模型机设计报告

班级 信安 2201 姓名 高铭辰 学号 202208060115

一、设计目的

完整、连贯地运用《电路与电子学》所学到的电路、模电和数电知识，熟练掌握现代 EDA 工具基本使用方法，为后续课程学习和今后从事相关工作打下良好的基础或做下一些铺垫。

二、设计内容

1. **算术单元 (au.v)**：负责执行所有算术运算，如加法和减法。这是计算机的核心部分之一，处理数值计算任务。
2. **控制信号 (con_signal.v) 和 指令解码器 (ins_decode.v)**：这两个模块共同构成控制流。指令解码器解析输入的指令并确定其含义，而控制信号模块根据这些指令生成相应的控制信号来指挥其他部分的操作。
3. **程序计数器 (pc.v) 和 指令寄存器 (ir.v)**：程序计数器负责追踪程序执行的位置，指示下一条执行的指令。指令寄存器存储当前执行的指令，确保在执行期间指令的可用性。
4. **寄存器组 (reg_group.v)**：存储运算过程中的数据和中间结果。这是数据存储和访问的关键部分。
5. **程序状态字 (psw.v)**：记录程序执行过程中的状态信息，如条件标志等，对程序的控制和分支决策至关重要。
6. **多路选择器 (mux2_1.v 和 mux3_1.v)**：根据控制信号选择正确的数据和控制路径，确保正确的数据和信号被送到计算机的合适部分。

三、详细设计

3.1 模型机的设计思路

1. **控制流**：由 `con_signal.v` 和 `ins_decode.v` 模块构成，这一部分是系统的大脑。`ins_decode.v`（指令解码器）负责解读输入的机器指令，确定它们代表的操作（如加法或减法）。然后，`con_signal.v`（控制信号）根据这些解析出的指令生成相应的控制信号，这些信号指示系统的其他部分如何响应这些指令，例如激活算术单元进行计算或将数据写入寄存器。
2. **数据流**：数据在 `au.v`（算术单元）、`reg_group.v`（寄存器组）和 `psw.v`（程序状态字）之间流动。`au.v` 处理所有的算术运算，如加法和减法。`reg_group.v` 存储操作数和运算结果，而 `psw.v` 记录并更新程序的状态，如运算后的标志位（比如零标志或溢出标志）。
3. **程序执行控制**：`pc.v`（程序计数器）和 `ir.v`（指令寄存器）共同管理程序的执行流。`pc.v` 跟踪下一条要执行的指令的位置，确保程序按顺序执行。与此同时，`ir.v` 存储当前正在执行的指令，保证指令在整个执行周期内可用。
4. **信号路由**：`mux2_1.v` 和 `mux3_1.v`（多路选择器）负责根据控制信号选择正确的数据和控制路径。它们确保正确的数据和指令被送到系统的正确部分，以执行特定的操作。

