湖南大學

数字电路与逻辑设计

课程实验报告

题　　目：Quartus II 软件的基本操作、译码器

学生姓名：魏子铖

学生学号：201726010308

专业班级：软件 1703

完成时间：2018.11.2

实验一 Quartus II 软件的基本操作、译码器

班级 软件 1703 姓名 魏子铖 学号 201706010308

一、实验目的

熟悉 Quartus II 软件的基本操作，了解各种设计方法（原理图设计、文本设计、波形设计）

二、实验内容

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。

采用的软件工具是 Quartus II 软件仿真平台。

2. 实验步骤

1) 新建工程文件

- 异或门的 VHDL 工程文件名为 myxor1
- 异或门的逻辑图工程文件名为 myxor2
- 3-8 译码器的 VHDL 工程文件名为 mydecoder3_8_1
- 3-8 译码器的逻辑图工程文件名为 mydecoder3_8_2
- 指令译码器的 VHDL 工程文件名为 command_decoder

2) 编写源代码或画逻辑图

完成后保存文件，文件名分别为 myxor1.vhd，myxor2.bdf，mydecoder3_8_1.vhd，mydecoder3_8_2.bdf，command_decoder.vhd，分别位于五个不同的工程文件下。

3) 编译与调试

确定文件名与工程文件名相同，点击【processing】-【start compilation】进行文件编译，文件编译成功。

4) 波形仿真及验证

为每个工程新建一个 vector waveform file。

按照程序所述插入所有输入和输出节点。任意设置输入波形，点击保存按钮保存，然后【start simulation】，得到波形仿真图。

5) 查看 RTL Viewer：【Tools】-【netlist viewer】-【RTL viewer】。

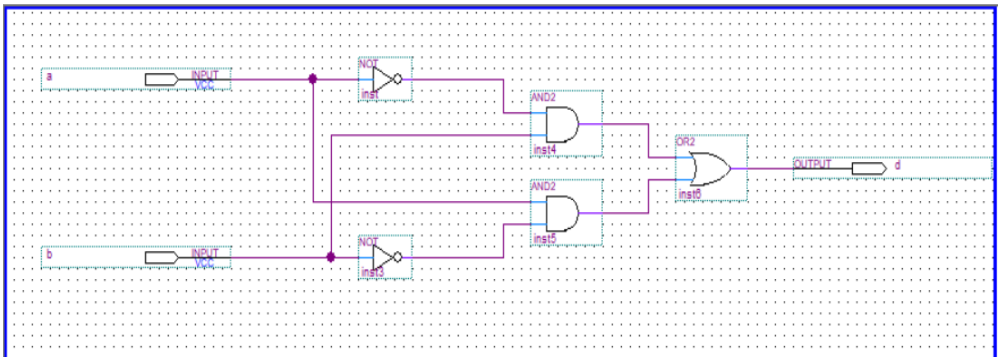
四、实验过程

1. 编译过程

异或门源代码：

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3
4   ENTITY myxor1 IS
5       PORT(a, b: in STD_LOGIC;
6                  d: out STD_LOGIC);
7   END myxor1;
8
