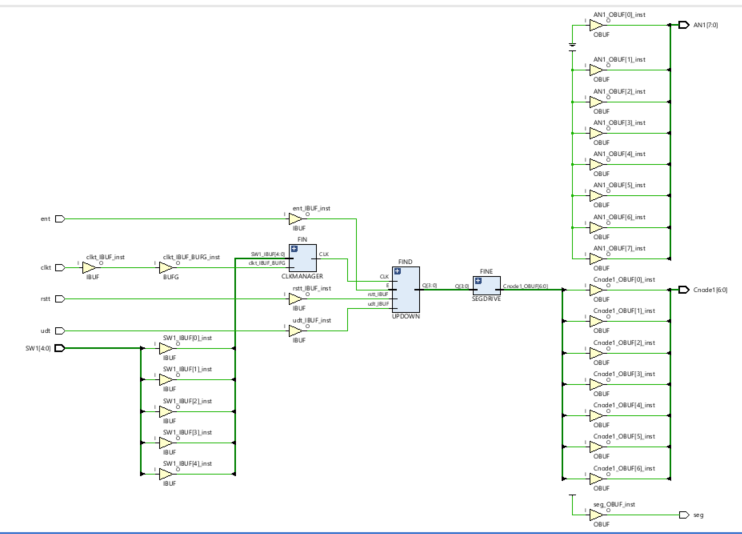
**ECE3300L Lab3 Group H Report (Sherwin Sathish & Mohamed Hamida)**

**SCHEMATIC():**

****

***Xdc for top.v:***

## This file is a general .xdc for the Nexys A7-100T

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

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

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

##Switches

set\_property -dict { PACKAGE\_PIN J15 IOSTANDARD LVCMOS33 } [get\_ports { SW1[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

set\_property -dict { PACKAGE\_PIN L16 IOSTANDARD LVCMOS33 } [get\_ports { SW1[1] }]; #IO\_L3N\_T0\_DQS\_EMCCLK\_14 Sch=sw[1]

set\_property -dict { PACKAGE\_PIN M13 IOSTANDARD LVCMOS33 } [get\_ports { SW1[2] }]; #IO\_L6N\_T0\_D08\_VREF\_14 Sch=sw[2]

set\_property -dict { PACKAGE\_PIN R15 IOSTANDARD LVCMOS33 } [get\_ports { SW1[3] }]; #IO\_L13N\_T2\_MRCC\_14 Sch=sw[3]

set\_property -dict { PACKAGE\_PIN R17 IOSTANDARD LVCMOS33 } [get\_ports { SW1[4] }]; #IO\_L12N\_T1\_MRCC\_14 Sch=sw[4]

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { udt }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { ent }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

#set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { ent }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

#set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { cinbit }]; #IO\_L24N\_T3\_34 Sch=sw[8]

#set\_property -dict { PACKAGE\_PIN U8 IOSTANDARD LVCMOS18 } [get\_ports { SW[9] }]; #IO\_25\_34 Sch=sw[9]

#set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { SW[10] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

#set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { SW[11] }]; #IO\_L23P\_T3\_A03\_D19\_14 Sch=sw[11]

#set\_property -dict { PACKAGE\_PIN H6 IOSTANDARD LVCMOS33 } [get\_ports { SW[12] }]; #IO\_L24P\_T3\_35 Sch=sw[12]

#set\_property -dict { PACKAGE\_PIN U12 IOSTANDARD LVCMOS33 } [get\_ports { SW[13] }]; #IO\_L20P\_T3\_A08\_D24\_14 Sch=sw[13]

#set\_property -dict { PACKAGE\_PIN U11 IOSTANDARD LVCMOS33 } [get\_ports { SW[14] }]; #IO\_L19N\_T3\_A09\_D25\_VREF\_14 Sch=sw[14]

#set\_property -dict { PACKAGE\_PIN V10 IOSTANDARD LVCMOS33 } [get\_ports { SW[15] }]; #IO\_L21P\_T3\_DQS\_14 Sch=sw[15]