3.2 模型机的整体架构

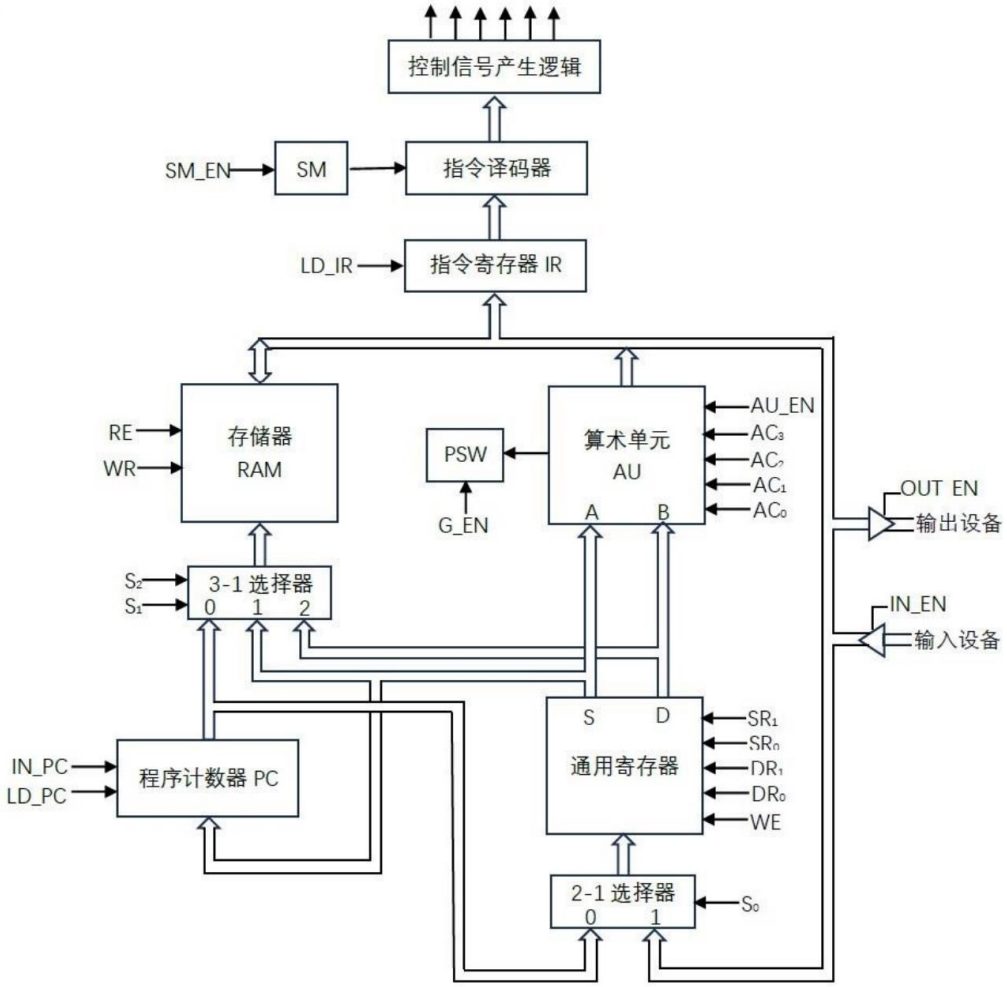
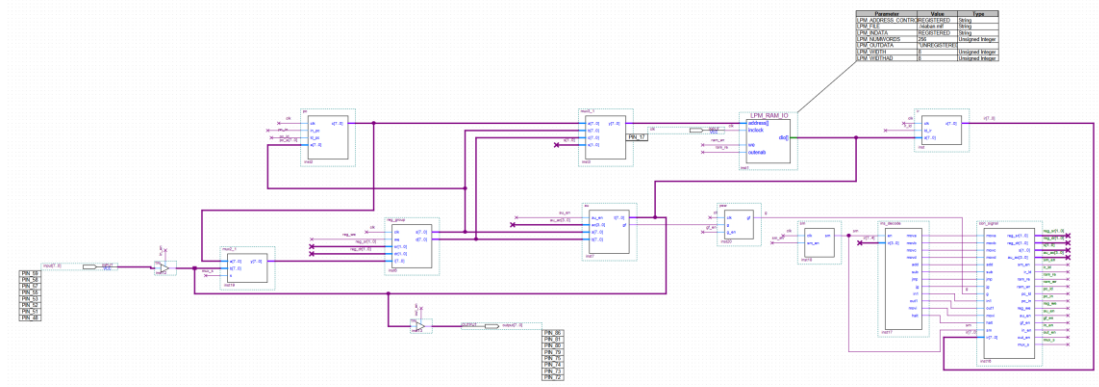


图 1　模型机结构框架

3.3 各模块的具体实现

（此部分必须各模块功能、接口设计以及功能实现）

整体架构



3.3.1 程序计数器 PC

(1)模块功能

存放当前欲执行指令在 RAM 中的存储地址。LD_PC=1 时，PC 装载跳转地址，IN_PC=1 时，PC 进行自加 1 操作。

(2)接口设计

```
input clk, in_pc, ld_pc,
input [7:0] a,
output reg [7:0] c
```

(3)功能实现

```
initial begin
    c = 8'b00000000;
end

always @(negedge clk) begin
    if (in_pc && ~ld_pc)
        c <= c + 1'b1;
    else if (~in_pc && ld_pc)
        c <= a;
end
```