9   ARCHITECTURE myxor OF myxor1 IS
10  BEGIN
11      d <= ((NOT a) AND b) OR (a AND (NOT b));
12  END myxor;
```

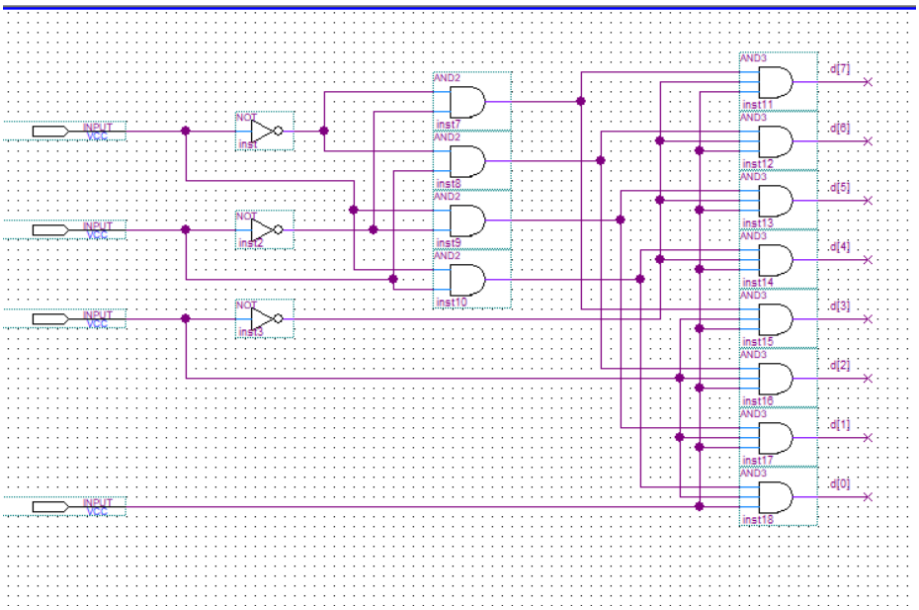
异或门的逻辑图：



3-8 译码器源代码：

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3
4   ENTITY mydecoder3_8_1 IS
5       PORT(a: in STD_LOGIC_VECTOR(2 DOWNTO 0);
6            d: out STD_LOGIC_VECTOR(7 DOWNTO 0));
7   END mydecoder3_8_1;
8
9   ARCHITECTURE decoder OF mydecoder3_8_1 IS
10  BEGIN
11      d <= "10000000" WHEN a = "000" ELSE
12           "01000000" WHEN a = "001" ELSE
13           "00100000" WHEN a = "010" ELSE
14           "00010000" WHEN a = "011" ELSE
15           "00001000" WHEN a = "100" ELSE
16           "00000100" WHEN a = "101" ELSE
17           "00000010" WHEN a = "110" ELSE
18           "00000001";
19  END decoder;
```

3-8 译码器逻辑图



指令译码器源代码：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY command_decoder IS
    PORT(a: in STD_LOGIC_VECTOR(7 DOWNTO 0);
         e: in STD_LOGIC;
         MOVA, MOVB, MOVC, ADD, SUB, AND0, NOT0,
         SHR, SHL, JMP, JZ, JC, IN0, OUT0, NOP,
         HALT: out STD_LOGIC);
END command_decoder;

