Sam Lee

CPE 166 – 03

Advanced Logic Design Lab

Wednesday: 5PM - 7:50 PM

Lecture Professor: Jing Pang

Lab Professor: Eric Carmi

Lab #4 Report

Contents

[Introduction 3](#_30j0zll)

[Part 1: Microprocessor Data Path Design 3](#_1fob9te)

[Design Purpose 3](#_3znysh7)

[Engineering Data 3](#_2et92p0)

[Source Code](#_tyjcwt) 7

[Simulation Waveforms](#_3dy6vkm) 12

[Results Discussion](#_lnxbz9) 14

[Part 2 : Microprocessor Control Path Design](#_1t3h5sf) 15

[Design Purpose 1](#_4d34og8)5

[Engineering Data](#_2s8eyo1) 15

[Source Code](#_17dp8vu) 16

[Simulation Waveforms](#_35nkun2) 18

[Results Discussion](#_26in1rg) 18

[Part 3: Final 4-Bit Microprocessor Design](#_1ksv4uv) 18

[Design Purpose](#_44sinio) 19

[Engineering Data](#_2jxsxqh) 19

[Source Code](#_z337ya) 20

[User Constraint File](#_3j2qqm3) 27

[Results Discussion](#_1y810tw) 28

[Conclusion](#_4i7ojhp) 29

# Introduction

A microprocessor is a computer processor that incorporates the functions of a central processing unit on a single integrated circuit (IC). The microprocessor is a multipurpose, clock driven, register based, digital integrated circuit that accepts binary data as input, processes it according to instructions stored in its memory, and provides results as output. Microprocessors contain both combinational logic and sequential digital logic. Microprocessors operate on numbers and symbols represented in the binary number system. For this lab we needed to build a simplified microprocessor in which it would take roughly 6 inputs and would output a value. Specifically, M0, M1, M2 and Cin inputs, one input switch SW1 and also a clock input. The SW1 switch serves as asynchronous reset function. The design needed to implement the following logic equation: R2 = M0 + (not M1) + Cin. Once everything was set up; the final step would be to download the design onto the NEXYS 4DDR board to view the results.

# Part 1: Microprocessor Data Path Design

## Design Purpose:

The purpose of this lab was to design a simplified version a microprocessor using VHDL in which would take in 6 inputs: M0, M1, M2, Cin, SW1, and Clk and one output: dout. The design should implement the following logical equation if it is successful: R2 = M0 + (not M1) + Cin. The logic ‘1’ on “SW1” switch will force the controller to enter its first state. After “SW1” becomes logic ‘0’, the finite state machine will proceed forward automatically into the next state after the rising edge of the clock. To summarize it all up, The LED display will be on if the final R2 result is '1' and will be off if final result R2 is ‘0’.

## Engineering Data:

In simple terms, we can think of the microprocessor design for this lab as being two distinct blocks. One block will be the data path block which is responsible for processing all of the arithmetic that is needed for each distinct input. The second block will be the finite state machine; and it will be responsible for controlling the state of the circuit. As explained before, the logic value ‘1’ will kickstart the controller into entering its first state. Once the logic value from SW1 goes back to ‘0’ it will automatically continue into its next state; until the final value is calculated. Figure 1 shows the detailed datapath circuit broken down into different circuit elements.

## 

## Figure 1: A simplified microprocessor data path circuit.

## A D flip flop tracks the input, making transitions with match those of the input D. The D stands for “data”; this flip-flop stores the value that is on the data line. It can be thought of as a basic memory cell. A D flip-flop can be made from a set/reset flip-flop by tying the set to the reset through an inverter. The result may be clocked.

## Figure 2: D flip-flop truth table and circuit diagram

## 

## 

## MUX 2to1 and MUX 4to1:

## A multiplexer or mux is a common digital circuit used to mix a lot of signals into just one. If you want multiple sources of data to share a single, common data line, you’d use a multiplexer to run them into that line. Multiplexers come in all sorts of shapes and sizes, but they’re all made out of logic gates. Every multiplexer has at least one select line, which is used to select which input signal gets relayed to the output. In a 2-to-1 multiplexer, there’s just one select line. More inputs means more select lines: a 4-to-1 multiplexer would have 2 select lines, an 8-to-1 has 3, and so on.

