

**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 1日

Lab 1- Design of Multiplexer and Application of SWORD Board Displays

**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:** 2020-03-01 **Instructor:** 洪奇军

1. **Method and Experimental Steps**

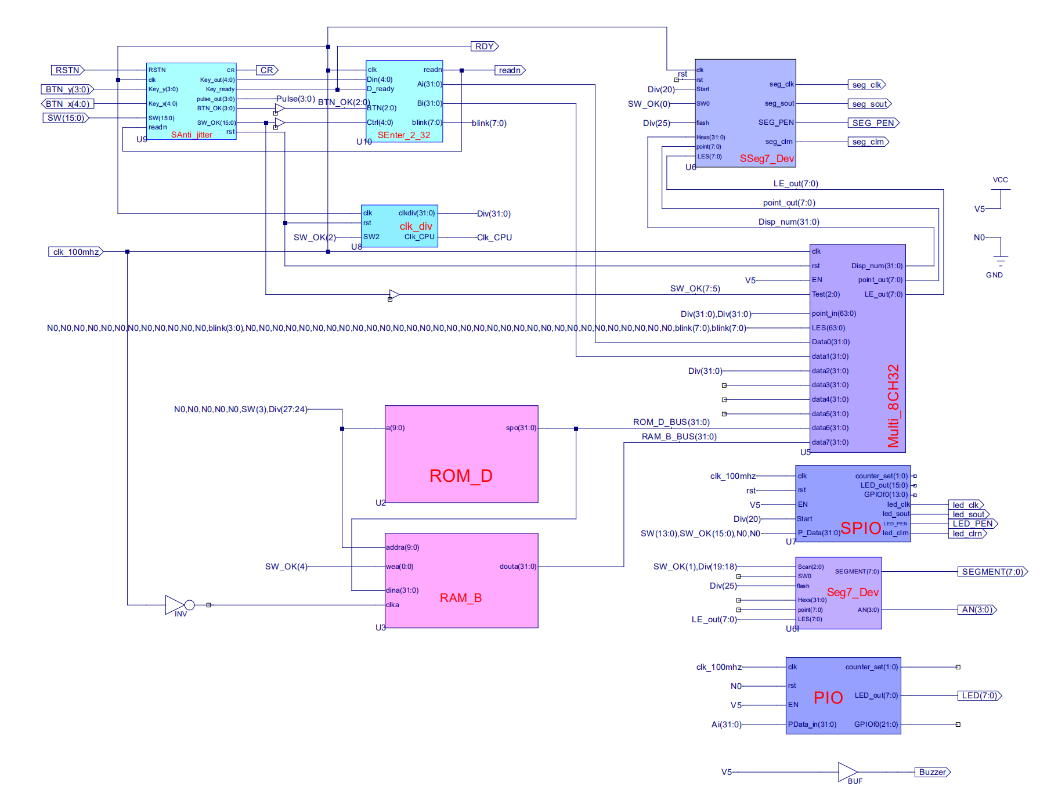


Figure 1 - OExp01\_MUX.sch

This depicts the completion of lab 1. All modules in the schematic above were sourced from course material, except for ROM\_D and RAM\_B. These two components were generated by the IP CORE generator, and settings referenced from the slides were used. The ucf provided in the courseware was used and is linked to the top module of this project. There were minimal warning messages with the synthesis process, but implementation was successful with zero errors and zero warnings. A programmable file has been generated and is ready for testing on the SWORD board.

