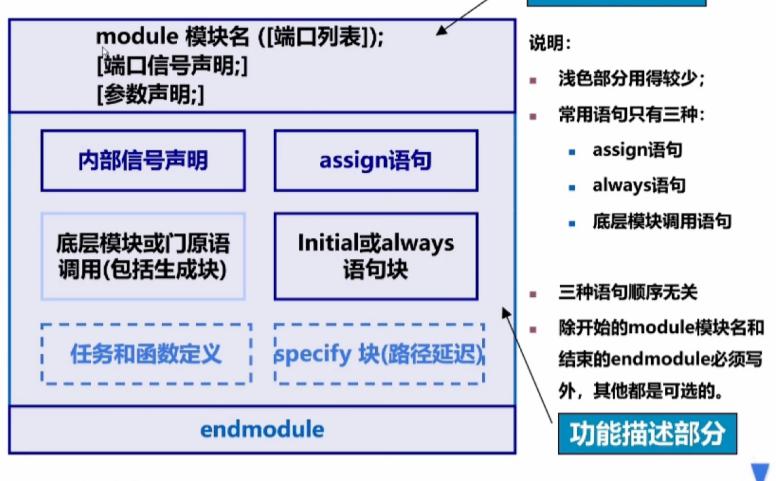


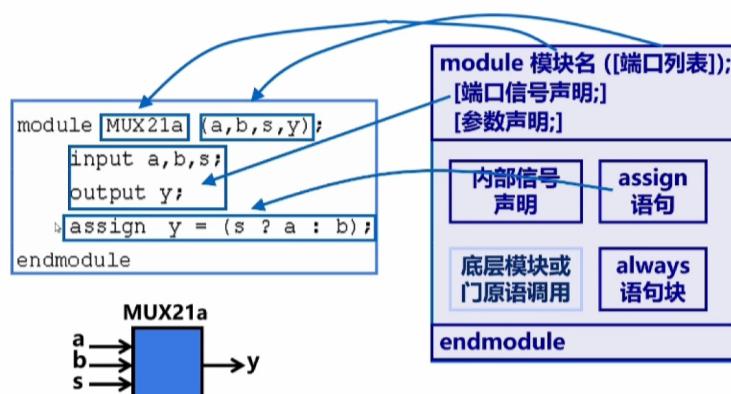
## Verilog模块结构



① ② ③ ④ ⑤ ⑥ ⑦

## Verilog模块结构

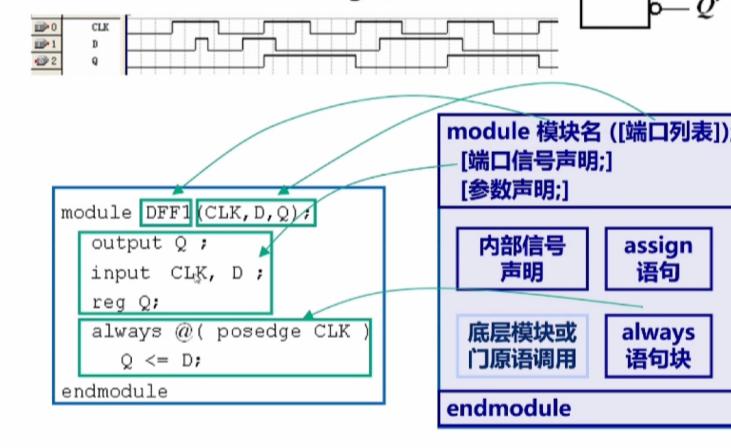
### 2选1多路选择器的Verilog描述



① ② ③ ④ ⑤ ⑥ ⑦

## Verilog程序结构

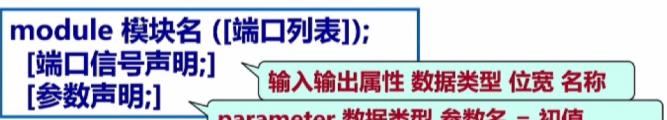
### 例2：边沿D触发器的Verilog描述



① ② ③ ④ ⑤ ⑥ ⑦

## Verilog模块结构

### 1. 模块说明部分



- 模块名是指电路的名字, 由用户指定, 最好与文件名一致  
(特别是在Quartus II软件中调试时);
- 端口列表是指电路的输入/输出信号名称列表, 信号名由用户指定, 各名称间用逗号隔开;
- 端口信号声明是要说明端口信号的输入输出属性、信号的数据类型, 以及信号的位宽; 输入输出属性有input, output, inout三种, 信号的数据类型常用的有wire和reg两种; 信号的位宽用[n1:n2]表示; 同一类信号之间用逗号隔开;
- 参数声明要说明参数的名称和初值

