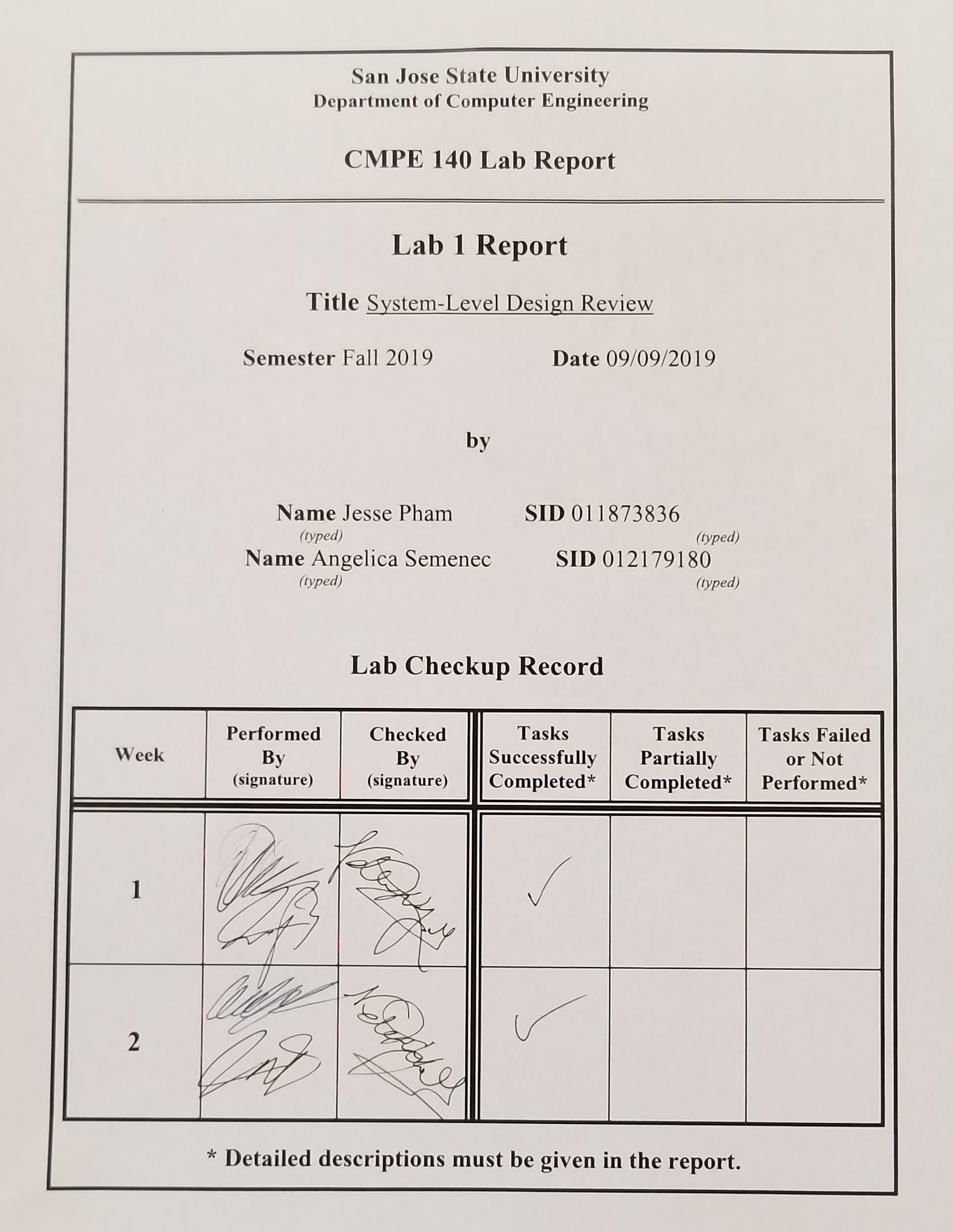
****

**Purpose**

The purpose of this assignment is to review system-level design through Verilog and the Basys-3 FPGA by creating a logical circuit that computes the factorial product of the input, displaying the given factorial number to the seven-segment FPGA display, and verifying its validity using waveform simulations.

**Design Methodology**

Using the six modules provided in the lab manual (register, counter, multiplexer, buffer, comparator, and multiplier), the Datapath (DP) and the Control Unit (CU) were designed. The CU has five inputs to the DP (EN, load\_cnt, load\_reg, sel0, and sel1), and two outputs from the DP (ERROR and DONE). The Algorithmic State Machine (ASM) was then constructed to determine which signals will be active and inactive during each state. The ASM chart is shown in *Figure 1* below. The ASM provides a reference to coding the CU.v file and the DP design, shown in *Figure 3,* provides a reference to coding the DP.v file. Since the highest valid value in the design is 12, the output is required to have 32 bits. If any number larger than 12 is input, the ERROR flag will be set and the program will return to the idle state, state 0. Due to only having 4 digits present on the seven-segment display, the HILO\_MUX.v provides the method of displaying the 8 digits by allowing the user to select either the high bits or the low bits to be displayed. The right-most switch on the FPGA board (V17) has been programmed to control the HILO\_sel input. When the input is LOW, the 4-digit display will project the lower 16 bits of the value as 4 hexadecimal values, however, when the input is HIGH, the upper 16 bits will be displayed. The ERROR flag, DONE flag, as well as the factorial functionality works as expected with the exception of 0!. Switches W15, W17, W16, and V16 were used to provide a value input to the system.

