CPE166 Advanced Logic Design

Lab Number 4

Anthony Chavez

Table of Contents

Introduction

Lab 4 was a four-part lab that served as an introduction to advanced logic design in Verilog. Part one of the lab was a SRAM Design and the last three parts were a Simplified Microprocessor Design. All code and test bench simulations were written and tested in the Xilinx Vivado IDE.

Laboratory Report

Part 1 – SRAM Design

Design Purpose:
In the SRAM design, we had to write 4'b1010, a four-bit binary number with the value of 10, into the 32 address locations of the SRAM. After the writing was complete, we had to read data from the RAM. The SRAM is based on the following function table.

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

Verilog Design:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/13/20
// Module Name: ram
// Description: Random Access Memory
// Project Part Number: Lab 4, Part 1
//////////////////////////////////////////////////////////////////////////////////


module ram( address, data, cs, we, oe );
    input cs, we, oe;
    input[4:0] address;
    inout[3:0] data;

    reg[3:0] data_out;
    reg[3:0] mem[0:31]; // 32 address lines with depth=4

    assign data = (cs && oe && !we) ? data_out : 4'bzzzz;  //High-z

    always@(cs or we or data or address)    //Read Data
    begin
```

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

    always@(cs or we or data or address)    //Write Data
    begin
        if(cs && we)
            mem[address] = data;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/13/20
// Module Name: mem_fsm
// Description: memory finite state machine
// Project Part Number: Lab 4, Part 1
//////////////////////////////////////////////////////////////////////////////////


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 = 1'b0;
    reg we = 1'b0;
    reg oe = 1'b0;
    reg [4:0] address = 5'b0000;
    reg [3:0] data_reg = 4'b0000;
    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
```

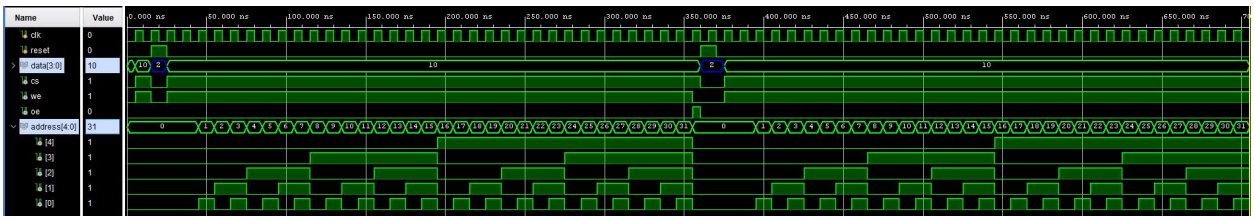
```
            s1: begin
                state <= s2;
                address <= 0;
                end
            s2: begin
                address <= address + 1;
                if(address == 31)
                    state <= s3;
                else
                    state <= s2;
                end
            s3: begin
                state <= s4;
                address <= 0;
                end
            s4: begin
                address <= address + 1;
                state <= s4;
                end
            default: begin
                state <= s1;
                address <= 0;
                end
        endcase
    end

    always@(state) begin
        case (state)
            idle: begin //high impedance (Z)
                cs = 0;
                we = 0;
                oe = 0;
                data_reg = 4'bZZZZ;
            end
            s1: begin //set 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 //set reading
                cs = 1;
                we = 0;
                oe = 1;
```

```
            data_reg = 4'bZZZZ;
        end
        s4: begin //reading
            cs = 1;
            we = 0;
            oe = 1;
            data_reg = 4'bZZZZ;
        end
        default: begin
            cs = 0;
            we = 0;
            oe = 0;
            data_reg = 4'bzzzz;
        end
      endcase
  end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Module Name: top
// Author: Anthony Chavez
// Date Last Revised: 11/13/20
// Module Name: top
// Description: Top Design of SRAM Design
// Project Part Number: Lab 4, Part 1
//////////////////////////////////////////////////////////////////////////////////


module top(cs, we, oe, data, clk, address, reset);
   output cs;
   output  we;
   output  oe;
   output [4:0] address;
   inout [3:0] data = 4'b0000;
   input  clk, reset;

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

Verilog Testbench Design and Simulation Waveforms:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Module Name: top
// Author: Anthony Chavez
// Date Last Revised: 11/11/20
// Module Name: top_tb
// Description: Top Design of SRAM Design testbench
// Project Part Number: Lab 4, Part 1
//////////////////////////////////////////////////////////////////////////////////


module top_tb;
    reg clk, reset;
    wire[3:0] data;
    wire cs, we, oe;
    wire[4:0] address;

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

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

    initial begin
        reset = 0;
        #15 reset = 1;
        #10 reset = 0;
        #335 reset = 1;
        #10 reset = 0;
        #335 $stop;
    end
endmodule
```



