**San Jose State University**

**Department of Computer Engineering**

**CMPE 125**

**===================================================**

**Lab 7 Report**

**System-level Design**

**Date: 11/18/2012**

**by**

**Name: Christian Moreano SID: 007029269**

**Name: Matt Balhorn SID: 006085014**

**Lab Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tasks** | **Designed by (print name)** | **Verified by (print name)** | **\*Completion Status** |
| **1** | **Christian** | **Matthew** | **A** |
| **2** | **Matthew** | **Christian** | **A** |
| **3** | **Matthew** | **Christian** | **A** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Performed by (print name)** | **Validated by (print name)** | **\*Completion Status** |
| **4** | **Christian** | **Matthew** | **A** |

**\* Enter the following:**

**A – if the task was successfully completed**

**B – if the task was partially completed**

**X – if the task was failed or not performed**

**If you entered B or X, detailed description about the incompletion or failure must be given in the report.**

**CMPE 125 Lab #7 Report**

**Christian Moreano, Matthew Balhorn**

*Computer Engineering Department, College of Engineering*

*San Jose State University, San Jose, CA 95192*

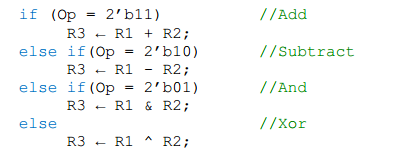
*E-mail:* xtian071989@gmail.com

**1. Introduction**

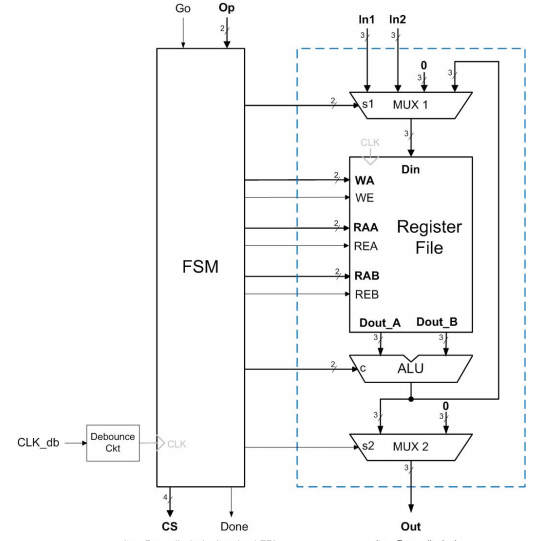
The object of this lab is to construct a simple calculator that performs the following operations: add , subtract , AND , and XOR. The design consists of a datapath , and a control unit. the short design description as follows: when idling , the calculator watches for the GO signal , and once the signal is detected , the calculator will do the following:

1) Load 3 bit inputs operands In1 , and In2 into the register file

2) Depending on the 2 bit operator control input do the following:



3) Output the 3 bit result Out , along with the flag signal Done, then return to idle mode, and wait for the Go signal again.

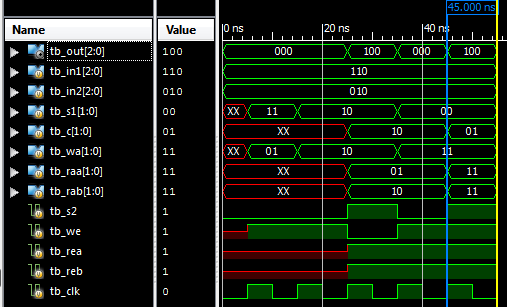


**Figure 0 : Design Schematic**

This lab was design based on the above schematic.

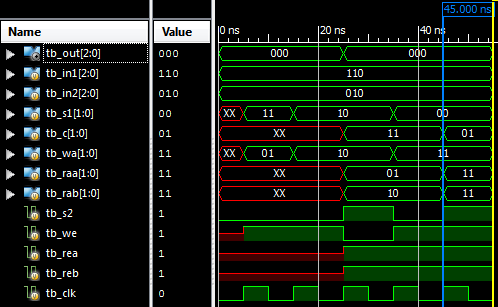
**2. Datapath**

The datapath consists of two multiplexers , a register file , and an ALU , as shown in the *Figure ?: Design Schematic.* To successfully execute and obtain an output from the datapath , we need the following control inputs: In1, In2 , s1 , WA, WE , RAA , REA , RAB , REB , c , s2 in order to receive our output named Out. In the figure , the first multiplexer allows data selection with s1 (2 bits) , and its output will go to the register file. In the register file , we have registers R0 through R3 , here we can either write to data to a register with WA , or read from register using with RAA/B , we can also disable or enable write operations with WE , and disable/enable read operations with REA/B.



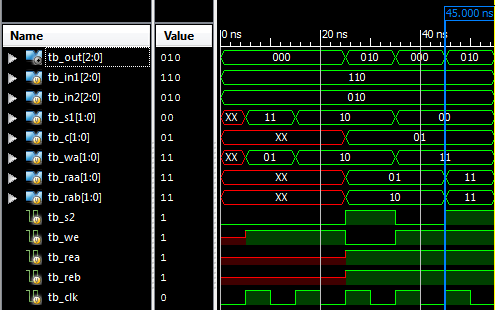
**Figure 1 : Datapath simulation using iSim, Subtraction (Christian Moreano)**

For the above simulation , our inputs are as follows tb\_in1 = 110 , tb\_in2 = 010. In the first rising edge clock , we set tb\_s1 = 11 , this allows tb\_in1 to go through to the register file and get written into R1 (11) , which is done by enabling WE , and setting WA to 11. Second rising edge clock , we set tb\_s1 = 10 , this allows tb\_in2 to go through to the register file and get written into R2 (10) , which is done by enabling WE , and setting WA to 10. Third rising edge clock , we enable REA , REB , and read data at location RAA = 01 and RAB = 10 and send them as inputs for the ALU , here our AUL operator is manipulated by the signal tb\_c , and the signal for the above simulation is 10 , which is for subtraction. ( We enabled tb\_s2 for showing purposes) Then the ALU output is routed back to first MUX , and now our select tb\_s1 changes to 00 to allow the ALU output to go to the register , where it gets written to register R3 , and gets read by RAA and RAB and then sent to ALU with tb\_c as 01 ( AND ) operation, and finally tb\_s2 is enabled to allow the output to go through. Below are the results of the other operations.



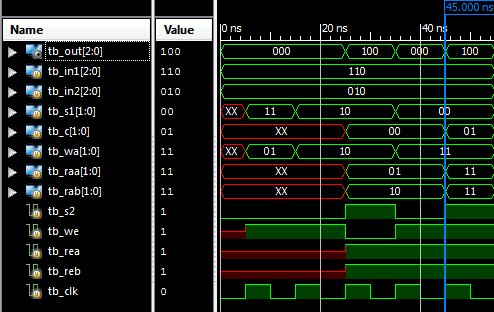
**Figure 2 : Datapath simulation using iSim, Addition (Christian Moreano)**

In the above waveform 110 and 010 are our In1 , and in2 , with the operation of 11 , which is addition, with a corresponding output of 000 at the final stage that starts at 45 ns.



**Figure 3 : Datapath simulation using iSim, AND (Christian Moreano)**

In the above waveform 110 and 010 are our In1 , and in2 , with the operation of 01 , which is AND, with a corresponding output of 010 at the final stage that starts at 45 ns.



**Figure 4 : Datapath simulation using iSim, XOR (Christian Moreano)**

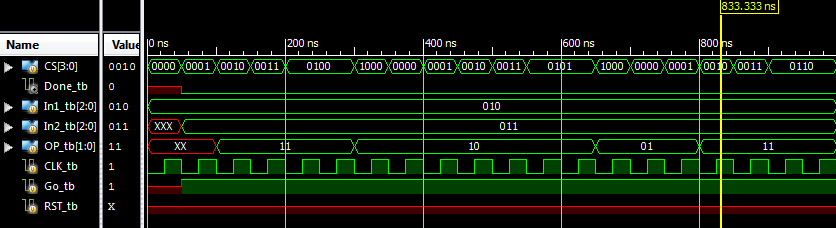
In the above waveform 110 and 010 are our In1 , and in2 , with the operation of 00 , which is XOR, with a corresponding output of 100 at the final stage that starts at 45 ns.

