模型机设计报告

班级 计科 1907 姓名 杨杰 学号 201908010705

一、设计目的

完整、连贯地运用《数字逻辑》所学到的知识，熟练掌握 EDA 工具基本使用方法，为学习好后续《计算机原理》课程做铺垫。

二、设计内容

① 按照给定的数据通路、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；

② 要求灵活运用各方面知识，使得所设计的计算机具有较佳的性能；

③ 对所设计计算机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）数据格式

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。

（2）寻址方式

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。

寄存器直接寻址

操 作 码	R1	R2

当R1 和R2 均不是“11”时，R1 和R2 分别表示两个操作数所在寄存器的地址（寄存器编号），其中R1 为目标寄存器地址，R2 为源寄存器地址。

R1 或R2 的值	指定的寄存器
00	A 寄存器
01	B 寄存器
10	C 寄存器

寄存器间接寻址

操 作 码	R1/11	R2/11

当 R1 或 R2 中有一个为“11”时，表示相应操作数的地址在 C 寄存器中。

（3）**指令系统**

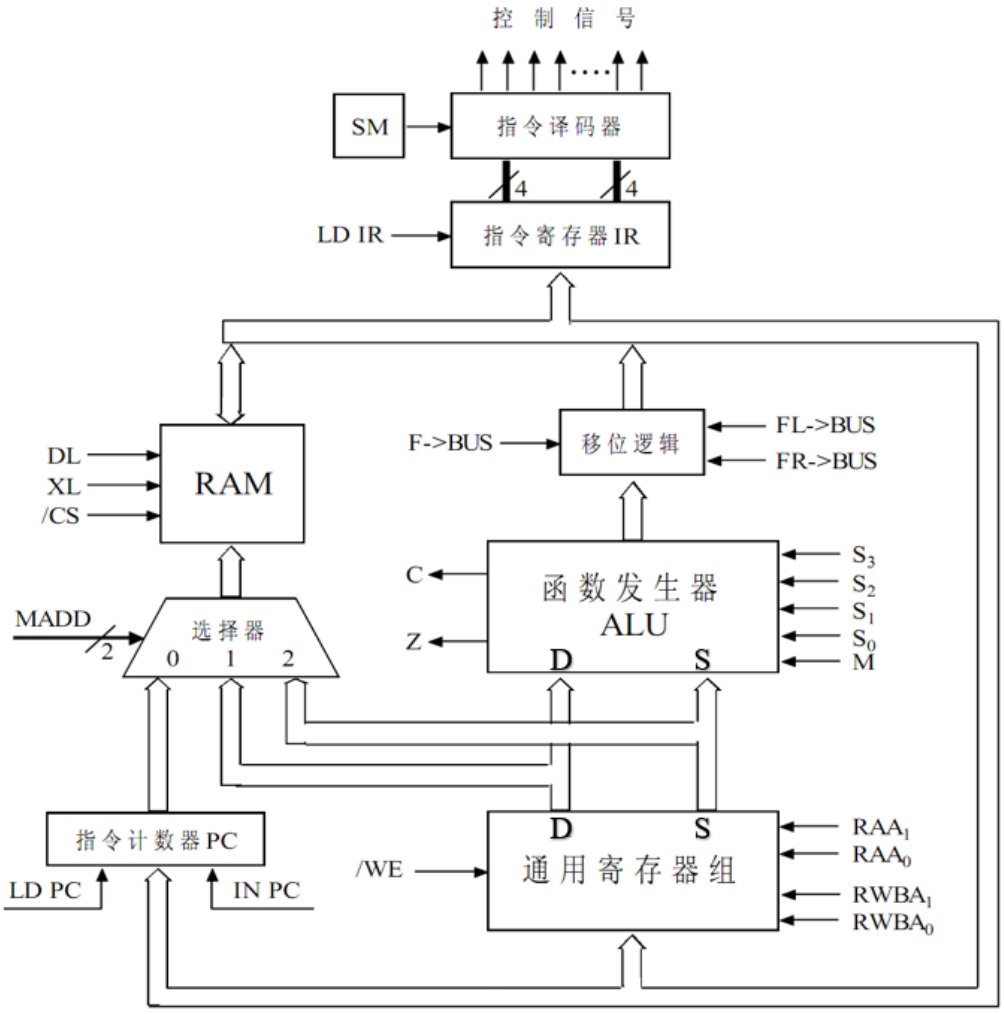
指令系统有 16 条指令，具体格式见指令系统表。应该指出的是，各条指令的编码形式可以多种多样。为了叙述方便，下面采用汇编符号对指令进行描述，其中 R1 和 R2 分别表示“目标”和“源”寄存器，M 表示地址在寄存器C 中的存贮单元。

表1 指令系统表

汇编符号	功能	编码
MOV R1，R2	（R2）→ R1	1111 R1 R2
MOV M，R2	（R2）→（C）	1111 11 R2
MOV R1，M	（（C））→（R1)	1111 R1 11
ADD R1，R2	（R1）+（R2）→ R1	1001 R1 R2
SUB R1，R2	（R1）-（R2）→ R1	0110 R1 R2
OR 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	无操作	0111 00 00
HALT	停机	1000 00 00

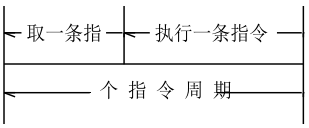
（4）数据通路设计

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。



（5）**指令周期与工作脉冲设置**

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。

整个系统由分部件构成，分部件之间通过导线连接，而各个分部件又实现各自的功能，同样包括时序部件和组合部件：

组合部件：

（1）指令译码器

功能：通过指令系统表设计一个由指令码映射到指令信号的译码器。对 IR 传送的机器码进行译码，产生相应的控制信号以及为下一个部件提供输入，其中指令码为 8 位二进制码，译码器的输出端为指令操作。

（2）选择器

功能：从多个输入：PC，寄存器输出 S 口，寄存器输出 D 口中选择一个输入，并将信息直接传输到 RAM。

（3）控制器

功能：根据译码器传送来的输入，cf，cz 标志以及 IR 传送来的输入产生各个分部件的控制信号，指导各个分部件工作。

（4）移位逻辑

功能：移位逻辑需要实现 RSR、RSL 左右移位操作，还需提供 MOVA、MOVB、ADD、SUB、OR、NOT、OUT 指令执行时，将数据传输至 BUS 总线的通路。

（5）函数发生器 ALU

功能：从通用寄存器组中读取内容；在 S3～ S0 和 M 的控制下，实现运算功；ADD 和 SUB 加减法指令影响状态位 Cf 和 Zf。

时序部件：

（1）指令寄存器 IR

功能：将总线上传送来的数据进行存储，并传送给下一个部件译码器。

（2）cf 和 zf 寄存器

功能：将 c 标志和 z 标志存储，并传送给部件控制器。

（3）指令计数器 PC

功能：当执行一条指令时，首先需要根据 PC 中存放的指令地址，将指令由 RAM 读取至指令寄存器 IR 中，此过程称为“取指令”，与此同时，PC 中的地址自动加 1。跳转指令如 JMP、JZ、JC 让程序跳转至指定地址去执行，这时 PC 需要装载跳转地址。

（4）时钟发生器 SM

功能：模型机中所有指令的执行都是一个周期完成取指令，一个周期执行指令。SM 则用来区分当前周期是取指令还是执行

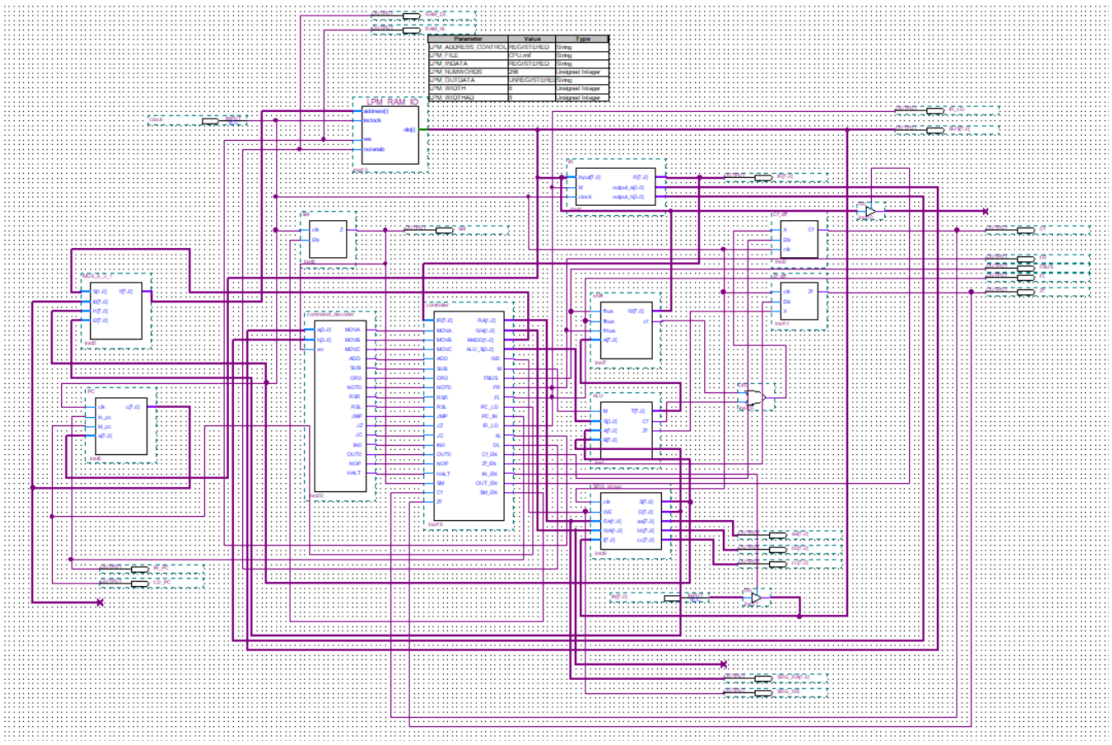
（5）通用寄存器组

功能：寄存器具有快速读写的功能特点，而通用寄存器组则是实现的对三块寄存器的读写操作，根据控制信号对 A，B，C 寄存器实现读写，并将数据传送给 ALU 函数发生器或选择器。

（6）RAM

功能：RAM 可以根据 quartus 直接生成 LPM_RAM_IO，实现读外接文件 mif 的读写。

整体设计展示：



3.2 各模块的具体实现

（此部分必须有模块的接口设计，功能实现，功能的仿真验证等内容。）

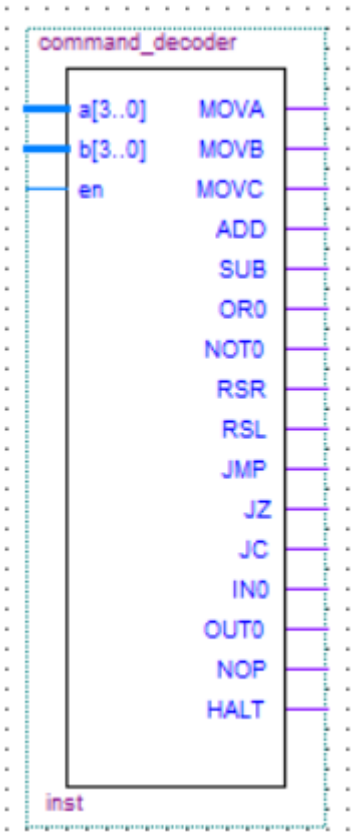
组合部件：

1. 指令译码器：

（1）部件功能：

通过指令系统表设计一个由指令码映射到指令信号的译码器。对 IR 传送的机器码进行译码，产生相应的控制信号以及为下一个部件提供输入，其中指令码为 8 位二进制码，译码器的输出端为指令操作。

（2）接口设计：



输入：

a,b 表示由 IR 传送来的数据

EN 是使能信号

输出：

MOVA 用来传送（R2）→ R1 操作的使能信号

MOVB 用来传送（R2）→（C）操作的使能信号

MOVC 用来传送（（C））→R1 操作的使能信号

ADD 用来传送（R1）+（R2）→R1 操作的使能信号

SUB 用来传送（R1）-（R2）→ R1 操作的使能信号

OR0 用来传送（R1）∨（R2）→ R1 操作的使能信号

NOT0 用来传送 R1 /（R1）→ R1 操作的使能信号

RSL 和 RSR 用来传送循环左右移位操作的使能信号

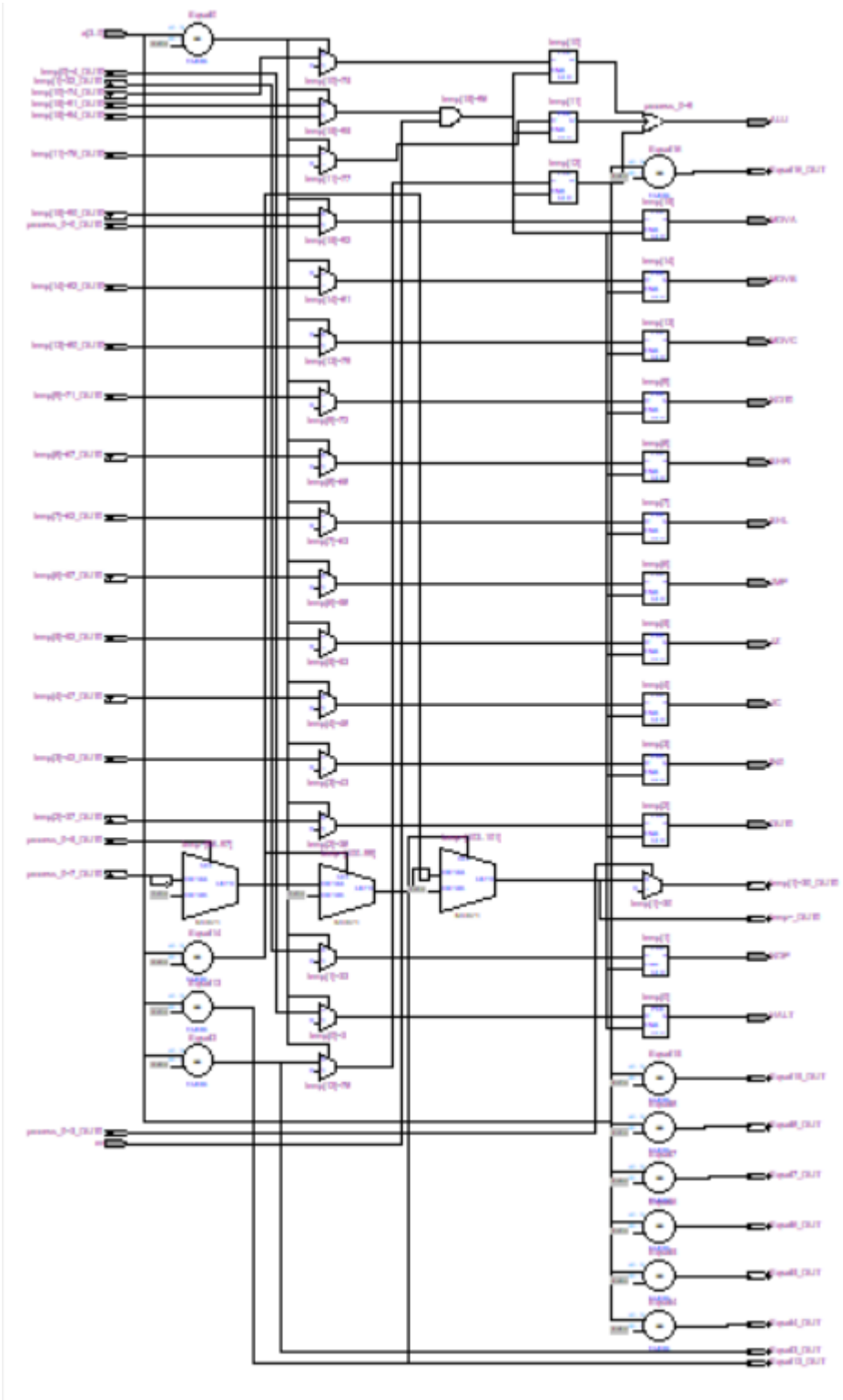
JMP，JC，JZ 则用来传送 PC 跳转到指定 address 操作的使能信号

