实验一 组合逻辑设计

一、实验目的

1 掌握 Verilog 语言和 Vivado、Logisim 开发平台的使用；

2 掌握基础组合逻辑电路的设计和测试方法。

二、实验内容（用 Logisim 或 Vivado 实现）

1 基础门电路（多输入门电路、复用器等）的设计和测试；

2 基础功能模块（编码器、译码器等）的设计与测试。

三、实验要求

1 掌握 Vivado 与 Logisim 开发工具的使用，掌握以上电路的设计和测试方法；

2 记录设计和调试过程（Verilog 代码/电路图/表达式/真值表，Vivado 仿真结果，Logisim 验证结果等）；

3 分析 Vivado 仿真波形/Logism 验证结果，注重输入输出之间的对应关系。

四、实验过程及分析

一、多输入门电路

一、Verilog 代码

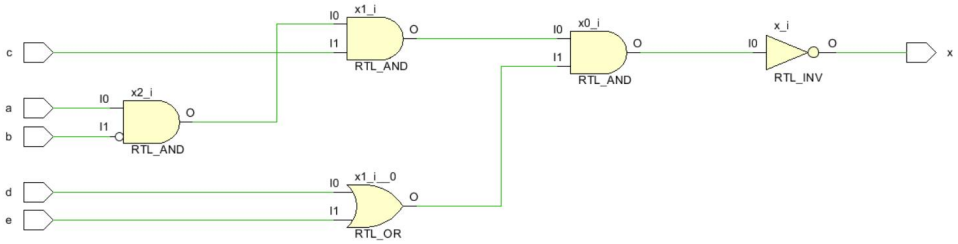
```
module and5(
    input a, b, c, d, e,
    output x
    );
    assign x = ~(a & (~b) & c & (d | e));
endmodule
```

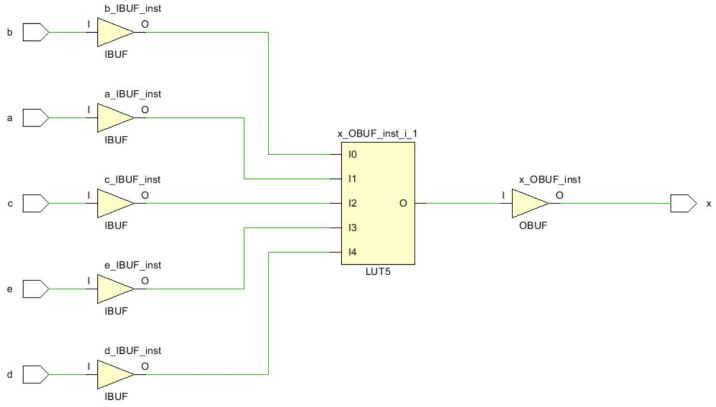
```
module sim_and5;
    reg a, b, c, d, e;
    wire x;
    and5 and5(a,b,c,d,e,x);
    initial
    begin
        a=0;b=0;c=0;d=0;e=0;
        fork
            repeat(100) #10 a=~a;
            repeat(50) #20 b = ~b;
            repeat(25) #40 c = ~c;
            repeat(10) #100 d = ~d;
            repeat(5) #200 e = ~e;
        join
    end
endmodule
```

and5:这是一个逻辑门模块，实现了一个复杂的逻辑表达式，用于计算输出信号 x。输入端包括 a, b, c, d, 和 e，均为 input 类型，表示输入端。输出端 x 声明为 output 类型，表示输出端。通过 assign 语句，x 的值计算为 x=~(a· ~b· c· (d+e))。该逻辑门模块的功能是将输入信号进行逻辑计算，然后计算输出信号 x 的值。

