工科高等院校电路系列课程教材

数字电路EDA入门

——VHDL程序实例集

张亦华　延　明　编著

北京邮电大学出版社
www.buptpress.com

责任编辑：孙伟玲　徐夙琨
封面设计：燕铭设计 YIMING DESIGN

工·科·高·等·院·校·电·路·系·列·课·程·教·材

电子电路与计算机辅助分析设计

电子电路学习指导与解题指南

电子测量与电子电路实验

电路分析基础解题指南

通信电子电路

数字逻辑设计实验技术与EDA工具

数字电路EDA入门——VHDL程序实例集

ISBN 7-5635-0673-X
9 787563 506736 >

TN79
Z299

ISBN 7-5635-0673-X/TN·284
定价：17.00元

数字电路 EDA 入门

——VHDL 程序实例集

张亦华　延　明　编著

北京邮电大学出版社
·北京·

内 容 提 要

本书是数字电路电子设计自动化(EDA)入门的工具书。其内容主要包括：用 VHDL 设计的基本组合电路、时序电路、数字综合电路、电路图输入法要领概述、实用 VHDL 语句等；附录部分介绍了 VHDL 基本知识和基本术语，中小规模集成电路等。

本书提供的所有程序代码都经过 MAX + plus Ⅱ 9.23 软件和 PLD 器件的编译、仿真、下载和实际测量，可以作为进一步开发的参考。大部分实例电路都是在设计数字电路时经常使用的电路。本书为那些想快速步入 EDA 设计大门的读者提供了一个仿制、借鉴、研究、创新的良好工作平台。

本书可作为大学本科和专科院校通信、电子工程类专业的 EDA 实验教材，也可供从事 EDA 工作的技术人员作程序开发的参考书。

图书在版编目(CIP)数据

数字电路 EDA 入门：VHDL 程序实例集/张亦华，延明编著. —北京：北京邮电大学出版社，2003
ISBN 7-5635-0673-X

Ⅰ. 数… Ⅱ. ①张…②延… Ⅲ. ①数字电路—电路设计：计算机辅助设计②硬件描述语言，VHDL—程序设计 Ⅳ. TN790.2

中国版本图书馆 CIP 数据核字(2003)第 010620 号

出 版 者：北京邮电大学出版社(北京市海淀区西土城路 10 号)
邮编：100876 发行部电话/传真：62282185/62283578
网址：http://www.buptpress.com
经 销：各地新华书店
印 刷：北京忠信诚胶印厂
印 数：4 000 册
开 本：787 mm × 1 092 mm 1/16
印 张：10.5
字 数：275 千字
版 次：2003 年 3 月第 1 版 2003 年 3 月第 1 次印刷
书 号：ISBN 7-5635-0673-X/TN·284
定 价：17.00 元

前　言

本书的编写遵循“基本、实用、工具、创新”四条原则。其目的，既为学习电子设计自动化(EDA)人员提供一种快速有效的入门工具，一个仿制、借鉴、研究、创新的工作平台，亦为从事 EDA 的技术人员提供一本关于基本数字电路电子设计自动化的检索开发工具书。

本书中的一部分实例是在多年教学、科研的基础上开发整理出来的，另一部分是业内经常使用的 VHDL 程序模块。所有实例都在 MAX + plus Ⅱ 软件平台上进行过编译、通过仿真，用可编程逻辑器件 EPM7128S(或者 FLEX10K)下载、实际测量，因而均可以作为进一步开发的参考。

如果说学习、掌握电子设计自动化技术有捷径的话，那就应该从仿制、理解、实践、创新开始。学习 VHDL 应遵循边学边练的原则，其语法规则只有在实践中才能完全领悟、掌握。

本书的出版得益于电路中心各个时期从事 EDA 教学工作的老师们的教学成果，没有他们的宝贵经验，也就没有本书的顺利出版。本书的出版也得益于广大同学的 EDA 学习与实践，他们为本书的编写提供了丰富的实践经验。本书在准备、编写过程中得到了学校各级领导，教务处、电路中心教师和工程技术人员的热情支持与帮助，在此向他们表示诚挚的谢意，特别向安德宁、徐惠民、赵廷瑞、赵振纲教授，以及肖冰、郭莉、袁宝库、魏学军、姬红强、袁东明等老师表示衷心的感谢！

本书的编写是我们从事 EDA 教学工作的一个阶段总结，真诚的希望广大读者能对书中存在的问题提出宝贵的批评和建议。

作　者

2003 年 1 月

目　录

第一章　VHDL设计基础

第一节　VHDL设计入门

由于本书大部分设计实例均以VHDL语言进行描述，因此在每一个实例的“设计任务”段落中都略去“用VHDL语言进行设计”这一限定文字。

用VHDL描述数字电路有行为描述、数据流描述和结构描述三种模式，其核心模式是行为描述模式，下面分别加以介绍。

实例1-1　行为描述模式

一、设计任务

设计一个有高进位c1、低进位c0的10位二进制全加器电路。

二、算法设计

若使用几个低位加法器组合求解，描述太繁琐，因而考虑用抽象的上层行为描述模式设计该加法器。

三、VHDL源程序

1．文件名：adder1.vhd

2．源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder1 is
port (a,b: in   std_logic_vector(9 downto 0) ;
        co: in   std_logic_vector(9 downto 0);
        c1: out std_logic;
       sum: out std_logic_vector (10 downto 0));
end ;

architecture jg of adder1 is
signal a_temp:std_logic_vector(10 downto 0);
signal b_temp:std_logic_vector(10 downto 0);
signal sum_temp:std_logic_vector(10 downto 0);
  begin
process
  begin
    a_temp <= '0' & a;
```

