**San Jose State University**
**Department of Computer Engineering**

## CMPE 140 Lab Report

# Lab 1 Report

**Title** System-Level Design Review

**Semester** Spring **Date** 02/04/2020

**by**


**Name** Thien Hoang **SID** 012555673

**Name** Phat Le **SID** 012067666


## Lab Checkup Record

| Week | Performed By (signature) | Checked By (signature) | Tasks Successfully Completed* | Tasks Partially Completed* | Tasks Failed or Not Performed* |
|---|---|---|---|---|---|
| 1 | x TH x | | 100% | | |
| 2 | TH | | 100% | | |

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

**Authors**

| | |
|---|---|
|  | Thien Hoang<br><br>I was born and raised in Ho Chi Minh City, Viet Name. I came to the United States when I was 16 years old. Before transferring to SJSU, I lived in Houston Texas with my family. I am currently a 5th-year Computer Engineering major at San Jose State University. I have an interest in building the circuit and aspire to be a circuit designer in the future. I enjoy riding a motorcycle on the weekend. It helps me release my stress |
|  | Phat Le<br><br>I am an international student from Vietnam, also a transfer student from Seattle ,and currently a senior Computer Engineering student at San Jose State University. My primary goal is to invent a device that will have huge improvements in humanity. By understanding the way computer software and hardware works, I believe my dream can become true one day. I also enjoy playing RPG console video games and singing alone with my guitar. Sounds very introvert right?! |

**Introduction**

The purpose of this lab is to review system-level design from CMPE 125 course by designing, functionally verifying, and FPGA prototyping the factorial computation. The system would begin the computation after receiving *Go* input and produce the *Done* signal and the result when the process is completed. Furthermore, the *Error* signal is also produced when the input number is greater than 12.

**Design Methodology**

In this lab, we will have two parts. In the first part, we will design and test the control unit and datapath for computing the factorial of n. In the second part of this lab, build the fully integrated factorial generator that contains the control unit and datapath. By looking at the factorial generator module, the system will start executing when the "Go" signal is received and output "Done" when the executing is completed. We also have an additional case that input entered is greater than 12 an "Err" signal will be set indicating the error.

*Table 1: List of all the module files and their function*

| Module | Function |
|---|---|
| clk_gen.v | This module is used to generate the clock signal. |
| button_debouncer.v | This module is used to generate the clock signal manually. |
| Factorial.v | This is the top-level module that connects the datapath module and control unit module together. |
| bin2hex32.v | This module is converting binary to a 32-bit hexadecimal value. |
| HILO_MUX.v | This module is selecting High a low mode. If it is =1 then selected the high mode if it is = 0 then selected the low mode |
| hex_to_7seg.v | This module is converting a single 4 digit hex value to 7 segment display LED. |
| led_mux.v | This module is selected for the appropriate on board LEDs to activate. |

| | |
|---|---|
| DataPath.v | This module is datapath for the factorial generator. |
| CNT.v | This module is a down counter with parallel load control and an enable signal. |
| CMP.v | This module is a comparator with a greater than output. |
| REG.v | This module is a data register with a load control signal. |
| MUL.v | This module is a combinational multiplier for unsigned integers. |
| MUX.v | This module is a 2-to-1 multiplexer. |
| ControlUnit.v | This module provides control signal inputs to the datapath based on the states transition and the feedback signal. |
| Factorial_contraints.xdc | This is the constraint file that decides all the LEDs and switches on the FPGA board. |
| Factorial_tb.v | This is the testbench module that checks multiple cases of the inputs and outputs on the Factorial module. |
| Factorial_FPGA.v | This module is combining all the functions and executing to FPGA. |

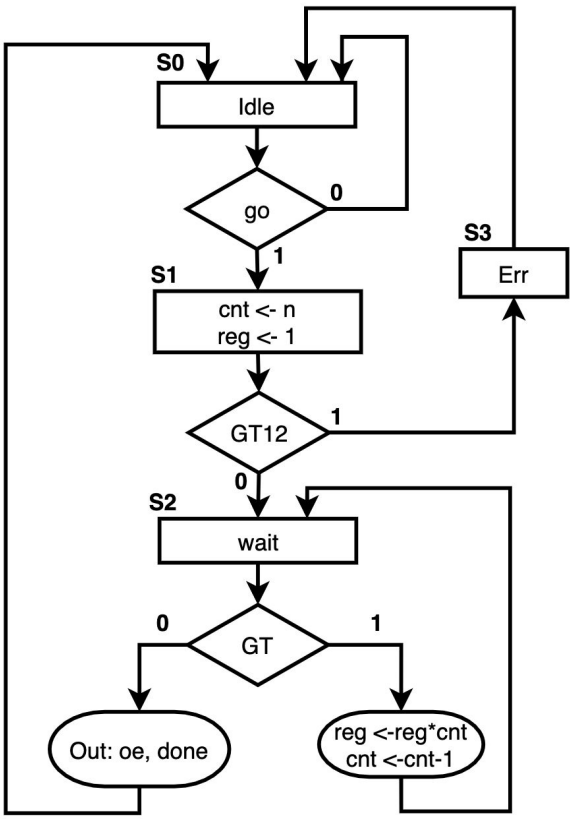


*Figure 1: ASM chart describing the Control Unit system's cycle by cycle*

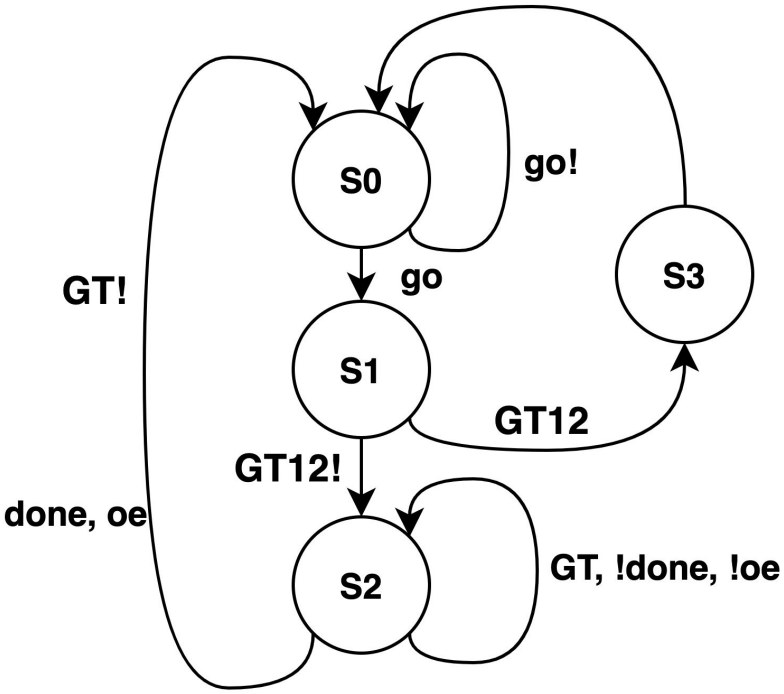


*Figure 2: State transition diagram  extracted from the ASM chart above*

*Table 2: Output Logic of the FSM extracted from the ASM chart above*

| Current State | Next State | Input | | | Output | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | go | GT12 | GT | sel | ld_cnt | en | ld_reg | oe | done | Err |
| S0 | S0 | 0 | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | S1 | 1 | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S1 | S2 | x | 0 | x | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| | S3 | x | 1 | x | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| S2 | S0 | x | x | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| | S2 | x | x | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| S3 | S0 | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