The first waveform was the initial design, but I decided to modify the design to the waveform shown in the second snippet. The initial design had a 32 by 8 ram design, but I changed this to a 32 by 4.

Result Discussion:
Comparing the initial waveform with my modified version of the SRAM design, I was successful in implementing the design. One problem I came across was only being able to display the address lines with “1010” written in to each of them, but I had several no-states for each address line until the data was written in the address. I went to office hours and was able to understand and fix my mistake.

Part 2 – Microprocessor Data Path Design

Design Purpose:
This is part 1 of 3 of designing a simplified microprocessor. The goal is to use structural hierarchical design method to implement the data path circuit of the microprocessor.

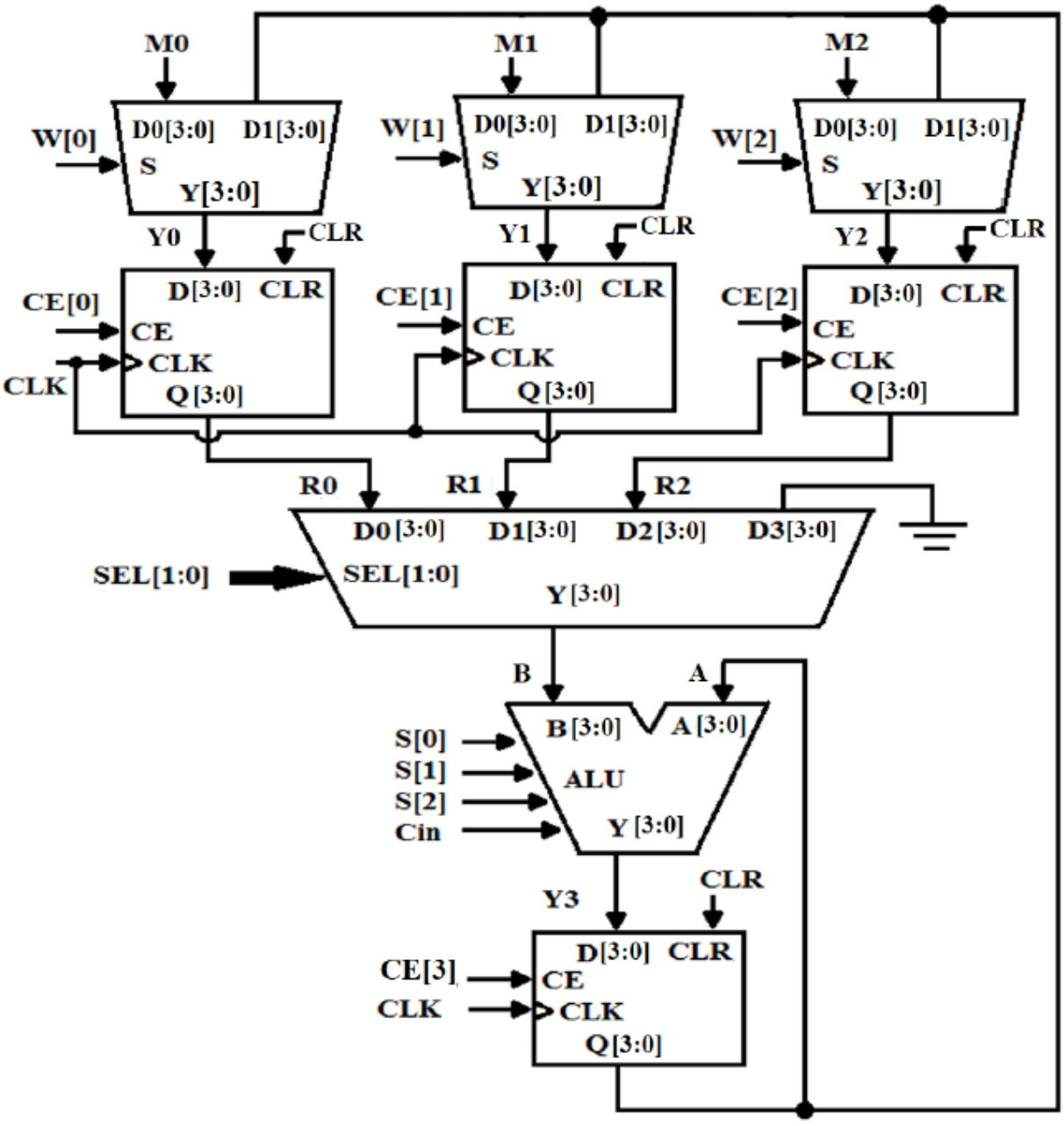


Figure 4-2. Simplified microprocessor data path circuit

Here we have the circuit diagram for the data path of the processor.

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

Here is the truth table for the Arithmetic Logic Unit (ALU) of the processor.

Verilog Design:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/19/20
// Module Name: muxb
// Description: multiplexer with 2 4-bit inputs
// Project Part Number: Lab 4, Part 2
//////////////////////////////////////////////////////////////////////////////////


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

    always@(d0 or d1 or s) begin
        y = (s) ? d1 : d0;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/15/20
// Module Name: dFlipFlop
// Description: D Flip Flop with CLR and CLK enable
// Project Part Number: Lab 4, Part 2
//////////////////////////////////////////////////////////////////////////////////