**Tasks Successfully Accomplished**

1. Design the system’s datapath using the six building blocks (counter, register, comparator, multiplier, multiplexer, and buffer).
2. Draw the two-piece system block diagram which shows all the relevant signals for the Control Unit and the Datapath
3. Create the ASM chart to describe the operations of the datapath
4. Draw the bubble diagram to show the Next State Logic of the control unit
5. Create an output table showing the output logic of the control unit
6. Create the Verilog Design code for the factorial accelerator
7. Create the testbench file and verified the code

**Tasks Not Completed**

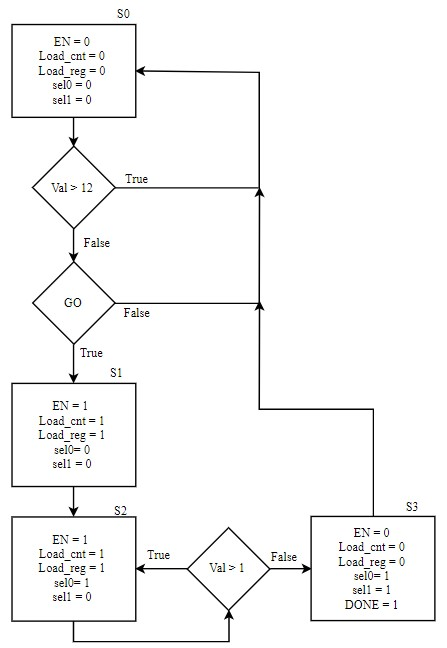
1. Zero Factorial
   1. The expected results for an input of 0 was not accurately met with the experimental result. When using 0 as an input, the expected result is 1, however, the actual results our FPGA produced was a 0. This error is believed to lie within the Control Unit because as the input passes through to the multiply module, it multiplies 0 and 1 and produces the value 0, which is stored into the register and outputs it through the buffer. The State Machine states to keep the control signal *sel0* as 1 to output the value in the register, however, *sel0* should be set to 0 in order to output a value of 1. This is an edge case that was not discovered in time before the demonstration.

**Conclusion**

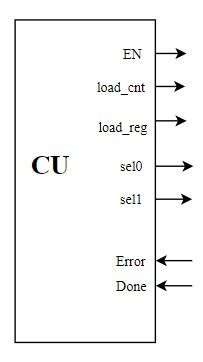
The factorial functionality was working as expected with the exception of 0!. This value did not consistently display the result to be 1, but instead displayed the result as a value of 0 approximately half of the time. The datapath works as intended, however, there were some complications regarding the control unit which may be the cause of the incorrect output for the input value of 0!. Overall, this lab utilizes the concepts of system-level design and provides a thorough opportunity for students to review the concepts as well as Verilog programming. Although there were complications within the experiment, the experiment was fully discussed and exercised to have a better understanding of system-level design methodologies.

**Appendix**

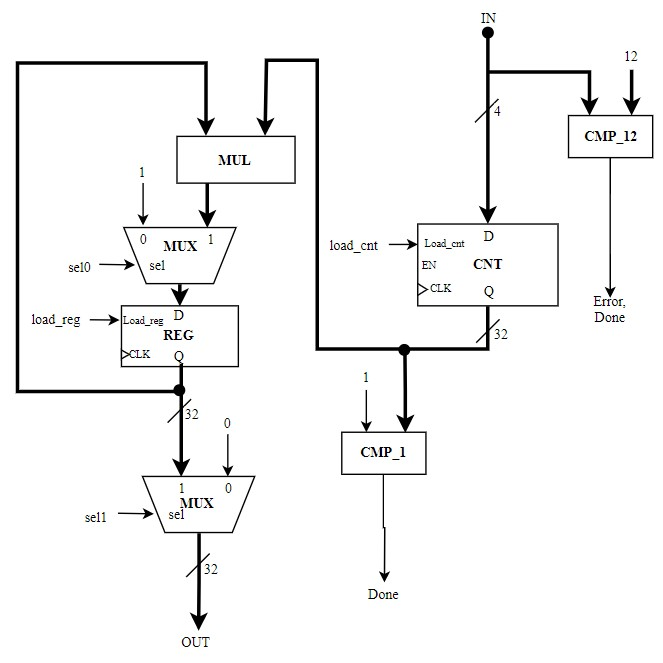
1. **Diagrams and Tables**

****

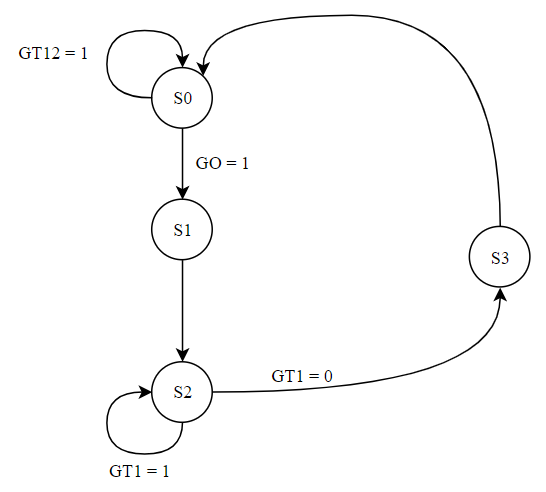
*Figure 1: Displaying the ASM Chart, describing the cycle-by-cycle operations of the datapath*

