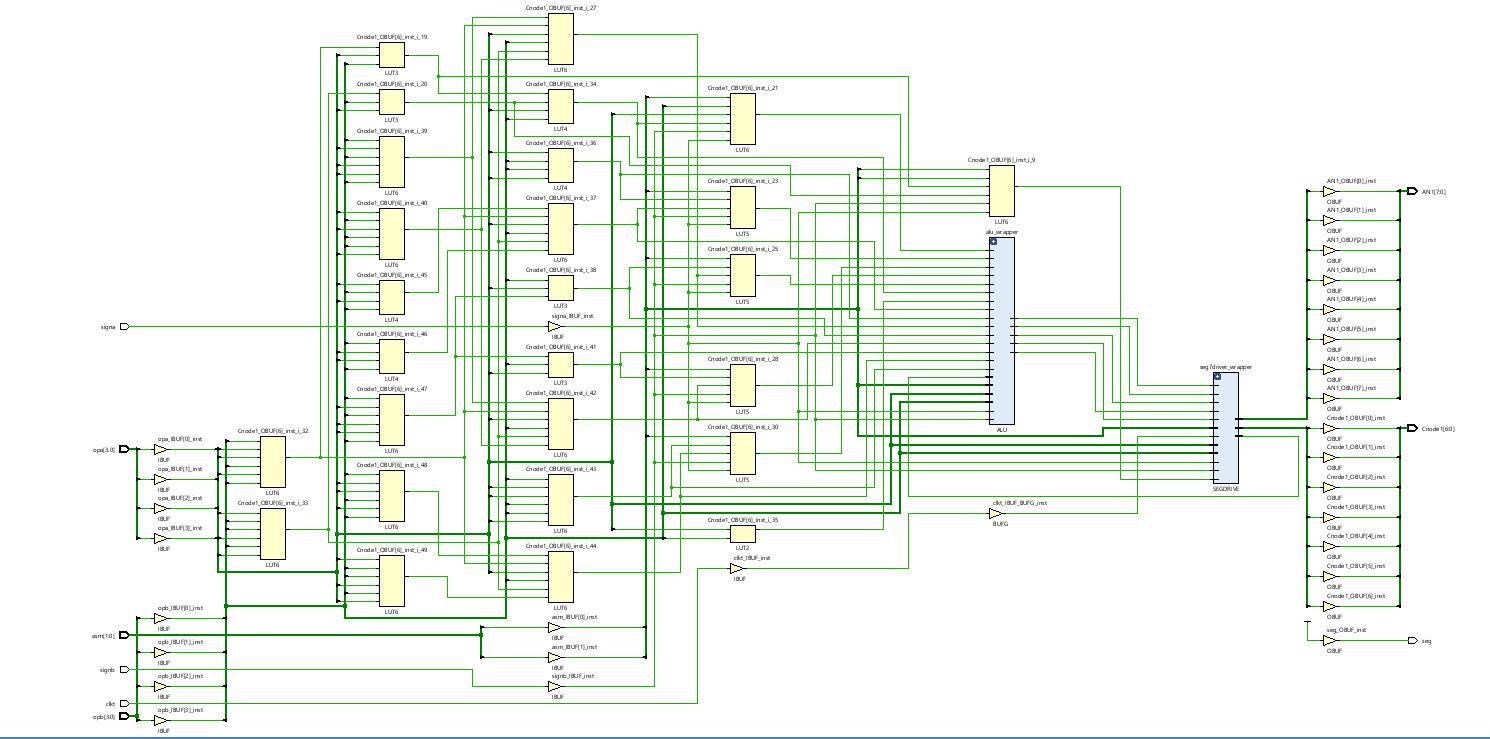
**ECE3300L Lab7 Group H Report (Sherwin Sathish & Mohamed Hamida)**

**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 { opb[0] }]; #IO\_L24N\_T3\_RS0\_15 Sch=sw[0]

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

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

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

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

set\_property -dict { PACKAGE\_PIN T18 IOSTANDARD LVCMOS33 } [get\_ports { opa[0] }]; #IO\_L7N\_T1\_D10\_14 Sch=sw[5]

set\_property -dict { PACKAGE\_PIN U18 IOSTANDARD LVCMOS33 } [get\_ports { opa[1] }]; #IO\_L17N\_T2\_A13\_D29\_14 Sch=sw[6]

set\_property -dict { PACKAGE\_PIN R13 IOSTANDARD LVCMOS33 } [get\_ports { opa[2] }]; #IO\_L5N\_T0\_D07\_14 Sch=sw[7]

set\_property -dict { PACKAGE\_PIN T8 IOSTANDARD LVCMOS18 } [get\_ports { opa[3] }]; #IO\_L24N\_T3\_34 Sch=sw[8]