```
    b_temp <= '0' & b;
    sum_temp <= a_temp + b_temp + co;
    sum <= sum_temp(9 downto 0);
    c1 <= sum_temp(10);
  end process;
  end jg;
```

四、程序说明

1. 用行为描述模式设计加法器,可以降低设计难度。行为描述用于表示输入与输出之间转换的行为,不需要包含任何结构方面的信息。

2. 设计者只需编制出源程序,而挑选电路方案的工作则由计算机系统自动完成。

3. 最终选取的电路方案的优化程度,往往取决于综合软件的技术水平和器件的支持能力。也就是说,最终选取的电路方案占用的 PLD 器件资源不一定是最少的。

4. 设计策略,首先考虑用行为描述模式设计电路,如果设计的结果不能满足资源占有率的要求,则应改变描述模式。

5. 本程序用"和(sum)"的最高位作为高进位,由于规定设计的是 10 位加法器,所以应设置"和(sum)"的位数为 11 位。

实例 1-2　数据流描述模式

一、设计任务

设计一个实现逻辑函数: f = ab + cd 的逻辑电路。

二、算法设计

使用数据流描述模式设计电路。

三、VHDL 源程序

1. 文件名:and_or.vhd

2. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity and_or is
port (a,b,c,d: in std_logic;
            f: out std_logic);
end ;

architecture sjl of and_or is
  begin
process
  begin
  f<=(a and b)or(c and d);
end process;
```

```
end;
```

四、程序说明

1. 用数据流描述模式设计电路与用传统的逻辑方程设计电路很相似，显见，

$$f = ab + cd$$

和

```
f <= (a and b)or(c and d)
```

是很相似的。它们的差别仅在于描述逻辑运算的逻辑符号及表达方式略有不同。数据流描述亦表示行为，但含有结构信息，如进程间的通信等，通常用并行语句进行描述。应当说明的是，有的描述形式究竟属于哪一种模式是难以界定的，但这绝对不会影响对具体描述的应用。

2. 设计中只要有了布尔代数表达式就很容易将它转换为 VHDL 的数据流表达式。转换方法是用 VHDL 中的逻辑运算符号置换布尔逻辑运算符即可。例如，用 or 置换“+”；用“<=”置换“=”。

实例 1-3 结构描述模式

一、设计任务

设计一个实现逻辑函数：f = ab + cd 的逻辑电路。

二、算法设计

使用结构描述模式设计电路。

三、VHDL 源程序

1. 文件名：and_or_gate.vhd

2. 源程序

```
entity and_or_gate is
port (a,b,c,d: in bit ;
              f:out bit);
end;
architecture and_or_gate1 of and_or_gate is
component ym
     port(
     a,b: in bit;
       c: out bit);
end component;
component hm
     port(
     a,b: in bit;
       c: out! bit);
end component;
signal temp1,temp2:bit;
begin
u1: ym
port map (a=> a , b=> b,c=> temp1);
```

```
u2: ym
port map (a => c , b => d,c => temp2 );
u3: hm
port map (temp1,temp2,c => f);
end;
```

四、程序说明

1. 用结构描述模式设计电路的步骤

(1) 调用已有元件,通过 component 语句实现。

(2) 用设计的端口名称替换被调用元件的端口名称,这一步通过例化语句实现。

2. 结构描述模式的特点

(1) 在已有元件的端口之间进行连接,若多个元件的端口被命名为同一名称,则表示这几个端口是并接在一起的。

(2) 结构描述实质是用文字描述电路原理图中各元件的连接关系。

五、被调用元件的 VHDL 源程序

1. 被调用元件 ym 的 VHDL 源程序

```
entity ym is
port (a,b: in bit;
        c:out bit);
end;
architecture yml of ym is
  begin
  c <= a and b;
end;
```

2. 被调用元件 hm 的 VHDL 源程序

```
entity hm is
port (a,b: in bit;
        c: out bit);
end;
architecture hml of hm is
  begin
  c <= a or b;
end;
```

第二节 VHDL 的框架结构

实例 1-4 VHDL 的基本框架结构

一、设计任务

描述一个与门:f = a and b。

二、算法设计

使用数据流模式描述。

三、VHDL源程序

1. 文件名:ym.vhd

2. 源程序

```
entity ym is
port (a,b:in bit;
        c:out bit);
end;
architecture yml of ym is
  begin
  c <= a and b;
end;
```

四、程序说明

VHDL语言的最基本结构是由设计实体部分和设计结构体部分组成。无论是简单的设计还是复杂的设计,这两部分都必须存在。

1. 从entity开始到关键字architecture之前是设计实体部分,主要用于描述器件(无论是复杂还是简单的器件)的外端口(外貌),即输入、输出端口的数量、名称、类型和端口使用的数据类型。

2. 从关键字architecture开始到结束是设计结构体部分,主要用于描述元件内部各个逻辑器件或者功能部件的连接关系。

实例1-5 VHDL的较复杂的框架结构

一、设计任务

描述一个系统,它含有:

(1) 1个3位二进制代符号位的减法器。

(2) 1个分频系数等于6的分频器。

(3) 1个2输入端"与非门"。

(4) 1个只有数据输入端、时钟控制端和输出端的D触发器。

(5) 1个1位二进制数比较器。

二、算法设计

系统分为5个电路部分。为了设计整齐,将电路设计文件分为相互独立的5个部分进行描述,它们是:

