第四章 运算操作符和属性

- VHDL的运算操作符：
- 属性：
- 用户自定义属性
- 操作符扩展
- 通用属性语句
- 例子

4.1 运算操作符

6种预定义的运算操作符：

- 赋值运算符；
- 逻辑运算符；
- 算术运算符；
- 关系运算符；
- 移位运算符；
- 并置运算符；

- 赋值运算符

类型	操作符	功能
赋值运算符	<=	对 SIGNAL 赋值
	:=	对 VARIABLE、CONSTANT、GENERIC 赋值，也可以用于赋初始值
	=>	对矢量中的某些位赋值，或者对某些位之外的其它位（常用 OTHERS 表示）赋值

例：

```
signal x: std_logic;          variable y: std_logic_vector(3 downto 0);
signal w: std_logic_vector(0 TO 7);
 x<='1';       ----通过<=将 '1'赋给信号x;
 y:="0000";  ----通过：=将值 "0000"赋给变量y;
 w1<=(0=>'1', OTHERS=>'0');  --信号w1的赋值;
 w2<=(OTHERS=>'0'); --信号w2的赋值;
```

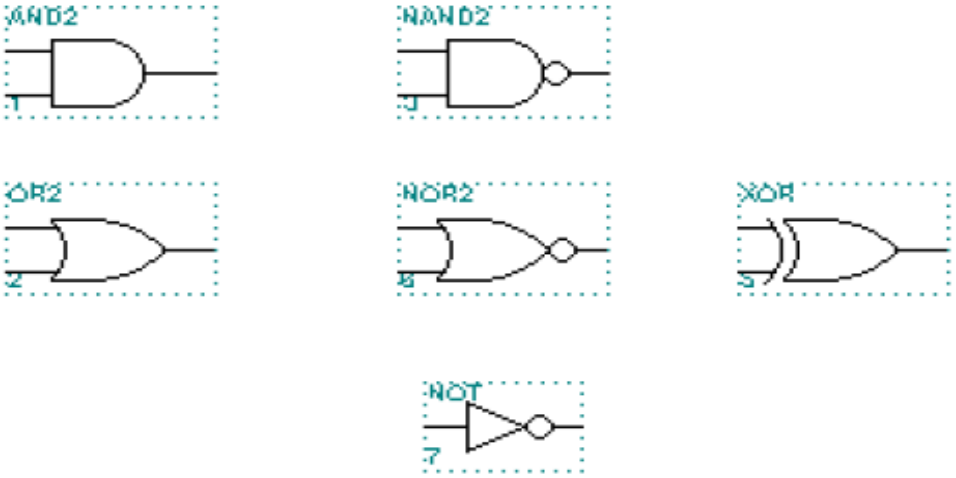
- 逻辑运算符

高运算优先级

类型	操作符	功能	操作数数据类型
逻辑操作符	NOT	非	BIT，STD_LOGIC，STD_ULOGIC或这些类型的扩展（BIT_VECTOR，STD_LOGIC_VECTOR，STD_ULOGIC_VECTOR）
	AND	与	
	OR	或	
	NAND	与非	
	NOR	或非	
	XOR	异或	

低运算优先级

- AND
- NAND
- OR
- NOR
- XOR
- NOT



算术运算符

类型	操作符	功能	操作数数据类型
算术操作符	+	加	INTEGER，SIGNED，REAL（不可综合），UNSIGNED；若声明ieee.std_logic_signed包集或声明了iee.std_logic_unsigned包集，还可对STD_LOGIC_VECTOR类型的数据进行加法和减法运算。 注意：对MOD和REM来说，操作数只能是整数类型。
	-	减	
	*	乘	
	/	除	
	**	指数运算	
	MOD	取模	
	REM	取余	
	ABS	取绝对值	

注意：书上的“可综合”是指EDA工具的工艺库（如IP库、基本器件库）中是否直接包含相应的功能模块或器件，如果包含，则称为可综合，否则就称为不可综合。对于某些运算功能，理论上是可以用电路实现的，但代价太大，不常使用，因此工艺库中一般就不包含这些功能器件，在这里就称为不可综合。另外，可综合的电路必须是规格固定的。