module dFlipFlop(clk, clr, ce, D, Q);
    input clk, clr, ce;
    input[3:0] D;
    output[3:0] Q;
    reg[3:0] Q;

    always@(posedge clk) begin
        if(clr)
            Q <= 0;
        else if(ce)
            Q <= D;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/15/20
// Module Name: mux4
// Description: 4 to 1 Multiplexor
// Project Part Number: Lab 4, Part 2
//////////////////////////////////////////////////////////////////////////////////


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;
```

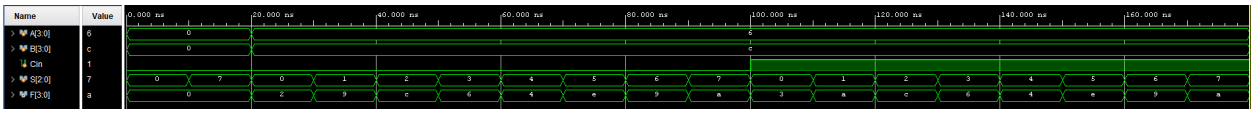
```
            2'b11: Y = D3;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/19/20
// Module Name: alu
// Description: Arithmetic Logic Unit
// Project Part Number: Lab 4, Part 2
//////////////////////////////////////////////////////////////////////////////////


module alu(A, B, Cin, S, F);
    input[3:0] A, B;
    input Cin;
    input[2:0] S;
    output[3:0] F;
    reg[3:0] F;

    always@(S) begin
        case (S)
            3'b000: F = A + B + Cin;
            3'b001: F = A + ~B + Cin;
            3'b010: F = B;
            3'b011: F = A;
            3'b100: F = A & B;
            3'b101: F = A | B;
            3'b110: F = ~A;
            3'b111: F = A ^ B;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/19/20
// Module Name: datapath
// Description: datapath of processor
// Project Part Number: Lab 4, Part 2
//////////////////////////////////////////////////////////////////////////////////


module datapath(M0, M1, M2, Cin, CLR, W, CE, SEL, S, CLK, R0, R1, R2);
    input[3:0] M0, M1, M2;
    input Cin, CLR;
    input[2:0] W;
    input[3:0] CE;
    input[1:0] SEL;
    input[2:0] S;
    input CLK;
    output[3:0] R0, R1, R2;
    wire[3:0] Y0, Y1, Y2, Y3;
    wire[3:0] ground = 0;
    wire[3:0] A, B;

