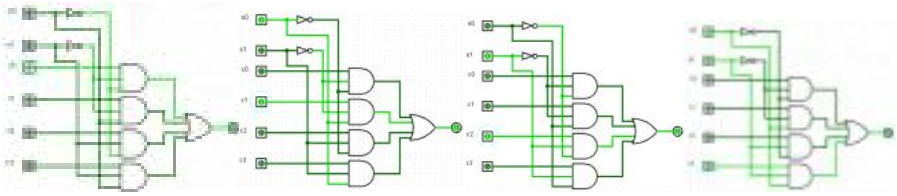
1. 数据选择器实验

一、实验内容

了解数据选择器的原理，使用 Verilog 实现数据选择器，并下板进行验证

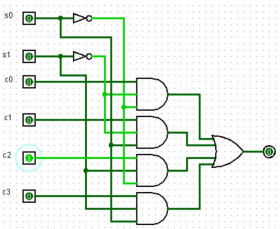
二、硬件逻辑图

1. 一路数据选择器



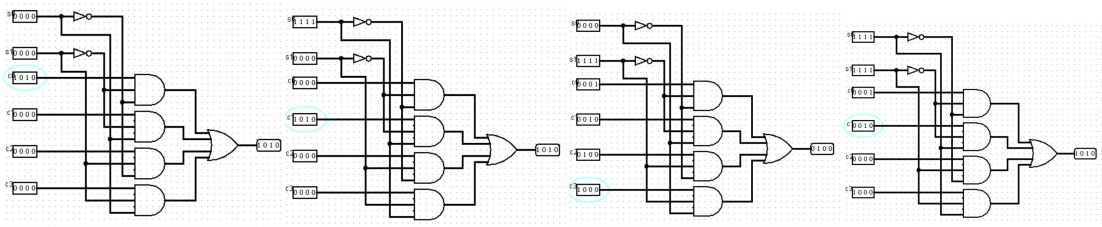
00　　01　　10　　11

当限制值为 00 时实验其他路不能输出，其他错误值同理



2. 四路数据选择器

四路数据选择器限制输入端本应该是一位的输入，由于 logicsim 门的输入位数需要匹配，所以此处也设置成四位的输入。且四位输入是每一对应位决定相应输入的对应位能否被输出



00　　01　　10　　11

从图 10 可以看出，当限制置为 10 的时候，仅有 s2 位置的信号被输出

三、模块建模

对于输出的每一位分别进行判断，当符合其对应的特定输入的时候就将其赋值给输出端口
利用条件控制语句进行判断来进行赋值

```
module selector41(
    input [3:0] iC0,
    input [3:0] iC1,
    input [3:0] iC2,
    input [3:0] iC3,
    input iS1,
    input iS0,
    output [3:0]oZ
    );
    assign oZ[0] = (~iS0 && ~iS1)? iC0[0]:
                   (iS0 && ~iS1)? iC1[0]:
                   (~iS0 && iS1)? iC2[0]:iC3[0];
    assign oZ[1] = (~iS0 && ~iS1)? iC0[1]:
                   (iS0 && ~iS1)? iC1[1]:
                   (~iS0 && iS1)? iC2[1]:iC3[1];
    assign oZ[2] = (~iS0 && ~iS1)? iC0[2]:
                   (iS0 && ~iS1)? iC1[2]:
                   (~iS0 && iS1)? iC2[2]:iC3[2];
    assign oZ[3] = (~iS0 && ~iS1)? iC0[3]:
                   (iS0 && ~iS1)? iC1[3]:
                   (~iS0 && iS1)? iC2[3]:iC3[3];
endmodule
```

四、测试模块建模

对于每个限制信号运行 160s 的时间，每隔 40s 将一个信号送入相应输入端口，观察输出信号是哪个端口对应的信号

```
`timescale 1ns / 1ns
module selector41_tb;
    reg [3:0] iC0;
    reg [3:0] iC1;
    reg [3:0] iC2;
    reg [3:0] iC3;
    reg iS1;
    reg iS0;
    reg [3:0] oZ;
```

```
selector41 sel(.iC0(iC0), .iC1(iC1), .iC2(iC2),.iC3(iC3), .iS0(iS0), .iS1(iS1), .oZ(oZ));

initial
begin
    iS0 = 0;
    #40 iS0 = 0;
    #40 iS0 = 0;
    #40 iS0 = 0;

    #40 iS1 = 1;
    #40 iS1 = 1;
    #40 iS1 = 1;
    #40 iS1 = 1;

    #40 iS0 = 0;
    #40 iS0 = 0;
    #40 iS0 = 0;
    #40 iS0 = 0;

    #40 iS1 = 1;
    #40 iS1 = 1;
    #40 iS1 = 1;
    #40 iS1 = 1;
end

initial
begin
    iS1 = 0;
    #40 iS1 = 0;
    #40 iS1 = 0;
    #40 iS1 = 0;

    #40 iS1 = 0;
    #40 iS1 = 0;
    #40 iS1 = 0;
    #40 iS1 = 0;

    #40 iS1 = 1;
    #40 iS1 = 1;
    #40 iS1 = 1;
    #40 iS1 = 1;

    #40 iS1 = 1;
    #40 iS1 = 1;
```