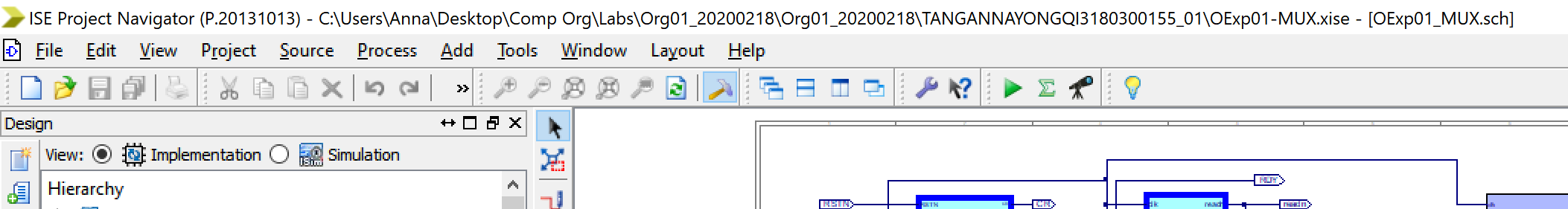


Figure 2 - TANGANNAYONGQI3180300155\_01 Folder Name

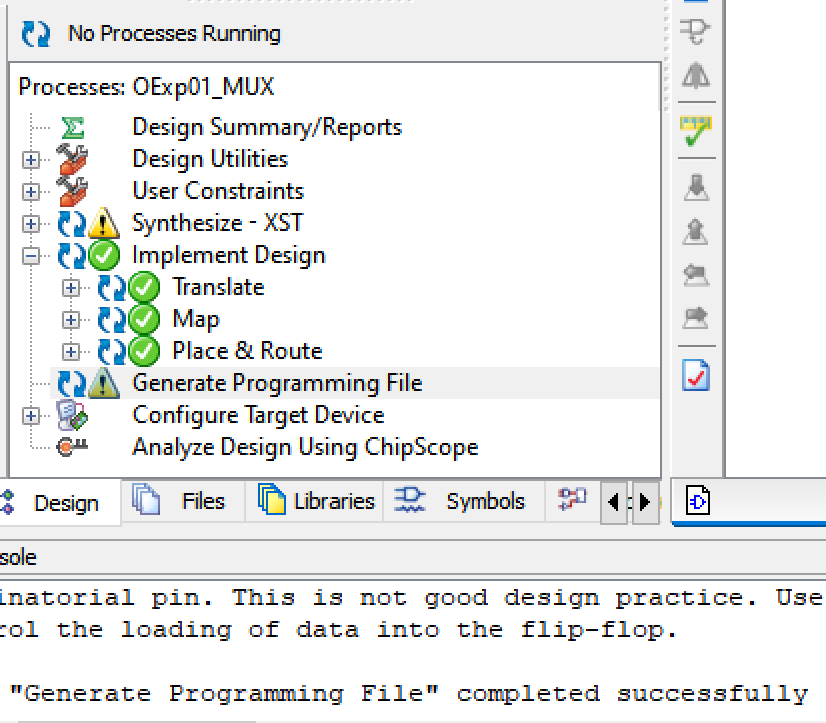
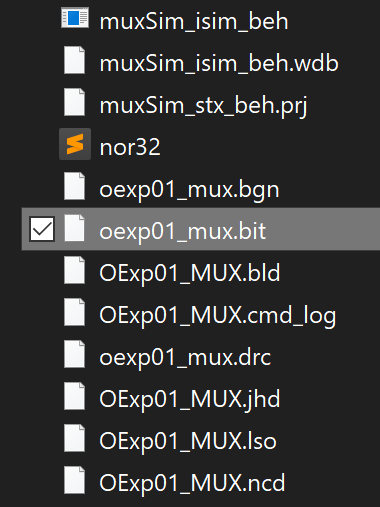


