实验一 Quartus II 软件的基本操作、译码器

班级 计科 1903 姓名 陈旭 学号 201914020128

一、实验目的

1. 熟悉译码器的工作原理。
2. 熟悉 Quartus II 软件的基本操作，了解各种设计方法（原理图涉及，文本设计、波形设计）。

二、实验内容

使用原理图和 VHDL 语言两种方式，完成以下内容，并验证其功能的正确性。

1. 用逻辑图和 VHDL 语言设计一个异或门。
2. 用逻辑图和 VHDL 语言设计一个 3-8 译码器。
3. 用 VHDL 语言设计模型机指令译码器。

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实现异或门和 3-8 译码器。
采用的软件工具是 Quartus II。

2、实验软件操作步骤

- **异或门**

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\experiment\twoin_xor 设置 project name 为 twoin_xor）-【next】（设置文件名 twoin_xor）-【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（twoin_xor.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 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：enable））-右击-【value】-【count】（如设置 binary；startvalue=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】。

- **3-8 译码器**

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\experiment1\three_eightdecoder 设置 project name 为 three_eightdecoder）-【next】（设置文件名 three_eightdecoder）-【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（three_eightdecoder.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。按照程序所述插入节点 enable 和两个节点向量 inputs，x。（enable 和 inputs 为输入节点（向量），x 为输出节点向量）。(操作为：右击 -【insert】-【insert node or

bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】)。任意设置 inputs 和 enable 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；startvalue=000; endvalue=111;count value=2.0ns)，同理设置 name b（如 000, 1, 10），保存)。然后【start simulation】，出 name C 的输出图。

5. 功能仿真判断代码的实现无误后时序仿真。
6. 查看 RTL Viewer：【Tools】-【netlist viewer】-【RTL viewer】。

- **指令译码器**

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\experiment1\cmddecoder 设置 project name 为 cmddecoder）-【next】(设置文件名 cmddecoder）-【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（cmddecoder.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。按照程序所述插入节点 mov1, mov2, mov3, add, sub, or1, not1, rsr, rsl, jmp, jz, jc, in1, out1, nop, halt enable cmdar_code。（enable cmdar_code 为输入节点（向量），mov1, mov2, mov3, add, sub, or1, not1, rsr, rsl, jmp, jz, jc, in1, out1, nop, halt 为输出节点向量)。(操作为：右击 -【insert】-【insert node or bus】-【node finder】(pins=all;【list】)-【>>】-【ok】-【ok】)。按顺序设置 cmdar_code 和 enable 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；startvalue=000; endvalue=111;count value=2.0ns)，同理设置 name b（如 00000000, 1, 10)，保存)。然后【start simulation】，出 name C 的输出图。
5. 功能仿真判断代码的实现无误后时序仿真。
6. 查看 RTL Viewer：【Tools】-【netlist viewer】-【RTL viewer】

