数字系统设计

郑海永

上课时间地点：周 2/9 10 节/3308

电子信息科学与技术 2010 级和 2011 级

中国海洋大学 电子工程系

2013 年 5 月

VHDL

目录

内容提要

内容提要

顺序语句

顺序语句用于在进程语句中描述进程或子程序的行为

1. WAIT 语句
2. 变量赋值语句
3. 信号赋值语句
4. IF 语句
5. CASE 语句
6. NULL 语句
7. LOOP 语句
8. NEXT 语句
9. EXIT 语句
10. 过程调用语句
11. RETURN 语句
12. 断言语句与REPORT 语句

④ IF 语句

```
IF 布尔表达式 THEN
    { 顺序语句 }
[ { ELSIF 布尔表达式 THEN
    { 顺序语句 } } ]
[ ELSE
    { 顺序语句 } ]
END IF;
```

```
IF sum <= 100 THEN
  sum := sum +10;
END IF;
```

```
IF nickel_in THEN
  deposited <= total_10;
ELSIF dime_in THEN
  deposited <= total_15;
ELSIF quarter_in THEN
  deposited <= total_30;
ELSE
  deposited <= total_error;
END IF;
```

```
IF ctrl1 = '1' THEN
  IF ctrl2 = '0' THEN
    mux_out <= "0010";
  ELSE
    mux_out <= "0001";
  END IF;
ELSE
  IF ctrl2 = '1' THEN
    mux_out <= "1000";
  ELSE
    mux_out <= "0100";
  END IF;
END IF;
```

```
IF sum <= 100 THEN
  sum := sum +10;
END IF;
```

```
IF nickel_in THEN
  deposited <= total_10;
ELSIF dime_in THEN
  deposited <= total_15;
ELSIF quarter_in THEN
  deposited <= total_30;
ELSE
  deposited <= total_error;
END IF;
```

```
IF ctrl1 = '1' THEN
  IF ctrl2 = '0' THEN
    mux_out <= "0010";
  ELSE
    mux_out <= "0001";
  END IF;
ELSE
  IF ctrl2 = '1' THEN
    mux_out <= "1000";
  ELSE
    mux_out <= "0100";
  END IF;
END IF;
```

```
IF sum <= 100 THEN
  sum := sum +10;
END IF;
```

```
IF nickel_in THEN
  deposited <= total_10;
ELSIF dime_in THEN
  deposited <= total_15;
ELSIF quarter_in THEN
  deposited <= total_30;
ELSE
  deposited <= total_error;
END IF;
```

```
IF ctrl1 = '1' THEN
  IF ctrl2 = '0' THEN
    mux_out <= "0010";
  ELSE
    mux_out <= "0001";
  END IF;
ELSE
  IF ctrl2 = '1' THEN
    mux_out <= "1000";
  ELSE
    mux_out <= "0100";
  END IF;
END IF;
```

8-3 优先级编码器 74LS148

8-3 优先级编码器 74LS148

- *ei_n* 允许输入端（工作状态选择端）。
  1. 当 $ei\_n = 0$ 时，编码器工作；
  2. 反之，编码器不工作（不允许编码）。
- *eo_n* 允许输出端。
  - 当允许编码（$ei\_n = 0$）但无信号输入时（1111_1111），**eo_n = 0**。
- *gs_n* 编码群输出端。
  - 当不允许编码（$ei\_n = 1$）或虽允许编码（$ei\_n = 0$）但无信号输入时，**gs_n = 1**。
  - 换言之，当允许编码（$ei\_n = 0$）且有信号输入时，**gs_n = 0**。

8-3 优先级编码器 74LS148

ei_n	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	a_2	a_1	a_0	gs_n	eo_n
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	x	x	x	x	x	x	x	0	1	1	1	0	1
0	x	x	x	x	x	x	0	1	1	1	0	0	1
0	x	x	x	x	x	0	1	1	1	0	1	0	1
0	x	x	x	x	0	1	1	1	1	0	0	0	1
0	x	x	x	0	1	1	1	1	0	1	1	0	1
0	x	x	0	1	1	1	1	1	0	1	0	0	1
0	x	0	1	1	1	1	1	1	0	0	1	0	1
0	0	1	1	1	1	1	1	1	0	0	0	0	1

