CPE166 Advanced Logic Design

Lab 4 Report:

Simplified Microprocessor Design

Student Name: Andy Cha

Date: 5-10-2020

California State University, Sacramento

TABLE OF CONTENTS

Section **Page**

Title Page …………………………………………………………………........................... 1

Table Of Contents …………………………………………………………………................... 2

Introduction ……………………………………………………………….......................... 3

Part 1: Microprocessor Data Path Design …..……….....…………………………………….. 4

Design Purpose ……………….……………………………………………………….……….. 4

Design Source Codes ………………………………………………………………………… 5-9

Simulation Waveform ……………………………...…………………………………….... 10

Result Discussion ……………………………………………………………………………….. 10

Part 2: Microprocessor Control Path Design ………………………………………………… 11

Design Purpose ………………………………………………………………………………. 11

Design Source Codes …………………………………………………………………...… 12-14

Simulation Waveform ……………………………...…………………………………….... 15

Result Discussion ……………………………………………………………………………….. 15

Part 3: Final Simplified Microprocessor Design …………………………………………….... 16

Design Purpose ………………………………………………………………………………. 16

Design Source Codes …………………………………………………………………...… 17

Simulation Waveform ……………………………...…………………………………….... 18

Result Discussion ……………………………………………………………………………….. 18

Conclusion ……………………………………………………………………………………. 19

Introduction

A microprocessor is a computer processor that incorporates the functions of a central processing unit on a single integrated circuit (IC). Microprocessors contain both combinational logic and sequential digital logic. For this lab we needed to build a simplified microprocessor in which it would take some inputs and would output a value. The design needed to implement the following logic equation: R2 = M0 + (not M1) + Cin. The lab will be divided into three parts to reduce the amount of errors at the end. The first part of this lab will be to create the datapath. The datapath contains the ALU, mux2to1, mux4to1, and D-flip-flop. For the second part, we will create the FSM to determine the internal inputs for datapath. The third part of this lab will be to create a top file where we combine the datapath and FSM with three added inputs: m0, m1, and m2.

Part 1: Microprocessor Data Path Design

Design Purpose:

This part of the lab will be to create the datapath circuit as shown in the lab manual. It consists of 3 mux2to1 circuits, 4 D-flip-flops, 1 mux4to1, and the ALU. The ALU will compute the functions as shown in the lab manual depending on the inputs ‘a’ and ‘b’. To test the datapath, we will create a testbench file giving values to m0, m1, m2 , ce, w, s, sel, and cin. Each of these internal signals will determine how data will be passed through each of these circuits and that is how we will be able to track if our output is correct.

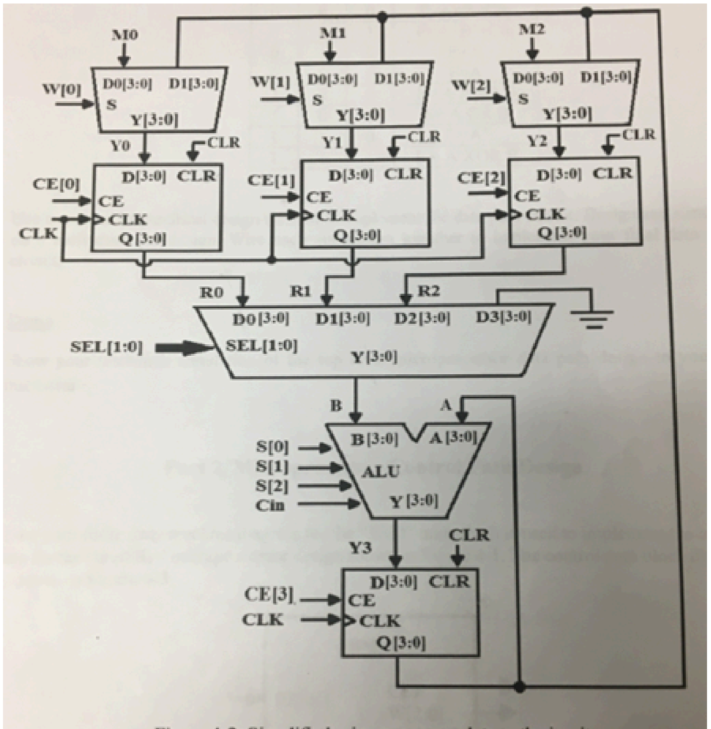
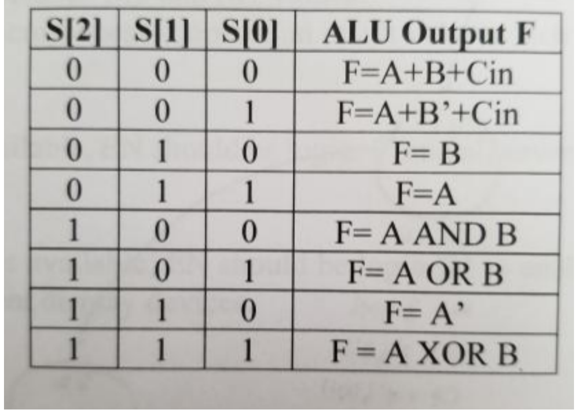


Figure 4-2. Simplified microprocessor data path circuit

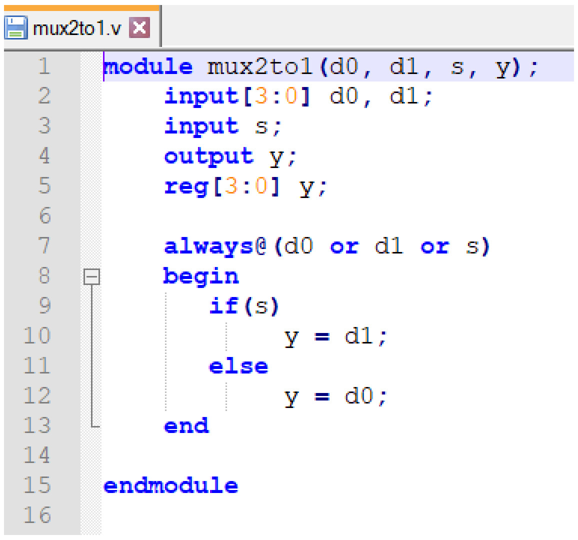
Table 4-1. ALU truth table



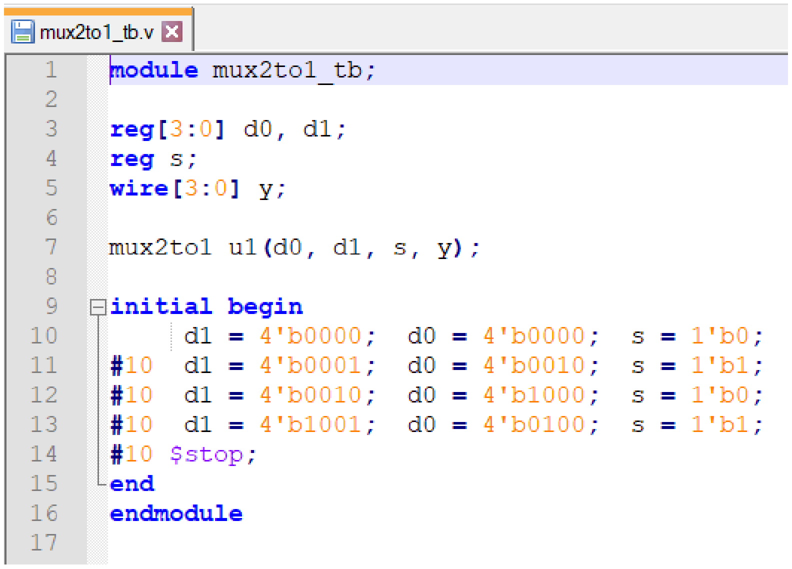
S[2]	S[1]	S[0]	ALU Output F
0	0	0	F=A+B+Cin
0	0	1	F=A+B’+Cin
0	1	0	F= B
0	1	1	F=A
1	0	0	F= A AND B
1	0	1	F= A OR B
1	1	0	F= A’
1	1	1	F = A XOR B