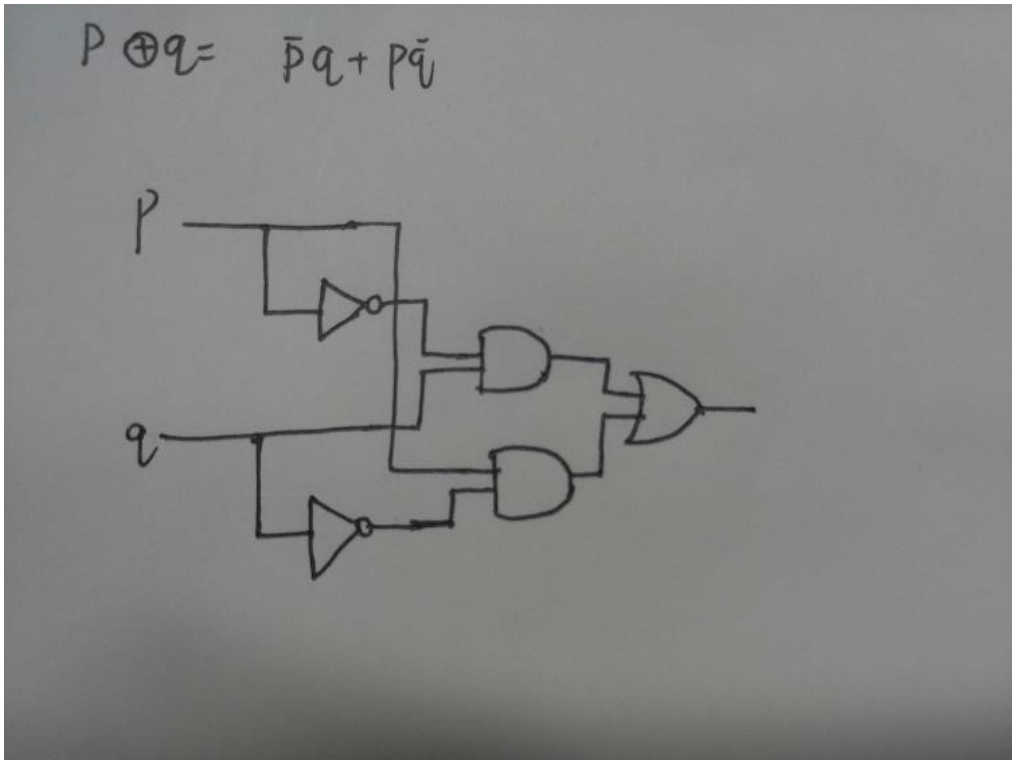
四、实验过程

1、编译过程

a）源代码（VHDL 设计）和原理图如图

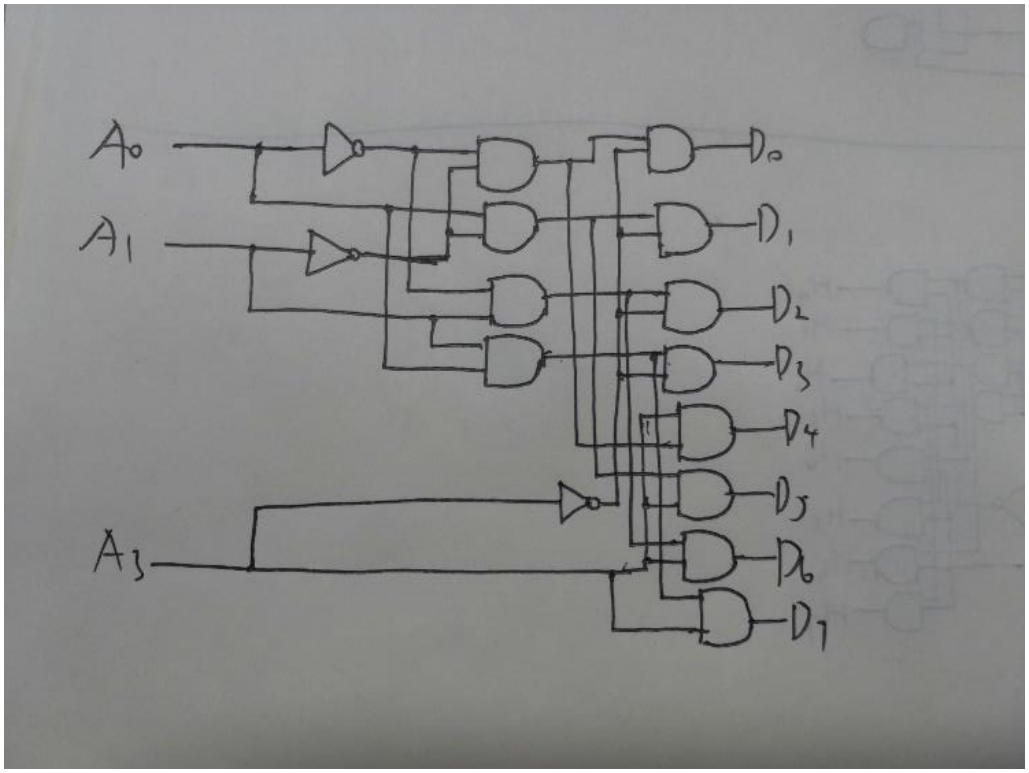
- 异或门

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

entity twoin_xor is
    port(
        a,b: in std_logic;
        c: out std_logic
    );
end entity twoin_xor;

architecture twoin_xor of twoin_xor is
    begin
        c <= a xor b;
end architecture twoin_xor;
```



- **3-8 译码器**

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

entity three_eightdecoder is
    port(
        inputs: in std_logic_vector(2 downto 0);
        enable: in std_logic;
        x: out std_logic_vector(7 downto 0)
    );
end entity three_eightdecoder;

architecture three_eightdecoder of three_eightdecoder is
    signal temps: std_logic_vector (2 downto 0);
    begin
        temps(0) <= not inputs(0);
        temps(1) <= not inputs(1);
        temps(2) <= not inputs(2);
        x(7) <= inputs(2) and inputs(1) and inputs(0) and enable;
        x(6) <= inputs(2) and inputs(1) and temps(0) and enable;
        x(5) <= inputs(2) and temps(1) and inputs(0) and enable;
        x(4) <= inputs(2) and temps(1) and temps(0) and enable;
        x(3) <= temps(2) and inputs(1) and inputs(0) and enable;
        x(2) <= temps(2) and inputs(1) and temps(0) and enable;
        x(1) <= temps(2) and temps(1) and inputs(0) and enable;
        x(0) <= temps(2) and temps(1) and temps(0) and enable;
end architecture three_eightdecoder;
```

