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 25 日

Lab 4 – IP Core Design CPU/IP2CPU

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: 2020-03-25 **Instructor:** 洪奇军

1. Method and Experimental Steps

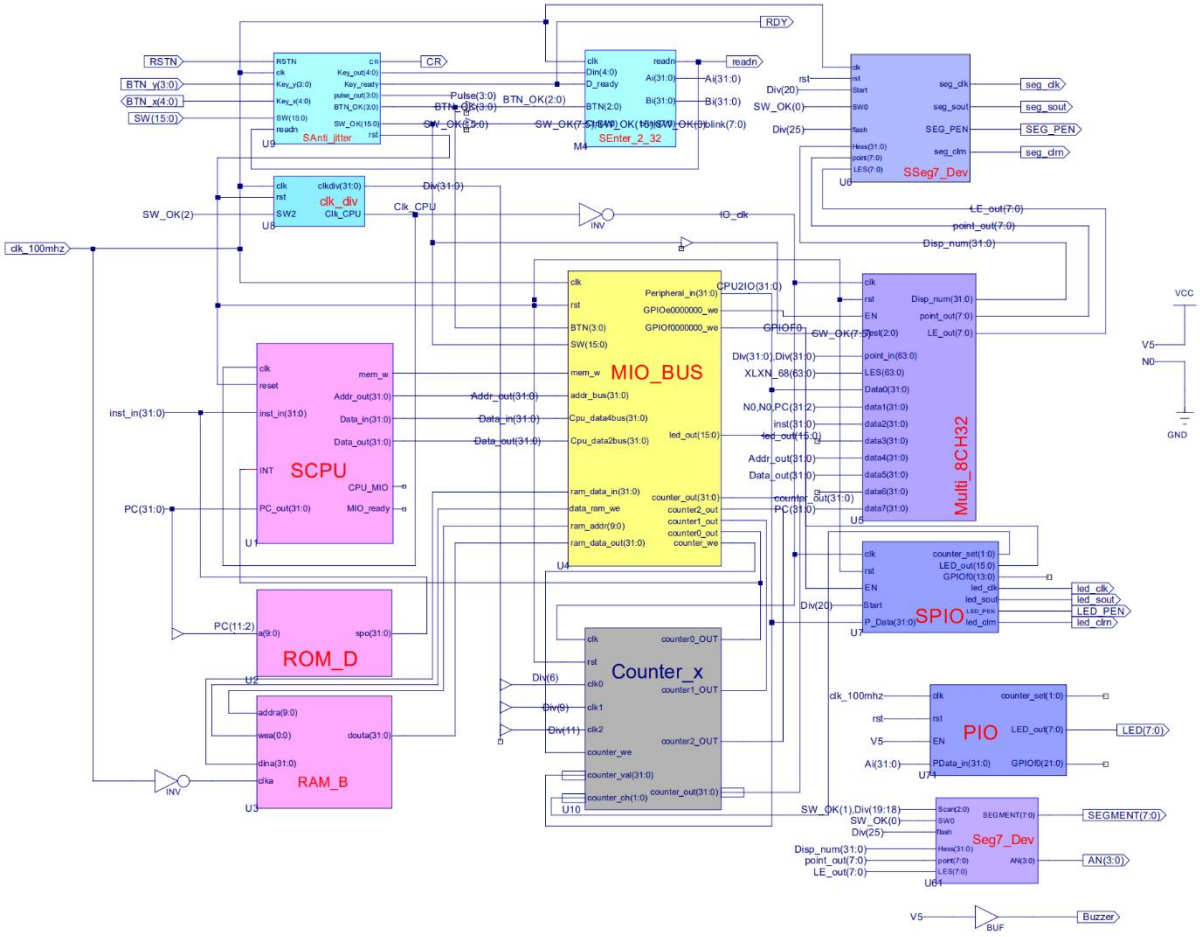

Figure 1 - topMod.sch

This depicts the completion of lab 4. I made two separate projects for this lab, with one being a CPU and the other being an ALU. The CPU is like the one that was built in lab 3, but with modifications to the SCPU module. Here, the CPU consists of the control unit and the data path. The data path is responsible for handling ALU operations, such as calculations, register maintenance and counting. The control unit decodes instructions sent from input and handles any external and internal signals. These topics were briefly examined in the Digital Logic Design course, but will be explored to a further extent here.
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.