ARCHITECTURE decoder OF command_decoder IS
SIGNAL R1, R2: STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL      s: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL s1, s2: STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL     s3: STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN
    PROCESS(a, e)
    BEGIN
        IF (e = '1') THEN
            R1 <= a(3) & a(2);
            R2 <= a(1) & a(0);
            s <= a(7) & a(6) & a(5) & a(4);
            s1 <= s & R1;
            s2 <= s & R2;
            s3 <= s & R1 & R2;
        ELSE
            R1 <= "00";
            R2 <= "00";
            s <= "0000";
            s1 <= "000000";
            s2 <= "000000";
            s3 <= "00111100";
```

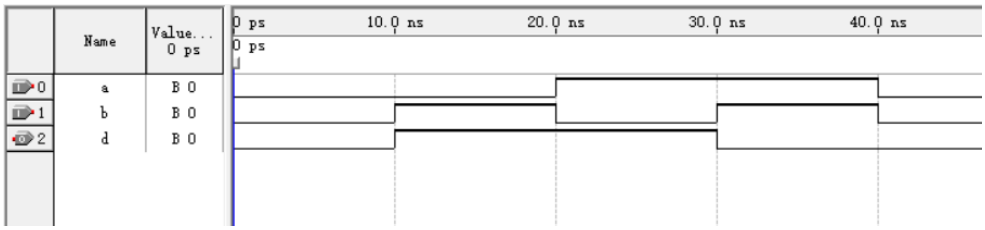
```
35          END IF;
36      END PROCESS;
37  WITH s3 SELECT
38      MOVA <= '1' WHEN "00110000",
39              '1' WHEN "00110001",
40              '1' WHEN "00110010",
41              '1' WHEN "00110100",
42              '1' WHEN "00110101",
43              '1' WHEN "00110110",
44              '1' WHEN "00111000",
45              '1' WHEN "00111001",
46              '1' WHEN "00111010",
47              '0' WHEN OTHERS;
48
49  WITH s1 SELECT
50      MOVB <= '1' WHEN "001111",
51              '0' WHEN OTHERS;
52
53  WITH s2 SELECT
54      MOVC <= '1' WHEN "001111",
55              '0' WHEN OTHERS;
56
57  WITH s SELECT
58      ADD <= '1' WHEN "1001",
59             '0' WHEN OTHERS;
60
61  WITH s SELECT
62      SUB <= '1' WHEN "0110",
63             '0' WHEN OTHERS;
64
65  WITH s SELECT
66      AND0 <= '1' WHEN "1110",
67              '0' WHEN OTHERS;
68
69  WITH s SELECT
70      NOT0 <= '1' WHEN "0101",
71              '0' WHEN OTHERS;
72
73  WITH s2 SELECT
74      SHR <= '1' WHEN "101000",
75             '0' WHEN OTHERS;
76
77  WITH s2 SELECT
78      SHL <= '1' WHEN "101011",
79             '0' WHEN OTHERS;
80
81  WITH s3 SELECT
82      JMP <= '1' WHEN "00010000",
83             '0' WHEN OTHERS;
84
85  WITH s3 SELECT
86      JZ <= '1' WHEN "00010001",
87            '0' WHEN OTHERS;
88
89  WITH s3 SELECT
90      JC <= '1' WHEN "00010010",
91            '0' WHEN OTHERS;
92
93  WITH s SELECT
94      IN0 <= '1' WHEN "0010",
95             '0' WHEN OTHERS;
96  WITH s SELECT
97      OUT0 <= '1' WHEN "0100",
98              '0' WHEN OTHERS;
```

