模型机设计报告

班级 信安 2101 班 姓名 孙照海 学号 202109070105

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

按给定的数据格式和指令系统，使用 EDA 工具设计一台用硬连线逻辑控制的简易计算机系统。

要求灵活运用各方面知识，使得所设计的计算机系统具有较佳的性能。

对所做设计的性能指标进行分析，整理出设计报告。

三、详细设计

3.1 设计的整体架构

1）数据字采用 8 位二进制定点补码表示，其中最高位（第 7 位）为符号位，小数点可视为最左或最右，其数值表示范围分别为：-1≤X＜+1 或-128≤X＜+127。指令的高 4 位为操作码，低 4 位分别用 2 位表示目的寄存器和源寄存器的编号，或表示寻址方式。共有 2 种寻址方式。

2）指令系统有 16 条指令，具体格式见指令系统表。应该指出的是，各条指令的编码形式可以多种多样。为了叙述方便，下面采用汇编符号对指令进行描述，其中 R1 和 R2 分别表示“目标”和“源”寄存器，M 表示地址在寄存器 C 中的存贮单元。

表1 指 令 系 统 表

汇编符号	功能	编码
MOV R1, R2	(R2) → R1	1100 R1 R2
MOV M, R2	(R2) → (C)	1100 11 R2
MOV R1, M	(C) →R1	1100 R1 11
ADD R1, R2	(R1) + (R2) → R1	1001 R1 R2
SUB R1, R2	(R1) - (R2) → R1	0110 R1 R2
AND R1, R2	(R1) & (R2) → R1	1011 R1 R2
NOT R1	/(R1) → R1	0101 R1 XX
RSR R1	(R1) 循环右移一位→ R1	1010 R1 00
RSL R1	(R1) 循环左移一位→ R1	1010 R1 11
JMP add	add → PC	0011 00 00, address
JZ add	结果为 0 时 add → PC	0011 00 01, address
JC add	结果有进位时 add → PC	0011 00 10, address
IN R1	(开关 7-0) → R1	0010 R1 XX
OUT R1	(R1) → 发光二极管 7-0	0100 R1 XX
NOP		0111 00 00
HALT	停机	1000 00 00

3）计算机的工作过程可以看作是许多不同的数据流和控制流在机器各部分之间的流动，数据流所经过的路径称作机器的数据通路。数据通路不同，指令执行所经过的操作过程就不同，机器的结构也就不一样。如何设计一个好的数据通

路已经超出了本课程的范围，在此我们不予讨论。我们假设所设计的计算机的数据通路如图 1 所示。

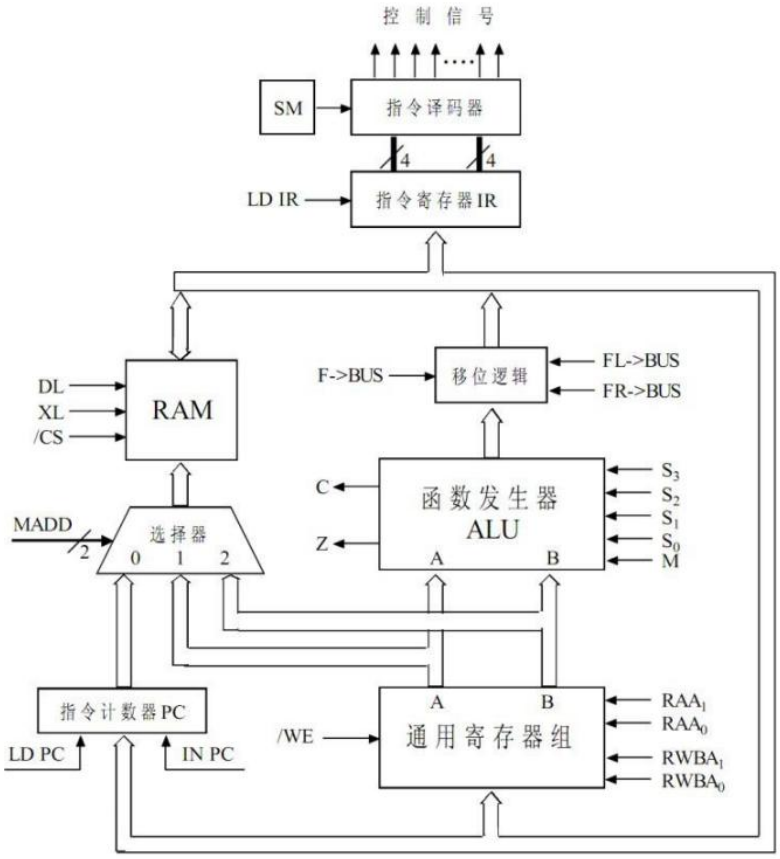


图 1　模型机数据通路

4）指令寄存器 IR 接收到一条机器指令后，这条指令就被译码执行。指令通过译码产生出的各种控制信号在时钟信号的配合下控制着指令执行的全过程。为此，需要将执行每条指令所需的全部基本微操作的控制信号罗列出来，进行综合分析、化简，并落实到不同的周期、节拍之中，然后用各种逻辑门电路实现。以下是所用基本控制信号列表。

表 2　基本控制信号及功能表

序号	信号	功能
1	LD PC	当 LD PC=1 时，将 BUS 上的数据写入 PC
2	IN PC	当 IN PC=1 时，PC 进行自加 1 操作
3	MADD	选择 RAM 地址来源。00：指令计数器 PC，01：A 口，10：B 口

4	XL	对存储器 RAM 进行写操作
5	DL	对存储器 RAM 进行读操作
6	LD IR	允许把 BUS 上的数据写入指令寄存器 IR
7	RAA1、RAA0	选择寄存器 A、B、C 中的一个作为源寄存器
8	RWBA1、RWBA0	选择寄存器 A、B、C 中的一个作为目的寄存器
9	/WE	允许把 BUS 上的数据写入通用寄存器组，低电平有效
10	M	M=1，ALU 进行算术逻辑运算；M=0，ALU 进行数据传输
11	S3～S0	控制 ALU 执行不同的操作
12	F→BUS	将移位逻辑的输入数据直接传至输出
13	FL→BUS	移位逻辑的输入数据循环左移一位传至输出且 D7 送至 C_f
14	FR→BUS	移位逻辑的输入数据循环右移一位传至输出且 D0 送至 C_f
15	CF_EN	允许 C_f 保存的使能信号
16	ZF_EN	允许 Z_f 保存的使能信号
17	SM_EN	允许 SM 翻转的使能信号
18	IN_EN	执行 IN 指令时，允许外部设备输入的使能信号
19	OUT_EN	执行 OUT 指令时，允许寄存器值输出到外部设备的使能信号

1. 数据数据传送类指令的执行过程

寄存器之间的传送

MOV　R1，R2

要求完成的操作为（R2）→R1，执行过程为：

由 R2 的编码通过 RAA1、RAA0 从通用寄存器组 A 口读出 R2 的内容，在 S3～S0 和 M 的控制下，经 ALU 送入总线 BUS；由/WE 控制和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写入通用寄存器 R1。

寄存器到内存的传送

MOV　M，R2

要求完成的操作为（R2）→（C），执行过程为：

由 M 的编码 11 通过 RWBA1、RWBA0 从通用寄存器 B 口读出 C 寄存器中的地址，在 MADD=2 的控制下，地址通过选择器到达存储器 RAM 的地址输入端；由 R2 的编码通过 RAA1、RAA0 从通用寄存器组 A 口读出 R2 的内容，在 S3～S0 和 M 的控制下，经 ALU 送入总线 BUS，并在/CS 和 XL 控制下将 BUS 上的数据写入存储器 RAM。

内存到寄存器的传送

MOV　R1，M

要求完成的操作为（（C））→R1，执行过程为：

由 M 的编码 11 通过 RAA1、RAA0 从通用寄存器 A 口读出 C 寄存器中的地址，在 MADD=1 的控制下，地址通过选择器到达存储器 RAM 的地址输入端，/CS 和 DL 使数据出现在 BUS 上；由/WE 控制和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写入通用寄存器 R1。

2、算术逻辑运算类指令的执行过程

ADD　R1，R2

SUB　R1，R2

OR　R1，R2

这类指令的执行过程为：

由 R2 的编码通过 RAA1、RAA0 从通用寄存器组 A 口读出 R2 的内容，由 R1 的编码通 过 RWBA1、RWBA0 从通用寄存器组 B 口读出 R1 的内容，在 S3～S0 和 M 的控制下，经 ALU 送入总线 BUS；由/WE 控制和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写 入通用寄存器 R1。其中 ADD 和 SUB 指令影响状态位 Cf 和 Zf。

3、移位指令的执行过程

RSR　　R1

RSL　　R1

这类指令的执行过程为：

由 R1 的编码通过 RWBA1、RWBA0 从通用寄存器组 B 口读出 R1 的内容，在 S3～S0 和 M 的控制下通过 ALU，经移位逻辑循环右移或循环左移后送入总线 BUS；再由/WE 控制 和 R1 的编码选择 RWBA1、RWBA0，将 BUS 上的数据写入通用寄存器 R1。