**

*Figure 2: Displaying the input and output signals of the Control Unit*

**

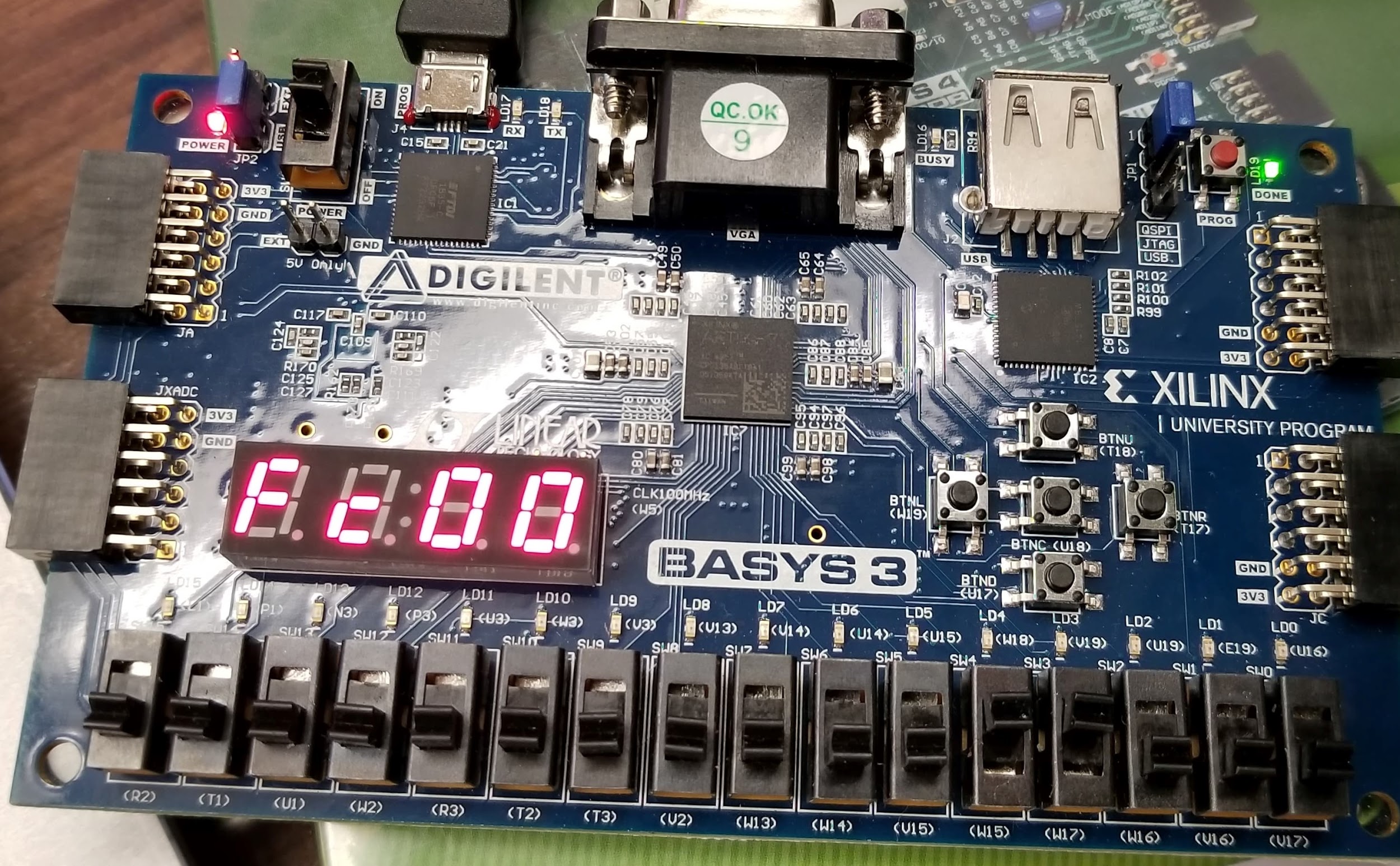
*Figure 3: Displaying the parts, signals, and paths of the Datapath section in the lab experiment*