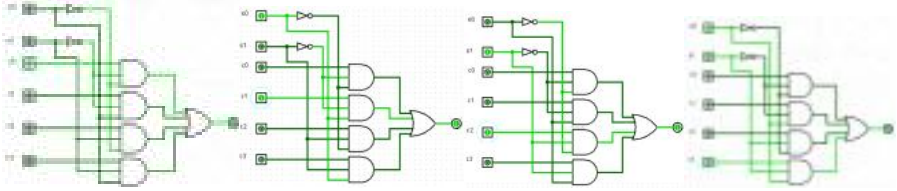
```
        #40 iS1 = 1;
        #40 iS1 = 1;
    end

    initial
    begin
        iC0 = 4'b0001;
        #40 iC1 = 4'b0010;
        #40 iC2 = 4'b0011;
        #40 iC3 = 4'b0100;

        #40 iC0 = 4'b0001;
        #40 iC1 = 4'b0010;
        #40 iC2 = 4'b0011;
        #40 iC3 = 4'b0100;

        #40 iC0 = 4'b0001;
        #40 iC1 = 4'b0010;
        #40 iC2 = 4'b0011;
        #40 iC3 = 4'b0100;

        #40 iC0 = 4'b0001;
        #40 iC1 = 4'b0010;
        #40 iC2 = 4'b0011;
        #40 iC3 = 4'b0100;
    end
endmodule
```

