第4章 表 达 式

本章讲述在Verilog HDL中编写表达式的基础。

表达式由操作数和操作符组成。表达式可以在出现数值的任何地方使用。

4.1 操作数

操作数可以是以下类型中的一种：

1) 常数
2) 参数
3) 线网
4) 寄存器
5) 位选择
6) 部分选择
7) 存储器单元
8) 函数调用

4.1.1 常数

前面的章节已讲述了如何书写常量。下面是一些实例。

```
256,7                  非定长的十进制数。
4'b10_11, 8'h0A        定长的整型常量。
'b1, 'hFBA             非定长的整数常量。
90.00006                 实数型常量。
"BOND"              串常量；每个字符作为8位ASCII值存储。
```

表达式中的整数值可被解释为有符号数或无符号数。如果表达式中是十进制整数，例如，12被解释为有符号数。如果整数是基数型整数（定长或非定长），那么该整数作为无符号数对待。下面举例说明。

```
12是01100的5位向量形式（有符号）
-12是10100的5位向量形式（有符号）
5'b01100是十进制数12（无符号）
5'b10100是十进制数20（无符号）
4'd12是十进制数12（无符号）
```

更为重要的是对基数表示或非基数表示的负整数处理方式不同。非基数表示形式的负整数作为有符号数处理，而基数表示形式的负整数值作为无符号数。因此- 44和-6'o54 （十进制的44等于八进制的54）在下例中处理不同。

```
integer Cone;
...
Cone = -44/4
Cone = -6'o54/ 4;
```

注意－44和－6'o54以相同的位模式求值；但是－44作为有符号数处理，而－6'o54作为无符

号数处理。因此第一个字符中Cone的值为－11，而在第二个赋值中Cone的值为1073741813[⊖]。

4.1.2 参数

前一章中已对参数作了介绍。参数类似于常量，并且使用参数声明进行说明。下面是参数说明实例。

```
parameter LOAD = 4'd12, STORE = 4'd10;
```

LOAD 和*STORE*为参数的例子，值分别被声明为 12和10。

4.1.3 线网

可在表达式中使用标量线网（1位）和向量线网（多位）。下面是线网说明实例。

```
wire [0:3] Prt;   //Prt 为4位向量线网。
wire Bdq;          /Bbq 是标量线网。
```

线网中的值被解释为无符号数。在连续赋值语句中，

```
assign Prt = -3;
```

*Prt*被赋于位向量1101，实际上为十进制的13。在下面的连续赋值中，

```
assign Prt = 4'HA;
```

*Prt*被赋于位向量1010，即为十进制的10。

4.1.4 寄存器

标量和向量寄存器可在表达式中使用。寄存器变量使用寄存器声明进行说明。例如：

```
integer TemA, TemB;
reg [1:5] State;
time Que [1:5];
```

整型寄存器中的值被解释为有符号的二进制补码数，而 reg寄存器或时间寄存器中的值被解释为无符号数。实数和实数时间类型寄存器中的值被解释为有符号浮点数。

```
TemA = -10;     /TemA值为位向量10110，是10的二进制补码。
TemA = 'b1011;    /TemA值为十进制数11。

State = -10;      /State值为位向量10110，即十进制数22。
State = 'b1011;    /State值为位向量01011，是十进制值11。
```

4.1.5 位选择

位选择从向量中抽取特定的位。形式如下：

```
net_or_reg_vector [bit_select_expr]
```

下面是表达式中应用位选择的例子。

```
State [1] && State [4]        /寄存器位选择。
Prt [0] | Bbq                   线网位选择。
```

如果选择表达式的值为x、z，或越界，则位选择的值为x。例如*State* [x]值为x。

4.1.6 部分选择

在部分选择中，向量的连续序列被选择。形式如下：

⊖ 因为Cone 为非定长整型变量，基数表示形式的负数在机内以补码形式出现。—— 译者注

```
net_or_reg_vector [msb_const_expr:lsb_const_expr]
```

其中范围表达式必须为常数表达式。例如。

```
State [1:4]          /寄存器部分选择。
Prt [1:3]            /线网部分选择。
```

选择范围越界或为x、z时，部分选择的值为x。

4.1.7 存储器单元

存储器单元从存储器中选择一个字。形式如下：

```
memory [word_address]
```

例如：

```
reg [1:8] Ack, Dram [0:63];
. . .
Ack = Dram [60];   /存储器的第60个单元。
```

不允许对存储器变量值部分选择或位选择。例如，

```
Dram [60] [2]        不允许。
Dram [60] [2:4]      也不允许。
```