但是，标准的 ALU 模块没有移位功能，需要在该模块出口与总线接口处增加一部分电 路以实现相应的移位功能。

4、转移类指令的执行过程

JMP　　add

JZ　　add

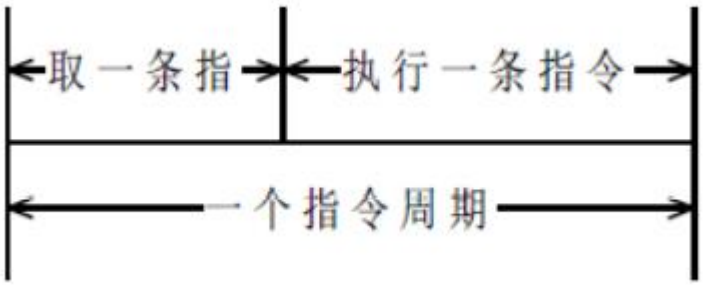
JC　　add

这类指令为双字节指令，第一字节为指令码，第二字节为转移目标地址。这类指令的执 行过程为：

在 MADD=0 的控制下，程序计数器 PC 中的地址通过选择器到达存储器 RAM 的地址 输入端，在/CS 和 DL 控制下转移地址从 RAM 中读出并送入 BUS；如果条件满足（IN PC=0） 则在 LD PC 允许下将 BUS 上的地址打 PC，否则 PC 加 1 计数。

当数据通路设计好之后，就要进行详细电路设计，这时需要考虑其它各种因素，比如进行触发器 Cf 和 Zf 的设置。

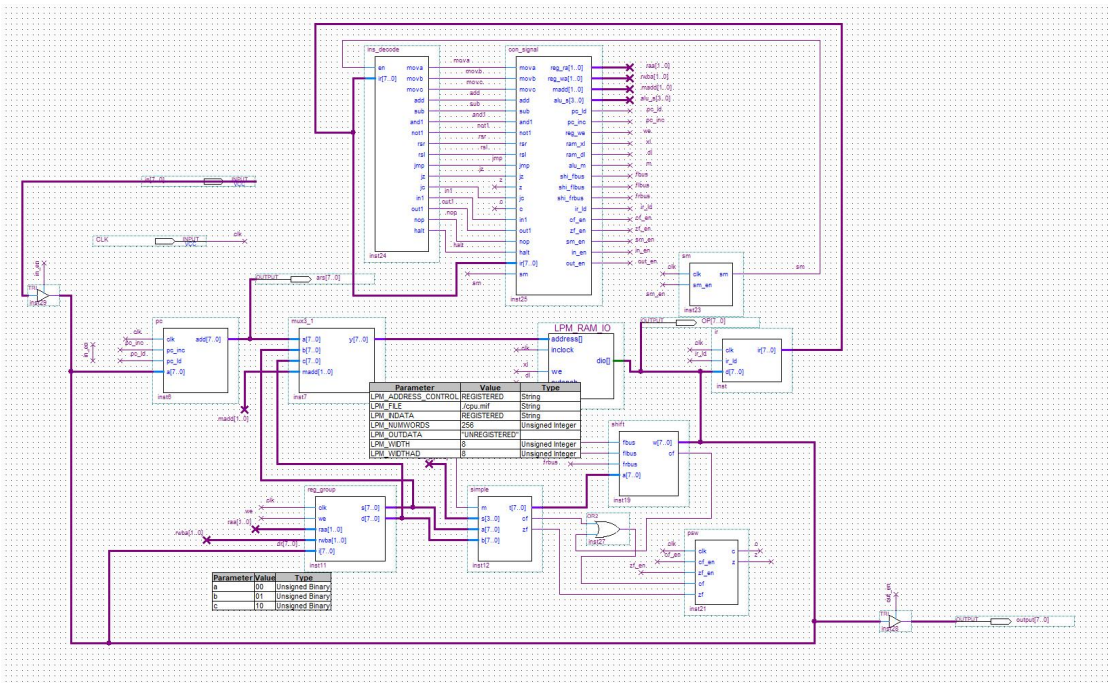
指令同期与数据通路结构、指令执行方式有关。指令可以串行执行，也可以并行执行。 本设计采用串行工作方式，即“读取—执行—再读取—再执行……”。串行工作方式虽然工 作速度和主机效率都要差一些，但它的控制简单。因此，本机指令周期可以确定为：



读取指令的时间随所使用的 RAM 的性能而异。执行一条指令所需工作脉冲的个数与宽 度要依据控制流和数据流所经过的路径与各级门的最大延迟而定。例如，本机中写入 RAM 和寄存器组的操作显然不能发生在“执行阶段”的任意时刻，它必须是在运算结果已经产生， 并被传送到总线的适当时刻才能“写”，这就需要工作脉冲来控制时序。

3.2 各模块的具体实现

1. 设计的整体架构



2. 指令译码器

源代码

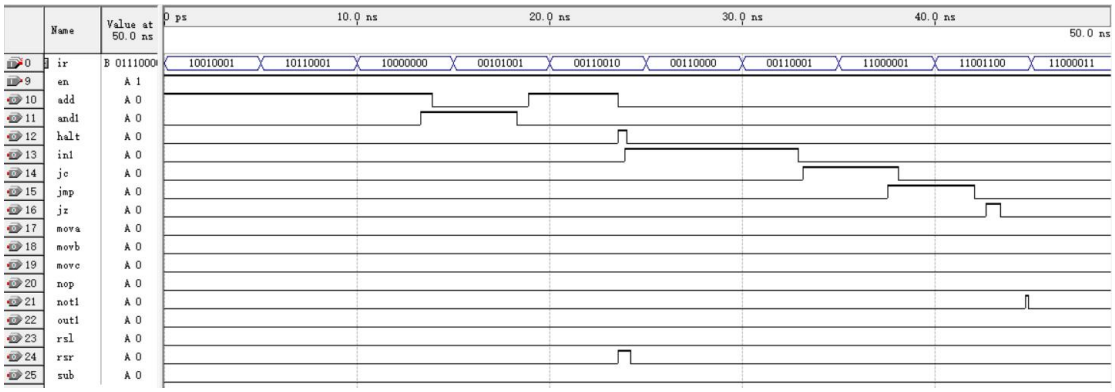
```
module ins_decode(en,ir,mova,movb,movc,add,sub,and1,not1,rsr,rsl,jmp,jz,jc,in1,out1,nop,halt);
    input [7:0] ir;
    input en;
    output mova,movb,movc,add,sub,and1,not1,rsr,rsl,jmp,jz,jc,in1,out1,nop,halt;
    reg mova,movb,movc,add,sub,and1,not1,rsr,rsl,jmp,jz,jc,in1,out1,nop,halt;
    always@(ir or en)
    begin
    mova=0;movb=0;movc=0;
    add=0;sub=0;and1=0;not1=0;
    rsr=0;rsl=0;
    jmp=0;jz=0;jc=0;
    in1=0;out1=0;
    nop=0;halt=0;
        if (en)
            begin
                if(ir[7:4]==4'b1100)//1100 refers to move action
                    begin
                        if(ir[3:2]==2'b11) movb=1;
                        else if(ir[1:0]==2'b11) movc=1;
                        else mova=1;
                    end
                else if(ir[7:4]==4'b1001) add=1;
                else if(ir[7:4]==4'b0101) not1=1;
                else if(ir[7:4]==4'b1011) and1=1;
                else if(ir[7:4]==4'b0110) sub=1;
                else if(ir[7:4]==4'b1010)
                    begin
                        if(ir[1:0]==2'b11) rsl=1;
                        else rsr=1;
                    end
                else if(ir[7:4]==4'b0011)//0011 refers to jump action
                    begin
                        if(ir[1:0]==2'b00) jmp=1;
                        else if(ir[1:0]==2'b10) jc=1;
                        else if(ir[1:0]==2'b01) jz=1;
                    end
                else if(ir[7:4]==4'b1000) halt=1;
                else if(ir[7:4]==4'b0010)in1=1;
                else if(ir[7:4]==4'b0100)out1=1;
                else if(ir[7:4]==4'b0111)nop=1;
                else ;
            end
        else ;
    end
endmodule
```

调试结果：

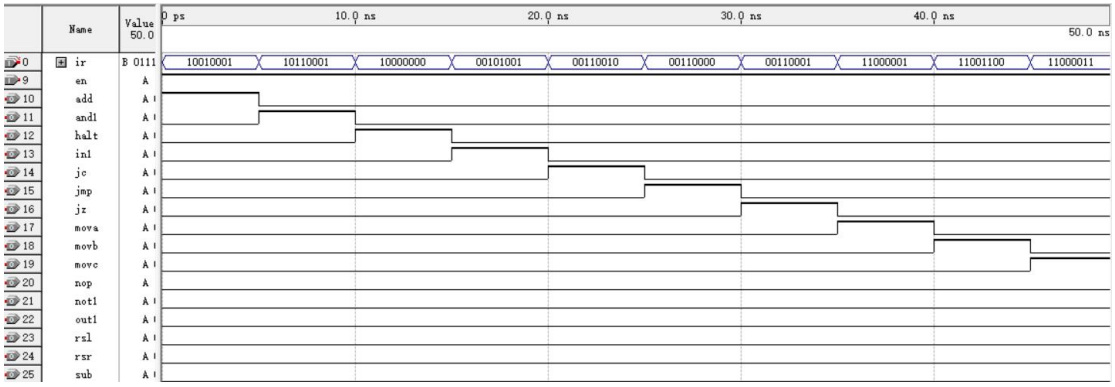
Flow Summary

Flow Status	Successful - Mon Nov 21 00:10:42 2022
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	insdecode
Top-level Entity Name	insdecode
Family	Cyclone II
Device	EP2C5T144C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	30 / 4,608 (< 1 %)
Total combinational functions	30 / 4,608 (< 1 %)
Dedicated logic registers	0 / 4,608 (0 %)
Total registers	0
Total pins	25 / 89 (28 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)