**3. Finite State Machine (Control Unit)**

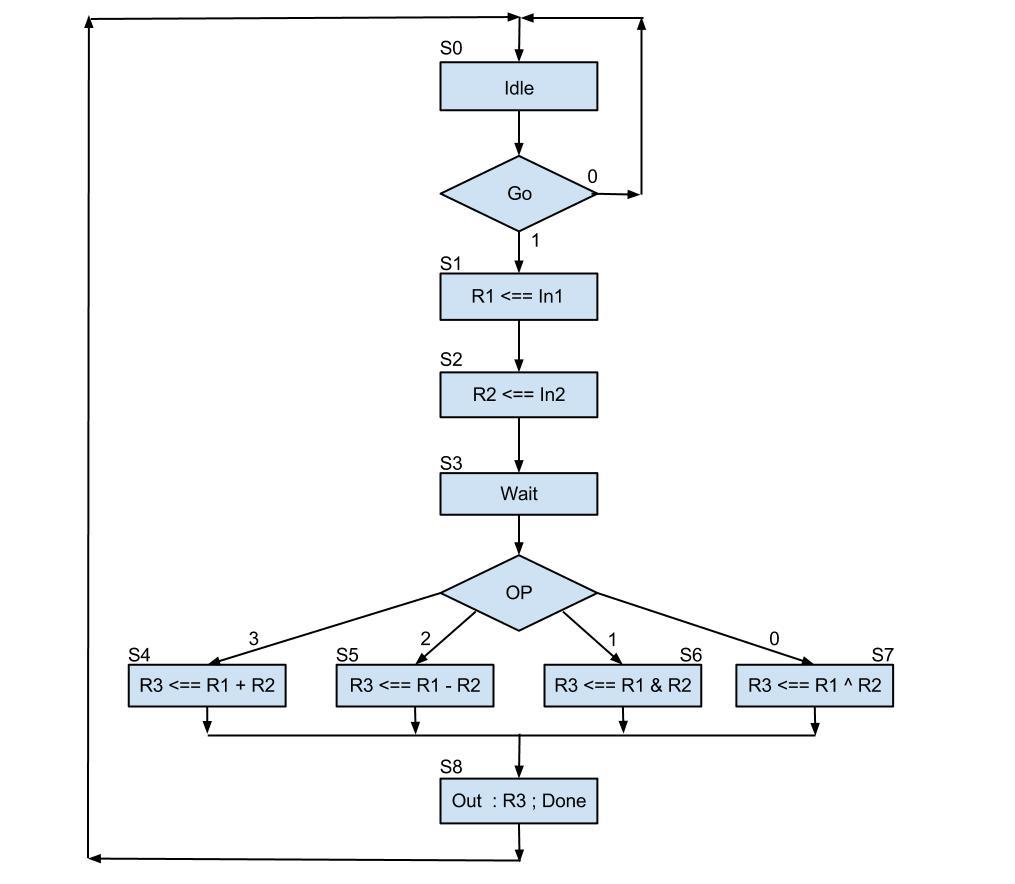
The Control Unit is a Finite State Machine which takes a clock signal, 2 inputs, Go, which is the enable signal that allows the state machine to get out of the initial state, and OP, which is a 2-bit control code that picks the operation on the data inputs in the Data Path. On each state, the Control Unit changes the control signals to accommodate for the OP control signal. These controls are as follows:

|  |  |
| --- | --- |
| State | Control signals |
| State 0 (initial state) |  |
| State 1 | s1 = 00; WE = 1; WA = 01; |
| State 2 | s1 = 01; WE = 1; WA = 10; |
| State 3 | Wait State; If OP = 11, S4; If OP = 10, S5; If OP = 01, S6; If OP = 00, S7; |
| State 4 | RAA = 01, REA = 1, RAB = 10, REB = 1, s1 = 11, WE = 1 WA = 11; C = 11 |
| State 5 | RAA = 01, REA = 1, RAB = 10, REB = 1, s1 = 11, WE = 1 WA = 11; C = 10 |
| State 6 | RAA = 01, REA = 1, RAB = 10, REB = 1, s1 = 11, WE = 1 WA = 11; C = 01 |
| State 7 | RAA = 01, REA = 1, RAB = 10, REB = 1, s1 = 11, WE = 1 WA = 11; C = 00 |
| State 8 | Done = 1; WE = 0; REA = 0; REB = 0; s2 = 1; |

The waveform for the verification of the FSM follows:



**Figure 5: FSM waveform with each of the 4 OP codes**

The waveform shows the operation of the FSM with each of the 

**Figure 6 : ASM Chart**

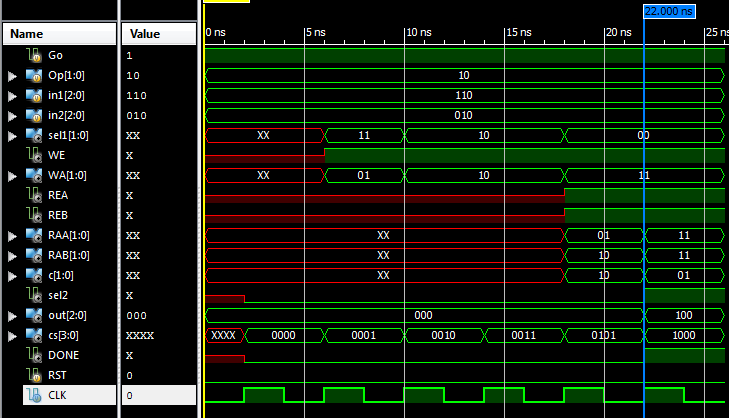
The above chart describes the state transition and their respective outputs

**4. Validations - Setup & Test Plan**

Before proceeding , we used iSim to simulate and see our testbench results. Below are waveforms generated for all four operators , + , - , & and XOR. Observe the the current state transitions ( cs ) , also notice that at 22 ns reading from R3 to the ALUE starts , and an output is displayed.



**Figure 7: FSM/Datapath Showing Addition Operation (Christian Moreano)**



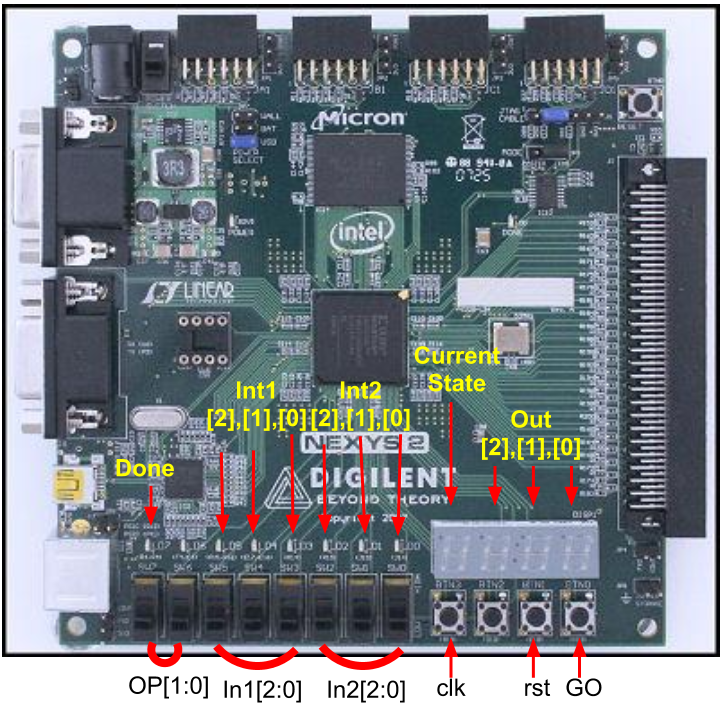
**Figure 8: FSM/Datapath Showing Subtraction Operation (Christian Moreano)**



