

模型机设计报告

班级 软件 1801 姓名 肖云杰 学号 201826010113

一、设计目的

本课程力图以“培养学生现代数字系统设计能力”为目标，贯彻以 CPU 设计为核心，以层次化、模块化设计方法为抓手的组织思路，培养学生设计与实现数字系统的能力。

本设计要求在进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的成果（包括已经设计好的功能部件和调试方法），设计出一个简易计算机系统。

二、设计内容

（一）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；

（二）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；

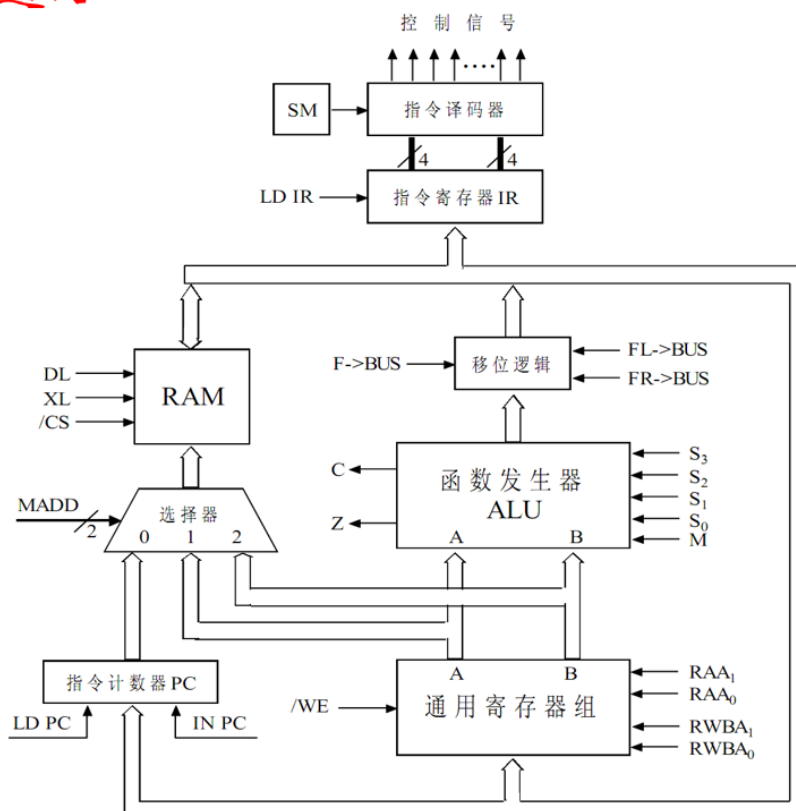
（三）对所设计计算机的性能指标进行分析，整理出设计报告。

3.1 设计的整体架构

表1 指令系统表

汇编符号	功能	编码
MOV R1, R2	$(R2) \rightarrow R1$	1111 R1 R2
MOV M, R2	$(R2) \rightarrow (C)$	1111 11 R2
MOV R1, M	$((C)) \rightarrow R1$	1111 R1 11
ADD R1, R2	$(R1) + (R2) \rightarrow R1$	1001 R1 R2
SUB R1, R2	$(R1) - (R2) \rightarrow R1$	0110 R1 R2
OR R1, R2	$(R1) \vee (R2) \rightarrow R1$	1011 R1 R2
NOT R1	$\neg (R1) \rightarrow R1$	0101 R1 XX
RSR R1	$(R1)$ 循环右移一位 $\rightarrow R1$	1010 R1 00
RSL R1	$(R1)$ 循环左移一位 $\rightarrow R1$	1010 R1 11
JMP add	$add \rightarrow PC$	0001 00 00, address
JZ add	结果为 0 时 $add \rightarrow PC$	0001 00 01, address
JC add	结果有进位时 $add \rightarrow PC$	0001 00 10, address
IN R1	$(\text{开关 } 7-0) \rightarrow R1$	0010 R1 XX
OUT R1	$(R1) \rightarrow \text{发光二极管 } 7-0$	0100 R1 XX
NOP	$(PC) + 1 \rightarrow PC$	0111 00 00
HALT	停机	1000 00 00

(四) 数据通路



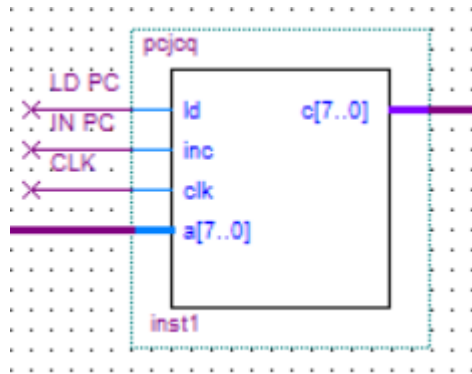
(五) 控制信号

表 2 基本控制信号及功能表

序号	信号	功能
1	IN PC	与 LD PC 配合使用，为 1 时 PC 加 1 计数，为 0 时加载 BUS 上的数据。
2	LD PC	当 IN PC=1 允许对 PC 加 1 计数，否则允许把 BUS 上的数据打入 PC。
3	LD IR	允许把 BUS 上的数据打入指令寄存器 IR。
4	/WE	允许把 BUS 上的数据打入通用寄存器组，低电平有效。
5	F→BUS	ALU 的运算结果通过移位门直接送到总线 BUS 的对应位。
6	FL→BUS	ALU 的运算结果通过移位门左移一位送到总线 BUS，且 F7 送 C _f 。
7	FR→BUS	ALU 的运算结果通过移位门右移一位送到总线 BUS，且 F0 送 C _f 。
8	/CS	允许访问存储器，低电平有效。
9~10	MADD	存储器 RAM 地址来源。0：指令计数器，1：通用寄存器 A 口，2：B 口。
11	DL	读存储器 RAM。
12	XL	写存储器 RAM。
13	M	M=1，表示 ALU 进行逻辑运算操作。
14~17	S ₃ ~S ₀	使 ALU 执行各种运算的控制位。
18	HALT	此位为“1”时停机，下次输入人工操作。

3.2 各模块的具体实现

1. 指令计数器



其功能表如下

CLK	IN PC	LD PC	功能
	1	0	$PC \leftarrow PC+1$
	0	1	$PC \leftarrow \text{输入}$

功能描述:

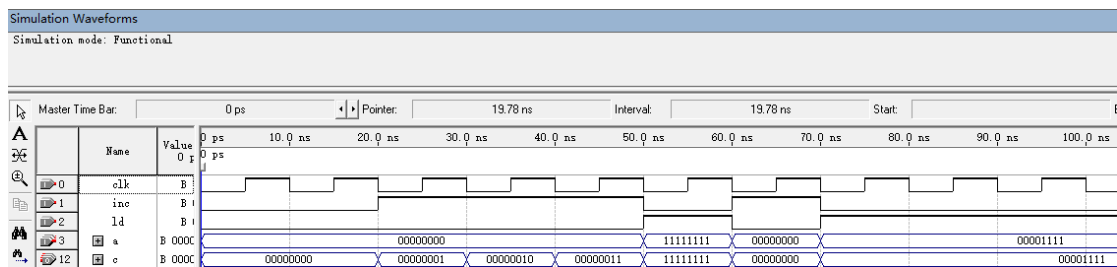
1. 在时钟下降沿, 若 $ld=1$, $inc=0$, 则 pc 输出值 = 输入值
2. 在时钟下降沿, 若 $inc=0$, $ld=0$, 则 pc 输出=原 $pc+1$;

```

CPU.bdf | pcjoq.vhd
1  library ieee;
2  use ieee.std_logic_1164.all, ieee.std_logic_unsigned.all;
3
4  entity pcjoq is
5  port(a : in std_logic_vector(7 downto 0);
6        ld, inc, clk : in std_logic;
7        c : out std_logic_vector(7 downto 0));
8  end pcjoq;
9
10 architecture behave of pcjoq is
11  signal tem : std_logic_vector(7 downto 0);
12  begin
13  process(ld, inc, clk)
14  begin
15      if (clk'event and clk = '0') then
16          if inc = '1' then
17              if tem = "11111111" then
18                  tem <= "00000000";
19              else
20                  tem <= tem + 1;
21              end if;
22          elsif ld = '1' then
23              tem <= a;
24          end if;
25          end if;
26      end process;
27      c <= tem;
28  end;

```

功能仿真:



c) 结果分析及结论

0-20ns: 两个功能选择都为低电平，无功能，因此，输出不变，模块没有问题

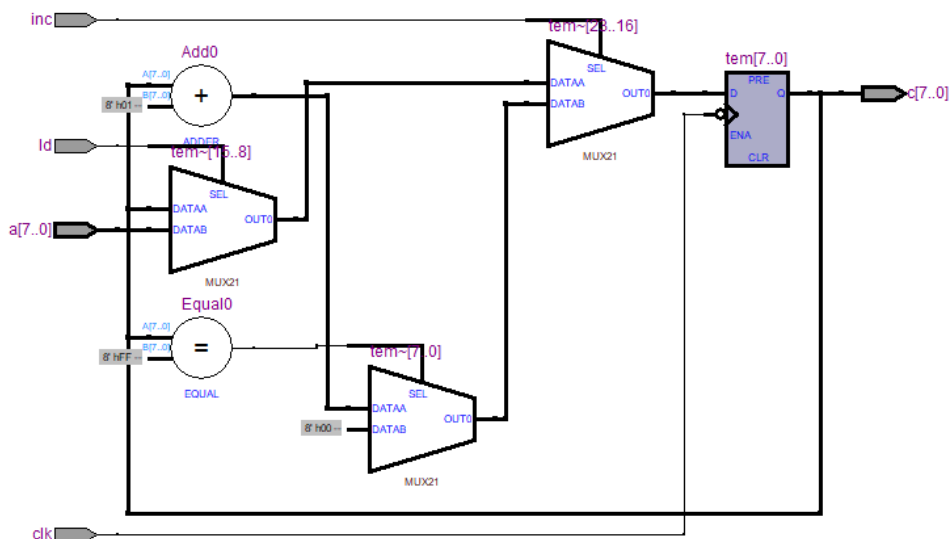
20-50ns: inc为高电平，功能为自增一，因此，在每个时钟下降沿，数据进行自增1

50-60ns: 设置为极限数据

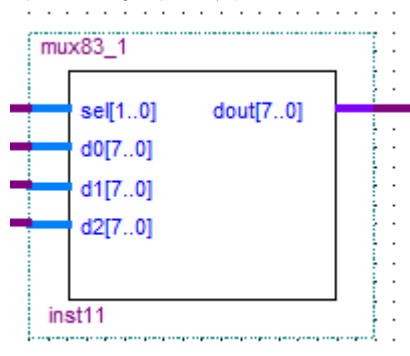
60-70ns: 判断溢出归零操作

70-100ns: ld为1，为置值功能，测试正确。

STL 视图



2. 8路3选1多路选择器



其功能如下:

根据 sel , dout 选择 合适的 d0, d1, d2 输出。

当 sel = "00" dout 选择 d0

当 sel = "01" dout 选择 d1

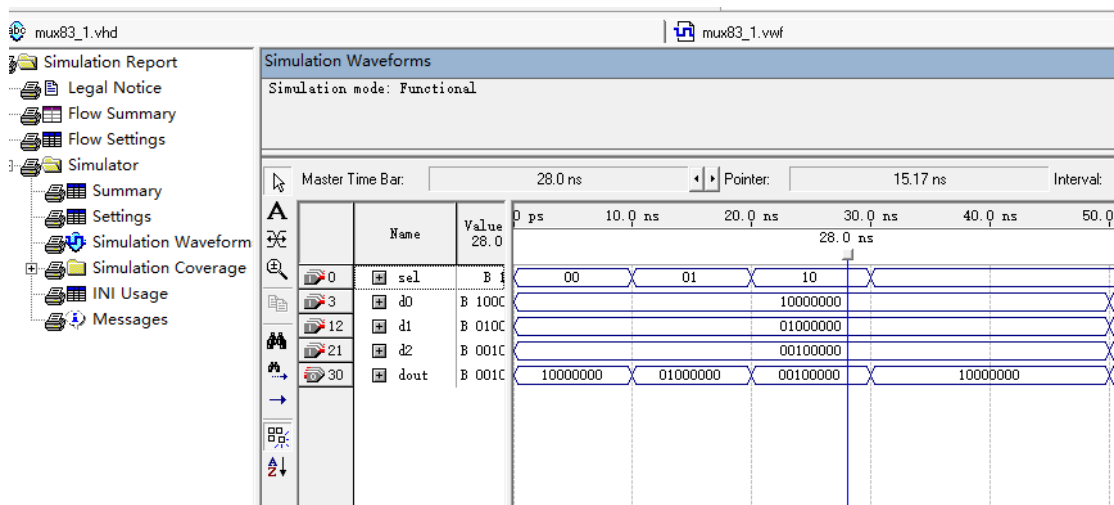
其他情况 dout 选择 d2

```

CPU.bdf | mux83_1.vhd
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  entity mux83_1 is
4  port(  sel :in STD_LOGIC_VECTOR(1 downto 0);
5         d0,d1,d2:in STD_LOGIC_VECTOR(7 downto 0);
6         dout:out STD_LOGIC_VECTOR(7 downto 0));
7  end mux83_1;
8  architecture rtl of mux83_1 is
9  begin
10     dout <=
11     d0 when sel="00" else
12     d1 when sel="01" else
13     d2;
14 end rtl;
15

```

功能仿真：



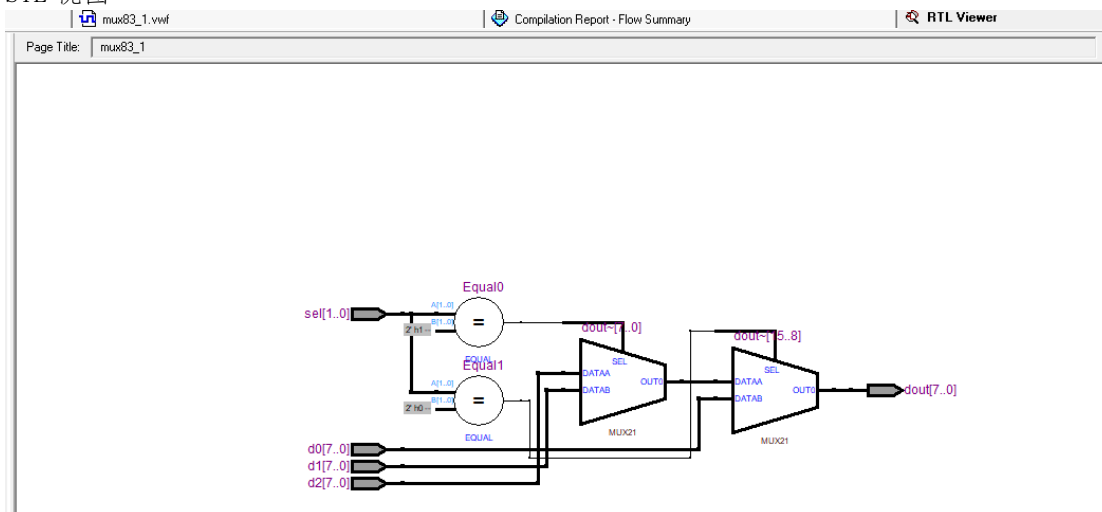
结果分析及结论

0-10ns: 选择信号为00，选择d0输出，正确

10-20ns: 选择信号为01，选择d1输出，正确

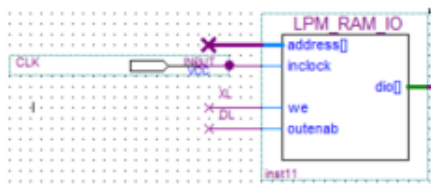
20-30ns: 选择信号为 10，选择 d2 输出，正确

STL 视图



3. RAM

LPM_RAM_IO的符号及功能如下：



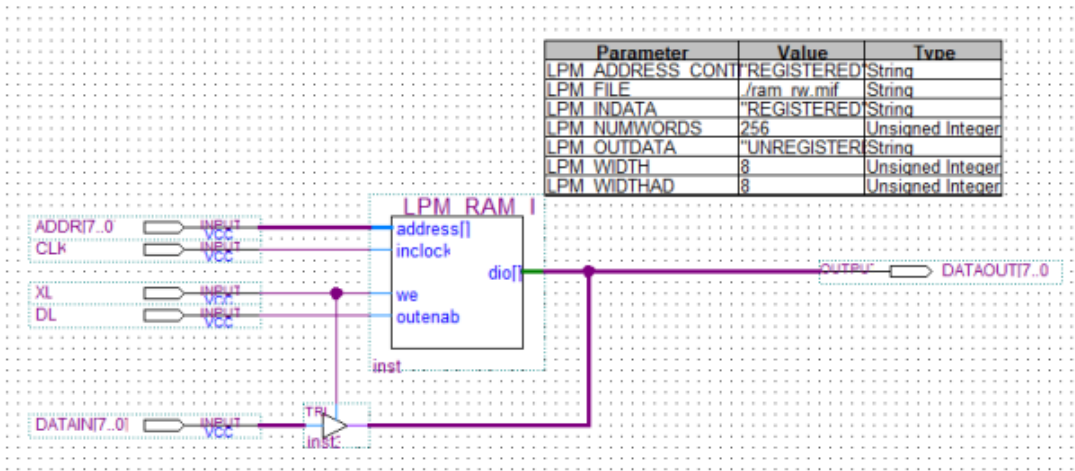
CLK	we	outenab	功能
	0	0	Dio<=高阻态Z
	1	0	Dio的数据写入address所指定的存储单元
	0	1	address所指定的存储单元数据从dio输出

功能描述：

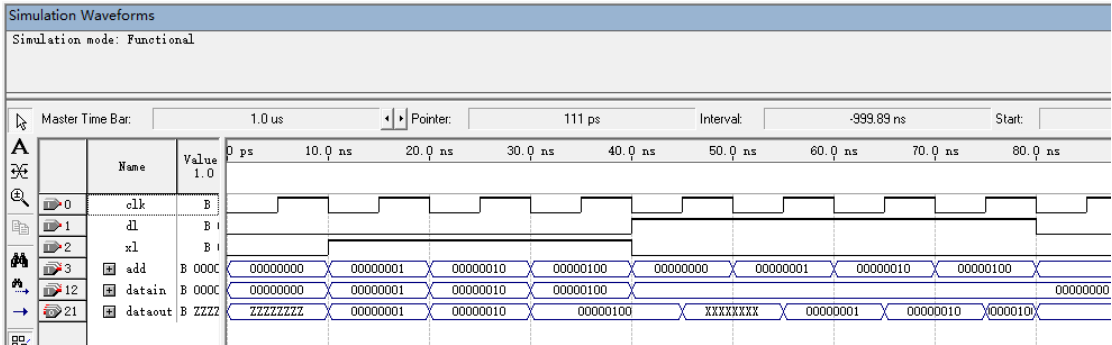
- 在 clk 时钟上升沿有效：
- 当 we=0 outenab =0 ， 输出为高阻态；
- 当 we=1 outenab =0 ， 将 dio 的数据写入地址 address
- 当 we=0 outenab =1 ， 将 address 的数据输出

采用 lpm 定制

将RAM和寄存器配合实现对RAM的读写操作，参考实现如图所示：



功能仿真如下：

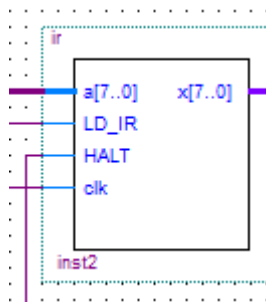


0-10ns: 两个功能信号都为低电平，输出为高阻态
10-40ns: we (x1) 为1，写入，将00000001写入00000001地址，将00000010写入00000010地

