=============================================================

**Lab 7 Report**

**System-level Design (1)**

**Date \_\_\_\_4/22/17\_\_\_\_\_\_\_**

**by**

**Name \_\_\_\_Anahit Sarao\_\_\_\_\_\_  SID \_\_008435583\_\_\_\_\_\_**

**Name \_\_\_\_Maxwell Cheshier\_\_\_\_\_\_  SID \_\_009193717\_\_\_**

**Lab Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tasks** | **Designed by (print name)** | **Verified by (print name)** | **\*Completion Status** |
| **1** | Both | Ryan | A |
| **2** | Maxwell | Ryan | A |
| **3** | Anahit | Ryan | A |

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Performed by (print name)** | **Validated by (print name)** | **\*Completion Status** |
| **4** | **N/A** | **Ryan** | **X** |

**\* 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.**

**San Jose State University**

**Department of Computer Engineering**

**CMPE 125 Spring 2017**

***Introduction***

This lab had four key tasks to it. The first task was to verify the functionality of the datapath. The data path which performs four arithmetic operations, AND, OR, Addition, Subtract. The testbench could either be eyeballing or self-checking, self-checking was preferred as it would make later tasks less complex. The next three tasks are a subset of a larger design task, which was is design and test a calculator. To be able to design and functionally test the calculator a control unit which acts as a finite state machine and was designed and tested with a self-checking test bench. Once the control unit was properly tested, it was incorporated with the data path to create an entire system which was then verified with a self-checking test bench. Lastly, it was all verified on the FPGA using a variety of switches and buttons to test the calculator functionality.

The parts listed below were fully completed in lab:

* Verification stimuli and design of Datapath
* Verification stimuli and design of Calculator
* Verification stimuli and design of Control Unit

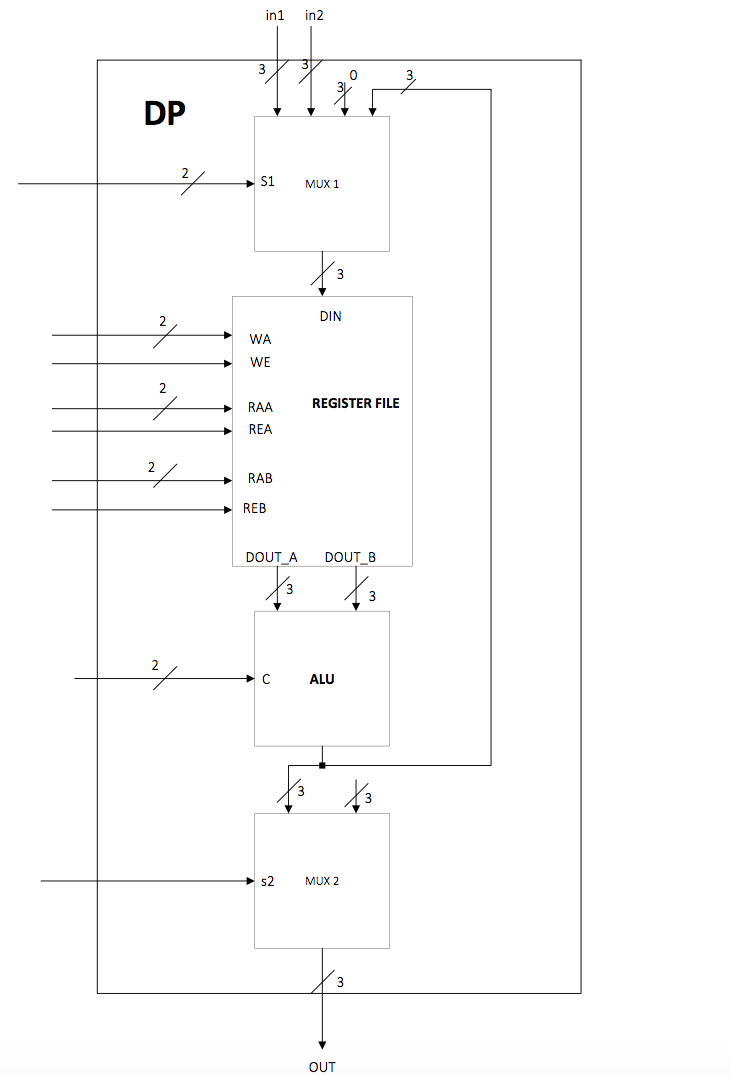
This part was not completed during lab:

* Hardware verification of Calculator

***Design Methodology***

*Datapath:* The datapath was designed with a mux, register file, ALU, and an additional mux. The first mux received three inputs, In1, In2, and s1. The two-bit input, s1, would select which input to receive choosing between In1, In2, 0, and the output of the ALU. The register file had several inputs, write enable, write address, two read enables, two read addresses, clock, and din from the mux. It then output two numbers into the ALU. The ALU would receive these numbers and depending on the value of c, it would perform either an addition, subtraction, XOR, or AND operation. Lastly, those values would go to the second mux where s2 would select between the values input from the ALU or 0. The resulting values would then be the output. Figure one shows the data path block diagram.

