

***Tasks***

This lab focuses on the designing a factorial calculator for any value *n*. To fully complete and understand the lab three tasks per component are implemented: design, simulation validation, FPGA validation. In order to validate the waveform testbench code was written in relation to the design files. The waveform outputs were compared to the function tables for each component. To validate the factorial using hardware an xdc was programmed to the Nexys4 DDR board. The constraints file allows the utilization of switches, leds and the seven segment display. The reset signal was not being tested fully through the simulations. However corrections were made to test the reset signal as active and not active. It is recommended to be test all control signals. Figure six in the appendix shows the simulation waveform output for each signal.

The list of tasks were fully completed in lab:

* Waveform validation of factorial
* Design system datapath using basic building blocks
* Design block diagrams
* Construct output table
* Hardware validation using Nexys4 DDR board
* Stimuli and Hardware Verification of factorial

The list of tasks were not fully completed in lab but were finished afterword:

* Simulation verification of the reset button

Table 1: Output Table

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CS | Go | Sel1 | CE | UD | LD1 | LD2 | Sel2 | Is\_GT | NS | Done |
| S0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | S0 | 0 |
| S0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | - | S1 | 0 |
| S1 | - | 0 | 1 | 0 | 1 | 1 | 0 | - | S2 | 0 |
| S2 | - | 1 | 0 | 0 | 0 | 0 | 0 | 0 | S3 | 0 |
| S2 | - | 1 | 0 | 0 | 0 | 0 | 0 | 1 | S4 | 0 |
| S3 | - | 0 | 0 | 0 | 0 | 0 | 1 | - | S0 | 1 |
| S4 | - | 1 | 1 | 0 | 0 | 1 | 0 | - | S2 | 0 |

***Appendix***

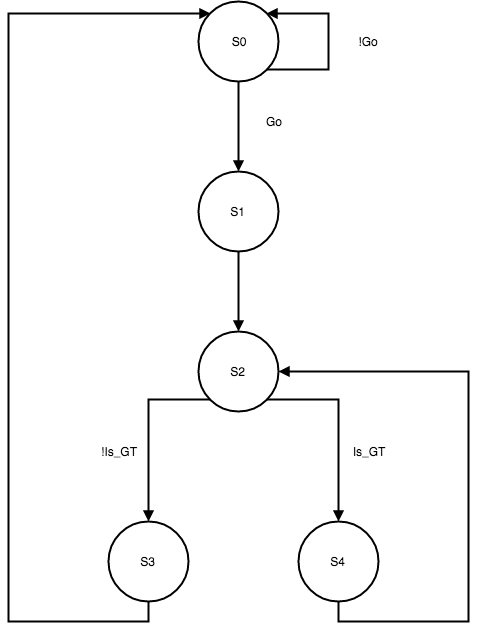
******

Figure 1:State Transition Diagram(Bubble Diagram)

