实验三 简易模型机中组合器件的实现

班级 计科 1903 姓名 陈旭 学号 201914020128

一、实验目的

1. 了解简易模型机的内部结构和工作原理。
2. 分析模型机的功能，设计 ALU 和移位逻辑。
3. 分析模型机的工作原理，设计模型机控制信号产生逻辑。

二、实验内容

1. 用 VHDL 语言设计ALU(函数发生器)。
2. 用 VHDL 语言设计移位逻辑。
3. 用 VHDL 语言设计控制信号产生逻辑。
4. 理解简易模型机的结构和工作原理。

三、实验方法

1、实验方法

- 采用基于 FPGA 进行数字逻辑电路设计的方法。
- 采用基本逻辑门电路和组合逻辑电路实现 ALU(函数发生器)，移位逻辑，控制信号产生逻辑。
- 采用的软件工具是 Quartus II。

2、实验软件操作步骤

1. ALU

1、 新建，编写源代码。

(1)选择保存项和芯片类型：【File】-【new project wizard】-【next】（设置文件路径为 C:\Users\86150\Desktop\mylearn\vscodec++\logicandcomputerdesignfundamentals\experiment3\alu 设置 project name 为 alu）-【next】（设置文件名 alu）-【next】（设置芯片类型为【cyclone-EP1CT144C8】）-【finish】

(2). 新建：【file】-【new】（第二个 VHDL File）-【OK】

2、 写好源代码，保存文件（alu.vhd）。

3、 编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译，编译结果有警告，编译成功。

4、 波形仿真及验证。新建一个 vector waveform file。按照程序所述插入 S, M, inx， iny，Cf, Zf, outbus 七个节点（inx，iny 为八位输入节点向量，S 为四位输出节点向量， outbus 为八位输出节点向量,Cf,Zf 为输出结点，M 为输入节点）。(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】)。任意设置 inx, iny, S, M 的输入波形…点击保存按钮保存。（操作为：点击 name（如：enable））-右击-【value】-【count】（如设置 binary；start value=0; count value=5.0ns），同理设置 name b（如 0, 1, 5），保存）。然后【start simulation】，出 name C 的输出图。

5、 时序仿真和功能仿真。

6、 查看 RTL Viewer：【Tools】－【netlist viewer】－【RTL viewer】。

2. 移位逻辑

1、 新建，编写源代码。

(1)选择保存项和芯片类型：【File】－【new project wizard】－【next】（设置文件路径为 C:\Users\86150\Desktop\mylearn\vscodec++\logicandcomputerdesignfundamentals\experiment3\rslogic 设置 project name 为 rslogic）－【next】（设置文件名 rslogic）－【next】（设置芯片类型为【cyclone-EP1CT144C8】）－【finish】

(2). 新建：【file】－【new】（VHDL File）－【OK】

2、 写好源代码，保存文件（rslogic.vhd）。

3、 编译与调试。确定源代码文件为当前工程文件，点击【processing】－【start compilation】进行文件编译，编译结果有警告，编译成功。

4、 波形仿真及验证。新建一个 vector waveform file。按照程序所述插入 fbus, frrbus, frlbus, inbus, outbus2, Cf 六个节点（inbus 为八位输入节点向量, outbus2 为八位输出节点向量,Cf 为输出结点，fbus, frrbus, frlbus 为输入节点）。(操作为：右击 －【insert】－【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】)。任意设置 inx, iny, S, M 的输入波形…点击保存按钮保存。(操作为：点击 name（如：enable））-右击-【value】-【count】(如设置 binary; start value=0; count value=5.0ns)，同理设置 name b（如 0, 1, 5），保存）。然后【start simulation】，出 name C 的输出图。

5、 时序仿真和功能仿真。

6、 查看 RTL Viewer：【Tools】－【netlist viewer】－【RTL viewer】。

3. 控制信号产生逻辑

1. 新建，编写源代码。

(1)选择保存项和芯片类型：【File】－【new project wizard】－【next】（设置文件路径为 C:\Users\86150\Desktop\mylearn\vscodec++\logicandcomputerdesignfundamentals\experiment3\signal_generate 设置 project name 为 signal_generate）－【next】（设置文件名 signal_generate）－【next】（设置芯片类型为【cyclone-EP1CT144C8】）－【finish】

(2). 新建：【file】－【new】（VHDL File）－【OK】

2. 写好源代码，保存文件（rslogic.vhd）。
3. 编译与调试。确定源代码文件为当前工程文件，点击【processing】－【start compilation】进行文件编译，编译结果有警告，编译成功。
4. 波形仿真及验证。新建一个 vector waveform file。按照程序所述插入 sm, mov1, mov2, mov3, add, sub, or1, not1, rsr, rsl, jmp, jz, jc, in1, out1, Zf, Cf, nop, halt，in_en, out_en, ldpc, inc, we, xl, dl, m, fbus, frr, frl, ld, Cf_en, Zf_en, Sm_en，ir, wa, ra, madd, S 等节点。(操作为：右击 －【insert】－【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】)。任意设置结点的输入波形…点击保存按钮保存。(操作为：点击 name（如：enable））-右击-【value】-【count】(如设置 binary; start value=0; count value=5.0ns)，同理设置 name b（如 0, 1, 5），保存）。然后【start simulation】，出 name C 的输出图。
5. 时序仿真和功能仿真。
6. 查看 RTL Viewer：【Tools】－【netlist viewer】－【RTL viewer】。

四、实验过程

a) 编译过程

- **源代码（VHDL 设计）如下**

1. ALU

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

entity alu is
    port(
        S: in std_logic_vector(3 downto 0);
        M: in std_logic;
        inx, iny: in std_logic_vector(7 downto 0);
        Cf, Zf: out std_logic:='0';
        outbus: out std_logic_vector(7 downto 0):="00000000");
end alu;

architecture alu of alu is
    signal result:std_logic_vector(8 downto 0):="000000000";
    signal sa, sb:std_logic_vector(8 downto 0);
    begin
        sa<='0'&inx;
        sb<='0'&iny;
        process(S, M, inx, iny)
        begin
            if (M='0') then --math compute or zhitong
                if (S="1001") then
                    outbus<=inx+iny;
                    result<=sa+sb;
                    Cf<=result(8);
                    if (result="000000000") then
                        Zf<='1';
                    else Zf<='0';
                    end if;
                elsif(S="0110") then
                    outbus<=iny-inx;
                    result<=sb-sa;
                    Cf<=result(8);
                    if(result="000000000") then
                        Zf<='1';
                    else Zf<='0';
                    end if;
                elsif(S="1010") then
                    Cf<='0';
```

```
                    Zf<='0';
                    outbus<=iny;
                elsif(S="0100") then
                    Cf<='0';
                    Zf<='0';
                    outbus<=iny;
                else
                    outbus<=iny;
                    Cf<='0';
                    Zf<='0';
                end if;
            else --logic compute
                if(S="1011") then
                    outbus<= inx or iny;
                    Cf<='0';
                    Zf<='0';
                elsif(S="0101") then
                    outbus<=not iny;
                    Cf<='0';
                    Zf<='0';
                end if;
            end if;
        end process;
    end architecture alu;
```

2. 移位逻辑

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

entity rslogic is
    port(
        fbus, frlbus, frrbus:in std_logic;
        inbus:in std_logic_vector(7 downto 0);
        outbus2: out std_logic_vector(7 downto 0);
        Cf:out std_logic:='0'
    );
end rslogic;

architecture rslogic of rslogic is
    begin
        process(fbus, frlbus, frrbus)
```

```
            begin
            if(fbus='1') then
                outbus2<=inbus;
                Cf<='0';
            else
                if (frlbus='1') then
                    outbus2<=inbus(6 downto 0)&inbus(7);
                    Cf<=inbus(7);
                elsif(frrbus='1') then
                    outbus2<=inbus(0)&inbus(7 downto 1);
                    Cf<=inbus(0);
                else
                    outbus2<="ZZZZZZZZ";
                    Cf<='0';
                end if;
            end if;
            end process;
    end architecture rslogic;
```