址，将00000100写入00000100地址。

40-80ns；读出，从地址内读出数据，且时钟上升沿有效

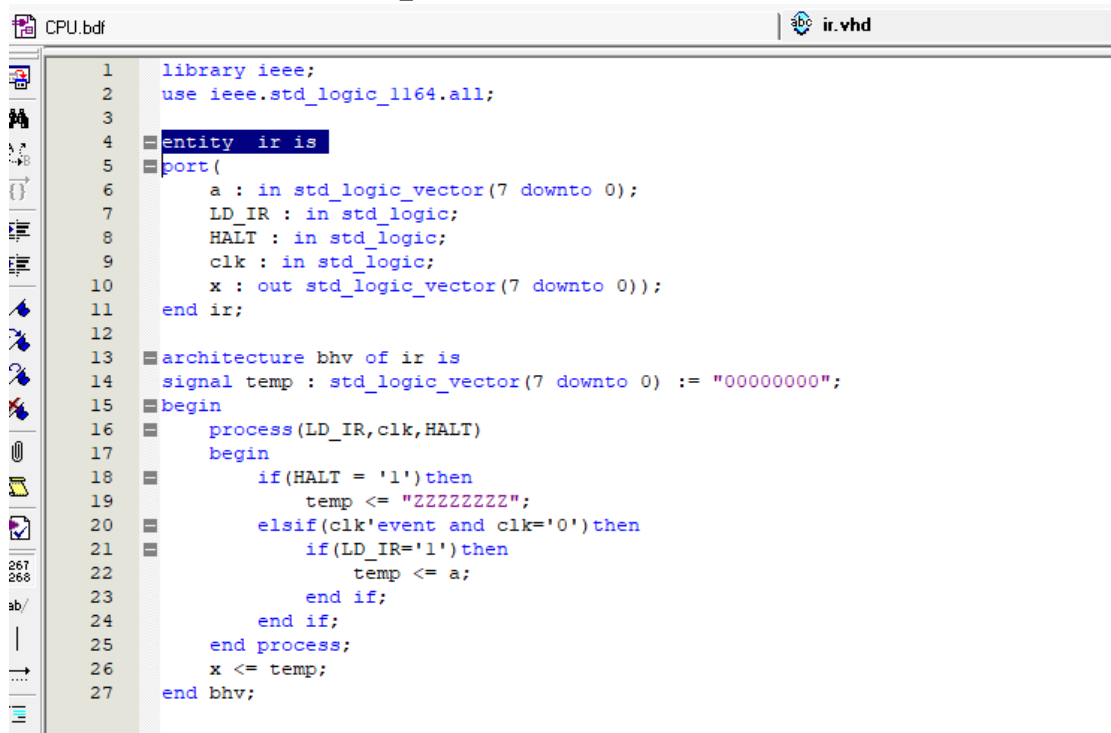
4. 指令寄存器



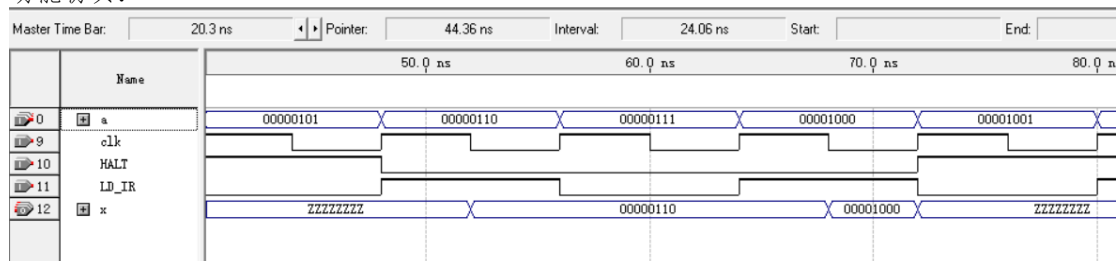
功能描述：

由其功能表得知，当 halt = 1，输出为高阻态

当 时钟处于下降沿， 且 ld_ir=1，输出为 输入 a；

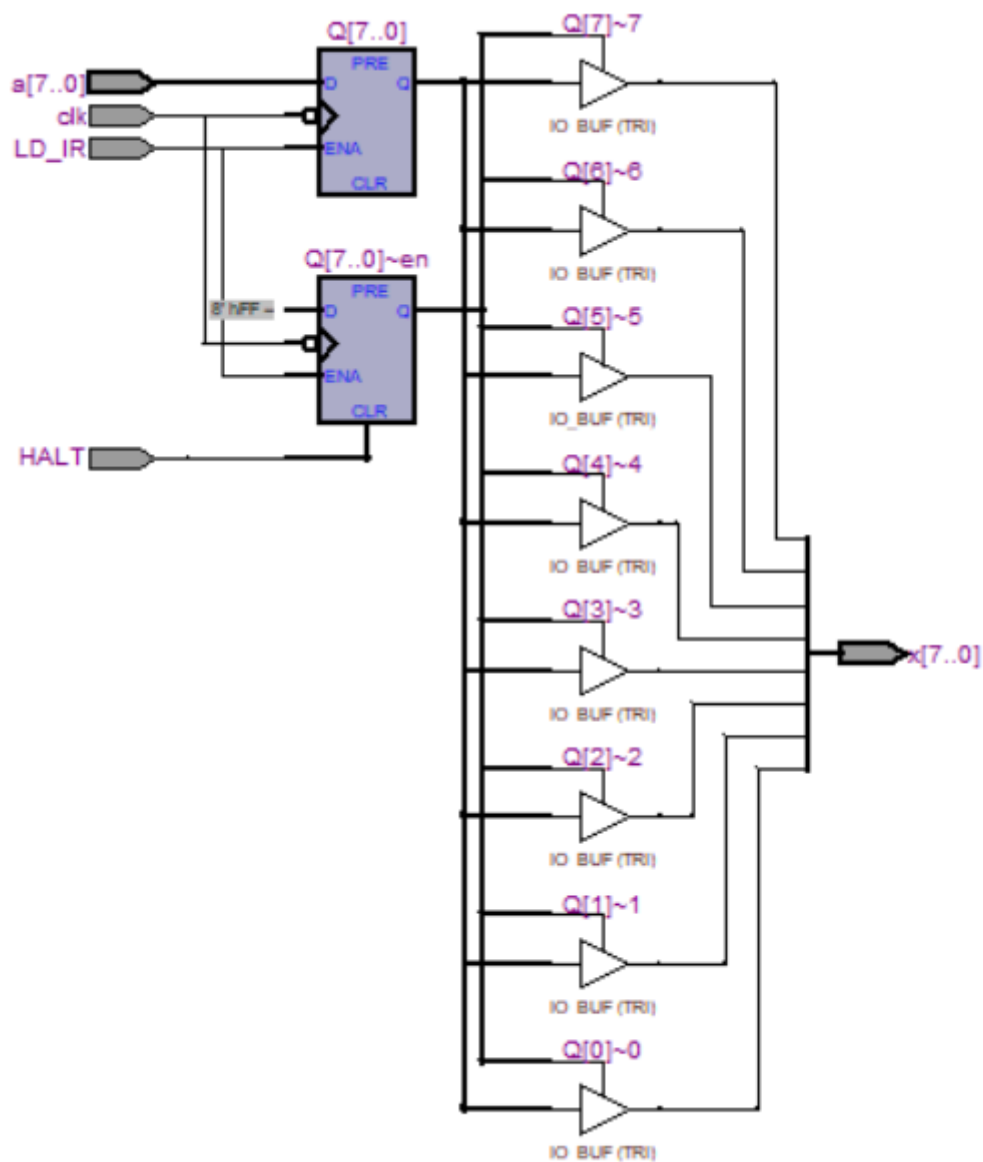


功能仿真：

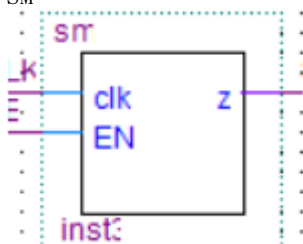


仿真结果正确

St1 视图





5. SM



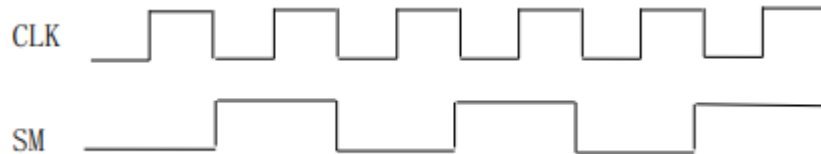
当 SM=0 时，该周期完成取指令，当 SM=1 时，该周期执行指令

Sm 功能如下:

CLK	EN	功能
	1	SM \leftrightarrow SM取反
	0	SM不变

Clk 时钟下降沿有效，当 en = 1，sm 取反
当 en = 0，sm 不变；

在实际使用中，我们根据 SM 的功能去掉 en，简化电路



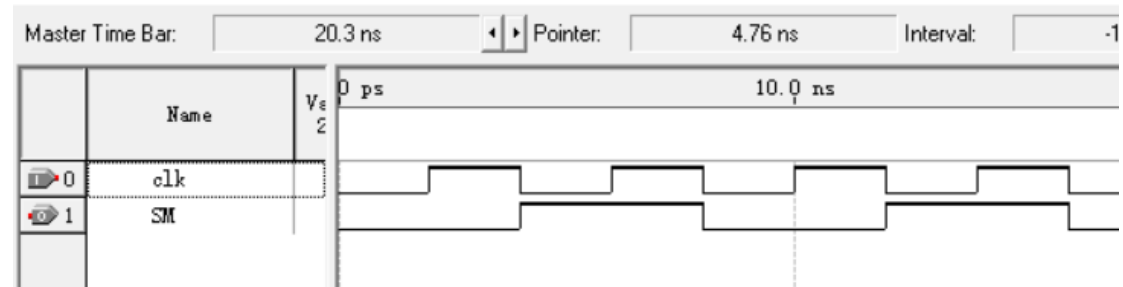
CPU.bdf

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity SM is
7  port (
8      clk : in std_logic;
9      SM : out std_logic);
10 end SM;
11
12 architecture beh of SM is
13     signal S : std_logic := '0';
14 begin
15     process (clk)
16     begin
17         if (clk'event and clk='0') then
18             if (S = '0') then
19                 S <= '1';
20             else
21                 S <= '0';
22             end if;
23         end if;
24     end process;
25     SM <= S;
26 end beh;

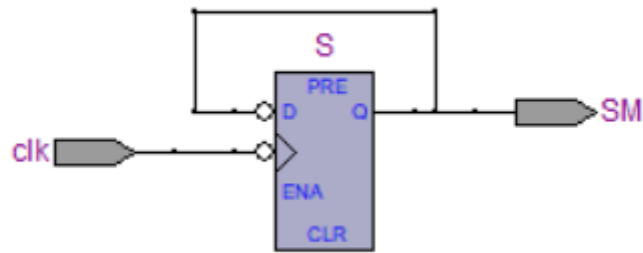
```

功能仿真：



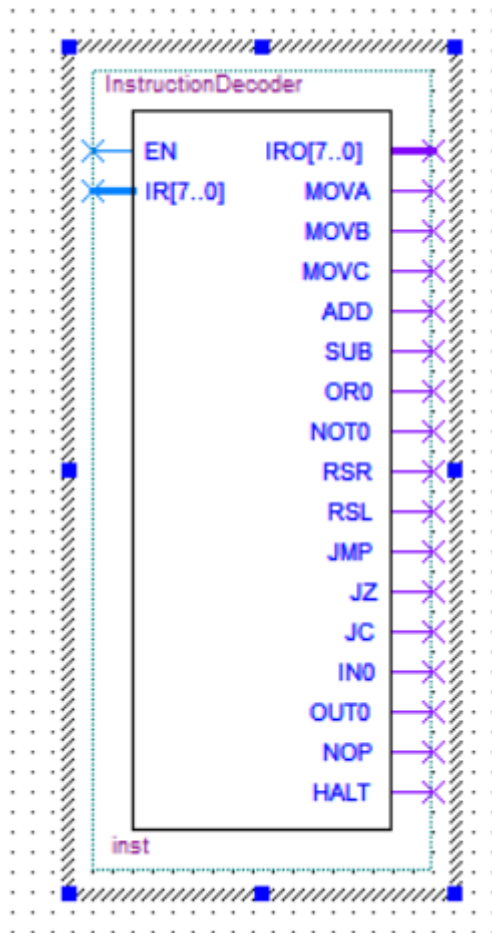
对比上图，发现功能仿真正确。

STL 视图



6. 指令译码器

功能： 当使能端为高电平，通过输入的指令，选择合适的功能接口输出



