wire [3:0] Y;
integer k;

mux4 u2(D0, D1, D2, D3, SEL, Y);

initial begin
    for (k = 0; k < 262144; k = k+1)
        #10 {D0, D1, D2, D3, SEL} = k;
    #10 $stop;
end
endmodule
```

```verilog
// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 04: Part 02-03
// 4-bit multiplexer (MUX4)

module mux4(D0, D1, D2, D3, SEL, Y);
    input [3:0] D0, D1, D2, D3;
    input [1:0] SEL;
    output [3:0] Y;
    reg [3:0] Y;

    always @ (D0 or D1 or D2 or D3 or SEL)
    begin
        case(SEL)
            2'b00: Y = D0;
            2'b01: Y = D1;
            2'b10: Y = D2;
            2'b11: Y = D3;
        endcase
    end
endmodule
```

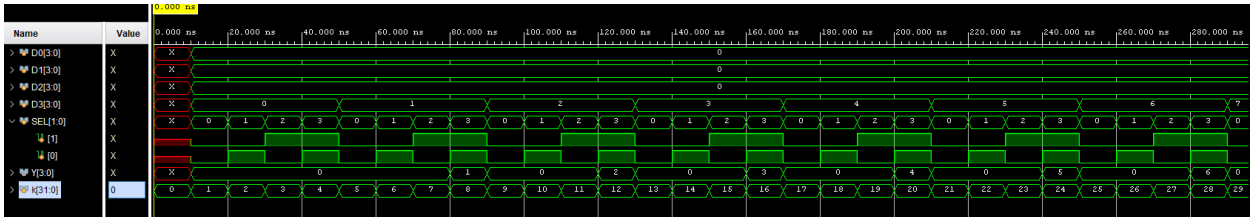

Figure 8. mux4 Verilog, testbench, and simulation

D Flip-Flop Design

Next, I was able to check the block for the d flipflop design shown below. With this, I was able to write the Verilog and testbench shown in figure 10.

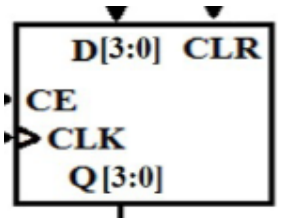

Figure 9. D Flip-Flop block

```
// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 04: Part 02-03
// D-Flip Flop (dff.v)

module dff(D, CLR, CE, CLK, Q);
    input [3:0] D;
    input CLR, CE, CLK;
    output reg [3:0] Q;

    always@(posedge CLK or posedge CLR)
    begin
        if (CLR)
            Q <= 0;
        else if (CE)
            Q <= D;
    end
endmodule
```

```
// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 04: Part 02-03
// D-Flip Flop (dff.v)


module dff_tb;
    reg [3:0] D;
    reg CLR, CE, CLK;
    wire [3:0] Q;

    dff u1(D, CLR, CE, CLK, Q);

    initial CLK = 0;
    always #10 CLK = ~CLK;

    initial begin
    CLR = 1; D = 1; CE = 1;
    #10 CLR = 0;
    #10 CE = 0;
    #50 $stop;
    end
endmodule
```

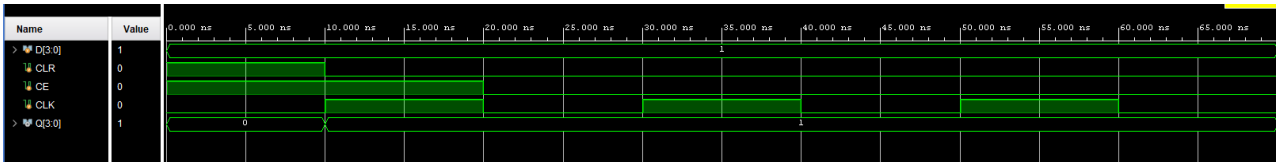

Figure 10. D flipflop Verilog, testbench, and simulation

ALU Design

For the next design, I need to design the ALU diagram shown below.

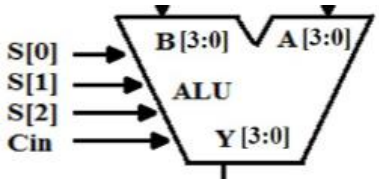

Figure 11. ALU block

There is a truth table for the ALU diagram inside the data path circuit shown below.

S[2]	S[1]	S[0]	ALU Output F
0	0	0	F = A + B + Cin
0	0	1	F = A + B’ + Cin
0	1	0	F = B
0	1	1	F = A
1	0	0	F = A AND B
1	0	1	F = A OR B
1	1	0	F = A’
1	1	1	F = A XOR B

