San Jose State University
Department of Computer Engineering

CMPE 140 Lab Report

Lab 1 Report

Title System-Level Design Review

Semester Spring 2019 **Date** _____

by

Name Darren Truong (typed) **SID** 010421463 (typed)

Name Alam Figueroa (typed) **SID** 012100387 (typed)

Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1	DT Alam		100%		
2	Alam DT		100%		

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

Introduction

The purpose of this lab is to become familiarized with system-level design again by developing a factorial accelerator. The lab was split into several tasks to accomplish this. The datapath block diagram, CU-DP block diagram, ASM chart, next state logic bubble diagram, and output table were designed as the tasks for the first week. The second week was to code in Verilog the datapath, control unit, CU-DP module, and all the testbenches to verify that the system functions properly. The system was also validated on the FPGA board.

Design Methodology

The task of developing a factorial module was accomplished by incorporating the modules listed in the following chart to form a datapath and having a control unit manage the overall process of the machine. The system starts execution upon receiving an external input "*Go*" and outputs a "*Done*" signal when the execution is completed. In addition, an "*Error*" signal is set using a combinational comparator when an input greater than 12 is entered, this automatically leads to the last state where the "*Done*" signal is set.

Table 1: Modules and descriptions used for Lab 1

Module	**Description**
FACTORIAL_FPGA	Module that implements the nfactorial module with the FPGA board.
nfactorial	Module that combines the datapath and control unit.
nfactorial_dp	Datapath module for the factorial.
control_unit	Control unit for the factorial.
counter	Countdown register module.
comp	Comparator module used to check if n > 12 or if the countdown has reached 1.
mux	2 input mux that loads either 1 or the multiplication result to the register module.
register	32-bit register used to store the result of the factorial or keeps track of the multiplication during the factorial process.
buffer	Tri-state buffer module that controls the

	datapath output.
clk_gen	Module used as clock for the button_debouncer and led_mux
button_debouncer	Module used to turn a button into a manual clock.
bin2hex32	Takes the 32-bit value from the factorial outputs it as 8 4-bit digits in hex.
HILO_MUX	Assigns which 4 digits will be displayed.
hex_to_7seg	Converts 4-bit values into 8-bit numbers that will indicated which LED segments to turn on.
led_mux	Takes the 8-bit number from hex_to_7seg and uses it to turn on the desired LED segments.

Table 2: Control Unit Output Table

State	Err	GT	sel	Load_cnt	EN	Load_reg	OE	Error	Done
S0	0	-	0	0	0	0	0	0	0
S0	1	-	0	0	0	0	0	1	0
S1	-	-	1	1	1	1	0	0	0
S2	-	0	0	0	0	0	0	0	0
S2	-	1	0	0	0	0	0	0	0
S3	-	-	0	0	0	0	1	0	1
S4	-	-	0	0	1	1	0	0	0

Simulation Methodology

The following waveform depicts 12 of the 13 inputs to factorial with their respective result.

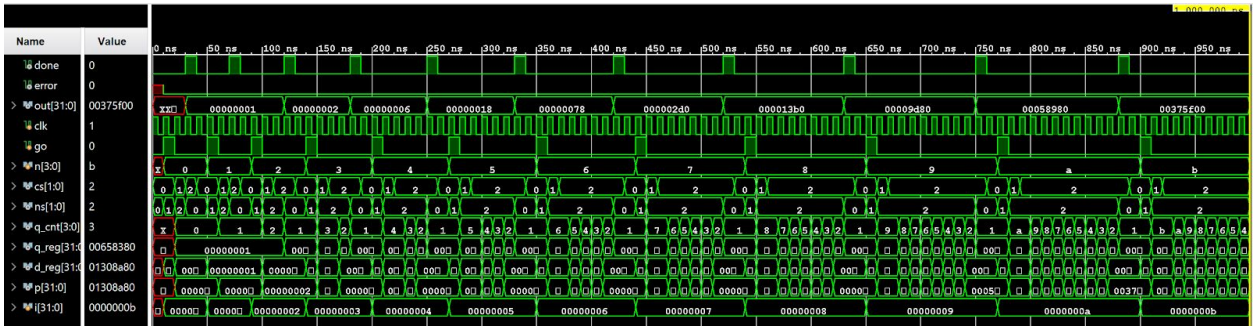

Figure 1: Factorial Module simulation with n = 0 - n = 11(0xb)

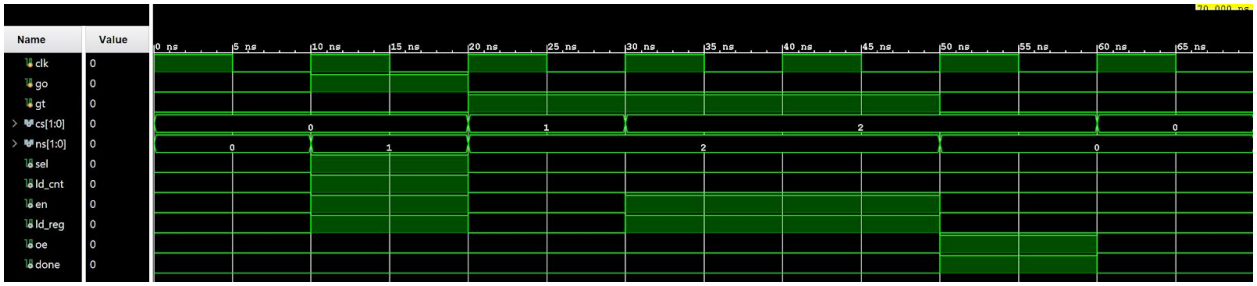

Figure 2: Waveforms depicting the behavior of the control unit for a sample test n = 4

