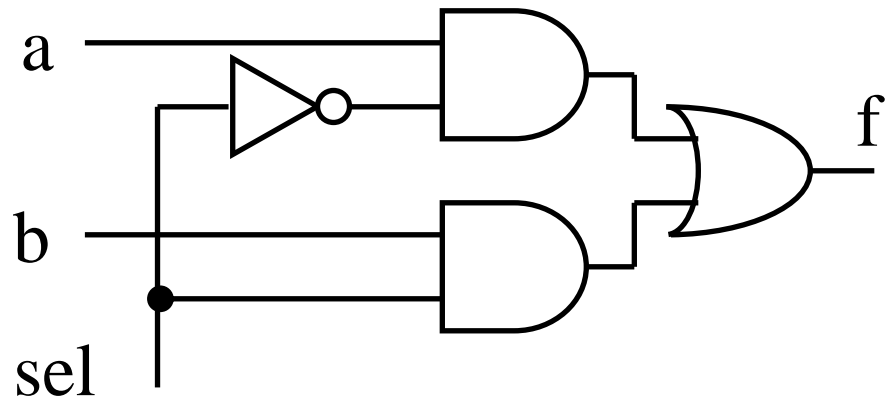
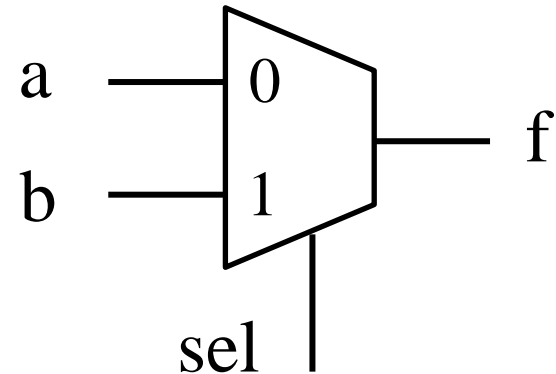


示例— Mux with Assign

```
module mux2_1(  
    output f,  
    input a, b, sel );
```

```
    assign f = (a & ~sel) | (b & sel);
```

```
endmodule
```



HDL主要特征

- HDL语言既包含一些高级程序设计语言的结构形式，同时也兼顾描述硬件线路连接的具体构件
- 通过使用结构级或行为级描述可以在不同的抽象层次描述设计
 - 五个抽象层次：系统级、算法级、寄存器传输级、逻辑（门）级、电路（开关）级
- HDL语言是**并发**的，即具有在同一时刻执行多个任务的能力
- HDL语言有**时序**的概念

Verilog HDL与 C语言的比较

- 虽然Verilog的某些语法与C语言接近，但存在本质上的区别
 - Verilog是一种硬件语言，最终是为了产生实际的硬件电路或对硬件电路进行仿真
 - C语言是一种软件语言，是控制硬件来实现某些功能
- 利用Verilog编程时，要时刻记着：Verilog是硬件描述语言，要将其与**硬件电路对应**起来

C	Verilog
procedures	modules
variables	wires/regs
parameters	ports
control (if,case,?:)	control (if,case,?:)

Verilog HDL基本语法

- 空白符（间隔符）
 - 主要起分隔文本的作用，可以使文本错落有致，便于阅读与修改
 - 包括空格、制表符、换行符及换页符
- 注释符
 - 改善程序的可读性，在编译时不起作用
 - 多行注释符：以/*开始到*/结束
 - 单行注释符：以//开始到行尾结束

Verilog HDL基本语法 (续1)

- 关键字

- Verilog语言内部已经使用的词，例如，module、endmodule、input、output、wire、reg、and等
- 其中的字母都是小写，例如Input不是关键字

- 标识符

- 用于对象（如模块名、电路的输入与输出端口、变量等）命名
- 以字母或下划线 “_”开始，字母、下划线、数字等的组合
- 字母大小敏感，如in，IN是不同的标识符
- 不能与关键词相同

Verilog HDL基本语法 (续2)

- 逻辑值集合
 - 0: 低电平、逻辑0或“假”
 - 1: 高电平、逻辑1或“真”
 - x/X: 不确定的值（未知状态）
 - z/Z: 高阻态
- 常量与符号常量

