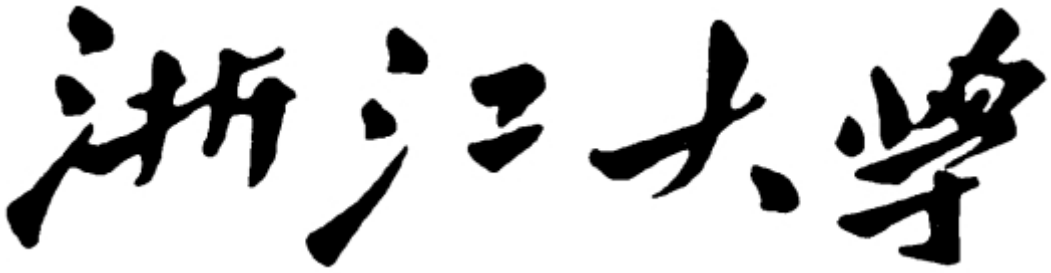


****本科实验报告****

课程名称:	**计算机组成**
姓 名:	臧可
学 院:	计算机科学与技术学院
系:	计算机科学与技术系
专 业:	计算机科学与技术
学 号:	3180102095
指导教师:	姜晓红

2020年 05 月 08 日

浙江大学实验报告

课程名称： 计算机组成 实验类型： 综合

实验项目名称： Lab3-single cycle CPU implementation

学生姓名： 臧可 专业：计算机科学与技术 学号：3180102095

同组学生姓名： None 指导老师： 姜晓红

实验地点：家里 实验日期： 2020年 05 月 08日

一、 实验目的和要求

Please do the project step by step according to the lab tutorial.

Object: Implement a combinational logic that enable the right switches on FPGA board control the eight Led lamp. Say, if switch being state on “010”, then only the 2nd Led lamp is on.

Requirement: using the logical design way avoiding software coding like case statement.

1. **Object: Design data path components and perform timing simulation**
    - ALU
    - Register Files

    Requirement:

    - ALU
        - Arrange the ALU of logic experiment eight
        - Logic diagram input and simulation
    - Register Files
        - Optimization logic experiment Regs
        - Behavior description and simulation results
2. **Object: Design a 18+ instruction data path**
    - Arithmetic logic component with general calculation function
    - General purpose register
    - Have the best possible path required for universal counting

    Requirement:

    - Using logic schematic design to realize data path
        - ALU and Regs call modules designed by Exp04
    - Replace Exp04's data path core
    - This experiment is completed on the basis of Exp04
3. **Object:Design a controller with 18+ instructions**
    - One of the main components of CPU
    - Register transfer controller: code to command
    - Design the controller test plan
    - Design the controller test program

    Requirement:

    - Design CPU controller
        - According to the analysis and discussion of the theory class, the controller designing the five data paths of the experiment
        - Schematic diagram or HDL description can be
            - But it must be described with a function expression structure
        - Simulation test controller module
    - Integrate and replace the validated data path module
        - Replace the SCPU_ctrl.ngc core in Experiment 5 (Exp05)

        - Module name: Top_OExp06_OwnSCPU.sch
    - Test controller module
        - Design test program (MIPS assembly) test:

- OP decoding test:
  - R-format, memory access instruction, branch instruction, branch instruction
    - Operation control test: Function decoding test

4. **Object:Set top-level module**

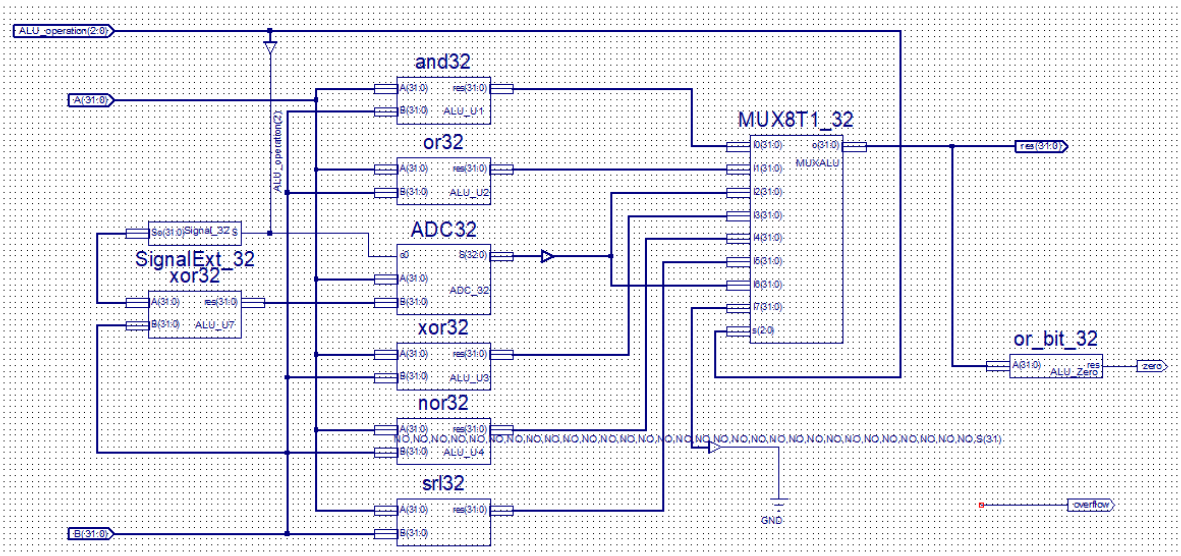
    - New CPU with integrated replacement verification
    - CPU logic symbols need to be modified