(1) 由1个block b1块语句描述3位二进制代符号位的减法器。

(2) 由1个block b2块语句描述分频系数等于6的分频器。

(3) 由1个并发语句描述2输入端"与非门"。

(4) 由1个并发语句process p1描述D触发器。

(5) 由1个并发语句process p2描述1位比较器。

三、VHDL源程序

1. 文件名:jg.vhd

2. 源程序

```
library ieee;          --调用资源库语句。
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity jg is       --开始描述设计实体。
generic(x:integer := 2);        --类属参数说明语句。
     port (din1,din2:in std_logic_vector(0 to x);
          clk,d,j,k:in bit;
        yout, qout,f:out bit;
             bjz, fh:out std_logic;
               dout:out std_logic_vector(0 to x));
end ;

architecture zh of jg is       --开始描述结构体。
  signal temp :std_logic_vector(0 to x) ;       --对结构体内使用的对象或元件作说明。
  signal q ,y :bit;
  signal   r :integer range 0 to 5;
  begin      --开始各子电路描述。
--描述减法器模块开始。
b1:block begin
p:process
  begin
  if din1 > din2 then
  fh <= '0';
  temp <= (din1 - din2 );
  else
  fh <= '1';
  temp <= (din2 - din1 );
  end if;
  dout <= temp;
end process ;
end block b1;
--描述减法器模块结束,描述分频器开始。
b2:block
  begin
process(clk)
  begin
if clk'event and clk = '1' then
  if r = 5 then
```

```
      r<=0;
      y<='1';
    else
      r<=r+1;
      y<='0';
    end if;
  end if;
      yout<=y;
    end process;
  end block b2 ;
  --描述分频器结束。
  --下为描述与非门的并发语句。
  f<=not(j and k);
  p1: process(clk)        --描述 D 触发器的 process p1 进程。
    begin
    if clk'event and clk='1' then
      q<=d;
      qout<=q;
    end if;
  end process p1;
  p2:process        --描述比较器的 process p2 进程。
  begin
  if j>k then
    bjz<='1';
  else
    bjz<='0';
  end if;
  end process p2;
  end;
```

四、程序说明

1. 一个复杂的 VHDL 源文件都需要调用库资源,以减少编程量。同样复杂的 VHDL 源文件也必须有 entity(设计实体)、architecture(结构体)这两个最基本部分。一个 VHDL 源文件容许存在多个结构体,但是在编译时,必须用语句指定一个结构体供软件系统编译使用。使用 MAX + plus Ⅱ 软件对 VHDL 语言进行编译时,该软件是自动选择几何位置排列在最后的结构体作为当前结构体进行编译的,因此,对多结构体的调用语句不再叙述。

结构体是由若干并发语句组成的,而并发语句可由若干顺序语句构成。

2. 块语句、process 语句都是并发语句,软件系统是异步处理各个不同的并发语句的,在物理硬件上,不同的并发语句描述芯片内部不同路径上传递的信号和电路结构。不同的并发语句之间可以通过全局性信号来通信。

块语句、process 语句既可以用来描述一个独立的电路，也可以用来描述一个大电路中的一个子电路。

并发语句内部可以嵌套并发语句，例如在块语句的内部可以嵌入 process 语句，但是，在 process 语句的内部，各语句却是顺序执行的。

3. 本程序在设计实体说明部分引入一条“类属参数说明语句”，语句以关键字 generic 开头。该语句用来定义一个通用参数，使程序的通用性好，修改更方便。

第二章　用 VHDL 设计组合电路

第一节　建立组合电路的方法

实例 2-1　借助真值表设计组合电路

一、设计任务

设计一个四选一选择器。

二、算法设计

借助真值表进行设计。

三、VHDL 源程序

1．文件名：xzq4_1.vhd

2．端口图与真值表

端口图如图 2-1 所示，真值表如表 2-1 所示。

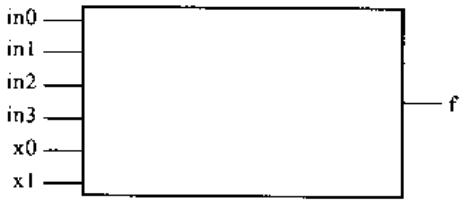


图 2-1　端口图

表 2-1　真值表

in0	in1	in2	in3	x1	x0	f
in0	–	–	–	0	0	in0
–	in1	–	–	0	1	in1
–	–	in2	–	1	0	in2
–	–	–	in3	1	1	in3

3．源程序

```
entity xzq4_1 is
  port(in0,in1,in2,in3:in bit;
                x0,x1:in bit;
                   f2:out bit);
end;

architecture a of xzq4_1 is
begin
  f<=(in0 and ((not x1 )and(not x0))) or
      (in1 and ((not x1 )and     x0)) or
      (in2 and (     x1  and(not x0))) or
      (in3 and (     x1  and     x0));
end;
```

四、程序说明

1. bit 数据类型是 VHDL 语言 IEEE 标准的缺省数据类型,不需要调用程序库(包)语句。

2. end 语句中的实体名 xzq4_1 和结构体名 a 可以省略。

3. 按真值表(可用"与或"结构实现)要求,用 VHDL 语言逻辑表达式方式描述四选一选择器,将 f = '1'的行用 VHDL 最小项表达式表达出来即可。这种描述方法和传统的由真值表变为最小项表达式的设计方法是相同的,只是用 VHDL 语言进行描述无须化简(由计算机进行化简);而用传统设计方法描述时,常常要对最小项表达式进行化简,以使设计电路简化。

4. 源程序中 x0,x1 是选择控制信号。

实例 2-2 用 VHDL 的逻辑表达式设计组合电路

一、设计任务

设计一个函数电路:y = abc + ef 。

二、算法设计

用 VHDL 的逻辑表达式进行描述。

三、VHDL 源程序

1. 文件名:hs.vhd

2. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity hs is
  port(a,b,c,e,f:in std_logic;
                 y:out std_logic);
end;

architecture a of hs is
begin
    y <= (a and b and c)or (e and f)
end;
```

四、程序说明

使用 VHDL 语言的逻辑表达式设计函数电路是很方便的,只要用 VHDL 语言的逻辑符号置换布尔方程中相应的逻辑符号即可。

实例 2-3 用 VHDL 的算数表达式设计组合电路

一、设计任务

设计 1 位全加器电路。

二、算法设计

用 VHDL 的算数表达式进行设计。

三、VHDL 源程序

1. 文件名：full_adder.vhd

2. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;        --为了进行加法运算必须调用该库。

entity full_adder is
  port(a,b,c0:in std_logic;
       sum,c1:out std_logic);
end ;

architecture a of full_adder is
signal sum1: std_logic_vector(1 downto 0);
signal a1,b1,c01: std_logic_vector(1 downto 0);

begin
a1 <= '0' & a;
b1 <= '0' & b;
c01 <= '0' & c0;        --必须先进行并位运算才能进行下式运算。
sum1 <= a1 + b1 + c01;
sum <= sum1(0);         --求和。
c1  <= sum1(1);         --求高进位。
end ;
```