当某一输入端有低电平输入，且比它优先级高的输入端没有低电平输入时，输出端才输出相应输入端的代码。

```
ENTITY prioty_encoder IS
  PORT(ei_n: IN Bit;
       d: IN Bit_Vector(7 DOWNTO 0);
       eo_n, gs_n: OUT Bit;
       a: OUT Bit_Vector(2 DOWNTO 0));
END prioty_encoder;
ARCHITECTURE encoder OF prioty_encoder IS
BEGIN
  PROCESS(ei_n,d)
  BEGIN
    IF (ei_n = '1') THEN
      eo_n <= '1';
      gs_n <= '1';
      a <= "111";
    ELSIF (d = B"1111_1111") THEN
      eo_n <= '0';
      gs_n <= '1';
      a <= "111";
    ELSE
      eo_n <= '1';
      gs_n <= '0';
      IF (d(0) = '0') THEN
        a <= "111";
      ELSIF (d(1) = '0') THEN
        a <= "110";
      ELSIF (d(2) = '0') THEN
        a <= "101";
      ELSIF (d(3) = '0') THEN
        a <= "100";
      ELSIF (d(4) = '0') THEN
        a <= "011";
      ELSIF (d(5) = '0') THEN
        a <= "010";
      ELSIF (d(6) = '0') THEN
        a <= "001";
      ELSE
        a <= "000";
      END IF;
    END IF;
  END PROCESS;
END encoder;
```

不完整的条件语句

组合逻辑电路

在IF 语句中，允许缺少ELSE 分支。

- 在组合逻辑电路的设计中，这种描述将在综合之后产生锁存器。
- 尽量不要在组合逻辑电路的设计中落掉IF 语句的ELSE 分支。

时序逻辑电路

- 在时序逻辑电路的设计中，这种描述将在综合之后引进寄存器。
- VHDL 综合其引进的一个时序元件保持当前状态值。
- 虽然锁存器比寄存器结构简单，工作速度快。
- 但锁存器容易产生毛刺，且时序分析困难。
- 因此在CPLD/FPGA 中构造了很多寄存器，却没有构造可供使用的锁存器。
- 如果要在CPLD/FPGA 中产生锁存器，则需要在寄存器之外增加组合逻辑才能实现，会占用更多的芯片资源。
- 然而由于锁存器占用面积小、速度快，在CPU、寄存器堆、存储器、FIFO 和其他存储元件的设计中，常常使用D 锁存器。

不完整的条件语句

在IF 语句中，允许缺少ELSE 分支。

组合逻辑电路

- 在组合逻辑电路的设计中，这种描述将在综合之后产生锁存器。
- 尽量不要在组合逻辑电路的设计中落掉IF 语句的ELSE 分支。

时序逻辑电路

- 在时序逻辑电路的设计中，这种描述将在综合之后引进寄存器。
- VHDL 综合其引进的一个时序元件保持当前状态值。
- 虽然锁存器比寄存器结构简单，工作速度快。
- 但锁存器容易产生毛刺，且时序分析困难。
- 因此在 CPLD/FPGA 中构造了很多寄存器，却没有构造可供使用的锁存器。
- 如果要在 CPLD/FPGA 中产生锁存器，则需要在寄存器之外增加组合逻辑才能实现，会占用更多的芯片资源。
- 然而由于锁存器占用面积小、速度快，在 CPU、寄存器堆、存储器、FIFO 和其他存储元件的设计中，常常使用 D 锁存器。

⑤ CASE 语句

```
CASE 选择表达式 IS
    { WHEN 值域 =>
        { 顺序语句 } }
    [ WHEN OTHERS =>
        { 顺序语句 } ]
END CASE;
```

3-8 译码器 74LS138

3-8 译码器 74LS138

- 74LS138 三个输入使能端 g1、$g2a\_n$、$g2b\_n$。
  1. g1 高电平有效
  2. $g2a\_n$ 低电平有效
  3. $g2b\_n$ 低电平有效
- 只有在所有使能端都为有效电平（$g1\,g2a\_n\,g2b\_n = 100$）时，才进行相应译码，输出信号低电平有效；否则，译码器停止译码，输出无效电平（高电平）。