Figure 1: Datapath Block Diagram



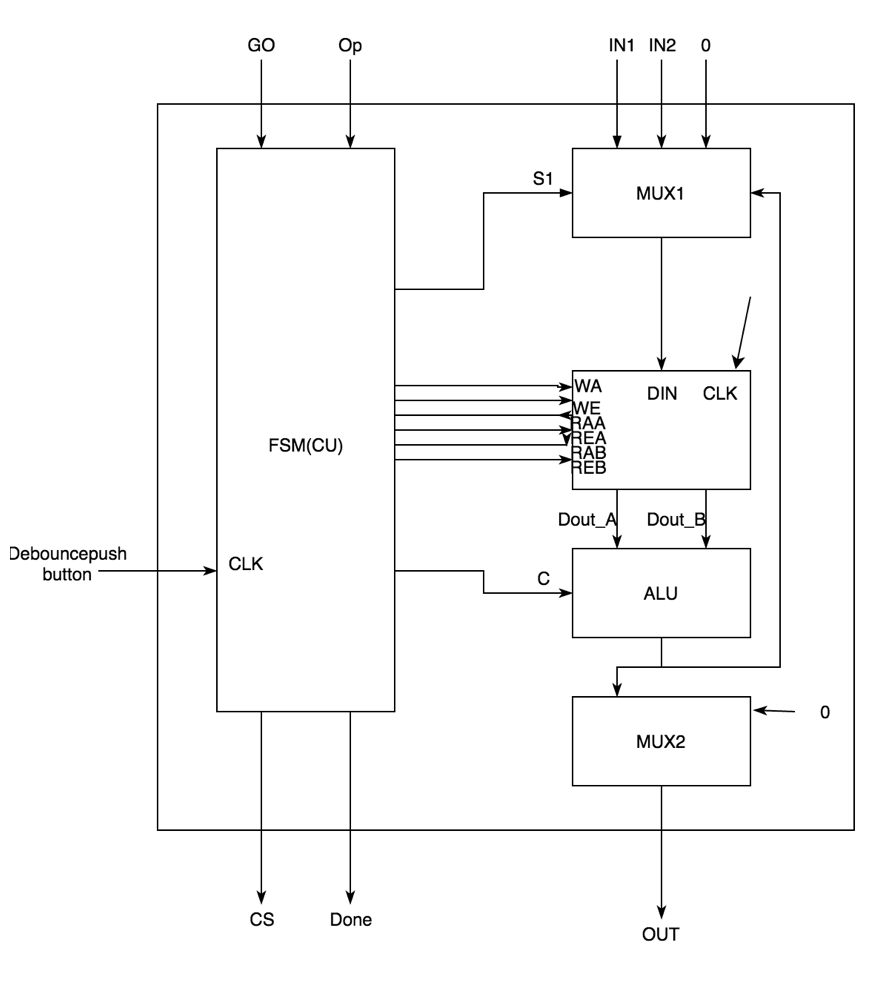
*Calculator:* The calculator is a composition of two smaller designs. The data path and control unit acting as one module allows simulation verification and hardware verification. The control unit acts as an infinite state machine which has a total of 8 states. Table one shows the OP code functions for each binary represented state. External signals allow the control unit to utilize the datapath functions to generate an output.

|  |  |
| --- | --- |
| **Operation Control Input** | **Performed Operations** |
| 00 | XOR: R1^R2 |
| 01 | AND: R1&R2 |
| 10 | Subtraction: R1-R2 |
| 11 | Addition: R1+R2 |

Table 1. Operation Control Input and Performed Operations

The control unit and data path are integrated as one module. Looking at figure two describes the how the control unit is a map to the datapath.

Figure 2: Top Level Design of FPGA block diagram



*Calculator sub-module:* The control unit is a submodule of the calculator which integrates it self with the datapath. Control unit is a Finite State Machine which has 8 state registers. The code is responsible to knowing the next state and current state logic on positive edge triggers. The code also determines the output of the state dependent on operation codes. Overall this FSM has nine state if the starting state, S0, is counted. The machine is static till the signal GO is given. After GO the machine is designed to iterate through each state while being instructed using OP codes to perform certain functions at each state. The control unit leaves the last state as the done state. Shown below is the control unit block diagram in figure 3, head over to the appendix to find the ASM flow chart and state transition diagram.

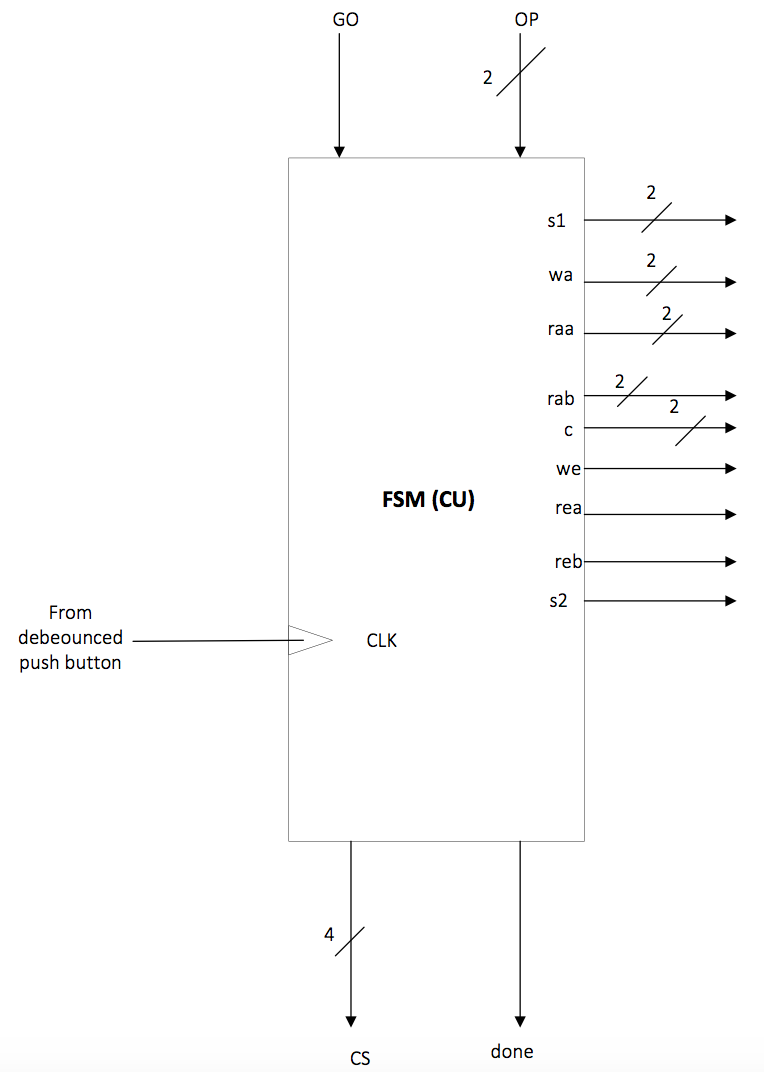


Figure 4: Control Unit block diagram

***Simulation Test Plan***

*Datapath:* The datapath was tested using a self-checking test bench. The test-bench would iterate through several values to test for all cases. It iterates through each integer value for In1 and In2 and then iterates through each operation. A register for the expected output was created in this lab and was used for verification when comparing the output to the expected result. Additionally, the value of the data in write address three of the register file was compared to the expected value. That is because the value stored in write address three of the register file contained the result of the operation between In1 and In2 or whatever was chosen by s1.

