

Lab4

Lab4

CPU数据通路与控制器实现

Lab4-1

数据通路实现

PC

ImmGen

顶层模块：DataPath

CPU替换与下板验证

数据通路替换集成

SCPU替换集成

上板准备

VGA信号调试

Lab4-2

控制器实现

模块实现

仿真结果

CPU集成

模块实现

功能仿真

下板验证

CPU指令拓展

数据通路重实现

控制器重实现

模块代码

功能仿真

SOC平台仿真

下板验证

CPU指令中断

数据通路修改

控制器修改

下板验证

CPU数据通路与控制器实现

Lab4-1

数据通路实现

PC

```

1 `timescale 1ns / 1ps
2
3 module REG32(
4     input clk, rst, CE, [31:0] D,
5     output reg [31:0] Q
6 );
7     always @(posedge clk or posedge rst) begin // clk上升沿触发
8         if (rst) Q <= 0; // rst == 1时复位寄存器
9         else if (CE) Q <= D; // CE == 1时使能, 写入寄存器
10    end
11 endmodule

```

ImmGen

根据注释所写对不同格式的指令集进行立即数的编码提取

31	30	25 24	21	20	19	15 14	12 11	8	7	6	0	opcode	R-type
		funct7		rs2		rs1	funct3		rd		opcode	I-type	
		imm[11:0]			rs1		funct3		rd		opcode	S-type	
		imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode		B-type	
		imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode		U-type	
				imm[31:12]					rd		opcode	J-type	
		imm[20]	imm[10:1]	imm[11]	imm[19:12]				rd		opcode		

```

1 `timescale 1ns / 1ps
2
3 module ImmGen (
4     input wire [1:0] ImmSel, //立即数操作控制
5     input wire [31:0] inst_field, //指令数据域[31:7]
6     output reg [31:0] Imm_out //立即数输出
7 );
8     always@(*) begin
9         case(ImmSel)
10             2'b00: Imm_out = {{20{inst_field[31]}}, inst_field[31:20]};
11                 //addi\lw(I)
12             2'b01: Imm_out = {{20{inst_field[31]}}, inst_field[30:25],
13             inst_field[11:7]}; //sw(s)
14             2'b10: Imm_out = {{20{inst_field[31]}}, inst_field[7],
15             inst_field[30:25], inst_field[11:8], 1'b0}; //beq(b)
16             2'b11: Imm_out = {{20{inst_field[31]}}, inst_field[19:12],
17             inst_field[20], inst_field[30:21], 1'b0}; //jal(j)
18         endcase
19     end
20 endmodule

```

顶层模块：DataPath

根据数据通路原理图，完成对顶层模块DataPath的设计

同时为了确保VGA能够显示，加入 `Defines.vh` 宏定义

后续想法：子模块中的 `add_32` 和 `MUX` 其实在模块代码书写时可以采用功能性描述，而不是结构性描述，便于debug

```

1 `include "Defines.vh"
2 `timescale 1ns / 1ps

```

```

3
4 module DataPath(
5     input wire Branch, //Beq指令
6     input wire Jump, //J指令
7     input wire [31:0] Data_in, //存储器输入
8     input wire [1:0] MemtoReg, //Regs写入数据源控制
9     input wire ALUSrc_B, //ALU端口B输入选择
10    input wire [1:0] ImmSel, //ImmGen操作控制
11    input wire [31:0] inst_field, //指令数据域[31:7]
12    input wire [2:0] ALU_operation, //ALU操作控制
13    input wire clk, //寄存器时钟
14    input wire rst, //寄存器复位
15    input wire Regwrite, //寄存器写入
16    output [31:0] ALU_out, //ALU运算输出
17    output [31:0] Data_out, //CPU数据输出
18    output [31:0] PC_out, //PC指针输出
19    `RegFile_Regs_output
20 );
21
22 // Module wires definition
23 wire [31:0] Imm_out;
24 wire [31:0] add_32_0_c;
25 wire [31:0] add_32_1_c;
26 wire [31:0] MUX2T1_32_1_o;
27 wire [31:0] MUX4T1_32_0_o;
28 wire [31:0] MUX2T1_32_3_o;
29 wire [31:0] MUX2T1_32_0_o;
30 wire [31:0] Rs1_data;
31 wire [31:0] Rs2_data;
32 wire [31:0] res;
33 wire zero;
34 wire [31:0] Q;
35
36 // Module instances
37 ImmGen Immgen (
38     // Input
39     .ImmSel(ImmSel),
40     .inst_field(inst_field),
41     // Output
42     .Imm_out(Imm_out)
43 );
44
45 add_32 add_32_0 (
46     // Input
47     .a(Q),
48     .b(32'b100),
49     // Output
50     .c(add_32_0_c)
51 );
52
53 add_32 add_32_1 (
54     // Input
55     .a(Q),
56     .b(Imm_out),
57     // Output

```

```

58     .c(add_32_1_c)
59 );
60
61     MUX2T1_32 MUX2T1_32_1 (
62         // Input
63         .I0(add_32_0_c),
64         .I1(add_32_1_c),
65         .s(Branch & zero),
66         // Output
67         .o(MUX2T1_32_1_o)
68 );
69
70     MUX4T1_32 MUX4T1_32_0 (
71         // Input
72         .s(MemtoReg),
73         .I0(res),
74         .I1(Data_in),
75         .I2(add_32_0_c),
76         .I3(add_32_0_c),
77         // Output
78         .o(MUX4T1_32_0_o)
79 );
80
81     MUX2T1_32 MUX2T1_32_3 (
82         // Input
83         .I0(MUX2T1_32_1_o),
84         .I1(add_32_1_c),
85         .s(Jump),
86         // Output
87         .o(MUX2T1_32_3_o)
88 );
89
90     MUX2T1_32 MUX2T1_32_0 (
91         // Input
92         .I0(Rs2_data),
93         .I1(Imm_out),
94         .s(ALUSrc_B),
95         // Output
96         .o(MUX2T1_32_0_o)
97 );
98
99     Regs Regs (
100         // Input
101         .clk(clk),
102         .rst(rst),
103         .Rs1_addr(inst_field[19:15]),
104         .Rs2_addr(inst_field[24:20]),
105         .wt_addr(inst_field[11:7]),
106         .wt_data(MUX4T1_32_0_o),
107         .RegWrite(RegWrite),
108         // Output
109         .Rs1_data(Rs1_data),
110         .Rs2_data(Rs2_data),
111         `RegFile_Regs_Arguments
112 );