3-8 译码器 74LS138

$g1$	$g2a\_n$	$g2b\_n$	c	b	a	y_n_7	y_n_6	y_n_5	y_n_4	y_n_3	y_n_2	y_n_1	y_n_0
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
			0	0	1	1	1	1	1	1	1	0	1
			0	1	0	1	1	1	1	1	0	1	1
			0	1	1	1	1	1	1	0	1	1	1
			1	0	0	1	1	1	0	1	1	1	1
			1	0	1	1	1	0	1	1	1	1	1
			1	1	0	1	0	1	1	1	1	1	1
			1	1	1	0	1	1	1	1	1	1	1

```
1  ENTITY decoder_3_8 IS
2    PORT(g1, g2a_n, g2b_n: IN Std_Logic;
3         a, b, c: IN Std_Logic;
4         y_n: OUT Std_Logic_Vector(7 DOWNTO 0));
5  END decoder_3_8;
6  ARCHITECTURE behavl_decoder OF decoder_3_8 IS
7  BEGIN
8    PROCESS(g1, g2a_n, g2b_n, a, b, c)
9      VARIABLE temporary: Std_Logic_Vector(2 DOWNTO 0);
10   BEGIN
11     temporary := g1 & g2a_n & g2b_n;
12     IF temporary = "100" THEN
13       temporary := c & b & a;
14       CASE temporary IS
15         WHEN "000" => y_n <= B"1111_1110";
16         WHEN "001" => y_n <= B"1111_1101";
17         WHEN "010" => y_n <= B"1111_1011";
18         WHEN "011" => y_n <= B"1111_0111";
19         WHEN "100" => y_n <= B"1110_1111";
20         WHEN "101" => y_n <= B"1101_1111";
21         WHEN "110" => y_n <= B"1011_1111";
22         WHEN "111" => y_n <= B"0111_1111";
23         WHEN OTHERS => y_n <= (OTHERS => '1');
24       END CASE;
25     ELSE
26       y_n <= (OTHERS => '1');
27     END IF;
28   END PROCESS;
29 END behavl_decoder;
```

三相三拍顺序脉冲发生器

```
1  ENTITY sequencer_33 IS
2    PORT(clk, ind: IN Std_Logic;
3         a, b, c: OUT Std_Logic);
4  END sequencer_33;
5  ARCHITECTURE behavl_seq OF sequencer_33 IS
6    SIGNAL x: Std_Logic_Vector(2 DOWNTO 0);
7  BEGIN
8    PROCESS(clk,ind)
9    BEGIN
10     IF ind = '1' THEN
11       x <= (OTHERS => '0');
12     ELSIF (clk'Event AND clk='1') THEN
13       CASE x IS
14         WHEN "001" => x <= "010";
15         WHEN "010" => x <= "100";
16         WHEN "100" => x <= "001";
17         WHEN OTHERS => x <= "001";
18       END CASE;
19     END IF;
20   END PROCESS;
21   a <= x(0);
22   b <= x(1);
23   c <= x(2);
24 END behavl_seq;
```

不完整的条件语句

- 在CASE 语句中出现的所有信号赋值语句，都应当在所有WHEN 子句中出现。
- 即每一个WHEN 子句中都应当对这些信号赋值。

这样综合之后才不会产生锁存器。

```
CASE s IS
  WHEN '0' => a <= '1'; b <= '0';
  WHEN OTHERS => a <= '0';
END CASE;
```

```
CASE s IS
  WHEN '0' => a <= '1'; b <= '0';
  WHEN OTHERS => a <= '0'; b <= b;
END CASE;
```

不完整的条件语句

- 在CASE 语句中出现的所有信号赋值语句，都应当在所有WHEN 子句中出现。
- 即每一个WHEN 子句中都应当对这些信号赋值。

这样综合之后才不会产生锁存器。

```
1 CASE s IS
2   WHEN '0' => a <= '1'; b <= '0';
3   WHEN OTHERS => a <= '0';
4 END CASE;
```

```
1 CASE s IS
2   WHEN '0' => a <= '1'; b <= '0';
3   WHEN OTHERS => a <= '0'; b <= b;
4 END CASE;
```

不完整的条件语句

- 在CASE 语句中出现的所有信号赋值语句，都应当在所有WHEN 子句中出现。
- 即每一个WHEN 子句中都应当对这些信号赋值。

这样综合之后才不会产生锁存器。

```
1 CASE s IS
2   WHEN '0' => a <= '1'; b <= '0';
3   WHEN OTHERS => a <= '0';
4 END CASE;
```

```
1 CASE s IS
2   WHEN '0' => a <= '1'; b <= '0';
3   WHEN OTHERS => a <= '0'; b <= b;
4 END CASE;
```

IF 语句与 CASE 语句

4 选 1 多路选择器

IF

```
1  ARCHITECTURE behavl_if OF mux_4_1 IS
2  BEGIN
3    p_if: PROCESS(sel,a,b,c,d)
4    BEGIN
5      IF sel = "00" THEN
6        y <= a;
7      ELSIF sel = "01" THEN
8        y <= b;
9      ELSIF sel = "10" THEN
10       y <= c;
11     ELSE
12       y <= d;
13     END IF;
14   END PROCESS p_if;
15 END behavl_if;
```