```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity InstructionDecoder is
7 port(
8   EN : in std_logic;
9   IR : in std_logic_vector(7 downto 0);
10  IRO : out std_logic_vector(7 downto 0);
11  MOVA : out std_logic;
12  MOVB : out std_logic;
13  MOVC : out std_logic;
14  ADD : out std_logic;
15  SUB : out std_logic;
16  ORO : out std_logic;
17  NOT0 : out std_logic;
18  RSR : out std_logic;
19  RSL : out std_logic;
20  JMP : out std_logic;
21  JZ : out std_logic;
22  JC : out std_logic;
23  IN0 : out std_logic;
24  OUT0 : out std_logic;
25  NOP : out std_logic;
26  HALT : out std_logic;
27 end InstructionDecoder ;
28
29 architecture RTL of InstructionDecoder is
30 begin
31   IRO <= IR;
32   MOVA <= EN and IR(7) and IR(6) and IR(5) and IR(4) and not( IR(3) and IR(2)) and not( IR(1) and IR(0));
33   MOVB <= EN and IR(7) and IR(6) and IR(5) and IR(4) and IR(3) and IR(2) and not( IR(1) and IR(0));
34   MOVC <= EN and IR(7) and IR(6) and IR(5) and IR(4) and not( IR(3) and IR(2)) and IR(1) and IR(0);
35   ADD <= EN and IR(7) and (not IR(6)) and (not IR(5)) and IR(4) and not( IR(3) and IR(2)) and not( IR(1) and IR(0));
36   SUB <= EN and (not IR(7)) and IR(6) and IR(5) and (not IR(4)) and not( IR(3) and IR(2)) and not( IR(1) and IR(0));
37   ORO <= EN and IR(7) and (not IR(6)) and IR(5) and IR(4) and not( IR(3) and IR(2)) and not( IR(1) and IR(0));
38   NOT0 <= EN and (not IR(7)) and IR(6) and (not IR(5)) and IR(4) and not( IR(3) and IR(2));
39   RSR <= EN and IR(7) and (not IR(6)) and IR(5) and (not IR(4)) and not( IR(3) and IR(2)) and (not IR(1)) and (not IR(0));
40   RSL <= EN and IR(7) and (not IR(6)) and IR(5) and (not IR(4)) and not( IR(3) and IR(2)) and IR(1) and IR(0);
41   JMP <= EN and (not IR(7)) and (not IR(6)) and (not IR(5)) and IR(4) and (not IR(3)) and (not IR(2)) and (not IR(1)) and (not IR(0));
42   JZ <= EN and (not IR(7)) and (not IR(6)) and (not IR(5)) and IR(4) and (not IR(3)) and (not IR(2)) and (not IR(1)) and IR(0);
43   JC <= EN and (not IR(7)) and (not IR(6)) and (not IR(5)) and IR(4) and (not IR(3)) and (not IR(2)) and IR(1) and (not IR(0));
44   IN0 <= EN and (not IR(7)) and (not IR(6)) and IR(5) and (not IR(4)) and not( IR(3) and IR(2));
45   OUT0 <= EN and (not IR(7)) and IR(6) and (not IR(5)) and (not IR(4)) and not( IR(3) and IR(2));
46   NOP <= EN and (not IR(7)) and IR(6) and IR(5) and IR(4) and (not IR(3)) and (not IR(2)) and (not IR(1)) and (not IR(0));
47   HALT <= EN and IR(7) and (not IR(6)) and (not IR(5)) and (not IR(4)) and (not IR(3)) and (not IR(2)) and (not IR(1)) and (not IR(0));
48 end RTL ;

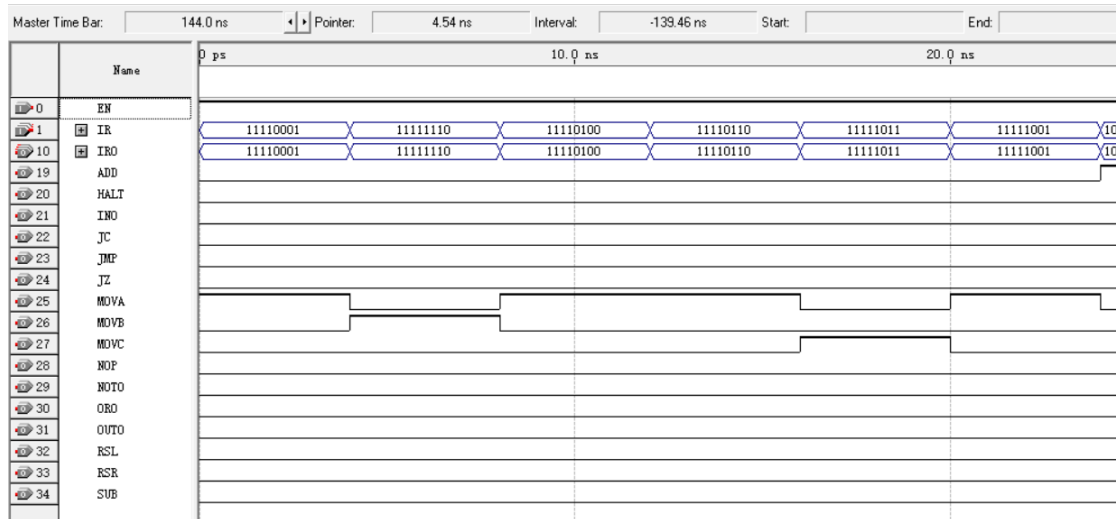
```

对应功能表

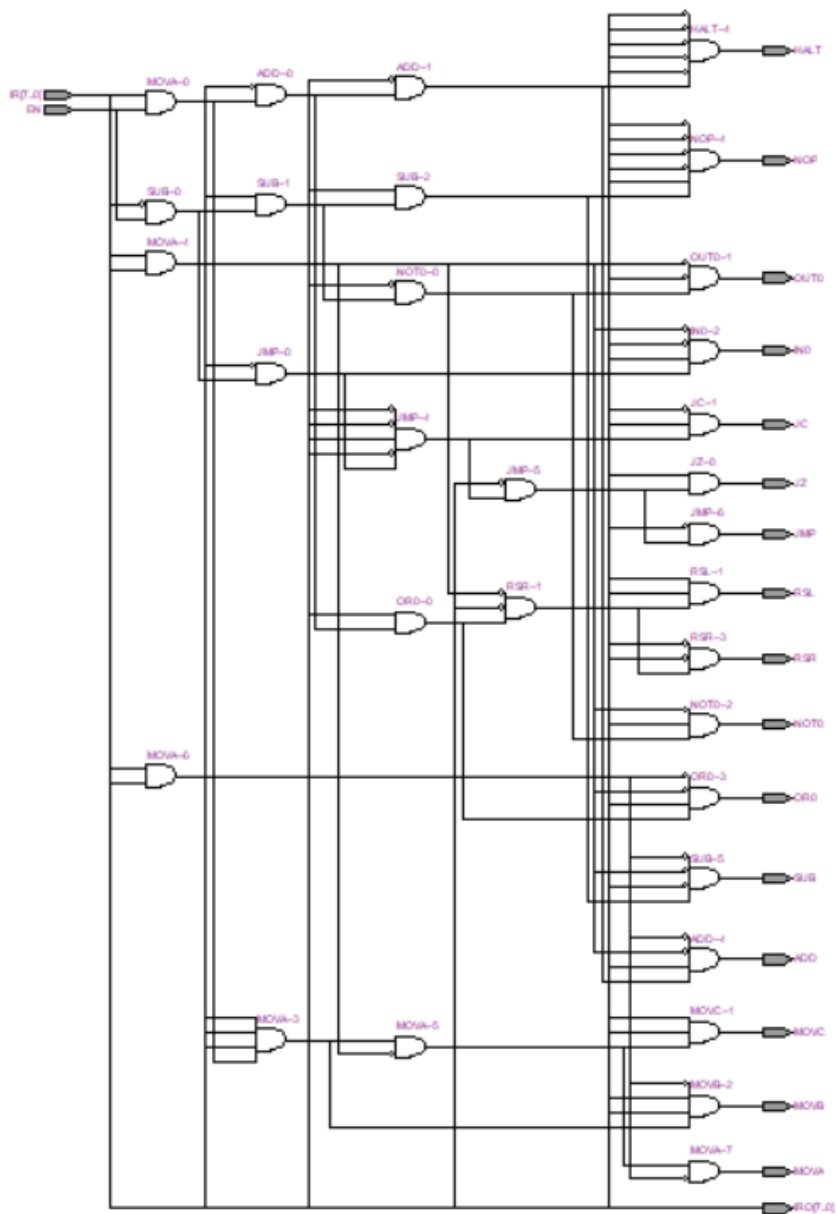
表1 指令系统表

汇编符号	功能	编码
MOV R1, R2	$(R2) \rightarrow R1$	1111 R1 R2
MOV M, R2	$(R2) \rightarrow (C)$	1111 11 R2
MOV R1, M	$((C)) \rightarrow R1$	1111 R1 11
ADD R1, R2	$(R1) + (R2) \rightarrow R1$	1001 R1 R2
SUB R1, R2	$(R1) - (R2) \rightarrow R1$	0110 R1 R2
OR R1, R2	$(R1) \vee (R2) \rightarrow R1$	1011 R1 R2
NOT R1	$\neg (R1) \rightarrow R1$	0101 R1 XX
RSR R1	$(R1)$ 循环右移一位 $\rightarrow R1$	1010 R1 00
RSL R1	$(R1)$ 循环左移一位 $\rightarrow R1$	1010 R1 11
JMP add	add $\rightarrow PC$	0001 00 00, address
JZ add	结果为 0 时 add $\rightarrow PC$	0001 00 01, address
JC add	结果有进位时 add $\rightarrow PC$	0001 00 10, address
IN R1	(开关 7-0) $\rightarrow R1$	0010 R1 XX
OUT R1	$(R1) \rightarrow$ 发光二极管 7-0	0100 R1 XX
NOP	$(PC) + 1 \rightarrow PC$	0111 00 00
HALT	停机	1000 00 00

仿真结果：



STL 视图:



7. 控制信号逻辑

需要根据每一个指令控制信号发出对应的简易计算机系统的各个控制信号。

控制信号发生器需要完成：

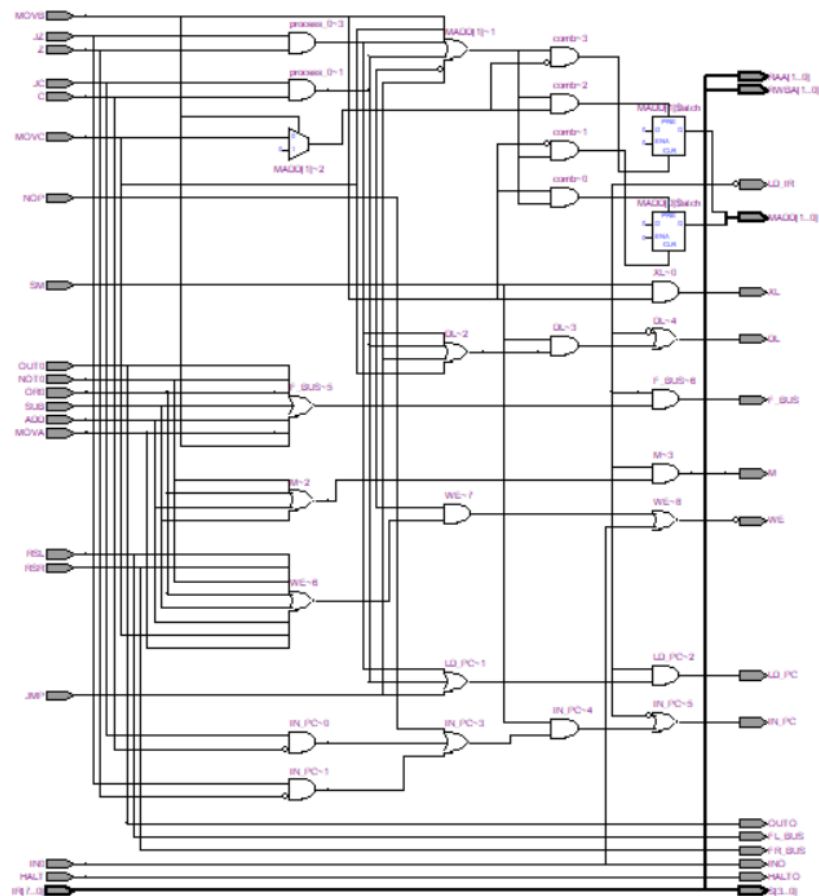
指令控制信号和简易计算机系统的各个控制信号的一一对应。