```

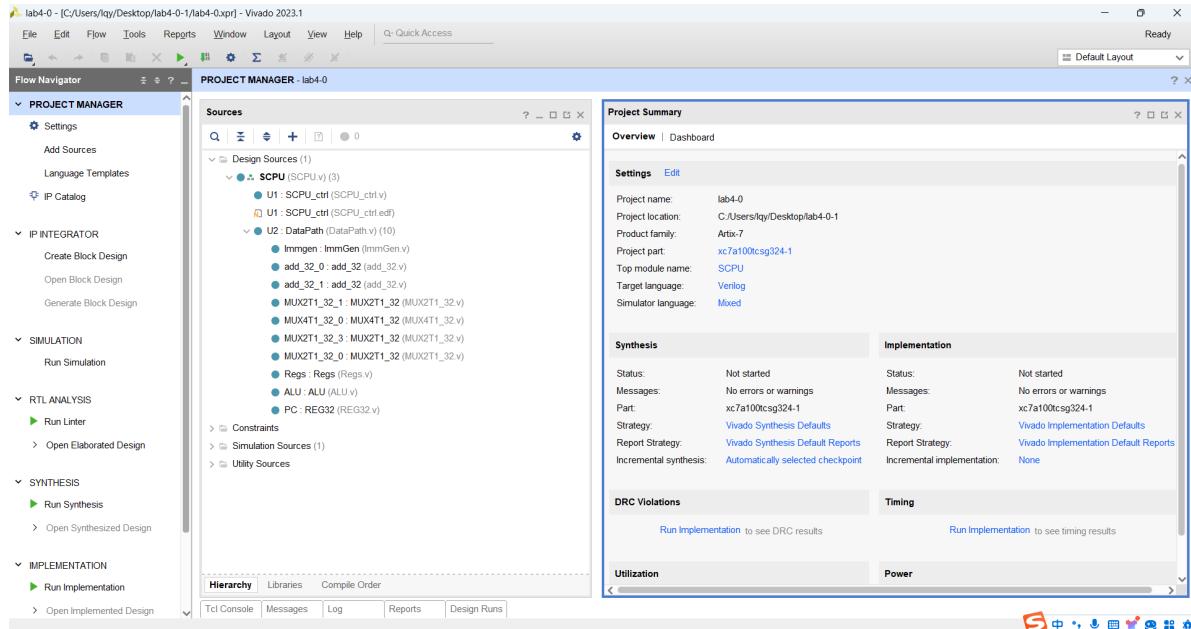
```

113
114     ALU ALU (
115         // Input
116         .A(Rs1_data),
117         .ALU_operation(ALU_operation),
118         .B(MUX2T1_32_0_o),
119         // Output
120         .res(res),
121         .zero(zero)
122     );
123
124     REG32 PC (
125         // Input
126         .clk(clk),
127         .rst(rst),
128         .CE(1'b1),
129         .D(MUX2T1_32_3_o),
130         // Output
131         .Q(Q)
132     );
133
134     // Output
135     assign ALU_out = res;
136     assign Data_out = Rs2_data;
137     assign PC_out = Q;
138
139 endmodule

```

CPU替换与下板验证

数据通路替换集成



SCPU替换集成

与上述同理，进行SCPU的打包、替换

上板准备

重新生成ROM和RAM并烧录初始化文件

VGA信号调试

将 Define.vh 通过预编译指令 `include` 到 VGA.v 中，并在合适位置进行宏调用以完成32个通用寄存器的VGA显示

```

1 `include "defines.vh"
2 `timescale 1ns / 1ps
3
4 module VGA(
5     input wire clk_25m,
6     input wire clk_100m,
7     input wire rst,
8     input wire [31:0] pc,
9     input wire [31:0] inst,
10    input wire [31:0] alu_res,
11    input wire mem_wen,
12    input wire [31:0] dmem_o_data,
13    input wire [31:0] dmem_i_data,
14    input wire [31:0] dmem_addr,
15    `VGA_Regs_Input
16    output wire hs,
17    output wire vs,
18    output wire [3:0] vga_r,
19    output wire [3:0] vga_g,
20    output wire [3:0] vga_b
21 );
22    wire [9:0] vga_x;
23    wire [8:0] vga_y;
24    wire video_on;
25    VgaController vga_controller(
26        ...
27    );
28    wire display_wen;
29    wire [11:0] display_w_addr;
30    wire [7:0] display_w_data;
31    VgaDisplay vga_display(
32        ...
33    );
34    VgaDebugger vga_debugger(
35        ...
36        .pc          (pc          ),
37        .inst         (inst         ),
38        ...
39        .alu_res      (alu_res      ),
40        `RegFile_Regs_Arguments ,
41        ...
42    );
43 endmodule