**Figure 9: FSM/Datapath Showing AND Operation (Christian Moreano)**

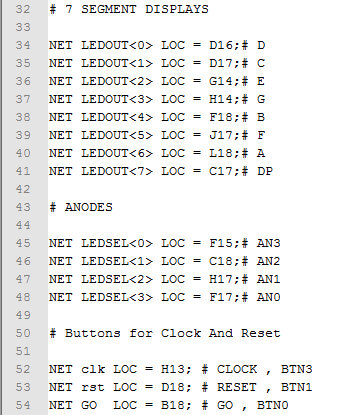
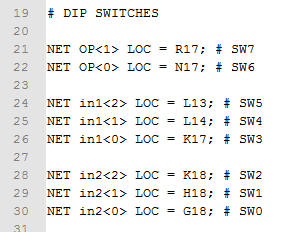
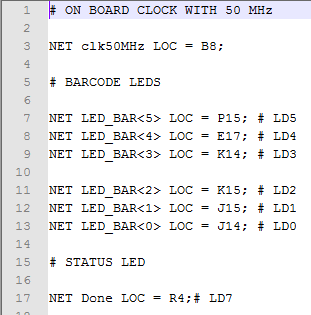


**Figure 10: FSM/Datapath Showing XOR Operation (Christian Moreano)**



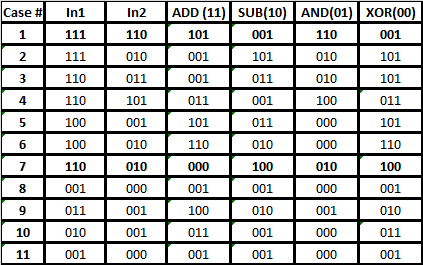
**Figure 11: Board Set-Up**

The above is a picture of the NEXYs 2 board , here we point out the setup for this lab ,



**Figure 12a : Barcode LED Figure 12b : Switches Figure 12c: Display & Buttons**

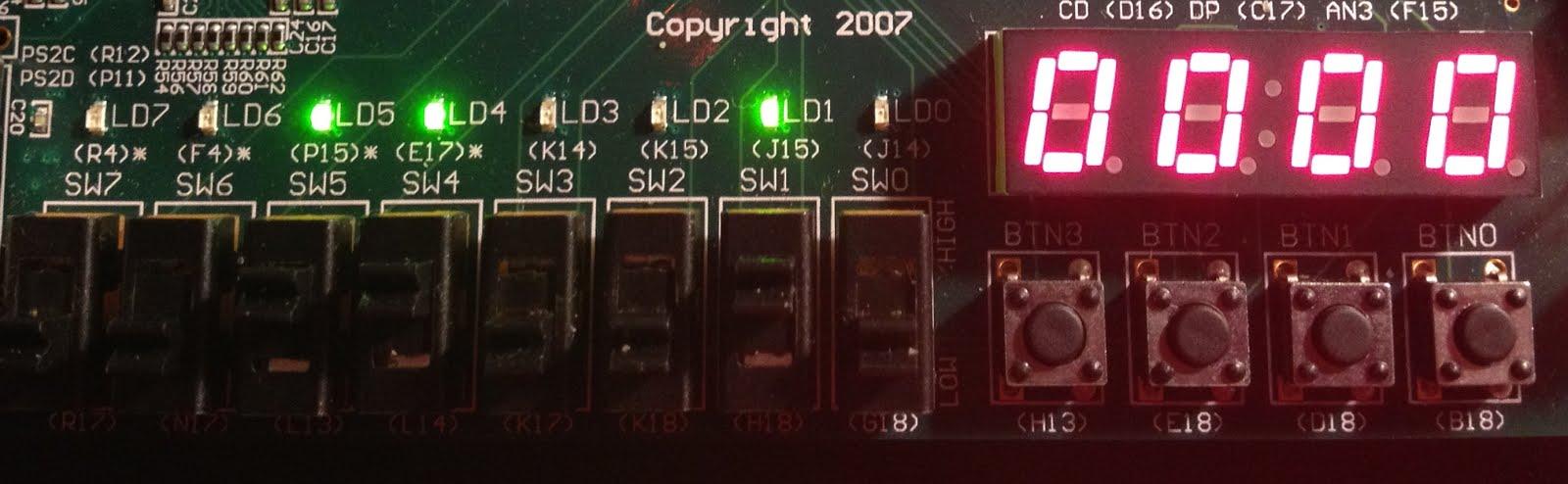
The three figures above shows to which port is each input and output mapped into the NEXYS 2.



**Table 1 : Test Cases**

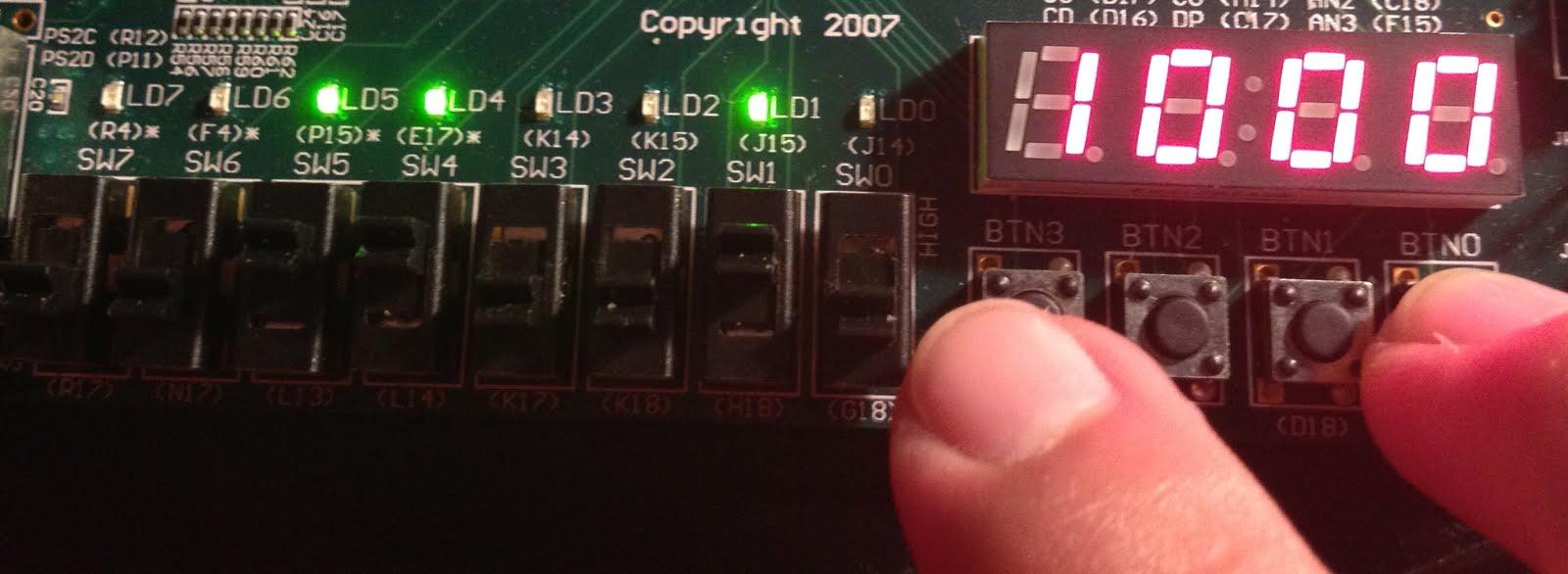
The above table shows the cases that were created for this lab , and have been verified and reinforced by our hardware verification using the NEXYS 2 , and the verilog modules created for this lab. Upon synthesising and creating the bit programming file , Digilent’s Adept software was used to program the board with the generated bit file.Below are the results of this experiment.

**Testing Case # 7**



**Figure 13 : Case 7 State 0**

First we feed out two binary inputs using the switches , upon pressing the clock button ( BTN3) , theres no state transition.



