第八章 VHDL的可综合性

主讲人：徐向民教授

本章目录

- VHDL语言结构向硬件的映射
- VHDL类型
- VHDL对象
- 运算符
- 顺序语句
- 并行语句

VHDL语言结构向硬件的映射

✓EDA工业界普遍认为,有效的VHDL建模风格是控制综合结果最为有力的手段。

✓为了建立有效的VHDL代码，设计师应了解VHDL语言结构与综合结果的关系。

✓应该指出的是，由于综合算法的不同，对于同样的硬件描述，不同的CAD综合工具可能会得到不同的综合结果。

VHDL语言在创立时，主要是为了满足仿真的需要。自从VHDL被用于综合以来，都是对VHDL的子集进行处理，这就是所谓的可综合的VHDL子集。不同综合工具支持的可综合子集不尽相同，通常有如下要求：

（1）延时描述（after语句、 wait for语句）等被忽略。

现在的所有综合工具都忽略源代码中的延时语句，有些工具干脆把这些语句处理为语法错误。大部分工具忽略延时语句后，给出警告提示。而综合时间约束则在综合过程中通过综合命令输入。

（2）支持有限类型

VHDL具有丰富的类型定义，但是有些类型不具备硬件对应物，不可能被综合，如文件类型。

通常可综合类型包括位、位矢量、布尔量、整数、枚举类型、数组等。其余像浮点数类型、记录类型等只能得到有限支持，而时间类型等完全不能被综合。

（3）进程的书写要服从一定的限制。

在仿真时，VHDL进程可以任意书写。而在综合时，通常要求一个进程内只能有一个有效时钟，有的工具还有进一步的限制。

（4）可综合代码应该是同步式的设计。

现在的EDA综合工具普遍推荐使用同步设计风格，即整个芯片电路的状态只能在时钟信号有效时发生改变。

当然设计师也可能尝试其他风格的设计，如异步设计，但这时综合工具产生的结果往往还需要设计师进一步优化或调整。

VHDL类型

VHDL语言中的对象有常量（constant)、信号（signal)、和变量（variable)三种，它们都必须定义为如下某种类型。类型定义说明了对象可以使用的数值，并隐含表示了可以对其进行的操作。

1、可综合数据类型
2、可综合子集

VHDL类型

1、可综合数据类型

面向综合的建模都支持这样一些类型：枚举类型、整数、一维数组。比较先进的综合工具现在一般也可以处理二维数组和简单的记录类型。

（1）枚举类型

枚举类型通过列出所有可能的取值来定义，例如：

```
type Boolean is (FALSE , TRUE);
type State_type is (HALT,READY,RUN,ERROR);
type Std_ulogic is ('U','X','0','1','Z', '-');
```

以上Std_ulogic的定义实际是对‘0’ ‘1’等字符进行了重载，由于这个定义已经成为IEEE标准，因此综合时不会产生额外硬件。而对于抽象层次更高的Boolean和 State_type则需要进行状态编码。

一般来说，状态编码是把状态值编码为位矢量（如bit_vector），矢量长度是能够表示所有状态的最短位宽。

例如，State_type的4个状态值可以分别校编码为“00”，“01”，“10”和“11”。

VHDL类型

（2）整数类型

可综合的整数类型定义总是有界的，例如：

```
type  My_integer  is  Integer  range 0 to 255;
subtype  Byte_int  is  Integer  range -128 to 127;
```

对整数类型进行综合时，综合工具首先将其翻译为位矢量，矢量长度仍取能够满足需要的最短位宽。

建议类型定义时明确指出整数的范围，以便于综合工具进行优化。否则大部分综合工具按32位处理。

综合后的电路中，整数以矢量形式出现，但通常只能以整个矢量为单位访问，即不能单独访问每一位。

VHDL类型

（3）数组类型

现在的综合工具都能够处理一维数组，例如：

```
type  Word  is  arry  （31 downto 0)  of   Bit;
type  My_RAM  is  array  (1023 downto  0 )  of   Word;
```

对于Word类型，综合工具通常将其综合为总线。My_RAM类型实际是二维的，这种用两个一维数组代替一个两维数组是常用的综合建模技巧。现在先进的综合工具如synospys DC可以将其综合为RAM，一般的综合工具至少可以把它综合为寄存器。

VHDL类型

（4）记录类型

记录类型在定义复杂数据类型时非常方便，能够把不同数据类型的数据组织在一起统一访问。

但是，EDA工业界对综合工具是否应该支持记录类型还没有统一意见，因此大多数综合工具不提供这种能力或只能把组合了简单数据类型的记录进行综合。

VHDL类型

2、可综合子集

VHDL在1989年首次公布时，就提供了两个程序包：

Standard 和 TextIO

其中定义了各种预定义数据类型。

1992年，IEEE颁布了标准程序包Std_logic_1164，其中定义了9值数据类型Std_ulogic,即相应的决断类型Std_logic。

2004年，IEEE批准了一种修订标准IEEE 1076.6-2004，该标准提供了VHDL RTL综合子集的重要扩展。新改进版包括VHDL几乎每一个特性，能被用于在RTL级进行建模并综合。此处还包括触发器和锁存器建模的扩展语义导引。

