



# Verilog HDL语言规范

主讲：何宾

**Email: [hebin@mail.buct.edu.cn](mailto:hebin@mail.buct.edu.cn)**

**2014.06**

# Verilog HDL表达式

表达式是将操作数和操作符联合起来使用的一种Verilog HDL语言结构，通过运算得到一个结果。

□ 表达式可以在出现数值的任何地方使用。

# Verilog HDL表达式

## --操作符

Verilog HDL中的操作符按功能可以分为下述类型：

- 算术操作符
- 按位操作符
- 关系操作符
- 归约操作符
- 相等操作符
- 移位操作符
- 逻辑操作符
- 条件操作符
- 连接和复制操作符

# Verilog HDL表达式

## --操作符

按运算符所带操作数的个数可分为三类：

- 单目操作符
- 双目操作符
- 三目操作符

# Verilog HDL表达式

## --操作符

### Verilog HDL支持的操作符列表

{ } { }	并置和复制
一元+ 一元-	一元操作符
+ - * / **	算术运算符
%	取模运算符
> >= < <=	关系操作符
!	逻辑非
&&	逻辑与
	逻辑或

# Verilog HDL表达式

## --操作符

==	逻辑相等
!=	逻辑不相等
===	条件(case)相等
!==	条件(case)不相等
~	按位取反
&	按位与
	按位或
^	按位异或
^~或者~^	按位异或非
&	规约与

~&	规约与非
	规约或
~	规约或非
^	规约异或
^~或者~^	规约异或非
<<	逻辑左移
>>	逻辑右移
<<<	算术左移
>>>	算术右移
?:	条件

# Verilog HDL表达式

## --操作符

### 实数支持的操作符

一元 '+' 和一元 '-'	一元操作符
+ - * / **	算术运算符
> >= < <=	关系操作符
! &&	逻辑
== !=	逻辑相等
?:	条件

# Verilog HDL表达式

## --操作符

### 操作符的优先级


下表给出了所有操作符的优先级。同一行内的操作符具有相同的优先级。表中优先级从高向低进行排列。

- 除条件操作符从右向左关联外，其余所有操作符自左向右关联。
- 当表达式中有不同优先级的操作符时，先执行高优先级的操作符。
- 圆扩号能够用于改变优先级的顺序。




# Verilog HDL表达式

## --操作符

+ - ! ~ & ~&   ~  ^ ~^ ^~ (一元)	最高优先级
**	
* / %	
+ - (二元)	
<< >> <<< >>>	
< <= > >=	
== != === !==	
& (二元)	

# Verilog HDL表达式

## --操作符

$\wedge$ $\wedge\sim$ $\sim\wedge$ (二元)	
(二元)	
&&	
?: (条件操作符)	最低优先级
{ } { }	

# Verilog HDL表达式

## --操作符

### 表达式中使用整数

- 在表达式中，可以使用整数作为操作数，一个整数可以表示为：
  - 没有宽度，没有基数的整数，比如：12。
  - 没有宽度，有基数的整数，比如：'d12, 'sd12。
  - 有宽度，有基数的整数，比如：16'd12, 16'sd12。

# Verilog HDL表达式

## --操作符

- 对于一个没有基数标识的整数的负数值的理解不同于一个带有基数标识的整数的负数值。
- 对于没有基数表示的整数，将其理解为以二进制补码存在的负数。
- 带有无符号基数表示的一个整数，将其理解为一个无符号数。

# Verilog HDL表达式 --操作符

## 整数表达式中使用整数的Verilog HDL描述例子

integer IntA;

IntA = -12 / 3;                   // 结果是-4

IntA = -'d 12 / 3;               // 结果是1431655761 ( 12取反)/3(32位 )

IntA = -'sd 12 / 3;              // 结果是-4

IntA = -4'sd 12 / 3;            // -4'sd12 是四位的负数1100 , 等效于-4。 -(-4) = 4。 4/3=1

# Verilog HDL表达式

## --操作符

### 算术操作符

下表给出了二元操作符的定义列表

$a+b$	a加b
$a-b$	a减b
$a*b$	a乘b
$a/b$	a除b
$a\%b$	a模b
$a**b$	a的b次幂乘

# Verilog HDL表达式

## --操作符

- 整数除法截断任何小数部分。例如： $7/4$ 结果为1。
- 对于除法和取模运算，如果第二个操作数为0，则整个结果的值为x。
- 对于取模操作，取第一个操作数的符号。
- 取模操作符求出与第一个操作符符号相同的余数。 $7\%4$ 结果为3，而： $-7\%4$  结果为-3。