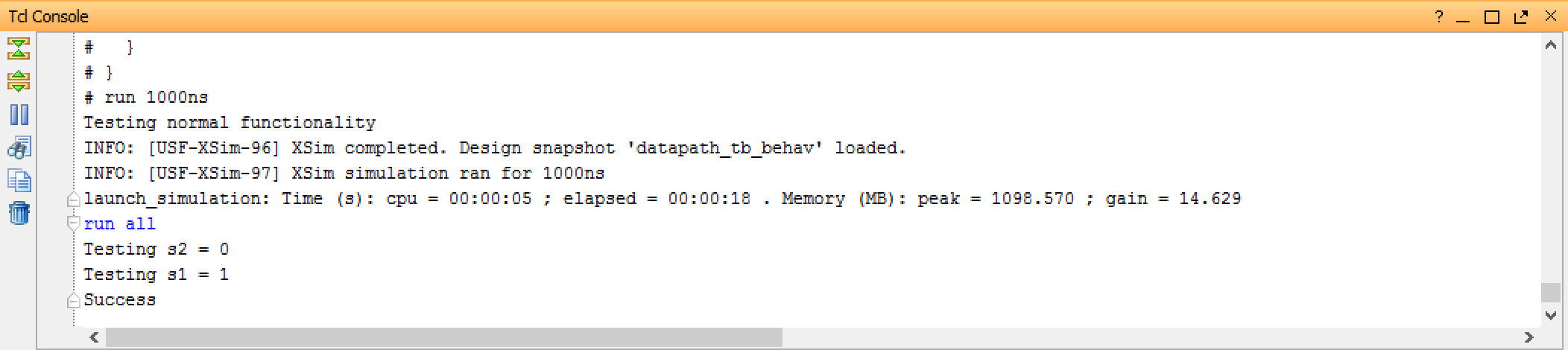
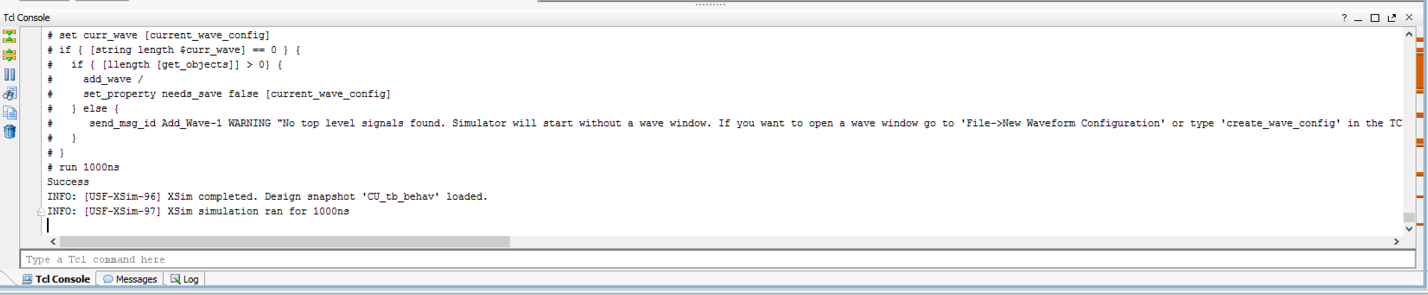


Figure 4: Console Output of Datapath Simulation

*Calculator:*

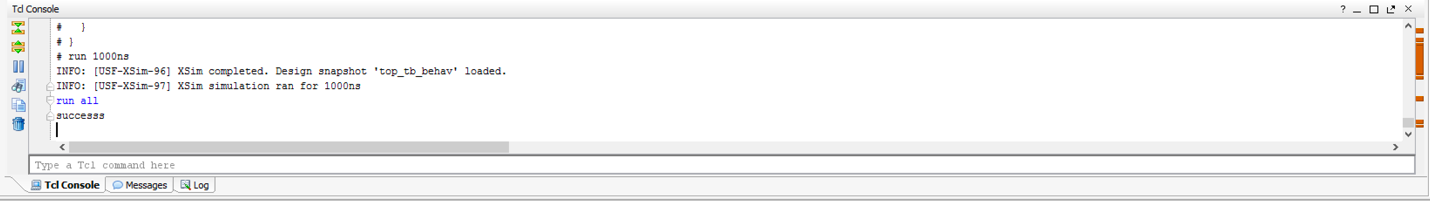
The test bench was designed to work with the clock to and respond to signals such as GO and DONE. All signals are low and 0 at the beginning. By using a layered of nested for loops it was possible to achieve all possible outputs. The most inner loop checks for AND, XOR, Addition and Subtraction using if statements where the output is matched against the expected output. This top level module utilized the control unit and datapath. If at any point the output is not correct the test bench is halted. Below in figure five a screenshot is shown the output success message meaning all 8 states performed as intended and all code from the datapath and control unit were also successful.

Figure5: Console Output of Calculator Simulation



*Control Unit:* This part was integrating the and testing the control unit. Figure six shows the success message. This

Figure5: Console Output of Control Unit Simulation



***FPGA Test Plan***

To attempt to validate the board, three new modules were added to be able to use the seven-segment display, a button as a clock bounce. The XDC file was set to use the switches, RGB LED and seven segment display. However, we were unable to get this part to work as the bit stream was not fully being generating.

***Conclusion***

In conclusion, the calculator worked as intended. All stimulation verification tests were passed for all stages. By looking at the code a workflow was hand drawn and presented to the TA. The state transition diagram and flowchart were acceptable and fulfilled the lab requirements Each submodule was working and efficient, however the bit stream was the only problem. A long list of errors is being generated during the bit stream generation. Some minor changes to the test bench were applied during demonstration time to make the code more efficient. Extra figures are provided within the appendix to show verification in detail.