****

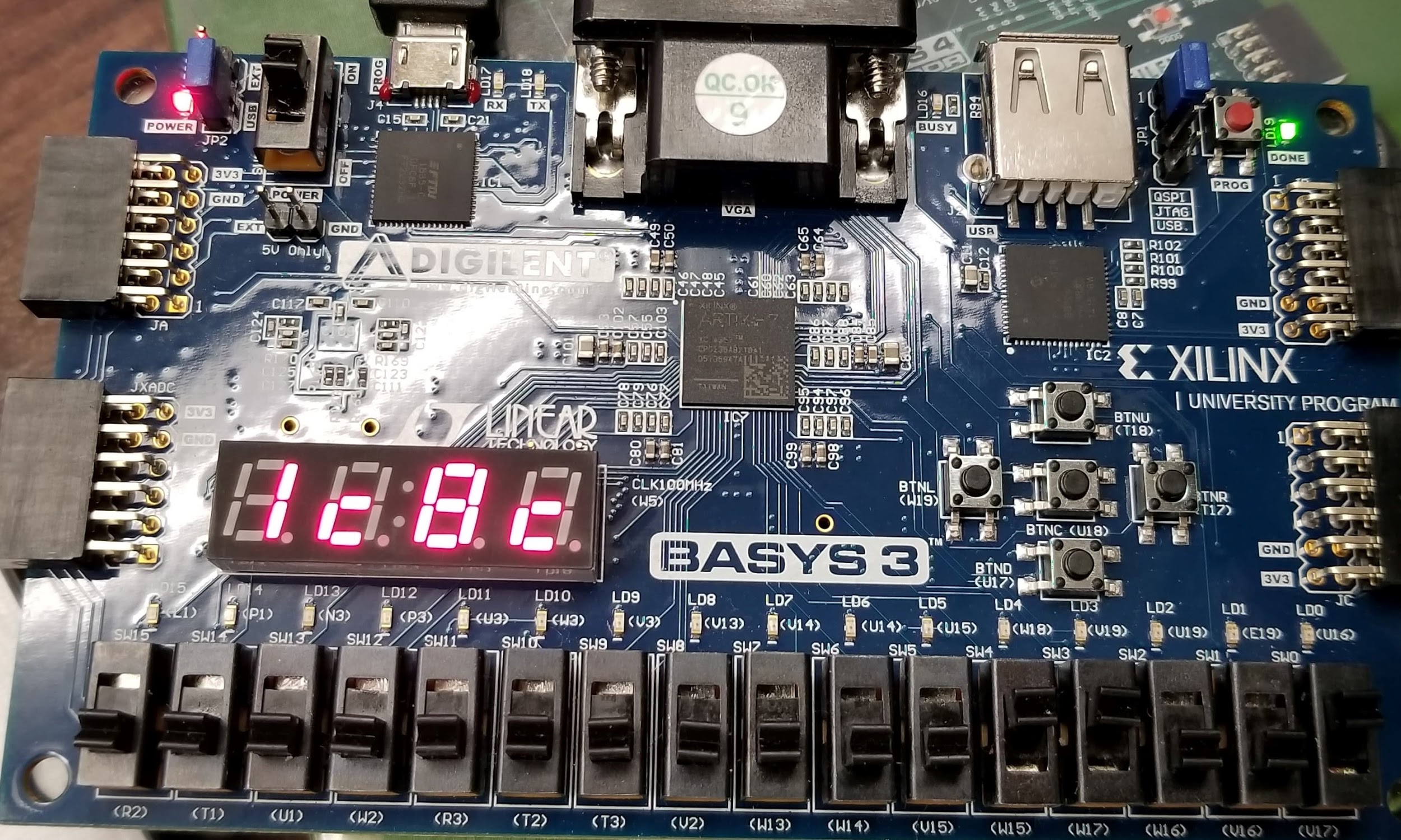
*Figure 4: Displaying the bubble diagram for the next state logic of the control unit*

Table 1: Displaying the output logic for the next state logic of the control unit

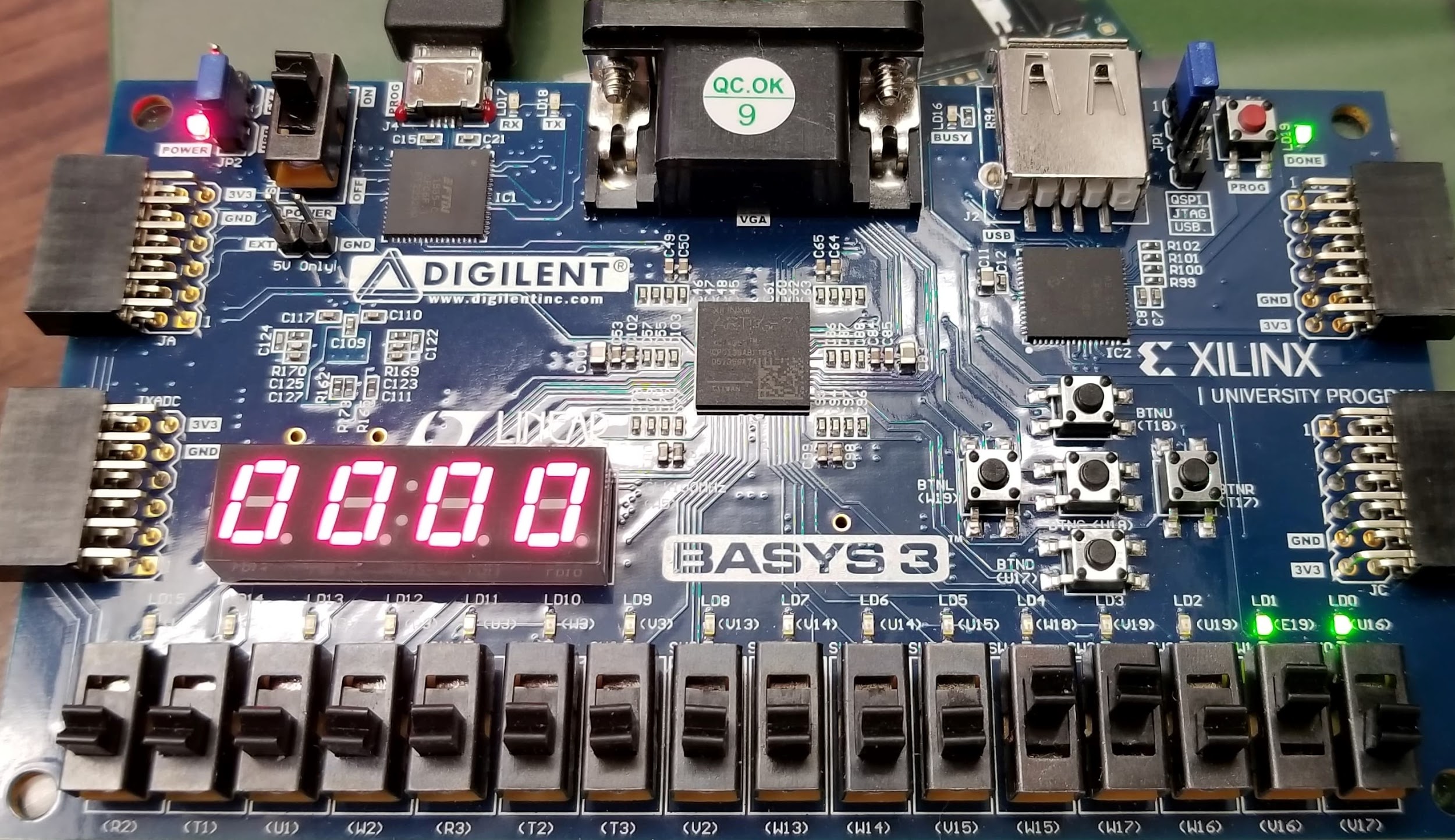
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **State** | **Next State** | **GO** | **GT1** | **GT12** | **en** | **load\_cnt** | **load\_reg** | **sel0** | **sel1** | **done** | **error** |
| 0 | 0 | 0 | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | x | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | x | x | x | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 3 | x | 0 | x | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | 2 | x | 1 | x | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 3 | 0 | x | x | x | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

