



2022

西安交通大学 信通学院

电子技术实验2

张翠翠

zhangcuicui@mail.xjtu.edu.cn

4 Verilog语法基础



◆软件安装

◆仿真验证

- 文件夹路径混乱
- 数字开头或含有非法字符的路径导致仿真报错
- 激励供给不完备

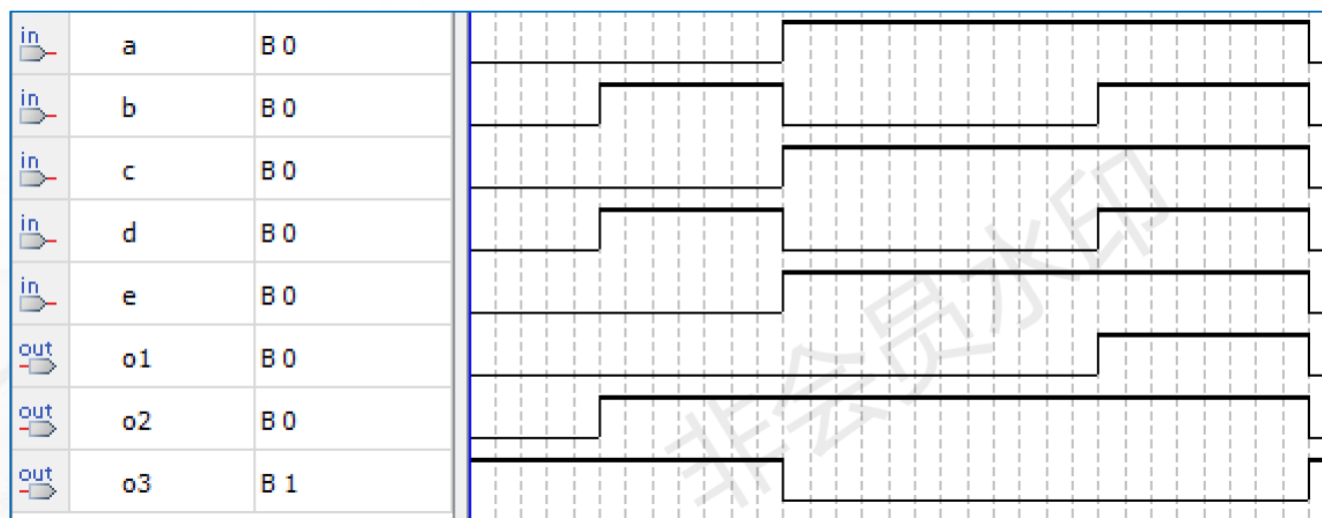
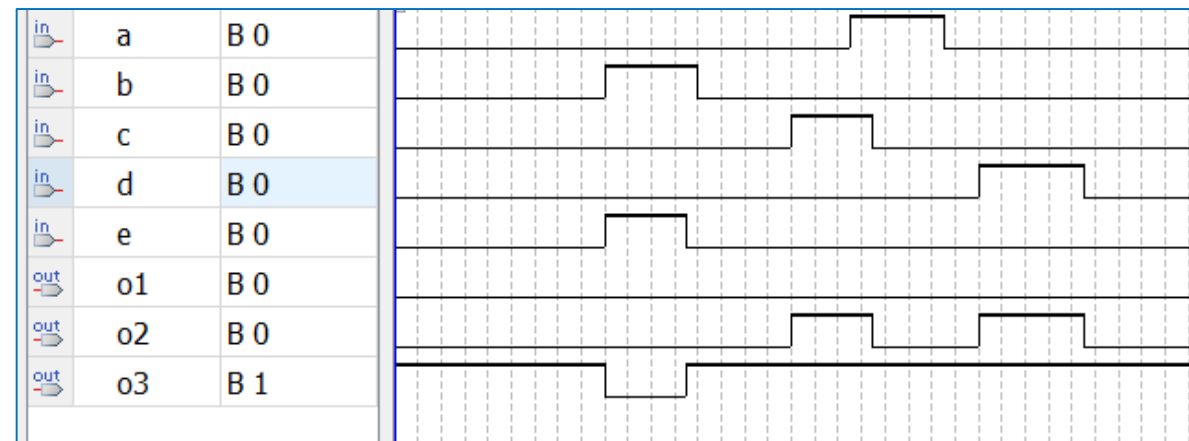
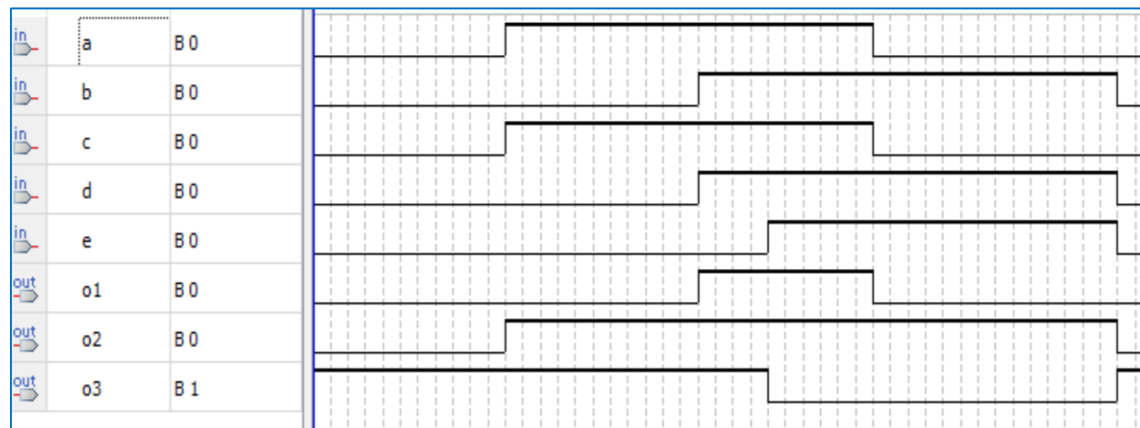
➤ 避免机械式的操作