***Appendix***

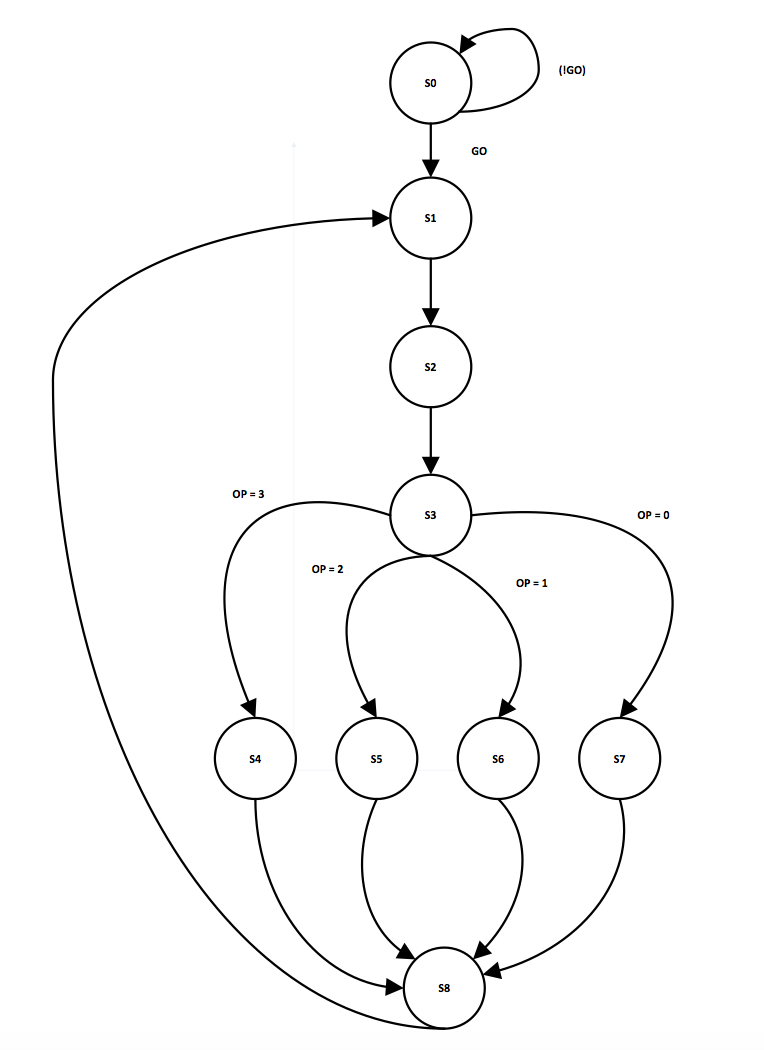


Figure 7: Control Unit FSM state transition diagram

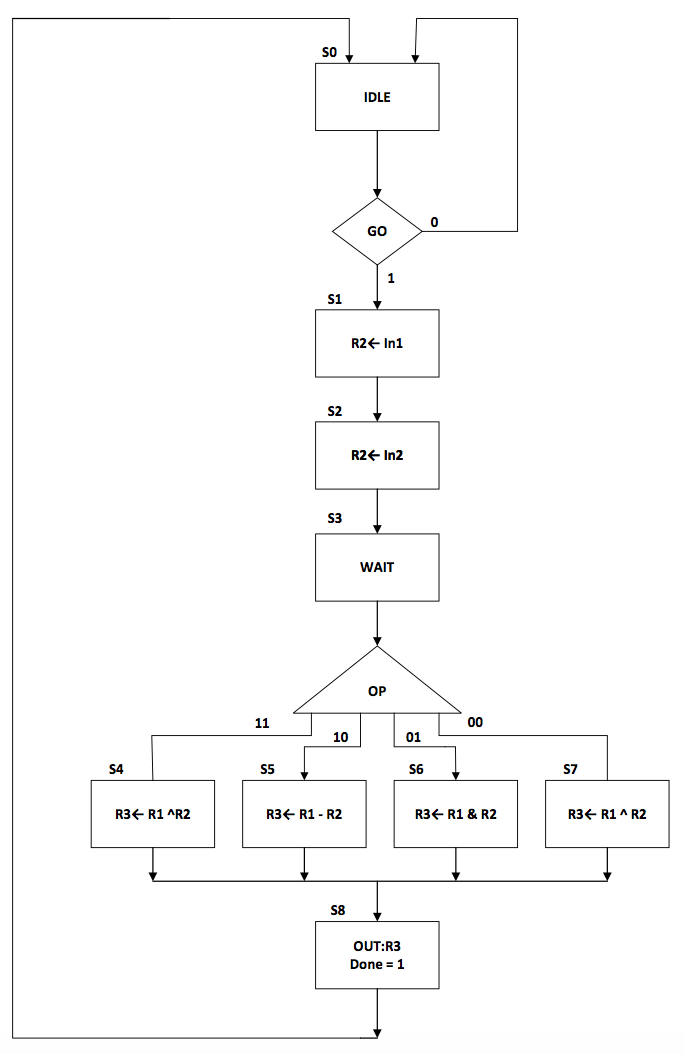


Figure 8: Control Unit ASM flow chart

Figure 2: FSM logic table for control unit

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | Outputs | | | | | | | | | |
| CS | S1 | WA | WE | RAA | REA | RAB | REB | C | S2 | Done |
| S0 | 01 | 00 | 0 | 0 | 0 | 00 | 0 | 00 | 0 | 0 |
| S1 | 11 | 01 | 11 | 1 | 0 | 00 | 0 | 00 | 0 | 0 |
| S2 | 10 | 10 | 0 | 1 | 0 | 00 | 0 | 00 | 0 | 0 |
| S3 | 01 | 00 | 1 | 0 | 0 | 00 | 0 | 00 | 0 | 0 |
| S4 | 00 | 11 | 1 | 1 | 1 | 10 | 1 | 00 | 0 | 0 |
| S5 | 00 | 11 | 1 | 1 | 1 | 10 | 1 | 01 | 0 | 0 |
| S6 | 00 | 11 | 1 | 1 | 1 | 10 | 1 | 10 | 0 | 0 |
| S7 | 00 | 11 | 1 | 1 | 1 | 10 | 1 | 11 | 0 | 0 |
| S8 | 01 | 00 | 0 | 0 | 1 | 11 | 1 | 10 | 1 | 1 |