在存储器中读取一个位或部分选择一个字的方法如下：将存储器单元赋值给寄存器变量，然后对该寄存器变量采用部分选择或位选择操作。例如， *Ack* [2] 和*Ack* [2:4]是合法的表达式。

4.1.8 函数调用

表达式中可使用函数调用。函数调用可以是系统函数调用（以 $字符开始）或用户定义的函数调用。例如：

```
$time + SumOfEvents (A, B)
/*$time是系统函数，并且SumOfEvents是在别处定义的用户自定义函数。*/
```

第10章将详细介绍函数。

4.2 操作符

Verilog HDL中的操作符可以分为下述类型：

1) 算术操作符
2) 关系操作符
3) 相等操作符
4) 逻辑操作符
5) 按位操作符
6) 归约操作符 ⊖
7) 移位操作符
8) 条件操作符
9) 连接和复制操作符

下表显示了所有操作符的优先级和名称。操作符从最高优先级（顶行）到最低优先级（底行）排列。同一行中的操作符优先级相同。

⊖ 归约操作符为一元操作符，对操作数的各位进行逻辑操作，结果为二进制数。 —— 译者

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。下面的表达式：

```
A + B - C
```

等价于：

```
(A + B ) - C            //自左向右
```

而表达式：

```
A ? B : C ? D : F
```

等价于：

```
A ? B : (C ? D : F)     //从右向左
```

圆扩号能够用于改变优先级的顺序，如以下表达式：

```
(A ? B : C) ? D : F
```

4.2.1 算术操作符

算术操作符有：

- +（一元加和二元加）
- －（一元减和二元减）
- *（乘）
- /（除）
- %（取模）

整数除法截断任何小数部分。例如：

```
7/4   结果为 1
```

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
```

如果算术操作符中的任意操作数是X或Z，那么整个结果为X。例如：

```
'b10x1 + 'b01111结果为不确定数'bxxxxx
```

1. 算术操作结果的长度

算术表达式结果的长度由最长的操作数决定。在赋值语句下，算术操作结果的长度由操作符左端目标长度决定。考虑如下实例：

```
reg [0:3] Arc, Bar, Crt;
reg [0:5] Frx;
. . .
Arc = Bar + Crt;
Frx = Bar + Crt;
```

第一个加的结果长度由 *Bar*，*Crt*和*Arc*长度决定，长度为 4位。第二个加法操作的长度同样由*Frx*的长度决定（*Frx*、*Bat*和*Crt*中的最长长度），长度为6位。在第一个赋值中，加法操作的溢出部分被丢弃；而在第二个赋值中，任何溢出的位存储在结果位 Frx[1]中。

在较大的表达式中，中间结果的长度如何确定？在 Verilog HDL中定义了如下规则：表达式中的所有中间结果应取最大操作数的长度（赋值时，此规则也包括左端目标）。考虑另一个实例：

```
wire [4:1] Box, Drt;
wire [1:5] Cfg;
wire [1:6] Peg;
wire [1:8] Adt;
. . .
assign Adt = (Box + Cfg) + (Drt + Peg);
```

表达式左端的操作数最长为 6，但是将左端包含在内时，最大长度为 8。所以所有的加操作使用8位进行。例如：*Box*和*Cfg*相加的结果长度为8位。

2. 无符号数和有符号数

执行算术操作和赋值时，注意哪些操作数为无符号数、哪些操作数为有符号数非常重要。无符号数存储在：

- 线网
- 一般寄存器
- 基数格式表示形式的整数

有符号数存储在：

- 整数寄存器
- 十进制形式的整数

下面是一些赋值语句的实例：

```
reg [0:5] Bar;
integer Tab;
. . .
Bar = -4'd12;   /寄存器变量Bar的十进制数为52，向量值为110100。
Tab = -4'd12;   /整数Tab的十进制数为-12，位形式为110100。

-4'd12 / 4      /结果是1073741821。
-12 / 4         /结果是-3
```

因为Bar是普通寄存器类型变量，只存储无符号数。右端表达式的值为 'b110100（12的二进制补码）。因此在赋值后， Bar存储十进制值52。在第二个赋值中，右端表达式相同，值为 'b110100，但此时被赋值为存储有符号数的整数寄存器。 Tab存储十进制值－ 12（位向量为

110100)。注意在两种情况下，位向量存储内容都相同；但是在第一种情况下，向量被解释为无符号数，而在第二种情况下，向量被解释为有符号数。

下面为具体实例：