➤ 缺乏自主思考

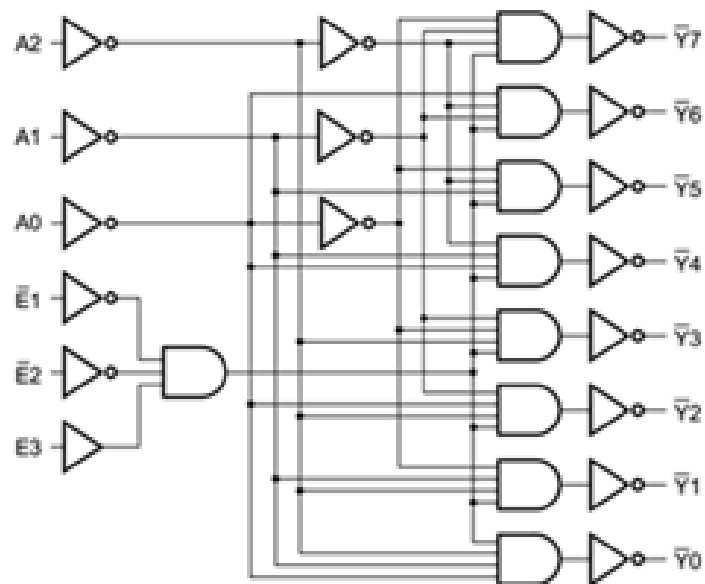
✓ 需要对仿真结果和软件操作进行总结和思考



上节实验总结



与或非逻辑的仿真结果示例

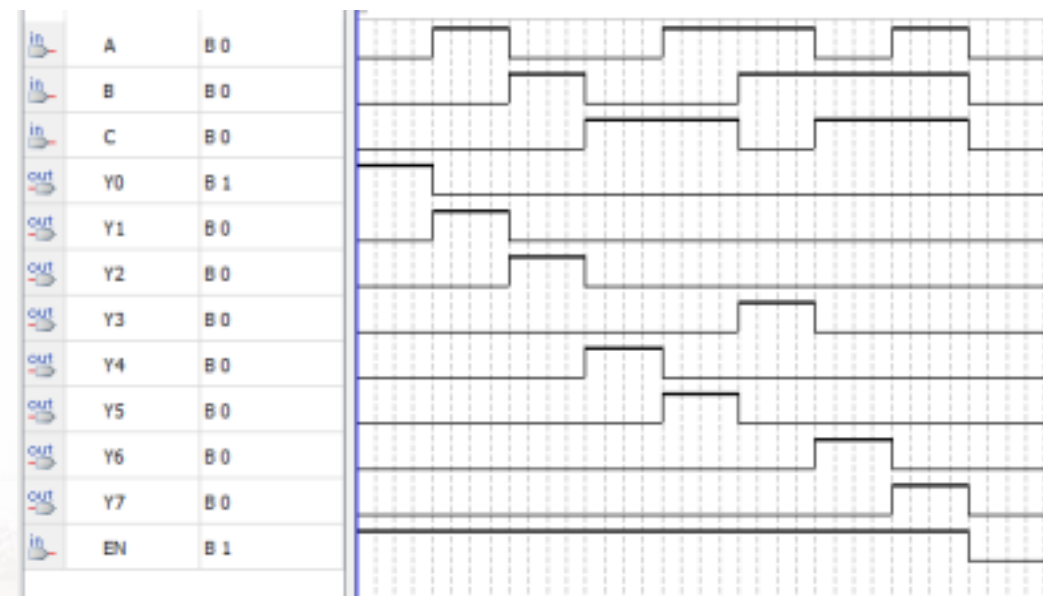
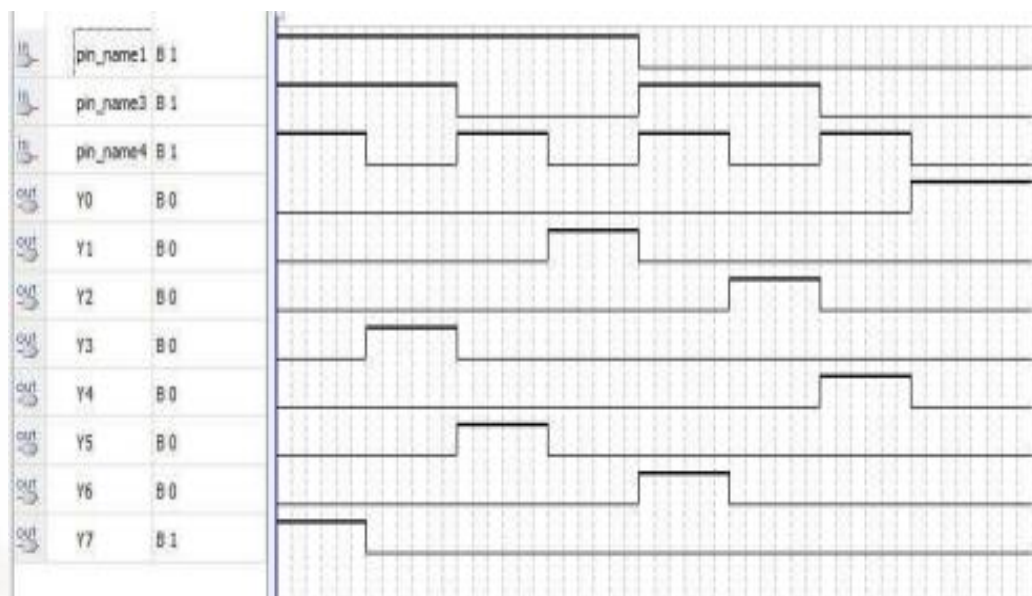


