

**Introduction**

In this lab, we are assigned to build a system level design with emphasis on the control portion. To fulfill our goal, we build a calculator with different states and a go signal to indicate when our CPU can start doing the calculation. Then we made a couple testbench and also a self checking testbench to check our progress.

**Design Methodology**

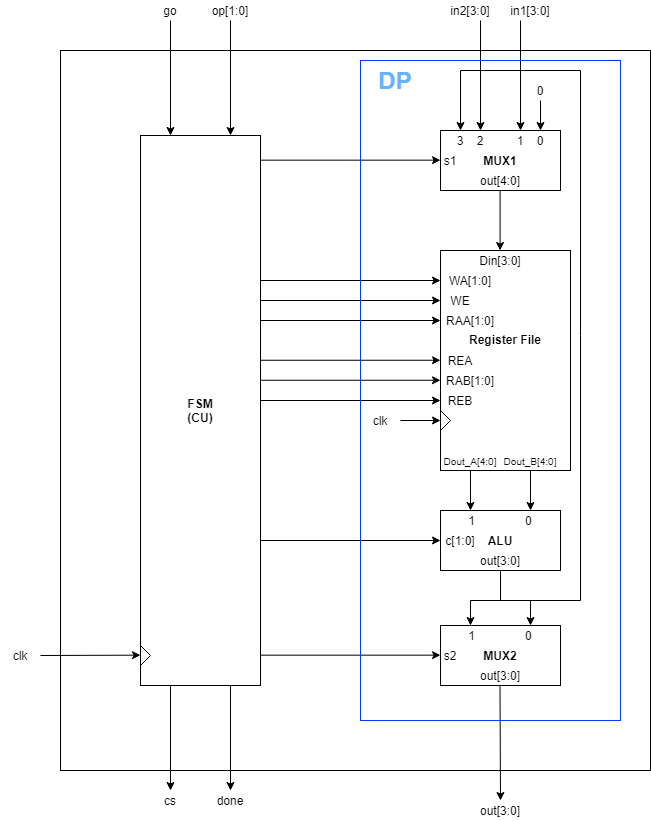
The design of this lab first begins with understanding the timing behind how our small calculator works. First, we need to load two numbers into the register. Since we can only load a single number at a time, we must use at least 2 clock cycles for this portion of the calculator. Once the numbers have been loaded into the register, we can then follow through with performing our operation.

In order to use the calculator, a state machine must be generated through the control unit to correctly perform the calculation each time a number is input. The control unit will output the appropriate control signals depending on which state it is in. To initialize the state machine, the go signal must be a logic high. This initializes the control signals for the calculator and then it sends the state machine through each of its different states. Initially, the first 3 states are the same regardless of the operation. After this, it will go into either state 4, 5, 6 or 7 depending on the OP code given, which decides which operation to perform. The final state is used to output the result to the 7 segment displays, and then it will cycle in state 1 repeatedly until a new go signal is given.

In table 1 shown below, the modules used in the design of our small calculator shown.

*Table 1. Module Definitions*

|  |  |
| --- | --- |
| **Modules** | **Function** |
| small\_calculator\_dp | Implements calculator with datapaths |
| small\_calculator\_cu | Implements calculator with control unit |
| control\_unit.v | Implements control unit for calculator |
| mux4.v | Implements 4 to 1 mux |
| register\_file.v | Implements register file |
| alu.v | Implements alu to perform calculation |
| mux2.v | Implements 2 to 1 mux |
| clk\_gen.v | Generates clock signal |
| button\_debouncer.v | Generates clock signal via button press |
| sevenSegDisplay.v | Generates decimal number on 7-segment LED on FPGA board |
| self\_checking\_tb.v | Testbench for calculator with datapath |
| go\_clk\_tb.v | Testbench for control unit functionality |
| fsm\_calc\_tb.v | Full self checking testbench for calculator with control unit |

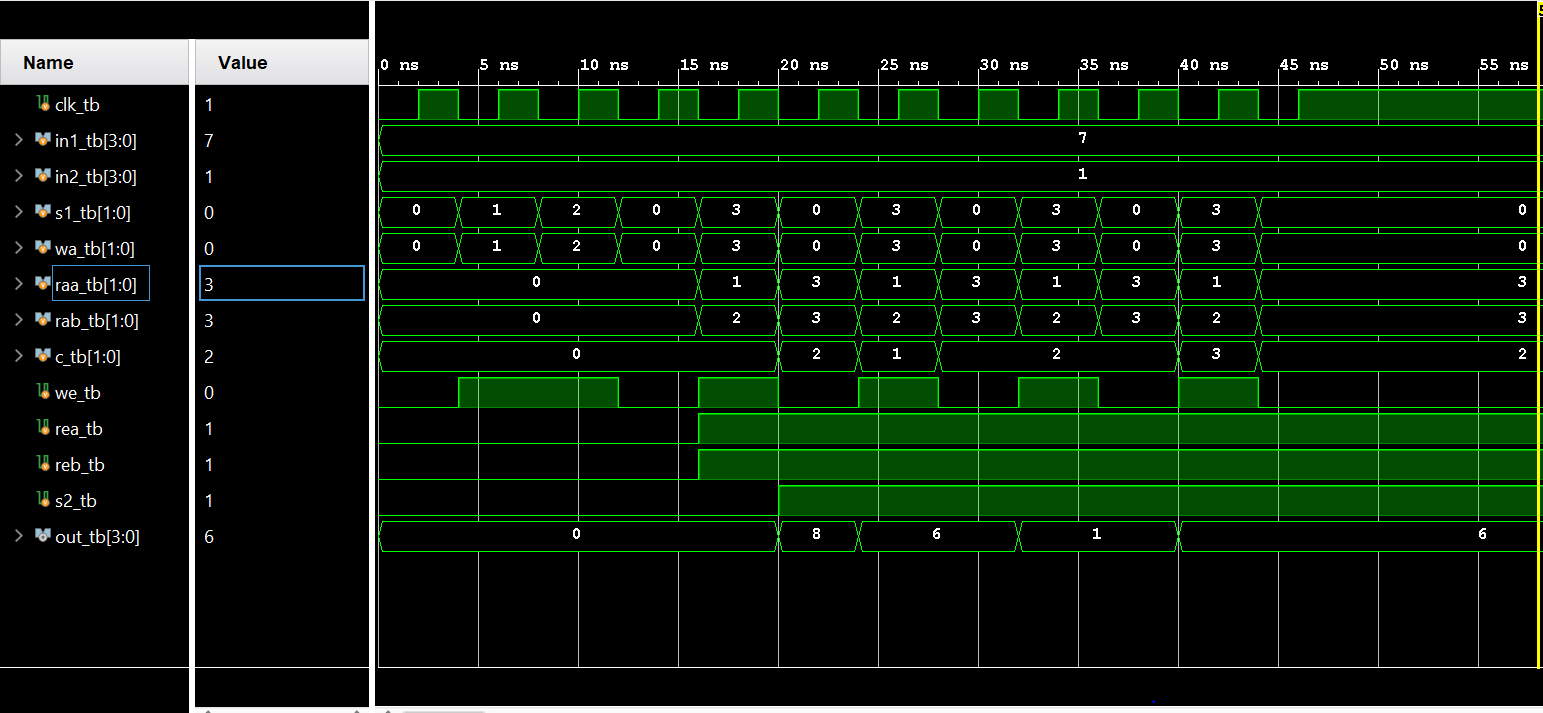


*Schematic 1. Schematic of the small calculator high level design*

**Simulation Results**

Task 1:

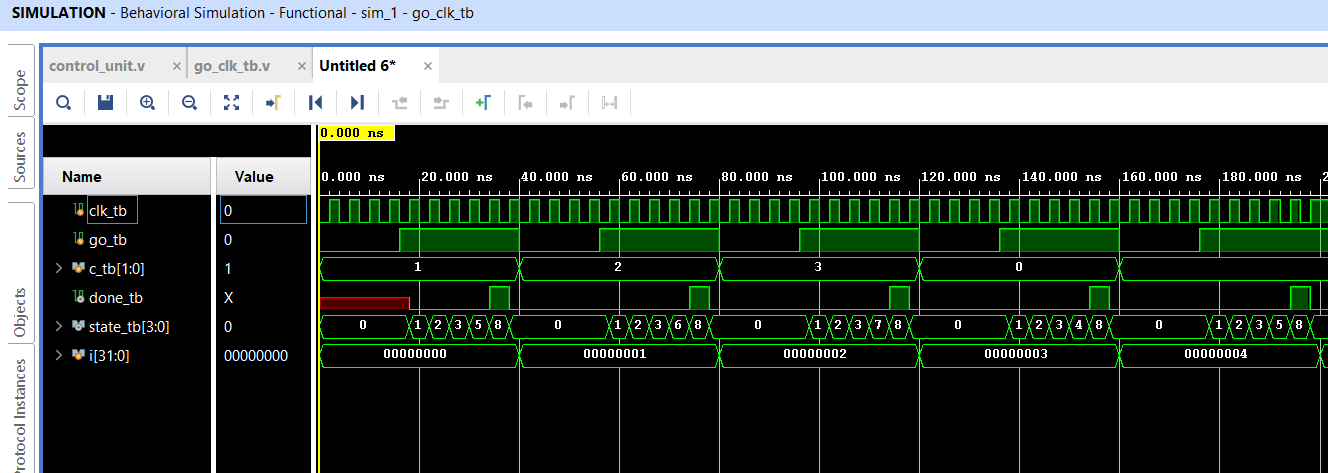
The first waveform shows the calculation being done using input 1, input 2, and c which determines the operation. Here, we are determining the correctness of our datapath when we are doing the calculations.



*Figure 1. Waveform from task 1*

Task 2:

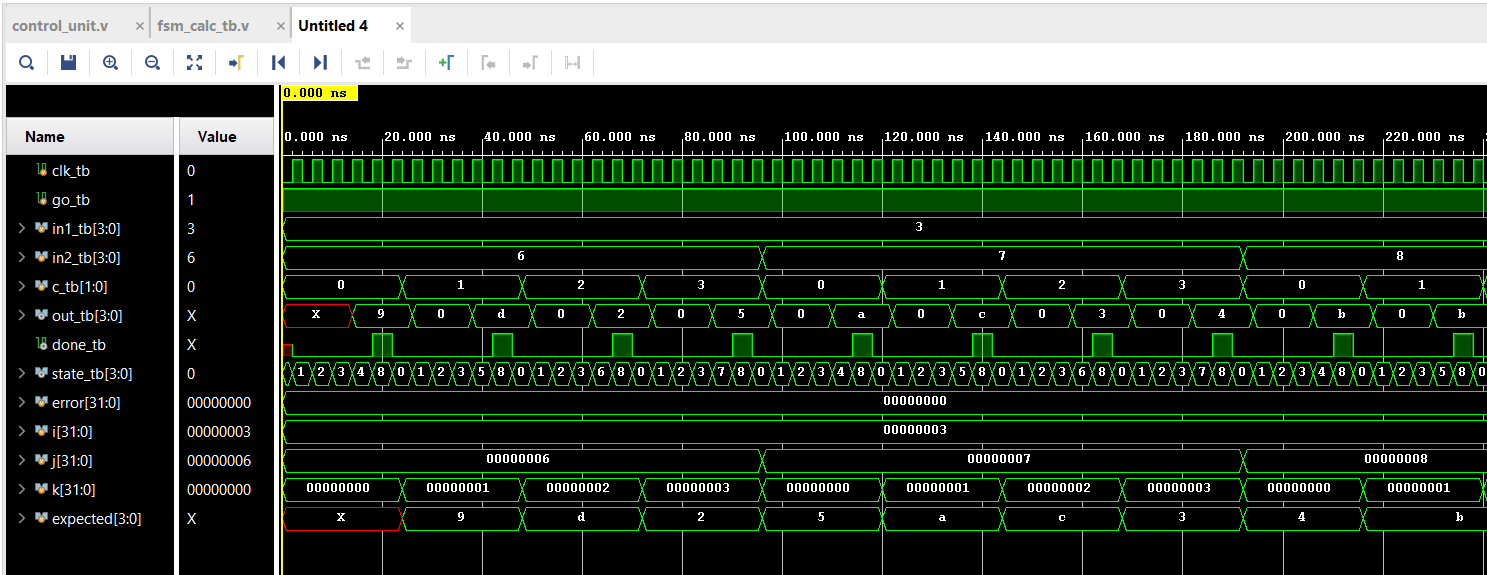
The waveform shown below is a simple testbench to show that our control unit is working as intended. By passing in clock signals without the go active, the state should not advance anywhere. However, after setting go to active and sending clock signals, then we are able to see the control unit advance to other states. The state it advances to is determined by what c is, which is responsible for choosing what operation the ALU is going to perform.



*Figure 2. Waveform from Control Unit*

Task 3:

The final waveform shows what we combined in the last 2 weeks. In it, we see the calculation done using the two inputs and different operations determined by c. With different c values, the state will range from 4 to 7. The done flag will be active when the state reaches 8. The error counter will increment when either the calculation is not equal to the expected value, or if the done flag sets or resets at the wrong state.