```
1  // Vigomar Kim Algador
2  // CpE166 Section 03
3  // Laboratory 04: Part 02-03
4  // ALU
5
6  module ALU(A, B, S, Cin, Y);
7      input [3:0] A, B;
8      input [2:0] S;
9      input Cin;
10     output [3:0] Y;
11     reg [3:0] Y;
12
13     always@(S) begin
14     case(S)
15         0: Y = A + B + Cin;
16         1: Y = A + (~B) + Cin;
17         2: Y = B;
18         3: Y = A;
19         4: Y = A&B;
20         5: Y = A|B;
21         6: Y = ~A;
22         7: Y = A^B;
23     endcase
24     end
25 endmodule
```

```
1  // Vigomar Kim Algador
2  // CpE166 Section 03
3  // Laboratory 04: Part 02-03
4  // ALU
5
6  module ALU_tb;
7      reg [3:0] A, B;
8      reg [2:0] S;
9      reg Cin;
10     wire [3:0] Y;
11     integer i,j;
12
13     ALU uut(B, A, S, Cin, Y);
14
15     initial begin
16     A = 0; B = 0; Cin = 0; S = 0; #10;
17
18     for (i = 0; i < 2; i = i+1) begin
19         for (j = 0; j < 8; j = j+1) begin
20             A = 6; B = 12; Cin = i; S = j; #10;
21         end
22     end
23     #10 $stop;
24     end
25 endmodule
```

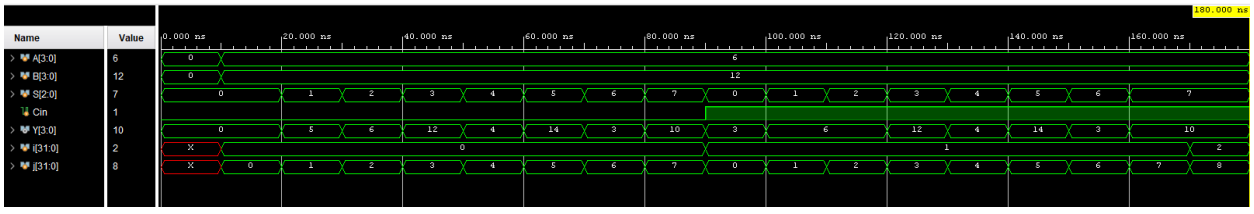

Figure 12. ALU Verilog, testbench, and simulation

Final Microprocessor Data Path Design

Finally, I need to connect all the components I did to create the Datapath that was shown in figure 5. Below is the full Verilog, testbench, as well as the final simulation for the Datapath.

```
// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 04: Part 02-03
// Datapath Verilog


module datapath(M0, M1, M2, Cin, CLR, W, CE, SEL, S, CLK, R0, R1, R2);
    input [3:0] M0, M1, M2, CE;
    input [2:0] W, S;
    input [1:0] SEL;
    input Cin, CLR, CLK;
    output [3:0] R0, R1, R2;

    wire [3:0] Y0, Y1, Y2, Y3, A, B;

    mux2 m1 (.D0(M0), .D1(A), .S(W[0]), .Y(Y0));
    mux2 m2 (.D0(M1), .D1(A), .S(W[1]), .Y(Y1));
    mux2 m3 (.D0(M2), .D1(A), .S(W[2]), .Y(Y2));

    dff d1 (.D(Y0), .CLR(CLR), .CE(CE[0]), .CLK(CLK), .Q(R0));
    dff d2 (.D(Y1), .CLR(CLR), .CE(CE[1]), .CLK(CLK), .Q(R1));
    dff d3 (.D(Y2), .CLR(CLR), .CE(CE[2]), .CLK(CLK), .Q(R2));

    mux4 m4 (.D0(R0), .D1(R1), .D2(R2), .D3(0), .SEL(SEL), .Y(B));

    ALU a1 (.A(A), .B(B), .S(S), .Cin(Cin), .Y(Y3));

    dff d4 (.D(Y3), .CLR(CLR), .CE(CE[3]), .CLK(CLK), .Q(A));

endmodule
```

```
`timescale 1ns / 1ps
// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 04: Part 02-03
// Datapath Testbench


module datapath_tb;
    reg [3:0] M0, M1, M2, CE;
    reg [2:0] W, S;
    reg [1:0] SEL;
    reg Cin, CLR, CLK;
    wire [3:0] R0, R1, R2;

    datapath uut (M0, M1, M2, Cin, CLR, W, CE, SEL, S, CLK, R0, R1, R2);

    initial CLK = 0;
    always #5 CLK = ~CLK;

    initial begin
            M0=4'b0000; M1=4'b0000; M2=0; Cin=0; CLR=1; W=0; CE=0; SEL=3; S=0;
        #15 M0=4'b0110; M1=4'b1100; M2=0; Cin=0; CLR=0; W=0; CE=1; SEL=0; S=2;