    muxb g1(.s(W[0]),.d0(M0),.d1(A),.y(Y0));
    muxb g2(.s(W[1]),.d0(M1),.d1(A),.y(Y1));
    muxb g3(.s(W[2]),.d0(M2),.d1(A),.y(Y2));

    dFlipFlop g4(.clk(CLK), .clr(CLR), .ce(CE[0]), .D(Y0), .Q(R0));
    dFlipFlop g5(.clk(CLK), .clr(CLR), .ce(CE[1]), .D(Y1), .Q(R1));
    dFlipFlop g6(.clk(CLK), .clr(CLR), .ce(CE[2]), .D(Y2), .Q(R2));

    mux4 g7(.D0(R0), .D1(R1), .D2(R2), .D3(ground), .sel(SEL), .Y(B));

    alu g8(.A(A), .B(B), .Cin(Cin), .S(S), .F(Y3));
    dFlipFlop g9(.clk(CLK), .clr(CLR), .ce(CE[3]), .D(Y3), .Q(A));
endmodule
```

Verilog Testbench Design and Simulation Waveforms:

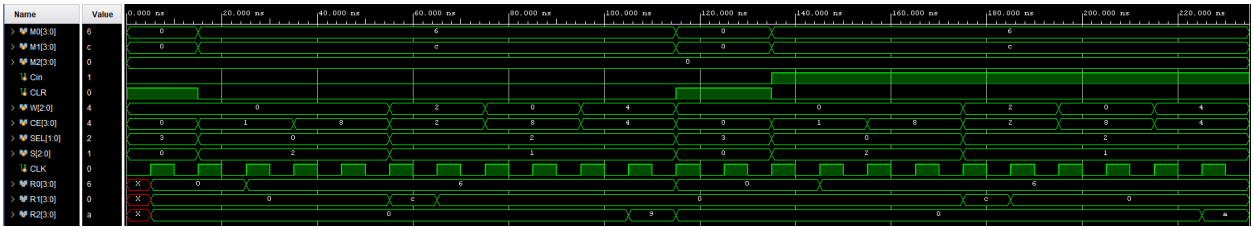
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/15/20
// Module Name: alu_tb
// Description: Arithmetic Logic Unit testbench
// Project Part Number: Lab 4, Part 2
//////////////////////////////////////////////////////////////////////////////////


module alu_tb;
    reg[3:0] A, B;
    reg Cin;
    reg[2:0] S;
    wire[3:0] F;

    alu uut(.A(A), .B(B), .Cin(Cin), .S(S), .F(F));

    initial begin
        A = 4'b0000; B = 4'b0000; Cin = 1'b0; S = 4'b0000;
        #10 S = 4'b111; //allow correct value for first alu function
        //without Cin
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b000;
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b001;
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b010;
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b011;
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b100;
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b101;
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b110;
        #10 A = 4'b0110; B = 4'b1100; Cin = 0; S = 3'b111;
        //with Cin
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b000;
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b001;
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b010;
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b011;
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b100;
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b101;
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b110;
        #10 A = 4'b0110; B = 4'b1100; Cin = 1; S = 3'b111;
        #10 $stop;
    end
endmodule
```



This waveform and testbench confirms the correct computations are being made by the ALU of the processor. If we calculate each computation by hand or calculator we will get the same result.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/18/20
// Module Name: datapath_tb
// Description: datapath of processor testbench
// Project Part Number: Lab 4, Part 2
//////////////////////////////////////////////////////////////////////////////////