CASE

```
1  ARCHITECTURE behavl_case OF mux_4_1 IS
2  BEGIN
3    p_case: PROCESS(sel,a,b,c,d)
4    BEGIN
5      CASE sel IS
6        WHEN "00" => y <= a;
7        WHEN "01" => y <= b;
8        WHEN "10" => y <= c;
9        WHEN "11" => y <= d;
10     END CASE;
11   END PROCESS p_case;
12 END behavl_case;
```

IF 语句与 CASE 语句

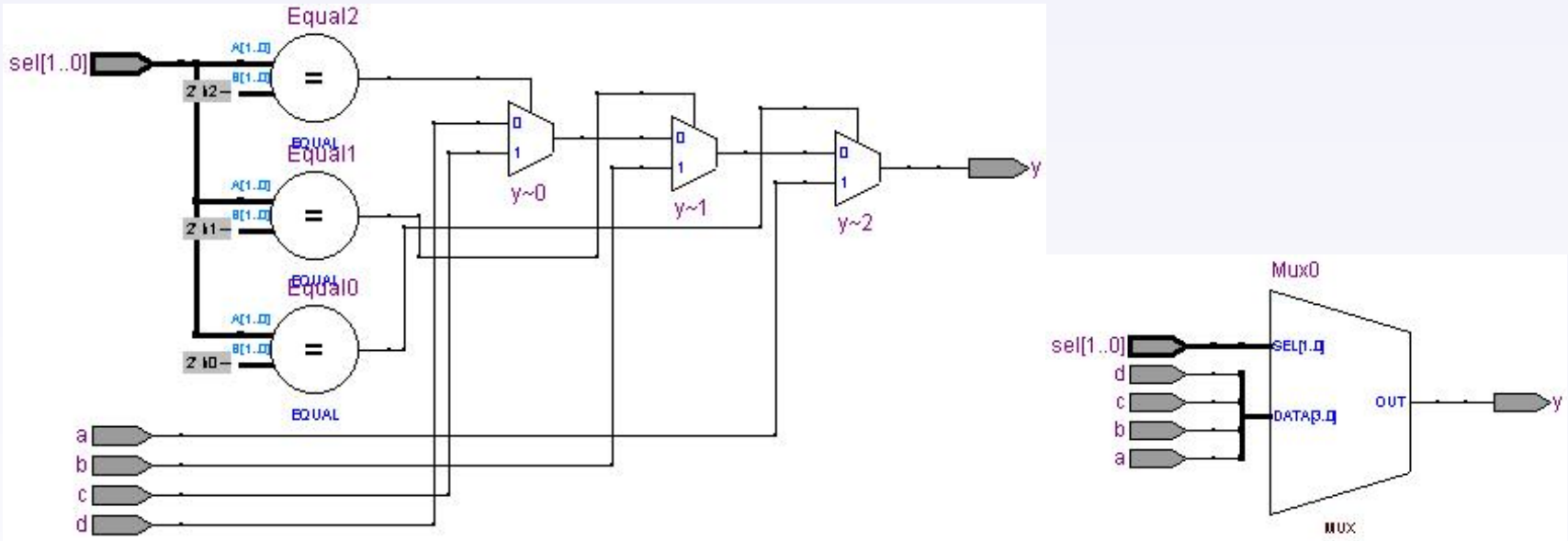
4 选 1 多路选择器

IF

```
1  ARCHITECTURE behavl_if OF mux_4_1 IS
2  BEGIN
3    p_if: PROCESS(sel,a,b,c,d)
4    BEGIN
5      IF sel = "00" THEN
6        y <= a;
7      ELSIF sel = "01" THEN
8        y <= b;
9      ELSIF sel = "10" THEN
10       y <= c;
11     ELSE
12       y <= d;
13     END IF;
14   END PROCESS p_if;
15 END behavl_if;
```

CASE

```
1  ARCHITECTURE behavl_case OF mux_4_1 IS
2  BEGIN
3    p_case: PROCESS(sel,a,b,c,d)
4    BEGIN
5      CASE sel IS
6        WHEN "00" => y <= a;
7        WHEN "01" => y <= b;
8        WHEN "10" => y <= c;
9        WHEN "11" => y <= d;
10     END CASE;
11   END PROCESS p_case;
12 END behavl_case;
```



- IF 语句对应一个优先级编码的多级选择组合电路。
- CASE 语句对应一个单级的多路选择器。
- 单级多路选择器比优先级编码的多路选择组合电路速度更快。
- 在不需要使用优先级编码结构时尽量采用CASE 语句，不仅可以加快仿真速度，而且能够综合出工作速度更快的电路。