```
 99    WITH s3 SELECT
100        NOP <= '1' WHEN "01110000",
101               '0' WHEN OTHERS;
102    WITH s3 SELECT
103        HALT <= '1' WHEN "10000000",
104                '0' WHEN OTHERS;
105
106    END decoder;
```

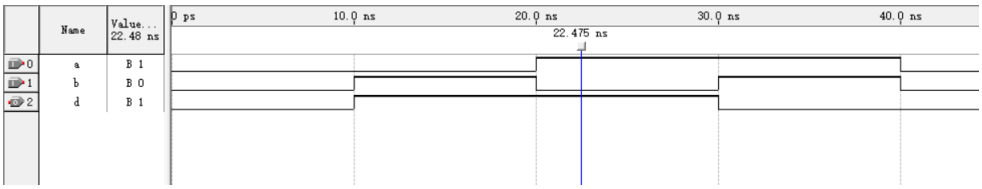
编译、调试过程：编写完成后进行全编译，编译成功，没有出现错误。
结果分析及结论：源代码及原理图没有出现语法或结构上的错误。

2. 波形仿真

a) 异或门仿真图：



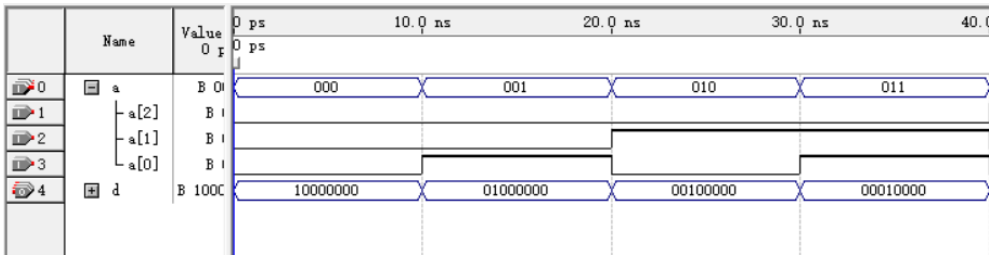
(VHDL)



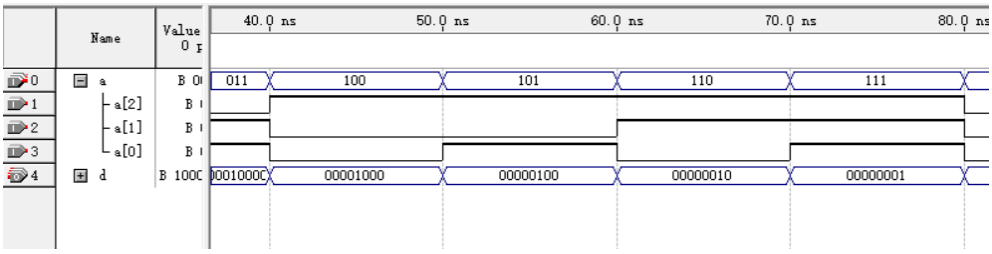
逻辑图

结果分析及结论：由异或运算的定义知，当输入的 a, b 相同时，输出 d 为 0，否则，输出 d 为 1。由波形仿真图可以明确地看出结果时正确的。

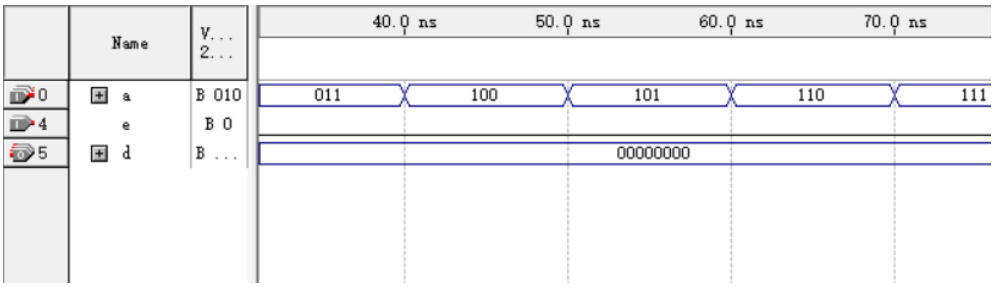
b) 3-8 译码器仿真图



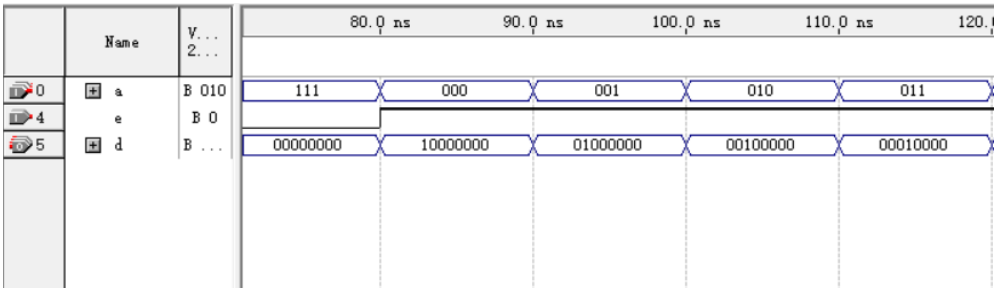
(VHDL)



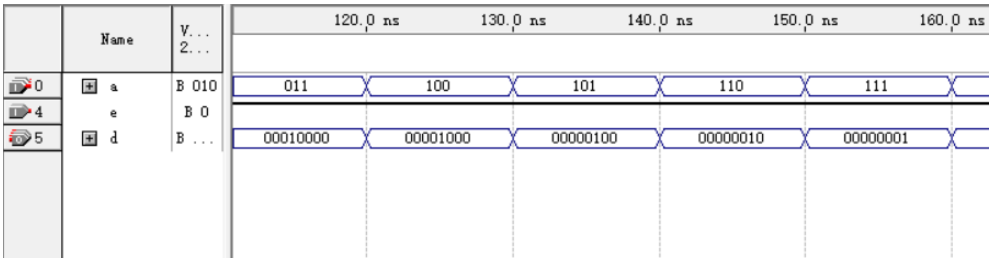
（VHDL）



逻辑图



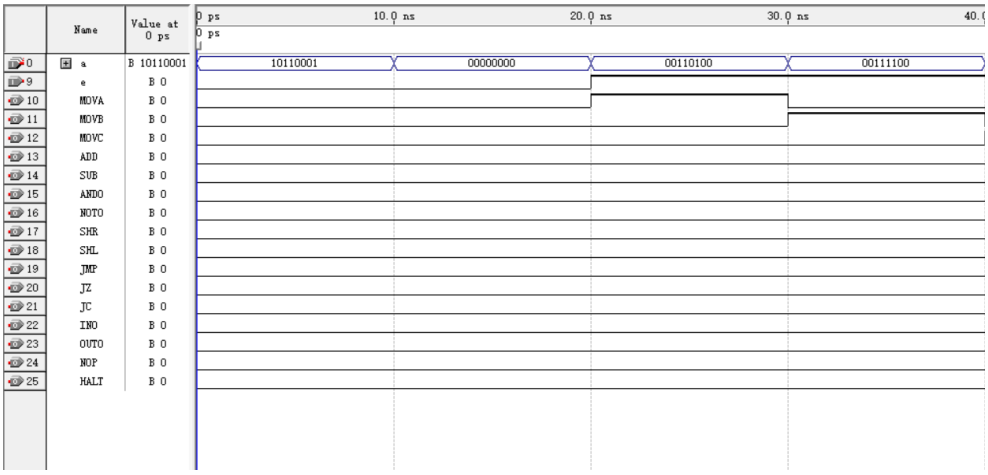
逻辑图