    Requirement:

    - implement no less than the following instructions
        - R-Type: add, sub, and, or, xor, nor, slt, srl *, jr, jalr, eret;
        - I-Type: addi, andi, ori, xori, lui, lw, sw, beq, bne, slti
        - J-Type: J, Jal *;
    - Design test program (MIPS assembly) test CPU logic symbols

二、 实验内容和原理

Logic schematic input design ALU

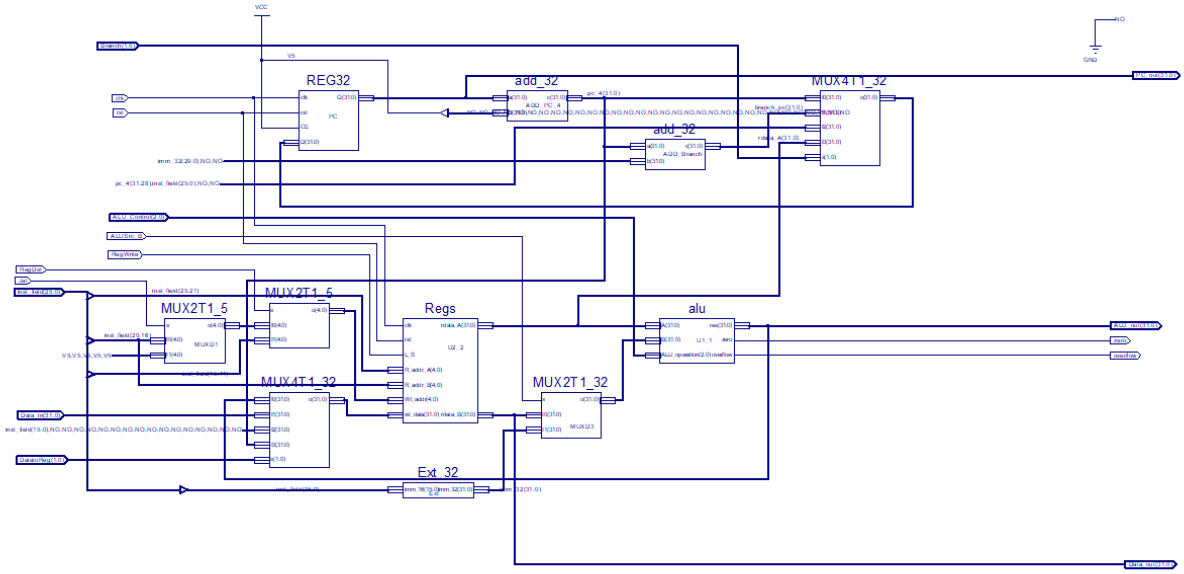


Behavior description design Register files

```
module Regs(input clk,rst,L_S,
                input [4:0] R_addr_A, R_addr_B, Wt_addr,
                input [31:0] wt_data,
                output [31:0] rdata_A, rdata_B
    );
     reg [31:0] register[1:31];
     integer i;


     assign rdata_A = (R_addr_A == 0)?0:register[R_addr_A];
     assign rdata_B = (R_addr_B == 0)?0:register[R_addr_B];
```

```
12
13
14      always@(posedge clk or posedge rst)
15         begin if(rst==1)
16                     for(i=1;i<32;i=i+1)
17                         register[i] <= 0;
18                 else if((Wt_addr != 0) && (L_S == 1))    //0地址保留给zero
19                     register[Wt_addr] <= wt_data;
20         end
21
22 endmodule
```