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

set\_property -dict { PACKAGE\_PIN R16 IOSTANDARD LVCMOS33 } [get\_ports { asm[0] }]; #IO\_L15P\_T2\_DQS\_RDWR\_B\_14 Sch=sw[10]

set\_property -dict { PACKAGE\_PIN T13 IOSTANDARD LVCMOS33 } [get\_ports { asm[1] }]; #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 { opc[0] }]; #IO\_L18N\_T2\_A11\_D27\_14 Sch=led[5]

#set\_property -dict { PACKAGE\_PIN U17 IOSTANDARD LVCMOS33 } [get\_ports { opc[1] }]; #IO\_L17P\_T2\_A14\_D30\_14 Sch=led[6]

#set\_property -dict { PACKAGE\_PIN U16 IOSTANDARD LVCMOS33 } [get\_ports { opc[2] }]; #IO\_L18P\_T2\_A12\_D28\_14 Sch=led[7]

#set\_property -dict { PACKAGE\_PIN V16 IOSTANDARD LVCMOS33 } [get\_ports { opc[3] }]; #IO\_L16N\_T2\_A15\_D31\_14 Sch=led[8]

#set\_property -dict { PACKAGE\_PIN T15 IOSTANDARD LVCMOS33 } [get\_ports { opc[4] }]; #IO\_L14N\_T2\_SRCC\_14 Sch=led[9]

#set\_property -dict { PACKAGE\_PIN U14 IOSTANDARD LVCMOS33 } [get\_ports { opc[5] }]; #IO\_L22P\_T3\_A05\_D21\_14 Sch=led[10]

#set\_property -dict { PACKAGE\_PIN T16 IOSTANDARD LVCMOS33 } [get\_ports { opc[6] }]; #IO\_L15N\_T2\_DQS\_DOUT\_CSO\_B\_14 Sch=led[11]

#set\_property -dict { PACKAGE\_PIN V15 IOSTANDARD LVCMOS33 } [get\_ports { opc[7] }]; #IO\_L16P\_T2\_CSI\_B\_14 Sch=led[12]

#set\_property -dict { PACKAGE\_PIN V14 IOSTANDARD LVCMOS33 } [get\_ports { signc1 }]; #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]

**Code Detail:**

`timescale 1ns / 1ps

module ALU(

input [3:0] opa,

input [3:0] opb,

input signa,

input signb,

input [1:0] asm,

output [7:0] opc,

output signc1

);

reg [7:0] ropc=0;

reg signc=0;

always@(\*)

begin

case(asm)

2'b00:

begin

ropc <= 0;

signc <=0;

end

2'b01://ADD

begin

if(opa==0 && opb==0)

begin

ropc=0;

signc=0;

end

else if(signa==0 && signb==1)

begin

if(opa<opb)

begin

ropc <= opb-opa;

signc <=1;

end

else if(opa>opb)

begin

ropc <= opa-opb;

signc <=0;

end

else

begin

ropc <=0;

signc <=0;

end

end

else if(signa==1 && signb==0)

begin

if(opa<opb)

begin

ropc <= opb-opa;

signc <=0;

end

else if(opa>opb)

begin

ropc <= opa-opb;

signc <=1;

end

else

begin

ropc <=0;

signc <=0;

end

end

else if(signa==1 && signb==1)

begin

ropc <=opa+opb;

signc <=1;

end

else

begin

ropc <=opa+opb;

signc <=0;

end

end

2'b10://SUBTRACT

begin

if(opa==0 && opb==0)

begin

ropc=0;

signc=0;

end

else if(signa==0 && signb==0)

begin

if(opa<opb)

begin

ropc <= opb-opa;

signc <=1;

end

else if(opa>opb)

begin

ropc <= opa-opb;

signc <=0;

end

else

begin

ropc <=0;

signc <=0;

end

end

else if(signa==1 && signb==1)

begin

if(opa<opb)

begin

ropc <= opb-opa;

signc <=0;

end

else if(opa>opb)

begin

ropc <= opa-opb;

signc <=1;

end

else

begin

ropc <=0;

signc <=0;

end

end

else if(signa==1 && signb==0)

begin

ropc <=opa+opb;

