实验二 多路复用器及控制信号产生逻辑的实现

班级 计科 1808 姓名 张继伟 学号 201808010829

一、实验目的

1．熟悉多路复用器以及模型机的工作原理。
2．学会使用 VHDL 语言设计多路复用器。
3．掌握 generic 的使用，能运用设计参数化多路复用器。
4．学会使用 VHDL 语言设计模型机控制信号产生逻辑。

二、实验内容

1、使用 VHDL 语言设计多路复用器。
2、掌握 generic 的使用，能运用设计参数化多路复用器。
3、使用 VHDL 语言设计模型机控制信号产生逻辑。

三、实验方法

（一）实验方法

采用基于 FPGA 进行数字逻辑电路设计的方法。
采用的软件工具是 Quartus II。

1．多路复用器，又名多路选择器、多路开关

多路复用器是一个组合电路，它可以从多个输入中选择一个输入，并将信息直接传输到输出。选择哪一条输入线由一组输入变量控制，它们被称为选择输入。通常，2n 条输入线要 n 个选择输入，选择输入的位组合决定选择哪个输入线。例如 n=1 的 2-1 多路复用器。这个复用器有两个信息输入 I0 和 I1，一个单独的选择输入 S，电路的真值表如表 1 所示。

S	I_0	I_1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

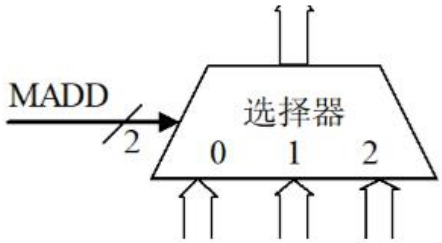
表 1 2-1 多路复用器真值表 分析真值表可知，如果选择输入 S=0，多路复用器输出为 I0 的值；

如果选择：

输入 S=1，多路开关输出 I1 的值。这样，S 不是选择输入 I0 就是选择输入 I1 到输出 Y。通过这些讨论，可以看出，2-1 多路复用器输出 Y 的方程式为

$$Y = \overline{S}I_0 + SI_1$$

2．8 重 3-1 多路复用器



3．运用 generic 设计参数化加法器，再调用定制为 4 位加法器。

a）参数化加法器

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

entity par_add is    generic(n:integer:=4);
   port(a,b:in std_logic_vector(n downto 0);
           s:out std_logic_vector(n downto 0);
           c:out std_logic);
 end par_add;

architecture exa of par_add is signal t:std_logic_vector(n+1 downto 0);
 begin
    t<=('0'&a)+('0'&b);
    s<=t(n downto 0);
    c<=t(n+1); end exa;
```

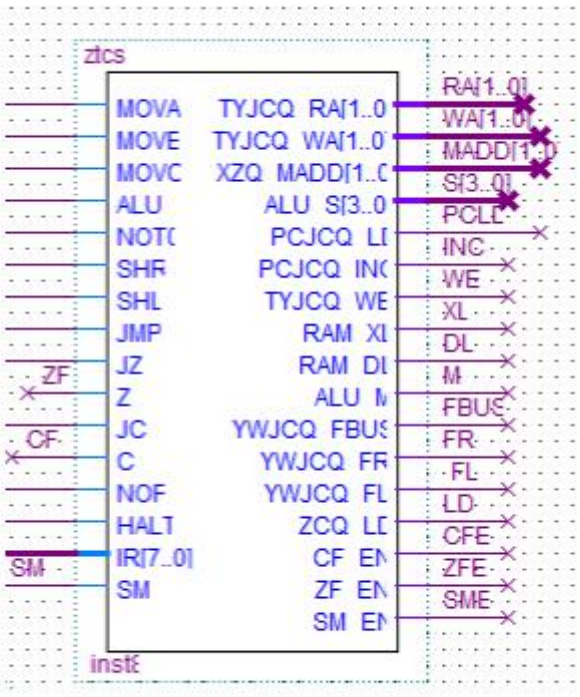
b）设计参数化加法器，定制为 4 位加法器

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

entity add3 is
   port(x,y:in std_logic_vector(3 downto 0);
           cin:in std_logic;
           s:out std_logic_vector(3 downto 0);
           cout:out std_logic); end add3;

architecture exa of add3 is component par_add is
   generic(n:integer:=4);
   port(a,b:in std_logic_vector(n downto 0);
         s:out std_logic_vector(n downto 0);
           c:out std_logic); end component;
 signal sum:std_logic_vector(3 downto 0);
 signal mid:std_logic_vector(4 downto 0);
 signal c3:std_logic;
 begin
      g0:par_add generic map(n=>3)port map(a=>x,b=>y,s=>sum,c=>c3);
      mid<=(c3&sum)+("0000"&cin);
      s<=mid(3 downto 0);
      cout<=mid(4); end exa;
```

