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

**Lab \_\_1\_\_ Report**

**Title \_\_\_\_\_\_\_\_\_\_\_\_Factorial Calculator\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester \_\_\_\_\_Fall 2016\_\_\_\_\_\_\_\_ Date \_\_\_\_September 16, 2016\_\_\_**

**by**

**Name \_\_\_\_Nadim Sarras\_\_\_\_\_ SID \_\_008382608\_\_\_\_\_\_\_\_\_\_\_**

**Name \_\_\_\_\_\_Kevin Manan\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ SID**  008355464

**Lab Record**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Performed by (print name)** | **Checked by (print name)** | **Successfully Completed** | **Partially Completed\*** | **Failed or Not Performed\*** |
| KM | NS | X |  |  |
| NS | KM | X |  |  |

**\* Detailed descriptions must be given in the report.**

**San Jose State University**

**Department of Computer Engineering**

**CMPE 140 Lab Report**

**Purpose:**

The purpose of the factorial calculator lab is to review Verilog from the long summer. It is useful to have a quick review of the fundamental of Verilog in order to proceed to the new MIPs framework.

**Materials and Software**

1. Xilinx Vivado 2015.2
2. Sublime Text
3. Xilinx Nexys 4 Board

**Task**

The team had to create an ASM, state transition table, and datapath. After the outlines are complete, the team creates the modules for the data path and test it using a test bench. The control unit was the following step including the top level.

Task 1 Data Path: The data path module was configured of an architecture utilizing the modules; multiplexer, 32 bit register, multiplier, decrement module, comparator, and an output enabled buffer.

Task 2 Data Path + CU Block Diagram: As shown in Figure 5, the connections between the Data Path and the Control Unit are displayed. The Control Unit will send critical control signals to the Data Path that will control the flow of data, these control signals are,  SEL, OE, CNT\_EN, LD\_REG, LD\_CNT. In return the Data Path sends the Control Unit the n > 1 signal which signifies to the Control Unit what state to iterate to next.

Task 3 ASM Chart: The ASM chart displays the states of the design and signifies what operation of data manipulation occurs in every state.

Task 4 Bubble Diagram: This diagram in Figure 6 provides a visual representation of how each control signals the flow of states.

Task 5 Output Table: Displays Control Signal Outputs based on inputs. The output table also displays the Next State logic as determined by the inputs.

Task 6 Verilog Design and Test Bench: An input would be fed into the data path. The input would then be fed into the multiplier module along with an initial value of 1. Afterwards the value would loop back around into the register and into the multiplier while receiving a new decremented input. This would continue until the Control Unit send the Done signal to the Data Path, when the input can no longer be decremented. The Test Bench was designed to test for inputs 1-10 into the design. The waveforms would display the input value, the done signal, and the result. The result displayed by the testbench was then compared to an expected result. If all inputs caused an output that matched the expected result then the test was successful for all cases.

Task 7 FPGA Design: The FPGA was programmed to showcase our architecture. As shown in Figure 2, the slide switches were programmed to be the input. The FPGA board was also programmed to include a GO, reset, and debounce button. After debouncing the FPGA a few times the output was displayed on the LCD, verifying the factorial value of the selected input.

**Discussion and Results**

The test bench for the data path and top level showed that the simulation circuits would work. The implementation to the FPGA board worked and was able to perform 10! without issues.