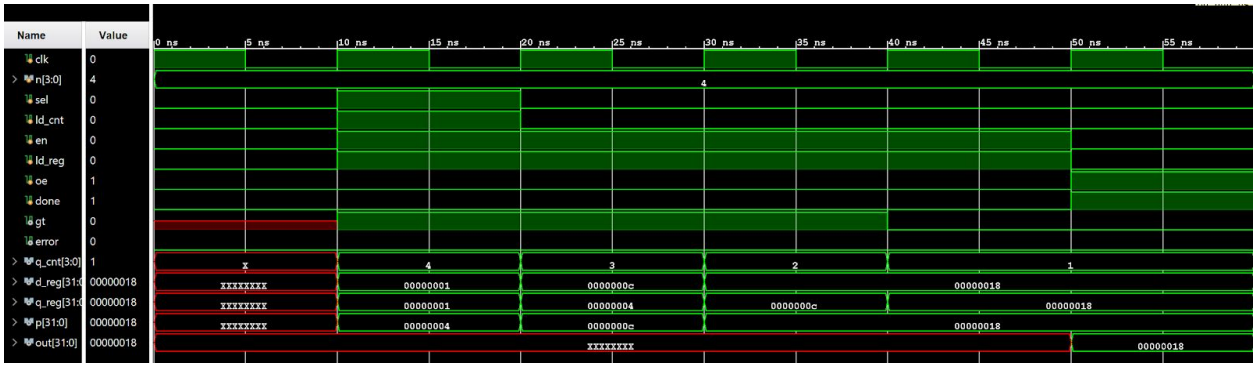

Figure 3: Factorial FSM Datapath for sample test n = 4

FPGA Validation

The FPGA board was set up so that switches 0 to 3 controlled input “n” and switch 5 controlled HILO_sel, with the LEDs above them lighting up when set active. When HILO_sel is set high, then the first four digits are displayed on the 7-segment display. The last four digits are displayed when HILO_sel is set low. The done flag is set to LED 15 and error is set to LED 14. The go signal is set to the left push button and the manual clock is set to the right push button.

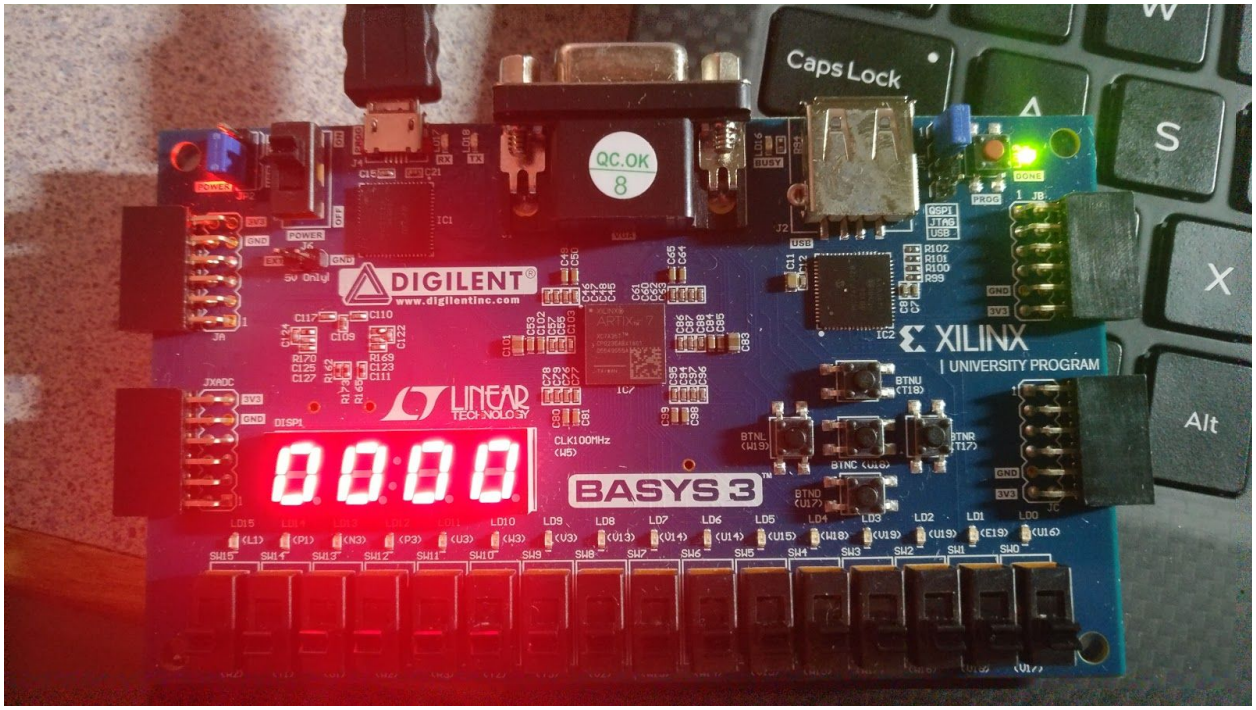

Figure 4: FPGA board programmed with factorial accelerator and on idle state.

