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

```
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

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

【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;
```

【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 9-1 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

例化名 ： 元件名 GENERIC MAP(类属表)

【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;
```

【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

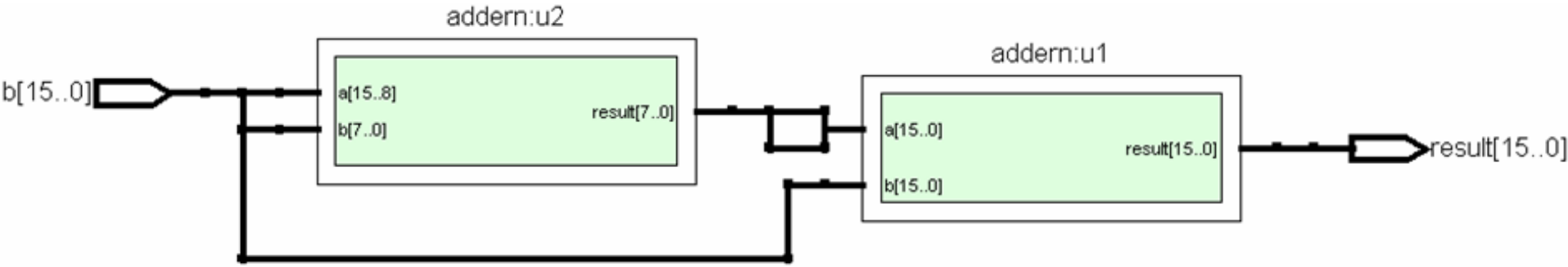

图 9-1　例 9-4 的 RTL 电路图

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

1. 结构体的一般语言格式

```
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;
```

2. 结构体说明语句

9.2 结 构 体

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

9.3 子 程 序

9.3.1 函数

```
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;
```

接下页

接上页

```
  END FUNCTION max;                --结束 FUNCTION 语句
END;                               --结束 PACKAGE  BODY 语句
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