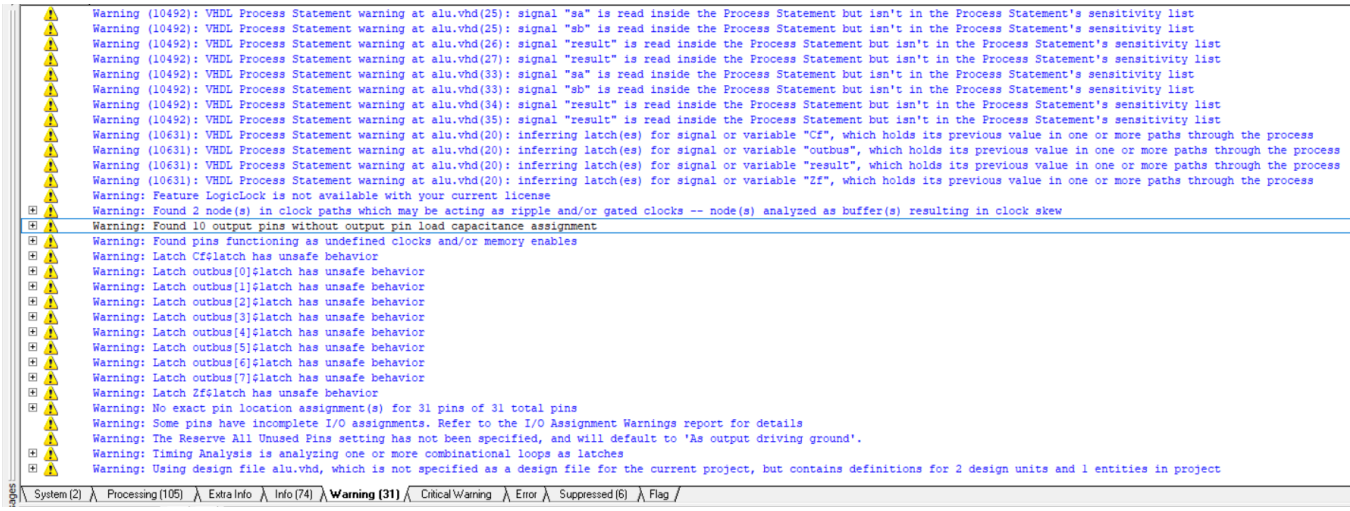
3. 控制信号产生逻辑

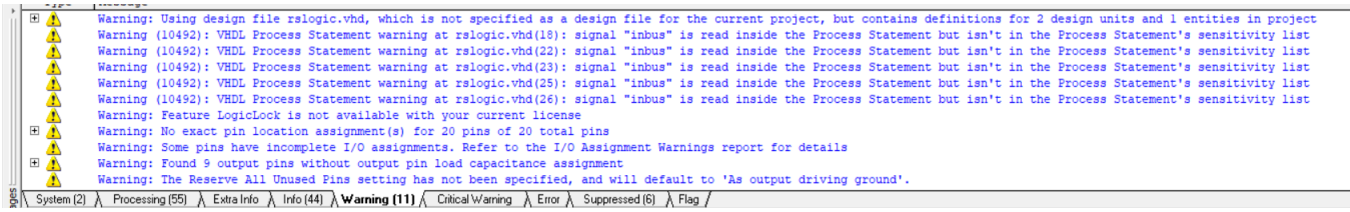
```
library ieee;
use ieee.std_logic_1164.all;

entity signal_generate is
    port(
        sm, mov1, mov2, mov3, add, sub, or1, not1, rsr, rsl, jmp, jz, j
c, in1, out1, Zf, Cf, nop, halt: in std_logic;
        ir:in std_logic_vector(7 downto 0);
        ra, wa, madd:out std_logic_vector(1 downto 0);
        s: out std_logic_vector(3 downto 0);
        in_en, out_en, ldpc, inc, we, xl, dl, m, fbus, frr, frl, ld, Cf
_en, Zf_en, Sm_en: out std_logic
    );
end signal_generate;

architecture signal_generate of signal_generate is
    signal in0: std_logic:='1';
    begin
        process(sm, mov1, mov2, mov3, add, or1, sub, not1, rsr, rsl, jm
p, jz, jc, in1, out1, Zf, Cf, nop, halt, ir)
```

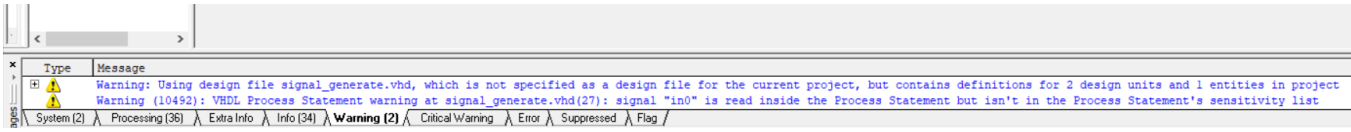
```
        begin
            ld<=not sm;
            ldpc<=(jc and Cf) or (jz and Zf) or jmp;
            inc <= (jz and (not Zf)) or (jc and (not Cf)) or nop or (no
t sm);
            if (ir(7 downto 4)="0010") then
                in0<='1';
            else
                in0<='0';
            end if;
            we <= not(mov1 or mov3 or add or sub or or1 or not1 or rsr
or rsl or in0) or (not sm);
            ra <= ir(1 downto 0);
            wa <= ir(3 downto 2);
            if (sm='1' and mov3='1') then
                madd<="01";
            elsif (sm='1' and mov2='1') then
                madd<="10";
            else
                madd<="00";
            end if;
            s<=ir(7 downto 4);
            xl<=mov2;
            dl<=mov3 or jmp or (Zf and jz) or (Zf and jc) or (not sm);
            if ((ir(7 downto 4)="1001") or (ir(7 downto 4)="0110") or (
ir(7 downto 4)="1010") or (ir(7 downto 4)="1111") or (ir(7 downto 4)="0
100")) then
                m<='0';
            else
                m<='1';
            end if;

            frl<=rsl;
            frr<=rsr;
            fbus<= mov1 or mov2 or add or sub or or1 or not1 or rsr or
rsl;
            Cf_en<=add or sub or or1 or rsr or rsl;
            Zf_en<=add or sub or or1 ;
            Sm_en<=not halt;
            in_en<=in1;
            out_en<=out1;
```

```
        end process;
    end architecture signal_generate;
```