```

Lab4-2

控制器实现

模块实现

这个模块的实现主要是根据 `opcode` 来确定一坨参数，然后 `ALU_Control` 再依靠 `opcode` 和 `func3` 和 `func7` 来确定，只需要写一堆 `case` 条件语句即可

```

1 `timescale 1ns / 1ps
2
3 module SCPU_ctrl(
4     input[4:0]OPcode, //Opcode-----inst[6:2]
5     input[2:0]Fun3, //Function----inst[14:12]
6     input Fun7, //Function----inst[30]
7     input MIO_ready, //CPU Wait not use
8     output reg [1:0]ImmSel, //立即数选择控制
9     output reg ALUSrc_B, //源操作数2选择
10    output reg [1:0]MemtoReg, //写回数据选择控制
11    output reg Jump, //jal
12    output reg Branch, //beq
13    output reg Regwrite, //寄存器写使能
14    output reg MemRW, //存储器读写使能
15    output reg [2:0]ALU_Control, //alu控制
16    output CPU_MIO //not use
17 );
18
19     reg [1:0] ALUop;
20     wire [3:0] Fun;
21     assign Fun = {Fun3,Fun7};
22     `define CPU_ctrl_signals {ImmSel, ALUSrc_B, MemtoReg, RegWrite, MemRW,
Branch, Jump, ALUop}
23     always@(*) begin
24         case(OPcode)
25             5'b01100: begin `CPU_ctrl_signals = {2'b00, 1'b0, 2'b00, 1'b1,
1'b0, 1'b0, 1'b0, 2'b10}; end //ALU
26             5'b00000: begin `CPU_ctrl_signals = {2'b00, 1'b1, 2'b01, 1'b1,
1'b0, 1'b0, 1'b0, 2'b00}; end //load
27             5'b01000: begin `CPU_ctrl_signals = {2'b01, 1'b1, 2'b00, 1'b0,
1'b1, 1'b0, 1'b0, 2'b00}; end //store
28             5'b11000: begin `CPU_ctrl_signals = {2'b10, 1'b0, 2'b00, 1'b0,
1'b0, 1'b1, 1'b0, 2'b01}; end //beq
29             5'b11011: begin `CPU_ctrl_signals = {2'b11, 1'b0, 2'b10, 1'b1,
1'b0, 1'b0, 1'b1, 2'b00}; end //jump
30             5'b00100: begin `CPU_ctrl_signals = {2'b00, 1'b1, 2'b00, 1'b1,
1'b0, 1'b0, 1'b0, 2'b11}; end //ALU(addi;;;;)
31             default: begin `CPU_ctrl_signals = {2'b00, 1'b1, 2'b00, 1'b1,
1'b0, 1'b0, 1'b0, 2'b11}; end
32         endcase
33     end
34
35     always@(*) begin
36         case(ALUop)
37             2'b00: ALU_Control = 3'b010; //add计算地址
38             2'b01: ALU_Control = 3'b110; //sub比较条件

```

```

39      2'b10:
40          case(Fun)
41              4'b0000: ALU_Control = 3'b010 ; //add
42              4'b0001: ALU_Control = 3'b110; //sub
43              4'b1110: ALU_Control = 3'b000; //and
44              4'b1100: ALU_Control = 3'b001; //or
45              4'b0100: ALU_Control = 3'b111; //slt
46              4'b1010: ALU_Control = 3'b101; //srl
47              4'b1000: ALU_Control = 3'b011; //xor
48              default: ALU_Control = 3'bxxx;
49      endcase
50
51      2'b11:
52          case(Fun3)
53              3'b000: ALU_Control = 3'b010; //addi
54              3'b010: ALU_Control = 3'b111; //slti
55              3'b100: ALU_Control = 3'b011; // xori
56              3'b110: ALU_Control = 3'b001; //ori
57              3'b111: ALU_Control = 3'b000; //andi
58              3'b101: ALU_Control = 3'b101; //srlti
59              default: ALU_Control = 3'bxxx;
60      endcase
61  end
62
63  assign CPU_MIO = MIO_ready;
64
65 endmodule

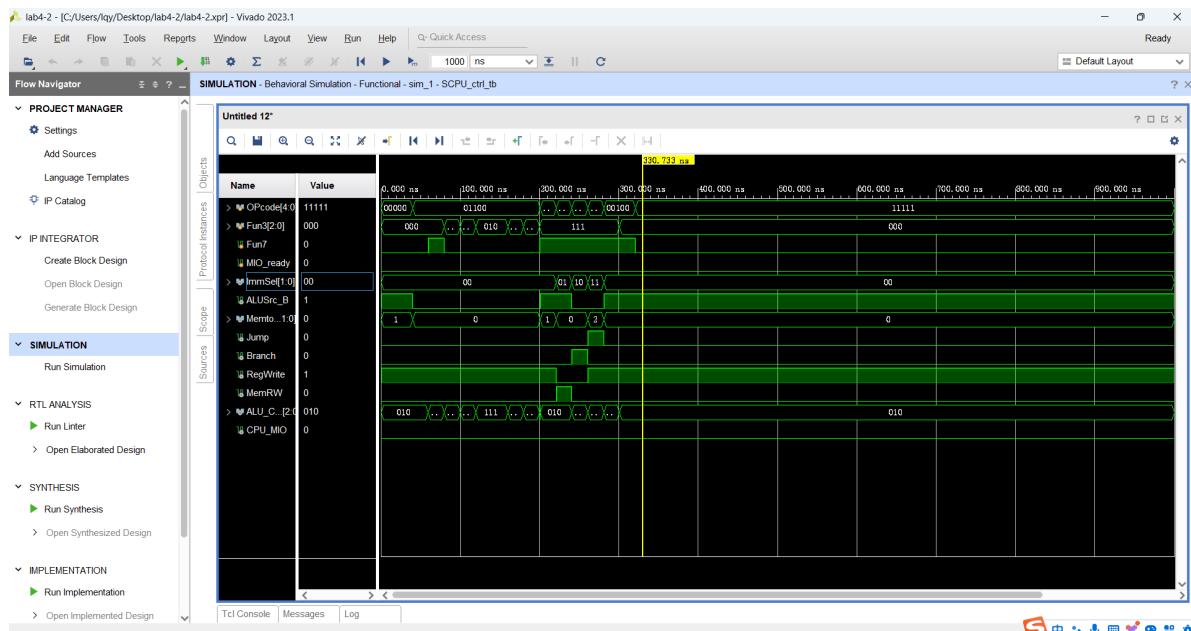
```

仿真结果

经过仔细比对PDF和自己的仿真结果，最终不断修改模块代码，以确保逻辑正确

注意到原本按照PDF上面的写法对ALU控制器进行套用PDF的模版，但是发现只使用 `assign Fun = {Fun3, Fun7};` 这一句会导致 `ALU_Control` 只会随着 `opcode` 改变而不会随着 `ALUop` 改变