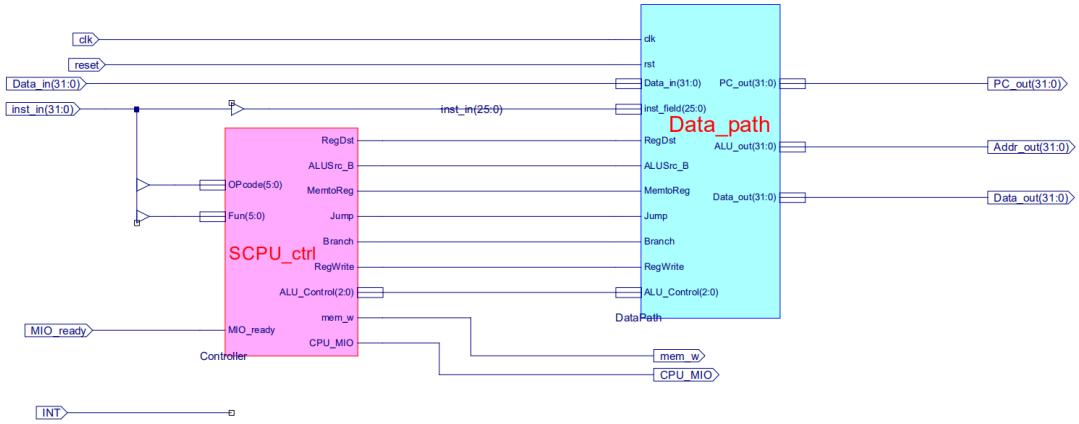

Figure 2 – U1 SCPU Module Schematic

Figure 3 - TANGANNAYONGQI3180300155_04IP2CPU

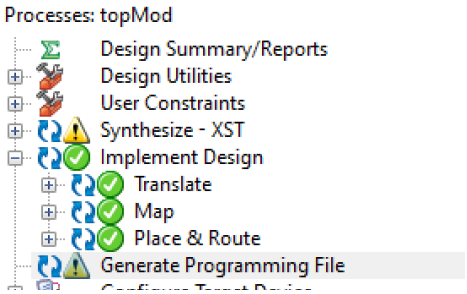

Figure 4 - .bit file generation

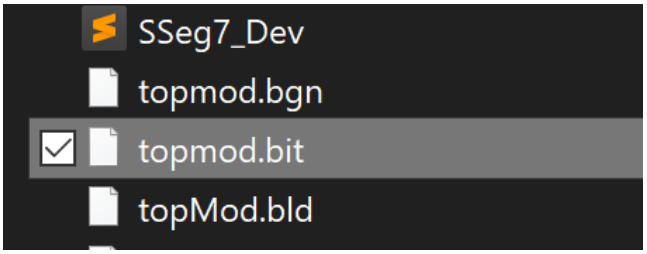

Figure 5 - .bit file generated in directory

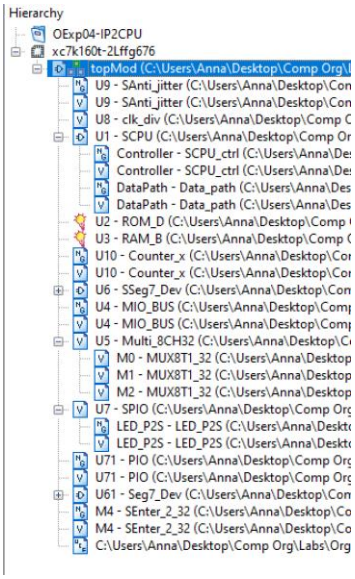

Figure 6 - file hierarchy

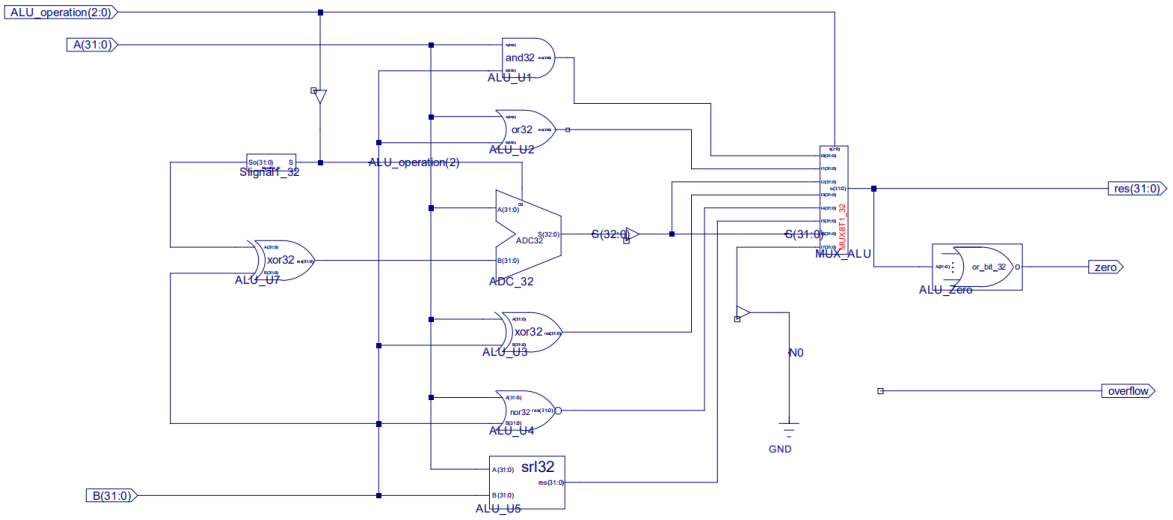

Figure 7 – ALU.sch

The schematic above is the second project of lab 4; the ALU. The ALU is capable of performing logical, arithmetic and shift operations. Examples include add, subtract, AND, OR, NOR, and so on. It takes three inputs A, B, and ALU_Operation which