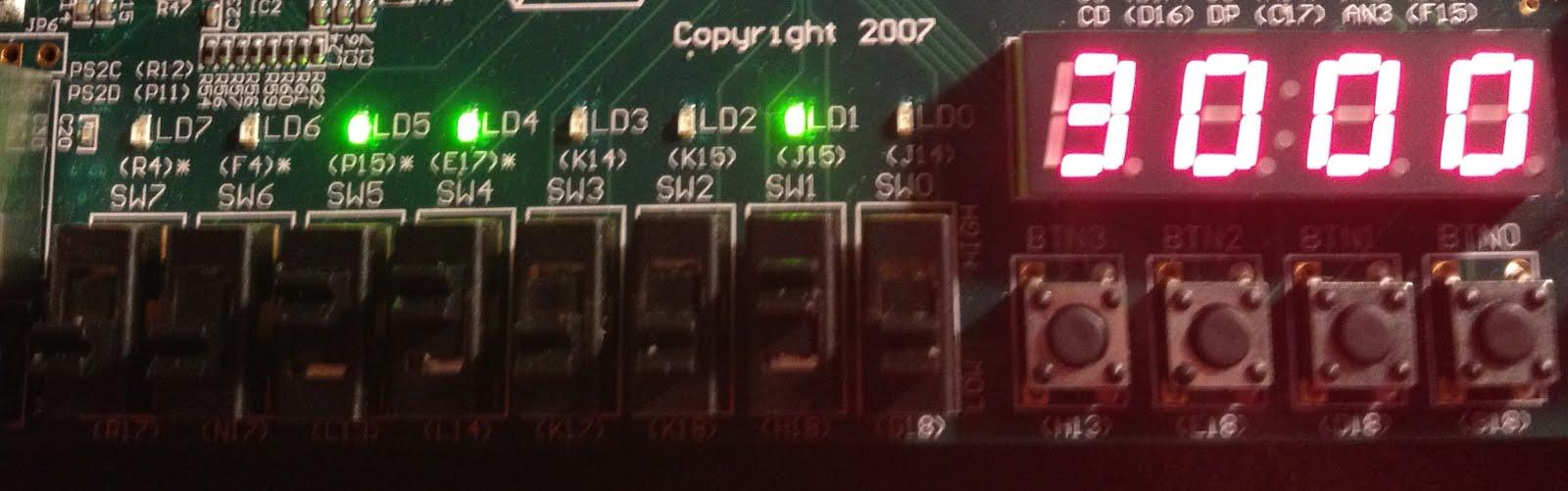
**Figure 13a : Case 7 State 1**

While holding the GO button ( BTN0) , the clock button is pressed ( BTN3) for a state transition from S0 to S1.Where In1 (110) is written into R1 register.



**Figure 13b : Case 7, State 2**

Internally , at S2 , the second binary value ( 010 ), In2 is written into the R2 register



**Figure 13c : Case 7 ,State 3**

Theres nothing interesting at S3.



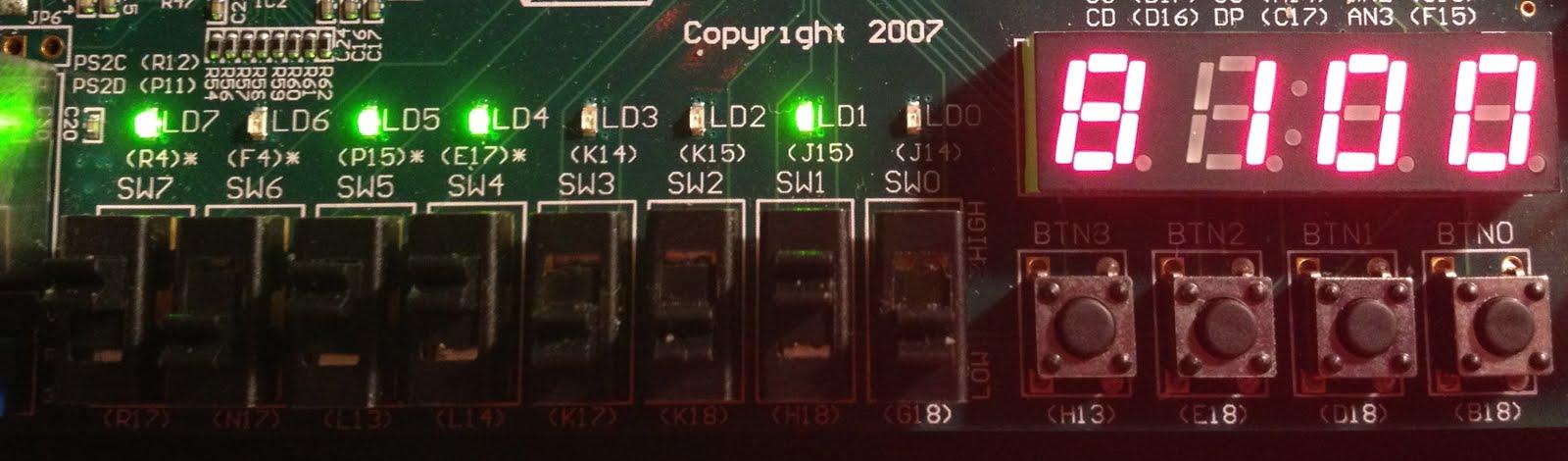
**Figure 13d : Case 7 ,State 7**

Since the OP chosen is XOR , the output is routed back internally , and written into R3



**Figure 13e : Case 7 State 8 , XOR**

At this state , we simply read from R3 and AND with itself , to display the result , also notice the Done status LED.



**Figure 13f : Case 7 State 8 , SUBTRACTION**

Above is another result , in this case , the operator is subtraction ( 10 )



**Figure 13g : Case 7 State 8 , AND**

Above is another result , in this case , the operator is AND ( 01 )



**Figure 13h : Case 7 State 8 , ADD**

Above is another result , in this case , the operator is ADD ( 11 )

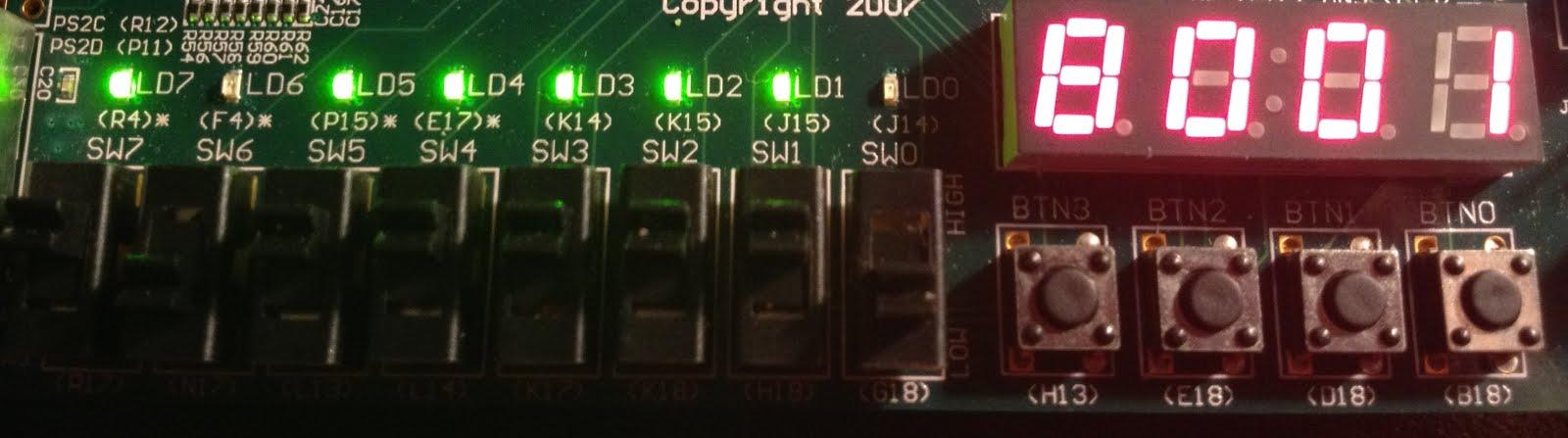
**Testing Case # 1**

Heres another hardware validation , at S8 level . with In1 = 111 , and In2 = 1110.