因此经过一番调试，在 `assign Fun = {Fun3, Fun7};` 前面加了一句 `wire [3:0] Fun;` 才使得结果正确



CPU集成

模块实现

```

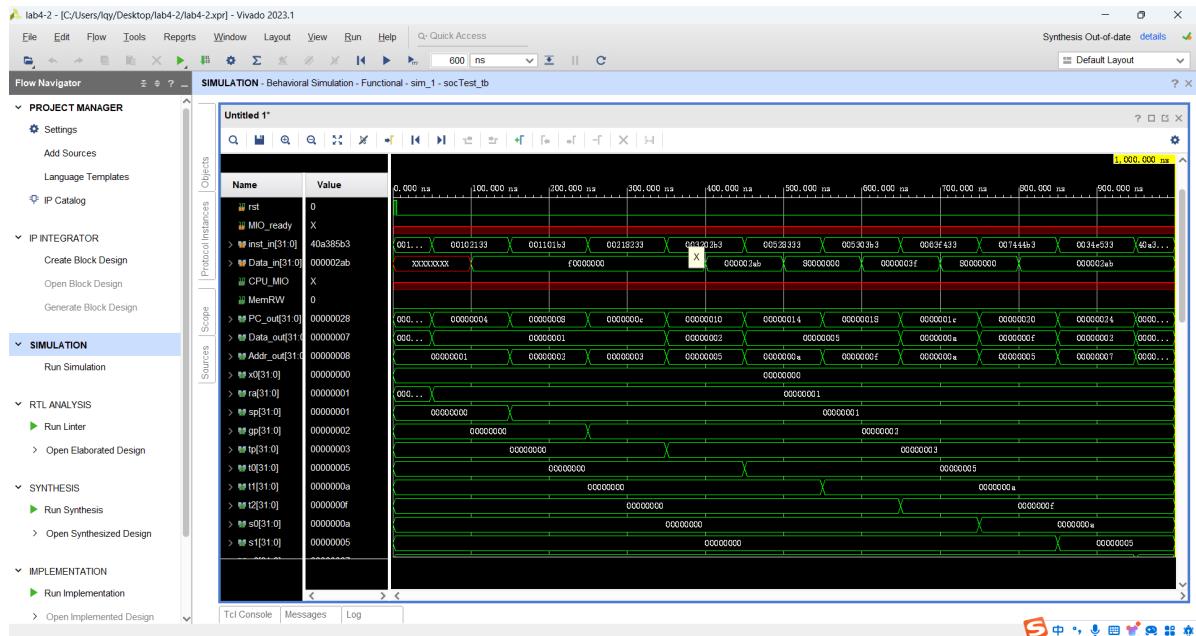
1 module socTest(
2     input clk,
3     input rst
4 );
5
6     /* SCPU 中接出 */
7     wire [31:0] Addr_out;
8     wire [31:0] Data_out;
9     wire      CPU_MIO;
10    wire      MemRW;
11    wire [31:0] PC_out;
12    /* RAM 接出 */
13    wire [31:0] douta;
14    /* ROM 接出 */
15    wire [31:0] spo;
16
17    SCPU u0(
18        /*YOUR_CODE*/
19        .Data_in(douta),
20        .MIO_ready(CPU_MIO),
21        .clk(clk),
22        .inst_in(spo),
23        .rst(rst),
24        .Addr_out(Addr_out),
25        .CPU_MIO(CPU_MIO),
26        .Data_out(Data_out),
27        .MemRW(MemRW),
28        .PC_out(PC_out)
29    );
30
31    RAM_B u1(
32        /*YOUR_CODE*/
33        .clk(~clk),
34        .wea(MemRW),
35        .addr(a(Addr_out[11:2])),
36        .dina(Data_out),
37        .douta(douta)
38    );
39
40    ROM_D u2(
41        /*YOUR_CODE*/
42        .a(PC_out[11:2]),
43        .spo(spo)
44    );
45 endmodule