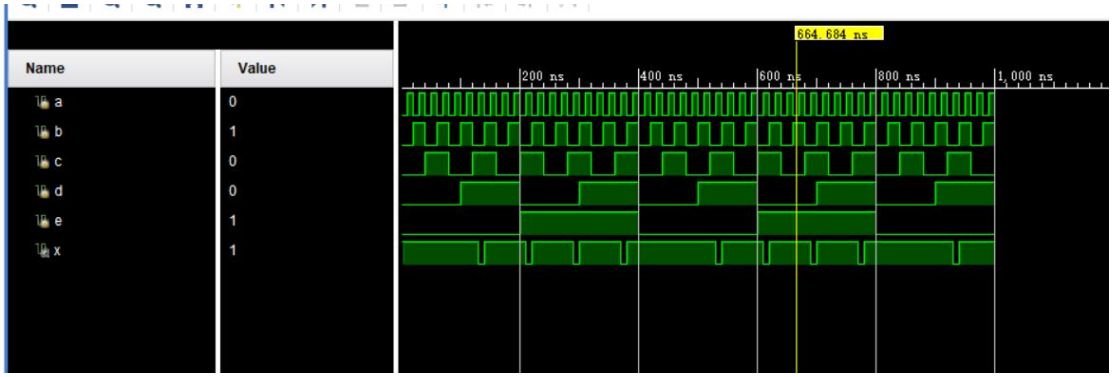
sim_and5:这是一个仿真模块，用于测试 and5 模块。输入端包括 a, b, c, d, 和 e，都被声明为 reg，表示寄存器。输出端 x 声明为 wire，表示一个输出线。and5 模块实例化为 and5，并通过输入 a, b, c, d, e 和输出 x 连接到 sim_and5 模块。在 initial 块中，通过 fork-join 结构模拟不同的输入情况。每个输入信号都以不同的时间间隔（通过 repeat 命令指定）取反，模拟不同的输入值。

二、电路图





三、Vivado 仿真结果



对每一个信号来讲，高电平代表 1，低电平代表 0。容易看出，仿真波形满足 x=~(a·~b· c· (d+e))。

二、复用器

一、Verilog 代码

```
module mux41(
    input D0,input D1,input D2,input D3,
    input [1:0] S,
    output Y
    );
    reg temp;
    always@(*)
    begin
    case(S)
        2'b00 : temp = D0;
        2'b01 : temp = D1;
        2'b10 : temp = D2;
        2'b11 : temp = D3;
        default:temp=D0;
     endcase
     end
     assign Y = temp;
endmodule
```

```
module sim_mux41;
    reg D0,D1,D2,D3;
    reg [1:0] S;
    wire Y;
    mux41 mux(D0,D1,D2,D3,S,Y);
    initial
    begin
        D0=0;D1=0;D2=0;D3=0;S=2'b00;
        fork
            repeat(100) #10 D0=~D0;
            repeat(50) #20 D1 = ~D1;
            repeat(25) #40 D2 = ~D2;
            repeat(10) #100 D3 = ~D3;
            repeat(5) #200 S = S+1;
        join
    end
endmodule
```