        #20 CLR=0; W=3'b000; CE=4'b1000; SEL=2'b00; S=3'b010;
        #20 CLR=0; W=3'b010; CE=4'b0010; SEL=2'b10; S=3'b001;
        #20 CLR=0; W=3'b000; CE=4'b1000; SEL=2'b10; S=3'b001;
        #20 CLR=0; W=3'b100; CE=4'b0100; SEL=2'b10; S=3'b001;

        #100 M0=4'b0000; M1=4'b0000; M2=0; Cin=0; CLR=1; W=0; CE=0; SEL=3; S=0;
        #20 M0=4'b0110; M1=4'b1100; M2=0; Cin=1; CLR=0; W=0; CE=1; SEL=0; S=2;

        #20 CLR=0; W=3'b000; CE=4'b1000; SEL=2'b00; S=3'b010;
        #20 CLR=0; W=3'b010; CE=4'b0010; SEL=2'b10; S=3'b001;
        #20 CLR=0; W=3'b000; CE=4'b1000; SEL=2'b10; S=3'b001;
        #20 CLR=0; W=3'b100; CE=4'b0100; SEL=2'b10; S=3'b001;

        #20 $stop;
    end
endmodule
```

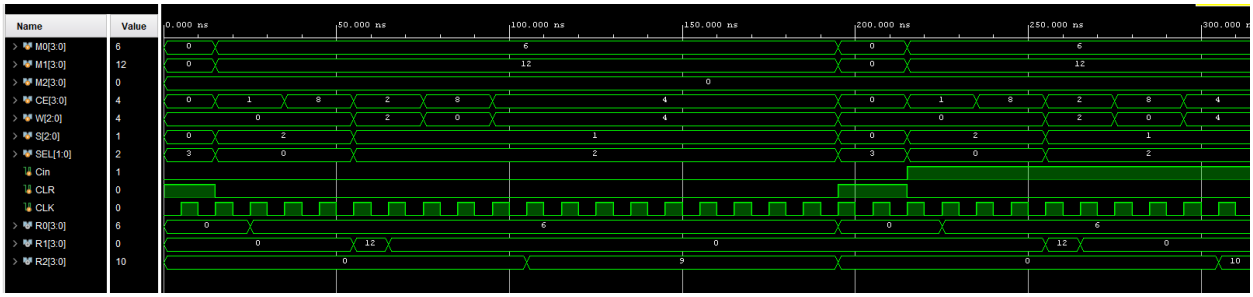

Figure 13. Final Datapath Verilog, testbench, and simulation

Result Discussion

In this part of the laboratory, I was tasked to write the Datapath for the simple microprocessor design. I was able to easily write and design each component of the Datapath circuit in which I learned from the previous laboratories and lecture including multiplexer and D flip-flop. I had encountered a problem when trying to do the whole final Datapath design as the output doesn't match to supposedly correct value from the equation R2 = M0 + (not M1) + Cin. On the other hand, I successfully find the error in my project and was able to generate the write output shown in the figure 13 simulation above.

CONCLUSION

This part of laboratory introduced to more topics using Verilog. This laboratory was divided into 2 parts: SRAM and Microprocessor Design. The first part of the laboratory, I was able to design SRAM project with 3 files: RAM, FSM, and the top combining the RAM and FSM. This part of the laboratory was simple as I was able to use the knowledge I learned from the lecture with the RAM and FSM from the previous laboratory. In the second part of the laboratory, Microprocessor design, I was only able to write the Datapath without the FSM as I ran into issues with Datapath and ran out of time with demonstration. Within the Datapath, I was tasked to use the structural hierarchy to design the circuit. I was able to write the components such as the multiplexer, D flip-flop, and ALU. However, the final step which is combining the whole component got me stuck with the testbench as the output I got doesn't match to the correct output. With this, I checked each component all over again and test it one-by-one to check for any errors. After several times, I was able to fix it and got the correct output. This part of the laboratory was definitely challenging as I am designing an application to the previous topics discussed.