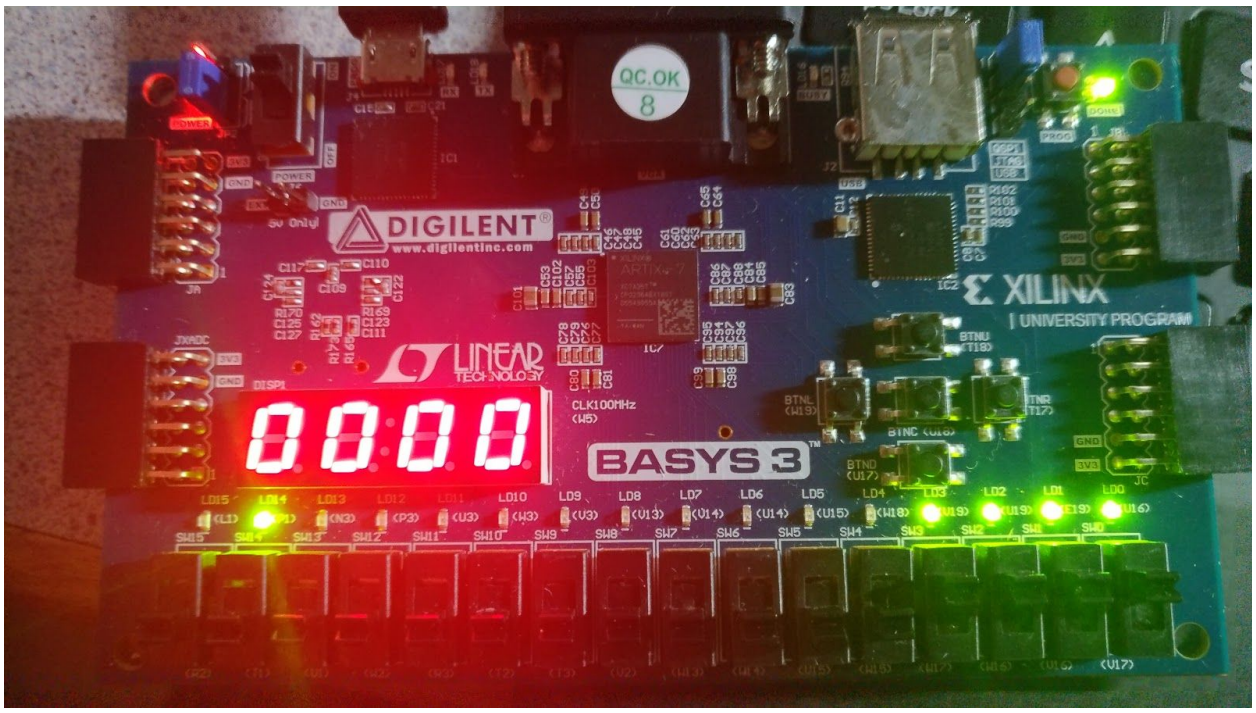

Figure 5: n value is set to 15. Error flag is active.

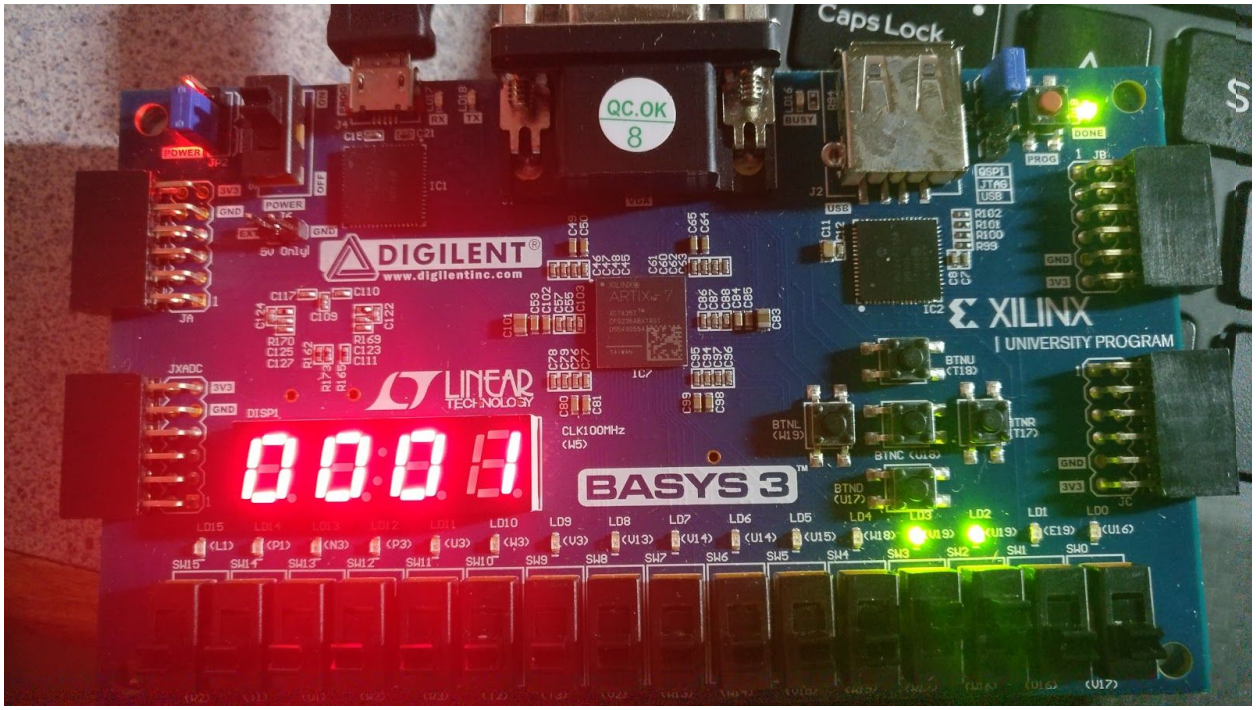

Figure 6: n value is set to 12. Go signal was sent and one clock cycle has been ticked.