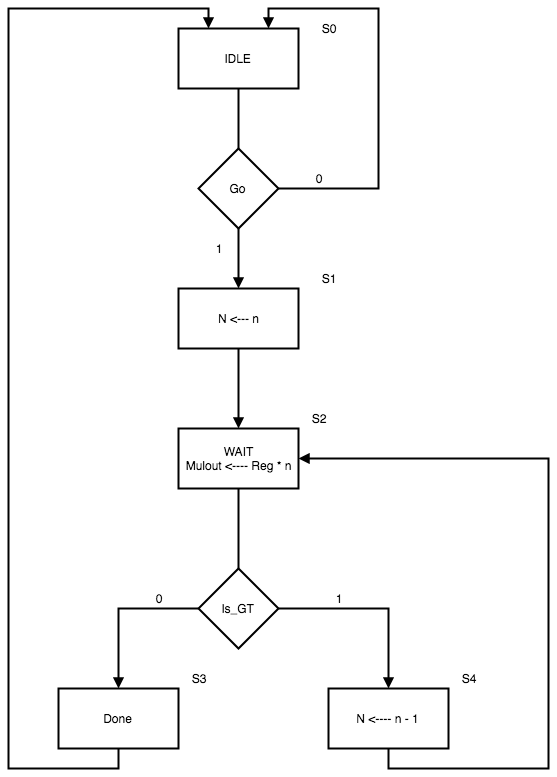


Figure 2: Control Unit ASM

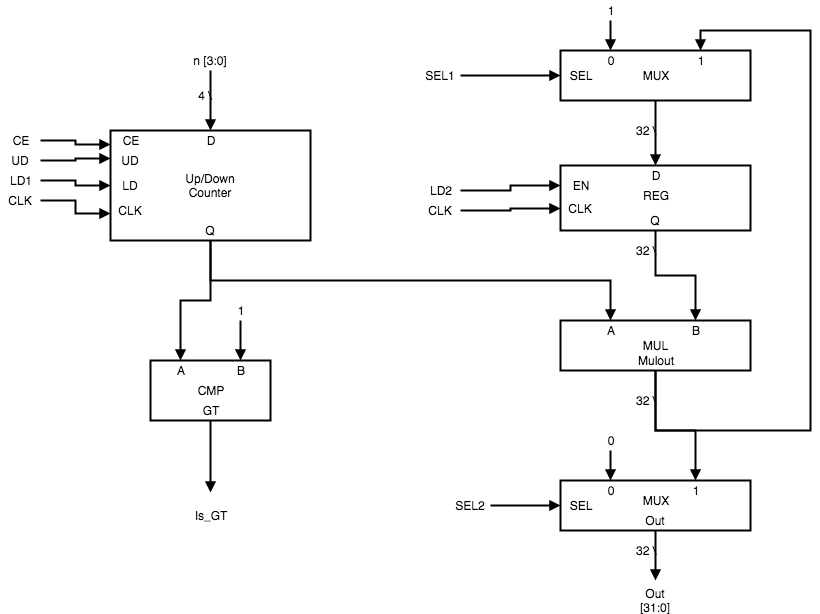


Figure 3: ASM

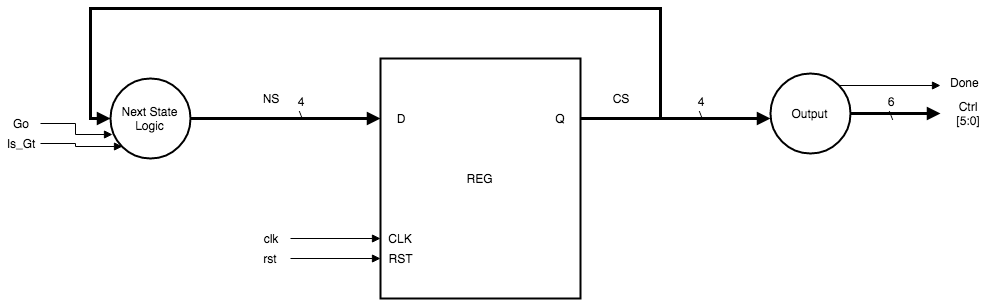


Figure 4: Register Block Diagram

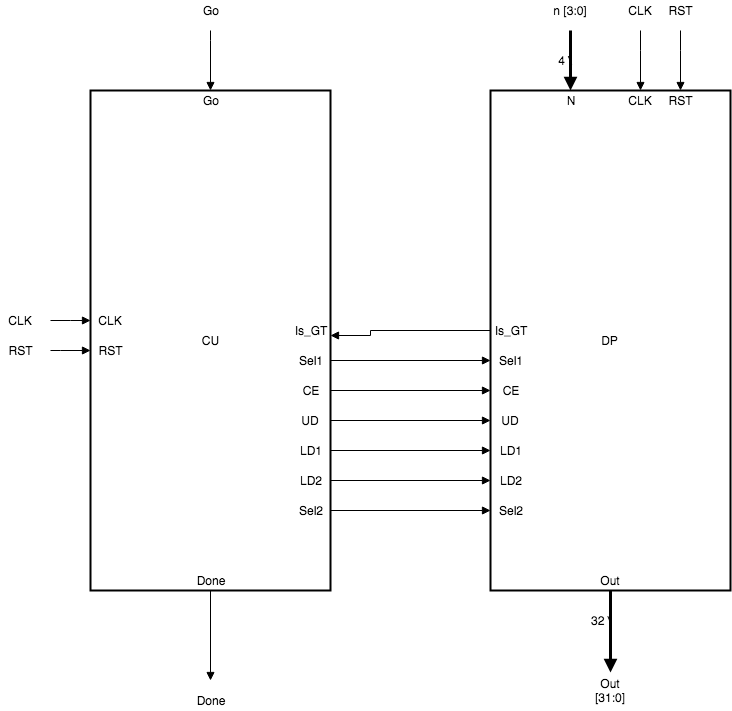


Figure 5: Control Unit and Datapath Block Diagram

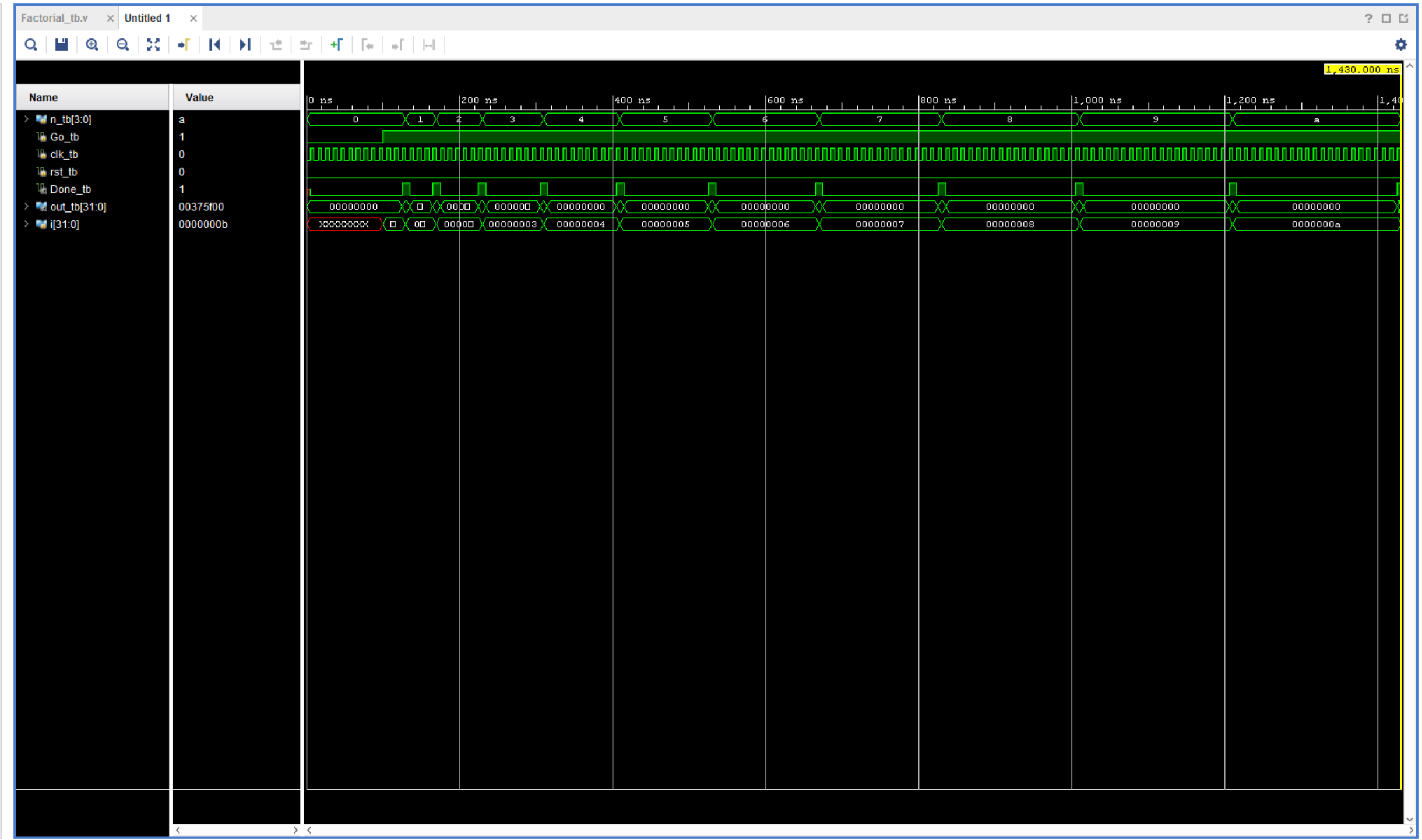


Figure 6: Factorial Testbench Waveform

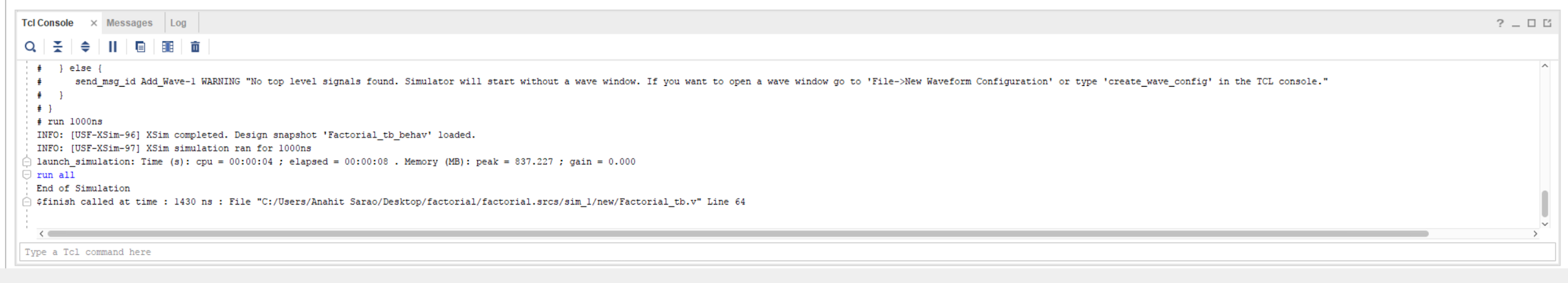


Figure 7: Simulation Validation Console Output of Testbench

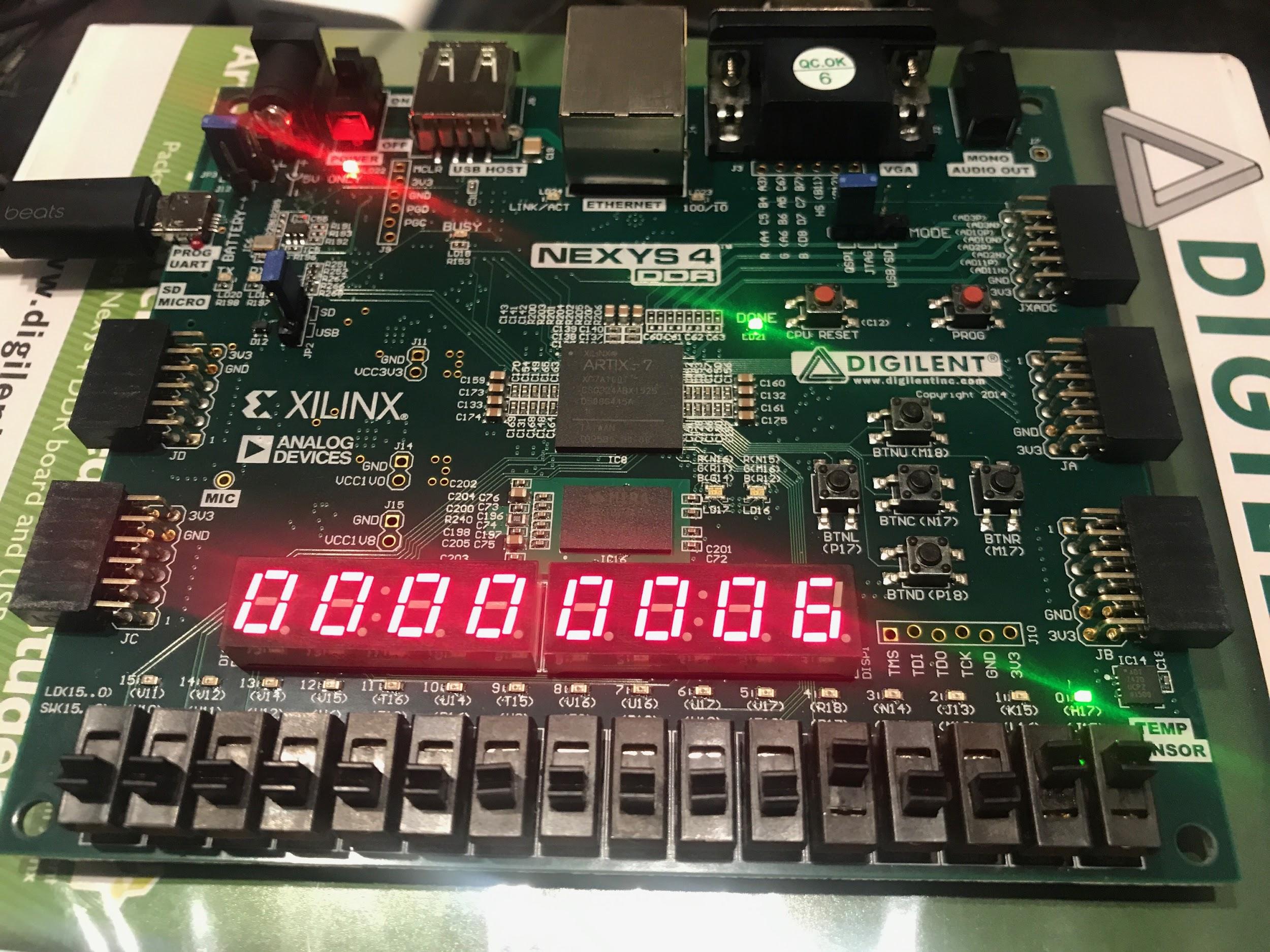


Figure 8: Factorial of 3

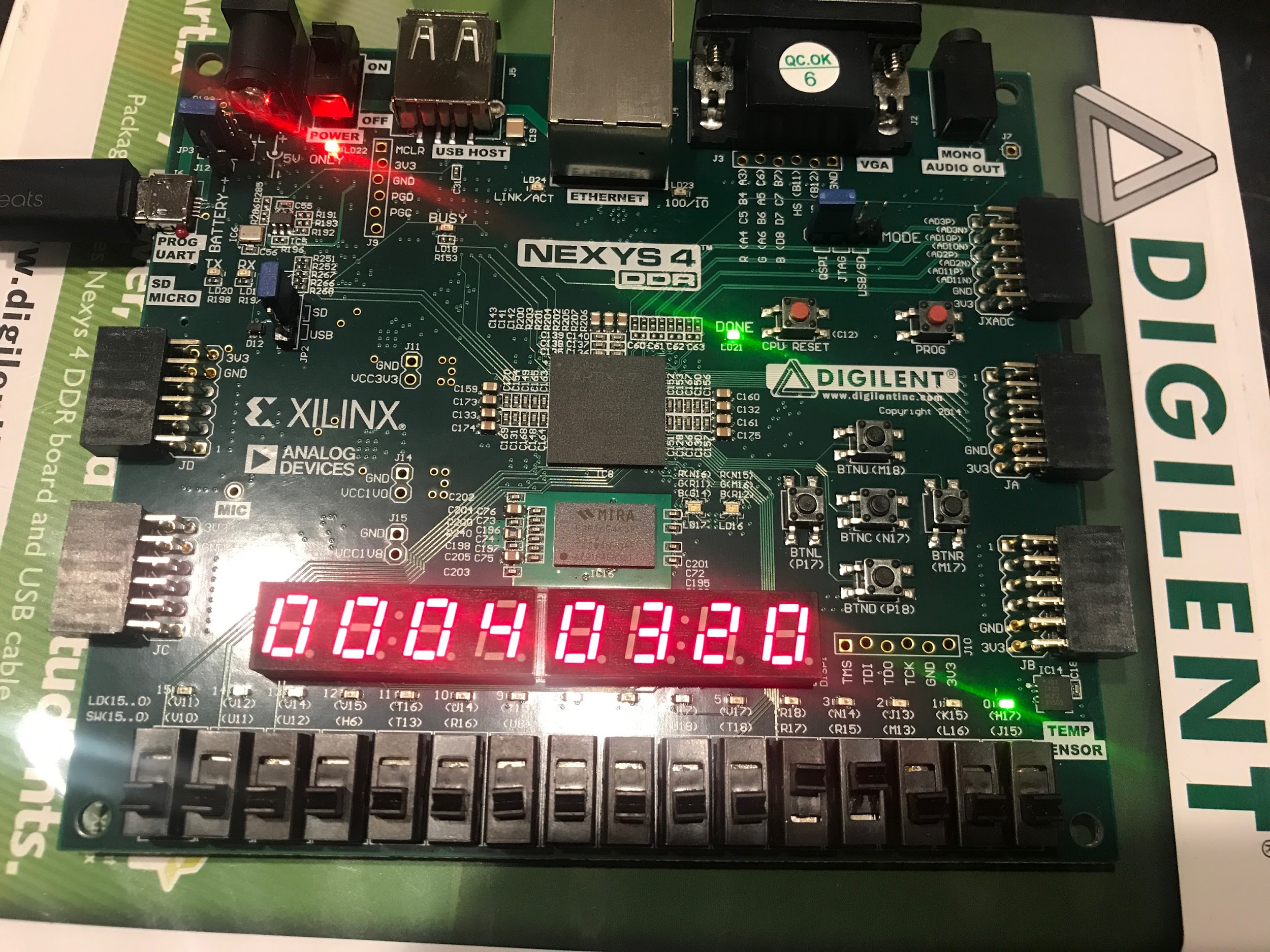


Figure 9: Factorial of 8

***Source Code***

set\_property -dict { PACKAGE\_PIN E3 IOSTANDARD LVCMOS33 } [get\_ports { clk100MHz }];

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

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { n[0] }];

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { n[1] }];

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { n[2] }];

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { n[3] }];

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { Done }];

set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { rst }];

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

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { go }];

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[0] }];

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[1] }];

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[2] }];

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[3] }];

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[4] }];

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[5] }];

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[6] }];

set\_property -dict { PACKAGE\_PIN H15 IOSTANDARD LVCMOS33 } [get\_ports { LEDOUT[7] }];

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[0] }];

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[1] }];

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[2] }];

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[3] }];

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[4] }];

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[5] }];

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[6] }];

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { LEDSEL[7] }];

module cu\_tb();

reg Go\_tb, clk\_tb, rst\_tb, Is\_Gt\_tb;

wire Done\_tb;

wire [5:0] ctrl\_tb;