第一段代码是一个 4:1 多路复用器（MUX41），第二段代码是一个测试模块（sim_mux41），用于模拟 MUX41 的操作。

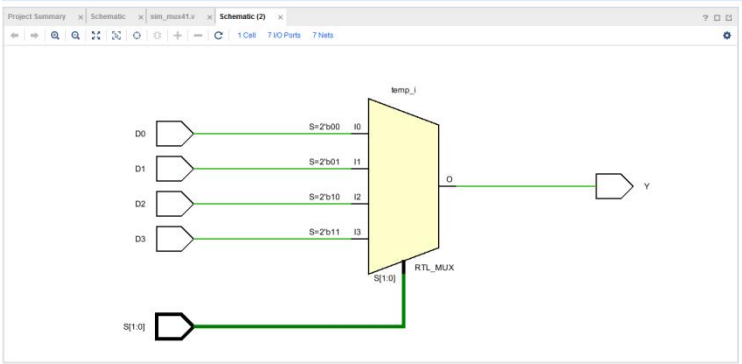
MUX41 模块分析：

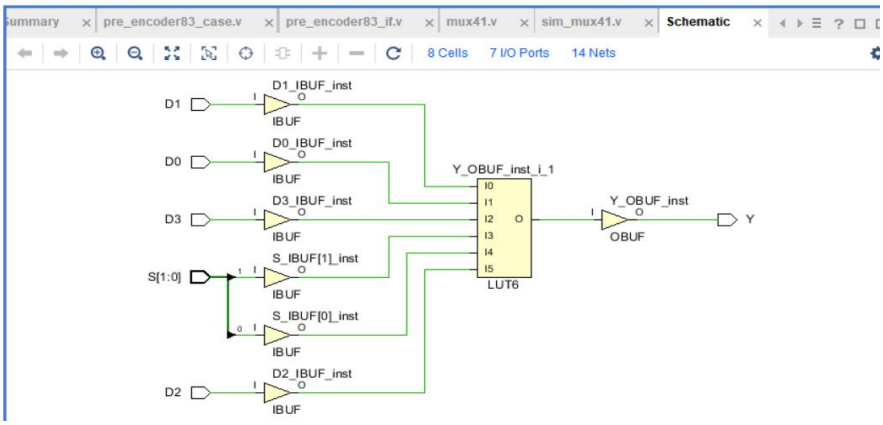
输入为 D0, D1, D2, D3（四个数据输入），S（一个 2 位的选择输入）；输出为 Y（选择输入 S 所指定的其中一个数据输入）；在模块内部，有一个寄存器 temp 用于存储选择的数据；使用 always 块，对输入信号 S 进行检测；使用 case 语句根据 S 的不同取值，将对应的数据输入复制到 temp 中。2'b00 对应 D0，2'b01 对应 D1，2'b10 对应 D2，2'b11 对应 D3；如果 S 不匹配上述任何情况，default 分支将选择 D0；最后，使用 assign 语句将 temp 的值分配给输出端口 Y。

仿真模块分析：

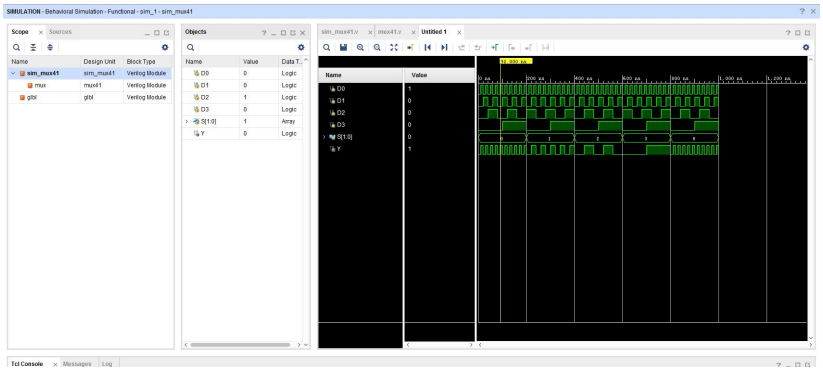
定义了测试所需的信号：D0, D1, D2, D3, S 以及输出信号 Y；创建了一个 MUX41 模块的实例 mux，并将信号连接到它的输入和输出；使用 initial 块，在仿真开始时初始化输入信号的值；使用 fork 和 join 块，对输入信号进行不同的变化和反转，以模拟不同的输入情况；例如，D0, D1, D2, 和 D3 被反转，并且选择信号 S 以不同的频率递增。这个测试模块的目的是模拟 MUX41 的行为，测试不同的输入组合，以确保 MUX41 的功能正常。

二、电路图





三、Vivado 仿真结果



对每一个信号来讲，高电平代表 1，低电平代表 0。容易看出，该仿真波形满足四路复用器的逻辑表达式 Y = (S1 * S0 * D3) + (S1 * !S0 * D2) + (!S1 * S0 * D1) + (!S1 * !S0 * D0)。