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

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

    initial begin
        M0=0; M1=0; M2=0; Cin=0; CLR=1; W=0; CE=0; SEL=2'b11; S=0;
        //without Cin
        #15 M0=4'b0110; M1=4'b1100; M2=0; Cin=0; CLR=0; W=0; CE=4'b0001; SEL=0;
S=3'b010;
        #20 CLR=0; W=0; CE=4'b1000; SEL=0; 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;
        //with Cin
        #20 M0=0; M1=0; M2=0; Cin=0; CLR=1; W=0; CE=0; SEL=2'b11; S=0;
        #20 M0=4'b0110; M1=4'b1100; M2=0; Cin=1; CLR=0; W=0; CE=4'b0001; SEL=0;
S=3'b010;
        #20 CLR=0; W=0; CE=4'b1000; SEL=0; 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
```



This testbench and waveform was used to verify the data path component of the processor was functioning properly.

Result Discussion:

Based on the hand computations and the waveforms verifying our hand-written work, we can conclude the data path component of the processor is functioning properly. Next we will be able to program the finite state machine the processor to control the data path circuit.

Part 3 – Microprocessor Control Path Design

Design Purpose:
In order to control the data path circuit, we needed to design a finite state machine to handle this functionality.

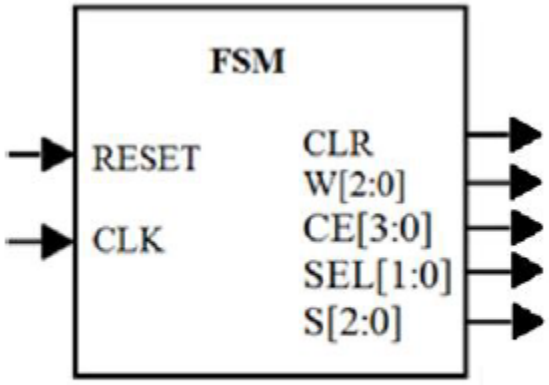
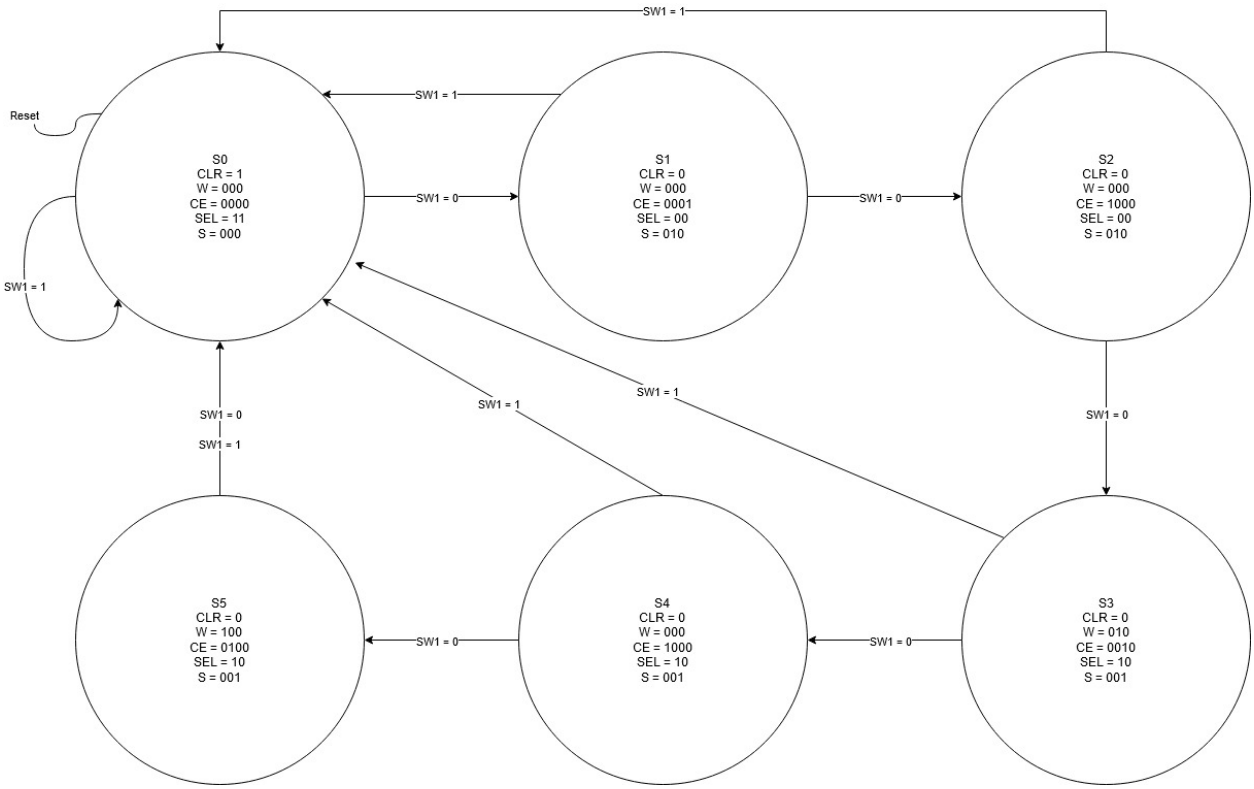


Figure 4-3. Simplified microprocessor control path block diagram

Here is the block diagram for the FSM circuit of the processor.



Here is the FSM diagram I implemented for the FSM circuit.

Verilog Design:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/18/20
// Module Name: fsm
// Description: finite state machine of processor
// Project Part Number: Lab 4, Part 3
//////////////////////////////////////////////////////////////////////////////////


module fsm(RESET, CLK, CLR, W, CE, SEL, S);
    input RESET, CLK;
    output CLR;
    output[2:0] W;
    output[3:0] CE;
    output[1:0] SEL;
    output[2:0] S;

    reg CLR = 1'b0;
    reg[2:0] W = 3'b000;
    reg[3:0] CE = 4'b0000;
    reg[1:0] SEL = 2'b00;
    reg[2:0] S = 2'b000;
    reg[2:0] state;
    parameter s0 = 3'b000, s1 = 3'b001, s2 = 3'b010, s3 = 3'b011, s4 = 3'b100, s5 = 3'b101;

    always@(posedge CLK or posedge RESET) begin
        if(RESET) begin
            state <= s0;
        end
        else
            case(state)
                s0: begin
                    state <= s1;
                end
                s1: begin
                    state <= s2;
                end
                s2: begin
                    state <= s3;
                end
                s3: begin
                    state <= s4;
                end
```