4．控制信号产生逻辑



1、 实验步骤

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+设置 project name 为 xor2）-【next】（设置文件名 xor2.vhd—在【add】）-【properties】（type=AHDL）-【next】（family=FLEX10K；name=EPF10K10TI144-4）-【next】-【finish】

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

2、写好源代码，保存文件（xor2.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 a,b,c 三个节点（a、b 为输入节点，c 为输出节点）。(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】)。任意设置 a,b 的输入波形…点击保存按钮保存。（操作为：点击 name（如：A））-右击-【value】-【clock】（如设置 period=200；offset=0），同理设置 name B（如 120，,60），保存）。然后【start simulation】，出 name C 的输出图。

5、时序仿真或功能仿真。

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

四、实验过程

一、8 重 3-1 多路复用器

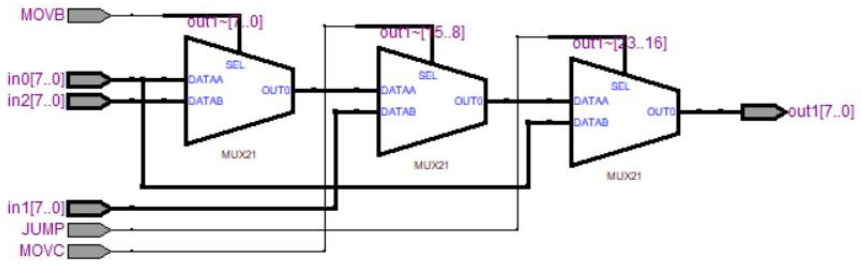
1、编译过程

a）源代码如图（VHDL 设计）

```
1   library IEEE;
2   use ieee.std_logic_1164.all;
3
4   entity fjt_8_3_1 is
5   port(
6       JUMP,MOVB,MOVC:in std_logic;
7       in0,in1,in2:in std_logic_vector(7 downto 0);
8       out1:out std_logic_vector(7 downto 0)
9   );
10  end fjt_8_3_1;
11
12  architecture struct of fjt_8_3_1 is
13  begin
14      process(in0,in1,in2,JUMP,MOVB,MOVC)
15      begin
16          if JUMP='1' then
17              out1<=in0;
18          elsif MOVC='1' then
19              out1<=in1;
20          elsif MOVB='1' then
21              out1<=in2;
22          else
23              out1<=in0;
24          end if;
25      end process;
26  end struct;
```