用户将能够以多种不同风格编写一个RTL模型，每一种都符合标准。这项标准化将最终帮助RTL确认。

VHDL类型

该项标准主要支持以下类型的综合：

a) bit , boolean , bit_vector

b) character , string

c) integer

d) std_ulogic , std_ulogic_vector ,
std_logic , std_logic_vector

e) signed , unsignd

VHDL对象

VHDL语言中有三类对象，常量(constant)，变量(variable)，信号(signal)，它们是VHDL代码中的数据的载体。

1、常量

常量仅被计算一次。在很多情况下，可以通过使用常量引导综合器获得优化的结果。在综合过程中，常量被处理的方式很多，主要有下述情况：

1）用于描述真值表、ROM等，或被用于信号赋值，常量在综合时会形成对应的硬件。

2）作为算术运算的一个操作数出现时，综合工具常会对这一算术运算实施特定的优化措施。当然，这样一来常量与综合结果中的硬件就不是一一对应了。

例如：优化综合工具用左移一位实现乘2操作，右移一位实现除法操作。

VHDL对象

3）常量在作为条件表达式的一部分时综合工具会对整个语法结构进行布尔优化。

4）常量传播。在下面的VHDL代码中，由于数组ROM和ROM（5）的索引都是常量，因此WORD4实际上也成为常数，在进一步优化中，WORD4将作为常量被处理。这就是常量传播。

```
constant ROM : ROM_TYPE := Read("Rom_file.dat");
signal WORD4 : Bit_vector (3 downto 0);
begin
    WORD4 <= ROM(5);
```

VHDL对象

1、变量和信号

变量和信号有着不同的仿真行为,同样在综合过程中,它们也会产生不同的结果。

1）一般来说，尽量使用变量能够获得比较好的综合结果，因为这样做使得优化的余地较大。但要注意，并不是所有的综合工具都支持变量的综合。

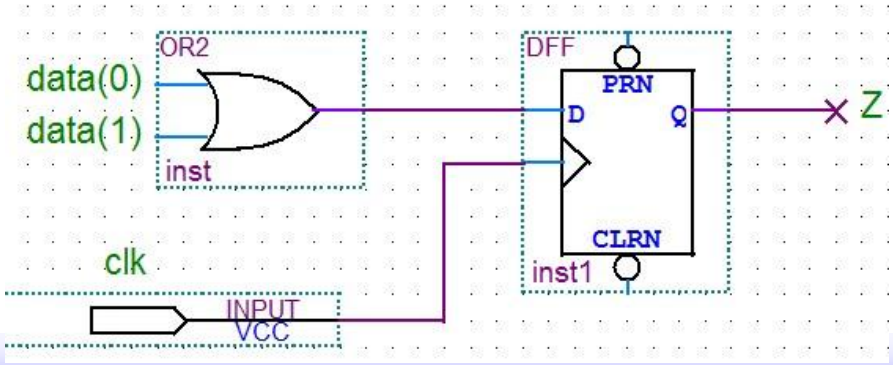
2）使用信号可以较好地保持综合前后在I/O上的一致性（这时把进程内对信号的读写统称为I/O），而且在需要锁存中间结果的时候，经常有必要使用信号。

下面用一个例子来说明变量与信号的不同综合结果：

VHDL对象

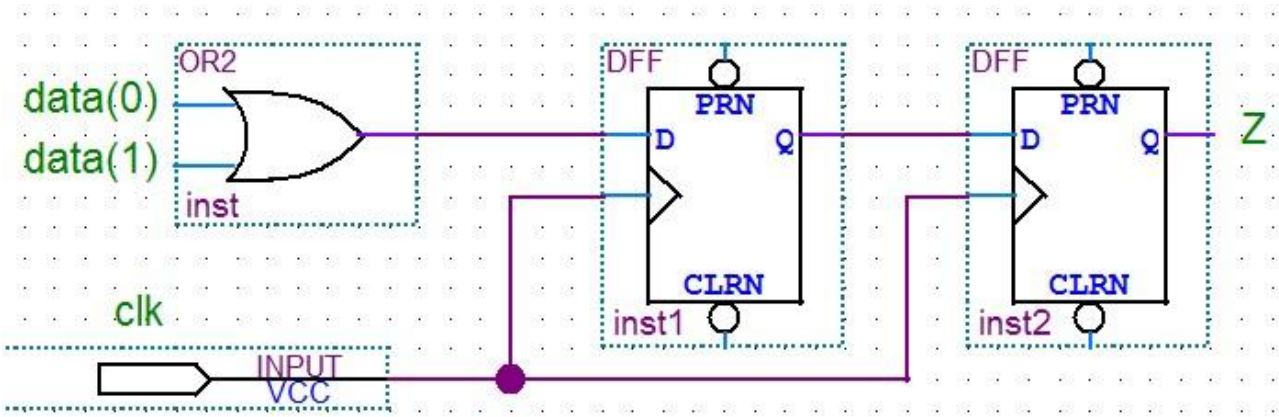
--结构体A，用变量实现算法

```
entity  var_sig is
    port(data : in  bit_vector (1  downto  0) ;   clk : in bit;    z : out bit);
    constant  k1 : bit_vector := “01”;
    constant  k2 : bit_vector := “10”;
end  var_sig;
architecture  A  of  var_sig  is
begin
     var : process
           variable  a1 , a2 :bit_vector (1  downto  0);
           variable  a3 : bit;
      begin
           wait until  clk = ‘1’ and clk’ event ;
           a1 := data  and  k1;
           a2 := data  and  k2;
           a3 := a1(0)  or  a2(1);
           z  <=  a3;
      end process var;
end  A
```



VHDL对象

```
--结构体B，用信号实现算法
architecture B of var_sig is
  signal a1 , a2 : bit_vector (1 downto 0);
  signal a3 : bit;
  begin
      a1 <= data and k1 ;
      a2 <= data and k2;
  sig : process
    begin
         wait until clk = '1' and clk' event;
          a3 <= a1(0) or a2(1);
          z  <= a3;
     end process sig;
end B
```