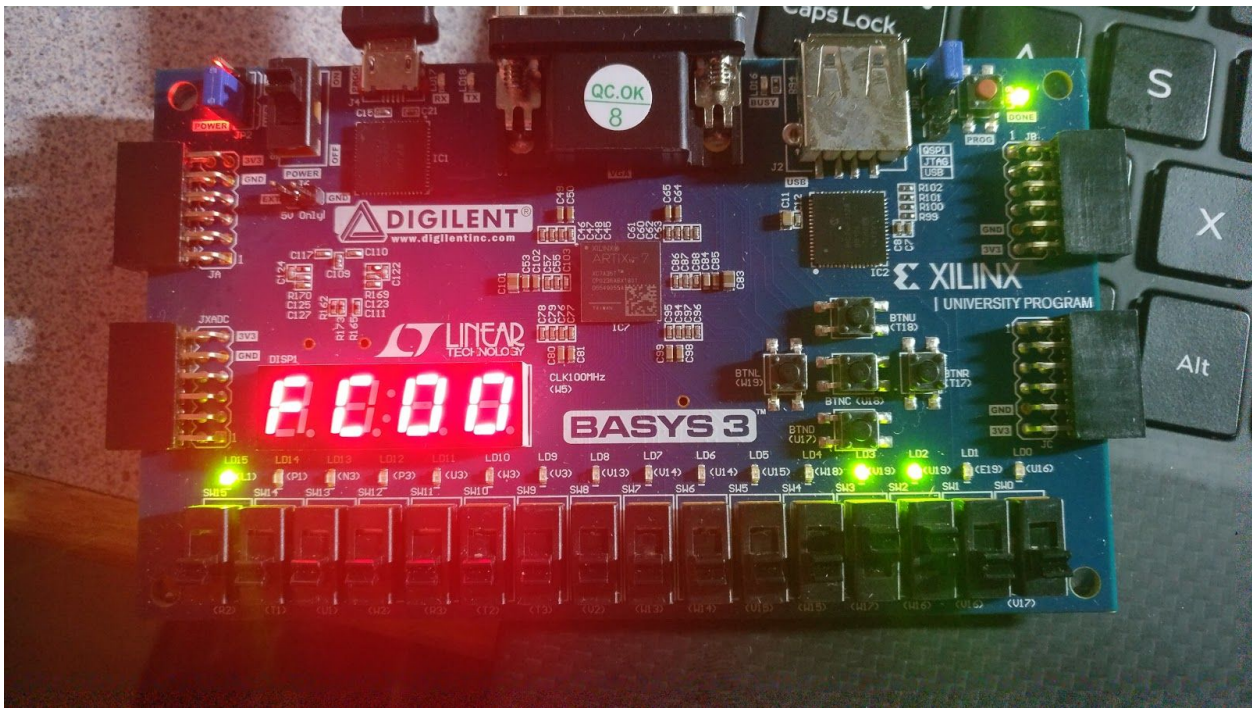

Figure 7: Done signal is active. HILO_sel is set to low. Display is showing the last four digits of the result.

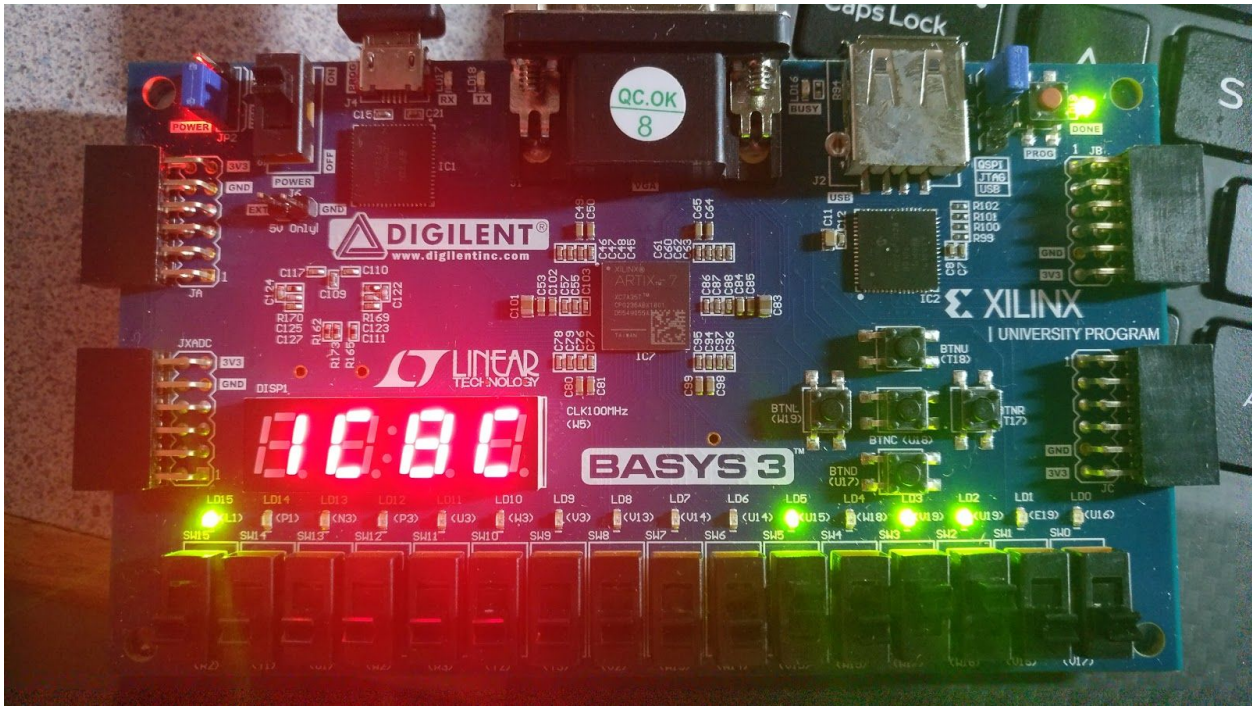

Figure 8: HILO_sel is set to high. Display is showing the first four digits of the result.

Successful Tasks

- Constructed diagram of system datapath
- Constructed diagram of CU-DP system block diagram
- Constructed ASM chart of datapath operation
- Constructed bubble diagram of next state logic
- Constructed output table of control unit
- Design of Factorial Accelerator datapath
- Verified functionality of datapath
- Design of Factorial Accelerator control unit
- Verified functionality of control unit
- Integration of datapath and control unit to create working system of Factorial Accelerator
- Factorial Accelerator validated on FPGA board

Conclusion

The factorial accelerator's datapath and control module functioned as expected on the testbench simulation. These modules combined leads to a functionally working factorial accelerator that was successfully validated on the FPGA board.