b)编译、调试过程

c) RTL 视图



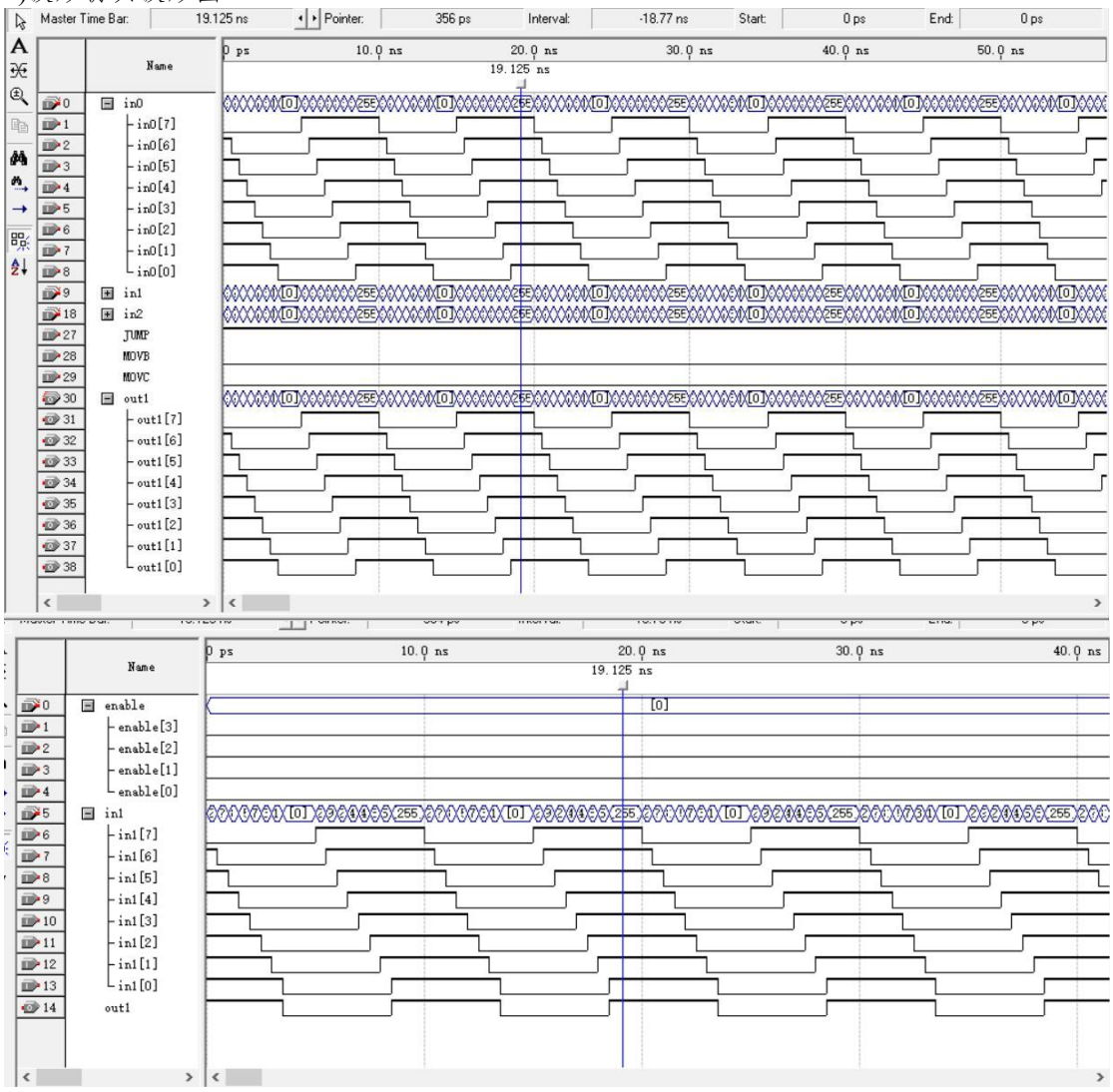
d)结果分析及结论

只要掌握了 3-1 多路复用器的原理，八重 3-1 多路复用器采用数据流描述相比结构描述能极大的简化代码长度。

2、 波形仿真

a)波形仿真过程（详见实验步骤）

b)波形仿真波形图



c)结果分析及结论

当输入为 00 时输出 i0

当输入为 01 时输出 i1

当输入为 10 时输出 i2

3、 时序仿真

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

Registered Performance | tpd | tsu | tco | th | Custom Delays

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	17.757 ns	in1[2]	out1[2]
2	N/A	None	17.010 ns	in2[7]	out1[7]
3	N/A	None	16.816 ns	MOVC	out1[7]
4	N/A	None	16.662 ns	JUMP	out1[2]
5	N/A	None	16.642 ns	MOVC	out1[2]
6	N/A	None	16.521 ns	JUMP	out1[7]
7	N/A	None	16.439 ns	in1[7]	out1[7]
8	N/A	None	15.712 ns	MOVB	out1[2]
9	N/A	None	15.708 ns	in2[2]	out1[2]
10	N/A	None	15.571 ns	MOVB	out1[7]
11	N/A	None	14.896 ns	JUMP	out1[6]
12	N/A	None	14.876 ns	MOVC	out1[6]
13	N/A	None	14.820 ns	in0[2]	out1[2]
14	N/A	None	14.429 ns	in2[0]	out1[0]
15	N/A	None	13.946 ns	MOVB	out1[6]
16	N/A	None	13.837 ns	in0[7]	out1[7]
17	N/A	None	13.397 ns	JUMP	out1[4]
18	N/A	None	13.377 ns	MOVC	out1[4]
19	N/A	None	13.271 ns	in2[6]	out1[6]
20	N/A	None	13.146 ns	JUMP	out1[0]
21	N/A	None	13.126 ns	MOVC	out1[0]
22	N/A	None	12.766 ns	JUMP	out1[3]
23	N/A	None	12.763 ns	JUMP	out1[5]
24	N/A	None	12.751 ns	JUMP	out1[1]