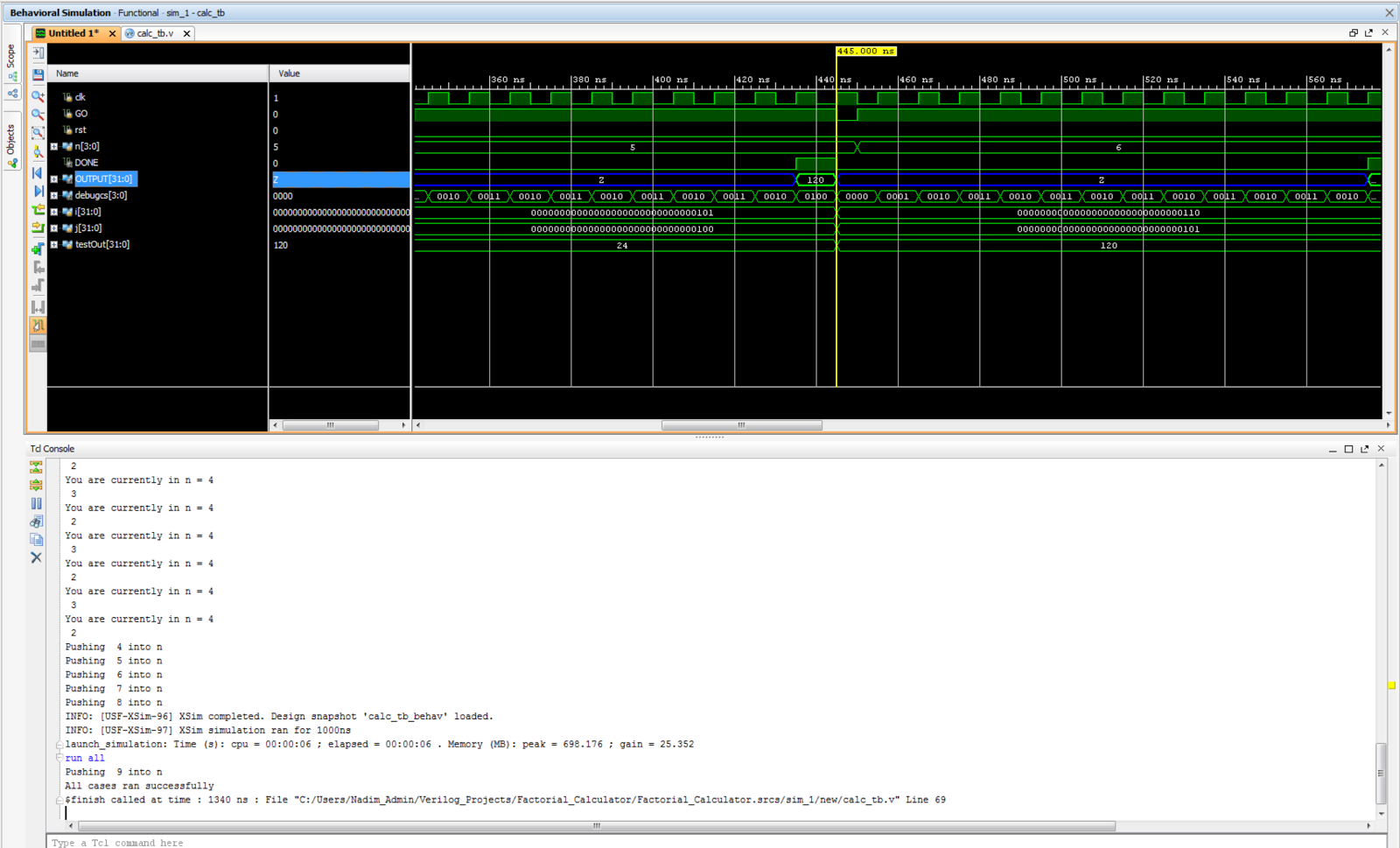


Figure 1: Output Waveform for Top Level Test Bench / with console output.

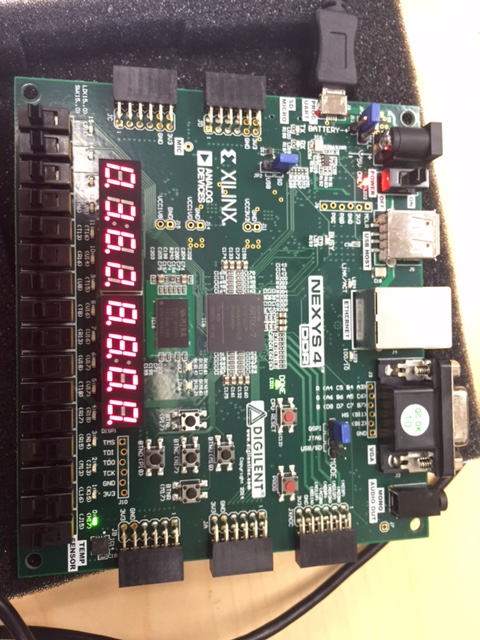


Figure 2: FPGA Board output of factorial calculator, ten factorial selected

**Conclusion**

The lab was a success and we were able to complete the assignment as well as achieve the desired outputs. The lab successfully reviewed our ability in Verilog and the team was able to accomplish the lab with an entire summer without practice.