Design Source Code:

Multiplexer 2-to-1:

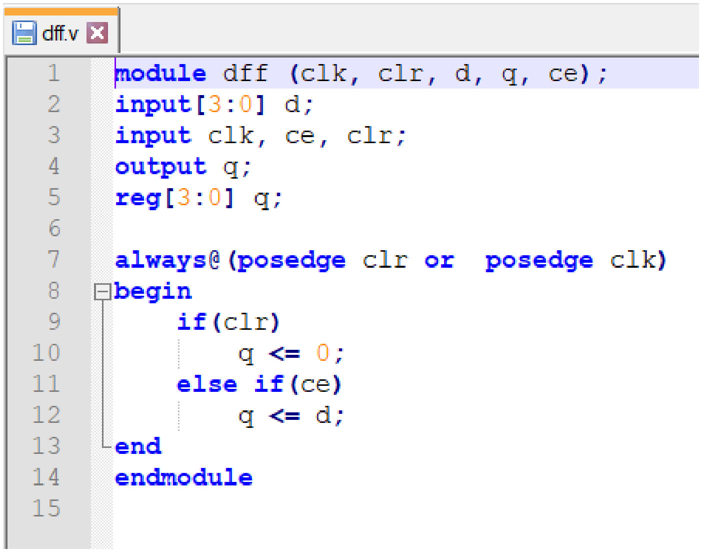

mux2to1.v

```
module mux2to1(d0, d1, s, y);
    input[3:0] d0, d1;
    input s;
    output y;
    reg[3:0] y;

    always@(d0 or d1 or s)
    begin
       if(s)
            y = d1;
       else
            y = d0;
    end

endmodule
```

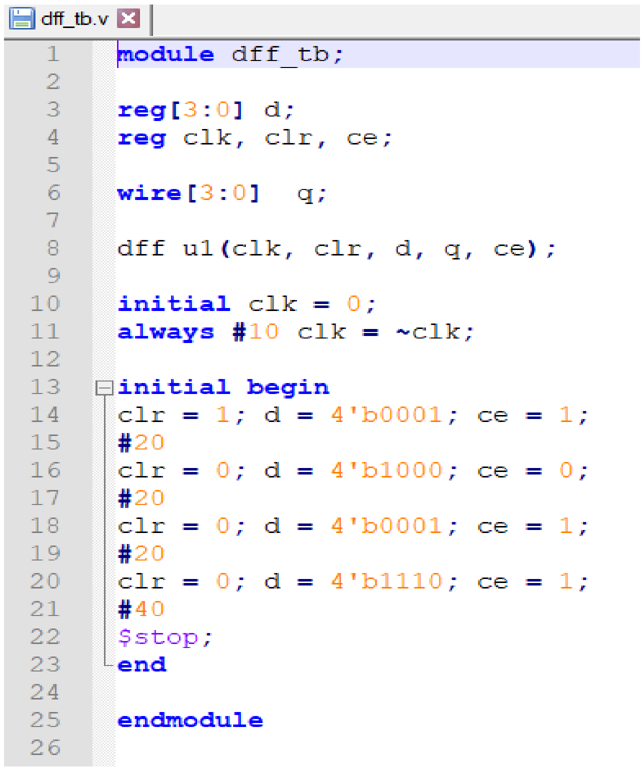

mux2to1_tb.v

```
module mux2to1_tb;

reg[3:0] d0, d1;
reg s;
wire[3:0] y;

mux2to1 u1(d0, d1, s, y);

initial begin
     d1 = 4'b0000;  d0 = 4'b0000;  s = 1'b0;
#10  d1 = 4'b0001;  d0 = 4'b0010;  s = 1'b1;
#10  d1 = 4'b0010;  d0 = 4'b1000;  s = 1'b0;
#10  d1 = 4'b1001;  d0 = 4'b0100;  s = 1'b1;
#10 $stop;
end
endmodule
```

D flip-flop:

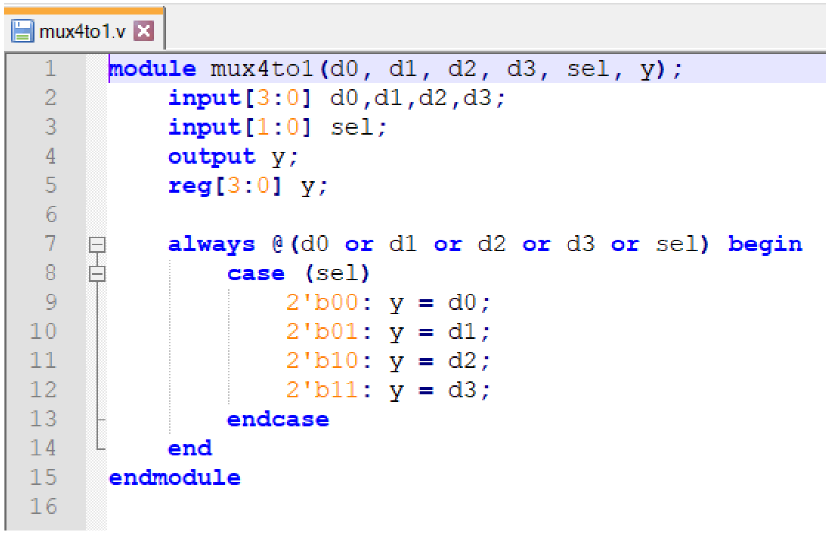


```verilog
module dff (clk, clr, d, q, ce);
input[3:0] d;
input clk, ce, clr;
output q;
reg[3:0] q;

always@(posedge clr or  posedge clk)
begin
    if(clr)
        q <= 0;
    else if(ce)
        q <= d;
end
endmodule
```