*Figure 3. Waveform from Calculator*

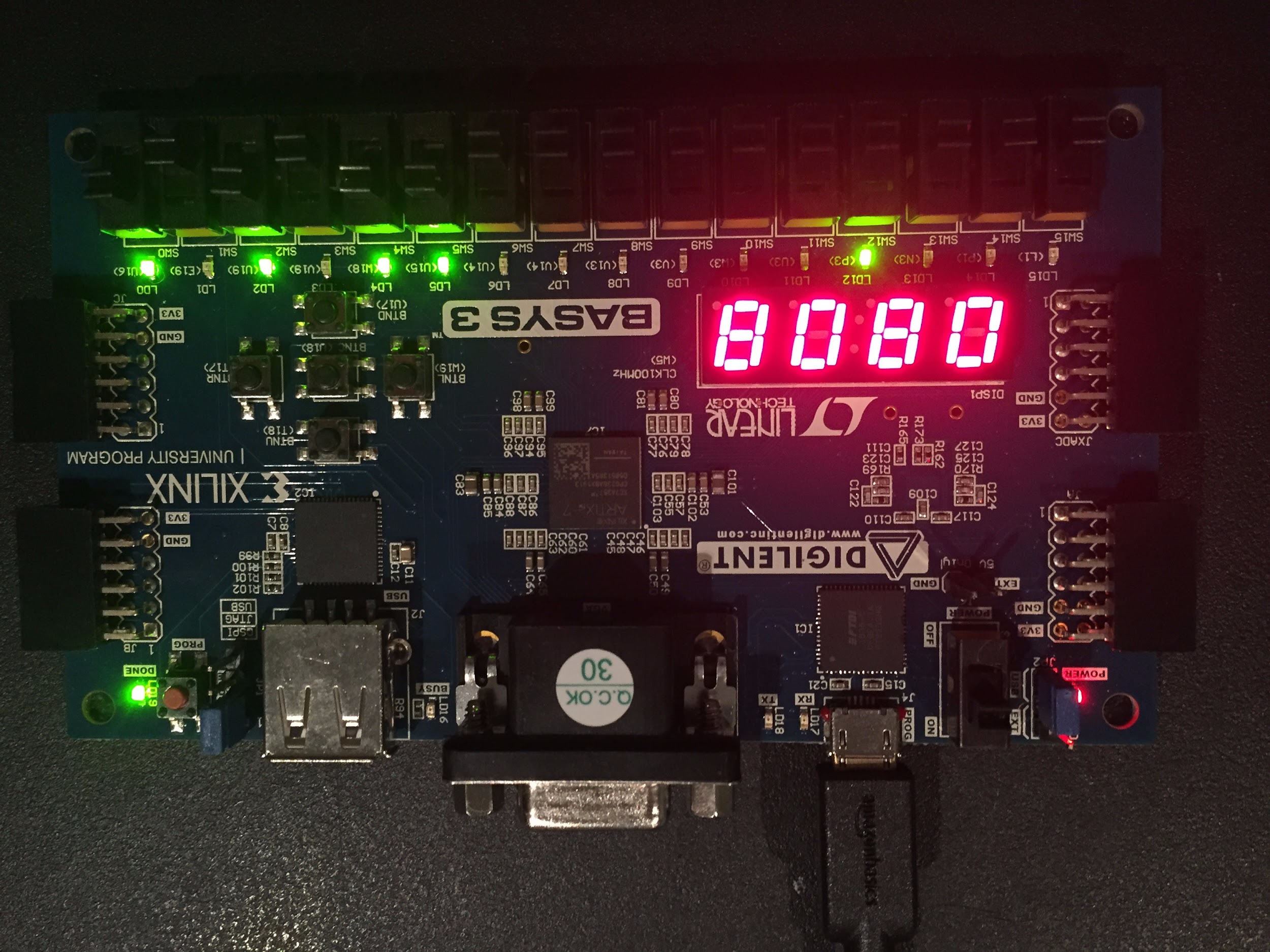
**FPGA Validation**

To validate the calculator on the FPGA board, figures 4 through 7 will show our results.

The two leftmost switches are our control switches, the 4th led from the left is the “done” signal, the center button is the debounced clock button and the right button is the “go” signal. The 8 rightmost switches represent the two 4 bit inputs to our calculator.

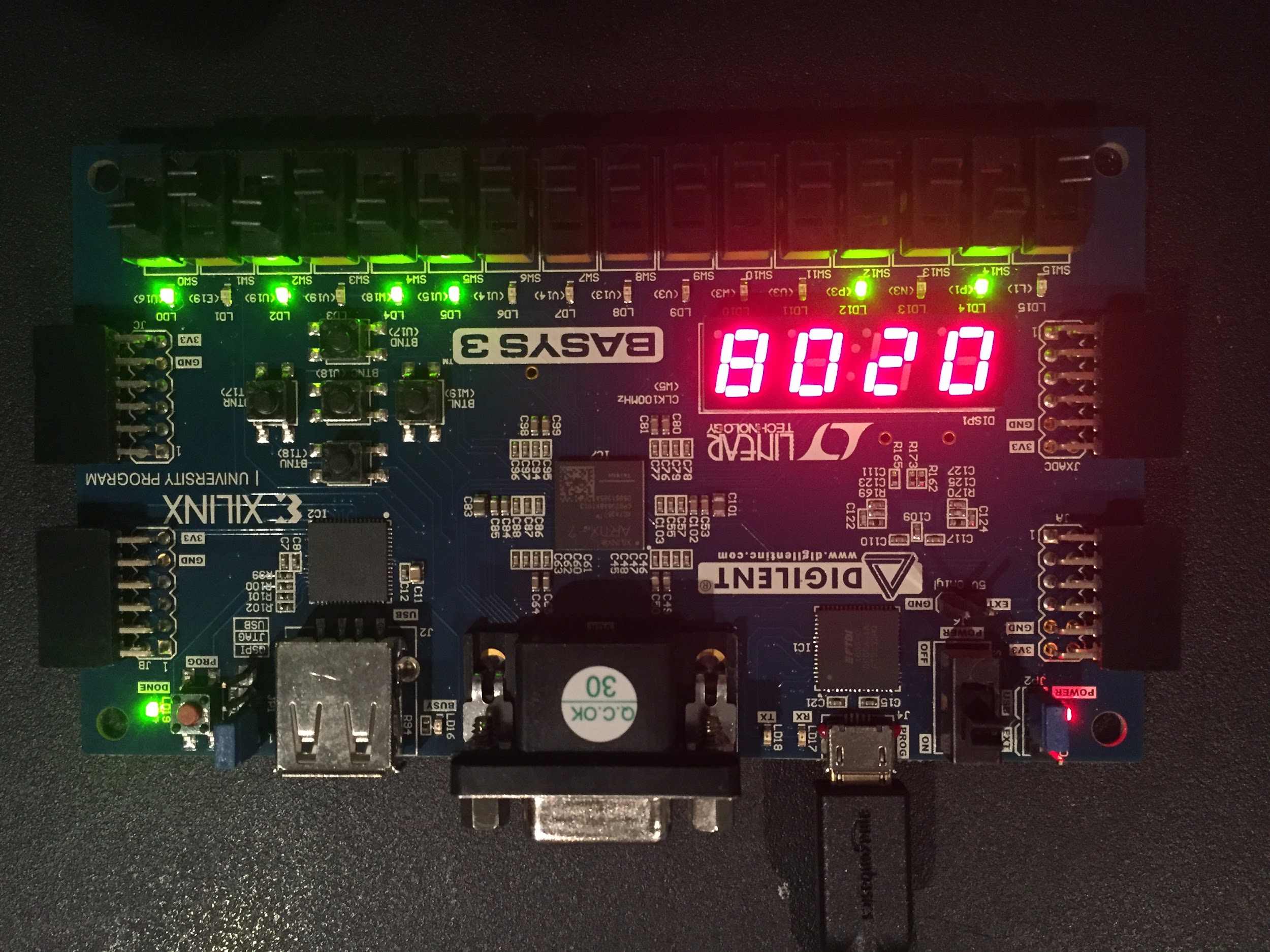
On the 7 segment display, the two leftmost numbers represent the output, and the rightmost number represents the state while the second to the right is not used and always displays a zero.