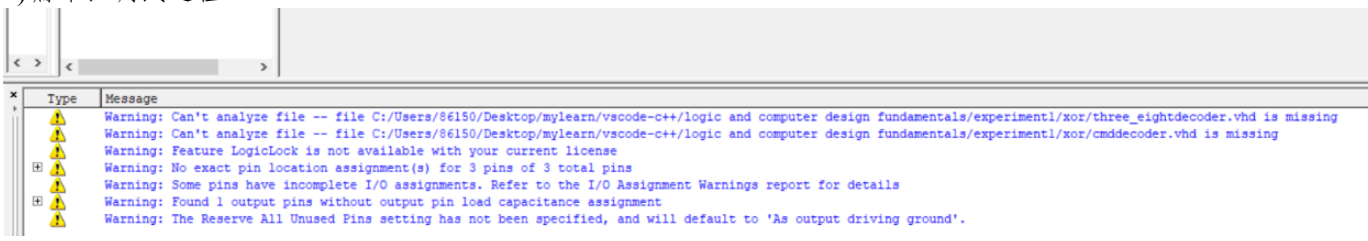


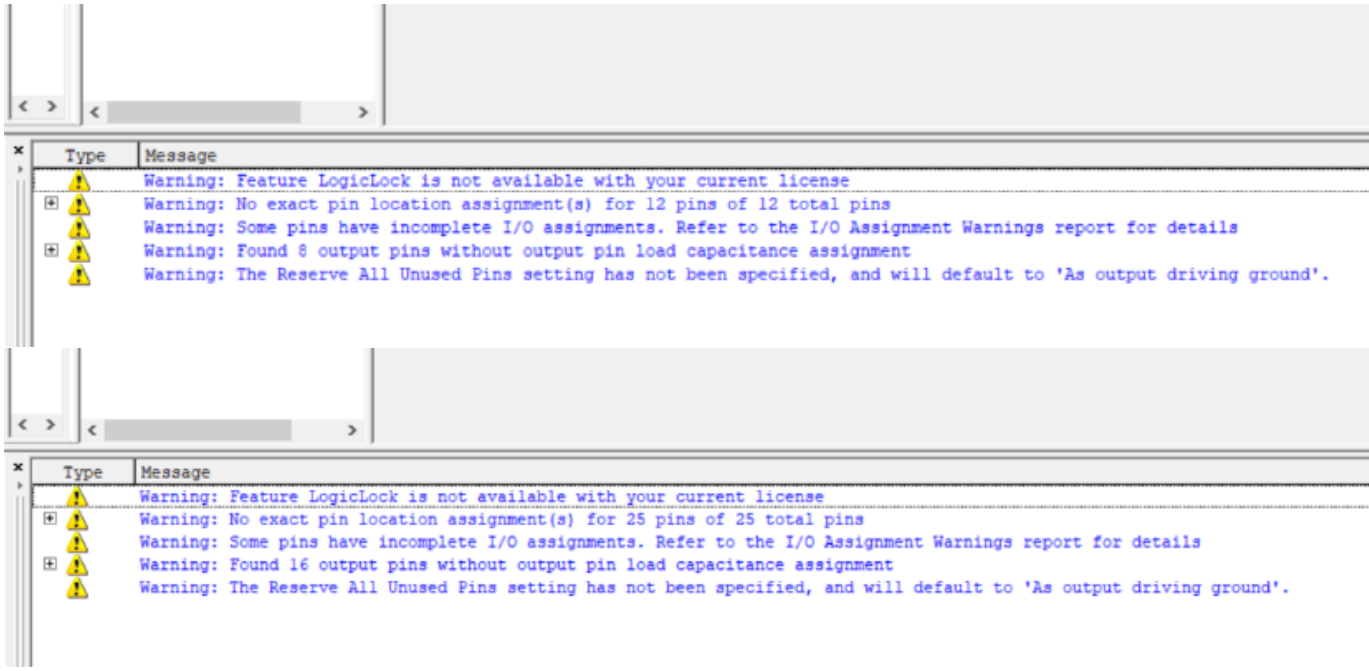
- 指令译码器

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

entity cmddecoder is
    port(
        cmdar_code: in std_logic_vector(7 downto 0);
        enable: in std_logic;
        mov1, mov2, mov3, add, sub, or1, not1, rsr, rsl, jmp, jz, jc, in1, out1, nop, h
alt: out std_logic
    );
end entity cmddecoder;

architecture cmddecoder of cmddecoder is
    signal cmdcode, tempcmd: std_logic_vector(3 downto 0);
    signal r1code,r2code, tempr1, tempr2: std_logic_vector(1 downto 0);
    signal r1, r2: std_logic;
    begin
            r1 <= not (cmdar_code(4) and cmdar_code(5) and enable);
            r2 <= not (cmdar_code(6) and cmdar_code(7) and enable);
            cmdcode(0) <= cmdar_code(0);
            cmdcode(1) <= cmdar_code(1);
            cmdcode(2) <= cmdar_code(2);
            cmdcode(3) <= cmdar_code(3);
            r1code(0) <= cmdar_code(4);
            r1code(1) <= cmdar_code(5);
            r2code(0) <= cmdar_code(6);
            r2code(1) <= cmdar_code(7);
            tempcmd <= not cmdcode;
```

