# 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。

1. 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】

1. 写好源代码，保存文件（twoin\_xor.vhd）。
2. 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。
3. 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。
4. 时序仿真和功能仿真。
5. 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】

1. 写好源代码，保存文件（three\_eightdecoder.vhd）。
2. 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。
3. 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。
4. 功能仿真判断代码的实现无误后时序仿真。
5. 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】

1. 写好源代码，保存文件（cmddecoder.vhd）。
2. 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。
3. 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。
4. 功能仿真判断代码的实现无误后时序仿真。
5. 查看RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】

四、实验过程

1. 编译过程
2. **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

    );

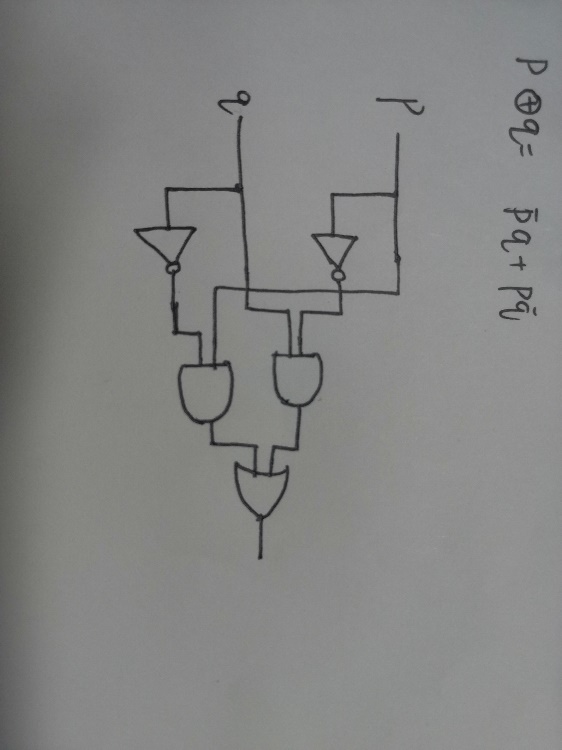
end entity twoin\_xor;

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;

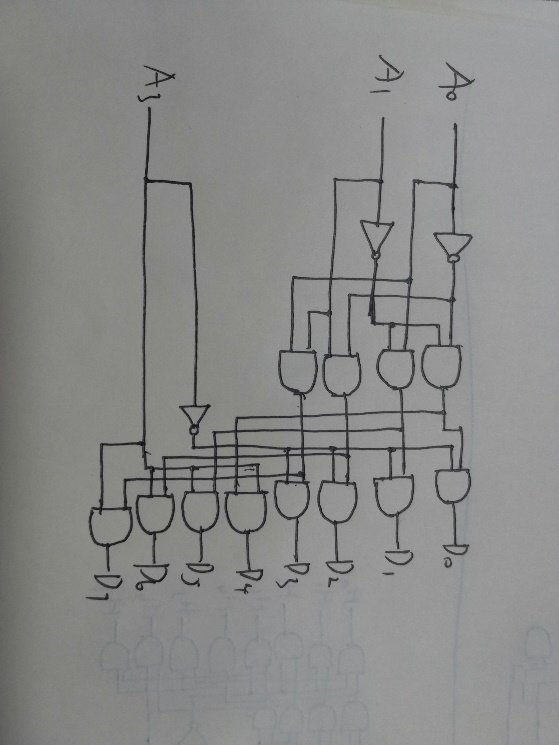
        x(3) <= temps(2) and inputs(1) and inputs(0) and enable;

        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, halt: 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 r2 and enable;

            sub <= tempcmd(0) and cmdcode(1) and cmdcode(2) and tempcmd(3) and r1 and r2 and enable;

            or1 <= cmdcode(0) and tempcmd(1) and cmdcode(2) and cmdcode(3) and r1 and r2 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 tempr2(0) and tempr2(1) and enable;

            rsl <= cmdcode(0) and tempcmd(1) and cmdcode(2) and tempcmd(3) and r1 and r2code(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 enable;