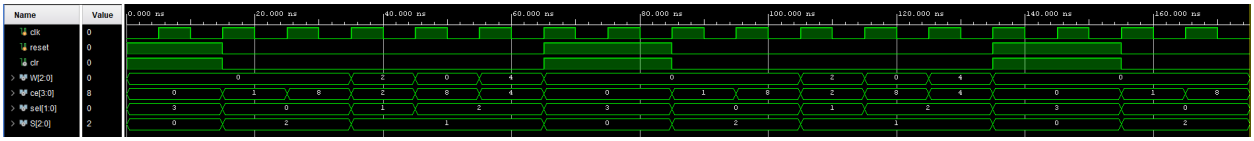
```
            s4: begin
                state <= s5;
            end
            s5: begin
                state <= s0;
            end
            default: begin
                state <= s0;
            end
        endcase
end

always@(state) begin
    case(state)
        s0: begin
            CLR <= 1'b1;
            W = 3'b000;
            CE = 4'b0000;
            SEL <= 2'b11;
            S = 3'b000;
        end
        s1: begin
            CLR <= 1'b0;
            W = 3'b000;
            CE = 4'b0001;
            SEL <= 2'b00;
            S = 3'b010;
        end
        s2: begin
            CLR <= 1'b0;
            W = 3'b000;
            CE = 4'b1000;
            SEL <= 2'b00;
            S = 3'b010;
        end
        s3: begin
            CLR <= 1'b0;
            W = 3'b010;
            CE = 4'b0010;
            SEL <= 2'b01;
            S = 3'b001;
        end
        s4: begin
            CLR <= 1'b0;
            W = 3'b000;
            CE = 4'b1000;
```

```
            SEL <= 2'b10;
            S = 3'b001;
        end
        s5: begin
            CLR <= 1'b0;
            W = 3'b100;
            CE = 4'b0100;
            SEL <= 2'b10;
            S = 3'b001;
        end
        default: begin
            CLR <= 1'b1;
            W = 3'b000;
            CE = 4'b0000;
            SEL <= 2'b11;
            S = 3'b000;
        end
    endcase
  end
endmodule
```

