CPE 166 Advance Logic Design

Harpreet Sidhu

Lab Section #3

Lab 4 Report

David Quintanilla

Table of Contents

Introduction

Lab 4 introduced the concept of designing a simplified microprocessor. Consisting of several components the microprocessor introduced topics such the SRAM Design and ALU and revisited topics such as FSM's and top level designs for projects. In Part 1, I wrote a 4 bit number into the 32 address locations of the SRAM. After I finished writing the data I had to read the data from the RAM design. Following the table given in the lab instructions, Part 1 was fairly simple to complete. Part 2, dealt with creating the data path design for the microprocessor. This included creating two different types of MUX's as well as creating D-Flipflops. The data path in part 2 was very complicated and required several hours of debugging. Using Table 4-1 in the lab instructions I created an ALU that implemented the data path circuit. Part 3 revisited FSM and ask to implement the control path for the microprocessor design.

Lastly, Part 4 was the final top-level design for the microprocessor. Using the formula M0 + (not M1) + Cin to check our results. This lab took longer than most labs, however with several hours of debugging this lab taught me how to properly design a microprocessor.

Part 1: SRAM Design

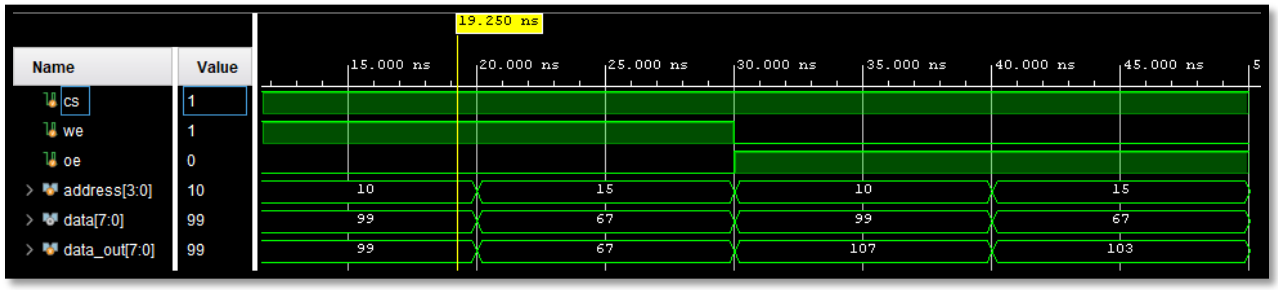
Ram Design (Verilog)

```
module ram ( address, data, cs, we, oe );
   input cs, we, oe ;
   input [3:0] address ;
   inout [7:0] data ;
   reg [7:0] data_out ;
   reg [7:0] mem [0:1023]; //10-bit address bus, 8-bit data bus.

   assign data = (cs && oe && !we) ? data_out : 8'bzzzzzzzzz;

   always @ (cs or we or data or address)

   if ( cs && we )

   mem[address] = data;

   always @ (cs or we or data or address or data)

      begin
         if (cs && !we && oe)
         data_out = mem[address];
   end
endmodule
```

Ram Design(Test Bench)

```
module ram_tb();

reg cs, we, oe;

reg [3:0] address;
wire [7:0] data;

reg [7:0] data_out ;

ram uut (.cs(cs) , .we(we), .oe(oe) , .address(address) ,.data(data) );


 assign data = (cs && oe && !we) ?  8'bzzzzzzzz : data_out;

initial
begin

    #10 we = 1'b1; cs = 1'b1; oe = 1'b0; address = 4'b1010; data_out = 8'b1100011;
    #10 we = 1'b1; cs = 1'b1; oe = 1'b0; address = 4'b1111; data_out = 8'b1000011;
    #10 we = 1'b0; cs = 1'b1; oe = 1'b1; address = 4'b1010; data_out = 8'b1101011;
    #10 we = 1'b0; cs = 1'b1; oe = 1'b1; address = 4'b1111; data_out = 8'b1100111;
    #10 $stop;


   end
   initial $monitor($time, ". we = %d, oe = %d, we = %d, address = %d, data_out = %d",
we,cs,oe,address,data_out);
endmodule
```