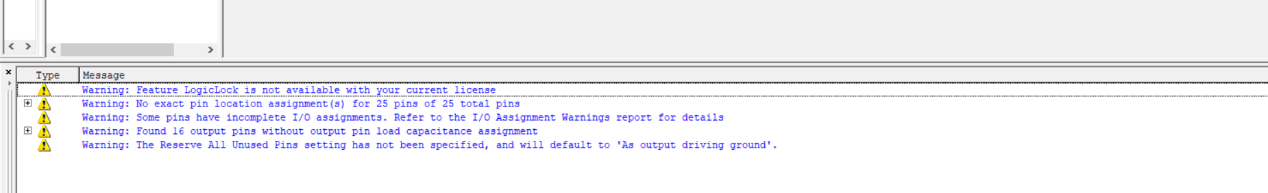
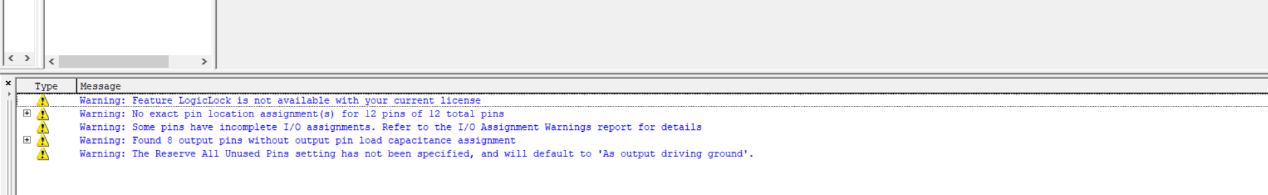
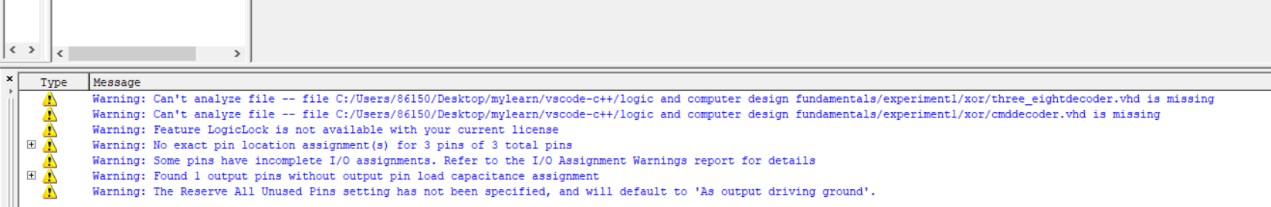
            out1 <= tempcmd(0) and cmdcode(1) and tempcmd(2) and tempcmd(3) and r1 and enable;

            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



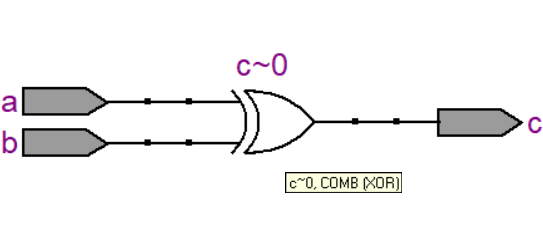
自上至下依次为异或门、3-8译码器、指令译码器的编译器提示信息。

三者编译器均给出警告，但均能够编译通过。

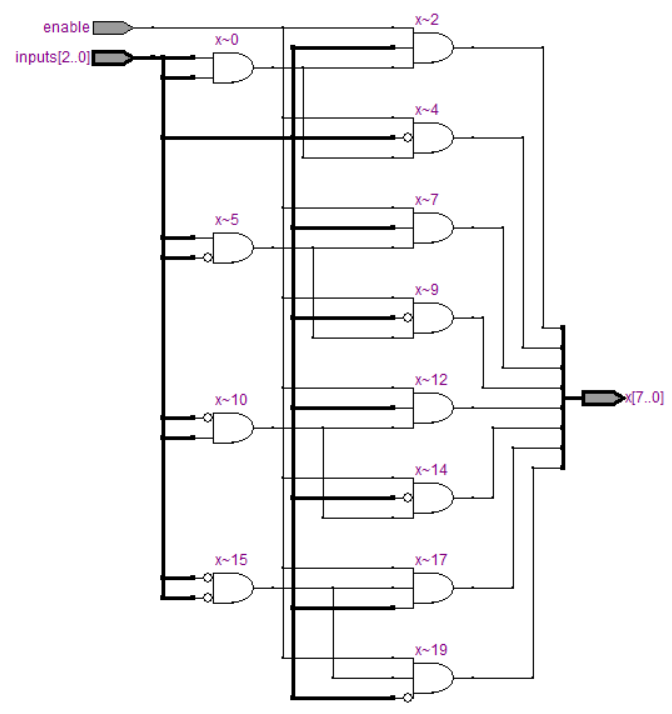
三者资源消耗基本都趋近于0.