## LEDs

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

#set\_property -dict { PACKAGE\_PIN K15 IOSTANDARD LVCMOS33 } [get\_ports { sumbit[1] }]; #IO\_L24P\_T3\_RS1\_15 Sch=led[1]

#set\_property -dict { PACKAGE\_PIN J13 IOSTANDARD LVCMOS33 } [get\_ports { sumbit[2] }]; #IO\_L17N\_T2\_A25\_15 Sch=led[2]

#set\_property -dict { PACKAGE\_PIN N14 IOSTANDARD LVCMOS33 } [get\_ports { sumbit[3] }]; #IO\_L8P\_T1\_D11\_14 Sch=led[3]

#set\_property -dict { PACKAGE\_PIN R18 IOSTANDARD LVCMOS33 } [get\_ports { coutbit }]; #IO\_L7P\_T1\_D09\_14 Sch=led[4]

#set\_property -dict { PACKAGE\_PIN V17 IOSTANDARD LVCMOS33 } [get\_ports { LED[5] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

#set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { LED[6] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

#set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { LED[7] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

#set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { LED[8] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

#set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { LED[9] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { LED[10] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

#set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { LED[11] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

#set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { LED[12] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

#set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { LED[13] }]; #IO\_L22N\_T3\_A04\_D20\_14 Sch=led[13]

#set\_property -dict { PACKAGE\_PIN V12 IOSTANDARD LVCMOS33 } [get\_ports { LED[14] }]; #IO\_L20N\_T3\_A07\_D23\_14 Sch=led[14]

#set\_property -dict { PACKAGE\_PIN V11 IOSTANDARD LVCMOS33 } [get\_ports { LED[15] }]; #IO\_L21N\_T3\_DQS\_A06\_D22\_14 Sch=led[15]

## RGB LEDs

#set\_property -dict { PACKAGE\_PIN R12 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_B }]; #IO\_L5P\_T0\_D06\_14 Sch=led16\_b

#set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_G }]; #IO\_L10P\_T1\_D14\_14 Sch=led16\_g

#set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { LED16\_R }]; #IO\_L11P\_T1\_SRCC\_14 Sch=led16\_r

#set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_B }]; #IO\_L15N\_T2\_DQS\_ADV\_B\_15 Sch=led17\_b

#set\_property -dict { PACKAGE\_PIN R11 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_G }]; #IO\_0\_14 Sch=led17\_g

#set\_property -dict { PACKAGE\_PIN N16 IOSTANDARD LVCMOS33 } [get\_ports { LED17\_R }]; #IO\_L11N\_T1\_SRCC\_14 Sch=led17\_r

##7 segment display

set\_property -dict { PACKAGE\_PIN T10 IOSTANDARD LVCMOS33 } [get\_ports { Cnode1[6] }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

set\_property -dict { PACKAGE\_PIN R10 IOSTANDARD LVCMOS33 } [get\_ports { Cnode1[5] }]; #IO\_25\_14 Sch=cb

set\_property -dict { PACKAGE\_PIN K16 IOSTANDARD LVCMOS33 } [get\_ports { Cnode1[4] }]; #IO\_25\_15 Sch=cc

set\_property -dict { PACKAGE\_PIN K13 IOSTANDARD LVCMOS33 } [get\_ports { Cnode1[3] }]; #IO\_L17P\_T2\_A26\_15 Sch=cd

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { Cnode1[2] }]; #IO\_L13P\_T2\_MRCC\_14 Sch=ce

set\_property -dict { PACKAGE\_PIN T11 IOSTANDARD LVCMOS33 } [get\_ports { Cnode1[1] }]; #IO\_L19P\_T3\_A10\_D26\_14 Sch=cf

set\_property -dict { PACKAGE\_PIN L18 IOSTANDARD LVCMOS33 } [get\_ports { Cnode1[0] }]; #IO\_L4P\_T0\_D04\_14 Sch=cg