b)编译、调试过程



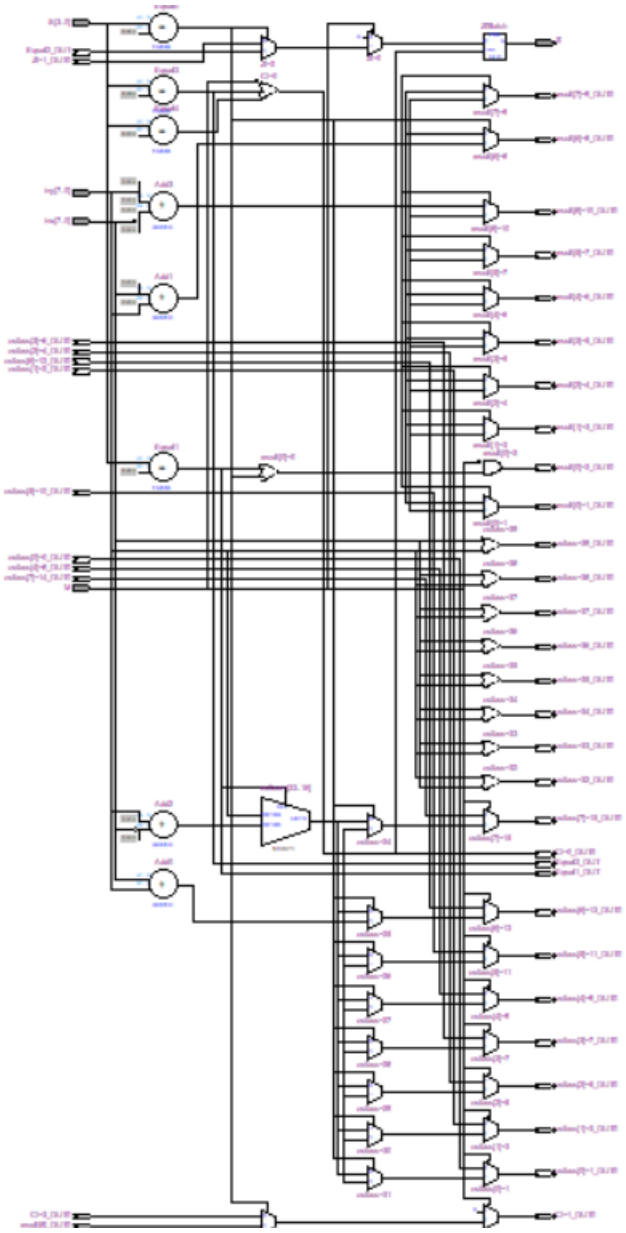




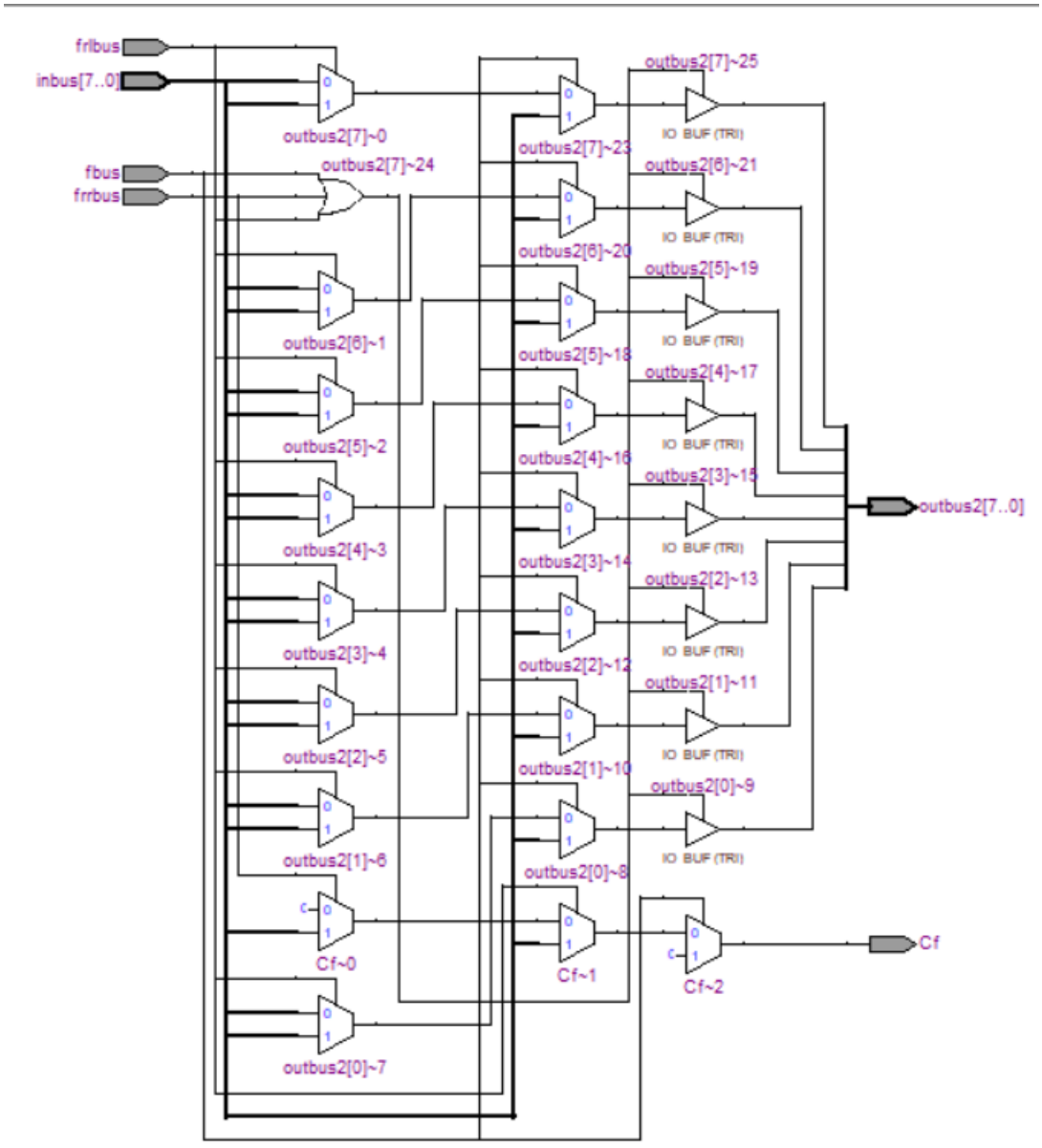
自上至下依次为 ALU，移位逻辑，控制信号产生逻辑的编译器提示信息。
三者编译器均给出警告，但均能够编译通过。
三者资源消耗基本都趋近于 0.