Factorial\_CU DUT(.Go(Go\_tb), .clk(clk\_tb), .rst(rst\_tb), .Is\_Gt(Is\_Gt\_tb), .Done(Done\_tb), .ctrl(ctrl\_tb));

parameter

IDLE = 6'b0\_0\_0\_0\_0\_0,

S1 = 6'b0\_1\_0\_1\_1\_0,

S2 = 6'b1\_0\_0\_0\_0\_0,

S3 = 6'b1\_0\_0\_0\_0\_1,

S4 = 6'b1\_1\_0\_0\_1\_0;

task tick; begin #5 clk\_tb = 1; #5 clk\_tb = 0; end endtask

initial

begin

clk\_tb = 0; rst\_tb = 0; Go\_tb = 1; Is\_Gt\_tb = 1;

tick;

if(ctrl\_tb != IDLE) $display("Error with IDLE");

tick;

if(ctrl\_tb != S1) $display("Error with S1");

tick;

if(ctrl\_tb != S2) $display("Error with S2");

tick;

if(ctrl\_tb != S4) $display("Error with S4");

tick;

Is\_Gt\_tb = 0;

tick;

if(ctrl\_tb != S3)

begin

$display("Error with S3");

if(Done\_tb != 1) $display("Error with Done");

end

$display("End of Simulation");

$finish;

end

endmodule

module dp\_tb();

reg [3:0] n\_tb;

reg [5:0] ctrl\_tb;

reg clk\_tb, rst\_tb;

wire [31:0] out\_tb;

wire Is\_GT\_tb;

integer i;

Factorial\_DP DUT( .n(n\_tb), .ctrl(ctrl\_tb), .clk(clk\_tb), .rst(rst\_tb), .out(out\_tb), .Is\_GT(Is\_GT\_tb));

parameter

IDLE = 6'b0\_0\_0\_0\_0\_0,

S1 = 6'b0\_1\_0\_1\_1\_0,

S2 = 6'b1\_0\_0\_0\_0\_0,

S3 = 6'b1\_0\_0\_0\_0\_1,

S4 = 6'b1\_1\_0\_0\_1\_0;

task tick; begin #5 clk\_tb = 1; #5 clk\_tb = 0; end endtask

initial

begin

clk\_tb = 0; rst\_tb = 0;

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

begin

n\_tb = i;

ctrl\_tb = IDLE; tick;

ctrl\_tb = S1; tick;

while(Is\_GT\_tb)

begin

ctrl\_tb = S2; tick;

ctrl\_tb = S4; tick;

end

ctrl\_tb = S3; tick;

case(i)

4'd0: begin if(out\_tb != 0) $display("Error with 0"); end

4'd1: begin if(out\_tb != 1) $display("Error with 1"); end

4'd2: begin if(out\_tb != 2) $display("Error with 2"); end

4'd3: begin if(out\_tb != 6) $display("Error with 3"); end

4'd4: begin if(out\_tb != 24) $display("Error with 4"); end

4'd5: begin if(out\_tb != 120) $display("Error with 5"); end

4'd6: begin if(out\_tb != 720) $display("Error with 6"); end

4'd7: begin if(out\_tb != 5040) $display("Error with 7"); end

4'd8: begin if(out\_tb != 40320) $display("Error with 8"); end

4'd9: begin if(out\_tb != 362880) $display("Error with 9"); end

4'd10: begin if(out\_tb != 3628800) $display("Error with 10"); end

endcase

end

$display("End of Sim");

$finish;

end

endmodule

module Factorial\_tb();

reg [3:0] n\_tb;

reg Go\_tb, clk\_tb, rst\_tb;

wire Done\_tb;

wire [31:0] out\_tb;

integer i;

Factorial DUT (.n(n\_tb), .Go(Go\_tb), .clk(clk\_tb), .rst(rst\_tb),.Done(Done\_tb), .out(out\_tb));

task tick;

begin

#5 clk\_tb = 1;

#5 clk\_tb = 0;

end

endtask

initial

begin

n\_tb = 0; clk\_tb = 0; rst\_tb = 0;

Go\_tb = 0; tick; tick; tick; tick; tick; tick; tick; tick; tick; tick;

Go\_tb = 1;

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

begin

n\_tb = i;

tick;

while(!Done\_tb)

begin

tick;

end

case(i)

4'd0: begin if(out\_tb != 1) $display("Error with 0"); end

4'd1: begin if(out\_tb != 1) $display("Error with 1"); end

4'd2: begin if(out\_tb != 2) $display("Error with 2"); end

4'd3: begin if(out\_tb != 6) $display("Error with 3"); end

4'd4: begin if(out\_tb != 24) $display("Error with 4"); end

4'd5: begin if(out\_tb != 120) $display("Error with 5"); end

4'd6: begin if(out\_tb != 720) $display("Error with 6"); end

4'd7: begin if(out\_tb != 5040) $display("Error with 7"); end

4'd8: begin if(out\_tb != 40320) $display("Error with 8"); end

4'd9: begin if(out\_tb != 362880) $display("Error with 9"); end

4'd10: begin if(out\_tb != 3628800) $display("Error with 10"); end

endcase

end

$display("End of Simulation");