四、程序说明

1. 使用 VHDL 语言的算数运算表达式设计全加器是很方便的。请注意：在 VHDL 描述中，要用并位运算解决求进位值 c1，否则对高进位 c1 的描述就要采用其他较复杂的方法。

2. 在程序中，用并位运算是为了满足代入符“<=”左、右两侧运算对象的位数相同。

3. 一般来说，组合电路可以使用行为描述、数据流描述和结构描述中的一种或者几种描述模式进行描述。设计时，只要描述方便、易于理解、便于记忆，程序经编译、仿真、测试效果好的描述模式就是好的描述模式。

有时，程序代码虽然简单，但是占用器件资源却多；相反，有时，程序代码虽然复杂，但是占用器件资源却少。因此，不能简单的用程序代码简单还是复杂来判断程序代码的优劣，而是要对程序代码进行综合评价，才能得出比较科学的结论。

第二节　描述组合电路的 VHDL 程序实例

实例 2-4　基本逻辑门

一、设计任务

构成基本逻辑门。

二、算法设计

用 VHDL 语言的逻辑标识符(赋值语句)描述。

三、VHDL 源程序

1. 文件名:jbm.vhd

2. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity jbm is
  port( a,b : in bit;
  f1,f2,f3,f4,f5,f : out bit);
end ;

architecture a of jbm is
begin
--以下部分构造基本逻辑门。
      f1 <= a and b;        --构成与门。
      f2 <= a or b;         --构成或门。
      f <= not a;           --构成非门。
      f3 <= a nand b;       --构成与非门。
      f4 <= a nor b;        --构成异或门。
      f5 <= not(a xor b);   --构成异或非门,即同门。
end ;
```

四、程序说明

1. 应注意 VHDL 语言的各种逻辑标识符号的含义。

2. 由基本逻辑门可以实现任意函数功能。

3. 注意 not 逻辑标识符号的正确使用方法。例如:'1' <= not '0';表达式是错误的,这是因为逻辑表达式中的运算对象必须是信号或者变量,不能是常量,而'1'、'0'是常量。

实例 2-5　用 Oc 门实现线与功能

一、设计任务

用一个 Oc 门设计一个实现线与功能的电路,令 d0、d1、d2 为输入信号,f 为输出信号。

二、算法设计

用 if 顺序语句描述。

三、VHDL 源程序

1. 文件名:xian_yu.vhd

2. 端口图

线与电路端口图如图 2-2 所示。

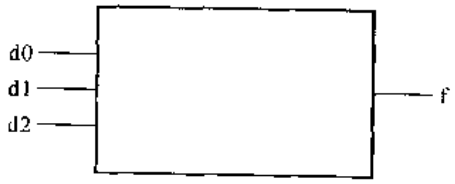


图 2-2　线与电路端口图

3. 源程序

```
library ieee;
```

```
use ieee.std_logic_1164.all;
entity xian_yu is
  port(d0,d1,d2 : in std_logic;
    f: out std_logic);
end ;
architecture a of xian_yu is
begin
  process
  begin
    if ((d0 and d1 and d2) = '1') then
      f<='Z';
    else
      f<='0';
    end if;
  end process;
end a;
```

四、程序说明

1. 文件描述用一个 Oc 门实现信号 d0、d1、d2 的线与功能。

2. 实际应用时，应在输出端接一个上拉电阻，若用 Altera 的 PLD 芯片，推荐使用的电阻值为 1 kΩ。

3. 字符'Z'中的 Z 必须大写。

4. if 顺序语句置于并发语句 process 的内部。if 顺序语句按书写顺序执行。

实例 2-6　带使能端的三态门

一、设计任务

设计一个三态门，令该三态门的输入信号为 data，使能信号为 oe，输出信号为 tri_output。

二、算法设计

用 if 顺序语句描述。

三、VHDL 源程序

1. 文件名：tri.vhd

2. 端口图

带使能端的三态门端口图如图 2-3 所示。

图 2-3　带使能端的三态门端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
entity tri is
  port (oe : in std_logic;
```

```
        data: in std_logic;
 tri_output: out std_logic);
end ;

architecture a of tri is
begin

  process (oe, data)
  begin
    if oe = '0' then
     tri_output <= 'Z';
    else
     tri_output <= data;
    end if;
  end process;
end a;
```

四、程序说明

1. 文件描述一个三态门。

2. 实际应用时，若 output 为高阻，且在输出端接一个上拉电阻，则输出端逻辑值为'1'；如果在输出端接一个下拉电阻，则输出端逻辑值为'0'。若用 Altera 的 PLD 芯片，推荐使用的电阻值为 1 kΩ。

3. 字符'Z'中的 Z 必须大写。

实例 2-7　三选一数据选择器

一、设计任务

描述一个三选一数据选择器电路，令 a、b、c 为输入信号，e1、e2 为选择信号，fout 为输出信号。

二、算法设计

用 if 顺序语句描述。

三、VHDL 源程序

1. 文件名：mux3.vhd

2. 端口图

三选一数据选择器端口图如图 2-4 所示。

图 2-4　三选一数据选择器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
entity mux3 is
  port(a,b,c,e1,e2:in bit;
               fout:out bit);
end ;
```

```
architecture a1 of mux3 is
begin
  process (a,b,c,e1,e2)
  begin
    if (e1 = '1') then
      fout <= a;
    elsif (e2 = '1') then
      fout <= b;
    else
      fout <= c;
    end if;
  end process;
end ;
```

四、程序说明

程序中 e1、e2 为选择信号，当 e1e2 = "10"或者 e1e2 = "11"，a 输出；当 e1e2 = "01"，b 输出；当 e1e2 = "00"，c 输出。

实例 2-8　四选一数据选择器(1)

一、设计任务

描述一个四选一数据选择器电路，令 din 为输入矢量信号，a、b 为选择信号，q 为输出信号。

二、算法设计

用 case 顺序语句描述。

三、VHDL 源程序

1. 文件名：mux4.vhd

2. 端口图

四选一数据选择器(1)端口图如图 2-5 所示。

图 2-5　四选一数据选择器(1)端口图

3. 源程序