b) 结果分析及结论

当输入为 00 时输出 i0

当输入为 01 时输出 i1

当输入为 10 时输出 i2

二、参数化多路复用器定制为 8-1 多路复用器

4、编译过程

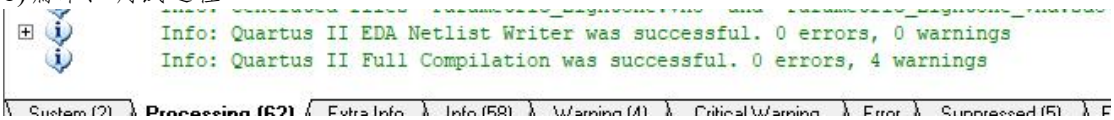
a）源代码如图（VHDL 设计）

fjt_canshuhua.vhd

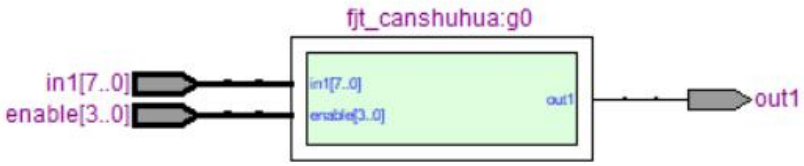
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity fjt_canshuhua is
7  generic(n:integer:=4;
8          m:integer:=2);
9  port(
10         in1:in std_logic_vector(n-1 downto 0);
11         enable:in std_logic_vector(m-1 downto 0);
12         out1:out std_logic
13 );
14 end fjt_canshuhua;
15
16 architecture struct of fjt_canshuhua is
17 begin
18     process(in1,enable)
19     begin
20     out1<=in1(conv_integer(enable));
21     end process;
22 end struct;
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith.all;
5
6  entity fjt_8_1 is
7  port(
8      in1:in std_logic_vector(7 downto 0);
9      enable:in std_logic_vector(3 downto 0);
10     out1:out std_logic);
11 end fjt_8_1;
12
13 architecture struct of fjt_8_1 is
14 component fjt_canshuhua
15     generic(n:integer:=4;
16             m:integer:=2);
17     port(
18             in1:in std_logic_vector(n-1 downto 0);
19             enable:in std_logic_vector(m-1 downto 0);
20             out1:out std_logic
21     );
22 end component;
23
24 begin
25     g0:fjt_canshuhua generic map(8,4) port map(in1,enable,out1);
26 end struct;
```

b)编译、调试过程



c) RTL 视图



d)结果分析及结论

学习了如何使用参数化方法设计多路复用器

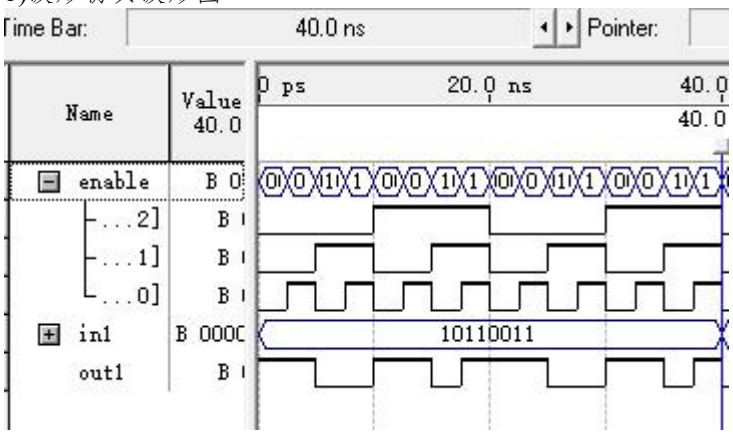
学会了 generic 函数设定参数

学会了使用 conv_integer 函数实现单一变量的多路复用器输出

5、 波形仿真

a)波形仿真过程（详见实验步骤）

b)波形仿真波形图



c)结果分析及结论

当输入为 000 时，输出右边第一位

当输入为 001 时，输出右边第二位

当输入为 010 时，输出右边第三位

当输入为 011 时，输出右边第四位

当输入为 100 时，输出右边第五位

当输入为 101 时，输出右边第六位

当输入为 110 时，输出右边第七位

当输入为 111 时，输出右边第八位

6、 时序仿真

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）

b)时序仿真图

Registered Performance | tpd | tsu | tco | th | Custom Delays

	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	12.946 ns	enable[0]	out1
2	N/A	None	12.252 ns	in1[0]	out1
3	N/A	None	10.129 ns	in1[1]	out1

d) 结果分析及结论

三、模型机的控制信号产生逻辑

7、编译过程

a）源代码如图（VHDL 设计）

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

entity fjt_kongzhixinhao is
port(
    IR:in std_logic_vector(7 downto 0);
    MOVA,MOVB,MOVC,ALU,NOT0,SHR,SHL,JMP,JZ,Z,JC,C,NOP,HALT,SM:in std_logic;
    SME,LD_IR,DL,XL,F,FRL,FRR,CF,ZF,M,LD_PC,IN_PC,N_WE,N_CS,CFE,ZFE:out std_logic;
    MADD,RAA,RWBA:out std_logic_vector(1 downto 0):="00";
    S:out std_logic_vector(3 downto 0)
    );
end fjt_kongzhixinhao;

architecture struct of fjt_kongzhixinhao is
signal command:std_logic_vector(3 downto 0);
begin
    command<=IR(7 downto 4);
    S(3)<=IR(7);
    S(2)<=IR(6);
    S(1)<=IR(5);
    S(0)<=IR(4);
    process(MOVA,MOVB,MOVC,ALU,NOT0,SHR,SHL,JMP,JZ,Z,JC,C,NOP,HALT,SM,IR)
        begin
        if SM='0' then
            LD_IR<='1';
            DL<='1';
            XL<='0';
            N_CS<='0';
                MADD<="00";
                LD_PC<='0';
                IN_PC<='1';
                N_WE<='1';
        elsif SM='1' then
            if MOVA='1' then
                RAA<=IR(1 downto 0);
                RWBA<=IR(3 downto 2);
                MADD<="00";
                LD_PC<='0';
                IN_PC<='0';
                N_WE<='0';
                XL<='0';
                DL<='1';
                M<='1';
                N_CS<='0';
                F<='1';
                FRR<='0';
                FRL<='0';
                LD_IR<='0';
                CFE<='0';
                ZFE<='0';
                SME<='1';
            elsif MOVB='1' then
                RAA<=IR(1 downto 0);
                RWBA<="11";
                MADD<="10";
                LD_PC<='0';
                IN_PC<='0';
                N_WE<='1';
```

```
61                  XL<='1';
62                  DL<='0';
63                  M<='1';
64                  N_CS<='0';
65                  F<='1';
66                  FRR<='0';
67                  FRL<='0';
68                  LD_IR<='0';
69                  CFE<='0';
70                  ZFE<='0';
71                  SME<='1';
72              elsif MOVC='1' then
73                  RAA<="11";
74                  RWBA<=IR(3 downto 2);
75                  MADD<="01";
76                  LD_PC<='0';
77                  IN_PC<='0';
78                  N_WE<='0';
79                  XL<='0';
80                  DL<='1';
81                  M<='1';
82                  N_CS<='0';
83                  F<='1';
84                  FRR<='0';
85                  FRL<='0';
86                  LD_IR<='0';
87                  CFE<='0';
88                  ZFE<='0';
89                  SME<='1';
90              elsif ALU='1' then
91                   RAA<=IR(1 downto 0);
92                   RWBA<=IR(3 downto 2);
93                   MADD<="00";
94                   LD_PC<='0';
95                   IN_PC<='0';
96                   N_WE<='0';
97                   XL<='0';
98                   DL<='0';
99                   if command="1011" then
100                      M<='0';
101                  else
102                      M<='1';
103                  end if;
104                  N_CS<='1';
105                  F<='1';
106                  FRR<='0';
107                  FRL<='0';
108                  LD_IR<='0';
109                  if command="1001" then
110                      CFE<='1';
111                      ZFE<='0';
112                  elsif command="0110" then
113                      CFE<='0';
114                      ZFE<='1';
115                  else
116                      ZFE<='0';
117                      CFE<='0';
118                  end if;
119                  SME<='1';
120              elsif NOT0='1' then
```