## Figure 3: MUX 2to1 and MUX 4to1 Schematic and Truth Table

## 

## 

## 

## 

## ALU:

## An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU). Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory. An ALU performs basic arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

## Figure 4: ALU Schematic and Truth Table

## 

## 

## Source Code:

## D Flip Flop Verilog and Testbench Code:

## `timescale 1ns / 1ns

## module dff (clk, clr, d, q, ce);

## 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

## `timescale 1ns / 1ns

## 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 = 1; ce = 1;

## #20 clr = 0;

## #20 ce = 0;

## #40 $stop;

## end

## endmodule

## MUX 2to1 and MUX 4to1 Verilog and Testbench Code:

## `timescale 1ns / 1ps

## module mux2to1(d0, d1, s, y);

## input[3:0] d0, d1;

## input s;

## output reg[3:0] y;

## always@(d0 or d1 or s) begin

## if(s) y = d1;

## else y = d0;

## end

## endmodule

## `timescale 1ns / 1ns

## module mux2to1\_tb;

## reg[3:0] d0, d1;

## reg s;

## wire[3:0] y;

## mux2to1 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

## `timescale 1ns / 1ps

## module mux4to1(d0, d1, d2, d3, sel, y);

## input[3:0] d0,d1,d2,d3;

## input[1:0] sel;

## output 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 / 1ns

## module mux4to1\_tb;

## reg[3:0] d0, d1, d2, d3;

## reg[1:0] sel;

## wire[3:0] y;

## mux\_4to1 u1(d0, d1, d2, d3, sel, 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

## ALU Verilog and Testbench Code: `timescale 1ns / 1ps

## module alu(a, b, s, cin, y);

## input[3:0] a, b;

## input[2:0] s;

## input cin;

## output reg[3:0] y;

## 

## always@(a or b or s or cin or y) 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

## `timescale 1ns / 1ns

## 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 = 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 = 5; 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

## Datapath Verilog and Testbench Code:

## `timescale 1ns / 1ps

## 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,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;

## 

## 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));

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

## 

## mux4to1 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

## `timescale 1ns / 1ps

## 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 = 1'b0; m2 = 4'b0000;

## m0 = 4'b0101; m1 = 4'b0011;

## cin = 1;

## end

## always #1 clk = ~clk;

## 

## initial begin

## #8; clr = 1'b0;

## #2; ce = 4'b0011; sel = 2'b00; s = 3'b010;

## #2; ce = 4'b1011;

## #2; ce = 4'b0011; sel = 2'b01; s = 3'b001;

## #2; ce = 4'b1000;

## #2; ce = 4'b0100; w = 3'b100;

## #2; w = 3'b000; ce = 4'b0000; sel = 2'b00; s = 3'b000;

## #8; $stop;

## end

## endmodule

## Simulation Waveforms:

## Figure 5: This figure is the simulation of the D flip flop.

## 

## Figure 6: This figure is the simulation of the MUX 2to1 and MUX 4to1.

## 

## 

## Figure 7: This figure is the simulation of the ALU.

## 

## Figure 8: This figure is the simulation of the Datapath.

## 

## Results Discussion:

# Part 1 had most of the work for this lab but it was mostly easy work because we had most of the components from previous parts of the lab, making it work together and making sure it worked together properly was the tricky part. 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, and getting the timing to calculate that properly was a bit tricky but after some time we got it done no problems.

# Part 2: Microprocessor Control Path Design

## Design Purpose:

## Once the data path module is built, we can proceed to building the finite state machine. We will design this FSM based on the conditions set. Figure 9 shows the block diagram of the finite state machine with its outputs.

## Engineering Data:

## A **finite state machine** is a computation model that can be implemented with hardware or software and can be used to simulate [sequential logic](https://brilliant.org/wiki/logic-gates/) and some computer programs. This particular finite state machine controls all of the inputs to the other modules. The state machine has 5 states: reset state, initial loading state, F = B (pass through) state, F = A + not B + Cin state, load R2 and enable/loop state. The FSM is just a Moore FSM, each state going directly into another without any consideration for input.

Figure 9: This is the schematic for the FSM block diagram.