使能端			输入			输出							
$\bar{E}1$	$\bar{E}2$	$E3$	A2	A1	A0	$\bar{Y}7$	$\bar{Y}6$	$\bar{Y}5$	$\bar{Y}4$	$\bar{Y}3$	$\bar{Y}2$	$\bar{Y}1$	$\bar{Y}0$
H	L	H	x	x	x	H	H	H	H	H	H	H	H
x	H	x	x	x	x	H	H	H	H	H	H	H	H
x	x	L	x	x	x	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	L
			L	L	H	H	H	H	H	H	H	L	H
			L	H	L	H	H	H	H	H	L	H	H
			L	H	H	H	H	H	H	L	H	H	H
			H	L	L	H	H	H	L	H	H	H	H
			H	L	H	H	H	L	H	H	H	H	H
			H	H	L	H	L	H	H	H	H	H	H
			H	H	H	L	H	H	H	H	H	H	H

38译码器的电路原理和真值表



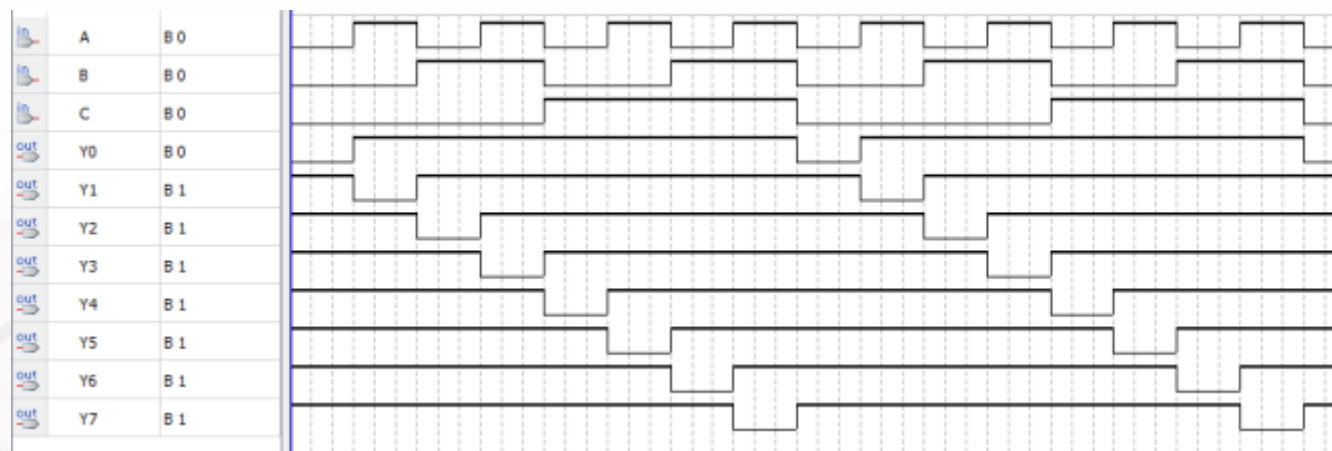
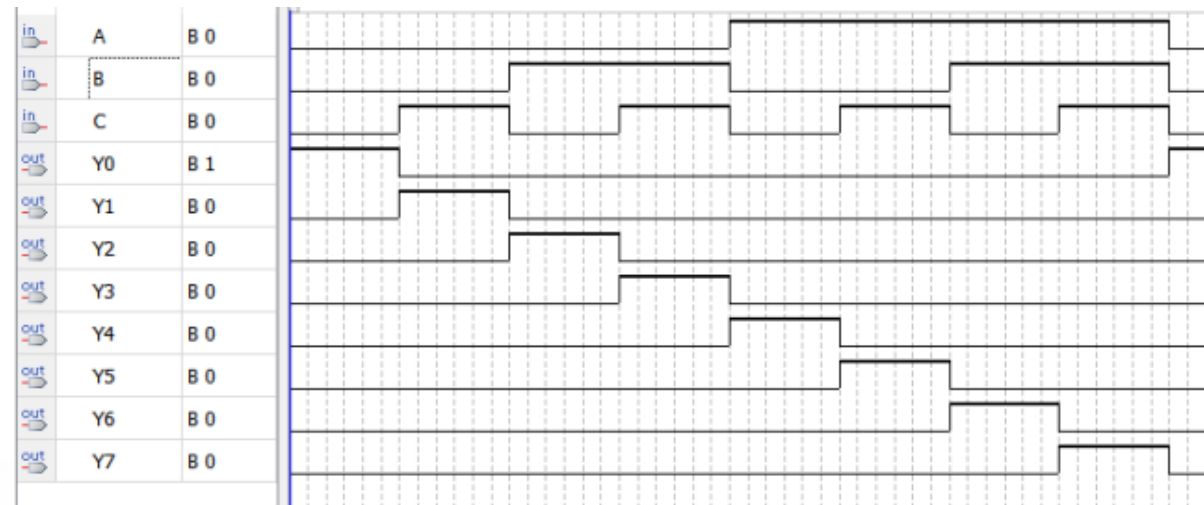
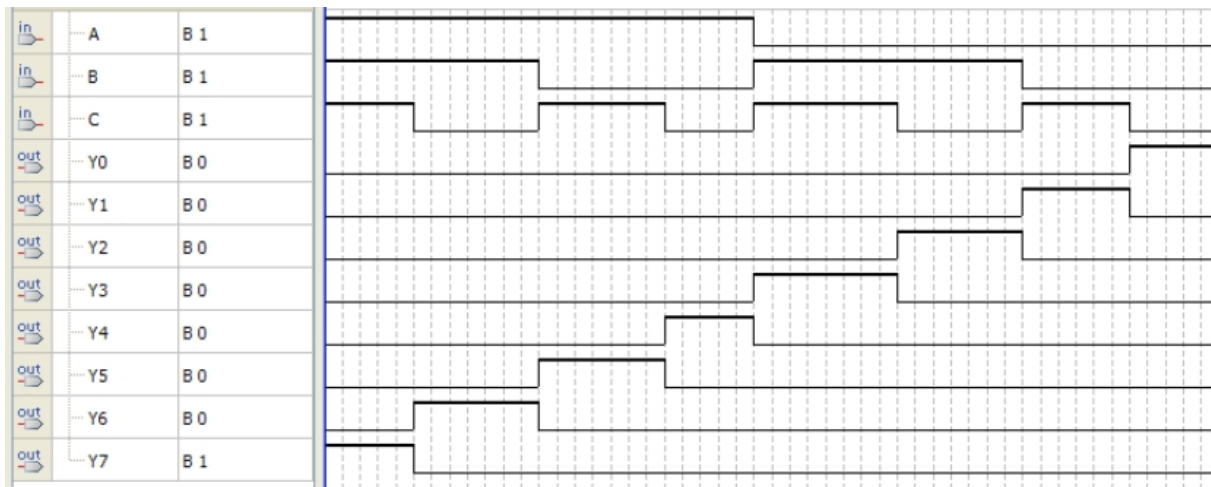
上节实验总结



38译码器的仿真结果示例



上节实验总结



38译码器的仿真结果示例



C 目录

CONTENTS

忠 果 敦 精
恕 毅 笃 勤
任 力 励 求
事 行 志 学

01

Verilog语法基础

02

实验内容

03

实验报告要求

04

下一次实验内容