Figure 4 - .bit file found in directory

Figure 3 - .bit file generation

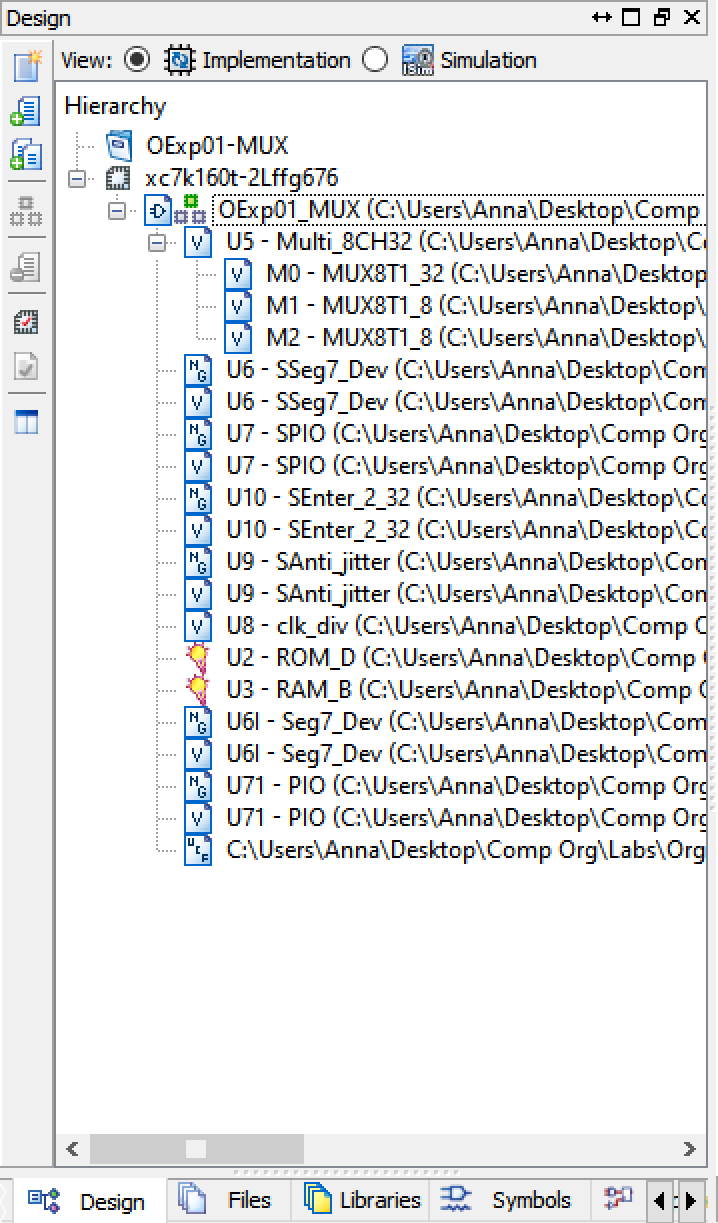


Figure 5 - file hierarchy

To see the Verilog code for each component of top module, please refer to the end of this report.

1. **Simulations and Observations**

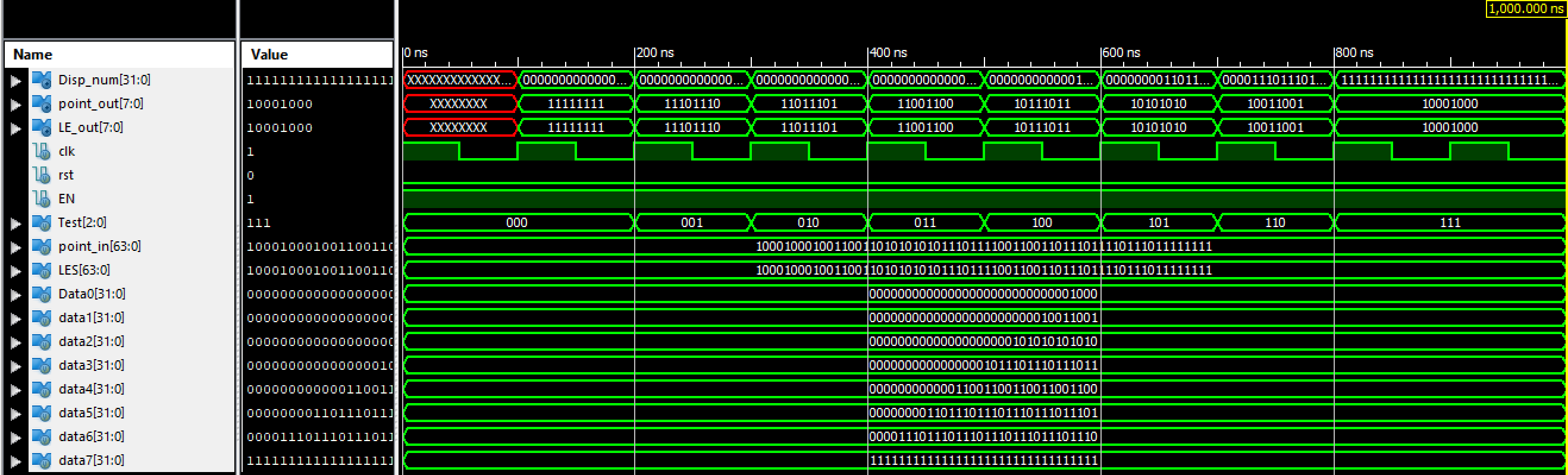


Figure 6 - Multi\_CH32 module

The picture above shows a simulation of the 8 to 32-bit signal multiplexer. As shown, it is functioning correctly and is consistent with its outputs.

*Verilog Test Module for Simulation*

module muxSim;

// Inputs

reg clk;

reg rst;

reg EN;

reg [2:0] Test;

reg [63:0] point\_in;

reg [63:0] LES;

reg [31:0] Data0;

reg [31:0] data1;

reg [31:0] data2;

reg [31:0] data3;

reg [31:0] data4;

reg [31:0] data5;

reg [31:0] data6;

reg [31:0] data7;

// Outputs

wire [7:0] point\_out;

wire [7:0] LE\_out;

wire [31:0] Disp\_num;

// Instantiate the Unit Under Test (UUT)

Multi\_8CH32 uut (

.clk(clk),

.rst(rst),

.EN(EN),

.Test(Test),

.point\_in(point\_in),

.LES(LES),

.Data0(Data0),

.data1(data1),

.data2(data2),

.data3(data3),

.data4(data4),

.data5(data5),

.data6(data6),

.data7(data7),

.Disp\_num(Disp\_num),

.point\_out(point\_out),

.LE\_out(LE\_out)

);

initial begin

// Initialize Inputs

clk = 0;

rst = 0;

EN = 1;

Test = 0;

point\_in = 64'h8899AABBCCDDEEFF;

LES = 64'h8899AABBCCDDEEFF;

Data0 = 32'h8;

data1 = 32'h99;

data2 = 32'hAAA;

data3 = 32'hBBBB;

data4 = 32'hCCCCC;

data5 = 32'hDDDDDD;

data6 = 32'hEEEEEEE;

data7 = 32'hFFFFFFFF;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

#100;