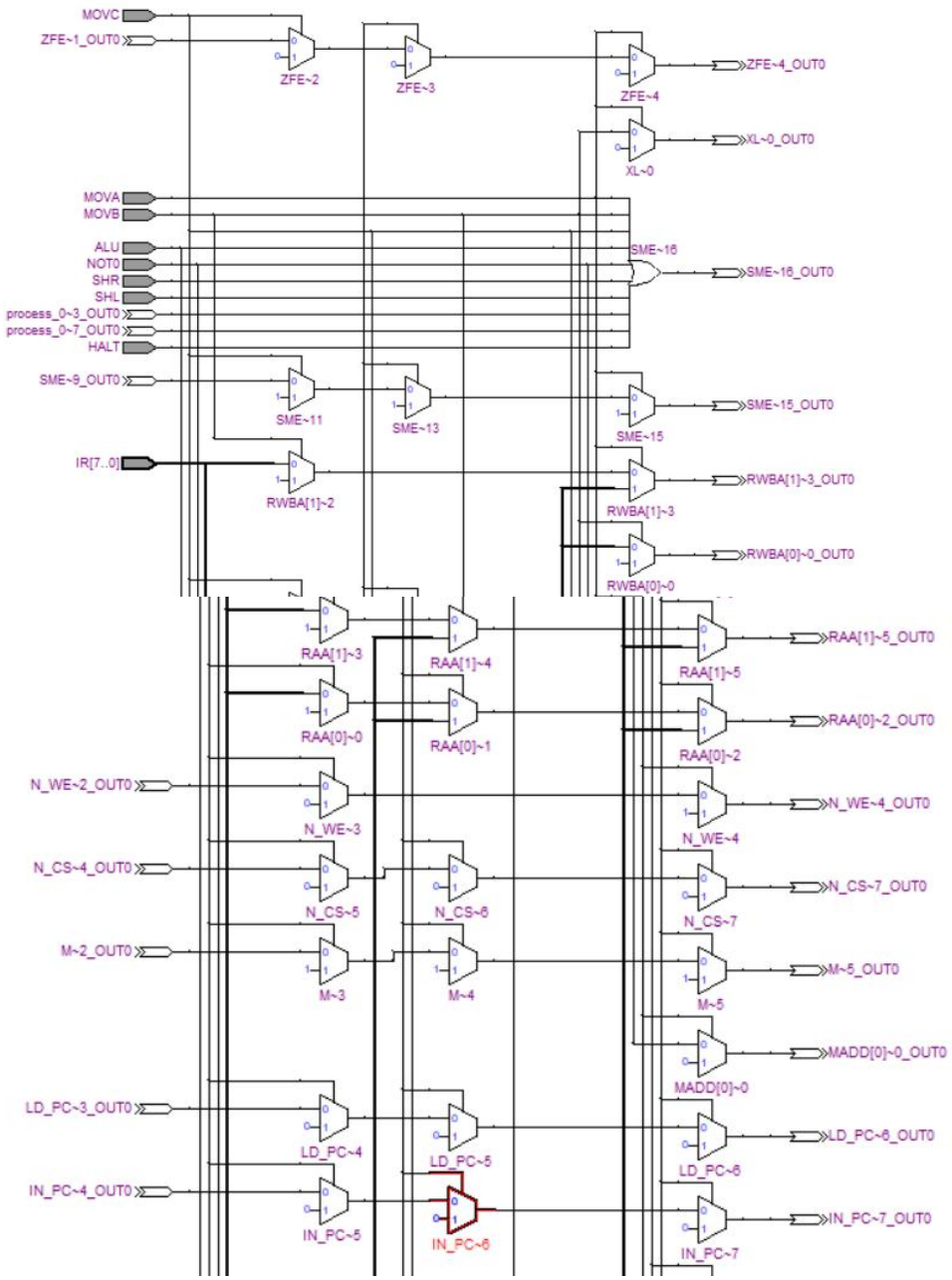
```
121                 RAA<=IR(1 downto 0);
122                 RWBA<=IR(3 downto 2);
123                 MADD<="00";
124                 LD_PC<='0';
125                 IN_PC<='0';
126                 N_WE<='0';
127                 XL<='0';
128                 DL<='0';
129                 M<='1';
130                 N_CS<='1';
131                 F<='1';
132                 FRR<='0';
133                 FRL<='0';
134                 LD_IR<='0';
135                 CFE<='0';
136                 ZFE<='0';
137                 SME<='1';
138             elsif SHR='1' then
139                 RAA<=IR(1 downto 0);
140                 RWBA<=IR(3 downto 2);
141                 MADD<="00";
142                 LD_PC<='0';
143                 IN_PC<='0';
144                 N_WE<='0';
145                 XL<='0';
146                 DL<='0';
147                 M<='1';
148                 N_CS<='1';
149                 F<='0';
150                 FRR<='1';
151                 FRL<='0';
152                 LD_IR<='0';
153                 CFE<='0';
154                 ZFE<='0';
155                 SME<='1';
156             elsif SHL='1' then
157                 RAA<=IR(1 downto 0);
158                 RWBA<=IR(3 downto 2);
159                 MADD<="00";
160                 LD_PC<='0';
161                 IN_PC<='0';
162                 N_WE<='0';
163                 XL<='0';
164                 DL<='0';
165                 M<='1';
166                 N_CS<='1';
167                 F<='0';
168                 FRR<='0';
169                 FRL<='1';
170                 LD_IR<='0';
171                 CFE<='0';
172                 ZFE<='0';
173                 SME<='1';
174             elsif JMP='1' or (JZ='1' and Z='1') or (JC='1'and C='1') then
175                 RAA<=IR(1 downto 0);
176                 RWBA<=IR(3 downto 2);
177                 MADD<="00";
178                 LD_PC<='1';
179                 IN_PC<='0';
180                 N_WE<='1';
```