c) RTL视图

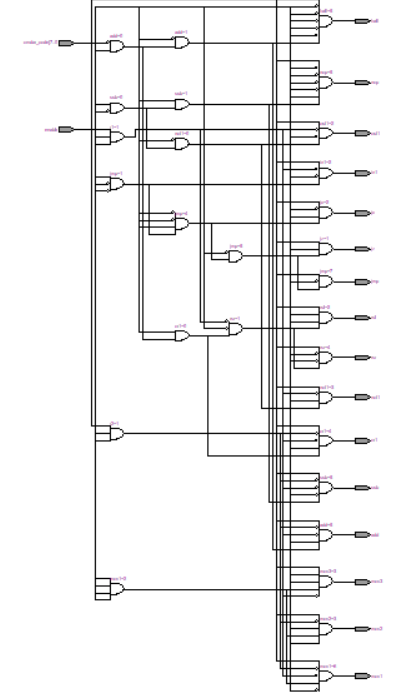
* 异或门



* 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

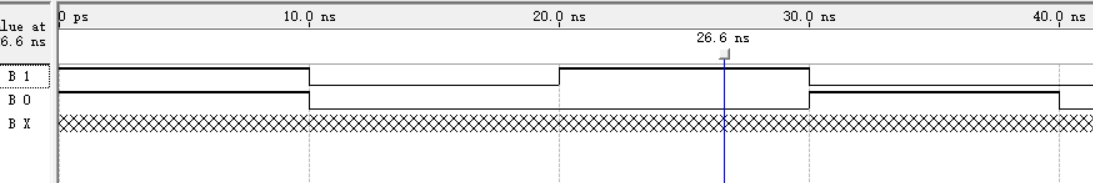
当输入为 1000 00 00 halt输出为1 其他指令输出为 0

1. 波形仿真