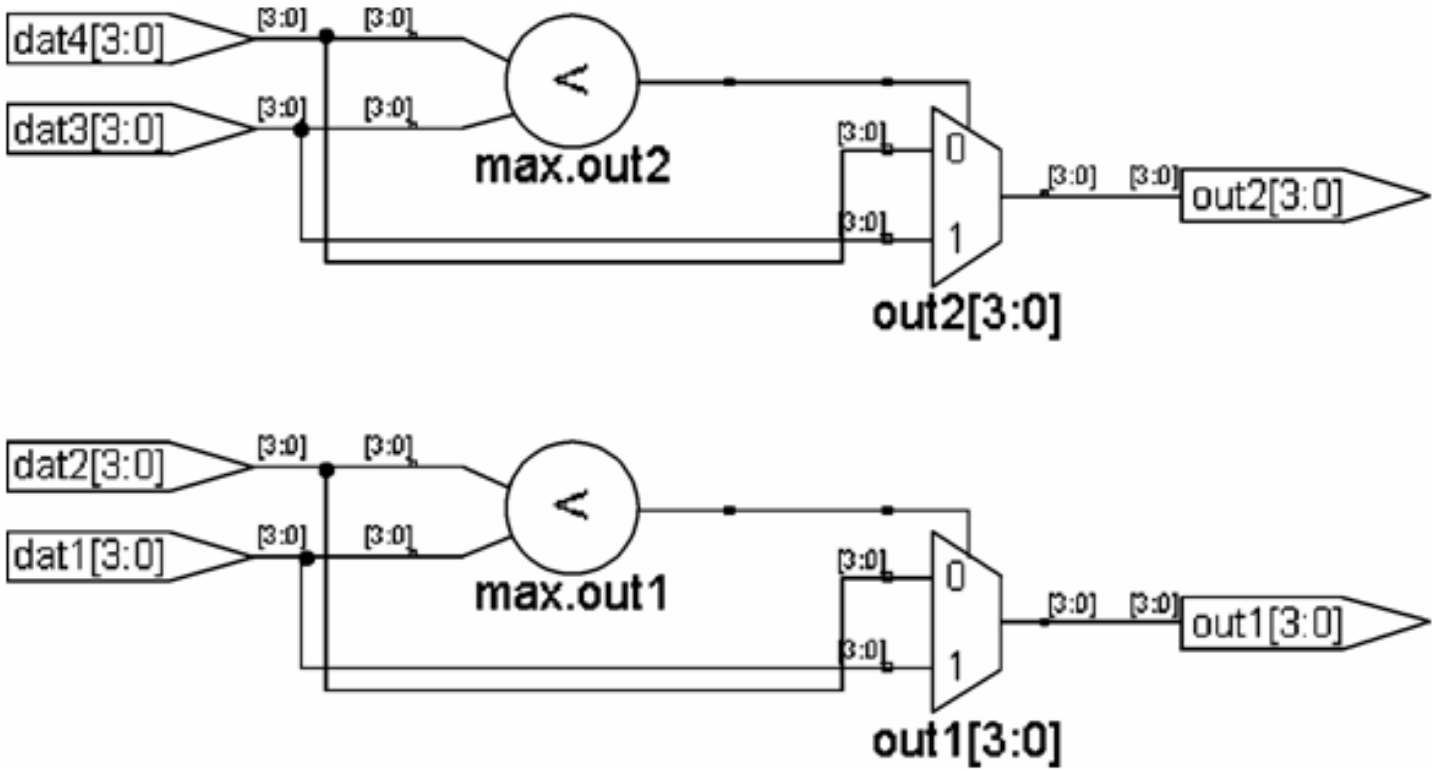

图 9-2　例 9-5 的逻辑电路图

【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 y  ;
 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】

```
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
```

接下页

9.3 子 程 序

接上页

```
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 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
```

接下页

接上页

```
   -- 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
  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  ;
  --  以下是程序包体
LIBRARY IEEE ;
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
```

接下页

接上页

```
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

【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;

 FUNCTION conv_std_logic_vector(a: integer, integer size)
          RETURN std_logic_vector ;
 FUNCTION conv_integer(a: std_logic_vector) RETURN integer ;
```

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;
```

9.3 子 程 序

表 9-1 IEEE 库类型转换函数表

函数名	功 能
所在程序包: STD_LOGIC_1164	
to_stdlogicvector(A)	由 bit_vector 类型转换为 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 类型转换成 bit 类型
所在程序包: STD_LOGIC_ARITH	
conv_std_logic_vector(A, 位长)	将 integer 转换成 std_logic_vector 类型，A 是整数
conv_integer(A)	将 std_logic_vector 转换成 integer
conv_unsigned(A, 位长)	将 unsigned，signed，integer 类型转换为指定位长的 unsigned 类型
conv_signed(A, 位长)	将 unsigned，signed，integer 类型转换为指定位长的 signed 类型
所在程序包: STD_LOGIC_UNSIGNED	
conv_integer(A)	由 std_logic_vector 转换成 integer

【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;
  END LOOP;
```

接下页

9.3 子 程 序

9.3.3 转换函数

接上页

```
  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

决断函数不可综合，主要用于**VHDL**仿真中解决信号被多个驱动源驱动时，驱动信号间的竞争问题。

当多个驱动源都同时产生一个处理事项，只有其中一个驱动源的信号值能赋给被驱动的信号。

决断函数输入一般是单一变量，多个驱动源的信号值组成非限定数组，
多个信号驱动源，其信号值组成的未限定数组可依次类推。

决断函数调用后返回的是单一信号值，称决断信号值。

9.3 子 程 序

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);
```

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 ;    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, sign1);      -- 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

std_logic_1164　　Numeric_Bit

Numeric_Std　　Math _Real　　Math _Complex

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 VHDL程序包

```
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容
END  程序包名;
```

9.5 VHDL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 VHDL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

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)

```
SIGNAL d1,d2,d3,d4,d5, : INTEGER RANGE 0 TO 255;--全部定义为整数类型
 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)--也是整数类型!
 d4 <= 8#376#      ;          -- (八进制表示，等于 254)
 d5 <= 16#A#E3     ;          -- (十六进制表示，等于 16#A000#)
```