RAM Wave Form



MEM_FSM (Verilog)

```
module mem_fsm ( clk, reset, address, data, cs, we, oe );
    input clk, reset;
    output [5:0] address;
    inout [7:0] data;
    output cs, we, oe;
    reg cs, we, oe;
    reg [5:0] address;
    reg [7:0] data_reg;
    reg [2:0] state;
    parameter idle = 3'b000, s1= 3'b001, s2 = 3'b010, s3 = 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 == 32)
        state <= s3;
        else
        state <= s2;
        end
      s3:begin
        if (address == 0)
        state <= s1;
        else
        begin
        address <= address -1;
        state <=s3;
        end
        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 = 8'bZZZZZZZZ;
            end
        s1: begin //writing
            cs = 1;
            we = 1;
            oe = 0;
            data_reg = 8'b11000011;
            end
        s2: begin //writing
            cs = 1;
            we = 1;
            oe = 0;
            data_reg = 8'b11000011;
            end
        s3: begin //reading
            cs = 1;
            we = 0;
            oe = 1;
```

```
            end
        default: begin
            cs = 0;
            we = 0;
            oe = 0;
            data_reg = 8'bzzzzzzzz;
            end
      endcase
   end
endmodule
```

Top (Verilog)

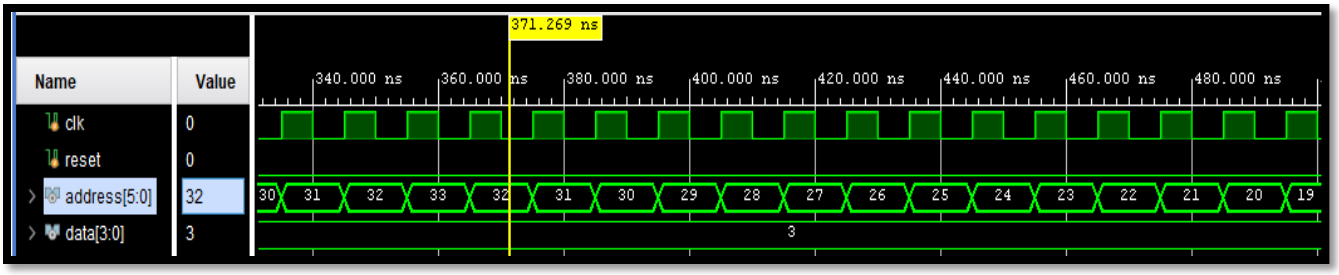
```
        module top(clk, reset, address, data);

input clk, reset;
output [5:0] address;
output [3:0] data;



   mem_fsm g1(.clk(clk),.address(address),.data(data),.cs(cs),.we(we),.oe(oe));
   ram     g2(.address(address),.data(data),.cs(cs),.we(we),.oe(oe));

endmodule
```

Top (Test Bench)

```
module top_tb();

reg clk, reset;
wire  [5:0] address;
```

```
wire  [3:0] data;


top uut(.clk(clk), .reset(reset), .address(address), .data(data));

always
begin

#5 clk = ~clk;
end
initial
begin

clk = 0;

#10 reset = 1;
#10 reset = 0;

#700 $stop;

end

endmodule
```

Top (Wave Form)



Part 2: Microprocessor Data Path Design

ALU Design (Verilog)

```
module ALU(A, B, S, cin, y);
        input  A, B, cin;
        input  [2:0] S;
        output [3:0] y;

        reg    [3:0] y;
always@( S or A or B or cin )
        begin
                case (S)
                        0 : y = A + B + cin;
                        1 : y = A + ~B + cin;
                        2 : y = B;
                        3 : y = A;
                        4 : y = (A & B);
                        5 : y = (A | B);
                        6 : y = ~A;
                        7 : y = (A ^ B);
                        default : y = 0;
```