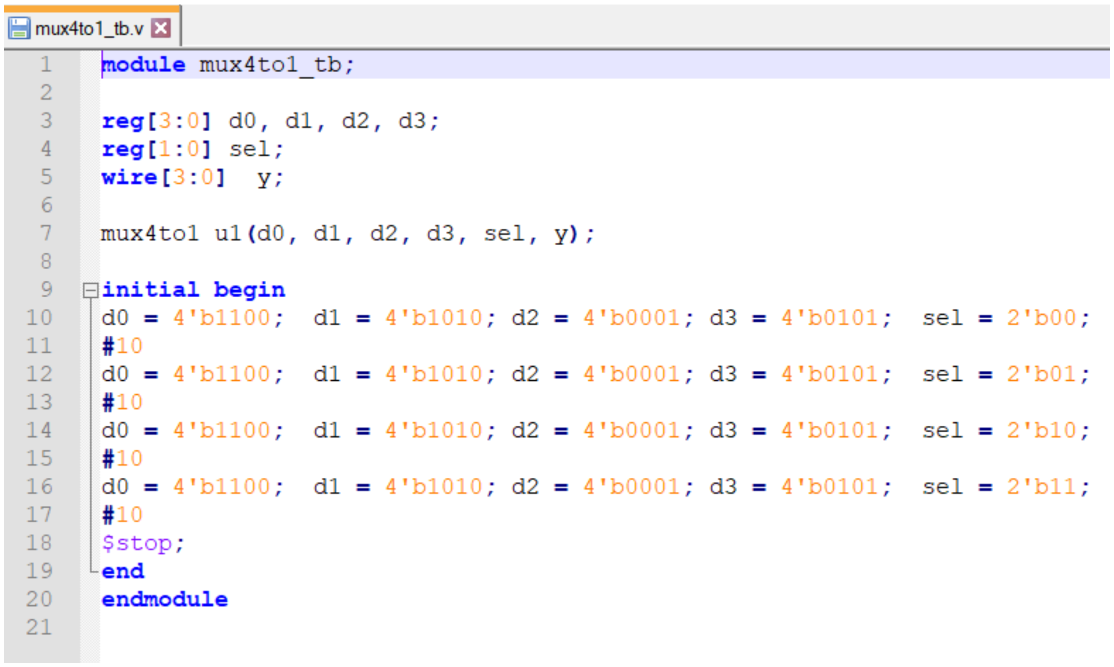
```verilog
module dff_tb;

reg[3:0] d;
reg clk, clr, ce;

wire[3:0]  q;

dff u1(clk, clr, d, q, ce);

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

initial begin
clr = 1; d = 4'b0001; ce = 1;
#20
clr = 0; d = 4'b1000; ce = 0;
#20
clr = 0; d = 4'b0001; ce = 1;
#20
clr = 0; d = 4'b1110; ce = 1;
#40
$stop;
end

endmodule
```

Multiplexer 4-to-1:



mux4to1.v

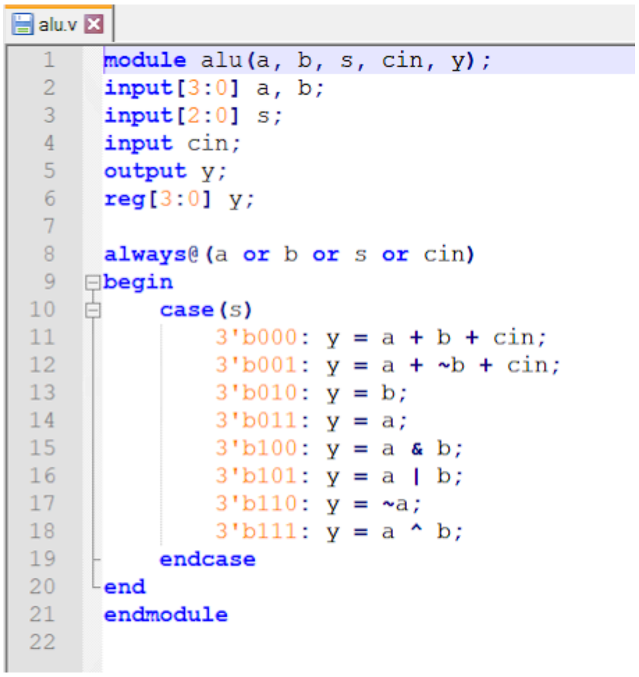
```
module mux4to1(d0, d1, d2, d3, sel, y);
    input[3:0] d0,d1,d2,d3;
    input[1:0] sel;
    output y;
    reg[3:0] y;

    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

```



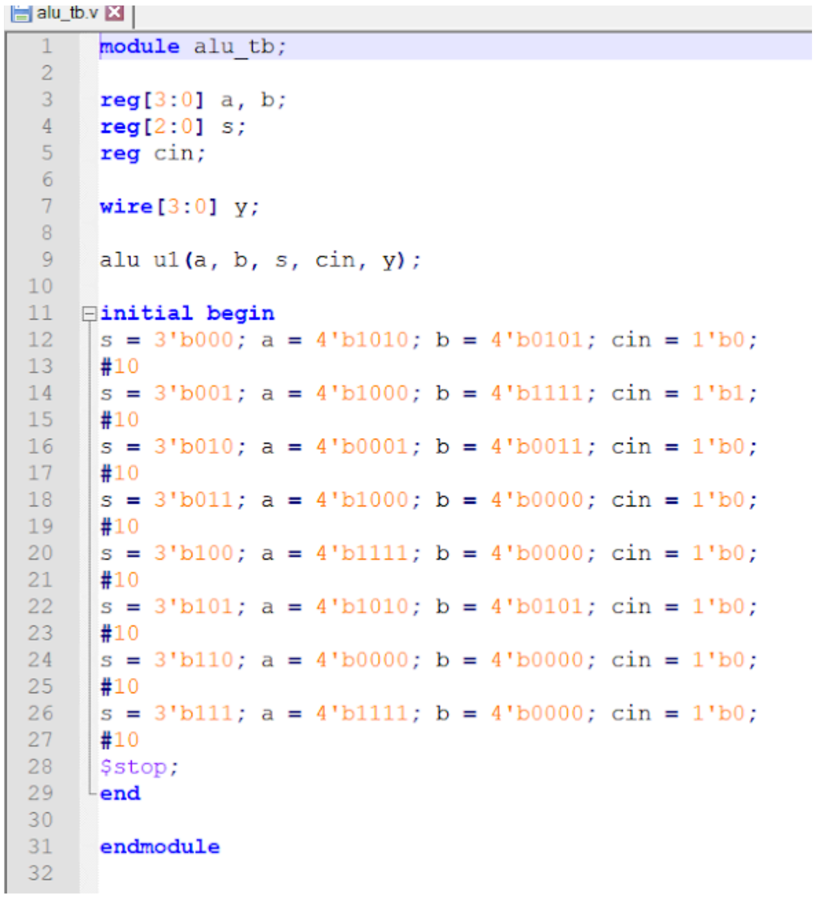
mux4to1_tb.v

```
module mux4to1_tb;

reg[3:0] d0, d1, d2, d3;
reg[1:0] sel;
wire[3:0]  y;

mux4to1 u1(d0, d1, d2, d3, sel, y);

initial begin
d0 = 4'b1100;  d1 = 4'b1010; d2 = 4'b0001; d3 = 4'b0101;  sel = 2'b00;
#10
d0 = 4'b1100;  d1 = 4'b1010; d2 = 4'b0001; d3 = 4'b0101;  sel = 2'b01;
#10
d0 = 4'b1100;  d1 = 4'b1010; d2 = 4'b0001; d3 = 4'b0101;  sel = 2'b10;
#10
d0 = 4'b1100;  d1 = 4'b1010; d2 = 4'b0001; d3 = 4'b0101;  sel = 2'b11;
#10
$stop;
end
endmodule

```

ALU:

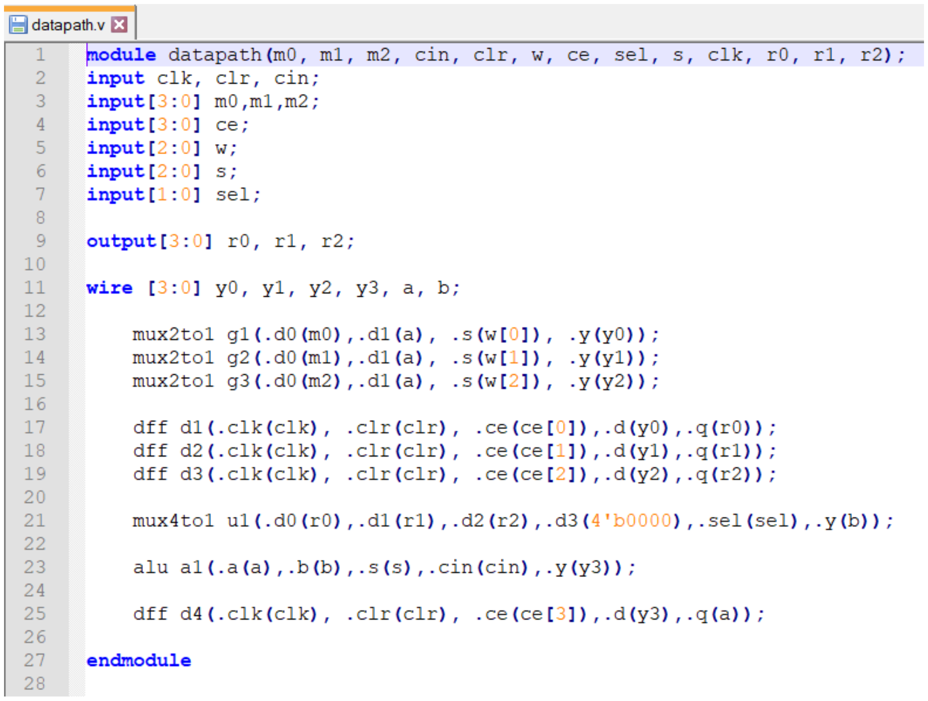


```verilog
module alu(a, b, s, cin, y);
input[3:0] a, b;
input[2:0] s;
input cin;
output y;
reg[3:0] y;

always@(a or b or s or cin)
begin
    case(s)
        3'b000: y = a + b + cin;
        3'b001: y = a + ~b + cin;
        3'b010: y = b;
        3'b011: y = a;
        3'b100: y = a & b;
        3'b101: y = a | b;
        3'b110: y = ~a;
        3'b111: y = a ^ b;
    endcase
end
endmodule
```



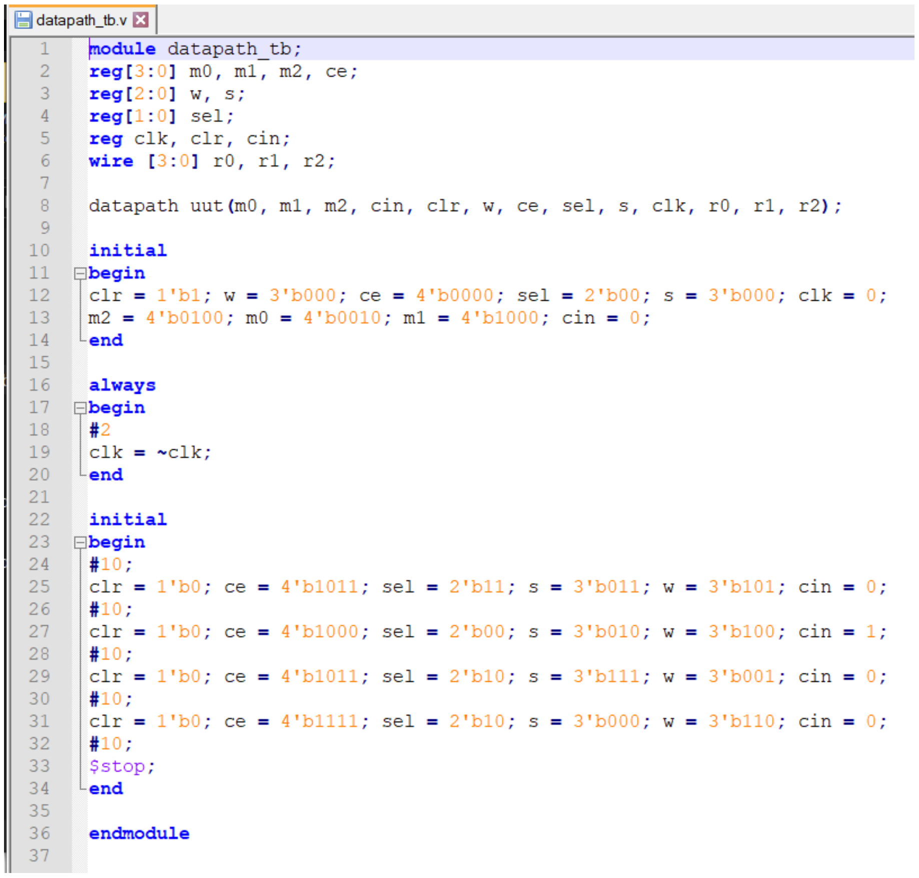
```verilog
module alu_tb;

reg[3:0] a, b;
reg[2:0] s;
reg cin;

wire[3:0] y;

alu u1(a, b, s, cin, y);

initial begin
s = 3'b000; a = 4'b1010; b = 4'b0101; cin = 1'b0;
#10
s = 3'b001; a = 4'b1000; b = 4'b1111; cin = 1'b1;
#10
s = 3'b010; a = 4'b0001; b = 4'b0011; cin = 1'b0;
#10
s = 3'b011; a = 4'b1000; b = 4'b0000; cin = 1'b0;
#10
s = 3'b100; a = 4'b1111; b = 4'b0000; cin = 1'b0;
#10
s = 3'b101; a = 4'b1010; b = 4'b0101; cin = 1'b0;
#10
s = 3'b110; a = 4'b0000; b = 4'b0000; cin = 1'b0;
#10
s = 3'b111; a = 4'b1111; b = 4'b0000; cin = 1'b0;
#10
$stop;
end

endmodule
```

Datapath:



datapath.v

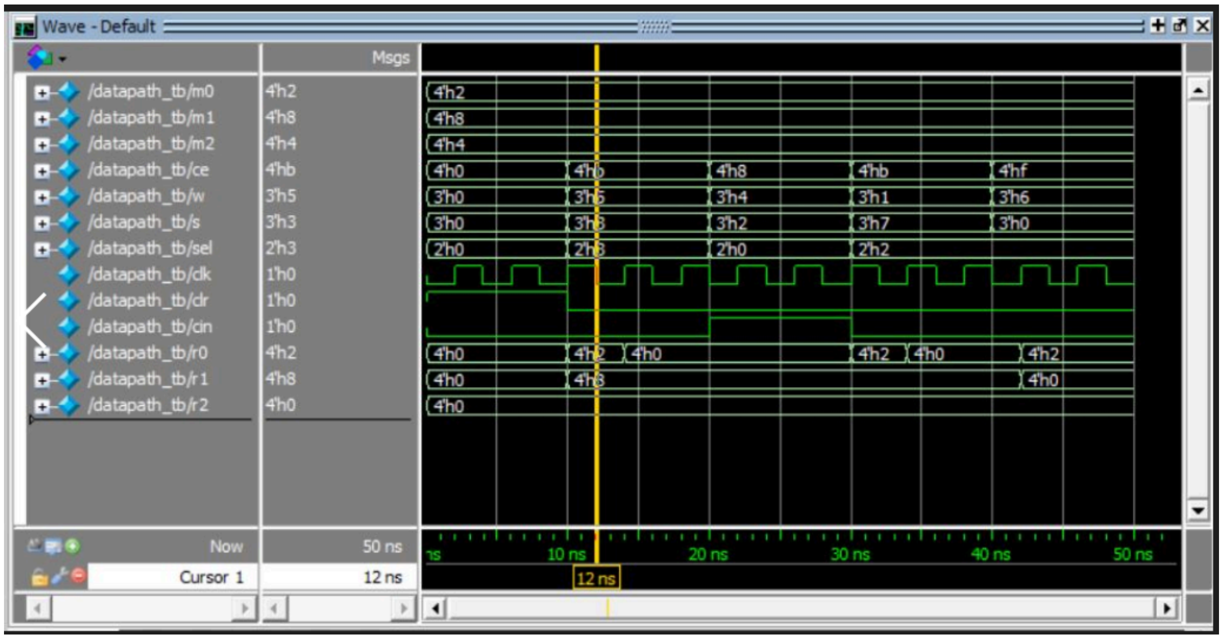
```
module datapath(m0, m1, m2, cin, clr, w, ce, sel, s, clk, r0, r1, r2);
input clk, clr, cin;
input[3:0] m0,m1,m2;
input[3:0] ce;
input[2:0] w;
input[2:0] s;
input[1:0] sel;

output[3:0] r0, r1, r2;

wire [3:0] y0, y1, y2, y3, a, b;

    mux2to1 g1(.d0(m0),.d1(a), .s(w[0]), .y(y0));
    mux2to1 g2(.d0(m1),.d1(a), .s(w[1]), .y(y1));
    mux2to1 g3(.d0(m2),.d1(a), .s(w[2]), .y(y2));

    dff d1(.clk(clk), .clr(clr), .ce(ce[0]),.d(y0),.q(r0));
    dff d2(.clk(clk), .clr(clr), .ce(ce[1]),.d(y1),.q(r1));
    dff d3(.clk(clk), .clr(clr), .ce(ce[2]),.d(y2),.q(r2));

    mux4to1 u1(.d0(r0),.d1(r1),.d2(r2),.d3(4'b0000),.sel(sel),.y(b));

    alu a1(.a(a),.b(b),.s(s),.cin(cin),.y(y3));

    dff d4(.clk(clk), .clr(clr), .ce(ce[3]),.d(y3),.q(a));

endmodule

```



datapath_tb.v

```
module datapath_tb;
reg[3:0] m0, m1, m2, ce;
reg[2:0] w, s;
reg[1:0] sel;
reg clk, clr, cin;
wire [3:0] r0, r1, r2;

datapath uut(m0, m1, m2, cin, clr, w, ce, sel, s, clk, r0, r1, r2);

initial
begin
clr = 1'b1; w = 3'b000; ce = 4'b0000; sel = 2'b00; s = 3'b000; clk = 0;
m2 = 4'b0100; m0 = 4'b0010; m1 = 4'b1000; cin = 0;
end

always
begin
#2
clk = ~clk;
end

initial
begin
#10;
clr = 1'b0; ce = 4'b1011; sel = 2'b11; s = 3'b011; w = 3'b101; cin = 0;
#10;
clr = 1'b0; ce = 4'b1000; sel = 2'b00; s = 3'b010; w = 3'b100; cin = 1;
#10;
clr = 1'b0; ce = 4'b1011; sel = 2'b10; s = 3'b111; w = 3'b001; cin = 0;
#10;
clr = 1'b0; ce = 4'b1111; sel = 2'b10; s = 3'b000; w = 3'b110; cin = 0;
#10;
$stop;
end

endmodule

```

Simulation Waveform:



In the waveform, we see that: r0 = 'a' because w[0] is '1' and ce[0] is '1'. r1 is = m2 because w[1] is '0' therefore the data passed through is M1 instead of 'a'. r2 = 0 because ce[2] is '0' so neither M2 or a is passed to r2. The 4-to-1 mux passes D3 which is '0000' because sel = '111' therefore b is '0000' the alu function performed is F = A because 's' = '011'.

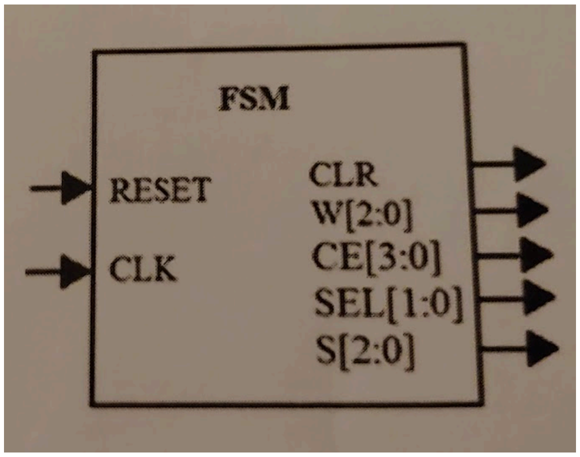
Result Discussion:

This part of the lab is a lot of work but it was manageable. The MUX selects properly, the flip flops store data properly, and the ALU does all of the calculations properly. The data path waveform shows the simplified processor doing the R2 = M0 + (not M1) + Cin calculation. The hard part of this lab was creating the testbench for the datapath and telling whether or not it was correct because I didn't know what 'a' was.

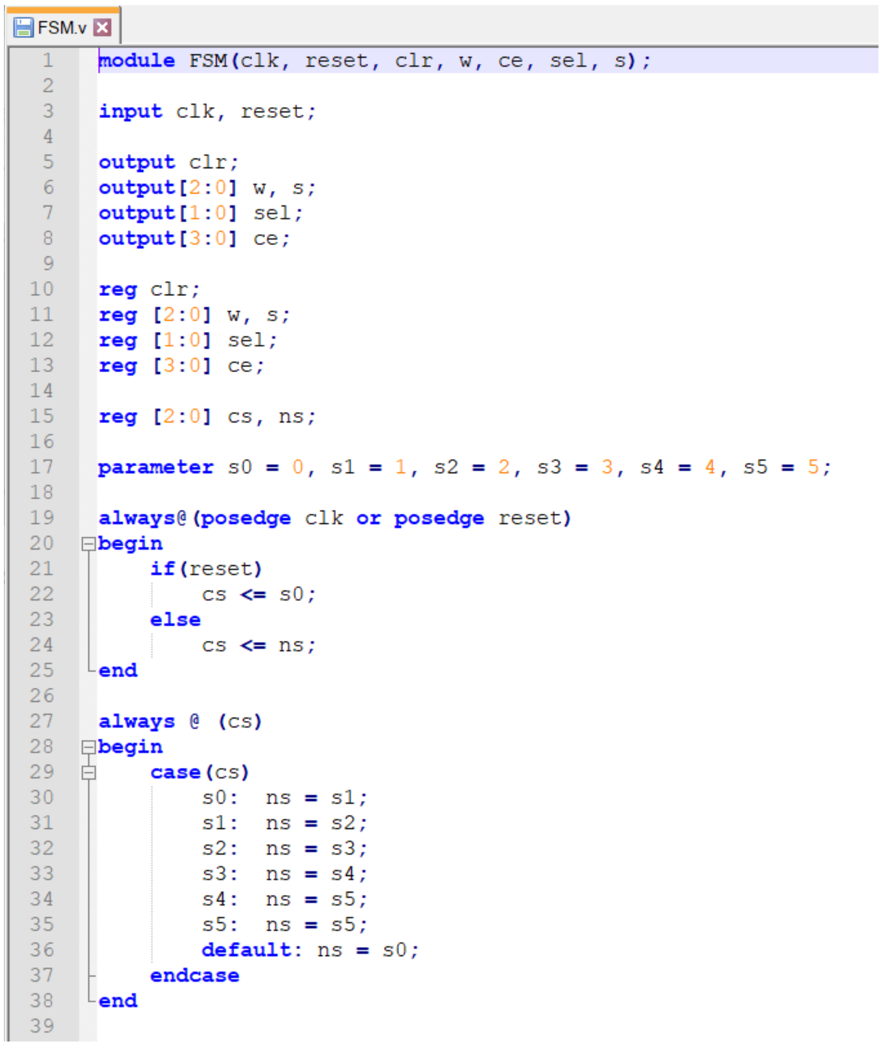
Part 2: Microprocessor Control Path Design