IN0 传送向 R1 写入操作的指令

OUT0 传送由寄存器中读出 R1 的操作的指令

HALT 传送停机指令

RTL 视图



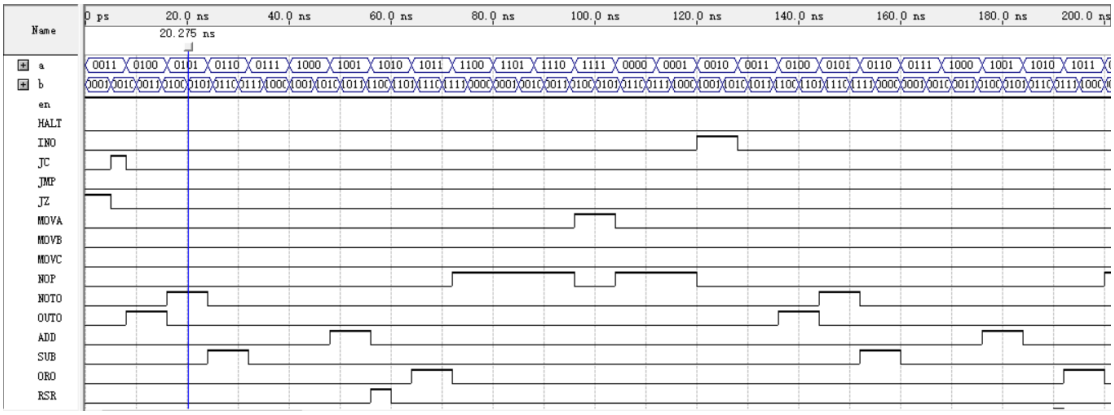
（3）功能实现：

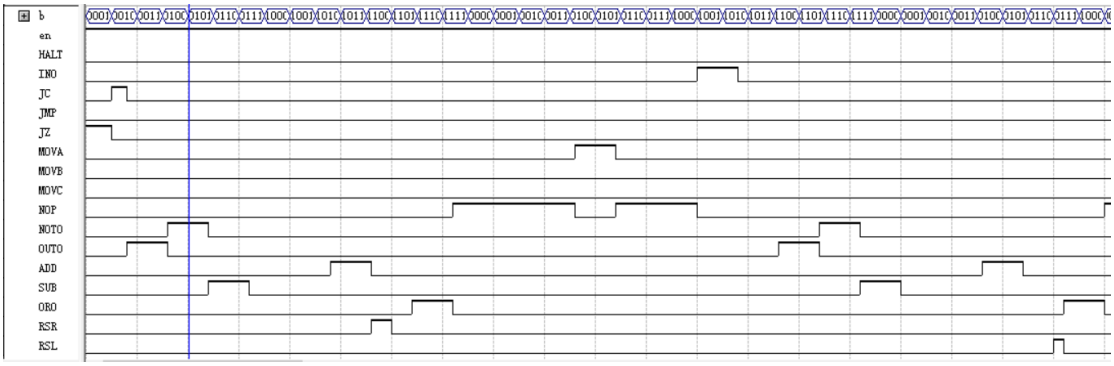
对每一种输入情况进行译码，对应每一种二进制编码，存储在一个临时变量 temp 里，在对 temp 讨论输出结果。

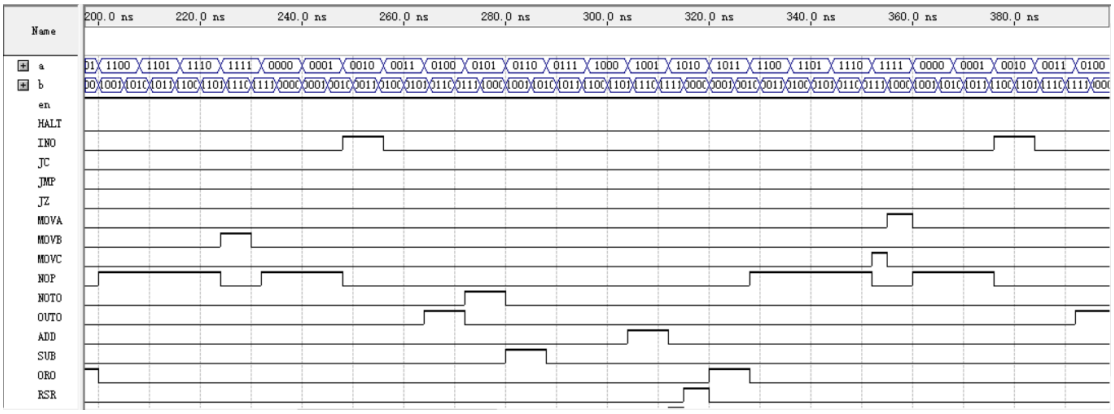
VHDL 设计：

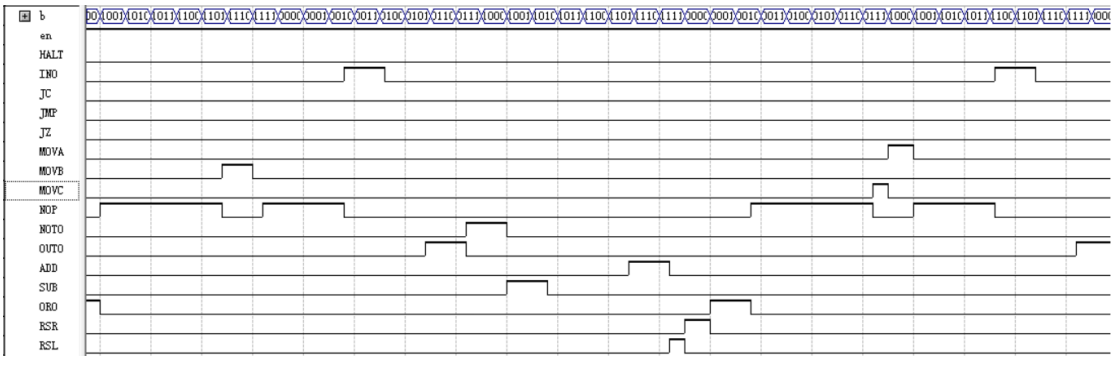
```
library ieee;
use ieee.std_logic_1164.all;

entity command_decoder is
port(a,b: in std_logic_vector(3 downto 0);
      EN: in std_logic;
      MOVA,MOVB,MOVC,ADD,SUB,OR0,NOT0,RSR,RSL,JMP,JZ,JC,IN0,OUT0,NOP,HALT:out std_logic);
end command_decoder;

architecture structural of command_decoder is
     signal temp: std_logic_vector(15 downto 0):="0000000000000000";
     signal prev: std_logic_vector(3 downto 0);
     signal r1,r2: std_logic_vector(1 downto 0);
begin
     temp<="0000000000000000";
     prev<=a;
     r1<=a(3)&a(2);
     r2<=b(1)&b(0);
     if EN='1' then
         case prev is
         when "1111"=>
             if r1="11" and r2="11" then temp<="1000000000000000";
             elsif r1="11" and r2="11" then temp<="0010000000000000";
             elsif r1 ="11" and r2="11" then temp<="0100000000000000";
             end if;
         when "1001"=> temp<="0001000000000000";
         when "0110"=> temp<="0000100000000000";
         when "1011"=> temp<="0000010000000000";
         when "0101"=> temp<="0000001000000000";
         when "1010"=>
             if r2="00" then temp<=   "0000000100000000";
             elsif r2="11" then temp<="0000000010000000";
             end if;
         when "0011"=>
         if r1="00" then
             if r2="00" then temp<=   "0000000001000000";
             elsif r2="01" then temp<="0000000000100000";
             elsif r2="10" then temp<="0000000000010000";
             end if;
         end if;
         when "0010"=> temp<="0000000000001000";
         when "0100"=> temp<="0000000000000100";
         when "0111"=>
         if(r1="00" and r2="00") then temp<="0000000000000010";
         end if;
         when "1000"=>
         if(r1="00" and r2="00") then temp<="0000000000000001";
         end if;
         when others=>temp<="0000000000000010";
         end case;
     end if;

     if temp(15)='1' then MOVA <='1';
     else MOVA<='0';
     end if;

     if temp(14)='1' then MOVB <='1';
     else MOVB<='0';
     end if;

     if temp(13)='1' then MOVC <='1';
     else MOVC<='0';
     end if;

     if temp(12)='1' then ADD<='1';
     else ADD<='0';
     end if;
     if temp(11)='1' then SUB<='1';
     else SUB<='0';
     end if;
     if temp(10)='1' then OR0<='1';
     else OR0<='0';
     end if;

     if temp(9)='1' then NOT0 <='1';
     else NOT0<='0';
     end if;

     if temp(8)='1' then RSR <='1';
     else RSR<='0';
     end if;

     if temp(7)='1' then RSL <='1';
     else RSL<='0';
     end if;

     if temp(6)='1' then JMP <='1';
     else JMP<='0';
     end if;

     if temp(5)='1' then JZ <='1';
     else JZ<='0';
     end if;

     if temp(4)='1' then JC <='1';
     else JC<='0';
     end if;

     if temp(3)='1' then IN0 <='1';
     else IN0<='0';
     end if;

     if temp(2)='1' then OUT0 <='1';
     else OUT0<='0';
     end if;

     if temp(1)='1' then NOP<='1';
     else NOP<='0';
     end if;

     if temp(0)='1' then HALT <='1';
     else HALT<='0';
     end if;
   end process;
end structural;
```

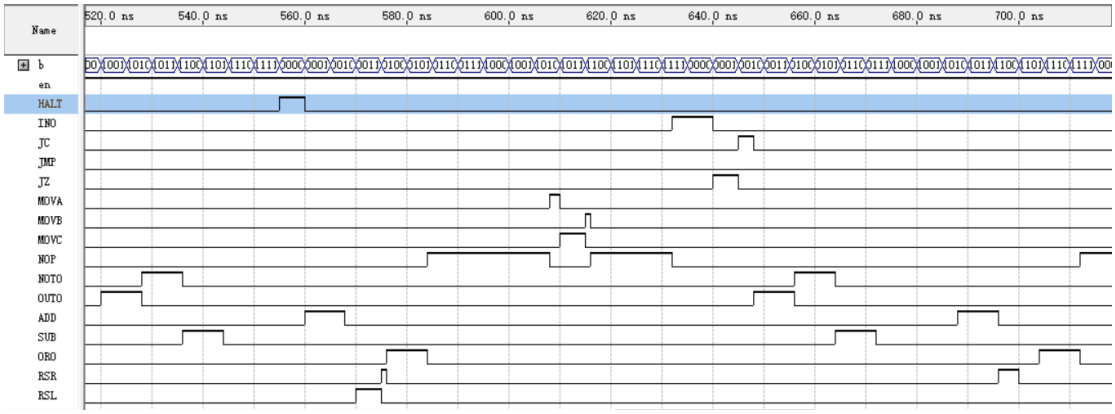
（4）功能仿真验证：


Name
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
140.0 ns
160.0 ns
180.0 ns
200.0 ns
20.275 ns
a
b
en
HALT
INO
JC
JMP
JZ
MOVA
MOVB
MOVC
NOP
NOTO
OUTO
ADD
SUB
ORO
RSR


b
en
HALT
INO
JC
JMP
JZ
MOVA
MOVB
MOVC
NOP
NOTO
OUTO
ADD
SUB
ORO
RSR
RSL


Name
200.0 ns
220.0 ns
240.0 ns
260.0 ns
280.0 ns
300.0 ns
320.0 ns
340.0 ns
360.0 ns
380.0 ns
a
b
en
HALT
INO
JC
JMP
JZ
MOVA
MOVB
MOVC
NOP
NOTO
OUTO
ADD
SUB
ORO
RSR