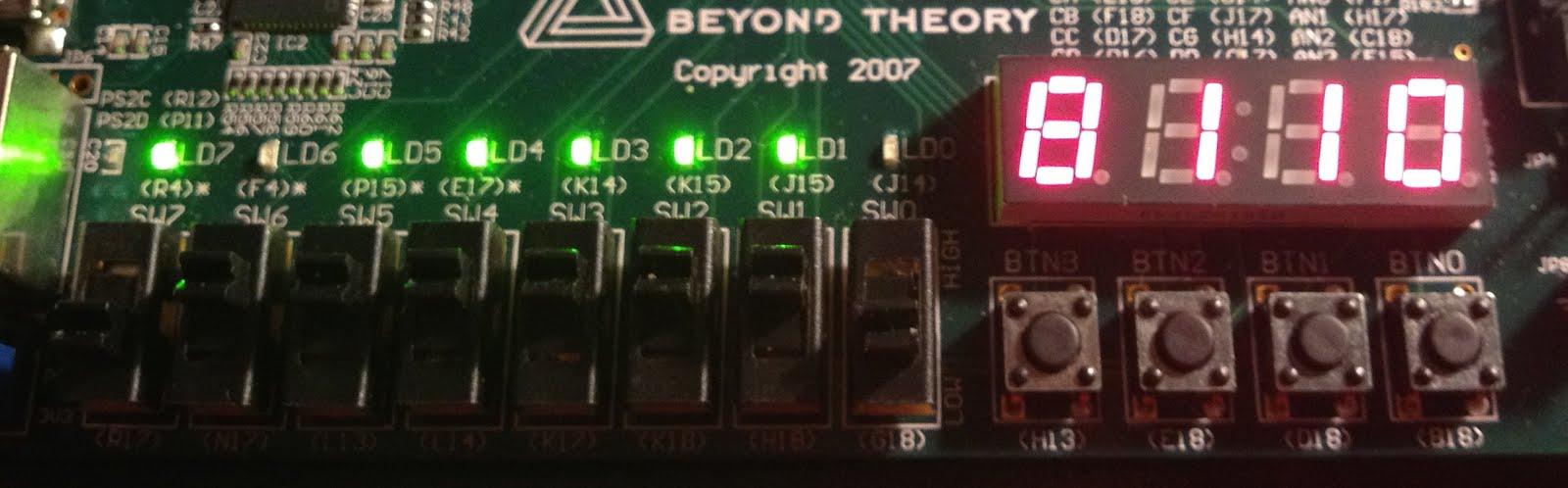
**Figure 14 : Case 1 State 8 , ADD**

We can observe in the above that In1 = 111 , and In2 = 110. with the operator ‘+’ , yields 101



**Figure 14a : Case 1 State 8 , SUBTRACT**

We can observe in the above that In1 = 111 , and In2 = 110. with the operator ‘-’ , yields 001



**Figure 14b : Case 1 State 8 , AND**

We can observe in the above that In1 = 111 , and In2 = 110. with the operator ‘&’ , yields 110



**Figure 14c : Case 1 State 8 , XOR**

We can observe in the above that In1 = 111 , and In2 = 110. with the operator ‘XOR’ , yields 001

**5. Conclusion**

In this lab we have learn how to build a simple calculator based on the given data path and control unit.

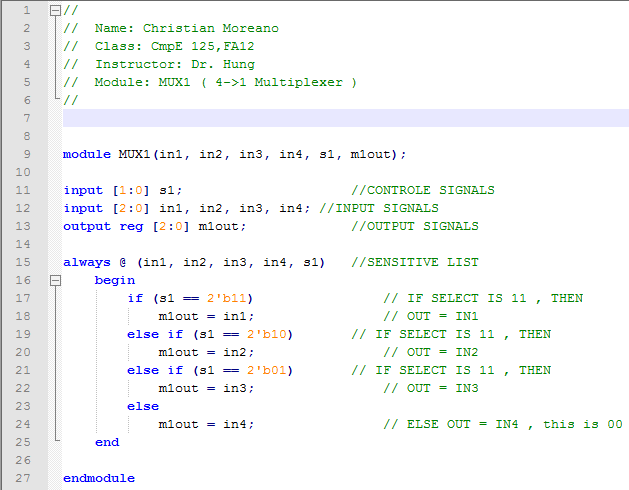
**6. References**

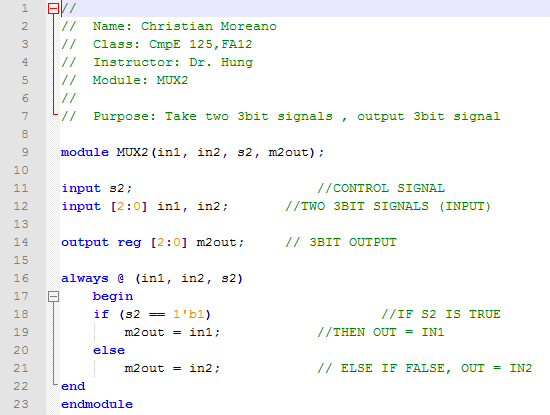
[1] D. Hung, *CMPE 125 Lecture Materials* , Computer Engineering Department, College of Engineering, San Jose State University, 2012.

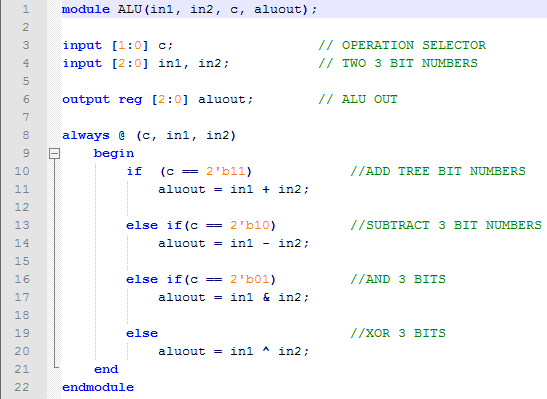
# 7. Appendix

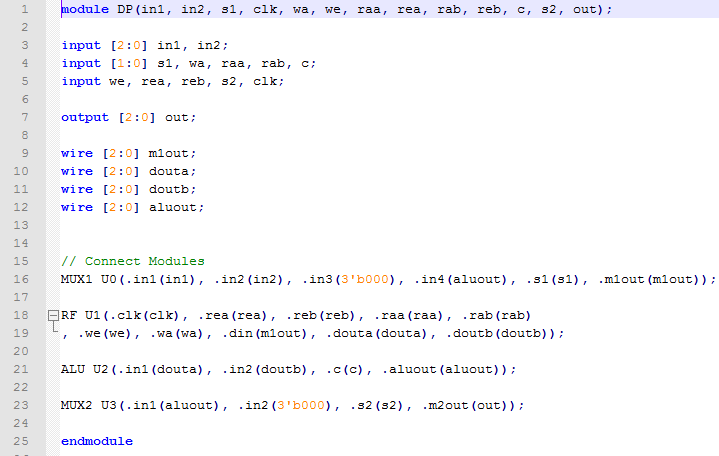
**Source Code: these are in picture format , since traditional copy and paste requires formatting**

