

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机组成 |
| 姓 名： | TANG ANNA YONGQI |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术（中加班）留学生 |
| 学 号： | 3180300155 |
| 生活照： |  |
| 指导教师： | 刘海风，洪奇军 |

2020年 3月 6日

Lab 2 – 7-Segment Display Module

**Name:** Anna Yongqi Tang **ID:** 3180300155 **Major:** 计算机科学与技术（中加班）留学生

**Course:** Computer Organization

**Date:** 2020-03-06 **Instructor:** 洪奇军

1. **Method and Experimental Steps**

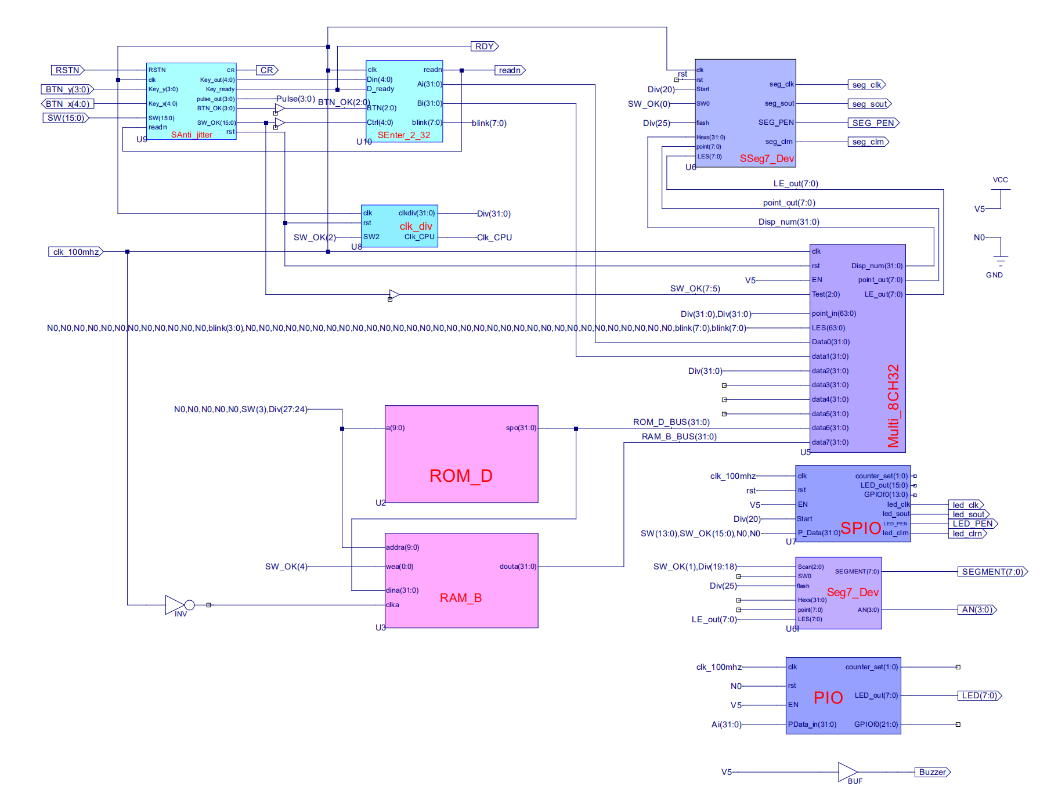


Figure 1 - top.sch

This depicts the completion of lab 2. Lab 2 was a reiteration of lab 1, but with improvements to the SSeg7\_Dev module, Seg7\_Dev module and the SPIO module. The ucf for this program came from the courseware and is linked to the top module of this project. Synthesis had minimal warnings, and implementation was successful. A programmable file has been generated and is ready for testing.