are signal lines that specify the type of operation to perform. It has three outputs res, zero and overflow. This ALU module was based off of the one that was designed in the Digital Logic Design course.

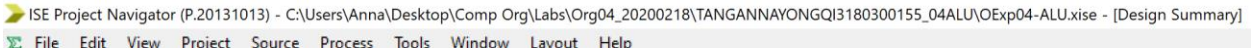

Figure 8 - TANGANNAYONGQI3180300155_04ALU

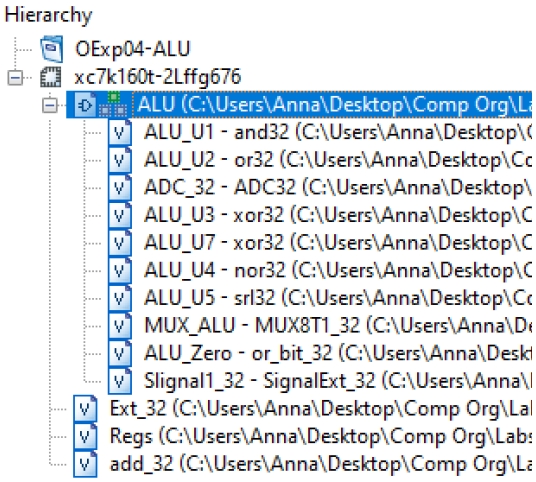

Figure 9 - file hierarchy

2. Simulations and Observations

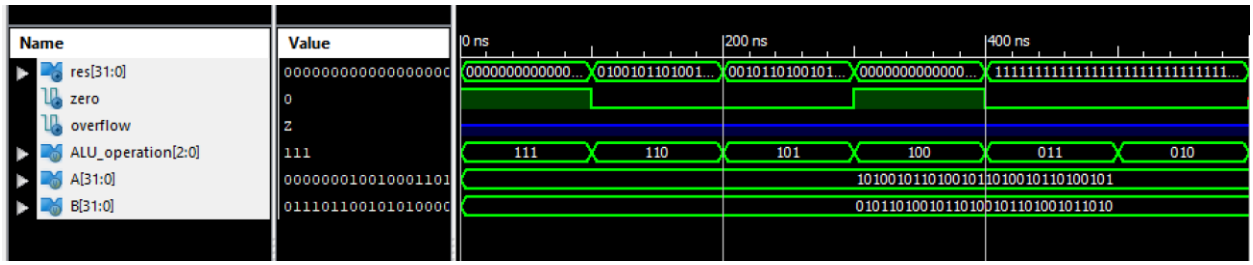

Figure 10 – ALU Simulation

Simulation parameters were taken from the lab PowerPoint. Some of the outcomes differed from the one presented in the slides, and I was unable to track down the issue that caused the difference.