① ② ③ ④ ⑤ ⑥ ⑦

## Verilog模块结构

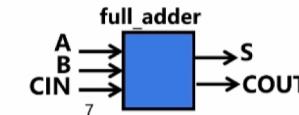
例: module full\_adder (A,B,CIN,S,COUT);  
input [3:0] A,B;  
input CIN;  
output reg [3:0] S;  
output COUT;

位宽如果不做说明的话, 默认是1位;  
数据类型不做说明的话, 默认是wire型的。



S位宽为4位, 对应信号为S[3]、S[2]、S[1]、S[0]

根据模块说明部分, 我们可以得出电路符号



① ② ③ ④ ⑤ ⑥ ⑦

## 2. assign语句

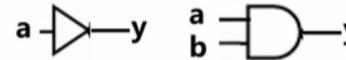
assign语句称作连续赋值语句

基本格式:

★ assign 赋值目标 = 表达式

例: assign y=a;

assign y=a&b;



module 模块名 ([端口列表]);

[端口信号声明];
 内部信号声明
 assign语句
 底层模块或门原语调用
 always语句块
endmodule

特点:

- 之所以称为连续赋值语句是指其总是处于激活状态, 只要表达式中的操作数有变化, 立即进行计算和赋值。(与连续赋值语句对应的另一种语句称为过程赋值语句)
- 赋值目标必须是wire型的, wire表示电路间的连线。reg属于寄存器类型。

## 2. assign语句

详见夏宇闻教材第6章

Verilog具有丰富的表达式运算功能, 可用于assign语句

操作类型	操作符	执行的操作
算术	*	乘
	/	除
	+	加
	-	减
	%	取模
	**	求幂
逻辑	!	逻辑求反
	&&	逻辑与
		逻辑或
关系	>	大于
	<	小于
	>=	大于等于
	<=	小于等于
等价	==	相等
	!=	不等
	== ==	case 相等
	!= !=	case 不等

操作类型	操作符	执行的操作
按位	~	按位求反
	&	按位与
		按位或
	^	按位异或
	^ ~ 或 ~ ^	按位同或
按减	&	按减与
	~ &	按减与非
	-	按减或
	-	按减或非
	^	按减异或
	^ -  或 ~ ^	按减同或
移位	>>	右移
	<<	左移
	>>>	算术右移
	<<<	算术左移
拼接	[ ]	拼接
复制	[ ( ) ]	复制
条件	? :	条件

① ② ③ ④ ⑤ ⑥ ⑦

## 2. assign语句

*	乘法
/	除法
+	加法
-	减法
%	求余
**	求幂

例：  
 $Y=5\%2;$  求余, 结果为1  
 $Y=2^{**}3;$  求幂, 结果为8

说明  
 ● 加减乘除、求幂的操作数可以是实数也可以是整数，求余运算的操作数只能是整数。  
 ● 求余运算结果取第一个操作数的符号；

## 2. assign语句

### (5) 按位运算符

~	按位非
&	按位与
	按位或
^	按位异或
~^	按位同或
^~	$\neg(a \wedge b)$

例：  
 $Y=\sim 4' b1001;$  结果为0110  
 $Y=4' b1001 \& 4' b0111;$  结果为0001  
 $Y=4' b1001 | 4' b0111;$  结果为1111  
 $Y=3' b001 | 4' b0111;$  结果为0111  
 $Y=3' b001 | 4' b0111 \& 3' b101;$  结果为0101

说明

- 按位运算的操作数是1位或多~~位~~二进制数。
- 按位非的操作数只有一个，将该数的每一位求非运算。
- 其它按位运算的操作数有2个或多个，将两个操作数对应的位两两运算；
- 如果操作数位宽不同，位宽小的会自动左添0补齐；
- 结果与操作数位宽相同；