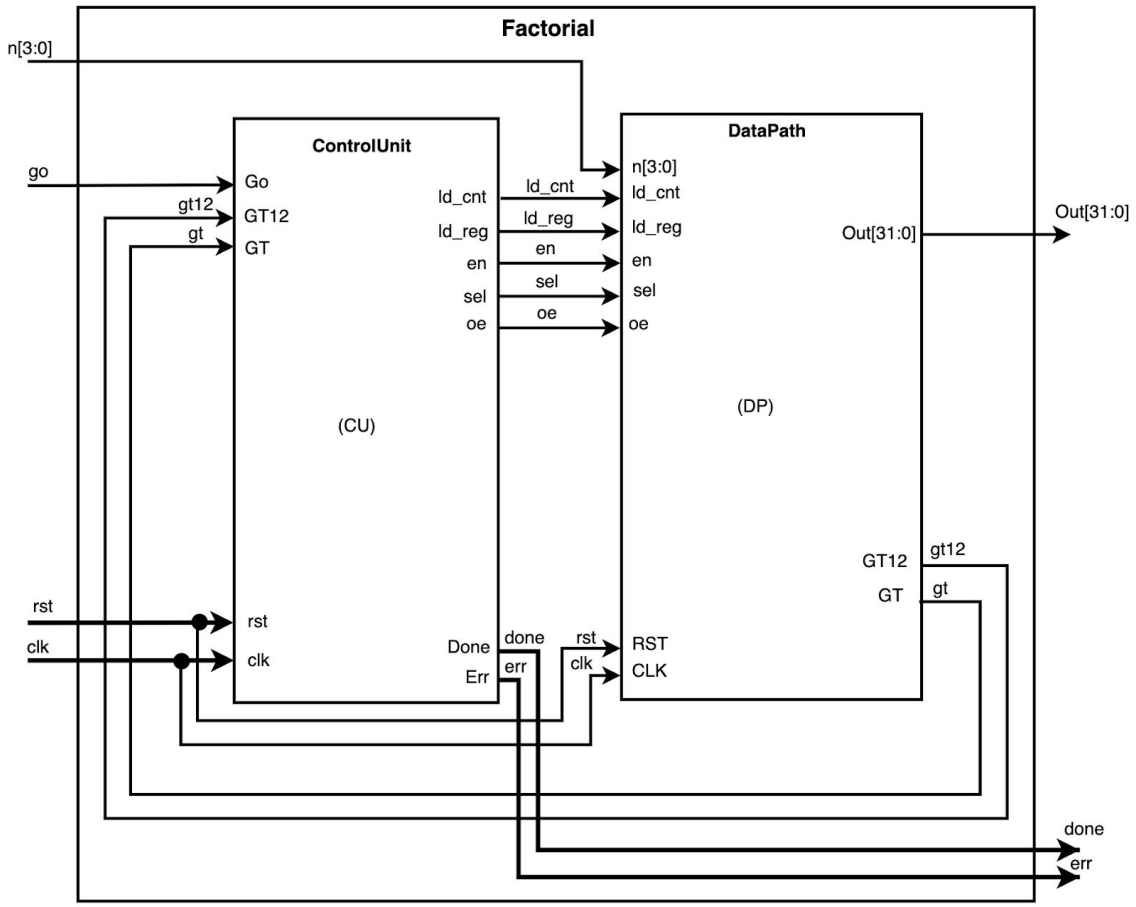


*Figure 3: Block diagram for top-level Factorial module*

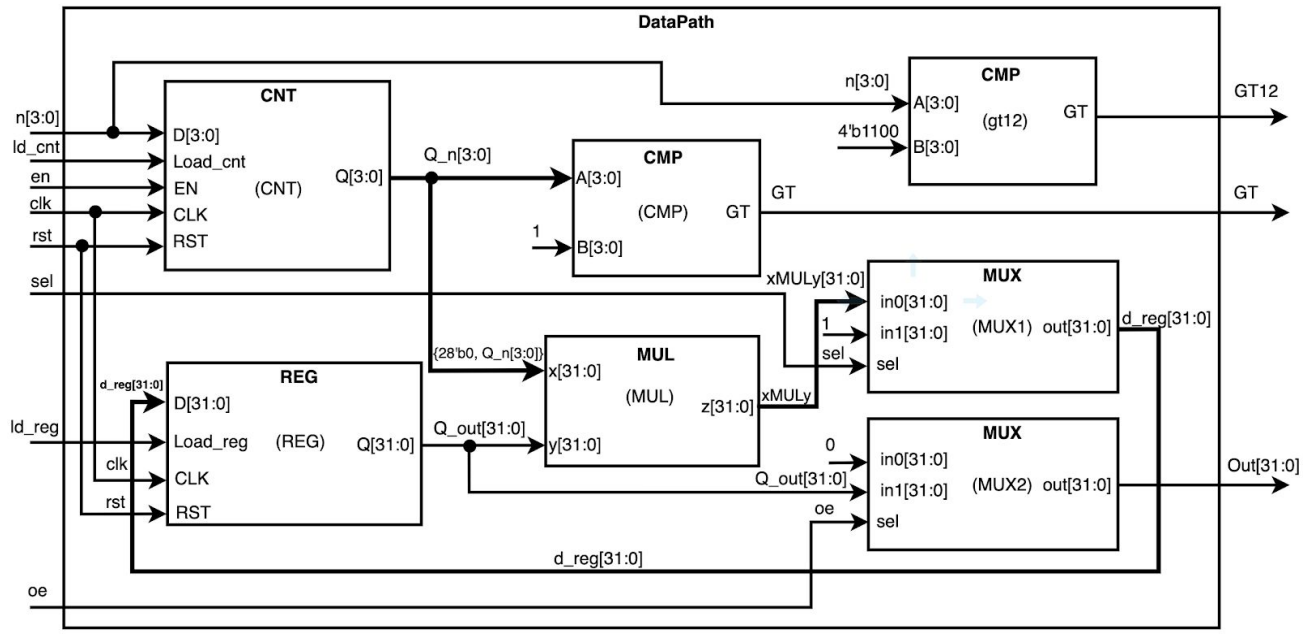


*Figure 4:Block diagram for Factorial datapath*

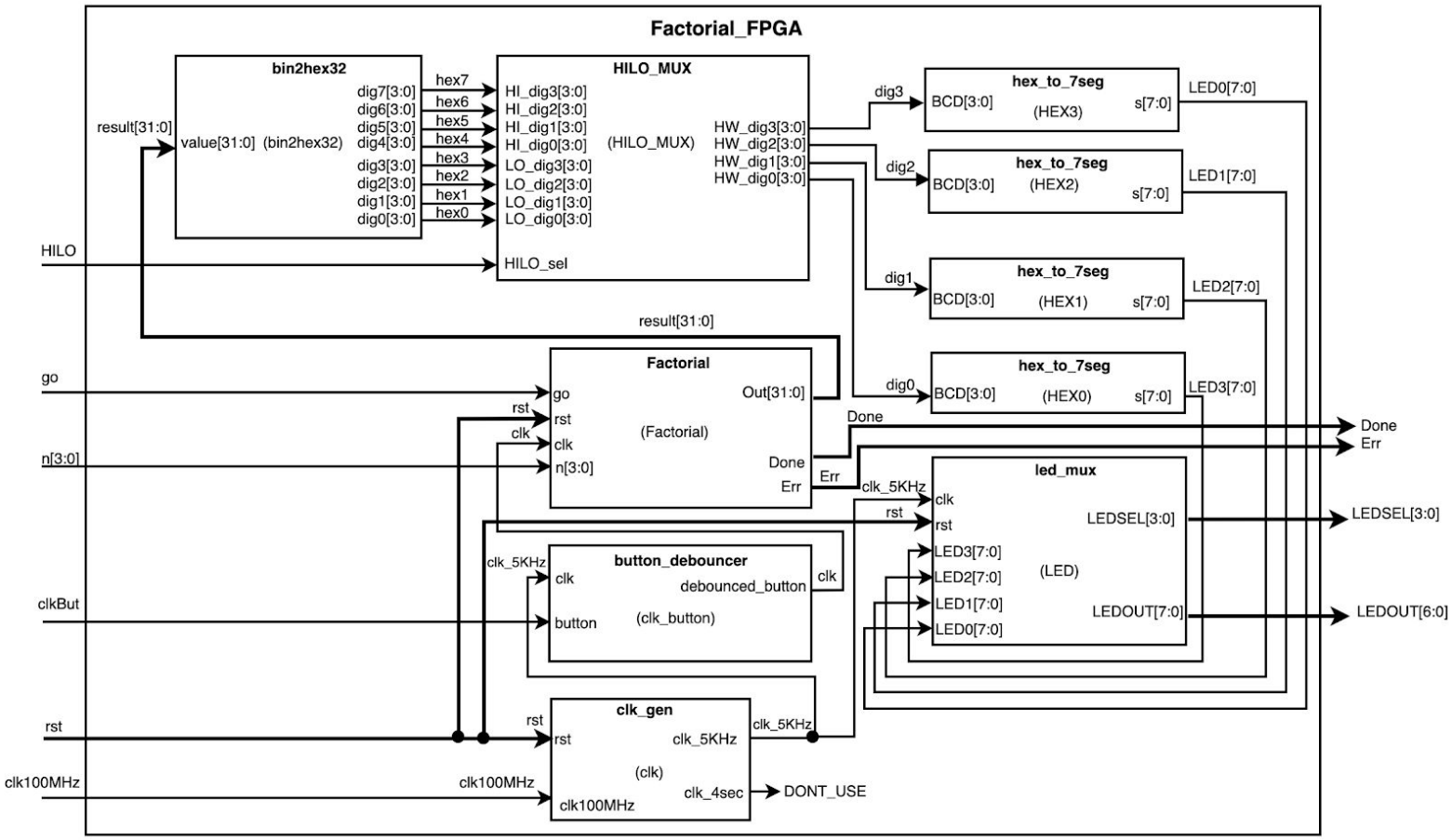


*Figure 5: Block diagram for top-level FPGA module*

## Simulation Result

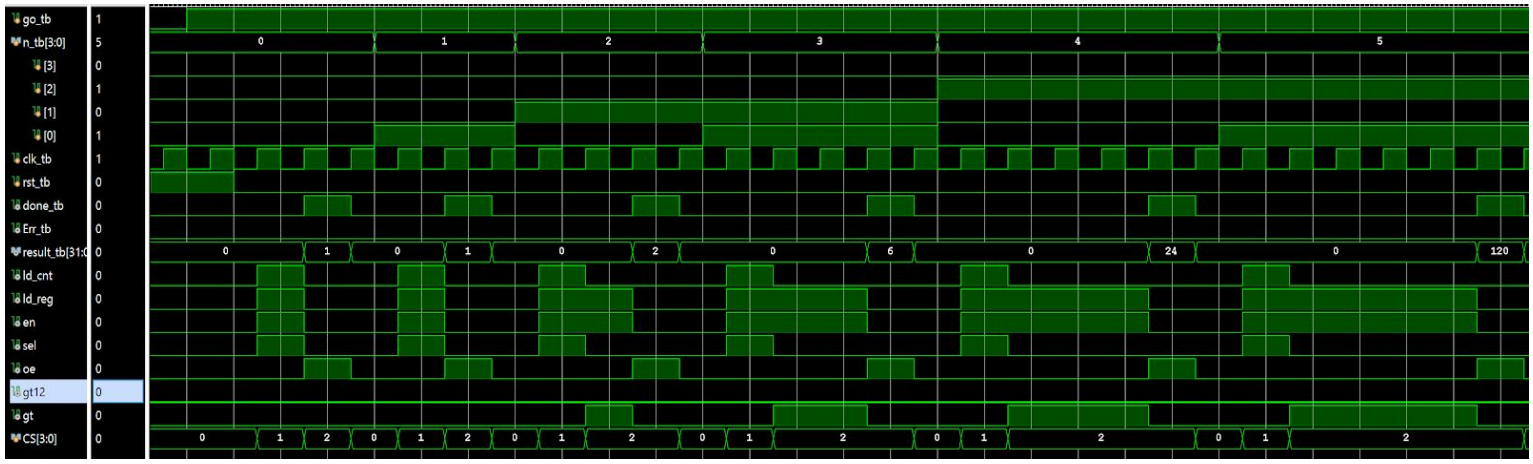


*Figure 6: Simulation waveforms from n = 0 to n = 5*

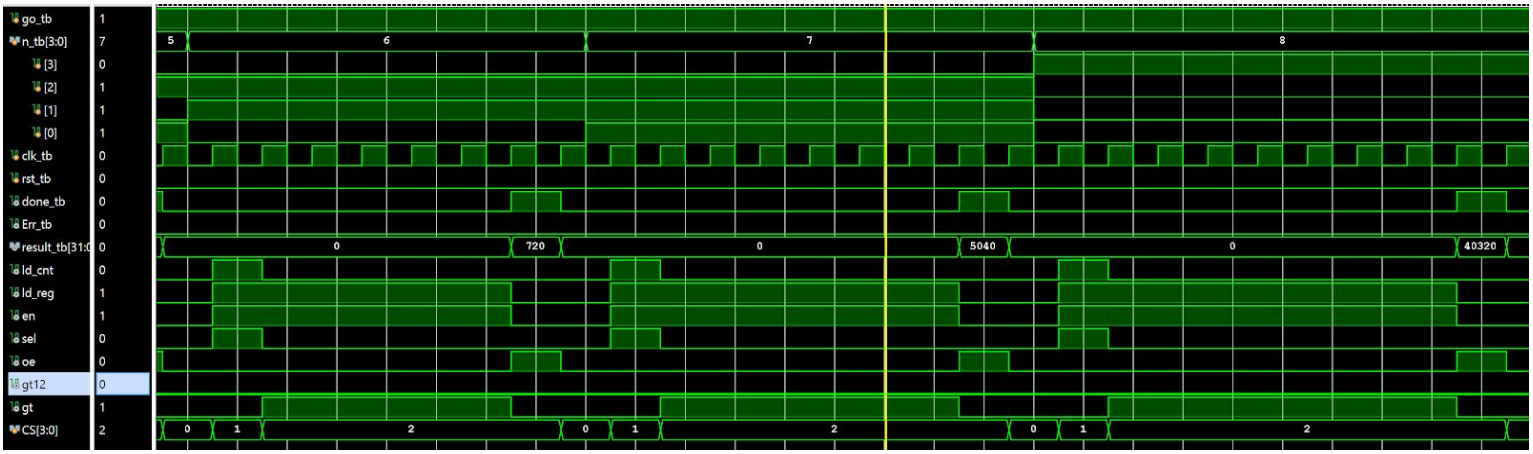


*Figure 7:Simulation waveforms from n = 6 to n = 8*

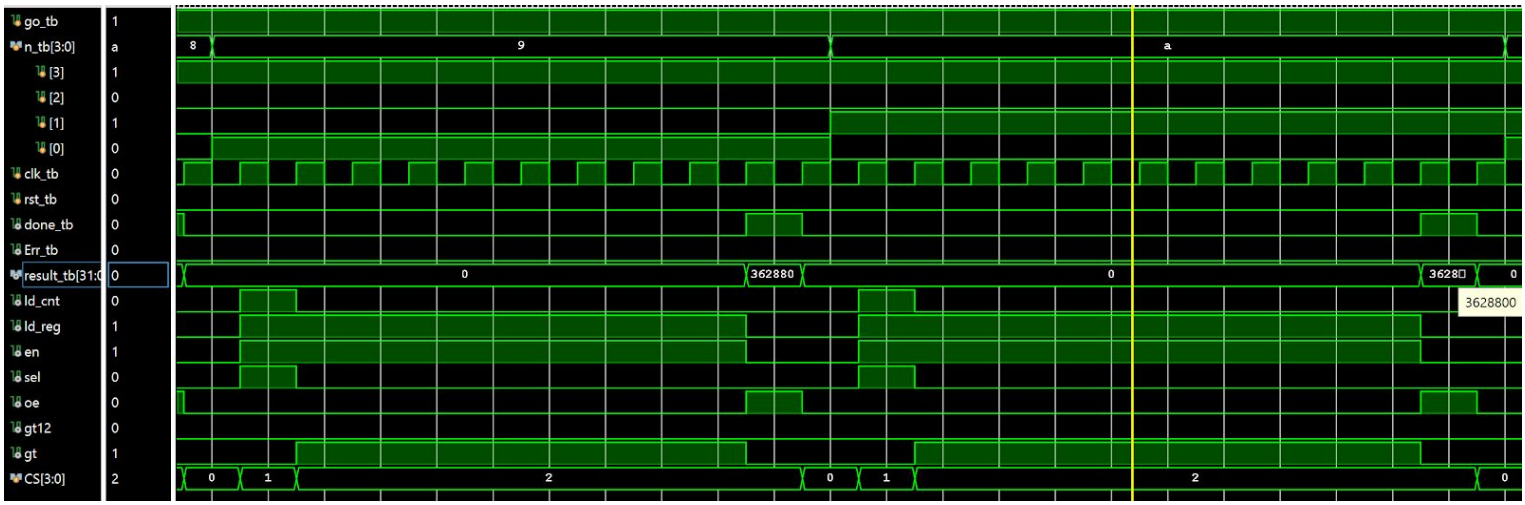