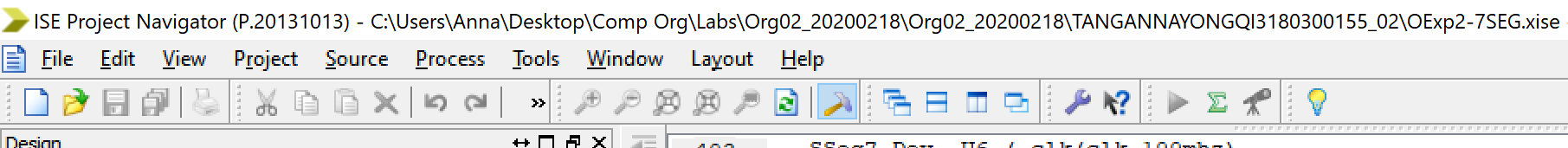


Figure - TANGANNAYONGQI3180300155\_02

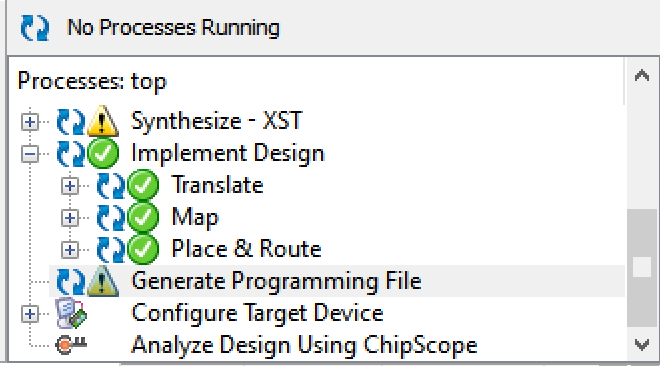
 

Figure 3 - .bit file generation Figure 4 - .bit file found in direction

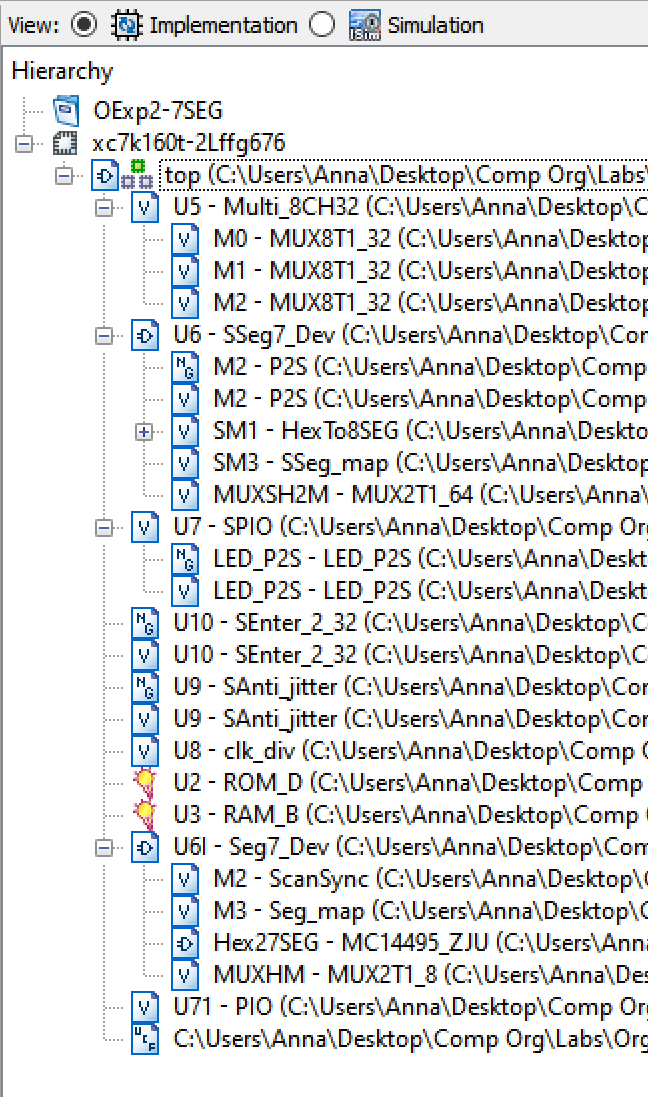
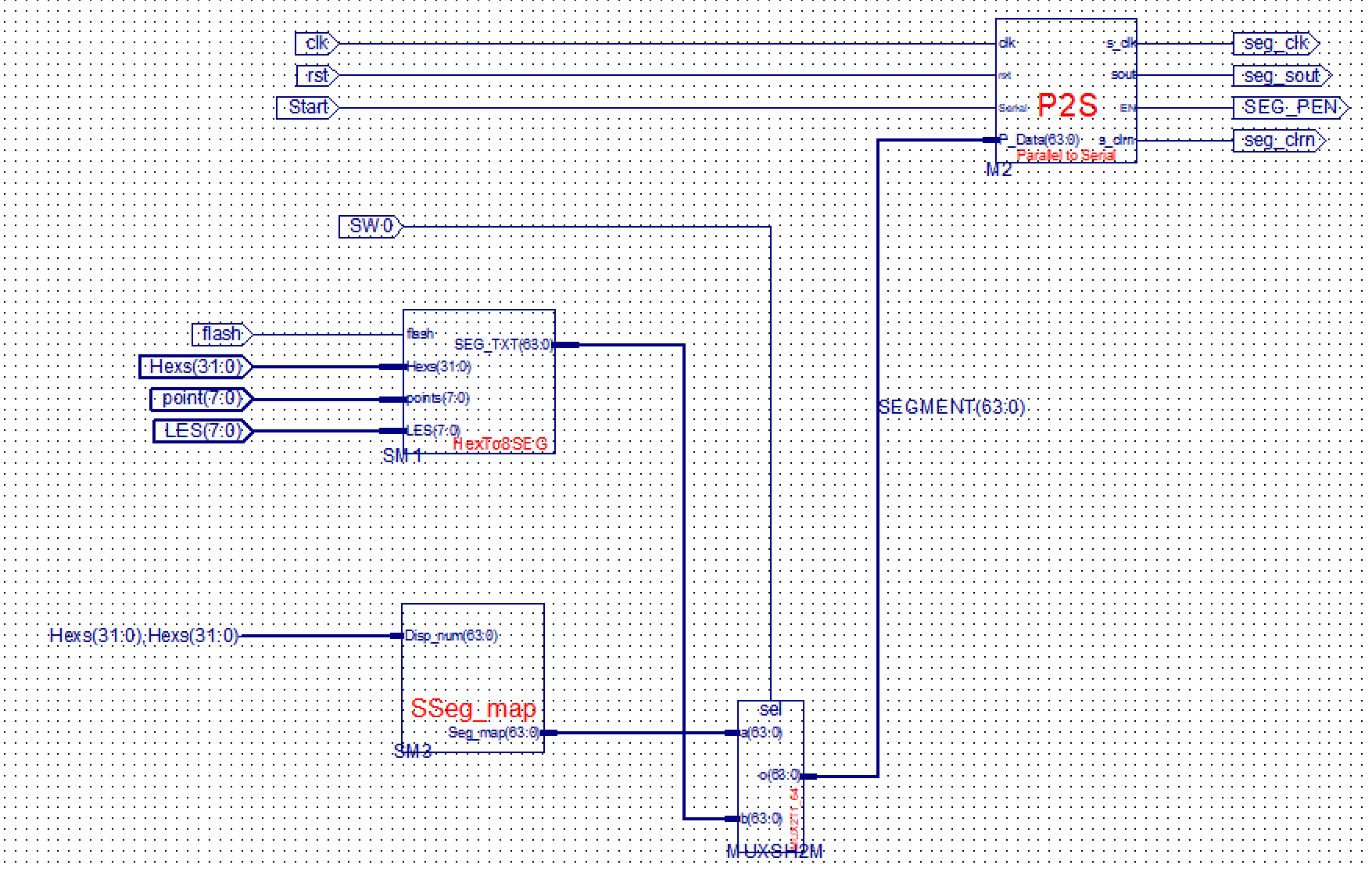


Figure 5 - file hierarchy

Most of the modules and the .ucf file that were used for this lab came from lab 1. Source code for the modules that have not been updated can be found in the lab report for lab 1. Modules that have been changed will be included in this report. Code for submodules can be found at the end of the report.