ALU Simulation

```
module ALU_ALU_sch_tb();

// Inputs
   reg [2:0] ALU_operation;
   reg [31:0] A;
   reg [31:0] B;

// Output
   wire [31:0] res;
   wire zero;
   wire overflow;

// Bidirs

// Instantiate the UUT
   ALU UUT (
            .ALU_operation(ALU_operation),
            .res(res),
            .zero(zero),
            .overflow(overflow),
            .A(A),
            .B(B)
   );
// Initialize Inputs
   initial begin
            A = 0;
            B = 0;
            ALU_operation = 0;
            A=32'hA5A5A5A5;
      B=32'h5A5A5A5A;
      ALU_operation =3'b111;
      #100;
      ALU_operation =3'b110;
      #100;
      ALU_operation =3'b101;
```

```
	#100;
	ALU_operation =3'b100;
	#100;
	ALU_operation =3'b011;
	#100;
	ALU_operation =3'b010;
	#100;
	ALU_operation =3'b001;
	#100;
	ALU_operation =3'b000;
	#100;
	A=32'h01234567;
	B=32'h76543210;
	ALU_operation =3'b111;
	end

endmodule
```

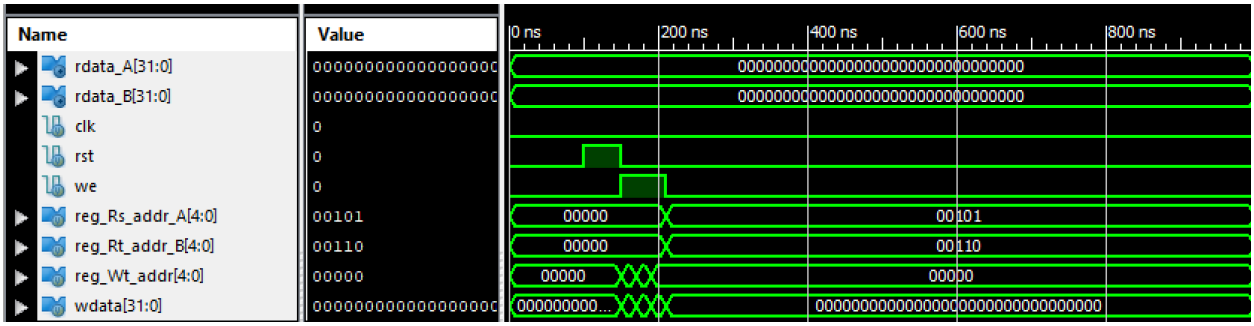

Figure 11 – Regs Simulation

Simulation parameters were taken from the lab PowerPoint. They were consisted with the one presented in the slides.

ALU Regs Simulation

```
module RegsSim;

	// Inputs
	reg clk;
	reg rst;
	reg we;
	reg [4:0] reg_Rs_addr_A;
	reg [4:0] reg_Rt_addr_B;
	reg [4:0] reg_Wt_addr;
	reg [31:0] wdata;

	// Outputs
	wire [31:0] rdata_A;
	wire [31:0] rdata_B;

	// Instantiate the Unit Under Test (UUT)
	Regs uut (
		.clk(clk),
```

```
        .rst(rst),
        .we(we),
        .reg_Rs_addr_A(reg_Rs_addr_A),
        .reg_Rt_addr_B(reg_Rt_addr_B),
        .reg_Wt_addr(reg_Wt_addr),
        .wdata(wdata),
        .rdata_A(rdata_A),
        .rdata_B(rdata_B)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 0;
        we = 0;
        reg_Rs_addr_A = 0;
        reg_Rt_addr_B = 0;
        reg_Wt_addr = 0;
        wdata = 0;

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

        // Add stimulus here
        rst = 1;
        #50;
        rst = 0;
        we = 1;
        reg_Rs_addr_A = 0;
        reg_Rt_addr_B = 0;
        reg_Wt_addr = 5;
        wdata = 32'hA5A5A5A5;
        #20;
        we = 1;
        reg_Rs_addr_A = 0;
        reg_Rt_addr_B = 0;
        reg_Wt_addr = 6;
        wdata = 32'h55AA55AA;
        #20;
        we = 1;
        reg_Rs_addr_A = 0;
        reg_Rt_addr_B = 0;
        reg_Wt_addr = 0;
        wdata = 32'hAAAA5555;
        #20;
        we = 0;
        reg_Rs_addr_A = 5;
        reg_Rt_addr_B = 6;
        reg_Wt_addr = 0;
        wdata = 0;
        #20;
    end
```

```
endmodule
```