## 2. assign语句

例：  
 $Y=!(3>2)$  逻辑非, 结果为0  
 $Y=(2<3) \&& (5>6);$  逻辑与, 结果为0  
 $Y=(2<3) || (5>6);$  逻辑或, 结果为1  
 $Y=(2<3) \&& 1' bx;$  逻辑与, 结果为x  
 $Y=(2+3) || (3-3);$  逻辑或, 结果为1  
有不确定x时注意

说明

- 逻辑型运算的结果可能是1（逻辑真）、0（逻辑假）、x（不确定）；
- 逻辑运算的操作数可以是任意表达式，表达式的结果被当做逻辑值处理，只有1、0、x三种情况，非0、x即1；
- 表达式最好加括号。

## 2. assign语句

### (6) 缩减运算符

&	缩减与
~&	缩减与非
	缩减或
~	缩减或非
^	缩减异或
~^	缩减同或
^~	

例：  
 $Y=\& 4' b1001;$  结果为0  
 $Y=\sim \& 4' b1001;$  结果为1

参与运算的‘’的进位数 $\Rightarrow$   
偶数 $\Rightarrow$ 1

说明

- 缩减运算的操作数是1位或多~~位~~二进制数；
- 缩减运算的操作数只有一个，将该数的各位自左至右进行逻辑运算，结果只有一位。

## 2. assign语句

例：  
 $Y=(3>2)$  结果为1  
 $Y=(3<2);$  结果为0  
 $Y=(3>=2);$  结果为1  
 $Y=(3<=2);$  结果为0  
 $Y=(3<=1' bx);$  结果为x

说明

- 关系运算的结果可能是1（逻辑真）、0（逻辑假）、x（不确定）；

## 2. assign语句

### (7) 移位运算符

>>	右移
<<	左移
>>>	算术右移
<<<	算术左移

例：  
 $Y=4' b1001 >> 1;$  结果为0100  
 $Y=4' b1001 >>> 1;$  结果为1100  
带符号，算术移位时在左边补符号位  
算术左移时保留符号位，其与左移相同

格式 操作数 移位符 n;

说明

- 移位运算的操作数是1位或多~~位~~二进制数；
- 向左或向右移n位；
- 只有对有符号数的算术右移自动补符号位；
- 其他移位均自动补0。

④ ⑤ ⑥ ⑦ ⑧ ⑨

循环移位运算

## 2. assign语句

例：  
 $Y=(3==2);$  结果为0  
 $Y=(3!=2);$  结果为1  
 $Y=(3==3);$  结果为1  
 $Y=(1' b1 == 1' bx);$  结果为x  
 $Y=(1' bx == 1' bx);$  结果为x  
 $Y=(1' b1 === 1' bx);$  结果为0  
 $Y=(1' bx === 1' bx);$  结果为1

说明

- 等于和不等于运算的结果可能是1（逻辑真）、0（逻辑假）、x（不确定）；  
对于x或z，认为是不确定的值，比较结果为x；
- case等和case不等的结果只能是1或0，对于x、z认为是确定的值，参加比较；

## 2. assign语句

## 2. assign语句

(8) 拼接复制运算符 格式 {操作数1, 操作数2, ...}

{	拼接
{}	复制拼接

例:

$Y = \{4' b1001, 2' b11\};$  结果为100111

$Y = \{4\{2' b01\}\};$  结果为01010101

$Y = \{\{4\{2' b01\}\}, 2' b11\};$

说明

- 将多个操作数拼接起来;
- 将操作数复制n遍并拼接起来;
- 可以组合使用。



## 3. always语句块

边沿敏感: (posedge 信号名) 信号上升沿到来

(negedge 信号名) 信号下降沿到来

例: (posedge clk)

例: (negedge clk)

电平敏感: (信号名列表) 信号列表中的任一个信号有变化

例: (a,b,c) 当a,b,c中有一个发生变化

说明: 逗号可以换成or 例: (a or b or c)

上升沿是0变1的过程  
下降沿是1变0的过程  
波特图对应上升

## 2. assign语句

### (9) 条件运算符

? : 用于条件赋值 格式一 表达式1 ? 表达式2 : 表达式3

例:

$Y = a ? b : c;$  如果a=1, 则y=b;

如果a=0, 则y=c.

如果a=x, 则y=x.