```
library ieee;
use ieee. std_logic_1164.all;
entity mux4_case is
  port( a,b: in std_logic;
        din: in std_logic_vector( 0 to 3);
          q: out std_logic);
end ;
architecture a of mux4_case is
signal sel : std_logic_vector( 0 to 1);
```

```
begin
  process
    begin
    sel <= a & b;
    case sel is
      when "00" => q <= din(0);
      when "01" => q <= din(1);
      when "10" => q <= din(2);
      when "11" => q <= din(3);
      when others => q <= 'Z';        -- 见程序说明。
    end case;
  end process ;
end a;
```

四、程序说明

1. std_logic 数据类型,除'0'、'1'值外,还有其他值,用 others 穷尽所有可能的组合值。

2. case 顺序语句置于并发语句 process 的内部。case 顺序语句按书写顺序执行。

3. 本程序中的 din 为输入矢量信号,这是一个 4 位矢量信号。矢量信号在图中用黑粗线表示。

实例 2-9　四选一数据选择器(2)

一、设计任务

描述一个四选一数据选择器电路,令 input 为输入矢量信号,a、b 为选择信号,s 为输出信号。

二、算法设计

用并行条件信号赋值语句及选择信号赋值语句描述。

三、VHDL 源程序

1. 文件名:mux4_2.vhd

2. 端口图

四选一数据选择器(2)端口图如图 2-6 所示。

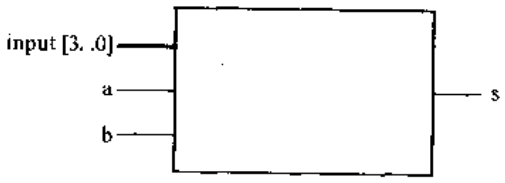


图 2-6　四选一数据选择器(2)端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_2 is
port(input:in std_logic_vector(3 downto 0);
      a,b:in std_logic ;
        s:out std_logic) ;
end ;
```

```
architecture with_ when of mux4_2 is
  signal sel : std_ logic_ vector(1 downto 0);
  begin
    sel <= a & b;
    s <= input(0) when sel = "00" else
         input(1) when sel = "01" else
         input(2) when sel = "10" else
         input(3) ;        - - 见程序说明。
end ;
architecture with_ select of mux4 is
  signal sel : std_ logic_ vector (1 downto 0);
  begin
    sel <= a & b;
    with sel select
      s <= input(0) when "00" ,
           input(1) when "01" ,
           input(2) when "10" ,
           input(3) when "11" ,
           'Z' when others;
end ;
```

四、程序说明

1. 本程序中含有两个结构体 with_ when 和 with_ select，MAX + plus Ⅱ 软件系统自动执行几何位置处于最后的结构体 with_ select。

2. 结构体 with_ when 是用并行条件信号赋值语句描述四选一数据选择器。注意，最后一个输出 input(3)不含 when 子句；在 s 表达式中只有一个分号(;)。

3. 结构体 with_ select 是用并行选择信号赋值语句描述四选一数据选择器。注意，选择信号赋值语句中的选择条件与 case 语句相似，不允许条件重叠和涵盖不全。由于 a、b 的值除了'1'、'0'外还有其他 7 个值，所以要用 when others 代表其他值，以穷尽所有可能值。

4. 由本例可知，VHDL 语言的描述能力很强，同一个设计任务，可以用不同的语句进行描述，这种强描述功能对设计者选择描述方法是很方便的。

5. 本程序中的 input 为输入 4 位矢量信号。

实例 2-10　芯片内两个节点相接

一、设计任务

将芯片的两个节点 a 和 b 相接。

二、算法设计

用信号赋值语句描述。

三、VHDL 源程序

1. 文件名:jdlj.vhd

2. 端口与内部示意图

端口与内部示意如图 2-7 所示。

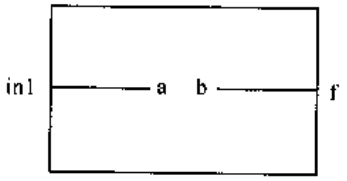


图 2-7 端口与内部示意图

3. 源程序

```
entity jdlj is
  port(in1: in bit ;
       f: out bit);
end ;

architecture a of jdlj is
signal a,b :bit;
begin
    a <= in1 ;
    b <= a ;
    f <= b ;
end ;
```

四、程序说明

1. bit 数据类型是 VHDL 语言 IEEE 标准的缺省数据类型,不需要调用程序库(包)语句。程序中,用三句并行信号赋值语句描述电路节点连接。

2. end 语句中的实体名 jdlj、结构体名 a 可以省略。

3. 用 PLD 器件 EPM7128SLC84-15 实现直通线机构的内部逻辑方程:

$$f = in1 \oplus gnd$$

该逻辑方程由开发软件编译 VHDL 程序后给出,并且已经被软件系统简化。连接线的本质是,输入端 in1 通过一个异或门和输出端 f 相接。

实例 2-11 3 线-8 线译码器

一、设计任务

描述一个 3 线-8 线译码器,使能端为 g1、g2a、g2b,地址选择端为 a、b、c,输出端为总线 y。

二、算法设计

用 case 语句描述电路,利用真值表辅助,很容易编写出程序。

三、VHDL 源程序

1. 文件名:decoder3_8.vhd

2. 端口图

3 线-8 线译码器端口图如图 2-8 所示。

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
```

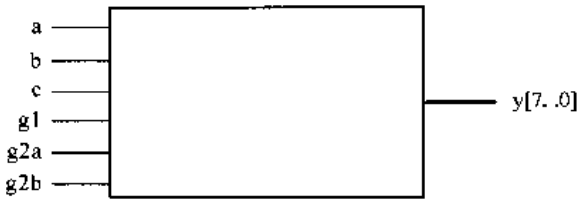


图 2-8　3 线-8 线译码器端口图