In figure 4 below, an addition operation is shown. Input 2 is 0011b and input 1 is 0101b which is 5 + 3. In the eighth state, the output is displayed and the done signal is active. The output shows the correct result.



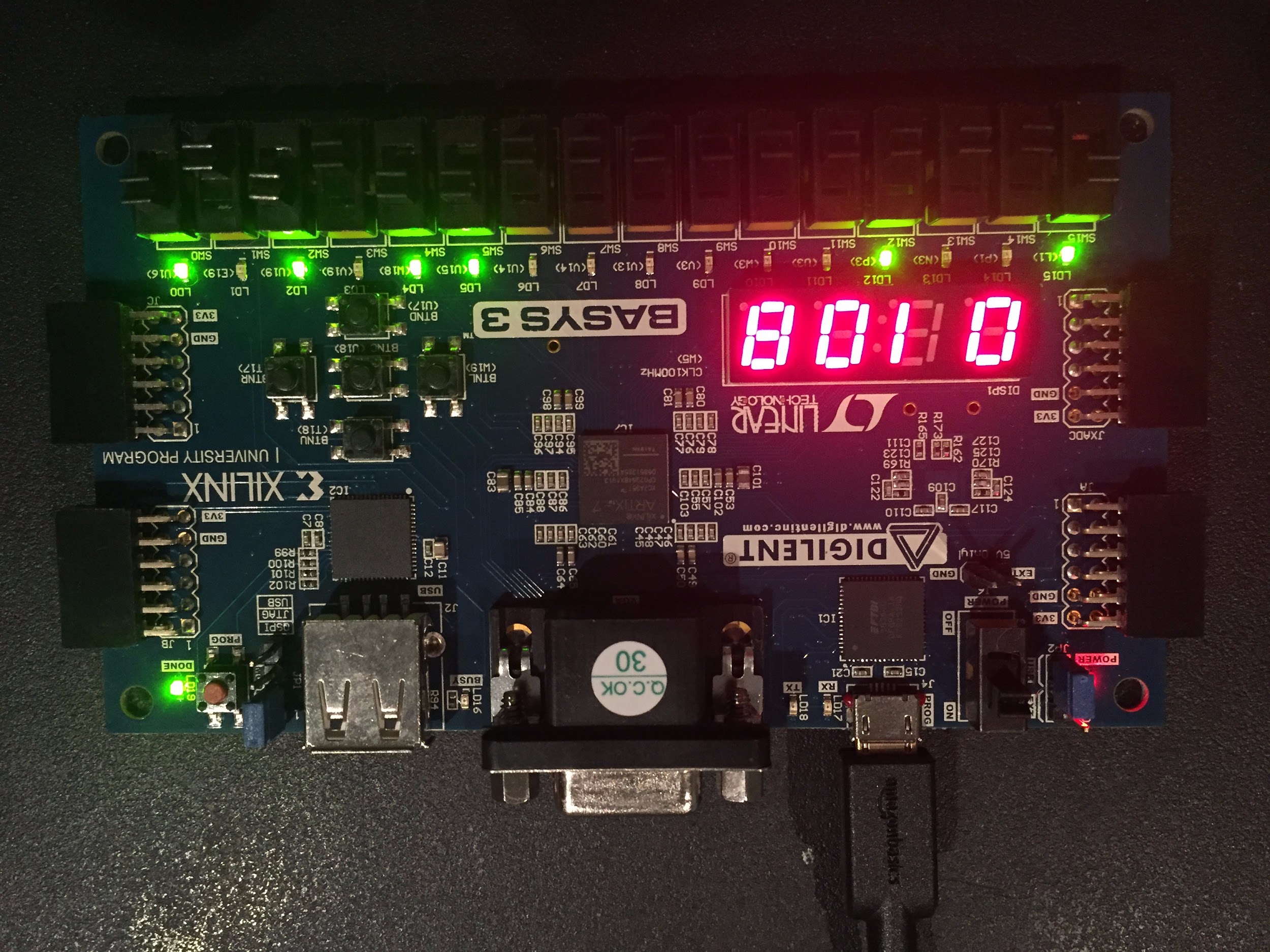
*Figure 4. Addition of the numbers 3 and 5*

In figure 5, a subtraction operation is shown because the OP code is 01. Using the same numbers, the operation is 5 - 3 = 2. The correct output is shown again.