```

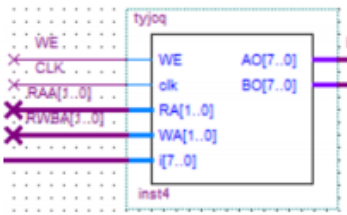
CPU.bdf | control_signal.vhd
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity control_signal is
7 port(
8   IR : in std_logic_vector(7 downto 0);
9   SM, MOVA, MOVVB, MOVC, ADD, SUB, OR0, NOT0, RSR, RSL, JMP, JZ, JC, IN0, OUT0, NOP, HALT, Z, C : in std_logic;
10  S : out std_logic_vector(3 downto 0);
11  RAA, RWBA, MADD: out std_logic_vector(1 downto 0);
12  IN_PC, LD_PC, WE, M, F_BUS, FR_BUS, LD_IR, DL, XL, HALTO, INO, OUTO : out std_logic;
13 end control_signal;
14
15 architecture beh of control_signal is
16 begin
17   process (SM, JMP, JC, C, JZ, Z, MOVVB, MOVC)
18   begin
19     if (MOVVB = '1') then
20       MADD <= "01";
21     elsif (MOVC = '1') then
22       MADD <= "10";
23     elsif ((not SM or JMP or (JC and C) or (JZ and Z)) = '1') then
24       MADD <= "00";
25     end if;
26   end process;
27   S <= IR(7 downto 4);
28   RAA <= IR(1 downto 0);
29   RWBA <= IR(3 downto 2);
30   IN_PC <= '1' when (SM='0') or ((JC='1' and C='0') or (JZ='1' and Z='0') or NOP='1') and SM='1' else '0';
31   LD_PC <= '1' when SM='1' and (JMP='1' or (JC='1' and C='1') or (JZ='1' and Z='1')) else '0';
32   WE <= '0' when ((SM='1' and (MOVA='1' or MOVC='1' or ADD='1' or SUB='1' or OR0='1' or NOT0='1' or RSR='1' or RSL='1')) or INO='1') else '1';
33   M <= '1' when (SM='1' and (ADD='1' or SUB='1' or OR0='1' or NOT0='1')) else '0';
34   F_BUS <= '1' when (SM='1' and (MOVA='1' or MOVB='1' or ADD='1' or SUB='1' or OR0='1' or NOT0='1' or OUTO='1')) else '0';
35   FR_BUS <= '1' when RSL='1' else '0';
36   LD_IR <= '0' when SM='1' else '1';
37   DL <= '1' when (SM='0' or (SM='1' and (MOVC='1' or JMP='1' or (JC='1' and C='1') or (JZ='1' and Z='1')))) else '0';
38   XL <= '1' when (SM='1' and MOVVB='1') else '0';
39   HALTO <= '1' when HALT='1' else '0';
40   INO <= '1' when INO='1' else '0';
41   OUTO <= '1' when OUTO='1' else '0';
42 end beh;

```

RTL 如图



8. 通用寄存器



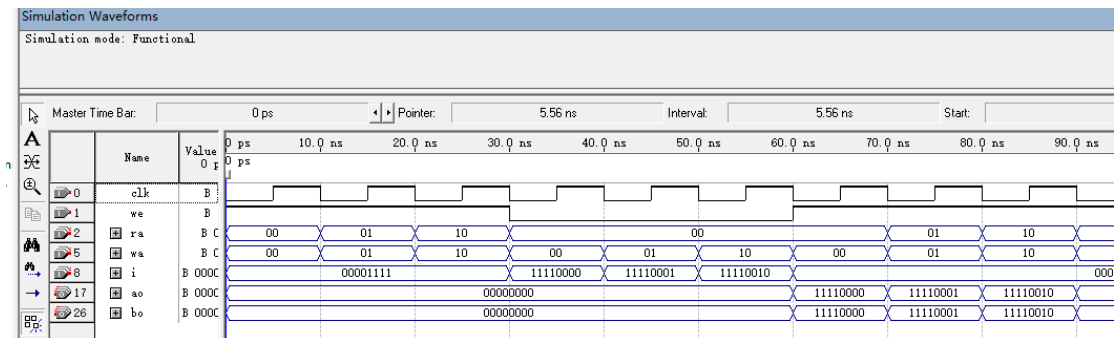
CLK	WE	RAA[1..0]	RWBA[1..0]	功能
	1	00或01或10	00或01或10	<p>根据RAA[1..0]的值从A,B,C中选择一个寄存器的值由AO口输出</p> <p>如RAA[1..0]=00, AO<=A寄存器的值</p> <p>RAA[1..0]=01, AO<=B寄存器的值</p> <p>RAA[1..0]=10, AO<=C寄存器的值</p> <p>根据RWBA[1..0]的值从选择A,B,C中选择一个寄存器的值由BO口输出,</p> <p>如RWBA[1..0]=00, BO<=A寄存器的值</p> <p>RWBA[1..0]=01, BO<=B寄存器的值</p> <p>RWBA[1..0]=10, BO<=C寄存器的值</p>
	0	XX	00或01或10	根据RWBA[1..0]的值,将外部输入写入A,B,C三个寄存器中的一个寄存器内。

```

CPU.bdf | Simulation Report - Simulation Waveforms | CPU.
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity tyjq is
5  port(
6      RAA : in std_logic_vector(1 downto 0);
7      RWBA : in std_logic_vector(1 downto 0);
8      I : in std_logic_vector(7 downto 0);
9      WE : in std_logic;
10     clk : in std_logic;
11     AO : out std_logic_vector(7 downto 0);
12     BO : out std_logic_vector(7 downto 0);
13 end tyjq;
14
15 architecture bhv of tyjq is
16     signal A : std_logic_vector(7 downto 0) := "10000000";
17     signal B : std_logic_vector(7 downto 0) := "11000001";
18     signal C : std_logic_vector(7 downto 0) := "10001001";
19     begin
20     process(clk,WE,A,B,C)
21     begin
22         if(clk'event and clk = '0')then
23             if(WE = '0') then
24                 if(RWBA = "00")then
25                     A <= I;
26                 elsif(RWBA = "01")then
27                     B <= I;
28                 elsif(RWBA = "10")then
29                     C <= I;
30                 end if;
31             end if;
32         end if;
33     end process;
34     AO <= A when RAA = "00" else
35         B when RAA = "01" else
36         C when RAA = "10" or RAA = "11";
37     BO <= A when RWBA = "00" else
38         B when RWBA = "01" else
39         C when RWBA = "10" or RWBA = "11";
40 end bhv;

```

仿真结果如下：



结果分析及结论

0-10ns: we为1, 选择输出, 此时选择的为A寄存器的值, 都为00000000

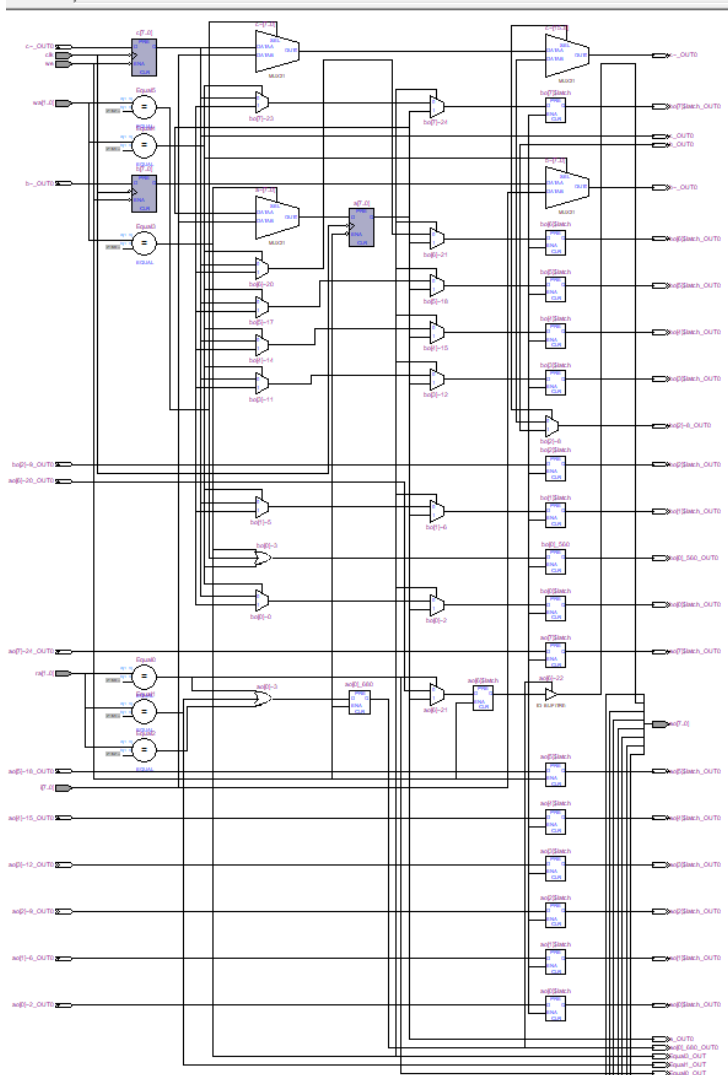
10-20ns: we为1, 选择输出, 此时选择的为B寄存器的值, 都为00000000

20-30ns: we为1, 选择输出, 此时选择的为C寄存器的值, 都为00000000

40-60ns: we为0, 按照wa, 将11110000, 11110001, 11110010分别赋值给三个寄存器。

60-90ns: we为1, 再次从三个寄存器取值, 观察, 正确。

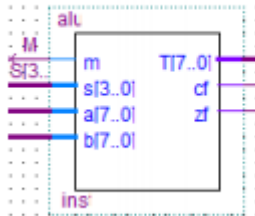
STL 视图:



9. ALU

ALU 根据 S3-S1 和 M 控制信号需要实现 ADD、SUB、OR、NOT 运算，并产生状态位 Cf 和 Zf。另外，ALU 在 MOVA、MOVB、RSR 和 RSL 四条指令执行时，提供将数据传输至 BUS 总线的通路。

ALU 模块图：



指令具体功能如下：

ADD R1, R2	$(R1) + (R2) \rightarrow R1$	1001 R1 R2
SUB R1, R2	$(R1) - (R2) \rightarrow R1$	0110 R1 R2
OR R1, R2	$(R1) \vee (R2) \rightarrow R1$	1011 R1 R2
NOT R1	$\neg (R1) \rightarrow R1$	0101 R1 XX