```
entity decoder3_8 is
  port (a , b , c,g1,g2a,g2 b:in std_logic;
                          y:out std_logic_vector (7 downto 0));
end ;
architecture rtl of decoder3_8 is
signal dz:std_logic_vector (2 downto 0);
begin
  dz <= c & b & a;
  process (dz,g1,g2a,g2b)
    begin
    if (g1 = '1'and g2a = '0' and g2b = '0') then
      case dz is
        when "000" => y <= "11111110";
        when "001" => y <= "11111101";
        when "010" => y <= "11111011";
        when "011" => y <= "11110111";
        when "100" => y <= "11101111";
        when "101" => y <= "11011111";
        when "110" => y <= "10111111";
        when "111" => y <= "01111111";
        when others  => y <= "XXXXXXXX";
      end case;
    else
      y <= "11111111";
    end if;
  end process;
end;
```

四、程序说明

1. 本程序描述的 3 线-8 线译码器与中小规模集成电路 74LS138 功能相同。

2. 功能仿真图如图 2-9 所示。

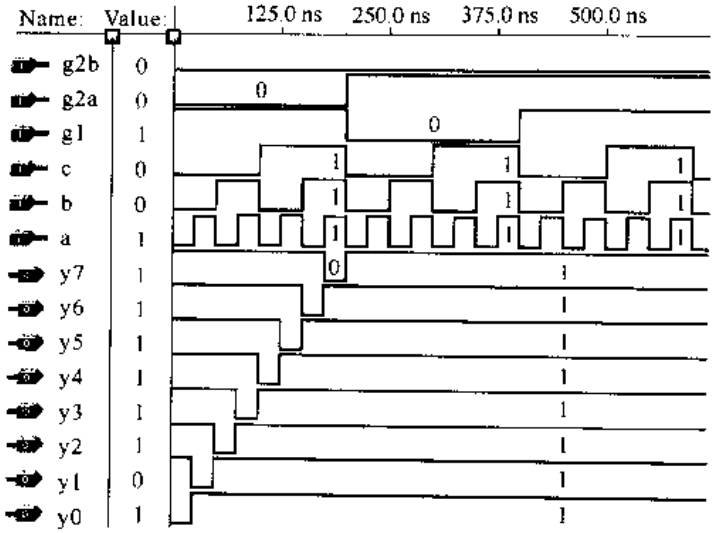


图 2-9　3 线-8 线译码器功能仿真图

实例 2-12　共阴极七段数码显示驱动电路

一、设计任务

描述一个七段数码显示的驱动电路。该电路有 BCD 码输入端 db,两个输入控制端 lt、bi,控制输出端 bio,输出端 segout(7 位)。

二、算法设计

用 if 语句描述电路,利用真值表辅助,很容易编写出程序。

三、VHDL 源程序

1. 文件名:display.vhd

2. 端口图

共阴极七段数码显示驱动电路端口图如图 2-10 所示。

图 2-10　共阴极七段数码显示驱动电路端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity display is
  port(lt : in bit ;
       bi : in bit ;
      bio : out bit ;
       db : in std_logic_vector(3 downto 0) ;
   segout : out std_logic_vector(6 downto 0)) ;

end ;
architecture a of display is
begin
```

```
process
begin
  if lt = '0' and bi = '0' then
    segout <= "0000000";
    bio <= '0';
  elsif lt = '0' and bi = '1' then
    segout <= "1111111";
    bio <= '1';
  elsif lt = '1' and bi = '0' and db = "0000" then
    segout <= "0000000";
    bio <= '0';
  elsif lt = '1' and bi = '1' and db = "0000" then
    segout <= "0111111";
    bio <= '1';
  elsif( lt = '1' and db = "0001" )then
    segout <= "0000110" ;
    bio <= '1';
  elsif( lt = '1' and db = "0010" )then
    segout <= "1011011" ;
    bio <= '1';
  elsif( lt = '1' and db = "0011" )then
    segout <= "1001111" ;
    bio <= '1';
  elsif( lt = '1' and db = "0100" )then
    segout <= "1100110" ;
    bio <= '1';
  elsif( lt = '1' and db = "0101" )then
    segout <= "1101101" ;
    bio <= '1';
  elsif( lt = '1' and db = "0110" )then
    segout <= "1111101" ;
    bio <= '1';
  elsif( lt = '1' and db = "0111" )then
    segout <= "0000111" ;
    bio <= '1';
  elsif( lt = '1' and db = "1000" )then
    segout <= "1111111" ;
    bio <= '1';
  elsif( lt = '1' and db = "1001" )then
```

```
        segout <= "1101111" ;
        bio <= '1';
      elsif( lt = '1' and db > "1001" )then
        segout <= "0000000";
        bio <= '0';
      end if;
    end process;
    end ;
```

四、程序说明

1. 本程序描述的共阴极七段数码显示驱动电路的功能是:

(1) 当 lt = '0' 和 bi = '0' ,则输出 segout <= "0000000",用于控制共阴极七段数码管全灭。

(2) 当 lt = '0' 和 bi = '1' ,则输出 segout <= "1111111",用于控制共阴极七段数码管全亮。

(3) 当 lt = '1' 和 bi = '0' ,若输入的 BCD 码 db = "0000",则输出 segout <= "0000000",用于控制共阴极七段数码管全灭,同时使控制输出信号 bio = '0',该信号可输入到其他共阴极七段数码管驱动电路的 bi 端,作为被控电路的灭零控制信号。若输入的 BCD 码 db 值大于"0000",则输出为非零数的译码,用于显示非零数。