**

*Figure 5: Displaying the lower four digits of 12!. Input bits B3, B2, B1, and B0 are assigned to switches W15, W17, W16, and V16 respectively, which uses the bit value 1100. The output is an expected 0xFC00.*

**

*Figure 6: Displaying the upper four digits of 12!. The output is the expected value of 0x1C8C.*

**

*Figure 7: Displaying the ERROR and DONE flag being triggered when entering an input of 1101 (1310).*

1. **Source Code**

|  |
| --- |
| **factorial\_FPGA.v** |
| module factorial\_FPGA(  input wire [3:0] IN,  input wire GO, rst, clk100MHz, HILO\_sel,  output done, error,  output wire [3:0] LEDSEL,  output wire [6:0] LEDOUT  );  wire clk\_5KHz;  wire dGO;  wire [31:0] OUT;  wire [3:0] dig0, dig1, dig2, dig3, dig4, dig5, dig6, dig7;  wire [3:0] HEX3, HEX2, HEX1, HEX0;  wire [6:0] LED3, LED2, LED1, LED0;  clk\_gen CLK (  .clk50MHz(clk100MHz),  .rst(rst),  .clk\_5KHz(clk\_5KHz));    button\_debouncer BD(.clk(clk\_5KHz), .button(GO), .debounced\_button(dGO));  CU\_DP the\_controlunit(.IN(IN), .GO(GO), .RST(rst), .clk(clk\_5KHz), .OUT(OUT), .done(done), .error(error));  bin2hex32 U1(  .value(OUT),  .dig0(dig0),  .dig1(dig1),  .dig2(dig2),  .dig3(dig3),  .dig4(dig4),  .dig5(dig5),  .dig6(dig6),  .dig7(dig7)  );  HILO\_MUX DUT(  .HI\_dig3(dig7),  .HI\_dig2(dig6),  .HI\_dig1(dig5),  .HI\_dig0(dig4),  .LO\_dig3(dig3),  .LO\_dig2(dig2),  .LO\_dig1(dig1),  .LO\_dig0(dig0),  .HILO\_sel(HILO\_sel),  .HW\_dig3(HEX3),  .HW\_dig2(HEX2),  .HW\_dig1(HEX1),  .HW\_dig0(HEX0)  );  hex2led U\_LD\_3(  .number(HEX3),  .s0(LED3[0]),  .s1(LED3[1]),  .s2(LED3[2]),  .s3(LED3[3]),  .s4(LED3[4]),  .s5(LED3[5]),  .s6(LED3[6])  );  hex2led U\_LD\_2(  .number(HEX2),  .s0(LED2[0]),  .s1(LED2[1]),  .s2(LED2[2]),  .s3(LED2[3]),  .s4(LED2[4]),  .s5(LED2[5]),  .s6(LED2[6])  );  hex2led U\_LD\_1(  .number(HEX1),  .s0(LED1[0]),  .s1(LED1[1]),  .s2(LED1[2]),  .s3(LED1[3]),  .s4(LED1[4]),  .s5(LED1[5]),  .s6(LED1[6])  );  hex2led U\_LD\_0(  .number(HEX0),  .s0(LED0[0]),  .s1(LED0[1]),  .s2(LED0[2]),  .s3(LED0[3]),  .s4(LED0[4]),  .s5(LED0[5]),  .s6(LED0[6])  );  LED\_MUX U3(  .clk(clk\_5KHz),  .rst(rst),  .LED0(LED0),  .LED1(LED1),  .LED2(LED2),  .LED3(LED3),  .LEDOUT(LEDOUT),  .LEDSEL(LEDSEL));  endmodule |

|  |
| --- |
| **CU\_DP.v** |
| module CU\_DP(  input [31:0] IN,  input GO, RST, clk,  output [31:0] OUT,  output done, error  );  wire GT1, GT12, en, load\_reg, load\_cnt, sel0, sel1;  CU control\_unit(.GO(GO), .GT1(GT1), .GT12(GT12), .clk(clk), .rst(RST),  .en(en), .load\_reg(load\_reg), .load\_cnt(load\_cnt), .sel0(sel0), .sel1(sel1),  .done(done), .error(error));    DP datapath(.IN(IN), .clk(clk), .EN(en), .load\_cnt(load\_cnt), .load\_reg(load\_reg),  .sel0(sel0), .sel1(sel1), .OUT(OUT), .GT1(GT1), .GT12(GT12));    endmodule |