```

CPU.bdf | Simulation Report - Simulation Waveforms | CPU.vwf | tyjo
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity ALU is
7  port(
8      M : in std_logic;
9      S : in std_logic_vector(3 downto 0);
10     A,B : in std_logic_vector(7 downto 0);
11     C,Z : out std_logic;
12     COUNT : out std_logic_vector(7 downto 0);
13 end ALU;
14
15 architecture str of ALU is
16     component parallel_adder_vhdl is
17     port(
18         A,B : in std_logic_vector(7 downto 0);
19         C0 : in std_logic;
20         S : out std_logic_vector(7 downto 0);
21         C8 : out std_logic);
22     end component;
23     signal sum,add,sub,subf : std_logic_vector(7 downto 0);
24     signal Ca,Cs,Cf : std_logic;
25     begin
26         process(M,S)
27         begin
28             if(M = '1') then
29                 if (S = "1001") then
30                     sum <= add;
31                     C <= Ca;
32                 elsif (S = "0110") then
33                     sum <= subf;
34                     C <= Cs;
35                 elsif (S = "1011") then
36                     sum <= A or B;
37                     C <= '0';
38                 elsif (S = "0101") then
39                     sum <= not A;
40                     C <= '0';
41                 end if;
42             else
43                 sum <= A;
44                 C <= '0';
45             end if;
46         end process;
47         process(sum)
48         begin
49             if (sum = "00000000") then
50                 Z <= '1';
51             else
52                 Z <= '0';
53             end if;
54         end process;
55         Al : parallel_adder_vhdl
56         port map (A, B, '0', add, Ca);

```

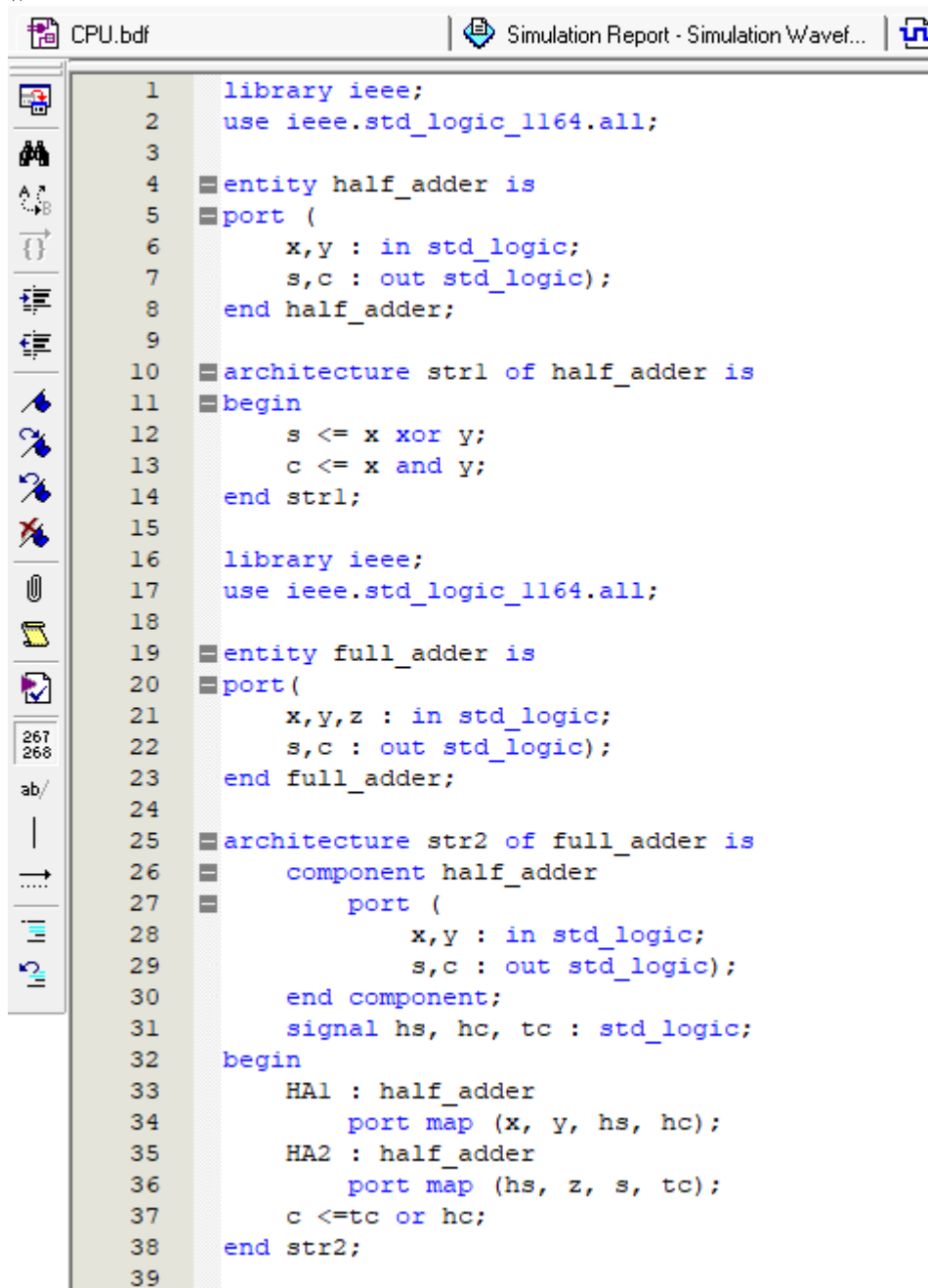


```

57     A2 : parallel_adder_vhdl
58     port map (A, B, '1', sub, Cs);
59     A3 : parallel_adder_vhdl
60     port map ("00000000", sub, '1', subf, Cf);
61     COUT <= sum;
62 end str;

```

全加器:

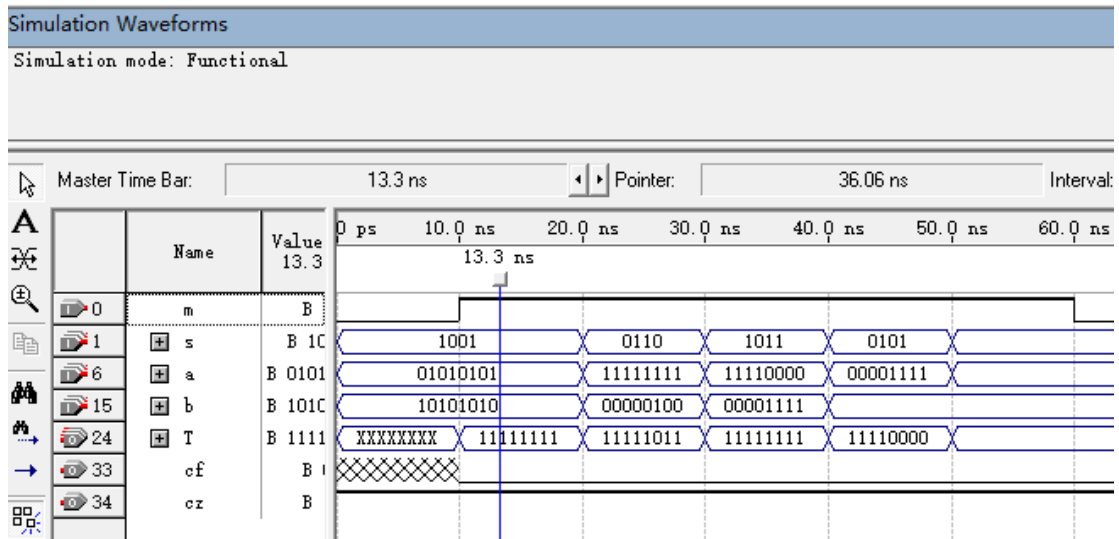