G <= conv_std_logic_vector(16#A#E3, 16); --假设 G 的类型是 std_logic_vector

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

```
'R' , 'a' , '*' , 'Z' , 'U' , '0' , '11' , '-' , 'L' …
TYPE STD_ULOGIC IS ( 'U', 'X', '0', '1', 'W', 'L', 'H', '-' )

"ERROR" , "Both S and Q equal to 1" , "X" , "BB$CC"
```

“B”、 “O”、 “X”

```
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 <= "101010101010"          --表达正确。这里可以略去 B，但不可加下划线
data6 <= "0AD0"                            --表达错误，缺 X。
```

9.7 VHDL文字规则

9.7.3 标识符及其表述规则

```
Decoder_1   ,   FFT   ,   Sig_N   ,   Not_Ack   ,   State0   ,   Idle
```

```
_Decoder_1              -- 起始为非英文字母
74LS164                 -- 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)**：包括实数类型、整数类型、枚举类型、时间类型。

- 复合类型**(Composite Type)**：可以由小的数据类型复合而成，如可由标量型复合而成。复合类型主要有数组型**(Array)**和记录型**(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 VHDL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 ;
```

```
SIGNAL a : BIT_VECTOR(7 TO 0);
```

9.8 数 据 类 型

9.8.1 VHDL预定义数据类型

4. 字符类型

5. 整数类型　　－2147483647～＋2147483647

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

9.8 数 据 类 型

9.8.1 VHDL预定义数据类型

7. 字符串类型

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

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

9.8.1 VHDL预定义数据类型

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

```
LIBRARY IEEE ;
USE IEEE.STD_LOIGC_ARITH.ALL ;
```

1. 无符号数据类型

```
UNSIGNED'("1000")
```

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

9.8 数 据 类 型

9.8.3 其他预定义标准数据类型

2. 有符号数据类型

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 数据类型转换示例

【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 数据类型转换示例

【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;
```

【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 ;
```

9.8 数 据 类 型

9.8.4 数据类型转换示例

表 9-2　可综合的数据类型归纳

数据类型	可综合的取值范围
bit, bit_ vector	'1', '0'
std_logic, std_logic_ vector	“X”, “0”, “1”“Z”
boolean	true,false
natural	0~2 147 483 647
integer	-2 147 483 647~+2 147 483 647
signed	-2 147 483 647~+2 147 483 647
unsigned	0~2 147 483 647
用户自定义类型	用户自定义数组或元素
数组类型（array）	可综合数据类型的组合
记录类型（record）	可综合数据类型的任意组合
子类型（subtype）	数据类型的子集