常量

- 整数型

- 十进制数形式表示，例如，30、-2
- 带基数形式表示，格式为：

〈+/-〉 〈位宽〉' 〈基数符号〉 〈数值〉

基数符号：十进制 D/d，二进制 B/b，八进制 O/o，十六进制 H/h

例如，-8' d101、5' o37、8' HeD, 8' b1001_001x

- 实数型常量

- 十进制记数法，例如，0.1、2.0、5.67
- 科学记数法，例如，23.1e2、5E-4

符号常量

- 用参数定义语句定义一个标识符来代表一个常量
 - 常用来定义变量的位宽及延时等
- 定义格式

parameter 参数名1=常量表达式1, 参数名2=常量表达式2,;

例如: parameter BIT=1, BYTE=8, PI=3.14;

变量数据类型

- **线网(net)型**：表示元件之间的物理连线
 - 输出值紧随输入值的变化而变化
 - 最常用类型是wire
- **寄存器(register)型**：表示抽象存储元件
 - 在赋新值以前保持原值
 - 只能在initial或always语句中被赋值
 - 最常用类型是reg
- **定义格式**

wire/reg [MSB:LSB] 变量名1, ..., 变量名n;

例如: wire a, b; reg[3:0] state;

Verilog HDL程序基本结构

- 由实现特定功能的模块构成

module 模块名 (端口名1, 端口名2, ...);

端口类型说明(input, output, inout);

参数定义(可选);

数据类型定义(wire, reg等);

} 说明部分

实例化低层模块和基本门级元件;

连续赋值语句 (assign) ;

过程块结构 (initial和always)

行为描述语句;

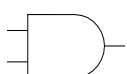




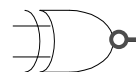
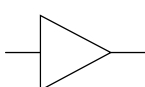
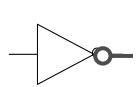
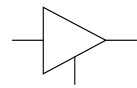
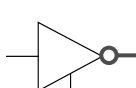
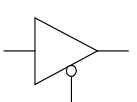
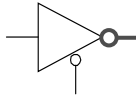
} 功能描述部分
顺序是任意的

endmodule

VerilogHDL描述组合逻辑电路

- 组合逻辑电路的门级描述
 - 使用内置的基本门级元件描述
- 组合逻辑电路的数据流描述
 - 使用连续赋值assign语句描述
- 组合逻辑电路的行为级描述
 - 使用always结构描述

基本门级元件

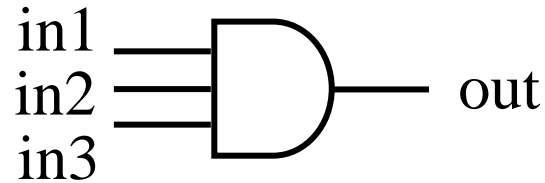
元件符号	功能说明	元件符号	功能说明
and 	多输入端与门	nand 	多输入端与非门
or 	多输入端或门	nor 	多输入端或非门
xor 	多输入端异或门	xnor 	多输入端异或非门
buf 	多输出端缓冲器	not 	多输出端反相器
bufif1 	高电平有效三态缓冲器	notif1 	高电平有效的三态反相器
bufif0 	低电平有效三态缓冲器	notif0 	低电平有效的三态反相器

多输入门和多输出门

- 多输入门：允许多个输入，但只有一个输出

– and, or, xor, nand, nor, xnor

实例名可忽略

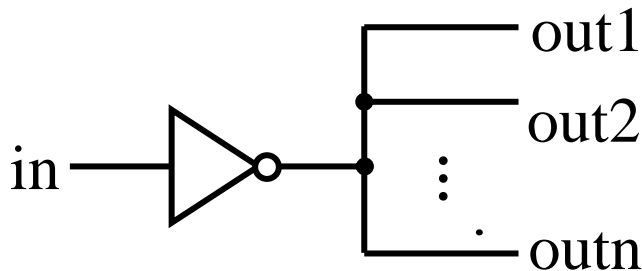


and A1(out, in1, in2, in3);

- 多输出门：允许有多个输出，但只有一个输入

– not, buf

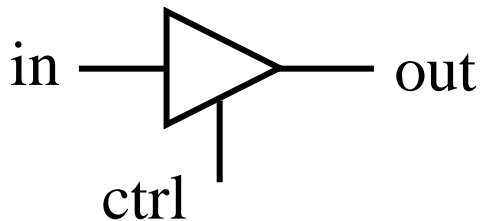
实例名可忽略



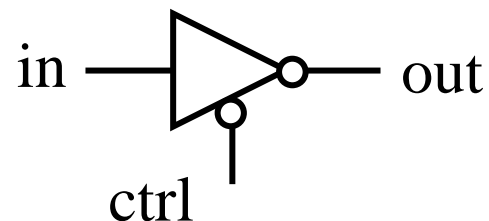
not B1(out1, out2, ..., in);