**Appendix**

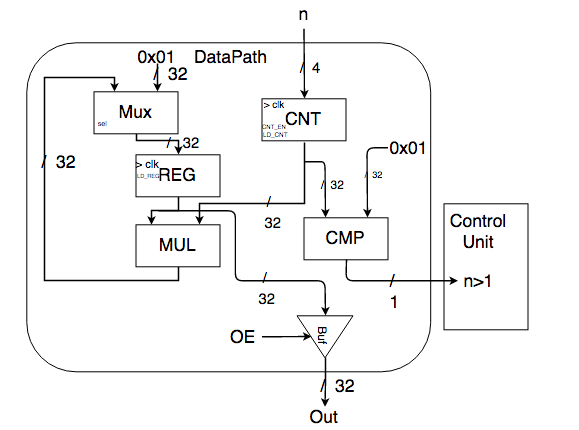


Figure 3: Data path design for Factorial circuit.

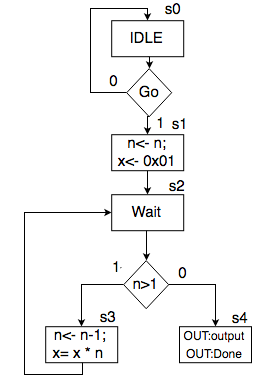


Figure 4: Arithmetic State Machine for the Factorial Circuit

https://lh3.googleusercontent.com/OUS5NKYK-LMiSkAK4xij_mgR3r7Y2QzDy8OgnyfihqT2FaUPDbdyF8ReYV-OHBEVxF5GF0-io-a1TKSJrpyCUI61j3n1-vZJybF0PB3ZgmqsykoJO61D6m6bvioPpS0-GgtllA4

Figure 5: Represents the top level design.

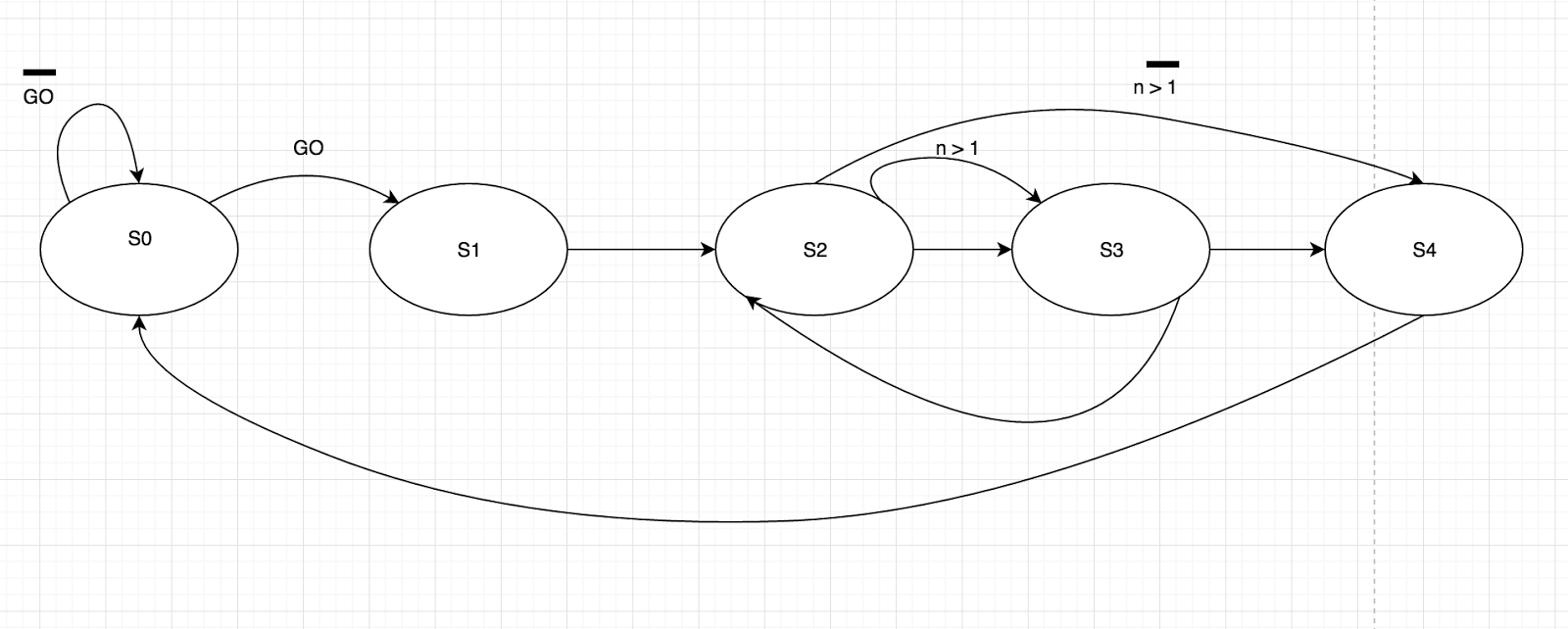


Figure 6: Mealy Machine representation.