*Figure 8:Simulation waveforms of n = 9 and n = 10*

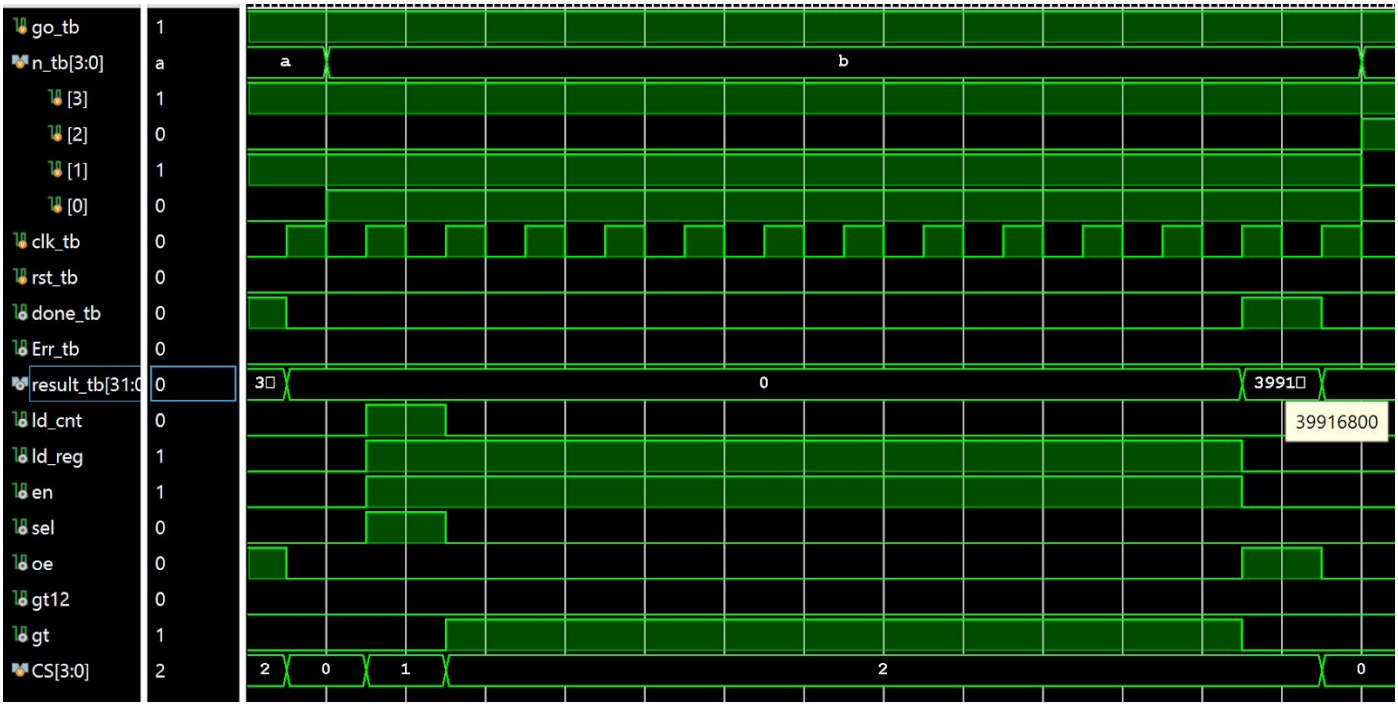


*Figure 9: Simulation waveforms of n = 11*

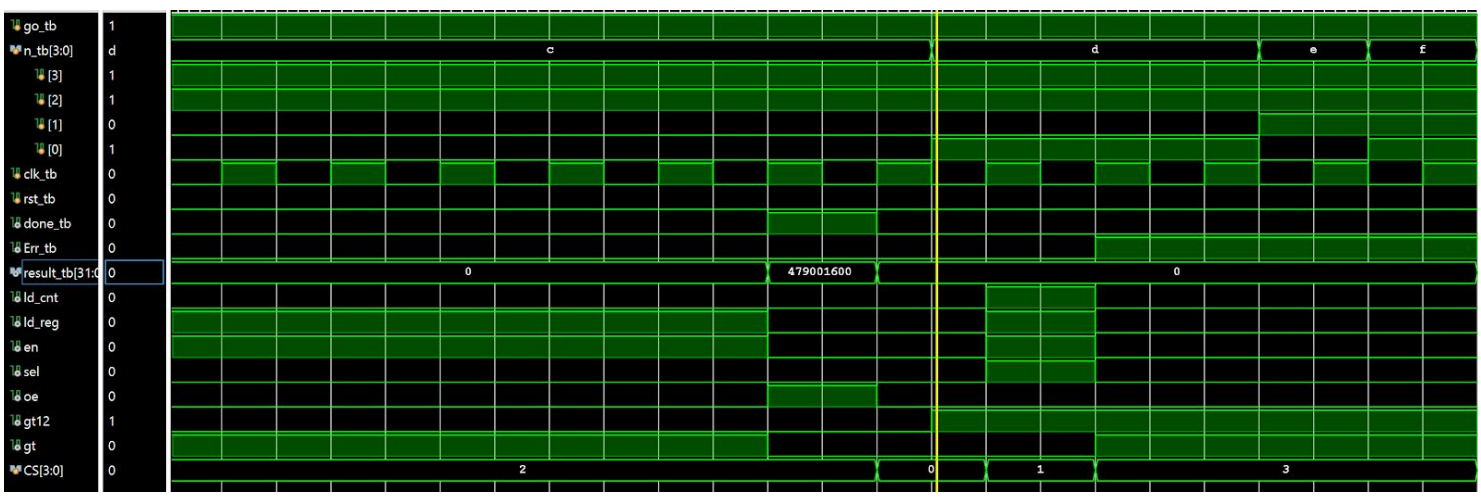


*Figure 10:Simulation waveforms from n = 12 to n = 15*

The testbench goes through all possible cases from n = 0 (0000) to n = 15(1111). As shown in *Figure 8,* the first one is when the input value (n_tb) is 0 which gives the result displayed in unsigned integer format (result_tb) as expected to be 1. According to *Figure 10*, when n_tb = 12 (c in Hex), the result is 479001600 which matches the result of !12. Furthermore, it can easily observe that all the result is only displayed at the end of the computation as the done signal (done_tb) is 1. In addition, the simulation also checks cases when n > 12. According to the waveform above, when n_tb = 13(d in Hex), the system would not output the final result but

display error signal(Err_tb = 1). Since all calculated results from all cases matched their actual values as shown in *Figure 8, 9, and 10*, functional verification was successful.