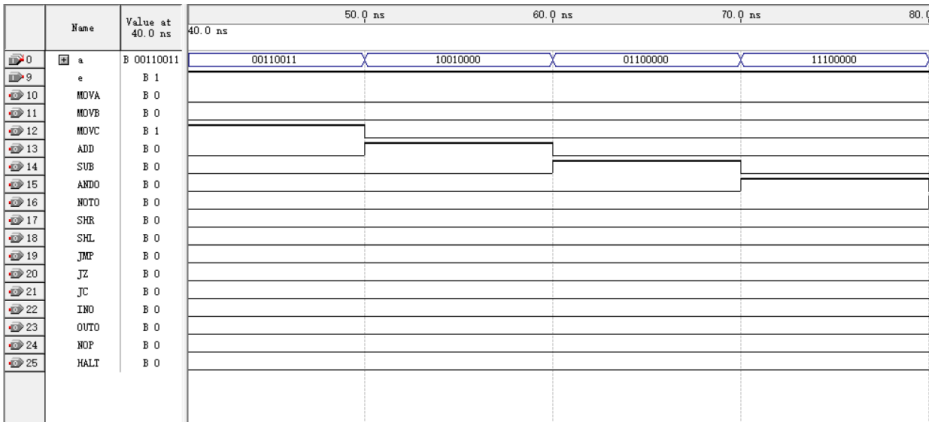


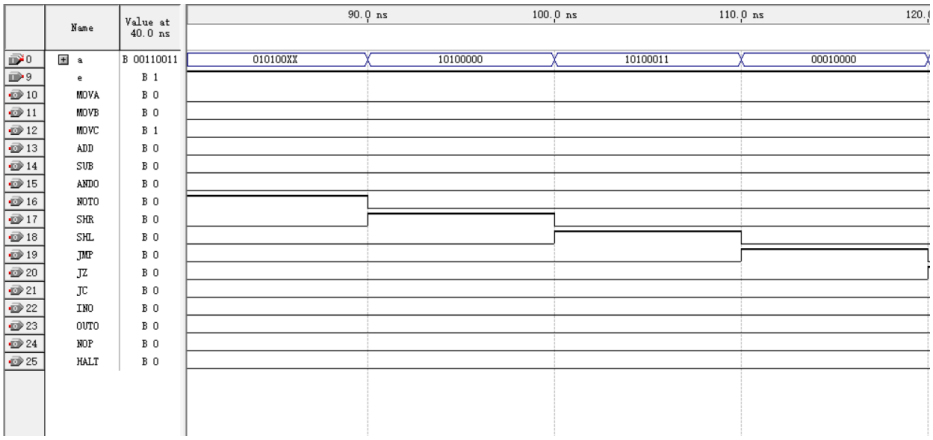
逻辑图

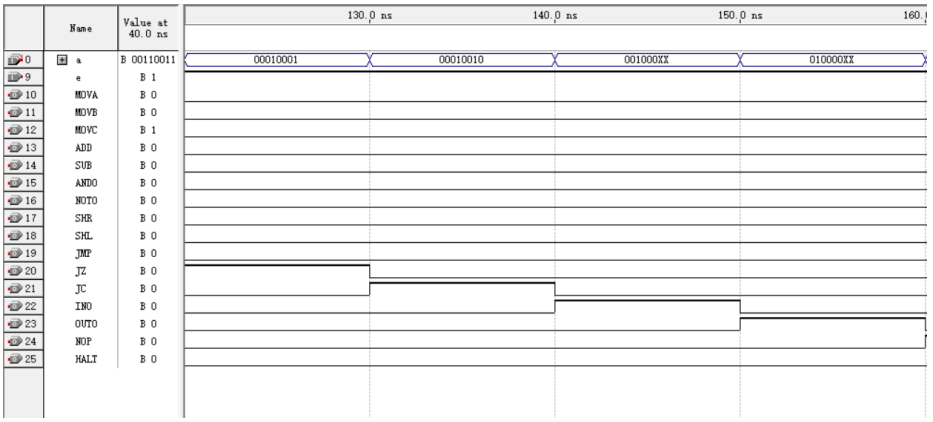
结果分析及结论：38 译码器是指将 3 位 2 进制数通过电路转换成八路不同状态的输出，输入 000 时输出 10000000，输入 001 时输出 01000000……在逻辑图的实现时加入了使能信号 e，当 e 为低电平时电路失效，e 为高电平时正常工作。由波形仿真图可以看出结果是正确的。

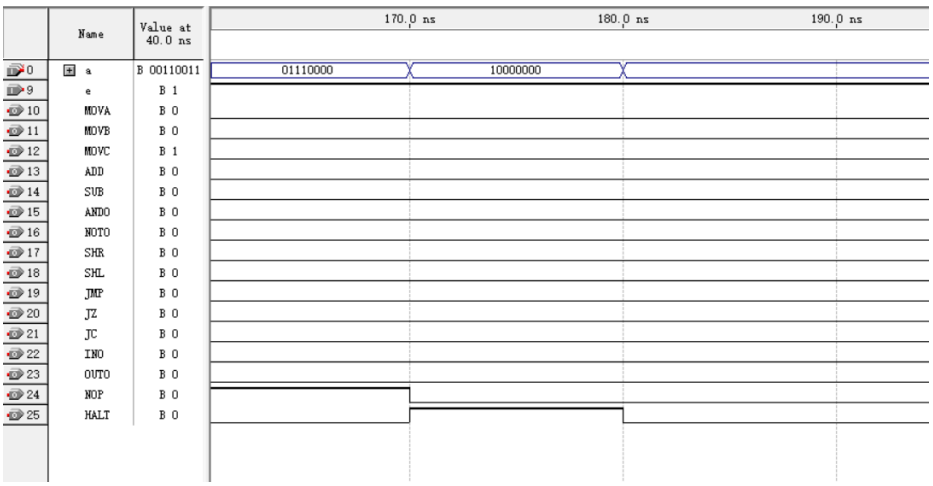
c) 指令译码器仿真图




Name
Value at 40.0 ns
50.0 ns
60.0 ns
70.0 ns
40.0 ns
0 a B 00110011
9 e B 1
10 MOVA B 0
11 MOVB B 0
12 MOVC B 1
13 ADD B 0
14 SUB B 0
15 ANDO B 0
16 NOTO B 0
17 SHR B 0
18 SHL B 0
19 JMP B 0
20 JZ B 0
21 JC B 0
22 INO B 0
23 OUTO B 0
24 NOP B 0
25 HALT B 0
00110011
10010000
01100000
11100000


Name
Value at 40.0 ns
90.0 ns
100.0 ns
110.0 ns
0 a B 00110011
9 e B 1
10 MOVA B 0
11 MOVB B 0
12 MOVC B 1
13 ADD B 0