初值

VHDL中有三种初值：

由类型或子类型定义可以得到的默认初值，定义对象时明确指定的初值和进程入口处显示地赋予对象的初值。

```
--设置初值的三种情况
--type  states  is(rst , fi , id , ie);
signal state : states ;                --信号STATE的默认初值是RST;
…..
signal  z : bit_vector (3 downto 0) := “0000”;        --明确指定的初值
……
P1: process (A , B)
        variable v1 , v2 : std_logic ;
begin
        v1 := ‘0’ ;  v2 := ‘1’;                        --赋初值
        ……
end  process P1;
```

初值

✓以上三种初值的前两种只在仿真时有意义,在综合时将被忽略。

✓第三种形式将被综合器处理，形成对应电路。

✓在集成电路设计中，复位时赋予各个信号初值是很有必要的，否则很有可能出现在不定态。因此无论在仿真还是在综合时，都建议使用系统化的方式给信号和变量赋初值，即上述的在进程入口处显示地赋予对象的初值。

运算符

VHDL提供了丰富的运算符，表中分类列出所有VHDL预定义运算符及相应的优先。

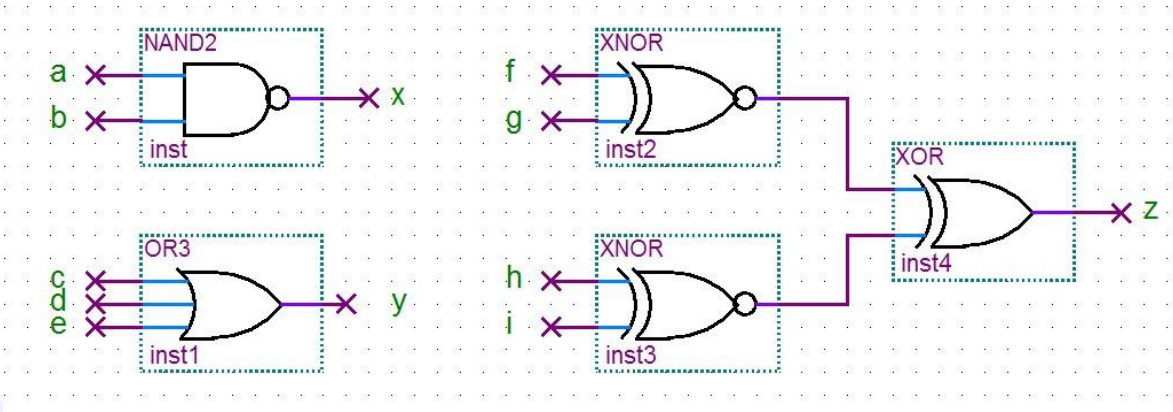
优先级顺序	运算操作符类型	操作符	功能
低	逻辑运算符	AND	逻辑与
		OR	逻辑或
		NAND	逻辑与非
		NOR	逻辑或非
		XOR	逻辑异或
	关系运算符	=	等号
		/=	不等号
		<	小于
		>	大于
		<=	小于等于
		>=	大于等于
	加、减、并置运算符	+	加
		-	减
		&	并置
	正负运算符	+	正
		-	负
	乘除运算符	*	乘
		/	除
		MOD	求模
		REM	取余
高	其他	**	指数
		ABS	取绝对值
		NOT	取反

运算符

1、逻辑运算符

逻辑运算符包括二元逻辑运算符以及NOT运算，操作数可以是bit和std_logic等类型的标量或同长度的矢量对象，也可以是boolean类型的对象。这些运算符综合时直接调用逻辑门单元实现即可，但经过优化后，这些运算符可能被合并或改变。

```
--这段VHDL代码中逻辑运算符综合的结果如图
Signal  x , a , b : bit_vector(3 downto 0);
Signal  y , c , d , e : std_logic;
Signal  z , f , g , h , I :boolean;
……
Begin
   x <= a nand b;
   y <= c or d or e;
   z <= (f xnor g) xor (h xnor i);
……
```



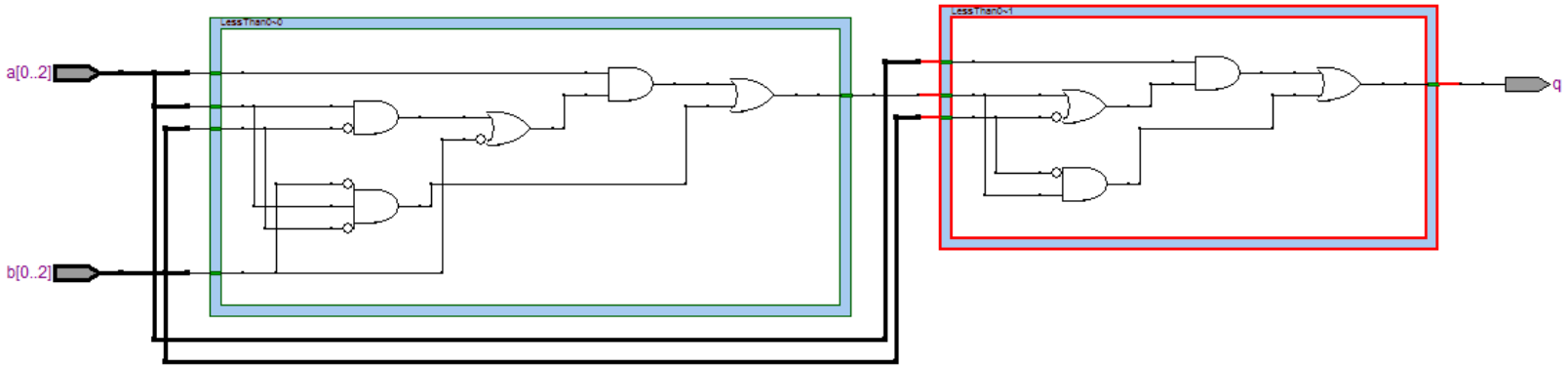
运算符

2、关系运算符

关系运算符的综合没有统一的方法，综合工具常利用被比较数的特点作特定的优化。

如下是对三位位宽的数据的”>”运算符的综合结果:

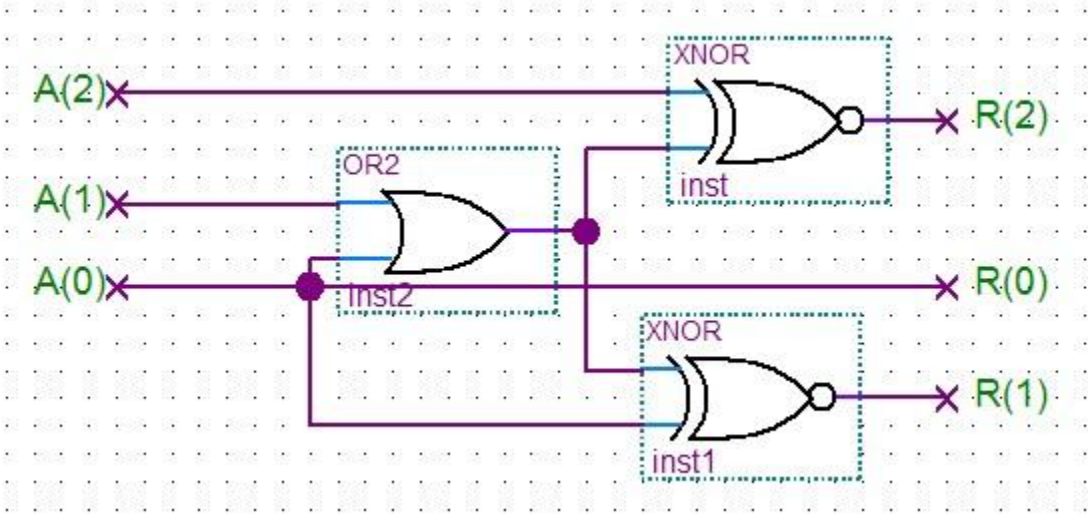
```
if   a>b then
        q <= '1';
    else
        q <= '0';
  end if ;
```



运算符

3、一元算术运算符

一元算术运算符有三个，即+（正），－（负）和abs（取绝对值）。对前两个运算符，综合工具大都可以用组合逻辑线路实现，如下的例子。abs运算符的处理比较复杂，大部分综合工具尚不能提供支持。



对R<= －A 综合的例子

运算符

4、二元算术运算符

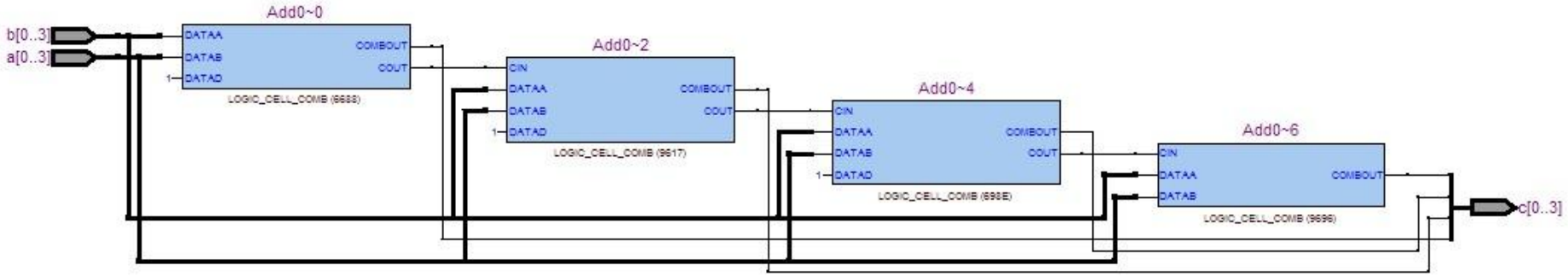
现在的综合工具,特别是高层次综合工具,特别是高层次综合工具,都能直接把加,减,乘运算综合为相应的电路,部分工具也支持除法运算。mod 和rem 运算符通常不被综合工具支持。

如果使用IEEE颁布的标准算术运算包std_logic_arith，那么还可以直接描述对bit或std_logic类型的标量和矢量对象进行算术运算的电路，并综合。

在综合过程中，综合器先把运算符映射为相应的加法器等综合库提供的专用运算部件，然后进行优化，如果运算可以用简单线路实现，综合器则用简单线路取代专用运算部件。

运算符

```
Entity  adder is
    port ( a , b : in integer range 0 to 15 ; c : out integer rang 0 to 15);
End adder;
Architecture alg of adder is
Begin
   c <=  a + b;
End alg;
```