⑥ NULL 语句

NULL [AFTER 时间表达式];

- 顺序语句且不会引起任何操作，下一条语句可以继续执行。
- 在IF 或CASE 语句中，在某个特定条件下，显式的说明没有任何操作和运算发生。
- 具有可选项AFTER 时间表达式的NULL 语句称为空事项处理语句，该语句可用于关闭被保护的决断信号的驱动源。

```
1 CASE en IS
2   WHEN '0' => out1 <= d;
3   WHEN OTHERS => NULL;
4 END CASE;
```

⑥ NULL 语句

NULL [AFTER 时间表达式];

- 顺序语句且不会引起任何操作，下一条语句可以继续执行。
- 在IF 或CASE 语句中，在某个特定条件下，显式的说明没有任何操作和运算发生。
- 具有可选项AFTER 时间表达式的NULL 语句称为空事项处理语句，该语句可用于关闭被保护的决断信号的驱动源。

```
1 CASE en IS
2   WHEN '0' => out1 <= d;
3   WHEN OTHERS => NULL;
4 END CASE;
```

⑥ NULL 语句

NULL [AFTER 时间表达式];

- 顺序语句且不会引起任何操作，下一条语句可以继续执行。
- 在IF 或CASE 语句中，在某个特定条件下，显式的说明没有任何操作和运算发生。
- 具有可选项AFTER 时间表达式的NULL 语句称为空事项处理语句，该语句可用于关闭被保护的决断信号的驱动源。

```
1 CASE en IS
2   WHEN '0' => out1 <= d;
3   WHEN OTHERS => NULL;
4 END CASE;
```

⑦ LOOP 语句

```
[ 语句标号: ][ 重复模式 ] LOOP
    { 顺序语句 }
END LOOP [ 语句标号 ];
```

重复模式

1. FOR 重复模式：FOR 循环变量 IN 离散范围
2. WHILE 重复模式：WHILE 布尔表达式
3. 重复模式缺少（无限循环）：循环体所有语句都重复执行直到其他的操作如EXIT、NEXT 或RETURN 语句使得循环退出。

⑦ LOOP 语句

```
[ 语句标号: ][ 重复模式 ] LOOP
    { 顺序语句 }
END LOOP [ 语句标号 ];
```

重复模式

1. FOR 重复模式：FOR 循环变量 IN 离散范围
2. WHILE 重复模式：WHILE 布尔表达式
3. 重复模式缺少（无限循环）：循环体所有语句都重复执行直到其他的操作如EXIT、NEXT 或RETURN 语句使得循环退出。

奇校验电路

```
1 ENTITY odd_check IS
2   PORT(z: IN Bit_Vector(7 DOWNTO 0);
3        odd: OUT Bit);
4 END odd_check;
```

```
1  ARCHITECTURE arch_for OF odd_check IS
2  BEGIN
3    p1: PROCESS(z)
4      VARIABLE tmp: Bit;
5    BEGIN
6      tmp := z(0);
7      l1: FOR i IN 1 TO 7 LOOP
8        tmp := tmp XOR z(i);
9      END LOOP l1;
10     odd <= tmp;
11   END PROCESS p1;
12 END arch_for;
```

```
1  ARCHITECTURE arch_while OF odd_check IS
2  BEGIN
3    p2: PROCESS(z)
4      VARIABLE tmp: Bit;
5      VARIABLE i: Integer;
6    BEGIN
7      i := 0;
8      tmp := z(i);
9      l2: WHILE i < 7 LOOP
10       i := i+1;
11       tmp := tmp XOR z(i);
12     END LOOP l2;
13     odd <= tmp;
14   END PROCESS p2;
15 END arch_while;
```

奇校验电路

```
1  ENTITY odd_check IS
2    PORT(z: IN Bit_Vector(7 DOWNTO 0);
3         odd: OUT Bit);
4  END odd_check;
```

```
1  ARCHITECTURE arch_for OF odd_check IS
2  BEGIN
3    p1: PROCESS(z)
4      VARIABLE tmp: Bit;
5    BEGIN
6      tmp := z(0);
7      l1: FOR i IN 1 TO 7 LOOP
8        tmp := tmp XOR z(i);
9      END LOOP l1;
10     odd <= tmp;
11   END PROCESS p1;
12 END arch_for;
```

```
1  ARCHITECTURE arch_while OF odd_check IS
2  BEGIN
3    p2: PROCESS(z)
4      VARIABLE tmp: Bit;
5      VARIABLE i: Integer;
6    BEGIN
7      i := 0;
8      tmp := z(i);
9      l2: WHILE i < 7 LOOP
10       i := i+1;
11       tmp := tmp XOR z(i);
12     END LOOP l2;
13     odd <= tmp;
14   END PROCESS p2;
15 END arch_while;
```