```
Bar = - 4'd12/4;
Tab = - 4'd12 /4;

Bar = - 12/4
Tab = - 12/4
```

在第一次赋值中，*Bar*被赋于十进制值61（位向量为111101)。而在第二个赋值中，*Tab*被赋于与十进制1073741821（位值为0011...11101)。*Bar*在第三个赋值中赋于与第一个赋值相同的值。这是因为*Bar*只存储无符号数。在第四个赋值中，*Bar*被赋于十进制值－3。

下面是另一些例子：

```
Bar = 4 - 6;
Tab = 4 - 6;
```

*Bar*被赋于十进制值62（－2的二进制补码)，而*Tab*被赋于十进制值－2（位向量为111110)。

下面为另一个实例：

```
Bar = -2 + (-4);
Tab = -2 + (-4);
```

*Bar*被赋于十进制值58（位向量为111010)，而*Tab*被赋于十进制值－6（位向量为111010)。

4.2.2 关系操作符

关系操作符有：

- >（大于）
- <（小于）
- >=（不小于）
- <=（不大于）

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。例如：

```
23 > 45
```

结果为假（0)，而：

```
52< 8'hxFF
```

结果为x。如果操作数长度不同，长度较短的操作数在最重要的位方向（左方）添 0补齐。例如：

```
'b1000 > = 'b01110
```

等价于：

```
'b01000 > = 'b01110
```

结果为假（0)。

4.2.3 相等关系操作符

相等关系操作符有：

- = =（逻辑相等）
- !=（逻辑不等）

• = = =（全等）

• != =（非全等）

如果比较结果为假，则结果为 0；否则结果为 1。在全等比较中，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。而在逻辑比较中，值 x和z具有通常的意义，且结果可以不为 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）。

如下例，假定：

```
Data = 'b11x0;
Addr = 'b11x0;
```

那么：

```
Data = = Addr
```

不定，也就是说值为 x，但：

```
Data = = = Addr
```

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
```

与下面的表达式相同：

```
4'b0010 = = 4'b0010
```

结果为真（1）。

4.2.4 逻辑操作符

逻辑操作符有：

• && (逻辑与)

• || (逻辑或)

• ！(逻辑非)

这些操作符在逻辑值 0或1上操作。逻辑操作的结构为 0或1。例如，假定：

```
Crd = 'b0; //为假
Dgs = 'b1; //为真
```

那么：

```
Crd && Dgs        结果为0（假）
Crd || Dgs        结果为1（真）
！ Dgs            结果为0（假）
```

对于向量操作，非0向量作为1处理。例如，假定：

```
A_Bus = 'b0110;
B_Bus = 'b0100;
```

那么：

```
A_Bus || B_Bus    结果为1
A_Bus && B_Bus    结果为 1
```

并且：

! A_Bus 与! B_Bus的结果相同。

结果为0。

如果任意一个操作数包含 x，结果也为 x。

!x　　结果为x

4.2.5 按位操作符

按位操作符有：

- ~（一元非）
- &（二元与）
- |（二元或）
- ^（二元异或）
- ~^, ^~（二元异或非）

这些操作符在输入操作数的对应位上按位操作，并产生向量结果。下表显示对于不同操作符按步操作的结果。

& 与	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

| 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

^ 异或	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

^~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

~ 非	0	1	x	z
	1	0	x	x

例如，假定，

```
A = 'b0110;
B = 'b0100;
```

那么：

A | B　　结果为0110
A & B　　结果为0100

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。

4.2.6 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；若如果存在位值为 x或z，结果为x；否则结果为1。

• ~& (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；如果存在位 x或z，结果为x；否则结果为0。

• ~| (归约或非)

与归约操作符|相反。

• ^ (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。

• ~^ (归约异或非)

与归约操作符^正好相反。

如下所示。假定，

```
A = 'b0110;
B = 'b0100;
```

那么：

```
|B          结果为1
& B         结果为0
~ A         结果为1
```

归约异或操作符用于决定向量中是否有位为 x。假定，

```
MyReg = 4'b01x0;
```

那么：

```
^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 !")
```

注意逻辑相等(==)操作符不能用于比较；逻辑相等操作符比较将只会产生结果 x。全等操作符期望的结果为值1。

4.2.7 移位操作符

移位操作符有：

• << (左移)

