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　4 月 18 日

Lab 5 – Controller Design

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: 2020-04-18 **Instructor:** 洪奇军

1. Method and Experimental Steps

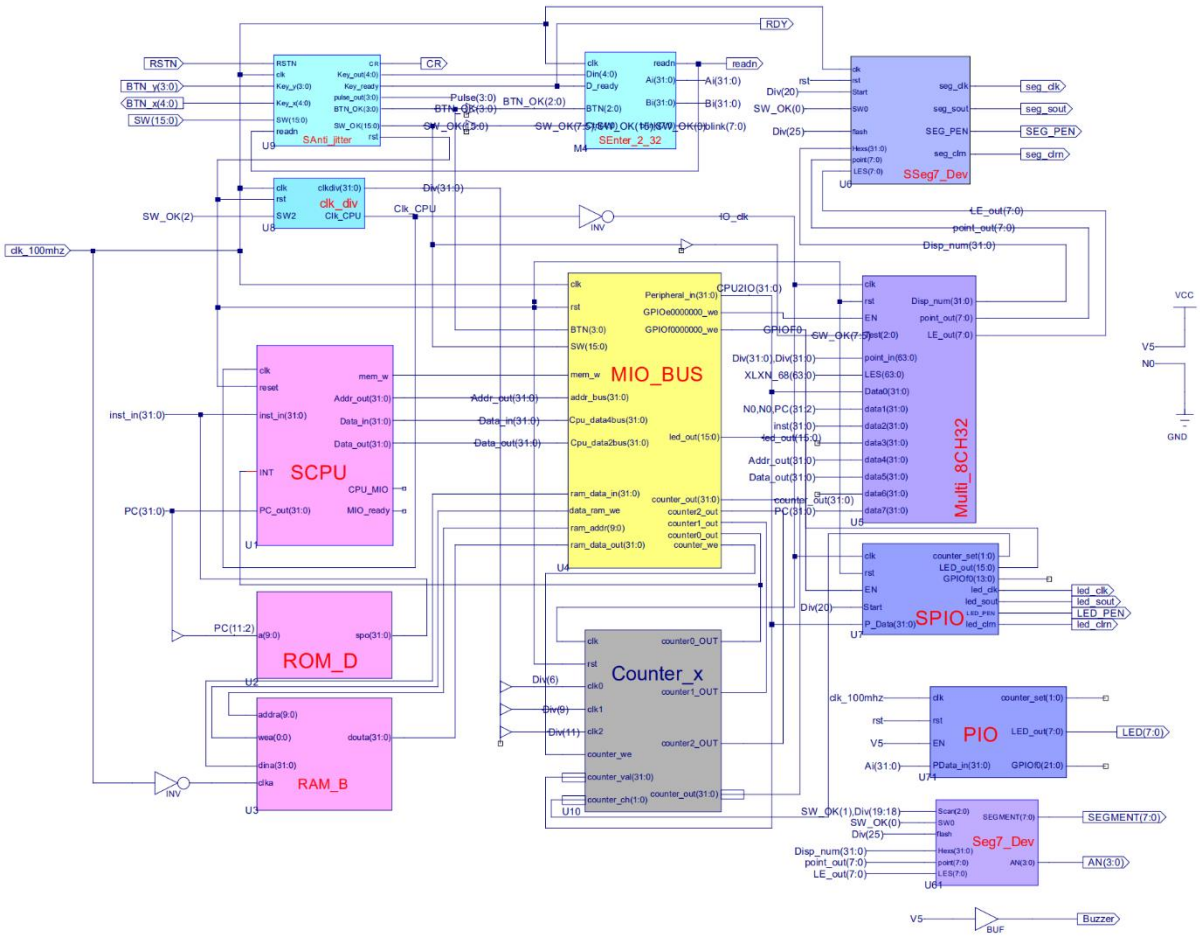

Figure 1 - topMod.sch

This depicts the completion of lab 6. The purpose of this week's experiment was to see how a controller in a processor works, in conjunction with the datapath. This CPU is a single-cycle implementation. The top module is from lab 4, and only the SCPU module was modified. Control is the hardware that tells the datapath what to do, when processing data. An extra control signal (ALU_Control) is added. The controller consists of the main decoder, and the ALUOp decoder. Currently, this CPU should be able to support the current instructions: add, sub, and, or, slt, nor, srl, xor, slti. Instruction memory and data memory were not implemented in this week's lab. The .ucf for this program came from the provided

courseware, and is linked to topMod.sch. Synthesis had minimal warnings, and implementation was successful. A programmable file has been generated and is ready for testing on the SWORD board.

