EDA 技术实用教程

第 9 章
VHDL结构与要素

9.1 实体

9.1.1 实体语句结构

实体说明单元的一般语句结构：

```
ENTITY  实体名  IS
 [GENERIC ( 参数名：数据类型 );]
 [PORT ( 端口表 );]
END ENTITY 实体名;
```


9.1 实体

9.1.2 参数传递说明语句

参数传递说明语句的一般书写格式如下：

```
GENERIC([ 常数名 : 数据类型 [ : 设定值 ]
{;常数名:数据类型[:设定值]});
```

9.1 实体

9.1.2 参数传递说明语句

【例9-1】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY andn IS
  GENERIC (  n : INTEGER );                  --定义类属参量及其数据类型
   PORT(a : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);--用类属参量限制矢量长度
         c : OUT STD_LOGIC);
 END;
 ARCHITECTURE behav OF andn IS
    BEGIN
      PROCESS (a)
          VARIABLE int : STD_LOGIC;
      BEGIN
         int := '1';
            FOR i IN a'LENGTH - 1 DOWNTO 0 LOOP  --循环语句
              IF a(i)='0' THEN   int := '0';
              END IF;
            END LOOP;
              c <=int ;
      END PROCESS;
END;
```


9.1 实体

9.1.2 参数传递说明语句

【例9-2】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY exn IS
   PORT(d1,d2,d3,d4,d5,d6,d7 : IN STD_LOGIC;
                               q1,q2  : OUT STD_LOGIC);
END;
 ARCHITECTURE exn_behav OF exn IS
     COMPONENT andn               --调用例10-1的元件调用声明
          GENERIC (  n : INTEGER);
         PORT(a : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
               C : OUT STD_LOGIC);
      END COMPONENT ;
 BEGIN
   u1: andn GENERIC MAP (n =>2) -- 参数传递映射语句，定义类属变量，n赋值为2
              PORT MAP (a(0)=>d1,a(1)=>d2,c=>q1);
   u2: andn GENERIC MAP (n =>5)     -- 定义类属变量，n赋值为5
              PORT MAP (a(0)=>d3,a(1)=>d4,a(2)=>d5,
                        a(3)=>d6,a(4)=>d7, c=>q2);
END;
```

9.1 实体

9.1.3 参数传递映射语句

【例9-3】

```
LIBRARY IEEE;                        --待例化元件
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_arith.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
ENTITY addern IS
    PORT (a, b: IN STD_LOGIC_VECTOR;
        result: out STD_LOGIC_VECTOR);
END addern;
 ARCHITECTURE behave OF addern IS
  BEGIN
     result <= a + b;
 END;
```

9.1.3 参数传递映射语句

【例9-4】

```
LIBRARY IEEE;                          --顶层设计
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_arith.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
ENTITY adders IS
    GENERIC(msb_operand: INTEGER := 15; msb_sum: INTEGER :=15);
    PORT(b: IN STD_LOGIC_VECTOR (msb_operand DOWNTO 0);
         result: OUT STD_LOGIC_VECTOR (msb_sum DOWNTO 0));
END adders;
 ARCHITECTURE behave OF adders IS
    COMPONENT addern
     PORT (   a, b: IN STD_LOGIC_VECTOR;
            result: OUT STD_LOGIC_VECTOR);
    END COMPONENT;
    SIGNAL a: STD_LOGIC_VECTOR (msb_sum /2 DOWNTO 0);
    SIGNAL twoa: STD_LOGIC_VECTOR (msb_operand DOWNTO 0);
   BEGIN
    twoa <= a & a;
   U1: addern PORT MAP (a => twoa, b => b, result => result);
   U2: addern  PORT MAP (a=>b(msb_operand downto msb_operand/2 +1),
           b=>b(msb_operand/2 downto 0), result => a);
END behave;
```

9.1 实体

9.1.3 参数传递映射语句

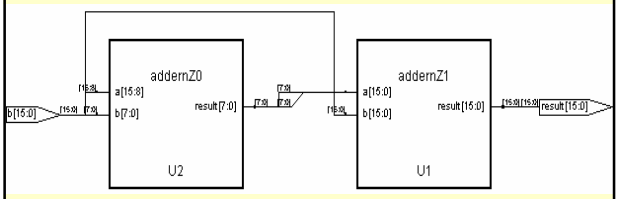


图9-1 例9-4的RTL电路图（Synplify综合）

9.1 实体

9.1.4 端口说明语句

```
PORT ( 端口名 : 端口模式  数据类型 ;
   { 端口名 : 端口模式 数据类型} );
```

9.2 结构体

10 KX 康芯科技

9.2 结构体

1. 结构体的一般语言格式

结构体的语句格式如下：

```
ARCHITECTURE 结构体名 OF 实体名 IS
    [说明语句]
BEGIN
    [功能描述语句]
END ARCHITECTURE 结构体名;
```

11 KX 康芯科技

9.2 结构体

2. 结构体说明语句

3. 功能描述语句结构

- 进程语句
- 信号赋值语句
- 子程序调用语句
- 元件例化语句

12 KX 康芯科技

9.3 子程序

9.3.1 函数

函数的语句表达格式如下：

```
FUNCTION 函数名(参数表) RETURN  数据类型        --函数首
FUNCTION  函数名(参数表)RETURN  数据类型 IS     -- 函数体
    [ 说明部分 ]
    BEGIN
    顺序语句 ;
  END FUNCTION 函数名;
```


【例9-5】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE  packexp IS                                  --定义程序包
   FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)  --定义函数首
     RETURN STD_LOGIC_VECTOR ;
   FUNCTION  func1 ( a,b,c : REAL )             --定义函数首
     RETURN REAL ;
   FUNCTION  "*"  ( a ,b : INTEGER )            --定义函数首
     RETURN INTEGER  ;
   FUNCTION  as2  (SIGNAL  in1 ,in2 : REAL )   --定义函数首
     RETURN REAL  ;
 END ;
PACKAGE  BODY packexp IS
   FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)   --定义函数体
     RETURN STD_LOGIC_VECTOR IS
   BEGIN
     IF a > b THEN RETURN  a;
       ELSE          RETURN  b;
     END IF;
   END FUNCTION max;         --结束FUNCTION语句
 END;                           --结束PACKAGE  BODY语句
```

（接下页）

```
LIBRARY IEEE; -- 函数应用实例
 USE IEEE.STD_LOGIC_1164.ALL;
 USE WORK.packexp.ALL ;
 ENTITY  axamp IS
   PORT(dat1,dat2 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dat3,dat4 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        out1,out2 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
  END;
  ARCHITECTURE bhv OF axamp IS
    BEGIN
     out1 <=  max(dat1,dat2);  --用在赋值语句中的并行函数调用语句
    PROCESS(dat3,dat4)
    BEGIN
     out2 <= max(dat3,dat4);      --顺序函数调用语句
    END PROCESS;
 END;
```

9.3 子程序

9.3.1 函数

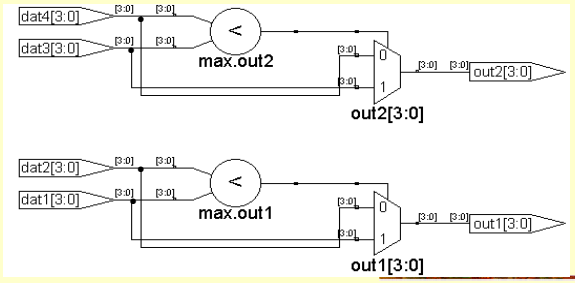


图9-2 例9-5的逻辑电路图

【例9-6】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY func IS
  PORT  ( a :   IN STD_LOGIC_VECTOR (0 to 2 ) ;
          m :  OUT STD_LOGIC_VECTOR (0 to 2 ) );
 END ENTITY func ;
 ARCHITECTURE demo OF func IS
 FUNCTION sam(x ,y ,z : STD_LOGIC) RETURN STD_LOGIC IS
 BEGIN
   RETURN ( x AND y ) OR z  ;
 END FUNCTION sam ;
 BEGIN
   PROCESS ( a )
   BEGIN
    m(0) <= sam( a(0),  a(1),  a(2) ) ;
    m(1) <= sam( a(2),  a(0),  a(1) ) ;
    m(2) <= sam( a(1),  a(2),  a(0) ) ;
   END PROCESS ;
END ARCHITECTURE demo ;
```