$Y = s1 ? (s0 ? d3 : d2) : (s0? d1 : d0);$

说明

- 根据表达式1的值, 决定运算结果;
- 如果表达式1值为1, 则结果等于表达式2;
- 如果表达式1值为0, 则结果等于表达式3;
- 如果表达式1值为x, 则结果为x;
- 可以嵌套。

## 3. always语句块

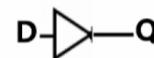
例: always @ (posedge CLK)  
 $Q=D;$

当CLK上升沿到来时, 激活该语句块,  
将D的值赋给Q;  
否则, 该语句块挂起, 即使D有变化,  
Q的值也保持不变, 直到下一次赋值。



例: always @ (D)  
 $Q=D;$

当D有变化时(不管是由1变0还是由0变1), 激活该语句块, 将D的值赋给Q;  
否则, 该语句块挂起, Q的值保持不变, 直到下一次赋值。



## 3. always语句块

always语句块又称过程块

基本格式:

always @ (敏感信号条件表)

各类顺序语句;

上升沿

例: always @ (posedge CLK)  
 $Q=D;$



特点:

- always语句本身不是单一的有意义的一条语句, 而是和下面的语句一起构成一个语句块, 称之为过程块; 过程块中的赋值语句称过程赋值语句;
- 该语句块不是总处于激活状态, 当满足激活条件时才能被执行, 否则被挂起, 挂起时即使操作数有变化, 也不执行赋值, 赋值目标值保持不变;
- 赋值目标必须是reg型的。即输出信号赋值为reg类型

## 3. always语句块

说明:

- 过程块中的赋值目标必须是reg型的。
- 由于always语句可以描述边沿变化, 在设计时序电路中得到广泛应用。
- always语句中还可以使用if、case、for循环等语句, 其功能更加强大。

例: always @ (posedge CLK)  
 $Q:=D;$

例: always @ (D)  
 $Q:=D;$

必须是reg型的

## 3. always语句块

●激活条件由敏感信号条件表决定, 当敏感条件满足时, 过程块被激活。

敏感条件有两种, 一种是边沿敏感, 一种是电平敏感。

在always块中, 应该使用阻塞赋值语句“:=”, 而不是非阻塞赋值“=>”, 除非明确需要描述并行执行的操作

## 3. always语句块

assign语句和always语句的主要区别:

- 连续赋值语句总是处于激活状态, 只要操作数有变化马上进行计算和赋值;
- 过程赋值语句只有当激活该过程时, 才会进行计算和赋值, 如果该过程不被激活, 即使操作数发生变化也不会计算和赋值。
- verilog规定assign中的赋值目标必须是wire型的, 而always语句中的赋值目标必须是reg型的。

例: assign Q=D

只要D发生变化, 马上进行计算和赋值; Q必须是wire型。

always @ (posedge clk)  
 $Q=D;$

只有当clk上升沿到来时, 才能激活该块语句, 才能进行计算和赋值; 否则, 即使D发生变化也不会计算和赋值。在未被激活时, Q必须是reg型。

### 3. always语句块

- always语句块中除了可以使用表达式赋值以外，还可以使用if, case等行为描述语句，还能够描述边沿变化，因此其功能比assign语句更强大（assign语句不能使用if等语句，也不能描述边沿变化）。

例：

```
module DFF2(CLK,D,RST,EN)
input CLK,D,RST,EN;
output Q;
reg Q;
always @ (posedge CLK or negedge RST)
begin
    if (!RST) Q<=0;
    else if (EN) Q<=D
end
endmodule
```

**! begin ... end**之间语句块是顺序执行的，其他语句块均为并行执行

### 3. always语句块

#### 综合举例：4位二进制加法计数器

```
module CNT4(CLK,Q);
input CLK;
output [3:0] Q;
reg [3:0] Q1;
always @ (posedge CLK)
begin
    Q1<=Q1+1;
end
assign Q=Q1;
endmodule
```

此程序中有always和assign两条语句，他们之间是并行的；此程序中有一个内部变量Q1，使用时要进行声明；

☆ 内部信号声明格式：

数据类型 位宽 信号名称 元素个数

### 3. always语句块

always语句块中如果有多条赋值语句必须将其用begin end包括起来，assign语句中没有begin end。**全加黑**

例：

```
module adder(a,b,cin,s,cout)
input a,b,cin;
output s,cout;
reg s,cout;
always @ (a,b,cin)
begin
    s=a^b^cin;
    cout=(a&b)|(a&cin)|(b&cin);
end
endmodule
```