*SSeg7\_Dev (U6)*



*SPIO Code (U7)*

module SPIO(input clk,

input rst,

input Start,

input EN,

input [31:0] P\_Data,

output reg[1:0] counter\_set,

output [15:0] LED\_out,

output wire led\_clk,

output wire led\_sout,

output wire led\_clrn,

output wire LED\_PEN,

output reg[13:0] GPIOf0

);

reg [15:0] LED;

assign LED\_out = LED;

always @(negedge clk or posedge rst) begin

if (rst) begin

LED <= 8'h2A;

counter\_set <= 2'b00;

end else begin

if (EN) begin

{GPIOf0[13:0], LED, counter\_set} <= P\_Data;

end else begin

LED <= LED;

counter\_set <= counter\_set;

end

end

end

LED\_P2S LED\_P2S(clk, rst, Start,

{~{LED[0], LED[1], LED[2], LED[3], LED[4], LED[5], LED[6], LED[7],

LED[8], LED[9], LED[10], LED[11], LED[12], LED[13], LED[14], LED[15]}},

led\_clk,

led\_clrn,

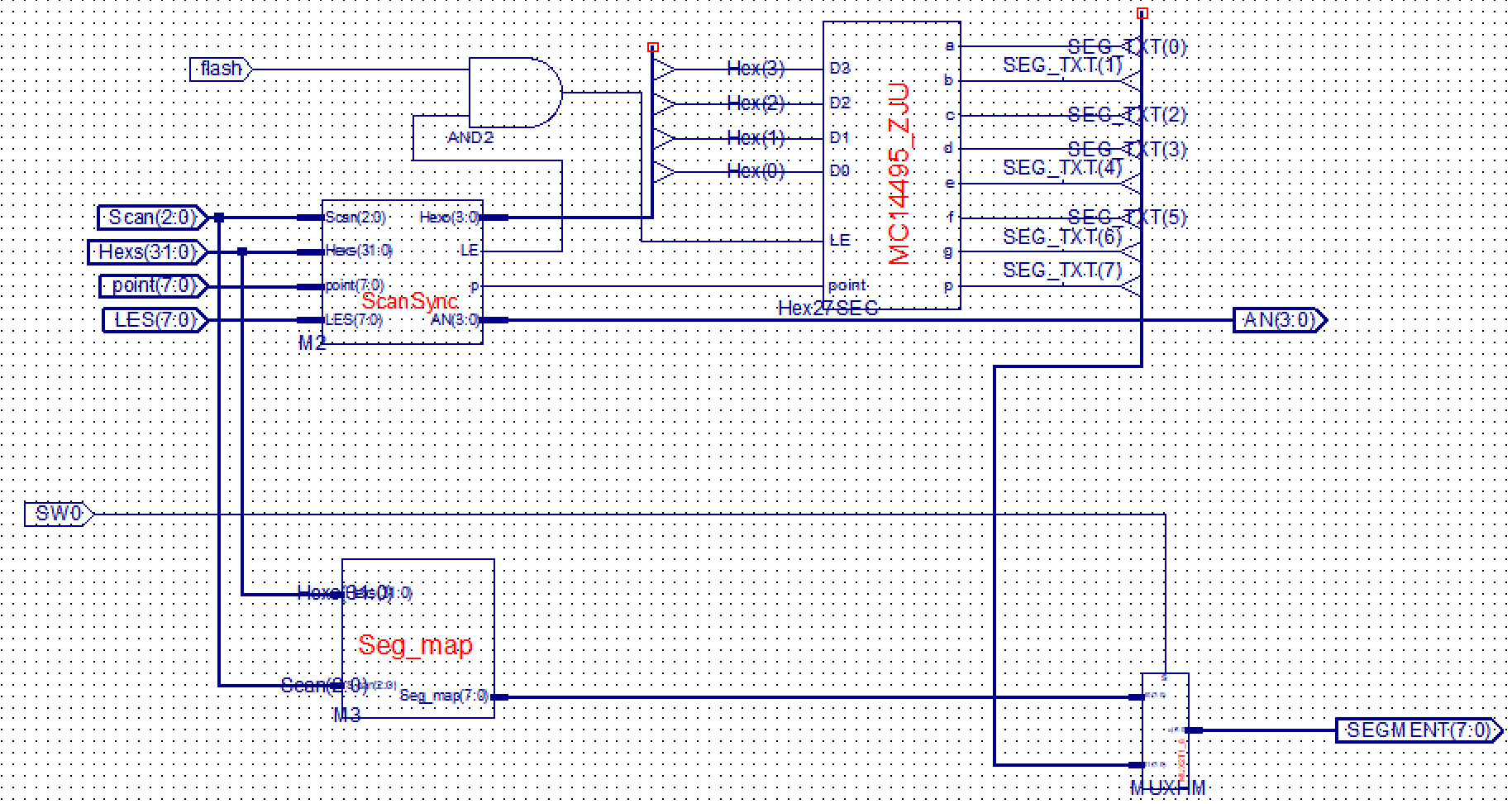
led\_sout,

LED\_PEN

);

endmodule

*Seg7\_Dev Code (U6l)*



*PIO Code (U7)*

module PIO(input wire clk,

input wire rst,

input wire EN,

input wire[31:0] PData\_in,

output reg[1:0] counter\_set,

output[7:0] LED\_out,

output reg[21:0]GPIOf0

);

reg [7:0] LED;

assign LED\_out = LED;

always @(negedge clk or posedge rst) begin

if (rst) begin

LED <= 8'h2A;

counter\_set <= 2'b0;

end else begin

if (EN) begin

{GPIOf0, LED, counter\_set} <= PData\_in;

end else begin

LED <= LED;

counter\_set <= counter\_set;