$finish;

end

endmodule

module Alu(input [31:0] in1, in2,

input [1:0] c,

output reg [31:0] aluout);

always @ (in1, in2, c)

begin

case (c)

2'b00:

begin

if(in2 == 0) begin aluout = 1; end

else begin aluout = in1 \* in2; end

end

2'b01: aluout = in1 & in2;

2'b10: aluout = 1;

default: aluout = in2; // 2'b11;

endcase

end

endmodule

module bcd\_to\_7seg(input [3:0] BCD,

output 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

4'b1001: begin s0=1; s1=0; s2=1; s3=0; s4=0; s5=0; s6=0; end

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

endcase

end

endmodule // end bcd\_to\_7seg

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

module button\_debouncer #(parameter depth = 16)(input wire clk, button,

output reg debounced\_button);

localparam history\_max = (2\*\*depth)-1;

reg [depth-1:0] history;

always @(posedge clk)

begin

history <= {button, history[depth-1:1]};

debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;

end

endmodule

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 // end clk\_gen

module comparater(input [31:0] A, B,

output reg greater);

always @(A,B)

begin

greater <= 0;

if(A > B)

begin

greater <= 1;

end

end

endmodule

module Dreg(input [31:0] D,

input en, clk, rst,

output reg [31:0] Q);

always @(posedge clk, posedge rst)

begin

if(rst)

begin

Q <= 32'd0;

end

else

begin

if(en) Q <= D;

else Q <= Q;

end

end

endmodule

module Factorial(input [3:0] n,

input Go, clk, rst,

output Done,

output [31:0] out);

wire [5:0] ctrl;

wire Is\_Gt;

Factorial\_CU U0 (.Go(Go), .clk(clk), .rst(rst), .Is\_Gt(Is\_Gt), .Done(Done), .ctrl(ctrl));

Factorial\_DP U1 ( .n(n), .ctrl(ctrl), .clk(clk), .rst(rst), .out(out), .Is\_GT(Is\_Gt));

endmodule

module Factorial\_CU(input Go, clk, rst, Is\_Gt,

output reg Done,

output reg [5:0] ctrl);

reg [3:0] ns, cs;

parameter

IDLE = 6'b0\_0\_0\_0\_0\_0,

S1 = 6'b0\_1\_0\_1\_1\_0,

S2 = 6'b1\_0\_0\_0\_0\_0,

S3 = 6'b1\_0\_0\_0\_0\_1,

S4 = 6'b1\_1\_0\_0\_1\_0;

always @(Go, Is\_Gt, Done, ns, cs)

begin

case(cs)

4'd0:

begin

Done = 0;

if(Go) begin ns = 4'd1; end

else begin ns = 4'd0; end

end

4'd1: begin Done = 0; ns = 4'd2; end

4'd2:

begin

if(Is\_Gt) begin ns = 4'd4; Done = 0; end

else begin ns = 4'd3; Done = 0; end

end

4'd3:

begin

Done = 1;

ns = 4'd0;

end

4'd4:

begin

Done = 0;

ns = 4'd2;

end

default: ns = 4'd0;

endcase

end

always @(posedge clk, posedge rst)

begin

if(rst)

cs <= 0;

else

cs <= ns;

end

always @(cs, ctrl) //{Sel1, ce, ud, ld1, ld2, sel2} = ctrl;

begin

case(cs)

4'd0: ctrl = IDLE;

4'd1: ctrl = S1;

4'd2: ctrl = S2;

4'd3: ctrl = S3;

4'd4: ctrl = S4;

endcase

end

endmodule

module Factorial\_DP(input [3:0] n,

input [5:0] ctrl,

input clk, rst,

output [31:0] out,

output Is\_GT);

wire [3:0] Cnt\_Reg;

wire [31:0] q\_out, aluout, q2\_out, mux\_out1, mux\_out2;

wire IS\_equal;

Ud\_Cnt\_4 u1(.D(n), .LD(ctrl[2]), .UD(ctrl[3]), .CE(ctrl[4]), .CLK(clk), .RST(rst), .Q(Cnt\_Reg));

comparater u2(.A({28'd0,Cnt\_Reg}), .B(32'd1), .greater(Is\_GT));

Dreg u3(.D(mux\_out1), .en(ctrl[1]), .clk(clk), .rst(rst), .Q(q\_out));

Alu u4(.in1(q\_out), .in2({28'd0,Cnt\_Reg}), .c(2'b00), .aluout(aluout));