# Verilog HDL表达式

## --操作符

### □ 对于幂乘运算

- ◆ 当其中的任何一个数是实数时，结果的类型也为实数。
- ◆ 如果幂乘的第一个操作数为0，并且第二个操作数不是正数；或者第一个操作数实负数，第二个操作数不是整数，则没有定义其结果。



# Verilog HDL表达式

## --操作符

下表给出了幂乘操作符规则。

op1 op2	负数<-1	-1	零	1	正数>1
正数	op1**op2	op2是奇数->-1 op2是偶数->1	0	1	op1**op2
零	1	1	1	1	1
负数	0	op2是奇数->-1 op2是偶数->1	x	1	0

# Verilog HDL表达式

## --操作符

- 对于一元操作，其优先级大于二元操作。

下表给出了一元操作符

$+m$	一元加 $m$ （和 $m$ 一样）
$-m$	一元减 $m$

- 在算术操作符中，如果任何操作数的位值是x或z，那么整个结果为x。
- 算术表达式结果的长度由最长的操作数决定。在赋值语句中，算术操作结果的长度由操作符左端目标长度决定。

# Verilog HDL表达式

## --操作符

### 算术操作的Verilog HDL描述例子

$$10\%3=1$$

$$-10\%3=-1$$

$$3^{**}2=9$$

$$0*0=1$$

$$0^{**}-1=x$$

$$-3.0^{**}2.0=9.0$$

$$11\%3=2$$

$$11\%-3=2$$

$$2^{**}3=8$$

$$2.0^{**}-3'sb1=0.5$$

$$9^{**}0.5=3.0$$

$$12\%3=0$$

$$-4'd12\%3=1$$

$$2^{**}0=1$$

$$2^{**}-3'sb1=0$$

$$9.0^{**}(1/2)=1.0$$

# Verilog HDL表达式

## --操作符

下表给出了算术操作数对数据类型的理解。

数据类型	理解
无符号网络	无符号
有符号网络	有符号，二进制补码
无符号寄存器	无符号
有符号寄存器	有符号，二进制补码
整数	有符号，二进制补码
时间	无符号
实数、实时时间	有符号，浮点

# Verilog HDL表达式 --操作符

在表达式中使用整数和寄存器数据类型的Verilog HDL描述的例子

```
integer intA;
```

```
reg [15:0] regA;
```

```
reg signed [15:0] regS;
```

```
intA = -4'd12;
```

```
regA = intA / 3;    //表达式是-4, intA是整数数据类型, regA的  
                   值是65532
```

# Verilog HDL表达式

## --操作符

```
regal = -4'd12;           // regA是65524
intA = regA / 3;          // 表达式的值为21841, regA是寄存器 类型数据
intA = -4'd12 / 3;        // 表达式的结果为1431655761, 是一个32位的
                           // 寄存器数据
regA = -12 / 3;            // 表达式结果-4, 一个整数类型, regA 是 65532
regS = -12 / 3;           // 表达式结果-4。regS是有符号寄存器
regS = -4'sd12 / 3;       // 表达式结果1。-4'sd12为4
```

# Verilog HDL表达式

## --操作符

### 关系操作符

#### 关系操作符列表

$a < b$	a小于b
$a > b$	a大于b
$a \leq b$	a小于等于b
$a \geq b$	a大于等于b

# Verilog HDL表达式

## --操作符

关系操作符有下面特点：

- 关系操作符的结果为真（1）或假（0）。
- 如果操作数中有一位为X或Z，那么结果为X。
- 如果关系运算存在无符号数时，将表达式看作是无符号数。当操作数长度不同时，位宽较短的操作数将0扩展到宽度较大的操作数的位宽范围。
- 如果关系运算都是有符号数时，将表达式看作是有符号的。当操作数长度不同时，位宽较短的操作数将符号扩展到宽度较大的操作数的宽度范围。