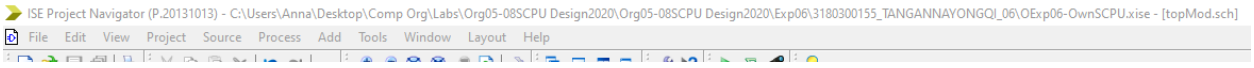

Figure 3 – 3180300155_TANGANNAYONGQI_06

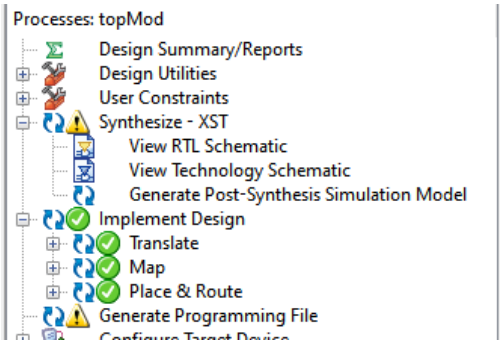

Figure 4 - .bit file generation

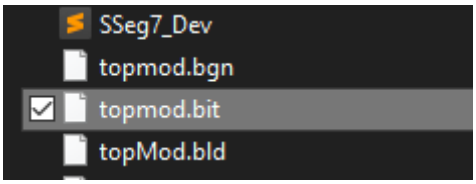

Figure 5 - .bit file generated in directory

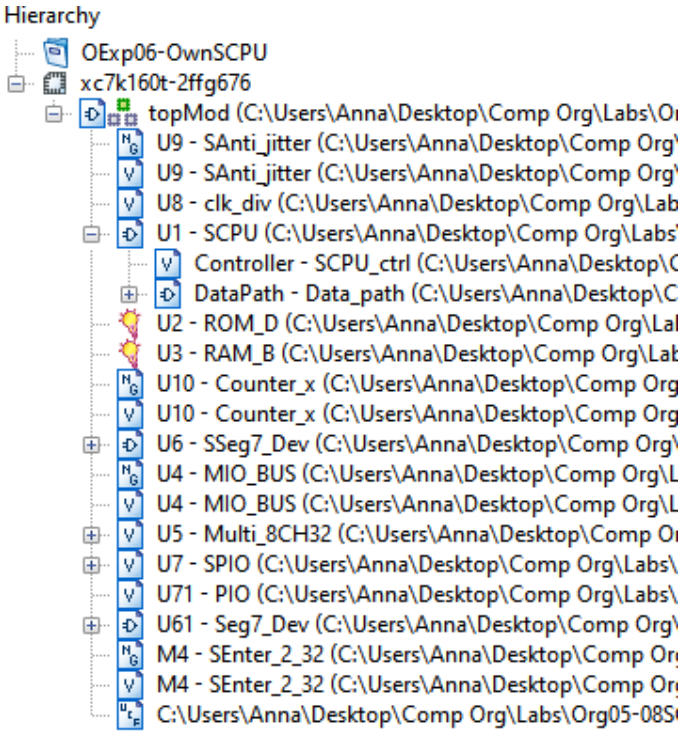

Figure 6 - file hierarchy

2. Simulations and Observations

This lab requires a MIPS program to be designed and tested on the CPU. The DEMO program and datapath testing will be performed later. There is an inconsistency with the simulation below, compared to the one presented in the PowerPoint. I was unable to track down what caused the issue, and used the Verilog Test Fixture presented in the slides as a benchmark.