```
    tempr1 <= not r1code;
    tempr2 <= not r2code;
    mov1 <= cmdcode(0) and cmdcode(1) and cmdcode(2) and cmdcode(3) and tempr1(
        0) and r1 and r2 and enable;
    mov2 <= cmdcode(0) and cmdcode(1) and cmdcode(2) and cmdcode(3) and r1code(
        0) and r1code (1) and r2 and enable;
    mov3 <= cmdcode(0) and cmdcode(1) and cmdcode(2) and cmdcode(3) and r1 and
        r2code(0) and r2code(1) and enable;
    add <= cmdcode(0) and tempcmd(1) and tempcmd(2) and cmdcode(3) and r1 and r
        2 and enable;
    sub <= tempcmd(0) and cmdcode(1) and cmdcode(2) and tempcmd(3) and r1 and r
        2 and enable;
    or1 <= cmdcode(0) and tempcmd(1) and cmdcode(2) and cmdcode(3) and r1 and r
        2 and enable;
    not1 <= tempcmd(0) and cmdcode(1) and tempcmd(2) and cmdcode(3) and r1 and
        enable;
    rsr <= cmdcode(0) and tempcmd(1) and cmdcode(2) and tempcmd(3) and r1 and t
        empr2(0) and tempr2(1) and enable;
    rsl <= cmdcode(0) and tempcmd(1) and cmdcode(2) and tempcmd(3) and r1 and r
        2code(0) and r2code(1) and enable;
    jmp <= tempcmd(0) and tempcmd(1) and cmdcode(2) and cmdcode(3) and tempr1(0
        ) and tempr1 (1) and tempr2(0) and tempr2(1) and enable;
    jz <= tempcmd(0) and tempcmd(1) and cmdcode(2) and cmdcode(3) and tempr1(0)
         and tempr1 (1) and tempr2(0) and r2code(1) and enable;
    jc <= tempcmd(0) and tempcmd(1) and cmdcode(2) and cmdcode(3) and tempr1(0)
         and tempr1 (1) and r2code(0) and tempr2(1) and enable;
    in1 <= tempcmd(0) and tempcmd(1) and cmdcode(2) and tempcmd(3) and r1 and e
        nable;
    out1 <= tempcmd(0) and cmdcode(1) and tempcmd(2) and tempcmd(3) and r1 and
        enable;
    nop <= tempcmd(0) and cmdcode(1) and cmdcode(2) and cmdcode(3) and tempr1(0
        ) and tempr1(1) and tempr2(0) and tempr2(1) and enable;
    halt <= cmdcode(0) and tempcmd(1) and tempcmd(2) and tempcmd(3) and tempr1(
        0) and tempr1(1) and tempr2(0) and tempr2(1) and enable;
end architecture cmddecoder;
```

b)编译、调试过程

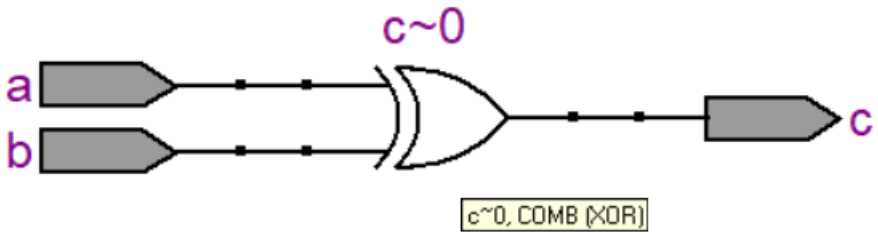




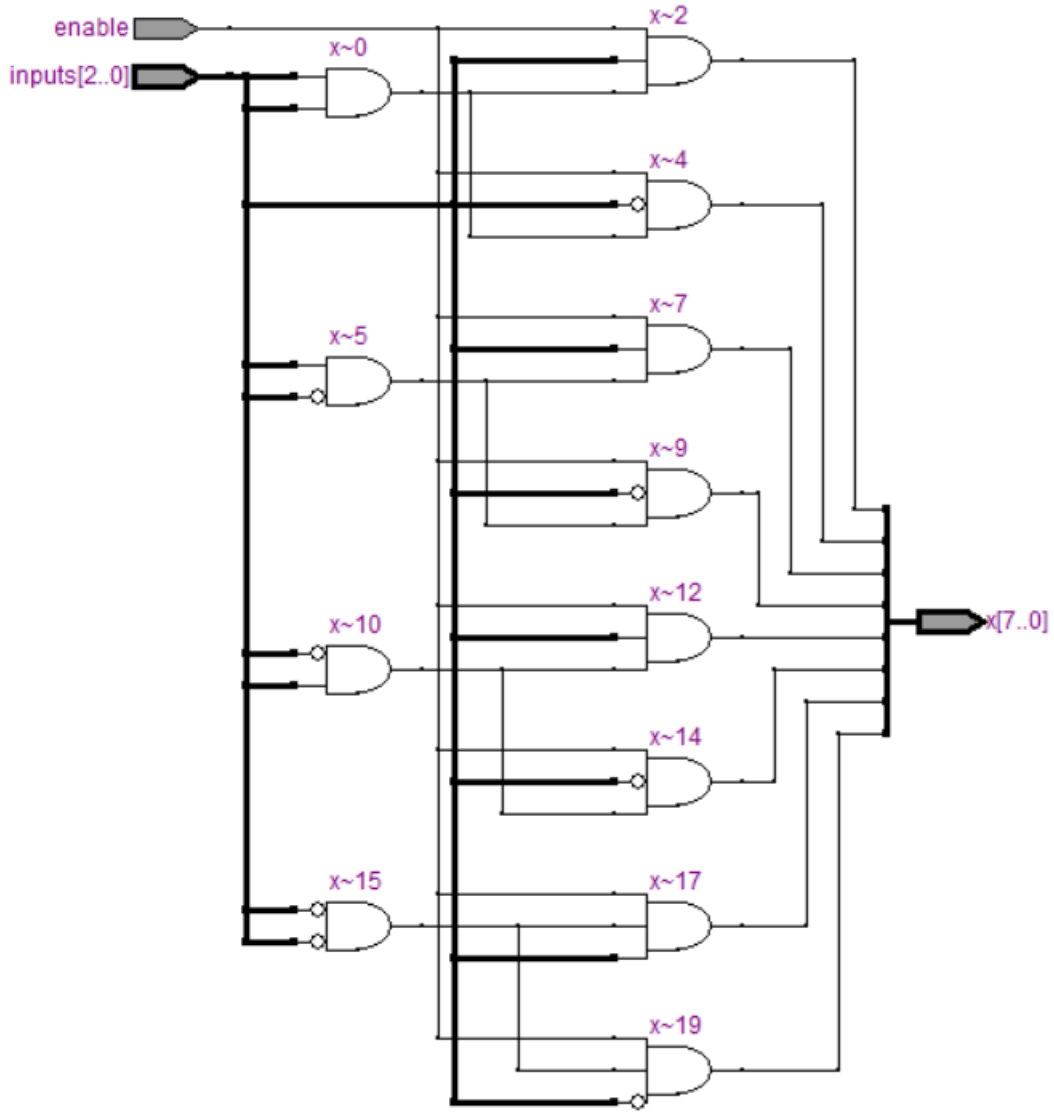
自上至下依次为异或门、3-8 译码器、指令译码器的编译器提示信息。
三者编译器均给出警告，但均能够编译通过。
三者资源消耗基本都趋近于 0.

c) RTL 视图

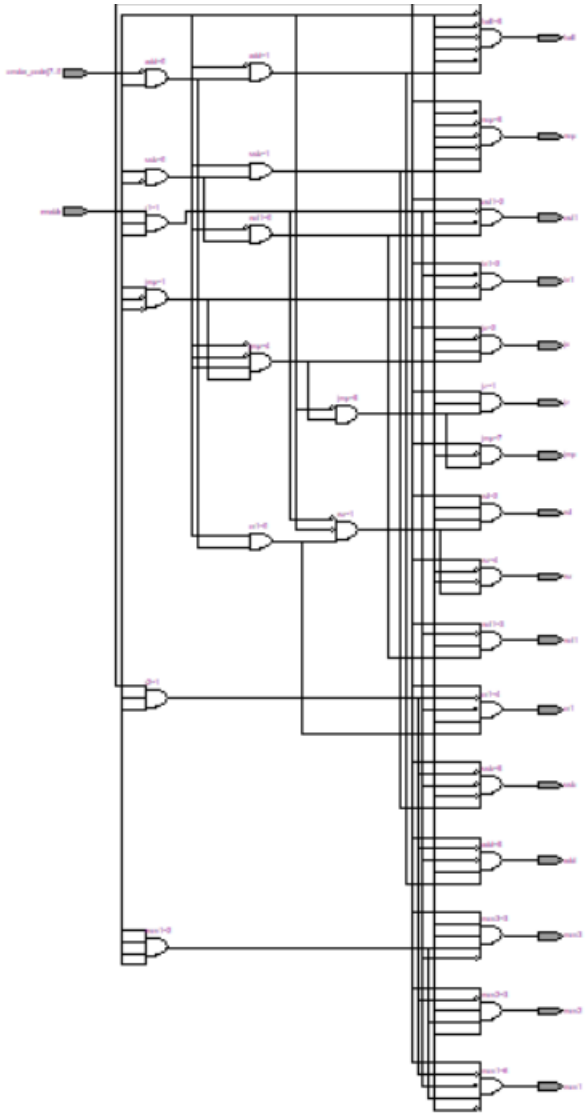
- 异或门



- 3-8 译码器



- 指令译码器



d)结果分析及结论

- 异或门

当输入a，b为 0 1时输出为 1
当输入a，b为 1 0时输出为 1
当输入a，b为 1 1时输出为 0
当输入a，b为 0 0时输出为 0

- 3-8 译码器

当使能 enable 为 0 时，输出恒为 0.
当使能 enable 为 1 时
当输入为 000 时，输出为 10000000