# Verilog HDL表达式

## --操作符

- 所有关系运算符的优先级相同，但是比算术运算符的优先级要低。
- 如果操作数中有实数，则将所有操作数转换为实数。然后进行关系运算。

### 关系操作符的Verilog HDL描述例子

$a < \text{foo} - 1$  等价于  $a < (\text{foo} - 1)$

$\text{foo} - (1 < a)$  不等价于  $\text{foo} - 1 < a$

# Verilog HDL表达式

## --操作符

### 相等操作符

下表给出了相等关系操作符列表

<b>a===b</b>	<b>a等于b，包含x和z</b>
<b>a! ==b</b>	<b>a不等于b，包含x和z</b>
<b>a==b</b>	<b>a等于b，结果可能未知（比较不包括x和z）</b>
<b>a!=b</b>	<b>a不等于b，结果可能未知（比较不包括x和z）</b>

# Verilog HDL表达式

## --操作符

### 相等关系操作符有下面特点：

- 相等操作符有相同的优先级。
- 如果相等操作中存在无符号数，当操作数长度不同时，长度较短的操作数将0扩展到宽度较大的操作数的范围。
- 如果相等操作中都是有符号数，当操作数长度不同时，长度较短的操作数将符号扩展到宽度较大的操作数的范围。

# Verilog HDL表达式

## --操作符

- 如果操作数中间有实数，则将所有操作数都转换为实数。然后进行相等运算。
- 如果比较结果为假，则结果为0；否则结果为1。
  - 在===和!==比较中，值x和z严格按位比较。也就是说，不进行解释，并且结果一定可知。这个比较可用于case语句描述中。
  - 在==和!=比较中，值x和z具有通常的意义，且结果可以为x。也就是说，在逻辑比较中，如果两个操作数之一包含x或z，结果为未知的值（x）。这个用于逻辑比较中。

# Verilog HDL表达式

## --操作符

### 相等关系操作符Verilog HDL描述的例子

```
Data= 'b11x0;
```

```
Addr= 'b11x0;
```

那么：`Data==Addr`不定，也就是说值为x；但`Data===Addr`比较结果为真，也就是说值为1。

□ 如果操作数的长度不相等，长度较小的操作数在左侧添0补位，

例如：`2'b10 == 4'b0010`，与后面的表达式相同：`4'b0010 == 4'b0010`，结果为真（1）。

# Verilog HDL表达式

## --操作符

### 逻辑操作符

- 符号&&（逻辑与）和符号||（逻辑或）用于逻辑的连接。
- 逻辑比较的结果为1（真）或者0（假）。当结果模糊的时候，为x。&&（逻辑与）的优先级大于||（逻辑或）。逻辑操作的优先级低于关系操作和相等操作。
- 符号!（逻辑非）是一元操作符。
- 这些操作符在逻辑值0/1上操作。逻辑操作的结构为0或1

# Verilog HDL表达式 --操作符

## 逻辑关系操作的Verilog HDL描述例子1

假设alpha=237, beta=0

regA = alpha && beta; // regA设置为0

regB = alpha || beta; // regB设置为1



# Verilog HDL表达式 --操作符

## 逻辑关系操作的Verilog HDL描述例子2

`a < size-1 && b != c && index != lastone`

为了便于理解和查看设计，推荐使用下面的方法描述上面给出的逻辑操作

`(a < size-1) && (b != c) && (index != lastone)`

## 逻辑关系操作的Verilog HDL描述例子3

`if (!inword)` 也可以表示为：

`if (inword == 0)`



# Verilog HDL表达式

## --操作符

### 按位操作符

下表给出对于不同操作符按位操作的结果

&（二元按位与）	0	1	x	z
	0	0	0	0
	1	1	x	x
	x	x	x	x
	z	x	x	x
^（二元按位异或）	0	1	x	z
	0	1	x	x
	1	0	x	x
	x	x	x	x
	z	x	x	x
~(一元非)	1	0	x	x

# Verilog HDL表达式

## --操作符

(二元按位或)	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x
^~ (二元按位异或非)	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