c) RTL 视图

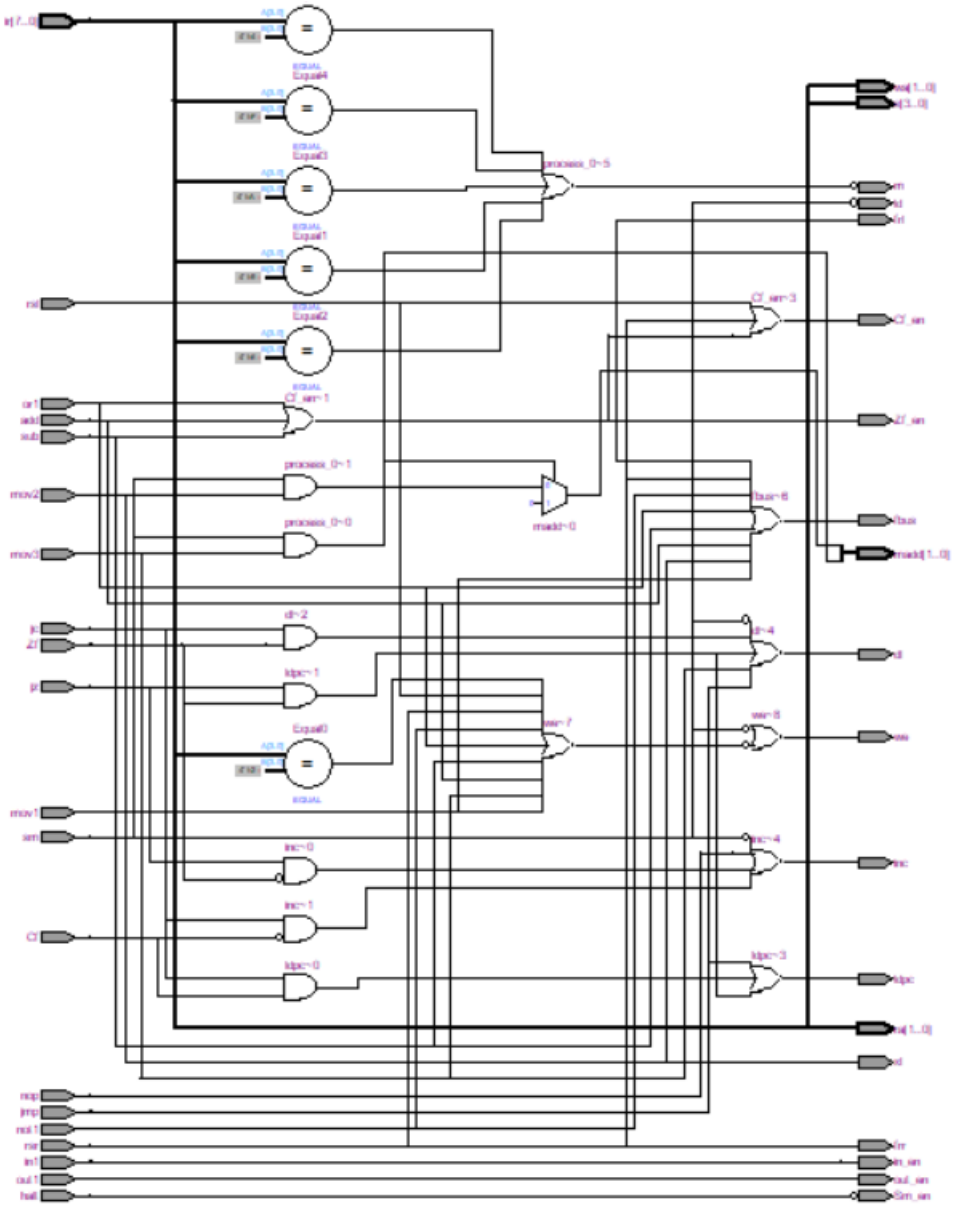
1. ALU



2. 位移逻辑


frlbus
inbus[7..0]
fbus
frrbus
outbus2[7]~0
outbus2[7]~24
outbus2[6]~1
outbus2[5]~2
outbus2[4]~3
outbus2[3]~4
outbus2[2]~5
outbus2[1]~6
Cf~0
outbus2[0]~7
outbus2[7]~25
outbus2[7]~23
outbus2[6]~21
outbus2[6]~20
outbus2[5]~19
outbus2[5]~18
outbus2[4]~17
outbus2[4]~16
outbus2[3]~15
outbus2[3]~14
outbus2[2]~13
outbus2[2]~12
outbus2[1]~11
outbus2[1]~10
outbus2[0]~9
outbus2[0]~8
Cf~1
Cf~2
IO BUF (TRI)
outbus2[7..0]
Cf

3. 控制信号产生逻辑



d)结果分析及结论

- ALU

  当输入M为0，执行算术运算或者直通；此时

  当S为“1001”执行inx+iny；

  当S为“0110”执行iny-inx；

  当S为“1001”、“1010”、“0100”或者其他，直通。

  当输入M为1，执行逻辑运算；此时

  当S为“1011”执行inx or iny；

  当S为“0110”执行not iny；

- 移位逻辑

  当输入 fbus 为 1 时，直通。

  当输入 fbus 为 0 时：

  frlbus 为 1 时循环左移

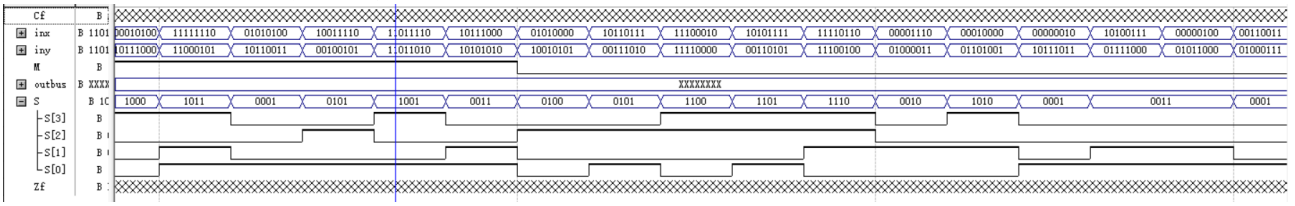
frrbus 为 0 时循环右移
否则输出为高阻态(“ZZZZZZZZ”)