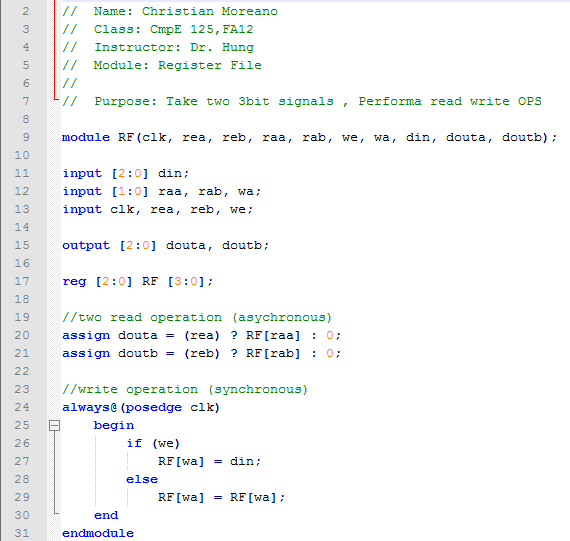
Christian Moreano SC

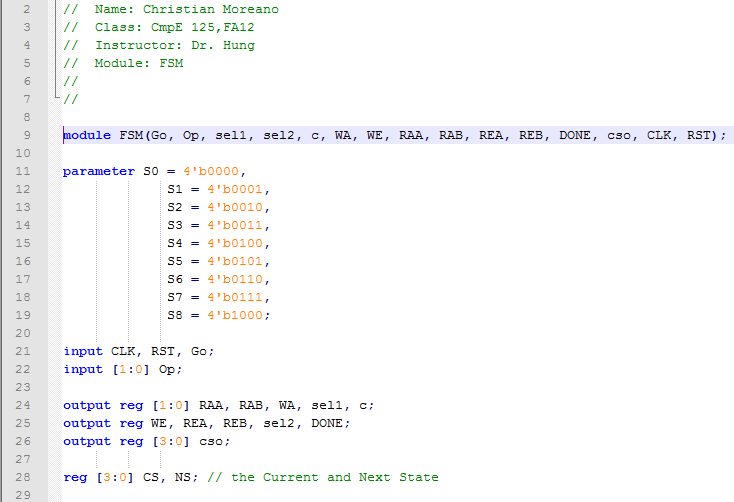


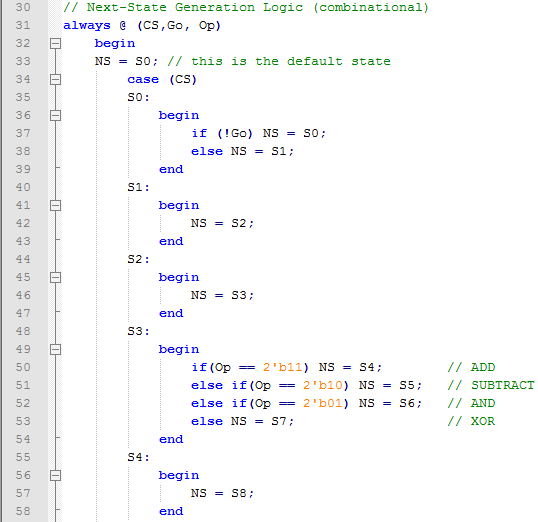


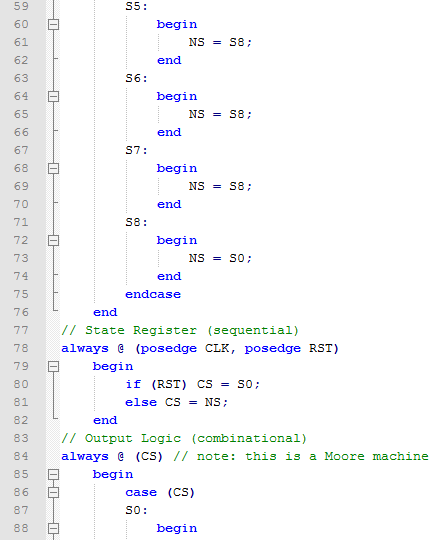


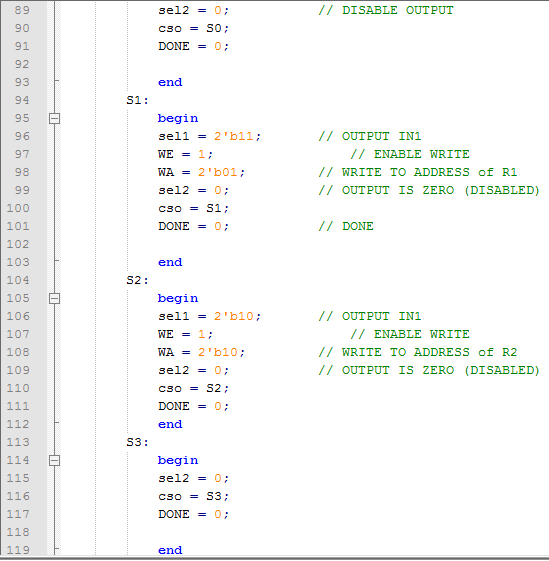


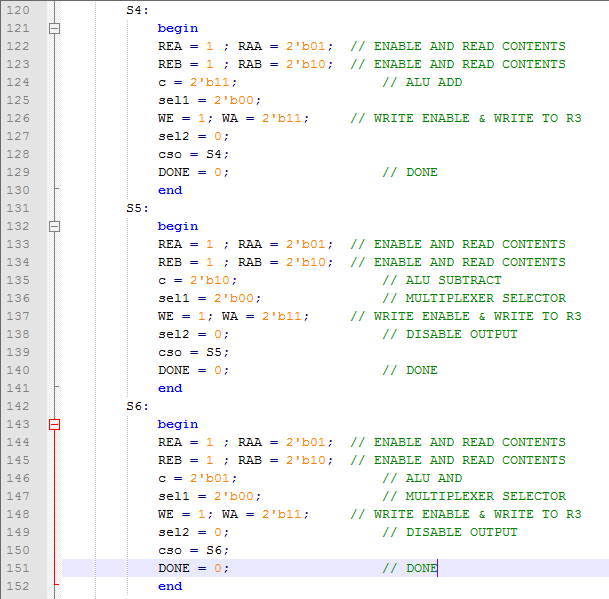


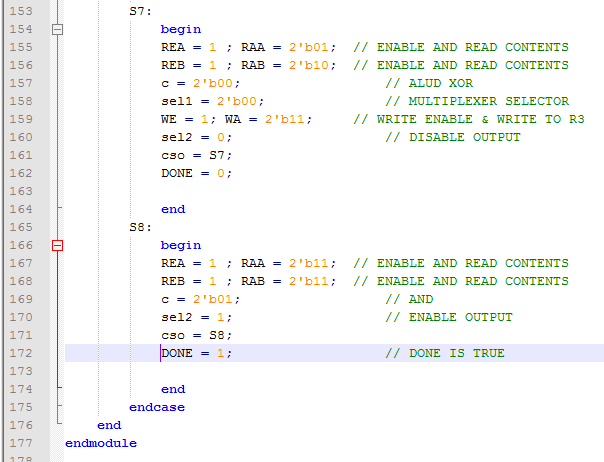




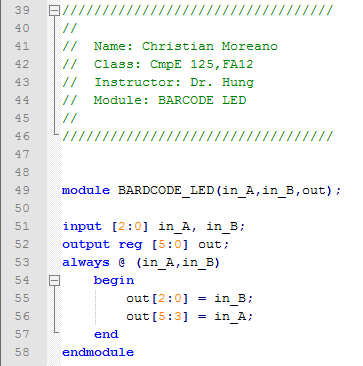


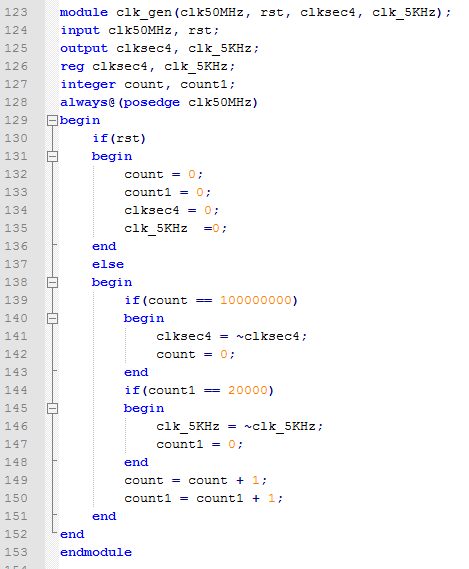


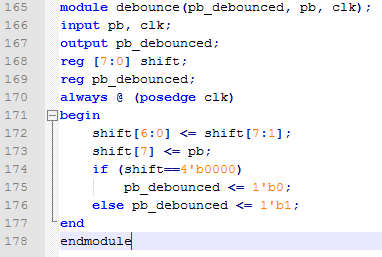


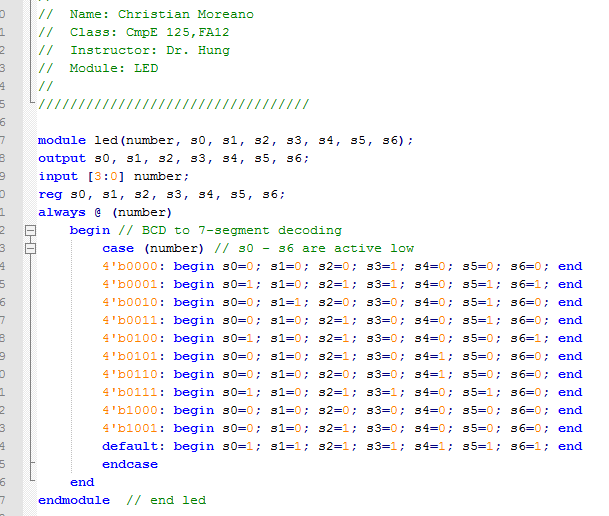


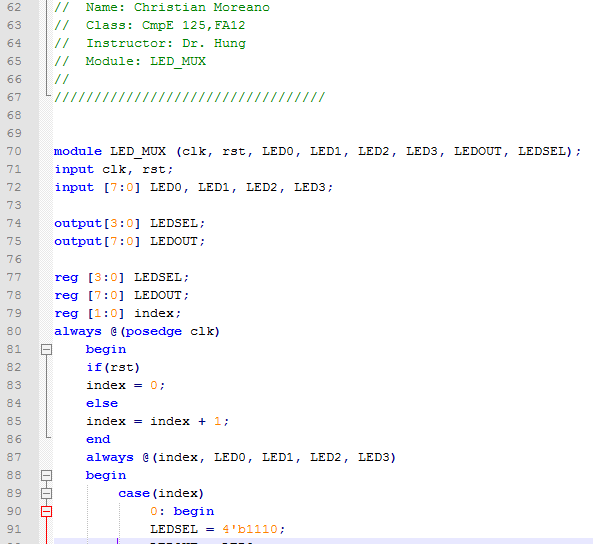
UTILITY MODULES

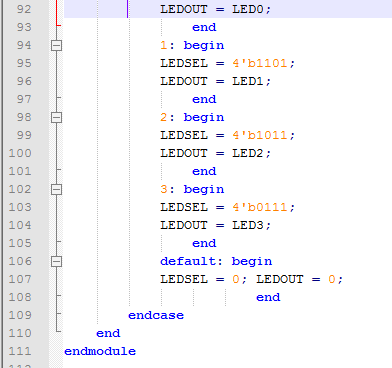






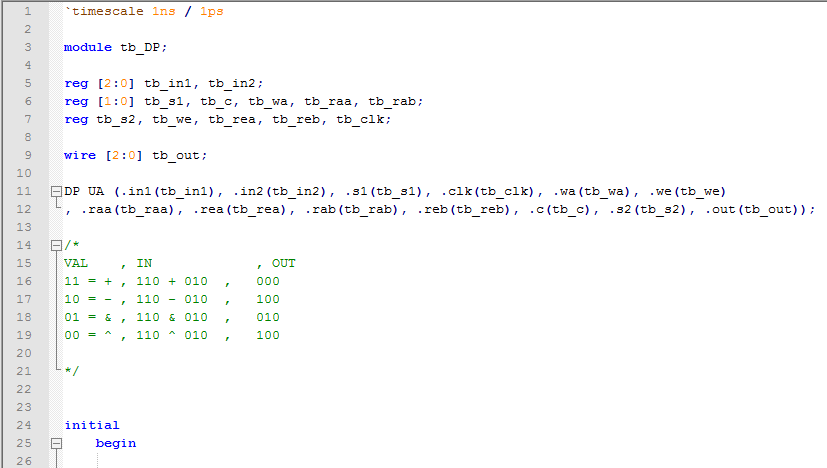


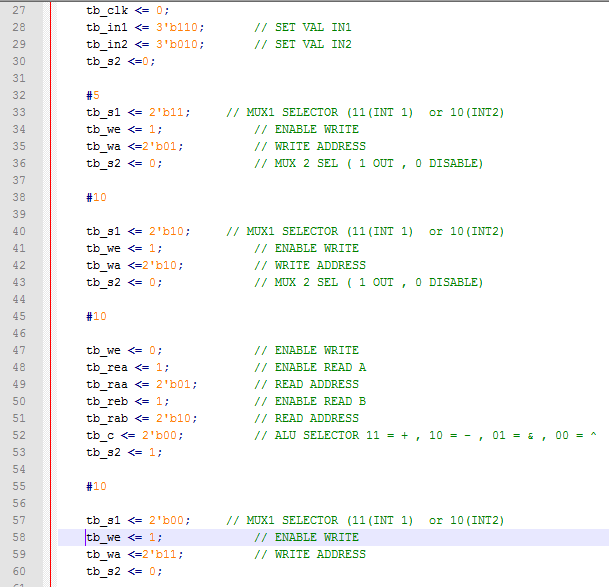


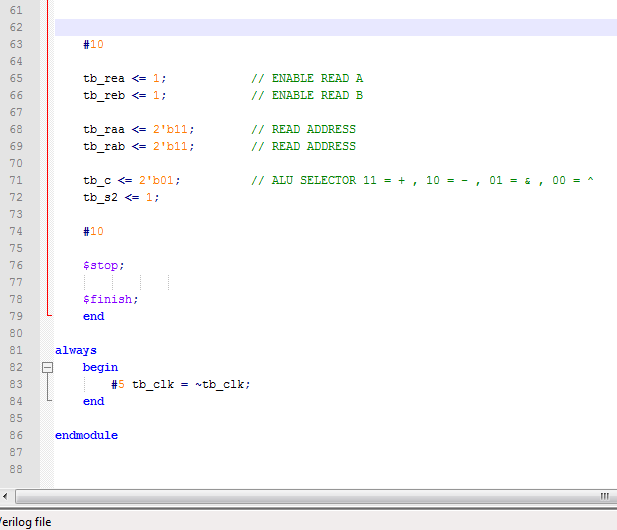


UTILITY MODULE END

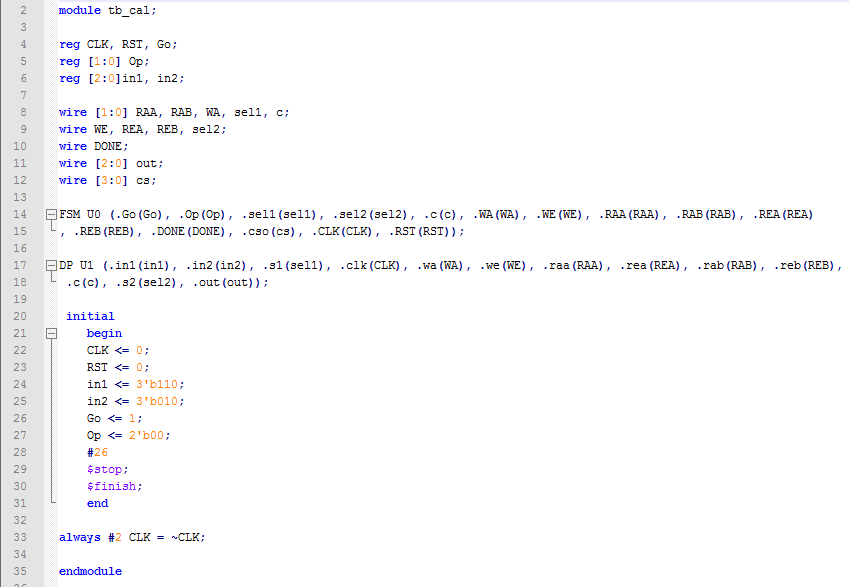
TESTBENCH DP BEGIN

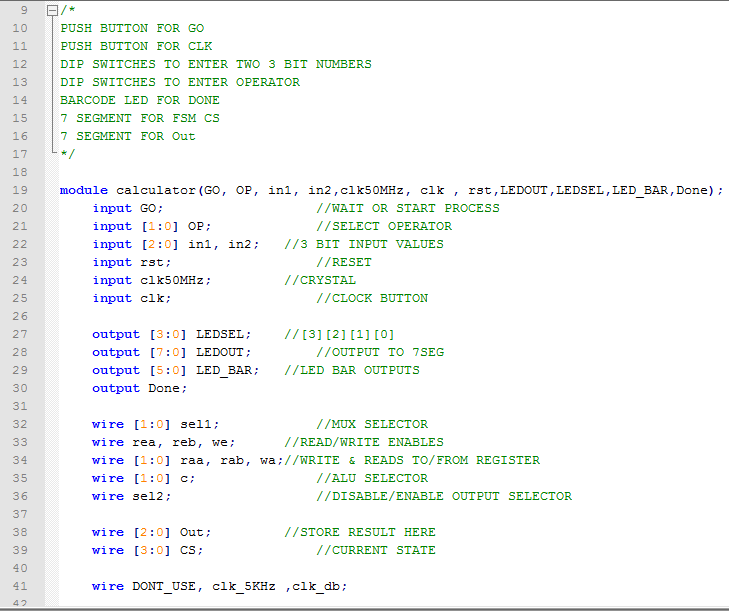


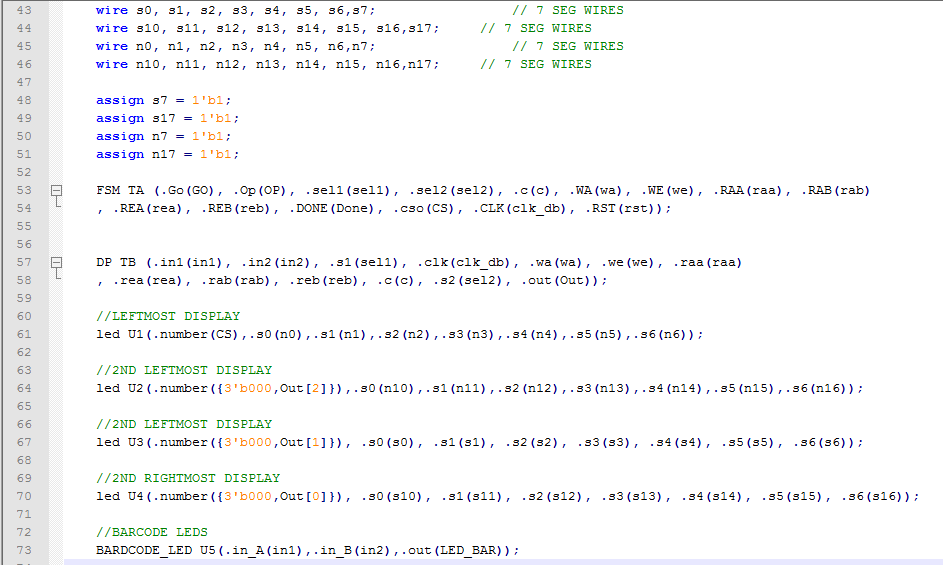


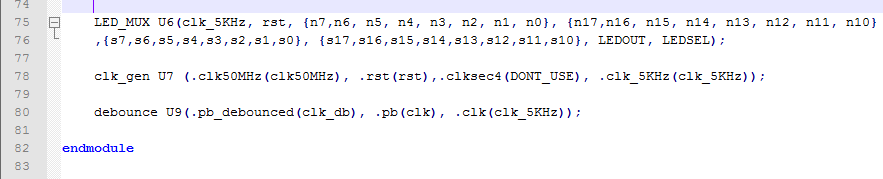


Tb\_CALC









**MATTHEW BALLHORN SC**

**FSM(Control Unit + DataPath integration)**

module FSM(In1,In2,Go,OP,Done,CS,CLK,RST,LEDOut,LEDSel,pb);

input Go,CLK,RST,pb;

input [1:0] OP;

input [2:0] In1,In2;

output reg Done;

output reg [3:0] CS;

output[7:0] LEDOut;

output[3:0] LEDSel;