**FPGA Validation**

For the FPGA validation process, the rightmost 4 switches are 4-bit input n, the leftmost switch is HILO_select. The center button is the clock, the left button is reset input, and the right button is go input. There was also a done and an error LED signals on the left. As shown in *Figure 13, 14, 15* the Done LED lid as soon as the result appeared in seven-segment LEDs. According to *Figure 12,* the Err LED *also* lid when input is greater than 12 which is as expected. Since all the output results from the seven-segments matched actual values, the FPGA validation process was successful.

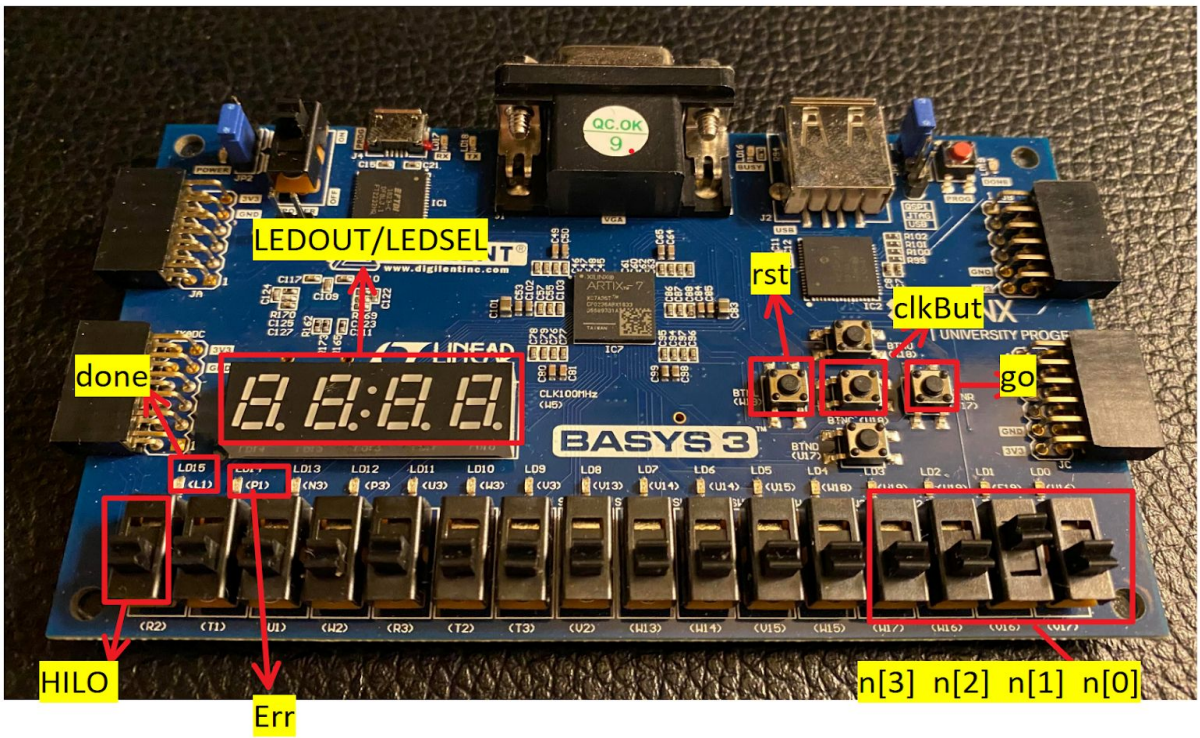


*Figure 11: Digilent Basys 3 FPGA Board environment setup*

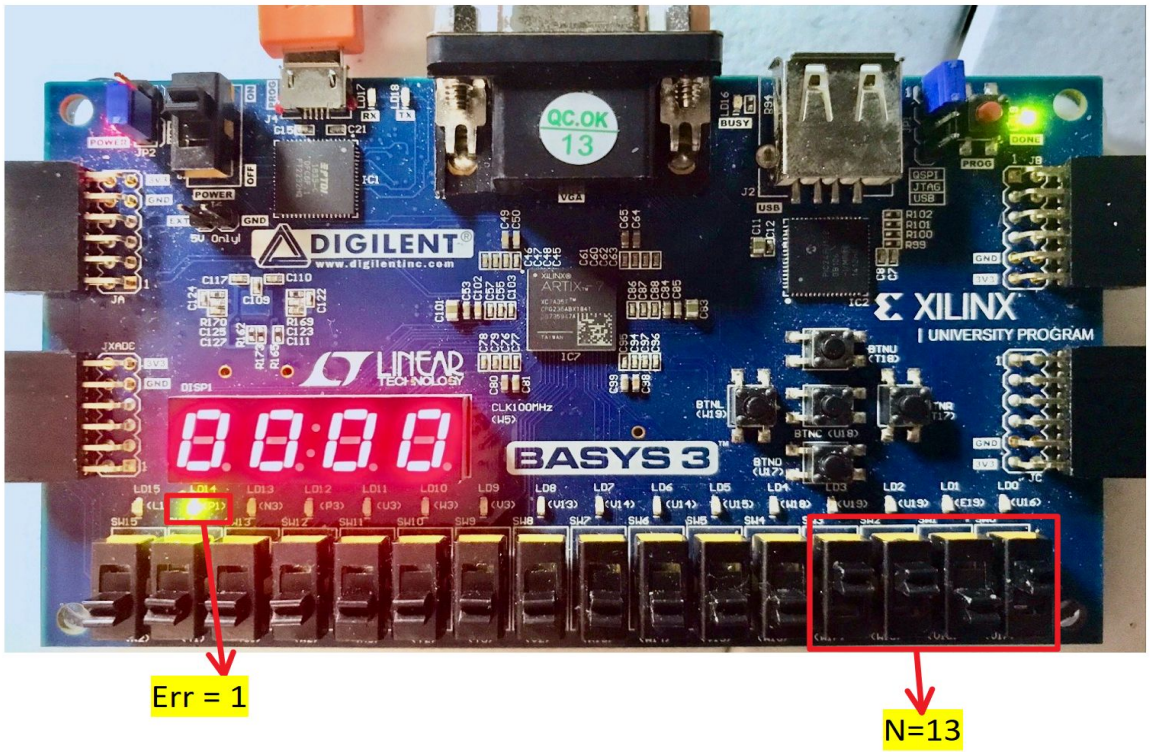


*Figure 12: The Factorial Generator showing the result of 13!*
*n =13, Done=0 , Err =1*