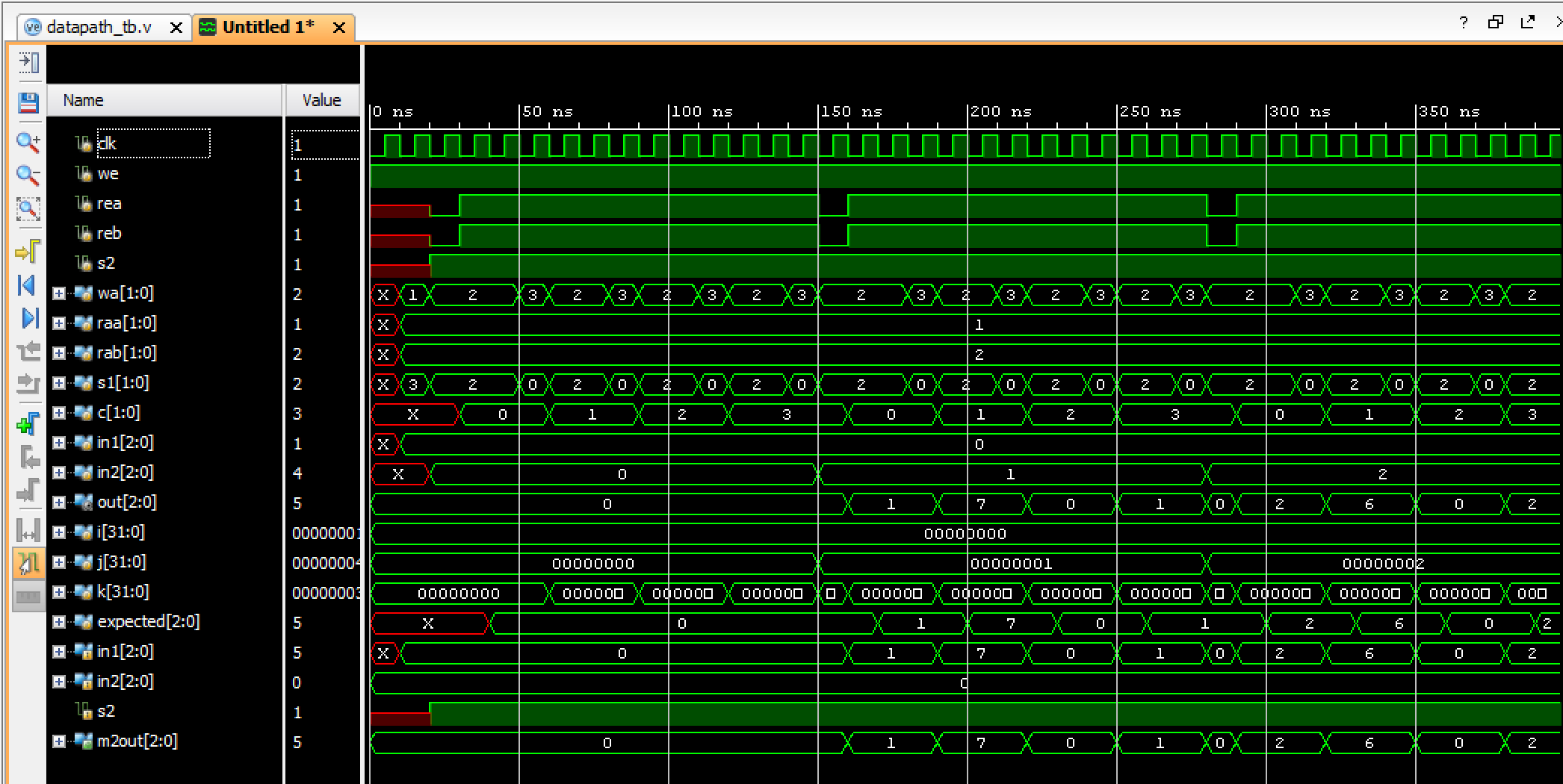


Figure 9: Waveform Stimuli for Datapath

Figure 10: Waveform Stimuli for Calculator

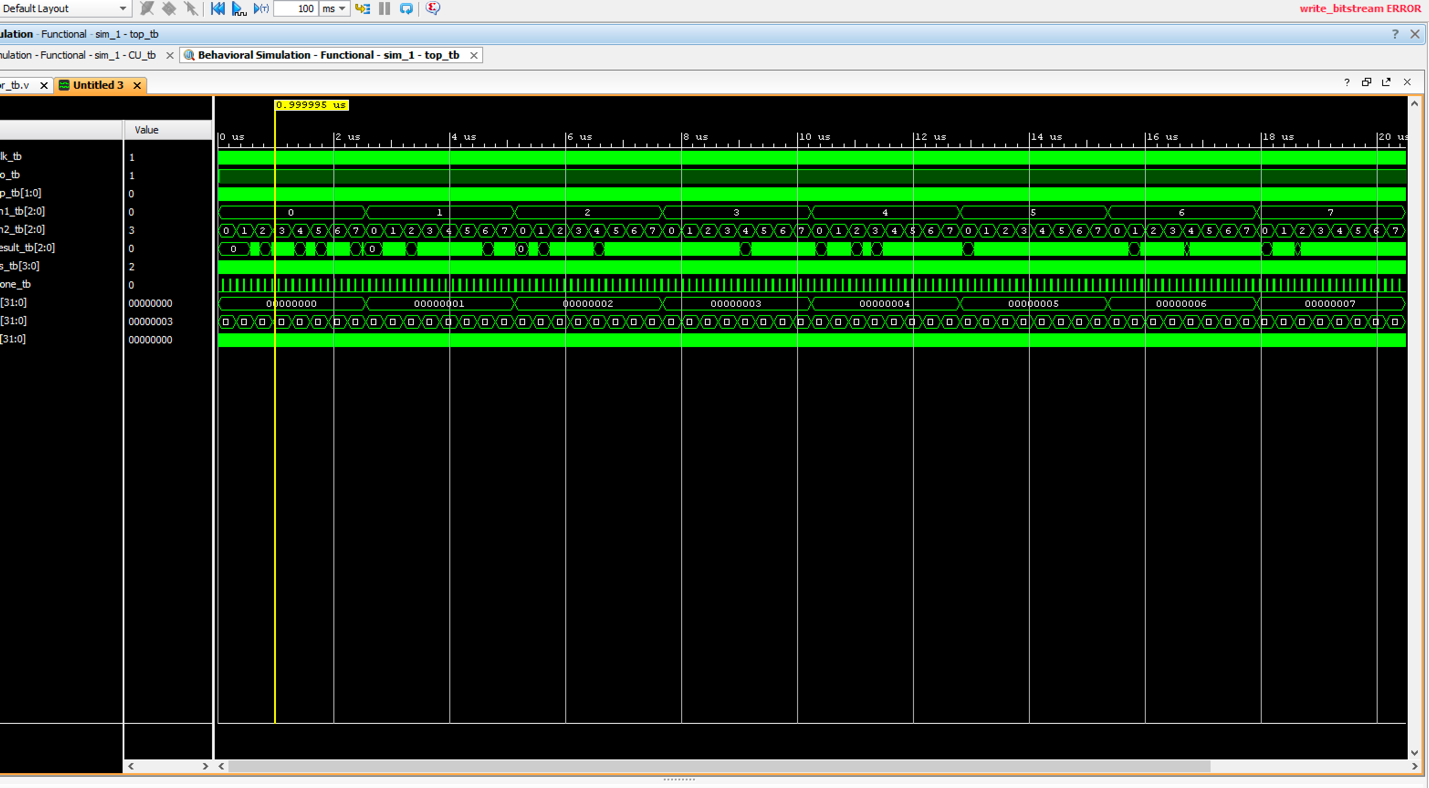
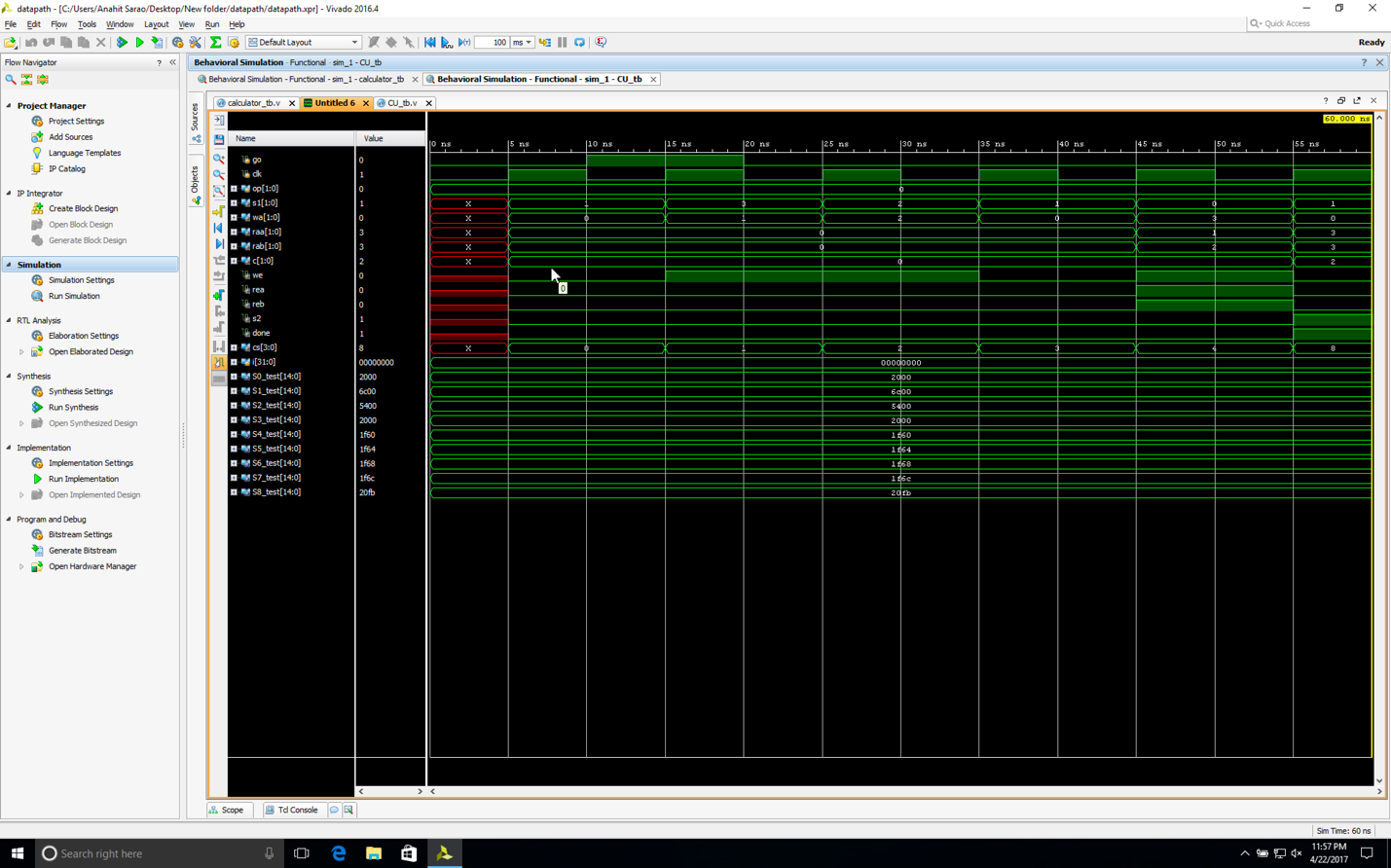


Figure 11: Waveform Stimuli for Control Unit



***Source Code***

Control Unit tb

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/19/2017 12:45:40 PM

// Design Name:

// Module Name: CU\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module CU\_tb();

reg go, clk;

reg[1:0] op;

wire [1:0] s1, wa, raa, rab, c;

wire we, rea, reb, s2, done;

wire [3:0] cs;

integer i;