3.3.2 3-1 选择器

(1)模块功能

选择 RAM 的地址来源,其有 3 个输入，每个输入 8 位,分别接程序计数器 PC 和通用寄存器的 S 口和 D 口，由控制信号 S2、S1 选择其中一个传送至 RAM 的地址
输入端。

(2)接口设计

```
input [7:0] a,
input [7:0] b,
input [7:0] c,
input [1:0] s,
output reg [7:0] y
```

(3)功能实现

```
always @*
begin
  case (s)
    2'b00: y = a;
    2'b01: y = b;
    2'b10: y = c;
    default: y = a;
  endcase
end
```

3.3.3 指令寄存器 IR

(1)模块功能

存放当前执行的指令

(2)接口设计

```
input clk, ld_ir,
input [7:0] a,
output reg [7:0] x
```

(3)功能实现

```
initial begin
x = 8'b00000000;
```

```
end
always @(negedge clk) begin
    if (ld_ir)
        x <= a;
end
```

3.3.4 SM

(1)模块功能

指示当前周期是取指周期还是执行周期。SM=0，表示当前是取指周期，SM=1，表示当前是执行周期。

(2)接口设计

```
input clk, sm_en,
output reg sm
```

(3)功能实现

```
initial begin
sm = 1'b0;
end

always @(negedge clk) begin
    if (sm_en)
        sm <= ~sm;
end
```

3.3.5 指令译码器

(1)模块功能

解析指令。根据指令寄存器 IR 提供的 8 位指令编码，解析是哪条指令。

(2)接口设计

```
input wire en;
input wire [3:0] ir;
output mova, movb, movc, movd, add, sub, jmp, jg, in1, out1, movi, halt;
reg mova, movb, movc, movd, add, sub, jmp, jg, in1, out1, movi, halt;
```

(3)功能实现

```
always @* begin
if (en) begin
   mova = ir == 4'b0100;
   movb = ir == 4'b0101;
   movc = ir == 4'b0110;
   movd = ir == 4'b0111;
   add  = ir == 4'b1000;
   sub  = ir == 4'b1001;
   jmp  = ir == 4'b1010;
   jg   = ir == 4'b1011;
   in1  = ir == 4'b1100;
   out1 = ir == 4'b1101;
   movi = ir == 4'b1110;
   halt = ir == 4'b1111;
end
else begin
   {mova, movb, movc, movd, add, sub, jmp, jg, in1, out1, movi, halt} = 12'b0;
end
end
```

3.3.6 控制信号产生逻辑

(1)模块功能

根据指令译码器解析的指令，产生该指令执行所需的控制信号。

(2)接口设计

```
input mova, movb, movc, movd, add, sub, jmp, jg, g, in1, out1, movi, halt, sm,
input [7:0] ir,
output reg [1:0] reg_sr, reg_dr, s,
output reg [3:0] au_ac,
output reg sm_en, ir_ld, ram_re, ram_wr, pc_ld, pc_in, reg_we, au_en, gf_en, in_en, out_en,
mux_s
```

(3)功能实现

```
always @(*) begin

    sm_en = ~halt;
    ir_ld = ~sm;
    au_ac = ir[7:4];
    reg_sr = ir[1:0];
    reg_dr = ir[3:2];
    in_en = in1;
    out_en = out1;
    gf_en = sub;
    au_en = add | sub | mova | movb | out1;
    mux_s = mova | movc | movi | add | sub | in1;
    ram_wr = movb;
    ram_re = (~sm) | movc | movi;

    pc_ld = jmp | (jg & g);
    pc_in = (~sm) | movi;
    reg_we = mova | movc | movd | movi | add | sub | in1;

    if (sm && movb) s = 2'b10;
    else if (sm && movc) s = 2'b01;
    else s = 2'b00;
end
```

3.3.7 2-1 选择器

(1)模块功能

选择通用寄存器的数据来源。

(2)接口设计

```
input [7:0] a,
input [7:0] b,
input s,
output reg [7:0] y
```

(3)功能实现

```
always @(a, b, s)
begin
   if (s == 0)
      y = a;
   else
      y = b;
end
```