9.8 数 据 类 型

9.8.4 数据类型转换示例

【例 9-21】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL ;
USE IEEE.STD_LOGIC_ARITH.ALL ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL ;
USE IEEE.STD_LOGIC_SIGNED.ALL ;
ENTITY CNT10 IS
  PORT (A,B : IN  UNSIGNED(3 downto 0);
        C,D : IN    SIGNED(3 downto 0);
        RAB : OUT UNSIGNED(3 downto 0);
        RCD : OUT   SIGNED(3 downto 0);
     RU,RS  : OUT   BOOLEAN);
END ;
ARCHITECTURE rtl of CNT10 is
BEGIN
RAB<= A+B ; RCD<= C+D ; RU<= (A>B)  ; RS<= (C>D) ;
END;
```

9.8 数 据 类 型

9.8.4 数据类型转换示例

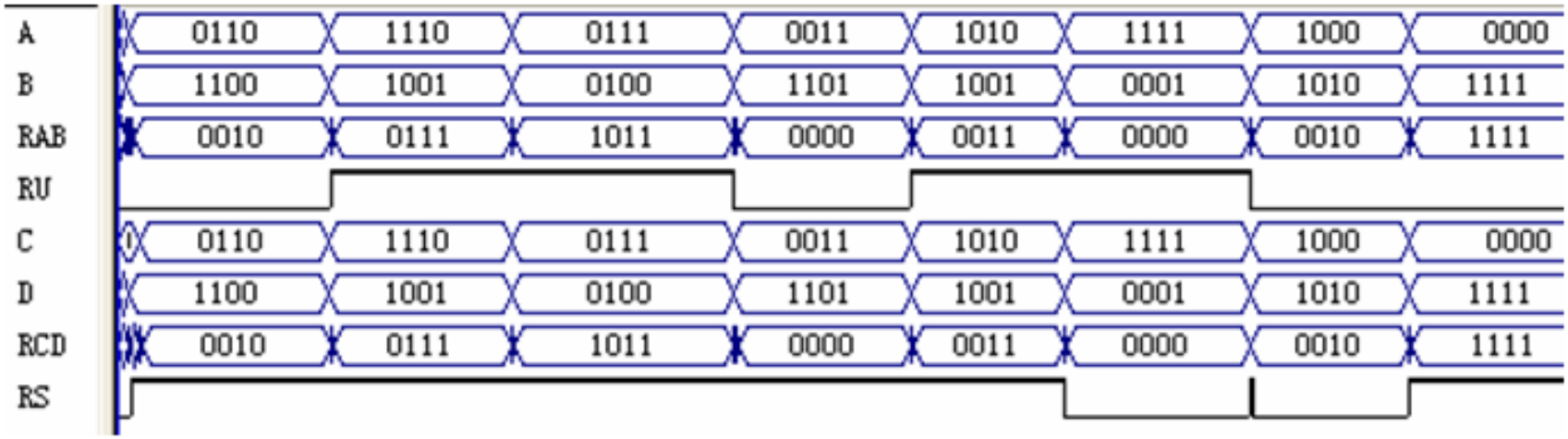

图 9-3　例 9-21 的仿真结果

9.9 VHDL操作符

9.9.1 逻辑操作符

逻辑操作符**(Logical Operator)**

关系操作符**(Relational Operator)**

算术操作符**(Arithmetic Operator)**

符号操作符**(Sign Operator)**

重载操作符**(Overloading Operator)**。

```
A and B and C and D    或    (A or B)xor C
```

9.9 VHDL操作符

9.9.1 逻辑操作符

表 9-3　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
	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

9.9 VHDL操作符

9.9.1 逻辑操作符

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 VHDL操作符

9.9.1 逻辑操作符

表 9-4　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 VHDL操作符

【例 9-22】

```
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 STD_LOGIC，而 l 的数据类型是
...                           -- 布尔量，因而不能相互作用，表达错误
```

9.9 VHDL操作符

9.9.2 关系操作符

“ = ”(等于)、“/=”(不等于)、“ >”(大于)、

“< ”(小于)、“>=”(大于等于)和“<=”(小于等于)

```
'1' = '1';    "101" = "101";    "1" > "011";    "101" < "110";
```

9.9 VHDL操作符

9.9.2 关系操作符

【例 9-23】

```
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 VHDL操作符

9.9.2 关系操作符

【例 9-24】

```
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 VHDL操作符

9.9.3 算术操作符

表 9-5　算术操作符分类表

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

9.9 VHDL操作符

9.9.3 算术操作符

1. 求和操作符

【例 9-25】

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

【例 9-26】

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

9.9 VHDL操作符

9.9.3 算术操作符

1. 求和操作符

【例 9-27】

```
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 VHDL操作符

9.9.3 算术操作符

2. 求积操作符

***(乘)、 /(除)、 MOD(取模) 、 RED(取余)**

3. 符号操作符 **“+”和“—”**

```
z := x * (—y);
```

4. 混合操作符 **“**”** **“ABS”**

【例 9-28】

```
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 VHDL操作符

9.9.3 算术操作符

5. 移位操作符　　**SLL、SRL、SLA、SRA、ROL 、ROR**

标识符　移位操作符　移位位数 ；

【例 9-29】

```
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、设计实体的概念。
9-2 举例说明**GENERIC**说明语句和**GENERIC**映射语句有何用处。
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**STRING**、**TIME**、**REAL**、**BIT**数据类型中，**VHDL**综合器支持哪些类型？
9-6 详细说明例**9-29**中的语句作用和程序实现的功能。
9-7 表达式**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**中有哪三种数据对象？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。
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？