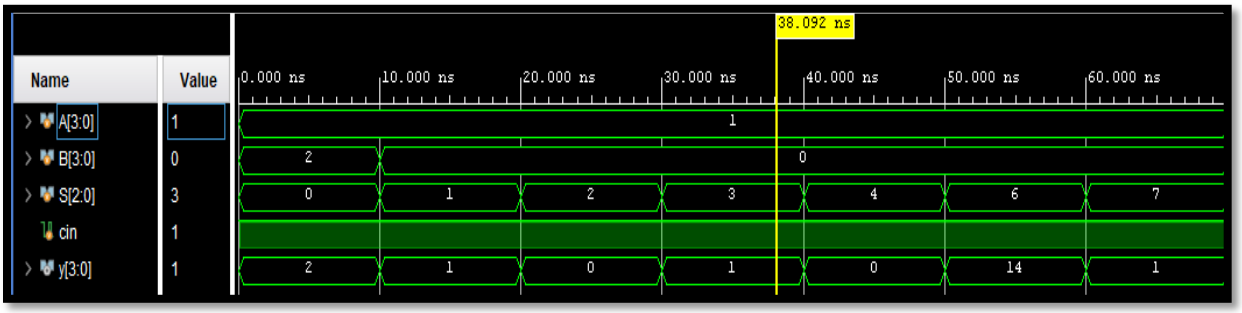
```
            endcase
        end
endmodule
```

ALU Design (Test Bench)

```
module ALU_tb;
reg[3:0] A, B;
reg[2:0] S;
reg cin;
wire[3:0] y;

ALU uut(A, B, S, cin, y);

initial begin

    S = 0; A = 1; B = 2; cin = 1;

  #10 S = 1; A = 1; B = 0; cin = 1;

  #10 S = 2; A = 1; B = 0; cin = 1;

  #10 S = 3; A = 1; B = 0; cin = 1;

  #10 S = 4; A = 1; B = 0; cin = 1;

  #10 S = 6; A = 1; B = 0; cin = 1;

  #10 S = 7; A = 1; B = 0; cin = 1;
```

```
  #10 $stop;
  end
endmodule
```

ALU Design (Wave Form)



Part 3 Microprocessor Control Path Design

FSM (Verilog)

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

 output reg  clr;
 output reg [2:0] w, s;
 output reg [1:0] sel;
 output 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 ) begin
cs <= s0;
end else begin
cs <= ns;
end

end

always@(cs) begin

case(cs)
	s0 : begin
	clr = 1'b1;
	w = 3'b100;
	s = 3'b010;
	sel = 2'b00;
	ce = 4'b0000;
	ns <= s1;
	end

	s1 : begin
	clr = 1'b0;
	w = 3'b100;
	s = 3'b010;
	sel = 2'b00;
	ce = 4'b1111;
	ns <= s2;
	end
```

```
s2 : begin
clr = 1'b0;
w = 3'b100;
s = 3'b010;
sel = 2'b00;
ce = 4'b1111;
ns <= s3;
end

s3 : begin
clr = 1'b0;
w = 3'b100;
s = 3'b001;
sel = 2'b01;
ce = 4'b1111;
ns <= s4;
end

s4 : begin
clr = 1'b0;
w = 3'b100;
s = 3'b001;
sel = 2'b01;
ce = 4'b0111;
ns <= s5;
end