Design Purpose:

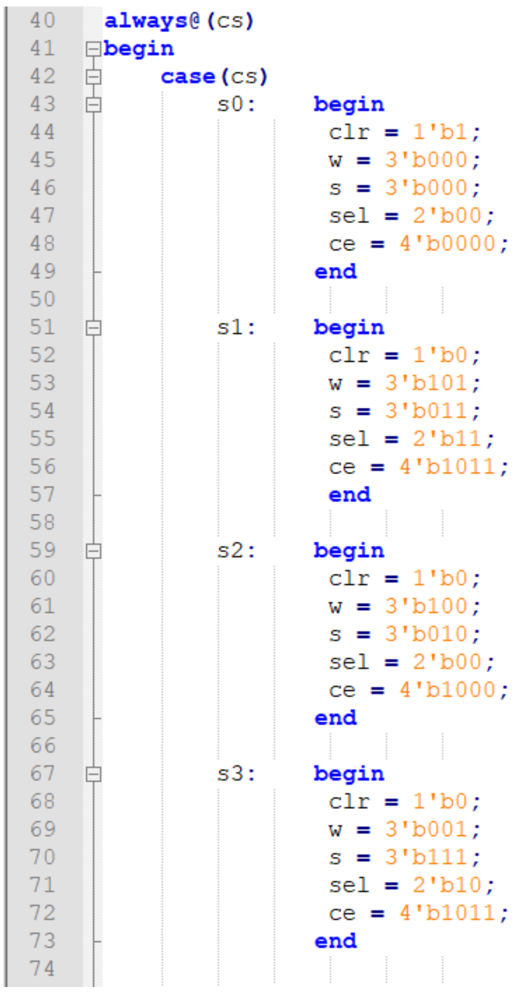
Once the data path module is built, we can proceed to building the finite state machine. The FSM will determine the internal signals for the datapath. It will also have a reset and a clk. The FSM will have 6 states each with different random data to confirm that each state transitions correctly.



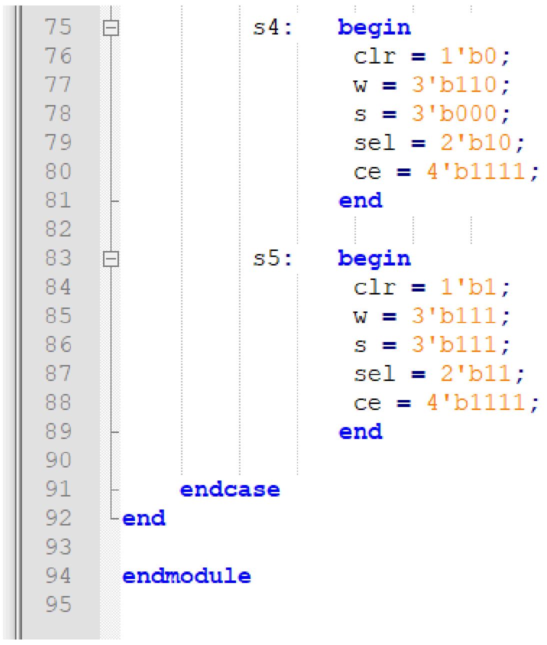
Design Source Code:



```verilog
module FSM(clk, reset, clr, w, ce, sel, s);

input clk, reset;

output clr;
output[2:0] w, s;
output[1:0] sel;
output[3:0] ce;

reg clr;
reg [2:0] w, s;
reg [1:0] sel;
reg [3:0] ce;

reg [2:0] cs, ns;

parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3, s4 = 4, s5 = 5;

always@(posedge clk or posedge reset)
begin
    if(reset)
        cs <= s0;
    else
        cs <= ns;
end

always @ (cs)
begin
    case(cs)
        s0:  ns = s1;
        s1:  ns = s2;
        s2:  ns = s3;
        s3:  ns = s4;
        s4:  ns = s5;
        s5:  ns = s5;
        default: ns = s0;
    endcase
end

```

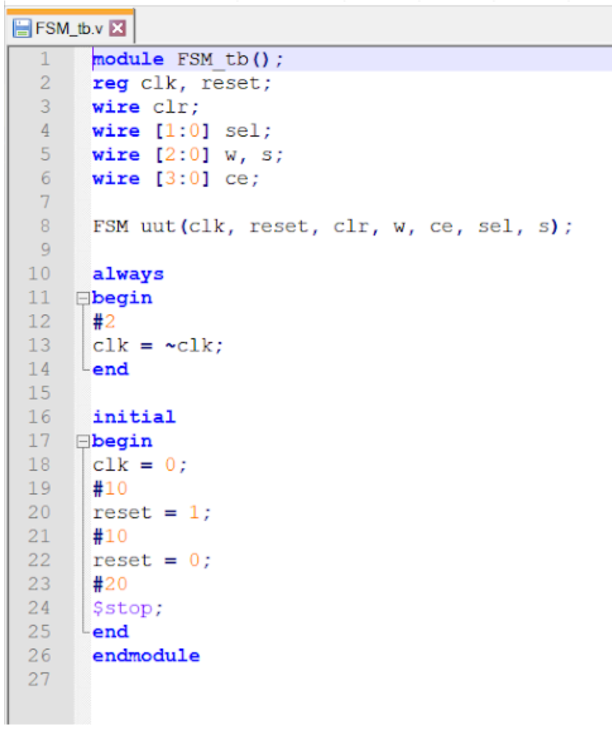


```
always@(cs)
begin
    case(cs)
        s0:     begin
                 clr = 1'b1;
                 w = 3'b000;
                 s = 3'b000;
                 sel = 2'b00;
                 ce = 4'b0000;
                end

        s1:     begin
                 clr = 1'b0;
                 w = 3'b101;
                 s = 3'b011;
                 sel = 2'b11;
                 ce = 4'b1011;
                 end

        s2:     begin
                 clr = 1'b0;
                 w = 3'b100;
                 s = 3'b010;
                 sel = 2'b00;
                 ce = 4'b1000;
                end

        s3:     begin
                 clr = 1'b0;
                 w = 3'b001;
                 s = 3'b111;
                 sel = 2'b10;
                 ce = 4'b1011;
                end