Part 01

Verilog语法基础

- 电路单元module
- 数据类型
- Verilog操作符
- 过程块和过程赋值
- 条件语句



◆一个设计实体中包含哪些要素？

- ✓ module 模块定义
- ✓ 模块的端口定义
- ✓ 模块的内部逻辑实现

```
module test(  
    input a,  
    input b,  
    input c,  
    input d,  
    input e,  
  
    output o1,  
    output o2,  
    output o3  
);  
  
assign o1 = a&b;  
assign o2 = c|d;  
assign o3 = ~e;  
  
endmoudle
```



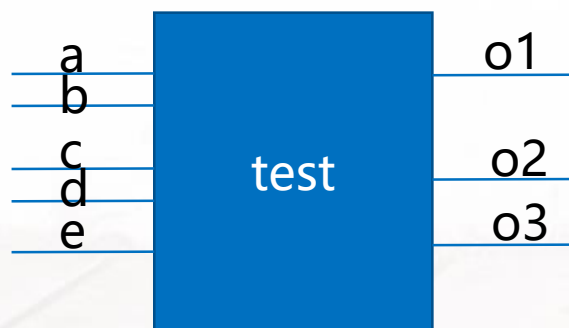
◆ 模块 (module)

- **module**能够表示：
 - 物理块，如**IC**或**ASIC**单元
 - 逻辑块，如一个**CPU**设计的**ALU**部分
 - 整个系统
- 每一个模块的描述从关键词**module**开始，有一个名称（如**SN74LS74**，**DFF**，**ALU**等等），由关键词**endmodule**结束。

```
module test(  
    input a, b, c, d, e,  
    output o1, o2, o3  
);  
assign o1 = a&b;  
assign o2 = c|d;  
assign o3 = ~e;  
endmoudle
```

