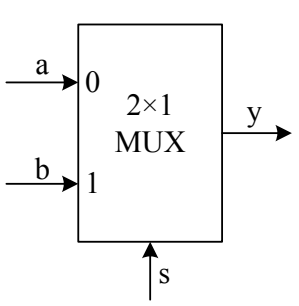
实验一　选择器

2018 年秋季学期

选择器是数字逻辑系统的常用电路，是组合逻辑电路中的主要组成元件之一，它是由几路数据输入、一位或多位的选择控制端，和一路数据输出所组成的。多路选择器从多路输入中，选取其中一路将其传送到输出端，由选择控制信号决定输出的是第几路输入信号。

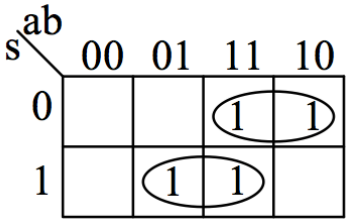
本次实验将介绍几种常用的多路选择器的设计方法；Verilog 语言中的 always 语句块、if-else 语句和 case 语句的使用等。最后请读者自行设计一个多路选择器，熟悉电路设计的基本流程和 Quartus 的使用。

1.1　2 选 1 多路选择器

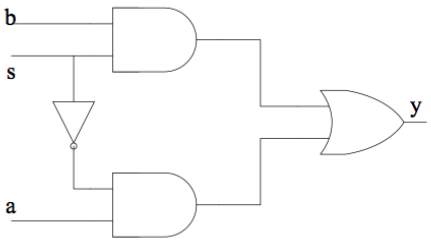


s	a	b
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

(a) 2 选 1 选择器模块和真值表



(b) 2 选 1 选择器的卡诺图



(c) 2 选 1 选择器的逻辑电路

图 1-1: 2 选 1 选择器

图 1-1(a)是 2 选 1 选择器的模块图和真值表，图中 *a* 和 *b* 为输入端；*y* 为输出端，*s* 是选择端，选择两个输入的其中一个输出。当 *s* 为 0 时，*y* 的输出值

为 a。当 s 为 1 时，y 的输出值为 b。

图 1-1(b)是 2 选 1 选择器的卡诺图，根据卡诺图可以得出 2 选 1 选择器的的表达式为 $y = (\sim s\&a)|(s\&b)$。根据表达式画出其逻辑电路如图 1-1(c)所示。

根据图 1-1(c)的逻辑电路图，利用 Verilog HDL 实现 2 选 1 选择器的逻辑电路，如表 1-1所示。

表 1-1: 2 选 1 选择器激励代码

```
module mux21(a,b,s,y);
  input   a,b,s;          // 声明3个wire型输入变量a,b,和s，其宽度为1位。
  output  y;             // 声明1个wire型输出变量y，其宽度为1位。

  assign  y = (~s&a)|(s&b);  // 实现电路的逻辑功能。

endmodule
```

为此工程设计一个激励代码，对设计的选择器的功能进行仿真。激励代码如表 1-2所示。

表 1-2: 2 选 1 选择器激励代码

```
‘timescale 10 ns/ 1 ps      // 设置时间尺度和时间精度
module mux21_vlg_tst();     // 测试代码的端口参数列表为空
       reg a;               // 输入变量声明为reg型变量
       reg b;
       reg s;
       wire y;              // 输出变量声明为wire型变量

  mux21 i1 (                 // 对要测试的模块进行实例化
            .a(a),
            .b(b),
            .s(s),
            .y(y));

  initial    begin          //  初始化输入变量
     s=0; a=0; b=0; #10;    // 将s，a和b均初始化为“0”，维持10个时间单位
               b=1; #10;    // 将b改为“1”，s和a的值不变，继续保持“0”，
                            // 维持10个时间单位 即10*10ns=100ns
          a=1; b=0; #10;    // 将a，b分别改为“1”和“0”，s的值不变，
                            // 继续保持“0”，维持10个时间单位
               b=1; #10;    // 将b改为“1”，s和a的值不变，维持10个时间单位
     s=1; a=0; b=0; #10;    // 将s，a，b分别变为“1,0,0”，维持10个时间单位
               b=1; #10;
          a=1; b=0; #10;
               b=1; #10;
  end
endmodule
```

上述代码分析与综合后的仿真结果如图 1-2所示，由图中可以看出，当 $s = 0$ 时，$y = a$，即 y 随着 a 值的改变而改变，此时的 b 值无论如何改变都不影响 y 的值。当 $s = 1$ 时，$y = b$，即 y 随着 b 值的改变而改变，此时的 a 值无论如何改变都不影响 y 的值。