```



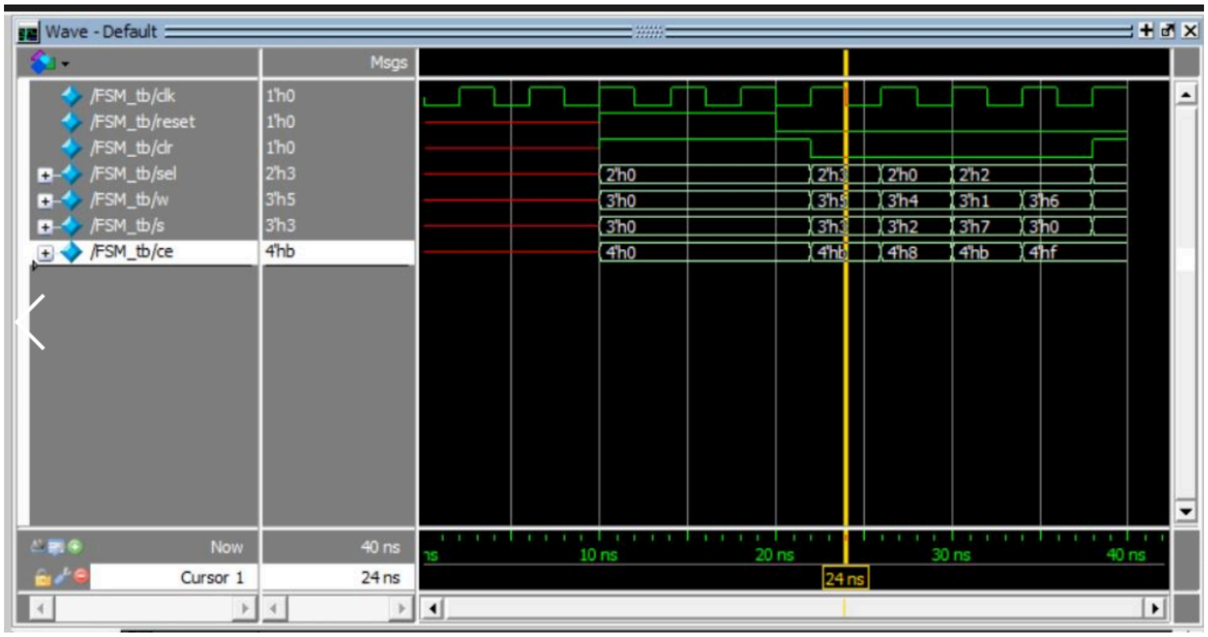
```
        s4:     begin
                 clr = 1'b0;
                 w = 3'b110;
                 s = 3'b000;
                 sel = 2'b10;
                 ce = 4'b1111;
                end

        s5:     begin
                 clr = 1'b1;
                 w = 3'b111;
                 s = 3'b111;
                 sel = 2'b11;
                 ce = 4'b1111;
                end

    endcase
end

endmodule

```

Testbench:



```
module FSM_tb();
reg clk, reset;
wire clr;
wire [1:0] sel;
wire [2:0] w, s;
wire [3:0] ce;

FSM uut(clk, reset, clr, w, ce, sel, s);

always
begin
#2
clk = ~clk;
end

initial
begin
clk = 0;
#10
reset = 1;
#10
reset = 0;
#20
$stop;
end
endmodule
```

Simulation Waveform:



Here we can see that the FSM shows the correct data for each state accordingly from the testbench.

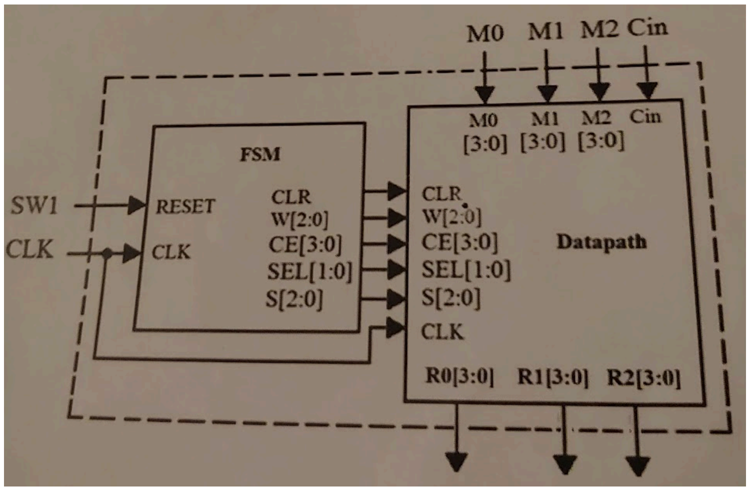
Result Discussion:

The point of the FSM is to control the internal signals to the data path. Since I chose random numbers, all I had to do was create a simple FSM so I can put it together with the datapath circuit for the next part.

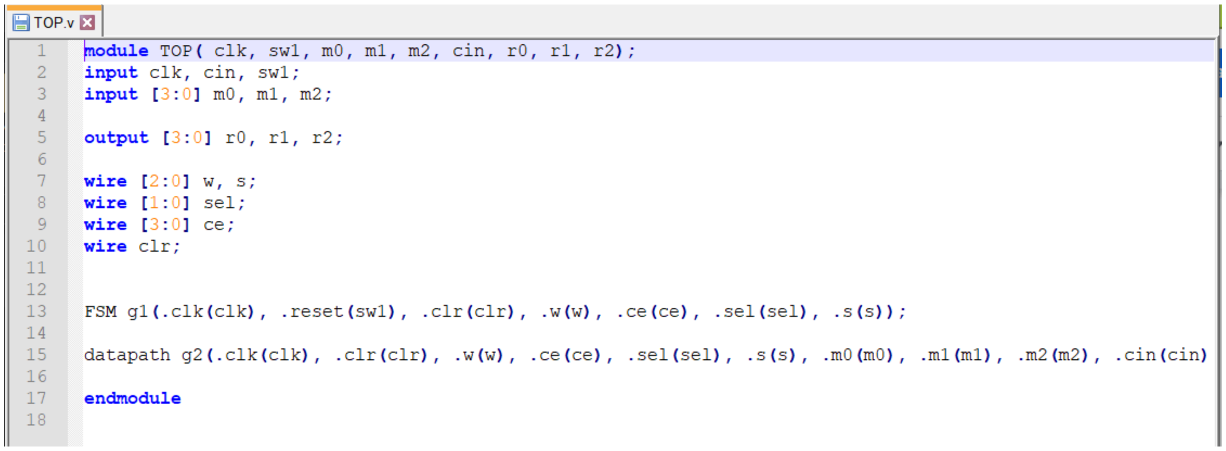
Part 3: Final Simplified Microprocessor Design

Design Purpose:

This part of the lab is to put the FSM together with the datapath with a “top” file. The top will determine the inputs m0, m1, m2, and Cin. The output should display r0, r1, and r2. The main focus of this last part is to learn how to integrate the data path and control path to design a simplified microprocessor.