功能仿真波形



时序仿真波形



结果分析及结论：

分析：功能仿真是指在不考虑器件延迟和布线情况下最理想的，对源代码进行的的逻辑功能检验。由波形可知，对于输入状态的改变，输出实时改变，没有延迟。

当 en 为 0 时，不管 ir 为何值，16 个输出全为 0

当 en 为 1 时：

当 ir=11000001 时，mova 输出为 1；

当 ir=11001100 时，movb 输出为 1；

当 ir=11000011 时，movc 输出为 1；

当 ir=10010001 时，add 输出为 1；

当 ir=01100001 时，sub 输出为 1；

当 ir=10110001 时，and1 输出为 1；

当 ir=01010000 时，not1 输出为 1；

当 ir=10100000 时，rsr 输出为 1；

当 ir=10100011 时，rsl 输出为 1；

当 ir=00110000 时，jmp 输出为 1；

当 ir=00110001 时，jz 输出为 1；

当 ir=00110010 时，jc 输出为 1；

当 ir=00101001 时，in1 输出为 1；

当 ir=01000001 时，out1 输出为 1；

当 ir=01110000 时，nop 输出为 1；

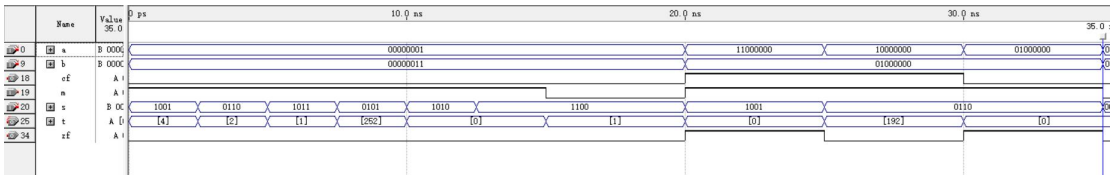
当 ir=10000000 时，halt 输出为 1；

3. ALU

源代码

```
    input [7:0] a,b;
    output [7:0] t;
    output cf,zf;
    reg [7:0] t;
    reg cf, zf;

always @(m, s, a, b, t, cf, zf)
begin
    cf = 1'b0;
    zf = 1'b0;
    t = 8'b00000000;
    if(m==1'b0)
    begin
        if(s==4'b1010) t=b;
        else if(s==4'b1100||s==4'b0100) t=a;
    end
    else if(m==1'b1)
            case(s[3:0])
                4'b1001:
                begin
                    {cf, t} =a+b;
                    if(t == 0)
                        zf = 1;
                    else
                        zf = 1;
                end
                4'b0110:
                begin
                    {cf, t} = b-a;
                    if(t==0) zf=1'b1;
                end
                4'b1011:                t = a & b;
                4'b0101:                t = ~b;

                default:
                begin
                    t = 8'b00000000;
                    cf = 0;
                    zf = 0;
                end
            end
        endcase
end
endmodule
```

功能仿真波形



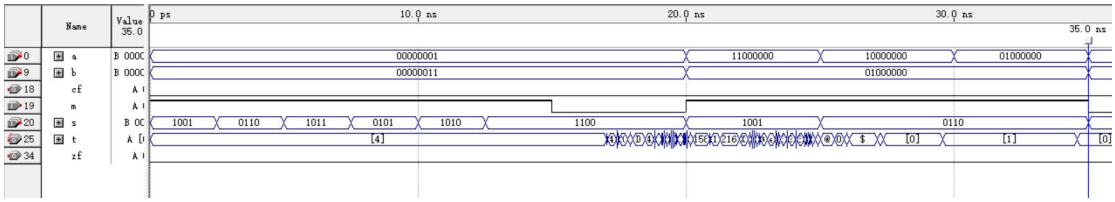
结果分析及结论：

分析：功能仿真是指不考虑器件延时和布线延时的理想情况下对源代码进行逻辑功能的验

证。由仿真波形可得，对于输入状态的变化，输出结果实时变化，没有延迟，其结果与电路设计的真值表的结果相对应。

当控制信号 m 为 1，s 为 1001 时，执行 t=a+b
当控制信号 m 为 1，s 为 0110 时，执行 t=b-a
当控制信号 m 为 1，s 为 1011 时，执行 t=a&b
当控制信号 m 为 1，s 为 0101 时，执行 t=~b
当控制信号 m 为 1，s 为 1010 时，执行 t=b
当控制信号 m 为 0，s 为 1100 时，执行 t=a
有进位和借位时 cf 为 1，否则为 0；
和为 0 或差为 0 时 zf 为 1，否则为 0；

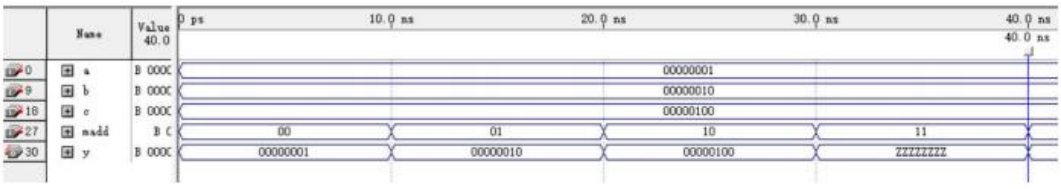
时序仿真波形



4. mux3_1
源代码

```
module mux3_1(a,b,c,madd,y);
    input[7:0]a;
    input[7:0]b;
    input[7:0]c;
    input[1:0]madd;
    output[7:0]y;
    reg[7:0]y;
    always@(*)
    begin
        y<=8'b0000_0000;
    case(madd)
            2'b00 : y<=a;
            2'b01 : y<=b;
            2'b10 : y<=c;
            default : ;
        endcase

    end
endmodule
```

功能仿真波形



结果分析及结论：
分析：功能仿真是指不考虑器件延时和布线延时的理想情况下对源代码进行逻辑功能的验证。由仿真波形可得，对于输入状态的变化，输出结果实时变化，没有延迟，其

结果与电路设计的真值表的结果相对应。

madd=00 时，y=a

madd=01 时，y=b

madd=10 时，y=c

madd=11 时，y 输出高阻态

结论：功能仿真操作简单，能体现和验证实验的功能，但忽略延迟的影响会使结果与实际结 果有一定误差。

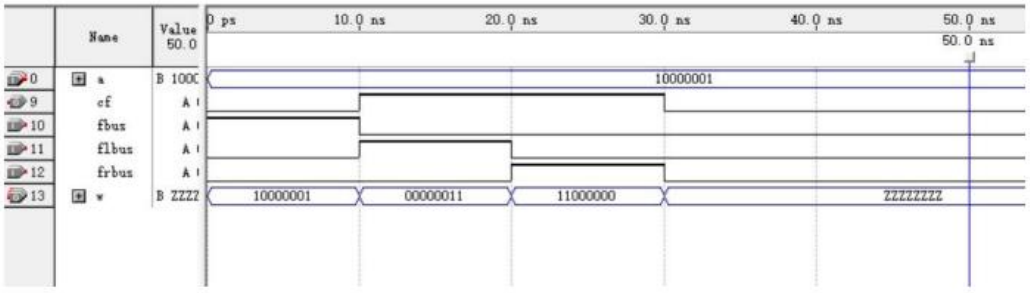
时序仿真波形

5. 移位逻辑

源代码

```
module shift(fbus,flbus,frbus,a,w,cf);
    input fbus;
    input flbus;
    input frbus;
    input [7:0]a;
    output [7:0]w;
    output cf;
    reg [7:0]w;
    reg cf;
    always@(fbus,flbus,frbus,a)
        begin
            cf=1'b0;
            if(fbus==1'b1) w[7:0]=a[7:0];
            else if(frbus==1'b1)
                begin
                    w[7:0]={a[0],a[7:1]};
                    cf=a[0];
                end
            else if(flbus==1'b1)
                begin
                    w[7:0]={a[6:0],a[7]};
                    cf=a[7];
                end
            else w[7:0]=8'hZZ;
        end
endmodule
```