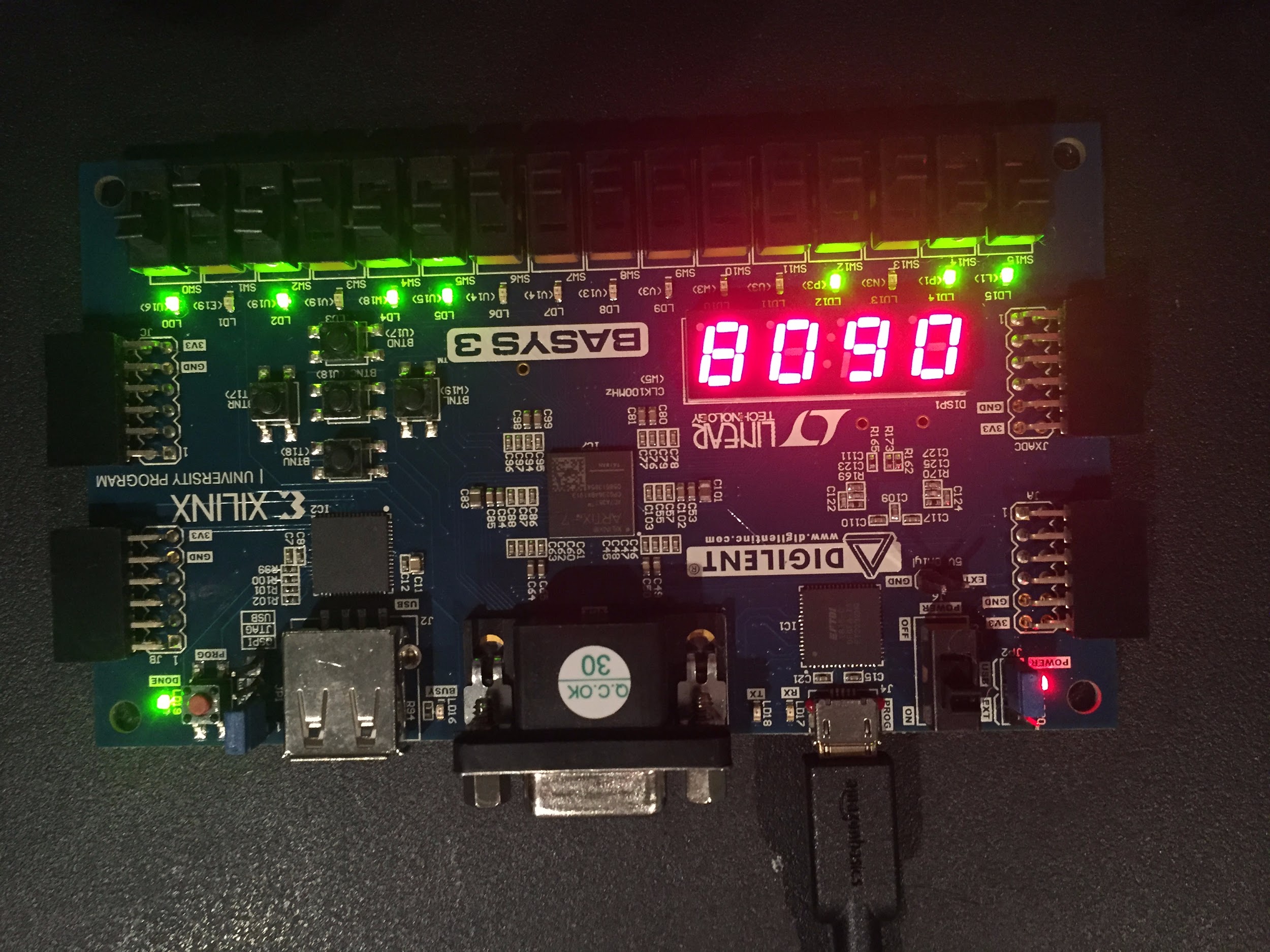
*Figure 5. Subtraction of the numbers 3 and 5*

In figure 6, an AND operation is shown because the OP code is 10. Using the same numbers, the operation is 0101b • 0011b = 0001b = 1. The correct output is shown again.



*Figure 6. AND of the numbers 3 and 5*

In figure 7, an XOR operation is shown because the OP code is 10. Using the same numbers, the operation is 0101b ^ 0011b = 0110b = 6. The correct output is shown again.

****

*Figure 7. XOR of the numbers 3 and 5*

From our FPGA validation, our calculations were shown to be correct.

**Conclusion**

This lab was successful because we were able to build a system level design with emphasis on the control portion. We also built a calculator with different states and a go signal to indicate when our CPU can start doing the calculation. At the end, we made testbenches to verify the calculator and the control unit and also a self checking testbench to check our progress.

Overall, we were able to complete all four tasks of the lab. We also implemented the calculator with the eight different stages on the FPGA board. To do this we added a debounced button to manually control our clock signal while using a state machine to generate the control signals. The control unit made a state machine to appropriately apply the correct timing to our calculator. This lab expanded on previous labs where we used modules such as registers and it required knowledge on how to appropriately add information into a register and also knowledge on how to extract this information in the correct sequence.

**Appendix**

|  |
| --- |
| **small\_calculator\_dp.v** |
| **module small\_calculator\_dp (**  **input wire clk,**  **input wire go,**  **input wire [3:0] in1,**  **input wire [3:0] in2,**  **input wire [1:0] c,**    **output wire [3:0] out,**  **output wire done,**  **output wire [3:0] state**  **);**    **wire [3:0] mux1out;**  **wire [3:0] douta;**  **wire [3:0] doutb;**  **wire [3:0] aluout;**    **wire [1:0] s1;**  **wire [1:0] wa;**  **wire [1:0] raa;**  **wire [1:0] rab;**  **wire we;**  **wire rea;**  **wire reb;**  **wire s2;**  **wire [1:0] cout\_cu;**  **control\_unit cu(**  **.go (go),**  **.clk(clk),**  **.c(c),**  **.s1(s1),**  **.wa(wa),**  **.raa(raa),**  **.rab(rab),**  **.c\_out(cout\_cu),**  **.we(we),**  **.rea(rea),**  **.reb(reb),**  **.s2(s2),**  **.done(done),**  **.state(state)**  **);**  **mux4 #(**  **.WIDTH(4)**  **) MUX1 (**  **.sel (s1),**  **.in0 (4'b0),**  **.in1 (in1),**  **.in2 (in2),**  **.in3 (aluout),**  **.out (mux1out)**  **);**  **register\_file RF (**  **.clk (clk),**  **.rea (rea),**  **.reb (reb),**  **.we (we),**  **.raa (raa),**  **.rab (rab),**  **.wa (wa),**  **.din (mux1out),**  **.douta (douta),**  **.doutb (doutb)**  **);**  **alu ALU (**  **.c (cout\_cu),**  **.in0 (douta),**  **.in1 (doutb),**  **.alu\_out (aluout)**  **);**  **mux2 #(**  **.WIDTH(4)**  **) MUX2 (**  **.sel (s2),**  **.in0 (4'b0),**  **.in1 (aluout),**  **.out (out)**  **);**  **endmodule** |