Appendix

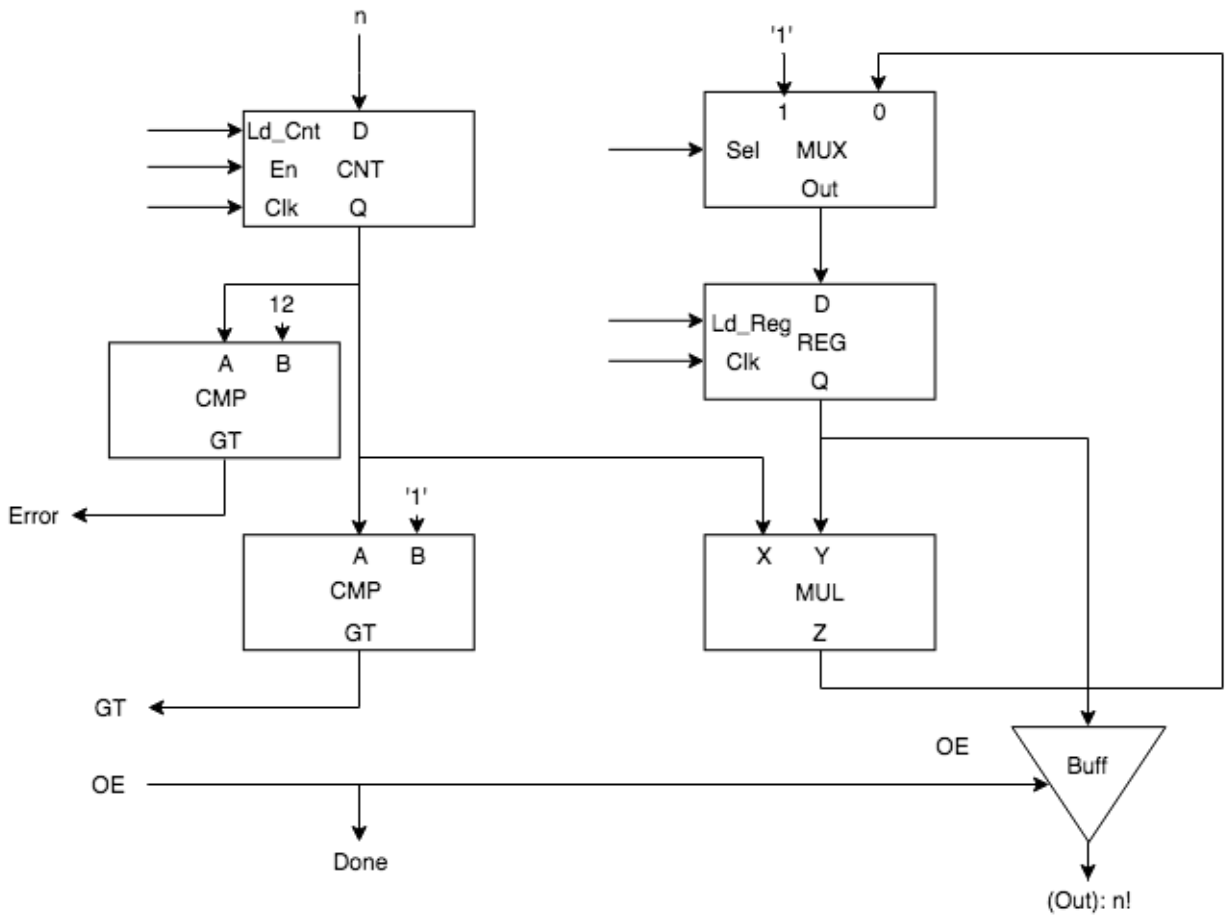

Figure 9: Datapath

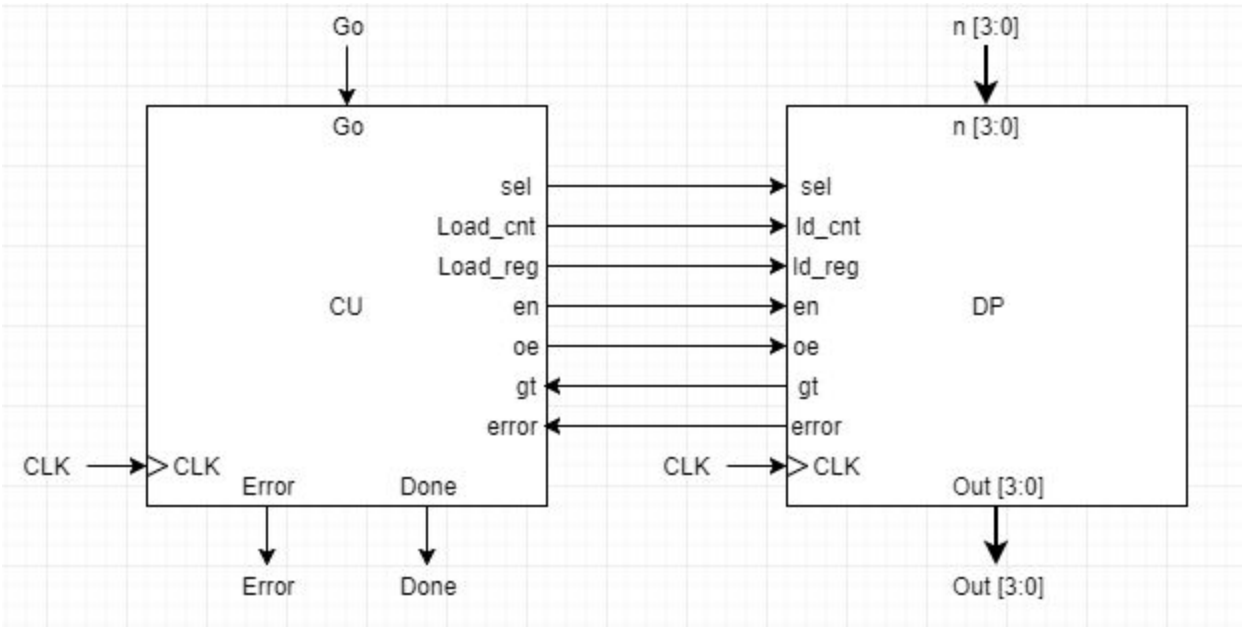

Figure 10: CU-DP model for the Factorial Accelerator.