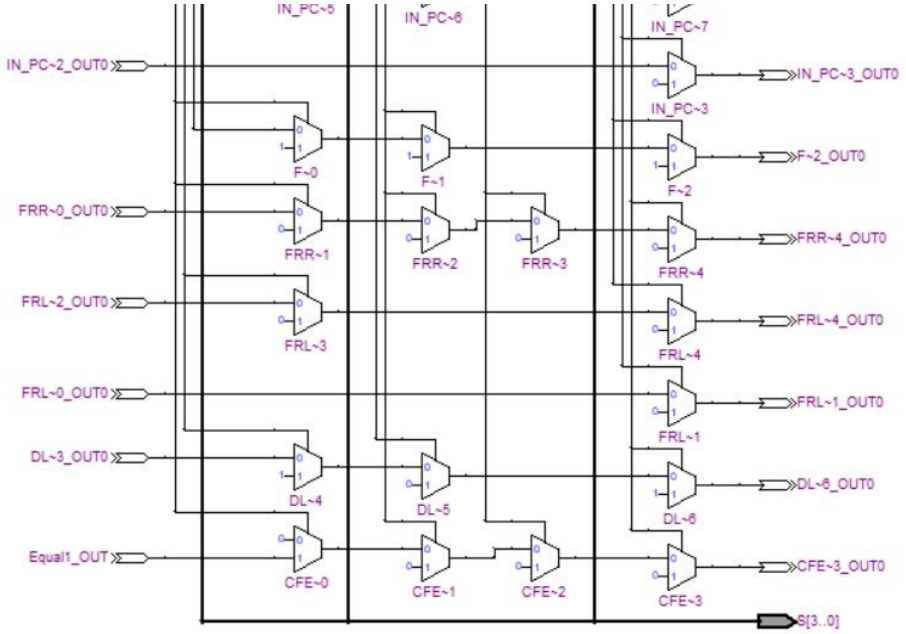
```
181                 XL<='0';
182                 DL<='1';
183                 M<='0';
184                 N_CS<='0';
185                 F<='0';
186                 FRR<='0';
187                 FRL<='0';
188                 LD_IR<='0';
189                 CFE<='0';
190                 ZFE<='0';
191                 SME<='1';
192             elsif NOP='1' or (JZ='1' and Z='0') or (JC='1'and C='0') then
193                 RAA<=IR(1 downto 0);
194                 RWBA<=IR(3 downto 2);
195                 MADD<="00";
196                 LD_PC<='1';
197                 IN_PC<='1';
198                 N_WE<='1';
199                 XL<='0';
200                 DL<='0';
201                 M<='0';
202                 N_CS<='0';
203                 F<='0';
204                 FRR<='0';
205                 FRL<='0';
206                 LD_IR<='0';
207                 CFE<='0';
208                 ZFE<='0';
209                 SME<='1';
210             elsif HALT='1' then
211                     RAA<=IR(1 downto 0);
212                     RWBA<=IR(3 downto 2);
213                     MADD<="00";
214                     LD_PC<='0';
215                     IN_PC<='0';
216                     N_WE<='1';
217                     XL<='0';
218                     DL<='0';
219                     M<='0';
220                     N_CS<='1';
221                     F<='0';
222                     FRR<='0';
223                     FRL<='0';
224                     LD_IR<='0';
225                     CFE<='0';
226                     ZFE<='0';
227                     SME<='0';
228                 end if;
229             end if;
230         end process;
231     end struct;
```