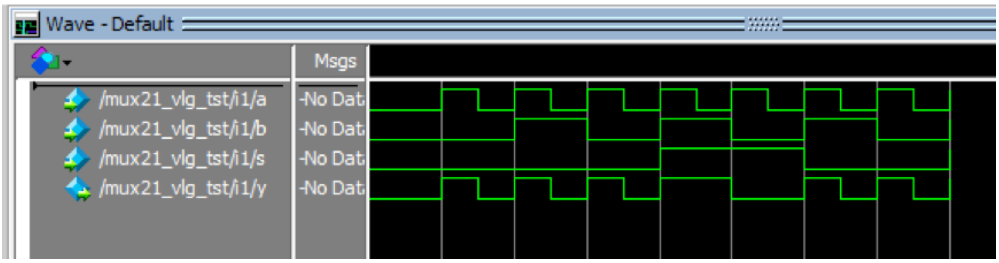


图 1-2: 2 选 1 选择器仿真结果

2 选 1 多路选择器的行为在 Verilog 中也可以用 if 语句来实现这一行为。用 if 语句来设计选择器的程序清单如表 1-3所示：

表 1-3: 2 选 1 选择器 if 语句实现

```
1  module mux21b(a,b,s,y);
2    input   a,b,s;
3    output reg  y;   // y在always块中被赋值，一定要声明为reg型的变量
4
5    always @ (*)
6      if(s==0)
7         y = a;
8      else
9         y = b;
10 endmodule
```

在 Verilog 中，各语句是并发执行的，模块中所有的 assign 语句、always 语句块和实例化语句，其执行顺序不分先后。而 if 语句是顺序执行的语句，其执行过程中必须先判断 if 后的条件，如果满足条件则执行 if 后的语句，否则执行 else 后的语句。Verilog 语法规定，顺序执行的语句必须包含中 always 块中，always 块中的语句按照它们中代码中出现的顺序执行。

☞ **always 语句块的使用**

always 块的语句格式如下：

always @(< 敏感事件列表 >)

各可执行的语句；

……

其中敏感事件列表中列出了所有影响 always 块中输出的信号清单，也就是说，如果敏感事件列表中的任何一个变量发生了变化，都要执行 always 语句块中的语句。如 always @ (a or b or s) 表示：只要 a、b、s 中的任何一个变量发生了变化，就立刻执行 always 语句块中的语句。

为了方便起见，敏感列表也可以用“*”代替，如 always @ (*)，(*) 号将自动包含 always 语句块中右边的语句或条件表达式中的所有信号。如表 1-3，只要 always 语句块中表达式右边出现的变量 a 和 b，或者条件表达式中出现的变量 s，这三个变量中的任何一个变量发生了变化，就立刻执行 always 语句块中的语句。

always 语句还有另外一种形式，即：always 后面不带任何有关敏感事件列表的信息，只有“always”这个保留字，那么这个时候表明在任何情况下都执行 always 语句块中的语句。

另外，always 块中的输出信号必须被描述成 reg 型，而不是默认为 wire 型。

☞ 关于 if 语句

if 语句是 Verilog HDL 中常用的条件语句，和 else 语句配对使用。当然，if 语句也可以不和 else 语句配对单独使用。

但是，如果 if 语句在使用时没有 else 语句与其配对则会发生这样的情况：编译器判断 if 后面的条件表达式是否满足，如果满足则执行其后的语句，那如果条件表达式不满足呢？这时，编译器就会自动产生一个寄存器来寄存当前的值，在条件不满足时保输出的过去值。这样就会产生用户没有设计的多余的寄存器出来。因此建议读者在使用 if 语句的时候要加上 else 语句与其配对。防止产生多余的寄存器。

另外，编译器默认 if 语句的功能语句只有一条，如果有多条功能语句，要把这些语句用关键词“begin”和“end”将其“括”起来。如：

```
1 if(s==0)
2    y = a;  x = b;
3 else
4    y = b; x = a;
```

是错误的写法，应改为:

```
1 if(s==0)
```

```
  begin    y = a;   x = b;   end
else
  begin   y = b;    x = a;   end
```

在使用中也可以用条件判断语句代替 if 语句，如果此时不用顺序语句就不需要 always 语句块，比如也可以使用“?”来代替 if 语句，其用法如下：

```
assign y = s ? b : a;
```

其含义如下：如果 s = 1，那么 y = b；否则 y = a。则此 2 选 1 选择器代码可另写如下：

表 1-4: 2 选 1 选择器 assign 语句实现

```
module mux21c(a,b,s,y);
  input    a,b,s;
  output   y;    // y不用声明为reg型的了。

  assign   y = s ? b : a;

endmodule
```

1.2 4 选 1 多路选择器

4 选 1 多路选择器的模块图和真值表如图 1-3所示，$a_0 - a_3$ 为 4 个输入端，s_0 和 s_1 是选择端，y 是输出端，根据 s_0 和 s_1 值的不同，y 选择 $a_0 - a_3$ 中的一个输出，具体请见真值表。