s5 : begin
clr = 1'b0;
w = 3'b100;
```

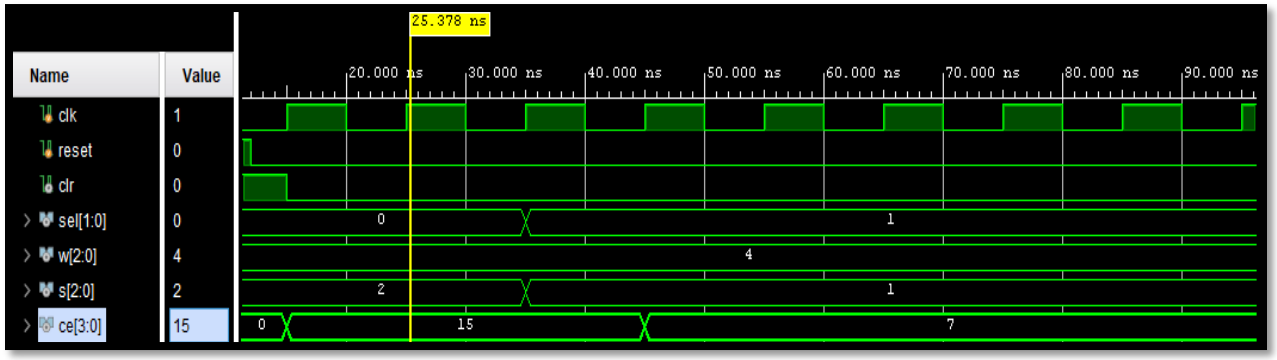
```
        s = 3'b001;
        sel = 2'b01;
        ce = 4'b0111;
        ns <= s5;
        end

   //default : ns = s0;
 endcase
 end
 endmodule
```

FSM (Test Bench)

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

    FSM uut(.clk(clk), .reset(reset), .clr(clr), .ce(ce), .w(w), .s(s), .sel(sel));
    always
    begin
        #5 clk = ~clk;
        end
        initial
        begin
        clk = 0;
        #2 reset = 1;
        #10 reset = 0;
```

```
        #150

        $stop;
    end
endmodule
```

FSM (Wave Form)



Data Path (Verilog)

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

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

 MUX2_1 g1(.d0(m0),.d1(a), .s(w[0]), .y(y0));
 MUX2_1 g2(.d0(m1),.d1(a), .s(w[1]), .y(y1));
 MUX2_1 g3(.d0(m2),.d1(a), .s(w[2]), .y(y2));
```

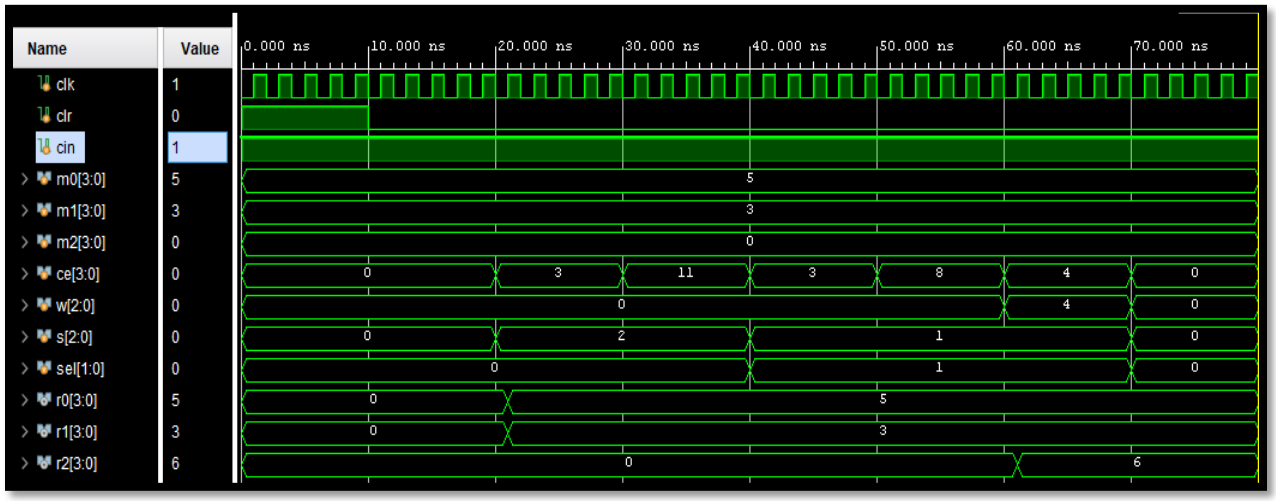
```
DFLIP_A d1(.clk(clk), .clr(clr), .ce(ce[0]),.d(y0),.q(r0));
DFLIP_A d2(.clk(clk), .clr(clr), .ce(ce[1]),.d(y1),.q(r1));
DFLIP_A d3(.clk(clk), .clr(clr), .ce(ce[2]),.d(y2),.q(r2));
DFLIP_A d4(.clk(clk), .clr(clr), .ce(ce[3]),.d(y3),.q(a));