◆ 模块端口

- ✓ 端口定义了该模块的输入输出引脚pin
- ✓ 模块通过端口与外部通信



端口在模块
后的括号中
列出

```
module test(a,b,c,d,e,o1,o2,o3);  
  input a, b, c, d, e;  
  output o1, o2, o3;  
  assign o1 = a&b;  
  assign o2 = c|d;  
  assign o3 = ~e;  
endmodule
```

端口有三种类型：
input、
output、inout

◆ 模块实例化

- 一个模块中可以包含其它模块，在一个模块中通过模块实例化来调用另一个模块。

```
module test(  
    input a, b, c, d, e,  
    output o1, o2, o3  
);  
    and u1(o1,a,b);  
    or u2(o2,c,d);  
    not u3(o3,e);  
endmodule
```

模块实例化

- ✓ 每个实例都有自己的名字(u1, u2, u3)。实例名是每个对象唯一的标记，通过这个标记可以查看每个实例的内部。
- ✓ 实例中端口的次序与模块定义的次序相同。
- ✓ 模块实例化与调用程序不同。每个实例都是模块的一个完全的拷贝，相互独立、并行。

◆ Verilog采用四值逻辑系统

0（低电平）

1（高电平）

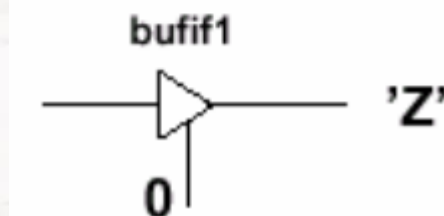
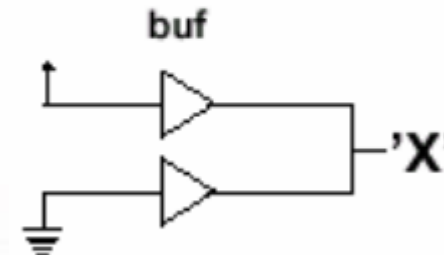
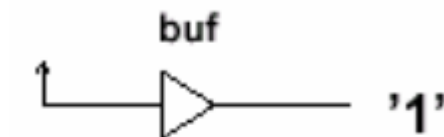
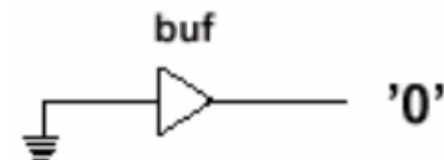
X——不确定的值

Z——高阻态

◆ Verilog中常数的表示

◆ Verilog主要有三类数据类型

- ✓ net(线网): 表示器件之间的物理连接
- ✓ register(寄存器): 表示抽象存储元件
- ✓ parameters(参数): 运行时的常数



◆ Verilog中常数的表示——整数 和 实数

- 整数表示为：<size>'<base><value>
 - size**：表示占用的二进制位宽bit。缺省为32位
 - base**：数基，可为2进制(b)、8进制(o)、10进制(d)、16进制(h)
缺省为10进制
 - value**：是所选数基内任意有效数字，包括X、Z。
- 当数值value大于指定的大小时，截去高位。如 2'b1101表示的是2'b01

8'd12	8表示8bit，d表示是十进制。 该例表示值12
12'H83A	12表示12bit，H表示是十六进制。 该例表示值为16进制的83A
4'b0111	4表示4bit，b表示二进制。 该例表示二进制0111，即7。

◆ Verilog中常数的表示——整数 和 实数

- ✓ 实数可用科学表示法或十进制表示
- ✓ 科学表示法表示方式：

<尾数> <**e或E**> <指数>， 表示： 尾数×**10**指数

例： 32E-4表示0.0032；

4.1e3表示4100.