如果操作数长度不相等，长度较小的操作数在最左侧添0补位。

例如：'b0110^'b10000，与如下式的操作相同：'b00110^'b10000，  
结果为'b10110。

# Verilog HDL表达式

## --操作符

### 归约操作符

□ 归约操作符在单一操作数的所有位上操作，并产生1位结果。归约操作符有：

#### ◆ & (归约与)

- 如果存在位值为0, 那么结果为0。
- 否则如果存在位值为x或z, 结果为x。
- 否则结果为1。

# Verilog HDL表达式

## --操作符

### ◆ ~& (归约与非)

- 与归约操作符&相反。

### ◆ |(归约或)

- 如果存在位值为1，那么结果为1。
- 否则如果存在位x或z，结果为x。
- 否则结果为0。

# Verilog HDL表达式

## --操作符

### ◆ $\sim$ | (归约或非)

- 与归约操作符|相反。

### ◆ $\wedge$ (归约异或)

- 如果存在位值为x或z, 那么结果为x。
- 否则如果操作数中有偶数个1, 结果为0。
- 否则结果为1。

### □ $\sim\wedge$ (归约异或非)

- 与归约操作符 $\wedge$ 正好相反。

# Verilog HDL表达式

## --操作符

归约异或操作符用于决定向量中是否有位为x。

下表给出了一元规约操作结果的列表。

操作数	$\&$	$\sim\&$	$ $	$\sim $	$\wedge$	$\sim\wedge$
4'b0000	0	1	0	1	0	1
4'b1111	1	0	1	0	0	1
4'b0110	0	1	1	0	0	1
4'b1000	0	1	1	0	1	0

# Verilog HDL表达式 --操作符

## 归约异或操作符的Verilog HDL描述的例子

假定：

`MyReg=4'b01x0;`

则：

`^MyReg`结果为x

上述功能使用如下的if语句检测：

`if (^MyReg===1'bx)`

`$display("There is an unknown in the vector MyReg !")`

# Verilog HDL表达式

## --操作符

注：

逻辑相等(==)操作符不能用于比较操作，这是因为逻辑相等操作符比较操作将只会产生结果x。全等操作符期望的结果为值1。



# Verilog HDL表达式

## --操作符

### 移位操作符

移位操作符包括：

- << (逻辑左移)
- >> (逻辑右移)
- <<< (算术左移)
- >>> (算术右移)

移位操作符左侧的操作数将移动右侧操作数所指定的位数，它是一个逻辑移位。空闲位补0。如果右侧操作数的值为x或z，则移位操作的结果为x。

# Verilog HDL表达式

## --操作符

### 移位操作符的Verilog HDL描述例子1

假设start的值为4'b0001。

```
module ashift;
```

```
reg [3:0] start, result;
```

```
initial begin
```

```
    start = 1;
```

```
    result = (start << 2); //结果是4'b0100
```

```
end
```

```
endmodule
```

# Verilog HDL表达式 --操作符

## 移位操作符的Verilog HDL描述例子2

假设start的值为4'b1000。

```
module ashift;
```

```
reg signed [3:0] start, result;
```

```
initial begin
```

```
start = 4'b1000;
```

```
result = (start >>> 2);    //结果是1110
```

```
end
```

```
endmodule
```

# Verilog HDL表达式

## --操作符

### 条件操作符

条件操作符将根据条件表达式的值来选择表达式，格式如下：

**`cond_expr ? expr1:expr2`**

- 如果`cond_expr`为真(即值为1)，选择`expr1`。
- 如果`cond_expr`为假(值为0)，选择`expr2`。
- 如果`cond_expr`为x或z，结果将是按以下逻辑`expr1`和`expr2`按位操作的值：0与0得0，1与1得1，其余情况为x。

# Verilog HDL表达式 --操作符

## 条件操作符的Verilog HDL描述例子1

```
wire[0:2]Student=Marks>18 ? Grade_A:Grade_C;
```

计算表达式Marks>18的值:

- (1) 如果结果为真, 则将Grade\_A赋值给Student。
- (2) 如果结果为假, 则将Grade\_C赋值给Student。

## 条件操作符的Verilog HDL描述例子2

```
always#5 Ctr=(Ctr!=25)?(Ctr+1):5;
```

过程赋值中的表达式表明:

- (1) 如果Ctr不等于25, 则将Ctr的值加1赋给Ctr。
- (2) 如果Ctr等于为25, 则将Ctr值重新置为5

# Verilog HDL表达式

## --操作符

### 连接和复制操作

连接操作是将位宽较小的表达式合并形成位宽较大的表达式的一种操作。其描述格式如下：

**{expr1,expr2,...,exprN}**

由于非定长常数的位宽未知,所以不允许连接非定长常数。

# Verilog HDL表达式

## --操作符

复制操作就是将一个表达式复制多次的操作，其描述格式如下：

```
{ replication_constant {expr}}
```

其中：

□ replication\_constant

为非负数、非z和非x的常数，表示复制的次数。

□ expr

为需要复制的表达式。

# Verilog HDL表达式

## --操作符

注：

包含有复制的连接表达式，不能出现在分配的左侧操作数，也不能连接到output或者input端口上。



# Verilog HDL表达式 --操作符

## 连接操作的Verilog HDL描述的例子

`{a, b[3:0], w, 3'b101}`

等效于

`{a, b[3], b[2], b[1], b[0], w, 1'b1, 1'b0, 1'b1}`

## 复制操作的Verilog HDL描述的例子

`{4{w}}`

等效于

`{w, w, w, w}`

# Verilog HDL表达式 --操作符

## 复制和连接操作的Verilog HDL描述的例子

$\{b, \{3\{a, b\}\}\}$

等效于

$\{b, a, b, a, b, a, b\}$

# Verilog HDL表达式

## --操作符

复制操作可以复制值为0的常数。在参数化代码时，这是非常有用的。

带有0复制常数的复制，被认为是大小为0，并且被忽略。这样一个复制，只能在至少有一个连接操作数是正数的连接中。

# Verilog HDL表达式 --操作符

## 复制和连接操作分配限制的Verilog HDL描述的例子

```
parameter P = 32;
```

```
// 下面对于1到32是合法的。
```

```
assign b[31:0] = { {32-P{1'b1}}, a[P-1:0] } ;
```

```
// 对于P=32来说，下面是非法的。因为0复制单独出现在一个连接中。
```

```
assign c[31:0] = { {{32-P{1'b1}}}, a[P-1:0] }
```

```
// 对P=32来说，下面是非法的。
```

```
initial
```

```
$displayb({32-P{1'b1}}, a[P-1:0]);
```

# Verilog HDL表达式 --操作符

## 复制操作Verilog HDL描述的例子

```
result = {4{func(w)}} ;
```

等效为：

```
y = func(w) ;
```

```
result = {y, y, y, y} ;
```

# Verilog HDL表达式

## --操作符

**在表达式中需要指定一些类型的操作数。最简单的操作数包括：网络、变量、参数，还包括以下：**

- 如果要求一个向量网络、向量寄存器、整数或者时间变量或者参数的单个比特位时，则需要使用位选择操作数。
- 如果要求一个向量网络、向量寄存器、整数或者时间变量或者参数的某些相邻的比特位，则需要使用部分选择操作数。

# Verilog HDL表达式

## --操作符

- 可以引用数组元素或者一个数组元素的位选择/部分，作为一个操作数。
- 其它操作数的一个连接也可以指定为一个操作数。
- 一个函数调用也是一个操作数。

# Verilog HDL表达式

## --操作符

### 向量位选择和部分选择寻址

如果位选择/部分选择超出地址范围，或者位选择为x/z，则返回的结果为x。

一个标量，或者一个类型为实数或者实时时间的变量或者参数，位选择或者部分选择是无效的。



# Verilog HDL表达式

## --操作符

对于部分选择，有两种类型：

□ 常数部分选择。

表示为：

`vect[msb_expr:lsb_expr]`

其中：

`msb_expr`和`lsb_expr`为常数的整数表达式。

# Verilog HDL表达式

## --操作符

□ 索引部分选择。

表示为：

```
reg [15:0] big_vect;
```

```
reg [0:15] little_vect;
```

```
big_vect[lsb_base_expr +: width_expr]
```

```
little_vect[msb_base_expr +: width_expr]
```

```
big_vect[msb_base_expr -: width_expr]
```

```
little_vect[lsb_base_expr -: width_expr]
```