奇校验电路

```
1  ENTITY odd_check IS
2    PORT(z: IN Bit_Vector(7 DOWNTO 0);
3         odd: OUT Bit);
4  END odd_check;
```

```
1  ARCHITECTURE arch_for OF odd_check IS
2  BEGIN
3    p1: PROCESS(z)
4      VARIABLE tmp: Bit;
5    BEGIN
6      tmp := z(0);
7      l1: FOR i IN 1 TO 7 LOOP
8        tmp := tmp XOR z(i);
9      END LOOP l1;
10     odd <= tmp;
11   END PROCESS p1;
12 END arch_for;
```

```
1  ARCHITECTURE arch_while OF odd_check IS
2  BEGIN
3    p2: PROCESS(z)
4      VARIABLE tmp: Bit;
5      VARIABLE i: Integer;
6    BEGIN
7      i := 0;
8      tmp := z(i);
9      l2: WHILE i < 7 LOOP
10       i := i+1;
11       tmp := tmp XOR z(i);
12     END LOOP l2;
13     odd <= tmp;
14   END PROCESS p2;
15 END arch_while;
```

对比分析

- FOR 循环变量 IN 离散范围
  1. 循环变量已经隐含声明，不必再用变量声明语句来声明。
  2. 循环变量可以出现在表达式中，但不能对循环变量进行赋值操作。
- WHILE 布尔表达式
  1. 布尔表达式中的控制变量必须事先声明并且赋予初值。
  2. 布尔表达式中的控制变量的值必须在循环体中进行更新，否则一旦进入循环就无法结束。
- 由于 VHDL 综合器不支持无法确定次数的LOOP 语句，所以
  1. FOR 循环语句可以被综合。
  2. WHILE 或重复模式缺少（无限循环）的循环语句不能被综合，只能进行仿真。
- 为了提高仿真速度，尽可能在 VHDL 代码中使用数组运算来替代FOR 循环。

```
TYPE number IS RANGE n DOWNTO 0;
TYPE num_vector IS ARRAY (x DOWNTO y) OF number;
```

```
FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
  VARIABLE tmp: num_vector;
BEGIN
  FOR i IN tmp'Range LOOP
    tmp(i) := a(i) + b;
  END LOOP;
  RETURN tmp;
END;
```

```
FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
  VARIABLE tmp: num_vector;
BEGIN
  tmp := (OTHERS => b);
  RETURN a+tmp;
END;
```

采用数组整体相加的描述要比每个元素分别相加的描述仿真速度快

```
1 TYPE number IS RANGE n DOWNTO 0;
2 TYPE num_vector IS ARRAY (x DOWNTO y) OF number;
```

```
1 FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
2   VARIABLE tmp: num_vector;
3 BEGIN
4   FOR i IN tmp'Range LOOP
5     tmp(i) := a(i) + b;
6   END LOOP;
7   RETURN tmp;
8 END;
```

```
1 FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
2   VARIABLE tmp: num_vector;
3 BEGIN
4   tmp := (OTHERS => b);
5   RETURN a+tmp;
6 END;
```

采用数组整体相加的描述要比每个元素分别相加的描述仿真速度快

```
1 TYPE number IS RANGE n DOWNTO 0;
2 TYPE num_vector IS ARRAY (x DOWNTO y) OF number;
```

```
1 FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
2   VARIABLE tmp: num_vector;
3 BEGIN
4   FOR i IN tmp'Range LOOP
5     tmp(i) := a(i) + b;
6   END LOOP;
7   RETURN tmp;
8 END;
```

```
1 FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
2   VARIABLE tmp: num_vector;
3 BEGIN
4   tmp := (OTHERS => b);
5   RETURN a+tmp;
6 END;
```

采用数组整体相加的描述要比每个元素分别相加的描述仿真速度快

```
TYPE number IS RANGE n DOWNTO 0;
TYPE num_vector IS ARRAY (x DOWNTO y) OF number;
```

```
FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
  VARIABLE tmp: num_vector;
BEGIN
  FOR i IN tmp'Range LOOP
    tmp(i) := a(i) + b;
  END LOOP;
  RETURN tmp;
END;
```

```
FUNCTION my_add (a:num_vector; b:number) RETURN num_vector IS
  VARIABLE tmp: num_vector;
BEGIN
  tmp := (OTHERS => b);
  RETURN a+tmp;
END;
```