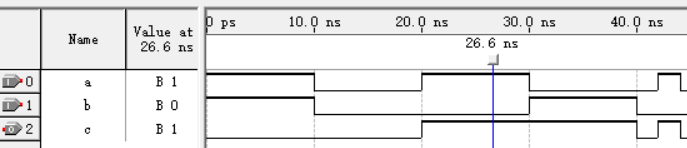
enable调为1时

* 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

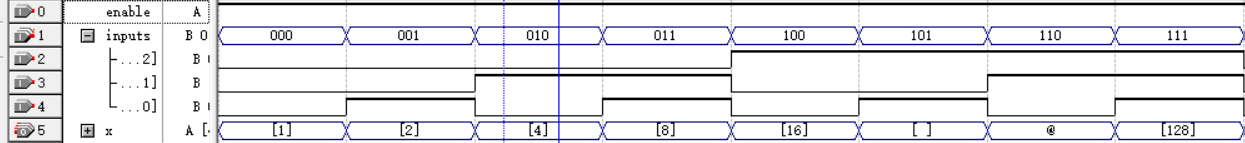
20-30ns时a,b输入为1 0，c输出为1

30-40ns时a,b输入为0 1，c输出为1

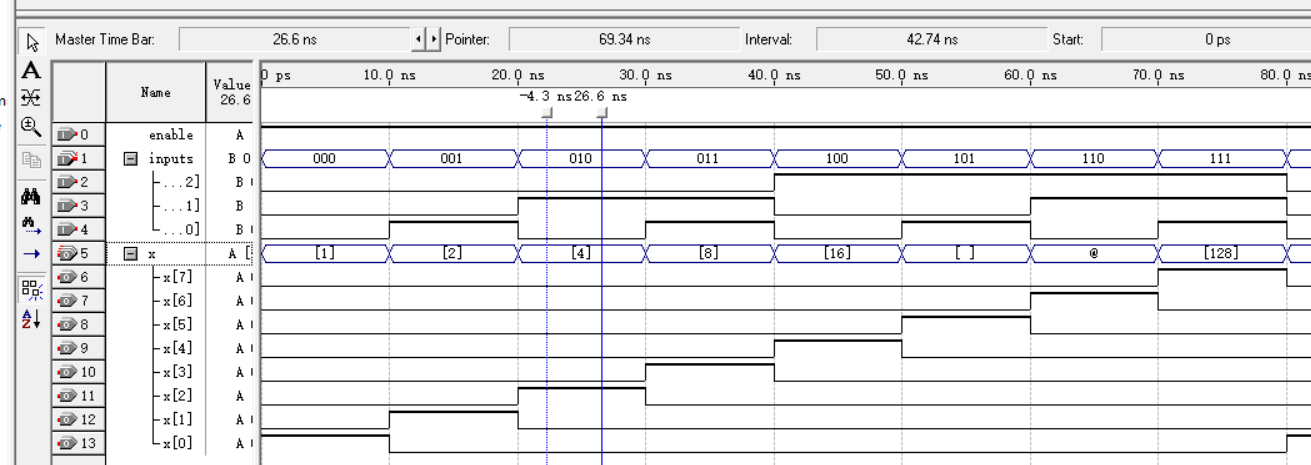