|  |
| --- |
| **register\_file.v** |
| **module register\_file (**  **input wire clk,**  **input wire rea,**  **input wire reb,**  **input wire we,**  **input wire [1:0] raa,**  **input wire [1:0] rab,**  **input wire [1:0] wa,**  **input wire [3:0] din,**  **output wire [3:0] douta,**  **output wire [3:0] doutb**  **);**  **reg [3:0] RegFile[3:0];**  **always @ (posedge clk) begin**  **if(we) begin**  **RegFile[wa] <= din;**  **end**  **else begin**  **RegFile[wa] <= RegFile[wa];**  **end**  **end**  **assign douta = (rea == 1'b1) ? RegFile[raa] : 4'b0;**  **assign doutb = (reb == 1'b1) ? RegFile[rab] : 4'b0;**  **endmodule** |

|  |
| --- |
| **mux4.v** |
| **module mux4 #(parameter WIDTH = 4) (**  **input wire [1:0] sel,**  **input wire [WIDTH-1:0] in0,**  **input wire [WIDTH-1:0] in1,**  **input wire [WIDTH-1:0] in2,**  **input wire [WIDTH-1:0] in3,**  **output reg [WIDTH-1:0] out**  **);**  **always @ (\*) begin**  **case(sel)**  **2'b00: out = in0;**  **2'b01: out = in1;**  **2'b10: out = in2;**  **2'b11: out = in3;**  **endcase**  **end**    **endmodule** |

|  |
| --- |
| **mux2.v** |
| **module mux2 #(WIDTH = 4) (**  **input wire sel,**  **input wire [WIDTH-1:0] in0,**  **input wire [WIDTH-1:0] in1,**  **output wire [WIDTH-1:0] out**  **);**  **assign out = (sel == 1'b1) ? in1 : in0;**  **endmodule** |

|  |
| --- |
| **alu.v** |
| **module alu (**  **input wire [1:0] c,**  **input wire [3:0] in0,**  **input wire [3:0] in1,**  **output reg [3:0] alu\_out**  **);**  **always @ (in0, in1, c) begin**  **case(c)**  **2'b00: alu\_out = in0 + in1;**  **2'b01: alu\_out = in0 - in1;**  **2'b10: alu\_out = in0 & in1;**  **2'b11: alu\_out = in0 ^ in1;**  **endcase**  **end**  **endmodule** |

|  |
| --- |
| **button\_debouncer.v** |
| **module button\_debouncer #(parameter depth = 16) (**  **input wire clk, /\* 5 KHz clock \*/**  **input wire button, /\* Input button from constraints \*/**  **output reg debounced\_button**  **);**    **localparam history\_max = (2\*\*depth)-1;**  **/\* History of sampled input button \*/**  **reg [depth-1:0] history;**  **always @ (posedge clk) begin**  **/\* Move history back one sample and insert new sample \*/**  **history <= { button, history[depth-1:1] };**    **/\* Assert debounced button if it has been in a consistent state throughout history \*/**  **debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;**  **end**    **endmodule** |

|  |
| --- |
| **clk\_gen.v** |
| **`timescale 1ns / 1ps**  **module clk\_gen (**  **input wire clk100MHz,**  **input wire rst,**  **output reg clk\_4sec,**  **output reg clk\_5KHz**  **);**  **integer count1, count2;**  **always @ (posedge clk100MHz) begin**  **if (rst) begin**  **count1 = 0; clk\_4sec = 0;**  **count2 = 0; clk\_5KHz = 0;**  **end else begin**  **if (count1 == 200000000) begin**  **clk\_4sec = ~clk\_4sec;**  **count1 = 0;**  **end**  **if (count2 == 10000) begin**  **clk\_5KHz = ~clk\_5KHz;**  **count2 = 0; end**  **count1 = count1 + 1;**  **count2 = count2 + 1;**  **end**  **end**  **endmodule** |

|  |
| --- |
| **sevenSegDisplay.v** |
| **module sevenSegDisplay(**  **input clk100MHz,**  **input [3:0]led0,**  **input [3:0]led1,**  **input [3:0]led2,**  **input [3:0]led3,**  **output reg [7:0]led\_out,**  **output reg [3:0]led\_pins**  **);**  **reg clk\_5KHz;**  **reg [1:0]led\_sel;**  **reg [3:0]led\_out\_bcd;**  **integer count1, count2;**  **always @ (posedge clk100MHz) begin**  **if(count1 == 200000000) begin**  **count1 = 0;**  **end**  **if (count2 == 10000) begin**  **clk\_5KHz = ~clk\_5KHz;**  **count2 = 0;**  **end**  **count1 = count1 + 1;**  **count2 = count2 + 1;**  **end**  **always @ (posedge clk\_5KHz) begin**  **led\_sel = led\_sel + 1;**  **case(led\_sel)**  **2'b00: begin**  **led\_pins = 4'b0111;**  **led\_out\_bcd = led0;**  **end**  **2'b01: begin**  **led\_pins = 4'b1011;**  **led\_out\_bcd = led1;**  **end**  **2'b10: begin**  **led\_pins = 4'b1101;**  **led\_out\_bcd = led2;**  **end**  **2'b11: begin**  **led\_pins = 4'b1110;**  **led\_out\_bcd = led3;**  **end**  **endcase**  **case(led\_out\_bcd)**  **4'd0: led\_out = 8'b11000000;**  **4'd1: led\_out = 8'b11111001;**  **4'd2: led\_out = 8'b10100100;**  **4'd3: led\_out = 8'b10110000;**  **4'd4: led\_out = 8'b10011001;**  **4'd5: led\_out = 8'b10010010;**  **4'd6: led\_out = 8'b10000010;**  **4'd7: led\_out = 8'b11111000;**  **4'd8: led\_out = 8'b10000000;**  **4'd9: led\_out = 8'b10010000;**  **4'b10: led\_out = 8'b01111111;**  **default: led\_out = 8'b11111111;**  **endcase**  **end**  **endmodule** |