- 控制信号产生逻辑
  根据输入信号产生按表格对应的控制信号，如 ld 为 not sm;等

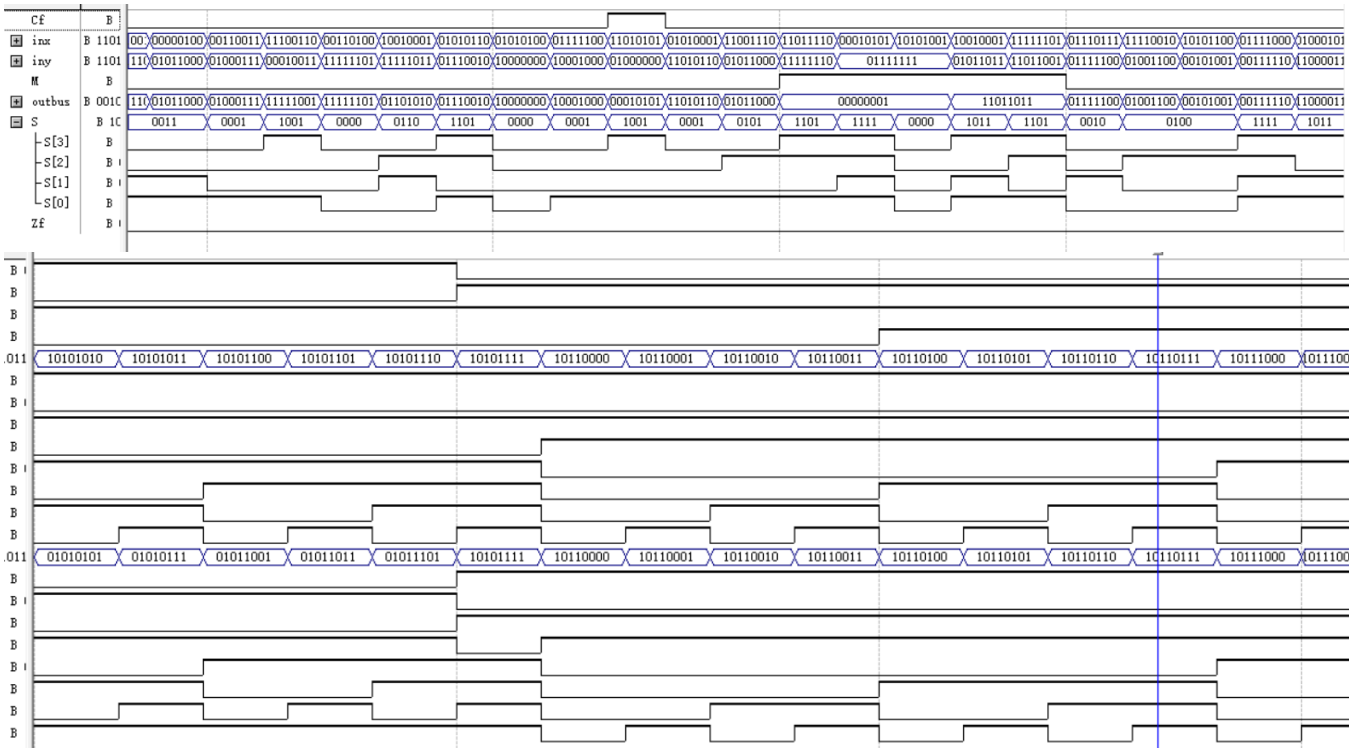
d) 波形仿真

1. ALU

a) 波形仿真过程（过程详见实验步骤）



b) 波形仿真波形图



c) 结果分析及结论

以第一个周期为例

当输入M为0，执行算术运算或者直通；此时

当S为“1001”执行inx+iny, 输出正确

当S为“0110”执行iny-inx, 输出正确

当S为“1001”、“1010”、“0100”或者其他，直通，输出正确

当输入M为1，执行逻辑运算；此时

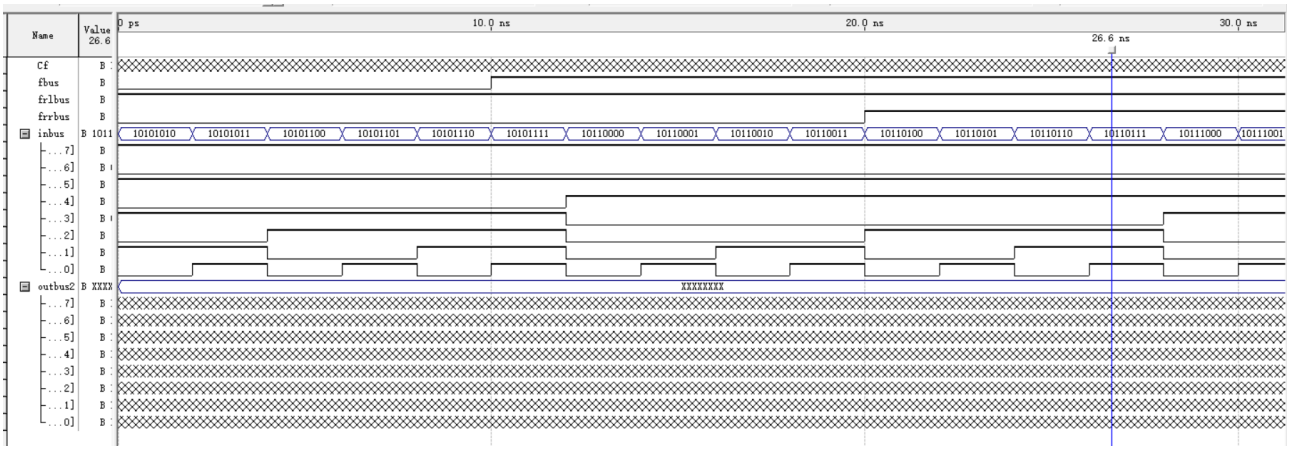
当S为“1011”执行inx or iny，输出正确;

当S为“0110”执行not iny;，输出正确

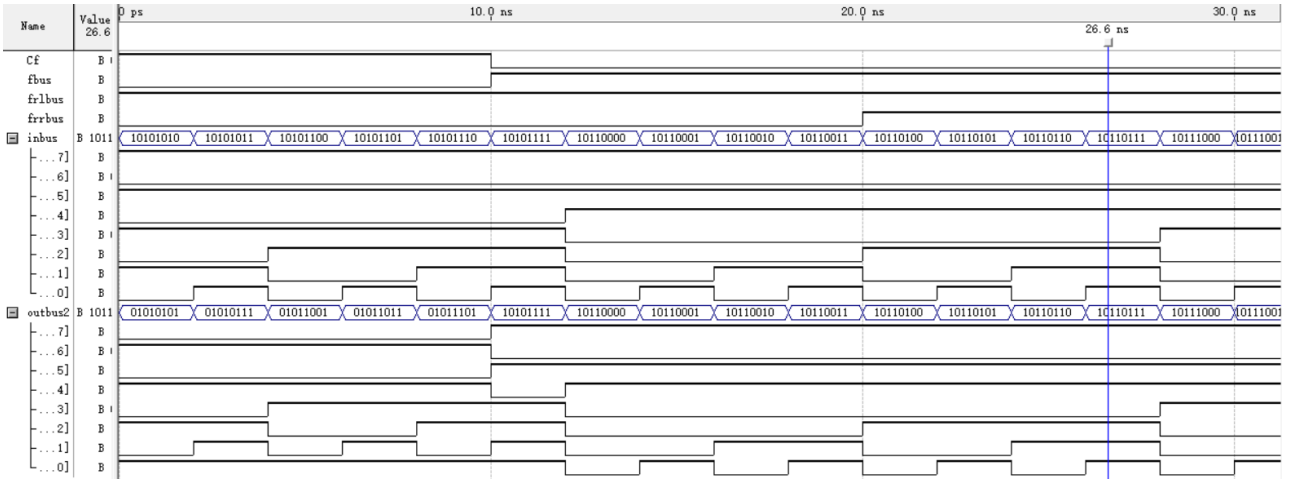
结果正确

2. 移位逻辑

a) 波形仿真过程（过程详见实验步骤）



b) 波形仿真波形图



c) 结果分析及结论

由上图得：

当输入 fbus 为 1 时，直通，输出正确

当输入 fbus 为 0 时：

frlbus 为 1 时循环左移，输出正确

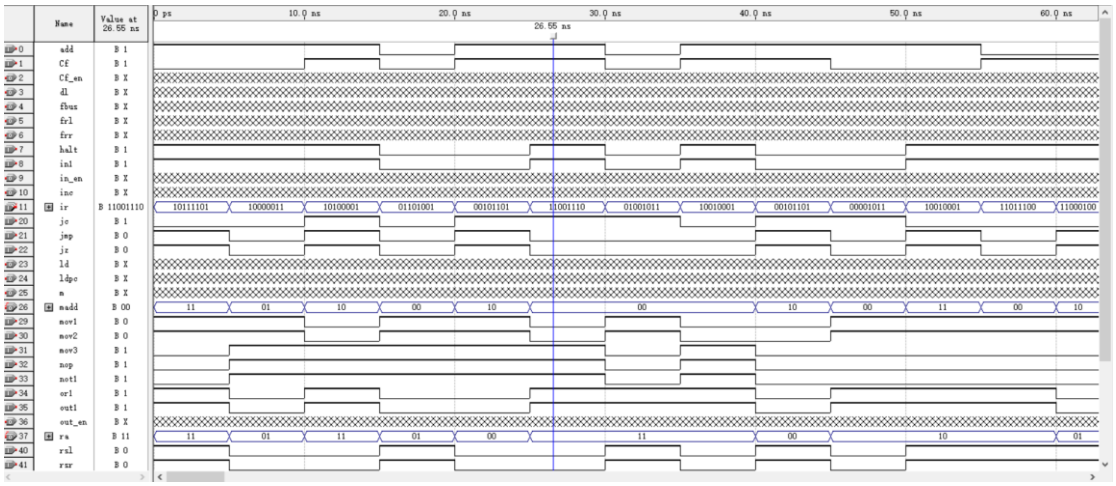
frrbus 为 0 时循环右移，输出正确

否则输出为高阻态(“ZZZZZZZZ”)，输出正确(上图因为图片大小等原因未能显示，但是输出的黑色粗线的确存在)

