浙江大学

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

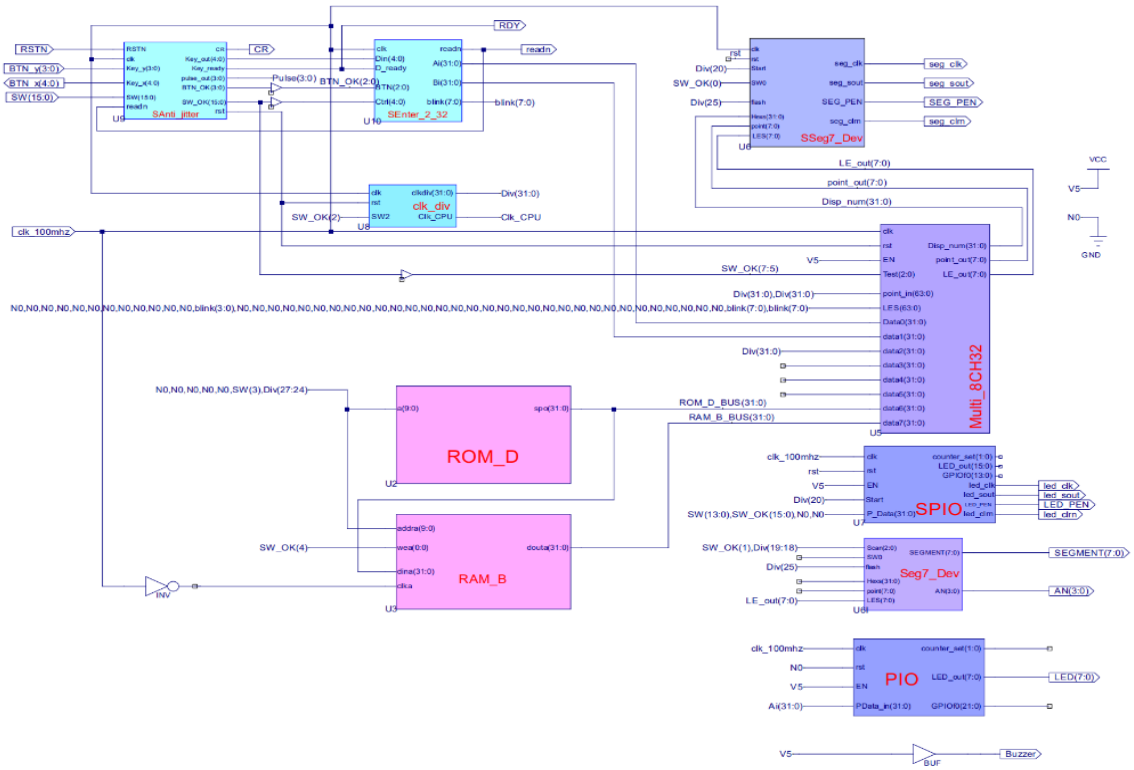

Figure 1 - OExp01_MUX.sch

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.

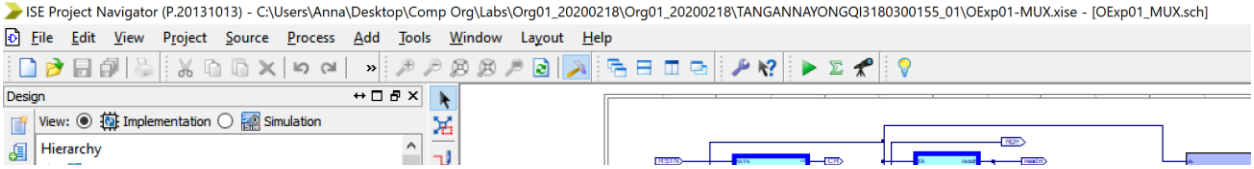

Figure 2 - TANGANNAYONGQI3180300155_01 Folder Name

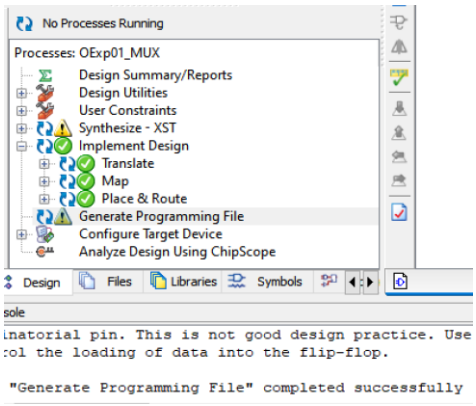

Figure 3 - .bit file generation

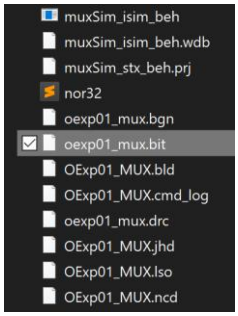

Figure 4 - .bit file found in directory

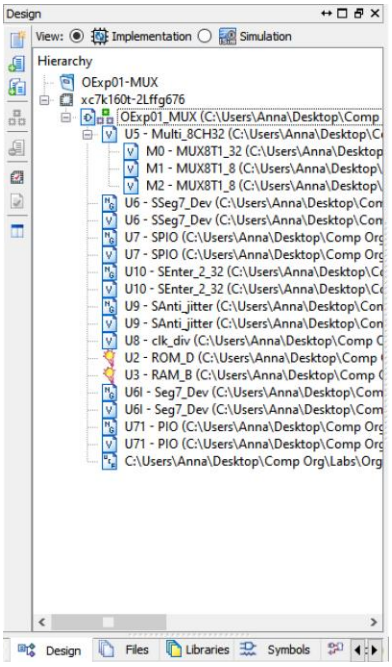

Figure 5 - file hierarchy

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

2. Simulations and Observations

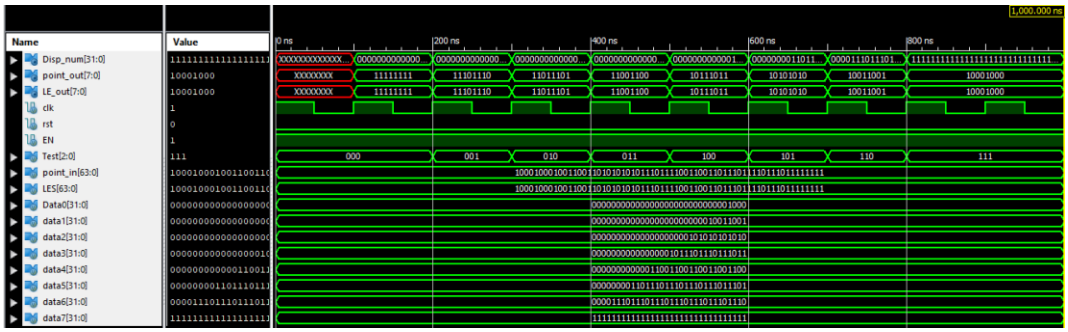

Figure 6 - Multi_CH32 module

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
```

3. 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.

4. 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.