Verilog Testbench Design and Simulation Waveforms:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/18/20
// Module Name: fsm_tb
// Description: finite state machine of processor testbench
// Project Part Number: Lab 4, Part 3
//////////////////////////////////////////////////////////////////////////////////


module fsm_tb;
  reg clk, reset;
  wire clr;
  wire[2:0] W;
  wire[3:0] ce;
  wire[1:0] sel;
  wire[2:0] S;

  fsm uut(.RESET(reset), .CLK(clk), .CLR(clr), .W(W), .CE(ce), .SEL(sel), .S(S));

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

```
    initial begin
        reset = 1;
        #15 reset = 0;
        #50 reset = 1;
        #20 reset = 0;
        #50 reset = 1;
        #20 reset = 0;
        #20 $stop;
    end
endmodule
```



This testbench and waveform was used to verify my finite state machine was implemented properly and matched my drawn FSM diagram.

Result Discussion:
Not much was done in this portion of the design process for the microprocessor design. However, the waveform matches my drawn FSM diagram and the code was modified accordingly upon testing the final design in the next portion of the microprocessor design process.

Part 4 – Final 4-bit Microprocessor Design

Design Purpose:
This is the final portion of the simplified 4-bit microprocessor. In this part of the design, I combined the finite state machine and data path circuits together to create the final top design. The design was to implement the following operation: R2 = M0 + (not M1) + Cin.

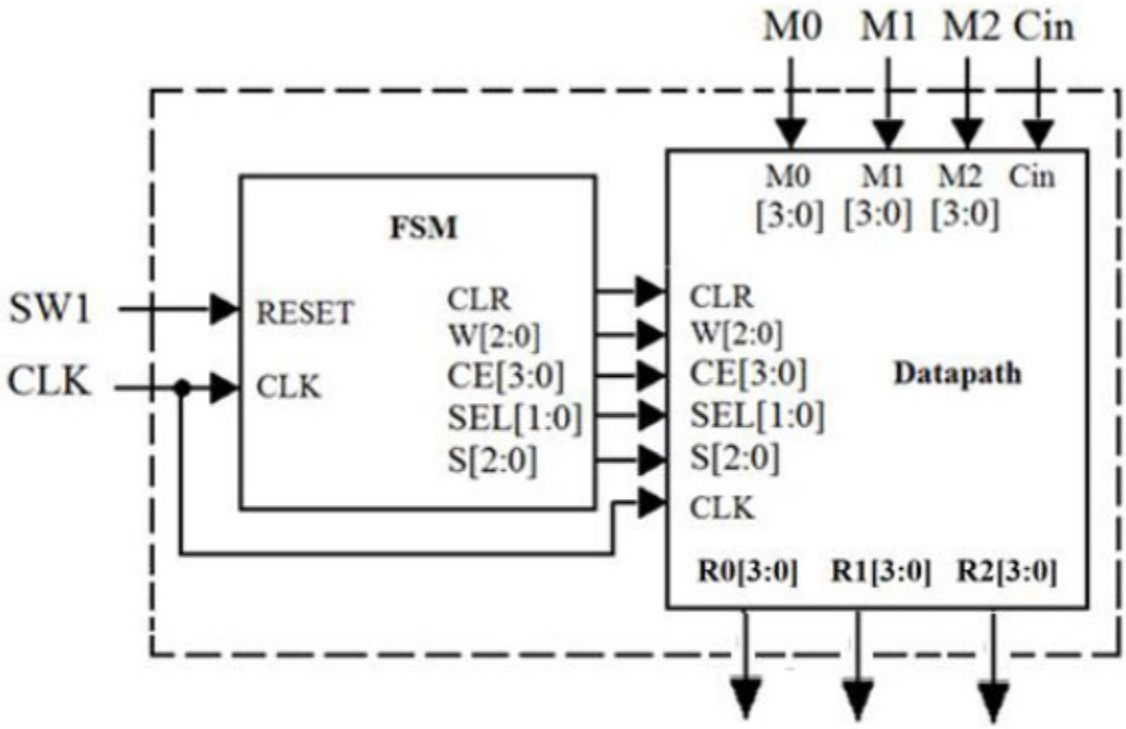


Figure 4-1. Simplified microprocessor block diagram

Here is the block diagram for the top design.