五、实验结果

（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

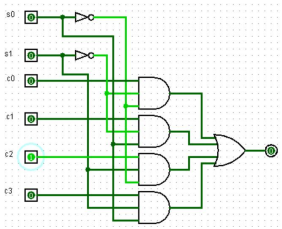
Logicsim 逻辑验证图：

一位数字选择器

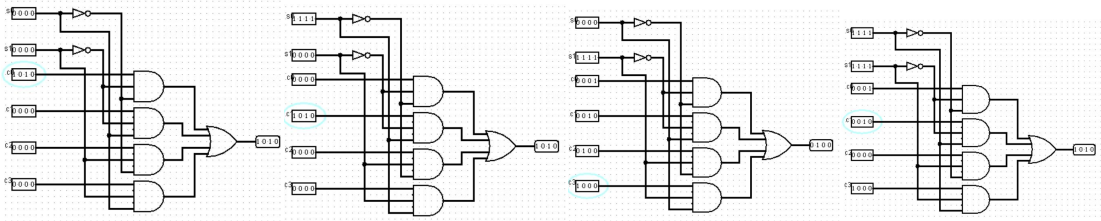


00　　01　　10　　11

当限制值为 00 时实验其他路不能输出，其他错误值同理



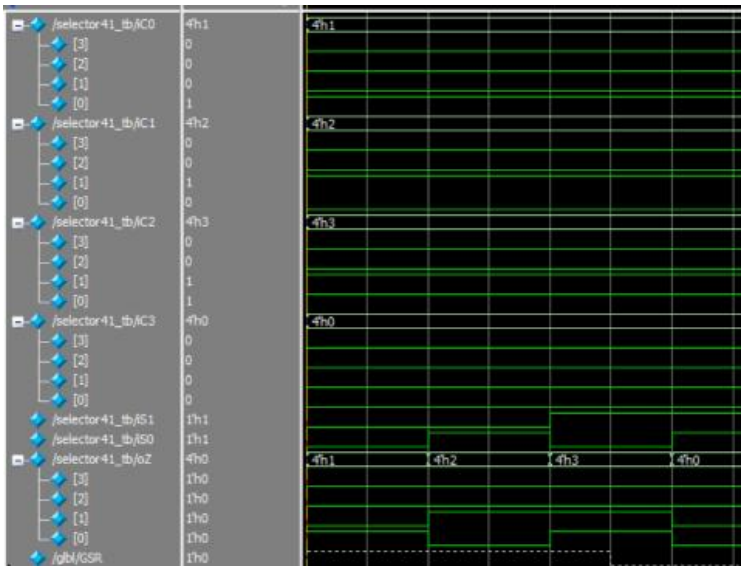
四位数字选择器



00　　01　　10　　11

可以看到在限制条件下仅指定位信息可以被选择

Modelsim 仿真波形图



各个输入端的信号各不相同，可以看到在相应限制下一些端口才会被输出，从输出端可以看到相应的波形

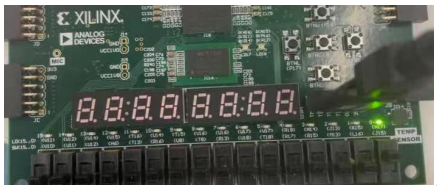
实际下板

其中从右到左，每四位代表一个四位的输入信号，由右到左分别是 c0 c1 c2 c3 输入端

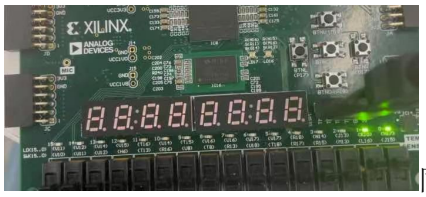
右上方中间键和右方向键分别为是 s1 和 s0 控制端

从右到左前四位是输出端

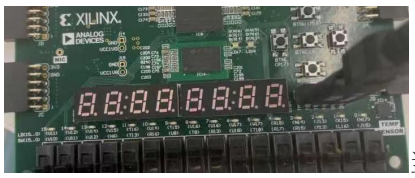
1. 控制端为 00



当控制端为 00 时，第一位输入信号可以输出

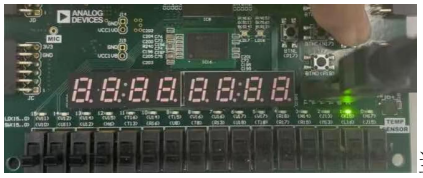


同上，测试第二组数据



当 00 限制时，d1 输入端口输入信号，输出端口没有反应

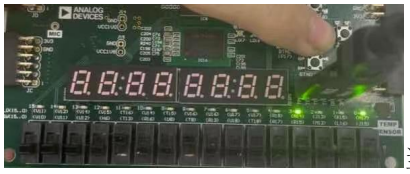
2. 控制端为 01



当 01 限制时，d2 输入端输入信号，输出端正常输出


同上，测试第二组

3. 控制端为 10



当 10 限制时，d3 输入信号，输出端正常输出

 同上，测试第二组

4. 控制端为 11

 当 11 限制时，d3 输入信号，输出端正常输出

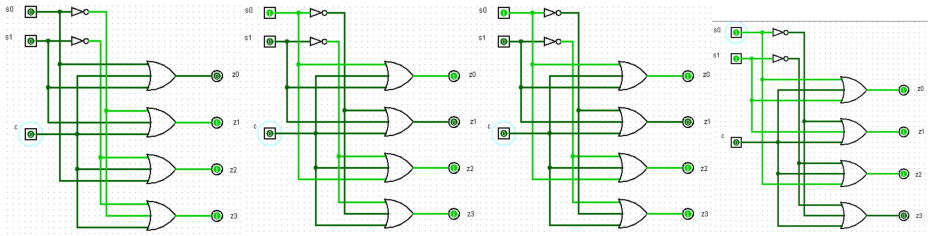
 同上，测试第二组

2. 数据分配器实验

一、实验内容

用 verilog 语言实现数据分配器。其功能与多路选择器相反，单路输入，多路输出，当输出端口不是指定的端口时，输出值为 1

二、硬件逻辑图



00　　01　　10　　11

可以看到，当输入端口为指定端口输入时可以将输入端口的信息输出
而其他的位置的位置保持为 1

三、模块建模

对于每一位进行判断，若符合限制则进行输出否则输出 1

```
module de_selector14(
      input iC,
      input iS1,
      input iS0,
      output oZ0,
      output oZ1,
      output oZ2,
      output oZ3
      );
    assign oZ0 = (~iS1 && ~iS0)? iC:1;

    assign oZ1 = (~iS1 && iS0)? iC:1;

    assign oZ2 = (iS1 && ~iS0)? iC:1;

    assign oZ3 = (iS1 && iS0)? iC:1;

endmodule
```

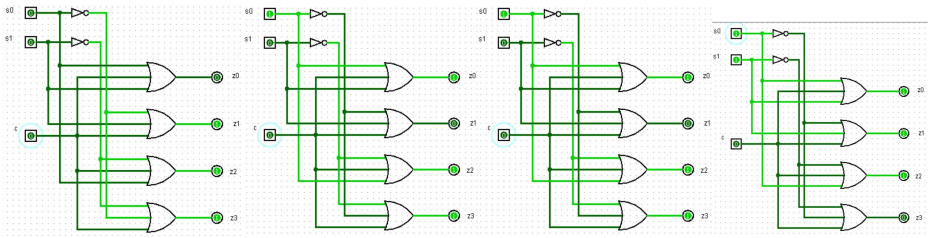
四、测试模块建模

初始时对于四个输入信号都赋值为 0，通过改变控制端的控制信号观察是否会在相应端口进行输出

```
`timescale 1ns / 1ns
module de_selector14_tb;
    reg iC;
    reg iS1;
    reg iS0;
    wire oZ0;
    wire oZ1;
    wire oZ2;
    wire oZ3;

    de_selector14        deS
    (.iC(iC), .iS1(iS1), .iS0(iS0), .oZ0(oZ0), .oZ1(oZ1), .oZ2(oZ2), .oZ3(oZ3));

    initial
    begin
        iS0 = 0;
        #40 iS0 = 1;
        #40 iS0 = 0;
        #40 iS0 = 1;
    end


    initial
    begin
        iS1 = 0;
        #40 iS1 = 0;
        #40 iS1 = 1;
        #40 iS1 = 1;
    end

    initial
    begin
        iC = 0;
    end

endmodule
```