9.3 子程序

9.3.2 重载函数

【例9-7】（MaxplusII不支持本例）

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
PACKAGE  packexp IS                                    --定义程序包
 FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)   --定义函数首
   RETURN STD_LOGIC_VECTOR ;
 FUNCTION  max( a,b : IN BIT_VECTOR)          --定义函数首
   RETURN BIT_VECTOR ;
 FUNCTION  max( a,b : IN INTEGER )            --定义函数首
   RETURN INTEGER ;
 END;
PACKAGE BODY packexp IS
 FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)    --定义函数体
   RETURN STD_LOGIC_VECTOR IS
 BEGIN
   IF a > b THEN RETURN  a;                        （接下页）
```

```
ELSE          RETURN  b;     END IF;
    END FUNCTION max;                    --结束FUNCTION语句
FUNCTION  max( a,b : IN INTEGER)            --定义函数体
     RETURN INTEGER IS
   BEGIN
     IF a > b THEN RETURN  a;
       ELSE          RETURN  b;     END IF;
    END FUNCTION max;                          --结束FUNCTION语句
    FUNCTION  max( a,b : IN BIT_VECTOR)          --定义函数体
     RETURN BIT_VECTOR IS
   BEGIN
     IF a > b THEN RETURN  a;
       ELSE          RETURN  b;     END IF;
    END FUNCTION max;                        --结束FUNCTION语句
  END;                                    --结束PACKAGE  BODY语句

  -- 以下是调用重载函数max的程序:
  LIBRARY IEEE ;
  USE IEEE.STD_LOGIC_1164.ALL ;
   USE WORK.packexp.ALL;
  ENTITY  axamp IS
```

（接下页）

```
PORT(a1,b1 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          a2,b2 : IN BIT_VECTOR(4 DOWNTO 0);
          a3,b3 : IN INTEGER RANGE 0 TO 15;
              c1 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
              c2 : OUT BIT_VECTOR(4 DOWNTO 0);
              c3 : OUT INTEGER RANGE 0 TO 15);
   END;
  ARCHITECTURE bhv OF axamp IS
    BEGIN
    c1 <=  max(a1,b1); --对函数max( a,b : IN STD_LOGIC_VECTOR)的
调用
    c2 <=  max(a2,b2); --对函数max( a,b : IN BIT_VECTOR) 的调用
    c3 <=  max(a3,b3); --对函数max( a,b : IN INTEGER)  的调用
 END;
```


【例9-8】

```
  LIBRARY IEEE ;                                -- 程序包首
  USE IEEE.std_logic_1164.all ;
  USE IEEE.std_logic_arith.all ;
  PACKAGE STD_LOGIC_UNSIGNED is
  function "+" (L : STD_LOGIC_VECTOR ; R : INTEGER)
                    return STD_LOGIC_VECTOR ;
  function "+" (L : INTEGER;  R : STD_LOGIC_VECTOR)
                    return STD_LOGIC_VECTOR ;
  function "+" (L : STD_LOGIC_VECTOR ; R : STD_LOGIC )
  return STD_LOGIC_VECTOR ;
  function SHR (ARG : STD_LOGIC_VECTOR ;
    COUNT : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR ;
  ...
  end STD_LOGIC_UNSIGNED  ;

  LIBRARY IEEE ;                              --  程序包体
```

（接下页）

```
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
package body STD_LOGIC_UNSIGNED is
function maximum (L, R :  INTEGER) return  INTEGER  is
begin
    if L > R then  return L;
        else         return R;
    end if;
end;
function "+" (L : STD_LOGIC_VECTOR ;  R : INTEGER)
return  STD_LOGIC_VECTOR  is
Variable result : STD_LOGIC_VECTOR (L'range) ;
Begin
  result :=     UNSIGNED(L) + R ;
  return std_logic_vector(result) ;
end ;
...
end STD_LOGIC_UNSIGNED ;
```


9.3 子程序

9.3.3 转换函数

表9-1 IEEE库类型转换函数表

函数名	功能
程序包: STD_LOGIC_1164	
to_stdlogicvector(A)	由bit_vector类型的A转换为std_logic_vector
to_bitvector(A)	由std_logic _vector转换为bit_vector
to_stdlogic (A)	由bit转换成std_logic
to_bit(A)	由std_logic转换成bit
程序包: STD_LOGIC_ARITH	
conv_std_logic_vector(A, 位长)	将整数integer转换成std_logic_vector类型，A是整数
conv_integer(A)	将std_logic_vector转换成整数integer
程序包: STD_LOGIC_UNSIGNED	
conv_integer(A)	由std_logic_vector转换成integer

9.3 子程序

9.3.3 转换函数

【例9-9】

```
LIBRARY IEEE;
USE IEEE. std_logic_1164.ALL;
ENTITY exg IS
  PORT (a,b : in bit_vector(3 downto 0);
        q   : out std_logic_vector(3 downto 0));
end ;
architecture rtl of exg is
begin
  q<= to_stdlogicvector(a and b);--将位矢量数据类型转换成标准逻辑位矢量数据
end;
```

9.3 子程序

9.3.3 转换函数

【例9-10】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;--注意使用了此程序包
ENTITY  axamp IS
   PORT(a,b,c : IN integer range 0 to 15 ;
        q   : OUT std_logic_vector(3 downto 0) );
  END;
ARCHITECTURE bhv OF axamp IS
    BEGIN
     q  <=   conv_std_logic_vector(a,4)  when  conv_integer(c)=8
else
conv_std_logic_vector(b,4) ;
END;
```


【例9-11】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE n_pack IS
  SUBTYPE  nat  IS  Integer  range  0  to  255;   --定义一个
Integer的子类型
    TYPE Bit8 IS array (7 downto 0) OF std_logic;-- 定义一
个数据类型
  FUNCTION nat_to_Bit8 (s: nat) RETURN Bit8;
End n_pack;
  PACKAGE BODY n_pack IS
  FUNCTION nat_to_Bit8 (s: nat) RETURN Bit8 IS
    VARIABLE Din: Integer range 255 downto 0;
    VARIABLE Rut: Bit8;
    VARIABLE Rig: Integer :=2**7;
 BEGIN
    Din := s;
  FOR I in 7 downto 0 LOOP
    IF Din/Rig > 1 THEN Rut(i) := '1'; Din := Din-Rig;
      ELSE Rut (i):= '0';  END IF;
        Rig := Rig / 2;
```

（接下页）

```
END LOOP;
  RETURN Rut;
 END nat_to_Bit8;
END n_pack;

LIBRARY IEEE; -- 用户定义转换函数应用实例
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.n_pack.ALL ;
ENTITY  axamp IS
   PORT(dat : IN nat;   --注意数据类型的定义
        ou  : OUT Bit8); --注意数据类型的定义
  END;
ARCHITECTURE bhv OF axamp IS
    BEGIN
     ou <=  nat_to_Bit8(dat);
END;
```

9.3 子程序

9.3.4 决断函数

9.3.5 过程

```
PROCEDURE 过程名(参数表)                    -- 过程首

PROCEDURE 过程名(参数表) IS
    [说明部分]
    BIGIN                                   -- 过程体
      顺序语句;
  END PROCEDURE 过程名;
```


9.3 子程序

9.3.5 过程

```
PROCEDURE  pro1  (VARIABLE  a, b :  INOUT REAL) ;
PROCEDURE  pro2 (CONSTANT  a1 :  IN INTEGER ;
                   VARIABLE  b1 :  OUT INTEGER ) ;
PROCEDURE  pro3  (SIGNAL sig :  INOUT BIT) ;
```

【例9-12】

```
PROCEDURE  prg1(VARIABLE value:INOUT BIT_VECTOR(0 TO 7)) IS
BEGIN
 CASE value IS
  WHEN "0000" => value: "0101" ;
  WHEN "0101" => value: "0000" ;
  WHEN OTHERS => value: "1111" ;
    END CASE ;
END PROCEDURE  prg1 ;
```


9.3 子程序

9.3.5 过程

【例9-13】