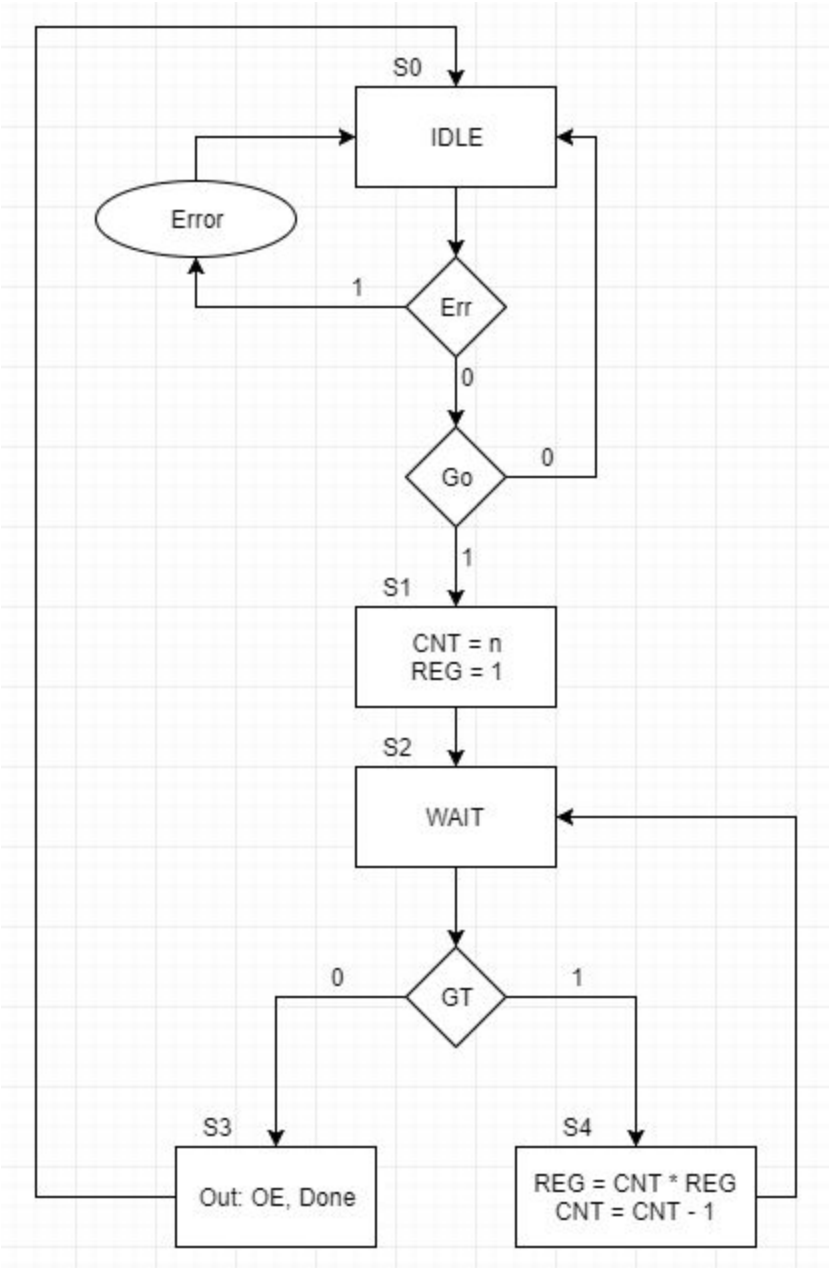

Figure 11: ASM Chart for Factorial Accelerator.

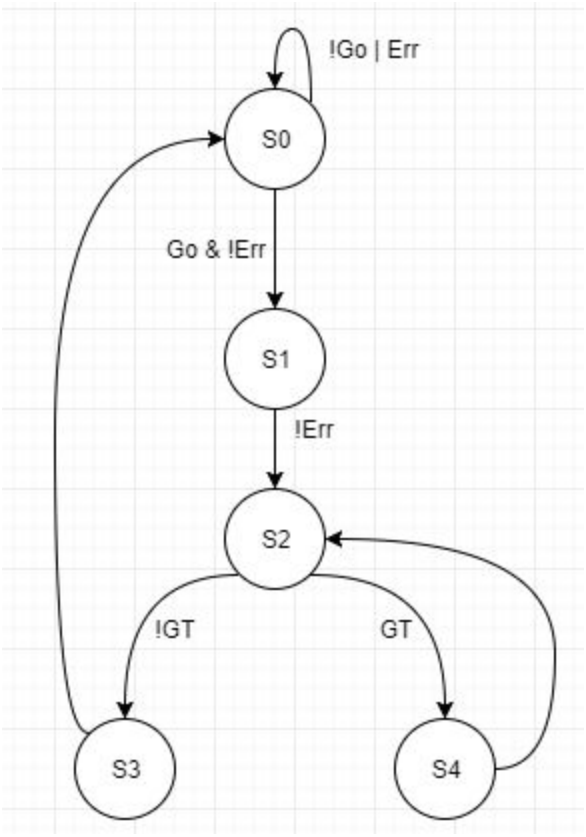

Figure 12: Next State Logic Bubble Diagram for Factorial Accelerator.