Mux u6 (.in1(32'd1), .in2(aluout), .sel(ctrl[5]), .out(mux\_out1));

Mux u7 (.in1(32'd0), .in2(aluout), .sel(ctrl[0]), .out(out));

endmodule

module Factorial\_fpga(input [3:0] n,

input go, clk100MHz, control, rst,

output Done,

output [7:0] LEDOUT, LEDSEL );

supply1 [7:0] vcc;

wire DONT\_USE, clk\_5KHz;

wire [31:0] out;

wire [3:0] dig0, dig1, dig2, dig3, dig4, dig5, dig6, dig7;

wire [6:0] out0, out1, out2, out3, out4, out5, out6, out7;

Factorial U0 (.n(n), .Go(go), .clk(debouncedButton), .rst(rst), .Done(Done), .out(out));

bin2bcd32 U1 (.value(out), .dig0(dig0), .dig1(dig1), .dig2(dig2), .dig3(dig3), .dig4(dig4), .dig5(dig5), .dig6(dig6), .dig7(dig7));

showNumber U2 (.dig0(dig0), .dig1(dig1), .dig2(dig2), .dig3(dig3), .dig4(dig4), .dig5(dig5), .dig6(dig6), .dig7(dig7),

.out0(out0), .out1(out1), .out2(out2), .out3(out3), .out4(out4), .out5(out5), .out6(out6), .out7(out7));

led\_mux U3(clk\_5KHz, rst, {1'b1, out7}, {1'b1, out6}, {1'b1, out5}, {1'b1, out4}, {1'b1, out3}, {1'b1, out2}, {1'b1, out1},{1'b1, out0}, LEDOUT, LEDSEL);

clk\_gen U4(.clk100MHz(clk100MHz), .rst(rst), .clk\_4sec(DONT\_USE), .clk\_5KHz(clk\_5KHz));

button\_debouncer U5(.clk(clk\_5KHz), .button(control), .debounced\_button(debouncedButton));

Endmodule

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

module Mux(input [31:0] in1, in2,

input sel,

output reg [31:0] out);

always @(in1, in2, sel, out)

begin

if(sel)

begin

out = in2;

end

else

begin

out = in1;

end

end

endmodule

module showNumber(input [3:0] dig0, dig1, dig2, dig3, dig4, dig5, dig6, dig7,

output [6:0] out0, out1, out2, out3, out4, out5, out6, out7);

bcd\_to\_7seg IN1( .BCD(dig0), .s0(out0[0]), .s1(out0[1]), .s2(out0[2]), .s3(out0[3]), .s4(out0[4]), .s5(out0[5]), .s6(out0[6]));

bcd\_to\_7seg IN2( .BCD(dig1), .s0(out1[0]), .s1(out1[1]), .s2(out1[2]), .s3(out1[3]), .s4(out1[4]), .s5(out1[5]), .s6(out1[6]));

bcd\_to\_7seg IN3( .BCD(dig2), .s0(out2[0]), .s1(out2[1]), .s2(out2[2]), .s3(out2[3]), .s4(out2[4]), .s5(out2[5]), .s6(out2[6]));

bcd\_to\_7seg IN4( .BCD(dig3), .s0(out3[0]), .s1(out3[1]), .s2(out3[2]), .s3(out3[3]), .s4(out3[4]), .s5(out3[5]), .s6(out3[6]));

bcd\_to\_7seg IN5( .BCD(dig4), .s0(out4[0]), .s1(out4[1]), .s2(out4[2]), .s3(out4[3]), .s4(out4[4]), .s5(out4[5]), .s6(out4[6]));

bcd\_to\_7seg IN6( .BCD(dig5), .s0(out5[0]), .s1(out5[1]), .s2(out5[2]), .s3(out5[3]), .s4(out5[4]), .s5(out5[5]), .s6(out5[6]));

bcd\_to\_7seg IN7( .BCD(dig6), .s0(out6[0]), .s1(out6[1]), .s2(out6[2]), .s3(out6[3]), .s4(out6[4]), .s5(out6[5]), .s6(out6[6]));

bcd\_to\_7seg IN8( .BCD(dig7), .s0(out7[0]), .s1(out7[1]), .s2(out7[2]), .s3(out7[3]), .s4(out7[4]), .s5(out7[5]), .s6(out7[6]));

endmodule

module Ud\_Cnt\_4(input [3:0] D,

input LD, UD, CE, CLK, RST,

output reg [3:0] Q);

always @(posedge CLK, posedge RST)

begin

if(RST) Q <= 4'd0;

else if(CE)

begin

if(LD) Q <= D;

else

begin

case(UD)

1: Q <= Q + 1;

0: Q <= Q - 1;

endcase

end

end

else Q <= Q;

end

endmodule