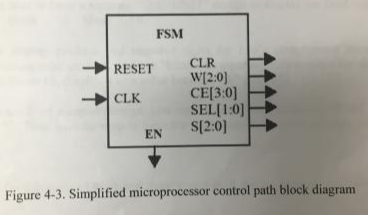
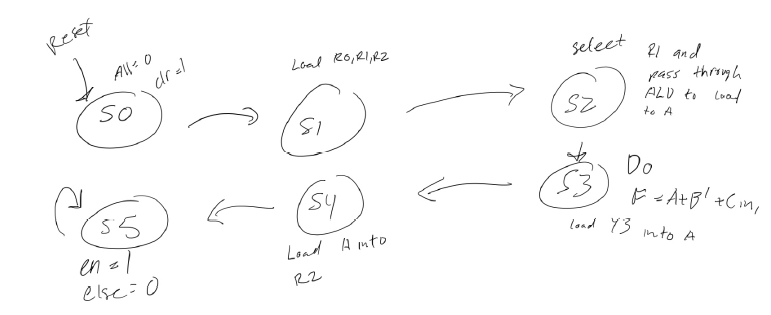


Figure 10: This is the drawing of the FSM that was implemented in order for the simulation to work.



## Source Code:

FSM Verilog and Testbench Code:

`timescale 1ns / 1ps

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

input clk, reset;

output reg en, 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;

en = 1'b0;

w = 3'b000;

s = 3'b000;

sel = 2'b00;

ce = 4'b0000;

ns <= s1;

end

s1 : begin

clr = 1'b0;

en = 1'b0;

w = 3'b101;

s = 3'b000;

sel = 2'b00;

ce = 4'b0111;

ns <= s2;

end

s2 : begin

clr = 1'b0;

en = 1'b0;

w = 3'b100;

s = 3'b010;

sel = 2'b11;

ce = 4'b1000;

ns <= s3;

end

s3 : begin

clr = 1'b0;

en = 1'b0;

w = 3'b100;

s = 3'b001;

sel = 2'b10;

ce = 4'b1000;

ns <= s4;

end

s4 : begin

clr = 1'b0;

en = 1'b0;

w = 3'b000;

s = 3'b000;

sel = 2'b00;

ce = 4'b0100;

ns <= s5;

end

s5 : begin

clr = 1'b0;

en = 1'b1;

w = 3'b000;

s = 3'b000;

sel = 2'b00;

ce = 4'b0000;

ns <= s5;

end

default : ns = s0;

endcase

end

`timescale 1ns / 1ps

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, ce, w, s, sel);

always

begin

#5 clk = ~clk;

end

initial

begin

clk = 0;

#2 reset = 1;

#10 reset = 0;

#150

$stop;

end

endmodule

## Simulation Waveforms:

## Figure 11: Below is the simulation for the finite state machine that was implements for the microprocessor design.

## 

## Results Discussion:

## The FSM works as expected. Even though it took a bit to create, everything works properly: the FSM controls the inputs of the data path to load and do the ALU calculations, and turns on the EN on the last state, continuously staying on the last state. One of the hardest part of just implementing correctly to show the correct inputs and outputs.

## Part 3: Final 4-Bit Microprocessor Design

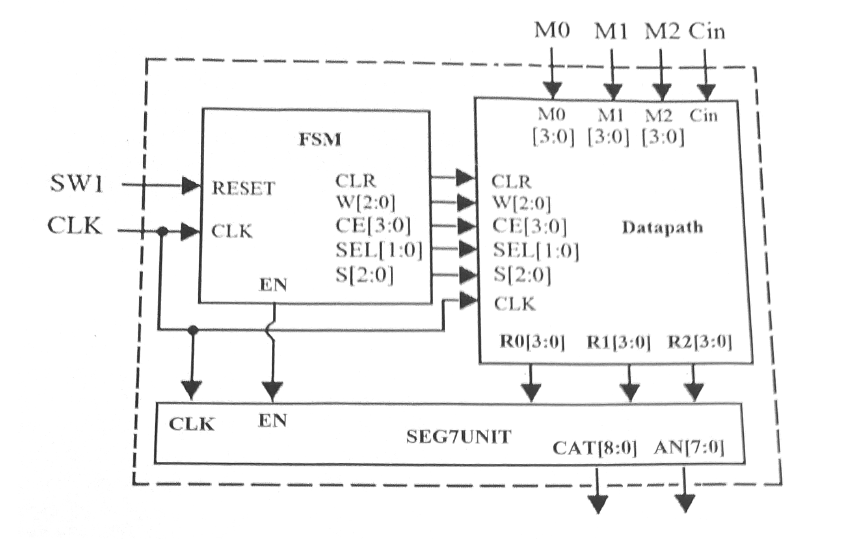
Design Purpose:

For this part of the lab, it requires to implement the top level design with the seven segment display along with the dathpath and finite state machine and implement all of the concepts from part 1 and part 2 for this lab. The final result should be able to display the result using the switches on the FPGA board.