|  |
| --- |
| **selfchecking\_tb.v** |
| **`timescale 1ns / 1ps**  **module self\_checking\_tb;**  **reg clk\_tb; //**  **reg [3:0] in1\_tb; //**  **reg [3:0] in2\_tb; //**  **reg [1:0] s1\_tb; //**  **reg [1:0] wa\_tb; //**  **reg [1:0] raa\_tb;//**  **reg [1:0] rab\_tb;//**  **reg [1:0] c\_tb; //alu control**  **reg we\_tb; //write enable**  **reg rea\_tb; //read enable a**  **reg reb\_tb; //read enable b**  **reg s2\_tb;**  **wire [3:0] out\_tb;**    **small\_calculator\_dp DUT(**  **.clk(clk\_tb),**  **.in1(in1\_tb),**  **.in2(in2\_tb),**  **.s1(s1\_tb),**  **.wa(wa\_tb),**  **.raa(raa\_tb),**  **.rab(rab\_tb),**  **.c(c\_tb),**  **.we(we\_tb),**  **.rea(rea\_tb),**  **.reb(reb\_tb),**  **.s2(s2\_tb),**  **.out(out\_tb)**  **);**      **task clock;**  **begin**  **clk\_tb = 0;**  **#2;**  **clk\_tb = 1;**  **#2;**  **end**  **endtask**    **initial begin**  **clk\_tb = 0;**  **rea\_tb = 0;**  **reb\_tb = 0;**  **raa\_tb = 2'b00;**  **rab\_tb = 2'b00;**  **c\_tb = 2'b00;**  **s2\_tb = 0;**  **in1\_tb = 4'b0111; //7**  **in2\_tb = 4'b0001; //1**  **clk\_tb = 0;**  **///////////////////////////////////////////////////////////////////////////////**  **wa\_tb = 2'b00;**  **s1\_tb = 2'b00;**  **we\_tb = 0;**  **clock;**    **we\_tb = 1;**  **wa\_tb = 2'b01; //2nd clock cycle**  **s1\_tb = 2'b01;**  **clock;**    **s1\_tb = 2'b10;**  **wa\_tb = 2'b10;**  **clock;**    **s1\_tb = 2'b00;**  **we\_tb = 0;**  **wa\_tb = 2'b00;**  **clock;**    **//add**  **we\_tb = 1;**  **rea\_tb = 1;**  **reb\_tb = 1;**  **raa\_tb = 2'b01;**  **rab\_tb = 2'b10;**  **c\_tb = 2'b00;**  **s1\_tb = 2'b11;**  **wa\_tb = 2'b11;**  **clock;**    **s1\_tb = 2'b00;**  **wa\_tb = 2'b00;**  **we\_tb = 0;**  **raa\_tb = 2'b11;**  **rab\_tb = 2'b11;**  **c\_tb = 2'b10;**  **s2\_tb = 1;**  **clock;**  **//subtract**  **we\_tb = 1;**  **rea\_tb = 1;**  **reb\_tb = 1;**  **raa\_tb = 2'b01;**  **rab\_tb = 2'b10;**  **c\_tb = 2'b01;**  **s1\_tb = 2'b11;**  **wa\_tb = 2'b11;**  **clock;**    **s1\_tb = 2'b00;**  **wa\_tb = 2'b00;**  **we\_tb = 0;**  **raa\_tb = 2'b11;**  **rab\_tb = 2'b11;**  **c\_tb = 2'b10;**  **s2\_tb = 1;**  **clock;**    **//and**  **we\_tb = 1;**  **rea\_tb = 1;**  **reb\_tb = 1;**  **raa\_tb = 2'b01;**  **rab\_tb = 2'b10;**  **c\_tb = 2'b10;**  **s1\_tb = 2'b11;**  **wa\_tb = 2'b11;**  **clock;**    **s1\_tb = 2'b00;**  **wa\_tb = 2'b00;**  **we\_tb = 0;**  **raa\_tb = 2'b11;**  **rab\_tb = 2'b11;**  **c\_tb = 2'b10;**  **s2\_tb = 1;**  **clock;**    **//xor**  **we\_tb = 1;**  **rea\_tb = 1;**  **reb\_tb = 1;**  **raa\_tb = 2'b01;**  **rab\_tb = 2'b10;**  **c\_tb = 2'b11;**  **s1\_tb = 2'b11;**  **wa\_tb = 2'b11;**  **clock;**    **s1\_tb = 2'b00;**  **wa\_tb = 2'b00;**  **we\_tb = 0;**  **raa\_tb = 2'b11;**  **rab\_tb = 2'b11;**  **c\_tb = 2'b10;**  **s2\_tb = 1;**  **clock;**  **end**  **endmodule** |

|  |
| --- |
| **go\_clk\_tb.v** |
| **`timescale 1ns / 1ps**  **module go\_clk\_tb;**  **reg clk\_tb;**  **reg go\_tb;**  **reg [3:0] in1\_tb;**  **reg [3:0] in2\_tb;**  **reg [1:0] c\_tb=0;**    **wire [3:0] out\_tb;**  **wire done\_tb;**  **wire [3:0] state\_tb;**    **small\_calculator\_dp DUT(**  **.clk(clk\_tb),**  **.go(go\_tb),**  **.in1(in1\_tb),**  **.in2(in2\_tb),**  **.c(c\_tb),**    **.out(out\_tb),**  **.done(done\_tb),**  **.state(state\_tb)**  **);**    **task clock;**  **begin**  **clk\_tb = 0;**  **#2;**  **clk\_tb = 1;**  **#2;**  **end**  **endtask**    **integer i;**    **initial begin**  **for(i=0; i<5; i=i+1)**  **begin**  **go\_tb=0;**  **c\_tb=c\_tb+1;**  **clock;**  **clock;**  **clock;**  **clock;**  **go\_tb=1;**  **clock;**  **clock;**  **clock;**  **clock;**  **clock;**  **clock;**  **end**  **end**    **endmodule** |

|  |
| --- |
| **fsm\_calc\_tb.v** |
| **`timescale 1ns / 1ps**  **module fsm\_calc\_tb;**  **reg clk\_tb;**  **reg go\_tb;**  **reg [3:0] in1\_tb;**  **reg [3:0] in2\_tb;**  **reg [1:0] c\_tb;**    **wire [3:0] out\_tb;**  **wire done\_tb;**  **wire [3:0] state\_tb;**  **integer error=0;**    **small\_calculator\_dp DUT(**  **.clk(clk\_tb),**  **.go(go\_tb),**  **.in1(in1\_tb),**  **.in2(in2\_tb),**  **.c(c\_tb),**    **.out(out\_tb),**  **.done(done\_tb),**  **.state(state\_tb)**  **);**    **task clock;**  **begin**  **clk\_tb = 0;**  **#2;**  **clk\_tb = 1;**  **#2;**  **end**  **endtask**  **//go = 0**  **// couple clock**  **//go =1**  **//couple**  **task checkDone;**  **begin**  **if(4 <= state\_tb | state\_tb <= 8)**  **begin**  **if(done\_tb==0)**  **begin**  **error=error+1;**  **end**  **end**  **if(0 <= state\_tb | state\_tb <= 3)**  **begin**  **if(done\_tb==1)**  **begin**  **error=error+1;**  **end**  **end**  **end**  **endtask**    **integer i;**  **integer j;**  **integer k;**  **reg[3:0] expected;**  **initial begin**  **for(i=3; i<16; i=i+1)**  **begin**  **for(j=6; j<16; j=j+1)**  **begin**  **for(k=0; k<4; k=k+1)**  **begin**  **in1\_tb=i;**  **in2\_tb=j;**  **c\_tb=k;**  **go\_tb=1;**  **clock;**  **clock;**  **clock;**  **clock;**  **clock;**  **clock;**    **//check add**  **if(k==0)**  **begin**  **expected=i+j;**  **if(expected != out\_tb)**  **begin**  **error=error+1;**  **end**  **end**    **//check sub**  **if(k==1)**  **begin**  **expected=i-j;**  **if(expected != out\_tb)**  **begin**  **error=error+1;**  **end**  **end**    **//check and**  **if(k==2)**  **begin**  **expected=i&j;**  **if(expected != out\_tb)**  **begin**  **error=error+1;**  **end**  **end**    **//check xor**  **if(k==3)**  **begin**  **expected=i^j;**  **if(expected != out\_tb)**  **begin**  **error=error+1;**  **end**  **end**  **//checkDone;**  **go\_tb=0;**  **end**  **end**  **end**  **end**    **endmodule** |