- 加法、减法、乘法可以综合成逻辑电路；（见例6.8/12.1/12.2等）
- 除法：除数为2^n时可通过移位操作实现（见例9.1），容易综合；其它类型除数时则比较复杂（见例9.4），是否能综合还取决于具体的除数；
- 指数运算：只有当底数和指数都是静态数值（常量或GENERIC参数）时才可综合；
- MOD和REM：只能用于整数类型。两者区别在于符号不同，如果有两个操作数a和b，表达式a REM b的符号与a相同；表达式a MOD b的符号与b相同。例如： 7 REM -2=1、-7 REM 2=-1、-7 MOD 2=1、7 MOD -2=-1。
  “mod”、“rem” 操作的综合限制：若右操作数为2n，则可用移位电路实现，即可综合的，其它情况则不可综合。
- ABS: 不可综合。

关系操作符

类型	操作符	功能	操作数数据类型
关系操作符	=	等于	关系运算符左右两边操作数的数据类型必须相同，适用于前面所述的所有数据类型。
	/=	不等于	
	<	小于	
	>	大于	
	<=	小于等于	
	>=	大于等于	

- 移位操作符

语法结构：<左操作数><移位操作符><右操作数>

类型	操作符	功能	操作数数据类型
移位操作符	sll	逻辑左移，右端空出的位置填 0	左操作数必须是 BIT_VECTOR 类型，右操作数必须是 INTEGER 类型（可加正/负号）
	srl	逻辑右移，左端空出的位置填 0	
	sla	算术左移，复制最右端的位填充到右端空出的位置	
	sra	算术右移，复制最左端的位填充到左端空出的位置	
	rol	循环逻辑左移，将左端移出的为依次填充到右端空出的位置	
	ror	循环逻辑右移，将右端移出的为依次填充到左端空出的位置	

例： 令x<=“01001”， 那么：

y<=x ssl 2; ---y<=“00100”;

y<=x sla 2; ---y<=“00111”;

y<=x rol 2; ---y<=“00101”;

- 并置运算符：

类型	操作符	功能	操作数数据类型
并置运算符	&	用于位的拼接	支持逻辑运算的任何数据类型，BIT，STD_LOGIC，STD_ULOGIC或这些类型的矢量扩展
	(,,)		

例：使用并置运算符对信号赋值

Z<=X & “1001”； ----若X<=“101”，则Z<=“1011001”；

Z<=(‘1’,’0’,’1’,’0’); ----Z<=“1010”;

位的连接的几种方法

```
Signal a, b, c, d : std_logic;
Signal q : std_logic_vector(3 downto 0);

q<=a&b&c&d;                    --正确使用并置运算符；

q<=(a, b, c, d);               --另一种位连接的方法；

q<=(3=>a, 2=>b, 1=>c, 0=>d);
        --指定位的脚标来进行位连接；

q<=a&a&c&d;          --如果相同的信号要使用多次；

q<=(1=>c, 0=>d, others=>a);
        --采用others进行位的连接；注意others的说
        --明只能放在最后。
```

各种操作符的优先级问题

- NOT和算术运算符中的ABS、**的优先级相同，是所有运算符中优先级最高的。
- 运算符*、/、MOD、REM的优先级相同 ，低于NOT、ABS和**运算符
- 六种移位运算符的优先级相同，高于关系运算符。
- 并置运算符的优先级与加、减运算符相同，高于移位运算符；

在VHDL中，左右没有优先组合的区别，一个表达式中如果有多个逻辑运算符，运算顺序的不同可能会影响运算结果，就需要用括号来解决组合顺序的问题。

4.2 属性（ATTRIBUTE）

- 属性是指从指定的客体或对象（如entity /type /architecture等）中获取关心的数据或信息。
- 利用属性可以使VHDL源代码更加简明扼要，易于理解；
- 语法： 对象’属性
- 预定义的属性：数值类属性和信号类属性

- 数值类属性：获取数组、块或一般数据的相关信息。
- VHDL预先定义的、可综合的数值类属性：

  left：索引的左边界值

  right：索引的右边界值

  high：索引的上限值

  low：索引的上限值

  length：索引的长度值

  range：索引的位宽范围

  reverse_range：索引的反向位宽范围

例：

variable my_vector : bit_vector (5 downto -5);