功能仿真波形

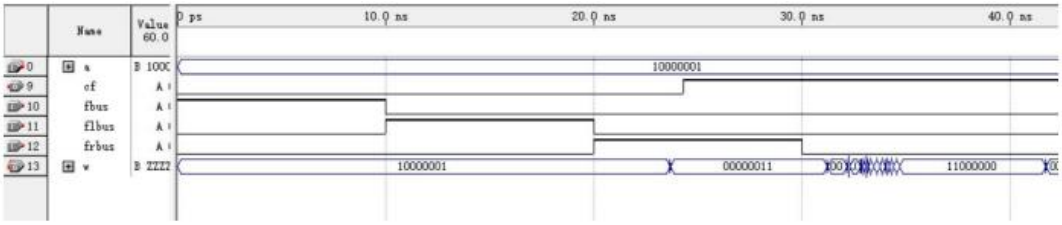


结果分析及结论：

分析：功能仿真是指在不考虑器件延迟和布线情况下最理想的，对源代码进行的的逻辑功能检验。由波形可知，对于输入状态的改变，输出实时改变，没有延迟。

当 fbus=1,frbus=0,flbus=0,不执行位移操作，输出等于输入，cf 不改变
当 fbus=0,frbus=1,flbus=0,执行右位移操作，输出等于输入右移移位，有进位 cf=1
当 fbus=0,frbus=0,flbus=1,执行左位移操作，输出等于输入左移移位，cf 不改变
当控制信号全为 0 时，输出为高阻态
结论：功能仿真操作简单，能够体现实验最基本的逻辑功能，但是忽略了延迟所造成额冒险冲突，相比较实际有所不同。

时序仿真波形

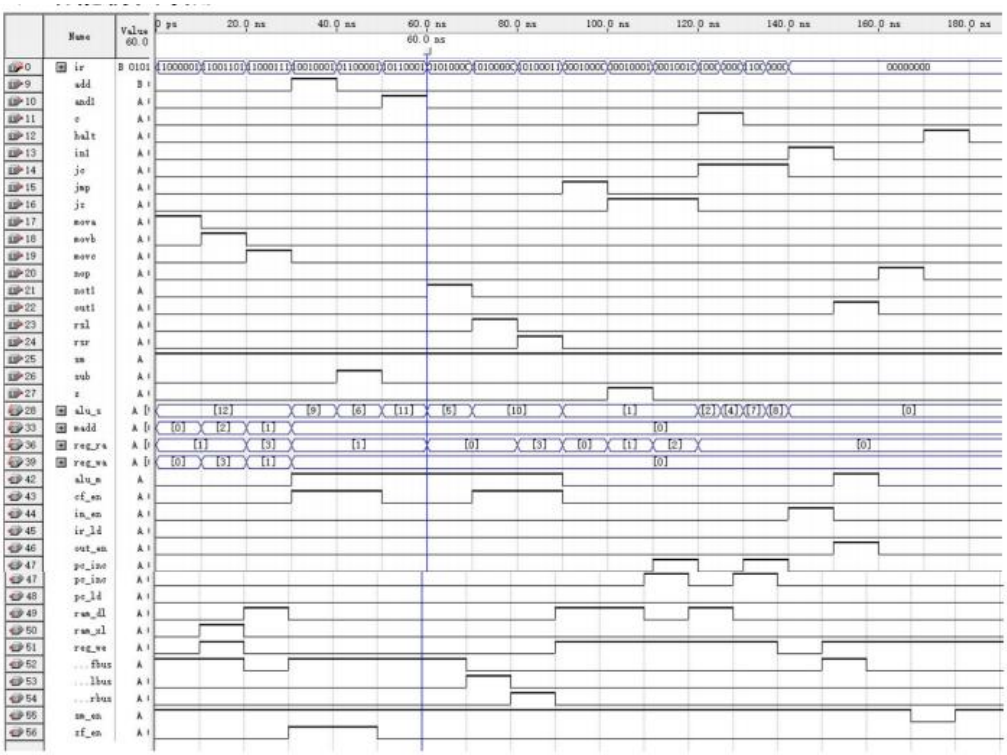


6. con_signal

源代码

```
module con_signal(mova,movb,movc,add,sub,and1,not1,rsr,rsl,jmp,jz,z,jc,c,in1,out1,nop,halt,ir,sm,
    reg_ra,reg_wa,madd,alu_s,pc_ld,pc_inc,reg_we,ram_xl,ram_dl,alu_m,shi_fbus,shi_flbus,shi_frbus,ir_ld,cf_en,zf_en,sm_en,in_en,out_en);
    input [7:0] ir;
    input mova,movb,movc,add,sub,and1,not1,rsr,rsl,jmp,jz,z,jc,c,in1,out1,nop,halt,sm;
    output [1:0]reg_ra;
    output [1:0]reg_wa;
    output [1:0]madd;
    output [3:0]alu_s;
    output pc_ld,pc_inc,reg_we,ram_xl,ram_dl,alu_m,shi_fbus,shi_flbus,shi_frbus,ir_ld,cf_en,zf_en,sm_en,in_en,out_en;
    reg [1:0]reg_ra;
    reg [1:0]reg_wa;
    reg [1:0]madd;
    reg[3:0]alu_s;
    reg pc_ld,pc_inc,reg_we,ram_xl,ram_dl,alu_m,shi_fbus,shi_flbus,shi_frbus,ir_ld,cf_en,zf_en,sm_en,in_en,out_en;
    always@(mova,movb,movc,add,sub,and1,not1,rsr,rsl,jmp,jz,z,jc,c,in1,out1,nop,halt,sm,ir)
        begin
            sm_en=~halt;
            ir_ld=~sm;
            ram_dl=(~sm)|movc|jmp|(jz&z)|(jc&c);
            ram_xl=movb;
            shi_fbus=mova|movb|add|sub|and1|not1|out1;
            shi_flbus=rsl;
            shi_frbus=rsr;
            alu_s=ir[7:4];
            alu_m=and1|not1|add|sub;
            cf_en=add|sub|rsr|rsl;
            zf_en=add|sub;
            pc_inc=(~sm)|(jc&(~c))|(jz&(~z));
            pc_ld=jmp|(jz&z)|(jc&c);
            reg_we=~(sm&(mova|movc|add|sub|and1|not1|rsr|rsl|in1));
            in_en=in1;
            out_en=out1;
            reg_ra=ir[1:0];
            reg_wa=ir[3:2];
            if(sm==1&&movc==1) madd=01;
            else if(sm==1&&movb==1) madd=2'b10;
            else madd=00;
        end
endmodule
```

功能仿真波形



结果分析及结论：

分析：功能仿真是指不考虑器件延时和布线延时的理想情况下对源代码进行逻辑功能的验证。由仿真波形可得，对于输入状态的变化，输出结果实时变化，没有延迟，其结果与电路设计的真值表的结果相对应。

当 mova 指令执行时，shi_fbus 和 sm_en 输出 1，其他输出为 0，madd 输出 00，alu_s 输出为 1100，reg_ra 输出 01，reg_wa 输出 00

当 movb 指令执行时，ram_xl 和 shi_fbus 和 reg_we 和 sm_en 输出为 1，其他输出为 0，madd 输出为 10，alu_s 输出为 1100，reg_ra 输出 01，reg_wa 输出 11

当 movc 指令执行时，ram_dl 和 sm_en 输出为 1，其他输出为 0，madd 输出 01，alu_s 输出 1100，reg_ra 输出 11，reg_wa 输出 01

当 add 指令执行时，shi_fbus，alu_en，cf_en，zf_en，sm_en 输出为 1，其他输出为 0，alu_s 为 1001，reg_ra 输出 01，reg_wa 输出 00

当 sub 指令执行时，shi_fbus 和 alu_m，cf_en，zf_en 和 sm_en 输出为 1，其他输出为 0，alu_s 输出 0110，reg_ra 输出 01，reg_wa 输出 00

当 and1 指令执行时，shi_fbus 和 alu_m 和 sm_en 输出 1，其他输出 0，alu_s 输出 1011，reg_ra 输出 01，reg_wa 输出 00

当 not1 指令执行时，

shi_fbus 和 alu_m 和 sm_en 输出 1，其他输出 0，

alu_s 输出 0101， reg_ra 输出 00，reg_wa 输出 00

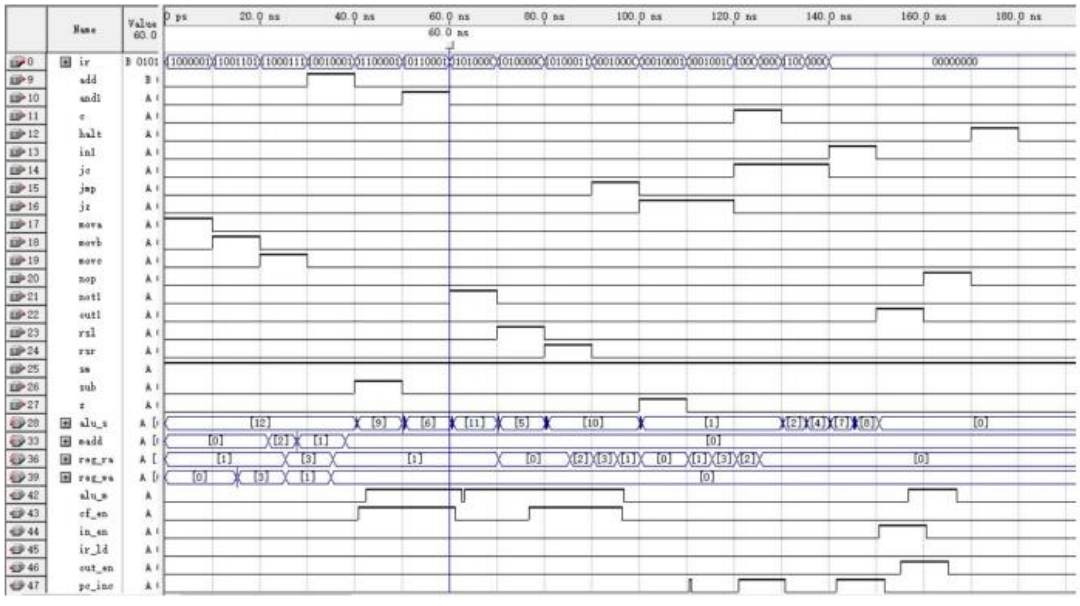
当 rsl 指令执行时，shi_flbus 和 alu_m 和 cf_en 和 sm_en 输出 0，其他输出 0，alu_s 输出 1010，reg_ra 和 reg_wa 输出 00