三、编码器

一、Verilog 代码

```
module pre_encoder83_case(I,Y);
    input I;
    output Y;
    wire[7:0] I;
    reg[3:1] Y;

    always @(I) begin
        case(I)
            8'b0000_0001 : Y = 3'b000;
            8'b0000_001X : Y = 3'b001;
            8'b0000_01XX : Y = 3'b010;
            8'b0000_1XXX : Y = 3'b011;
            8'b0001_XXXX : Y = 3'b100;
            8'b001X_XXXX : Y = 3'b101;
            8'b01XX_XXXX : Y = 3'b110;
```

```
module encoder83_case(I,Y);
    input I;
    output Y;
    wire[7:0] I;
    reg[2:0] Y;

    always @(I) begin
        case(I)
            8'b0000_0001 : Y = 3'b000;
            8'b0000_0010 : Y = 3'b001;
            8'b0000_0100 : Y = 3'b010;
            8'b0000_1000 : Y = 3'b011;
            8'b0001_0000 : Y = 3'b100;
            8'b0010_0000 : Y = 3'b101;
            8'b0100_0000 : Y = 3'b110;
            8'b1000_0000 : Y = 3'b111;
            default : Y=3'b000;
        endcase
    end
endmodule
```

```
module pre_encoder83_if(I,Y);
    input I;
    output Y;
    wire[7:0] I;
    reg[2:0] Y;

    always @(I) begin
        if(I[7]==1) Y=3'b111;
        else if(I[6]==1)Y=3'b110;
        else if(I[5]==1)Y=3'b101;
        else if(I[4]==1)Y=3'b100;
```

```
module sim_encoder83();
    reg[7:0] x;
    wire[2:0] y_assign,y_case,y_pre_case,y_pre_if;
    integer i;

    initial begin
        x = 1;
        for(i=0;i<7;i=i+1)# 10 x = x*2;
        #10 x =  128;
        while(x>0) #5 x=x-1;
    end

    encoder83_assign encoder83_assign_1(x,y_assign);
    encoder83_case encoder83_case_1(x,y_case);
    pre_encoder83_case pre_encoder_case_1(.I(x),.Y(y_pre_case));
    pre_encoder83_if pre_encoder83_if_1(.I(x),.Y(y_pre_if));
endmodule
```

```
module encoder83_assign(I,Y);
    input I;
    output Y;
    wire[7:0] I;
    wire[2:0] Y;
    assign Y={I[4]|I[5]|I[6]|I[7],I[2]|I[3]|I[6]|I[7],I[1]|I[3]|I[5]|I[7]};
endmodule
```

pre_encoder83_case：实现了一个 3 位的预编码器，根据输入信号 I 的特定模式进行编码，并将编码结果输出到 Y。该模块的作用是将 8 位输入 I 根据不同的模式映射到 3 位的输出 Y。每个模式都对应一个特定的 3 位编码，未匹配的输入将导致 Y 被设置为 3'b000。

pre_encoder83_if：实现了一个 3 位的预编码器。根据输入信号 I 的特定位的值，将其编码为 3 位的输出信号 Y。这个模块的作用是将 8 位输入 I 根据每个位的值映射到 3 位的输出 Y，其中每个位的值对应一个特定的 3 位编码。如果输入的位宽不足 8 位或所有位都为 0，则将 Y 设置为 3'b000。

encoder83_case，它实现了一个 3 位的编码器。根据输入信号 I 的特定模式，将其编码为 3 位的输出信号 Y。这个模块的作用是将 8 位输入 I 根据不同的模式映射到 3 位的输出 Y。每个模式都对应一个特定的 3 位编码，未匹配的输入将导致 Y 被设置为 3'b000。是 pre_encoder83_case 的实例化。

encoder83_assign，它实现了一个 3 位的编码器。根据输入信号 I 的特定位的值，将其编码为 3 位的输出信号 Y，这是通过使用连续赋值 (assign) 语句来实现的。这个模块的作用是将 8 位输入 I 根据不同位的逻辑或运算结果映射到 3 位的输出 Y，其中每一位 Y[2]、Y[1]、Y[0] 的值由 I 的不同位进行逻辑或运算得出。未匹配的输入位将导致相应位的 Y 为 0。**这种方法使用了逻辑运算来实现编码，而不是使用 case 语句，可以有效地减小逻辑复杂性，尤其适用于大型编码器。**