习　题

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。

9-16 利用循环语句和移位操作符实现移位相加方式的纯组合电路**8**位乘法器设计。

实验与设计

9-1 乐曲硬件演奏电路设计

（1）实验目的：

（2）实验原理：

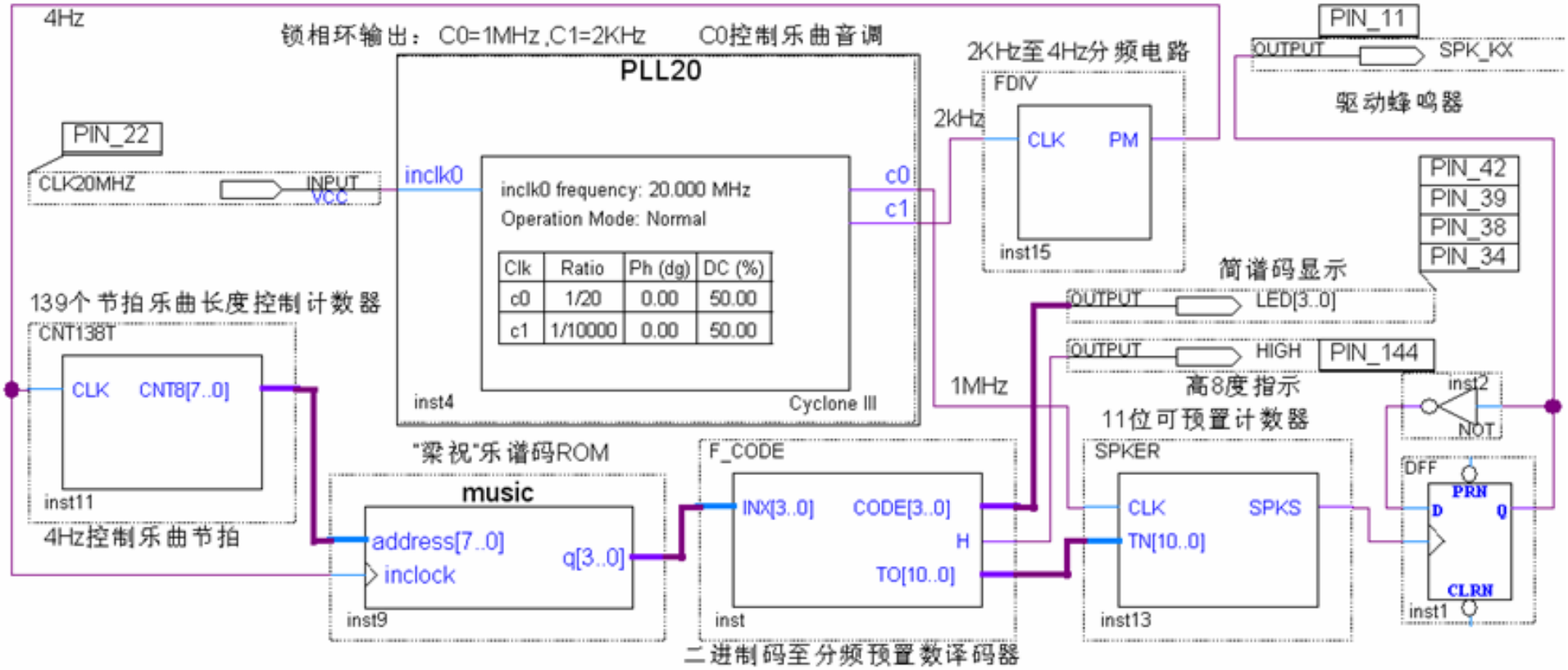

图 9-4 乐曲演奏电路顶层设计

实验与设计

9-1 乐曲硬件演奏电路设计

（1）实验目的：

（2）实验原理：

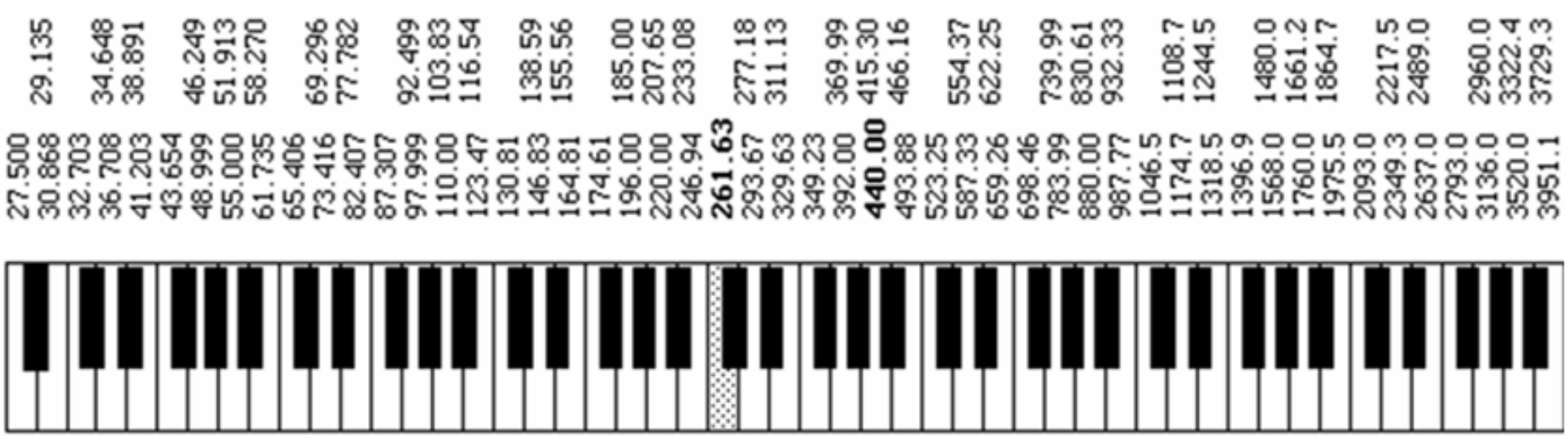

图 9-5 电子琴音阶基频对照图（单位 Hz）

实验与设计

9-1 乐曲硬件演奏电路设计

（3）实验内容1：

（4）实验内容2：

（5）实验内容3：

（6）实验内容4：

（7）实验内容5：

（8）实验内容6：

（9）实验内容7：

（10）实验报告：5E+系统的演示文件：/KX_7C5EE+/EXPERIMENTs/EXP4_Music/。

【例 9-30】

```
  WIDTH = 4 ; //“梁祝”乐曲演奏数据
  DEPTH = 256 ; //实际深度 139
  ADDRESS_RADIX = DEC ; //地址数据类是十进制
  DATA_RADIX = DEC ;     //输出数据的类型也是十进制
  CONTENT  BEGIN         //注意实用文件中要展开以下数据，每一组占一行
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

9-1 乐曲硬件演奏电路设计

（3）实验内容1：

（4）实验内容2：

（5）实验内容3：

（6）实验内容4：

（7）实验内容5：

（8）实验内容6：

（9）实验内容7：

（10）实验报告：5E+系统的演示文件：/KX_7C5EE+/EXPERIMENTs/EXP4_Music/。