signc <=1;

end

else

begin

ropc <=opa+opb;

signc <=0;

end

end

2'b11: //MULTIPICATION

begin

ropc = opa \* opb;

if(opa==0 && opb==0)

begin

ropc=0;

signc=0;

end

else if(signa==1 && signb==1)

signc <= 0;

else if(signa==0 && signb==0)

signc <= 0;

else

signc <= 1;

end

endcase

end

assign opc = ropc;

assign signc1 = signc;

endmodule

`timescale 1ns / 1ps

module alu\_tb();

reg [3:0] opatb;

reg [3:0] opbtb;

reg signatb;

reg signbtb;

reg [1:0] asmtb;

wire [7:0] opctb;

wire signctb;

ALU alu\_tb(

.opa(opatb),

.opb(opbtb),

.signa(signatb),

.signb(signbtb),

.asm(asmtb),

//output to 7seg

.opc(opctb),

.signc1(signctb)

);

initial

begin: TEST

asmtb = 3;

opatb = 4;

opbtb = 2;

signatb = 0;

signbtb = 0;

#100

asmtb = 2;

opatb = 4;

opbtb = 4;

signatb = 0;

signbtb = 1;

#100

asmtb = 1;

opatb = 9;

opbtb = 1;

signatb = 0;

signbtb = 0;

#100

asmtb = 3;

opatb = 1;

opbtb = 9;

signatb = 1;

signbtb = 0;

#1000

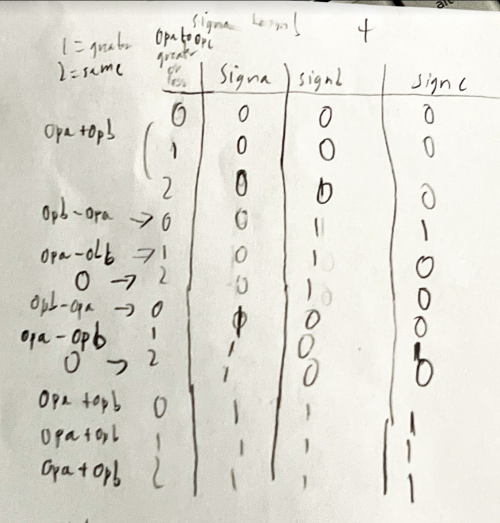
$finish;