FACTORIAL_FPGA.v

```
`timescale 1ns / 1ps
module FACTORIAL_FPGA(
    input wire [3:0] n,
    input wire Go, button, clk100MHz, HILO_sel,
    output wire [3:0] LEDSEL,
    output wire [7:0] LEDOUT,
    output wire [15:0] LED
    );

    wire Done, Error, CLK, clk_5KHz;
    wire [31:0] Out;
    wire [3:0] dig0, dig1, dig2, dig3, dig4, dig5, dig6, dig7;
    wire [3:0] out0, out1, out2, out3;
    wire [7:0] s0, s1, s2, s3;

    assign LED[3:0] = n;
    assign LED[5] = HILO_sel;
    assign LED[14] = Error;
    assign LED[15] = Done;

    nfactorial facto (
        .go (Go),
        .clk (CLK),
        .n (n),
```

```
        .done (Done),
        .error (Error),
        .out (Out)
    );

    clk_gen clk (
        .clk100MHz (clk100MHz),
        .clk_5KHz (clk_5KHz)
    );

    button_debouncer debouncer (
        .clk (clk_5KHz),
        .button (button),
        .debounced_button (CLK)
    );

    bin2hex32 BinToHex (
        .value (Out),
        .dig0 (dig0),
        .dig1 (dig1),
        .dig2 (dig2),
        .dig3 (dig3),
        .dig4 (dig4),
        .dig5 (dig5),
        .dig6 (dig6),
        .dig7 (dig7)
        );

    HILO_MUX highlow (
        .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 (out3),
        .HW_dig2 (out2),
        .HW_dig1 (out1),
        .HW_dig0 (out0)
        );

    hex_to_7seg digit0 (
        .HEX (out0),
        .s (s0)
        );

    hex_to_7seg digit1 (
        .HEX (out1),
        .s (s1)
        );

    hex_to_7seg digit2 (
        .HEX (out2),
        .s (s2)
        );

    hex_to_7seg digit3 (
        .HEX (out3),
        .s (s3)
        );

    led_mux display (
```

```
        .clk (clk_5KHz),
        .LED3 (s3),
        .LED2 (s2),
        .LED1 (s1),
        .LED0 (s0),
        .LEDSEL (LEDSEL),
        .LEDOUT (LEDOUT)
        );

endmodule
```

nfactorial.v

```
module nfactorial(
      input clk,
      input go,
      input [3:0] n,
      output done,
      output error,
      output [31:0]out,
      //Troubleshooting variables
      output [31:0] q_cnt,
      output [31:0] q_reg,
      output [31:0] d_reg,
      output [31:0] p,
      output [1:0] ns,
      output [1:0] cs,
      output gt
    );

    wire [5:0] ctrl;
    //wire [1:0] ns,cs;
    //wire gt;

    assign done = ctrl[0];

    nfactorial_dp DP(
      .clk      (clk),
      .sel      (ctrl[5]),
      .ld_cnt   (ctrl[4]),
      .en       (ctrl[3]),
      .ld_reg   (ctrl[2]),
      .oe       (ctrl[1]),
      .done     (ctrl[0]),
      .n        (n),
      .gt       (gt),
      .error    (error),
       //outputs used for verification
      .q_cnt    (q_cnt),
      .q_reg    (q_reg),
      .d_reg    (d_reg),
      .p        (p),
      .out      (out)
    );

    control_unit CU(
      .clk      (clk),
      .go       (go),
      .gt       (gt),
```

```
      .error    (error),
      .out      (ctrl),
      .ns       (ns),
      .cs       (cs)
    );

endmodule
```

nfactorial_dp.v

```
`timescale 1ns / 1ps

module nfactorial_dp(
        input clk,
        input sel, ld_cnt,en,ld_reg,oe,done,
        input [3:0] n,
        output gt,
        output error,
        //outputs used for verification
        output [3:0] q_cnt,
        output [31:0] q_reg,
        output [31:0] d_reg,
        output [31:0] p,
        output [31:0] out
    );

    parameter WIDTH = 4;

    //Control Signals input from CU
    /*wire sel = ctrl[5];
    wire ld_cnt = ctrl[4];
    wire en = ctrl[3];
    wire ld_reg = ctrl[2];
    wire oe = ctrl[1];
    wire done = ctrl[0];*/

    //Counter output
    //wire error;
    //wire [WIDTH-1:0] q_cnt;
    //Multiplier output
    //wire [31:0] p;
    //Register
    //wire [31:0] d_reg;
    //wire [31:0] q_reg;

    counter CNT(
        .clk    (clk),
        .en     (en),
        .ld     (ld_cnt),
        .d      (n),
        .q      (q_cnt)
    );

    comp CMPERR(
        .a     (n),
        .b     (12),
        .gt    (error)
    );

    comp CMP(
        .a      (q_cnt),
```

```
        .b      (1),
        .gt     (gt)
    );

    mux2 #(32) MUX(
        .sel    (sel),
        .in0    (p),
        .in1    (1),
        .out    (d_reg)
    );


    register #(32)REG(
        .ld     (ld_reg),
        .clk    (clk),
        .d      (d_reg),
        .q      (q_reg)
    );


    /*multiplier#(32) MUL(
        .a      (q_cnt),
        .b      (q_reg),
        .p      (p)
    );*/
    assign p = q_cnt * q_reg;

    buff #(32)BUF(
        .oe     (oe),
        .d      (q_reg),
        .out    (out)
    );

endmodule
```

control_unit.v

```
`timescale 1ns / 1ps

module control_unit(
    input clk,
    input go,
    input gt,
    input error,
       output reg [5:0] out,
       //Verification variables
       output reg [1:0] cs, ns
    );
    // internal constants (states)
    // IDLE = S0, WAIT = S3, ADD = S4, SUB = S5....etc
    parameter IDLE = 2'b00, ST1 = 2'b01, ST2 = 2'b10, ERR =  2'b11;

    // internal registers
    //reg [2:0] ns;
    //reg [2:0] cs;

    initial // start at IDLE;
        begin
            cs = IDLE;
        end
```

```
    always @ (cs,go,gt,error) // FSM state transitions
       begin
           case(cs)
               IDLE: begin // IF GO is active, go to ST1, if not stay idle
                       if (go)
                         ns = ST1;
                       else
                         ns = IDLE;
                     end
               ST1: begin
                       if (error)
                           ns = ERR;
                       else
                       ns = ST2;
                    end
               ST2: begin
                       if (gt)
                         ns = ST2;
                       else
                         ns = IDLE;
                     end
                ERR: ns = IDLE;
               default: ns = IDLE;
           endcase
       end

     // Output logic combinatorial
     always @ (ns,cs,gt)
       begin
           case(ns)
               IDLE: //out = 6'b000000;
                     begin
                       if(cs == ST2)
                         out = 6'b000011;
                       else
                         out = 6'b000000;
                     end//*/
               ST1:  out = 6'b111100;
               ST2:  begin
                       if  (gt == 1)
                         out = 6'b001100;
                       else
                         out = 6'b000011;
                     end
                ERR: out = 6'b000001;
           endcase
           #1;
      end

      // Sequential
     always @ (posedge clk)
       begin
           cs <= ns; // change states on positive edge clock
       end
endmodule
```