Logic schematic input design Data_path



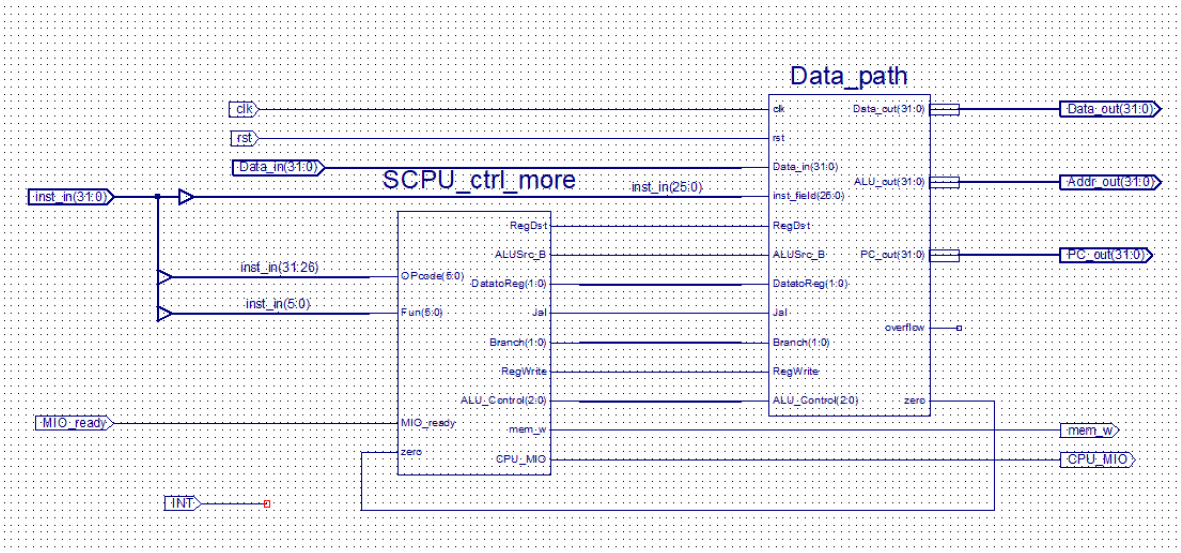
SCPU_ctrl HDL description structure

```
1  module SCPU_ctrl_more(input [5:0]OPcode,
2                        input [5:0]Fun,
3                        input MIO_ready,
4                        input zero,
5                        output reg RegDst,
6                        output reg ALUSrc_B,
7                        output reg [1:0] DatatoReg,
8                        output reg Jal,
9                        output reg [1:0]Branch,
10                       output reg RegWrite,
11                       output reg mem_w,
12                       output reg [2:0]ALU_Control,
13                       output reg CPU_MIO
14     );
15     `define
CPU_ctrl_signals{RegDst,ALUSrc_B,DatatoReg,Jal,Branch,RegWrite,mem_w,CPU_MI
O,ALU_Control}
16         always@*begin
17             case(OPcode)
18                 6'b000000:begin
```

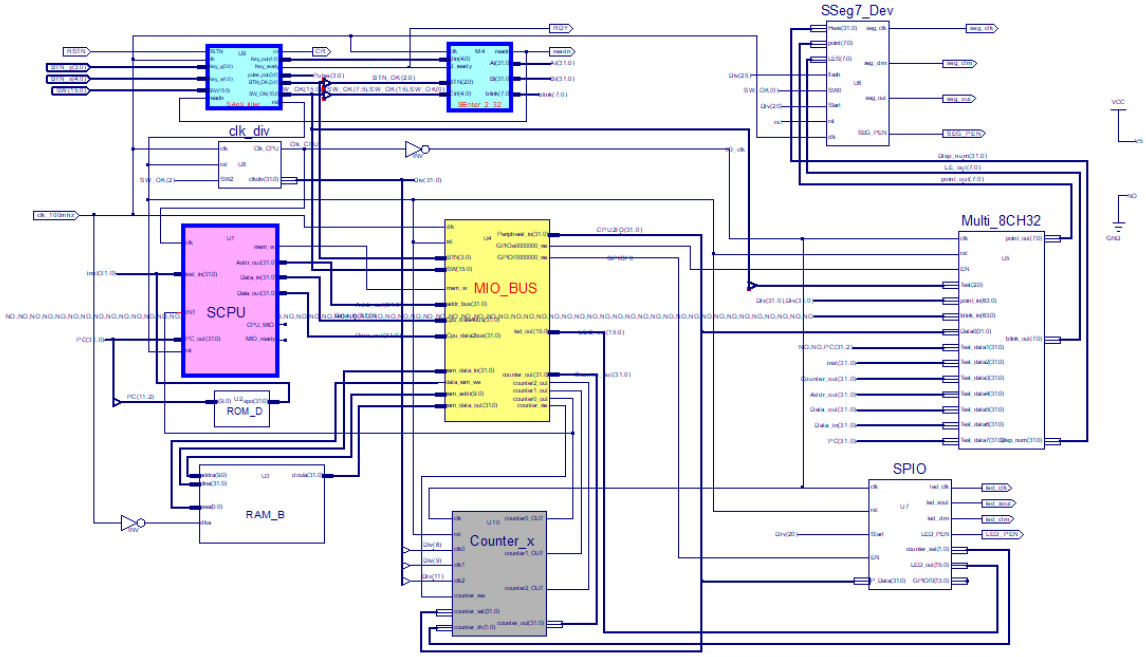
```
                    case(Fun)
                        6'b100000:begin `CPU_ctrl_signals = 13'b10000_00100_010;end //add
                        6'b100010:begin `CPU_ctrl_signals = 13'b10000_00100_110;end //sub
                        6'b100100:begin `CPU_ctrl_signals = 13'b10000_00100_000;end //and
                        6'b100101:begin `CPU_ctrl_signals = 13'b10000_00100_001;end //or
                        6'b000010:begin `CPU_ctrl_signals = 13'b10000_00100_101;end //srl
                        6'b101010:begin `CPU_ctrl_signals = 13'b10000_00100_111;end //slt
                        6'b100111:begin `CPU_ctrl_signals = 13'b10000_00100_100;end //nor
                        6'b100110:begin `CPU_ctrl_signals = 13'b10000_00100_011;end //xor