MUX4_1 g4(.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));

Endmodule
```

Data Path (Test Bench)

```
`timescale 1ns / 1ps
module MUX_D_Combo_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;

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

 initial begin
 clr = 1'b1; w = 3'b000; ce = 4'b0000; sel = 2'b00; s = 3'b000; clk = 1'b0; m2 = 4'b0000;
 m0 = 4'b0101; m1 = 4'b0011;
 cin = 1;
 end
 always #1 clk = ~clk;
```

```
 initial begin
 #10; clr = 1'b0;
 #10; ce = 4'b0011; sel = 2'b00; s = 3'b010;
 #10; ce = 4'b1011;
 #10; ce = 4'b0011; sel = 2'b01; s = 3'b001;
 #10; ce = 4'b1000;
 #10; ce = 4'b0100; w = 3'b100;
 #10; w = 3'b000; ce = 4'b0000; sel = 2'b00; s = 3'b000;
 #10; $stop;
 end
endmodule
```

Data Path (Wave Form)



MUX2_1 (Verilog)

```
module MUX2_1(d0, d1, s, y);
input [3:0]        d1, d0;
input s;
output reg [3:0] y;
```

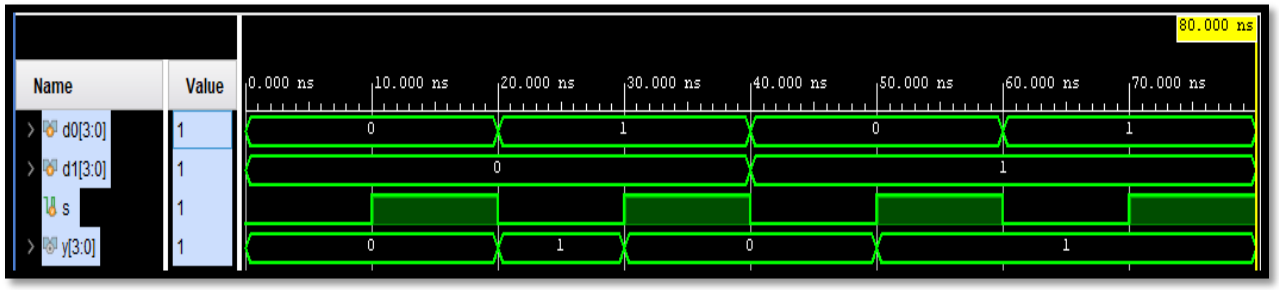
```
always@( s or d1 or d0 )
begin
  if (s)
        y = d1;
else
        y = d0;
end

endmodule
```

MUX2_1 (Test Bench)

```
module MUX2_1_tb();

reg[3:0] d0, d1;
reg s;
wire[3:0] y;
MUX2_1 u1(d0, d1, s, y);

initial begin
 d1 = 0; d0 = 0; s = 0;
 #10 d1 = 0; d0 = 0; s = 1;
 #10 d1 = 0; d0 = 1; s = 0;
 #10 d1 = 0; d0 = 1; s = 1;
 #10 d1 = 1; d0 = 0; s = 0;
 #10 d1 = 1; d0 = 0; s = 1;
 #10 d1 = 1; d0 = 1; s = 0;
 #10 d1 = 1; d0 = 1; s = 1;
 #10 $stop;
```