顺序语句

顺序语句只能在进程中出现，而且其出现顺序直接影响到硬件行为。VHDL能够描述非常复杂的数字电路，很大程度上是由于具有丰富的顺序语句。

1、if语句

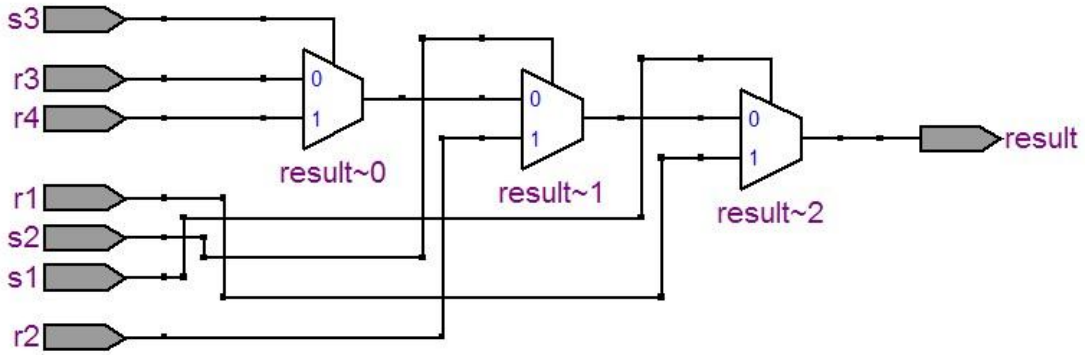
(1)if语句包含了条件所有可能的取值,称之为完全if语句。

这时综合器用多路选择器或基本逻辑门的组合来实现电路。用多路选择器实现电路时，if…elsif…else中隐含的优先关系会被消去，这是设计师应该注意的问题。

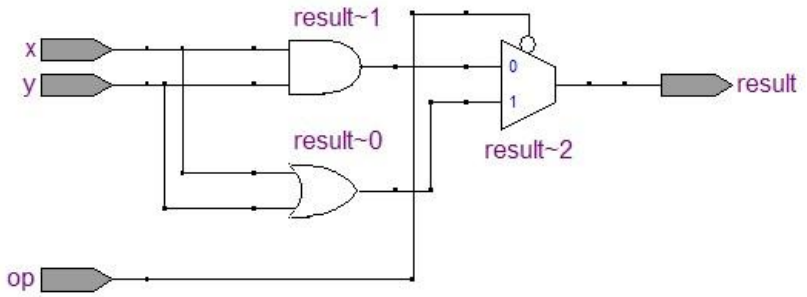
如下例中，他们分别表示P1和P2综合得到的结果。

顺序语句

```
P1: process(s1,s2,s3,r1,r2,r3,r4)
begin
   if s1 = '1' then   result <= r1;
   elsif  s2='1' then  result <= r2;
   elsif  s3/='1' then result <= r3;
   else  result <= r4;
 end if;
end process P1;
```




```
P2 : process(op,x,y)
begin
  if  op='0' then
          result <= x  or  y;
  else
          result <= x  and  y;
   end if;
 end process P2;
```



顺序语句

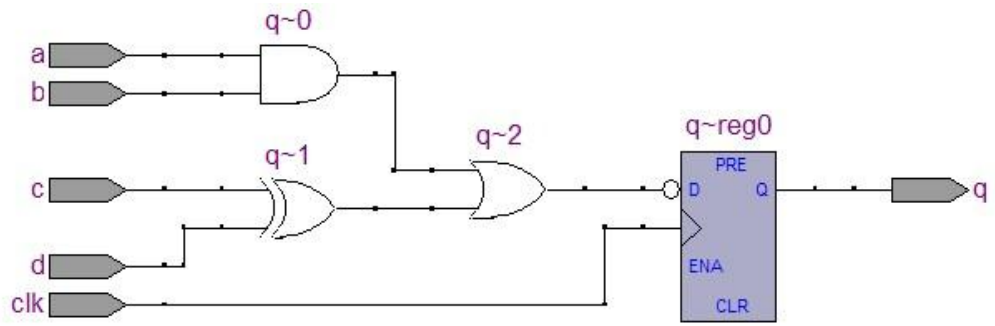
1、if语句

(2)if语句条件未包含所有可能出现的情况,称之为不完全if语句。

此时有效条件是对某信号的跳变进行检测，并且在条件满足时是对信号进行赋值操作，那么会生成触发器。如果赋值号右边为一复杂表达式，则综合器先用组合逻辑电路计算表达式，计算结果送入触发器的数据输入。

如下是对进程FF进行寄存器推断的例子：

```
FF:process (clk,a,b,c,d)
 begin
    if (clk = '1' and clk' event )then
          q <=(a and b) nor (c xor d);
    end if;
 end process FF;
```