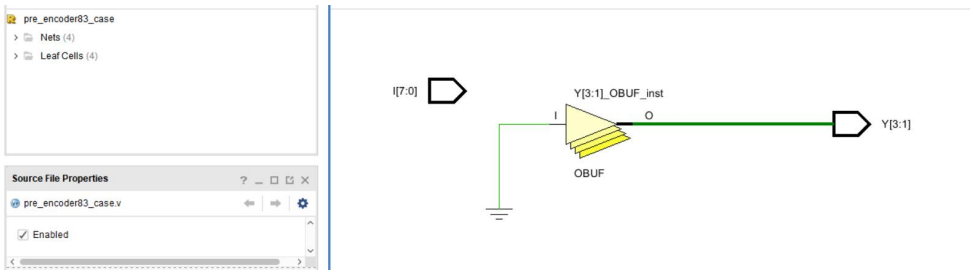
sim_encoder83:此代码段主要用于模拟编码器的功能，以及对 x 进行一系列操作。x 是一个 8 位的寄存器。y_assign、y_case、y_pre_case 和 y_pre_if 都是 3 位宽的线，用于保存编码器的输出。i 是一个整数变量，用于循环中。

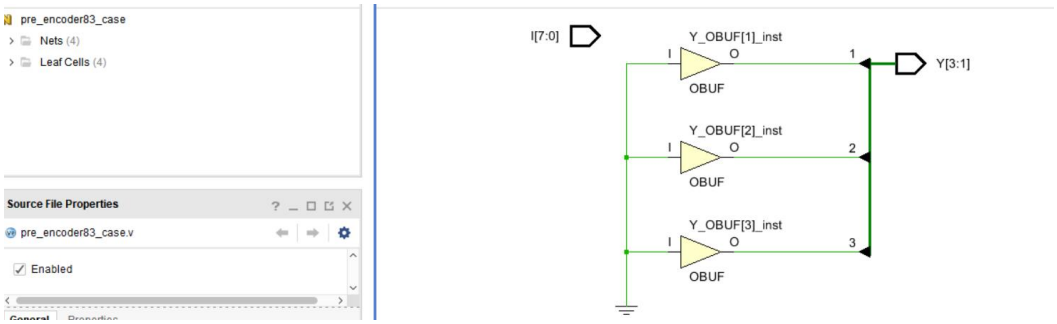
分析循环代码块：首先，x 被设置为 1；然后，有一个 for 循环，从 0 到 6，每次迭代 x 的值翻倍（左移 1 位）；接下来，等待 10 个时间单位；然后，将 x 设置为 128；接着，有一个 while 循环，只要 x 大于 0，就每 5 个时间单位减小 x 的值。最后一