• >> (右移)

移位操作符左侧操作数移动右侧操作数表示的次数，它是一个逻辑移位。空闲位添 0补位。如果右侧操作数的值为x或z, 移位操作的结果为x。假定：

```
reg [0：7] Qreg；
. . .
Qreg = 4'b0111;
```

那么:

```
Qreg >> 2 是 8'b0000_0001
```

Verilog HDL中没有指数操作符。但是，移位操作符可用于支持部分指数操作。例如，如果要计算$Z^{NumBits}$的值，可以使用移位操作实现，例如:

```
32'b1 << NumBits  //NumBits必须小于32。
```

同理，可使用移位操作为2-4解码器建模，如

```
wire [0:3] DecodeOut = 4'b1 <<Address [0:1];
```

Address[0:1] 可取值0,1,2和3。与之相应，*DecodeOut*可以取值4'b0001、4'b0010、4'b0100和4'b1000，从而为解码器建模。

4.2.8 条件操作符

条件操作符根据条件表达式的值选择表达式，形式如下:

```
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。

如下所示:

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

计算表达式*Marks* > 18; 如果真, *Grade_A* 赋值为*Student*; 如果*Marks* < =18, *Grade_C* 赋值为*Student*。下面为另一实例：

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

过程赋值中的表达式表明如果 *Ctr*不等于25, 则加1；否则如果*Ctr*值为25时, 将*Ctr*值重新置为5。

4.2.9 连接和复制操作

连接操作是将小表达式合并形成大表达式的操作。形式如下:

```
{expr1, expr2, . . ,exprN}
```

实例如下所示:

```
wire [7:0] Dbus;
wire [11:0] Abus;

assign Dbus [7:4] = {Dbus [0], Dbus [1], Dbus[2], Dbus[3]};
       //以反转的顺序将低端4位赋给高端4位。
assign Dbus = {Dbus [3:0], Dbus [7:4]};
       //高4位与低4位交换。
```

由于非定长常数的长度未知, 不允许连接非定长常数。例如, 下列式子非法：

```
{Dbus,5}     //不允许连接操作非定长常数。
```

复制通过指定重复次数来执行操作。形式如下:

```
{repetition_number {expr1, expr2, ...,exprN}}
```

以下是一些实例：

```
Abus = {3{4'b1011}};   //位向量12'b1011_1011_1011)
Abus = {{4{Dbus[7]}}, Dbus}; /*符号扩展*/
```

{3{1'b1}} 结果为111
{3{Ack}} 结果与{Ack, Ack, Ack}相同。

4.3 表达式种类

常量表达式是在编译时就计算出常数值的表达式。通常，常量表达式可由下列要素构成：

1) 表示常量文字，如'b10和326。

2) 参数名，如RED的参数表明：

```
parameter RED = 4'b1110;
```

标量表达式是计算结果为1位的表达式。如果希望产生标量结果，但是表达式产生的结果为向量，则最终结果为向量最右侧的位值。

习题

1. 说明参数*GATE_DELAY*，参数值为5。
2. 假定长度为64个字的存储器，每个字8位，编写Verilog 代码，按逆序交换存储器的内容。即将第0个字与第63个字交换，第1个字与第62个字交换，依此类推。
3. 假定32位总线*Address_Bus*，编写一个表达式，计算从第11位到第20位的归约与非。
4. 假定一条总线*Control_Bus* [15:0]，编写赋值语句将总线分为两条总线：*Abus* [0:9]和*Bbus* [6:1]。
5. 编写一个表达式，执行算术移位，将*Qparity*中包含的8位有符号数算术移位。
6. 使用条件操作符，编写赋值语句选择*NextState*的值。如果*CurrentState*的值为*RESET*，那么*NextState*的值为*GO*；如果*CurrentState*的值为*GO*，则*NextState*的值为*BUSY*；如果*CurrentState*的值为*BUSY*；则*NextState*的值为*RESET*。
7. 使用单一连续赋值语句为图2-2所示的2-4解码器电路的行为建模。[提示：使用移位操作符、条件操作符和连接操作符。]
8. 如何从标量变量*A*，*B*，*C*和*D*中产生总线*BusQ*[0:3]？如何从两条总线*BusA* [0:3]和*BusY* [20:15]形成新的总线BusR[10:1]?