14 SUB B 0
15 ANDO B 0
16 NOTO B 0
17 SHR B 0
18 SHL B 0
19 JMP B 0
20 JZ B 0
21 JC B 0
22 INO B 0
23 OUTO B 0
24 NOP B 0
25 HALT B 0
010100XX
10100000
10100011
00010000


Name
Value at 40.0 ns
130.0 ns
140.0 ns
150.0 ns
0 a B 00110011
9 e B 1
10 MOVA B 0
11 MOVB B 0
12 MOVC B 1
13 ADD B 0
14 SUB B 0
15 ANDO B 0
16 NOTO B 0
17 SHR B 0
18 SHL B 0
19 JMP B 0
20 JZ B 0
21 JC B 0
22 INO B 0
23 OUTO B 0
24 NOP B 0
25 HALT B 0
00010001
00010010
001000XX
010000XX


Name
Value at 40.0 ns
170.0 ns
180.0 ns
190.0 ns
0 a B 00110011
9 e B 1
10 MOVA B 0
11 MOVB B 0
12 MOVC B 1
13 ADD B 0
14 SUB B 0
15 ANDO B 0
16 NOTO B 0
17 SHR B 0
18 SHL B 0
19 JMP B 0
20 JZ B 0
21 JC B 0
22 INO B 0
23 OUTO B 0
24 NOP B 0
25 HALT B 0
01110000
10000000

结果分析及结论：