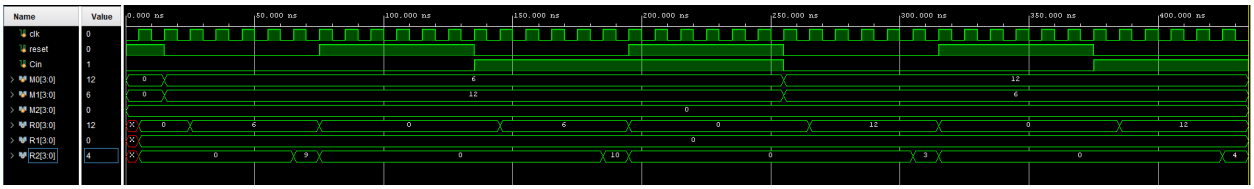
Verilog Design:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/18/20
// Module Name: top
// Description: top level design of 4-bit processor
// Project Part Number: Lab 4, Part 4
//////////////////////////////////////////////////////////////////////////////////


module top(CLK, SW1, M0, M1, M2, Cin, R0, R1, R2);
    input CLK, SW1, Cin;
    input[3:0] M0, M1, M2;
    output[3:0] R0, R1, R2;
    wire CLR;
    wire[2:0] W;
    wire[3:0] CE;
    wire[1:0] SEL;
    wire[2:0] S;

    fsm g1(.RESET(SW1), .CLK(CLK), .CLR(CLR), .W(W), .CE(CE), .SEL(SEL), .S(S));
    datapath g2(.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));
endmodule
```

Verilog Testbench Design and Simulation Waveforms:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Author: Anthony Chavez
// Date Last Revised: 11/18/20
// Module Name: top_tb
// Description: top level design of 4-bit processor testbench
// Project Part Number: Lab 4, Part 4
//////////////////////////////////////////////////////////////////////////////////


module top_tb;
    reg clk, reset, Cin;
    reg[3:0] M0, M1, M2;
    wire[3:0] R0, R1, R2;

    top uut(.CLK(clk), .SW1(reset), .M0(M0), .M1(M1), .M2(M2), .Cin(Cin), .R0(R0),
.R1(R1), .R2(R2));
```

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

    initial begin
        reset = 1; Cin = 0; M0 = 4'b0000; M1 = 4'b0000; M2 = 4'b0000;
        #15 reset = 0; Cin = 0; M0 = 4'b0110; M1 = 4'b1100; M2 = 4'b0000;
        #60 reset = 1;
        #60 reset = 0; Cin = 1; M0 = 4'b0110; M1 = 4'b1100; M2 = 4'b0000;
        #60 reset = 1;
        #60 reset = 0; Cin = 0; M0 = 4'b1100; M1 = 4'b0110; M2 = 4'b0000;
        #60 reset = 1;
        #60 reset = 0; Cin = 1; M0 = 4'b1100; M1 = 4'b0110; M2 = 4'b0000;
        #60 $stop;
    end
endmodule
```



This testbench and waveform was used to verify the top design was implemented properly. For the most part, the design is correct.

Result Discussion:
Based on the final waveform results, I am happy with the final product. However, the last two test cases were not the correct results I expected. According to my handwritten calculations and using a calculator in programmer mode, the third test case was supposed to be 5 and the last being 6. I went to my lab instructor's office hours and we tried to debug the program, but we were unsuccessful in finding the error. We both decided to assume the design was functioning properly due to the first two test cases being correct and the logic of the data path circuit and the finite state machine circuit were properly configured.

Conclusion

In part one of this lab I was successfully able to implement the SRAM design. This allowed me to prepare for the simplified 4-bit microprocessor design as I was able to incorporate the same finite state machine design concept. In addition, I was able to apply all the knowledge I gained from Lab 2 and from the lecture to implement the microprocessor design. This felt like a good way to end the lab assignments. Although some test cases didn't display the correct result, I assume the microprocessor is still functioning properly. Overall, I feel the lab was helpful in understanding how to apply the knowledge gained in the previous lab projects to implement more advanced designs.