各属性如下：

my_vector’left	5
my_vector’right	-5
my_vector’high	5
my_vector’low	-5
my_vector’length	11
my_vector’range	(5 downto -5)
my_vector’reverse_range	(-5 to 5)

- 枚举类型数值的属性：通常是不可综合的

  VAL(pos)： 指定位置（pos）的值

  POS(value)：给定数值的位置序号

  LEFTOF(value)：给定数值的左侧值

  VAL(row,column)：位于行、列位置的值

信号类属性：VHDL预定义的，对于信号s：

EVENT： 如果s值发生了变化，则返回值为true，否则为false XXX， 常用于时钟信号的判定。

STABLE： 如果s值保持不变，则返回值为true，否则为false；

ACTIVE： 如果s值为‘1’，则返回值为true，否则为false；

QUIET<time>： 如果在指定的time内s值保持不变，则返回值为true，否则为false；

LAST_EVENT： 返回从上一次事件发生的时间到当前时间的时间差；

LAST_ACTIVE： 最后一次s=‘1’到当前所经历的时间长度值；

LAST_VALUE： 最后一次变化前s的值；

除EVENT和STABLE属性是可以综合的之外，其它的属性都不可综合，仅用于仿真。

例： 对信号clk是否出现上述沿进行判断

IF（clk’EVENT and clk=‘1’）..

IF（NOT clk’STABLE and clk=‘1’）..

4.3 用户自定义的属性

- 用户自定义属性的声明，语法如：

```
ATTRIBUTE attribute_name: attribute_type;
```

attribute_type可以是任何数据类型，包括预定义的数据类型。

- 用户自定义属性的描述，语法如：

```
ATTRIBUTE attribute_name OF target_name: class IS value;
```

class可以是数据类型、信号、变量、函数、实体或构造体等。

例：

attribute number_of_in； integer; --属性声明；

attribute number_of_in of nand3: signal is 3; --属性描述；

inputs<=nand3’number_of_in; --属性调用，返回值为3；

例：枚举类型编码：通常采用顺序编码的方式

TYPE color IS (red, green, blue, white);

默认的编码是：

red=“00”; green=“01”; blue=“10”; white=“11”;

可对编码次序进行修改：

attribute enum_coding of color: type is “11 00 10 01”;

(值可选，如 type is “1000 0001 0100 0010”)

4.4 操作符扩展（或重载操作符）：

即用户自定义操作符，可以与预定义的操作符具有相同的名称。对已存在的操作符重新定义，可进行不同类型操作数之间的运算。

例：对一个整数和一个1位的二进制数进行加法运算

```
function “+” (a: integer; b: bit) return integer is
begin
    if (b=‘1’) then return a+1；
    else return a;
    end if;
end “+”,
```

（续前） 对函数“+”的调用：

```
signal inp1,outp: integer RANGE 0 TO 15;
signal inp2: bit;
……
    outp<=3+inp1+inp2;
……
```

预定义的加法操作符（两个整数相加）

用户自定义的、经过扩展的加法运算操作符，可对一个整数和一个**bit**型数据进行加法运算

4.5 通用属性语句（GENERIC）

用于指定常规参数，所指定的参数是静态的，方便设计人员进行参数修改，可增加代码的灵活性和可重用性。

- generic语句必须在ENTITY中进行声明；
- generic语句所指定的参数是全局的，不仅可以在ENTITY内部使用，还可以在后面的整个设计中使用。
- 主要用来为设计实体指定参数，如端口宽度、器件延时等；

语法结构：

GENERIC （参数名:参数类型:=参数值）；

如：

```
entity my_entity is
   GENERIC (n:integer:=8);
   port (...);
end my_entity;
```

也可以在一个 GENERIC 语句中指定多个参数，如：

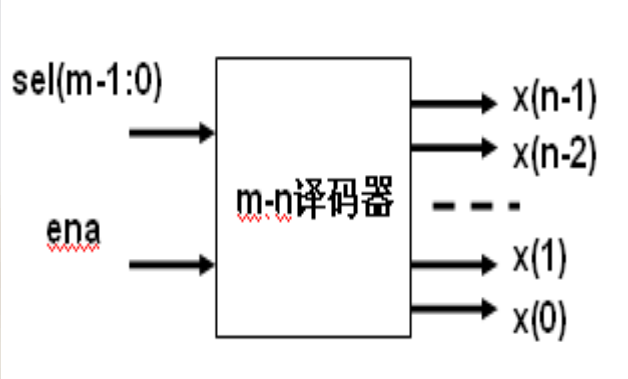
```
GENERIC (n: integer:=8; vector: bit_vector:="1001");
```