五、实验结果

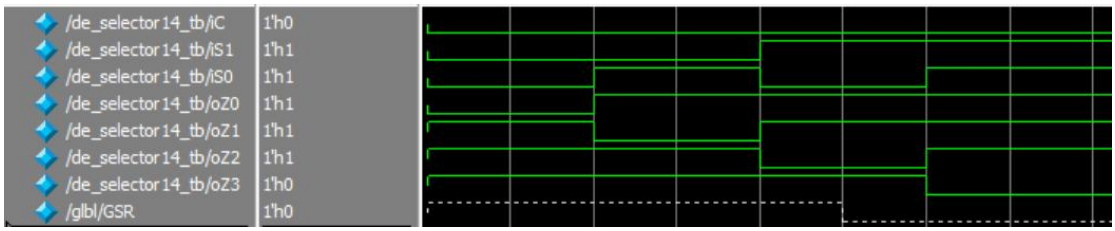
Logicsim 逻辑验证图



00　　01　　10　　11

可以看到，当输入端口为指定端口输入时可以将输入端口的信息输出
而其他的位置的位置保持为 1

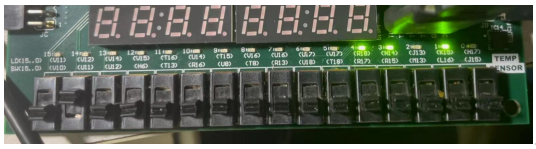
modelsim 仿真波形图

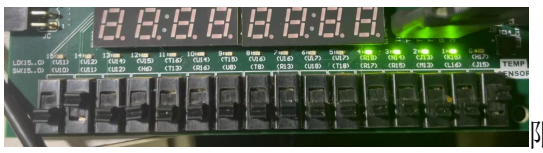


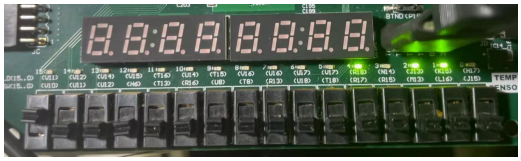
可以看到当输入端低电平时，通过依次改变限制条件，输出端依次变为低电平，实现了相应逻辑

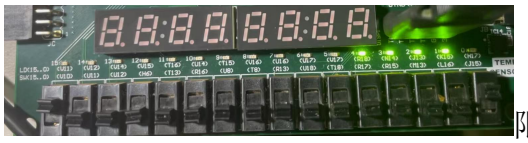
下板验证
输入管脚为右侧第一个，再往左依次四个是输出管脚的最低位到最高位，最左侧两个管脚分别是 s1 和 s0