段实例化了四个不同的模块。

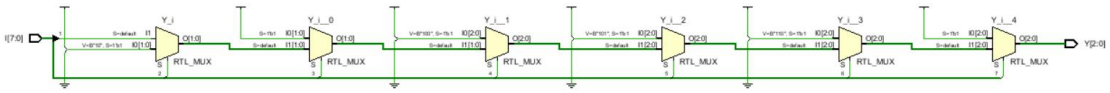
二、电路图

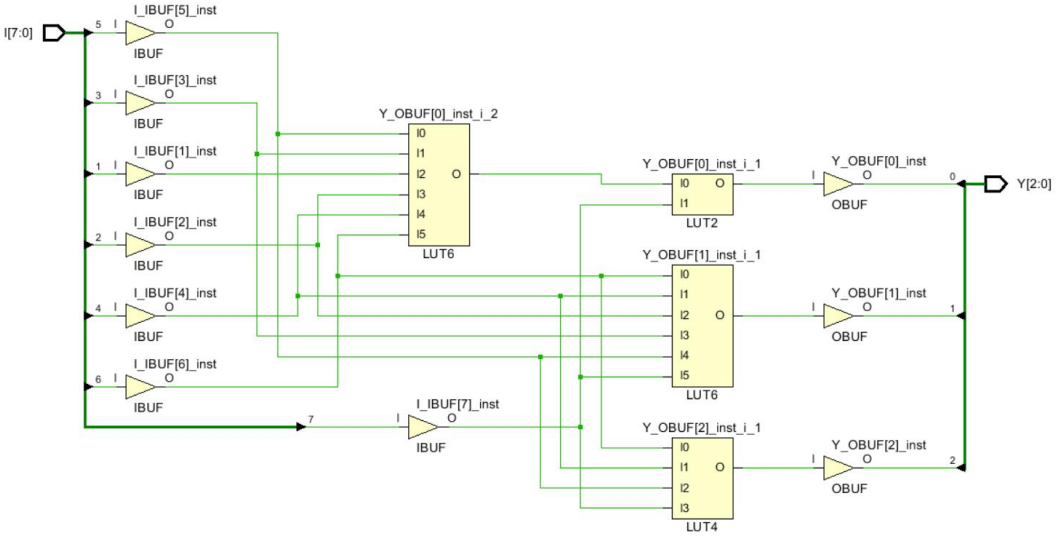
pre_encoder83_case



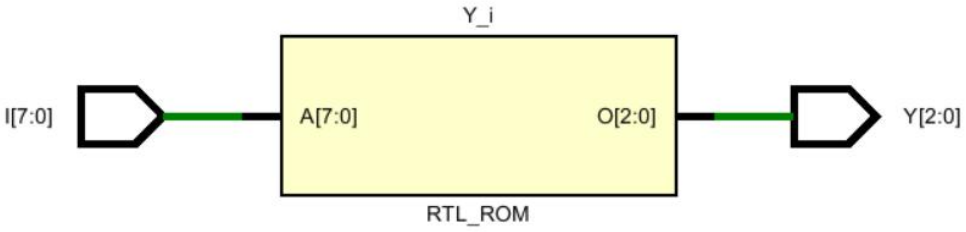


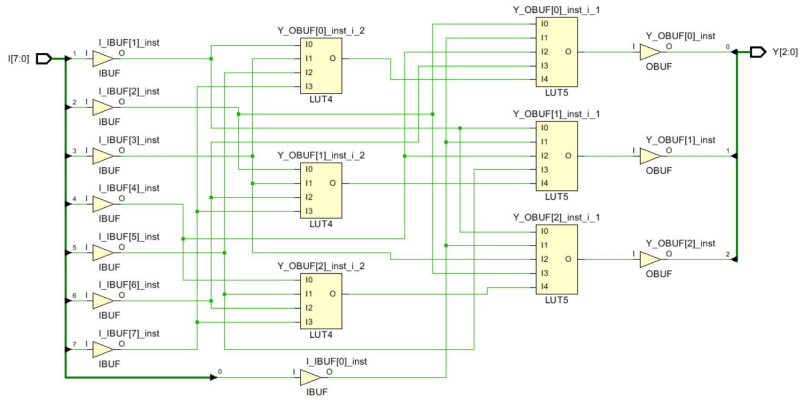
pre_encoder83_if

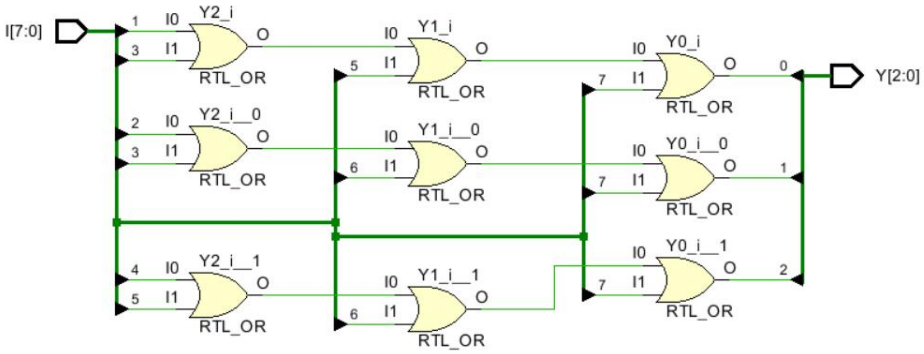




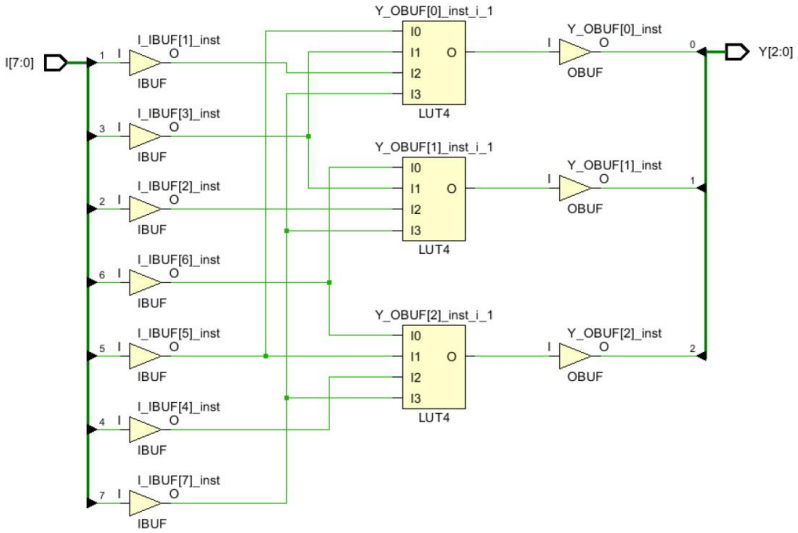
encoder83_case





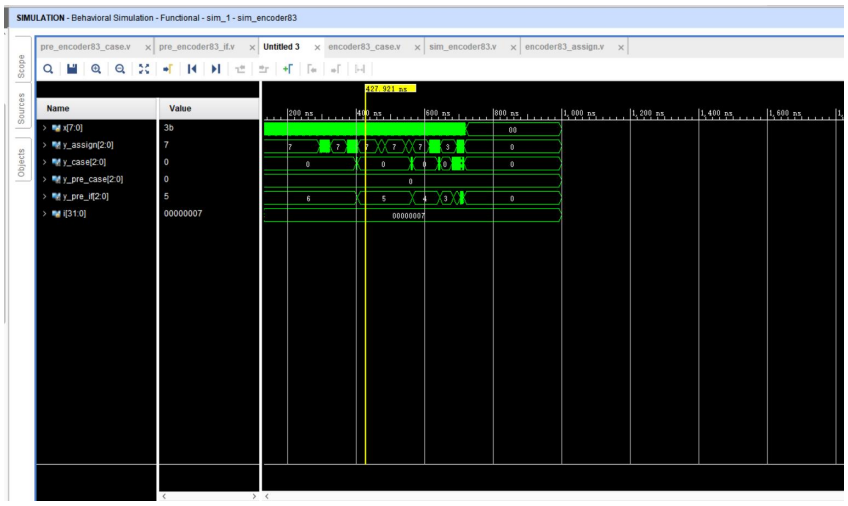


encoder83_assign



三、Vivado 仿真结果

pre_encoder83_case



对每一个信号来讲，高电平代表 1，低电平代表 0。容易看出时序波形满足 Y2=I4+I5+I6+I7、Y1=I2+I3+I6+I7、Y0=I1+I3+I5+I7

四、译码器

一、Verilog 代码

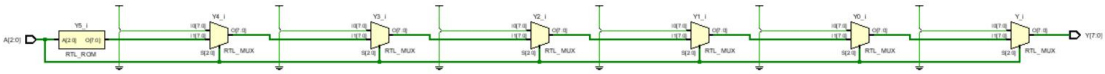
```
module decoder38 (
    input [2:0] A,  // 3位输入信号
    output [7:0] Y  // 8位输出信号
);

assign Y = (A == 3'b000) ? 8'b00000001 :
           (A == 3'b001) ? 8'b00000010 :
           (A == 3'b010) ? 8'b00000100 :
           (A == 3'b011) ? 8'b00001000 :
           (A == 3'b100) ? 8'b00010000 :
           (A == 3'b101) ? 8'b00100000 :
           (A == 3'b110) ? 8'b01000000 :
                           8'b10000000;
endmodule
```