# Verilog HDL表达式

## --操作符

其中：

- ◆ `msb_expr`和`lsb_expr`为常数的整数表达式，可以在运行的时候改变。
- ◆ `width_expr`为正常数表达式。

# Verilog HDL表达式

## --操作符

### 数组部分选择的Verilog HDL描述例子

```
reg [31: 0] big_vect;
```

```
reg [0 :31] little_vect;
```

```
integer sel;
```

```
big_vect[ 0 +: 8]    // == big_vect[ 7 : 0]
```

```
big_vect[15 -: 8]    // == big_vect[15 : 8]
```

```
little_vect[ 0 +: 8]  // == little_vect[0 : 7]
```

```
little_vect[15 -: 8]  // == little_vect[8 :15]
```

```
dword[8*sel +: 8]    // 帶有固定宽度的变量部分选择
```

# Verilog HDL表达式 --操作符

例：数组初始化、位选择和部分选择的Verilog HDL描述

## 例子

```
reg [7:0] vect;
```

```
vect = 4;      // 用00000100填充，msb是7，lsb是0
```

- ◆ 如果addr=2，则vect[addr]返回1。
- ◆ 如果addr超过范围，则vect[addr]返回x。
- ◆ 如果addr是0、1、3~7，则vect[addr]返回0。
- ◆ vect[3:0]返回0100。

# Verilog HDL表达式

## --操作符

- ◆ `vect[5:1]`返回00010。
- ◆ `vect[返回x的表达式]`返回x。
- ◆ `vect[返回z的表达式]`返回x。
- ◆ 如果addr的任何一位是x或者z，则addr的值为x。

# Verilog HDL表达式

## --操作符

### 数组和存储器寻址

对于

```
reg [7:0] mem_name[0:1023];
```

存储器地址表示为：

```
mem_name[addr_expr]
```

其中：

- `addr_expr`为任意整数表达式。
- `mem_name[mem_name[3]]`表示存储器的间接寻址。

# Verilog HDL表达式 --操作符

## 存储器寻址Verilog HDL描述的例子

```
reg [7:0] twod_array[0:255][0:255];
```

```
wire threed_array[0:255][0:255][0:7];
```

```
twod_array[14][1][3:0] //访问字的低四位
```

```
twod_array[1][3][6] //访问字的第6位
```

```
twod_array[1][3][sel] //使用可变的位选择
```

```
threed_array[14][1][3:0] //非法
```



# Verilog HDL表达式

## --操作符

**字符串** 字符串是双引号内的字符序列，用一串8位二进制ASCII码的形式表示，每一个8位二进制ASCII码代表一个字符。

Verilog HDL的操作符均可操作字符串。当给字符串所分配的值小于所声明字符串的位宽时，用0补齐左侧。

# Verilog HDL表达式 --操作符

## 字符串Verilog HDL描述的例子

```
module string_test;
```

```
reg [8*14:1] stringvar;
```

```
initial begin
```

```
    stringvar = "Hello world";
```

```
    $display("%s is stored as %h", stringvar, stringvar);
```

```
    stringvar = {stringvar, "!!!"};
```

```
    $display("%s is stored as %h", stringvar, stringvar);
```

```
end
```

```
endmodule
```

# Verilog HDL表达式 --操作符

仿真结果表示为：

Hello world is stored as 00000048656c6c6f20776f726c64

Hello world!!! is stored as 48656c6c6f20776f726c64212121

# Verilog HDL表达式

## --操作符

Verilog HDL中，所支持的字符串操作包括：复制、连接和比较。

- 通过分配实现复制。
- 通过连接操作符实现连接。
- 通过相等操作符实现比较。

当操作向量寄存器内字符串的值时，寄存器应该至少为 $8*n$ 比特位（ $n$ 是ASCII字符的个数），用于保存 $n$ 个8位的ASCII码。

# Verilog HDL表达式 --操作符

## 字符串连接的Verilog HDL描述

**initial begin**

s1 = "Hello";

s2 = " world!";

if ({s1,s2} == "Hello world!")