```
end
endmodule
```

MUX2_1 (Wave Form)



MUX4_1 (Verilog)

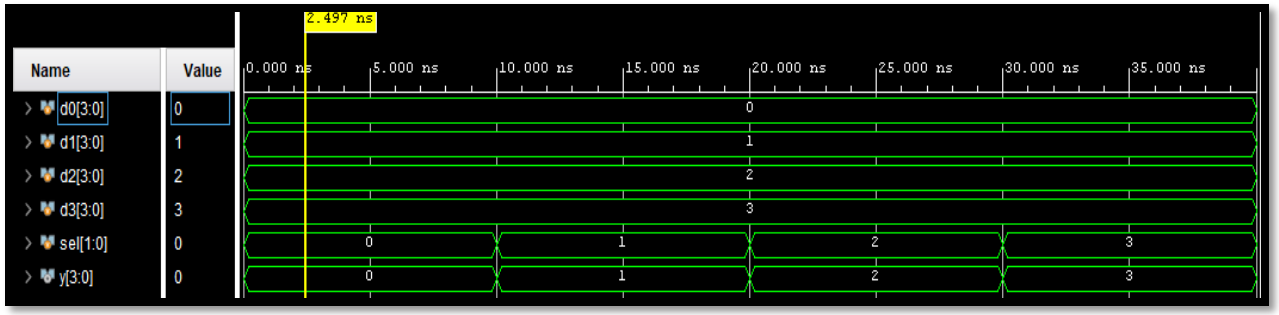
```
module MUX4_1(d0, d1, d2, d3, sel, y);

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



        always@( sel or d3 or d2 or d1 or d0 )
        begin
                case (sel)
                        0 : y = d0;
                        1 : y = d1;
                        2 : y = d2;
                        3 : y = d3;

                endcase
```

```
        end
endmodule
```

MUX4_1 (Test Bench)

```
module MUX4_1_tb;
reg[3:0] d0, d1, d2, d3;
reg[1:0] sel;
wire[3:0] y;

MUX4_1 u1(.d0(d0), .d1(d1), .d2(d2), .d3(d3), .sel(sel), .y(y));
initial
begin
    d0 = 0; d1 = 1; d2 = 2; d3 = 3; sel = 2'b00;
 #10 d0 = 0; d1 = 1; d2 = 2; d3 = 3; sel = 2'b01;
 #10 d0 = 0; d1 = 1; d2 = 2; d3 = 3; sel = 2'b10;
 #10 d0 = 0; d1 = 1; d2 = 2; d3 = 3; sel = 2'b11;
 #10 $stop;
end

endmodule
```

MUX4_1 (Wave Form)



DFLIP_A (Verilog)

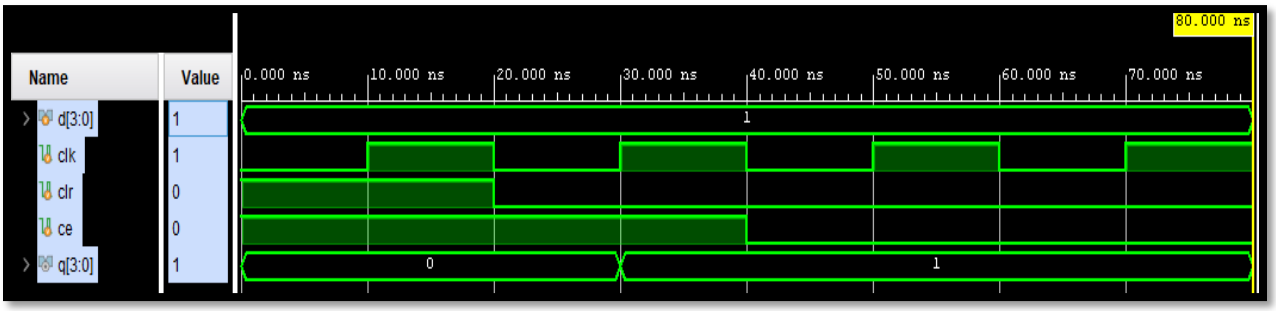
```
module DFLIP_A (clk, clr, d, ce, q);
   input[3:0] d;
   input clk, ce, clr;
   output reg[3:0] q;

   always@(posedge clr or posedge clk)

      begin

       if(clr) q <= 0;

       else if(ce) q <= d;

      end
endmodule
```

DFLIP_A (Test Bench)

```
module DFLIP_A_tb();