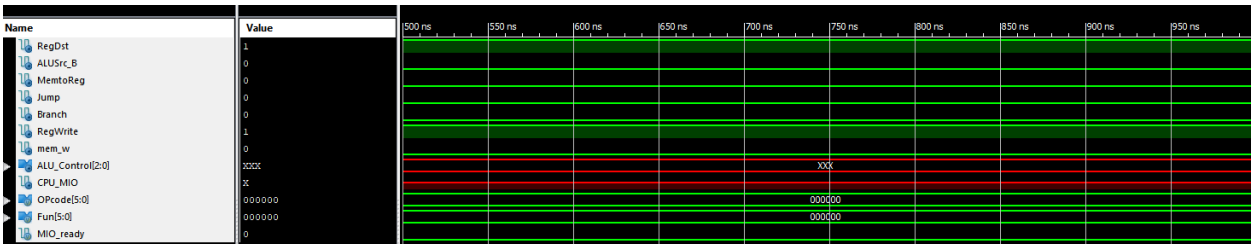

Figure 7 – Controller Simulation

Controller Simulation

```
module scpuctrlSim;

	// Inputs
	reg [5:0] OPcode;
	reg [5:0] Fun;
	reg MIO_ready;

	// Outputs
	wire RegDst;
	wire ALUSrc_B;
	wire MemtoReg;
	wire Jump;
	wire Branch;
	wire RegWrite;
	wire mem_w;
	wire [2:0] ALU_Control;
	wire CPU_MIO;

	// Instantiate the Unit Under Test (UUT)
	SCPU_ctrl uut (
		.OPcode(OPcode),
		.Fun(Fun),
		.MIO_ready(MIO_ready),
		.RegDst(RegDst),
		.ALUSrc_B(ALUSrc_B),
		.MemtoReg(MemtoReg),
		.Jump(Jump),
		.Branch(Branch),
		.RegWrite(RegWrite),
		.mem_w(mem_w),
		.ALU_Control(ALU_Control),
		.CPU_MIO(CPU_MIO)
	);
```

```
      initial begin
            // Initialize Inputs
            OPcode = 0;
            Fun = 0;
            MIO_ready = 0;

            #40;
            // Wait 100 ns for global reset to finish

            // Add stimulus here
            OPcode = 6'b000000; //ALU??,?? ALUop=2'b10; RegDst=1; RegWrite=1
            Fun = 6'b100000;  //add,??ALU_Control=3'b010
            #20;
            Fun = 6'b100010;  //sub,??ALU_Control=3'b110
            #20;
            Fun = 6'b100100;  //and,??ALU_Control=3'b000
            #20;
            Fun = 6'b100101;  //or,??ALU_Control=3'b001
            #20;
            Fun = 6'b101010;  //slt,??ALU_Control=3'b111
            #20;
            Fun = 6'b100111;  //nor,??ALU_Control=3'b100
            #20;
            Fun = 6'b000010;  //srl,??ALU_Control=3'b101
            #20;
            Fun = 6'b010110;  //xor,??ALU_Control=3'b011
            #20;
            Fun = 6'b111111;  //??
            #1;
            OPcode = 6'b100011;//load??,?? ALUop=2'b00, RegDst=0,
            #20;          // ALUSrc_B=1, MemtoReg=1, RegWrite=1
            OPcode = 6'b101011;
            #20; //store??,??ALUop=2'b00, mem_w=1, ALUSrc_B=1
            OPcode = 6'b000100;//beq??,?? ALUop=2'b01, Branch=1
            #20;
            OPcode = 6'b000010;//jump??,?? Jump=1
            #20;
            OPcode = 6'b001010; //slti??,??ALUop=2'b11, RegDst=0,
            #20;              //ALUSrc_B=1, RegWrite=1

            OPcode = 6'h3f;         //??
            Fun = 6'b000000;        //??
      end

endmodule
```

3. Conclusion

This lab was conceptually simple, but it was a bit troublesome to implement. The simulation was also a bit odd to understand. I am looking forward to implementing this completed lab onto the SWORD board, and seeing what output comes out after running the demo MIPS program.

4. Source Code

Controller-SCPU_ctrl.v

```
module SCPU_ctrl( input[5:0]OPcode, //OPcode
                  input[5:0]Fun,    //Function
                  input MIO_ready,  //CPU Wait
                  output reg RegDst,
                  output reg ALUSrc_B,
                  output reg MemtoReg,
                  output reg Jump,
                  output reg Branch,
                  output reg RegWrite,
                  output wire mem_w,
                  output reg [2:0]ALU_Control,
                  output reg CPU_MIO);
     reg MemRead,MemWrite,ALUop1,ALUop0;
   `define  CPU_ctrl_signals{RegDst,ALUSrc_B,MemtoReg,RegWrite,
            MemRead,MemWrite,Branch,Jump,ALUop1,ALUop0}
     assign mem_w = MemWrite&&(~MemRead);

           always@* begin
                 case(OPcode)
                 6'b001010:begin `CPU_ctrl_signals = 10'b0101_0000_11; end
                 6'b000000:begin `CPU_ctrl_signals = 10'b1001_0000_10; end
                 6'b100011:begin `CPU_ctrl_signals = 10'b0111_1000_00; end
                 6'b101011:begin `CPU_ctrl_signals = 10'b1100_0100_00; end
                 6'b000100:begin `CPU_ctrl_signals = 10'b1000_0010_01; end
                 6'b000010:begin `CPU_ctrl_signals = 10'b1000_0001_10; end
                 default: begin `CPU_ctrl_signals = 10'b0000_0000_00; end
                 endcase
                 end

           always @* begin
          case({ALUop1,ALUop0})
          2'b00: ALU_Control = 3'b010;              //add????
          2'b01: ALU_Control = 3'b110;              //sub????
          2'b10:
                 case(Fun)
                       6'b100000: ALU_Control = 3'b010;    //add
                       6'b100010: ALU_Control = 3'b110;    //sub
                       6'b100100: ALU_Control = 3'b000;    //and
                       6'b100101: ALU_Control = 3'b001;    //or
                       6'b101010: ALU_Control = 3'b111;    //slt
                       6'b100111: ALU_Control = 3'b100;    //nor:~(A | B)
                       6'b000010: ALU_Control = 3'b101;    //srl
                       6'b010110: ALU_Control = 3'b011;    //xor
                       default:   ALU_Control=3'bx;
                       endcase
          2'b11: ALU_Control = 3'b110;              //slti
         endcase
           end
```

```
endmodule
```