                        6'b001000:begin `CPU_ctrl_signals = 13'b10001_11000_000;end //jr
                        6'b001001:begin `CPU_ctrl_signals = 13'b10111_11100_010;end //jalr
                        default:  begin `CPU_ctrl_signals = 13'b00000_00000_000;end
                    endcase
                end
                6'b100011:begin `CPU_ctrl_signals = 13'b01010_00100_010;end //load
                6'b101011:begin `CPU_ctrl_signals = 13'b01000_00010_010;end //store
                6'b000100:begin
                    if(zero==1'b1) `CPU_ctrl_signals = 13'b00000_01000_110;
                    else           `CPU_ctrl_signals = 13'b00000_00000_110;
                end //beq
                6'b000101:begin
                    if(zero==1'b0) `CPU_ctrl_signals = 13'b00000_00000_110;
                    else           `CPU_ctrl_signals = 13'b00000_01000_110;
                end //bne
                6'b000010:begin `CPU_ctrl_signals = 13'b00000_10000_000;end //j
                6'b000011:begin `CPU_ctrl_signals = 13'b00111_10100_010;end //jal
                6'b001111:begin `CPU_ctrl_signals = 13'b00100_00100_010;end //lui
                6'b001010:begin `CPU_ctrl_signals = 13'b01000_00100_111;end //slti
                6'b001110:begin `CPU_ctrl_signals = 13'b01000_00100_011;end //xori
                6'b001111:begin `CPU_ctrl_signals = 13'b01000_00100_001;end //ori
                6'b001100:begin `CPU_ctrl_signals = 13'b01000_00100_000;end //andi
                6'b001000:begin `CPU_ctrl_signals = 13'b01000_00100_010;end //addi
                6'b010000:begin `CPU_ctrl_signals = 13'b10000_11100_010;end //eret
                default:  begin `CPU_ctrl_signals = 10'b00000_00000_000;end
            endcase
        end
```

```
55
56 endmodule
```