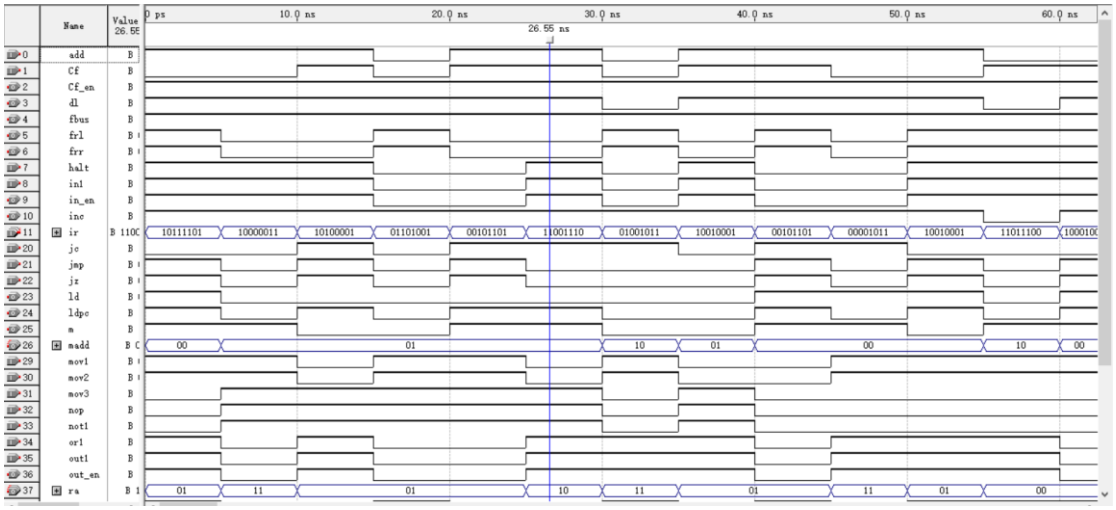
结果正确

3. **控制信号产生逻辑**

a) 波形仿真过程（过程详见实验步骤）



b) 波形仿真波形图



e) 时序仿真

- ALU

a) 时序仿真过程

做好上述步骤后，编译【classic timing analysis】-在 compilation report 中选择【timing analysis】

-【tpd】（引脚到引脚的延时）

b) 时序仿真图

tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	11.812 ns	S[2]	outbus[6]
2	N/A	None	11.452 ns	S[2]	outbus[3]
3	N/A	None	11.271 ns	S[0]	outbus[6]
4	N/A	None	11.181 ns	inx[6]	outbus[6]
5	N/A	None	11.057 ns	S[3]	outbus[6]
6	N/A	None	10.928 ns	iny[5]	outbus[6]
7	N/A	None	10.922 ns	S[1]	outbus[6]
8	N/A	None	10.911 ns	S[0]	outbus[3]
9	N/A	None	10.887 ns	S[2]	outbus[4]
10	N/A	None	10.796 ns	inx[0]	outbus[6]
11	N/A	None	10.718 ns	iny[0]	outbus[6]
12	N/A	None	10.697 ns	S[3]	outbus[3]
13	N/A	None	10.562 ns	S[1]	outbus[3]
14	N/A	None	10.519 ns	iny[6]	outbus[6]
15	N/A	None	10.436 ns	inx[0]	outbus[3]
16	N/A	None	10.428 ns	iny[1]	outbus[6]
17	N/A	None	10.381 ns	inx[1]	outbus[6]
18	N/A	None	10.366 ns	S[2]	outbus[7]
19	N/A	None	10.358 ns	iny[0]	outbus[3]
20	N/A	None	10.353 ns	S[2]	outbus[1]
21	N/A	None	10.346 ns	S[0]	outbus[4]
22	N/A	None	10.323 ns	iny[4]	outbus[6]
23	N/A	None	10.220 ns	inx[2]	outbus[6]
24	N/A	None	10.192 ns	iny[3]	outbus[6]
25	N/A	None	10.189 ns	iny[2]	outbus[6]
26	N/A	None	10.133 ns	inx[3]	outbus[6]
27	N/A	None	10.132 ns	S[3]	outbus[4]
28	N/A	None	10.125 ns	inx[5]	outbus[6]
29	N/A	None	10.124 ns	S[2]	outbus[5]
30	N/A	None	10.068 ns	iny[1]	outbus[3]
31	N/A	None	10.037 ns	S[2]	outbus[2]
32	N/A	None	10.021 ns	M	outbus[6]
33	N/A	None	10.021 ns	inx[1]	outbus[3]
34	N/A	None	10.008 ns	S[2]	outbus[0]

tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
34	N/A	None	10.008 ns	S[2]	outbus[0]
35	N/A	None	9.997 ns	S[1]	outbus[4]
36	N/A	None	9.871 ns	inx[0]	outbus[4]
37	N/A	None	9.860 ns	inx[2]	outbus[3]
38	N/A	None	9.859 ns	inx[4]	outbus[6]
39	N/A	None	9.829 ns	iny[2]	outbus[3]
40	N/A	None	9.825 ns	S[0]	outbus[7]
41	N/A	None	9.812 ns	S[0]	outbus[1]
42	N/A	None	9.810 ns	iny[3]	outbus[3]
43	N/A	None	9.793 ns	iny[0]	outbus[4]
44	N/A	None	9.745 ns	inx[3]	outbus[3]
45	N/A	None	9.692 ns	inx[6]	outbus[7]
46	N/A	None	9.611 ns	S[3]	outbus[7]
47	N/A	None	9.598 ns	S[3]	outbus[1]
48	N/A	None	9.583 ns	S[0]	outbus[5]
49	N/A	None	9.503 ns	iny[1]	outbus[4]
50	N/A	None	9.496 ns	S[0]	outbus[2]
51	N/A	None	9.476 ns	S[1]	outbus[7]
52	N/A	None	9.467 ns	S[0]	outbus[0]
53	N/A	None	9.463 ns	S[1]	outbus[1]
54	N/A	None	9.456 ns	inx[1]	outbus[4]
55	N/A	None	9.439 ns	iny[5]	outbus[7]
56	N/A	None	9.376 ns	iny[4]	outbus[4]
57	N/A	None	9.369 ns	S[3]	outbus[5]
58	N/A	None	9.337 ns	inx[0]	outbus[1]
59	N/A	None	9.307 ns	inx[0]	outbus[7]
60	N/A	None	9.299 ns	M	outbus[3]
61	N/A	None	9.295 ns	inx[2]	outbus[4]
62	N/A	None	9.282 ns	S[3]	outbus[2]
63	N/A	None	9.267 ns	iny[3]	outbus[4]
64	N/A	None	9.264 ns	iny[2]	outbus[4]
65	N/A	None	9.259 ns	iny[0]	outbus[1]
66	N/A	None	9.253 ns	S[3]	outbus[0]
67	N/A	None	9.234 ns	S[1]	outbus[5]