|  |
| --- |
| **CU.v** |
| module CU(  input GO, GT1, GT12, clk, rst,  output reg en, load\_cnt, load\_reg, sel0, sel1, done, error  );  reg [1:0] state, next\_state;  initial begin  next\_state = 0;  end  always @(state, GO, GT1, GT12) begin  case(state)  0: begin  if (GO == 1) begin  done = 0;  error = 0;  next\_state = 1;  end  else next\_state = 0;  if (GT12 == 1) begin  error = 1;  done = 1;  next\_state = 0;  end  end  1: begin  next\_state = 2;  end  2: begin  if (GT1 == 0) begin  next\_state = 3;  end  else if (GT1 == 1) begin  next\_state = 2;  end  end  3: begin  next\_state = 0;  done = 1;  end  endcase  end  always @(posedge clk, posedge rst) begin  if (rst == 1) begin  state <= 0;  end  else begin  state <= next\_state;  end  end  always @(state) begin  case(state)  0: begin  en = 0;  load\_cnt = 0;  load\_reg = 0;  sel0 = 0;  sel1 = 1;  end  1: begin  en = 1;  load\_cnt = 1;  load\_reg = 1;  sel0 = 0;  sel1 = 1;  end  2: begin  en = 1;  load\_cnt = 0;  load\_reg = 1;  sel0 = 1;  sel1 = 1;  end  3: begin  en = 0;  load\_cnt = 0;  load\_reg = 0;  sel0 = 1;  sel1 = 1;  end  endcase  end  endmodule |

|  |
| --- |
| **DP.v** |
| module DP(  input wire [31:0] IN,  input wire clk, EN, load\_cnt, load\_reg, sel0, sel1,  output wire [31:0] OUT,  output wire GT1, GT12  );  wire [31:0] Q\_count, Q\_reg, Q\_mux, Q\_mul;  count #(32) counter(.D(IN), .load\_cnt(load\_cnt), .en(EN), .clk(clk), .Q(Q\_count));  mux #(32) reg\_mux(.A(1), .B(Q\_mul), .sel(sel0), .OUT(Q\_mux));  multiply #(32) multiplier(.X(Q\_reg), .Y(Q\_count), .Z(Q\_mul));  register #(32) register0(.D(Q\_mux), .clk(clk), .load\_reg(load\_reg), .Q(Q\_reg));  compare #(32) compare\_1(.A(Q\_count), .B(1), .GT(GT1));  buffer #(32) buff(.A(0), .B(Q\_reg), .sel(sel1), .OUT(OUT));  compare #(32) compare\_12(.A(IN), .B(12), .GT(GT12));  endmodule |

|  |
| --- |
| **count.v** |
| module count  #(parameter WIDTH = 4)(  input [WIDTH - 1:0] D,  input load\_cnt, en, clk,  output reg [WIDTH - 1:0] Q  );  always @ (posedge clk)  if (en == 1) begin  if (load\_cnt == 0) begin  Q <= Q - 1;  end  else begin  Q <= D;  end  end  else begin  Q <= Q;  end  endmodule |

|  |
| --- |
| **mux.v** |
| module mux #(parameter W = 4)(  input [W-1:0] A, B,  input sel,  output reg [W-1:0] OUT  );  always @ (A, B, sel)  begin  case (sel)  1'b0: OUT = A;  1'b1: OUT = B;  endcase  end    endmodule |

|  |
| --- |
| **multiply.v** |
| module multiply  #(parameter WIDTH = 4)(  input [WIDTH - 1:0] X, Y,  output reg [WIDTH - 1:0] Z  );  always @ (X or Y) begin  Z = X \* Y;  end  endmodule |

|  |
| --- |
| **register.v** |
| module register #(parameter W = 4)(  input [W-1:0] D,  input clk, load\_reg,  output reg [W-1:0] Q  );    always @(posedge clk)  if (load\_reg == 1) begin  Q <= D;  end  endmodule |

|  |
| --- |
| **compare.v** |
| module compare  #(parameter WIDTH = 4)(  input [WIDTH - 1:0] A, B,  output reg GT  );  always @ (A or B) begin  if (A > B) begin  GT = 1;  end  else begin  GT = 0;  end  end  endmodule |

|  |
| --- |
| **buffer.v** |
| module buffer #(parameter W = 4)(  input [W-1:0] A, B,  input sel,  output reg [W-1:0] OUT  );  always @ (A, B, sel)  begin  case (sel)  1'b0: OUT = A;  1'b1: OUT = B;  endcase  end    endmodule |