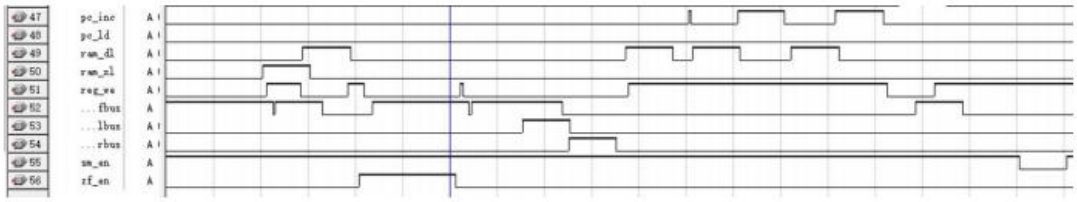
当 rsr 指令执行时，shi_frbus 和 alu_m 和 cf_en 和 sm_en 输出 1，其他输出 0，alu_s 输出 1010，reg_ra 输出 11，reg_wa 输出 00

当 jmp 指令执行时，ram_dl，pc_ld，reg_we 和 sm_en 输出 1，其他输出 0，alu_s 输出 0001，reg_ra 和 reg_wa 输出 00

当 jz 指令为 1 和 jc 指令为 1 时，若 z 和 c 为 1 时，ram_dl 和 pc_ld 和 reg_we 和 sm_en 输出为 1，其他输出为 0
当 z 和 c 为 0 时，pc_inc 和 reg_we 和 sm_en 输出 1，其他输出 0
当 inl 指令执行时，sm_en 和 in_en 输出 1，其他输出 0
当 outl 指令执行时，sm_en 和 out_en 输出 1，其他输出 0
当 nop 指令执行时，sm_en 输出 1，其他输出 0
当 halt 指令执行时，输出全为 0
结论：功能仿真操作简单，能体现和验证实验的功能，但忽略延迟的影响会使结果与实际结果有一定误差。

时序仿真波形



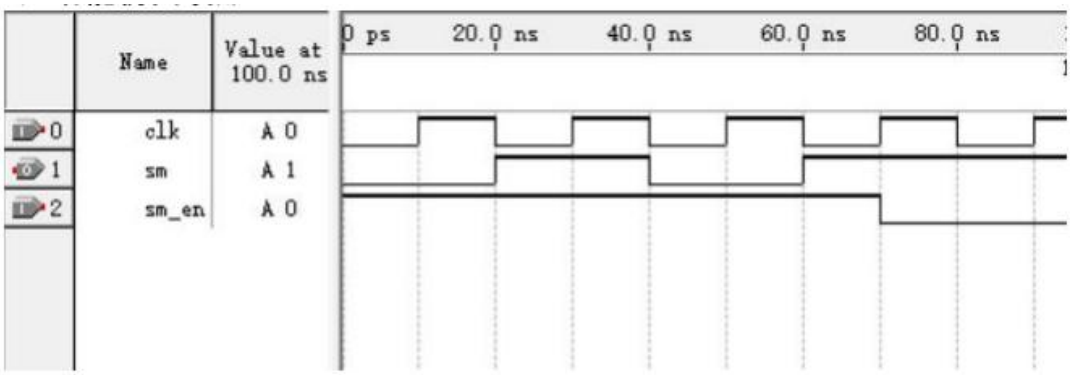


7. sm

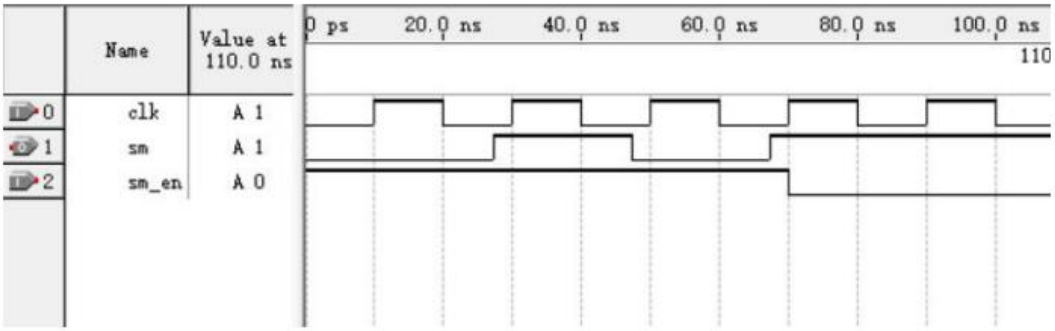
源代码

```
module sm(
  input clk,sm_en,
  output reg sm
);
initial sm=1'b0;
always @(negedge clk) begin
    if(sm_en)begin
        sm<=~sm;
    end
    else begin
        sm<=sm;
    end
end
endmodule
```

功能仿真波形



时序仿真波形



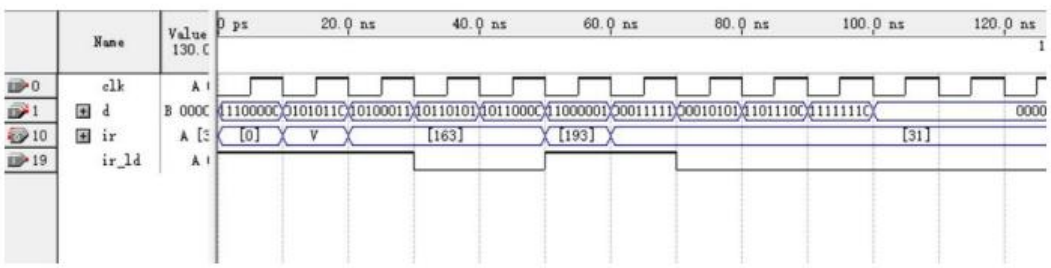
分析与结论：0-70ns：sm_en=1 时，在时钟下降沿，将 sm 进行取反，可以看到 20ns 时 sm 由 0 变为 1，40ns 时 sm 由 1 变为 0，60ns 时 sm 由 0 变为 1;70-140ns：sm_en=0 时，在时钟下降沿，sm 不变。可以看到 70-140ns 过程中 sm 值保持不变,符合该元件对于功能的要求。

8. ir

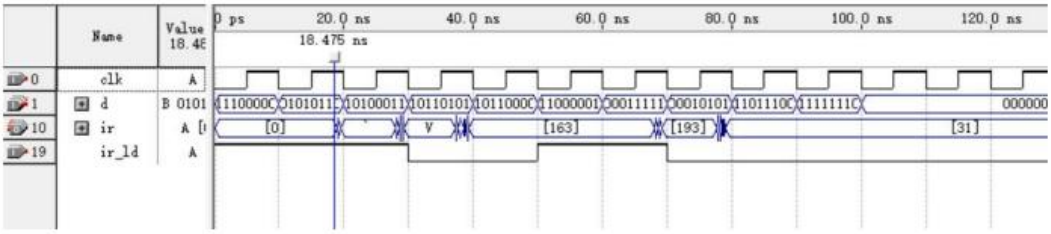
源代码

```
module ir(clk,ir_ld,d,ir);
    input clk,ir_ld;
    input [7:0] d;
    output reg [7:0] ir;
    always@(negedge clk)
        begin
            if(ir_ld) ir<=d;
            else ;
        end
endmodule
```

功能仿真波形



时序仿真波形



结果分析：对于功能仿真，在 0-30ns，ir_ld 为 1，在时钟下降沿将输入写入输出，当 ir_ld 为 0 时，输出保持不变，对于时序仿真，可以看到输出存在 9ns 左右的延迟，同时部分时刻输入的变化导致冒险出现，使得输出错误，输出的变化情况大致与功能仿真相同