|  |
| --- |
| **system-constraint.xdc** |
| **# Clock input**  **set\_property -dict {PACKAGE\_PIN W5 IOSTANDARD LVCMOS33} [get\_ports {clk\_FPGA}];**  **create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk\_FPGA}];**  **# Button (debouncer and go)**  **set\_property -dict {PACKAGE\_PIN U18 IOSTANDARD LVCMOS33} [get\_ports {button\_clk}]; # Center Button**  **set\_property -dict {PACKAGE\_PIN T17 IOSTANDARD LVCMOS33} [get\_ports {button\_go}]; #Right button**  **# Input Switches**  **set\_property -dict {PACKAGE\_PIN V17 IOSTANDARD LVCMOS33} [get\_ports {Din1[0]}]; # Din10**  **set\_property -dict {PACKAGE\_PIN V16 IOSTANDARD LVCMOS33} [get\_ports {Din1[1]}]; # Din11**  **set\_property -dict {PACKAGE\_PIN W16 IOSTANDARD LVCMOS33} [get\_ports {Din1[2]}]; # Din12**  **set\_property -dict {PACKAGE\_PIN W17 IOSTANDARD LVCMOS33} [get\_ports {Din1[3]}]; # Din13**  **set\_property -dict {PACKAGE\_PIN W15 IOSTANDARD LVCMOS33} [get\_ports {Din2[0]}]; # Din20**  **set\_property -dict {PACKAGE\_PIN V15 IOSTANDARD LVCMOS33} [get\_ports {Din2[1]}]; # Din21**  **set\_property -dict {PACKAGE\_PIN W14 IOSTANDARD LVCMOS33} [get\_ports {Din2[2]}]; # Din22**  **set\_property -dict {PACKAGE\_PIN W13 IOSTANDARD LVCMOS33} [get\_ports {Din2[3]}]; # Din23**  **set\_property -dict {PACKAGE\_PIN R2 IOSTANDARD LVCMOS33} [get\_ports {c\_fpga[1]}]; # c\_fpga0**  **set\_property -dict {PACKAGE\_PIN T1 IOSTANDARD LVCMOS33} [get\_ports {c\_fpga[0]}]; # c\_fpga1**  **# LED Output**  **set\_property -dict {PACKAGE\_PIN U16 IOSTANDARD LVCMOS33} [get\_ports {Dout1[0]}]; # Dout10**  **set\_property -dict {PACKAGE\_PIN E19 IOSTANDARD LVCMOS33} [get\_ports {Dout1[1]}]; # Dout11**  **set\_property -dict {PACKAGE\_PIN U19 IOSTANDARD LVCMOS33} [get\_ports {Dout1[2]}]; # Dout12**  **set\_property -dict {PACKAGE\_PIN V19 IOSTANDARD LVCMOS33} [get\_ports {Dout1[3]}]; # Dout13**  **set\_property -dict {PACKAGE\_PIN W18 IOSTANDARD LVCMOS33} [get\_ports {Dout2[0]}]; # Dout20**  **set\_property -dict {PACKAGE\_PIN U15 IOSTANDARD LVCMOS33} [get\_ports {Dout2[1]}]; # Dout21**  **set\_property -dict {PACKAGE\_PIN U14 IOSTANDARD LVCMOS33} [get\_ports {Dout2[2]}]; # Dout22**  **set\_property -dict {PACKAGE\_PIN V14 IOSTANDARD LVCMOS33} [get\_ports {Dout2[3]}]; # Dout23**  **set\_property -dict {PACKAGE\_PIN P3 IOSTANDARD LVCMOS33} [get\_ports {done\_LED}]; # done LED**  **set\_property -dict {PACKAGE\_PIN P1 IOSTANDARD LVCMOS33} [get\_ports {c\_LED[0]}]; # c LED**  **set\_property -dict {PACKAGE\_PIN L1 IOSTANDARD LVCMOS33} [get\_ports {c\_LED[1]}]; # c LED**  **# 7 Segment Display**  **set\_property -dict {PACKAGE\_PIN W7 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[0]}]; # CA**  **set\_property -dict {PACKAGE\_PIN W6 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[1]}]; # CB**  **set\_property -dict {PACKAGE\_PIN U8 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[2]}]; # CC**  **set\_property -dict {PACKAGE\_PIN V8 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[3]}]; # CD**  **set\_property -dict {PACKAGE\_PIN U5 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[4]}]; # CE**  **set\_property -dict {PACKAGE\_PIN V5 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[5]}]; # CF**  **set\_property -dict {PACKAGE\_PIN U7 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[6]}]; # CG**  **set\_property -dict {PACKAGE\_PIN V7 IOSTANDARD LVCMOS33} [get\_ports {led\_out\_fpga[7]}]; # DP**  **set\_property -dict {PACKAGE\_PIN U2 IOSTANDARD LVCMOS33} [get\_ports {led\_pins\_fpga[0]}]; # AN0**  **set\_property -dict {PACKAGE\_PIN U4 IOSTANDARD LVCMOS33} [get\_ports {led\_pins\_fpga[1]}]; # AN1**  **set\_property -dict {PACKAGE\_PIN V4 IOSTANDARD LVCMOS33} [get\_ports {led\_pins\_fpga[2]}]; # AN2**  **set\_property -dict {PACKAGE\_PIN W4 IOSTANDARD LVCMOS33} [get\_ports {led\_pins\_fpga[3]}]; # AN3** |