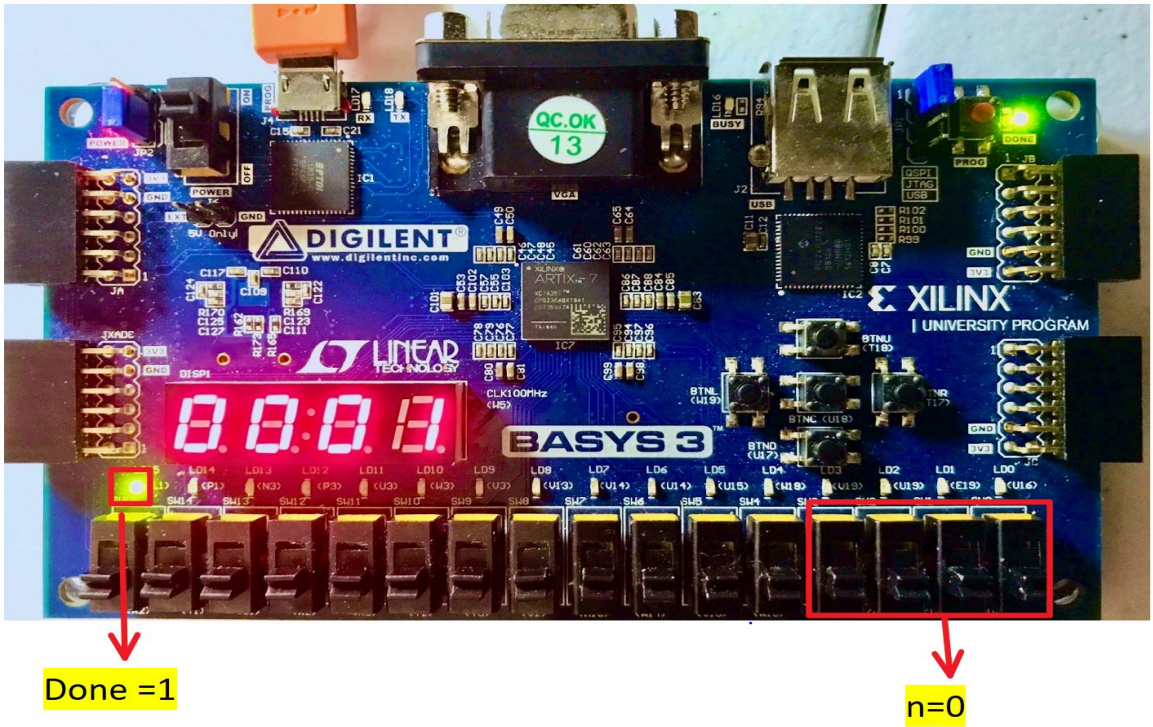


*Figure 13: The Factorial Generator showing the result of 0!*
*N =0, Done = 1 , Err = 0, result =1*

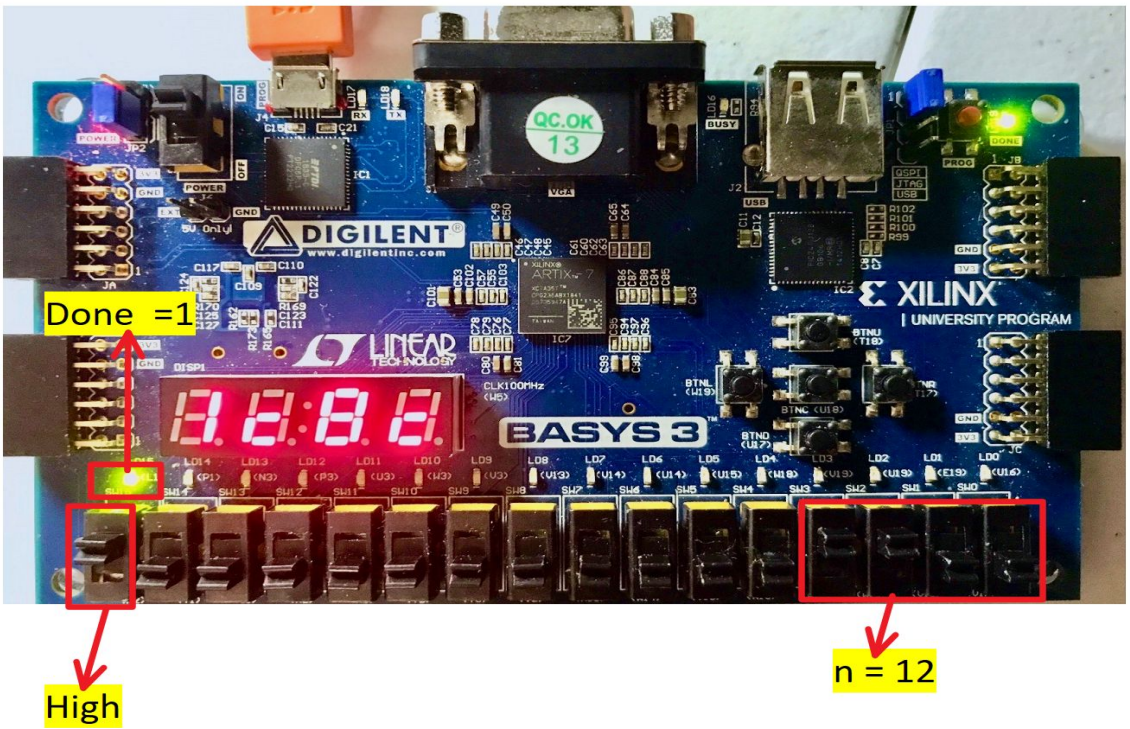


*Figure 14: The Factorial Generator showing the result of 12! With the high mode set N =12, Done = 1 , Err = 0, result =1c8cFc00*

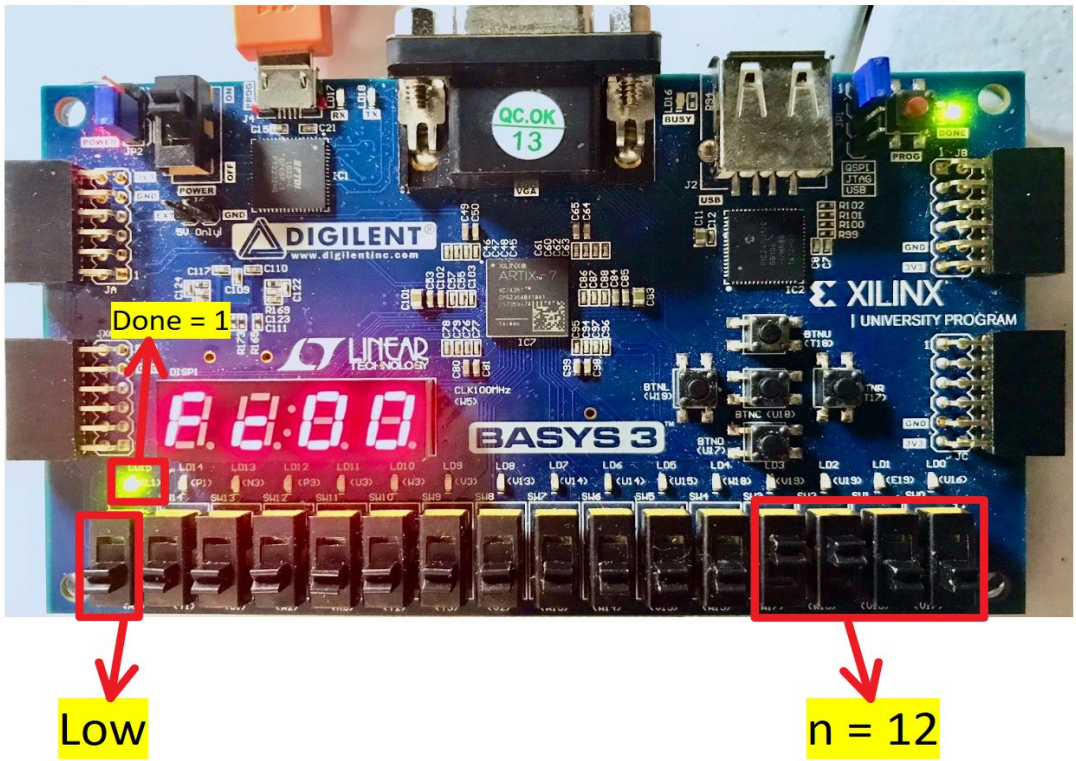


*Figure 15: The Factorial Generator showing the result of 12! With the low mode set N =12, Done = 1 , Err = 0, result =1c8cFc00*