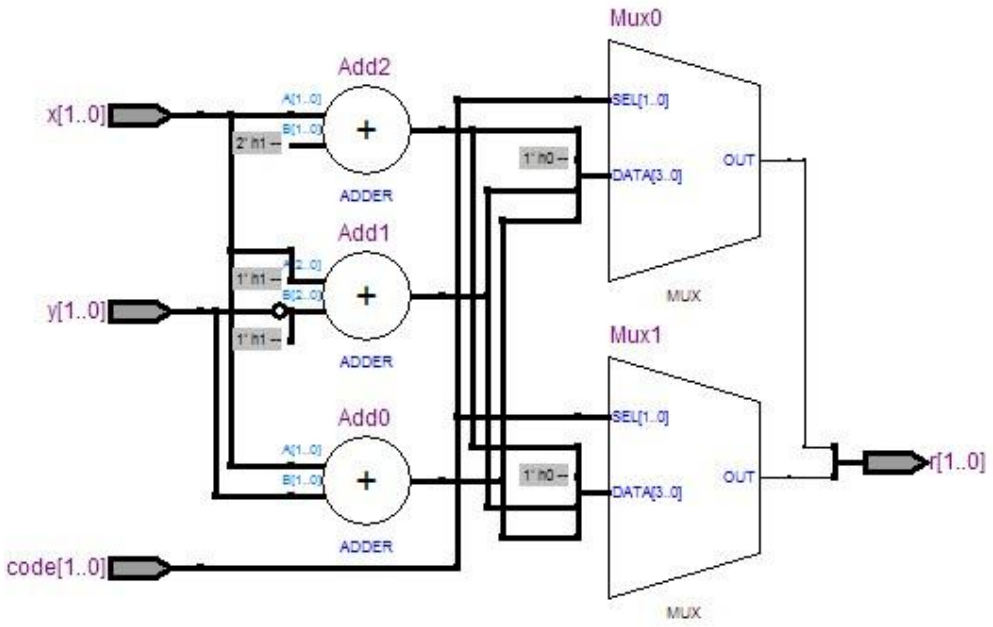


顺序语句

2、case语句

case语句与多路选择器电路的对应关系是显而易见的，但是，建模时要注意合理使用无关态和others语句，否则会造成电路的复杂化，甚至导致形成时序电路。

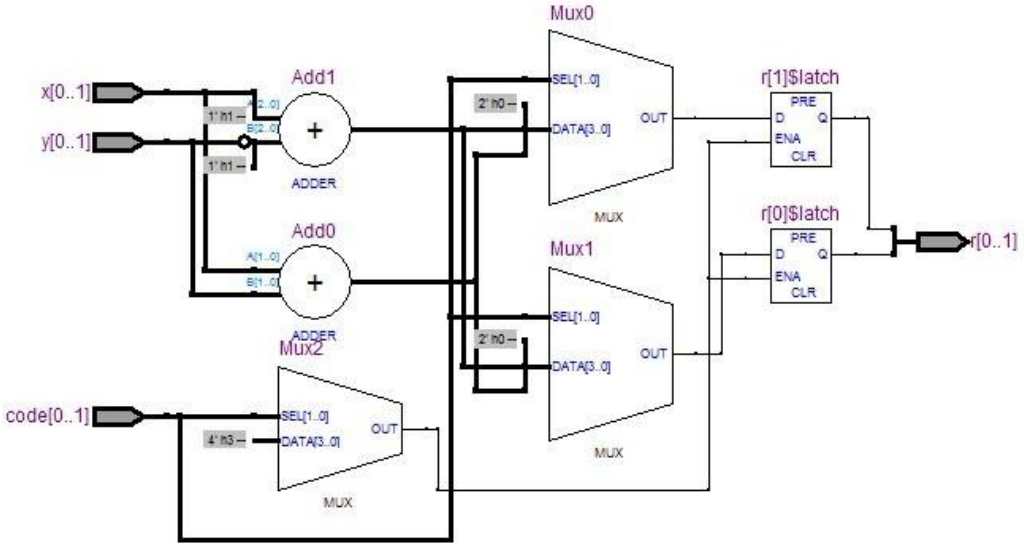
```
type code_type is (add,sub,rst,incx);
subtype  word is interger range 0 to 3;
signal code : code_type;
signal x,y,r : word;
……
p1: process (code,x,y)
begin
  case code  is
        when add => r <=x + y;
        when sub => r <= x – y;
        when rst  => r <= 0;
        when incx =>r <= x+1;
  endcase;
end process p1;
```



顺序语句

状态涵盖不完整时

```
type code_type is (add,sub,rst,incx);
 subtype  word is interger range 0 to 3;
 signal code : code_type;
 signal x,y,r : word;
 ……
 p1: process (code,x,y)
 begin
   case code  is
         when add => r <=x + y;
         when sub => r <= x – y;
         when others  => null;
endcase;
 end process p1;
```



顺序语句

3、循环语句

VHDL的循环语句有三种：for循环 、 while循环,和无限loop…end loop。在行为综合中，循环语句的处理是极其复杂的，这里从寄存器传输级的角度加以讨论。

在寄存器级进行综合，要求for， while循环的上下界必须是静态已知的。如下面两段代码:

顺序语句

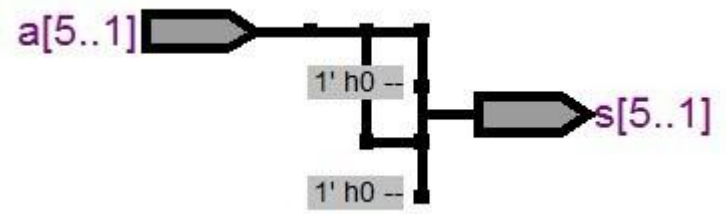
--由于上界不确定而不可综合

```
constant n :natural := 31;
signal rg,sum :natural range 0 to n;
signal clk : clk;
signal  a   : bit_vector(0 to n);
…
p1:process
 variable cpt : natural range 0 to n;
begin
   wait until clk = ‘1’ and clk’event;
   for  j in 1 to rg loop
        if a(j) = ‘0’ then
                cpt := cpt + 1;
        end if;
   end loop;
    sum <= cpt;
end process p1;
```

--可综合的循环语句

```
constant cond : bit_vector(1 to 5) :=“01101”;
 signal s, a :bit_vector (1 to 5);
 …
for i in  cond’range  loop
      next when cond(i) = ‘0’;
      s(i) <= a(i);
end loop;
…
```

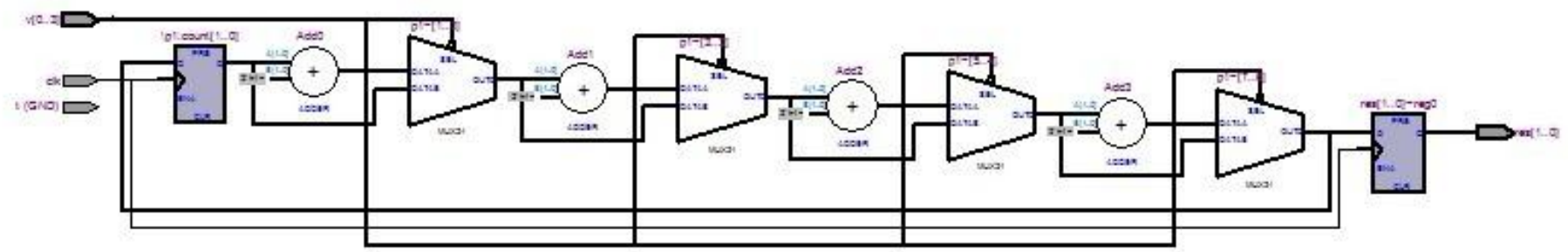
--这段代码通过使用next语句，形成了一
--个选择性的连线网络。



顺序语句

VHDL定义了next和exit语句来中断循环的正常执行，现在的综合工具都可以处理这两种电路结构。如下例中，这个电路信号中的‘1’进行计数，代码中使用了next语句。

```
signal v : bit_vector (0 to 3);
signal sum: natural range 0 to 3;
signal clk : bit;
…
p1: process
  variable count : natural range 0 to 3;
begin
if  clk = ‘1’ and clk’event then
    for  j in 0 to 3 loop
        next when v(j)=‘0’;
        count := count + 1;
    end loop;
end if;
end process p1;
```