	Slack	Required P2P Time	Actual P2P Time	From	To
67	N/A	None	9.234 ns	S[1]	outbus[5]
68	N/A	None	9.229 ns	iny[0]	outbus[7]
69	N/A	None	9.212 ns	iny[5]	outbus[5]
70	N/A	None	9.208 ns	inx[3]	outbus[4]
71	N/A	None	9.203 ns	M	outbus[4]
72	N/A	None	9.147 ns	S[1]	outbus[2]
73	N/A	None	9.118 ns	S[1]	outbus[0]
74	N/A	None	9.108 ns	inx[0]	outbus[5]
75	N/A	None	9.042 ns	iny[6]	outbus[7]
76	N/A	None	9.030 ns	iny[0]	outbus[5]
77	N/A	None	9.021 ns	inx[0]	outbus[2]
78	N/A	None	8.957 ns	inx[0]	outbus[0]
79	N/A	None	8.943 ns	iny[0]	outbus[2]
80	N/A	None	8.941 ns	iny[1]	outbus[1]
81	N/A	None	8.939 ns	iny[1]	outbus[7]
82	N/A	None	8.906 ns	inx[4]	outbus[4]
83	N/A	None	8.892 ns	inx[1]	outbus[7]
84	N/A	None	8.892 ns	iny[0]	outbus[0]
85	N/A	None	8.887 ns	inx[1]	outbus[1]
86	N/A	None	8.834 ns	iny[4]	outbus[7]
87	N/A	None	8.740 ns	iny[1]	outbus[5]
88	N/A	None	8.731 ns	inx[2]	outbus[7]
89	N/A	None	8.703 ns	iny[3]	outbus[7]
90	N/A	None	8.700 ns	iny[2]	outbus[7]
91	N/A	None	8.693 ns	inx[1]	outbus[5]
92	N/A	None	8.653 ns	iny[1]	outbus[2]
93	N/A	None	8.644 ns	inx[3]	outbus[7]
94	N/A	None	8.636 ns	inx[5]	outbus[7]
95	N/A	None	8.635 ns	iny[4]	outbus[5]
96	N/A	None	8.606 ns	inx[1]	outbus[2]
97	N/A	None	8.532 ns	inx[2]	outbus[5]
98	N/A	None	8.515 ns	M	outbus[1]
99	N/A	None	8.504 ns	iny[3]	outbus[5]
100	N/A	None	8.501 ns	iny[2]	outbus[5]

tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
89	N/A	None	8.703 ns	iny[3]	outbus[7]
90	N/A	None	8.700 ns	iny[2]	outbus[7]
91	N/A	None	8.693 ns	inx[1]	outbus[5]
92	N/A	None	8.653 ns	iny[1]	outbus[2]
93	N/A	None	8.644 ns	inx[3]	outbus[7]
94	N/A	None	8.636 ns	inx[5]	outbus[7]
95	N/A	None	8.635 ns	iny[4]	outbus[5]
96	N/A	None	8.606 ns	inx[1]	outbus[2]
97	N/A	None	8.532 ns	inx[2]	outbus[5]
98	N/A	None	8.515 ns	M	outbus[1]
99	N/A	None	8.504 ns	iny[3]	outbus[5]
100	N/A	None	8.501 ns	iny[2]	outbus[5]
101	N/A	None	8.445 ns	inx[3]	outbus[5]
102	N/A	None	8.433 ns	iny[7]	outbus[7]
103	N/A	None	8.423 ns	inx[2]	outbus[2]
104	N/A	None	8.415 ns	inx[5]	outbus[5]
105	N/A	None	8.404 ns	inx[7]	outbus[7]
106	N/A	None	8.399 ns	M	outbus[0]
107	N/A	None	8.386 ns	iny[2]	outbus[2]
108	N/A	None	8.370 ns	inx[4]	outbus[7]
109	N/A	None	8.359 ns	M	outbus[2]
110	N/A	None	8.209 ns	M	outbus[5]
111	N/A	None	8.171 ns	inx[4]	outbus[5]
112	N/A	None	8.103 ns	S[2]	Cf
113	N/A	None	7.823 ns	M	outbus[7]
114	N/A	None	7.403 ns	S[3]	Cf
115	N/A	None	7.242 ns	S[0]	Cf
116	N/A	None	7.138 ns	S[1]	Cf
117	N/A	None	7.008 ns	M	Cf
118	N/A	None	6.456 ns	S[2]	Zf
119	N/A	None	5.756 ns	S[3]	Zf
120	N/A	None	5.595 ns	S[0]	Zf
121	N/A	None	5.491 ns	S[1]	Zf
122	N/A	None	5.361 ns	M	Zf

c) 结果分析及结论

每个引脚只间相互传递产生的延时各不相同，挑选其中 p2p 时间的最大值，为 S [2]传递给 outbus[6]。为 11.812ns，故整体延时为 11.812ns。

- 移位逻辑

a) 时序仿真过程

做好上述步骤后，编译【classic timing analysis】-在 compilation report 中选择【timing analysis】-【tpd】（引脚到引脚的延时）

b) 时序仿真图

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	11.200 ns	frlbus	outbus2[7]
2	N/A	None	10.736 ns	frlbus	outbus2[3]
3	N/A	None	10.667 ns	frlbus	outbus2[4]
4	N/A	None	10.497 ns	frrbus	outbus2[7]
5	N/A	None	10.479 ns	inbus[2]	outbus2[3]
6	N/A	None	10.447 ns	inbus[3]	outbus2[4]
7	N/A	None	10.221 ns	inbus[3]	outbus2[3]
8	N/A	None	10.184 ns	fbus	outbus2[7]
9	N/A	None	10.033 ns	frrbus	outbus2[3]
10	N/A	None	9.964 ns	frrbus	outbus2[4]
11	N/A	None	9.880 ns	inbus[0]	outbus2[7]
12	N/A	None	9.766 ns	fbus	outbus2[4]
13	N/A	None	9.720 ns	fbus	outbus2[3]
14	N/A	None	9.547 ns	inbus[5]	outbus2[4]
15	N/A	None	9.543 ns	inbus[2]	outbus2[2]
16	N/A	None	9.512 ns	inbus[7]	outbus2[7]
17	N/A	None	9.356 ns	inbus[6]	outbus2[7]
18	N/A	None	9.342 ns	inbus[4]	outbus2[4]
19	N/A	None	9.155 ns	inbus[3]	outbus2[2]
20	N/A	None	9.126 ns	frlbus	outbus2[2]
21	N/A	None	9.111 ns	inbus[4]	outbus2[3]
22	N/A	None	9.095 ns	frlbus	outbus2[5]
23	N/A	None	8.900 ns	frlbus	outbus2[0]
24	N/A	None	8.890 ns	frlbus	outbus2[6]
25	N/A	None	8.579 ns	inbus[2]	outbus2[1]
26	N/A	None	8.507 ns	fbus	outbus2[5]
27	N/A	None	8.472 ns	inbus[1]	outbus2[2]
28	N/A	None	8.343 ns	frlbus	Cf
29	N/A	None	8.340 ns	frrbus	outbus2[2]
30	N/A	None	8.325 ns	inbus[0]	outbus2[0]
31	N/A	None	8.313 ns	frrbus	outbus2[5]
32	N/A	None	8.307 ns	inbus[6]	outbus2[5]
33	N/A	None	8.257 ns	inbus[5]	outbus2[5]