end

end

end

endmodule

1. **Simulations and Observations**

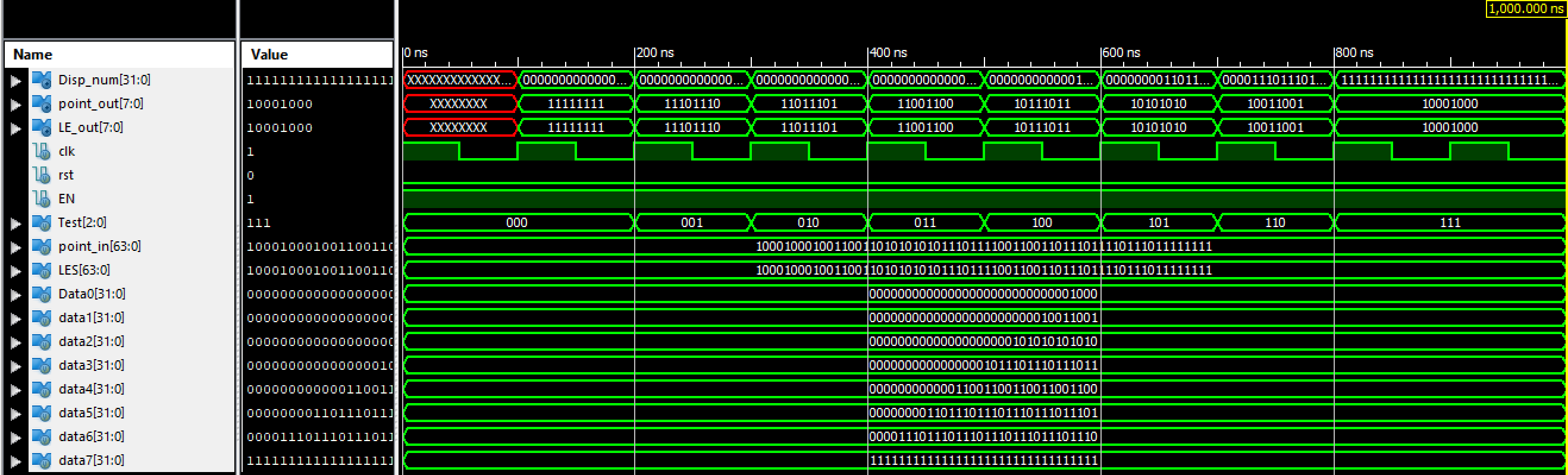


Figure 6 - Multi\_CH32 module

I used the same simulation file from lab 1 to simulate this module. As expected, the results are the same and the simulation passed.

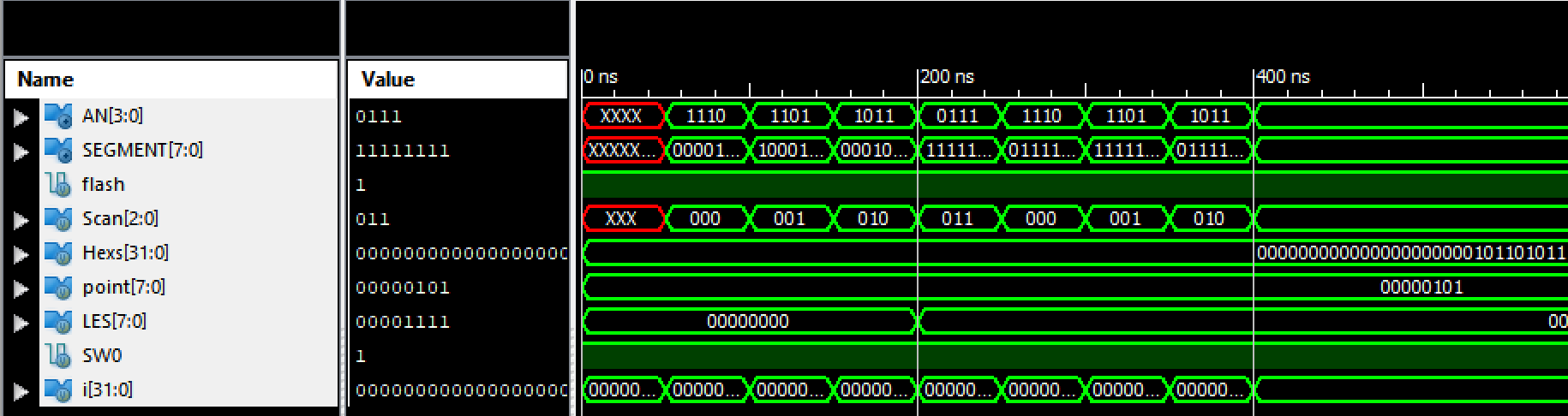


Figure 7 - Seg7\_Dev module

The above picture is a simulation of the enhanced seven-segment display module (Seg7\_Dev). It is not consistent with the sample provided in the PowerPoint, but it gives the expected results from the programmed parameters.

*Verilog Test Module for Simulation*

module Seg7\_Dev\_Seg7\_Dev\_sch\_tb();

// Inputs

reg flash;

reg [2:0] Scan;

reg [31:0] Hexs;

reg [7:0] point;

reg [7:0] LES;

reg SW0;

// Output

wire [3:0] AN;

wire [7:0] SEGMENT;

// Bidirs

// Instantiate the UUT

Seg7\_Dev UUT (

.flash(flash),

.Scan(Scan),

.Hexs(Hexs),

.point(point),

.LES(LES),

.AN(AN),

.SEGMENT(SEGMENT),

.SW0(SW0)

);

// Initialize Inputs

`ifdef auto\_init

initial begin

Scan = 0;

Hexs = 0;

point = 0;

LES = 0;

flash = 0;

SW0 = 0;

`endif

integer i;

initial begin

Hexs = 16'h05AF;

point = 4'b0101;

LES = 4'b0000;

SW0 = 1;

flash = 1;

for(i = 0; i < 4; i = i + 1) begin

#50;