当限制为 00 时，可以看到输入为 0，仅有 0 输出位为 0


当限制为 01 时，可以看到输入为 0，仅有 1 位输出为 0


限制为 01 时，输入为 1，1 输出端变为 1


限制为 10 时，输入为 0，2 输出端变为 0


限制为 10 时，输入为 1，2 输出端为 1


限制为 11 时，输入为 0，3 输出端为 0


限制为 11 时，输入变为 1，输出变为 1

、

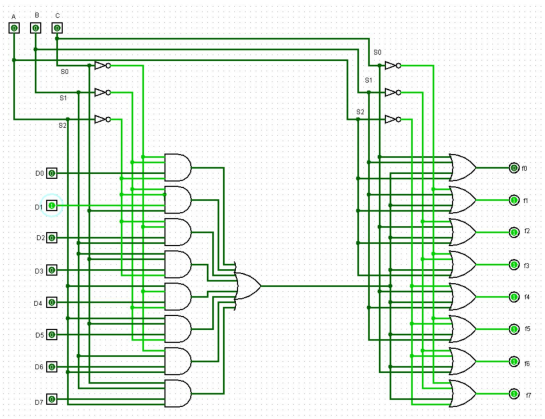
3.8 路数据传输实验

一、实验内容

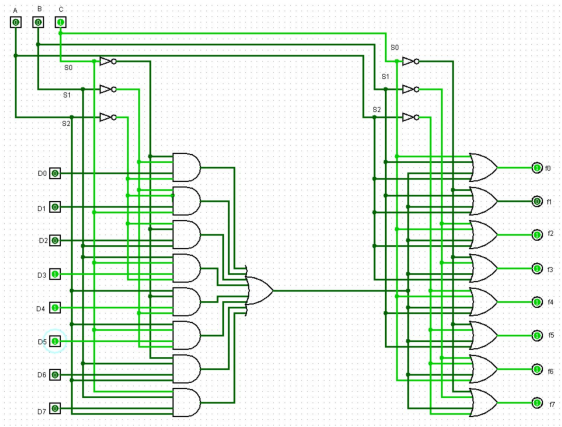
在实现数据选择器和分配器的基础上实现 8 路传输模块的建模，并下板进行验证

二、硬件逻辑图

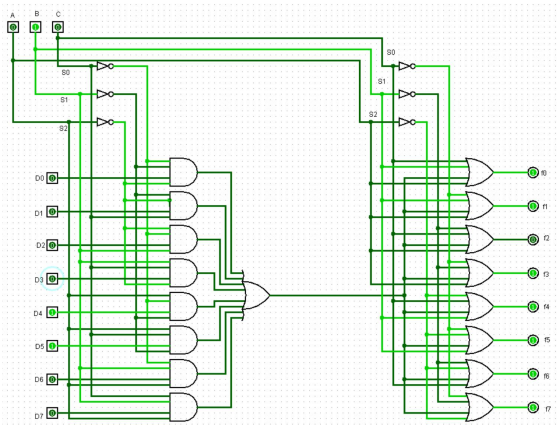
000



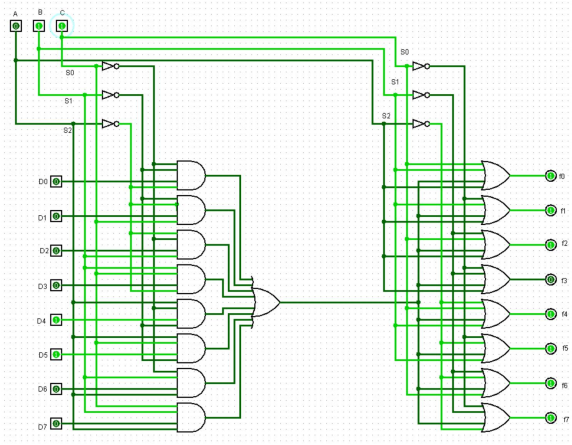
001



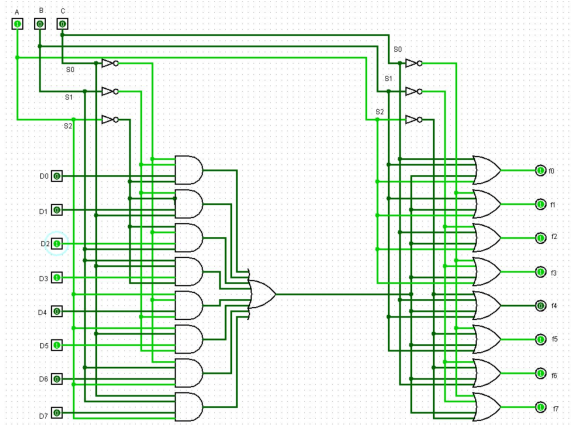
010



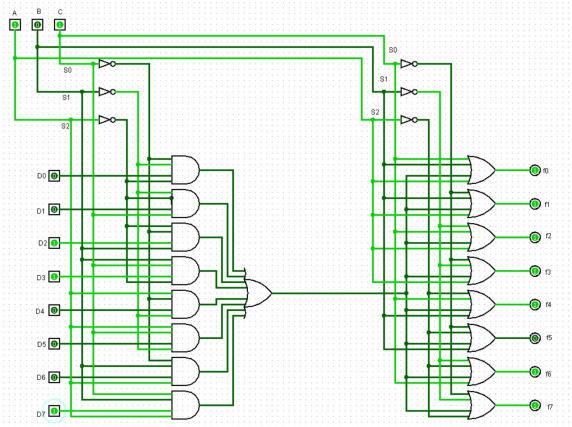
011


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

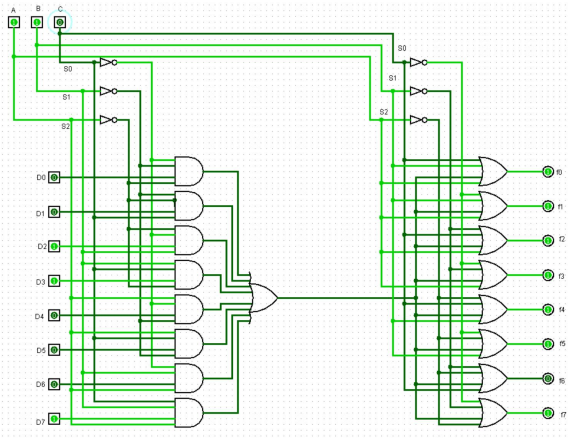
100


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

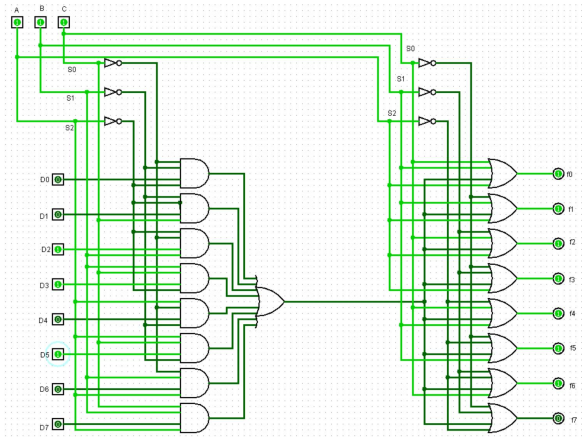
101


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

110



111



可以看到当控制端口限制时，仅有对应的输出端口会将信号传输（即 0 信号传输过去），其他端口信号并没有传输过去

三、模块建模

通过建立一个 8 位的数据选择器，一个 8 位的数据分配器，用 wire 类型的中间量来连接选择器的输出与分配器的输入实现相应功能

```
module selector81(    //8 位数据选择器
    input iC0,
    input iC1,
    input iC2,
    input iC3,
    input iC4,
    input iC5,
    input iC6,
    input iC7,
    input iS2,
    input iS1,
    input iS0,
```

```
    output oZ
    );
    assign oZ = (~iS2 && ~iS1 && ~iS0)? iC0:
                (~iS2 && ~iS1 && iS0)? iC1:
                (~iS2 && iS1 && ~iS0)? iC2:
                (~iS2 && iS1 && iS0)? iC3:
                (iS2 && ~iS1 && ~iS0)? iC4:
                (iS2 && ~iS1 && iS0)? iC5:
                (iS2 && iS1 && ~iS0)? iC6:iC7;


endmodule


module de_selector18(    //8 位数据分配器
    input iC,
    input iS2,
    input iS1,
    input iS0,
    output oZ0,
    output oZ1,
    output oZ2,
    output oZ3,
    output oZ4,
    output oZ5,