b
en
HALT
INO
JC
JMP
JZ
MOVA
MOVB
MOVC
NOP
NOTO
OUTO
ADD
SUB
ORO
RSR
RSL



仿真结果：

输入 1111 R1 R2 执行 MOV A 的操作，此时 MOVA=1

输入 1111 11 R2 执行 MOV B 的操作，此时 MOVB=1

输入 1111 R1 11 执行 MOVC 的操作，此时 MOVC=1

输入 1001 R1 R2 执行（R1）+（R2）→ R1 ADD=1

输入 0110 R1 R2 执行（R1）-（R2）→ R1 SUB=1

输入 1011 R1 R2 执行（R1）∨（R2）→ R1 OR0=1

输入 0101 R1 执行 /（R1）→ R1 NOT0=1

输入 1010 R1 00 执行循环右移一位 RSR=1

输入 1010 R1 11 执行循环左移一位 RSL=1

输入 0011 00 00，JMP 为 1

输入 0011 00 01，JZ 为 1

输入 0011 00 10，JC 为 1

输入 0010 R1XX 执行输入操作 IN0=1

输入 0100 R1XX 执行输出操作 OUT0=1

输入 01110000 执行 NOP 操作 NOP=1

输入 10000000 执行停机操作 HALT=1

对于其他输入，都译码为 NOP，跳过这条指令，跳转到下一条指令，NOP=1

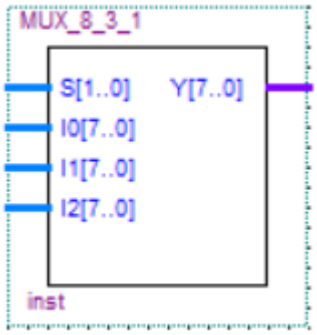
结论：仿真结果满足功能要求，设计正确

2. 选择器

（1）部件功能：

从多个输入：PC，寄存器输出 S 口，寄存器输出 D 口中选择一个输入，并将信息直接传输到 RAM。

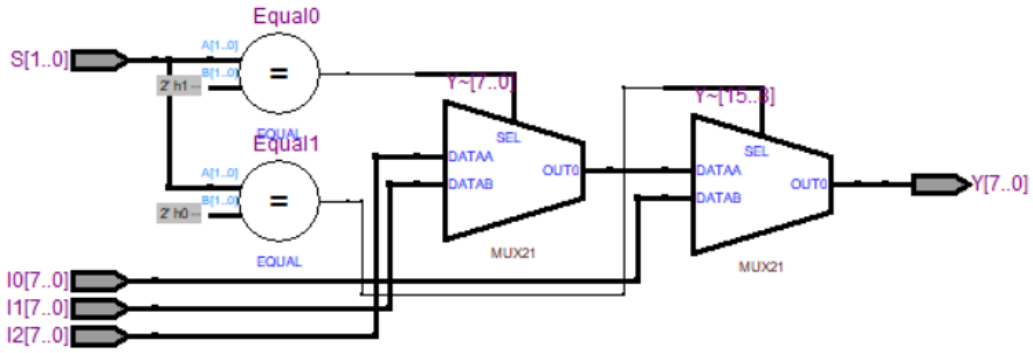
（2）接口设计：



输入：S 表示选择信号，I0，I1，I2 分别是来自 PC，寄存器 S 口，寄存器 D 口传送来的数据。

输出：Y 表示选择结果

RTL 视图



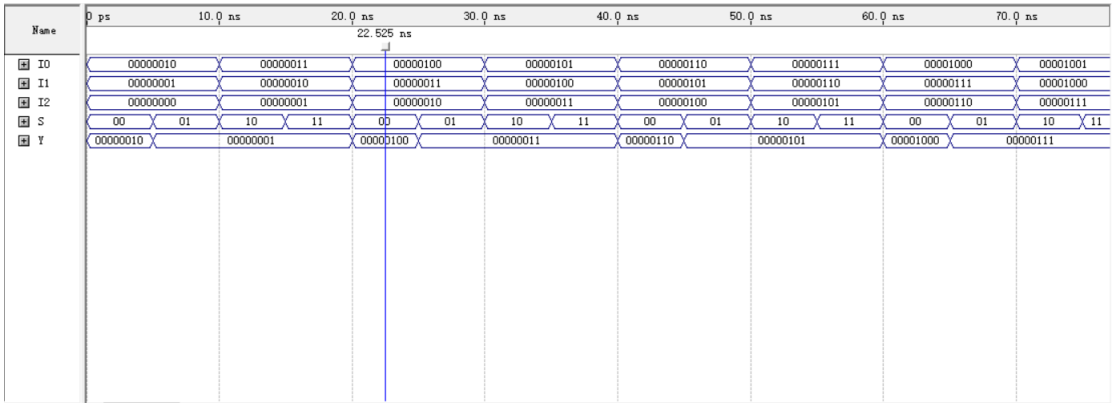
（3） 功能实现：

当 S=00，选择 I0，当 S=01 选择 I1，当 S=10 选择 I2

VHDL 设计：

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity MUX_8_3_1 is
4      port(S:in std_logic_vector(1 downto 0);
5           I0,I1,I2:in std_logic_vector(7 downto 0);
6           Y:out std_logic_vector(7 downto 0));
7  end entity;
8  architecture behavior of MUX_8_3_1 is
9  begin
10     Y<=I0 when S="00" else
11        I1 when S="01" else
12        I2 when S="10" else
13        "XXXXXXXX";
14 end behavior;
```

（4）功能仿真验证：



仿真结果：

时钟为 0.0ns 时,输入 S=00,选择输出 I0，结果为 Y=”00000010”

时钟为 5.0ns 时,输入 S=01,选择输出 I1，结果为 Y=”00000001”

时钟为 10.0ns 时,输入 S=10,选择输出 I2，结果为 Y=”00000001”

时钟为 15.0ns 时,输入 S=11,不进行选择，结果为 Y=”00000001”

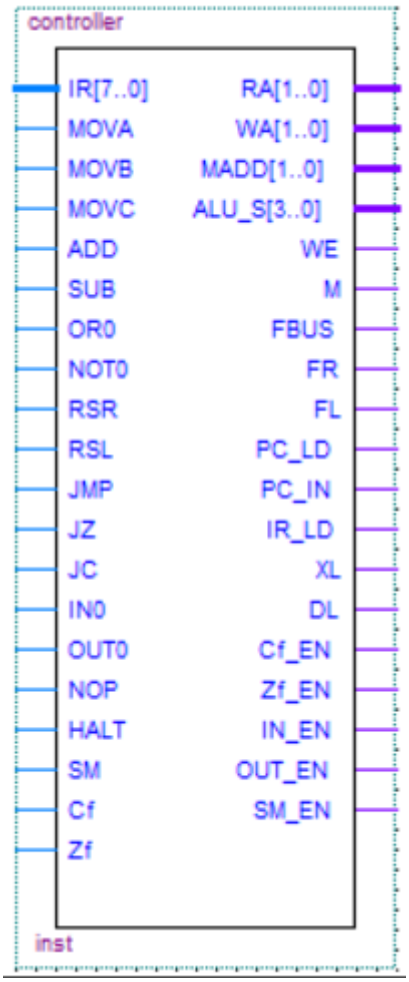
结论：仿真结果满足功能要求，设计正确

3. 控制器：

（1）部件功能：

根据译码器传送来的输入，cf，cz 标志以及 IR 传送来的输入产生各个分部件的控制信号，指导各个分部件有序工作。

（2）接口设计：



输入：

MOVA,MOVB,MOVC,ADD,SUB,OR0,NOT0,RSR,RSL,JMP,JZ,JC,IN0,OUT0,NOP,HALT 是由译码器传送的控制信号，用于控制信号的产生依据。

SM：由 SM 传送的时钟信号。

IR：由指令寄存器传送的 RAM 数据。

Cf,Zf：由 C 寄存器和 Z 寄存器传送的 C 标志和 Z 标志。

输出：

（1）WE，RA，WA 指向寄存器组的控制信号，指导寄存器组的读写操作。

（2）MADD：指向选择器的控制信号，指导选择器进行选择。

（3）ALU_S，M：指向函数发生器的控制信号，指导函数发生器工作。

（4）PC_LD,PC_IN：指向 PC 指令计数器的控制信号，指导 PC 进行加一和跳转操作。

（5）XL,DL：指向 RAM 的控制信号指导 RAM 进行读写或高阻状态。

（6）FBUS，FR，FL：指向移位逻辑的控制信号，指导移位逻辑进行左右移位，传送以及高阻状态的执行。

（7）IR_LD：指令寄存器的控制信号，指导 IR 进行载入。

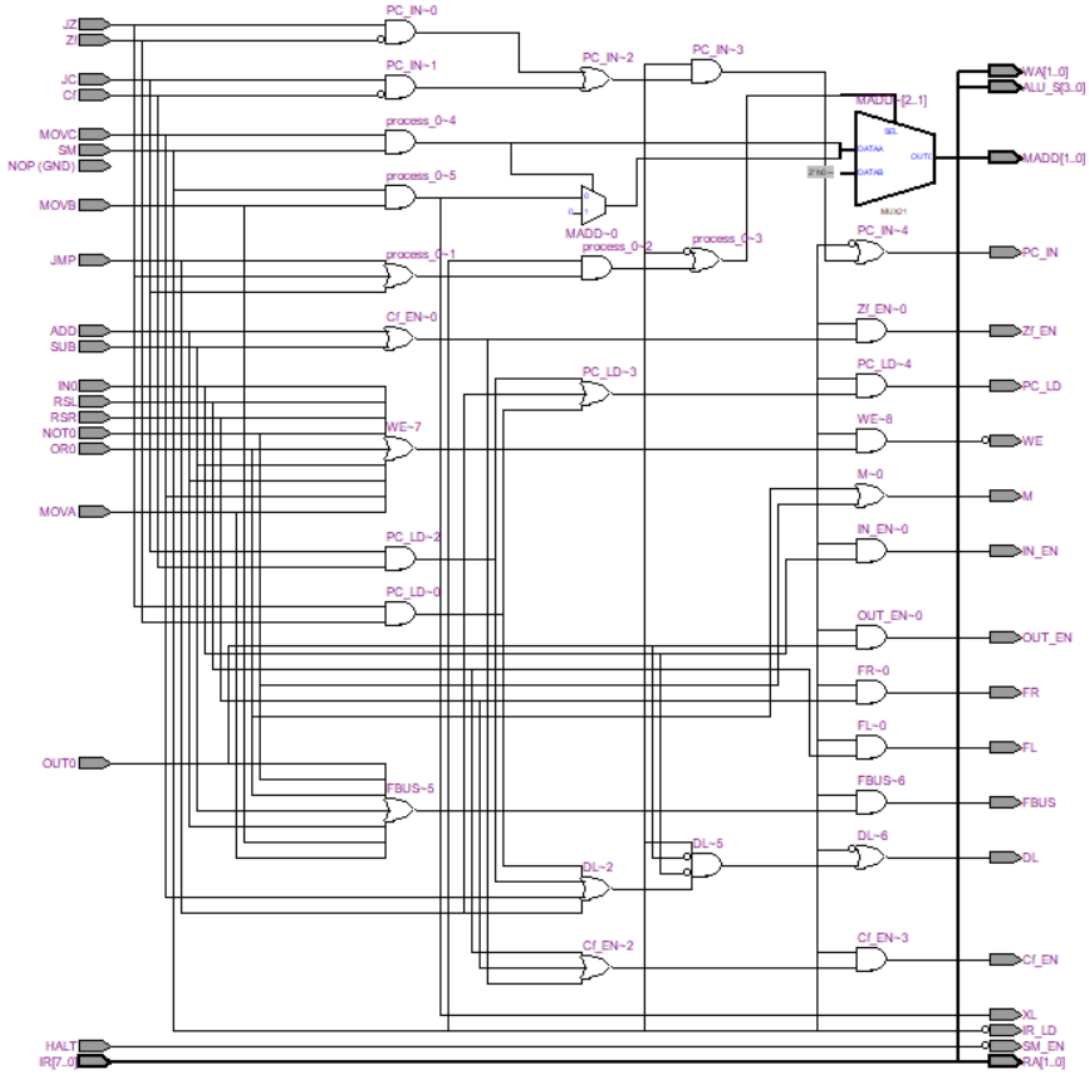
（8）CF_EN,ZF_EN：C,Z 寄存器的控制信号，指导 C，Z 寄存器的工作。

（9）SM_EN：SM 时钟发生器的控制信号，指导其工作。

（10）IN_EN:控制 IN0 指令的执行。

（11）OUT_EN:控制 OUT0 指令的执行。

RTL 视图



（3）功能实现

WE：当指令 MOVA，MOVC 传送指令，运算指令 ADD,SUB,OR0，取反 NOT0，左右移位 RSL，RSR，输入寄存器 IN0，以及 SM 为 1 时，WE 为 0，表示执行写入操作。

WE<=not((MOVA or MOVC or ADD or SUB or OR0 or NOT0 or RSR or RSL or IN0) and SM)

RA：取机器码 IR(1 downto 0)位，选择对 ABC 三个寄存器的读取

WA：取机器码 IR(3 downto 2)位，选择对 ABC 的写入和读取

MADD：当 SM=‘0’或(JZ and Zf) or (JC and Cf) or JMP 为 1 时，MADD=“00”选择第一个输入，接通 PC，当 SM 为 1 且 MOVC=1 时，MADD=“01”接通 S 口，当 SM=1 且 MOVB=1 时 MADD=“10”接通 D 口。

ALU_S 取机器码 IR(7 downto 4)高四位，由此进行相应运算判断。

M：当 IR(7 downto 4)=“1011” or IR(7 downto 4)="0101"即执行或非运算时为 1。

PC_LD 当(JZ and Zf) or (JC and Cf) or JMP 为 1 时 PC_LD 为 1 执行跳转操作.

PC_IN 当(JZ and (not Zf)) or (JC and (not Cf)) or(not SM)为 1 执行 PC 加一操作.

XL 当传送指令 MOVB=1 时为 1 写入 RAM

DL 当 MOVC or JMP or(Z and JZ) or (Z and JC) or (not SM)为 1 时执行读取操作

FR<=RSR;右移

FL<=RSL;左移

FBUS 当 MOVA or MOVB or ADD or SUB or OR0 or NOT0 or OUT0 即执行 AB 类传送指令、算术逻辑运算以及输出指令时 FBUS 为 1 表示提供数据通道。

IR_LD：not SM 即当取址周期时 IR_LD 为 1 进行数据载入

CF_EN：当 ADD or SUB or RSR or RSL 为 1,即执行算术运算以及左右移位时可能产生标志位这时开放 CF_EN 通道。

ZF_EN：当 ADD or SUB 为 1，即执行算术运算时可能产生标志位这时开放 ZF_EN 通道。

SM_EN 当 not HALT 为 1，即非停机时 SM 时钟持续发生。

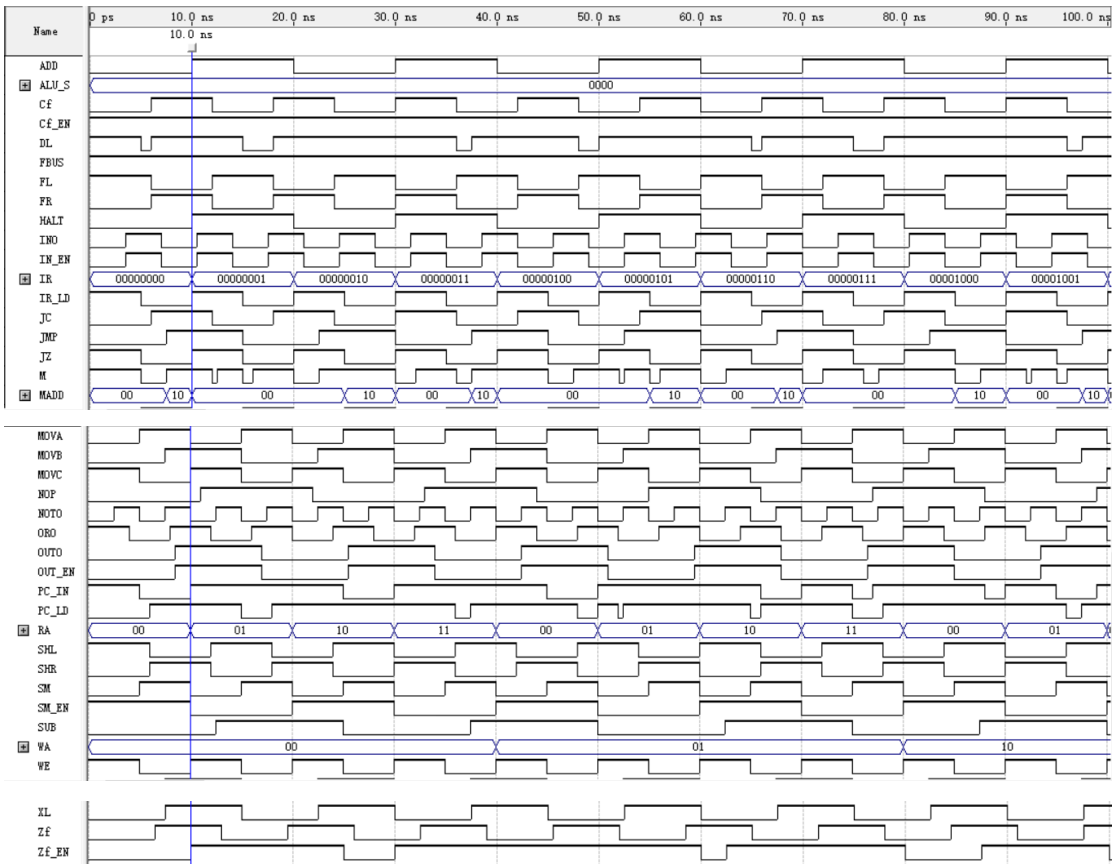
IN_EN:当 IN0 为 1 时。

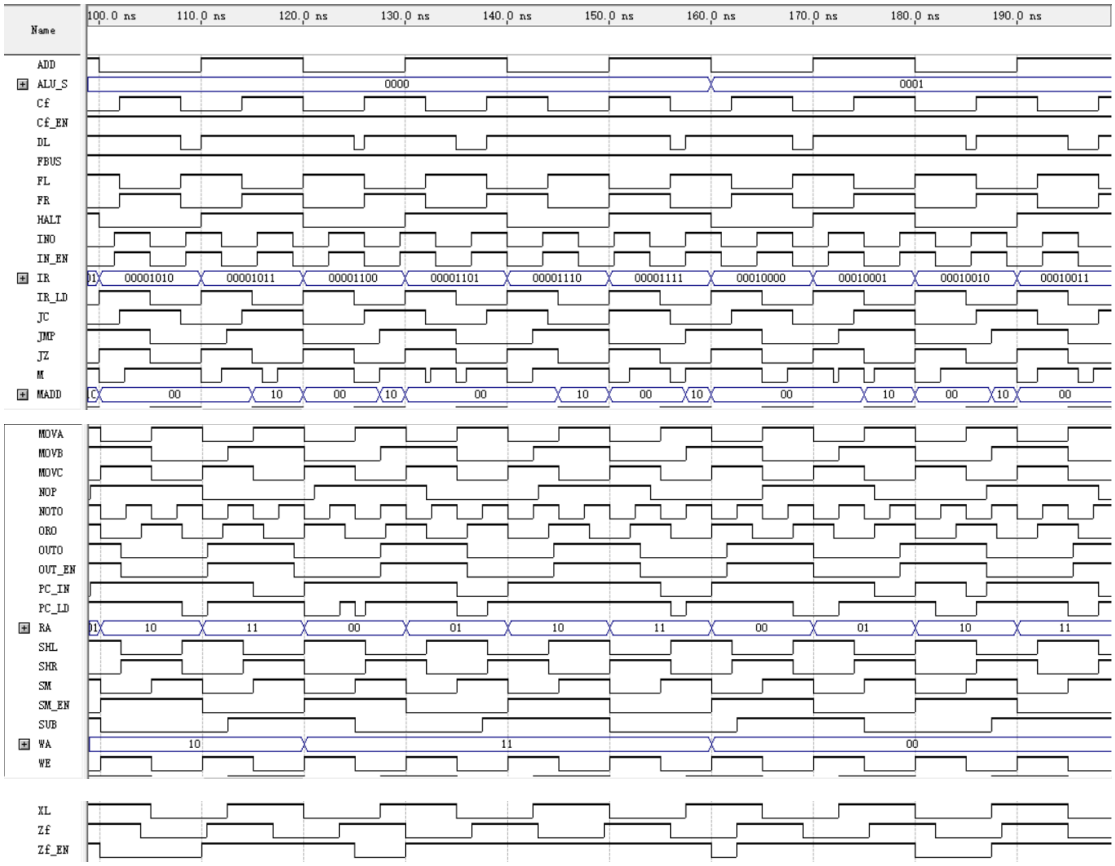
OUT_EN:当 OUT0 为 1 时。

VHDL 设计：