Scan = i;

end

LES = 4'b1111;

for(i = 0; i < 4; i = i + 1) begin

#50;

Scan = i;

end

end

endmodule

1. **Conclusion**

This week’s lab was more confusing than hard, because I struggled to understand the purpose of changing the module’s source files to improve functionality. The information provided in the slides were not helpful and caused much confusion. I still struggle to understand the use of a .ngc file, and what purpose they provide when implemented in a program. Developing a simulation file for the seven-segment display was also not easy, because I did not quite understand what each input was and what role they played. I overcame this difficulty by refreshing what I learned in my Logic course and by playing around with the input parameters. Overall, this was a straightforward lab and I look forward to implementing this onto the SWORD board.

1. **Source Code**

*SSeg7\_Dev Code (U6) – P2S (M2)*

module P2S(input wire clk, //parallel to serial

input wire rst,

input wire Serial,

input wire[DATA\_BITS-1:0] P\_Data,

output reg s\_clk,

output wire s\_clrn,

output wire sout,

output reg EN

);

DATA\_BITS = 64,

DATA\_COUNT\_BITS = 6;

endmodule

*SSeg7\_Dev Code (U6) – Hex28SEG (SM1)*

module HexTo8SEG(input [31:0] Hexs,

//input [2:0] Scan,

input [7:0] points,

input [7:0] LES,

input flash,

output[63:0] SEG\_TXT

);

Hex2Seg M0(.Hex(Hexs[3:0]),

.LE(LES[0] & flash),

.point(points[0]),

.flash(flash),

.Segment(SEG\_TXT[63:56])

);

Hex2Seg M1(.Hex(Hexs[7:4]),

.LE(LES[1] & flash),

.point(points[1]),

.flash(flash),

.Segment(SEG\_TXT[55:48])

);

Hex2Seg M2(.Hex(Hexs[11:8]),

.LE(LES[2] & flash),

.point(points[2]),

.flash(flash),

.Segment(SEG\_TXT[47:40])

);

Hex2Seg M3(.Hex(Hexs[15:12]),

.LE(LES[3] & flash),

.point(points[3]),

.flash(flash),

.Segment(SEG\_TXT[39:32])

);

Hex2Seg M4(.Hex(Hexs[19:16]),

.LE(LES[4] & flash),

.point(points[4]),

.flash(flash),

.Segment(SEG\_TXT[31:24])

);

Hex2Seg M5(.Hex(Hexs[23:20]),

.LE(LES[5] & flash),

.point(points[5]),

.flash(flash),

.Segment(SEG\_TXT[23:16])

);

Hex2Seg M6(.Hex(Hexs[27:24]),

.LE(LES[6] & flash),

.point(points[6]),

.flash(flash),

.Segment(SEG\_TXT[15:8])

);

Hex2Seg M7(.Hex(Hexs[31:28]),

.LE(LES[7] & flash),

.point(points[7]),

.flash(flash),

.Segment(SEG\_TXT[7:0])

);

endmodule

module Hex2Seg(input[3:0]Hex,

input LE,

input point,

input flash,

output[7:0]Segment

);

MC14495\_ZJU MSEG(.D0(Hex[0]),

.D1(Hex[1]),

.D2(Hex[2]),

.D3(Hex[3]),

.LE(LE & flash),

.point(point),

.a(Segment[7]),

.b(Segment[6]),

.c(Segment[5]),

.d(Segment[4]),

.e(Segment[3]),

.f(Segment[2]),

.g(Segment[1]),

.p(Segment[0])

);