|  |
| --- |
| **constraints.xdc** |
| ## This file is a general .xdc for the Basys3 rev B board  ## Clock signal  set\_property PACKAGE\_PIN W5 [get\_ports clk100MHz]  set\_property IOSTANDARD LVCMOS33 [get\_ports clk100MHz]  create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk100MHz]  ## Switches  set\_property PACKAGE\_PIN V17 [get\_ports HILO\_sel]  set\_property IOSTANDARD LVCMOS33 [get\_ports HILO\_sel]  set\_property PACKAGE\_PIN V16 [get\_ports {IN[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {IN[0]}]  set\_property PACKAGE\_PIN W16 [get\_ports {IN[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {IN[1]}]  set\_property PACKAGE\_PIN W17 [get\_ports {IN[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {IN[2]}]  set\_property PACKAGE\_PIN W15 [get\_ports {IN[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {IN[3]}]  #set\_property PACKAGE\_PIN V15 [get\_ports {IN[0]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[5]}]  #set\_property PACKAGE\_PIN W14 [get\_ports {sw[6]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[6]}]  #set\_property PACKAGE\_PIN W13 [get\_ports {sw[7]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[7]}]  #set\_property PACKAGE\_PIN V2 [get\_ports {sw[8]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[8]}]  #set\_property PACKAGE\_PIN T3 [get\_ports {sw[9]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[9]}]  #set\_property PACKAGE\_PIN T2 [get\_ports {sw[10]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[10]}]  #set\_property PACKAGE\_PIN R3 [get\_ports {sw[11]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[11]}]  #set\_property PACKAGE\_PIN W2 [get\_ports {sw[12]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[12]}]  #set\_property PACKAGE\_PIN U1 [get\_ports {sw[13]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[13]}]  #set\_property PACKAGE\_PIN T1 [get\_ports {sw[14]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[14]}]  #set\_property PACKAGE\_PIN R2 [get\_ports {sw[15]}]  #set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[15]}]  ##7 segment display  set\_property PACKAGE\_PIN W7 [get\_ports {LEDOUT[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[0]}]  set\_property PACKAGE\_PIN W6 [get\_ports {LEDOUT[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[1]}]  set\_property PACKAGE\_PIN U8 [get\_ports {LEDOUT[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[2]}]  set\_property PACKAGE\_PIN V8 [get\_ports {LEDOUT[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[3]}]  set\_property PACKAGE\_PIN U5 [get\_ports {LEDOUT[4]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[4]}]  set\_property PACKAGE\_PIN V5 [get\_ports {LEDOUT[5]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[5]}]  set\_property PACKAGE\_PIN U7 [get\_ports {LEDOUT[6]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[6]}]  #set\_property PACKAGE\_PIN V7 [get\_ports dp]  #set\_property IOSTANDARD LVCMOS33 [get\_ports dp]  set\_property PACKAGE\_PIN U2 [get\_ports {LEDSEL[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[0]}]  set\_property PACKAGE\_PIN U4 [get\_ports {LEDSEL[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[1]}]  set\_property PACKAGE\_PIN V4 [get\_ports {LEDSEL[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[2]}]  set\_property PACKAGE\_PIN W4 [get\_ports {LEDSEL[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[3]}]  # LEDs  set\_property PACKAGE\_PIN U16 [get\_ports {done}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {done}]  set\_property PACKAGE\_PIN E19 [get\_ports {error}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {error}]    ##Buttons  set\_property PACKAGE\_PIN U18 [get\_ports rst]  set\_property IOSTANDARD LVCMOS33 [get\_ports rst]  set\_property -dict {PACKAGE\_PIN W19 IOSTANDARD LVCMOS33} [get\_ports {GO}]; # Left Button |

|  |
| --- |
| **factorial\_tb.v** |
| module factorial\_tb();  reg [31:0] tb\_in;  reg tb\_clk, tb\_go, tb\_rst;  wire [31:0] tb\_out;  wire tb\_done, tb\_error;  reg [31:0] exp\_out;  reg exp\_done;  integer i, count;  CU\_DP DUT(.IN(tb\_in), .GO(tb\_go), .RST(tb\_rst), .clk(tb\_clk),  .OUT(tb\_out), .done(tb\_done), .error(tb\_error));    initial begin  tb\_rst = 0;  i = 0;  while (i < 16) begin  exp\_out = 1;  tb\_in = i;  tb\_go = 1;  tick;    exp\_done = 0;  while (tb\_done == 0) begin  tick;  end  exp\_done = 1;  count = i;  while (count > 1) begin  exp\_out = exp\_out \* count;  count = count - 1;  end  tick;  i = i + 1;  end  $finish;  end    task tick;  begin  tb\_clk = 0; #5;  tb\_clk = 1; #5;  end  endtask  endmodule |

|  |
| --- |
| **bin2hex32.v** |
| module bin2hex32(  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 & 4'hFF;  assign dig1 = value >> 4 & 4'hFF;  assign dig2 = value >> 8 & 4'hFF;  assign dig3 = value >> 12 & 4'hFF;  assign dig4 = value >> 16 & 4'hFF;  assign dig5 = value >> 20 & 4'hFF;  assign dig6 = value >> 24 & 4'hFF;  assign dig7 = value >> 28 & 4'hFF;  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 |

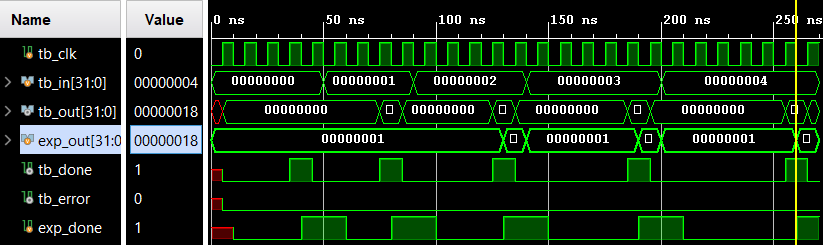
|  |
| --- |
| **clk\_gen.v** |
| module clk\_gen(clk50MHz, rst, clksec4, clk\_5KHz);  input clk50MHz, rst;  output clksec4, clk\_5KHz;  reg clksec4, clk\_5KHz;  integer count, count1;  always@(posedge clk50MHz)  begin  if(rst)  begin  count = 0;  count1 = 0;  clksec4 = 0;  clk\_5KHz =0;  end  else  begin  if(count == 100000000)  begin  clksec4 = ~clksec4;  count = 0;  end  if(count1 == 20000)  begin  clk\_5KHz = ~clk\_5KHz;  count1 = 0;  end  count = count + 1;  count1 = count1 + 1;  end  end  endmodule |