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

set\_property -dict { PACKAGE\_PIN J17 IOSTANDARD LVCMOS33 } [get\_ports { AN1[0] }]; #IO\_L23P\_T3\_FOE\_B\_15 Sch=an[0]

set\_property -dict { PACKAGE\_PIN J18 IOSTANDARD LVCMOS33 } [get\_ports { AN1[1] }]; #IO\_L23N\_T3\_FWE\_B\_15 Sch=an[1]

set\_property -dict { PACKAGE\_PIN T9 IOSTANDARD LVCMOS33 } [get\_ports { AN1[2] }]; #IO\_L24P\_T3\_A01\_D17\_14 Sch=an[2]

set\_property -dict { PACKAGE\_PIN J14 IOSTANDARD LVCMOS33 } [get\_ports { AN1[3] }]; #IO\_L19P\_T3\_A22\_15 Sch=an[3]

set\_property -dict { PACKAGE\_PIN P14 IOSTANDARD LVCMOS33 } [get\_ports { AN1[4] }]; #IO\_L8N\_T1\_D12\_14 Sch=an[4]

set\_property -dict { PACKAGE\_PIN T14 IOSTANDARD LVCMOS33 } [get\_ports { AN1[5] }]; #IO\_L14P\_T2\_SRCC\_14 Sch=an[5]

set\_property -dict { PACKAGE\_PIN K2 IOSTANDARD LVCMOS33 } [get\_ports { AN1[6] }]; #IO\_L23P\_T3\_35 Sch=an[6]

set\_property -dict { PACKAGE\_PIN U13 IOSTANDARD LVCMOS33 } [get\_ports { AN1[7] }]; #IO\_L23N\_T3\_A02\_D18\_14 Sch=an[7]

##Buttons

#set\_property -dict { PACKAGE\_PIN C12 IOSTANDARD LVCMOS33 } [get\_ports { rst }]; #IO\_L3P\_T0\_DQS\_AD1P\_15 Sch=cpu\_resetn

set\_property -dict { PACKAGE\_PIN N17 IOSTANDARD LVCMOS33 } [get\_ports { rstt }]; #IO\_L9P\_T1\_DQS\_14 Sch=btnc

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

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

#set\_property -dict { PACKAGE\_PIN M17 IOSTANDARD LVCMOS33 } [get\_ports { BTNR }]; #IO\_L10N\_T1\_D15\_14 Sch=btnr

#set\_property -dict { PACKAGE\_PIN P18 IOSTANDARD LVCMOS33 } [get\_ports { BTND }]; #IO\_L9N\_T1\_DQS\_D13\_14 Sch=btnd

**Code Detail:**

`timescale 1ns / 1ps

`timescale 1ns / 1ps

module CLKMANAGER

(

input clk,

input rst,

input [4:0] SW,

output reg clkout =0

);

reg [31:0] sel;

always@(posedge clk or posedge rst)

begin: DREG

if(rst)

sel<= 32'd0;

else

sel<=sel+1;

end

always@(SW)

begin

case(SW)

5'd0: clkout <= sel[0];

5'd1: clkout <= sel[1];

5'd2: clkout <= sel[2];

5'd3: clkout <= sel[3];

5'd4: clkout <= sel[4];

5'd5: clkout <= sel[5];

5'd6: clkout <= sel[6];

5'd7: clkout <= sel[7];

5'd8: clkout <= sel[8];

5'd9: clkout <= sel[9];

5'd10:clkout <= sel[10];

5'd11:clkout <= sel[11];

5'd12:clkout <= sel[12];

5'd13:clkout <= sel[13];

5'd14:clkout <= sel[14];

5'd15:clkout <= sel[15];

5'd16:clkout <= sel[16];

5'd17:clkout <= sel[17];

5'd18:clkout <= sel[18];

5'd19:clkout <= sel[19];

5'd20:clkout <= sel[20];