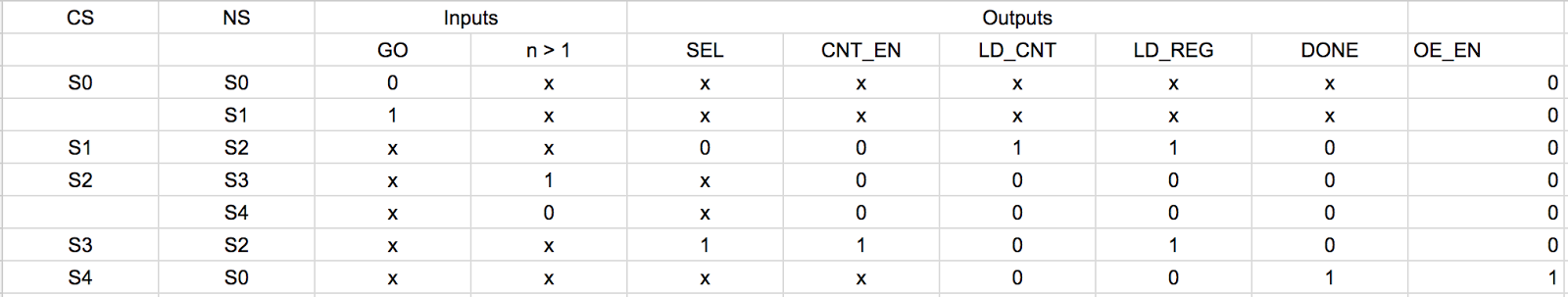


Figure 7: Output Table

**Source Code:**

***Top Level Calculator with FPGA components***

`timescale 1ns / 1ps

module FPGA\_factorial\_calc(

  input clk100MHz,

  input pb,

  input rst,

  output wire [7:0] LEDSEL,

  output wire [7:0] LEDOUT,

   input GO,

   input [3:0] n,

   output wire DONE

);

wire pb\_debounced;

wire clk\_5KHz;

wire NOT\_USED;

wire [3:0] dig0;

wire [3:0] dig1;

wire [3:0] dig2;

wire [3:0] dig3;

wire [3:0] dig4;

wire [3:0] dig5;

wire [3:0] dig6;

wire [3:0] dig7;

wire a0, a1, a2, a3, a4, a5, a6;

wire b0, b1, b2, b3, b4, b5, b6;

wire c0, c1, c2, c3, c4, c5, c6;

wire d0, d1, d2, d3, d4, d5, d6;

wire e0, e1, e2, e3, e4, e5, e6;

wire f0, f1, f2, f3, f4, f5, f6;

wire g0, g1, g2, g3, g4, g5, g6;

wire h0, h1, h2, h3, h4, h5, h6;

wire [31:0] OUTPUT;

wire [3:0] debugcs;

factorial\_calc f1(

   .clk(pb\_debounced),

   .GO(GO),

   .rst(rst),

   .n(n),

   .DONE(DONE),

   .OUTPUT(OUTPUT),

   .debugcs(debugcs)

   );

clk\_gen cuu1(

   .clk100MHz(clk100MHz),

   .rst(rst),

   .clk\_4sec(NOT\_USED),

   .clk\_5KHz(clk\_5KHz)

   );

debounce duu1(

       .pb\_debounced(pb\_debounced),

       .pb(pb),

       .clk(clk\_5KHz)

       );

bin2bcd32 buu1(

        .value(OUTPUT),

        .dig0(dig0),

        .dig1(dig1),

        .dig2(dig2),

        .dig3(dig3),

        .dig4(dig4),

        .dig5(dig5),

        .dig6(dig6),

        .dig7(dig7)

        );

       bcd\_to\_7seg DUT1(

               .BCD(dig0),

               .s0(a0),

               .s1(a1),

               .s2(a2),

               .s3(a3),

               .s4(a4),

               .s5(a5),

               .s6(a6)

               );

       bcd\_to\_7seg DUT2(

               .BCD(dig1),

               .s0(b0),

               .s1(b1),

               .s2(b2),

               .s3(b3),

               .s4(b4),

               .s5(b5),

               .s6(b6)