结果正确

* 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

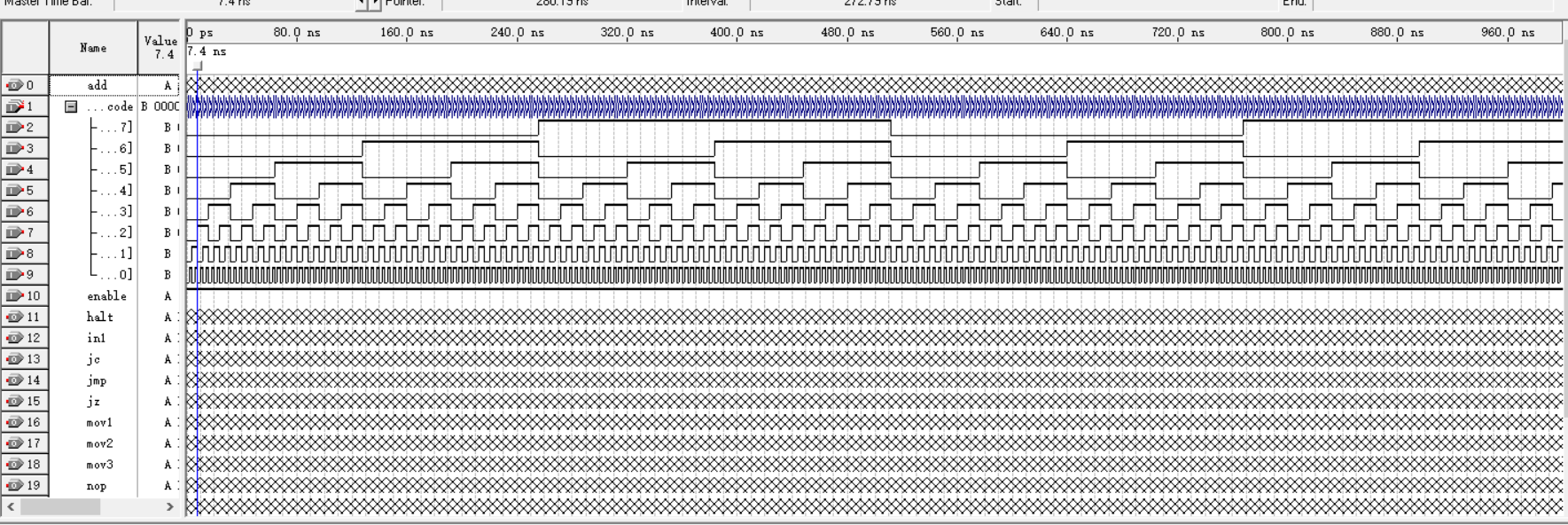
enable调为1，

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……

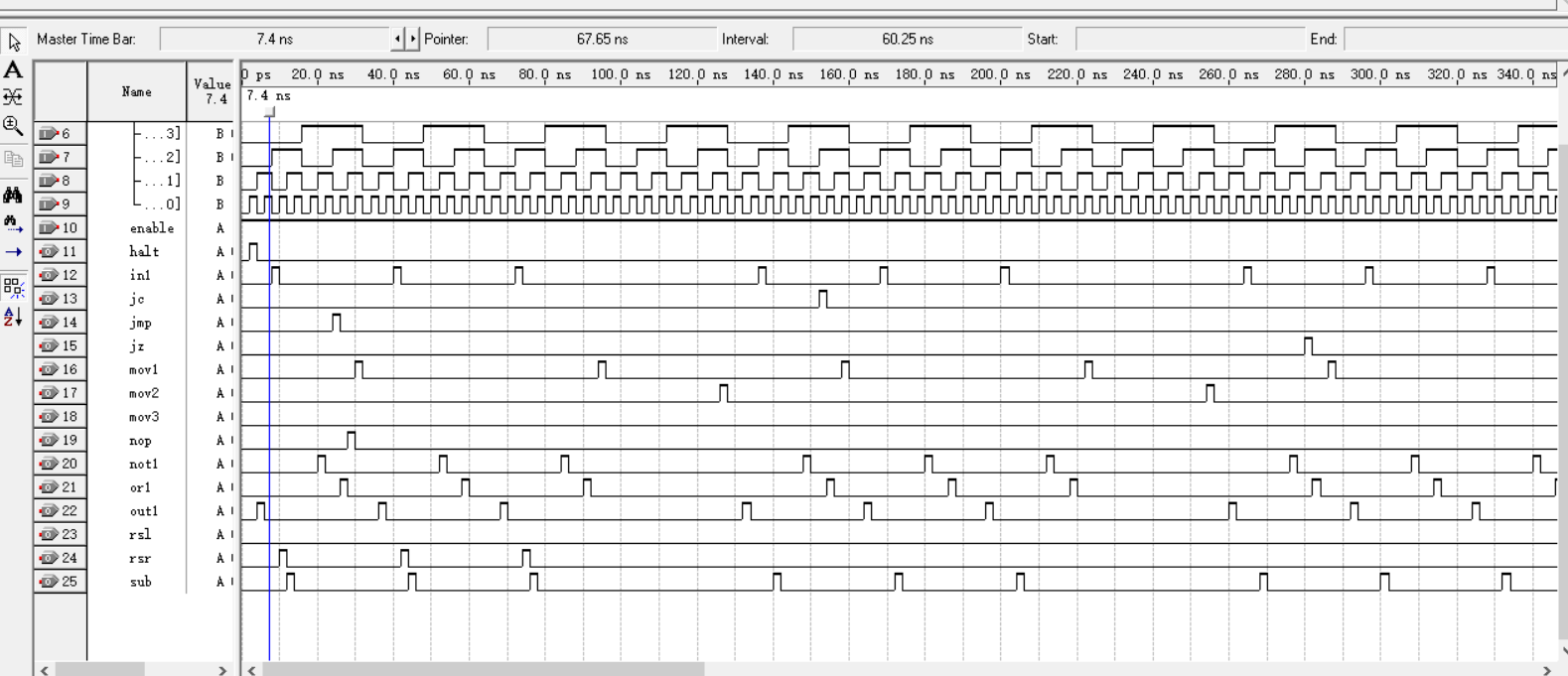
结果正确