当输入为 001 时，输出为 01000000
当输入为 010 时，输出为 00100000
当输入为 011 时，输出为 00010000
当输入为 100 时，输出为 00001000
当输入为 101 时，输出为 00000100
当输入为 110 时，输出为 00000010
当输入为 111 时，输出为 00000001

- 指令译码器

当使能 enable 为 0 时，所有输出恒为 0.
当使能 enable 为 1 时，有：

当输入为 1111 R1 R2 mov1 输出为1，其他指令输出为 0
当输入为 1111 11 R2 mov2 输出为1 其他指令输出为 0
当输入为 1111 R1 11 mov3输出为1 其他指令输出为 0
当输入为 1001 R1 R2 add 输出为1 其他指令输出为 0

当输入为 0110 R1 R2 sub 输出为1 其他指令输出为 0
当输入为 1011 R1 R2 or1 输出为1 其他指令输出为 0
当输入为 0101 R1 XX not1 输出为1 其他指令输出为 0
当输入为 1010 R1 00 rsr 输出为1 其他指令输出为 0
当输入为 1010 R1 11 rsl 输出为1 其他指令输出为 0
当输入为 0011 00 00 jmp 输出为1 其他指令输出为 0

当输入为 0011 00 01 jz 输出为1 其他指令输出为 0
当输入为 0011 00 10 jc 输出为1 其他指令输出为 0

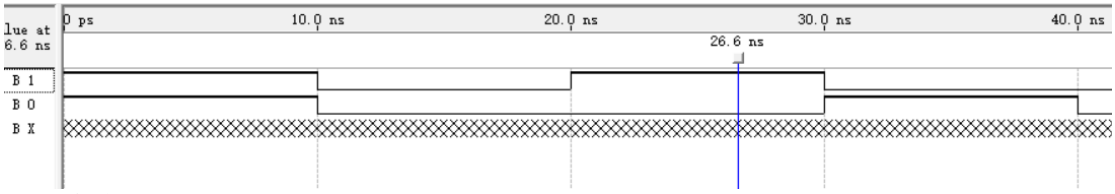
当输入为 0010 R1 XX in1 输出为1 其他指令输出为 0
当输入为 0100 R1 XX out1输出为1 其他指令输出为 0
当输入为 0111 00 00 nop 输出为1 其他指令输出为 0
当输入为 1000 00 00 halt输出为1 其他指令输出为 0

2、 波形仿真

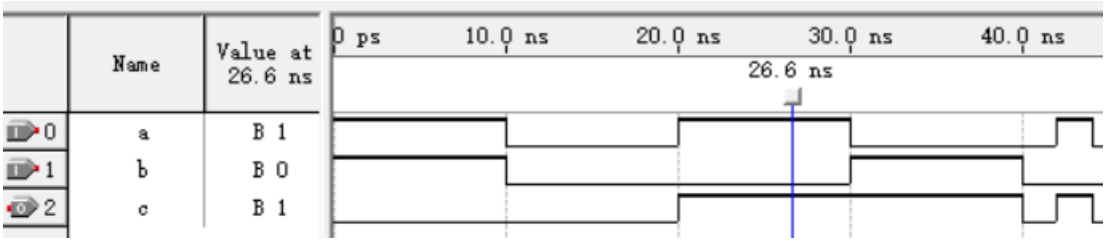
enable 调为 1 时

- 异或门

a)波形仿真过程（过程详见实验步骤）



b)波形仿真波形图

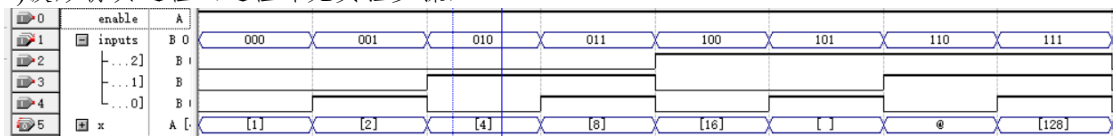


c)结果分析及结论

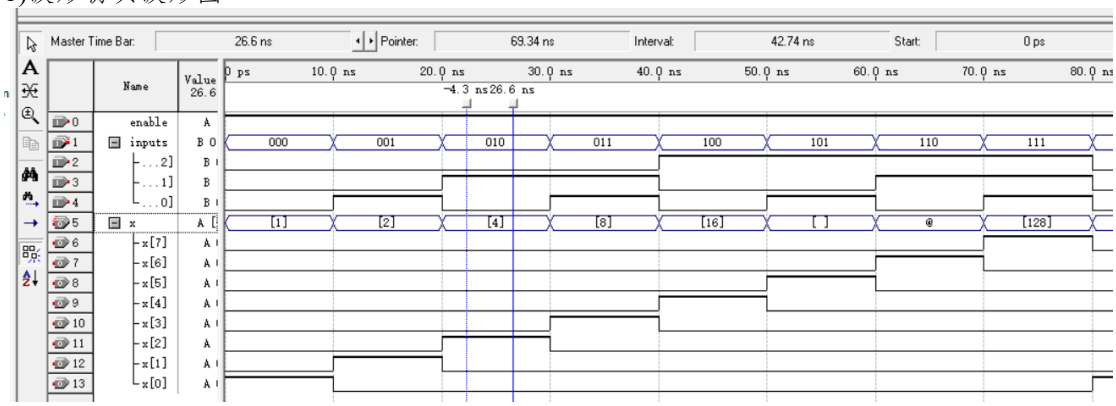
enable 调为 1，
0-10 ns 时 a,b 输入均为 1，c 输出为 0
10-20ns 时 a,b 输入均为 0，c 输出为 0
20-30ns 时 a,b 输入为 1 0，c 输出为 1
30-40ns 时 a,b 输入为 0 1，c 输出为 1
结果正确

- 3-8 译码器

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

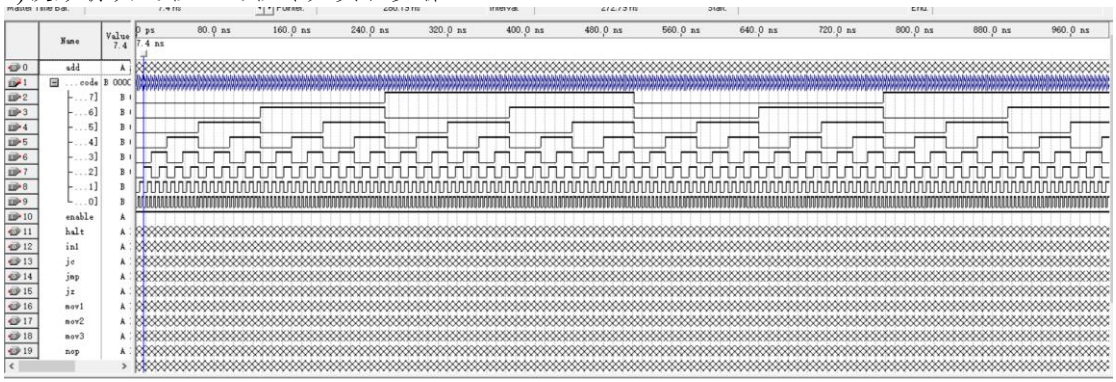
enable 调为 1，

0-80 ns 中，每隔 10ns 三位输入码变化一次，而相应的八位输出码也随之变化，呈现阶梯状，如 0-10ns 输入为 000，输出为 10000000；10-20ns 输入为 01000000……

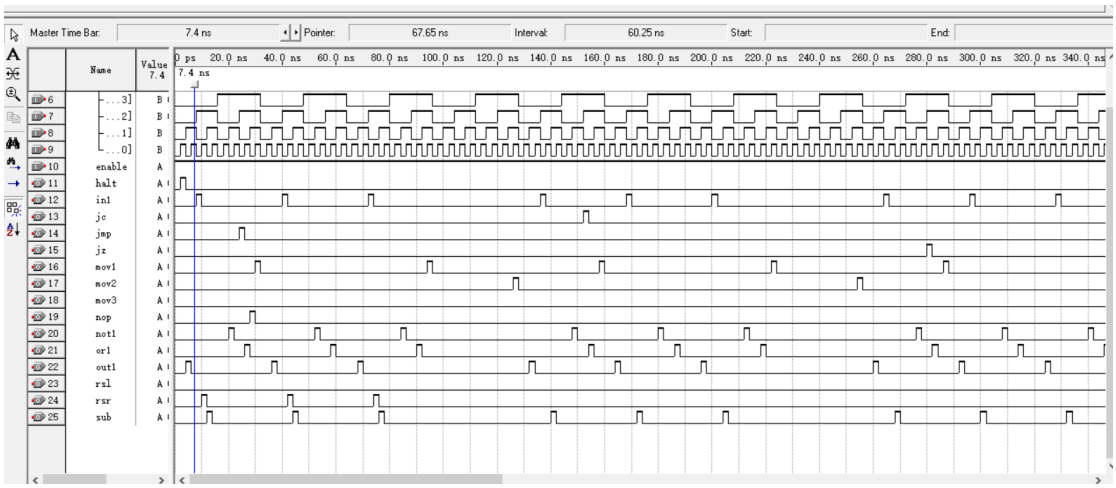
结果正确

- 指令译码器

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

enable调为1，当输入从00000000-11111111变化时，每隔2ns即变化一个周期时，如果产生对应的指令码则对应指令为1，其他为0。如图中，当输入为 0011 00 01时， jz 输出为1 其他指令输出为 0，以此类推其他仍如此，故结果正确

enable 调为 0 时

如下图知不论输入如何其指令译码器和 3-8 译码器输出均为 0

结果正确

3、 时序仿真(enable 为 1 时)

- 异或门

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	12.900 ns	A	C
2	N/A	None	12.400 ns	B	C

c) 结果分析及结论

A 引脚到 C 引脚的实际 p2p 时间为 12.900ns，二 B 引脚到 C 引脚的实际 p2p 时间为 12.400ns。A 比 B 慢 0.5ns，可由于结果是由时间长的那个决定，故整体为 12.900ns。

tpd (引脚到引脚的延时)

- 3-8 译码器

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	P2P Time	Time	From	To
1	N/A	None	10.789 ns	inputs[0]	x[4]
2	N/A	None	10.593 ns	inputs[0]	x[2]
3	N/A	None	10.524 ns	enable	x[4]
4	N/A	None	10.297 ns	enable	x[2]
5	N/A	None	10.246 ns	inputs[0]	x[7]
6	N/A	None	10.235 ns	inputs[0]	x[6]
7	N/A	None	10.001 ns	inputs[0]	x[0]
8	N/A	None	9.970 ns	enable	x[6]
9	N/A	None	9.967 ns	inputs[0]	x[5]
10	N/A	None	9.950 ns	enable	x[7]
11	N/A	None	9.892 ns	inputs[1]	x[4]