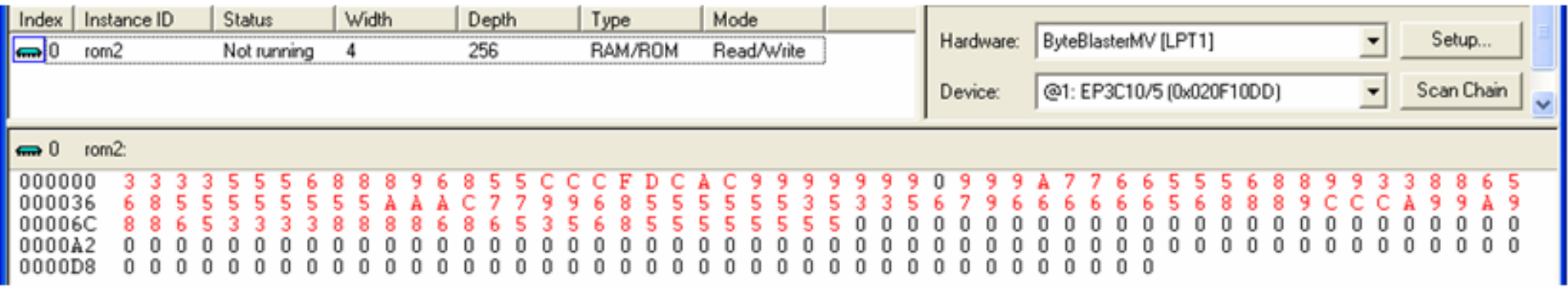

图 9-6 In-System Memory Content Editor 对 MUSIC 模块的数据读取

【例 9-31】

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

实验与设计

9-2 数字彩色液晶显示控制电路设计

（1）实验任务1：

基于5E+系统的基本控制演示示例是：/KX_7C5EE+/EXPERIMENTs/EXP13_COLOR_LCD/

（2）实验任务2：

（3）实验任务3：

（4）实验任务4：

（5）实验任务5：

（6）实验任务6：

演示示例：/KX_7C5EE+/DEMOs/EXPL9_Super_Mario2/,和/DEMOs/EXP7_LCD_light_GAME/。

实验与设计

9-3 GPS应用的通信电路设计

实验任务：

参考**GPS**模块使用文件：**/KX_7C5EE+/GPS**文件。常用的**GPS**模块是**UART**通信方式。可以用两种方式读取**GPS**模块中的数据：

1、软件方式。可以根据实验**6-8**，使用**FPGA**中的**8051**核与**GPS**通信，并将数据用液晶显示出来；

2、硬件方式，即不使用任何**CPU**。可以根据实验**7-6**的原理设计**UART**硬件特性模块，读取**GPS**模块的数据，并显示于数码管或液晶屏上。演示示例：**/KX_7C5EE+/EXPERIMENTs/EXP16_KX8051_FTEST_RS232/**和**/EXP17_KX8051_GPS_FTEST/**。

实验与设计

9-3 GPS应用的通信电路设计

实验任务：

演示示例：**/KX_7C5EE+/EXPERIMENTs/EXP16_KX8051_FTEST_RS232/**和**/EXP17_KX8051_GPS_FTEST/**。