```

基于原理图进行连线

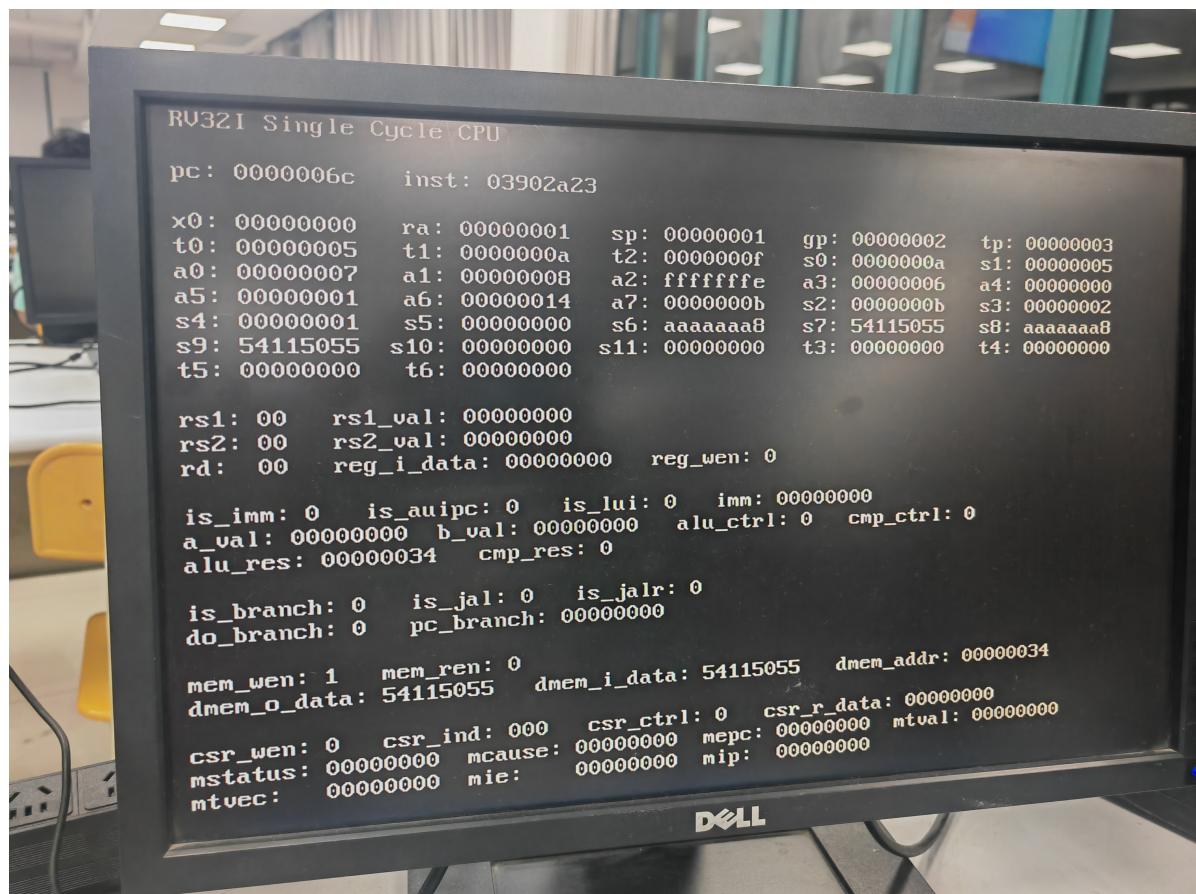
功能仿真

利用SOC平台进行功能仿真

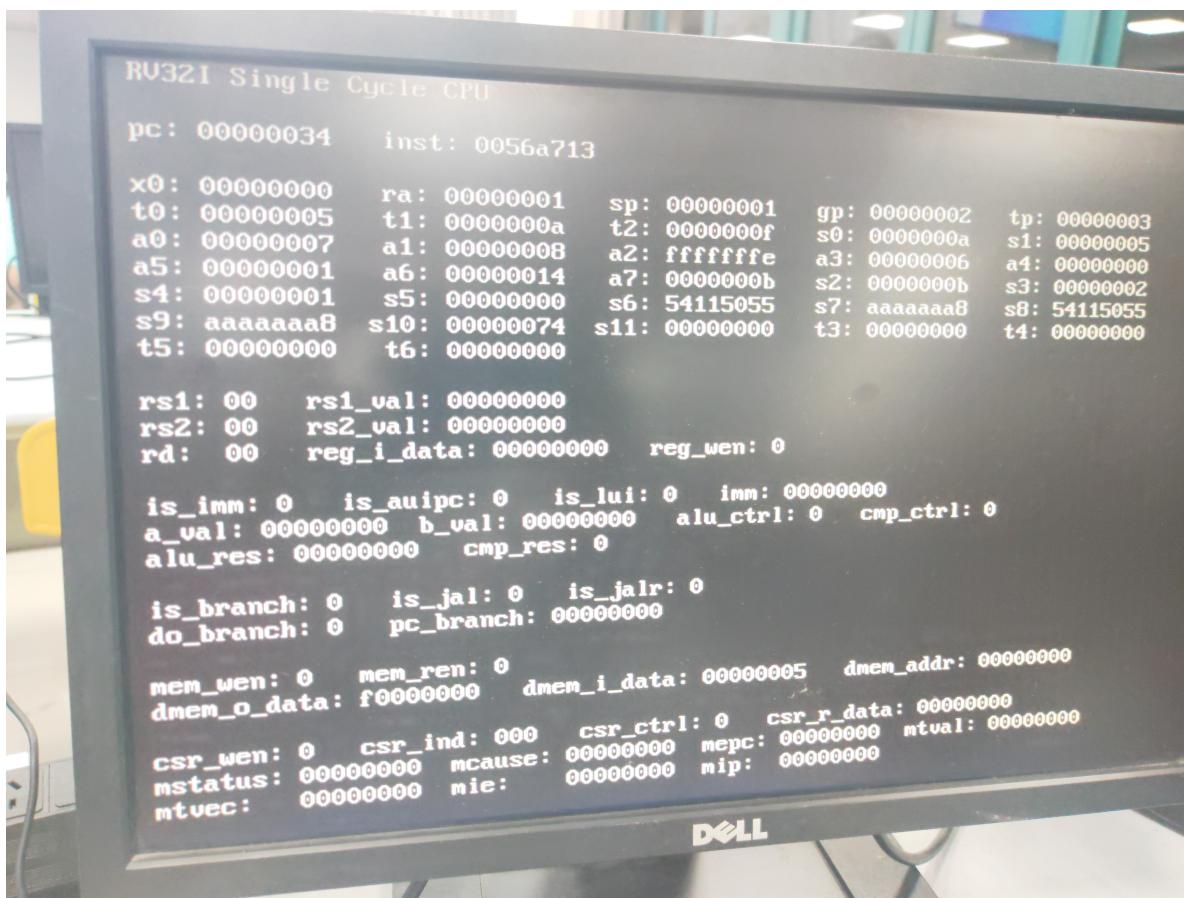


下板验证

经过检查，下板验证结果正确



特别注意到这里 s10 寄存器的值是74，符合预期结果，而且指令也完成了循环



CPU指令拓展

数据通路重实现

根据原理图进行连线

⚠️ 数据通路部分的多路选择器 MUX2T1_32 在模块定义时把控制信号错误写为32位，导致后续功能仿真时一致出现没有指令读出的情况，这个细小的问题困扰了我两天两夜之久，助教哥哥也花了一晚上才看出来问题所在 😅

```

1 `timescale 1ns / 1ps
2
3 module Data_path_more(
4     input [1:0] Jump, //J指令
5     input Branch, //Beq指令
6     input BranchN, //Bne指令
7     input ALUSrc_B, //ALU端口B输入选择
8     input [3:0] ALU_Control, //ALU操作控制
9     input clk, //寄存器时钟
10    input [1:0] MemtoReg, //Regs写入数据源控制
11    input [31:0] Data_in, //存储器输出
12    input [2:0] ImmSel, //ImmGen操作控制
13    input [31:0] inst_field, //指令数据域[31:7]
14    input rst, //寄存器复位
15    input RegWrite, //寄存器写信号
16    output wire [31:0] PC_out, //PC指针输出
17    output [31:0] ALU_out, //ALU运算输出
18    output [31:0] Data_out //CPU数据输出
19 );