CU DUT(.go(go), .clk(clk), .op(op), .s1(s1), .wa(wa), .raa(raa), .rab(rab),

.c(c), .we(we), .rea(rea), .reb(reb), .s2(s2), .done(done), .cs(cs));

task cycle;

begin

clk=0;

#5;

clk=1;

#5;

end

endtask

parameter S0\_test = 15'b01\_00\_0\_0\_0\_00\_00\_00\_0\_0;

parameter S1\_test = 15'b11\_01\_1\_0\_0\_00\_00\_00\_0\_0;

parameter S2\_test = 15'b10\_10\_1\_0\_0\_00\_00\_00\_0\_0;

parameter S3\_test = 15'b01\_00\_0\_0\_0\_00\_00\_00\_0\_0;

parameter S4\_test = 15'b00\_11\_1\_1\_1\_01\_10\_00\_0\_0;

parameter S5\_test = 15'b00\_11\_1\_1\_1\_01\_10\_01\_0\_0;

parameter S6\_test = 15'b00\_11\_1\_1\_1\_01\_10\_10\_0\_0;

parameter S7\_test = 15'b00\_11\_1\_1\_1\_01\_10\_11\_0\_0;

parameter S8\_test = 15'b01\_00\_0\_0\_0\_11\_11\_10\_1\_1;

initial

begin

clk=0;

go=0;

for(i =0; i<4; i=i+1)

begin

op=i;

cycle;

if(cs!=0)

begin

$display("Error with go operation");

$stop;

end

if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S0\_test)

begin

$display("Error initializing state 0");

$stop;

end

go=1;

cycle;

if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S1\_test)

begin

$display("Error changing from state 0 to state 1");

$stop;

end

go=0;

cycle;

if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S2\_test)

begin

$display("Error changing from state 1 to state 2");

$stop;

end

cycle;

if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S3\_test)

begin

$display("Error changing from state 2 to state 3");

$stop;

end

cycle;

case(op)

2'b00: if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S4\_test)

begin

$display("Error transitioning to State 4");

$stop;

end

2'b01: if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S5\_test)

begin

$display("Error transitioning to State 5");

$stop;

end

2'b10: if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S6\_test)

begin

$display("Error transitioning to State 6");

$stop;

end

2'b11: if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S7\_test)

begin

$display("Error transitioning to State 7");

$stop;

end

endcase

cycle;

if({s1, wa, we, rea, reb, raa, rab, c, s2, done}!= S8\_test)

begin

$display("Error transitioning to State 8");

$stop;

end

$display("Success");

$stop;

end

end

endmodule

**data path tb**

`timescale 1ns / 1ps

module datapath\_tb();

reg clk, we, rea, reb, s2;

reg [1:0] wa, raa, rab, s1, c;

reg [2:0] in1, in2;

wire [2:0] out;

DP DUT(.clk(clk), .we(we), .rea(rea), .reb(reb), .s2(s2),

.wa(wa), .raa(raa), .rab(rab), .s1(s1), .c(c),

.in1(in1), .in2(in2), .out(out));

task cycle;

begin

clk=0;

#5;

clk=1;

#5;

end

endtask

integer i=0;

integer j=0;

integer k=0;

reg [2:0] expected;

initial

begin

we=1;

cycle;

$display("Testing normal functionality");

raa=1;

rab=2;

for(i=0; i<8; i=i+1)

begin

wa=1;

in1=i;

s1=3;

cycle;

for(j=0; j<8; j=j+1)

begin

wa=2;

in2=j;

rea=0;

reb=0;

s1=2;

s2=1;

cycle;

for(k=0; k<4; k=k+1)

begin

s1=2;

wa=2;

rea=1;

reb=1;

c=k;

raa=1;

rab=2;

cycle;

case(c)

2'b00: expected = in1 + in2;

2'b01: expected = in1 - in2;

2'b10: expected = in1 & in2;

default: expected = in1 ^ in2; // 2'b11;

endcase

cycle;

if(out!=expected)

begin

$display ("Error: output does not equal expected");

$stop;

end

s1=0;

wa=3;

cycle;

if(DUT.U1.RegFile[3]!=expected)

begin

$display("error with r3");

$stop;

end;

end

end

end

$display("Testing s2 = 0");

in1=3; in2=2;

rea=0; reb=0;

wa=1;

c=0;

s1=3;

cycle;

wa=2;

rea=1; reb=1;

s2=0;

cycle;

if(out!=0)

begin

$display("Error s2 not set properly");

$stop;

end

$display("Testing s1 = 1");

in1=3; in2=2;

rea=0; reb=0;

wa=1;

c=0;

s1=1;

cycle;

wa=2;

rea=1; reb=1;

s2=1;

cycle;

if(out!=in1)

begin

$display("Error s1 not set properly");

$stop;

end

$display("Success");

$stop;

end

endmodule

**calculator-top-level**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/19/2017 02:00:54 PM

// Design Name:

// Module Name: calculator

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module top(Clk,go,cs,op,In1,In2,result,done);

input Clk;

input go;

input [1:0] op;

input [2:0] In1;

input [2:0] In2;

output [2:0] result;

output [3:0] cs;

output done;

wire WE;

wire REA;

wire REB;

wire [1:0] WA;

wire [1:0] RAA;

wire [1:0] RAB;

wire [1:0] S1;

wire [1:0] C;

wire S2;

DP DUT1(.in1(In1), .in2(In2), .s1(S1), .clk(Clk), .wa(WA), .we(WE), .raa(RAA), .rea(REA), .rab(RAB), .reb(REB), .c(C), .s2(S2), .out(result));