counter.v

```
`timescale 1ns / 1ps
```

```
module counter#(parameter WIDTH = 4)(
    input clk,
    input en,
    input ld,
    input [WIDTH-1:0] d,
    output reg [WIDTH-1:0] q
    );

always @ (posedge clk) begin
   if(en)//ce = 1
      begin
        if(ld)//ld = 1 Load
          q = d;
        else//ld = 0
          q = q-1;
      end
    end
endmodule
```

comp.v

```
`timescale 1ns / 1ps

module comp(
    input [3:0] a,
    input [3:0] b,
    output gt
  );
  assign gt = (a>b) ? 1 : 0;

endmodule
```

mux.v

```
module mux2 #(parameter 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
```

register.v

```
module register #(parameter WIDTH = 4)(
         input ld,
         input clk,
         input [WIDTH-1:0] d,
         output reg [WIDTH-1:0] q
```

```
    );

    always@(posedge clk)
    begin
        if(ld)
            q = d;
        else
            q = q;
    end

endmodule
```

buffer.v

```
module buff#(parameter WIDTH = 4)(
        input oe,
        input [WIDTH-1:0] d,
        output reg[WIDTH-1:0] out
    );

    always@(oe)
    begin
            out = d;
    end
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

```
`timescale 1ns / 1ps
module bin2hex32(
    input wire [31:0] value,
    output wire [3:0] dig0, dig1, dig2, dig3, dig4, dig5, dig6, 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
```

HILO_MUX.v

```
`timescale 1ns / 1ps
module HILO_MUX(
    input wire [3:0] HI_dig3, HI_dig2, HI_dig1, HI_dig0,
```

```
    input wire [3:0] LO_dig3, LO_dig2, LO_dig1, LO_dig0,
    input wire HILO_sel,
    output wire [3:0] HW_dig3, HW_dig2, HW_dig1, 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
```

hex_to_7seg.v

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

    always @ (HEX) begin
        case (HEX)
               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'b10001000;
               4'hB: s = 8'b10000000;
               4'hC: s = 8'b11000110;
               4'hD: s = 8'b11000000;
               4'hE: s = 8'b10000110;
               4'hF: s = 8'b10001110;
            default: s = 8'b01111111;
       endcase
    end

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)
               2'd0: led_ctrl <= {4'b1110, LED0};
               2'd1: led_ctrl <= {4'b1101, LED1};
               2'd2: led_ctrl <= {4'b1011, LED2};
               2'd3: led_ctrl <= {4'b0111, LED3};
            default: led_ctrl <= {4'b1111, 8'hFF};
        endcase
    end

endmodule
```

FACTORIAL.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}];

# Buttons
set_property -dict {PACKAGE_PIN W19 IOSTANDARD LVCMOS33} [get_ports {Go}]; # Left Button
set_property -dict {PACKAGE_PIN T17 IOSTANDARD LVCMOS33} [get_ports {button}]; # Right
Button

# Switches
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {n[0]}];  # Switch 0
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {n[1]}];  # Switch 1
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports {n[2]}];  # Switch 2
set_property -dict {PACKAGE_PIN W17 IOSTANDARD LVCMOS33} [get_ports {n[3]}];  # Switch 3
set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {HILO_sel}];  # Switch 5
# LEDs
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {LED[0]}];  # LED 0
set_property -dict {PACKAGE_PIN E19 IOSTANDARD LVCMOS33} [get_ports {LED[1]}];  # LED 1
set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports {LED[2]}];  # LED 2
set_property -dict {PACKAGE_PIN V19 IOSTANDARD LVCMOS33} [get_ports {LED[3]}];  # LED 3
set_property -dict {PACKAGE_PIN W18 IOSTANDARD LVCMOS33} [get_ports {LED[4]}];  # LED 4
set_property -dict {PACKAGE_PIN U15 IOSTANDARD LVCMOS33} [get_ports {LED[5]}];  # LED 5
set_property -dict {PACKAGE_PIN U14 IOSTANDARD LVCMOS33} [get_ports {LED[6]}];  # LED 6
set_property -dict {PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports {LED[7]}];  # LED 7
set_property -dict {PACKAGE_PIN V13 IOSTANDARD LVCMOS33} [get_ports {LED[8]}];  # LED 8
set_property -dict {PACKAGE_PIN V3  IOSTANDARD LVCMOS33} [get_ports {LED[9]}];  # LED 9
set_property -dict {PACKAGE_PIN W3  IOSTANDARD LVCMOS33} [get_ports {LED[10]}]; # LED 10
set_property -dict {PACKAGE_PIN U3  IOSTANDARD LVCMOS33} [get_ports {LED[11]}]; # LED 11
set_property -dict {PACKAGE_PIN P3  IOSTANDARD LVCMOS33} [get_ports {LED[12]}]; # LED 12
set_property -dict {PACKAGE_PIN N3  IOSTANDARD LVCMOS33} [get_ports {LED[13]}]; # LED 13
set_property -dict {PACKAGE_PIN P1  IOSTANDARD LVCMOS33} [get_ports {LED[14]}]; # LED 14
set_property -dict {PACKAGE_PIN L1  IOSTANDARD LVCMOS33} [get_ports {LED[15]}]; # LED 15

# 7 segment display
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
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
```