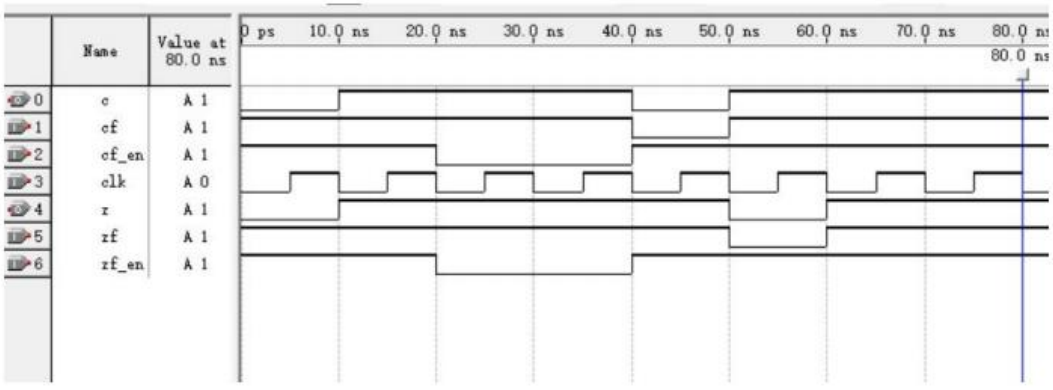
结论：元件设计符合设计要求，元件内部存在 9ns 左右的延迟

9.psw

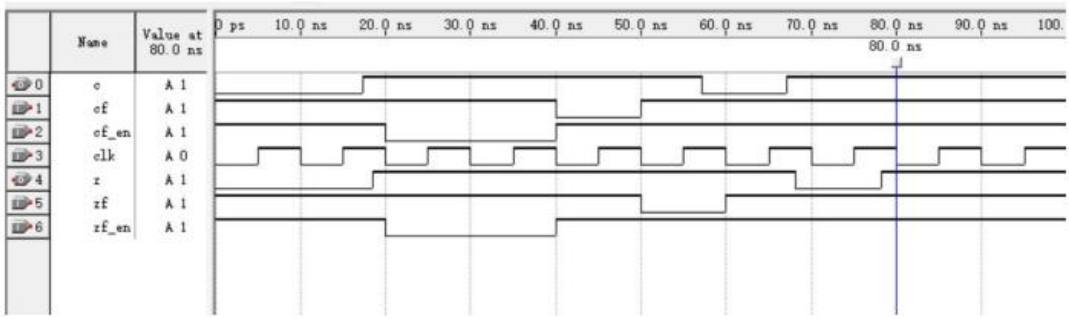
源代码

```
module psw(input clk,cf_en,zf_en,cf,zf,
output reg c,z);
initial begin c=1'b0;z=1'b0;end
always @(negedge clk)begin
    if(cf_en)begin
        c<=cf;
    end
    else begin
        c<=c;
    end
end
always @(negedge clk) begin
    if(zf_en)begin
        z<=zf;
    end
    else begin
        z<=z;
    end
end
endmodule
```

功能仿真波形



时序仿真波形



结果分析：对于功能仿真，0-20ns，cf_en 和 zf_en 为 1，在时钟下降沿，将 cf 和 zf 的值写入输出 c，z 中，20-40ns，cf_en 和 zf_en 为 0，输出 c 和 z 保持不变，对于时序仿真，其中输出 c 有 7ns 左右延迟，z 有 8ns 左右延迟，其输出变化与功能仿真大致相同

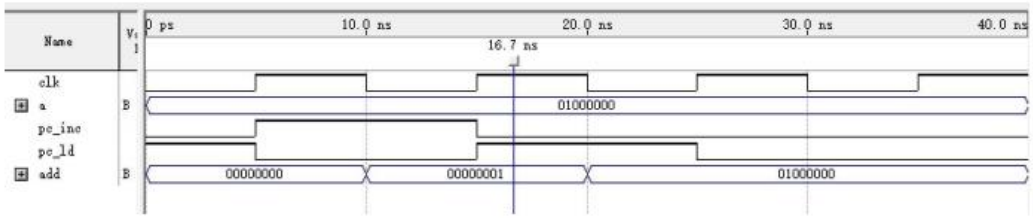
结论：元件设计符合要求，输出 c 有 7ns 延迟，输出 z 有 8ns 延迟

10. pc

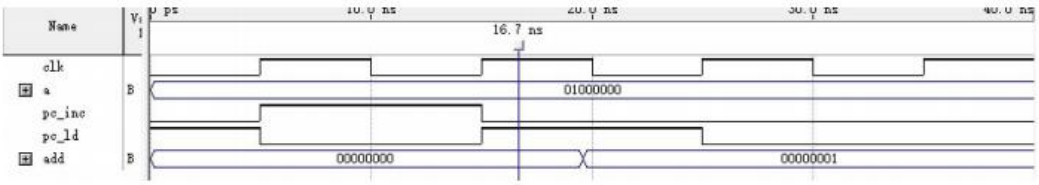
源代码

```
module pc(clk,pc_inc,pc_ld,a,add);
 input clk,pc_inc,pc_ld;
 input [7:0] a;
 output reg[7:0] add;
always @(negedge clk) begin
     if(pc_inc==1&&pc_ld==0)add<=add+1'b1;
     else if(pc_inc==0&&pc_ld==1)add<=a;
     else add<=add;
 end
 endmodule
```

功能仿真波形



时序仿真波形



结果分析：对于功能仿真，5-15ns， pc_inc 为 1，pc_ld 为 0，执行地址加 1 操作，15ns-25ns,pc_inc 为 0，pc_ld 为 1，执行写入操作，将输入写入到输出中，25-40ns，pc_inc 为 0，pc_ld 为 0，

数据保持不变，对于时序仿真，存在 9ns 左右的延迟，输出结果大致与功能仿真相同

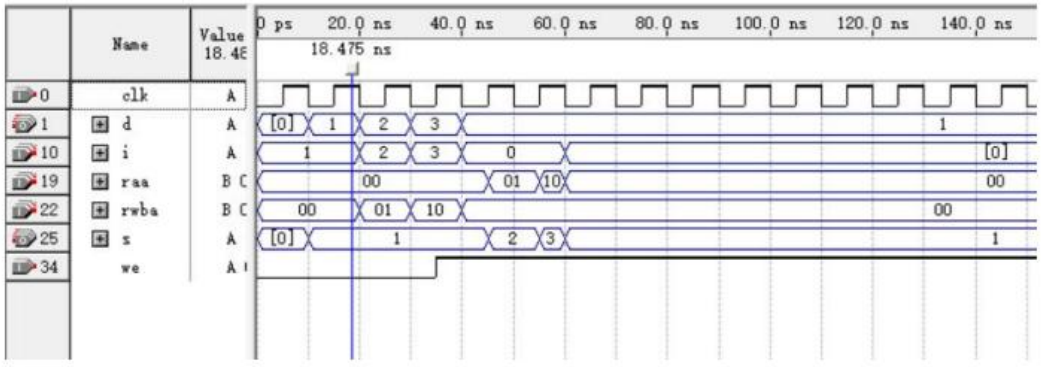
结论：元件设计符合要求，元件存在 9ns 左右的延迟

11. reg_group

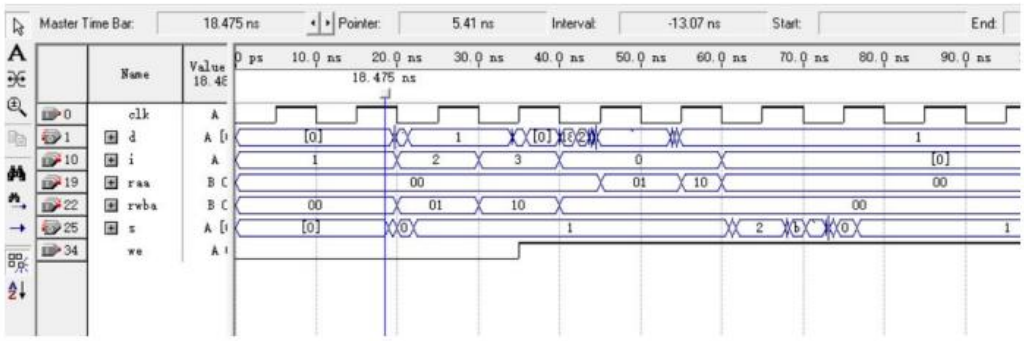
源代码

```
module reg_group(input clk,we,
input [1:0]raa,
input [1:0]rwba,
input [7:0]i,
output reg[7:0]s,output reg[7:0]d);
reg[7:0] A=8'b0000_0001,B=8'b0000_0010,C=8'b10000000;
parameter a=2'b00,b=2'b01,c=2'b10;

always @(raa or rwba or A or B or C)begin

    case (raa)
    a:begin s=A;end
    b:begin s=B;end
    c:begin s=C;end
    default:begin s=1'bz;end
    endcase

    case(rwba)
    a:begin d=A;end
    b:begin d=B;end
    c:begin d=C;end
    default:begin d=1'bz;end
    endcase
end

always @(negedge clk)begin

   if(we==1'b0)begin
        if(rwba==a)A<=i;
        else if(rwba==b)B<=i;
        else if(rwba==c)C<=i;
        else A<=A;
    end
    else A<=A;

end
endmodule
```