并行语句

VHDL的并行语句出现在在结构体内，可综合的并行语言结构包括进程、并行赋值语句、块语句、生成语句等。

1、进程

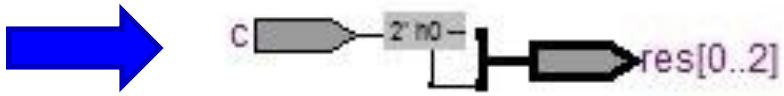
进程是VHDL中描述硬件行为最为有力的方式。进程内的语句属于顺序语句，而进程本身则属于并行语句。进程的综合是比较复杂的，主要有这样一些问题：

综合后的进程是用组合逻辑还是时序逻辑电路实现？
进程中的对象是否有必要用存储器部件？

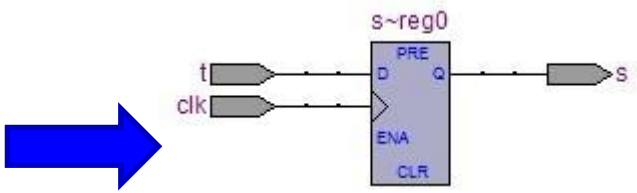
并行语句

1、进程

```
--综合后不需要存储器的VHDL进程
p1: process (c)
variable  a,b : bit ;
begin
      res <= a and b and c;
end process p1;
```




```
--需要存储器的VHDL进程
p2 : process
begin
wait until clk ='1' and clk'event;
      s <= t;
end process ;
```



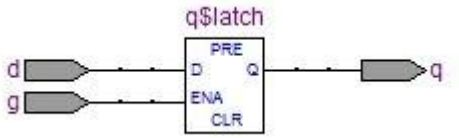
并行语句

1、进程

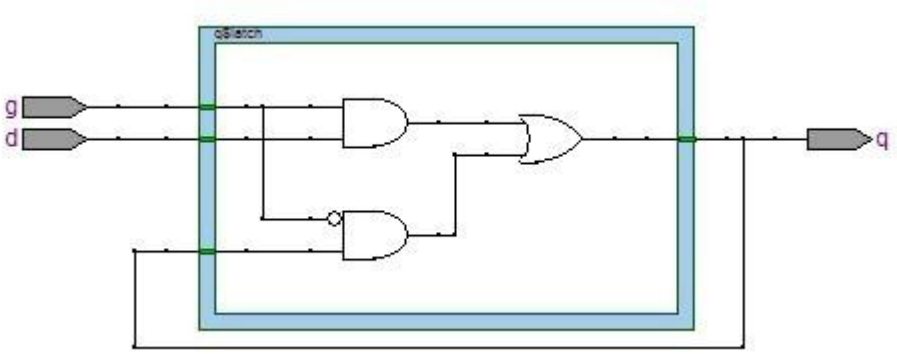
如果进程综合后的电路含有寄存器，那么自然就是时序电路。此外，在以下两种情况下，进程被综合为时序电路：

（1）进程中至少有一个信号不是if 或case中每种可能的分支上的赋值对象，如下面代码中的进程P1的信号。

```
p1: process (d,g)
 begin
   if  g = '1' then
         q <= d;
      end if;
  end process p1;
```



RTL级电路

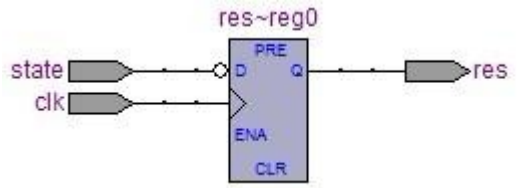


门级电路

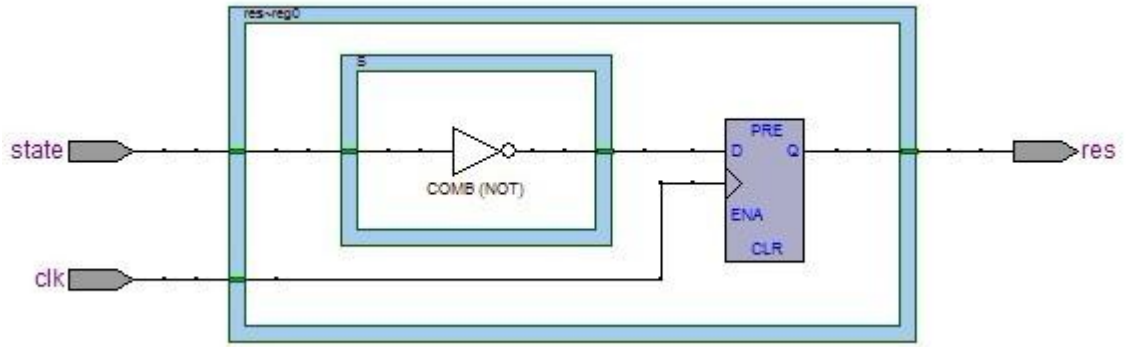
并行语句

（2）进程中所有被读访问的信号不在敏感列表中，如下代码 的进程P2；

```
signal state : t_state;
……
p2 : process
begin
   wait until clk='1' and clk'event;
   case  state  is   --state在被赋值之前先被读访问
      when stop => state <= go;
      when go    => state <= stop;
   end case ;
end process p2;
```