5'd21:clkout <= sel[21];

5'd22:clkout <= sel[22];

5'd23:clkout <= sel[23];

5'd24:clkout <= sel[24];

5'd25:clkout <= sel[25];

5'd26:clkout <= sel[26];

5'd27:clkout <= sel[27];

5'd28:clkout <= sel[28];

5'd29:clkout <= sel[29];

5'd30:clkout <= sel[30];

5'd31:clkout <= sel[31];

endcase

end

endmodule

The purpose of the clock manager is to have a counter and a 5-bit mux to output a clk with various levels of frequency depending on the integer div in this case. First we set up the counter by using an always loop that accounts for the reset, and sets up the different frequencies for all 32 values. Then we created an always loop that takes effect when the switches are changed, and we used a case statement for all 32 values of the 5-bit switches. Depending on the clkout that is selected by the switches, the frequency could vary for each clkout. We used the non-blocking assignment <= so that the code doesn’t have to be done in order from top to bottom.

`timescale 1ns / 1ps

module CLKMANAGER\_TB (

);

reg clk\_tb,rst\_tb;

reg [4:0] SW\_tb;

wire clkout\_tb;

CLKMANAGER COMP(

.clk(clk\_tb),

.rst(rst\_tb),

.SW(SW\_tb),

.clkout(clkout\_tb)

);

initial

begin

clk\_tb = 1;

end

always

begin

#5 clk\_tb = ~clk\_tb;

//#20 SW\_tb = SW\_tb+1;

end

initial

begin

rst\_tb = 1;

SW\_tb <= 0;

#10

rst\_tb = 0;

#100

SW\_tb = 4;

#100

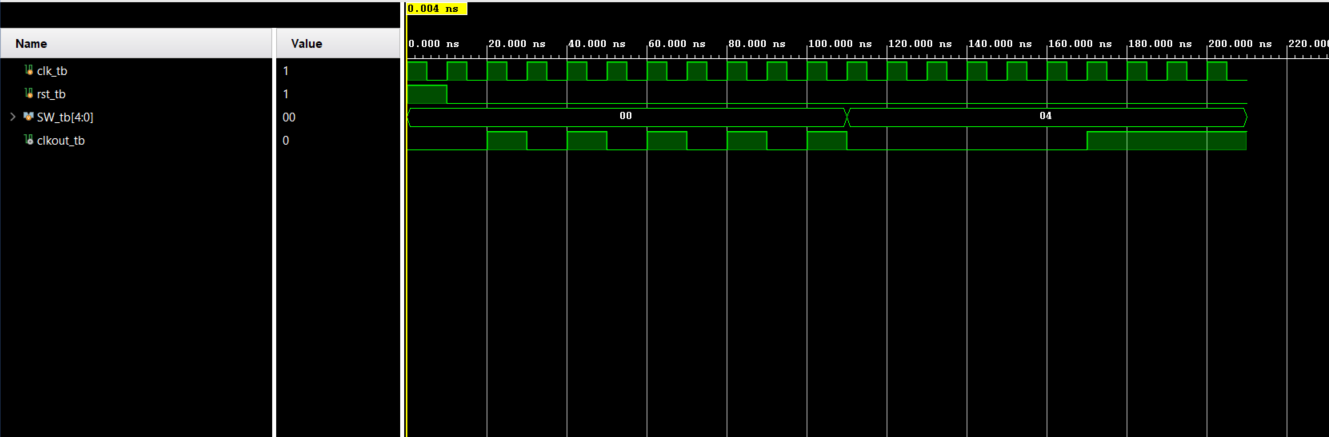
$finish;

end

endmodule

The purpose of this testbench is to test the code in the design source. We instantiate the variables of the design source code by calling that class,so that we can test them in this simulation source. We set the initial values for the inputs, and we use an always loop to make sure that the clock is toggling. We set a value for the switch to analyze the change in frequency.