功能仿真波形



时序仿真波形

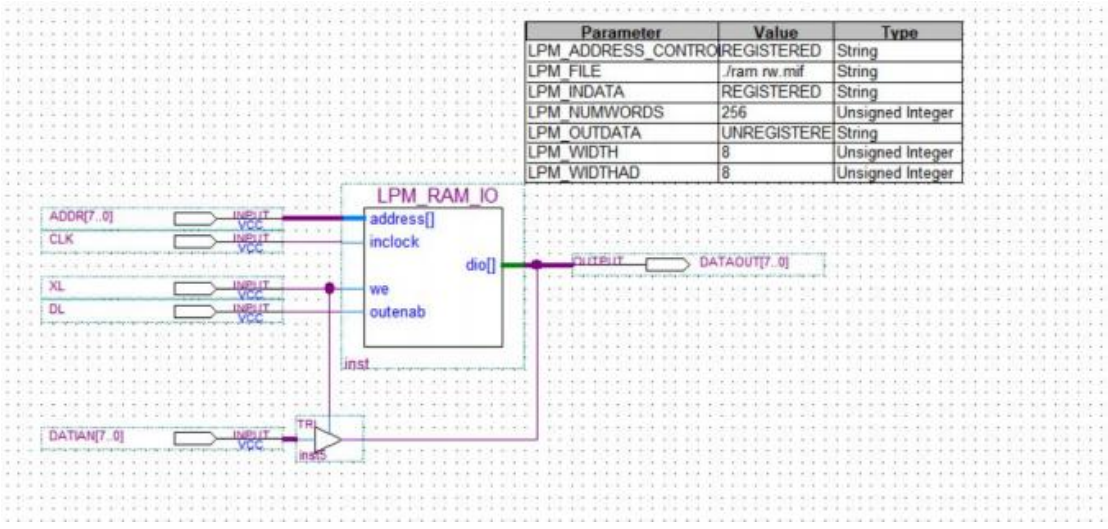


结果分析：对于功能仿真，在 0-35ns，we 为 0，进行写入操作，在每个下降沿，当 rwba=00,01,10 时，将输入 i 分别写入寄存器 A,B,C 中，而在 35-65ns，we 为 1，进行读取操作，在每个下降沿，根据 raa 和 rwba 的值 s，d 输出对应寄存器的值，当 raa=00，s 输出 A 的值，raa=01，s 输出 B 的值，raa=10，s 输出 C 的值，对于时序仿真，输出 s 存在 9ns 左右的延迟，输出 d 存在 10ns 左右延迟，其余输出结果大致与功能仿真相同，对于时序分析，可以得到时钟输出延迟 tco 为 13.487ns，建立时间 tsu 为 7.110ns，保持时间 th 为 0.479ns，电路延迟时间 tpd 为 16.353ns

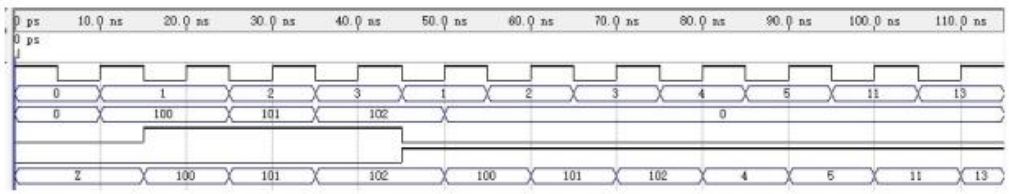
结论：元件设计符合要求，输出 s 存在 9ns 左右的延迟，输出 d 存在 10ns 左右延迟

12. ram

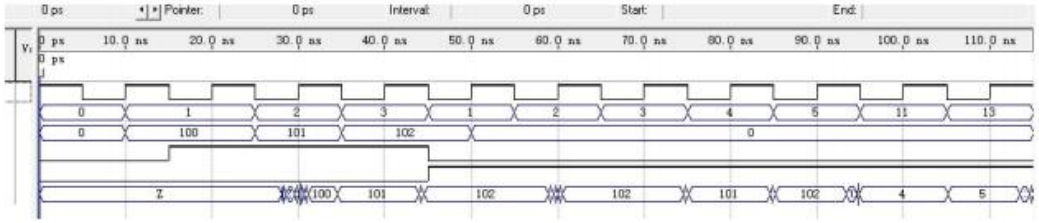
电路图



功能仿真波形



时序仿真波形



结果分析：对功能仿真，0-15ns，xl 和 dl 都为 0，输出为高阻态。15-45ns，xl 为 1，dl 为 0，进行写入操作，对于三个上升沿，分别将输入信号 100,101,102 分别写入地址 1,2,3 中。45-115ns，xl 为 0，dl 为 1，进行读操作，对于每个上升 沿，分别将 1,2,3,4,5,11,13 地址中的指令读出并输出，其中 4,5,11,13 地址中的 指令已提前存入对应的 mif 文件中，对于时序仿真，输出存在 11ns 左右的延迟，同时由于输入的改变导致某些位置出现冒险。
结论：元件设计符合要求，输出存在 11ns 左右延迟。

四、 系统测试

4.1 测试环境

开发软件:QuartusII 9.1 Build 35003/24/2010 SP 2 SJ Full Version
操作系统: Windows 11 家庭中文版
FPGA 学习板芯片信号: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。）

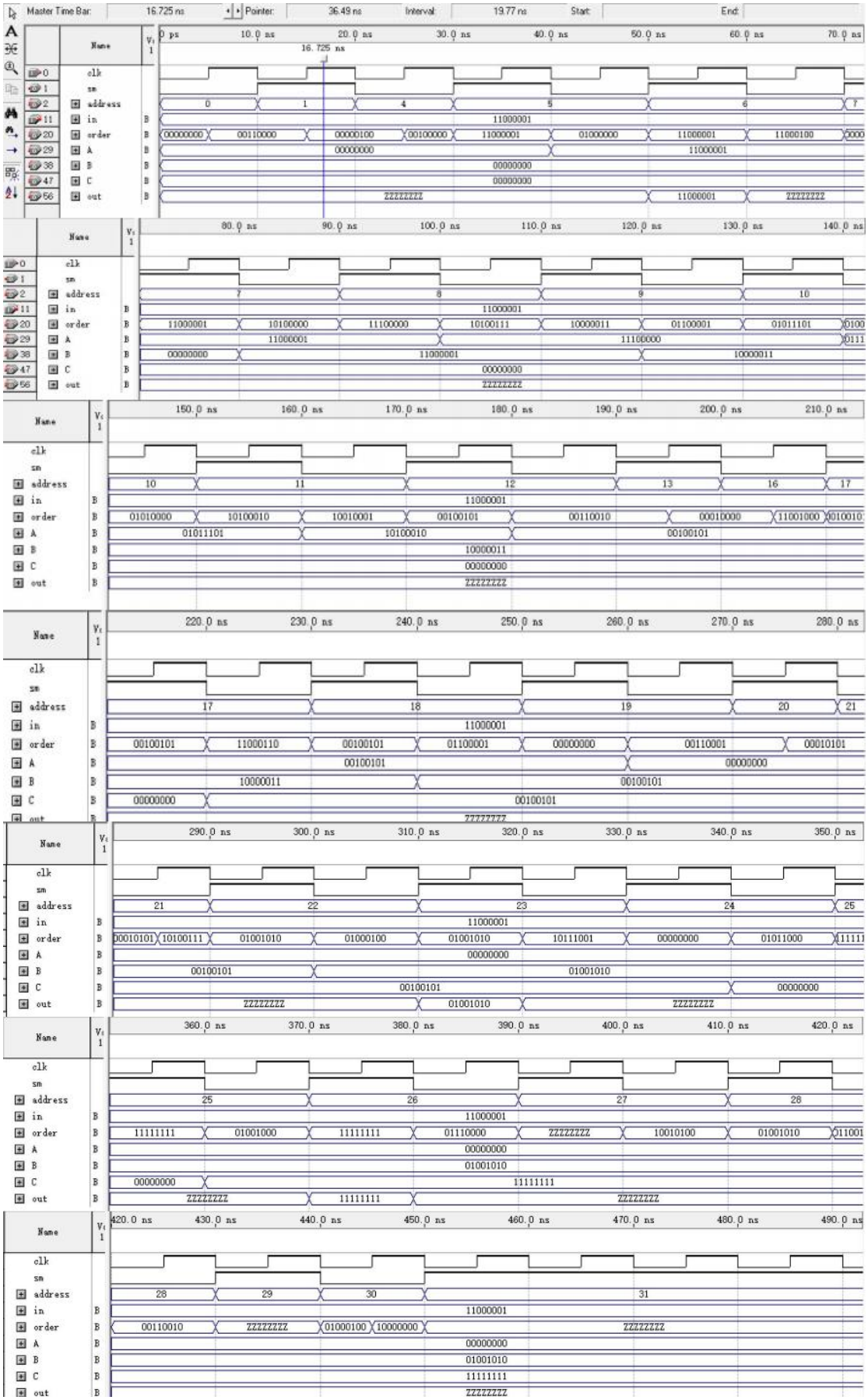
RAM 地址	指令	指令二进制编码	指令十六制编码	
0，1：	JMP 04H	(00110000 00000100)	(30H 04H)	
4：	IN A	(00100000)	(20H)	A=10000011 (83H)
			（注：A 中的 83H 是由外部输入的）	
5：	OUT A	(01000000)	(40H)	
6：	MOV M A	(11001100)	(CCH)	C=10000000 (80H)
		（注：C 中的 80H 是在通用寄存器组设计时对 C 初始化的值）		
7：	MOV B M	(11000111)	(C7H)	B=10000011 (83H)
8：	SUB A,B	(01100001)	(61H)	A=00000000 (00H), Z=1
9,10：	JZ 10H	(00110001 00010000)	(31H 10H)	
16：	NOT A	(01010000)	(50H)	A=11111111 (FFH)
17：	AND A,C	(10110010)	(B2H)	A=10000000 (80H)
18：	ADD A,B	(10010001)	(91H)	A=00000011 (03H),C=1
19,20：	JC 18H	(00110010 00011000)	(32H 18H)	
24：	NOP	(01110000)	(70H)	
25：	RSR A	(10100000)	(A0H)	A=10000001 (81)
26：	OUT A	(01000000)	(40H)	
27：	RSL B	(10100111)	(A7H)	B=00000111 (07H)
28：	MOV A,B	(11000001)	(C1H)	A=00000111 (07H)
29：	OUT A	(01000000)	(40H)	
30：	HALT	(10000000)	(80H)	
31：	ADD A,B	(10010001)	(91H)	
32：	OUT A	(01000000)	(40H)	