    output oZ6,
    output oZ7
    );
   assign oZ0 = (~iS2 && ~iS1 && ~iS0)? iC:1;

   assign oZ1 = (~iS2 && ~iS1 && iS0)? iC:1;

   assign oZ2 = (~iS2 && iS1 && ~iS0)? iC:1;

   assign oZ3 = (~iS2 && iS1 && iS0)? iC:1;

   assign oZ4 = (iS2 && ~iS1 && ~iS0)? iC:1;

   assign oZ5 = (iS2 && ~iS1 && iS0)? iC:1;

   assign oZ6 = (iS2 && iS1 && ~iS0)? iC:1;

   assign oZ7 = (iS2 && iS1 && iS0)? iC:1;
```

```
endmodule



module transmission8(
    input [7:0] iData,
    input A,
    input B,
    input C,
    output [7:0] oData
    );

    wire internal;    //wire 导线连接输出与输入

    selector81 sel81(
    .iC0(iData[0]),
    .iC1(iData[1]),
    .iC2(iData[2]),
    .iC3(iData[3]),
    .iC4(iData[4]),
    .iC5(iData[5]),
    .iC6(iData[6]),
    .iC7(iData[7]),
    .iS2(A),
    .iS1(B),
    .iS0(C),
    .oZ(internal)
    );

    de_selector18 deS18(
    .iC(internal),
    .iS2(A),
    .iS1(B),
    .iS0(C),
    .oZ0(oData[0]),
    .oZ1(oData[1]),
    .oZ2(oData[2]),
    .oZ3(oData[3]),
    .oZ4(oData[4]),
    .oZ5(oData[5]),
    .oZ6(oData[6]),
    .oZ7(oData[7])
    );
```

```
endmodule
```

四、测试模块建模

初始时将所有输入信号置为 0，每 40ns 从 000 到 111 依次变换输入限制，观察输出端口，哪一个输出端口为 0，哪一个输入被传出

```
`timescale 1ns / 1ns
module transmission8_tb;
    reg [7:0] iData;
    reg A;
    reg B;
    reg C;
    wire [7:0] oData;

    transmission8 trans8(    //实例化
    .iData(iData),
    .A(A),
    .B(B),
    .C(C),
    .oData(oData)

    );

    //依次变换三位限制，从 000 到 111
    initial
    begin
        A = 0;
        #40 A = 0;
        #40 A = 0;
        #40 A = 0;
        #40 A = 1;
        #40 A = 1;
        #40 A = 1;
        #40 A = 1;
    end

    initial
    begin
        B = 0;
        #40 B = 0;
        #40 B = 1;
```