end

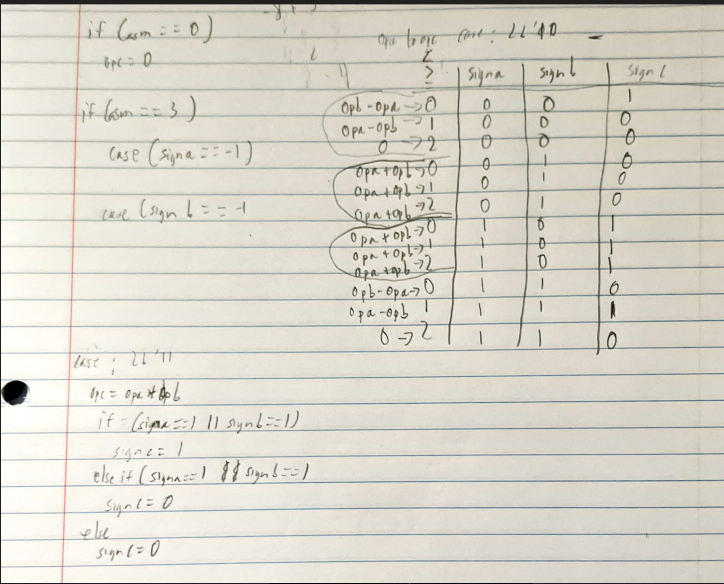
endmodule

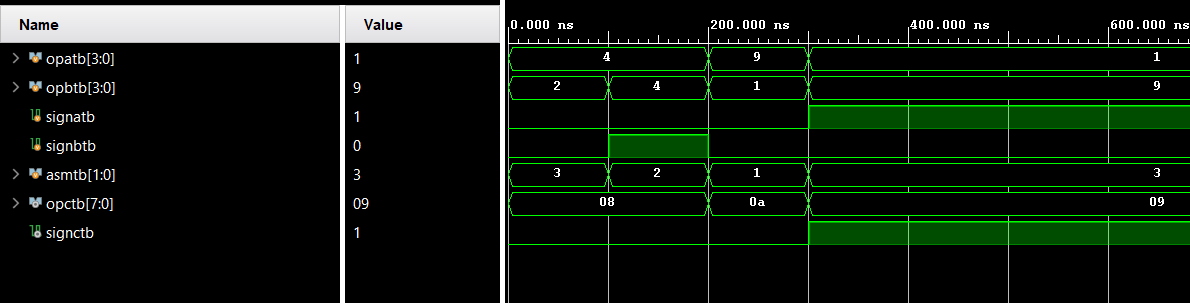
The alu code takes 2 single digit numbers and either adds, subtracts, or multiplies them to yield a 2 digit answer. To solve this we created truth tables to simplify the algorithm. We compared the signs first, then we set opa as the main digit, and created many cases based on opa compared to opb. At the end of the code we yield the opc and the signc values based on the different cases. The truth tables were used for addition and subtraction, but multiplication was simple because a size comparison between opa and opb wasn’t necessary. The truth tables and testbench are shown below:

**ADDITION:**



**SUBTRACT AND MULTIPLY:**





module bin2bcd(

input [7:0] bin,

output reg [7:0] bcd

);

integer i;

always @(bin) begin

bcd=0;

for (i=0;i<6;i=i+1) begin

// else

// begin

//Iterate once for each bit in input number

if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3; //If any BCD digit is >= 5, add three

if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;

bcd = {bcd[7:0],bin[5-i]};

//end

end //Shift one bit, and shift in proper bit from input

end

endmodule

`timescale 1ns / 1ps

module bin2bcd\_tb(

);

reg [31:0]bin\_tb;

wire [31:0] bcd\_tb;

bin2bcd COMP (

.bin(bin\_tb),

.bcd(bcd\_tb)

);

initial

begin

bin\_tb = 12;

//$finish

end

endmodule

The purpose of this code is to implement the double dabble algorithm. The “double dabble” algorithm is commonly used to convert a binary number to BCD. The binary number is left-shifted once for each of its bits, with bits shifted out of the MSB of the binary number and into the LSB of the accumulating BCD number. After every shift, all BCD digits are examined, and 3 is added to any BCD digit that is currently 5 or greater. This works because every left shift multiplies all BCD digits by two. Since BCD digits cannot exceed nine, a pre-shift number of five or more would result in a post-shift number of ten or more, which cannot be represented. Adding three to any BCD digit greater than five does two things: first, at the next shift, the 3 that was added becomes 6, and that accounts for the difference in binary and BCD codes. This code takes the opc output of the alu and converts it into BCD.

`timescale 1ns / 1ps

module segdisplaydriver(

input [31:0] inDigit,

output reg [6:0] Cnode,

output dp,

output reg [7:0] AN,

input nexysCLK, // 100MHz

output reg divided\_clk = 0 // 10kHz => 10ms period, 0.5ms ON, 0.5ms OFF

);

reg [3:0] singledigit = 0;

reg [3:0] refreshcounter = 0;

// Calculate division value = 100MHz / (2 \* desired frequency) - 1 => 10kHz => 4999

localparam div\_value = 25000;

integer counter\_value = 0;

always @(posedge nexysCLK)

begin

if (counter\_value == div\_value) // For every (div\_value) clock cycles, reset counter back to 0

counter\_value <= 0; // Use <= for parallel & same time, = for sequential - one after the other

else

counter\_value <= counter\_value + 1;

end

// divide clock

always @(posedge nexysCLK)

begin

if (counter\_value == div\_value)

divided\_clk <= ~divided\_clk; // Flip signal

else

divided\_clk <= divided\_clk; // Keep signal the same

end

/\*CLOCK DIVIDER CODE\*/

always @(posedge divided\_clk)

begin

refreshcounter <= refreshcounter + 1;

end

always @(refreshcounter)

begin

case(refreshcounter)

4'b0000: singledigit = inDigit[3:0]; // digit 1 value (right digit)

4'b0001: singledigit = inDigit[7:4]; // digit 2 value

4'b0010: singledigit = inDigit[11:8]; // digit 3 value

4'b0011: singledigit = inDigit[15:12]; // digit 4 value

4'b0101: singledigit = inDigit[19:16]; // digit 5 value missing 4’b0010

4'b0110: singledigit = inDigit[23:20]; // digit 6 value

4'b0111: singledigit = inDigit[27:24]; // digit 7 value

4'b1000: singledigit = inDigit[31:28]; // digit 8 value (left digit)

endcase

end

always @(refreshcounter)

begin

case(refreshcounter)