```
library ieee;
use ieee.std_logic_1164.all;
entity controller is
    port(IR:in std_logic_vector(7 downto 0):="00000000";
         MOVA,MOVB,MOVC,ADD,SUB,OR0,NOT0,RSR,RSL,JMP,JZ,JC,IN0,OUT0,NOP,HALT,SM,Cf,Zf:in std_logic:='0';
         RA,WA,MADD:out std_logic_vector(1 downto 0);
         ALU_S:out std_logic_vector(3 downto 0);
         WE,M,FBUS,FR,FL,PC_LD,PC_IN,IR_LD,XL,DL,Cf_EN,Zf_EN,IN_EN,OUT_EN:out std_logic;
         SM_EN:out std_logic:='1');
end controller;
architecture bhv of controller is
begin
process(MOVA,MOVB,MOVC,ADD,SUB,OR0,NOT0,RSR,RSL,JMP,JZ,JC,IN0,OUT0,NOP,HALT,IR,SM,Zf,Cf)
begin
RA<=IR(1 downto 0);
WA<=IR(3 downto 2);
ALU_S<=IR(7 downto 4);
if SM='0' or ((JC='1' OR JZ='1' OR JMP='1') and SM='1') then MADD<="00";
elsif MOVC='1' and SM='1' then MADD<="01";
elsif MOVB='1'and SM='1' then MADD<="10";
else MADD<="00";
end if;
M<=OR0 or NOT0;
WE<=not((MOVA or MOVC or ADD or SUB or OR0 or NOT0 or RSR or RSL or IN0) and SM);
FBUS<=(MOVA or MOVB or ADD or SUB or OR0 or NOT0 or OUT0) and SM;
FR<=RSR and SM;
FL<=RSL and SM;
Cf_EN<=(ADD or SUB or RSR or RSL) and SM;
Zf_EN<=(ADD or SUB) and SM;
IN_EN<=IN0 and SM;
OUT_EN<=OUT0 and SM;
SM_EN<=not HALT;
PC_LD<=(JMP or (JZ and Zf) or (JC and Cf)) and SM;
PC_IN<=(((JZ and not Zf) or (JC and not Cf)) and SM) or (not SM);
IR_LD<=not SM;
XL<=MOVB and SM;
DL<=((MOVC or JMP or (JC and Cf) or (JZ and Zf)) and (not IN0) and (not OUT0) and SM) or (not SM);
end process;
end bhv;
```

（4）功能仿真验证：





仿真结果：

时钟 15ns 时：指令 MOVA，MOVC，ADD，SUB，OR0，NOT0，RSR，RSL，IN0，SM=1，SRG_WE=1

任何时间都有 SRG_RA=IR 后 2 位，SRG_WA=IR 高 2 位

时钟 15ns：MUX_MADD：SM=‘0’，MUX_MADD=“00”；

时钟 20ns：SM 为 1 且 MOVC=1，MUX_MADD=“01”

时钟 10ns：SM=1 且 MOVB，MUX_MADD=“10”

任何时间 ALU_S=IR(7 downto 4)高四位

时钟 20ns：IR(7 downto 4)=“1011” or IR(7 downto 4)=“0101”，ALU_M=1

时钟 25ns：PC_LD=1，(Jz and Z) or (JC AND C) or JMP 为 1

时钟 15ns：PC_IN=1，(JZ and (NOT Z)) or (JC and (not C)) or NOP or(not SM)

时钟 20ns：RAM_XL=1,MOVB=1 时为 1 写入 RAM

时钟 25ns：RAM_DL=1,MOVC or JMP or(Z and JZ) or (Z and JC) or (not SM)为 1

时钟 45ns：shift_FR<=RSR

时钟 50ns：shift_FL<=RSL

时钟 50ns：shift_FBUS=1,MOVA or MOVB or ADD or SUB or OR0 or NOT0=1

时钟 15ns：IR_LD=1：not SM=1

时钟 55ns：CF_EN=1：ALU or RSR or RSL=1

时钟 60ns：ZF_EN=1：ADD or SUB=1

时钟 65ns：SM_EN=1,not HALT 为 1

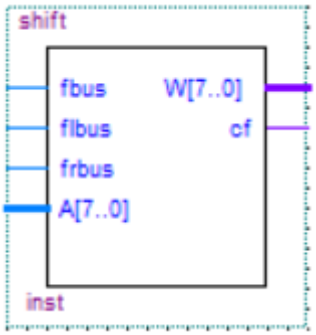
结论：仿真结果满足功能要求，设计正确

4. 移位逻辑

（1） 部件功能：

移位逻辑需要实现 RSR、RSL 左右移位操作，还需提供 MOVA、MOVB、ADD、SUB、OR、NOT、OUT 指令执行时，将数据传输至 BUS 总线的通路。

（2） 接口设计：



输入：

fbus：直通控制信号

frbus：左移控制信号

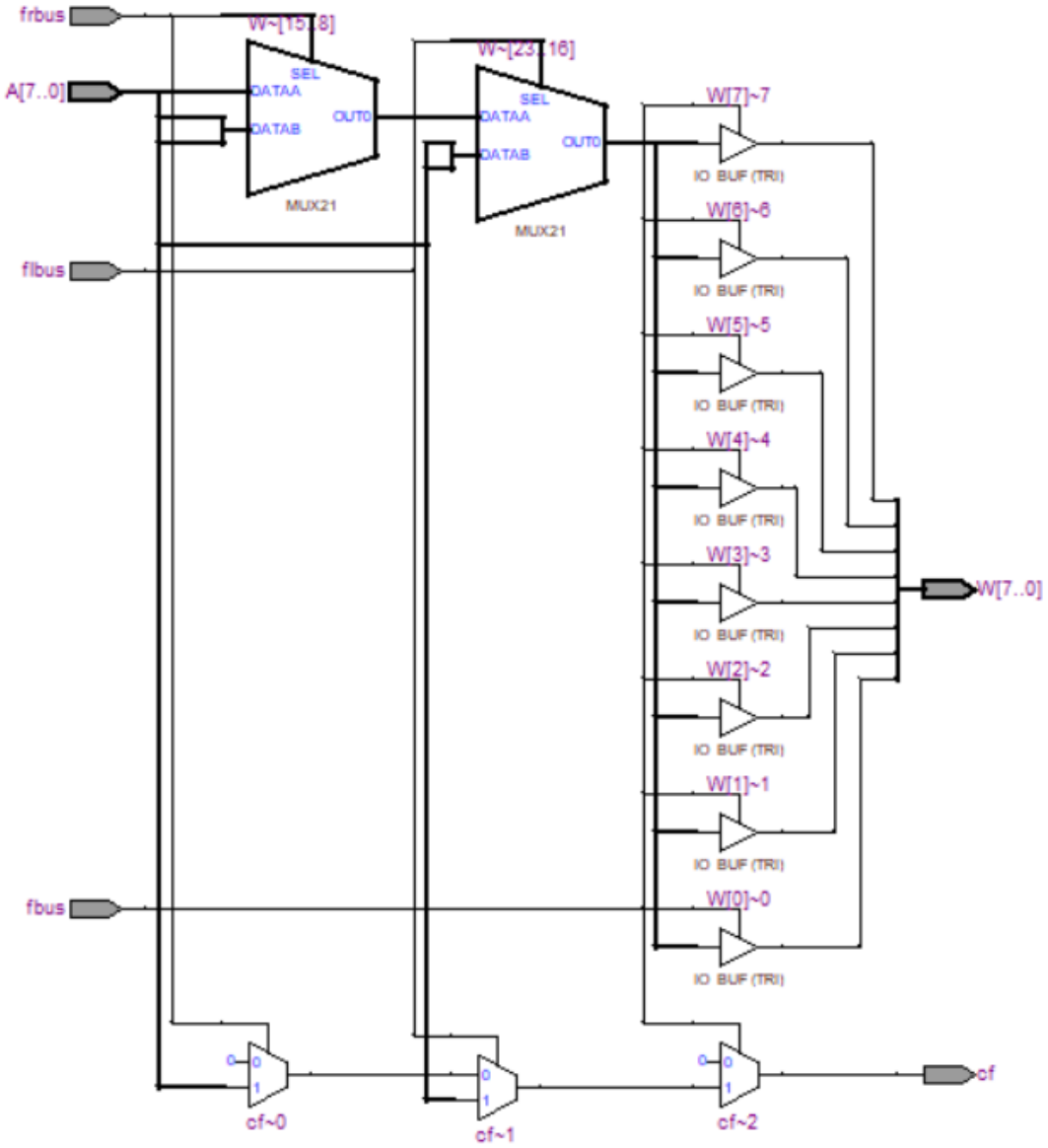
flbus：右移控制信号

输出：

W[7…0]：移位的结果

Cf：加减法产生的进退位

RTL 视图



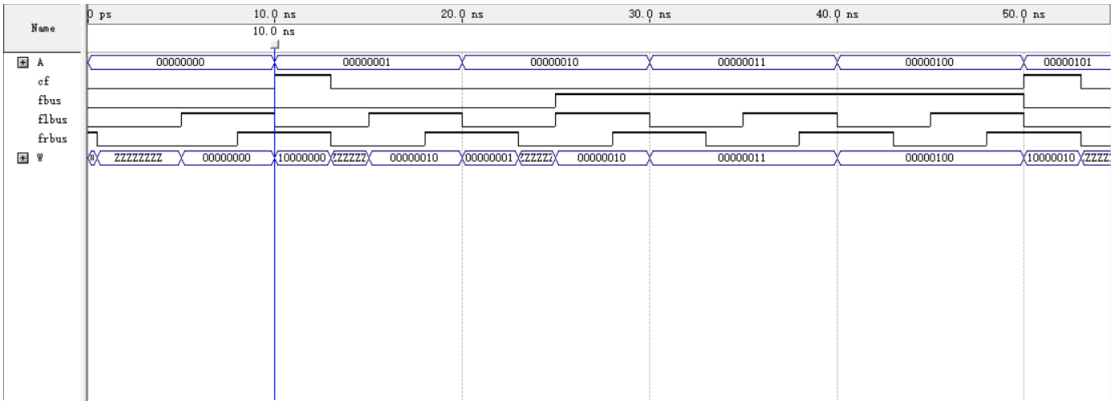
（3） 功能实现：

当 fbus=1 时，直通；当 flbus=1 时，执行左移位，将输入的 6 到 0 和 7 位赋值给输出；当 frbus=1 时，执行右移位，将输入的 0 和 7 到 1 位赋值给输出。并且判断移除的位是否为 1，是则 cf<=1，否则为 0；

VHDL 设计：

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity shift is
4       port(fbus,flbus,frbus:in std_logic;
5            A:in std_logic_vector(7 downto 0);
6            W:out std_logic_vector(7 downto 0);
7            cf:out std_logic:='0');
8   end shift;
9   architecture bhv of shift is
10  begin
11  process(fbus,flbus,frbus)
12      begin
13      if fbus='1' then
14          W<=A;
15          cf<='0';
16      elsif flbus='1' then
17          W<=A(6 downto 0)&A(7);
18          cf<=A(7);
19      elsif frbus='1' then
20          W<=A(0)&A(7 downto 1);
21          cf<=A(0);
22      else
23          W<="ZZZZZZZZ";
24          cf<='0';
25      end if;
26  end process;
27  end bhv;
```