reg WE,REA,REB,s2;

reg [1:0] WA,RAA,RAB,C,s1;

reg [3:0] NS;

wire Clk\_db,clk\_5KHz,clksec4;

wire [2:0] Out;

always@(Go, OP)

begin

NS = 4'b0000; Done = 0;

case(CS)

4'b0000: begin

if(!Go) NS = 4'b0000;

else NS = 4'b0001;

s2 = 0;

end

4'b0001: begin

NS = 4'b0010;

s1 = 2'b00;

WA = 2'b01;

WE = 1;

s2 = 0;

end

4'b0010: begin

NS = 4'b0011;

s1 = 2'b01;

WA = 2'b10;

WE = 1;

s2 = 0;

end

4'b0011: begin

if(OP == 00) NS = 4'b0111;

else if(OP == 01) NS = 4'b0110;

else if(OP == 10) NS = 4'b0101;

else NS = 4'b0100;

s2 = 0;

end

4'b0100: begin

NS = 4'b1000;

RAA = 2'b01; RAB = 2'b10;

REA = 1; REB = 1;

C = OP; s1 = 2'b00;

WA = 2'b11; WE = 1;

s2 = 0;

end

4'b0101: begin

NS = 4'b1000;

RAA = 2'b01; RAB = 2'b10;

REA = 1; REB = 1;

C = OP; s1 = 2'b00;

WA = 2'b11; WE = 1;

s2 = 0;

end

4'b0110: begin

NS = 4'b1000;

RAA = 2'b01; RAB = 2'b10;

REA = 1; REB = 1;

C = OP; s1 = 2'b00;

WA = 2'b11; WE = 1;

s2 = 0;

end

4'b0111: begin

NS = 4'b1000;

RAA = 2'b01; RAB = 2'b10;

REA = 1; REB = 1;

C = OP; s1 = 2'b00;

WA = 2'b11; WE = 1;

s2 = 0;

end

4'b1000: begin

NS = 4'b0000;

Done = 1;

end

endcase

end

always@(posedge Clk\_db, posedge RST)

begin

if(RST) CS = 4'b0000;

else CS = NS;