b)编译、调试过程

```
Info: Quartus II EDA Netlist Writer was successful. 0 errors, 0 warnings
Info: Quartus II Full Compilation was successful. 0 errors, 87 warnings
```

c) RTL 视图


MOVC
ZFE~1_OUT0
ZFE~2
ZFE~3
ZFE~4
ZFE~4_OUT0
XL~0
XL~0_OUT0
MOVA
MOVB
ALU
NOT0
SHR
SHL
process_0~3_OUT0
process_0~7_OUT0
HALT
SME~16
SME~16_OUT0
SME~9_OUT0
SME~11
SME~13
SME~15
SME~15_OUT0
IR[7..0]
RWBA[1]~2
RWBA[1]~3
RWBA[1]~3_OUT0
RWBA[0]~0
RWBA[0]~0_OUT0
RAA[1]~3
RAA[1]~4
RAA[1]~5
RAA[1]~5_OUT0
RAA[0]~0
RAA[0]~1
RAA[0]~2
RAA[0]~2_OUT0
N_WE~2_OUT0
N_WE~3
N_WE~4
N_WE~4_OUT0
N_CS~4_OUT0
N_CS~5
N_CS~6
N_CS~7
N_CS~7_OUT0
M~2_OUT0
M~3
M~4
M~5
M~5_OUT0
MADD[0]~0
MADD[0]~0_OUT0
LD_PC~3_OUT0
LD_PC~4
LD_PC~5
LD_PC~6
LD_PC~6_OUT0
IN_PC~4_OUT0
IN_PC~5
IN_PC~6
IN_PC~7
IN_PC~7_OUT0