（4）功能仿真验证：



仿真结果：

时钟为 3.0ns：fbus=0，frbus=0，flbus=0，输出高阻 W=ZZZZZZZZ，不产生进位 cf=0；

时钟为 10.0ns：frbus=1，执行右移输出 W=10000000，产生进位 cf=1；

时钟为 15.0ns，flbus=1，执行左移输出 W=00000010，cf=0；

时钟为 44.0ns，fbus=1，直通输出 W=00000100；

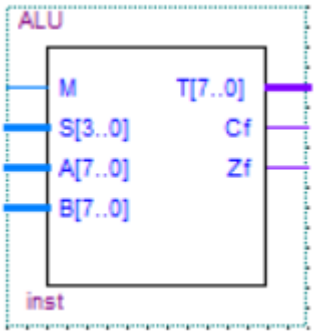
结论：仿真结果满足功能要求，设计正确

5. 函数发生器 ALU

（1）部件功能：

从通用寄存器组中读取内容;在 S3～S0 和 M 的控制下，实现运算功能;ADD 和 SUB 加减法指令影响状态位 Cf 和 Zf。

（2）接口设计：



输入：

S[3…0]为操作类型判断输入

M 是逻辑运算的使能信号

A[7…0]8 位数据接入

B[7…0]8 位数据接入

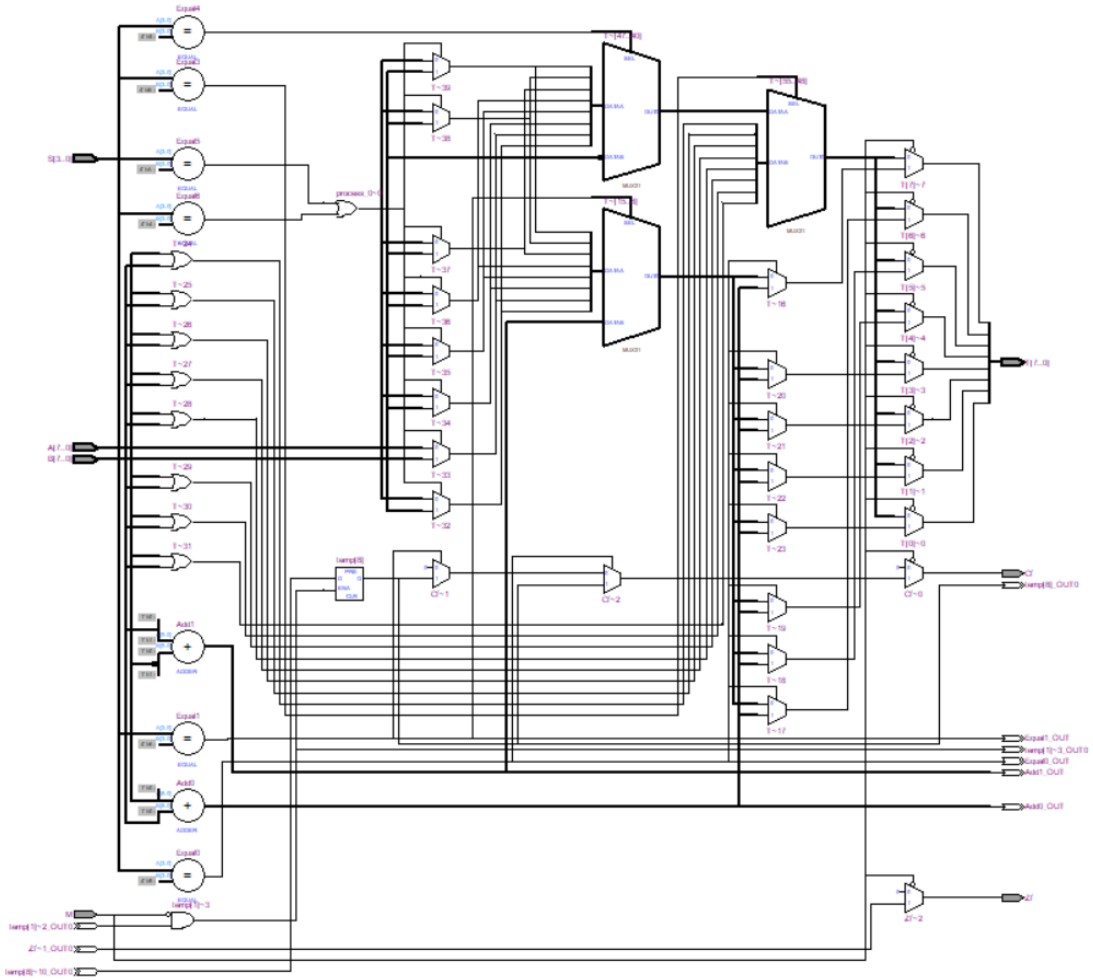
输出：

T[7…0]操作的结果

Cf 进位标志

Zf 结果为 0 标志

RTL 视图



（3）功能实现：

判断 M 输入和 S 输入，如果

M=’0’,S=”1001”，执行加法，将 R1+R2 赋值给 R1

M=’0’,S=”0110”，执行减法，将 R1-R2 赋值给 R1

M=’1’,S=”1011”，执行或运算，将（R1）∨（R2）赋值给 R1

M=’1’,S=”0101”，执行取反运算，/R1 赋值给 R1

判断加减法是否有最高位的进位或借位，如果有那就将 cf=1，否则为 cf=0

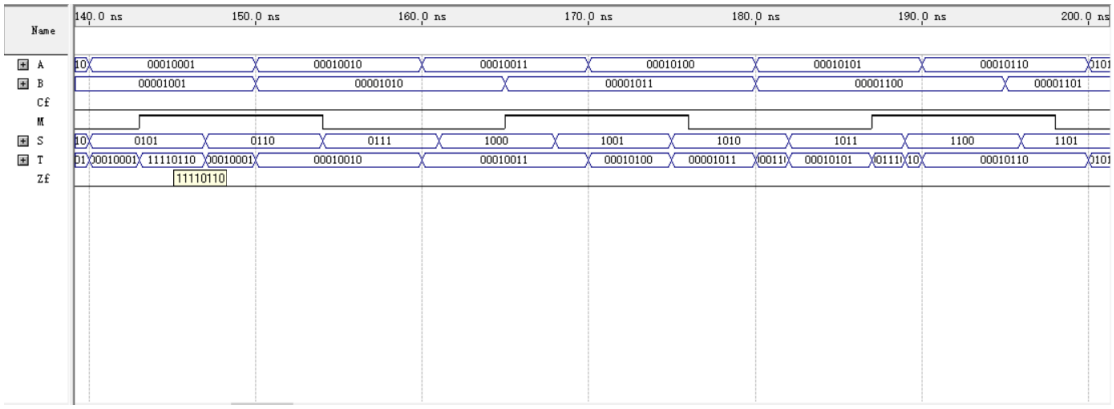
判断加减操作的结果是否为 0，如果是那就将 zf=1，否则 zf=0

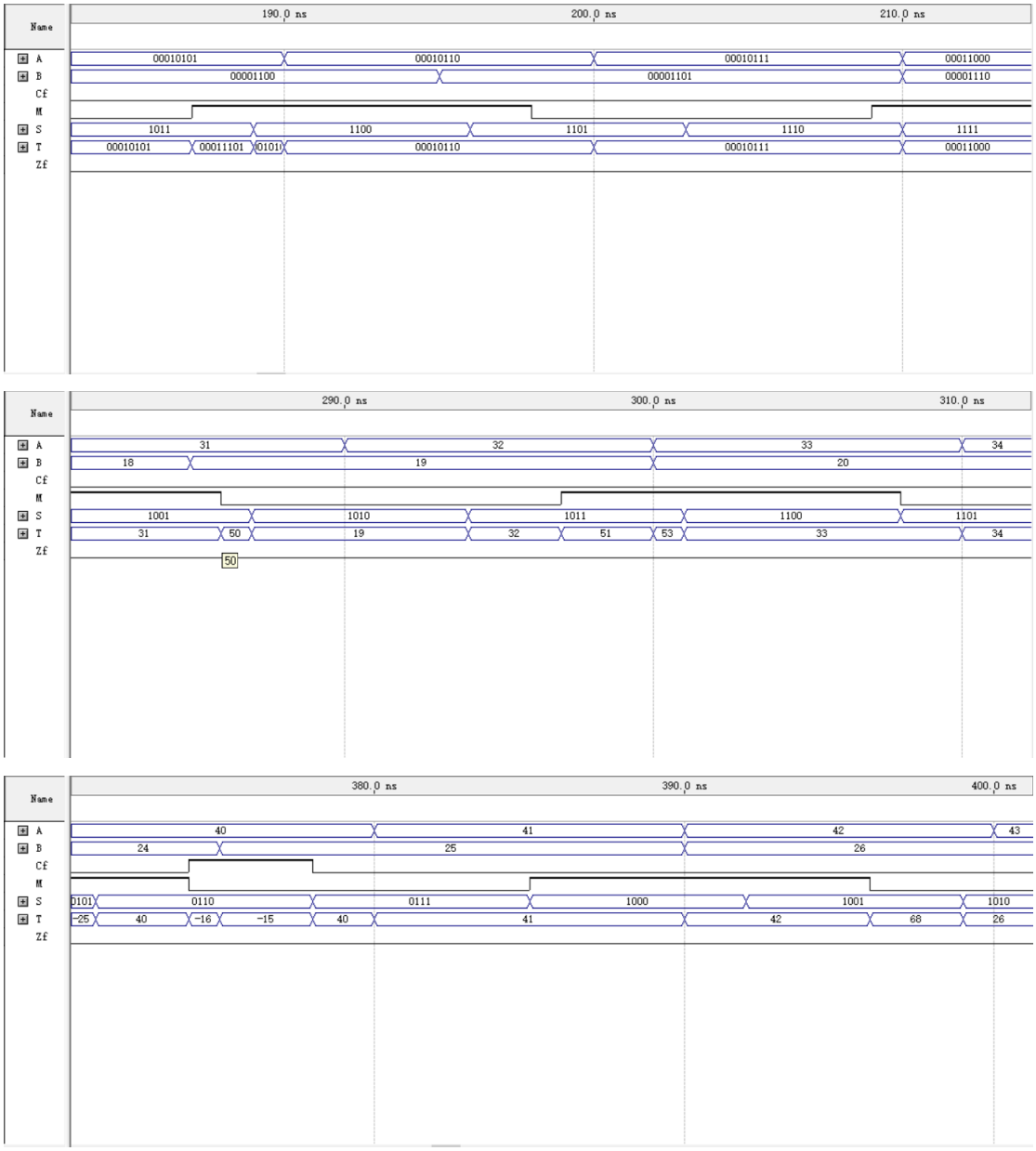
VHDL 设计：

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  entity ALU is
5      port(M:in std_logic;
6           S:in std_logic_vector(3 downto 0);
7           A,B:in std_logic_vector(7 downto 0);
8           T:out std_logic_vector(7 downto 0);
9           Cf:out std_logic:='0';
10          Zf:out std_logic:='0');
11 end ALU;
12 architecture bhv of ALU is
13 signal temp:std_logic_vector(8 downto 0);
14 begin
15     process(M,S,A,B)
16     begin
17     if M='0' then
18         if S="1001" then
19             T<=A+B;
20             temp<=('0'&A)+('0'&B);
21             Cf<=temp(8);
22             if temp="000000000" then Zf<='1';
23             else Zf<='0';
24             end if;
25         elsif S="0110" then
26             T<=B-A;
27             temp<=('0'&B)-('0'&A);
```

```
28                  Cf<=temp(8);
29                  if temp="000000000" then Zf<='1';
30                  else Zf<='0';
31                  end if;
32              else
33                  if S="1010" or S="0100" then T<=B;
34                  else T<=A;
35                  end if;
36                  Cf<='0';
37                  Zf<='0';
38              end if;
39          elsif M='1' then
40              if S="1011" then
41                  T<=A or B;
42              elsif S="0101" then
43                  T<= not B;
44              else
45                  if S="1010" or S="0100" then T<=B;
46                  else T<=A;
47                  end if;
48              end if;
49              Cf<='0';
50              Zf<='0';
51          end if;
52          end process;
53      end bhv;
```

（4）功能仿真验证：





仿真结果：

时钟为 143ns 时，A=00010001，B=00001001，此时 M=1,S=0101，执行非运算，T=11110110，cf=0，zf=0；

时钟为 187ns 时：A=00010101，B=00001100，此时 M=1,S=1011，执行或运算，结果 T=00011101，Cf=0，zf=0；

时钟为 286ns 时：A=31，B=19，此时 M=0,S=1001，执行加法，输出为 T=50，此时 cf=0；

时钟为 374ns 时：A=40，B=24，此时 M=0,S=0110，执行减法，输出 T=-16，进位 cf=1；

结论：仿真结果满足功能要求，设计正确

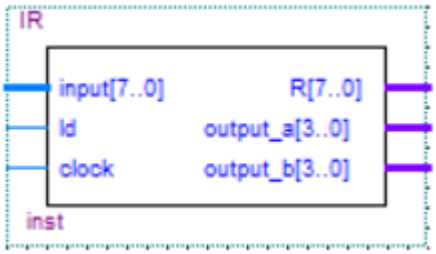
时序部件：

1. 指令寄存器 IR

（1）部件功能：

将总线上传送来的数据进行存储，并传送给下一个部件指令译码器。

（2）接口设计：



输入：

input[7…0]数据输入

clock 时钟信号

ld 控制信号，为 1 有效

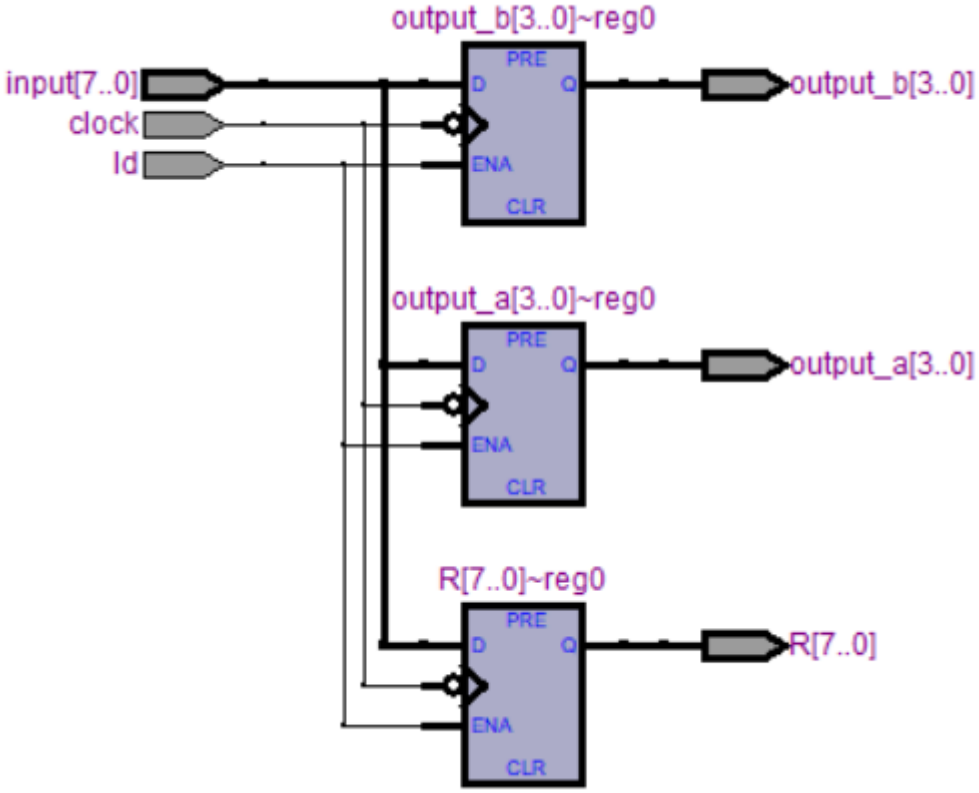
输出：

output_a[3…0]输出寄存器值的高四位

output_b[3…0]输出寄存器值的低四位

R[7…0]输出寄存器存储的值

RTL 视图



（3）功能实现：

当时钟下降沿时,判断 ld,如果 ld=1,那么将输入载入寄存器,否则不进行载入

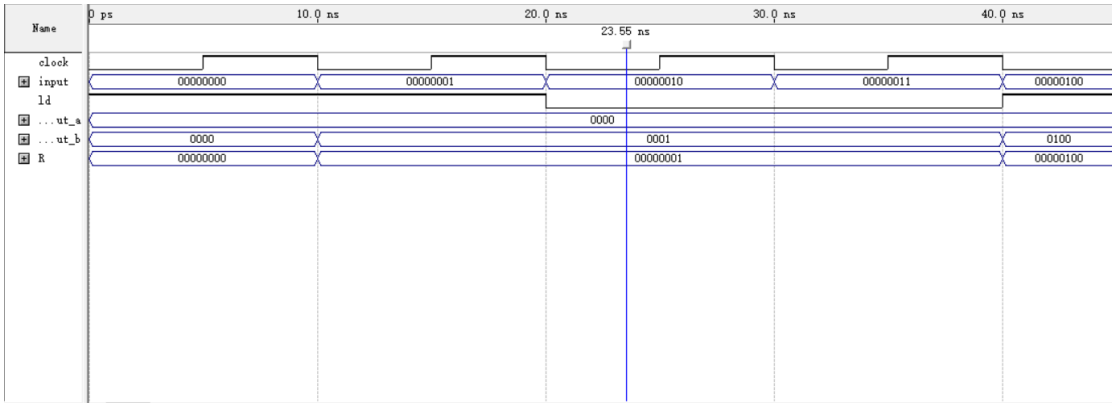
将存储值的高四位赋值给 output_a[3…0]输出寄存器值的高四位

output_b[3…0]输出寄存器值的低四位，R[7…0]输出寄存器存储的值。

VHDL 设计：

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity IR is
4      port(input:in std_logic_vector(7 downto 0);
5           ld,clock:in std_logic;
6           R:out std_logic_vector(7 downto 0);
7           output_a:out std_logic_vector(3 downto 0):="0000";
8           output_b:out std_logic_vector(3 downto 0):="0000");
9  end IR;
10 architecture bhv of IR is
11 begin
12     process(clock,ld)
13     begin
14         if clock'event and clock='0' then
15             if ld='1' then
16                 R<=input;
17                 output_a<=input(7 downto 4);
18                 output_b<=input(3 downto 0);
19             end if;
20         end if;
21     end process;
22 end bhv;
```

（4）功能仿真验证：



仿真结果：

时钟为 10.0ns：input=00000001，ld=1，此时下降沿输出 R=00000001，output_a=0000,out_b=0001;

时钟为 20.0ns：input=00000010，ld=0，下降沿时，输出保持不变，R=00000001，output_a=0000,out_b=0001;