```

CPU.bdf | Simulation Report - Simulation Wavef...
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity half_adder is
5  port (
6      x,y : in std_logic;
7      s,c : out std_logic);
8  end half_adder;
9
10 architecture str1 of half_adder is
11 begin
12     s <= x xor y;
13     c <= x and y;
14 end str1;
15
16 library ieee;
17 use ieee.std_logic_1164.all;
18
19 entity full_adder is
20 port(
21     x,y,z : in std_logic;
22     s,c : out std_logic);
23 end full_adder;
24
25 architecture str2 of full_adder is
26 component half_adder
27 port (
28     x,y : in std_logic;
29     s,c : out std_logic);
30 end component;
31 signal hs, hc, tc : std_logic;
32 begin
33     HA1 : half_adder
34     port map (x, y, hs, hc);
35     HA2 : half_adder
36     port map (hs, z, s, tc);
37     c <= tc or hc;
38 end str2;
39

```

仿真结果如下:



0-10ns: m=0, 无效

10-20ns: 功能为加法, 运行正确

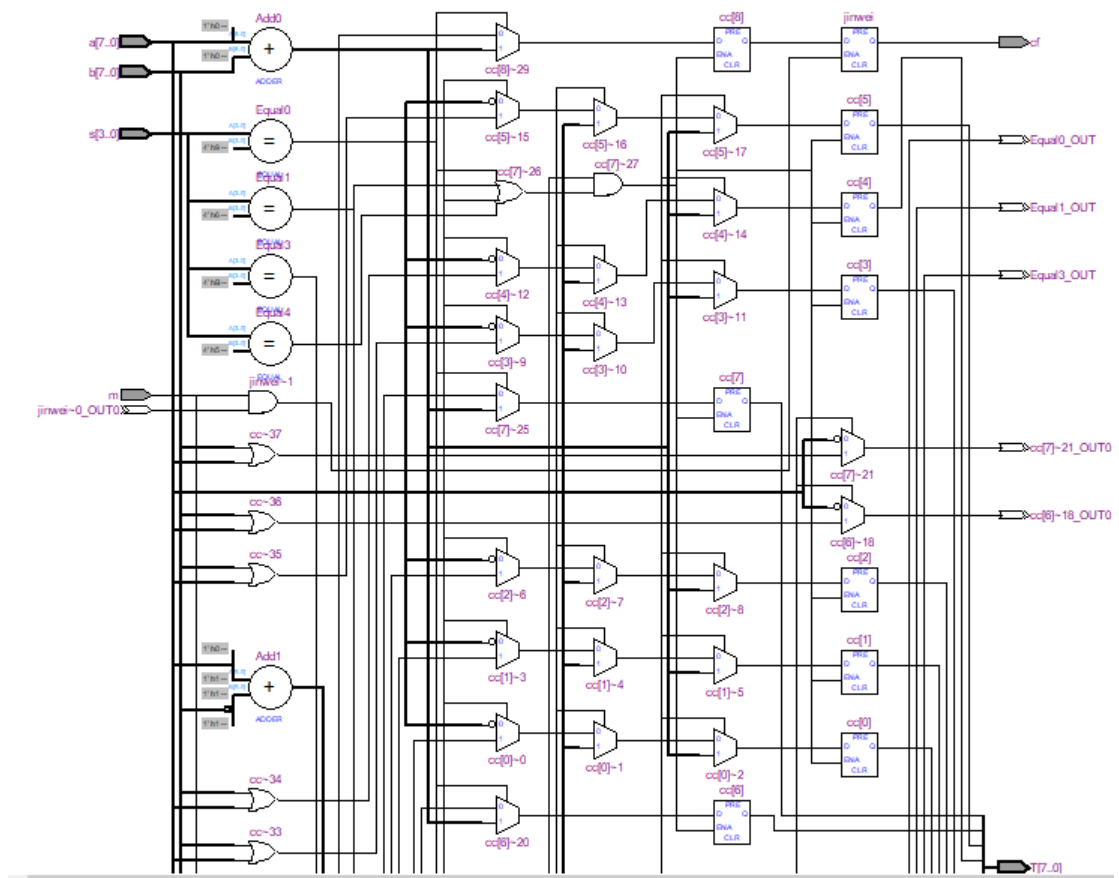
20-30ns: 功能为减法, 运行正确

30-40ns: 功能为求或, 运行正确

40-50ns: 功能为求非, 运行正确

STL 视图:

Page Title: alu_ex3

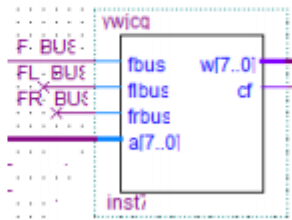


10. 移位逻辑

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。

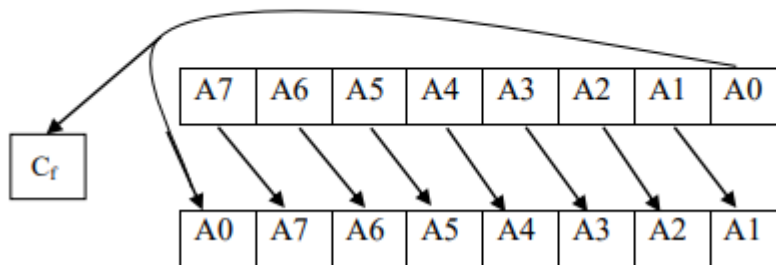
移位逻辑模块图：



指令具体功能如下：

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

RSR 循环移位操作具体如下：

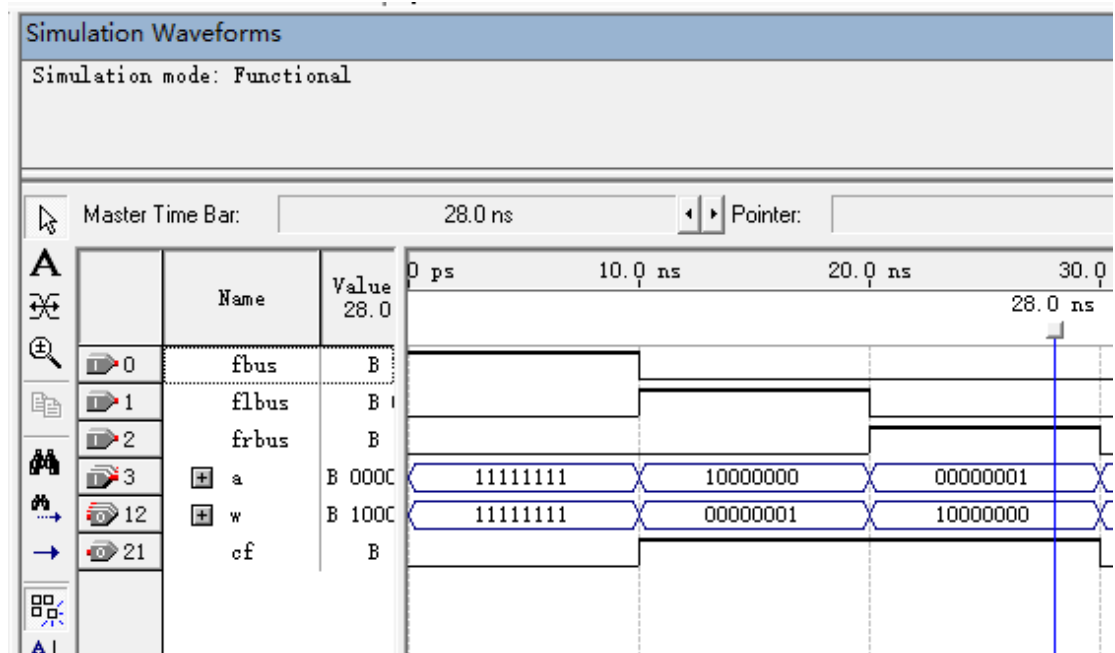


```

CPU.bdf | yiwei.vhd
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  entity yiwei is
5  port (fbus, flbus, frbus : in std_logic;
6        a : in std_logic_vector(7 downto 0);
7        cf : out std_logic;
8        w : out std_logic_vector(7 downto 0));
9  end yiwei;
10 architecture bhv of yiwei is
11 begin
12   process (fbus, flbus, frbus, a)
13   begin
14     if fbus='1' then
15       w<=a;
16       cf<='0';
17     elsif flbus='1' then
18       w<=a(6 downto 0) & a(7);
19       cf<=a(7);
20     elsif frbus='1' then
21       w<=a(0) & a(7 downto 1);
22       cf<=a(0);
23     else
24       w<="ZZZZZZZZ";
25       cf<='0';
26     end if;
27   end process;
28 end;
29

```

仿真结果如下：

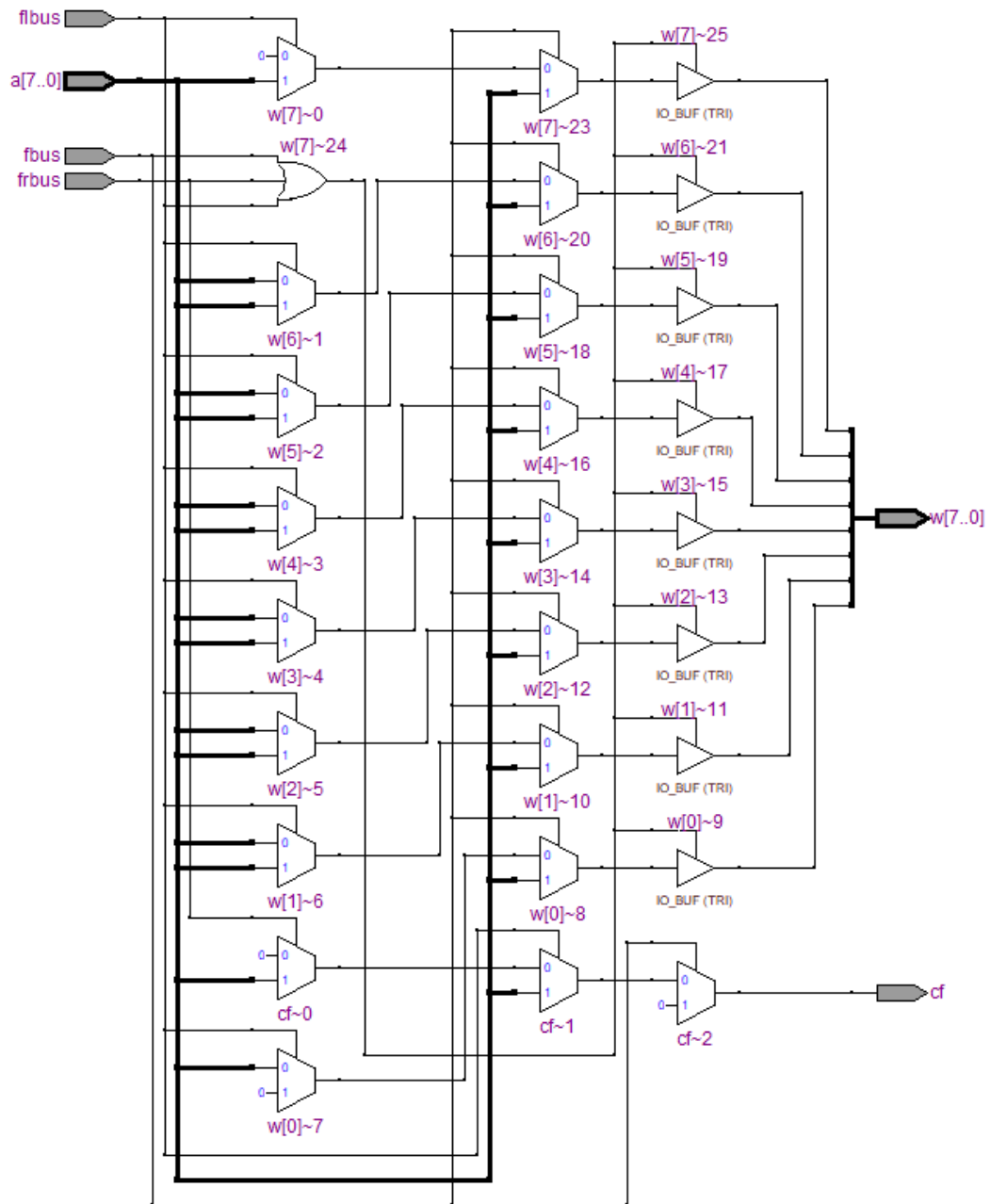


0-10ns: 简单输出

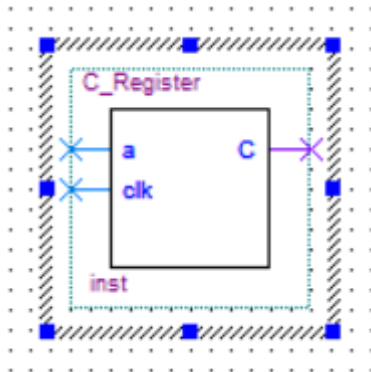
10-20ns: 左移, 将最左位补到最右位, 正确

20-30ns: 右移, 将最右位补到最左位, 正确

STL 视图



11.12 寄存器

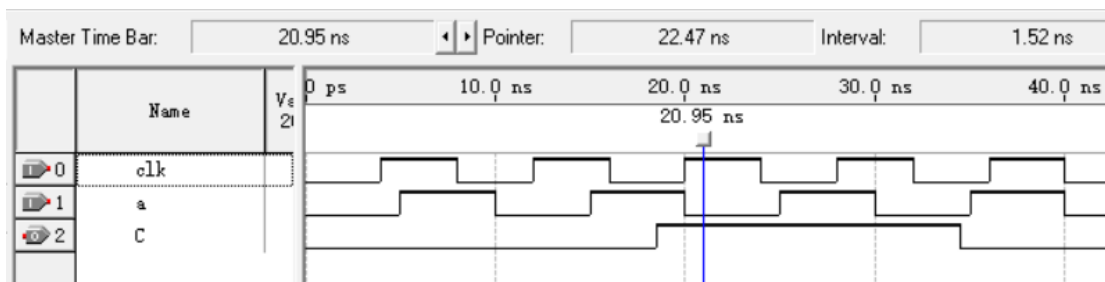


根据寄存器的功能可知：

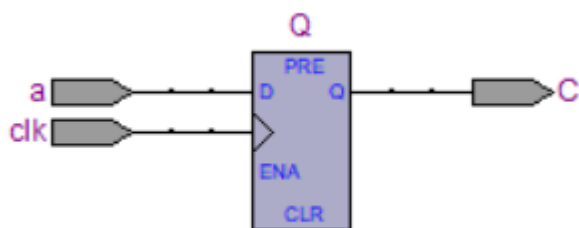
当处于时钟上升沿时，将输入数据 a 寄存在寄存器中
寄存器需要完成：对输入数 a 进行存储并且输出