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

DFLIP_A utt(.clk(clk), .clr(clr), .d(d), .q(q), .ce(ce));
```

```
initial
clk = 0;
always
#10 clk = ~clk;
initial begin
    clr = 1; d = 1; ce = 1;
    #20 clr = 0;
    #20 ce = 0;
    #40
    $stop;
    end
endmodule
```

DFLIP A (Wave Form)



Part 4: Final 4-bit Microprocessor Design

Top Final (Verilog)

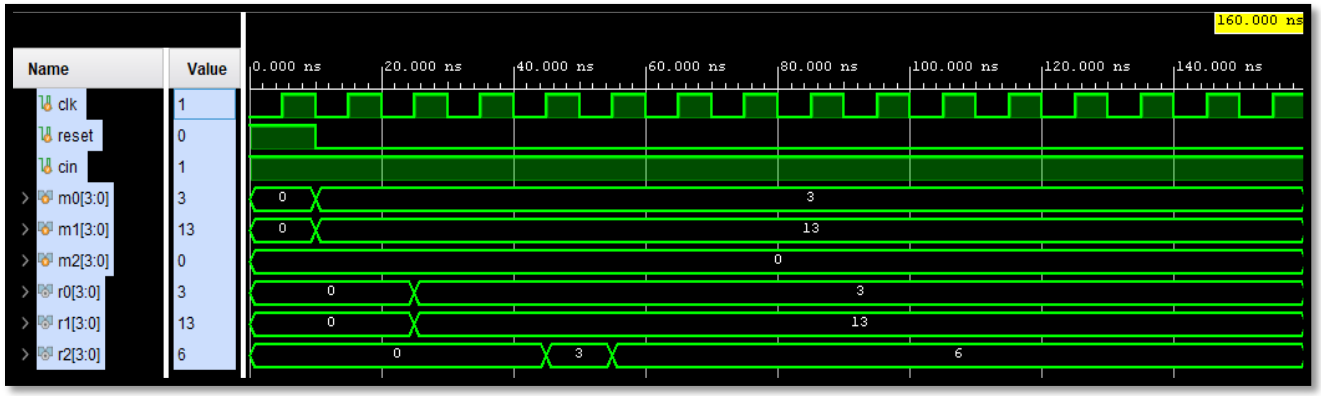
```
module Top_Final( clk, reset, m0, m1, m2, cin, r0, r1, r2);

 input clk, cin, reset;

 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 fsm1(.clk(clk), .reset(reset),  .clr(clr), .w(w), .ce(ce), .sel(sel), .s(s));

MUX_D_Combo mux_d_combo1(.clk(clk), .clr(clr), .w(w), .ce(ce), .sel(sel), .s(s), .m0(m0),

.m1(m1), .m2(m2), .cin(cin), .r0(r0), .r1(r1), .r2(r2));

Endmodule
```

Top Final (Test Bench)

```
module Top_Final_tb();

reg clk, reset, cin;

reg [3:0] m0,m1,m2;

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

Top_Final uut(.clk(clk), .reset(reset), .cin(cin), .m0(m0), .m1(m1), .m2(m2), .r0(r0), .r1(r1),

.r2(r2));

always begin

#5 clk = ~clk;

end

initial begin

    clk = 0; reset = 1; cin = 1; m0=0; m1=0; m2=0;

    //#10 reset = 0;

   // m0=4'b1000;

    //m1=4'b0001;
```

```
    //m2=4'b1111;

    //cin = 1;

    #10 m0 = 3; reset = 0;

    m1 = 13;

    //m2 = 2;

    #150 $stop;

end

endmodule
```

Top Final (Wave Form)



Conclusion

To conclude the main objective of this lab; which was to design a simple microprocessor that implemented the logic equation R2 = M0 +not (M1) + Cin was successfully created. The use of hierarchal design within the different modules was again used in order to link all modules together and create instances. It was again great practice for designing combinational and sequential circuits. I believe that this lab explored several topics that I was interested in on retouching and furthered my understanding of advance logic designs.