12	N/A	None	9.735 ns	enable	x[0]
13	N/A	None	9.671 ns	enable	x[5]
14	N/A	None	9.571 ns	inputs[1]	x[2]
15	N/A	None	9.554 ns	inputs[2]	x[4]
16	N/A	None	9.362 ns	inputs[2]	x[2]
17	N/A	None	9.347 ns	inputs[0]	x[1]
18	N/A	None	9.346 ns	inputs[1]	x[7]
19	N/A	None	9.335 ns	inputs[1]	x[6]
20	N/A	None	9.106 ns	inputs[1]	x[0]
21	N/A	None	9.070 ns	inputs[1]	x[5]
22	N/A	None	9.050 ns	enable	x[1]
23	N/A	None	9.012 ns	inputs[2]	x[7]
24	N/A	None	9.001 ns	inputs[2]	x[6]
25	N/A	None	8.769 ns	inputs[2]	x[0]
26	N/A	None	8.732 ns	inputs[2]	x[5]
27	N/A	None	8.490 ns	inputs[0]	x[3]
28	N/A	None	8.452 ns	inputs[1]	x[1]
29	N/A	None	8.193 ns	enable	x[3]
30	N/A	None	8.116 ns	inputs[2]	x[1]
31	N/A	None	7.596 ns	inputs[1]	x[3]
32	N/A	None	7.259 ns	inputs[2]	x[3]

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 inputs[0]传递给 x[4]。为 10.789ns，故整体延时为 10.789ns。

tpd (引脚到引脚的延时)

- 指令译码器

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	10.181 ns	cmdar_code[6]	mov1
2	N/A	None	9.942 ns	cmdar_code[7]	mov1
3	N/A	None	9.866 ns	cmdar_code[4]	not1
4	N/A	None	9.814 ns	cmdar_code[0]	mov1
5	N/A	None	9.735 ns	cmdar_code[3]	mov1
6	N/A	None	9.665 ns	cmdar_code[4]	rsr
7	N/A	None	9.613 ns	cmdar_code[5]	not1
8	N/A	None	9.515 ns	cmdar_code[6]	add
9	N/A	None	9.394 ns	cmdar_code[5]	rsr
10	N/A	None	9.354 ns	cmdar_code[2]	mov1
11	N/A	None	9.308 ns	cmdar_code[3]	rsr
12	N/A	None	9.280 ns	cmdar_code[6]	rsr
13	N/A	None	9.278 ns	cmdar_code[0]	rsr
14	N/A	None	9.276 ns	cmdar_code[7]	add
15	N/A	None	9.249 ns	cmdar_code[2]	rsr
16	N/A	None	9.195 ns	cmdar_code[0]	not1
17	N/A	None	9.189 ns	cmdar_code[1]	rsr
18	N/A	None	9.148 ns	cmdar_code[0]	add
19	N/A	None	9.141 ns	cmdar_code[4]	add
20	N/A	None	9.135 ns	cmdar_code[4]	jmp
21	N/A	None	9.135 ns	cmdar_code[2]	not1
22	N/A	None	9.117 ns	cmdar_code[2]	mov3
23	N/A	None	9.099 ns	cmdar_code[1]	not1
24	N/A	None	9.089 ns	cmdar_code[1]	mov1
25	N/A	None	9.081 ns	cmdar_code[1]	mov3
26	N/A	None	9.069 ns	cmdar_code[3]	add
27	N/A	None	9.049 ns	cmdar_code[4]	mov1
28	N/A	None	9.036 ns	cmdar_code[2]	jmp
29	N/A	None	9.020 ns	cmdar_code[3]	jmp
30	N/A	None	9.011 ns	cmdar_code[0]	mov3
31	N/A	None	8.995 ns	cmdar_code[4]	sub
32	N/A	None	8.916 ns	cmdar_code[4]	in1
33	N/A	None	8.903 ns	cmdar_code[6]	mov3
34	N/A	None	8.891 ns	cmdar_code[5]	jmp

tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
35	N/A	None	8.888 ns	cmdar_code[5]	add
36	N/A	None	8.866 ns	cmdar_code[7]	rsr
37	N/A	None	8.831 ns	cmdar_code[2]	add