采用数组整体相加的描述要比每个元素分别相加的描述仿真速度快

无限循环

```
1 sum := 1; j := 0;
2 l: LOOP
3   j := j+21;
4   sum := sum*10;
5   EXIT WHEN sum > 100;
6 END LOOP l;
```

⑧ NEXT 语句

NEXT 语句只能用于 LOOP 语句的循环体中，其作用是终止当前的一次循环迭代，并且开始下一次循环。

```
NEXT [ LOOP 语句标号 ][ WHEN 布尔表达式 ];
```

1. 当布尔表达式的值为True 时，NEXT 语句终止当前的一次循环迭代，并且跳转到LOOP 语句标号所标识的循环体尾部；如果LOOP 语句标号缺失，则跳转到当前循环体的尾部。
2. 当布尔表达式的值为False 时，NEXT 语句的执行等价于NULL，循环正常继续。
3. 当WHEN 布尔表达式缺失时，则等价于该可选项为WHEN True。

⑧ NEXT 语句

NEXT 语句只能用于 LOOP 语句的循环体中，其作用是终止当前的一次循环迭代，并且开始下一次循环。

NEXT [LOOP 语句标号][WHEN 布尔表达式];

1. 当布尔表达式的值为True 时，NEXT 语句终止当前的一次循环迭代，并且跳转到LOOP 语句标号所标识的循环体尾部；如果LOOP 语句标号缺失，则跳转到当前循环体的尾部。
2. 当布尔表达式的值为False 时，NEXT 语句的执行等价于NULL，循环正常继续。
3. 当WHEN 布尔表达式缺失时，则等价于该可选项为WHEN True。

⑨ EXIT 语句

EXIT 语句也只能用于 LOOP 语句的循环体中，其作用是终止当前的一次循环迭代，并且结束当前的循环。

```
EXIT [ LOOP 语句标号 ][ WHEN 布尔表达式 ];
```

1. 当布尔表达式的值为True 时，EXIT 语句终止当前的一次循环迭代，并且结束LOOP 语句标号所标识的循环；如果LOOP 语句标号缺失，则结束当前循环。
2. 当布尔表达式的值为False 时，EXIT 语句的执行等价于NULL，循环正常继续。
3. 当WHEN 布尔表达式缺失时，则等价于该可选项为WHEN True。

⑨ EXIT 语句

EXIT 语句也只能用于 LOOP 语句的循环体中，其作用是终止当前的一次循环迭代，并且结束当前的循环。

EXIT [LOOP 语句标号][WHEN 布尔表达式];

1. 当布尔表达式的值为True 时，EXIT 语句终止当前的一次循环迭代，并且结束LOOP 语句标号所标识的循环；如果LOOP 语句标号缺失，则结束当前循环。
2. 当布尔表达式的值为False 时，EXIT 语句的执行等价于NULL，循环正常继续。
3. 当WHEN 布尔表达式缺失时，则等价于该可选项为WHEN True。

奇校验电路

```
1  ENTITY odd_check IS
2    PORT(z: IN Bit_Vector(7 DOWNTO 0);
3         odd: OUT Bit);
4  END odd_check;
5  ARCHITECTURE arch_loop OF odd_check IS
6  BEGIN
7    p3: PROCESS(z)
8      VARIABLE: tmp: Bit;
9      VARIABLE: i: Integer;
10   BEGIN
11     i := 0;
12     tmp := z(i);
13     l3: LOOP
14       i := i+1;
15       tmp := tmp XOR z(i);
16       EXIT l3 WHEN i > 6;
17     END LOOP l3;
18     odd <= tmp;
19   END PROCESS p3;
20 END arch_loop;
```

⑩ 过程调用语句

过程调用语句可以启动一个过程体被仿真。

过程名 (实参表);

1. 传递给过程的参数类型必须与所调用的过程在其过程声明语句中声明的形式参数类型相同，而且必须是常量类或信号类。
2. 如果参数类型是变量类，则该变量是共享变量，或者该过程只能在进程或其他过程中被调用。
3. 每调用一次过程，相当于使用了一个电路模块。
4. 当前使用的电路模块不应当与前一次使用电路模块的输出信号相关，所以信号类形式参数的模式不能为BUFFER，而只能是IN、OUT或INOUT。

⑩ 过程调用语句

过程调用语句可以启动一个过程体被仿真。

过程名（实参表）;