3.3.8 通用寄存器

(1)模块功能

保存操作数和中间计算结果，其有 4 个 8 位寄存器 R0、R1、R2、R3，控制信号 SR1、SR0 选择 4 个寄存器中的一个作为源寄存器，源寄存器的数据从 S 口输出；DR1、DR0 选择 4 个寄存器中的一个作为目的寄存器，目的寄存器的数据从 D 口输出，WE 是写控制信号，在 DR1、DR0 的配合下，将其输入端的数据写入目的寄存器。

(2)接口设计

```
input clk, we,
input [1:0] sr, dr,
input [7:0] i,
output reg [7:0] s, d
```

(3)功能实现

```
always @(sr or dr or R0 or R1 or R2 or R3)
   begin
      case (sr)
            2'b00: s = R0;
            2'b01: s = R1;
            2'b10: s = R2;
            default: s = R3;
      endcase

      case (dr)
            2'b00: d = R0;
            2'b01: d = R1;
            2'b10: d = R2;
            default: d = R3;
      endcase
   end

always @(negedge clk)
   begin
      case({we,dr})
          3'b100: R0 <= i;
          3'b101: R1 <= i;
          3'b110: R2 <= i;
          3'b111: R3 <= i;
          default: ;
      endcase
   end
```

3.3.9 算术单元 AU

(1)模块功能

实现算术运算，并在执行 MOVA、MOVB、OUT 指令时负责将数据送至总线 BUS。

(2)接口设计

```
input au_en;
input [3:0] ac;
input [7:0] a;
input [7:0] b;
output [7:0] t;
output gf;
reg [7:0] t;
reg gf;
```

(3)功能实现

```
always @(*)
   begin
      t = 8'b00000000;
      gf = 1'b0;
      if (au_en == 1'b0)
         begin
         t = 8'hZZ;
         end
      else
         begin
         case (ac)
            4'b1000: begin
               t = a + b;
            end
            4'b1001: begin
               t = b - a;
               if(b[7]==1'b0&&a[7]==1'b1) gf = 1'b1;
               else if(b[7]==1'b1&&a[7]==1'b0) gf = 1'b0;
               else if(b>a) gf = 1'b1;
               else gf= 1'b0;
            end
            4'b0100, 4'b0101, 4'b1101: begin
               t = a;
            end
            default: begin
               t = 8'hZZ;
            end
         endcase
         end
   end
```

3.3.10 状态寄存器 PSW

(1)模块功能

存放 SUB 指令产生的状态位 G

(2)接口设计

```
input clk, g, g_en,
output reg gf
```

(3)功能实现

```
initial begin
      gf = 1'b0;
end


always @(negedge clk) begin
      if (g_en)
            gf <= g;
end
```

四、系统测试

4.1 测试环境

Quartus II 9.0sp1 Web Edition
Cyclone II：EP2C5T144C8

4.2 测试代码

（使用模型机实现的指令编写一至两个程序测试模型机的正确性。）

测试代码一：

通用寄存器组：R3 初始化为 07h，R0 初始化为 01h

RAM 的 mif 文件存放的内容：

RAM 地址	汇编指令	机器码 (RAM 存放内容)	说明
00	JMP	1010 0011	R3 的初始值为 07h，指令执行完，PC=07h
01		0001 0001	R0 的初始值为 01h，RAM 的 01 地址单元存放 11h
07	IN R1	1100 0100	外部引脚输入 01h，指令执行完，R1=01h
08	MOVA R2, R1	0100 1001	指令执行完，R2=01h
09	MOVC R1, M	0110 0100	RAM 的 01 单元内容 11h 赋给 R1，指令执行完，R1=11h
10	SUB R1, R2	1001 0110	指令执行完，R1=10h 且 G=1
11	MOVD R3, PC	0111 1100	取指后 PC 自加 1，此时 PC=12，那么 R3=12=0Ch
12	ADD R3, R1	1000 1101	指令执行完，R3=1Ch=28
13	JG	1011 0011	G=1，满足跳转条件，执行跳转，PC=28
28	MOVB M, R3	0101 0011	将 R3 中的数据写入 RAM
29	MOVC R1, M	0110 0100	将 RAM 的数据写入 R1，两条指令执行完，R1=R3=1Ch
30	MOVI #9	1110 0000	此为两字节指令，指令执行完后，R0=09H
31		0000 1001	
32	SUB R0, R1	1001 0001	指令执行完，R0=EDh 且 G=0
33	JG	1011 0011	G=0，不满足跳转条件，不执行跳转
34	OUT R0	1101 0000	指令执行完，输出引脚为 EDh
35	HALT	1111 0000	停机指令
36	JMP	1010 0011	此指令在停机指令后用于检测停机是否正确，正确的停机指令应该是停机后面的指令不执行