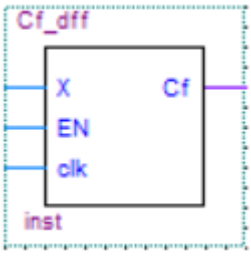
结论：仿真结果满足功能要求，设计正确

2. cf 寄存器

（1）部件功能：

将 c 标志保存，并传送给控制器。

（2）接口设计：



输入：

X：标志输入，数据来源

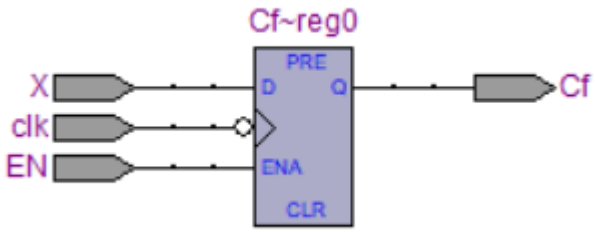
EN：使能控制信号，1 有效

clk 时钟信号

输出：

Cf，标志位对外输出

RTL 视图



（3）功能实现：

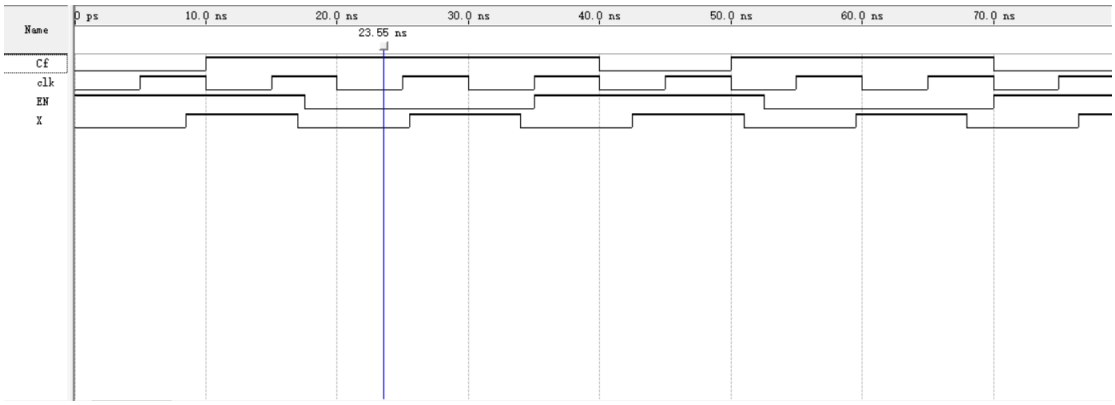
时钟下降沿时，判断 EN

如果 EN=1，载入数据，否则不进行载入

VHDL 设计：

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity Cf_dff is
4       port(X,EN,clk:in std_logic;
5            Cf:out std_logic);
6   end Cf_dff;
7   architecture bhv of Cf_dff is
8   begin
9   process(clk,en)
10  begin
11  if clk'event and clk='0' then
12      if EN='1' then Cf<=X;
13      end if;
14  end if;
15  end process;
16  end bhv;
```

（4）功能仿真验证：



仿真结果：

时钟为 10.0ns: 时钟下降沿，EN=1，X=1，将数据载入，输出 Cf=1

时钟为 20.0ns: 时钟下降沿，EN=0，数据不进行载入，输出保存的值 Cf=1，Cf 持续保持不变

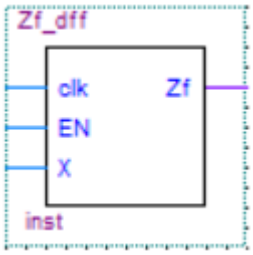
结论：仿真结果满足功能要求，设计正确.

zf 寄存器

（1）部件功能：

将 z 标志保存，并传送给控制器。

（2）接口设计：



输入：

X：标志输入，数据来源

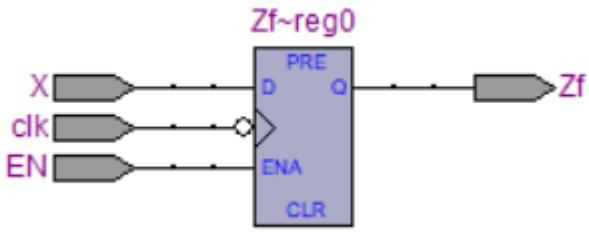
EN：使能控制信号，1 有效

clk：时钟信号

输出：

Zf：标志位对外输出

RTL 视图



（3）功能实现：

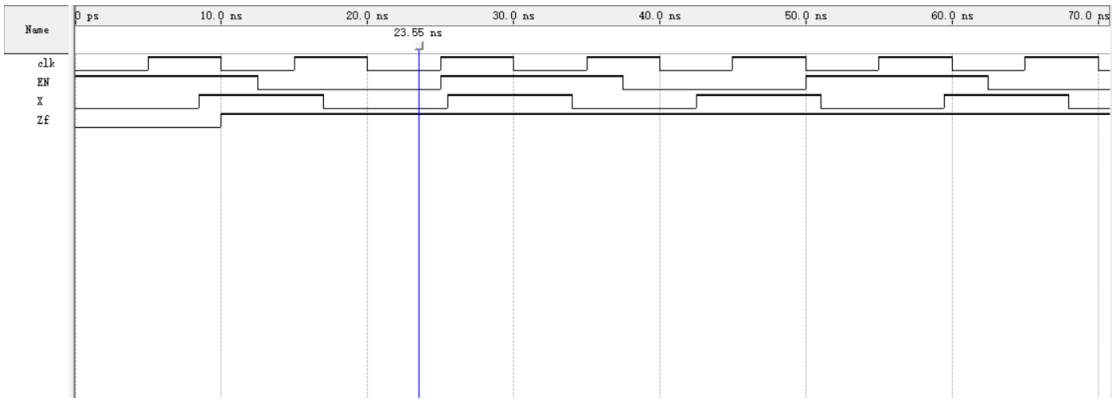
时钟下降沿时，判断 EN

如果 EN=1，载入数据，否则不进行载入

VHDL 设计：

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity Zf_dff is
4       port(clk,EN,X:in std_logic;
5            Zf:out std_logic);
6   end Zf_dff;
7   architecture bhv of Zf_dff is
8   begin
9   process(clk,EN)
10      begin
11      if clk'event and clk='0' then
12          if EN='1' then Zf<=X;
13          end if;
14      end if;
15  end process;
16  end bhv;
```

（4）功能仿真验证：



仿真结果：

时钟为 10.0ns: 时钟下降沿，EN=1，X=1，将数据载入，输出 Zf=1

时钟为 20.0ns: 时钟下降沿，EN=0，数据不进行载入，输出保存的值 Zf=1，Zf 持续保持不变

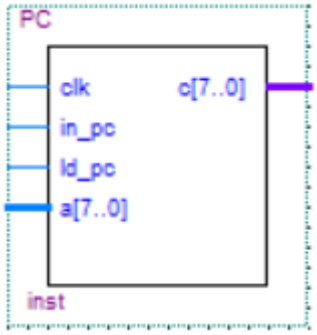
结论：仿真结果符合功能要求，设计成功。

3. 指令计数器 PC

（1）部件功能：

当执行一条指令时，首先需要根据 PC 中存放的指令地址，将指令由 RAM 读取至指令寄存器 IR 中，此过程称为“取指令”，与此同时，PC 中的地址自动加 1。跳转指令如 JMP、JZ、JC 让程序跳转至指定地址去执行，这时 PC 需要装载跳转地址。

（2）接口设计：



输入：

ld_pc:加载 pc 控制信号

in_pc: pc 加一控制信号

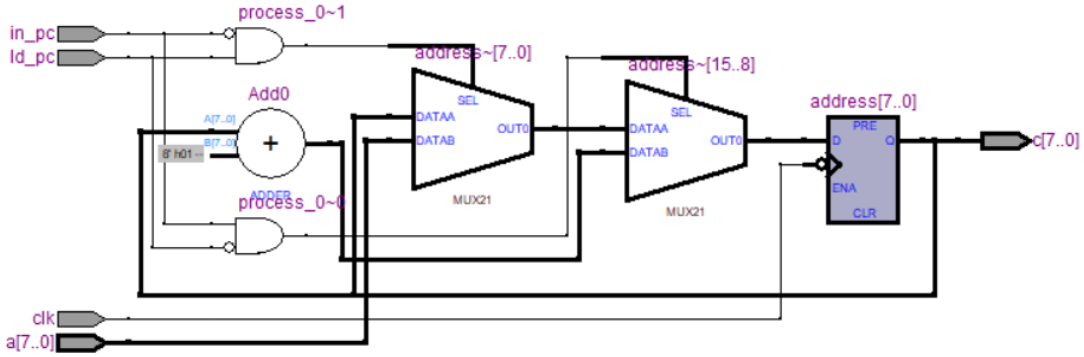
clk：时钟信号

a[7…0]:数据输入端口

输出：

c[7…0]:地址对外输出

RTL 视图



（3）功能实现：

在时钟下降沿进行判断：

如果 in_pc=1,ld_pc=0,进行 pc 加一操作，pc<=pc+1;

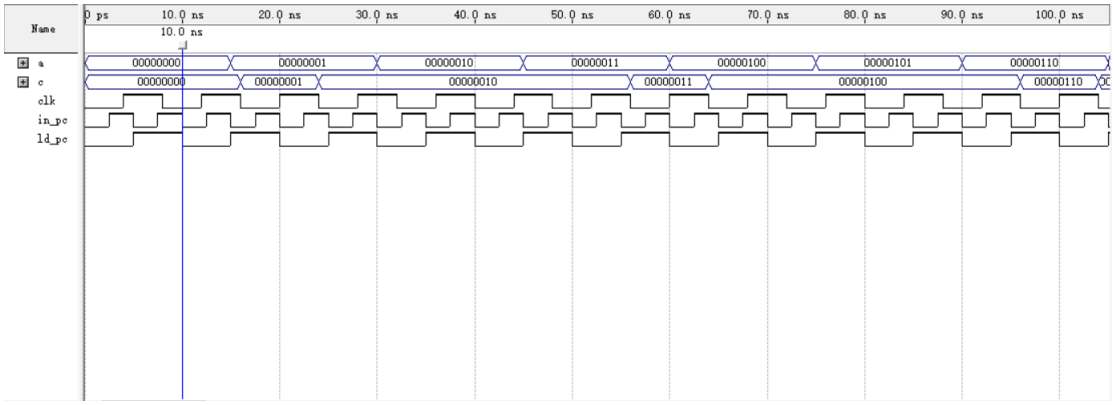
如果 ld_pc=1,in_pc=1,进行地址载入，pc<=address；

其他情况不进行操作

VHDL 设计：

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  entity PC is
5      port(clk,in_pc,ld_pc:in std_logic;
6           a:in std_logic_vector(7 downto 0);
7           c:out std_logic_vector(7 downto 0));
8  end entity;
9  architecture bhv of PC is
10 signal address:std_logic_vector(7 downto 0):="00000000";
11 begin
12 process(clk)
13 begin
14 if clk'event and clk='0' then
15     if in_pc='1' and ld_pc='0' then
16         address<=address+"00000001";
17     elsif in_pc='0' and ld_pc='1' then
18         address<=a;
19     else NULL;
20     end if;
21 end if;
22 end process;
23 c<=address;
24 end bhv;
```

（4）功能仿真验证：



仿真结果：

时钟为 8.0ns：时钟下降沿, in_pc=1,ld_pc=1,pc 保持不变,输出 c=00000000

时钟为 15.8ns: 时钟下降沿, in_pc=0,ld_pc=1,载入 pc,输入 a=00000001,输出 c=00000001

时钟为 24.0ns：时钟下降沿, in_pc=1,ld_pc=0,pc+1,时钟下降沿输出 c=00000010

时钟为 32.0ns：时钟下降沿, in_pc=0,ld_pc=0, pc 保持不变,输出 c=00000010

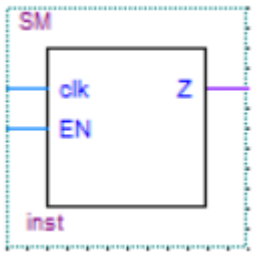
结论：仿真结果符合功能要求，设计成功

4. 时钟发生器 SM

（1）部件功能：

模型机中所有指令的执行都是一个周期完成取指令，一个周期执行指令。SM 则用来区分当前周期是取指令还是执行。

（2）接口设计：



输入：

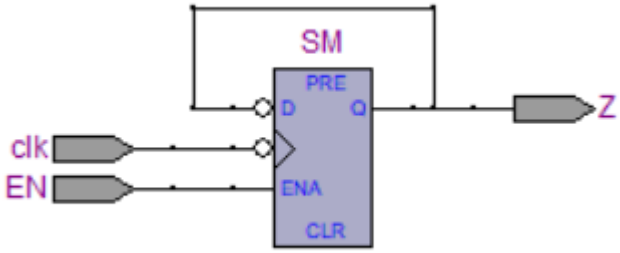
EN 使能信号 1 有效

clk：时钟信号

输出：

Z:变化的周期性时钟信号

RTL 视图



（3）功能实现：

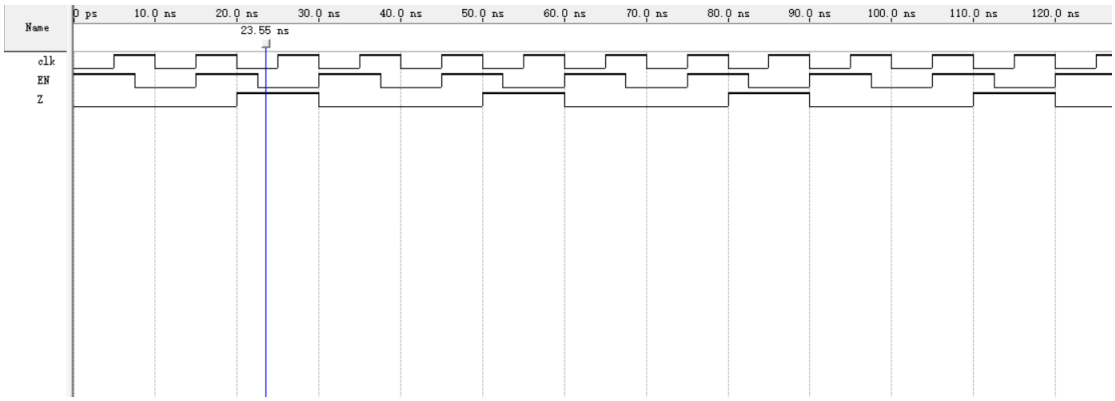
在时钟下降沿进行判断：

如果 EN=1，就对 SM 取反后输出，如果 EN=0，就输出 SM 的现有值。

VHDL 设计：

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   entity SM is
5       port(clk,EN:in std_logic;
6            Z:out std_logic);
7   end entity;
8   architecture bhv of SM is
9   signal SM:std_logic:='0';
10  begin
11  process(EN,clk)
12  begin
13      if(clk'event and clk='0') then
14          if(EN='1') then SM<=not SM;
15          else NULL;
16          end if;
17      end if;
18  end process;
19  Z<=SM;
20  end bhv;
```

（4）功能仿真验证：



仿真结果：

时钟为 10.0ns: 处于时钟下降沿，EN=0，SM 保持不变，SM=0

时钟为 20.0ns: 处于时钟下降沿，EN=1，对 SM 取反后输出，SM=1

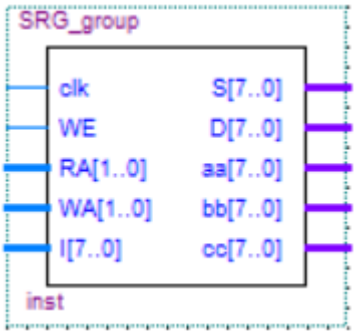
结论：仿真结果符合功能要求，设计成功

5. 通用寄存器组：

（1）部件功能：

寄存器具有快速读写的功能特点，而通用寄存器组则是实现的对三块寄存器的读写操作，根据控制信号对 A，B，C 寄存器实现读写，并将数据传送给 ALU 函数发生器或选择器。