1. 传递给过程的参数类型必须与所调用的过程在其过程声明语句中声明的形式参数类型相同，而且必须是常量类或信号类。
2. 如果参数类型是变量类，则该变量是共享变量，或者该过程只能在进程或其他过程中被调用。
3. 每调用一次过程，相当于使用了一个电路模块。
4. 当前使用的电路模块不应当与前一次使用电路模块的输出信号相关，所以信号类形式参数的模式不能为BUFFER，而只能是IN、OUT或INOUT。

⑪ RETURN 语句

RETURN 语句只能用于子程序中，用来结束当前的过程或函数。

RETURN [表达式];

1. 过程体中的RETURN 语句不能有表达式，而函数体中的RETURN 语句则必须有表达式（该表达式的值即为函数的返回值）。
2. 函数的返回语句必须是RETURN 语句，而过程结束时可以用RETURN 语句返回，也可以不用RETURN 语句。

⑪ RETURN 语句

RETURN 语句只能用于子程序中，用来结束当前的过程或函数。

RETURN [表达式];

1. 过程体中的RETURN 语句不能有表达式，而函数体中的RETURN 语句则必须有表达式（该表达式的值即为函数的返回值）。
2. 函数的返回语句必须是RETURN 语句，而过程结束时可以用RETURN 语句返回，也可以不用RETURN 语句。

⑫ 断言语句与 REPORT 语句

在进程和子程序中的断言语句被称为顺序断言语句。

ASSERT 布尔表达式 [REPORT 信息][SEVERITY 错误等级];

1. 当布尔表达式为True 时，ASSERT 语句不执行任何动作。
2. 只有当布尔表达式为False 时，才报告信息和错误等级，且所报告的信息只能是字符串类型的一段文字。
3. REPORT 子句缺失时，将报告缺省信息Assertion Violation。
4. 错误等级只能是标准程序包STANDARD 中定义的Severity_Level 类型，其取值范围为Note、Warning、Error 和Failure。
5. SEVERITY 子句缺失时，错误等级的缺省值为Error。

⑫ 断言语句与 REPORT 语句

在进程和子程序中的断言语句被称为顺序断言语句。

ASSERT 布尔表达式 [REPORT 信息][SEVERITY 错误等级];

1. 当布尔表达式为True 时，ASSERT 语句不执行任何动作。
2. 只有当布尔表达式为False 时，才报告信息和错误等级，且所报告的信息只能是字符串类型的一段文字。
3. REPORT 子句缺失时，将报告缺省信息Assertion Violation。
4. 错误等级只能是标准程序包STANDARD 中定义的Severity_Level 类型，其取值范围为Note、Warning、Error 和Failure。
5. SEVERITY 子句缺失时，错误等级的缺省值为Error。

R-S 触发器

```
1  ENTITY rsff IS
2    PORT(set_n, reset_n: IN Bit;
3         q, q_n: BUFFER Bit);
4  END rsff
5  ARCHITECTURE rsff_arch OF rsff IS
6  BEGIN
7    PORCESS(set_n, reset_n)
8      VARIABLE temporary: Std_Logic_Vector (1 DOWNTO 0);
9    BEGIN
10     temporary := set_n & reset_n;
11    ASSERT (temporary /= ”00”)
12       REPORT "Both␣set␣and␣reset␣equal␣to␣'0'."
13       SEVERITY Warning;
14     IF temporary = "10" THEN
15       q <= '0';
16       q_n <= '1';
17     ELSIF temporary = "01" THEN
18       q <= '1';
19       q_n <= '0';
20     END IF;
21   END PROCESS;
22 END rsff_arch;
```

REPORT 语句

VHDL'93 提供一种简短格式的顺序断言语句。

REPORT 信息 [SEVERITY 错误等级];

1. REPORT 语句等价于顺序语句中布尔表达式的值为False 的情况。
   ASSERT False REPORT 信息 [SEVERITY 错误等级];
2. 不同点是当SEVERITY 子句缺失时，错误等级的缺省值为Note。

```
1 REPORT "Hello!";
2 ASSERT False REPORT "Hello!" SEVERITY Note;
```

REPORT 语句

VHDL'93 提供一种简短格式的顺序断言语句。

REPORT 信息 [SEVERITY 错误等级];

1. REPORT 语句等价于顺序语句中布尔表达式的值为False 的情况。
   ASSERT False REPORT 信息 [SEVERITY 错误等级];
2. 不同点是当SEVERITY 子句缺失时，错误等级的缺省值为Note。

```
1 REPORT "Hello!";
2 ASSERT False REPORT "Hello!" SEVERITY Note;
```