```
   PROCEDURE  comp ( a, r : IN REAL;
                     m : IN INTEGER ;
                  v1, v2: OUT REAL) IS
   VARIABLE cnt : INTEGER ;
   BEGIN
   v1 := 1.6 * a ;                          -- 赋初始值
   v2 := 1.0 ;                              -- 赋初始值
   Q1 :  FOR  cnt  IN  1 TO m  LOOP
    v2 :=  v2 * v1 ;
EXIT  Q1  WHEN  v2 > v1;       -- 当v2 > v1，跳出循环LOOP
    END LOOP Q1
 ASSERT (v2 < v1 )
     REPORT  "OUT OF RANGE"            -- 输出错误报告
         SEVERITY  ERROR ;
END PROCEDURE  comp ;
```

【例9-14】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE axamp IS                                        过程首定义
 PROCEDURE nand4a (SIGNAL a,b,c,d : IN STD_LOGIC ;
                   SIGNAL  y : OUT  STD_LOGIC );
 END axamp;
PACKAGE BODY axamp IS                                --过程体定义
 PROCEDURE nand4a  (SIGNAL a,b,c,d : IN STD_LOGIC ;
                   SIGNAL  y : OUT  STD_LOGIC ) IS
 BEGIN
   y<= NOT(a AND b AND c AND d);
 RETURN;
END nand4a;
END axamp;
LIBRARY IEEE;                                          --主程序
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.axamp.ALL;
 ENTITY EX IS
  PORT( e,f,g,h : IN STD_LOGIC ;
                x : OUT  STD_LOGIC );
 END;
ARCHITECTURE bhv OF EX IS
    BEGIN
     nand4a(e,f,g,h,x) ;                          并行调用过程
END;
```

9.3 子程序

9.3.6 重载过程

【例9-15】

```
 PROCEDURE  calcu (  v1, v2 : IN REAL ;
                     SIGNAL out1 : INOUT INTEGER) ;
 PROCEDURE  calcu (  v1, v2 : IN INTEGER ;
                     SIGNAL out1 : INOUT REAL) ;
 ...
 calcu (20.15, 1.42, signl) ;   -- 调用第一个重载过程calcu
 calcu (23,  320, sign2 ) ;     -- 调用第二个重载过程 calcu
…
```


9.4 VHDL库

9.4.1 库的种类

1. IEEE库
2. STD库

```
LIBRARY STD ;
USE STD.STANDARD.ALL ;
```

3. WORK库
4. VITAL库

9.4 VHDL库

9.4.2 库的用法

```
USE 库名.程序包名.项目名；
    USE 库名.程序包名.ALL；
```

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.STD_ULOGIC ;
USE IEEE.STD_LOGIC_1164.RISING_EDGE ;
```

```
USE WORK.std_logic_1164.ALL;
```


9.5 程序包

定义程序包的一般语句结构如下：

```
PACKAGE   程序包名   IS                --   程序包首
      程序包首说明部分
END   程序包名;
PACKAGE BODY   程序包名   IS        --   程序包体
程序包体说明部分以及包体内
    END 程序包名;
```


9.5 程序包

【例9-16】

```
PACKAGE pacl IS                                  -- 程序包首开始
  TYPE byte IS RANGE 0 TO 255 ;                  -- 定义数据类型byte
  SUBTYPE nibble IS byte RANGE 0 TO 15 ;         -- 定义子类型nibble
CONSTANT byte_ff  : byte := 255 ;                -- 定义常数byte_ff
SIGNAL addend : nibble ;                         -- 定义信号addend
COMPONENT byte_adder                             -- 定义元件
PORT( a, b : IN byte ;
      c : OUT byte  ;
 overflow : OUT BOOLEAN ) ;
 END COMPONENT ;
 FUNCTION my_function (a : IN byte) Return byte ;   -- 定义函数
END pacl ;                                          -- 程序包首结束
```

【例9-17】

```
PACKAGE  seven IS
    SUBTYPE segments is BIT_VECTOR(0 TO 6) ;
    TYPE bcd IS RANGE 0 TO 9 ;
END seven ;
USE WORK.seven.ALL ; -- WORK库默认是打开的,
ENTITY decoder IS
    PORT (input: bcd; drive : out segments) ;
END decoder ;
ARCHITECTURE simple OF decoder IS
BEGIN
    WITH input SELECT
      drive <=  B"1111110"  WHEN 0 ,
                B"0110000"  WHEN 1 ,
                B"1101101"  WHEN 2 ,
                B"1111001"  WHEN 3 ,
                B"0110011"  WHEN 4 ,
                B"1011011"  WHEN 5 ,
                B"1011111"  WHEN 6 ,
                B"1110000"  WHEN 7 ,
                B"1111111"  WHEN 8 ,
                B"1111011"  WHEN 9 ,
                B"0000000"  WHEN  OTHERS ;
END simple ;
```

9.5 程序包

(1) STD_LOGIC_1164程序包。

(2) STD_LOGIC_ARITH程序包。

(3) STD_LOGIC_UNSIGNED和STD_LOGIC_SIGNED程序包。

(4) STANDARD和TEXTIO程序包。

9.6 配置

配置语句的一般格式如下：

```
CONFIGURATION 配置名 OF 实体名 IS
    配置说明
END 配置名;
```

9.7 VHDL文字规则

9.7.1 数字

整数 — 5， 678， 0， 156E2(=15600)， 45_234_287 (=45234287)

实数 — 1.335， 88_670_551.453_909(=88670551.453909)，1.0，44.99E-2(=0.4499)

以数制基数表示的文字

```
SIGNAL d1,d2,d3,d4,d5, : INTEGER RANGE 0 TO 255;
d1 <= 10#170# ;          -- (十进制表示，等于 170)
d2 <= 16#FE#    ;        -- (十六进制表示，等于 254)
d3 <= 2#1111_1110#;      -- (二进制表示，等于 254)
d4 <= 8#376#       ;     -- (八进制表示，等于 254)
d5 <= 16#E#E1   ;    -- (十六进制表示，等于2#1110000#，等于224)
```

物理量文字(VHDL综合器不接受此类文字) — 60s (60秒)， 100m (100米)， k (千欧姆)， 177A (177安培)

9.7 VHDL文字规则

9.7.2 字符串

(1) 文字字符串 — "ERROR"， "Both S and Q equal to 1"， "X"， "BB$CC"

(2) 数位字符串

```
data1 <= B"1_1101_1110"      -- 二进制数数组，位矢数组长度是9
data2 <= O"15"               -- 八进制数数组，位矢数组长度是6
data3 <= X"AD0"              -- 十六进制数数组，位矢数组长度是12
data4 <= B"101_010_101_010"  -- 二进制数数组，位矢数组长度是12
data5 <= "101_010_101_010"   --表达错误，缺B。
data6 <= "0AD0"              --表达错误，缺X。
```


9.7 VHDL文字规则

9.7.3 标识符

合法的标识符：Decoder_1， FFT， Sig_N， Not_Ack， State0 ， Idle

非法的标识符：

```
_Decoder_1          -- 起始为非英文字母
 2FFT               -- 起始为数字
Sig_#N              -- 符号"#"不能成为标识符的构成
Not-Ack             -- 符号"-" 不能成为标识符的构成
RyY_RST_            -- 标识符的最后不能是下划线"_"
data_ _BUS          -- 标识符中不能有双下划线
return              -- 关键词
```

9.7 VHDL文字规则

9.7.4 下标名

标识符(表达式)

```
SIGNAL  a, b : BIT_VECTOR (0 TO 3) ;
SIGNAL  m    : INTEGER  RANGE 0 TO 3 ;
SIGNAL  y, z : BIT ;
y <= a(m) ;              --  不可计算型下标表示
z <= b(3) ;       -- 可计算型下标表示
```


9.8 数据类型

- 标量型(Scalar Type) — 实数类型、整数类型、枚举类型、时间类型
- 复合类型(Composite Type) — 数组型(Array)、记录型(Record)
- 存取类型(Access Type) — 为给定的数据类型的数据对象提供存取方式
- 文件类型(Files Type) — 用于提供多值存取类型

9.8 数据类型

9.8.1 预定义数据类型

1. 布尔类型

```
TYPE BOOLEAN IS (FALSE, TRUE);
```

2. 位数据类型

```
TYPE BIT IS ('0', '1');
```

3. 位矢量类型

```
TYPE BIT_VECTOR IS ARRAY (Natural Range <> ) OF  BIT ;
```

4. 字符类型

'A'

5. 整数类型

－2147483647～＋2147483647

9.8 数据类型

6. 实数类型

1.0	十进制浮点数
0.0	十进制浮点数
65971.333333	十进制浮点数
65_971.333_3333	与上一行等价
8#43.6#e+4	八进制浮点数
43.6E－4	十进制浮点数

7. 字符串类型