Engineering Data:

For the display to happen we to had create three instances: FSM, datapath, and seg7unit, that were part of the top design.

Figure 13: This figure is the block diagram for the whole design.



The 7-segment display consists of seven LEDs arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a numerical digit to be displayed.

Figure 14: This figure is a common anode seven segment display device.

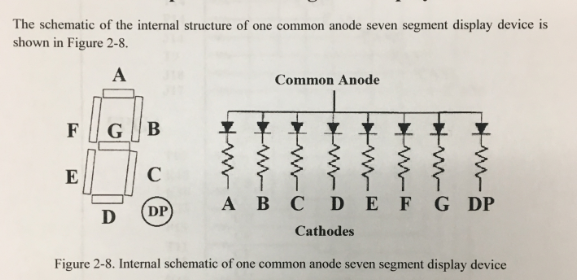


Figure 15: This figure below is the 8-digit seven segment displays on the NEXXYS4 DDR FPGA board.

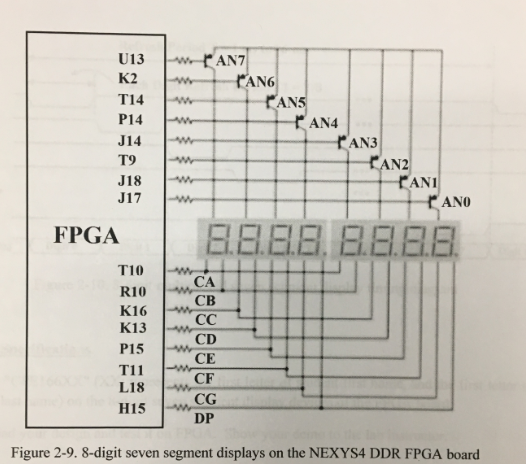


Figure 16: This figure is the seven segment display.

Source Code:

TOP Verilog and Testbench Code:

`timescale 1ns / 1ps

module TOP( clk, reset, m0, m1, m2, cin, seg, dig);

input clk, cin, reset;

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

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

output [7:0] seg, dig;

wire [2:0] w, s;

wire [1:0] sel;

wire [3:0] ce;

wire clr;

wire en;

reg sign2, sign1, sign0;

reg [3:0] num2, num1;

wire [3:0] num0;

//Twos complement controller for m0 and m1.

always@(m0 or m1) begin

if( m0[3] == 1'b1 ) begin

sign2 = 1'b1;

end else begin

sign2 = 1'b0;

end

if( m1[3] == 1'b1 ) begin

sign1 = 1'b1;

end else begin

sign1 = 1'b0;

end

case( m0 )

4'b0000 : num2 = 4'b0000;

4'b0001 : num2 = 4'b0001;

4'b0010 : num2 = 4'b0010;

4'b0011 : num2 = 4'b0011;

4'b0100 : num2 = 4'b0100;

4'b0101 : num2 = 4'b0101;

4'b0110 : num2 = 4'b0110;

4'b0111 : num2 = 4'b0111;

4'b1000 : num2 = 4'b1000;

4'b1001 : num2 = 4'b0111;

4'b1010 : num2 = 4'b0110;

4'b1011 : num2 = 4'b0101;

4'b1100 : num2 = 4'b0100;

4'b1101 : num2 = 4'b0011;

4'b1110 : num2 = 4'b0010;

4'b1111 : num2 = 4'b0001;

default : num2 = 4'b1111;

endcase

case( m1 )

4'b0000 : num1 = 4'b0000;

4'b0001 : num1 = 4'b0001;

4'b0010 : num1 = 4'b0010;

4'b0011 : num1 = 4'b0011;

4'b0100 : num1 = 4'b0100;

4'b0101 : num1 = 4'b0101;

4'b0110 : num1 = 4'b0110;

4'b0111 : num1 = 4'b0111;

4'b1000 : num1 = 4'b1000;

4'b1001 : num1 = 4'b0111;

4'b1010 : num1 = 4'b0110;

4'b1011 : num1 = 4'b0101;

4'b1100 : num1 = 4'b0100;

4'b1101 : num1 = 4'b0011;

4'b1110 : num1 = 4'b0010;