（2）接口设计：



输入：

WE 使能信号，0 有效为写入

clk：时钟信号

RA[1…0]:控制信号，控制数据从 S 口读出

WA[1…0]:控制信号，控制数据从 D 口读出

I[7…0]:数据输入

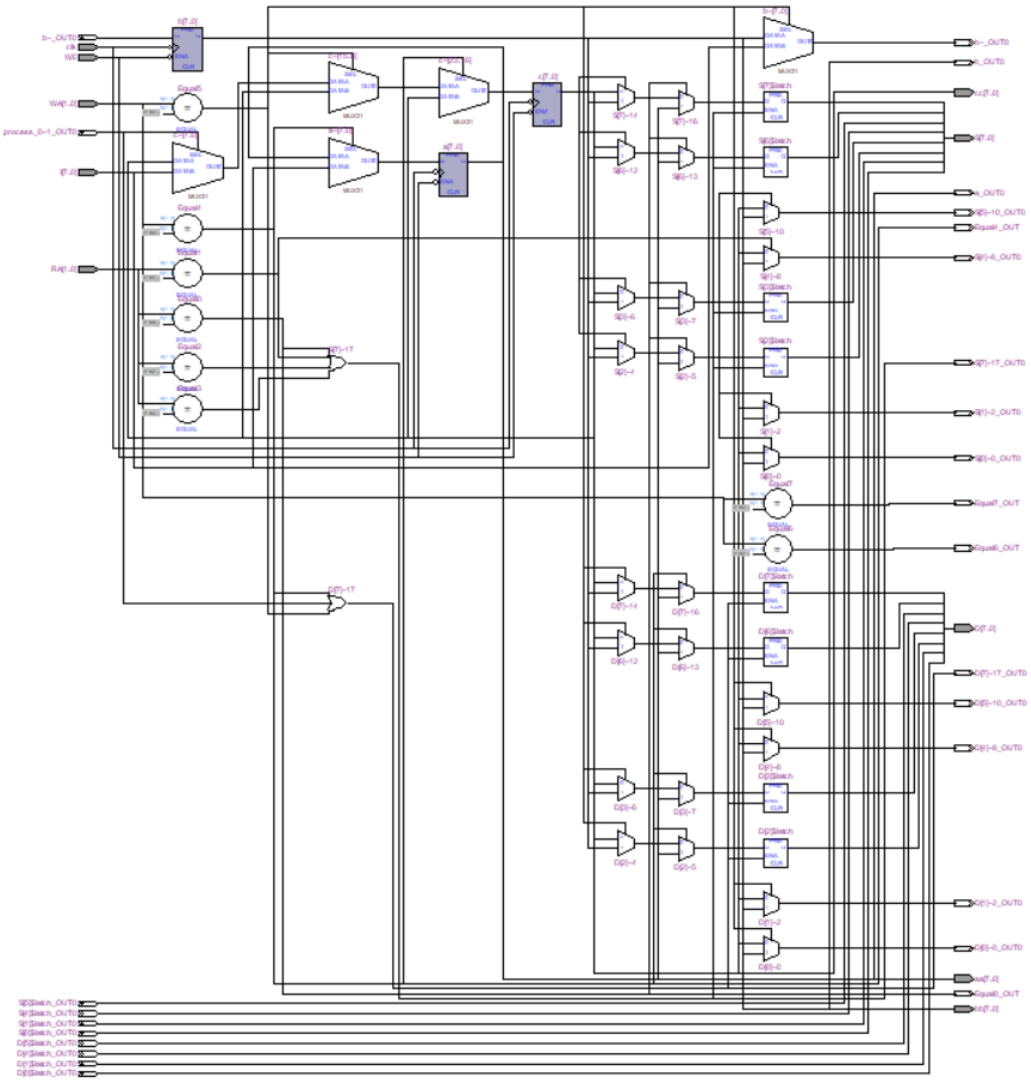
输出：

S[7…0]:从 S 口的数据输出

D[7…0]:从 D 口的数据输出

aa[7…0],bb[7…0],cc[7…0]向外界展示数据输出

RTL 视图



（3）功能实现：

无论 WE 为何值，都由 RA，WA 来判断输出

如果 RA=“00”A=a,如果 RA=“01”，A=b，否则 A=c

如果 WA=“00”B=a,如果 WA=“01”，B=b，否则 B=c

当 WE=0，执行写入

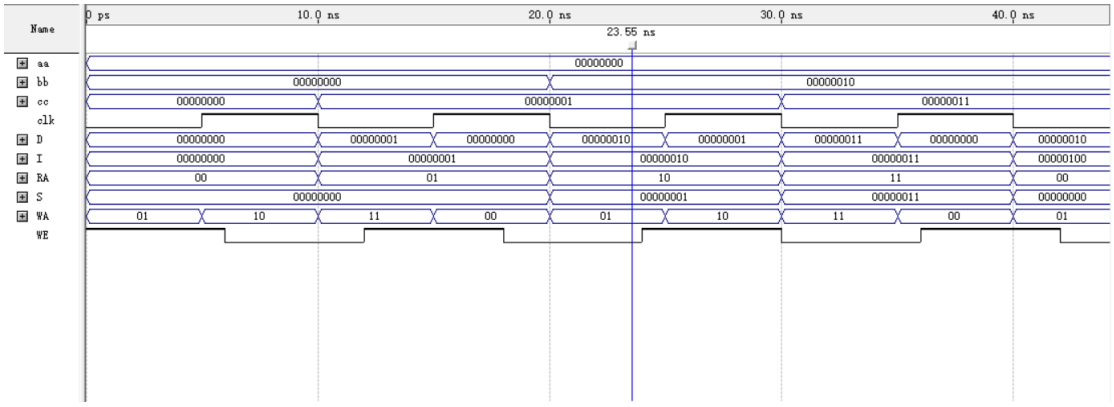
当时钟信号为下降沿：

如果 WA=“00”a<=I, 如果 WA=“01”，b<=I，否则 c<=I

VHDL 设计：

```
library ieee;
use ieee.std_logic_1164.all;
entity SRG_group is
    port(clk,WE:in std_logic;
         RA,WA:in std_logic_vector(1 downto 0);
         I:in std_logic_vector(7 downto 0);
         S,D:out std_logic_vector(7 downto 0);
         aa,bb,cc:out std_logic_vector(7 downto 0):="00000000");
end entity;
architecture bhv of SRG_group is
signal a,b,c:std_logic_vector(7 downto 0):="00000000";
begin
process(clk,RA,WA)
begin
if RA="00" then S<=a;
elsif RA="01" then S<=b;
elsif RA="10" or RA="11" then S<=c;
end if;
if WA="00" then D<=a;
elsif WA="01" then D<=b;
elsif WA="10" or WA="11" then D<=c;
end if;
if clk'event and clk='0' then
    if WE='0' then
        if WA="00" then a<=I;
        elsif WA="01" then b<=I;
        elsif WA="10" or WA="11" then c<=I;
        end if;
    end if;
end if;
end process;
aa<=a;
bb<=b;
cc<=c;
end bhv;
```

（4）功能仿真验证：



仿真结果：

时钟为 10.0ns: 处于时钟下降沿，WE=0，WA=11，将输入数据写入寄存器 C 中,cc=00000001

时钟为 15.0ns: 未处于时钟下降沿，WE=1，WA=00，RA=01,S 口读取 B 寄存器数据，D 口读取 A 寄存器数据，S=00000000，D=00000000

时钟为 40.0ns: 处于时钟下降沿，WE=1，WA=01，RA=00, S 口读取 A 寄存器数据，D 口读取 B 寄存器数据，S=00000000，D=00000010

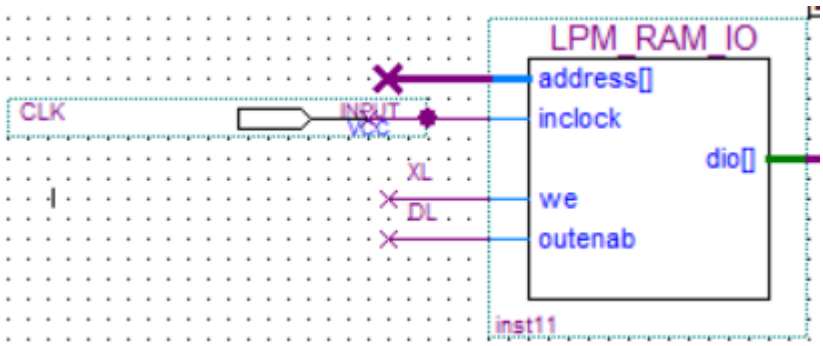
结论：仿真结果符合功能要求，设计成功

6. RAM

（1）部件功能：

RAM 可以根据 quartus 直接生成 LPM_RAM_IO，实现读外接文件 mif 的读写。

（2）接口设计：



输入：

inclock：时钟信号

address[7..0]:指定访问 RAM 的地址

outenab:读取控制信号，为 1 时将数据从 RAM 中读出

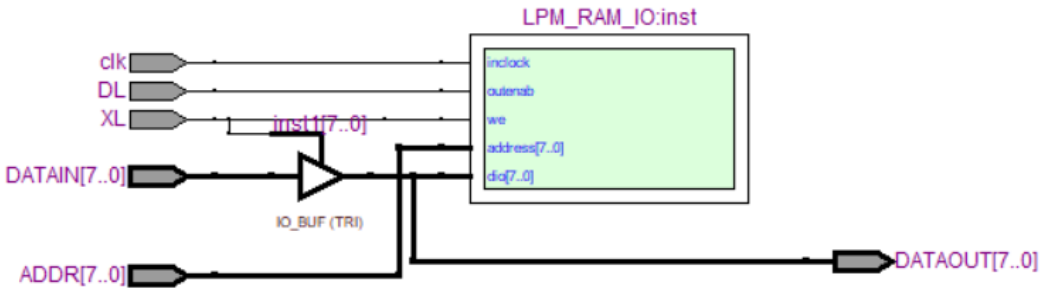
we:写入控制信号，为 1 时将数据写入 RAM

dio[]:将数据写入 RAM

输出：

dio[]:将数据从 RAM 中读出

RTL 视图



（3）功能实现：

CLK	We（XL）	outenab(DL)	功能
⎍	0	0	Dio<=高阻态Z
⎍	1	0	Dio的数据写入address所指定的存储单元
⎍	0	1	address所指定的存储单元数据从dio输出

四、系统测试

4.1 测试环境

开发软件：QuartusII 9.0sp1

操作系统：WINDOWS10 专业版

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。）

用 mif 初始化 RAM，按照指令系统表编写指令如下：

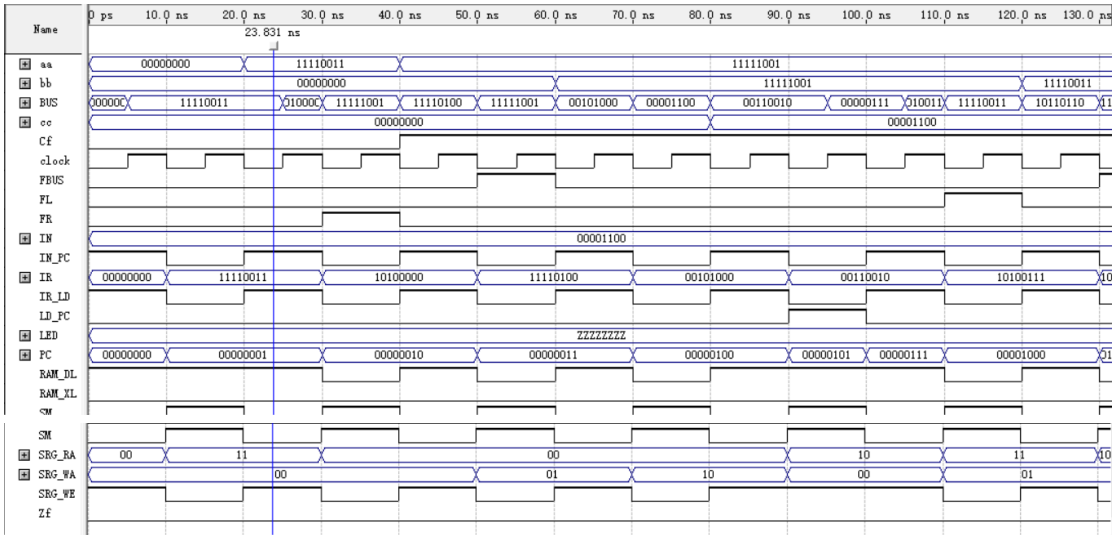
Addr	+000	+001	+010	+011	+100	+101	+110	+111
00000000	11110011	10100000	11110100	00101000	00110010	00000111	10000000	10100111
00001000	10110110	01010100	11111101	10010110	10000000	00110000	00010000	10000000
00010000	01000011	00110001	10000000	01100010	10000000	00000000	00000000	00000000

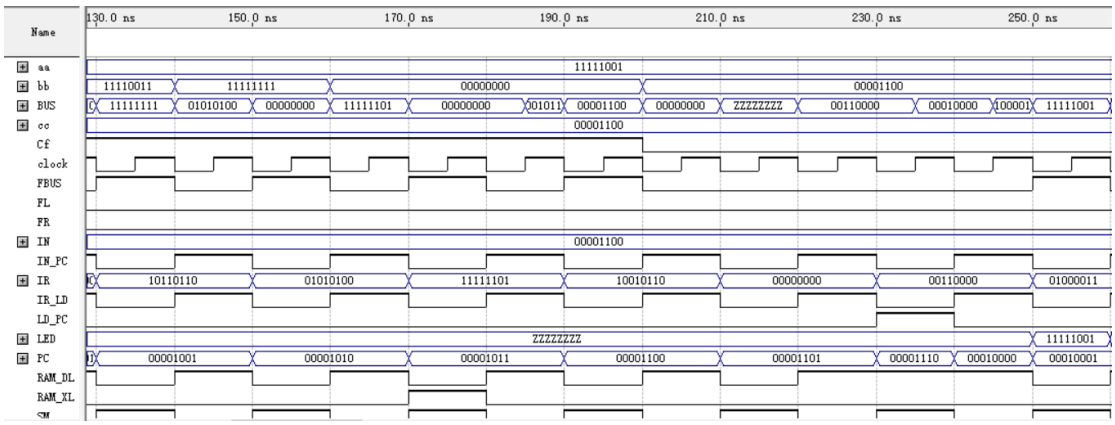
预估执行结果：

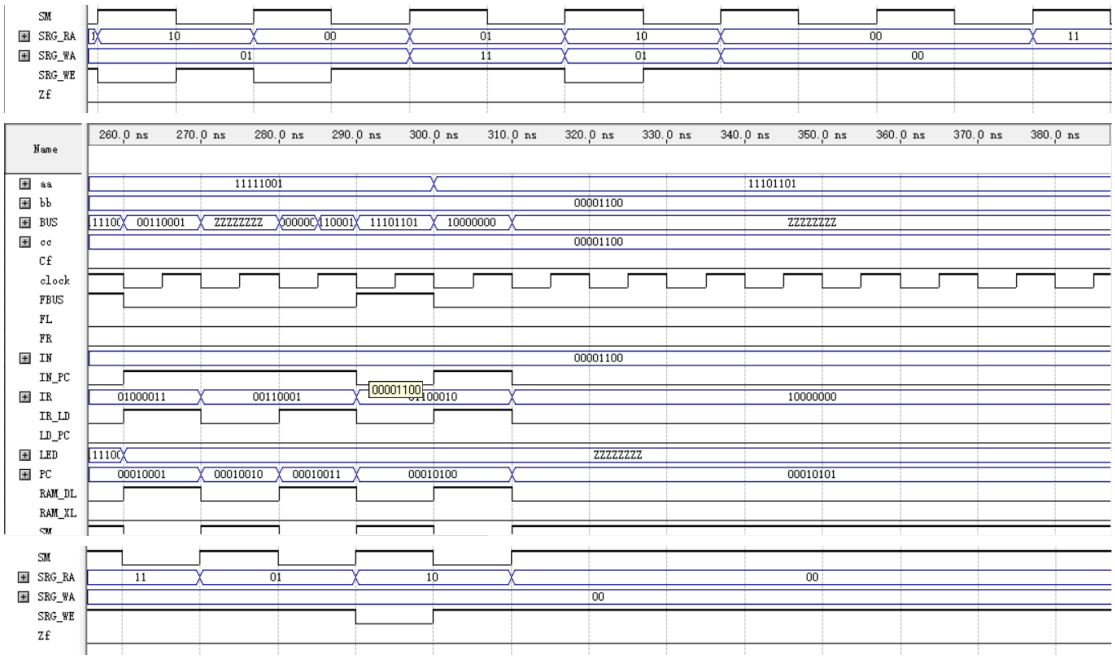
取指令		执行指令		
PC	RAM	aa	bb	cc
0	0	00000000	00000000	00000000
00000000	11110011	11110011	00000000	00000000
00000001	10100000	11111001	00000000	00000000
00000010	11110100	11111001	11111001	00000000
00000011	00101000	11111001	11111001	00001100
00000100	00110010	11111001	11111001	00001100
00000101	00000111	11111001	11111001	00001100
00000111	10100111	11111001	11110011	00001100
00001000	10110110	11111001	11111111	00001100
00001001	01010100	11111001	00000000	00001100
00001010	11111101	11111001	00000000	00001100
00001011	10010110	11111001	00001100	00001100
00001100	00000000	11111001	00001100	00001100
00001101	00110000	11111001	00001100	00001100
00001110	00010000	11111001	00001100	00001100
00010000	01000011	11111001	00001100	00001100
00010001	00110001	11111001	00001100	00001100
00010010	10000000	11111001	00001100	00001100
00010011	01100010	11101101	00001100	00001100
00010100	10000000	11101101	00001100	00001100

4.3 测试结果

功能仿真图：







波形分析：

第 1 条指令 MOV A M：将 RAM 中地址为 C 的数据写入寄存器 A

取址周期 SM=0：

时钟为 5.0ns 时，RAM_DL=1,RAM_XL=0，PC=00000000，时钟上升沿 RAM 读出指令

时钟为 10.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=11110011

执行周期 SM=1：

时钟为 15.0ns 时，SRG_WE=0，SRG_RA=11, SRG_WA=00，RAM 读出 00000000 对应数据 11110011

时钟为 20.0ns，时钟处于下降沿，A 寄存器载入总线数据，aa=11110011

指令执行完成.

第 2 条指令 RSR A：将寄存器 A 中的数据进行右移

取址周期 SM=0：

时钟为 25.0ns 时，RAM_DL=1,RAM_XL=0，PC=00000001，时钟上升沿 RAM 读出指令

时钟为 30.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=10100000

执行周期 SM=1：

时钟为 35.0ns 时，SRG_WE=0，SRG_RA=00, SRG_WA=00，读出寄存器 A 对应数据，通过移位逻辑进行右移后进入总线，同时产生 C 标志

时钟为 40.0ns，时钟处于下降沿，总线数据载入 A 寄存器，aa=11111001

指令执行完成.

第 3 条指令 MOV B A：将 A 寄存器中的数据载入到 B 寄存器中

取址周期 SM=0：

时钟为 45.0ns 时，RAM_DL=1,RAM_XL=0，PC=00000010，时钟上升沿 RAM 读出指令

时钟为 50.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=11110100

执行周期 SM=1：

时钟为 55.0ns 时，SRG_WE=0，SRG_RA=00, SRG_WA=01，读出寄存器 A 对应数据到总线

时钟为 60.0ns，时钟处于下降沿，总线数据载入 B 寄存器，bb=11111001

指令执行完成.

第 4 条指令 IN C：将外部输入数据写入寄存器 C

取址周期 SM=0：

时钟为 65.0ns 时，RAM_DL=1,RAM_XL=0，PC=00000011，时钟上升沿 RAM 读出指令

时钟为 70.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=00101000

执行周期 SM=1：

时钟为 75.0ns 时，SRG_WE=0，SRG_WA=10，外部输入送入总线

时钟为 80.0ns，时钟处于下降沿，C 寄存器载入总线数据，cc=00001100

指令执行完成.

第 5 条指令 JC：配合 C 标志，让 PC 跳转到指定地址

取址周期 SM=0：

时钟为 85.0ns 时，RAM_DL=1,RAM_XL=0，PC=00000100，时钟上升沿 RAM 读出指令

时钟为 90.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=00110010，同时 PC+1, PC=00000101

执行周期 SM=1：

时钟为 95.0ns 时，RAM_DL=1,RAM_XL=0，PC=00000101，时钟上升沿 RAM

读出跳转地址送入总线

时钟为 100.0ns，时钟处于下降沿，PC_IN=0,PC_LD=1，跳转地址载入 PC，PC=00000111

指令执行完成.