```
VARIABLE string_var : STRING (1 TO 7 ) ;
string_var := "a b c d" ;
```


9.8 数据类型

8. 时间类型

```
TYPE time IS RANGE －2147483647 TO 2147483647
    units
        fs ;  -- 飞秒，VHDL中的最小时间单位
        ps = 1000 fs ;          -- 皮秒
        ns = 1000 ps ;          -- 纳秒
        us = 1000 ns ;          -- 微秒
        ms = 1000 us ;          -- 毫秒
        sec = 1000 ms ;         -- 秒
        min = 60 sec ;          -- 分
        hr = 60 min ;           -- 时
end units ;
```


9.8 数据类型

9. 文件类型

```
PROCEDUER Readline (F: IN TEXT; L: OUT LINE);
PROCEDUER Writeline (F: OUT TEXT; L: IN LINE);
PROCEDUER Read (   L: INOUT LINE; Value: OUT std_logic;
Good: OUT BOOLEAN);
PROCEDUER Read (L: INOUT LINE; Value: OUT std_logic);
PROCEDUER Read (   L: INOUT LINE; Value: OUT std_logic_ vector;
Good: OUT BOOLEAN);
PROCEDUER Read (L: INOUT LINE; Value: OUT std_logic_ vector;
PROCEDUER Write (          L: INOUT LINE; Value: IN std_logic;
Justiaied: IN SIDE :=Right;field; IN WIDTH :=0);
PROCEDUER Write (L: INOUT LINE; Value: IN std_logic _ vector,
  Justiaied: IN SIDE :=Right;field; IN WIDTH :=0);
```

9.8 数据类型

9.8.2 IEEE预定义标准逻辑位与矢量

1. 标准逻辑位数据类型

2. 标准逻辑矢量数据类型

```
TYPE STD_LOGIC_VECTOR IS ARRAY ( NATURAL RANGE <> ) OF STD_LOGIC ;
```


9.8 数据类型

9.8.3 其他预定义标准数据类型

1. 无符号数据类型(UNSIGNED TYPE)　UNSIGNED'("1000")

```
VARIABLE var : UNSIGNED(0 TO 10) ;
SIGNAL   sig : UNSIGNED(5 TO 0) ;
```

2. 有符号数据类型(SIGNED TYPE)

```
SIGNED'("0101") 代表 +5, 5
SIGNED'("1011") 代表 –5
```

```
VARIABLE var ; SIGNED(0 TO 10);
```


9.8 数据类型

9.8.4 数组类型

```
TYPE 数组名IS ARRAY(数组范围)OF数据类型
```

```
TYPE stb IS ARRAY (7 DOWNTO 0) of STD_LOGIC ;
```

```
TYPE x is (low, high) ;
TYPE data_bus IS ARRAY (0 TO 7, x) of BIT ;
```

```
TYPE 数组名IS ARRAY (数组下标名RANGE <>)OF 数据类型 ;
```

9.8 数据类型

9.8.4 数组类型

【例9-18】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY  amp IS
   PORT (     a1, a2 : IN BIT_VECTOR(3 DOWNTO 0);
          c1, c2,c3 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
           b1, b2, b3 : INTEGER RANGE 0 TO 15;
         d1,d2,d3,d4 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)     );
END amp;
 d1 <= TO_STDLOGICVECTOR(a1 AND a2);                --(1)
 d2 < = CONV_STD_LOGIC_VECTOR(b1,4) WHEN CONV_INTEGER(b2)=9
         else   CONV_STD_LOGIC_VECTOR(b3,4);        --(2)
 d3 < = c1 WHEN CONV_INTEGER(c2)= 8 ELSE c3;       --(3)
 d4 < = c1 WHEN  c2 = 8 else c3;             --(4)
```


9.8 数据类型

9.8.4 数组类型

【例9-19】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY decoder3to8 IS
 PORT (  input: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
         output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END decoder3to8;
ARCHITECTURE behave OF decoder3to8 IS
  BEGIN
    PROCESS (input)
      BEGIN
          output <= (OTHERS => '0');
         output(CONV_INTEGER(input)) <= '1';
    END PROCESS;
END behave;
```


9.8 数据类型

【例9-20】

```
  FUNCTION To_bit ( s : std_ulogic; xmap : BIT := '0' ) RETURN BIT ;
  FUNCTION To_bitvector ( s : std_logic_vector ;
                          xmap : BIT := '0' ) RETURN BIT_VECTOR ;
  FUNCTION To_bitvector ( s : std_ulogic_vector ;
                          xmap : BIT := '0' ) RETURN BIT_VECTOR ;
```

下面是转换函数To_bitvector的函数体：

```
FUNCTION To_bitvector ( s : std_logic_vector ;
                        xmap : BIT := '0' )
                        RETURN BIT_VECTOR IS
    ALIAS sv : std_logic_vector(s'LENGTH-1 DOWNTO 0 ) IS s ;
    VARIABLE result : BIT_VECTOR(s'LENGTH-1 DOWNTO 0 );
BEGIN
    FOR i IN result'RANGE LOOP
        CASE sv(i) IS
            WHEN '0'|'L' => result(i) := '0';
            WHEN '1'|'H' => result(i) := '1';
            WHEN OTHERS => result(i) := xmap;
        END CASE ;
    END LOOP ;
    RETURN result ;
END ;
```

9.9 操作符

9.9.1 逻辑操作符

表9-2 VHDL操作符列表

类　型	操作符	功　能	操作数数据类型
算术操作符	+	加	整数
	—	减	整数
	&	并置	一维数组
	*	乘	整数和实数(包括浮点数)
	/	除	整数和实数(包括浮点数)
	MOD	取模	整数
	REM	取余	整数
	SLL	逻辑左移	BIT或布尔型一维数组
	SRL	逻辑右移	BIT或布尔型一维数组
	SLA	算术左移	BIT或布尔型一维数组

（接下页）

（接上页）

算术操作符	SRA	算术右移	BIT或布尔型一维数组
	ROL	逻辑循环左移	BIT或布尔型一维数组
	ROR	逻辑循环右移	BIT或布尔型一维数组
	**	乘方	整数
	ABS	取绝对值	整数
关系操作符	=	等于	任何数据类型
	/=	不等于	任何数据类型
	<	小于	枚举与整数类型，及对应的一维数组
	>	大于	枚举与整数类型，及对应的一维数组
	<=	小于等于	枚举与整数类型，及对应的一维数组
	>=	大于等于	枚举与整数类型，及对应的一维数组
逻辑操作符	AND	与	BIT，BOOLEAN，STD_LOGIC
	OR	或	BIT，BOOLEAN，STD_LOGIC
	NAND	与非	BIT，BOOLEAN，STD_LOGIC
	NOR	或非	BIT，BOOLEAN，STD_LOGIC
	XOR	异或	BIT，BOOLEAN，STD_LOGIC
	XNOR	异或非	BIT，BOOLEAN，STD_LOGIC
	NOT	非	BIT，BOOLEAN，STD_LOGIC
符号操作符	+	正	整数
	—	负	整数

9.9 操作符

9.9.1 逻辑操作符

表9-3 VHDL操作符优先级

运算符	优先级
NOT，ABS，**	最高优先级
* ，/ ，MOD， REM	↑
+(正号)， —(负号)	↑
+ ， — ， &	↑
SLL， SLA， SRL， SRA， ROL， ROR	↑
=， /=， <， <=， >， >=	↑
AND， OR， NAND， NOR， XOR， XNOR	最低优先级

9.9 操作符

9.9.1 逻辑操作符

【例9-21】

```
SIGNAL a , b, c : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL d, e, f, g : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
SIGNAL h, I, j, k : STD_LOGIC ;
SIGNAL l, m, n, o, p : BOOLEAN ;
...
a<=b AND c;  --b、c 相与后向a赋值，a、b、c的数据类型同属4位长的位矢量
d<=e OR f OR g ;              -- 两个操作符OR相同，不需括号
h<=(i NAND j)NAND k ;    -- NAND不属上述三种算符中的一种，必须加括号
l<=(m XOR n)AND(o XOR p); -- 操作符不同，必须加括号
h<=i AND j AND k ;            -- 两个操作符都是AND，不必加括号
h<=i AND j OR k ;             -- 两个操作符不同，未加括号，表达错误
a<=b AND e ;              -- 操作数b 与 e的位矢长度不一致，表达错误
h<=i OR l ;           -- i 的数据类型是位STD_LOGIC，而l的数据类型是
…            -- 布尔量BOOLEAN，因而不能相互作用，表达错误。
```


9.9 操作符

9.9.2 关系操作符

“ = ” (等于)　　“/= ” (不等于)

“>” (大于)　　“< ” (小于)

“>=” (大于等于)　　“<=” (小于等于)

9.9 操作符

9.9.2 关系操作符

【例9-22】