4.6 设计实例：

例1：通用译码器

•功能描述：

m-n译码器，$m=\log_2 n$



2-4 译码器的真值表

ena	sel	x
0	00	1111
1	00	1110
	01	1101
	10	1011
	11	0111

3-8 译码器的真值表

ena	sel	x
0	000	11111111
1	000	11111110
	001	11111101
	010	11111011
	011	11110111
	100	11101111
	101	11011111
	110	10111111
	111	01111111

- 端口描述

  两个输入端口：ena和sel(m-1:0)

  一个输出端口：x(n-1:0)

- 功能特征分析

1、根据真值表，确定ena/sel与x的关系：

当ena为‘0’时，x为“1111_1111”；

当ena为‘1’时， 8位的二进制数“1111_1111” 的某位变为‘0’，其它位保持‘1’。该位的确定取决于sel的值；

2、 确定端口规格：本例中m为3，n为8；

- 设计要点

1、如何根据sel的值确定需由‘1’变‘0’的位置? 分析真值表发现：将sel由二进制数转换成十进制数temp2， 则需由‘1’变为‘0’的位的位置为temp2。

2、如何将二进制数转换为十进制数？循环累加！

- 具体实现：

```
1-------------------------------------------------
2     library ieee;
3     use ieee.std_logic_1164.all;
4-------------------------------------------------
5     entity decoder is
6          port(  ena: IN std_logic;
7                 sel:  IN std_logic_vector(2 downto 0);
8                 x: OUT std_logic_vector(7 downto 0)  );
9     end decoder;
10----------------------------------------
```

```
11 architecture generic_decoder of decoder is
12 begin
13     process (ena, sel)
14         variable temp1: std_logic_vector( x'high downto 0 );
15         variable temp2: integer range 0 to x'high;
16     begin
17          temp1:=(others=>'1');
18          temp2:=0;
19          if (ena='1') then
20              FOR i IN sel'range loop
21                    if (sel(i)='1') then
22                         temp2:=2*temp2+1;
23                    else
24                         temp2:=2*temp2;
25                    end if;
26               END loop;
27               temp1(temp2):='0';
28          end if;
29          x<=temp1;
30      end process;
31end generic_decoder;
32----------------------------------------
```