请大家画出该模块的端口符号图和电路图

思考问题：  
在仿真时，begin 和 end之间的语句执行顺序如何？

### 3. always语句块

#### 举例比较：

##### ● 阻塞赋值【例8-6】

```
always @ (A,B)
begin
    M1=A;
    M2=B&M1;
    Q=M1|M2;
end
```

适用于组合逻辑，因为组合逻辑单输出依赖于当前的输入值

##### 设A、B同时由0变1

激活前：M1=0, M2=0, Q=0

激活后：

先计算A=1，马上赋值给M1  
再计算B&M1=1，马上赋值给M2  
再计算M1|M2=1，马上赋值给Q  
**算一条，赋值一条**

##### ● 非阻塞赋值【例8-7】

```
always @ (A,B)
begin
    M1<=A;
    M2<=B&M1;
    Q<=M1|M2;
end
```

适用于时序逻辑，因为时序逻辑的输出需要在特定的时间点更新

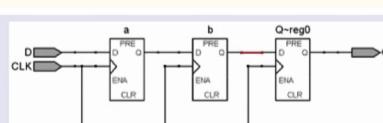
先计算A=1，(等待，不赋值)  
再计算B&M1=0，(等待，不赋值)  
再计算M1|M2=0，(等待，不赋值)  
过程结束

先赋值给M1=1  
再赋值给M2=0  
再赋值给Q=0

**挨个算，算完之后再赋值**

#### 应用：

- 设计组合电路时常用阻塞赋值；
- 设计时序电路时常用非阻塞赋值；
- 但不是绝对的。
- 不建议在一个always块中混合使用阻塞赋值和非阻塞赋值



#### 例：阻塞赋值实现的组合电路

```
module MY(A,B,C,Y)
input A,B,C;
output Y;
reg Y;
reg M;
```

always @ (A,B,C)

begin

    M=B|C;

    Y=A&M;

end

endmodule

#### 例：非阻塞赋值实现的移位寄存器

```
module DDF3(CLK,D,Q)
output Q;
input CLK,D;
reg a,b,Q;
always @ (posedge CLK)
begin
    a<=D;
    b<=a;
    Q<=b;
end
```

### 3. always语句块

#### 综合举例：4位二进制加法计数器

```
module CNT4(CLK,Q);
input CLK;
output [3:0] Q;
reg [3:0] Q1;
always @ (posedge CLK)
begin
    Q1<=Q1+1;
end
assign Q=Q1;
endmodule
```

此程序中有always和assign两条语句，他们之间是并行的；此程序中有一个内部变量Q1，使用时要进行声明；

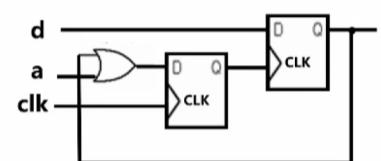
☆ 内部信号声明格式：

数据类型 位宽 信号名称 元素个数

### 4. 底层模块和门原语调用

#### 一、底层模块调用

##### 例：图示电路的描述

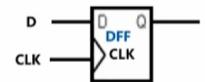


module 模块名 ([端口列表]);	[端口信号声明];
内部信号声明	assign 语句
底层模块或门原语调用	always 语句块
endmodule	

该电路是由两个D触发器和一个或门构成的，设计思路之一是先设计底层电路D触发器，然后再设计顶层电路，在顶层电路中可调用底层模块。

### 4. 底层模块和门原语调用

#### 底层模块描述



```
module DFF(CLK,D,Q)
output reg Q;
input CLK,D;
always @ (posedge CLK)
    Q<=D;
```

#### 底层模块调用格式

##### 底层模块名 例化名 (端口映射);

```
module examp(clk,d,a,q)
output q;
input clk,d,a;
```

wire d1;  
wire q1;

DFF dff1(.CLK(clk),.D(d1),.Q(q1));  
DFF dff2(q1,d,q);

or (d1,a,q);

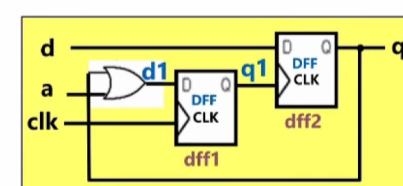
endmodule

### 4. 底层模块和门原语调用

#### 端口映射有两种方法：

##### 端口名关联法（命名法）

##### 位置关联法（顺序法）



#### 命名法格式：

(.底层端口名1(外接信号名1),.底层端口名2(外接信号名2),...)