tpd					
	Slack	Required P2P Time	Actual P2P Time	From	To
21	N/A	None	9.111 ns	inbus[4]	outbus2[3]
22	N/A	None	9.095 ns	frlbus	outbus2[5]
23	N/A	None	8.900 ns	frlbus	outbus2[0]
24	N/A	None	8.890 ns	frlbus	outbus2[6]
25	N/A	None	8.579 ns	inbus[2]	outbus2[1]
26	N/A	None	8.507 ns	fbus	outbus2[5]
27	N/A	None	8.472 ns	inbus[1]	outbus2[2]
28	N/A	None	8.343 ns	frlbus	Cf
29	N/A	None	8.340 ns	frrbus	outbus2[2]
30	N/A	None	8.325 ns	inbus[0]	outbus2[0]
31	N/A	None	8.313 ns	frrbus	outbus2[5]
32	N/A	None	8.307 ns	inbus[6]	outbus2[5]
33	N/A	None	8.257 ns	inbus[5]	outbus2[5]
34	N/A	None	8.232 ns	fbus	outbus2[6]
35	N/A	None	8.203 ns	inbus[1]	outbus2[0]
36	N/A	None	8.197 ns	frrbus	outbus2[0]
37	N/A	None	8.187 ns	frrbus	outbus2[6]
38	N/A	None	8.164 ns	inbus[7]	outbus2[0]
39	N/A	None	8.162 ns	frlbus	outbus2[1]
40	N/A	None	8.122 ns	inbus[0]	Cf
41	N/A	None	8.102 ns	fbus	outbus2[2]
42	N/A	None	8.084 ns	inbus[4]	outbus2[5]
43	N/A	None	8.081 ns	fbus	outbus2[0]
44	N/A	None	8.037 ns	inbus[6]	outbus2[6]
45	N/A	None	7.994 ns	inbus[7]	outbus2[6]
46	N/A	None	7.818 ns	inbus[5]	outbus2[6]
47	N/A	None	7.674 ns	frrbus	Cf
48	N/A	None	7.595 ns	inbus[0]	outbus2[1]
49	N/A	None	7.477 ns	inbus[1]	outbus2[1]
50	N/A	None	7.456 ns	inbus[7]	Cf
51	N/A	None	7.350 ns	fbus	outbus2[1]
52	N/A	None	7.329 ns	fbus	Cf
53	N/A	None	7.318 ns	frrbus	outbus2[1]

c) 结果分析及结论

每个引脚只间相互传递产生的延时各不相同，挑选其中 p2p 时间的最大值，为 frlbus 传递给 outbus2[7]。为 11.200ns，故整体延时为 11.200ns。

tpd (引脚到引脚的延时)

- 控制信号产生逻辑

c) 时序仿真过程

做好上述步骤后，编译【classic timing analysis】-在 compilation report 中选择【timing

analysis】-【tpd】（引脚到引脚的延时）

d) 时序仿真图

tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	31.200 ns	or1	we
2	N/A	None	29.500 ns	add	we
3	N/A	None	29.300 ns	sub	we
4	N/A	None	28.300 ns	rsr	we
5	N/A	None	27.400 ns	or1	fbus
6	N/A	None	27.100 ns	rsl	we
7	N/A	None	25.900 ns	ir[5]	we
8	N/A	None	25.700 ns	add	fbus
9	N/A	None	25.500 ns	sub	fbus
10	N/A	None	25.500 ns	ir[7]	we
11	N/A	None	24.500 ns	rsr	fbus
12	N/A	None	24.300 ns	ir[6]	we
13	N/A	None	23.300 ns	rsl	fbus
14	N/A	None	23.200 ns	jc	inc
15	N/A	None	22.900 ns	Zf	inc
16	N/A	None	21.800 ns	jz	inc
17	N/A	None	21.700 ns	or1	Zf_en
18	N/A	None	21.500 ns	or1	Cf_en
19	N/A	None	21.100 ns	Zf	dl
20	N/A	None	21.100 ns	mov1	we
21	N/A	None	21.100 ns	jc	ldpc
22	N/A	None	20.900 ns	ir[5]	m
23	N/A	None	20.900 ns	Cf	inc
24	N/A	None	20.700 ns	ir[7]	m
25	N/A	None	20.600 ns	jc	dl
26	N/A	None	20.200 ns	ir[4]	we
27	N/A	None	20.000 ns	add	Zf_en
28	N/A	None	19.800 ns	sub	Zf_en
29	N/A	None	19.800 ns	add	Cf_en
30	N/A	None	19.700 ns	ir[6]	m
31	N/A	None	19.600 ns	sub	Cf_en
32	N/A	None	19.600 ns	jz	dl
33	N/A	None	19.500 ns	nop	inc
34	N/A	None	19.300 ns	not1	we

tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
37	N/A	None	18.900 ns	Cf	ldpc
38	N/A	None	18.900 ns	ir[7]	s[3]
39	N/A	None	18.700 ns	Zf	ldpc
40	N/A	None	18.700 ns	ir[5]	s[1]
41	N/A	None	18.600 ns	rsr	Cf_en
42	N/A	None	18.400 ns	rsl	frl
43	N/A	None	18.300 ns	rsr	frr
44	N/A	None	17.700 ns	mov1	fbus
45	N/A	None	17.500 ns	jz	ldpc
46	N/A	None	17.400 ns	rsl	Cf_en
47	N/A	None	17.400 ns	ir[6]	s[2]
48	N/A	None	17.300 ns	ir[3]	wa[1]
49	N/A	None	16.700 ns	out1	out_en
50	N/A	None	16.600 ns	ir[2]	wa[0]
51	N/A	None	16.400 ns	not1	fbus
52	N/A	None	16.000 ns	in1	in_en
53	N/A	None	15.300 ns	halt	Sm_en
54	N/A	None	15.300 ns	sm	inc
55	N/A	None	15.000 ns	ir[4]	m
56	N/A	None	14.400 ns	mov3	we
57	N/A	None	14.100 ns	sm	we
58	N/A	None	14.000 ns	sm	ld
59	N/A	None	13.800 ns	mov3	dl
60	N/A	None	13.800 ns	ir[1]	ra[1]
61	N/A	None	13.700 ns	mov2	fbus
62	N/A	None	13.700 ns	mov2	xl
63	N/A	None	13.300 ns	sm	dl
64	N/A	None	13.300 ns	ir[4]	s[0]
65	N/A	None	12.800 ns	mov3	madd[1]
66	N/A	None	12.700 ns	mov2	madd[1]
67	N/A	None	12.700 ns	sm	madd[0]
68	N/A	None	12.500 ns	sm	madd[1]
69	N/A	None	12.500 ns	mov3	madd[0]
70	N/A	None	12.400 ns	ir[0]	ra[0]

d) 结果分析及结论

每个引脚只间相互传递产生的延时各不相同，挑选其中 p2p 时间的最大值，为 or1 传递给 we。为 31.200ns，故整体延时为 31.200ns。

tpd (引脚到引脚的延时)

五、实验结论

1、思考题

1） 输出为高阻态至数据总线。按照上图分析，当不工作时各传入信号均为 0，此时输出为高阻态，此时移位逻辑所在的线路无法通过，数据在通用寄存器和 RAM 等之间移动。

2) 使用触发器将其存储起来，便于后续处理和控制其他模块

3) 可以使用基础逻辑门来将某信号产生条件根据其逻辑函数表达式连接起来，然后根据输出波形随输入的变化而变化的情况来判断其正确性即可

2、 实验总结与实验心得

本次实验的内容与要求是简易模型机中组合器件的 VHDL 语言实现以及理解简易模型机的结构和工作原理。涉及的能力有 VHDL 组合逻辑电路编程能力，组合电路的设计与分析能力，实验报告的撰写能力，实验过程分析总结能力。通过使用软件实现硬件，既培养了编码能力，又增加了硬件设计能力和社会实践能力。该次实验使我对组合电路的设计与 CPU 中命令的传递与选择有了更深刻的认识，巩固了组合电路的设计相关的知识点，同时为时序电路的设计与编程实现奠定了基础。我因此受益良多。