第 6 条指令 RSL B：将寄存器 B 中的数据进行左移

取址周期 SM=0：

时钟为 105.0ns 时，RAM_DL=1,RAM_XL=0，PC=00000111，时钟上升沿 RAM 读出指令

时钟为 110.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=10100111

执行周期 SM=1：

时钟为 115.0ns 时，SRG_WE=0，SRG_RA=01, SRG_WA=01，读出寄存器 B 对应数据，通过移位逻辑进行左移后进入总线，同时产生 C 标志

时钟为 120.0ns，时钟处于下降沿，B 寄存器载入总线数据，bb=11110011，

指令执行完成.

第 7 条指令 OR B C：进行或逻辑运算

取址周期 SM=0：

时钟为 125.0ns 时，RAM_DL=1,RAM_XL=0，PC=00001000，时钟上升沿 RAM 读出指令

时钟为 130.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=10110110

执行周期 SM=1：

时钟为 135.0ns 时，SRG_WE=0，SRG_RA=10, SRG_WA=01，B 寄存器中的数据和 C 寄存器中的数据进入 ALU，经或运算后输出至总线

时钟为 140.0ns，时钟处于下降沿，B 寄存器载入总线数据，bb=11111111

指令执行完成.

第 8 条指令 NOT B：进行取反逻辑运算

取址周期 SM=0：

时钟为 145.0ns 时，RAM_DL=1,RAM_XL=0，PC=00001001，时钟上升沿 RAM 读出指令

时钟为 150.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=01010100

执行周期 SM=1：

时钟为 155.0ns 时，SRG_WE=0，SRG_RA=01, SRG_WA=01，将 B 寄存器中的数据送入 ALU，经取反运算后送入总线

时钟为 160.0ns，时钟处于下降沿，B 寄存器载入总线数据，bb=00000000，

指令执行完成.

第 9 条指令 MOV M B:将寄存器 B 中的数据写入 RAM 中地址为 C 的存储单元

取址周期 SM=0：

时钟为 165.0ns 时，RAM_DL=1,RAM_XL=0，PC=00001010，时钟上升沿 RAM 读出指令

时钟为 170.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=11111101

执行周期 SM=1：

时钟为 175.0ns 时，SRG_WE=0，SRG_RA=01, SRG_WA=11，将 B 寄存器中的数据送入总线，在 RAM_DL=0,RAM_XL=1 作用下，将总线数据写入 RAM 中地址为 00001100 的存储单元

指令执行完成.

第 10 条指令 ADD B C：进行加法算术运算

取址周期 SM=0：

时钟为 185.0ns 时，RAM_DL=1,RAM_XL=0，PC=00001011，时钟上升沿 RAM 读出指令

时钟为 190.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=10010110

执行周期 SM=1：

时钟为 195.0ns 时，SRG_WE=0，SRG_RA=10, SRG_WA=01，B 寄存器中的数据和 C 寄存器中的数据进入 ALU，经加法运算后输出至总线，同时清除 C 标志

时钟为 200.0ns，时钟处于下降沿，总线数据载入 B 寄存器，bb=00001100

指令执行完成.

第 11 条指令 NOP：空操作

取址周期 SM=0：

时钟为 205.0ns 时，RAM_DL=1,RAM_XL=0，PC=00001100，时钟上升沿 RAM

读出指令

时钟为 210.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，

IR=00000000，PC+1，PC=00001101

执行周期 SM=1：

时钟为 215.0ns 时，无操作

指令执行完成.

第 12 条指令 JMP：让 PC 跳转到指定地址

取址周期 SM=0：

时钟为 225.0ns 时，RAM_DL=1,RAM_XL=0，PC=00001101，时钟上升沿 RAM 读出指令

时钟为 230.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，

IR=00110000，PC+1，PC=00001110

执行周期 SM=1：

时钟为 235.0ns 时，RAM_DL=1,RAM_XL=0，PC=00001110，时钟上升沿 RAM 读出跳转地址送入总线

时钟为 240.0ns 时，时钟处于下降沿，PC_IN=0,PC_LD=1，跳转地址载入 PC，PC=00010000

指令执行完成.

第 13 条指令 OUT A：将 A 寄存器中的数据送入外部输出

取址周期 SM=0：

时钟为 245.0ns 时，RAM_DL=1,RAM_XL=0，PC=00010000，时钟上升沿 RAM 读出指令

时钟为 250.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=01000011

执行周期 SM=1：

时钟为 255.0ns 时，SRG_WE=1，SRG_RA=11, SRG_WA=00，A 寄存器数据载入总线，LED=11111001

指令执行完成.

第 14 条指令 JZ：配合 Z 标志，让 PC 跳转到指定地址

取址周期 SM=0：

时钟为 265.0ns 时，RAM_DL=1,RAM_XL=0，PC=00010001，时钟上升沿 RAM 读出指令

时钟为 270.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=00110001，PC+1，PC=00010010

执行周期 SM=1：

时钟为 275.0ns 时，因为 Z 标志为 0，所以 PC_IN=1,PC_LD=0，PC+1，PC=00010011

指令执行完成.

第 15 条指令 SUB A C：进行减法算术运算

取址周期 SM=0：

时钟为 285.0ns 时，RAM_DL=1,RAM_XL=0，PC=00010011，时钟上升沿 RAM 读出指令

时钟为 290.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=01100010

执行周期 SM=1：

时钟为 295.0ns 时，SRG_WE=0，SRG_RA=10, SRG_WA=00，A 寄存器中的数据和 C 寄存器中的数据进入 ALU，经减法运算后输出至总线，同时清除 C 标志

时钟为 300.0ns 时，时钟处于下降沿，总线数据载入 A 寄存器，aa=11101101

指令执行完成.

第 16 条指令 HALT：停机指令

取址周期 SM=0：

时钟为 305.0ns 时，RAM_DL=1,RAM_XL=0，PC=00010100，时钟上升沿 RAM 读出指令

时钟为 310.0ns 时，时钟处于下降沿，指令寄存器 IR 载入数据，IR=10000000

执行周期 SM=1：

RAM_DL=0,RAM_XL=0,PC_IN=0,PC_LD=0,IR_LD=0,SRG_WE=1，SRG_RA=00, SRG_WA=00,MUX=00,IN_EN=0,OUT_EN=0,SM_EN=0 系统所有使能信号失效，此时所有部件不工作.

指令执行完成.

结论：仿真结果满足功能要求，设计正确。

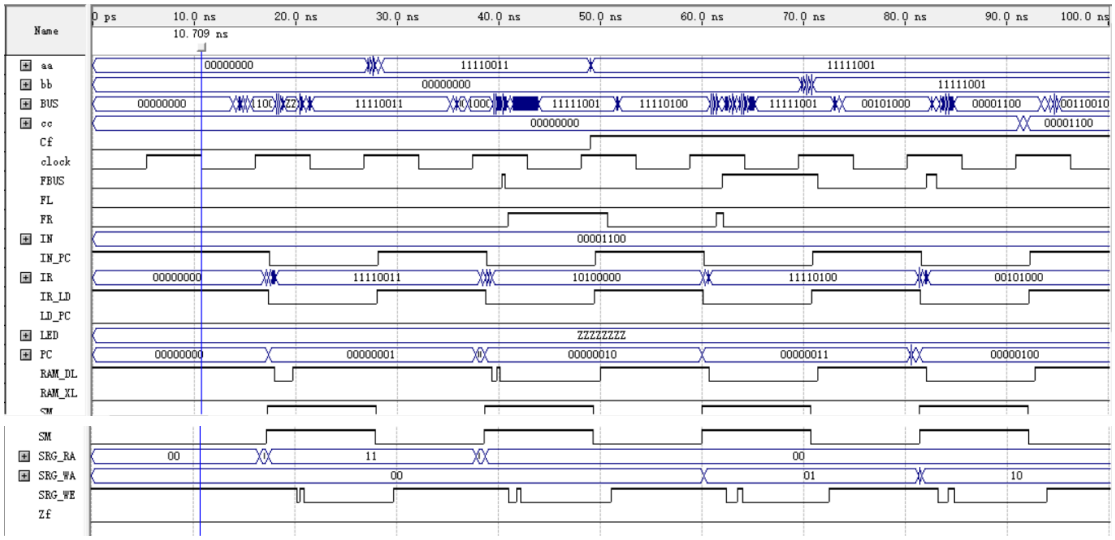
4.4 模型机性能分析

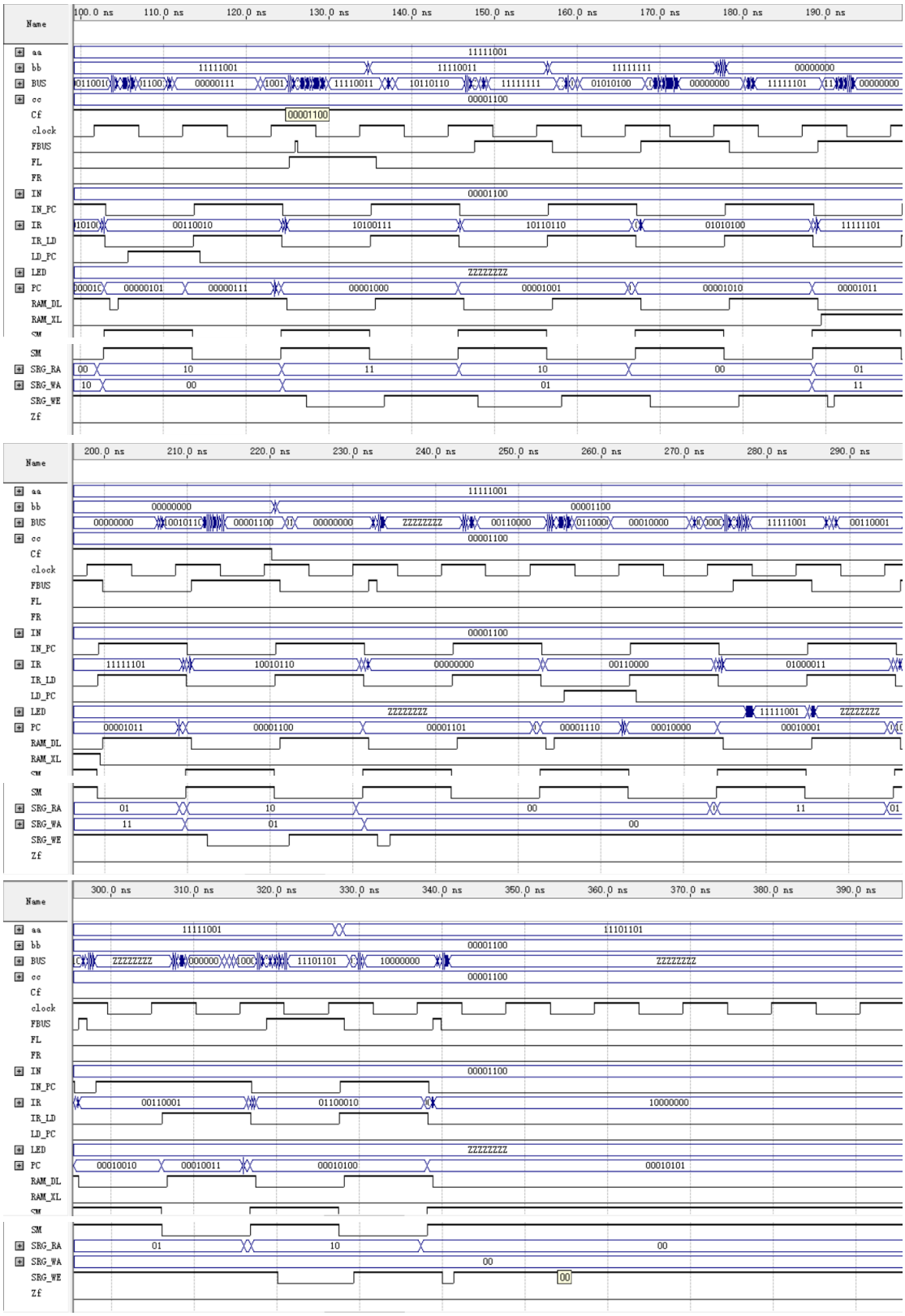
资源消耗：

Flow Status	Successful - Fri Jan 01 19:30:53 2021
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
Device	EP1C3T100C8
Timing Models	Final
Met timing requirements	No
Total logic elements	224 / 2,910 (8 %)
Total pins	17 / 65 (26 %)
Total virtual pins	0
Total memory bits	2,048 / 59,904 (3 %)
Total PLLs	0 / 1 (0 %)

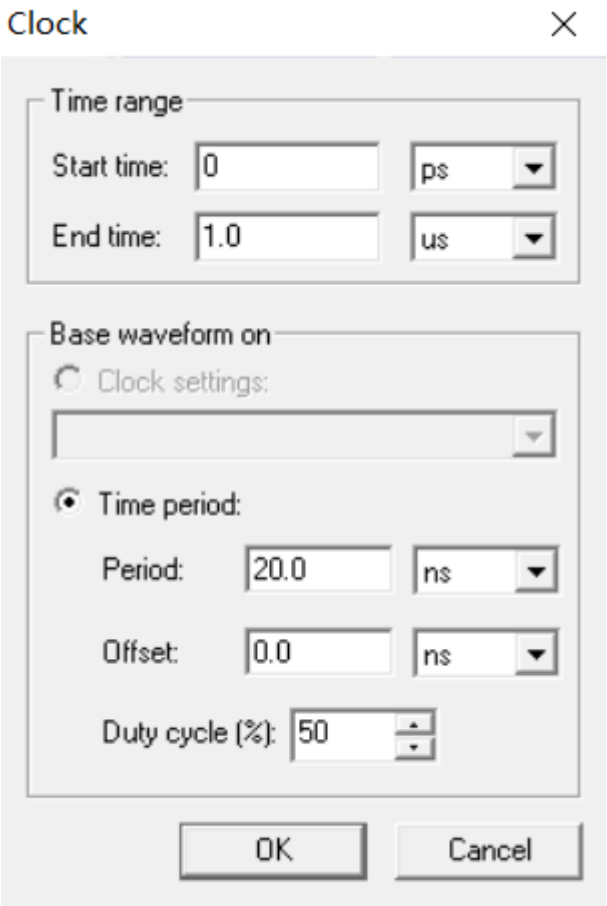
Total logical elements 总逻辑门数：224，占用率 8%

时序性能：





时序仿真成功执行时序指令，最后停机，此时最小的时钟信号为 20ns，所以说可以用 20ns 的时钟周期实现 16 条指令而不出现错误。



总的来说，224（8%）的逻辑门总数和 20ns 的最小时序时钟周期称得上是一个优秀的设计。

五、总结

（设计总结与心得）

1. 时间和空间是冲突的，在进行时序仿真时，我不断的改进电路，减少开销。当逻辑门开销是234时，时钟的最小周期是19ns，改进开销后，逻辑门数量减少到224此时最小时钟却提高到了20ns.我最后选择了后者，减少开销损失时间，因为我认为我的设计在这种情况下开销的价值大于时间。
2. 减少开销的方法：

   （1）一定情况下在VHDL中将if语句改为when select，这样的单变量选择可以减少开销，但是串行会增加延时。

   （2）优化结构：总控制器的设计最能反映你的逻辑，我们要合理的简化设计，精简逻辑，达到最优的效果。

3. 避免触发器的不必要生成

   无论是什么语句，都讲究逻辑的划分，在组合电路设计时，尽量将一个逻辑分为一个集合的划分，包含所有情况，相互之间没有交集，避免多余的触发器生成。

4. 中间变量的使用，我们在设计时序电路时，往往会有这样的情况，用一个内部变量来临时存储一个值，当需要改变的时候就去改变它，实际上，这个功能类似于寄存器，为什么不直接用寄存器的逻辑去设计呢？这样反而精简。

5. 交流的重要性：一个人毕竟不能持续的工作，相互的交流会让过程边的更加简单有趣，很多时候不经意间的交谈就激起了我的设计想法。

6. 前提到交流的重要性但是完全依赖就是禁区，我们更要独立的思考独立的设计，要做到这块CPU完全是自己自主设计的，这才是实验目的。还是那句话，talk is cheap，show me the code.

7. 感谢的话：很感谢这个实验，让我真正的体会到了什么叫做设计，什么叫把自己的idea去实现；胡老师真的是一个很有爱的老师，总是让我们顶着压力却能灿烂自信的微笑；助教虽然验收平时很严格但是这是对我们负责，让我们组合每个部件的时候更加精准，再一次感谢你们！