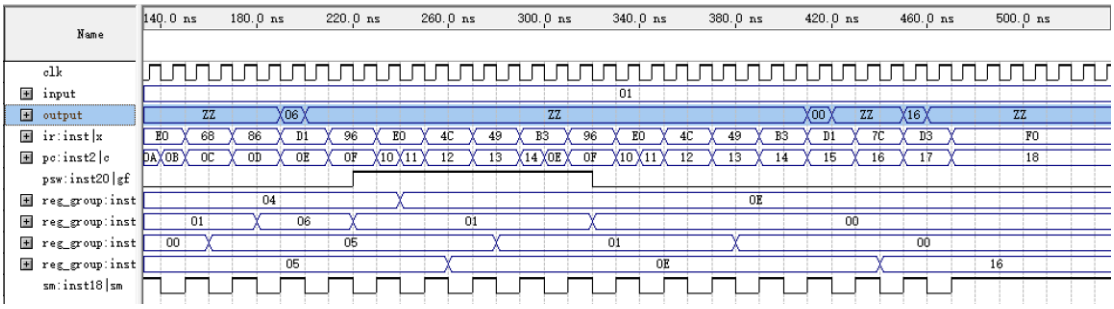
测试代码二：

	RAM地址	汇编指令	机器码(RAM存放内容)	说明
2	00	MOVI #8	11100000	R0=05H
3	01		00000101	
4	02	MOVA R3,R0	01001100	R3=05H
5	03	JMP	10100011	PC=05H
6	04		00000000	00H||08H
7	05	MOVI #4	11100000	R0=04H
8	06		00000100	
9	07	MOVB M,R3	01010011	R3写入RAM
10	08	IN R1	11000100	输入为01H,R1=01H
11	09	MOVI #4	11100000	R0=04H
12	10		00000100	
13	11	MOVC R2,M	01101000	R2=02H
14	12	ADD R1,R2	10000110	R1=06H
15	13	OUT R1	11010001	输出06H
16	14	SUB R1,R2	10010110	R1=01H,G=1||R1=0,G=0
17	15	MOVI #14	11100000	R0=0EH
18	16		00001110	
19	17	MOVA R3,R0	01001100	R3=0EH
20	18	MOVA R2,R1	01001001	R2=01H||R2=0
21	19	JG	10110011	G=1,满足,PC=0EH||G=0,不满足
22	20	OUT R1	11010001	输出0
23	21	MOVD R3,PC	01111100	R3=16H
24	22	OUT R3	11010011	输出16H
25	23	HALT	11110000	停机
26	24	OUT R3	11010011	应无变化,sm不变

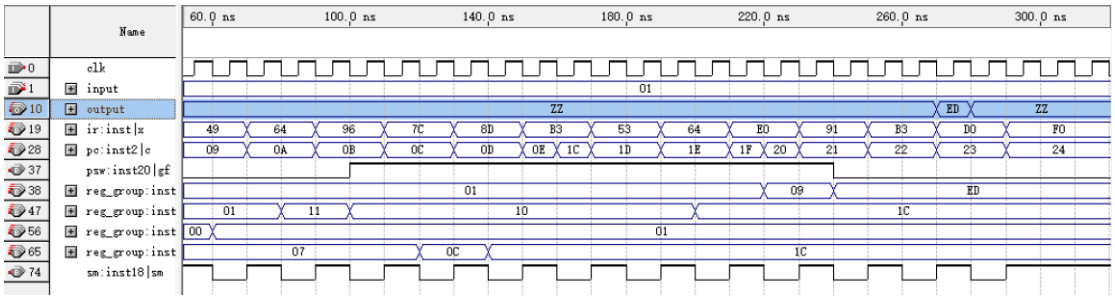
4.3 测试结果

(为结果显示完全，故用 16 进制显示输出)