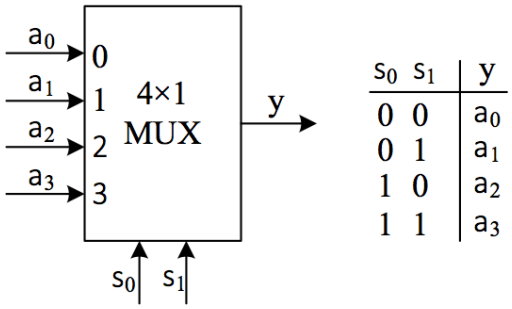


图 1-3: 4 选 1 选择器模块及真值表

Verilog 语言中的 case 语句可以综合出“多路复用器”的电路，它的可读性非常强。表 1-5所示的是用 case 语句实现 4 选 1 多路选择器的方法。

上述设计的测试代码如表 1-6所示：

表 1-5: 4 选 1 选择器 case 语句实现

```
module mux41(a,s,y);
  input  [3:0] a;  // 声明一个wire型输入变量a，其变量宽度是4位的。
  input  [1:0] s;  // 声明一个wire型输入变量s，其变量宽度是2位的。
  output reg y;   // 声明一个1位reg型的输出变量y。

  always @ (s or a)
    case (s)
      0: y = a[0];
      1: y = a[1];
      2: y = a[2];
      3: y = a[3];
      default: y = 1'b0;
    endcase

endmodule
```

表 1-6: 4 选 1 选择器激励代码

```
`timescale 10 ns/ 1 ps
module mux41_vlg_tst();
       reg [3:0] a;
       reg [1:0] s;
       wire y;

  mux41 i1 (
                .a(a),
                .s(s),
                .y(y));
   initial    begin
     s=2'b00; a=4'b1110; #10;
              a=4'b0001; #10;
     s=2'b01; a=4'b1110; #10;
              a=4'b0010; #10;
     s=2'b10; a=4'b1010; #10;
              a=4'b0100; #10;
     s=2'b11; a=4'b0111; #10;
              a=4'b1001; #10;
   end
endmodule
```

表 1-5中的程序的仿真图如图 1-4所示。

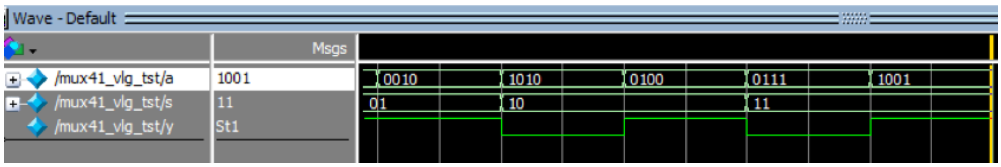


图 1-4: 4 选 1 选择器仿真图

☞ **case 语句的使用**

case 语句是以关键字 case 和一个被括起来的“选择表达式”开头，表达式的结果表示一个整数。下面是 case 选项，每个选项由选择列表和过程语句构成，选择列表可以是一个整数值，也可以是多个整数值，多个整数值之间以逗号分开，选择列表和过程语句之间以冒号连接，如：`0,1: y = a[0];`。

case 语句的执行过程是这样的：先计算出选择表达式的值，在 case 选项中找到和选择表达式值相同的第一个选择，然后执行此选择值后面的过程语句。

case 语句列出的选择列表，有时候不能全部包含选择表达式所有的可能值，这时关键词 default 就要被作为 case 语句的最后一个选项，它表示表达式中那些未被选择列表覆盖的所有其他值。一般情况下即使选择列表列出了选择表达式的所有选项，还是建议保留 default 这一选项。如果选择列表中没有包含选择表达式的所有选项，而此时又没有 default 选项的话，综合器会综合出一个锁存器以保存未被覆盖的情况下输出的过去值。这一般是不希望出现的情况，所以在 case 语句中建议无论如何保留 default 选项。

如果在满足某个表达式值时要执行多条语句，也要用关键词“begin”和“end”将这些语句其“括”起来。

1.3 实验内容

2 位 4 选 1 选择器

用 case 语句实现一个 2 位 4 选 1 的选择器，如图 1-5所示，选择器有 5 个 2 位输入端，分别为 X0, X1, X2, X3 和 Y，输出端为 F；X0, X1, X2, X3 是四个 2 位的输入变量。输出 F 端受控制端 Y 的控制，选择其中的一个 X 输出，当 Y = 00 时，输出端输出 X0，即 F = X0；当 Y = 01 时，输出端输出 X1，即 F = X1；以此类推。

选择开发板上的 SW0 和 SW1 作为控制端 Y，SW2—SW9 作为八位数据输入端 X0–X3，将两位的输出端 F 接到发光二极管 LEDR0 和 LEDR1 上显示

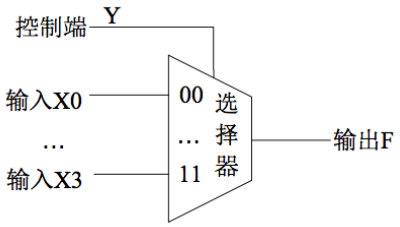


图 1-5: 2 位 4 选 1 选择器

输出，完成设计，对自己的设计进行功能仿真，并下载到开发板上验证电路性能。