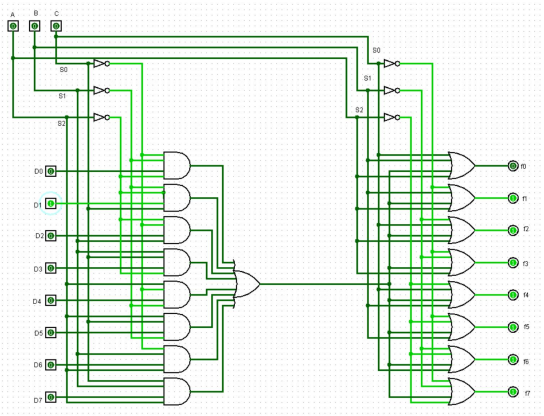
```
        #40 B = 1;
        #40 B = 0;
        #40 B = 0;
        #40 B = 1;
        #40 B = 1;
    end

    initial
    begin
        C = 0;
        #40 C = 1;
        #40 C = 0;
        #40 C = 1;
        #40 C = 0;
        #40 C = 1;
        #40 C = 0;
        #40 C = 1;
    end

    //初始将信号均赋值为 0，便于观察输出
    initial
    begin
        iData[0] = 0;
        iData[1] = 0;
        iData[2] = 0;
        iData[3] = 0;
        iData[4] = 0;
        iData[5] = 0;
        iData[6] = 0;
        iData[7] = 0;
    end
endmodule
```

五、实验结果

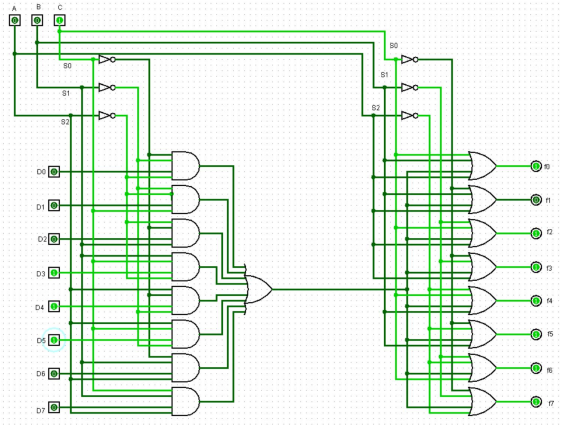
（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