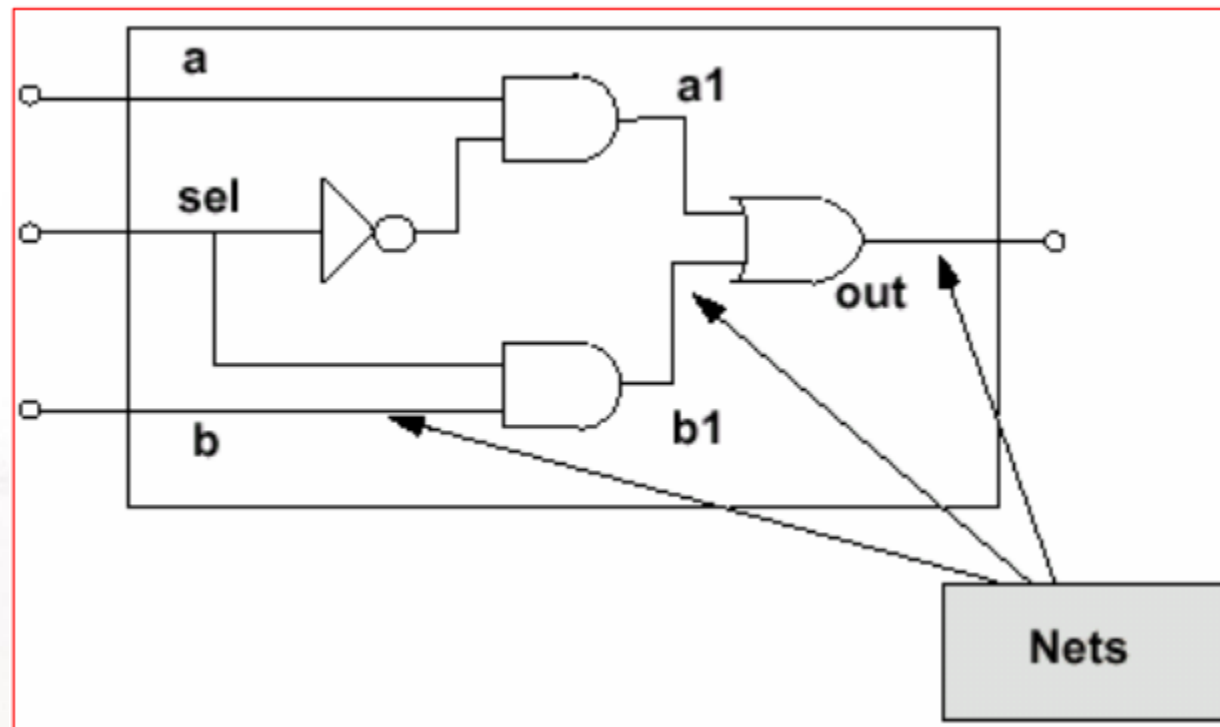
◆ 参数

- 用参数声明一个可变常量，常用于定义延时及宽度变量。
- 参数定义的语法：**parameter <list_of_assignment>;**
可一次定义多个参数，用逗号隔开。

```
module mod1( out, in1, in2);  
...  
parameter IDLE = 4'b0000;  
endmodule
```


◆ net线网类

- 物理连接或者叫连线，驱动它的可以是门和模块
- 不具有记忆性
- 由持续赋值语句assign赋值



◆ net线网类



net类型	功 能
wire, tri	标准内部连接线(缺省)
supply1, supply0	电源和地
wor, *trior	多驱动源线或
wand, *triand	多驱动源线与
*triereg	能保存电荷的net
*tri1, tri0	无驱动时上拉/下拉

- **wire**类型是最常用的类型，只有连接功能
 wire [31:0] w1, w2; // w1和w2是32bit的wire类型
 wire a, b; //a和b是1bit的wire类型
- 没有声明的缺省类型为 **1 位(标量)wire**类型
 assign **out1**=a&b; //out1若之前没有被声明，则默认为1bit的wire型



```

module test(
    input a,
    input b,
    input c,
    input d,
    input e,

    output o1,
    output o2,
    output o3
);

assign o1 = a&b;
assign o2 = c|d;
assign o3 = ~e;

endmoudle
  
```

```

module test(
    input a,
    input b,
    input c,
    input d,
    input e,

    output wire o1,
    output wire o2,
    output wire o3
);

assign o1 = a&b;
assign o2 = c|d;
assign o3 = ~e;

endmoudle
  
```

```

module test(a,b,c,d,e,o1,o2,o3);
    input a, b, c, d, e;
    output o1, o2, o3;

    wire o1,o2,o3;

    assign o1 = a&b;
    assign o2 = c|d;
    assign o3 = ~e;
endmoudle
  
```

该设计中可以省略；
但当o1、o2、o3不是
1bit wire时，或者是
别的数据类型时必须
要声明

```

module test(a,b,c,d,e,o1,o2,o3);
    input a, b, c, d, e;
    output [3:0] o1;
    output o2, o3;
    wire [3:0] o1;
    reg o2,o3;
    //逻辑表达
endmoudle
  
```

◆ register寄存器类

- 寄存器变量由关键字reg定义，常代表触发器的输出
- 在赋新值前保持旧值不变
- 在always块中使用过程赋值改变其值
- always块内被赋值的每一个信号都必须声明为reg型

◆ register声明

```
reg a; //1bit寄存器类变量a
```

```
reg [3: 0] v; // 4位寄存器变量v
```

```
reg [7: 0] m, n; // 两个8位寄存器变量m和n
```

◆ register寄存器类

寄存器类型	功能
reg	可定义无符号整数变量，可以是标量(1位)或矢量，是最常用的寄存器类型
integer	32位有符号整数变量，算术操作产生二进制补码形式的结果。通常用作不会由硬件实现的数据处理。
real	双精度的带符号浮点变量，用法与integer相同。
time	64位无符号整数变量，用于仿真时间的保存与处理
realtime	与real内容一致，但可以用作实数仿真时间的保存与处理



◆ 选择正确的数据类型

- 信号分为端口信号和内部信号。出现在端口列表中的是端口信号，其它为内部信号
- 输入端口只能是线网类型（net）
- 输入端口的驱动可以是线网类型或寄存器类型（net or register）
- 输出端口可以是线网类型也可以是寄存器类型（net or register）
- 过程块内赋值的为register类型，过程块外赋值的为net类型
- 过程块中的赋值（过程赋值）只能给register型赋值



操作符类型	符号
连接及复制操作符	{ } { }
一元操作符	! ~ & ^
算术操作符	* / % + -
逻辑移位操作符	<< >>
关系操作符	> < >= <=
相等操作符	== === != !==
按位操作符	& ^ ~^
逻辑操作符	 &&
条件操作符	? :