4'b0000: AN = 8'b11111110; // digit 1 ON (right digit)

4'b0001: AN = 8'b11111101; // digit 2 ON

4'b0010: AN = 8'b11111011; // digit 3 ON

4'b0011: AN = 8'b11110111; // digit 4 ON

4'b0100: AN = 8'b11101111; // digit 5 ON

4'b0101: AN = 8'b11011111; // digit 6 ON

4'b0110: AN = 8'b10111111; // digit 7 ON

4'b0111: AN = 8'b01111111; // digit 8 ON (left digit)

default: AN = 8'bZZZZZZZZ;

endcase

end

assign dp = 1'b1;

always@(singledigit)

begin

case (singledigit)

4'd0: Cnode<= 7'b0000001; //0

4'd1: Cnode<= 7'b1001111; //1

4'd2: Cnode<= 7'b0010010; //2

4'd3: Cnode<= 7'b0000110; //3

4'd4: Cnode<= 7'b1001100; //4

4'd5: Cnode<= 7'b0100100; //5

4'd6: Cnode<= 7'b0100000; //6

4'd7: Cnode<= 7'b0001111; //7

4'd8: Cnode<= 7'b0000000; //8

4'd9: Cnode<= 7'b0000100; //9

4'd10:Cnode<= 7'b0001000; //A

4'd11:Cnode<= 7'b1100000; //B

4'd12:Cnode<= 7'b0110001; //C

4'd13:Cnode<= 7'b1000010; //D

4'd14:Cnode<= 7'b1011000; //POSITIVE AKA 1110

4'd15:Cnode<= 7'b0010100; //NEGATIVE AKA 1111

default: Cnode = 7'b0000000; //DEFAULT CASE EVERYTHING ON

endcase

end

endmodule

`timescale 1ns / 1ps

module SEGDRIVE\_TB(

);

reg [31:0] 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 32 switches and depending on when the switches change value, one of the many 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 9, and the 32-bit switch value selects the specific digit that needs to be modified. In this particular code 0 represents on, while 1 represents off. The values of E and F are set to display the positive and negative “sign” that is desired from the alu.

`timescale 1ns / 1ps

module top(

//INPUTS FROM FPGA

input [3:0] opa,

input [3:0] opb,

input signa,

input signb,

input [1:0] asm,

//OUTPUT TO 7SEG DISPLAY

// output [7:0] opc,

//output signc1,

//7SEG DISPLAY INPUTS

input clkt,

output [6:0] Cnode1,

output [7:0] AN1,

output seg

);

wire [31:0] seg7Digit;

//Send input changes to 7seg display

assign seg7Digit[3:0] = opb;

assign seg7Digit[7:4] = signb + 14;

assign seg7Digit[11:8] = opa;

assign seg7Digit[15:12] = signa + 14;

assign seg7Digit[27:24] = signc1 + 14;

assign seg7Digit[31:28] = asm;

wire [7:0] opc;

ALU alu\_wrapper(

.opa(opa),

.opb(opb),

.signa(signa),

.signb(signb),

.asm(asm),

.opc(opc),

.signc1(signc1)

);

bin2bcd b2bwrap(

.bin(opc),

.bcd(seg7Digit[23:16])

);

SEGDRIVE seg7driver\_wrapper(

.nexysCLK(clkt),

.inDigit(seg7Digit),

.Cnode(Cnode1),

.dp(seg),

.AN(AN1)

);

endmodule

`timescale 1ns / 1ps

module top\_tb();

reg [3:0] opatb;

reg [3:0] opbtb;

reg signatb;

reg signbtb;

reg [1:0] asmtb;

wire [6:0] Cnode\_tb;

wire dp\_tb;

wire [7:0] AN\_tb;

top top\_t(

.opa(opatb),

.opb(opbtb),

.signa(signatb),

.signb(signbtb),

.asm(asmtb),

//output to 7seg

.Cnode1(Cnode\_tb),

.seg(dp\_tb),

.AN1(AN\_tb)

);

initial

begin: TEST

asmtb = 3;

opatb = 4;

opbtb = 2;

signatb = 0;

signbtb = 0;

#100

asmtb = 2;

opatb = 4;

opbtb = 4;

signatb = 0;

signbtb = 1;

#100

asmtb = 1;

opatb = 9;

opbtb = 1;

signatb = 0;

signbtb = 0;

#100

asmtb = 3;

opatb = 1;

opbtb = 9;

signatb = 1;

signbtb = 0;