```
ENTITY relational_ops_1 IS
  PORT ( a, b : IN BIT__VECTOR (0 TO 3) ;
           m : OUT BOOLEAN) ;
END relational_ops_1 ;
ARCHITECTURE example OF relational_ops_1 IS
BEGIN
  output <= (a = b) ;
END example ;
```

9.9 操作符

9.9.2 关系操作符

【例9-23】

```
ENTITY relational_ops_2 IS
  PORT (a, b : IN INTEGER RANGE 0 TO 3 ;
            m : OUT BOOLEAN) ;
END relational_ops_2 ;
ARCHITECTURE example OF relational_ops_2 IS
BEGIN
  output <= (a >= b) ;
END example ;
```


9.9 操作符

9.9.3 算术操作符

表9-4 算术操作符分类表

	类 别	算术操作符分类
1	求和操作符(Adding operators)	+(加)，－(减)， &(并置)
2	求积操作符(Multiplying operators)	*， / ， MOD ， REM
3	符号操作符(Sign operators)	+(正)， － (负)
4	混合操作符(Miscellaneous operators)	**， ABS
5	移位操作符(Shift operators)	SLL，SRL，SLA，SRA，ROL，ROR

9.9 操作符

9.9.3 算术操作符

1. 求和操作符

【例9-24】

```
VARIABLE a, b , c ,d , e ,f : INTEGER RANGE 0 TO 255 ;
...
a := b + c ;    d := e – f  ;
```

【例9-25】

```
PROCEDURE adding_e (a: IN INTEGER;  b: INOUT INTEGER ) IS
...
b := a + b ;
```

9.9 操作符

9.9.3 算术操作符

1. 求和操作符

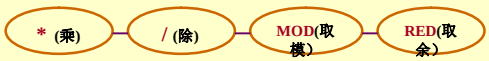
【例9-26】

```
PACKAGE example_arithmetic IS
   TYPE small_INt IS RANGE 0 TO 7 ;
END example_arithmetic ;
USE WORK.example_arithmetic.ALL ;
ENTITY arithmetic IS
   PORT (a, b : IN SMALL_INT ;
             c : OUT SMALL_INT) ;
END arithmetic ;
ARCHITECTURE example OF arithmetic IS
BEGIN
   c <= a + b ;
END example ;
```


9.9 操作符

9.9.3 算术操作符

2. 求积操作符



3. 符号操作符

z := x*(－y) ;

9.9 操作符

9.9.3 算术操作符

4. 混合操作符

【例9-27】

```
SIGNAL a, b : INTEGER RANGE －8 to 7 ;
SIGNAL    c : INTEGER RANGE 0 to 15 ;
SIGNAL    d : INTEGER RANGE 0 to 3 ;
a <= ABS(b) ;
 c <= 2 ** d ;
```

9.9 操作符

5. 移位操作符

【例9-28】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY decoder3to8 IS
    port (   input: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
             output: OUT BIT_VECTOR (7 DOWNTO 0));
END decoder3to8;
ARCHITECTURE behave OF decoder3to8 IS
BEGIN
output <=  "00000001" SLL CONV_INTEGER(input);    --被移位部分是常数!
  END behave;
```


习 题

9-1. 说明实体，设计实体概念。

9-2. 举例说明GENERIC说明语句和GENERIC映射语句有何用处，并举例说明。

9-3. 说明端口模式INOUT和BUFFER有何异同点。

9-4. 什么是重载？重载函数有何用处？

9-5. 在以下数据类型中，VHDL综合器支持哪些类型：

STRING、TIME、REAL、BIT

9-6. 详细说明例10-28中的语句作用和程序实现的功能。

9-7. 表式C <= A + B中，A、B和C的数据类型都是STD_LOGIC_VECTOR，是否能直接进行加法运算？说明原因和解决方法。

9-8. VHDL中有哪3种数据对象？详细说明它们的功能特点以及使用方法，举例说明数据对象与数据类型的关系。

习 题

9-9. 能把任意一种进制的值向一整数类型的数据对象赋值吗？如果能，怎样做？

9-10. 判断下列VHDL标识符是否合法，如果有误则指出原因：

16#0FA#， 10#12F#， 8#789#， 8#356#， 2#0101010#

74HC245 ， \74HC574\， CLR/RESET， \IN 4/SCLK\， D100%

9-11. 数据类型BIT、INTEGER和BOOLEAN分别定义在哪个库中？哪些库和程序包总是可见的？

9-12. 函数与过程的设计与功能上有什么区别？调用上有什么区别？

9-13. 回答有关Bit和Boolean数据类型的问题：

(1) 解释Bit和Boolean类型的区别；

(2) 对于逻辑操作应使用哪种类型？

(3) 关系操作的结果为哪种类型？

(4) IF语句测试的表达式是哪种类型？

习 题

9-14. 运算符重载函数通常要调用转换函数，以便能够利用已有的数据类型。下面给出一个新的数据类型AGE，并且下面的转换函数已经实现：

```
function CONV_INTEGER(ARG: AGE) return INTEGER;
```

仿照本章中的示例，利用此函数编写一个“+”运算符重载函数，支持下面的运算：

```
SIGNAL  a, c : AGE;
...
c <= a + 20;
```

9-15. 用两种方法设计8位比较器，比较器的输入是两个待比较的8位数A=[A7..A0]和B=[B7..B0]，输出是 D、E、F。当A=B时D=1；当A>B时E=1；当A<B时F=1。第一种设计方案是常规的比较器设计方法，即直接利用关系操作符进行编程设计；第二种设计方案是利用减法器来完成，通过减法运算后的符号和结果来判别两个被比较值的大小。对两种设计方案的资源耗用情况进行比较，并给以解释。

实 验 与 设 计

9-1 乐曲硬件演奏电路设计

(1) 实验目的：学习利用实验6-3的数控分频器设计硬件乐曲演奏电路。

(2) 实验原理：主系统由3个模块组成，例9-29是顶层设计文件，其内部有3个功能模块(如图9-3所示)：TONETABA.VHD、NOTETABS.VHD和SPEAKER.VHD。

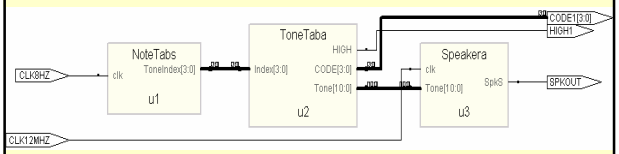


图9-3 硬件乐曲演奏电路结构（Synplify综合）

实 验 与 设 计

9-1 乐曲硬件演奏电路设计

(3) 实验内容1：定制例9-32的NoteTabs模块中的音符数据ROM“music”。该ROM中的音符数据已列在例9-33中。注意该例数据表中的数据位宽、深度和数据的表达类型。此外，为了节省篇幅，例中的数据都横排了，实用中应该以每一分号为一行来展开，否则会出错。

最后对该ROM进行仿真，确认例9-33中的音符数据已经进入ROM中。

(4) 实验内容2：根据给出的乘法器逻辑原理图及其各模块的VHDL描述，在QuartusII上完成全部设计，包括编辑、编译、综合和仿真操作等。给出仿真波形，并作出详细说明。

(5) 实验内容3：硬件验证。先将引脚锁定，使CLK12MHz与clock9相接，接受12MHz时钟频率（用短路帽在clock9接“12MHz”）；CLK8Hz 与clock2相接，接受4Hz频率；发音输出SPKOUT接Speaker；与演奏发音相对应的简谱码输出显示可由CODE1在数码管5显示；HIGH1为高八度音指示，可由发光管D5指示，最后向目标芯片下载适配后的SOF逻辑设计文件。实验电路结构图为NO.1。