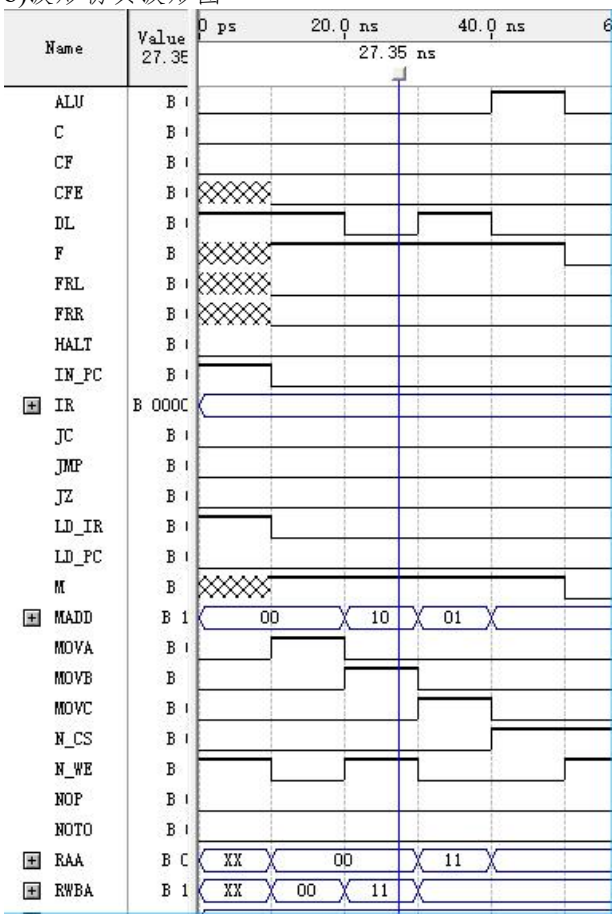
d)结果分析及结论

首先完成取指操作，并根据指令完成后续操作

8、 波形仿真

a)波形仿真过程（详见实验步骤）

b)波形仿真波形图



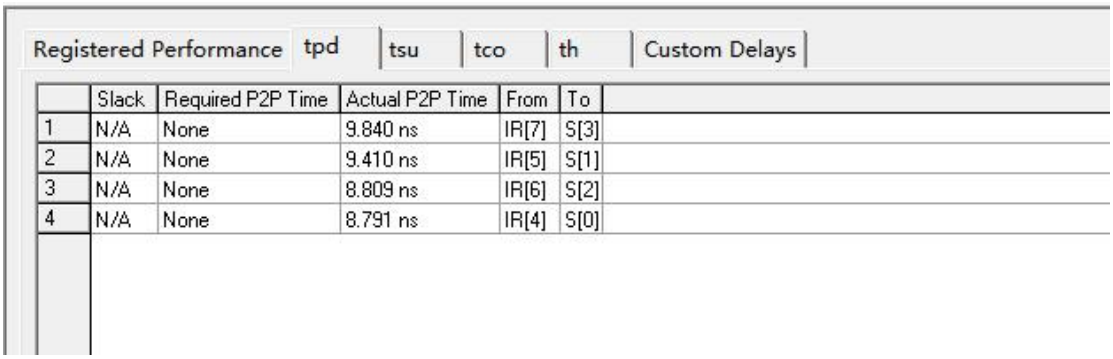
c)结果分析及结论

9、 时序仿真

e) 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	9.840 ns	IR[7]	S[3]
2	N/A	None	9.410 ns	IR[5]	S[1]
3	N/A	None	8.809 ns	IR[6]	S[2]
4	N/A	None	8.791 ns	IR[4]	S[0]

f) 结果分析及结论

tpd (引脚到引脚的延时)

五、实验结论

多路复用器的原理简单来说就是首先选择作用，输入 00，选择第 0 号输入位的数据，01 选择 1 号输入位的数据一次类推，至于几重多路复用器则代表可选数据的长度，例如一个 8 位二进制数则需要一个八重多路复用器才能实现选择。常用的函数 conv_integer 可实现二进制转十进制从而输出 vector 中内容实现多路复用器。

要掌握了 3-1 多路复用器的原理，八重 3-1 多路复用器采用数据流描述相比结构描述能极大的简化代码长度。

学习了如何使用参数化方法设计多路复用器。

学会了 generic 函数设定参数。

学会了使用 conv_integer 函数实现单一变量的多路复用器输出。