```

```

20
21 // Wires Definition.
22 // ImmGen_0 wires definition.
23 wire [31:0] imm_out;
24 // add_32_0 wires definition.
25 wire [31:0] add_32_0_c;
26 // add_32_1 wires definition.
27 wire [31:0] add_32_1_c;
28 // MUX4T1_32_0 wires definition.
29 wire [31:0] MUX4T1_32_0_o;
30 // MUX2T1_32_1 wires definition.
31 wire [31:0] MUX2T1_32_1_o;
32 // MUX4T1_32_1 wires definition.
33 wire [31:0] MUX4T1_32_1_o;
34 // MUX2T1_32_0 wire definition.
35 wire [31:0] MUX2T1_32_0_o;
36 // Regs_0 wires definition.
37 wire [31:0] Rs1_data;
38 wire [31:0] Rs2_data;
39 // PC wires definition.
40 wire [31:0] Q;
41 // ALU_0 wires definition.
42 wire [31:0] res;
43 wire zero;

44
45 // Module definition.
46 // ImmGen_0
47 ImmGen ImmGen_0 (
48     // Input.
49     .ImmSel(ImmSel),
50     .inst_field(inst_field),
51     // Output.
52     .Imm_out(imm_out)
53 );
54
55 // add_32_0
56 add_32 add_32_0 (
57     // Input.
58     .a(32'h4),
59     .b(Q),
60     // Output.
61     .c(add_32_0_c)
62 );
63
64 // add_32_1
65 add_32 add_32_1 (
66     // Input.
67     .a(Q),
68     .b(imm_out),
69     // Output.
70     .c(add_32_1_c)
71 );
72
73 // MUX4T1_32_0
74 MUX4T1_32 MUX4T1_32_0 (

```

```

75      // Input.
76      .s(MemtoReg),
77      .I0(res),
78      .I1(Data_in),
79      .I2(add_32_0_c),
80      .I3(imm_out),
81      // Output.
82      .o(MUX4T1_32_0_o)
83  );
84
85  // MUX2T1_32_1
86  MUX2T1_32 MUX2T1_32_1 (
87      // Input.
88      .s((Branch & zero) | (BranchN & !zero)),
89      .I0(add_32_0_c),
90      .I1(add_32_1_c),
91      // Output.
92      .o(MUX2T1_32_1_o)
93  );
94
95  // MUX4T1_32_1
96  MUX4T1_32 MUX4T1_32_1 (
97      // Input.
98      .s(Jump),
99      .I0(MUX2T1_32_1_o),
100     .I1(add_32_1_c),
101     .I2(res),
102     .I3(MUX2T1_32_1_o),
103     // Output.
104     .o(MUX4T1_32_1_o)
105  );
106
107 // MUX2T1_32_0
108 //assign MUX2T1_32_0_o = (ALUSrc_B ? imm_out : Rs2_data);
109 MUX2T1_32 MUX2T1_32_0 (
110     // Input.
111     .s(ALUSrc_B),
112     .I0(Rs2_data),
113     .I1(imm_out),
114     // Output.
115     .o(MUX2T1_32_0_o)
116  );
117
118 // Regs_0
119 Regs Regs_0 (
120     // Input.
121     .clk(clk),
122     .rst(rst),
123     .Rs1_addr(inst_field[19:15]),
124     .Rs2_addr(inst_field[24:20]),
125     .wt_addr(inst_field[11:7]),
126     .wt_data(MUX4T1_32_0_o),
127     .RegWrite(RegWrite),
128     // Output.
129     .Rs1_data(Rs1_data),

```

```

130      .Rs2_data(Rs2_data)
131  );
132
133  // PC
134  REG32 PC (
135      // Input.
136      .clk(clk),
137      .rst(rst),
138      .CE(1'b1),
139      .D(MUX4T1_32_1_o),
140      // Output.
141      .Q(Q)
142  );
143
144  // ALU_0
145  ALU ALU_0 (
146      // Input.
147      .A(Rs1_data),
148      .B(MUX2T1_32_0_o),
149      .ALU_operation(ALU_Control),
150      // Output.
151      .res(res),
152      .zero(zero)
153  );
154
155  // Output.
156  assign PC_out = Q;
157  assign ALU_out = res;
158  assign Data_out = Rs2_data;
159
160 endmodule

```

控制器重实现

模块代码

根据对应的控制信号表将对应信号写出来

中间标注为注释的一些指令是留存带扩展的指令，未实现

```

1 `timescale 1ns / 1ps
2
3 module SCPU_ctrl_more(
4     input [4:0] Opcode, //Opcode
5     input [2:0] Fun3, //Function
6     input Fun7, //Function
7     input MIO_ready, //CPU wait
8     output reg [2:0] ImmSel,
9     output reg ALUSrc_B,
10    output reg [1:0] MemtoReg,
11    output reg [1:0] Jump,
12    output reg Branch,
13    output reg BranchN,
14    output reg Regwrite,
15    output reg MemRW,

```

```

16     output reg [3:0] ALU_Control,
17     output CPU_MIO
18 );
19
20 `define CPU_ctrl_signals {ImmSel, ALUSrc_B, MemtoReg, RegWrite, MemRW,
Branch, BranchN, Jump, ALU_Control}
21 always@(*) begin
22     case(OPcode)
23         5'b01100: begin // R-type
24             case({Fun3, Fun7})
25                 4'b0000: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0010}; end // add
26                 4'b0001: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0110}; end // sub
27                 4'b0010: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1110}; end // sll
28                 4'b0100: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0111}; end // slt
29                 4'b0110: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1001}; end // sltu
30                 4'b1000: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1100}; end // xor
31                 4'b1010: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1101}; end // sr
32                 4'b1011: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1111}; end // sra
33                 4'b1100: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0001}; end // or
34                 4'b1110: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0000}; end // and
35             default: begin `CPU_ctrl_signals = {3'b010, 1'b0, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0010}; end
36         endcase
37     end
38     5'b00000: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b01, 1'b1,
1'b0, 1'b0, 1'b0, 2'b00, 4'b0010}; end // I-type, lw
39     5'b00100: begin // I-type, ALU
40         case(Fun3)
41             3'b000: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0010}; end // addi
42             3'b010: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0111}; end // slti
43             3'b011: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1001}; end // sltiu
44             3'b100: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1100}; end // xor
45             3'b110: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0001}; end // ori
46             3'b111: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b0000}; end // andi
47             3'b001: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b00,
1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1110}; end // slli
48             3'b101: begin
49                 case(Fun7)