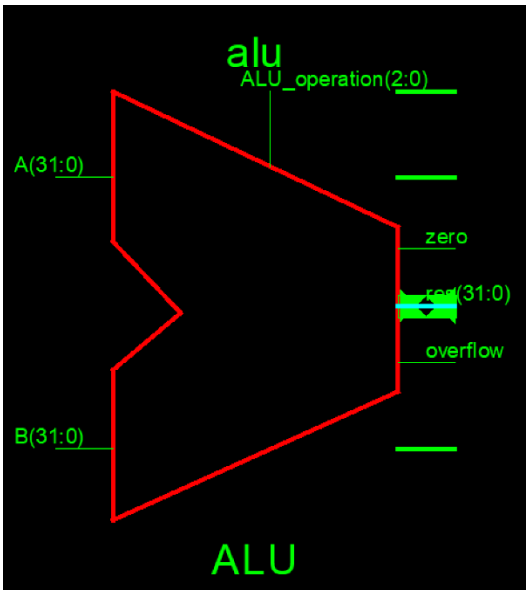

Figure 12 – ALU RTL symbol

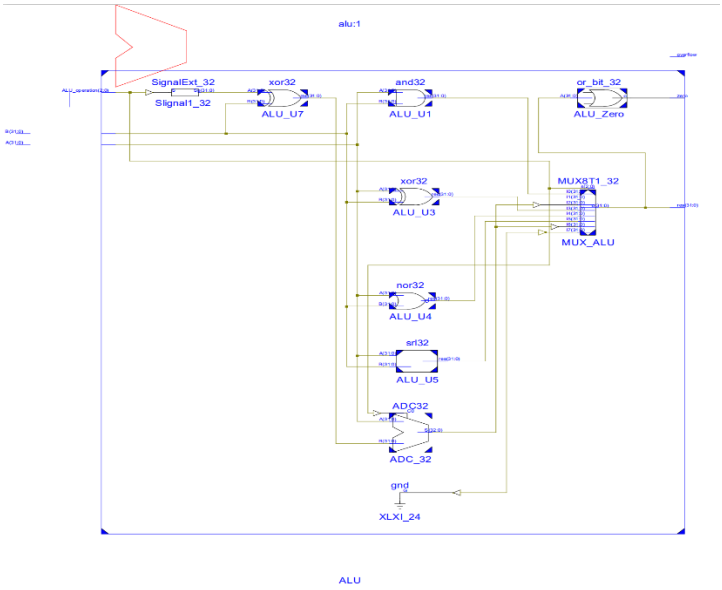

Figure 13 – ALU RTL

RTL schematic of the ALU. Here, we see the ALU as a module symbol and its digital logic structure.

3. Conclusion

In this week's lab, I was mostly interested in the CPU module. We haven't really gotten the chance to explore the data path and control unit in the theoretical section of this course, so I'm looking forward to that. I think the purpose of making the ALU in this lab is to prepare for the next experiments, where we focus on building the processor of the CPU. Overall, I found this lab to be very straightforward and simple to implement.

4. Source Code

All modules and components were either retrieved from lab 1, lab 2 or directly taken from the given files.

ALU Regs

```
module  Regs(input clk, rst, we,
             input [4:0] reg_Rs_addr_A, reg_Rt_addr_B, reg_Wt_addr,
             input [31:0] wdata,
             output [31:0] rdata_A,  rdata_B
             );
      reg [31:0] register [1:31];   // r1 - r31
      integer i;

      assign rdata_A = (reg_Rs_addr_A == 0) ? 0 : register[reg_Rs_addr_A];
      // read
```

```
      assign rdata_B = (reg_Rt_addr_B == 0) ? 0 : register[reg_Rt_addr_B];
      // read

      always @(posedge clk or posedge rst)  begin
            if (rst == 1) begin
                  for (i=1; i<32; i=i+1) begin
                        register[i] <= 0;       // reset
                  end
            end else if ((reg_Wt_addr != 0) && (we == 1)) begin
                  register[reg_Wt_addr] <= wdata;      // write
            end
      end

endmodule
```