SCPU



Top



三、 实验过程和数据记录及结果分析

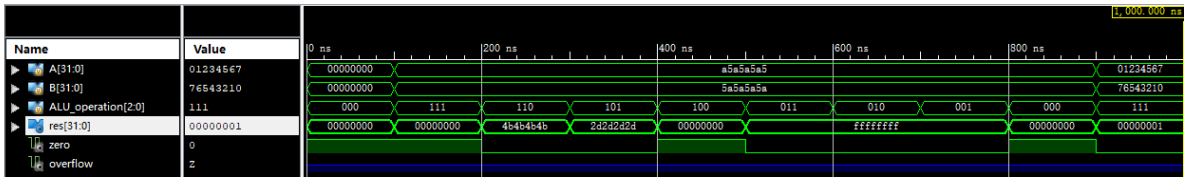
ALU

ALU Incentive code

```
1         initial begin
2          A = 0;
3          B = 0;
```

```
        ALU_operation = 0;
        #100;
        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
```

ALU_Simulation results



Simulation results are as expected

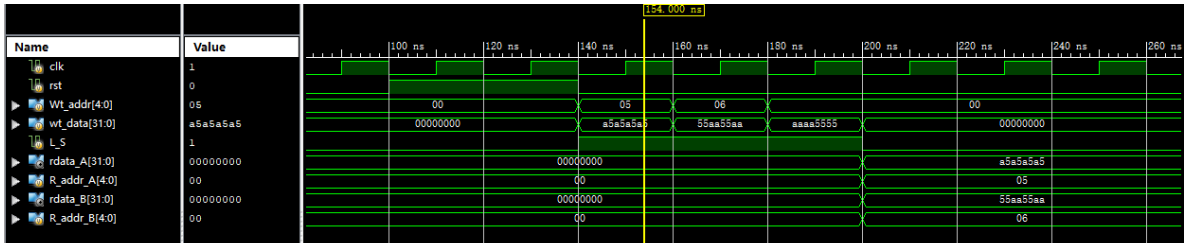
Regs Files

Regs Incentive code

```
    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 0;
        L_S = 0;
        R_addr_A = 0;
        R_addr_B = 0;
        Wt_addr = 0;
        wt_data = 0;

        // Wait 100 ns for global reset to finish
        #100;
        rst = 1;
        #40;
        rst = 0;
        L_S = 1;
        Wt_addr = 5;
        wt_data = 32'hA5A5A5A5;
```

```
        #20;
        Wt_addr = 6;
        wt_data = 32'h55AA55AA;
        #20;
        Wt_addr = 0;
        wt_data = 32'hAAAA5555;
        #20;
        L_S = 0;
        wt_data = 0;
        R_addr_A = 5;
        R_addr_B = 6;

    end

    always begin
        clk=0;#10;
        clk=1;#10;
    end