\$display("strings are equal");

**end**

# Verilog HDL表达式 --操作符

其中：

`s1 = 000000000048656c6c6f`

`s2 = 00000020776f726c6421`

`{s1,s2}=000000000048656c6c6f00000020776f726c6421`

注：

对于空字符串“”，将其认为和ASCII NUL(“\0”)等效，其值为0，而不同于字符串“0”。

# Verilog HDL表达式

## --操作符

**延迟表达式**      Verilog HDL中，延迟表达式的格式为用圆括号括起来的三个表达式，这三个表达式之间用冒号分隔开。

三个表达式依次代表最小、典型、最大延迟时间值。

# Verilog HDL表达式

## --操作符

延迟表达式的Verilog HDL描述例子。

$(a:b:c)+(d:e:f)$

表示：

- 最小延迟值为 $a+d$ 的和。
- 典型延迟值为 $b+e$ 的和。
- 最大延迟值为 $c+f$ 的和。



# Verilog HDL表达式 --操作符

分配min:typ:max格式值的Verilog HDL描述例子

```
val - (32'd 50: 32'd 75: 32'd 100)
```

# Verilog HDL表达式

## --表达式的位宽

为了对表达式求值时得到可靠的结果，控制表达式的位宽是非常重要的。

□ 在某些情况下采取最简单的解决方法。

◆ 比如：如果指定了两个16位的寄存器矢量的位排序方式和操作，那么结果就是一个16位的值。

# Verilog HDL表达式

## --表达式的位宽

□ 然而，在某些情况下，究竟有多少位参与表达式求值或者结果有多少位，并不容易看出来。

◆ 例如：两个16位操作数之间的算术加法，是应该使用16位求值还是该使用17位（允许进位位溢出）求值？

答案取决于被建模器件的类型以及那个设备是否处理进位位溢出来决定。

Verilog HDL利用操作数的位宽来决定有多少位参与表达式的求值。

# Verilog HDL表达式

## --表达式的位宽

### 表达式长度Verilog HDL描述的例子

`reg [15:0] a, b;`    // 16位寄存器类型

`reg [15:0] sumA;`    // 16位寄存器类型

`reg [16:0] sumB;`    // 17位寄存器类型

`sumA = a + b;`    // 结果16位

`sumB = a + b;`    // 结果17位

# Verilog HDL表达式

## --表达式的位宽

控制表达式位宽的规则已经公式化，因此在大多数实际情况下，都有一个简单的解决方法。

- 表达式位宽是由包含在表达式内的操作数和表达式所处的环境决定的。
- 自主表达式的位宽由它自身单独决定，比如延迟表达式。
- 环境决定型表达式的位宽由该表达式自己的位宽和它所处的环境来决定，比如：一个赋值操作中右侧表达式的位宽由它自己的位宽和赋值符左侧的位宽来决定。

# Verilog HDL表达式

## --表达式的位宽

下表说明了表达式的形式如何决定表达式结果的位宽。表中i、j、k都表示单操作数的表达式，而L(i)代表表达式i的位宽，op代表操作符。

表达式	结果值位宽	说明
不定长常数	与整数相同	
定长常数	与给定的位宽相同	
i op j, 操作符op为: + - * / % &   ^ ^~ or ~^	max (L(i),L(j))	

# Verilog HDL表达式

## --表达式的位宽

op i, 操作符op为: + - ~	L(i)	
i op j, 操作符op为: == != == != &&    > >= < <=	1位	在求表达式值时, 每个操作数的位宽都先变为max (L(i),L(j))
op i, 操作符op为: & ~&   ~ ^ ^~or ~^	1位	所有操作数都是自主表达式
i op j, 操作符op为: >> << ** >>> <<<	L(i)	j是自主表达式
i?j:k	max (L(j),L(k))	i是自主表达式
{i,...,j}	L(i)+...+L(j)	所有操作数都是自主表达式
{i{j,...,k}}	I*(L(i)+...+L(j))	所有操作数都是自主表达式

# Verilog HDL表达式

## --表达式的位宽

在表达式求值过程中，中间结果应当采用具有最大位宽操作数（如果是在复制语句中，也包括赋值符的左侧）的位宽。在表达式求值过程中要注意避免丢失数据的重要性。

保护进位位的Verilog HDL描述的例子

```
reg [15:0] a, b, answer; // 16-bit regs
```

```
answer = (a + b) >> 1; //不能正常操作
```