               );

       bcd\_to\_7seg DUT3(

               .BCD(dig2),

               .s0(c0),

               .s1(c1),

               .s2(c2),

               .s3(c3),

               .s4(c4),

               .s5(c5),

               .s6(c6));

       bcd\_to\_7seg DUT4(

               .BCD(dig3),

               .s0(d0),

               .s1(d1),

               .s2(d2),

               .s3(d3),

               .s4(d4),

               .s5(d5),

               .s6(d6));

       bcd\_to\_7seg DUT5(

               .BCD(dig4),

               .s0(e0),

               .s1(e1),

               .s2(e2),

               .s3(e3),

               .s4(e4),

               .s5(e5),

               .s6(e6));

       bcd\_to\_7seg DUT6(

               .BCD(dig5),

               .s0(f0),

               .s1(f1),

               .s2(f2),

               .s3(f3),

               .s4(f4),

               .s5(f5),

               .s6(f6));

       bcd\_to\_7seg DUT7(

               .BCD(dig6),

               .s0(g0),

               .s1(g1),

               .s2(g2),

               .s3(g3),

               .s4(g4),

               .s5(g5),

               .s6(g6));

       bcd\_to\_7seg DUT8(

               .BCD(dig7),

               .s0(h0),

               .s1(h1),

               .s2(h2),

               .s3(h3),

               .s4(h4),

               .s5(h5),

               .s6(h6));

led\_mux MUXDUT(

       .clk(clk\_5KHz),

       .rst(rst),

       .LED0({1'b0,h6,h5,h4,h3,h2,h1,h0}), // leftmost digit

       .LED1({1'b0,g6,g5,g4,g3,g2,g1,g0}),

       .LED2({1'b0,f6,f5,f4,f3,f2,f1,f0}),

       .LED3({1'b0,e6,e5,e4,e3,e2,e1,e0}),

       .LED4({1'b0,d6,d5,d4,d3,d2,d1,d0}),

       .LED5({1'b0,c6,c5,c4,c3,c2,c1,c0}),

       .LED6({1'b0,b6,b5,b4,b3,b2,b1,b0}),

       .LED7({1'b0,a6,a5,a4,a3,a2,a1,a0}), // rightmost digit

       .LEDSEL(LEDOUT),

       .LEDOUT(LEDSEL)

       );

endmodule

***Clock Generator Module***

module clk\_gen(clk100MHz, rst, clk\_4sec, clk\_5KHz);

input clk100MHz, rst;

output clk\_4sec, clk\_5KHz;

reg clk\_4sec, clk\_5KHz;

integer count, count1;

always@(posedge clk100MHz)

begin

if(rst)

begin

count = 0;

count1 = 0;

clk\_4sec = 0;

clk\_5KHz =0;

end

else

begin

if(count == 200000000)

begin

clk\_4sec = ~clk\_4sec;

count = 0;

end

if(count1 == 10000)

begin

clk\_5KHz = ~clk\_5KHz;

count1 = 0;

end

count = count + 1;

count1 = count1 + 1;

end

end

endmodule

***Debounce Module***

module debounce #(parameter width = 16) (

output reg pb\_debounced,

input wire pb,

input wire clk

);

localparam shift\_max = (2\*\*width)-1;

reg [width-1:0] shift;

always @ (posedge clk)

begin

shift[width-2:0] <= shift[width-1:1];

shift[width-1] <= pb;

if (shift == shift\_max)

pb\_debounced <= 1'b1;

else

pb\_debounced <= 1'b0;

end

endmodule

***Binary to Decimal Converter***

module bin2bcd32(

input wire [31:0] value,

output wire [3:0] dig0,

output wire [3:0] dig1,

output wire [3:0] dig2,

output wire [3:0] dig3,

output wire [3:0] dig4,

output wire [3:0] dig5,

output wire [3:0] dig6,

output wire [3:0] dig7

);

assign dig0 = value % 10;

assign dig1 = (value / 10) % 10;

assign dig2 = (value / 100) % 10;

assign dig3 = (value / 1000) % 10;

assign dig4 = (value / 10000) % 10;

assign dig5 = (value / 100000) % 10;

assign dig6 = (value / 1000000) % 10;

assign dig7 = (value / 10000000) % 10;

endmodule

***BCD to 7 segment display X 8***

module bcd\_to\_7seg(BCD, s0, s1, s2, s3, s4, s5, s6);

output s0, s1, s2, s3, s4, s5, s6;

input [3:0] BCD;

reg s0, s1, s2, s3, s4, s5, s6;

always @ (BCD)