EXAMPLE FOR CLKMANAGER:



`timescale 1ns / 1ps

module UPDOWN(

input clk1,

input rst1,

input en,

input ud,

output reg [3:0] tmp

);

always@(posedge clk1 or posedge rst1)

begin

if(en)

begin

if(rst1)

begin

if(ud)// 0 is up, 1 is down

tmp<= 4'd9;

else

tmp<= 4'd0;

end

else

begin

if(ud)

if(tmp==4'd0)

tmp <= 4'd9;

else

tmp <= tmp-1;

else

if(tmp==4'd9)

tmp <= 4'd0;

else

tmp <= tmp+1;

end

//tmp = tmp;

end

else

tmp=0;

end

endmodule

The purpose of this code is to count up or down in BCD. The always@(posedge or rst) was used because this is a sequential circuit , and the non-blocking assignment <= is used as well because this code runs in parallel. The if statement with en is used first because without the en the updown counter wouldn’t run. Next there is a if statement to see if the reset value is 1, if it is then we need to see whether the updown counter is 0 for up or 1 for down. This is because if we are counting from 9 to 0 then the value should reset to 9, whereas if we count from 0 to 9 it should reset to 0.

begin

always@(posedge clk1 or posedge rst1)

begin

if(en)

begin

if(rst1)

begin

if(ud)// 0 is up, 1 is down

tmp<= 4'd9;

else

tmp<= 4'd0;

end

else

begin

if(ud)

if(tmp==4'd0)

tmp <= 4'd9;

else

tmp <= tmp-1;

else

if(tmp==4'd9)

tmp <= 4'd0;

else

tmp <= tmp+1;

end

//tmp = tmp;

end

else

tmp=0;

end

The code above verifies the bcd because it checks whether the updown is 1 or 0, and based on what the maximum value for counting up and the minimum value for counting down is the tmp wire is reset to the beginning of the number counting scale. For example, the number counting scale is 0 to 9 for ud =0, so when it reaches 9 it will set back to 0 so that it doesn’t go to 10 and so on. The tmp is the wire that connects the updown counter results to the 4-bit output that goes to the 7seg.

`timescale 1ns / 1ps

module UPDOWN\_TB(

);

reg clk\_tb,rst\_tb;

reg en\_tb;

reg ud\_tb;

wire [3:0] cnt\_tb;

UPDOWN COMP (

.clk1(clk\_tb),

.rst1(rst\_tb),

.en(en\_tb),

.ud(ud\_tb),

.tmp(cnt\_tb)

);

initial

begin

clk\_tb = 0;

rst\_tb = 0;

ud\_tb = 0;

en\_tb = 1;

end

always

begin

#5 clk\_tb = ~clk\_tb;

end

always

begin

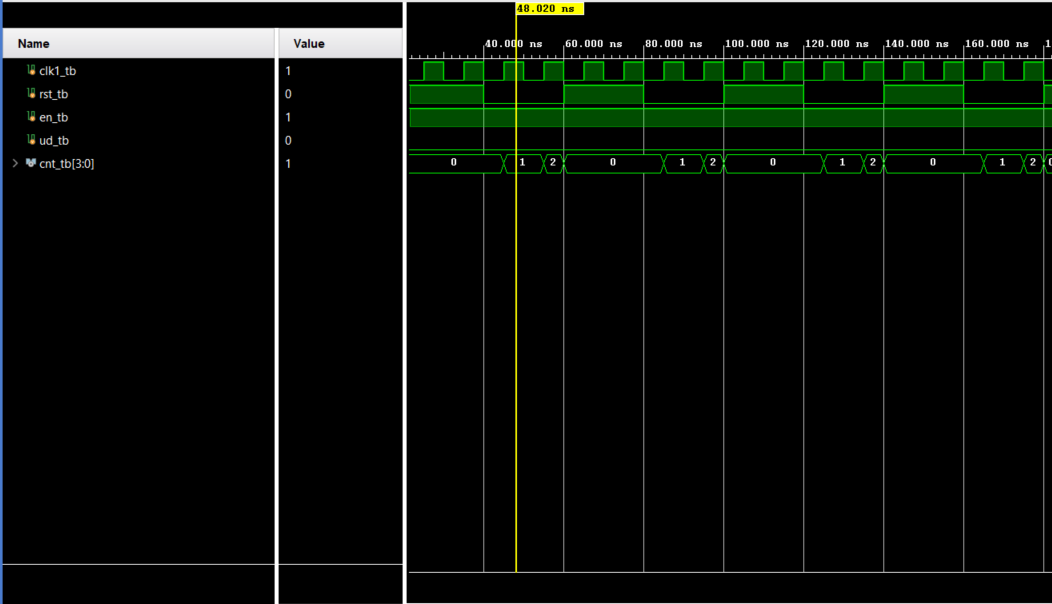
#20 rst\_tb = ~rst\_tb;