**Conclusion**

In sum, this lab helped us review the Verilog code learned from CMPE 125. Furthermore, by doing this lab, we understand the concept of converting 32 bits values to 8 hexadecimal values that can be displayed in only 4 seven-segment LEDs. At first, we were not able to get all the cases showing in the waveform because we were missing one clock at the end of each computation to transition from the last state to idle state. We also did not know the syntax on how to concatenate 28b'0 with the 4 bits input. With the help of our TA, we were able to get the correct waveforms. Overall, this lab is successfully finished and it was a good review of fundamental concepts for us.

## Appendix

### a. Source Code

**DataPath.v**

```
module DataPath(input [3:0] n,
                input ld_cnt,
                input en,
                input clk,
                input rst,
                input sel,
                input ld_reg,
                input oe,
                output [31:0] Out,
                output GT,
                output GT12
    );
    wire [3:0] Q_n;
    wire [31:0] d_reg, Q_out, xMULy;

    CMP#4 gt12(.A(n),
               .B(4'b1100),
               .GT(GT12));

    CNT#4 CNT( .D(n),
               .Load_cnt(ld_cnt),
               .EN(en),
               .CLK(clk),
               .RST(rst),
               .Q(Q_n));

    CMP#4 CMP( .A(Q_n),
               .B(1),
               .GT(GT));

    REG#32 REG( .D(d_reg),
                .Load_reg(ld_reg),
                .CLK(clk),
                .RST(rst),
                .Q(Q_out));

    MUL#32 MUL(.x({28'b0, Q_n[3:0]}),
               .y(Q_out),
               .z(xMULy));

    MUX#32 MUX1(.in0(xMULy),
                .in1(1),
                .sel(sel),
                .out(d_reg));

    MUX#32 MUX2(.in0(0),
                .in1(Q_out),
                .sel(oe),
                .out(Out));
```

```
endmodule
```

## CMP.v

```
module CMP #(parameter WIDTH = 4)(
            input wire [WIDTH-1:0] A,
            input wire [WIDTH-1:0] B,
            output wire GT
    );
    assign GT =(A>B) ? 1:0;

endmodule
```

## CNT.v

```
module CNT #(parameter Data_width = 4)
                (input Load_cnt, EN, CLK, RST,
                input [Data_width-1:0] D,
                output reg [Data_width-1:0]Q
    );

    always @(posedge CLK, posedge RST)
    begin
        if(RST) Q <= 0;
        else if(EN)
            begin
                if (Load_cnt) Q <= D;
                else Q <= Q - 1; //countdown
             end
        else Q <= Q;
   end
endmodule
```

## REG.v

```
module REG#(parameter WIDTH = 32)
    (input  CLK, RST,
     input  Load_reg,
     input  [WIDTH-1:0] D,
     output  reg[WIDTH-1:0] Q
    );
    always@(posedge CLK, posedge RST)
        if (RST) Q <= 0;
        else if(Load_reg) Q <= D;
        else Q <= Q;
```

```
endmodule
```

## MUL.v

```
module MUL #(parameter WIDTH = 32)(
                          input  wire [WIDTH-1:0] x,
                          input  wire [WIDTH-1:0] y,
                          output wire [WIDTH-1:0] z
    );
    assign z = x*y;
endmodule
```

## MUX.v

```
module MUX #(parameter WIDTH = 32) (
        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
```

## ControlUnit.v

```
  module ControlUnit(input Go,
                input clk, rst,
                input GT12,
                input GT,
                output reg sel,
                output reg ld_cnt,
                output reg ld_reg,
                output reg en,
                output reg oe,
                output reg Done,
                output reg Err
);

parameter S0 = 4'b0000,
          S1 = 4'b0001,
          S2 = 4'b0010,
          S3 = 4'b0011;

reg [3:0] NS, CS;
```

```
always@(CS, Go, GT12, GT)
begin
    case(CS)
        S0:
        begin
            if(!Go) NS = S0;
            else NS = S1;
        end

        S1:
        begin
            if(GT12) NS = S3;
            else NS = S2;
        end

        S2:
        begin
            if(GT) NS = S2;
            else NS = S0;
        end
    endcase
end

always@(posedge clk, posedge rst)
begin
    if(rst) CS = S0;
    else CS <= NS;
end

always@(CS, GT)
begin
    case(CS)
    S0://idle
    begin
        ld_cnt = 0;
        ld_reg = 0;
        sel = 0;
        en = 0;
        oe = 0;
        Done = 0;
        Err = 0;
    end

    S1://load state
    begin
        ld_cnt = 1; //load value
        ld_reg = 1; // let reg = 1;
        sel = 1;
        en = 1; // enable the count
        oe = 0;
        Done = 0;
        Err = 0;
    end

    S2:
```

```
        begin
            ld_cnt = 0;
            Err = 0;
            sel = 0; //select reg*cnt
            if(GT)
            begin
                ld_reg = 1;//load reg
                en = 1;// cnt-1
                oe = 0;
                Done = 0;
            end
            else begin
                ld_reg = 0;//stop loading
                en = 0;// stop count down
                oe = 1;
                Done = 1;
            end
        end

        S3:
        begin
            ld_cnt = 0; //dont care
            ld_reg = 0; // dont care
            sel = 0;//dont care
            en = 0; // dont care
            oe = 0;//dont care
            Done = 0;//dont care
            Err = 1;
        end
        endcase
    end
endmodule
```

## Factorial.v

```
  module Factorial(input go,
                   input [3:0] n,
                   input clk, rst,
                   output done,
                   output err,
                   output [31:0] Out
      );
      wire ld_cnt, ld_reg, en, sel, oe, gt12, gt;

      DataPath DP (.n(n),
                   .ld_cnt(ld_cnt),
                   .en(en),
                   .clk(clk),
                   .rst(rst),
                   .sel(sel),
                   .ld_reg(ld_reg),
                   .oe(oe),
                   .Out(Out),
```

```
                    .GT(gt),
                    .GT12(gt12));

    ControlUnit CU(.Go(go),
                   .clk(clk),
                   .rst(rst),
                   .GT12(gt12),
                   .GT(gt),
                   .sel(sel),
                   .ld_cnt(ld_cnt),
                   .ld_reg(ld_reg),
                   .en(en),
                   .oe(oe),
                   .Done(done),
                   .Err(err));
endmodule
```