4'b1111 : num1 = 4'b0001;

default : num1 = 4'b1111;

endcase

end

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

Datapath datapath1(clk, clr, w, ce, sel, s, m0, m1, m2, cin, r0, r1, r2);

assign num0 = r2;

Seg7Unit segdig( clk, sign2, sign1, sign0, num2, num1, num0, seg, dig);

endmodule

`timescale 1ns / 1ns

module top\_tb;

// Inputs

reg reset,clk, cin;

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

// Outputs

wire [3:0] R0,R1,R2;

wire [8:0] cat;

wire [7:0] an;

/\*

module top(da, db, dout, clk, rst, start);

\*/

// Instantiate the Unit Under Test (UUT)

// module Top( clk, reset, m0, m1, m2, cin, seg, dig);

Top uut(clk, reset, m0, m1, m2, cin, seg, dig);

//Top uut(reset, clk, m0, m1, m2, cin, cat,an);

initial clk = 0;

always #5 clk = ~clk;

initial begin

reset = 0;

#10;

m0 = 4'b1000;

m1 = 4'b1001;

m2 = 4'b1011;

cin = 1;

reset = 0;

#100;

reset = 1; #10;

#10;

m0 = 1;

m1 = 1;

m2 = 1;

cin = 1;

reset = 0;

#10; $stop;

end

endmodule

Seg7Unit Verilog Code:

module Seg7Unit( clk, sign2, sign1, sign0, num2, num1, num0, seg, dig);

input clk;

input sign2, sign1, sign0;

input [3:0] num2, num1, num0;

output [7:0] seg;

output [7:0] dig;

parameter N = 18;

reg [N-1:0] count;

reg [3:0] dd;

reg [7:0] seg;

reg [7:0] an;

always @ (posedge clk)

begin

count <= count + 1;

case(count[N-1:N-3])

3'b000 :

begin

dd = 4'd7;

an = 8'b11111110;

end

3'b001:

begin

dd = 4'd6;

an = 8'b11111101;

end

3'b010:

begin

dd = 4'd5;

an = 8'b11111011;

end

3'b011:

begin

dd = 4'd4;

an = 8'b11110111;

end

3'b100 :

begin

dd = 4'd3;

an = 8'b11101111;

end

3'b101:

begin

dd = 4'd2;

an = 8'b11011111;

end

3'b110:

begin

dd = 4'd1;

an = 8'b10111111;

end

3'b111:

begin

dd = 4'd0;

an = 8'b01111111;

end

endcase

end

assign dig = an;

always @ (dd)

begin

seg[7] = 1'b1;

case(dd)

4'd0 : begin

if( sign2 == 1'b1) begin

seg[6:0] = 7'b0111111; //to display F

end else begin

seg[6:0] = 7'b1111111;

end

end

4'd1 : begin

case( num2 )

4'b0000 : seg[6:0] = 7'b1000000; //0

4'b0001 : seg[6:0] = 7'b1111001; //1

4'b0010 : seg[6:0] = 7'b0100100; //2

4'b0011 : seg[6:0] = 7'b0110000; //3

4'b0100 : seg[6:0] = 7'b0011001; //4

4'b0101 : seg[6:0] = 7'b0010010; //5

4'b0110 : seg[6:0] = 7'b0000010; //6

4'b0111 : seg[6:0] = 7'b1111000; //7

4'b1000 : seg[6:0] = 7'b0000000; //8

4'b1001 : seg[6:0] = 7'b0011000; //9

4'b1010 : seg[6:0] = 7'b0001000; //A

4'b1011 : seg[6:0] = 7'b0000011; //b

4'b1100 : seg[6:0] = 7'b1000110; //c

4'b1101 : seg[6:0] = 7'b0100001; //d

4'b1110 : seg[6:0] = 7'b0000110; //E

4'b1111 : seg[6:0] = 7'b0001110; //F

default : seg[6:0] = 7'b1111111;

endcase

end

4'd3 : begin

if( sign1 == 1'b1) begin

seg[6:0] = 7'b0111111; //to display F

end else begin

seg[6:0] = 7'b1111111;

end

end

4'd4 : begin

case( num1 )