CU DUT2(.Clk(Clk),.Go(go),.Op(op),.sel1(S1),.WA(WA),.RAA(RAA),.RAB(RAB),.WE(WE),.REA(REA),.REB(REB),.c(C),.sel2(S2),.Done(done),.cs(cs));

button\_debouncer U4(.clk(clk\_5KHz), .button(control), .debounced\_button(debouncedButton));

endmodule

// this is the control unit

module CU(s1, wa, we, rea, reb, raa, rab, c, s2, done, go, clk, cs, op);

output reg [1:0] s1, wa, raa, rab, c;

output reg we, rea, reb, s2, done;

output reg [3:0] cs;

input clk, go;

input [1:0] op;

reg [3:0] ns; //next state

parameter S0 = 4'b0000; //state parameters

parameter S1 = 4'b0001;

parameter S2 = 4'b0010;

parameter S3 = 4'b0011;

parameter S4 = 4'b0100;

parameter S5 = 4'b0101;

parameter S6 = 4'b0110;

parameter S7 = 4'b0111;

parameter S8 = 4'b1000;

always @ (go, cs)

begin

case(cs)

S0: ns = (go)? S1: S0;

S1: ns = S2;

S2: ns = S3;

S3: case(op)

2'b00: ns = S4;

2'b01: ns = S5;

2'b10: ns = S6;

2'b11: ns = S7;

endcase

S4: ns = S8;

S5: ns = S8;

S6: ns = S8;

S7: ns = S8;

default: ns = 0;

endcase

end

always@(posedge clk)

begin

cs <= ns;

end

always@(cs)

begin

case(cs)

S0: begin s1=1; wa=0; we=0; rea=0; reb=0; raa=0; rab=0; c=0; s2=0; done=0; end

S1: begin s1=3; wa=1; we=1; rea=0; reb=0; raa=0; rab=0; c=0; s2=0; done=0; end

S2: begin s1=2; wa=2; we=1; rea=0; reb=0; raa=0; rab=0; c=0; s2=0; done=0; end

S3: begin s1=1; wa=0; we=0; rea=0; reb=0; raa=0; rab=0; c=0; s2=0; done=0; end

S4: begin s1=0; wa=3; we=1; rea=1; reb=1; raa=1; rab=2; c=0; s2=0; done=0; end

S5: begin s1=0; wa=3; we=1; rea=1; reb=1; raa=1; rab=2; c=1; s2=0; done=0; end

S6: begin s1=0; wa=3; we=1; rea=1; reb=1; raa=1; rab=2; c=2; s2=0; done=0; end

S7: begin s1=0; wa=3; we=1; rea=1; reb=1; raa=1; rab=2; c=3; s2=0; done=0; end

S8: begin s1=1; wa=0; we=0; rea=0; reb=0; raa=3; rab=3; c=2; s2=1; done=1; end

endcase

end

endmodule

module MUX1(in1, in2, in3, in4, s1, m1out);

input [2:0] in1, in2, in3, in4;

input [1:0] s1;

output reg [2:0] m1out;

always @ (in1, in2, in3, in4, s1)

begin

case (s1)

2'b11: m1out = in1;

2'b10: m1out = in2;

2'b01: m1out = in3;

default: m1out = in4;

endcase

end

endmodule //MUX1

module RF(clk, rea, reb, raa, rab, we, wa, din, douta, doutb);

input clk, rea, reb, we;

input [1:0] raa, rab, wa;

input [2:0] din;

output reg [2:0] douta, doutb;

reg [2:0] RegFile [3:0];

always @(rea, reb, raa, rab)

begin

if (rea)

douta = RegFile[raa];

else douta = 3'b000;

if (reb)

doutb = RegFile[rab];

else doutb = 3'b000;

end

always@(posedge clk)

begin

if (we)

RegFile[wa] = din;

else

RegFile[wa] = RegFile[wa];

end

endmodule //RF

module ALU (in1, in2, c, aluout);

input [2:0] in1, in2;

input [1:0] c;

output reg [2:0] aluout;

always @ (in1, in2, c)

begin

case (c)

2'b00: aluout = in1 + in2;

2'b01: aluout = in1 - in2;

2'b10: aluout = in1 & in2;

default: aluout = in1 ^ in2;

endcase

end

endmodule

module MUX2 (in1, in2, s2, m2out);

input [2:0] in1, in2;

input s2;

output reg [2:0] m2out;

always @ (in1, in2, s2)

begin

if (s2)

m2out = in1;

else

m2out = in2;

end

endmodule //MUX2

**calculatortb.v**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 04/19/2017 02:11:14 PM

// Design Name:

// Module Name: calculator\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module top\_tb;

reg Clk\_tb;

reg go\_tb;

reg [1:0] op\_tb;

reg [2:0] In1\_tb;

reg [2:0] In2\_tb;

wire [2:0] result\_tb;

wire [3:0] cs\_tb;

wire done\_tb;

integer a,b,c;

top DUT(.Clk(Clk\_tb),.go(go\_tb),.cs(cs\_tb),.op(op\_tb),.In1(In1\_tb),.In2(In2\_tb),.result(result\_tb),.done(done\_tb));

initial begin

Clk\_tb=1'b0;

go\_tb=1'b0;

op\_tb=2'b00;

In1\_tb=2'b00;

In2\_tb=2'b00;

for(a=0;a<8;a=a+1)

begin

for(b=0;b<8;b=b+1)

begin

In1\_tb=a;

In2\_tb=b;

for(c=0;c<4;c=c+1)

begin

op\_tb=c;

#10 Clk\_tb=1;

go\_tb=1'b1;

#10 Clk\_tb=0;

#10 Clk\_tb=1;

#10 Clk\_tb=0;

#10 Clk\_tb=1;

#10 Clk\_tb=0;

#10 Clk\_tb=1;

#10 Clk\_tb=0;

if(op\_tb==0 && done\_tb==1)

begin

if(result\_tb!=(In1\_tb^In2\_tb))

begin

$display("fail");

$stop;

end

end

if(op\_tb==1 && done\_tb==1)

begin

if(result\_tb!=(In1\_tb&In2\_tb))

begin

$display("fail");

$stop;

end

end

if(op\_tb==2 && done\_tb==1)

begin

if(result\_tb!=In1\_tb-In2\_tb)

begin

$display("fail");

$stop;

end

end

if(op\_tb==3 && done\_tb==1)

begin

if(result\_tb!=In1\_tb+In2\_tb)

begin

$display("fail");

$stop;

end

end

end

end

end

$display("successs");

$stop;

end

endmodule