DFF dff1(.CLK(clk),.D(d1),.Q(q1));

因为有名字对应，不必按底层模块的端口信号列表顺序

#### 顺序法格式：

(外接信号名1,外接信号名2,...)

DFF dff2(q1,d,q);

D	Q
DFF	CLK
module DFF(CLK,D,Q)	

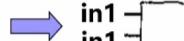
必须严格按照底层模块的端口信号列表顺序书写

## 4. 底层模块和门原语调用

Verilog语言提供已经设计好的门，称为门原语（primitive，共12个），这些门可直接调用，不用再对其进行功能描述。

门原语调用格式：门原语名 实例名（端口连接）

其中实例名可省略（和模块调用不同），端口连接只能采用顺序法，输出在前，输入在后。

例：and (out, in1, in2); 

端口连接中第一个是输出，其余是输入，输入个数不限。

与门等6个	and (与)	or (或)	xor (异或)
	nand (与非)	nor (或非)	xnor (同或)

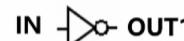
④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 4. 底层模块和门原语调用

### 非门和缓冲器



例：

not (OUT1, IN); 

buf b1\_2out(OUT1, OUT2, IN); 

端口列表中前面是输出，最后一个是输入，输出个数不限。

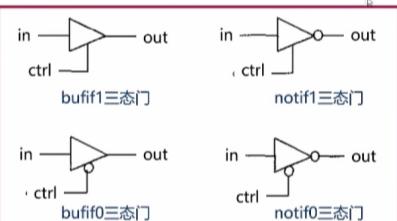
④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 4. 底层模块和门原语调用

### 三态门

bufif1 (控制端1有效缓冲器)      notif1 (控制端1有效非门)

bufif0 (控制端0有效缓冲器)      notif0 (控制端0有效非门)



例：

bufif1 b1 (out, in, ctrl);  
bufif0 b0 (out, in, ctrl);  
  
notif1 n1 (out, in, ctrl);  
notif0 n0 (out, in, ctrl);

端口列表中前面是输出，中间是输入，最后是使能端，输出个数不限。

④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 5. Verilog中的数据类型

Verilog中的数据类型分为两大类

### 线网类 (net类)

### 变量类 (variable类)

因连续赋值语句和过程赋值语句的激活特点不同，故赋值目标特点也不同，前者不需要保存，后者需要保存，因此规定两种数据类型，net型用于连续赋值的赋值目标或门原语的输出，且仿真时不需要分配内存空间，variable用于过程赋值的赋值目标，且仿真时需要分配内存空间。

④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 5. Verilog中的数据类型

### ● net类中的数据类型

wire (线型)	tri (三态)	tri0 (下拉电阻)	supply0 (地)
wand (线与)	triand (三态与)	tri1 (上拉电阻)	supply1 (电源)
wor (线或)	trior (三态或)	trireg (电容性线网)	

最常用的是wire

### ● variable类中的数据类型

reg (寄存器型)	
integer (整型)	time (时间型)
real (实型)	realtime (实时间型)

最常用的是reg

④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 5. Verilog中的数据类型

将一个信号定义成net型还是variable型，由以下两方面决定

- 使用何种赋值语句对该信号进行赋值，如果是连续赋值或门原语赋值或例化语句赋值，则定义成net型；如果是过程赋值则定义成variable型。
- 对于端口信号来说，input信号和inout信号必须定义成net型的；output信号可以是net型的也可以是variable型的，决定于如何对其赋值（同a）。



④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 5. Verilog中的数据类型

例：该图中d和e的赋值有三种方法

### (1) 使用连续赋值语句

assign d=a&b;  
assign e=d|c;

此时，d和e必须  
定义为net型的。

### (2) 使用门原语赋值

and (d,a,b);  
or (e,d,c);