```

Regs simulation results



Simulation results are as expected

Data_path

Data_path Incentive code

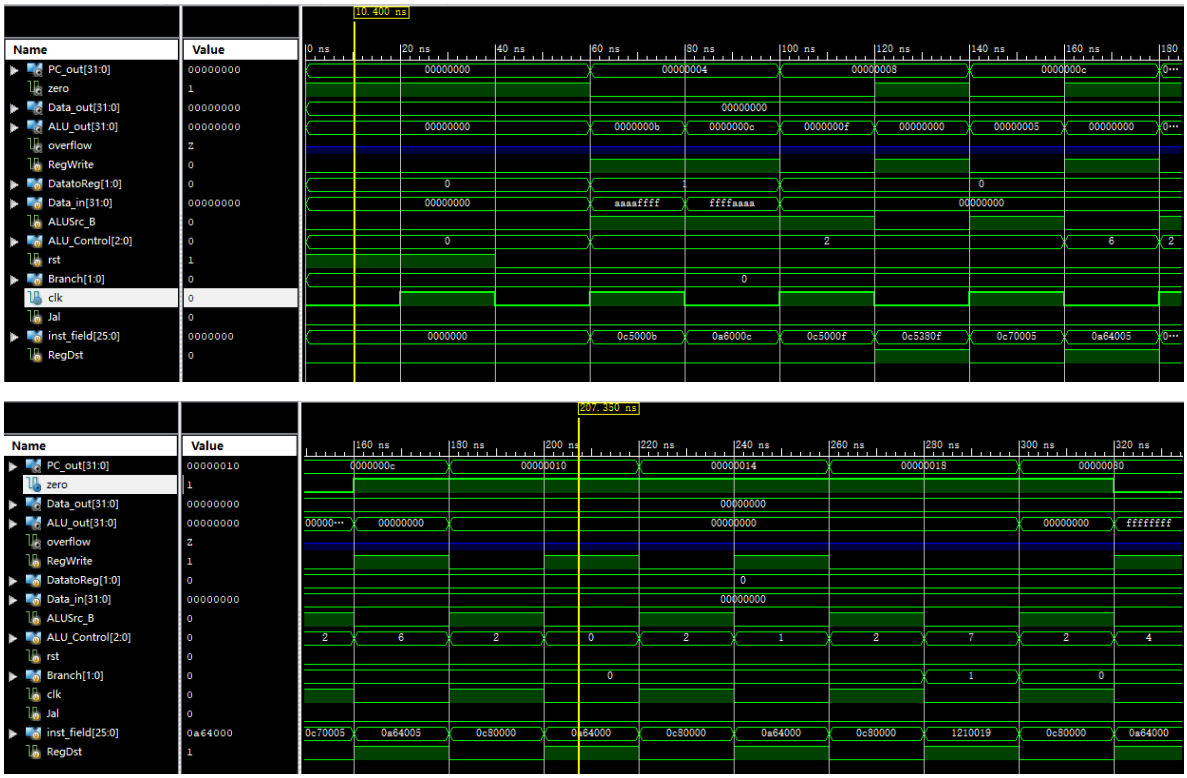
```
        initial begin
              RegWrite = 0;
              DatatoReg = 0;
              Data_in = 0;
              ALUSrc_B = 0;
              ALU_Control = 0;
              rst = 1;
              Branch = 0;
              clk = 0;
              Jal = 0;
              inst_field = 0;
              RegDst = 0;

              #40; rst=0;
              #20;
              //lw $5,11($6)
              Jal = 0;
              Branch = 0;
              RegWrite = 1;
              inst_field[25:21]=6; //rs
              inst_field[20:16]=5; //rt
```

```
                inst_field[15:0]=11;
                RegDst = 0;
                DatatoReg = 1;
                Data_in = 32'haaaaffff;
                ALUSrc_B = 1;
                ALU_Control=2;
                #20;
                //lw $6,12($5)
                inst_field[25:21]=5;
                inst_field[20:16]=6;
                inst_field[15:0]=12;
                Data_in = 32'hffffaaaa;
                #20;
                //sw $5,15($6)
                RegWrite = 0;
                inst_field[25:21]=6;
                inst_field[20:16]=5;
                inst_field[15:0]=15;
                DatatoReg = 0;
                Data_in = 0;
                #20;
                //add $7,$6,$5
                RegWrite = 1;
                inst_field[25:21]=6;
                inst_field[20:16]=5;
                inst_field[15:11]=7;
                RegDst = 1;
                ALUSrc_B = 0;
                #20;
                //sw $7,5($6)
                RegWrite = 0;
                inst_field[25:21]=6;
                inst_field[20:16]=7;
                inst_field[15:0]=5;
                RegDst = 0;
                ALUSrc_B = 1;
                #20;
                //sub $8,$5,$6
                RegWrite = 1;
                inst_field[25:21]=5;
                inst_field[20:16]=6;
                inst_field[15:11]=8;
                RegDst = 1;
                ALUSrc_B = 0;
                ALU_Control=6;
                #20;
                //sw $8,0($6)
                RegWrite = 0;
                inst_field[25:21]=6;
                inst_field[20:16]=8;
                inst_field[15:0]=0;
                RegDst = 0;
                DatatoReg = 0;
                ALUSrc_B = 1;
                ALU_Control=2;
                #20;
                //and $8,$5,$6
                RegWrite = 1;
```

```
                inst_field[25:21]=5;
                inst_field[20:16]=6;
                inst_field[15:11]=8;
                RegDst = 1;
                ALUSrc_B = 0;
                ALU_Control=0;
                #20;
                //sw $8,0($6)
                RegWrite = 0;
                inst_field[25:21]=6;
                inst_field[20:16]=8;
                inst_field[15:0]=0;
                RegDst = 0;
                ALUSrc_B = 1;
                ALU_Control=2;
                #20;
                //or $8,$5,$6
                RegWrite = 1;
                inst_field[25:21]=5;
                inst_field[20:16]=6;
                inst_field[15:11]=8;
                RegDst = 1;
                ALUSrc_B = 0;
                ALU_Control=1;
                #20;
                //sw $8,0($6)
                RegWrite = 0;
                inst_field[25:21]=6;
                inst_field[20:16]=8;
                inst_field[15:0]=0;
                RegDst = 0;
                ALUSrc_B = 1;
                ALU_Control=2;
                #20;
                //slt $25,$9,$1
                Branch = 1;
                RegWrite = 0;
                inst_field[25:21]=9;
                inst_field[20:16]=1;
                inst_field[15:0]=25;
                RegDst = 1;
                ALUSrc_B = 0;
                ALU_Control=7;
                #20;
                //beq $6,$8,0
                Branch = 0;
                RegWrite = 0;
                inst_field[25:21]=6;
                inst_field[20:16]=8;
                inst_field[15:0]=0;
                RegDst = 0;
                ALUSrc_B = 1;
                ALU_Control=2;
                #20;
                //nor $8,$5,$6
                RegWrite = 1;
                inst_field[25:21]=5;
                inst_field[20:16]=6;
```

```
                inst_field[15:11]=8;
                RegDst = 1;
                ALUSrc_B = 0;
                ALU_Control=4;
                #20;
        end

        always begin
        clk = 0; #20;
        clk = 1; #20;
        end
```