最高

优先级

最低



◆ 过程(procedural)块

- 过程块有两种：
 - **initial**块，只能执行一次。**不能综合，用在仿真中**
 - **always**块，**循环**执行
- 过程块中有下列部件
 - 过程赋值语句：在描述过程块中的数据流
 - 高级结构（循环，条件语句）：描述块的功能
 - 时序控制：控制块的执行及块中的语句



◆ 过程赋值(procedural assignment)

- 在过程块中的赋值称为**过程赋值**
- 被赋值的信号必须是寄存器类型（如reg类型）
- 赋值语句右边可以是任何有效的表达式
- 没有声明的信号缺省为wire类型。使用过程赋值给wire赋值会产生错误。

```
module adder (out, a, b, cin);  
    input a, b, cin;  
    output [1:0] out;  
    wire a, b, cin;  
    reg half_sum;  
    reg [1: 0] out;  
    always @( a or b or cin)  
    begin  
        half_sum = a ^ b ^ cin ; // OK  
        half_carry = a & b | a & !b & cin | !a & b & cin ; //  
        ERROR!  
        out = {half_carry, half_sum} ;  
    end  
endmodule
```

half_carry
没有声明

◆ 边沿敏感时序

- 时序控制@可以用在RTL级或行为级组合逻辑或时序逻辑描述中。
- 可以用关键字**posedge**和**negedge**限定信号敏感边沿。敏感表中可以有多个信号，用关键字**or**连接。

```
module reg_adder (out, a, b, clk);  
    input clk;  
    input [2: 0] a, b;  
    output [3: 0] out;  
    reg [3: 0] out;  
    reg [3: 0] sum;  
    always @( a or b) // 若a或b发生任何变化，执行  
        #5 sum = a + b;  
    always @( negedge clk) // 在clk下降沿执行  
        out = sum;  
endmodule
```

◆ 边沿敏感时序

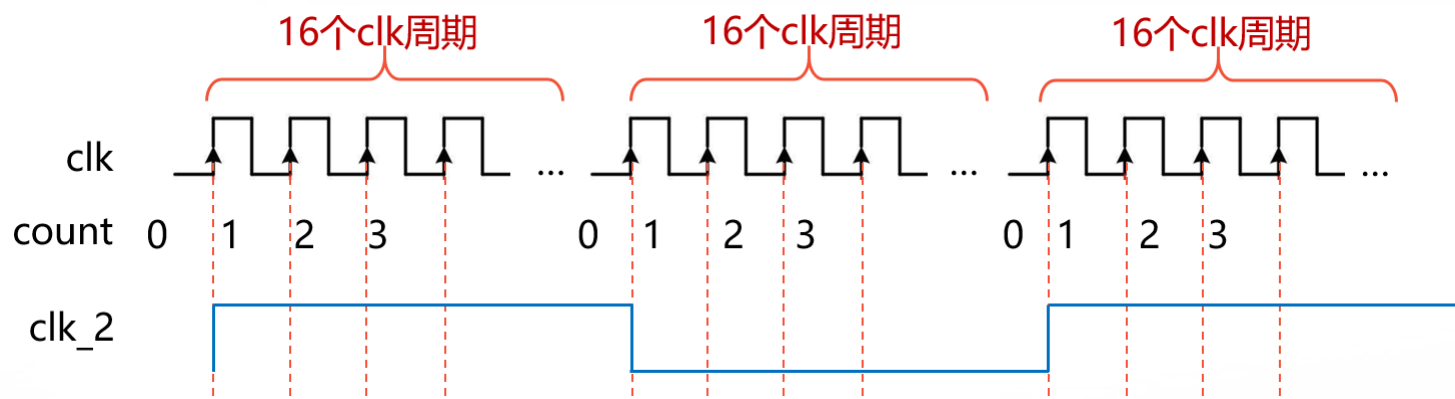
✓ 例：分频器。

对输入时钟做分频，以得到更低频率的时钟。

输入信号：clk（高频时钟），输出信号clk_2（低频时钟）。

```
reg [3:0] count;
always @(posedge clk) begin
    count <= count + 4'h1;
end

reg clk_2;
always @(posedge clk) begin
    if(count == 4'h1) begin
        clk_2 <= ~clk_2;
    end
    else begin
        clk_2 <= clk_2;
    end
end
```





◆ 持续赋值 (continuous assignment)

- 可以用持续赋值语句描述组合逻辑
- 持续赋值在过程块外使用
- 持续赋值用于net驱动
- 持续赋值只能在等式左边有一个简单延时说明
 - 只限于在表达式左边用#delay形式
- 持续赋值可以是显式或隐含的

语法: `<assign> [#delay] [strength] <net_name> = <expressions>;`

```
wire out;
```

```
assign out = a & b; // 显式
```

```
wire inv = ~in; // 隐含
```

◆ If语句

✓ 例1：二选一数控开关。

输入信号a,b,sel, 输出信号y
当sel为1时, 选择a路信号输出给y;
当sel为0时, 选择b路信号输出给y。

```
always @(a or b or sel) begin
    if(sel == 1'b1) y <= a;
    else            y <= b;
end
```

描述方式:

if(表达式)

begin

.....

end

else

begin

.....