```

```

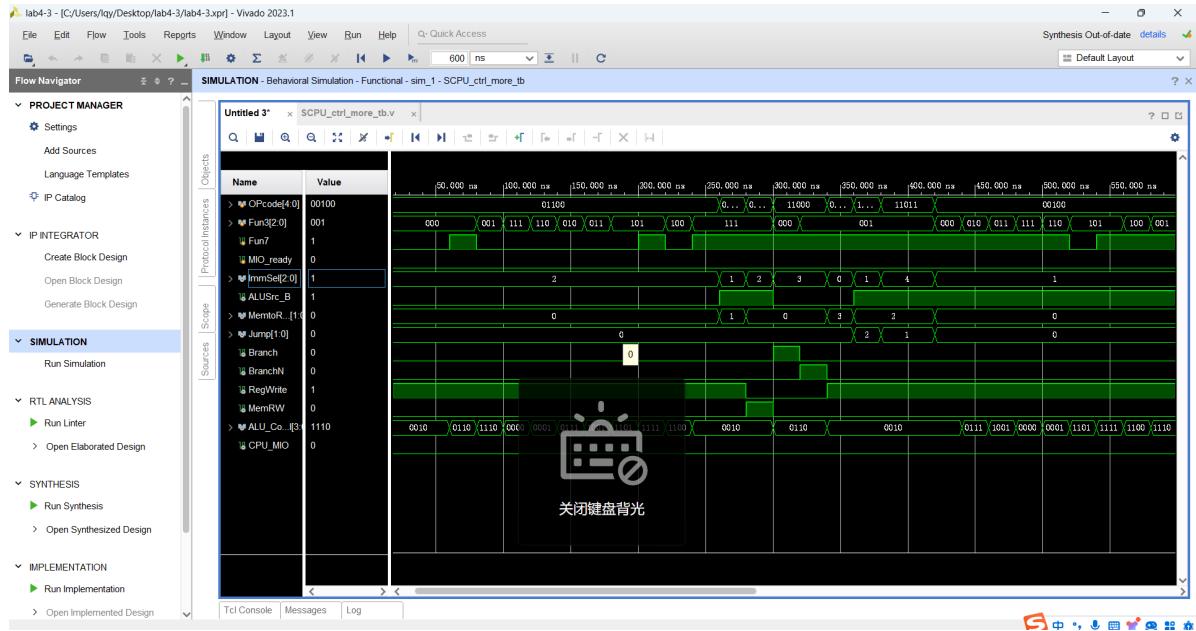
50                     1'b0: begin `CPU_ctrl_signals = {3'b001, 1'b1,
51                         2'b00, 1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1101}; end // srl
52                         1'b1: begin `CPU_ctrl_signals = {3'b001, 1'b1,
53                             2'b00, 1'b1, 1'b0, 1'b0, 1'b0, 2'b00, 4'b1111}; end // srai
54                             endcase
55                         end
56                     endcase
57                     5'b11001: begin `CPU_ctrl_signals = {3'b001, 1'b1, 2'b10, 1'b1,
58                         1'b0, 1'b0, 1'b0, 2'b10, 4'b0010}; end // I-type, jalr
59                     5'b01000: begin // S-type
60                         case(Fun3)
61                             //3'b000: begin `CPU_ctrl_signals =
62                             16'b010_1_xx_0_1_0_0_00_0010; end // sb
63                             //3'b001: begin `CPU_ctrl_signals =
64                             16'b010_1_xx_0_1_0_0_00_0010; end // sh
65                             3'b010: begin `CPU_ctrl_signals = {3'b010, 1'b1, 2'b00,
66                             1'b0, 1'b1, 1'b0, 1'b0, 2'b00, 4'b0010}; end // sw
67                             default: begin `CPU_ctrl_signals = {3'b010, 1'b1, 2'b00,
68                             1'b0, 1'b1, 1'b0, 1'b0, 2'b00, 4'b0010}; end
69                             endcase
70                         end
71                     5'b11000: begin // B-type
72                         case(Fun3)
73                             3'b000: begin `CPU_ctrl_signals = {3'b011, 1'b0, 2'b00,
74                             1'b0, 1'b0, 1'b1, 1'b0, 2'b00, 4'b0110}; end // beq
75                             3'b001: begin `CPU_ctrl_signals = {3'b011, 1'b0, 2'b00,
76                             1'b0, 1'b0, 1'b0, 1'b1, 2'b00, 4'b0110}; end // bne
77                             //3'b100: begin `CPU_ctrl_signals =
78                             16'b011_0_xx_0_x_1_0_00_0111; end // blt
79                             //3'b101: begin `CPU_ctrl_signals =
80                             16'b011_0_xx_0_x_1_0_00_0111; end // bge
81                             //3'b110: begin `CPU_ctrl_signals =
82                             16'b011_0_xx_0_x_1_0_00_1001; end // bltu
83                             //3'b111: begin `CPU_ctrl_signals =
84                             16'b011_0_xx_0_x_1_0_00_1001; end // bgeu
85                             default: begin `CPU_ctrl_signals = {3'b011, 1'b0, 2'b00,
86                             1'b0, 1'b0, 1'b1, 1'b0, 2'b00, 4'bxxxx}; end
87                             endcase
88                         end
89                     5'b11011: begin `CPU_ctrl_signals = {3'b100, 1'b1, 2'b10, 1'b1,
90                         1'b0, 1'b0, 1'b0, 2'b01, 4'b0010}; end // J-type, jal
91                     5'b00101: begin `CPU_ctrl_signals = {3'b000, 1'bx, 2'b11, 1'b1,
92                         1'bx, 1'b0, 1'b0, 2'b00, 4'bxxxx}; end // U-type, auipc
93                     5'b01101: begin `CPU_ctrl_signals = {3'b000, 1'b0, 2'b11, 1'b1,
94                         1'b0, 1'b0, 1'b0, 2'b00, 4'b0010}; end // U-type, lui
95                     default: begin `CPU_ctrl_signals = {3'bxxx, 1'bx, 2'bxx, 1'bx,
96                         1'bx, 1'bx, 1'bx, 2'bxx, 4'bxxxx}; end
97                     endcase
98                 end
99             assign CPU_MIO = MIO_ready;
100
101 endmodule