实 验 与 设 计

(6) 实验内容4：填入新的乐曲，如“采茶舞曲”、或其它熟悉的乐曲。操作步骤如下：

1、根据所填乐曲可能出现的音符，修改例9-3的音符数据表格，同时注意每一音符的节拍长短；

2、如果乐曲比较长，可增加模块NOTETABA中计数器的位数，如9位时可达512个基本节拍。

(7) 实验内容5：争取可以在一个ROM装上多首歌曲，可手动或自动选择歌曲。

(8) 实验内容6：根据此项实验设计一个电子琴，硬件测试可用电路结构图NO.3。

(9) 思考题1：用LFSR设计可编程分频器，对本实验中的音阶发生电路的可编程计数器(实现可编程分频功能)用LFSR替代。

(10) 思考题2：例9-30中的进程DelaySpkS对扬声器发声有什么影响？

(11) 思考题3：在电路上应该满足哪些条件，才能用数字器件直接输出的方波驱动扬声器发声？

(12) 实验报告：用仿真波形和电路原理图，详细叙述硬件电子琴的工作原理及其4个VHDL文件中相关语句的功能，叙述硬件实验情况。

【例9-29】

```
LIBRARY IEEE; -- 硬件演奏电路顶层设计
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Songer IS
    PORT (   CLK12MHZ : IN STD_LOGIC;             --音调频率信号
             CLK8HZ   : IN STD_LOGIC;             --节拍频率信号
               CODE1  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);-- 简谱码输出显示
               HIGH1  : OUT STD_LOGIC; --高8度指示
             SPKOUT   : OUT STD_LOGIC );--声音输出
 END;
ARCHITECTURE one OF Songer IS
   COMPONENT NoteTabs
     PORT ( clk    : IN STD_LOGIC;
         ToneIndex : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
    END COMPONENT;
    COMPONENT ToneTaba
        PORT ( Index :  IN  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
               CODE  : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
               HIGH  : OUT STD_LOGIC;
               Tone  : OUT  STD_LOGIC_VECTOR (10 DOWNTO 0) );
    END COMPONENT;
```

（接下页）

```
COMPONENT Speakera
        PORT  ( clk  : IN STD_LOGIC;
                Tone : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                SpkS : OUT STD_LOGIC  );
    END COMPONENT;
    SIGNAL Tone : STD_LOGIC_VECTOR (10 DOWNTO 0);
    SIGNAL ToneIndex : STD_LOGIC_VECTOR (3 DOWNTO 0);
 BEGIN
 u1 : NoteTabs  PORT MAP (clk=>CLK8HZ, ToneIndex=>ToneIndex);
 u2 : ToneTaba PORT MAP
(Index=>ToneIndex,Tone=>Tone,CODE=>CODE1,HIGH=>HIGH1);
  u3 : Speakera PORT MAP(clk=>CLK12MHZ,Tone=>Tone, SpkS=>SPKOUT
);
 END;
```

【例9-30】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Speakera IS
    PORT (   clk  : IN STD_LOGIC;
             Tone : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
             SpkS : OUT STD_LOGIC  );
END;
ARCHITECTURE one OF Speakera IS
    SIGNAL PreCLK, FullSpkS : STD_LOGIC;
BEGIN
 DivideCLK : PROCESS(clk)
        VARIABLE Count4 : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
    BEGIN
        PreCLK <= '0';  --  将CLK进行16分频，PreCLK为CLK的16分频
        IF Count4>11 THEN PreCLK <= '1';  Count4 := "0000";
        ELSIF clk'EVENT AND clk = '1' THEN  Count4 := Count4 + 1;
        END IF;
    END PROCESS;
    GenSpkS : PROCESS(PreCLK, Tone)-- 11位可预置计数器
        VARIABLE Count11 : STD_LOGIC_VECTOR (10 DOWNTO 0);
BEGIN
      IF PreCLK'EVENT AND PreCLK = '1' THEN
       IF Count11 = 16#7FF# THEN Count11 := Tone ; FullSpkS <= '1';
```

（接下页）

```
 ELSE Count11 := Count11 + 1; FullSpkS <= '0'; END IF;
        END IF;
   END PROCESS;
  DelaySpkS : PROCESS(FullSpkS)--将输出再2分频，展宽脉冲，使扬声器有足够功率发音
        VARIABLE Count2 : STD_LOGIC;
 BEGIN
    IF FullSpkS'EVENT AND FullSpkS = '1' THEN  Count2 := NOT Count2;
          IF Count2 = '1' THEN  SpkS <= '1';
          ELSE SpkS <= '0';  END IF;
        END IF;
    END PROCESS;
 END;
```


【例9-31】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ToneTaba IS
    PORT ( Index :  IN  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
           CODE  : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
           HIGH  : OUT STD_LOGIC;
           Tone  : OUT  STD_LOGIC_VECTOR (10 DOWNTO 0) );
END;
ARCHITECTURE one OF ToneTaba IS
BEGIN
    Search : PROCESS(Index)
    BEGIN
        CASE Index IS     -- 译码电路，查表方式，控制音调的预置数
  WHEN "0000" => Tone<="11111111111" ; CODE<="0000"; HIGH <='0';-- 2047
  WHEN "0001" => Tone<="01100000101" ; CODE<="0001"; HIGH <='0';-- 773;
  WHEN "0010" => Tone<="01110010000" ; CODE<="0010"; HIGH <='0';-- 912;
  WHEN "0011" => Tone<="10000001100" ; CODE<="0011"; HIGH <='0';--1036;
  WHEN "0101" => Tone<="10010101101" ; CODE<="0101"; HIGH <='0';--1197;
  WHEN "0110" => Tone<="10100001010" ; CODE<="0110"; HIGH <='0';--1290;
  WHEN "0111" => Tone<="10101011100" ; CODE<="0111"; HIGH <='0';--1372;
  WHEN "1000" => Tone<="10110000010" ; CODE<="0001"; HIGH <='1';--1410;
  WHEN "1001" => Tone<="10111001000" ; CODE<="0010"; HIGH <='1';--1480;
  WHEN "1010" => Tone<="11000000110" ; CODE<="0011"; HIGH <='1';--1542;
  WHEN "1100" => Tone<="11001010110" ; CODE<="0101"; HIGH <='1';--1622;
  WHEN "1101" => Tone<="11010000100" ; CODE<="0110"; HIGH <='1';--1668;
  WHEN "1111" => Tone<="11011000000" ; CODE<="0001"; HIGH <='1';--1728;
  WHEN OTHERS => NULL;
  END CASE;
  END PROCESS;
END;
```

【例9-32】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY NoteTabs IS
    PORT ( clk    : IN STD_LOGIC;
        ToneIndex : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
END;
ARCHITECTURE one OF NoteTabs IS
COMPONENT MUSIC                --音符数据ROM
 PORT(address : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      inclock : IN STD_LOGIC ;
            q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END COMPONENT;
    SIGNAL Counter :  STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
    CNT8 : PROCESS(clk, Counter)
    BEGIN
        IF Counter=138 THEN  Counter <= "00000000";
        ELSIF  (clk'EVENT  AND  clk  =  '1')  THEN  Counter  <=
Counter+1; END IF;
    END PROCESS;
  u1   :   MUSIC   PORT   MAP(address=>Counter   ,   q=>ToneIndex,
inclock=>clk);
 END;
```

【例9-33】