38	N/A	None	8.805 ns	cmdar_code[0]	jmp
39	N/A	None	8.795 ns	cmdar_code[3]	not1
40	N/A	None	8.783 ns	cmdar_code[4]	mov3
41	N/A	None	8.778 ns	cmdar_code[3]	mov3
42	N/A	None	8.767 ns	cmdar_code[6]	mov2
43	N/A	None	8.750 ns	cmdar_code[1]	add
44	N/A	None	8.724 ns	cmdar_code[5]	sub
45	N/A	None	8.687 ns	cmdar_code[4]	jz
46	N/A	None	8.661 ns	cmdar_code[7]	mov3
47	N/A	None	8.645 ns	cmdar_code[5]	in1
48	N/A	None	8.638 ns	cmdar_code[3]	sub
49	N/A	None	8.588 ns	cmdar_code[2]	jz
50	N/A	None	8.587 ns	cmdar_code[4]	rsl
51	N/A	None	8.579 ns	cmdar_code[2]	sub
52	N/A	None	8.572 ns	cmdar_code[3]	jz
53	N/A	None	8.559 ns	cmdar_code[3]	in1
54	N/A	None	8.539 ns	cmdar_code[5]	mov3
55	N/A	None	8.528 ns	cmdar_code[7]	mov2
56	N/A	None	8.500 ns	cmdar_code[2]	in1
57	N/A	None	8.500 ns	cmdar_code[1]	jmp
58	N/A	None	8.476 ns	cmdar_code[6]	halt
59	N/A	None	8.474 ns	cmdar_code[0]	halt
60	N/A	None	8.464 ns	cmdar_code[0]	in1
61	N/A	None	8.445 ns	cmdar_code[6]	or1
62	N/A	None	8.443 ns	cmdar_code[5]	jz
63	N/A	None	8.441 ns	cmdar_code[4]	jc
64	N/A	None	8.400 ns	cmdar_code[0]	mov2
65	N/A	None	8.399 ns	cmdar_code[4]	out1
66	N/A	None	8.385 ns	cmdar_code[1]	halt
67	N/A	None	8.375 ns	cmdar_code[1]	in1
68	N/A	None	8.357 ns	cmdar_code[0]	jz

	Slack	Required P2P Time	Actual P2P Time	From	To
68	N/A	None	8.357 ns	cmdar_code[0]	jz
69	N/A	None	8.342 ns	cmdar_code[2]	jc
70	N/A	None	8.336 ns	cmdar_code[4]	nop
71	N/A	None	8.326 ns	cmdar_code[3]	jc
72	N/A	None	8.321 ns	cmdar_code[3]	mov2
73	N/A	None	8.316 ns	cmdar_code[5]	rsl
74	N/A	None	8.314 ns	cmdar_code[4]	or1
75	N/A	None	8.301 ns	cmdar_code[6]	rsl
76	N/A	None	8.289 ns	cmdar_code[6]	jmp
77	N/A	None	8.251 ns	cmdar_code[7]	jmp
78	N/A	None	8.237 ns	cmdar_code[2]	nop
79	N/A	None	8.236 ns	cmdar_code[4]	mov2
80	N/A	None	8.230 ns	cmdar_code[3]	rsl
81	N/A	None	8.221 ns	cmdar_code[3]	nop
82	N/A	None	8.206 ns	cmdar_code[7]	or1
83	N/A	None	8.197 ns	cmdar_code[5]	jc
84	N/A	None	8.171 ns	cmdar_code[2]	rsl
85	N/A	None	8.146 ns	cmdar_code[5]	out1
86	N/A	None	8.140 ns	cmdar_code[0]	rsl
87	N/A	None	8.111 ns	cmdar_code[0]	jc
88	N/A	None	8.097 ns	cmdar_code[7]	sub
89	N/A	None	8.092 ns	cmdar_code[5]	nop
90	N/A	None	8.078 ns	cmdar_code[0]	or1
91	N/A	None	8.062 ns	cmdar_code[7]	halt
92	N/A	None	8.062 ns	cmdar_code[7]	rsl
93	N/A	None	8.061 ns	cmdar_code[5]	or1
94	N/A	None	8.037 ns	cmdar_code[0]	sub
95	N/A	None	8.006 ns	cmdar_code[0]	nop
96	N/A	None	7.999 ns	cmdar_code[3]	or1
97	N/A	None	7.983 ns	cmdar_code[5]	mov2