end

endmodule

The purpose of this testbench is to test the code in the design source. We instantiate the variables of the design source code by calling that class,so that we can test them in this simulation source. We set the initial values for the inputs, then we use an always loop to make sure that the clock is toggling, and I had an always loop for reset just to test it out.

EXAMPLE FOR UPDOWN:



`timescale 1ns / 1ps

module SEGDRIVE(

// input en,

//input [31:0] SW,

input [3:0] tmp\_SW,

output reg [6:0] Cnode,

output dp,

output wire [7:0] AN

);

//reg [31:0] temp;

//reg [3:0] tmp\_SW;

assign AN = 8'b11111110;

assign dp = 1'b1;

always@(tmp\_SW)

begin

case (tmp\_SW)

4'd0: Cnode <= 7'b0000001;

4'd1: Cnode <= 7'b1001111;

4'd2: Cnode <= 7'b0010010;

4'd3: Cnode <= 7'b0000110;

4'd4: Cnode <= 7'b1001100;

4'd5: Cnode <= 7'b0100100;

4'd6: Cnode <= 7'b0100000;

4'd7: Cnode <= 7'b0001111;

4'd8: Cnode <= 7'b0000000;

4'd9: Cnode <= 7'b0000100;

4'd10:Cnode <= 7'b0001000;

4'd11:Cnode <= 7'b1100000;

4'd12:Cnode <= 7'b0110001;

4'd13:Cnode <= 7'b1000010;

4'd14:Cnode <= 7'b0110000;

4'd15:Cnode <= 7'b0111000;

endcase

end

endmodule

`timescale 1ns / 1ps

module SEGDRIVE\_TB(

);

reg [3:0] tmp\_SW\_tb;

wire [6:0] Cnode\_tb;

wire dp\_tb;

wire [7:0] AN\_tb;

SEGDRIVE COMP (

.tmp\_SW(tmp\_SW\_tb),

.Cnode(Cnode\_tb),

.dp(dp\_tb),

.AN(AN\_tb)

);

initial

begin

//dp\_tb = 1;

tmp\_SW\_tb=0;

#10

tmp\_SW\_tb=4;

#1000

$finish;

end

endmodule

This code takes an input of 4 switches and depending on when the switches change value, one of the 16 possible outputs will be selected to display on the 7 segment display. AN and dp are assigned in the beginning to make sure it is only one digit without the decimal place being shown. The different cases range from 0 to f, and the 4-bit switch value selects the specific case. In this particular code 0 represents on, while 1 represents off.