Data_path simulation results



Simulation results are as expected

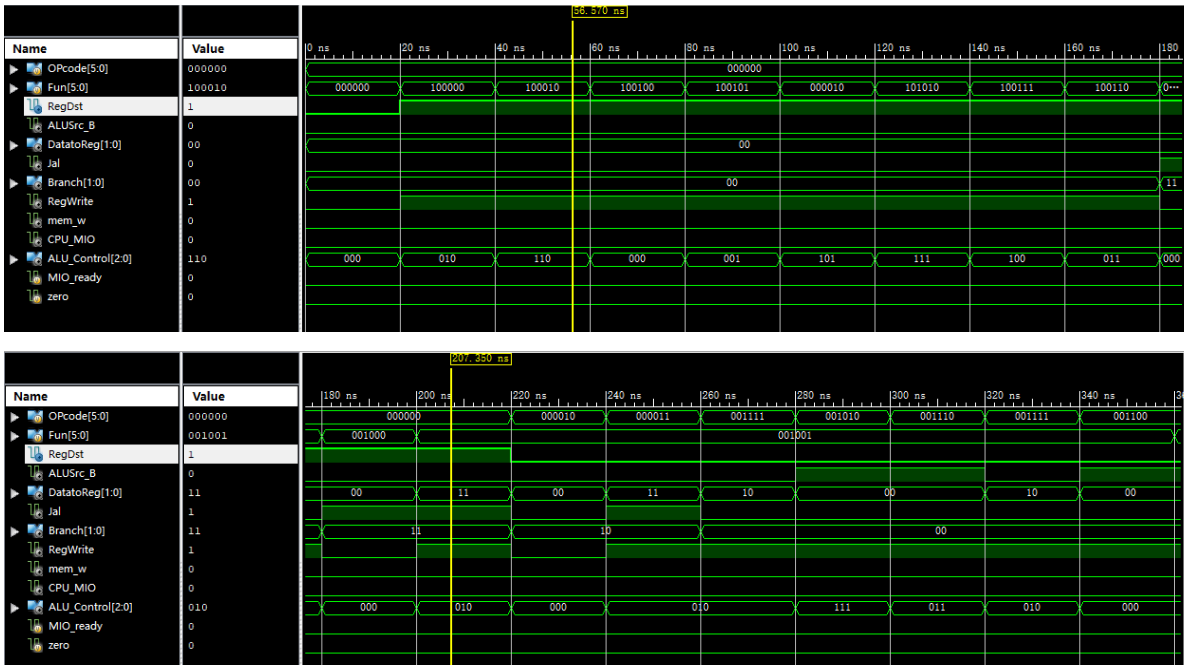
SCPU_ctrl

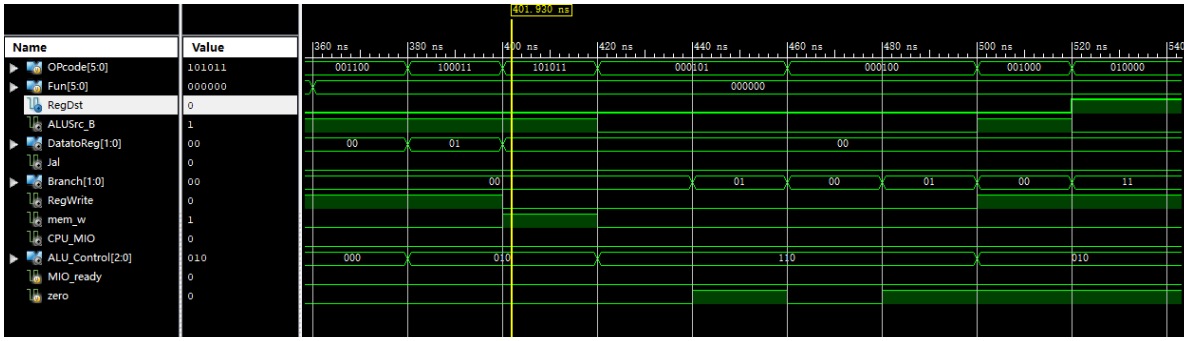
SCPU_ctrl Incentive code

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

        // Wait 20 ns for global reset to finish
        #20;