98	N/A	None	7.956 ns	cmdar_code[6]	sub
99	N/A	None	7.940 ns	cmdar_code[2]	mov2
100	N/A	None	7.924 ns	cmdar_code[1]	jz
101	N/A	None	7.920 ns	cmdar_code[1]	out1

tpd

	Slack	Required P2P Time	Actual P2P Time	From	To
104	N/A	None	7.855 ns	cmdar_code[0]	out1
105	N/A	None	7.842 ns	cmdar_code[2]	out1
106	N/A	None	7.840 ns	cmdar_code[1]	sub
107	N/A	None	7.807 ns	cmdar_code[1]	jc
108	N/A	None	7.797 ns	cmdar_code[2]	or1
109	N/A	None	7.770 ns	cmdar_code[3]	halt
110	N/A	None	7.716 ns	cmdar_code[1]	or1
111	N/A	None	7.709 ns	cmdar_code[3]	out1
112	N/A	None	7.708 ns	cmdar_code[6]	jz
113	N/A	None	7.666 ns	cmdar_code[7]	jz
114	N/A	None	7.640 ns	cmdar_code[1]	mov2
115	N/A	None	7.629 ns	cmdar_code[5]	halt
116	N/A	None	7.590 ns	cmdar_code[6]	jc
117	N/A	None	7.583 ns	cmdar_code[7]	nop
118	N/A	None	7.550 ns	cmdar_code[7]	jc
119	N/A	None	7.500 ns	cmdar_code[2]	halt
120	N/A	None	7.440 ns	cmdar_code[6]	nop
121	N/A	None	7.394 ns	cmdar_code[1]	nop
122	N/A	None	6.813 ns	enable	mov1
123	N/A	None	6.683 ns	enable	not1
124	N/A	None	6.284 ns	enable	rsr
125	N/A	None	6.147 ns	enable	add
126	N/A	None	5.875 ns	enable	jmp
127	N/A	None	5.614 ns	enable	sub
128	N/A	None	5.535 ns	enable	in1
129	N/A	None	5.522 ns	enable	mov3
130	N/A	None	5.427 ns	enable	jz
131	N/A	None	5.399 ns	enable	mov2
132	N/A	None	5.216 ns	enable	out1
133	N/A	None	5.206 ns	enable	rsl
134	N/A	None	5.181 ns	enable	jc
135	N/A	None	5.077 ns	enable	or1
136	N/A	None	5.076 ns	enable	nop
137	N/A	None	4.611 ns	enable	halt

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 cmdar_code[6]传递给 mov1。为 10.181ns，故整体延时为 10.181ns。

tpd (引脚到引脚的延时)

五、实验结论

1、思考题

1) 首先，发光二极管数码管是用发光二极管构成显示数码的笔划来显示数字，由于发二极管会发光，故 LED 数码管适用于各种场合。其次，液晶显示数码管是利用液晶材料在交

变电压的作用下晶体材料会吸收光线，而没有交变电场作用下有笔划不会听吸光，这样就可以来显示数码。再者，译码器可以用于内存寻址，程序计数器中保存了 CPU 将要执行的指令，通过译码器可以将其发送给相应的执行指令的组成部分中。

2) VHDL 语言描述译码器电路常用的方法有条件语句判断、逻辑表达式赋值信号。

两者都要用到的语句有

库引入、实体声明、端口方向、结构体、库，程序包的调用、进程语句

前者常用的语句有

选择赋值语句（when-s-select 语句）

条件信号赋值语句（**赋值目标信号 <= 表达式 1 WHEN 赋值条件 1 ELSE**）

if 语句

后者常用的语句有

基本逻辑关系(and, or, not)等

3) 原理图实现更加直观，实现起来简单易上手，步骤和方法就是根据基础的逻辑关系画出对应的逻辑门电路，但是可用程度较低，仅仅起到直观的效果，而且可以被 VHDL 语句实现中的查看 RTL 视图功能所取代。

VHDL 代码实现的步骤和方法即为根据逻辑关系设计 VHDL 程序，然后进行波形仿真，查看 RTL 视图等。VHDL 实现起来较为困难，需要一定的代码知识，直接实现起来可能显得不那么直观，并且出错率可能较大。但是功能强大，在生产生活应用的大背景下更有实际价值。

2、实验总结与实验心得

本次实验学习了译码器的实现。需要的能力有 VHDL 的简单编程能力，VHDL 的简单仿真能力，逻辑表达式设计与化简能力，基础电路的设计绘制能力，实验报告的设计能力，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。该次实验使我对数字电路与逻辑设计有了更深刻的认识，巩固了译码器相关的知识点。我因此受益良多。