begin // BCD to 7-segment decoding

case (BCD) // s0 - s6 are active low

4'b0000: begin s0=0; s1=0; s2=0; s3=1; s4=0; s5=0; s6=0; end

4'b0001: begin s0=1; s1=0; s2=1; s3=1; s4=0; s5=1; s6=1; end

4'b0010: begin s0=0; s1=1; s2=0; s3=0; s4=0; s5=1; s6=0; end

4'b0011: begin s0=0; s1=0; s2=1; s3=0; s4=0; s5=1; s6=0; end

4'b0100: begin s0=1; s1=0; s2=1; s3=0; s4=0; s5=0; s6=1; end

4'b0101: begin s0=0; s1=0; s2=1; s3=0; s4=1; s5=0; s6=0; end

4'b0110: begin s0=0; s1=0; s2=0; s3=0; s4=1; s5=0; s6=0; end

4'b0111: begin s0=1; s1=0; s2=1; s3=1; s4=0; s5=1; s6=0; end

4'b1000: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=0; s6=0; end

default begin s0=1; s1=0; s2=1; s3=0; s4=0; s5=0; s6=0; end

endcase

end

endmodule

***LED MUX***

module led\_mux (

input wire clk,

input wire rst,

input wire [7:0] LED0, // leftmost digit

input wire [7:0] LED1,

input wire [7:0] LED2,

input wire [7:0] LED3,

input wire [7:0] LED4,

input wire [7:0] LED5,

input wire [7:0] LED6,

input wire [7:0] LED7, // rightmost digit

output wire [7:0] LEDSEL,

output wire [7:0] LEDOUT

);

reg [2:0] index;

reg [15:0] led\_ctrl;

assign {LEDOUT, LEDSEL} = led\_ctrl;

always@(posedge clk)

begin

index <= (rst) ? 3'd0 : (index + 3'd1);

end

always @(index, LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7)

begin

case(index)

3'd0: led\_ctrl <= {8'b11111110, LED7};

3'd1: led\_ctrl <= {8'b11111101, LED6};

3'd2: led\_ctrl <= {8'b11111011, LED5};

3'd3: led\_ctrl <= {8'b11110111, LED4};

3'd4: led\_ctrl <= {8'b11101111, LED3};

3'd5: led\_ctrl <= {8'b11011111, LED2};

3'd6: led\_ctrl <= {8'b10111111, LED1};

3'd7: led\_ctrl <= {8'b01111111, LED0};

default: led\_ctrl <= {8'b11111111, 8'hFF};

endcase

end

endmodule

***Top Level Calculator***

`timescale 1ns / 1ps

module factorial\_calc(

   input clk,

   input GO,

   input rst,

   input [3:0] n,

   output wire DONE,

   output wire [31:0] OUTPUT,

   output wire [3:0] debugcs

   );

wire SEL;

wire CNT\_EN;

wire LD\_CNT;

wire LD\_REG;

wire compare;

dp d1(

   .n(n),

   .clk(clk),

   .out(OUTPUT),

   .Done(DONE),

   .SEL(SEL),

   .CNT\_EN(CNT\_EN),

   .LD\_CNT(LD\_CNT),

   .LD\_REG(LD\_REG),

   .OE\_EN(DONE),

   .compare(compare)

);

CU c1(

   .rst(rst),

   .clk(clk),

   .GO(GO),

   .n\_cmp\_1(compare),

   .SEL(SEL),

   .CNT\_EN(CNT\_EN),

   .LD\_CNT(LD\_CNT),

   .LD\_REG(LD\_REG),

   .DONE(DONE),

   .debugcs(debugcs)

);

endmodule

***Control Unit***

`timescale 1ns / 1ps

module CU(

   input clk,

   input rst,

   input GO,

   input n\_cmp\_1,

   output reg SEL,

   output reg CNT\_EN,

   output reg LD\_CNT,

   output reg LD\_REG,

   output reg DONE,

   output wire [3:0] debugcs

   );

reg [3:0] CS;

reg [3:0] NS;

assign debugcs = CS;

always@(CS, GO, n\_cmp\_1)

begin

   case(CS)

       3'b000: NS = ((GO) ? 3'b001 : 3'b000);

       3'b001: NS = 3'b010;

       3'b010: NS = ((n\_cmp\_1) ? 3'b011 : 3'b100);

       3'b011: NS = 3'b010;