#1000

$finish;

end

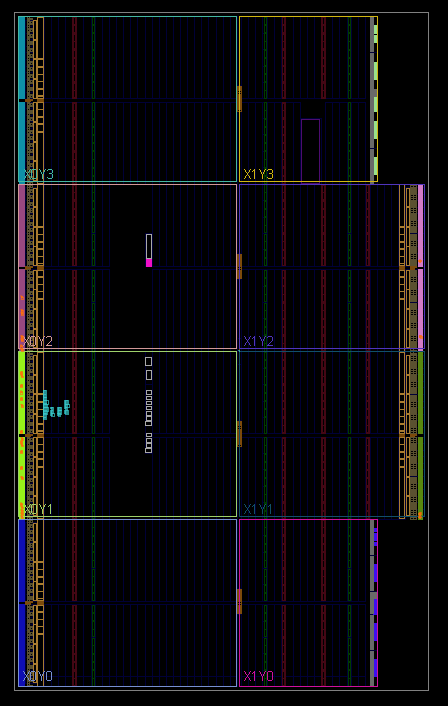
endmodule

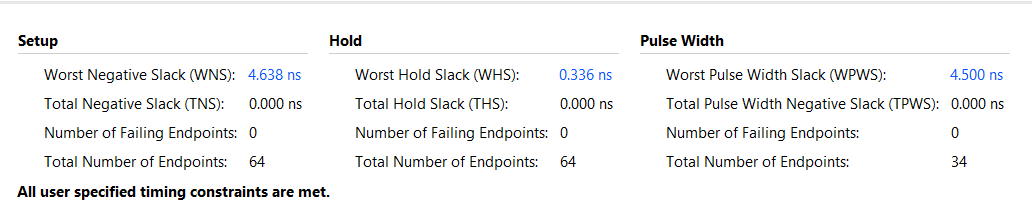
The top file instantiates the ALU input, 8-bit output of the BCD converter from the ALU output, and the SEGDRIVE input, Cnode, dp, and Anode. The wires are used to establish connections between the ALU output to the binary to bcd converter, and also establish the input and output connections to the SEGDRIVE. The assign is used to give specific digits on the FPGA the desired values, whether it’s an actual number or sign.

**Corner Cases/Error :**

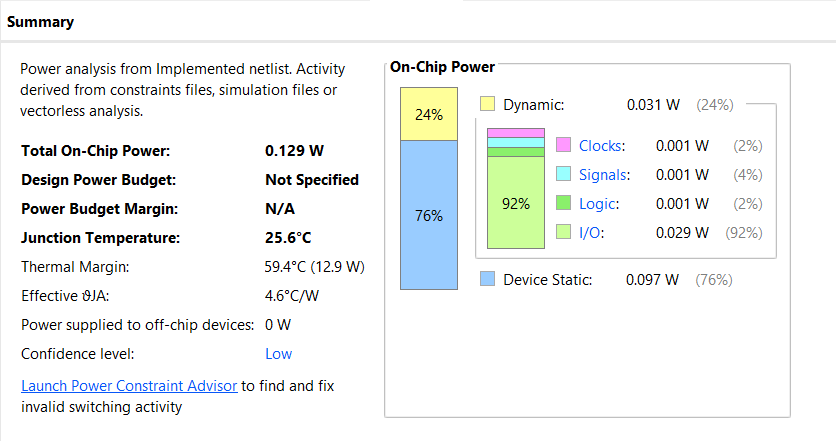
***One corner case can be A-D can be displayed, to cope this either the default case in the case statements should be 9, or the user simply just shouldn’t input those values.***

**IMPLEMENTED DESIGN/ TIMING SUMMARY:**

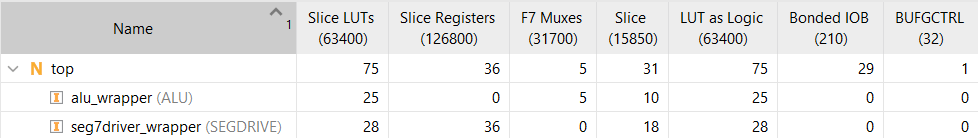
****

****

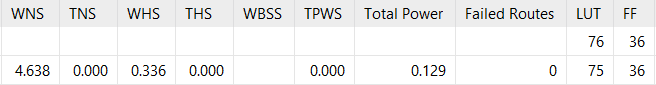
**POWER SUMMARY:**

****

**UTILIZATION:**

****

**RESOURCE USAGE:**

****