* 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

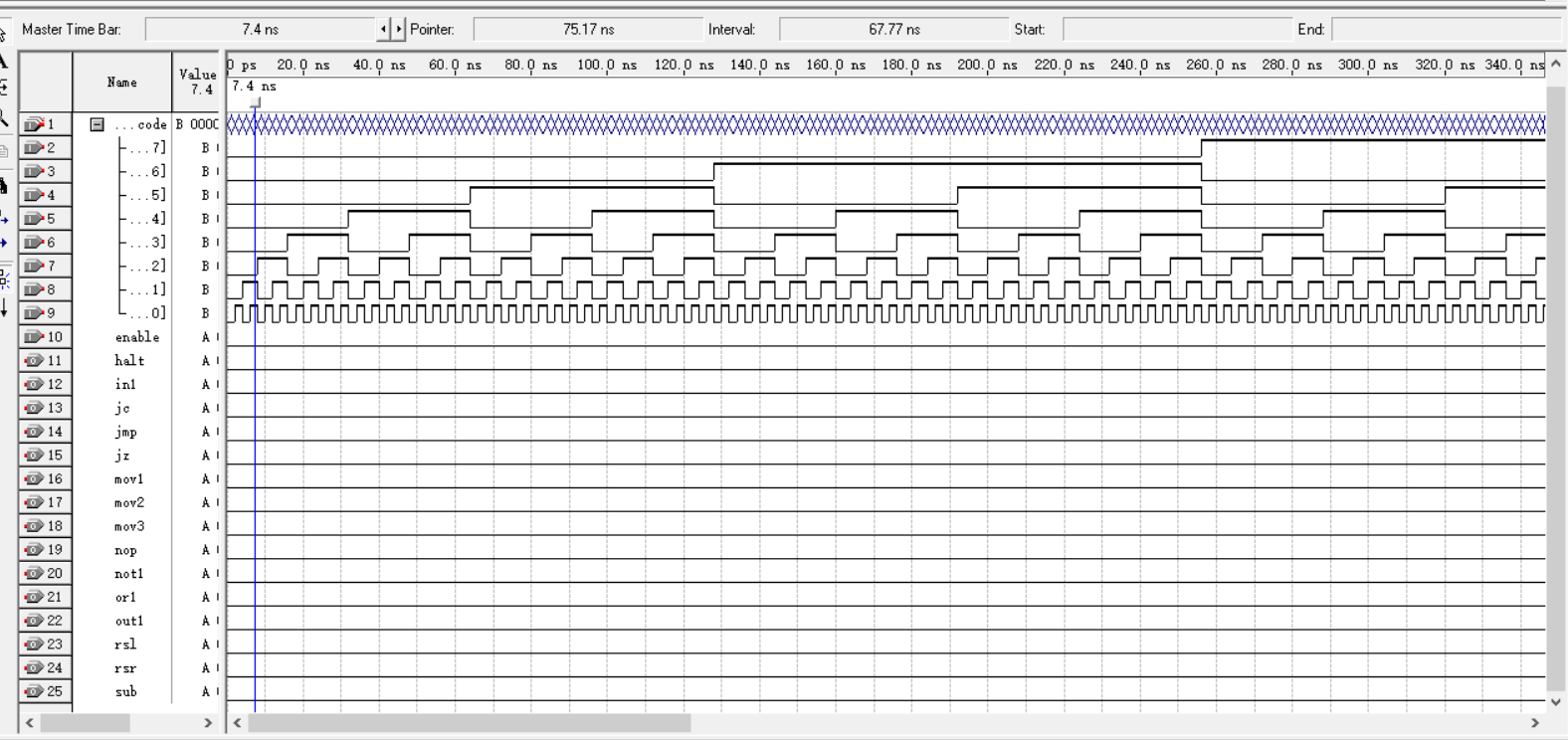
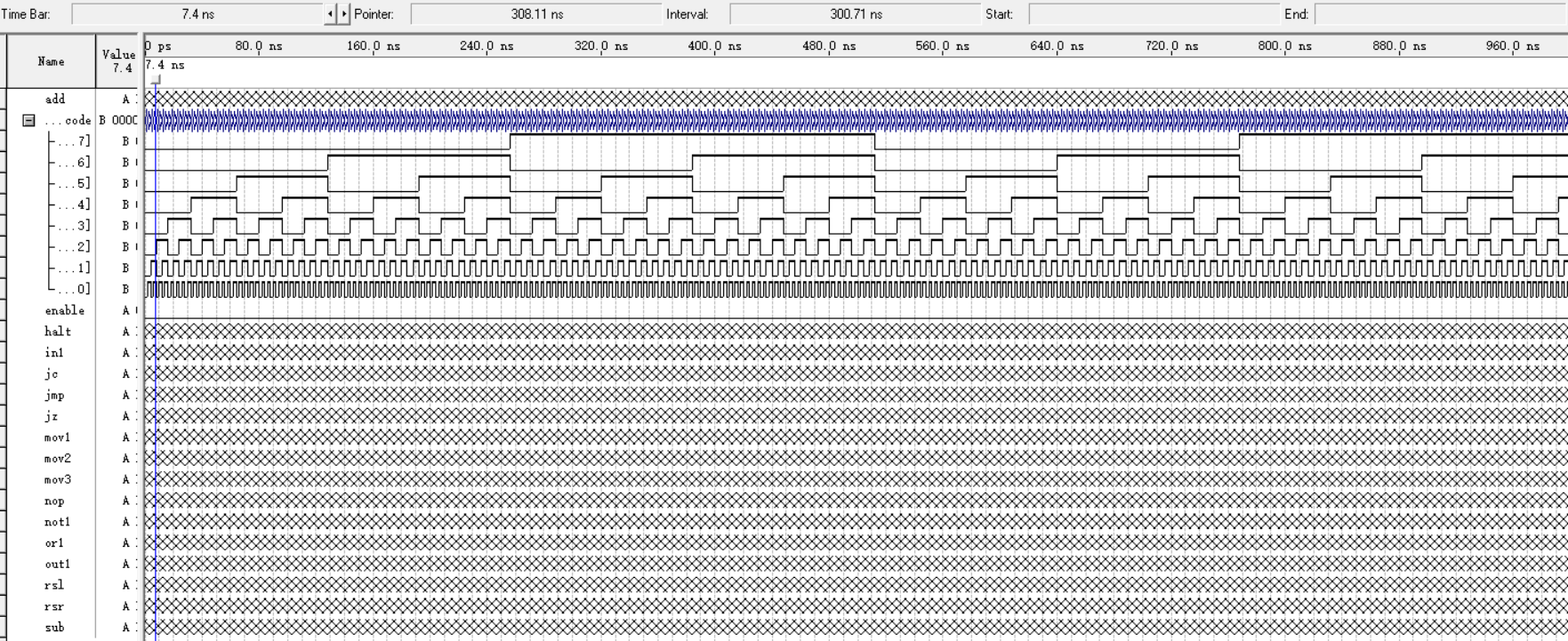
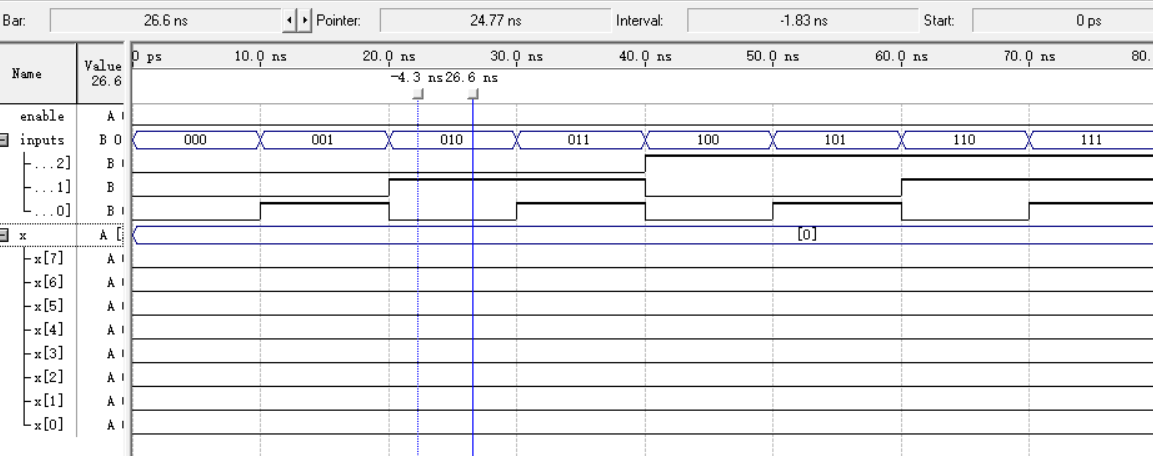
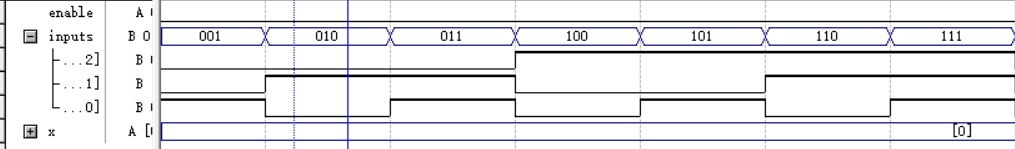