       3'b100: NS = 3'b000;

   endcase

end

always @(posedge clk, posedge rst)

begin

   CS = NS;

   if (rst)

   begin

   CS = 3'b000;

   end

end

always@(CS)

begin

   case(CS)

       3'b000: begin

               DONE = 1'b0;

               end

       3'b001: begin

               SEL = 1'b0;

               CNT\_EN = 1'b0;

               LD\_CNT = 1'b1;

               LD\_REG = 1'b1;

               DONE = 1'b0;

               end

       3'b010: begin

               CNT\_EN = 1'b0;

               LD\_CNT = 1'b0;

               LD\_REG = 1'b0;

               DONE = 1'b0;

               end

       3'b011: begin

               SEL = 1'b1;

               CNT\_EN = 1'b1;

               LD\_CNT = 1'b0;

               LD\_REG = 1'b1;

               DONE = 1'b0;

               end

       3'b100: begin

               LD\_CNT = 1'b0;

               LD\_REG = 1'b0;

               DONE = 1'b1;

               end

   endcase

end

endmodule

***Data Path***

module dp(

input [3:0] n,

input clk, SEL, CNT\_EN, LD\_CNT, LD\_REG, OE\_EN,

output [31:0] out,

output compare, Done

);

wire [31:0] mulOut;

wire [31:0] muxOut;

wire [31:0] regOut;

wire [31:0] cntOut;

MUX m1(

.a(1'b1), .b(mulOut),

.sel(SEL),

.x(muxOut)

);

REG r1(

.d(muxOut),

.clk(clk), .LD\_REG(LD\_REG),

.q(regOut)

);

MUL mul1(

.x(regOut), .y(cntOut),

.z(mulOut)

);

CNT c1(

.clk(clk), .CNT\_EN(CNT\_EN), .LD\_CNT(LD\_CNT),

.n(n),

.N(cntOut)

);

OE o1(

.a(regOut), .en(OE\_EN),

.b(out)

);

CMP comp1(

.A(cntOut), .B(1'b1),

.GT(compare));

endmodule

***MUX***

module MUX(

input [31:0] a, b,

input sel,

output reg [31:0] x);

always @ (\*) begin

if(sel)begin

x =b;

end else begin

x= a;

end

end

endmodule

***REG***

module REG(

input [31:0] d,

input clk, LD\_REG,

output reg [31:0] q

);

always @ ( posedge clk) begin

if(LD\_REG)begin

q<= d; //set the output with the input

end else begin

q<= q;

end

end

endmodule

***MUL***

module MUL(

input [31:0] x, y,

output reg [31:0] z

);

always @ (\*)begin

z= x\*y;

end

endmodule

***CNT***

module CNT(

input clk, CNT\_EN, LD\_CNT,

input [3:0] n,

output reg [31:0] N

);

always@(posedge clk)begin

if(LD\_CNT)

begin

N<= n;

end

else

begin

if(CNT\_EN)

begin

N<= N -1;

// N= N-1;

end

if(!CNT\_EN)

begin

N<= N;

end

end

end

endmodule

***OE Buffer***

module OE(

input [31:0] a,

input en,

output reg [31:0] b

);

always@ (\*)begin

if(en)begin

b=a;

end else begin

b= 32'bz;

end

end

endmodule

***Comparator Module***

module CMP (

input [31:0] A,

input [31:0] B,

output reg GT

);

always @ (\*)begin

//A= n  && B = 1

if(A>B)begin

GT = 1'b1;

end else begin

GT=1'b0;

end

end

endmodule

***FPGA Constraints***

# Based on Digilent's Nexys 4 DDR reference XDC,

# with extra pins not used in the labs removed.

# Clock signal (100 MHz)

set\_property -dict { PACKAGE\_PIN E3    IOSTANDARD LVCMOS33 } [get\_ports { clk100MHz }]; #IO\_L12P\_T1\_MRCC\_35 Sch=clk100mhz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports {clk100MHz}];

# Slide switches

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

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

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

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

# LEDs

set\_property -dict { PACKAGE\_PIN H17   IOSTANDARD LVCMOS33 } [get\_ports { DONE }]; #IO\_L18P\_T2\_A24\_15 Sch=led[0]

# 7-segment display

# Segment cathodes (common to all digits)

set\_property -dict { PACKAGE\_PIN K13   IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[0] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN K16   IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[1] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN P15   IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[2] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN L18   IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[3] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

set\_property -dict { PACKAGE\_PIN R10   IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[4] }]; #IO\_25\_14 Sch=cb

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

set\_property -dict { PACKAGE\_PIN T10   IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[6] }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