Logicsim 逻辑验证图

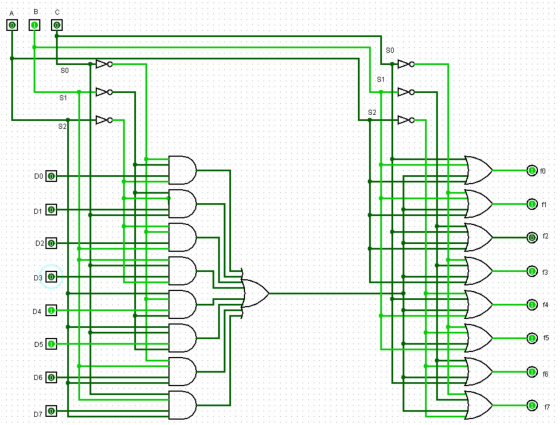
000


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

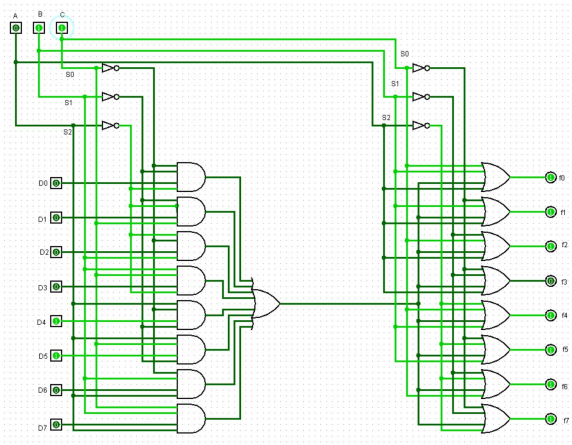
001


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

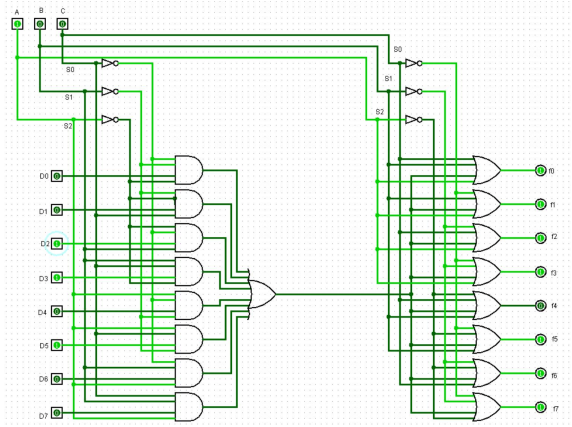
010


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

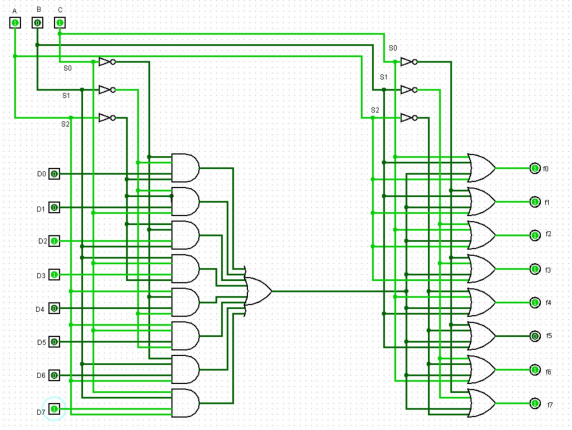
011


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

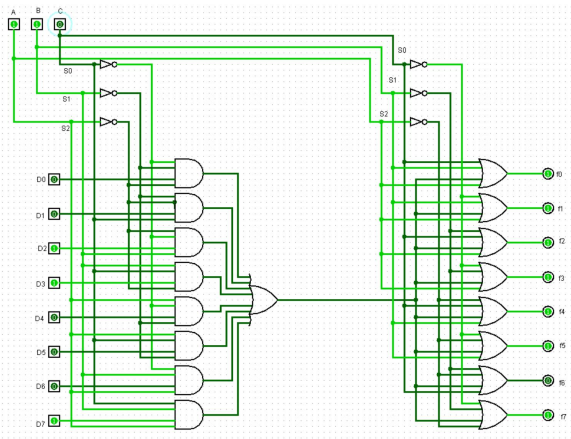
100


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

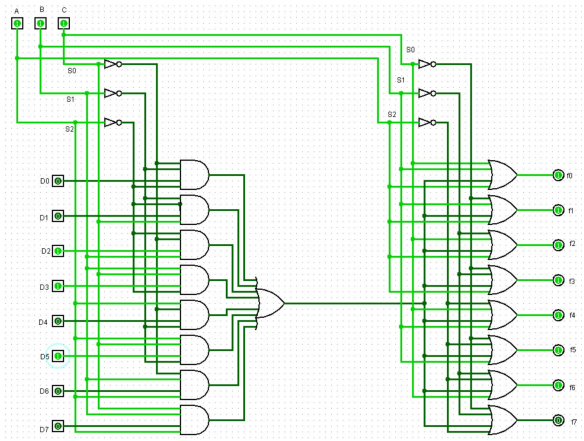
101


A
B
C
S0
S1
S2
D0
D1
D2
D3
D4
D5
D6
D7
f0
f1
f2
f3
f4
f5
f6
f7

110

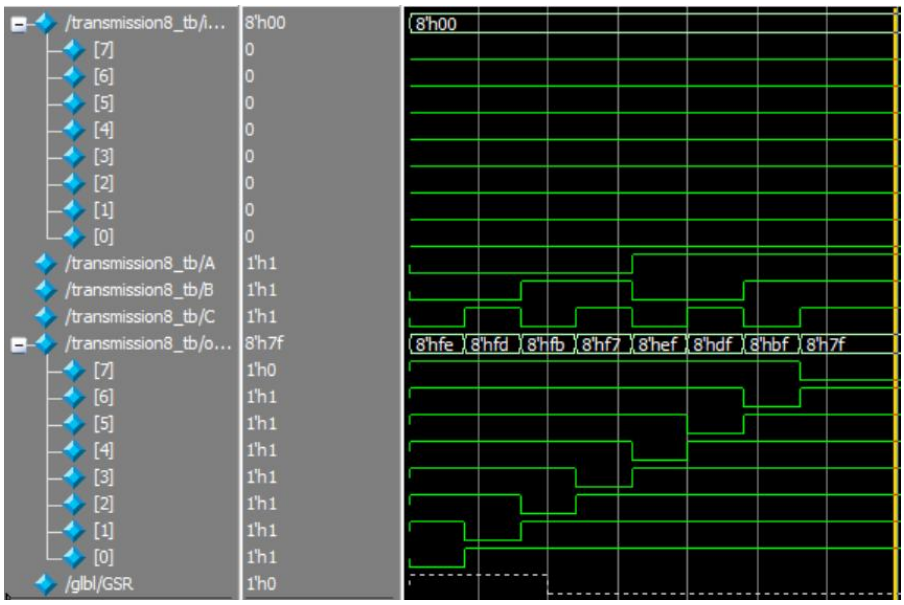


111



可以看到仅有限制的输入位才能将低电平信号传送出去，其他输入端的低电平无法传送

Modelsim 仿真波形图



从波形可以很明显看到，依次改变输入端的限制，输入端的低电平依次可以传输过去，而相应的其他输入端的低电位对输出端没有影响

下板验证


当限制为 001 时，1 位置有输入，1 位置输出信号为 1


当限制为 00 时，1 位置无输入，1 位置输出 0；且可以观察到其他位置信号输入为 1 不影响 1 位置输出为 0


当限制为 011 时，3 位置无输入，3 位置无输出，且 3 位置输入不影响其他位置输出


当限制为 011 时，3 位置有输入，3 位置有输出为 1，


当限制为 101 时，5 位置无输入，5 位置无输出，且 5 位置无输入不影响其他位置输出


当限制为 101 时，5 位置有输入，5 位置有输出


当限制为 111 时，7 位置有输入，7 位置有输出

当限制为 111 时，7 位置无输入，7 位置无输出，且 7 位置无输入不影响其他位置正常输出