```
WIDTH = 4 ; --"梁祝"乐曲演奏数据
DEPTH = 256 ;
ADDRESS_RADIX = DEC ;
DATA_RADIX = DEC ;
CONTENT  BEGIN
 00: 3 ; 01: 3 ; 02: 3 ; 03: 3; 04: 5; 05: 5; 06:  5;07: 6; 08: 8; 09: 8;
 10: 8 ; 11: 9 ; 12: 6 ; 13: 8; 14: 5; 15: 5; 16: 12;17: 12;18: 12; 19:15;
 20:13 ; 21:12 ; 22:10 ; 23:12; 24: 9; 25: 9; 26: 9; 27: 9; 28: 9; 29: 9;
 30: 9 ; 31: 0 ; 32: 9 ; 33: 9; 34: 9; 35:10; 36: 7; 37: 7; 38: 6; 39: 6;
 40: 5 ; 41: 5 ; 42: 5 ; 43: 6; 44: 8; 45: 8; 46: 9; 47: 9; 48: 3; 49: 3;
 50: 8 ; 51: 8 ; 52: 6 ; 53: 5; 54: 6; 55: 8; 56: 5; 57: 5; 58: 5; 59: 5;
 60: 5 ; 61: 5 ; 62: 5 ; 63: 5; 64:10; 65:10; 66:10; 67:12; 68: 7; 69: 7;
 70: 9 ; 71: 9 ; 72: 6 ; 73: 8; 74: 5; 75: 5; 76: 5; 77: 5; 78: 5; 79: 5;
 80: 3 ; 81: 5 ; 82: 3 ; 83: 3; 84: 5; 85: 6; 86: 7; 87: 9; 88: 6; 89: 6;
 90: 6 ; 91: 6 ; 92: 6 ; 93: 6; 94: 5; 95: 6; 96: 8; 97: 8; 98: 8; 99: 9;
100:12 ;101:12 ;102:12 ;103:10;104: 9;105: 9;106:10;107: 9;108: 8;109: 8;
110: 6 ;111: 5 ;112: 3 ;113: 3;114: 3;115: 3;116: 8;117: 8;118: 8;119: 8;
120: 6 ;121: 8 ;122: 6 ;123: 5;124: 3;125: 5;126: 6;127: 8;128: 5;129: 5;
130: 5 ;131: 5 ;132: 5 ;133: 5;134: 5;135: 5;136: 0;137: 0;138: 0;
END ;
```


实 验 与 设 计

9-2采用高速A/D的存储示波器设计

(1) 实验目的：学习利用FPGA控制高速ADC、示波器显示控制方法等。

(2) 实验原理：图9-4所示的是基于大学生电子设计竞赛赛题的存储示波器结构图，FPGA中的A/D采样控制器负责对A/D对模拟信号的采样控制，并将A/D转换好的数据送到FPGA的内部RAM中存储；RAM的地址信号由地址发生计数器产生。当完成1至数个周期的被测信号的采样后，在地址发生计数器的地址扫描下，将存于RAM中的数据通过外部的D/A进入示波器的Y端；与此同时，地址发生计数器的地址信号分频后通过另一个D/A构成锯齿波信号，进入示波器的X端。从而实现存储示波器的功能。

实 验 与 设 计

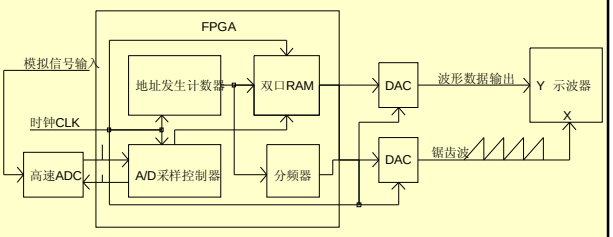


图9-4 存储示波器结构简图

实 验 与 设 计

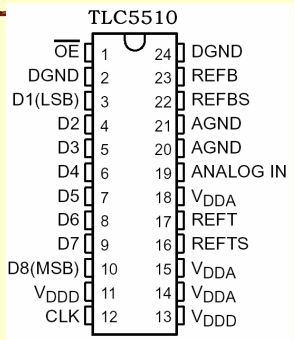


图9-5 TLC5510引脚图

实 验 与 设 计

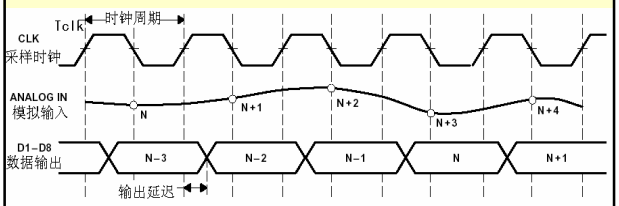


图9-6 TLC5510采样时序图

实 验 与 设 计

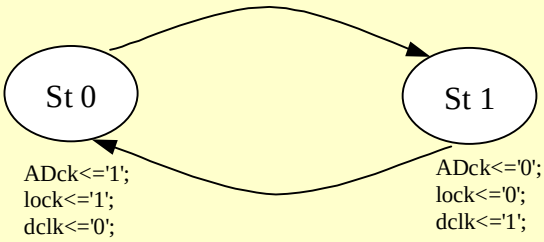


图9-7 TLC5510采样控制状态图

实 验 与 设 计

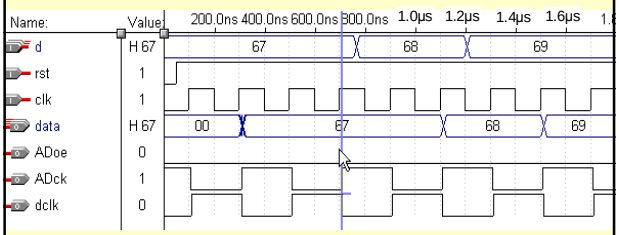


图9-8 A/D转换仿真波形

【例9-34】-- TLC5510 采样控制示例

```
library IEEE;
use     IEEE.STD_LOGIC_1164.ALL;
entity ad5510 is
     port( rst, clk : in  std_logic; --rst 复位;clk采样时钟输入
           d    : in    std_logic_vector(7 downto 0);-- 8位A/D数据
     Adck, ADoe  : out std_logic;-- Adck, Adoe分别为TLC5510的CLK和的OE
          data   : out    std_logic_vector(7 downto 0);-- 8位数据
          dclk   : out    std_logic  );            -- 数据输出锁存信号
end ad5510;
architecture ADCTRL of ad5510 is
        type adsstates is (sta0,sta1);             --定义两个状态变量
        signal    ads_state,next_ads_state : adsstates;
        signal    lock    : std_logic;
begin
ads : PROCESS( ads_state)   -- A/D 采样控制状态机
BEGIN
       CASE ads_state IS
  WHEN sta0 => ADck<='1'; lock<='1'; dclk<='0';next_ads_state <= sta1;
  WHEN sta1 => ADck<='0'; lock<='0'; dclk<='1';next_ads_state <= sta0;
  WHEN OTHERS => ADck<='0'; lock<='0'; dclk<='1';next_ads_state <= sta0;
    END CASE ;
END PROCESS;
PROCESS (CLK,rst)                                  (接下页 )
```

```
BEGIN
        IF RST ='0' THEN   ads_state <= sta0;
          ELSIF ( CLK'EVENT AND CLK='1')  THEN
             ads_state <= next_ads_state;  -- 在时钟上升沿，转换至下一状态
       END IF;
END PROCESS;
PROCESS (lock,rst) -- 此进程中，在lock的上升沿，将转换好的数据锁入
       BEGIN
        IF RST ='0' THEN   data <= (others => '0');
          ELSIF lock'EVENT AND lock='1' THEN   data <= D ;   END IF;
END PROCESS ;
ADoe <= '0';
end ADCTRL;
```

【例9-35】 -- TLC5510的另一种采样控制方法

```
...
lock <= clk;  ADck <= clk; dclk <= not lock; ADoe <= '0';
PROCESS (lock,rst) -- 此进程中，在lock的上升沿，将转换好的数据锁入
BEGIN
       if rst <= '0' then  data <= (others => '0');
       ELSIF lock'EVENT AND lock='1' THEN  data <= D ; END IF;
END PROCESS ;
end logi;
```


【例9-36】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY RESERV IS
      PORT(CLK, KEY1 : IN  STD_LOGIC;--时钟（20MHz）和采样/显示控制键
                TRAG : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);--扫描锯齿波输出
           DOUT : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);--波形输出
            ADIN   : IN  STD_LOGIC_VECTOR (7 DOWNTO 0)  ); --A/D采样数据输入
END;
ARCHITECTURE DACC OF RESERV IS
COMPONENT DPRAM   --采样数据存储用RAM
        PORT ( address : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
          inclock, wren: IN STD_LOGIC ;
                  data  : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
                     q   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END COMPONENT;
      SIGNAL    Q1   : STD_LOGIC_VECTOR (9 DOWNTO 0);
      SIGNAL MD0,DIN : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
PROCESS(CLK)
      BEGIN
        IF rising_edge(CLK) THEN  Q1 <= Q1 + 1;  END IF; --生成锯齿波计数器
END PROCESS;
process(CLK, ADIN)
begin
    if rising_edge(CLK) then DIN <= ADIN ; end if;  --采样数据锁存进RAM
end process;
   DOUT(9 DOWNTO 2)<=MD0; TRAG<=Q1; DOUT(1 DOWNTO 0) <= "00";
  u1 : DPRAM PORT MAP(data=>DIN, wren=>KEY1, address=>Q1, q=>MD0, inclock=>CLK);
END;
```