c)结果分析及结论

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

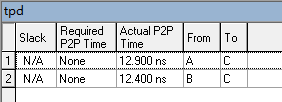
1. 时序仿真(enable为1时)

* 异或门

1. 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）

1. 时序仿真图



1. 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。

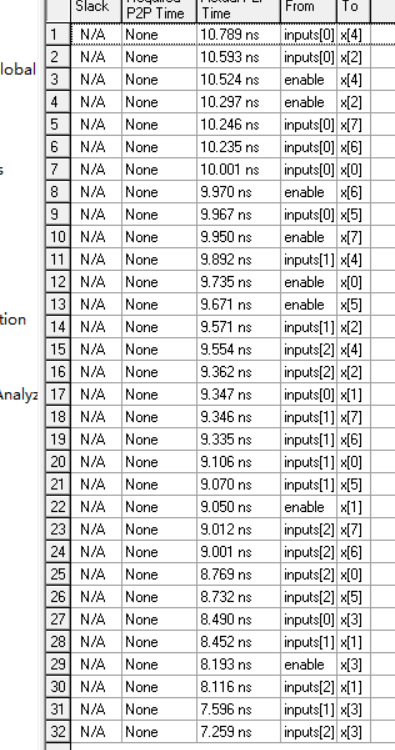
tpd (引脚到引脚的延时)