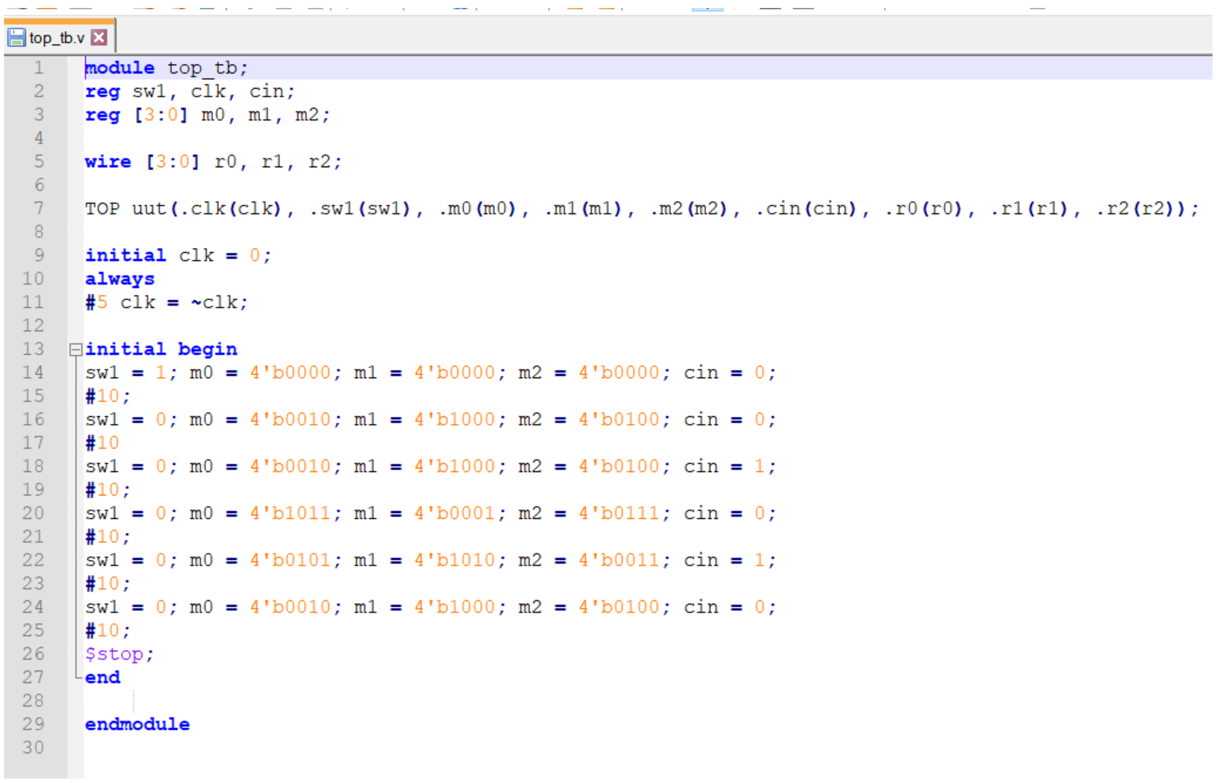


Design Source Code:


TOP.v

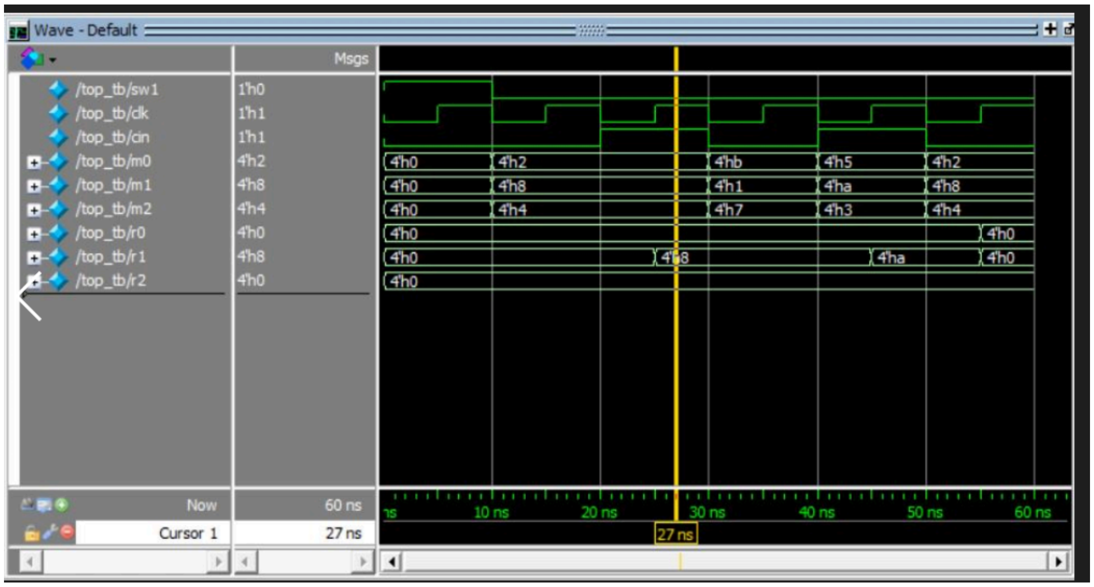
```
module TOP( clk, sw1, m0, m1, m2, cin, r0, r1, r2);
input clk, cin, sw1;
input [3:0] m0, m1, m2;

output [3:0] r0, r1, r2;

wire [2:0] w, s;
wire [1:0] sel;
wire [3:0] ce;
wire clr;


FSM g1(.clk(clk), .reset(sw1), .clr(clr), .w(w), .ce(ce), .sel(sel), .s(s));

datapath g2(.clk(clk), .clr(clr), .w(w), .ce(ce), .sel(sel), .s(s), .m0(m0), .m1(m1), .m2(m2), .cin(cin)

endmodule

```

Testbench:


top_tb.v

```
module top_tb;
reg sw1, clk, cin;
reg [3:0] m0, m1, m2;

wire [3:0] r0, r1, r2;

TOP uut(.clk(clk), .sw1(sw1), .m0(m0), .m1(m1), .m2(m2), .cin(cin), .r0(r0), .r1(r1), .r2(r2));

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

initial begin
sw1 = 1; m0 = 4'b0000; m1 = 4'b0000; m2 = 4'b0000; cin = 0;
#10;
sw1 = 0; m0 = 4'b0010; m1 = 4'b1000; m2 = 4'b0100; cin = 0;
#10
sw1 = 0; m0 = 4'b0010; m1 = 4'b1000; m2 = 4'b0100; cin = 1;
#10;
sw1 = 0; m0 = 4'b1011; m1 = 4'b0001; m2 = 4'b0111; cin = 0;
#10;
sw1 = 0; m0 = 4'b0101; m1 = 4'b1010; m2 = 4'b0011; cin = 1;
#10;
sw1 = 0; m0 = 4'b0010; m1 = 4'b1000; m2 = 4'b0100; cin = 0;
#10;
$stop;
end

endmodule

```

Simulation Waveform:



The top inserts m0, m1, and m2 into the datapath and the FSM determines the internal wiring.

Result Discussion:

This part wasn't hard to finish. The only thing we had to do is create a top file to connect the datapath and FSM. Then we just had to create a testbench to show the waveform. It is hard to see if the waveform is correct but if we look at the internal signals that the FSM is passing and what the inputs for the top file is, then we can compare the answer with the equation given in the lab manual to test if it is correct.

Conclusion

The main objective of this lab was to design a simple 4 bit microprocessor that implemented the logic equation R2 = M0 +not (M1) + Cin. We learned how to design data paths, control paths, and how to integrate both of them. The use of hierarchical design was used to complete this lab and reduce many errors. In all, this lab gave me great experience with combinational and sequential circuits. It took me a while to understand how the data path passed around data but as I looked at the D-flip-flop, multiplexers, and ALU separately it started to make sense. I learned how a simple microprocessor functions from this lab.