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

# Segment cathodes (common to all digits)

# Anodes per digit

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

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

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

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

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

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

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

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

# Pushbuttons

set\_property -dict {PACKAGE\_PIN N17 IOSTANDARD LVCMOS33} [get\_ports {rst}];

# Active high

# Active low

set\_property -dict { PACKAGE\_PIN M18   IOSTANDARD LVCMOS33 } [get\_ports { pb }]; #IO\_L4N\_T0\_D05\_14 Sch=btnu

set\_property -dict { PACKAGE\_PIN P17   IOSTANDARD LVCMOS33 } [get\_ports { GO }]; #IO\_L12P\_T1\_MRCC\_14 Sch=btnl

#=============================================================================

***Top Level Calculator Test Bench***

`timescale 1ns / 1ps

module calc\_tb;

reg clk;

reg GO;

reg rst;

reg [3:0] n;

wire DONE;

wire [31:0] OUTPUT;

wire [3:0] debugcs;

factorial\_calc DUT(.clk(clk), .GO(GO), .rst(rst), .n(n), .DONE(DONE), .OUTPUT(OUTPUT), .debugcs(debugcs));

integer i, j;

reg [31:0] testOut;

initial

begin

clk = 1'b0;

rst = 1'b1;

#5;

rst = 1'b0; //cycles thru reset

GO= 1'b0;

end

initial forever #5 clk = ~clk;

initial begin

#5; //waiting GO isn't set yet.

for(i=0; i<11; i=i+1)begin

#5;

n <= i;

GO <= 1'b1; //allows system to go to s1.

$display("Pushing %d into n", n);

// $display("Currently in State: %i", CS);

#10;

while(!DONE)begin

   if(n == 4'b0100)

   begin

   $display("You are currently in n = 4");

   $display("%d",debugcs);

   end

   #10;

//this is waiting for the circuit to be done. ASM s0->s4; when s4 DONE should be 1 & the loop will break.

end

testOut = 1'b1;

for(j= 0; j<n; j=j+1)begin

testOut = testOut \* (n-j); //ex. 5-0, 5-1, 5-2, ..., 5-4

end

if(OUTPUT != testOut)begin

$display("Your output is wrong. You got %i when it's suppose to be %i", OUTPUT, testOut);

$stop;

end

GO = 1'b0;

end

$display("All cases ran successfully");

$finish;

end

endmodule

***Data Path TestBench***

`timescale 1ns /1ps

module DP\_tb;

reg [7:0] N\_tb;

reg Load\_Reg\_tb, Load\_Cnt\_tb, En\_tb, Sel\_tb, OE\_tb, Clk\_tb;

wire GT\_tb;

wire [7:0] Output\_tb;

wire Done;

dp DUT(N\_tb, Clk\_tb, Sel\_tb, En\_tb, Load\_Cnt\_tb, Load\_Reg\_tb, OE\_tb, Output\_tb, GT\_tb, Done);

initial

begin

Clk\_tb <= 1;

   //s0

   N\_tb <= 2'b00;

   Load\_Reg\_tb <= 0;

   Load\_Cnt\_tb <= 0;

   En\_tb <= 0;

   Sel\_tb <= 0;

   #10;

   //s1

   N\_tb <= 3'b100;

   Load\_Reg\_tb <= 1;

   Load\_Cnt\_tb <= 1;

   En\_tb <= 0;

   Sel\_tb <= 0;

   OE\_tb <= 0;

   #10;

   //s2

   Load\_Reg\_tb <= 0;

   Load\_Cnt\_tb <= 0;

   En\_tb <= 0;

   Sel\_tb <= 0;

   OE\_tb <= 0;

   #10;

   while (GT\_tb == 1)

   begin

       //s3

       Load\_Reg\_tb <= 1;

       Load\_Cnt\_tb <= 0;

       En\_tb <= 1;

       Sel\_tb <= 1;

       OE\_tb <= 0;

       #10;

       //s2

       Load\_Reg\_tb <= 0;

       Load\_Cnt\_tb <= 0;

       En\_tb <= 0;

       Sel\_tb <= 0;

       OE\_tb <= 0;

       #10;

   end

//s4

Load\_Reg\_tb <= 0;

Load\_Cnt\_tb <= 0;

En\_tb <= 0;

Sel\_tb <= 0;

OE\_tb <= 1;

#10;

#0 $stop;

#0 $finish;

end

always #5 Clk\_tb = ~Clk\_tb;

endmodule