* 3-8译码器

1. 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）

1. 时序仿真图



1. 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。

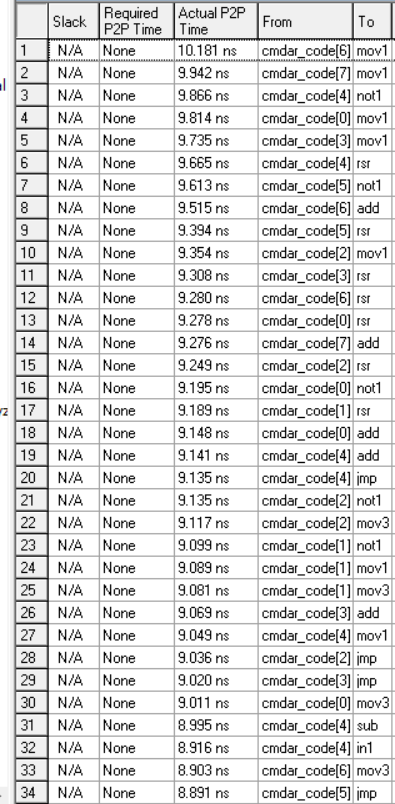
tpd (引脚到引脚的延时)

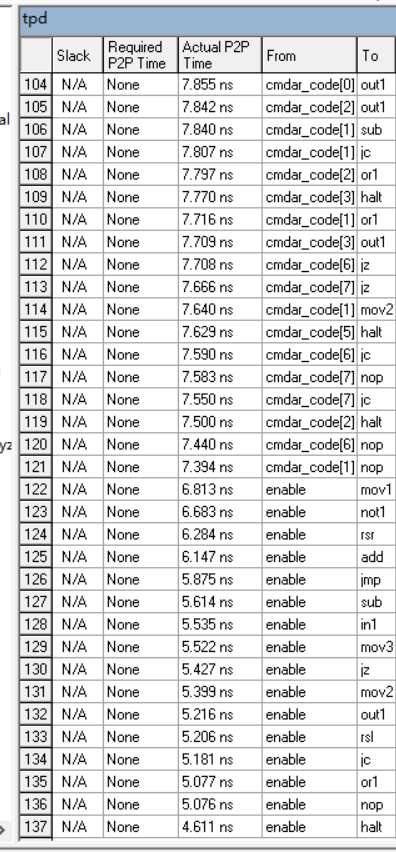
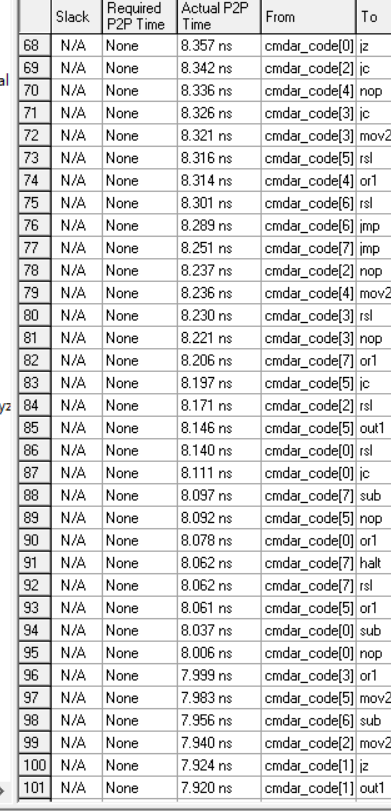
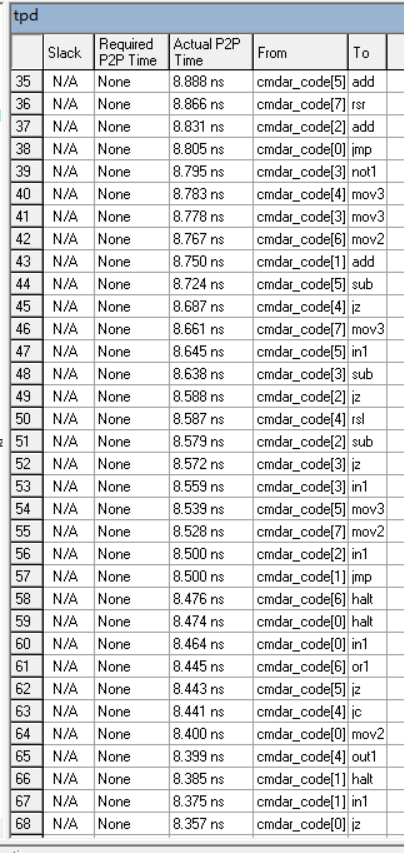
* 指令译码器

1. 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）

1. 时序仿真图





1. 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

1. 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。