end

✓ 例2：带使能的二选一数控开关。

输入信号a,b,sel,en, 输出信号y
当en为1时, 输出y符合例1中的规则;
当en为0时, y恒输出0。

```
always @(*) begin
    if(en == 1'b1) begin
        if(sel == 1'b1) y <= a;
        else            y <= b;
    end
    else begin
        y <= 0;
    end
end
```

- 可以多层嵌套。在嵌套if序列中, else和前面最近的if相关。
- 为提高可读性及确保正确关联, 使用begin...end块语句指定其作用域。

◆ case语句

• 例3：数据分配器

输入信号a,b,i, 输出信号y0,,y1,y2,y3;

当ab为00时, i输出给y0, y1y2y3输出0;

当ab为01时, i输出给y1, y0y2y3输出0;

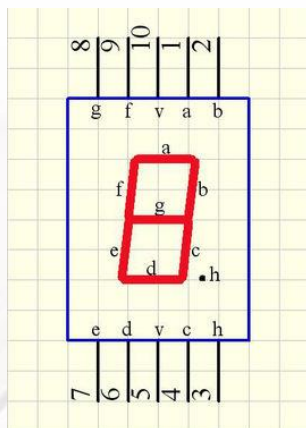
当ab为10时, i输出给y2, y0y1y3输出0;

当ab为11时, i输出给y3, y0y1y2输出0。

```
always @(*) begin
    case ({a,b})
        2'b00: {y3,y2,y1,y0} <= {3'b000,i};
        2'b01: {y3,y2,y1,y0} <= {2'b00,i,1'b0};
        2'b10: {y3,y2,y1,y0} <= {1'b0,i,2'b00};
        2'b11: {y3,y2,y1,y0} <= {i,3'b000};
        default: ;
    endcase
end
```

```
always @(k) begin
    case (k)
        4'b0000: seg7out = 7'b0000001;
        4'b0001: seg7out = 7'b1001111;
        4'b0010: seg7out = 7'b0010010;
        4'b0011: seg7out = 7'b0000110;
        4'b0100: seg7out = 7'b1001100;
        4'b0101: seg7out = 7'b0100100;
        4'b0110: seg7out = 7'b0100000;
        4'b0111: seg7out = 7'b0001111;
        4'b1000: seg7out = 7'b0000000;
        4'b1001: seg7out = 7'b0000100;
        4'b1010: seg7out = 7'b0001000;
        4'b1011: seg7out = 7'b1100000;
        4'b1100: seg7out = 7'b0110001;
        4'b1101: seg7out = 7'b1000010;
        4'b1110: seg7out = 7'b0110000;
        4'b1111: seg7out = 7'b0111000;
    endcase
end
```

• 例4：7段译码管





■ 循环语句

repeat: 将一块语句循环执行确定次数。

repeat (次数表达式) <语句>

while: 在条件表达式为真时一直循环执行

while (条件表达式) <语句>

forever: 重复执行直到仿真结束

forever <语句>

for: 在执行过程中对变量进行计算和判断，在条件满足时执行

for(赋初值; 条件表达式; 计算) <语句>

➤ 不常用，只有for语句是可以被综合的，其它三种只在仿真中使用。



Part 02

实验内容

- 带使能的数控开关
- 数据分配器

◆ 完成例2：带使能的二选一数控开关，练习if语句

1. 新建工程为lab31;
2. 为工程添加Verilog设计文件，完成代码编辑;
3. 编译
4. 为设计添加波形仿真文件vwf，给输入信号设置激励，对设计进行仿真。

```
always @(*) begin
    if(en == 1'b1) begin
        if(sel == 1'b1) y <= a;
        else            y <= b;
    end
    else begin
        y <= 0;
    end
end
```

✓ 思考：如何合理的设置激励，使得仿真结果逻辑完备且结果易读

◆ 完成例3：数据分配器 练习case语句

1. 新建工程为lab32;
2. 为工程添加Verilog设计文件，完成代码编辑;
3. 编译
4. 为设计添加波形仿真文件vwf，给输入信号设置激励，对设计进行仿真。

```
always @(*) begin
    case({a,b})
        2'b00: {y3,y2,y1,y0} <= {3'b000,i};
        2'b01: {y3,y2,y1,y0} <= {2'b00,i,1'b0};
        2'b10: {y3,y2,y1,y0} <= {1'b0,i,2'b00};
        2'b11: {y3,y2,y1,y0} <= {i,3'b000};
        default: ;
    endcase
end
```

✓ 思考：如何合理的设置激励，使得仿真结果逻辑完备且结果易读



Part 03

实验报告要求

- 实验内容
- 设计步骤
- 结果图示
- Verilog知识总结



◆ 实验报告应至少包含 （实验报告以pdf形式提交到助教邮箱）

1. Verilog中module的基本框架
2. Verilog中数据类型、赋值方法、条件语句
3. 实验过程中完成的代码和仿真结果图
4. 总结Verilog相关知识

实验报告模板 →

电子技术实验 2 实验报告

学号: „
班级: „
姓名: „

Verilog 语法基础

一 实验内容

二 设计步骤

三 结果图示（含代码设计和仿真结果）

四 Verilog 相关知识

4.1 Verilog 中 module 的基本框架

4.2 Verilog 中的主要数据类型和主要赋值方法

4.3 Verilog 中的两个条件语句