9-4 VGA动画图像显示控制电路设计

实验任务：

相关演示示例有：鼠标控制的**VGA**显示游戏：**/KX_7C5EE+/DEMOs/ EXPL12_PS2Mouse_VGA_GAME/**；键盘控制的两个游戏：**/KX_7C5EE+/DEMOs/EXPL1_VGA_GAME_ARK/**；和**/EXPL2_VGA_GAME_pong/**；

32位简单**CPU**设计：**/KX_7C5EE+/DEMOs/EXPL16_MIPS_COMPUTER_VGA/**。

实验与设计

9-5 PS2键盘控制模型电子琴电路设计

（1）实验原理：

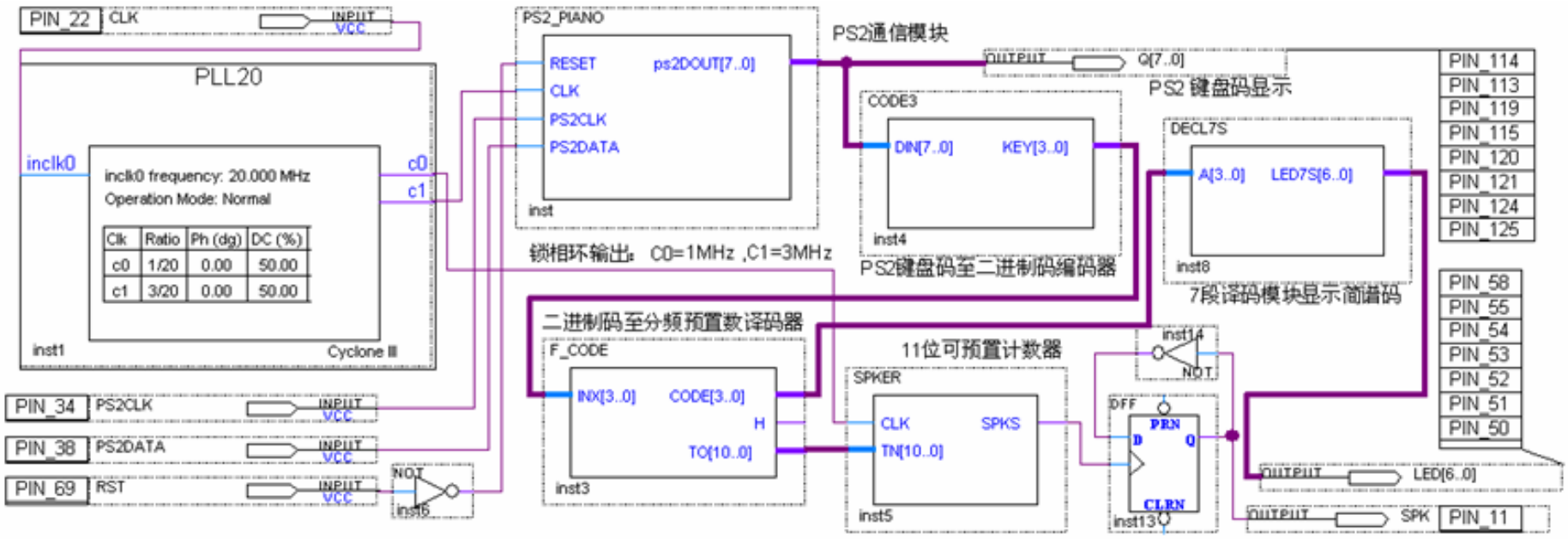

图 9-7　PS2 键盘控制模型电子琴电路顶层设计

表 9-6　PS2 键盘键控与输出码对照表

Key	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Data	1C	32	21	23	24	2B	34	33	43	3B	42	4B	3A	31	44
Key	P	Q	R	S	T	U	V	W	X	Y	Z	0	1	2	3
Data	4D	15	2D	1B	2C	3C	2A	1D	22	35	1A	45	16	1E	26
Key	4	5	6	7	8	9	`	-	=	\	]	;	’	,	.
Data	25	2E	36	3D	3E	46	0E	4E	55	5D	5B	4C	52	41	49
Key	/	[	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	KP 0
Data	4A	54	05	06	04	0C	03	0B	83	0A	01	09	78	07	70
Key	KP1	KP2	KP3	KP4	KP5	KP6	KP7	KP8	KP9	KP .	KP-	KP+	KP/	KP*	END
Data	69	72	7A	6B	73	74	6C	75	7D	71	7B	79	4A	7C	69