# Verilog HDL表达式

## --表达式的位宽

### 自主表达式的Verilog HDL描述的例子

```
reg [3:0] a;          $display('a*b=%h', a*b);
reg [5:0] b;          c = {a**b};
reg [15:0] c;         $display('a**b=%h', c);
initial begin         c = a**b;
a = 4'hF;             $display('c=%h', c);
b = 6'hA;             end
```

# Verilog HDL表达式

## --表达式的位宽

仿真器的输出结果:

**a\*b=16** // 由于长度为6, 所以'h96被截断到'h16。

**a\*\*b=1** // 表达式的长度为4

**c=ac61** // 表达式的长度为16

# Verilog HDL表达式

## --有符号表达式

为了得到可靠的结果，控制表达式的符号是非常重要的。可以使用两个系统函数来处理类型的表示：

- `$signed()`

返回相同宽度的有符号的值。

- `$unsigned()`

返回相同宽度的无符号的值。

# Verilog HDL表达式

## --有符号表达式

调用系统函数进行符号转换Verilog HDL描述的例子

```
reg [7:0] regA, regB;
```

```
reg signed [7:0] regS;
```

```
regA = $unsigned(-4);    // regA = 8'b11111100
```

```
regB = $unsigned(-4'sd4); // regB = 8'b00001100
```

```
regS = $signed (4'b1100); // regS = -4
```

# Verilog HDL表达式

## --有符号表达式

下面是表达式符号类型规则：

- 表达式的符号类型仅仅取决于操作数，与LHS（左侧）值无关。
- 简单十进制格式数值是有符号数。
- 基数格式数值是无符号数，除非符号（s）用于技术说明符。
- 无论操作数是何类型，其位选择结果为无符号型。

# Verilog HDL表达式

## --有符号表达式

- 无论操作数是何类型，其部分位选择结果为无符号型，即使部分位选择指定了一个完整的矢量。
- 无论操作数是何类型，连接（或复制）操作的结果为无符号型。
- 无论操作数是何类型，比较操作的结果（1或0）为无符号型。
- 通过类型强制转换为整型的实数为有符号型。

# Verilog HDL表达式

## --有符号表达式

- 任何自主操作数的符号和位宽由操作数自己决定，独立于表达式的其余部分。
- 对于非自主操作数遵循下面的规则：
  - ◆ 如果有任何操作数为实型，则结果为实型。
  - ◆ 如果有任何操作数为无符号型，则结果为无符号型。
  - ◆ 如果所有操作数为有符号型，则结果为有符号型。

# Verilog HDL表达式

## --分配和截断

如果右操作数的长度大于左操作数的长度，则将右操作数的MSB丢弃，以进行长度匹配。

当出现长度不匹配时，并不要求实现过程警告或者报告和分配长度不匹配的任何错误。截断符号表达式的符号位，可能会改变结果的符号。



# Verilog HDL表达式

## --分配和截断

### 位宽不匹配分配Verilog HDL描述的例子1

```
reg [5:0] a;
```

```
reg signed [4:0] b;
```

```
initial begin
```

```
    a = 8'hff;    //分配完后, a = 6'h3f
```

```
    b = 8'hff;    //分配完后, b = 5'h1f
```

```
end
```

# Verilog HDL表达式 --分配和截断

## 位宽不匹配分配Verilog HDL描述的例子2

```
reg [0:5] a;
```

```
reg signed [0:4] b, c;
```

```
initial begin
```

```
    a = 8'sh8f;    //分配完后, a = 6'h0f
```

```
    b = 8'sh8f;    //分配完后, b = 5'h0f
```

```
    c = -113;      //分配完后, c = 15
```

```
end
```

# Verilog HDL表达式

## --分配和截断

### 长度不匹配分配Verilog HDL描述的例子3

```
reg [7:0] a;
```

```
reg signed [7:0] b;
```

```
reg signed [5:0] c, d;
```

```
initial begin
```

```
    a = 8'hff;
```

```
    c = a;           //分配完后, c = 6'h3f
```

```
    b = -113;
```

```
    d = b;           //分配完后, d = 6'h0f
```

```
end
```