三态门

- 一个输出、一个数据输入和一个控制输入
 - notif0, notif1, bufif0, bufif1



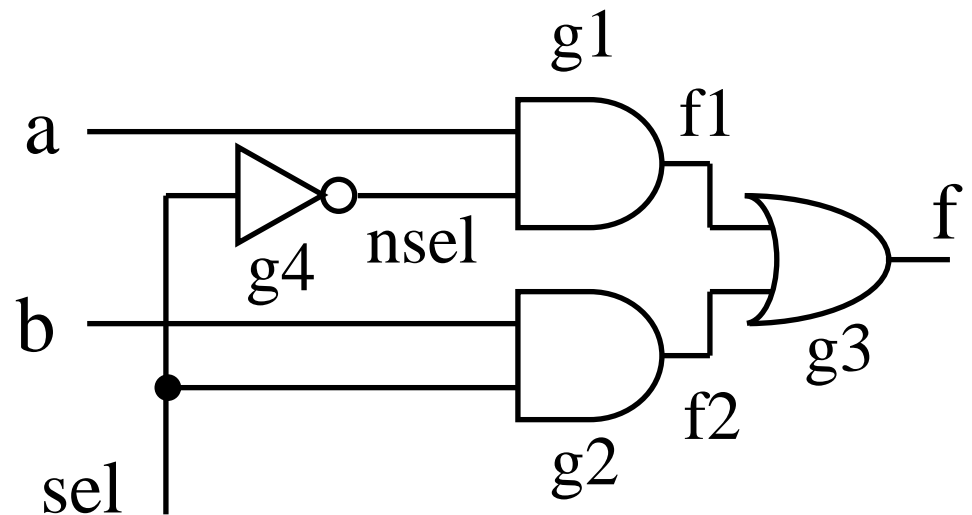
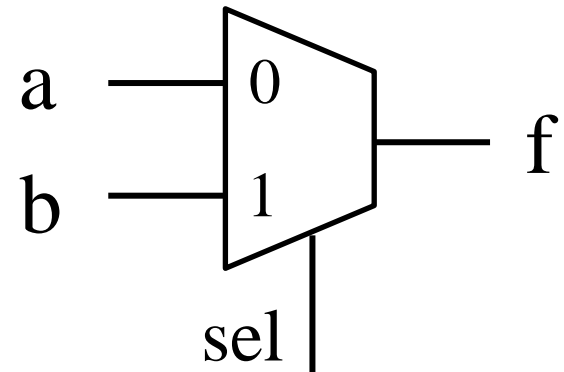
bufif1 B1(out, in, ctrl);



notif0 N1(out, in, ctrl);

示例— Mux with Primitives

```
module mux2_1( f, a, b, sel );  
    output f;  
    input a, b, sel;  
  
    and g1(f1, a, nsel),  
        g2(f2, b, sel);  
    or g3(f, f1, f2);  
    not g4(nsel, sel);  
  
endmodule
```