```
CPU.bdf | abc yiwei.vhd
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity C_Register is
5  port(
6      a : in std_logic;
7      clk : in std_logic;
8      C : out std_logic);
9  end C_Register;
10
11 architecture str of C_Register is
12     signal Q : std_logic := '0';
13 begin
14     process(clk)
15     begin
16         if(clk'event and clk='1')then
17             Q <= a;
18         end if;
19     end process;
20     C <= Q;
21 end str;
```

仿真结果如下：



RTL 为：



四、系统测试

4.1 测试环境

采用基于FPGA进行数字逻辑电路设计的方法。

采用的软件是 QuartusII

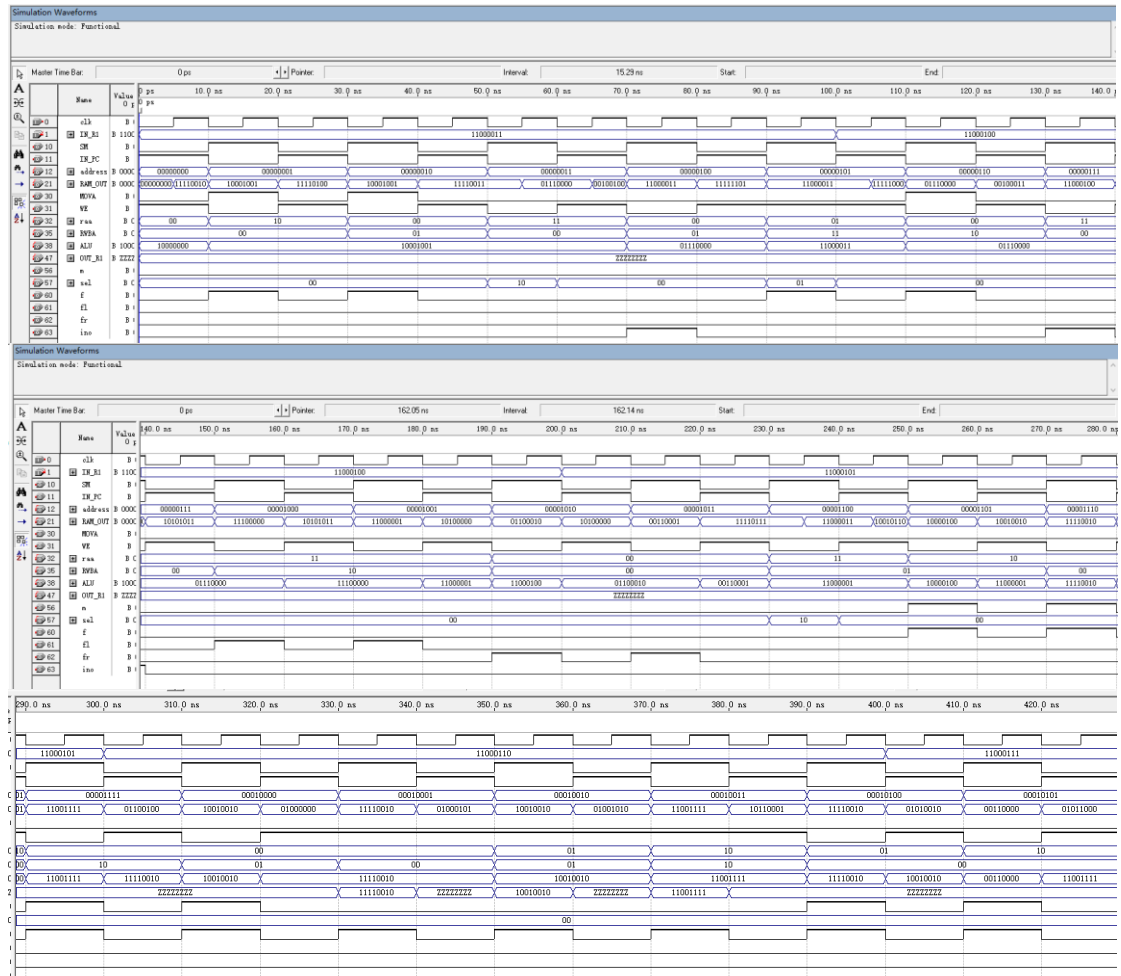
4.2 测试方法

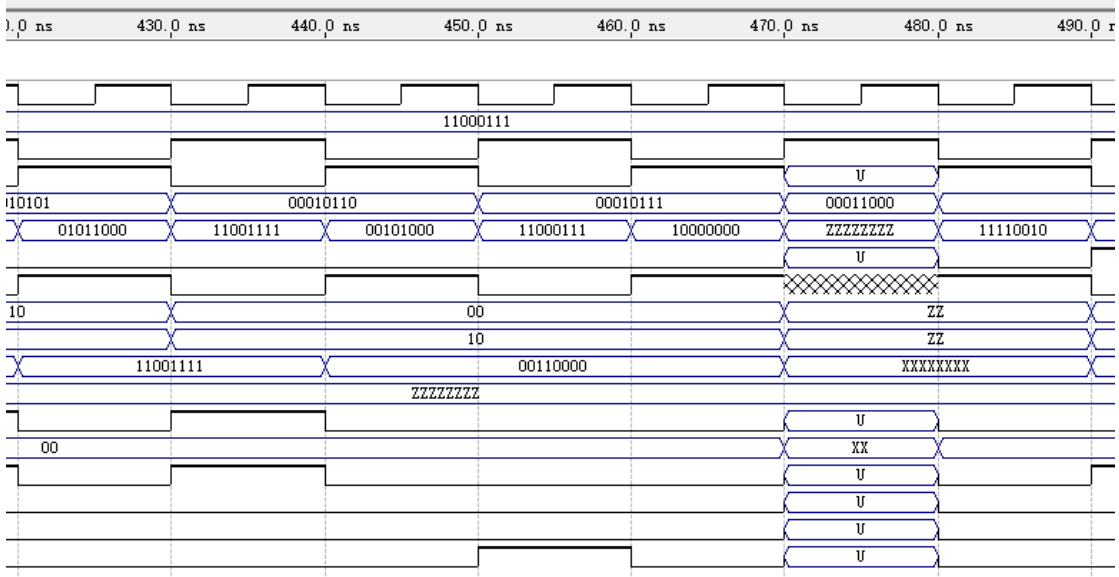
表1 指令系统表

汇编符号	功能	编码
MOV R1, R2	$(R2) \rightarrow R1$	1111 R1 R2
MOV M, R2	$(R2) \rightarrow (C)$	1111 11 R2
MOV R1, M	$((C)) \rightarrow R1$	1111 R1 11
ADD R1, R2	$(R1) + (R2) \rightarrow R1$	1001 R1 R2
SUB R1, R2	$(R1) - (R2) \rightarrow R1$	0110 R1 R2
OR R1, R2	$(R1) \vee (R2) \rightarrow R1$	1011 R1 R2
NOT R1	$\neg (R1) \rightarrow R1$	0101 R1 XX
RSR R1	$(R1)$ 循环右移一位 $\rightarrow R1$	1010 R1 00
RSL R1	$(R1)$ 循环左移一位 $\rightarrow R1$	1010 R1 11
JMP add	add $\rightarrow PC$	0001 00 00, address
JZ add	结果为 0 时 add $\rightarrow PC$	0001 00 01, address
JC add	结果有进位时 add $\rightarrow PC$	0001 00 10, address
IN R1	(开关 7-0) $\rightarrow R1$	0010 R1 XX
OUT R1	$(R1) \rightarrow$ 发光二极管 7-0	0100 R1 XX
NOP	$(PC) + 1 \rightarrow PC$	0111 00 00
HALT	停机	1000 00 00

针对功能表的每个功能，进行测试。

4.3 仿真结果





4.4 模型机性能分析

共用了 48 个周期完成 24 个功能的仿真。

5. 总结：

这次 cpu 实验，发现了几个之前测试能通过，但是整合起来错误的部件，发现了自身设计的一些逻辑漏洞，同时部分部件由于偷工减料，有些错误无法更改，需要重构。

新的部件 lpm 以及通用寄存器等，对我启发极大，学习制作自己简易的 cpu，实现简单的功能，提高了我的兴趣。