Test = 1;

#100;

Test = 2;

#100;

Test = 3;

#100;

Test = 4;

#100;

Test = 5;

#100;

Test = 6;

#100;

Test = 7;

end

always begin

clk = 1; #50;

clk = 0; #50;

end

endmodule

1. **Conclusion**

From this lab, I realized that this was a continuation and reinforcement of the content that we have learned from our Digital Logic Design course. Some of the modules that were used in building the top module were basically the same but enhanced for the purpose of this lab. This was the first instance that I used the IP Core generator, and I look forward to exploring more to what this does. I only encountered trivial difficulties, such as wiring. Overall, I found this to be a straightforward lab and I look forward to implementing this onto the SWORD board.

1. **Source Code**

*Multi\_CH32 Code (U5)*

module Multi\_8CH32(input clk,

input rst,

input EN,

input[2:0]Test,

input[63:0]point\_in,

input[63:0]LES,

input[31:0] Data0,

input[31:0] data1,

input[31:0] data2,

input[31:0] data3,

input[31:0] data4,

input[31:0] data5,

input[31:0] data6,

input[31:0] data7,

output [7:0] point\_out,

output [7:0] LE\_out,

output [31:0]Disp\_num

);

reg [31:0] disp\_data = 32'hAA5555AA;

reg [7:0] cpu\_blink = 8'b11111111, cpu\_point = 8'b0;

MUX8T1\_32 M0(.I0(disp\_data),

.I1(data1),

.I2(data2),

.I3(data3),

.I4(data4),

.I5(data5),

.I6(data6),

.I7(data7),

.s(Test),

.o(Disp\_num)

);

MUX8T1\_8 M1(.I0(cpu\_blink),

.I1(LES[15:8]),

.I2(LES[23:16]),

.I3(LES[31:24]),

.I4(LES[39:32]),

.I5(LES[47:40]),

.I6(LES[55:48]),

.I7(LES[63:56]),

.s(Test),

.o(LE\_out)

);

MUX8T1\_8 M2(.I0(cpu\_point),

.I1(point\_in[15:8]),

.I2(point\_in[23:16]),

.I3(point\_in[31:24]),

.I4(point\_in[39:32]),

.I5(point\_in[47:40]),

.I6(point\_in[55:48]),

.I7(point\_in[63:56]),

.s(Test),

.o(point\_out)

);

always @(posedge clk) begin

if (EN) begin

disp\_data <= Data0;

cpu\_blink <= LES[7:0];

cpu\_point <= point\_in[7:0];

end else begin

disp\_data <= disp\_data;

cpu\_blink <= cpu\_blink;

cpu\_point <= cpu\_point;

end

end

endmodule

*SSeg7\_Dev Code (U6)*

module SSeg7\_Dev(input clk,

input rst,

input Start,

input SW0,

input flash,

input[31:0]Hexs,

input[7:0]point,

input[7:0]LES,

output seg\_clk,

output seg\_sout,

output SEG\_PEN,

output seg\_clrn

);

endmodule

*Seg7\_Dev Code (U6l)*

module Seg7\_Dev(input[2:0] Scan,

input SW0,

input flash,

input[31:0]Hexs,

input[7:0]point,

input[7:0]LES,

output[7:0]SEGMENT,

output[3:0]AN

);

endmodule

*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

);

endmodule

*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

);

endmodule

*clk\_div Code (U8)*

module clk\_div(input clk,

input rst,

input SW2,

output reg[31:0]clkdiv,

output Clk\_CPU

);

always @ (posedge clk or posedge rst) begin

if (rst) clkdiv <= 0; else clkdiv <= clkdiv + 1'b1; end

assign Clk\_CPU=(SW2)? clkdiv[24] : clkdiv[2];

endmodule

*SAnti\_jitter Code (U9)*

module SAnti\_jitter(input wire clk,

input wire RSTN,

input wire readn,

input wire [3:0]Key\_y,

output reg[4:0] Key\_x,

input wire[15:0]SW,

output reg[4:0] Key\_out,

output reg Key\_ready,

output reg[3:0] pulse\_out,

output reg[3:0] BTN\_OK,

output reg[15:0]SW\_OK,

output reg CR,

output reg rst

);

endmodule

*SEnter\_2\_32 Code (U10)*

module SEnter\_2\_32(input clk,

input[2:0] BTN,

input [4:0] Ctrl, //{SW[7:5],SW[15],SW[0]}

input D\_ready,

input [4:0]Din,

output reg readn,

output reg[31:0]Ai=32'h87654321,

output reg[31:0]Bi=32'h12345678,

output reg [7:0 ]blink

);

endmodule

*Testing Constraints File (UCF)*

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