//assign Segment = {a,b,c,d,e,f,g,p}; //p,g,f,e,d,c,b,a

endmodule

*SSeg7\_Dev Code (U6) – SSeg\_map (SM3)*

module SSeg\_map(input[63:0]Disp\_num,

output[63:0]Seg\_map

);

assign Seg\_map = {Disp\_num[0], Disp\_num[4], Disp\_num[16], Disp\_num[25], Disp\_num[17], Disp\_num[5], Disp\_num[12], Disp\_num[24],Disp\_num[1], Disp\_num[6], Disp\_num[18], Disp\_num[27], Disp\_num[19], Disp\_num[7], Disp\_num[13], Disp\_num[26],Disp\_num[2], Disp\_num[8], Disp\_num[20], Disp\_num[29], Disp\_num[21], Disp\_num[9], Disp\_num[14], Disp\_num[28],Disp\_num[3], Disp\_num[10], Disp\_num[22], Disp\_num[31], Disp\_num[23], Disp\_num[11], Disp\_num[15], Disp\_num[30], Disp\_num[0], Disp\_num[4], Disp\_num[16], Disp\_num[25], Disp\_num[17], Disp\_num[5], Disp\_num[12], Disp\_num[24],Disp \_num[1], Disp\_num[6], Disp\_num[18], Disp\_num[27], Disp\_num[19], Disp\_num[7], Disp\_num[13], Disp\_num[26], Disp\_num[2], Disp\_num[8], Disp\_num[20], Disp\_num[29], Disp\_num[21], Disp\_num[9], Disp\_num[14], Disp\_num[28], Disp\_num[3], Disp\_num[10], Disp\_num[22], Disp\_num[31], Disp\_num[23], Disp\_num[11], Disp\_num[15], Disp\_num[30]};

endmodule

*SSeg7\_Dev Code (U6) – MUX2T1\_64 (MUXSH2M)*

module MUX2T1\_64(input[63:0]a,

input[63:0]b,

input sel,

output[63:0]o

);

assign o = sel? a : b;

endmodule

*SPIO Code (U7) – LED\_P2S*

Same as SSeg7\_Dev Code (U6) – P2S (M2).

*Seg7\_Dev Code (U6l) – ScanSync(M2)*

module ScanSync(Hexs,Scan,point,LES,Hexo,p,LE,AN);

input [31:0] Hexs;

input [2:0] Scan;

input [7:0] point;

input [7:0] LES;

output reg [3:0] Hexo;

output reg p,LE;

output reg [3:0] AN;

always@\* begin

case(Scan)

3'b000:begin Hexo<=Hexs[3:0]; AN<=4'b1110; p<=point[0]; LE<=LES[0];end

3'b001:begin Hexo<=Hexs[7:4]; AN<=8'b1101; p<=point[1]; LE<=LES[1];end

3'b010:begin Hexo<=Hexs[11:8]; AN<=8'b1011; p<=point[2]; LE<=LES[2];end

3'b011:begin Hexo<=Hexs[15:12]; AN<=8'b0111; p<=point[3]; LE<=LES[3];end

3'b100:begin Hexo<=Hexs[3:0]; AN<=4'b1110; p<=point[0]; LE<=LES[0];end

3'b101:begin Hexo<=Hexs[7:4]; AN<=8'b1101; p<=point[1]; LE<=LES[1];end

3'b110:begin Hexo<=Hexs[11:8]; AN<=8'b1011; p<=point[2]; LE<=LES[2];end

3'b111:begin Hexo<=Hexs[15:12]; AN<=8'b0111; p<=point[3]; LE<=LES[3];end

endcase

end

endmodule

*Seg7\_Dev Code (U6l) – SegMap(M3)*

module Seg\_map(Hexs,Scan,Seg\_map);

input [31:0] Hexs;

input [2:0] Scan;

output reg[7:0] Seg\_map;

always@\* begin

case(Scan)

3'b000:Seg\_map={Hexs[24],Hexs[12],Hexs[5],Hexs[17],Hexs[25],Hexs[16],Hexs[4],Hexs[0]};

endcase

end

endmodule

*Seg7\_Dev Code (U6l) – SegMap(M3)*

module MUX2T1\_8(input[7:0]I0,

input[7:0]I1,

input s,

output[7:0]o

);

assign o = s ? I1 : I0;

endmodule

*Testing Constraints File (UCF)*

Please refer to the UCF that was provided in the courseware.