## calculator_tb.v

```
module Factorial_tb;
    reg go_tb;
    reg [3:0] n_tb;
    reg clk_tb, rst_tb;
    wire done_tb, Err_tb;
    wire [31:0] result_tb;



    Factorial DUT(.go(go_tb),
                  .n(n_tb),
                  .clk(clk_tb),
                  .rst(rst_tb),
                  .done(done_tb),
                  .err(Err_tb),
                  .Out(result_tb));

    task ticktock;
    begin
        #5 clk_tb = ~clk_tb;
        #5 clk_tb = ~clk_tb;
    end
    endtask

    integer i;
    initial begin
        go_tb = 0;
        n_tb = 0;
        clk_tb = 0;
        rst_tb = 1;

        ticktock();
```

```
        ticktock();
        //begin the operation
        go_tb = 1;
        ticktock();
        //i=0;

        rst_tb = 0;
        ticktock();
        ticktock();

        for( i=0; i<16;i = i+1)
        begin
            n_tb = i;

            while(!done_tb && !Err_tb)
            begin
                ticktock();
            end
            ticktock();
        end

    end
endmodule
```

## Factorial_FPGA.v

```
module Factorial_FPGA(input wire go, rst, clkBut, clk100mHz,
                      input wire [3:0] n,
                      input HILO,
                      output wire done,
                      output wire [3:0] LEDSEL,
                      output wire [7:0] LEDOUT,
                      output wire Err
    );

    wire DONT_USE;
    wire clk_5KHz;
    wire debounced_clk;
    wire [3:0] dig3, dig2, dig1, dig0,
        hex7,hex6, hex5, hex4, hex3, hex2, hex1, hex0;
    wire [7:0] LED3, LED2, LED1, LED0;
    wire [31:0] result;

     clk_gen     clk(.clk100MHz(clk100mHz),
                .rst(rst),
                .clk_4sec(DONT_USE),
                .clk_5KHz(clk_5KHz));

    button_debouncer    clk_button(.clk(clk_5KHz),
                                   .button(clkBut),
                                   .debounced_button(debounced_clk));

    Factorial Factorial(.go(go),
```

```
                        .n(n),
                        .clk(debounced_clk),
                        .rst(rst),
                        .done(done),
                        .err(Err),
                        .Out(result));

    bin2hex32 bin2hex32(.value(result),
                        .dig0(hex0),
                        .dig1(hex1),
                        .dig2(hex2),
                        .dig3(hex3),
                        .dig4(hex4),
                        .dig5(hex5),
                        .dig6(hex6),
                        .dig7(hex7));

    HILO_MUX HILO_mux(  .HI_dig3(hex7),
                        .HI_dig2(hex6),
                        .HI_dig1(hex5),
                        .HI_dig0(hex4),
                        .LO_dig3(hex3),
                        .LO_dig2(hex2),
                        .LO_dig1(hex1),
                        .LO_dig0(hex0),
                        .HILO_sel(HILO),
                        .HW_dig3(dig3),
                        .HW_dig2(dig2),
                        .HW_dig1(dig1),
                        .HW_dig0(dig0)
                        );

    hex_to_7seg HEX3 (.number(dig3),
                      .s(LED3));

    hex_to_7seg HEX2 (.number(dig2),
                      .s(LED2));

    hex_to_7seg HEX1 (.number(dig1),
                      .s(LED1));

    hex_to_7seg HEX0 (.number(dig0),
                      .s(LED0));

    led_mux         LED(.clk(clk_5KHz),
                        .rst(rst),
                        .LED3(LED3),
                        .LED2(LED2),
                        .LED1(LED1),
                        .LED0(LED0),
                        .LEDSEL(LEDSEL),
                        .LEDOUT(LEDOUT));
endmodule
```

## clk_gen.v

```
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;
            count2 = 0;
            clk_5KHz = 0;
            clk_4sec = 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
```

## 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
```

## 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
```

## hex_to_7seg.v

```
module hex_to_7seg(input  wire [3:0] number,
                   output reg  [7:0] s);

    always @ (number) begin
        case (number)
                4'h0: s = 8'b11000000;
                4'h1: s = 8'b11111001;
                4'h2: s = 8'b10100100;
                4'h3: s = 8'b10110000;
                4'h4: s = 8'b10011001;
```

```
                4'h5: s = 8'b10010010;
                4'h6: s = 8'b10000010;
                4'h7: s = 8'b11111000;
                4'h8: s = 8'b10000000;
                4'h9: s = 8'b10010000;
                4'ha: s = 8'b10100000;
                4'hb: s = 8'b10000011;
                4'hc: s = 8'b10100111;
                4'hd: s = 8'b10100000;
                4'he: s = 8'b10000100;
                4'hf: s = 8'b10001110;
            default: s = 8'b01111111;
        endcase
    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 (
input wire clk,
input wire rst,
input wire [7:0] LED3,
input wire [7:0] LED2,
input wire [7:0] LED1,
input wire [7:0] LED0,
output wire [3:0] LEDSEL,
```

```
output wire [7:0] LEDOUT
);
reg [1:0] index;
reg [11:0] led_ctrl;
assign {LEDSEL, LEDOUT} = led_ctrl;
always @ (posedge clk) index <= (rst) ? 2'b0 : (index + 2'd1);
always @ (index, LED0, LED1, LED2, LED3) begin
case (index)
4'd0: led_ctrl <= {4'b1110, LED0};
4'd1: led_ctrl <= {4'b1101, LED1};
4'd2: led_ctrl <= {4'b1011, LED2};
4'd3: led_ctrl <= {4'b0111, LED3};
default: led_ctrl <= {8'b1111, 8'hFF};
endcase
end
endmodule
```

## Factorial_constraints.xdc

```
# Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33}           [get_ports
{clk100mHz}];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{clk100mHz}];

# input switches
#input1
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {n[0]}];
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {n[1]}];
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports {n[2]}];
set_property -dict {PACKAGE_PIN W17 IOSTANDARD LVCMOS33} [get_ports {n[3]}];

#HILO select
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports {HILO}];

#clock button
set_property   -dict   {PACKAGE_PIN   U18   IOSTANDARD   LVCMOS33}   [get_ports
{clkBut}];
#go button
set_property -dict {PACKAGE_PIN T17 IOSTANDARD LVCMOS33} [get_ports {go}];
#reset button
set_property -dict {PACKAGE_PIN W19 IOSTANDARD LVCMOS33} [get_ports {rst}];

 # output LED
set_property -dict {PACKAGE_PIN L1 IOSTANDARD LVCMOS33} [get_ports {done}];

 # output Err
set_property -dict {PACKAGE_PIN P1 IOSTANDARD LVCMOS33} [get_ports {Err}];

#LED selection
set_property   -dict   {PACKAGE_PIN   U2   IOSTANDARD   LVCMOS33}   [get_ports
{LEDSEL[0]}]; # AN0
```

```
set_property   -dict   {PACKAGE_PIN   U4   IOSTANDARD   LVCMOS33}   [get_ports
{LEDSEL[1]}]; # AN1
set_property   -dict   {PACKAGE_PIN   V4   IOSTANDARD   LVCMOS33}   [get_ports
{LEDSEL[2]}]; # AN2
set_property   -dict   {PACKAGE_PIN   W4   IOSTANDARD   LVCMOS33}   [get_ports
{LEDSEL[3]}]; # AN3

#LED output
set_property   -dict   {PACKAGE_PIN   W7   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[0]}]; # CA
set_property   -dict   {PACKAGE_PIN   W6   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[1]}]; # CB
set_property   -dict   {PACKAGE_PIN   U8   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[2]}]; # CC
set_property   -dict   {PACKAGE_PIN   V8   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[3]}]; # CD
set_property   -dict   {PACKAGE_PIN   U5   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[4]}]; # CE
set_property   -dict   {PACKAGE_PIN   V5   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[5]}]; # CF
set_property   -dict   {PACKAGE_PIN   U7   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[6]}]; # CG
set_property   -dict   {PACKAGE_PIN   V7   IOSTANDARD   LVCMOS33}   [get_ports
{LEDOUT[7]}]; # DP
```