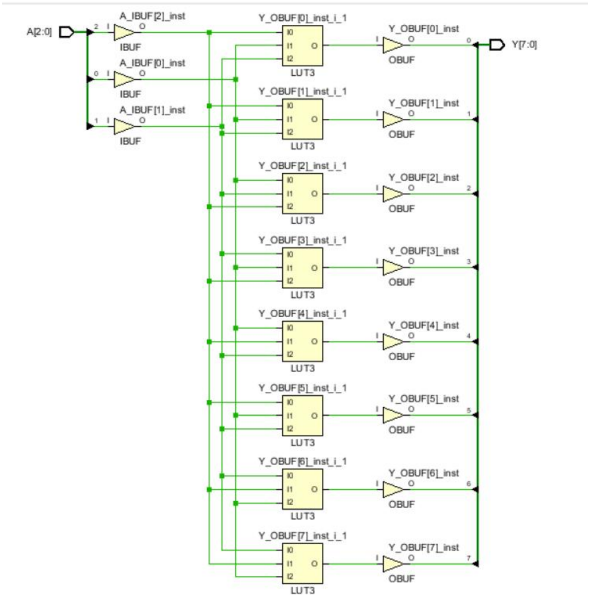
```
module sim_decoder38;
    reg [2:0] A;
    wire [7:0] Y;
    integer i;


    initial begin
        A = 3'b000;
        for (i = 0; i < 7; i = i + 1) begin
            #10 A = A + 1;
        end
    end
      // 实例化 decoder38 模块
      decoder38 decoder(A, Y);
endmodule
```

decoder38:decoder38 模块实现了一个 38 译码器，它有一个 3 位输入信号 A 和一个 8 位输出信号 Y。使用 assign 语句，根据输入信号 A 的值，将对应的 8 位输出模式赋值给 Y。这是一个典型的 38 译码器逻辑，其中每个输入值对应一个特定的输出位。当 A 等于 3'b000 时，Y 的最低位设置为 1，依此类推，当 A 等于其他二进制值时，不同的输出位被设置为 1。

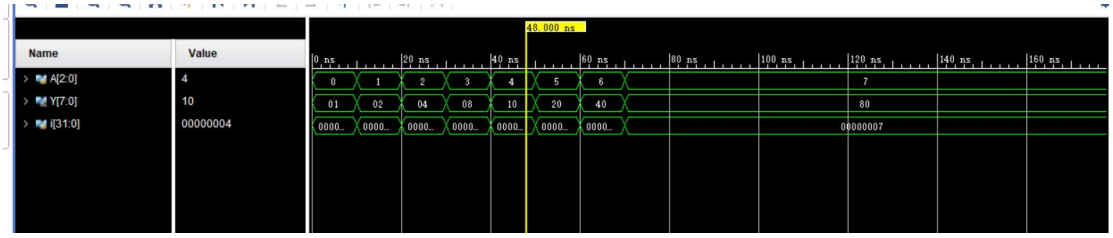
sim_decoder38:sim_decoder38 模块用于模拟 decoder38 模块。定义了输入寄存器 A 和输出线 Y，并使用 integer i 定义了一个整数变量用于循环。在 initial 块内，首先将 A 初始化为 3'b000。接下来，使用 for 循环递增 A 的值。

二、电路图





三、Vivado 仿真结果



对每一个信号来讲，高电平代表 1，低电平代表 0。容易看出，时序波形满足 D0 = A’ * B’ * C’、D1 = A * B’ * C’、D2 = A’ * B * C’、D3 = A * B * C’、D4 = A’ * B’ * C、D5 = A * B’ * C、D6 = A’ * B * C、D7 = A * B * C。

五、调试和心得体会

本次实验中，我复习了 Vivado 软件的操作，包括电路的设计和测试方法；电路图如何生成；Vivado 仿真时序图如何生成；学会了分析 Vivado 仿真波形。在本次实验中，我采用

Vivado 来实现，PPT 只有两个要求电路给出了代码，其余两个并没有给出，需要自己来编写，这个过程我通过和同学讨论，与老师交流，完成了代码的编写工作，受益匪浅。