实 验 与 设 计

9-2采用高速A/D的存储示波器设计

(3) 实验内容1：对例9-36给出仿真波形，并分析；引脚锁定后，进行硬件验证（包括定制LPM_RAM）。时钟首先输入13MHz。注意打开系统的+/-13V电源开关。用示波器的Y1（X）端接GWADDA板的5651 D/A输出；GWADDA板上5510 A/D口的“AIN”接受来自主系统模拟波形，即接主系统板上右侧“JP17”的“OUTPUT”端，然后将主系统板上“JP18”的“INPUT”端与系统右下角的时钟65536或32768HZ等相接。“JL11”的3针座短路“H_F”端，调节“JP15”电位器，使得主系统右侧的“WAVE OUT”端输出正常信号波形（用示波器监视，在4V上下）。GWADDA板上跳线JAD的短路帽插“20MHz”端，这时VHDL程序时钟CLK锁定在137脚（EP1C3）；此时AD5510和DA5651的工作时钟与FPGA的工作时钟完全一样，都是20MHz。电路图选择No.5，选择键1高电平为采样，低电平为存储显示。

波形输出：Y1（X）端接GWADDA板的5651的A输出端，作X作锯齿波输出；

波形输出：Y2（Y）端接GWADDA板的5651的B输出端，作Y作波形输出。

实 验 与 设 计

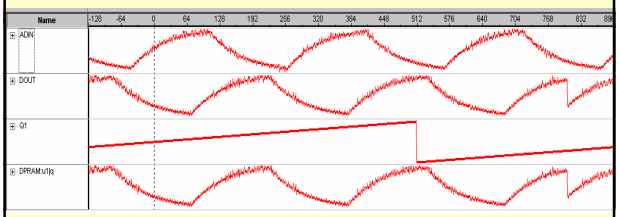


图9-9 存储示波器采样电路系统内SignalTapII数据波形

实 验 与 设 计

9-2采用高速A/D的存储示波器设计

(4) 实验内容2：图9-9所示的是此存储示波器采样电路系统内SignalTapII给出的数据波形，其中ADIN是直接来自5510的采样波形；Q1是RAM的地址波形；q是DPRAM输出的存储波形，显然此波形有缺陷，原因是每一次采如RAM中的波形数据都不是整数倍的波形。为了改善输出波形的显示效果，设计一个状态机，严格控制进入RAM中的数据在地址发生器一个扫描周期中含被采样信号整数个完整周期的采样数据。

实 验 与 设 计

9-3 循环冗余校验(CRC)模块设计

(1) 实验目的：设计一个在数字传输中常用的校验、纠错模块：循环冗余校验CRC模块，学习使用FPGA器件完成数据传输中的差错控制。

(2) 实验原理：CRC即Cyclic Redundancy Check 循环冗余校验，是一种数字通信中的信道编码技术。经过CRC方式编码的串行发送序列码，可称为CRC码，共由两部分构成：k位有效信息数据和r位CRC校验码。其中r位CRC校验码是通过k位有效信息序列被一个事先选择的r+1位"生成多项式"相"除"后得到的(r位余数即是CRC校验码)，这里的除法是"模2运算"。CRC校验码一般在有效信息发送时产生，拼接在有效信息后被发送；在接收端，CRC码用同样的生成多项式相除，除尽表示无误，弃掉r位CRC校验码，接收有效信息；反之，则表示传输出错，纠错或请求重发。

实 验 与 设 计

sdata：12位的待发送信息；　　datald：sdata的装载信号；
error：误码警告信号　；　　datafini：数据接收校验完成；
rdata：接收模块(检错模块)接收的12位有效信息数据；　　clk：时钟信号；
datacrc：附加上5位CRC校验码的17位CRC码，在生成模块被发送，在接收模块被接收；
hsend、hrecv：生成、检错模块的握手信号，协调相互之间关系；

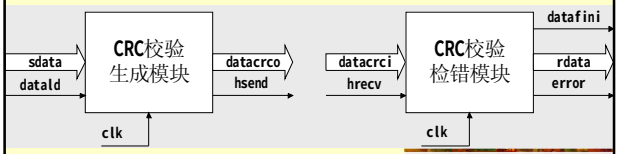


图9-10 CRC模块

【例9-37】

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;
ENTITY crcm IS
    PORT (clk, hrecv, datald : IN std_logic;
          sdata     : IN std_logic_vector(11 DOWNTO 0);
          datacrco  : OUT std_logic_vector(16 DOWNTO 0);
          datacrci  : IN std_logic_vector(16 DOWNTO 0);
            rdata   : OUT std_logic_vector(11 DOWNTO 0);
          datafini  : OUT std_logic;
          ERROR0, hsend  : OUT std_logic);
END crcm;
ARCHITECTURE comm OF crcm IS
    CONSTANT multi_coef : std_logic_vector(5 DOWNTO 0) := "110101";
            -- 多项式系数, MSB一定为'1'
    SIGNAL  cnt,rcnt : std_logic_vector(4 DOWNTO 0);
    SIGNAL  dtemp,sdatam,rdtemp   : std_logic_vector(11 DOWNTO 0);
    SIGNAL  rdatacrc: std_logic_vector(16 DOWNTO 0);
    SIGNAL  st,rt  : std_logic;
BEGIN
PROCESS(clk)
    VARIABLE crcvar : std_logic_vector(5 DOWNTO 0);
BEGIN
    IF(clk'event AND clk = '1') THEN
    IF(st = '0' AND datald = '1') THEN  dtemp <= sdata;
```

（接下页）

```
 sdatam <= sdata; cnt <= (OTHERS => '0');  hsend <= '0';  st <= '1';
       ELSIF(st = '1' AND cnt < 7) THEN  cnt <= cnt + 1;
       IF(dtemp(11) = '1') THEN  crcvar := dtemp(11 DOWNTO 6) XOR multi_coef;
          dtemp <= crcvar(4 DOWNTO 0) & dtemp(5 DOWNTO 0) & '0';
          ELSE  dtemp <= dtemp(10 DOWNTO 0) & '0';   END IF;
    ELSIF(st='1' AND cnt=7) THEN datacrco<=sdatam & dtemp(11 DOWNTO 7);
          hsend <= '1';  cnt <= cnt + 1;
       ELSIF(st='1' AND cnt=8) THEN   hsend<= '0';    st<='0';
       END IF;
   END IF;
END PROCESS;
PROCESS(hrecv,clk)
   VARIABLE rcrcvar : std_logic_vector(5 DOWNTO 0);
BEGIN
 IF(clk'event AND clk = '1') THEN
  IF(rt = '0' AND hrecv = '1') THEN  rdtemp <= datacrci(16 DOWNTO 5);
   rdatacrc <= datacrci;  rcnt <= (OTHERS => '0');
   ERROR0 <= '0';    rt <= '1';
   ELSIF(rt= '1' AND rcnt < 7) THEN  datafini <= '0';  rcnt <= rcnt + 1;
          rcrcvar := rdtemp(11 DOWNTO 6) XOR multi_coef;
          IF(rdtemp(11) = '1') THEN
           rdtemp <= rcrcvar(4 DOWNTO 0) & rdtemp(5 DOWNTO 0) & '0';
          ELSE  rdtemp <= rdtemp(10 DOWNTO 0) & '0';
          END IF;
       ELSIF(rt = '1' AND rcnt = 7) THEN  datafini <= '1';
          rdata <= rdatacrc(16 DOWNTO 5);  rt <= '0';
          IF(rdatacrc(4 DOWNTO 0) /= rdtemp(11 DOWNTO 7)) THEN
            ERROR0 <= '1'; END IF;
       END IF;
   END IF;
END PROCESS;
END comm;
```

实 验 与 设 计

9-3 循环冗余校验(CRC)模块设计

(3) 实验内容1：编译以上示例文件，给出仿真波形。

(4) 实验内容2：建立一个新的设计，调入crcm模块，把其中的CRC校验生成模块和CRC校验查错模块连接在一起，协调工作。引出必要的观察信号，锁定引脚，并在EDA实验系统上实现之。

(5) 思考题1：例中对st、rt有不妥之处，试解决之(提示：复位reset信号的引入有助于问题的解决)。

(6) 思考题2：如果输入数据、输出CRC码都是串行的，设计该如何实现(提示：采用LFSR)。

(7) 思考题3：在例子程序中需要8个时钟周期才能完成一次CRC校验，试重新设计使得在一个clk周期内完成。

(8) 实验报告：叙述CRC的工作原理，将设计原理、程序设计与分析、仿真分析和详细实验过程。