Mif 文件

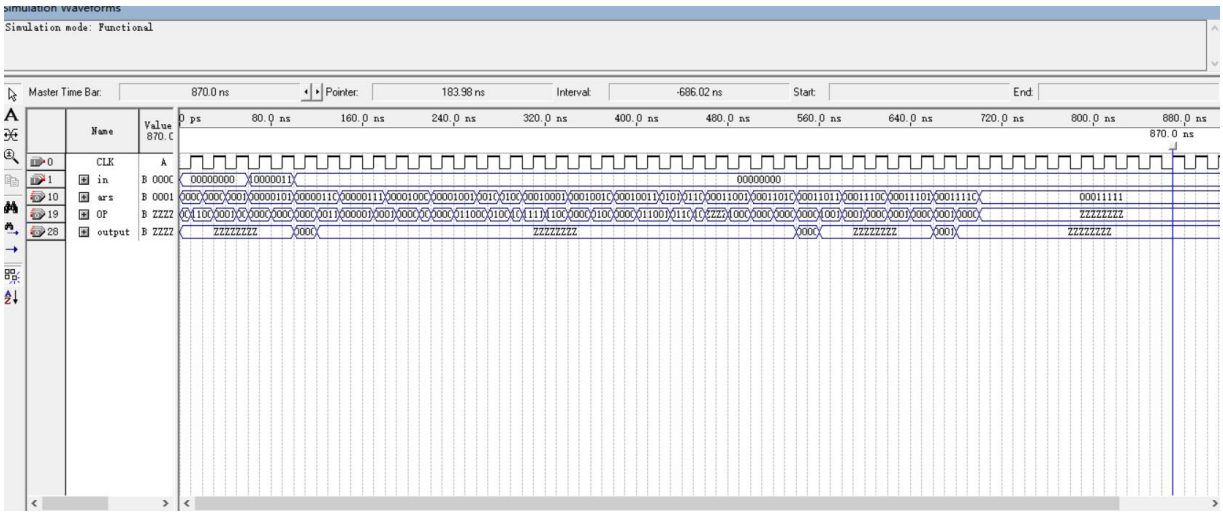
cpu.bdf* | cpu.mif | cpu.vwf* | Simulation Report - Simulation Waveforms

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	00110000	00000100	00000000	00000000	00100000	01000000	11001100	11000111
8	01100001	00110001	00010000	00000000	00000000	00000000	00000000	00000000
16	01010000	10110010	10010001	00110010	00011000	00000000	00000000	00000000
24	01110000	10100000	01000000	10100111	11000001	01000000	10000000	10010001
32	01000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
64	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
72	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
80	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
88	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
96	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
104	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
112	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
120	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
128	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
136	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

4.3 测试结果


Master Time Bar: 16.725 ns
Pointer: 36.49 ns
Interval: 19.77 ns
Start:
End:
Name
clk
sm
address
in
order
A
B
C
out

分析：
0-20ns，执行 JMP 指令，跳转到地址 00000100
20-40ns，执行 IN A 指令，外界输入 10000011 到寄存器 A
40-60ns，执行 OUT A 指令，输出 A 中的值 10000011 到外界
60-80ns，执行 MOV M,A 指令，C 的值为 10000000
80-100ns，执行 MOV B M 指令，B 的值为 100000011
100-120ns，执行 JZ 指令，跳转到地址 00010000
120-140ns，执行 NOT A 指令，A 的值为 11111111
140-160ns，执行 AND A C 指令，A 的值为 10000000
160-180ns，执行 ADD A B 指令，A 的值为 00000011，C=1
180-200ns，执行 JC 指令，跳转到地址 00011000
200-220ns，执行 NOP 指令，无变化
220-240ns，执行 RSR A 指令，A 的值为 10000001
240-260ns，执行 OUT A 指令，输出 A 10000001
260-280ns，执行 RSL B 指令，B 的值为 00000111
280-300ns，执行 MOV A B 指令，A 的值为 00000111
300-320ns，执行 OUT A 指令，输出 A 中的值 00000111 到外界
320-340ns，执行 HALT 指令，停机
340-360ns，执行 ADD AB 指令，不相与
360-380ns，执行 OUT A 指令，不输出



4.4 模型机性能分析

Flow Status	Successful - Wed Feb 22 22:28:53 2023
Quartus II Version	9.0 Build 184 04/29/2009 SP 1 SJ Web Edition
Revision Name	cpu
Top-level Entity Name	cpu
Family	Cyclone II
Device	EP2C5T144C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	219 / 4,608 (5 %)
Total combinational functions	218 / 4,608 (5 %)
Dedicated logic registers	43 / 4,608 (< 1 %)
Total registers	43
Total pins	33 / 89 (37 %)
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 %)

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	9.076 ns	in[5]	ir:inst|ir[5]	--	CLK	0
2	Worst-case tco	N/A	None	19.659 ns	reg_group:inst11|C[5]	OP[6]	CLK	--	0
3	Worst-case tpd	N/A	None	14.691 ns	in[7]	OP[7]	--	--	0
4	Worst-case th	N/A	None	-2.573 ns	in[2]	reg_group:inst11|B[2]	--	CLK	0
5	Clock Setup: 'CLK'	N/A	None	38.68 MHz (period = 25.850 ns)	reg_group:inst11|C[5]	lpm_ram_io:inst1|altram:sram|altsyncram:ram_block|altsyncram_rha1:auto_generated|ram_block1a0~porta_datain_reg7	CLK	CLK	0
6	Total number of failed paths								0

分析：此程序占用资源数为 219，时钟频率为 38.68MHz，性能比较良好，占用资源也比较少，并且设计出的程序符合实验要求。

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。

需要掌握的理论：理解模型机中数据的格式，在该模型及中数据采用 8 位二进制定点补码表示，其中最高位数位符号位。其指令共有 8 位，前 4 位为操作码，后四位分别表示目的寄存器和源寄存器。而对于该模型机的指令系统共有 16 条，在上述分析中已经阐述。同时我们应该了解简易模型机的内部细分结构和工作原理。同时熟悉多路复用器、移位逻辑和控制信号产生逻辑 sm,ir,psw,pc,reg_group 等器件的工作原理。具体来说就是学会其功能和指令译码器的汇编符号到编码，再到最后的功能。了解其具体元器件的引脚关系并且学会用 Verilog 语言在 quartus 上进行代码的编写。

遇到的困难：一开始很无从下手，只是把各个部分的代码完成，但是并不会组装。一开始把所有的连线都一一连在对应位置，不过这样看上去非常冗余不美观。后来学会了用标号来阐述输入输出引脚，这样看上去不仅清晰美观，而且可读性强。到运行 bdf 文件的时候各种顶层文件、路径名称以及引脚的报错。折腾好这些文件后开始调试代码，而每次的波形图总是不能完全正确地展示每个输出。在重复调试了每个模块后，终于能够运行不报错。不过并不清楚该如何减少空间占用内存和耗时。通过不断重写代码，能够有效地减小耗时和占用内存。不过到后边内存和耗时似乎不能同时兼得。

解决办法：在一开始不太会用 quartus 的波形仿真，通过上网搜教程并且询问老师和同学得以解决此问题。最头大的是不会分析时序仿真中出现的那么多冒险，通过广泛的在网上查阅资料，开始不断修改代码来减少冲突冒险的产生。

同时通过凌老师的录课找到了连接模型机的一些启发，可以顺着老师的思路去完成整个机器的连接。

5.2 对本实验内容、过程和方法的改进建议（可选项）。

希望实验指导书上可以不光有理想的输入输出关系，还可以有一些实际中会出现的问题如冲突的出现，以及对这一问题的分析和解决方法。