        // Add stimulus here
        Fun = 6'b100000;//add
```

```
13          #20; Fun = 6'b100010;//sub
14          #20; Fun = 6'b100100;//and
15          #20; Fun = 6'b100101;//or
16          #20; Fun = 6'b000010;//srl
17          #20; Fun = 6'b101010;//slt
18          #20; Fun = 6'b100111;//nor
19          #20; Fun = 6'b100110;//xor
20          #20; Fun = 6'b001000;//jr
21          #20; Fun = 6'b001001;//jalr
22          #20; OPcode = 6'b000010;//j
23          #20; OPcode = 6'b000011;//jal
24          #20; OPcode = 6'b001111;//lui
25          #20; OPcode = 6'b001010;//slti
26          #20; OPcode = 6'b001110;//xori
27          #20; OPcode = 6'b001111;//ori
28          #20; OPcode = 6'b001100;//andi
29          #20; Fun = 6'b000000;//interval
30          #20; OPcode = 6'b100011;//lw
31          #20; OPcode = 6'b101011;//sw
32          #20; OPcode = 6'b000101;//bne
33          zero = 0;//ne
34          #20; OPcode = 6'b000101;//bne
35          zero = 1;//eq
36          #20; OPcode = 6'b000100;//beq
37          zero = 0;//ne
38          #20; OPcode = 6'b000100;//beq
39          zero = 1;//eq
40          #20; OPcode = 6'b001000;//addi
41          #20; OPcode = 6'b010000;//eret
42
43      end
```

SCPU_ctrl simulation results

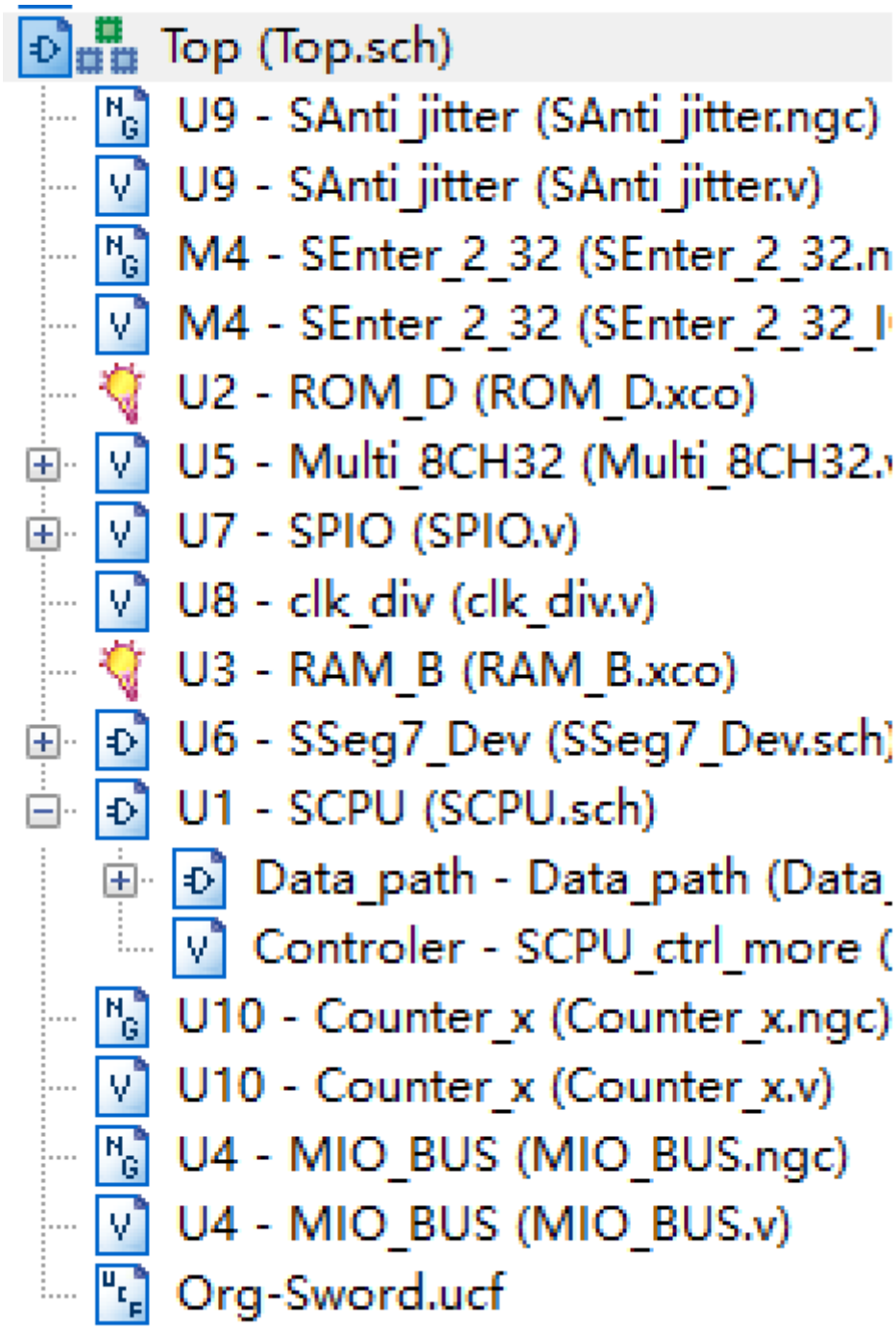




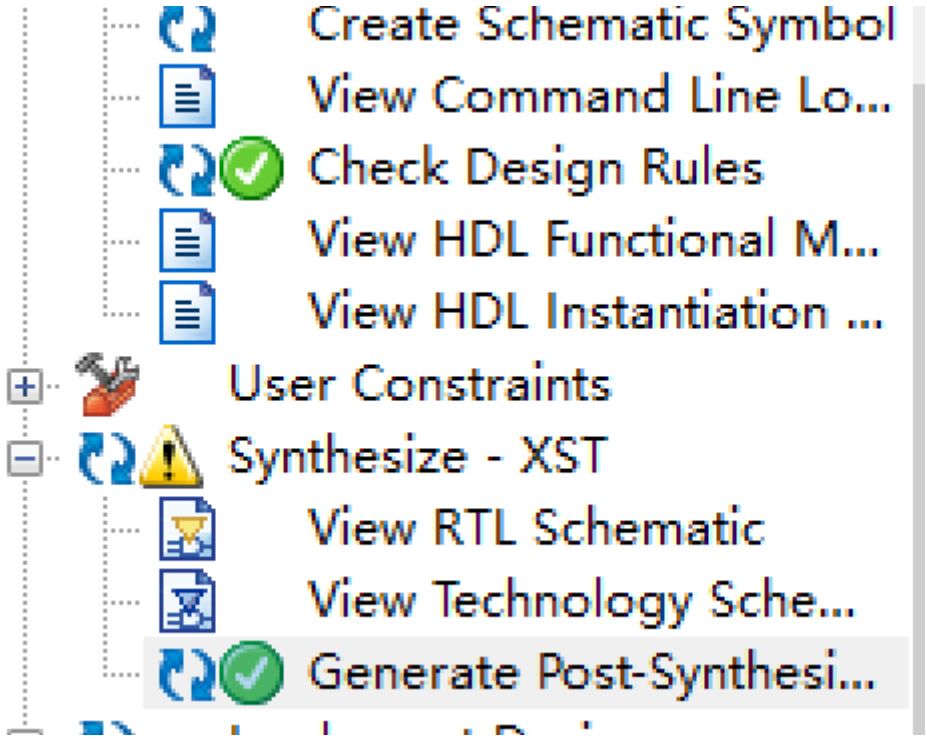
Simulation results are as expected

Top

Top-level structure



Verified



Generate Post-Synthesis Simulation Model successfully

四、 讨论与心得

1. Through this experiment, I have a deeper understanding of the design of the logic diagram. For example, you cannot connect the same line at the entrance and exit of a component. 1'b1 can be renamed with the line connected by vcc, 1'b0 can be renamed with the line connected by gnd.
2. In the process of simulation, I have a more in-depth understanding of the implementation of instructions.
3. I learned the verilog grammar of `define, and my skill of cable has also improved.