4'b0000 : seg[6:0] = 7'b1000000; //0

4'b0001 : seg[6:0] = 7'b1111001; //1

4'b0010 : seg[6:0] = 7'b0100100; //2

4'b0011 : seg[6:0] = 7'b0110000; //3

4'b0100 : seg[6:0] = 7'b0011001; //4

4'b0101 : seg[6:0] = 7'b0010010; //5

4'b0110 : seg[6:0] = 7'b0000010; //6

4'b0111 : seg[6:0] = 7'b1111000; //7

4'b1000 : seg[6:0] = 7'b0000000; //8

4'b1001 : seg[6:0] = 7'b0011000; //9

4'b1010 : seg[6:0] = 7'b0001000; //A

4'b1011 : seg[6:0] = 7'b0000011; //b

4'b1100 : seg[6:0] = 7'b1000110; //c

4'b1101 : seg[6:0] = 7'b0100001; //d

4'b1110 : seg[6:0] = 7'b0000110; //E

4'b1111 : seg[6:0] = 7'b0001110; //F

default : seg[6:0] = 7'b1111111;

endcase

end

4'd6 : begin

if( sign0 == 1'b1) begin

seg[6:0] = 7'b0111111; //to display F

end else begin

seg[6:0] = 7'b1111111;

end

end

4'd7 : begin

case( num0 )

4'b0000 : seg[6:0] = 7'b1000000; //0

4'b0001 : seg[6:0] = 7'b1111001; //1

4'b0010 : seg[6:0] = 7'b0100100; //2

4'b0011 : seg[6:0] = 7'b0110000; //3

4'b0100 : seg[6:0] = 7'b0011001; //4

4'b0101 : seg[6:0] = 7'b0010010; //5

4'b0110 : seg[6:0] = 7'b0000010; //6

4'b0111 : seg[6:0] = 7'b1111000; //7

4'b1000 : seg[6:0] = 7'b0000000; //8

4'b1001 : seg[6:0] = 7'b0011000; //9

4'b1010 : seg[6:0] = 7'b0001000; //A

4'b1011 : seg[6:0] = 7'b0000011; //b

4'b1100 : seg[6:0] = 7'b1000110; //c

4'b1101 : seg[6:0] = 7'b0100001; //d

4'b1110 : seg[6:0] = 7'b0000110; //E

4'b1111 : seg[6:0] = 7'b0001110; //F

default : seg[6:0] = 7'b1111111;

endcase

end

default : seg[6:0] = 7'b1111111; //blank

endcase

User Constraints File:

## Clock signal

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports {clk}]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk}];

##Switches

#m1

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { m0[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { m0[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { m0[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { m0[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

#m0

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { m1[0] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { m1[1] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { m1[2] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { m1[3] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

#m2

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { m2[8] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { m2[9] }]; #IO\_25\_34 Sch=sw[9]

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { m2[10] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { m2[11] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

#cin

set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { cin }]; #IO\_L24P\_T3\_35 Sch=sw[12]

#reset

set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { sw1 }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

##7 segment display

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { seg[0] }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { seg[1] }]; #IO\_25\_14 Sch=cb

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { seg[2] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { seg[3] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { seg[4] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { seg[5] }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { seg[6] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { seg[7] }]; #IO\_L19N\_T3\_A21\_VREF\_15 Sch=dp

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { dig[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { dig[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { dig[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { dig[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { dig[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { dig[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

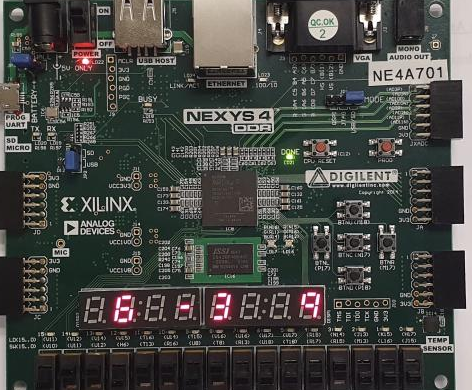
set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { dig[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { dig[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

Results Discussion:

This part of the lab was the hardest part because of the calculation errors that we kept getting, everything integrated properly in the end. The values add just like we would expect from the given examples, exporting it onto the board worked without issues and our seven segment output with a two's compliment representation of numbers works properly as well. The results above show M0 = 6, M1 = -3 (1101), and Cin = 1 adding up to 9, just like the example in the lab manual.

Figure 17: The figure is the final display of the design.



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. Once the modules were built, using the NEXYS 4DDR, I was able to verify that the design was working correctly. Overall, the objectives for this lab were successfully completed.

## 