指2 downto 0

过程？

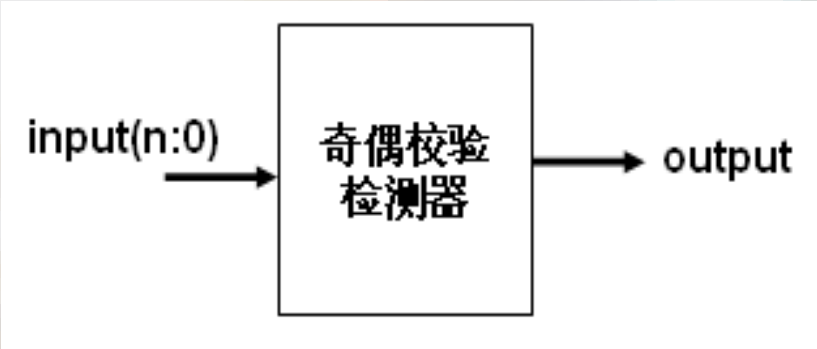
- 思考：

  如何使代码具有较强的通用性，以适应不同端口规格的译码器，即通用译码器？

GENERIC语句！

例2：通用奇偶校验检测器电路

- **功能描述：** 输入矢量中‘1’的个数为偶数时，电路输出‘0’，否则输出‘1’。



- 端口描述：

  一个输入端口：**input(n:0)**

  一个输出端口：**output**

- 功能特征分析

  1、需要统计输入矢量中‘1’的个数；

  2、功能固定，规格不固定，可以设计成通用构造体；

- 设计要点

  1、如何使用GENERIC语句来定义端口规格，以适应不同规格电路设计的需要？

  2、如何统计输入矢量中‘1’的个数？循环累加还是异或运算？代价如何？

- 具体实现：

```
1----------------------------------------------------
2 entity parity_det is
3          GENERIC (n: integer:=7);
4         port(  input: IN bit_vector(n downto 0);
5                    output: OUT BIT );
6  end parity_det;
7-----------------------------------------------
8   architecture parity OF parity_det is
9   begin
10      process(input)
11           variable temp: BIT;
12       begin
13           temp:='0';
14           FOR i IN input'range loop
15                temp:=temp XOR input(i);
16           END loop;
17           output<=temp;
18       end process;
19  end entity;
```

例3：通用奇偶校验发生器电路

- **功能描述：**

  输入矢量中‘1’的个数为偶数时，输出矢量中所增加的输出位的值为‘0’，否则输出‘1’。输出矢量中的其它位为输入矢量的值。

- 端口描述：

  一个输入端口：input(n-1:0)

  一个输出端口：output(n:0)

- 功能特征分析

1、输入矢量的宽度比输出矢量的宽度少一位；输出矢量的其它位由输入矢量直接赋值。

2、功能固定，规格不固定，可以设计成通用构造体。

3、需要统计输入矢量中‘1’的个数。

- 设计要点

1、如何使用GENERIC语句来定义端口规格，以适应不同规格电路设计的需要？

2、如何统计输入矢量中‘1’的个数？循环累加还是异或运算？代价如何？

- 具体实现：

```
1----------------------------------------------------
2 entity parity_gen is
3          GENERIC (n: integer:=7);
4        port(  input: IN bit_vector(n-1 downto 0);
5                  output: OUT bit_vector(n downto 0) );
6  end parity_gen;
7-----------------------------------------------
8   architecture parity OF parity_gen is
9   begin
10      process(input)
11           variable temp1: BIT;
12           variable temp2: BIT_VECTOR(output'range);
13       begin
14           temp1:='0';
15           FOR i IN input'range loop
16                temp1:=temp1 XOR input(i);
17                temp2(i):=input(i);
18           END loop;
19           temp2(output'HIGH):=temp1;
20           output<=temp2;
21       end process;
22  end entity;
```

小结

运算操作符

操作符类型	操作符	操作数类型
赋值运算	<= , := , =>	任意数据类型
逻辑运算	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_ULOGIC, STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR
算术运算	+, -, *, /, ** (mod, rem, abs)	INTEGER, SIGNED, UNSIGNED
比较运算	=, /=, <, >, <=, >=	任意数据类型
移位运算	ssl, srl, sla, sra, rol, ror	BIT_VECTOR
并置运算	&, (,,,)	STD_LOGIC, STD_LOGIC_VECTOR, SIGNED, STD_ULOGIC, UNSIGNED STD_ULOGIC_VECTOR

属　性

应用场合	属 性	返回值
常规类型数据	LOW	返回数组索引的下限值
	HIGH	返回数组索引的上限值
	LEFT	返回数组索引的左边界值
	RIGHT	返回数组索引的右边界值
	LENGTH	返回矢量的长度值
	RANGE	返回矢量的位宽范围
	REVERSE_RANGE	按相反的次序，返回矢量的位宽范围
枚举数据类型	VAL(pos)	返回指定位置（pos）的值
	POS(value)	给定数值（value），返回其位置序号
	LEFTOF(value)	给定数值（value），返回其左侧的值
	VAL(row,column)	返回给定行、列位置对应的值
信号	s'EVENT	如果 s 的值发生了变化，则返回值为 true，否则为 false；
	s'STABLE	如果 s 的值保持稳定没有发生变化，则返回值为 true，否则为 false；
	s'ACTIVE	如果当前 s 的值= '1'，则返回值为 true，否则为 false；

第4章 复习

1、VHDL的运算操作符有哪些？

2、VHDL中预定义的属性都是如何使用的？

3、VHDL中用户自定义属性的语法结构？

4、什么是操作符扩展，有何作用？

5、通用属性语句GENRIC的使用。

课后作业：

1、使用GENERIC语句改写例4.1，使其由3-8译码器成为通用译码器。

2、书后作业4.1题。