end

wire cs0,cs1,cs2,cs3,cs4,cs5,cs6,cs7;

wire o0,o1,o2,o3,o4,o5,o6,o7;

supply1[7:0] vcc;

DP U0(.In1(In1),.In2(In2),.s1(s1),.s2(s2),.WA(WA),.WE(WE),.RAA(RAA),.REA(REA),.RAB(RAB),.REB(REB),.C(C),.Out(Out),.CLK(CLK));

debounce U1(.pb\_debounced(Clk\_db),.pb(pb),.clk(clk\_5KHz));

clk\_gen U3(.clk50MHz(CLK),.rst(RST),.clksec4(clksec4),.clk\_5KHz(clk\_5KHz));

led U4(.number(CS),.s0(cs0),.s1(cs1),.s2(cs2),.s3(cs3),.s4(cs4),.s5(cs5),.s6(cs6));

led U5(.number(Out),.s0(o0),.s1(o1),.s2(o2),.s3(o3),.s4(o4),.s5(o5),.s6(o6));

LED\_MUX U6(.clk(clk\_5KHz),.rst(RST),.LED0({cs7,cs6,cs5,cs4,cs3,cs2,cs1,cs0}),.LED1({o7,o6,o5,o4,o3,o2,o1,o0}),.LED2(vcc),.LED3(vcc),.LEDOUT(LEDOut),.LEDSEL(LEDSel));

endmodule

module FSM\_tb();

reg [2:0] In1\_tb, In2\_tb;

reg [1:0] OP\_tb;

reg CLK\_tb,Go\_tb,RST\_tb;

wire [3:0] CS;

wire Done\_tb;

FSM U0(.In1(In1\_tb),.In2(In2\_tb),.Go(Go\_tb),.OP(OP\_tb),.Done(Done\_tb),.CS(CS),.CLK(CLK\_tb),.RST(RST\_tb));

initial

begin

CLK\_tb = 0;In1\_tb = 3'b010;

#50 Go\_tb = 1; In2\_tb = 3'b011;

#50 OP\_tb = 2'b11;

#50

#50

#50

#50 OP\_tb = 2'b10;

#200 OP\_tb = 2'b01;

#200 OP\_tb = 2'b00;

#50 $stop;

#20 $finish;

end

always

begin

#25 CLK\_tb = ~CLK\_tb;

end

endmodule