赋值语句

- 连续赋值语句

assign 变量名= 赋值表达式

- 只能对线网型变量进行赋值，不能对寄存器型变量进行赋值
- 仅用于描述组合逻辑

- 过程赋值语句

变量名= 赋值表达式

- 只能对寄存器数据类型的变量赋值
- 在always和initial语句内的赋值
- 可用于描述组合和时序逻辑

Verilog HDL运算符

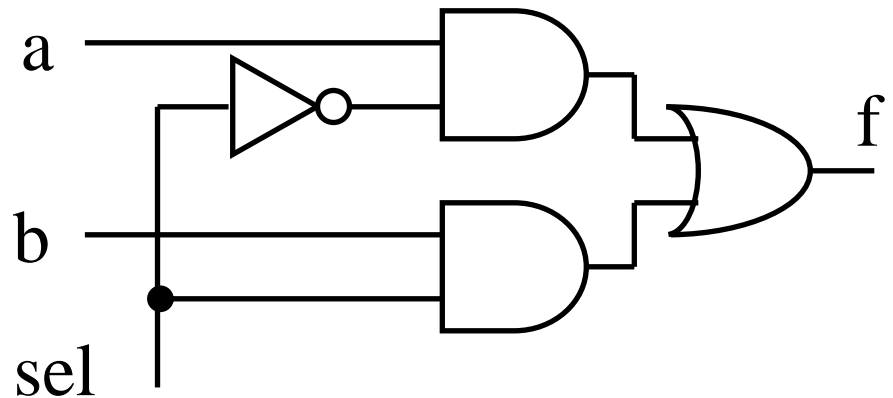
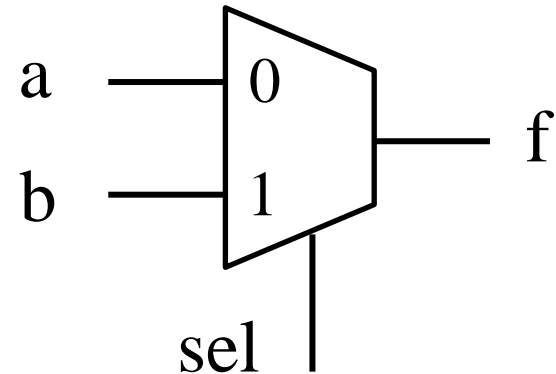
类型	符号	功能说明	类型	符号	功能说明
算术运算符	+ - * / %	二进制加 二进制减 二进制乘 二进制除 求模	关系运算符	> < >= <= == !=	大于 小于 大于或等于 小于或等于 等于 不等于
位运算符	~ & ^ ^~ 或 ~^	按位取反 按位与 按位或 按位异或 按位同或	缩位运算符	& ~& ~ ^ ^~ 或 ~^	缩位与 缩位与非 缩位或 缩位或非 缩位异或 缩位同或
逻辑运算符	! && 	逻辑非 逻辑与 逻辑或	移位运算符	>> <<	右移 左移

示例— Mux with Assign

```
module mux2_1(  
    output f,  
    input a, b, sel );
```

```
    assign f = (a & ~sel) | (b & sel);
```

```
endmodule
```



if条件语句

- 表达式一般为逻辑表达式或关系表达式
 - 对表达式的值进行判断，若为0，x，z，按假处理；若为1，则按真处理，执行指定语句
- if和else后可包含单个或多个语句，多句时用begin-end块语句括起来
- if语句嵌套使用时，注意if与else的配对关系

```
if (表达式) 语句1;  
if (表达式) 语句1;  
    else 语句2;  
if (表达式1) 语句1;  
    else if (表达式2) 语句2;  
    else if (表达式3) 语句3;  
        .....  
    else if (表达式n) 语句n;  
    else 语句n+1;
```

Case条件语句

case （敏感表达式）

值1: 语句1;

值2: 语句2;

.....

值n: 语句n;

default: 语句n+1;

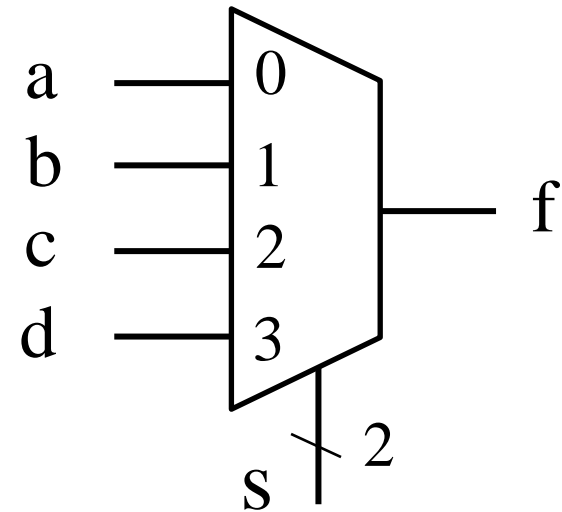
endcase

条件语句使用要点

- 描述组合电路时，应注意列出所有条件分支，否则编译器认为条件不满足时，会引进一个记忆单元（锁存器）来保持原值，从而产生时序电路而非组合电路
- 由于每个变量有4种取值，为包含所有分支，可在if语句后加上 **else**；在 **case**语句后加上 **default**

示例— Mux with Always (If)

```
module mux4_1(  
    output reg f,  
    input a, b, c, d,  
    input [1:0] s);  
  
    always @(a, b, c, d, s)  
        if (s == 2'b00) f = a;  
        else if (s == 2'b01) f = b;  
        else if (s == 2'b10) f = c;  
        else f = d;  
endmodule
```



示例— Mux with Always(Case)

```
module mux4_1(  
    input a, b, c, d,  
    input [1:0] s,  
    output reg f );  
    always @(a, b, c, d, s)  
        case (s)  
            2'b00: f = a;  
            2'b01: f = b;  
            2'b10: f = c;  
            default: f = d;  
        endcase  
endmodule
```