指令的汇编符号	指令的功能	指令的二进制编码
MOV R1，R2	(R2) → R1	0011 R1 R2
MOV M，R2	(R2) → (C)	0011 11 R2
MOV R1，M	((C)) →R1	0011 R1 11
ADD R1，R2	(R1) + (R2) → R1	1001 R1 R2
SUB R1，R2	(R1) - (R2) → R1	0110 R1 R2
AND R1，R2	(R1) ∧ (R2) → R1	1110 R1 R2
NOT R1	(R1) → R1	0101 R1 XX
SHR R1	(R1) 逻辑右移一位→ R1	1010 R1 00
SHL R1	(R1) 逻辑左移一位→ R1	1010 R1 11
JMP add	add → PC	0001 00 00，address
JZ add	结果为 0 时 add → PC	0001 00 01，address
JC add	结果有进位时 add → PC	0001 00 10，address
IN R1	(开关 7-0) → R1	0010 R1 XX
OUT R1	(R1) → 发光二极管 7-0	0100 R1 XX
NOP	(PC) +1 → PC	0111 00 00
HALT	停机	1000 00 00

根据输入的指令的二进制编码，将对应的功能输出高电平，对比波形仿真图可以发现结果是正确的。

五、实验结论

本次实验通过对指令译码器的编程，使我更加深刻的了解了计算机内部的执行操作的顺序，首先，用户输入指令代码，然后指令译码器转换成为机器能够识别的机器码，再转交给指令执行部件去执行，指令译码器将复杂的汇编代码简化成为了简单的二进制代码，从而大大增加了 cpu 的执行能力与运算速度。