(4) 当 lt = '1' 和 bi = '1' ,则输出信号按正常译码规律变化,支持数码管显示 0 ~ 9 数字。

2. 用于控制数码管的 7 个显示段亮、灭的信号 segout 的最高位用于控制数码管的 g 段,其控制顺序是:g、f、e、d、c、b、a。

实例 2-13 优先编码器

一、设计任务

描述一个优先编码器。该电路有 8 个输入端 d(8 位),3 个输出端 y(3 位)。

二、算法设计

用 if 语句描述电路,利用真值表辅助,很容易编写出程序。

三、VHDL 源程序

1. 文件名:encoder.vhd

2. 端口图与真值表

优先编码器端口图如图 2-11 所示,优先编码器的真值表如表 2-2 所示。

图 2-11 优先编码器端口图

表 2-2 真值表

d7	d6	d5	d4	d3	d2	d1	d0	y2	y1	y0
0	x	x	x	x	x	x	x	0	0	0
1	0	x	x	x	x	x	x	0	0	1
1	1	0	x	x	x	x	x	0	1	0
1	1	1	0	x	x	x	x	0	1	1
1	1	1	1	0	x	x	x	1	0	0

续表

d7	d6	d5	d4	d3	d2	d1	d0	y2	y1	y0
1	1	1	1	1	0	x	x	1	0	1
1	1	1	1	1	1	0	x	1	1	0
1	1	1	1	1	1	1	0	1	1	1

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
entity encoder is
    port(
    d: in std_logic_vector(0 to 7);
    y: out std_logic_vector(0 to 2));
end ;
architecture beh of encoder is
begin
  process(d)
  begin
    if d(7)   = '0' then y <= "000";
    elsif d(6) = '0' then y <= "001";
    elsif d(5) = '0' then y <= "010";
    elsif d(4) = '0' then y <= "011";
    elsif d(3) = '0' then y <= "100";
    elsif d(2) = '0' then y <= "101";
    elsif d(1) = '0' then y <= "110";
    elsif d(0) = '0' then y <= "111";
    end if;
  end process;
end ;
```

四、程序说明

1. 由优先编码器的真值表可知，输入信号 d7 的优先权最高，只要 d7 = '0'，无论其他输入端的值为何值，编码器输出结果都由 d7 = '0'决定。VHDL 语言可用 if 语句描述优先权特性，在 if 语句中最先描述 d7 = '0'这个优先编码条件。优先权级别越低，在语句中描述的顺序就越靠后。

2. 图 2-12 是优先编码器的仿真图。由图得知，在某个时刻，有 d7 = d3 = d1 = '0'，但是，输出结果，却是由 d7 控制。程序规定，在 d7 = '0'的条件下，输出 y 值为"000"。

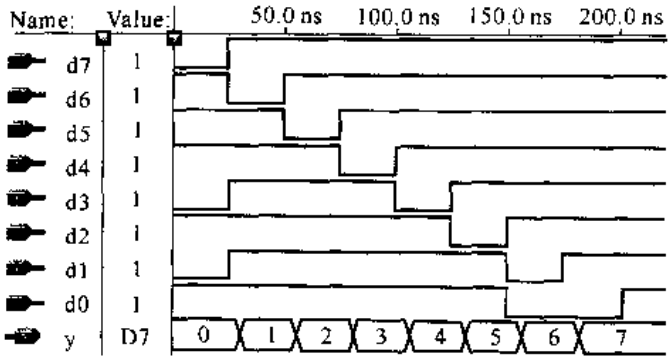


图 2-12 优先编码器的仿真图

实例 2-14 比 较 器

一、设计任务

设计一个含有 generic 的比较器，令 d1(8 位)、d2(8 位)为输入信号，clk 为时钟信号，f1(d1 > d2)、f2(d1 = d2)、f3(d1 < d2)为输出信号。

二、算法设计

由于两个比较信号位数多，为降低设计难度，拟采用行为描述模式描述电路。

三、VHDL 源程序

1. 文件名：display.vhd

2. 端口图

比较器端口图如图 2-13 所示。

图 2-13 比较器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity comp is
generic (x : integer := 8);
  port(d1,d2: in std_logic_vector( 0 to x-1);
    f1,f2,f3: out std_logic );
  end ;

architecture a of comp is
begin
  process (d1,d2)
    begin
      if d1 > d2 then
        f1 <= '1';
      else
        f1 <= '0';
       end if;
```

```
        if d1 = d2 then
          f2 <= '1';
        else
          f2 <= '0';
        end if;
        if d1 < d2 then
          f3 <= '1';
        else
          f3 <= '0';
        end if;
    end process ;
end;
```

四、程序说明

1. 用 generic 语句引入通用参数 x，赋其初值为 8。应用 generic 语句使程序通用性好，修改方便。

2. 由程序可知，用行为描述模式描述一个多位比较器比较简单。需要指出的是，当组合电路结构复杂时，电路容易产生冒险，这时应考虑加入钟控电路，以消除或者预防组合电路产生的冒险。

实例 2-15　二－十进制 BCD 译码器

一、设计任务

设计一个二-十进制 BCD 译码器。译码器输入 din 为 4 位二进制数，输出为以 4 位二进制数表示的两个十进制数 a、b。

二、算法设计

用行为描述模式描述译码器。

三、VHDL 源程序

1. 文件名：decode.vhd

2. 端口图

二-十进制 BCD 译码器端口图如图 2-14 所示。

图 2-14　二-十进制 BCD 译码器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity v2_bcdymq is
  port(din: in integer range 15 downto 0;
       a,b: out integer range 9 downto 0);
```