`timescale 1ns / 1ps

module top(

input clkt,

input rstt,

input ent,

input [4:0] SW1,

input udt,

output seg,

output [6:0] Cnode1,

output [7:0] AN1

);

wire slowclk\_out;

CLKMANAGER FIN(

.clk(clkt),

//.rst(rstt),

.SW(SW1),

.clkout(slowclk\_out)

);

wire [3:0] tmp\_cnt;

UPDOWN FIND(

.rst1(rstt),

.en(ent),

.clk1(slowclk\_out),

.tmp(tmp\_cnt),

.ud(udt)

);

SEGDRIVE FINE(

.tmp\_SW(tmp\_cnt),

.Cnode(Cnode1),

.dp(seg),

.AN(AN1)

);

endmodule

The top file basically instantiates the clock, switches, and clock out of the CLKMANAGER. It instantiates the reset, enable, clock, updown, and 4-bit output of the UPDOWN, and the SEGDRIVE switches, Cnode, dp, and Anode. Two wires are created to connect everything together. The slowclk\_out wire connects the output clk of the clk manager to the input of the updown counter clk. The wire tmp\_cnt connects the 4-bit output of the updown to the input of the switches, so the values of 0 to 9 can be displayed on the FPGA board.

This code represents the following schematic of a clkmanager that generates 32 random frequencies and then sends it out as the clock of the updown which counts 0 to 9 or 9 to 0 at different frequencies. The 4-bit output represents the BCD value, which will be connected to the switches of the 7-segment display, so that it could display the BCD on the 7-segment display.

**Corner Cases/Error :**

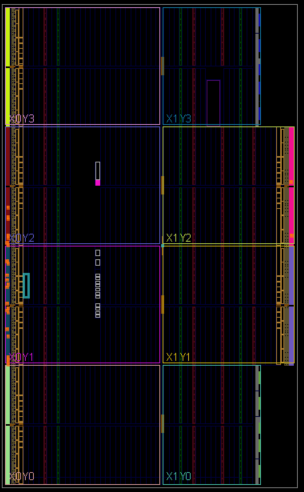
***One corner case can be the CLKMANAGER Testbench. At first the testbench wasn’t working because of :***

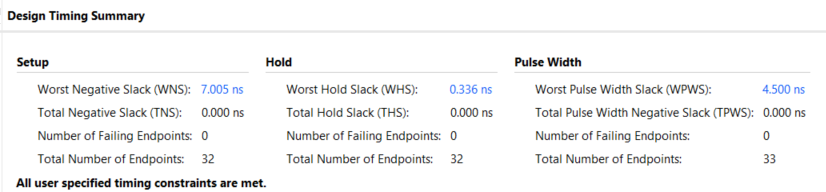
always@(SW)

begin

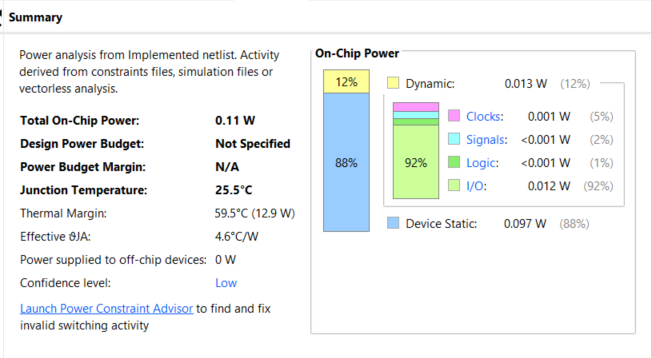
***The problem is that the loop can only run if the SW changes, so we had to change SW to posedge clk or posedge rst, so that the code could effectively work in the testbench.***

**IMPLEMENTED DESIGN/ TIMING SUMMARY:**

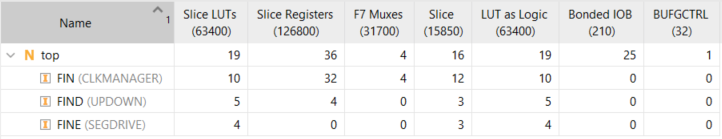
****

****

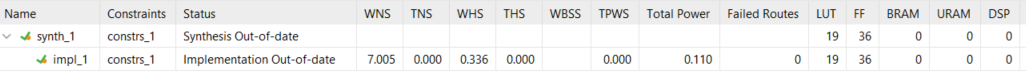
**POWER SUMMARY:**

****

**UTILIZATION:**

****

**RESOURCE USAGE:**

****