Key	BKSP	SPACE	TAB	CAPS	L SHFT	L CTRL	L CUI	L ALT	R SHFT	R CTRL	R CUI
Data	66	29	0D	58	12	14	1F	11	59	14	27

Key	R ALT	APPS	ENTER	ESC	INSERT	HOME	PG UP	DELETE	PG DN	NUM
Data	11	2F	5A	76	70	6C	7D	71	7A	77

Key	U ARROW	L ARROW	D ARROW	R ARROW	KP EN	SCROLL	PRNT SCRN	PAUSE
Data	75	6B	72	74	5A	7E	12 7C	77

实验与设计

9-5 PS2键盘控制模型电子琴电路设计

（2）实验内容：

此实验基于5E+系统的示例演示：/KX_7C5EE+/DEMOs/EXPL12_PS2Mouse_VGA_GAME/。 PS/2鼠标接5E+系统的上方的PS/2接口，再接VGA显示器，按键K8复位；移动鼠标即可在VGA上显示鼠标光标，并做游戏。

实验与设计

9-6 乒乓球游戏电路设计

(1) 实验内容**1**：

演示文件：**/KX_7C5EE+/DEMOs/EXPL15_PINPANG_GAME/PINPANG**。

(2) 实验内容**2**：。

9-7 基于CPLD的FPGA PS模式编程配置控制电路设计

（**1**）实验目的：

（**2**）实验原理：

（**3**）实验任务**1**：

（**4**）实验任务**2**：

示例评估板和示例演示程序：

/KX_7C5EE+/DEMOs/EXPL17_FPGA_PS_CONFIG/

实验与设计

9-8 基于M9K RAM型LPM移位寄存器设计

9-9 基于FT245BM的USB通信控制模块设计

实验任务：

示例评估板和示例演示程序：

/KX_7C5EE+/EXPERIMENTs/EXP38_USB_FT245/