```

功能仿真

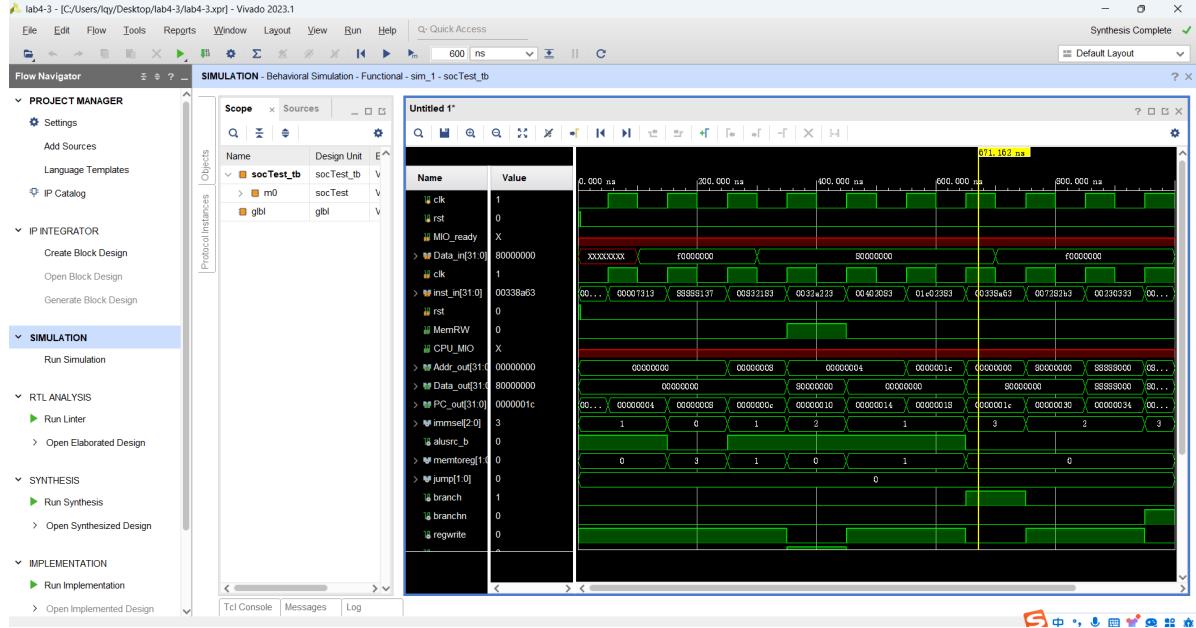
根据提供的正确仿真结果写出仿真代码，进行仿真后得到如下结果

原本对控制器参数表中未定义的控制信号全部赋值 X，后来根据仿真图将其修改为对应的默认值

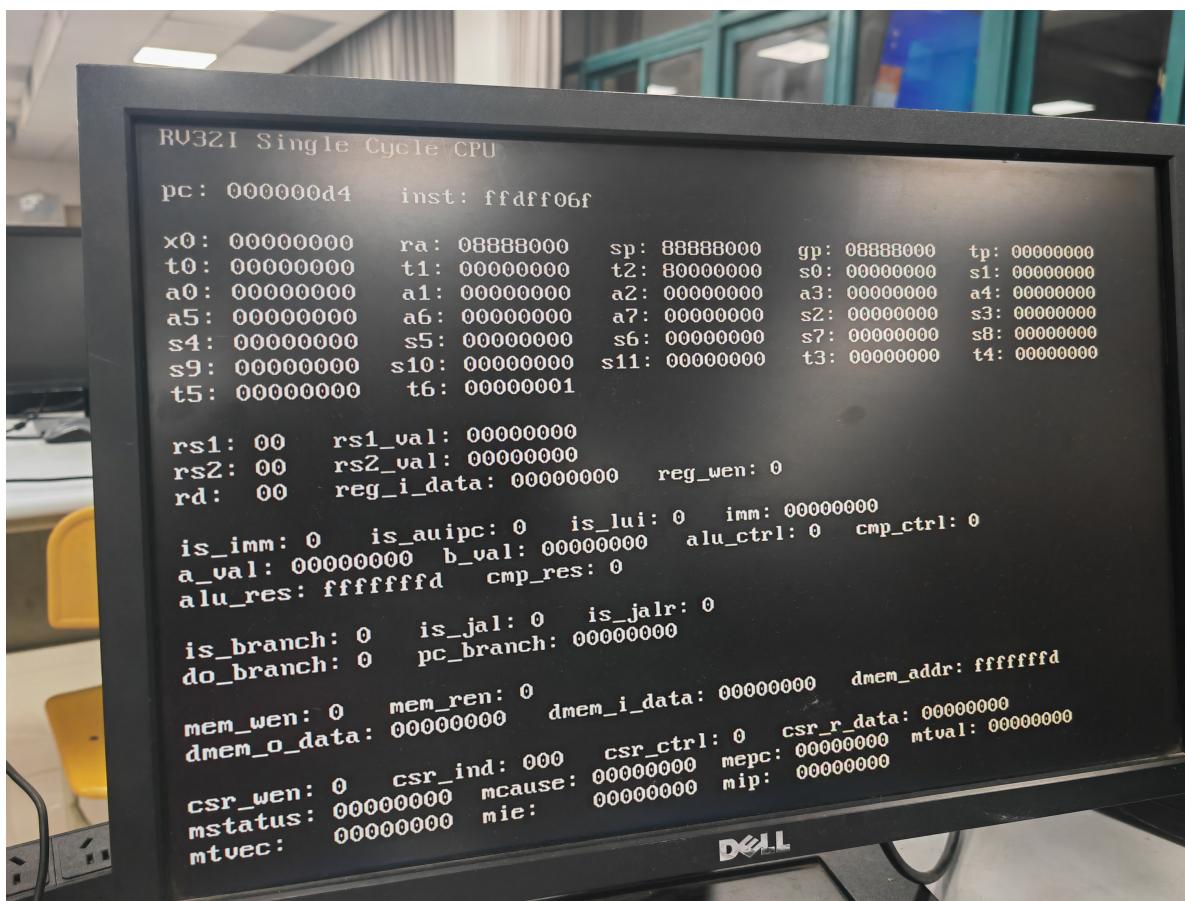


SOC平台仿真

通过SOC平台进行SCPU模块的仿真，结果如下：



下板验证



经过助教验收，但是注意到最后跳转到 error 部分，原因是在 PC=0x20 的 gp 寄存器应该是 0x80000000，仿真的结果也是这个，但是下板时VGA上的显示为 0x08888000，助教说是时序的问题，其他同学也有遇到，最后给了验收通过

CPU指令中断

这一部分不太会写

数据通路修改

控制器修改

下板验证