|  |
| --- |
| **HILO\_MUX.v** |
| module HILO\_MUX(  input wire [3:0] HI\_dig3,  input wire [3:0] HI\_dig2,  input wire [3:0] HI\_dig1,  input wire [3:0] HI\_dig0,  input wire [3:0] LO\_dig3,  input wire [3:0] LO\_dig2,  input wire [3:0] LO\_dig1,  input wire [3:0] LO\_dig0,  input wire HILO\_sel,  output wire [3:0] HW\_dig3,  output wire [3:0] HW\_dig2,  output wire [3:0] HW\_dig1,  output wire [3:0] HW\_dig0  );  assign HW\_dig3 = HILO\_sel ? HI\_dig3 : LO\_dig3;  assign HW\_dig2 = HILO\_sel ? HI\_dig2 : LO\_dig2;  assign HW\_dig1 = HILO\_sel ? HI\_dig1 : LO\_dig1;  assign HW\_dig0 = HILO\_sel ? HI\_dig0 : LO\_dig0;  endmodule |

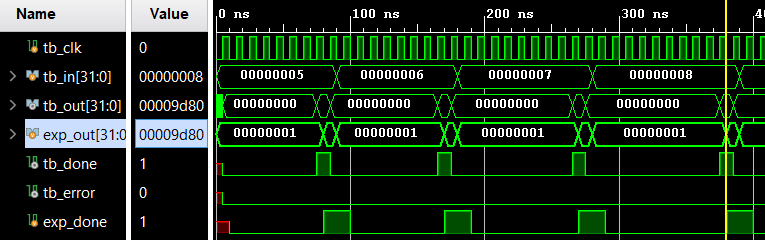
|  |
| --- |
| **LED\_MUX.v** |
| module LED\_MUX (clk, rst, LED0, LED1, LED2, LED3, LEDOUT, LEDSEL);  input clk, rst;  input [6:0] LED0, LED1, LED2, LED3;  output[3:0] LEDSEL;  output[6:0] LEDOUT;  reg [3:0] LEDSEL;  reg [6:0] LEDOUT;  reg [1:0] index;  always @(posedge clk)  begin  if(rst)  index = 0;  else  index = index + 1;  end  always @(index or LED0 or LED1 or LED2 or LED3)  begin  case(index)  0: begin  LEDSEL = 4'b1110;  LEDOUT = LED0;  end  1: begin  LEDSEL = 4'b1101;  LEDOUT = LED1;  end  2: begin  LEDSEL = 4'b1011;  LEDOUT = LED2;  end  3: begin  LEDSEL = 4'b0111;  LEDOUT = LED3;  end  default: begin  LEDSEL = 0; LEDOUT = 0;  end  endcase  end  endmodule |

1. **Waveforms**

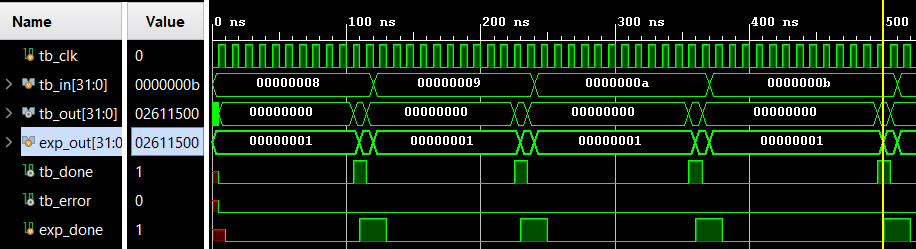
The waveform simulations below demonstrate that the results are being outputted as expected. In *Figure 8*, the input to the system (tb\_in) was a value of 4, which our expected value should be 4 x 3 x 2 x 1, which equals 24 (0x18). The waveform shows that at DONE, our experimental output (tb\_out) is 0x18, which is the result we were expecting.

****

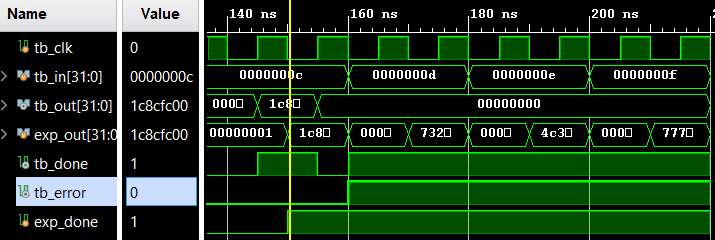
*Figure 8: Displaying the Waveform Simulation signals from inputs 0 to 4. Here, we can see that the expected output for input 4 is 0x18 and our output (tb\_out) is also 0x18. There are no errors and the DONE flag is raised*

**

*Figure 9: Displaying the Waveform Simulation from inputs 5 to 8.*

**

*Figure 10: Displaying the Waveform Simulation for inputs 9 to 11*

**

*Figure 11: Displaying the Waveform Simulation for inputs 12 to 15. Here, we can see that once 13, 14, and 15 are used as inputs, the ERROR flag (tb\_error) is set to HIGH, making the output equal 0 due to never leaving State 0 of the State Machine.*