测试结果一：

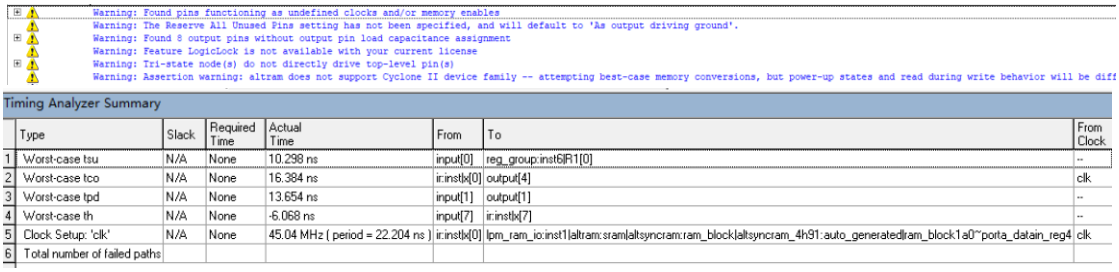


测试结果二：



4.4 模型机性能分析

Flow Status	Successful - Thu Dec 21 19:59:33 2023
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	Gcpu
Top-level Entity Name	Gcpu
Family	Cyclone II
Device	EP2C5T144C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	154 / 4,608 (3 %)
Total combinational functions	153 / 4,608 (3 %)
Dedicated logic registers	50 / 4,608 (1 %)
Total registers	50
Total pins	17 / 89 (19 %)
Total virtual pins	0
Total memory bits	2,048 / 119,808 (2 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)



	Type	Slack	Required Time	Actual Time	From	To	From Clock
1	Worst-case tsu	N/A	None	10.298 ns	input[0]	reg_group:inst6|R1[0]	--
2	Worst-case tco	N/A	None	16.384 ns	ir:inst|x[0]	output[4]	clk
3	Worst-case tpd	N/A	None	13.654 ns	input[1]	output[1]	--
4	Worst-case th	N/A	None	-6.068 ns	input[7]	ir:inst|x[7]	--
5	Clock Setup: 'clk'	N/A	None	45.04 MHz (period = 22.204 ns)	ir:inst|x[0]	lpm_ram_io:inst1|altram:sram|altsyncram:ram_block|altsyncram_4h91:auto_generated|ram_block1a0~porta_datain_reg4	clk
6	Total number of failed paths						

五、 实验总结、必得体会及建议

5.1 从需要掌握的理论、遇到的困难、解决的办法以及经验教训等方面进行总结。

（1）掌握的理论知识：我已经对简易模型机的内部构造和运行机制有了初步的了解，包括熟悉指令寄存器、状态寄存器、指令计数器以及寄存器的工作方式。同时，我也学会了如何用 Verilog 语言来编写电路设计。

（2）遇到的挑战：由于是首次尝试这项技能，对于各个模块的融合操作还不够熟练，给我带来了一定的困难。特别是在一次性尝试搭建完整系统时，往往难以确定错误的具体位置。

（3）解决策略：我采取了逐步迭代的方法来构建电路，即先搭建其中一部分，然后通过几个输入引脚来验证输出是否符合预期。这样做可以帮助我及时发现并纠正错误。对于其他问题，我通过上网搜索相关资料、询问同学并分析报告，从而找出电路的问题。在遇到不懂的地方时，我还向老师求助，不仅解决了问题，还学到了很多方法和技巧。

（4）学习心得：在电子电路学习中，动手实践非常重要。光靠理论学习是不够的，必须亲自使用软件进行仿真实验，这样才能更深入地理解课程内容。

5.2 对本实验内容、过程和方法的改进建议（可选项）。

无