RTL级



门级电路图

并行语句

2、信号赋值语句

信号赋值语句的处理是直截了当的，下面:

语句1被综合成一要硬连线；

对于语句2，R将被当作常数处理；

语句3被综合为组合逻辑电路。

当然语句1和语句3经过逻辑优化后，可能改变形式或者被消去。

```
(1)  S <=  A;
(2)  R<= '1';
(3)  T <= (B xor C) or (D and E) or (F xnor G);
```

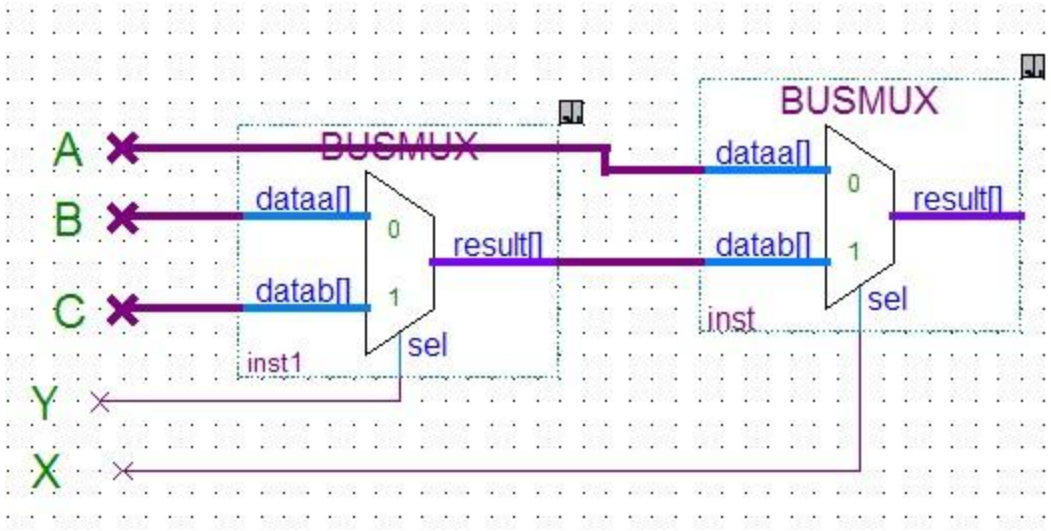
并行语句

3、条件和选择赋值语句

VHDL的并行语句有两种方式进行有条件的赋值，即条件赋值when…else和选择赋值with…select…when。实际上，这两种语法结构都可以改写为等价的顺序语句。

```
s <= a when x = '1';
     b when y = '1';
     else c;
```

```
if  x= '1' then s<=a;
elsif  y='1' then s<=b;
else s<=c;
end if;
```



上述的两种条件赋值语句是完全等份的。在综合后会用多选一网络实现。

并行语句

4、元件例化语句

元件例化语句提供了使用以前建立的模块的手段。在综合过程中，提供综合命令的控制。例化语句调用的元件可以用如下一些方式处理：

（1）展平，即取消层次。把元件本身的描述代入上一级描述，然后整体进行综合和优化。

（2）只把被例化元件综合一次，然后遇到例化这一元件的语句均使用这一综合结果，也就是说，在综合结果中加入一个相同的模块。

（3）只把被例化元件综合一次，但是在每次处理例化元件语句时，根据上下文的代码对元件的接口逻辑重新综合。

元件例化语句常与配置语句联合使用，可以通过配置语句引导综合工具选择适当的设计版本。

并行语句

5、块语句

块语句有把一些相关并行语句组织到一起和进行保护赋值两种作用。

第一种块语句在综合处理上与一般的并行语句没有区别，只是有些综合工具可以把一个块结构当作模块来处理，也就是提供了一种层次化的手段。

第二种块语句一般用来描述寄存器和三态器件。

下面两段代码描述了一个下降沿触发的触发器和三态总线连接。

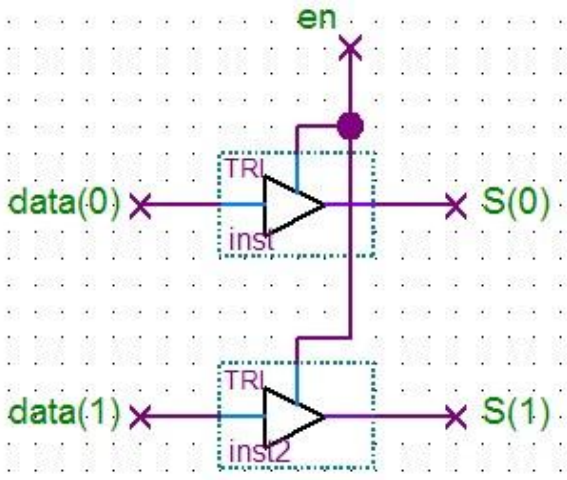
并行语句

--块语句描述的触发器

```
block1 : block(clk='0' and clk'event)
    signal r :bit;
begin
    r <= guarded data;
    s <= guarded r;
end block block1;
```

--三态总线连接的描述

```
signal en : std_ulogic;
signal data : std_ulogic_vector(1 downto 0);
signal s : std_logic_vector(1 downto 0)bus;
…
tri_state : block(en = '1')
 begin
     s <= guarded std_logic_vector(data);
end block tri_state;
```



三态总线描述的综合结果，由于data 是std_ulogic_vector类型，所以在赋值时进行了类型转换。