此时，d和e也必须  
定义为net型的。

### (3) 使用过程赋值语句

```
always @ (a,b,d,c)  
begin  
    d=a&b;  
    e=d|c;  
end
```

此时，d和e必须  
定义为variable型的。

### (3) 使用过程赋值语句

```
always @ (a,b,d,c)  
begin  
    d=a&b;  
    e=d|c;  
end
```

此时，d和e必须  
定义为variable型的。

④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 5. Verilog中的数据类型

Verilog中的数据类型分为两大类

### 线网类 (net类)

### 变量类 (variable类)

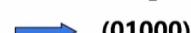
因连续赋值语句和过程赋值语句的激活特点不同，故赋值目标特点也不同，前者不需要保存，后者需要保存，因此规定两种数据类型，net型用于连续赋值的赋值目标或门原语的输出，且仿真时不需要分配内存空间，variable用于过程赋值的赋值目标，且仿真时需要分配内存空间。

④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 6. Verilog中数字的表示格式

无符号数的表示方法：<位宽>'<进制><数字>

例：2'b00 

5'd8 

有符号数的表示方法：<位宽>'<sb><数字>

例：8'sb10111011 

注意有符号数是按照补码表示的，即第一位是符号位。

比较：8'b10111011 

④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 7. 逻辑值

Verilog语言中的逻辑值有四种

- 1: 逻辑1, 高电平, 数字1
- 0: 逻辑0, 低电平, 数字0
- x: 不确定
- z: 高阻态

## 8. if语句

**条件表达式格式:** (计算表达式)  
 计算表达式可以是任意形式的表达式;  
 条件表达式的结果只有0和1两种, 如果计算表达式的值为0,  
 则条件表达式的值为0, 否则为1。

例如: 设 $a=1000$ ,  $b=0110$

条件表达式	计算表达式	结果
if ( $a==b$ )	0	0
if ( $a>b$ )	1	1
if ( $a$ )	1000	1
if ( $a*b$ )	11_0000(前两位被截掉)	0
if ( $a b$ )	1110	1
if ( $a&b$ )	0000	0

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 8. if语句

### 4种类型的if语句

- ① if (<条件表达式>) 语句;
- ② if (<条件表达式>) 真语句; else 假语句;
- ③ if (<条件表达式1>) 语句1;  
else if (<条件表达式2>) 语句2;  
else if (<条件表达式3>) 语句3;
- ④ if (<条件表达式1>) 语句1;  
else if (<条件表达式2>) 语句2;  
else if (<条件表达式3>) 语句3;  
else 默认语句;

计算条件表达式。  
 如果结果为真(1或非0值), 则执行真语句;  
 如果条件为假(0或x), 则执行假语句。

- 显然③、④种可比①、②种描述更复杂的条件关系。

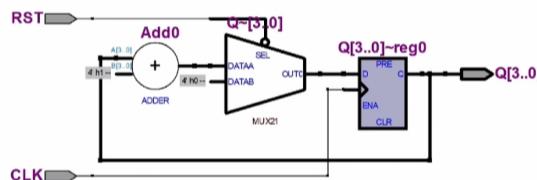
可以是一条语句, 也可以是一组语句

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 8. if语句

### 例: 计数器

```
always @ (posedge CLK)
  if (!RST) Q=0;
  else Q=Q+1;
```



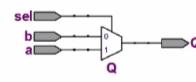
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 8. if语句

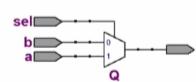
- 在用if语句设计“组合电路”时要注意, 如果条件不完整, 会综合出寄存器。不存在未定义的情况

使条件完整的两种方法:

1. 加else  
`always @ (a,b)
 if (sel) Q=a;
 else Q=b;`



2. 设初值  
`always @ (a,b)
 Q=a;
 if (sel) Q=b;`



① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

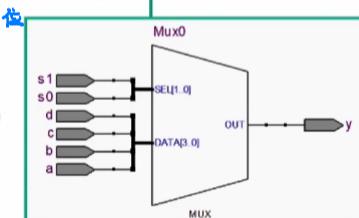
## 9. case语句

格式:	case (表达式) 取值1: 语句1; 取值2: 语句2; 取值3: 语句3; ... ... default: 默认语句; endcase	case (表达式) 取值1: 语句1; 取值2: 语句2; 取值3: 语句3; ... ... endcase
功能:	<p>如果表达式的值=取值1, 则执行语句1;    如果表达式的值=取值2, 则执行语句2;    如果表达式的值=取值3, 则执行语句3;    .....    如果表达式的值和上述取值都不相等, 则执行默认语句。    default语句可以不带。</p>	

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 9. case语句

例: module MUX41 (a,b,c,d,s1,s0,y);  
 input a,b,c,d,s1,s0; 均为1位  
 output reg y;  
 //reg y;  
 always @ (a,b,c,d,s1,s0)  
 begin  
 把s1与s0拼接  
 case ({s1,s0})
 2' b00 : y=a;
 2' b01 : y=b;
 2' b10 : y=c;
 2' b11 : y=d;
 default : y=0;
 endcase
 end
 endmodule



如果条件描述不完整, 则会综合出寄存器; 在设计组合电路时要注意使条件描述完整。  
 加default语句可以使条件完整。  
 如果条件描述完整也可以不加default语句。

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 10. Verilog语言的描述风格

### 三种描述方式

- 结构化描述 (也称门级描述) (全部用门原语和底层模块调用)
- 数据流级描述 (全部用assign语句)
- 行为级描述 (全部用always语句配合if、case语句等)

有些资料中提到另外一种描述方式: RTL级描述方式

- RTL级描述 (数据流级+行为级, 可综合)

可综合为门级语句

实际描述是三种混合的

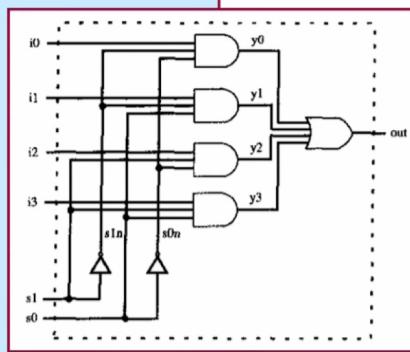
举例: 用门级描述、数据流描述、行为描述分别设计数据选择器

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

## 10. Verilog语言的描述风格

### 例 多路选择器的Verilog 门级描述

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
wire s1n, s0n;
wire y0, y1, y2, y3;
not (s1n, s1);
not (s0n, s0);
and (y0, i0, s1n, s0n);
and (y1, i1, s1n, s0);
and (y2, i2, s1, s0n);
and (y3, i3, s1, s0);
or (out, y0, y1, y2, y3);
endmodule
```



名称 英文	与门 AND	或门 OR	非门 NOT	与非门 NAND	或非门 NOR	异或门 XOR	同或门 XNOR
表达式	$Y=A \& B$	$Y=A B$	$Y=\sim A$	$Y=\sim(A \& B)$	$Y=\sim(A B)$	$Y=A \wedge B$	$Y=\sim A \& \sim B$ $  A \& B$
国内符号							
国际符号							
输入 A B	输出	输出	输出	输出	输出	输出	输出
0 0	0	1	1	1	0	1	1
0 1	0	1	1	1	0	1	0
1 0	0	1	0	1	0	1	0
1 1	1	1	0	0	0	0	0

## 10. Verilog语言的描述风格

### 例 多路选择器的Verilog 数据流级描述

```
module mux_to_1 (out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
assign out = (~s1 & ~s0 & i0)|(~s1 & s0 & i1)|(s1 & ~s0 & i2)|(s1 & s0 & i3);
endmodule
```

数据选择器的公式：

$$Y = i0(\overline{S1}S0) + i1(\overline{S1}S0) + i2(S1\overline{S0}) + i3(S1S0)$$

1-2 andgate 与门

1-6 nandgate 与非门

1-7 norgate 或非门

1-3 notgate 非门

1-5 xnorgate 同或门

1-1 orgate 异或门

1-4 xorgate 异或门

## 10. Verilog语言的描述风格

### 例 多路选择器的Verilog 行为级描述

```
module mux4_to_1(out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
begin
  case ({s1, s0})
    2'b00: out = i0;
    2'b01: out = i1;
    2'b10: out = i2;
    2'b11: out = i3;
    default: out = 1'b0;
  endcase
end
endmodule
```

## 11. 其它规定

### (1) 关键字

关键字即Verilog语言中预定义的有特殊含义的英文词语

### (2) 标识符

标识符即用户自定义的信号名、模块名等等；

注意关键字不能作标识符；

Verilog区别大小写（关键字都是小写）。

### (3) 文件取名和存盘

Verilog文件扩展名为.v； verilog不要求文件名和模块名一致，但QuartusII要求一致

### (4) 注释 //单行注释

/\* \*/多行注释