```
  end ;

architecture fpq1 of v2_bcdymq is
  begin
p1: process
    begin
        if din < 10 then
          a <= din ;
          b <= 0;
        else
          a <= din - 10 ;
          b <= 1 ;
        end if;
end process p1 ;
end ;
```

四、程序说明

1. 依据输入、输出之间的行为关系,用行为描述模式描述二 - 十进制 BCD 译码器。解决这类问题,通常用行为模式描述简单有效。

2. 本程序使用了整数数据类型,定义这种数据类型,要使用关键字"integer range〈数值范围〉"。

实例 2-16 并行奇校验电路

一、设计任务

设计一个 4 位并行奇校验电路,设输出为 p。当各输入信号同时输入的逻辑'1'的个数总和为奇数时,输出 p = 1;否则 p = 0。

二、算法设计

在传统设计中,奇校验电路往往用若干个"异或"门构成。由奇校验电路的真值表,很容易得到设计结果。

三、VHDL 源程序

1. 文件名:parity.vhd

2. 端口图

奇校验电路端口图如图 2-15 所示。

图 2-15 奇校验电路端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity parity is
    port(a0,a1,a2,a3: in bit;
                 p : out bit);
    end;
```

```
architecture a of parity is
begin
  process
   begin
  p <= a0 xor a1 xor a2 xor a3;
  end process ;
end;
```

四、程序说明

1. 比照 p 表达式,很容易扩大并行奇校验电路的输入位数,使电路扩容。

2. 输出信号 p 接到一个反相器的输入端,用该反相器的输出信息即可实现电路的并行偶校验功能。

实例 2-17　补码生成电路

一、设计任务

设计一个 8 位补码生成电路,设输入为 d,输出为 p。

二、算法设计

补码用于减法运算,其作用是把减法运算变成补码的加运算。考虑用行为描述模式描述补码生成电路。

三、VHDL 源程序

1. 文件名:patch.vhd

2. 端口图

补码生成电路端口图如图 2-16 所示。

图 2-16　补码生成电路端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity patch is
  port(d: in std_logic_vector(0 to 7);
       y: out std_logic_vector(0 to 7));
end ;
architecture a of patch is
begin
  process(d)
    begin
    y <= (not d) + 1;
end process;
end ;
```

四、程序说明

1. 按补码定义:补码 = 反码 + 1。本程序用补码定义描述其生成电路。

2. 注意,为了使 MAX + plus Ⅱ 软件接受对电路的描述,程序中必须设立调库命令:

```
use ieee.std_logic_signed.all;
```

实例 2-18 全 减 法 器

一、设计任务

设计一个 4 位全减法器,设被减数为 a,减数为 b,低借位为 c0,输出差为 d,高借位为 c1。

二、算法设计

考虑用行为描述模式进行描述。

三、VHDL 源程序

1. 文件名:jfq.vhd

2. 端口图

全减法器端口图如图 2-17 所示。

图 2-17 全减法器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity jianfaqi is
  port(a,b: in std_logic_vector(0 to 3);
       c0: in std_logic;
       c1: out std_logic;
        d: out std_logic_vector(0 to 3));
end ;
architecture a of jianfaqi is
begin
  process
    begin
    if a >= b + c0 then
      d <= a - (b + c0);
      c1 <= '0';
    else
      c1 <= '1';
      d <= ("10000" ) - (b + c0 - a);
    end if;
    end process;
end ;
```

四、程序说明

1. 用行为描述模式描述全减法器。两个数够减,高借位值 c1 为'0';两个数不够减,需要向高位借 1 位,这时高借位值 c1 为'1',由于有借位,所以差值是大于或等于零的数。

2. 注意,为了使 MAX + plus Ⅱ 软件接受对电路的描述,程序中必须设立调库命令:

use ieee.std_logic_signed.all;

实例 2-19　移　位　器

一、设计任务

设计一个 4 位组合移位器，设输入为 d(4 位)，输出为 f(4 位)，控制移位为 kz(2 位)。输出 f 依 kz 值条件产生位移。

二、算法设计

考虑用行为描述模式进行描述。

图 2-18　移位器端口图

三、VHDL 源程序

1. 文件名：shifter.vhd

2. 端口图

移位器端口图如图 2-18 所示。

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity shifter is
  port(kz: in std_logic_vector(1 downto 0);
       d: in std_logic_vector(3 downto 0);
       f: out std_logic_vector(3 downto 0));
end ;
architecture a of shifter is
signal y : std_logic_vector(3 downto 0);
begin
  process
    begin
    if kz = "00" then
    y(3 downto 0) <= d(3 downto 0);      --输入按对应位传给输出。
    elsif kz = "01" then
    y(3 downto 1) <= d(2 downto 0);      --输入传给输出的对应位发生改变。
    y(0) <= d(3);
    elsif kz = "10" then
    y(3 downto 2) <= d(1 downto 0);      --输入传给输出的对应位发生改变。
    y(1) <= d(3);
    y(0) <= d(2);
    elsif kz = "11" then
    y(3) <= d(0);      --输入传给输出的对应位发生改变。
    y(2) <= d(3);
    y(1) <= d(2);
```

```
        y(0) <= d(1);
        end if;
        end process;
        f <= y;
    end ;
```

四、程序说明

1. kz = "00"时,输入信号赋给输出信号。kz 值每变化一次,输出信号的值就变化一次。移位的本质是输入信号重新给输出信号赋值,只是赋值的位值不同。

2. 移位器若用时序电路实现,则被称为移位寄存器。

实例 2-20　只读存储器

一、设计任务

设计一个 8×8,输出 8 位,内存 8 组固定数据,地址信号为 rom_addr(3 位),使能信号为 ce 的只读存储器(ROM)。

二、算法设计

考虑用行为描述模式进行描述。

三、VHDL 源程序

1. 文件名:rom.vhd

2. 端口图

ROM 端口图如图 2-19 所示。

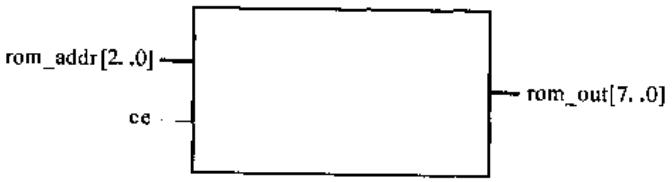


图 2-19　ROM 端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity rom is
  port(rom_out: out std_logic_vector(7 downto 0);
       rom_addr: in std_logic_vector(2 downto 0);
            ce: in std_logic);
end ;

architecture a of rom is
begin
  rom_out <="10000001" when rom_addr = "000" and ce = '0' else
            "10000010" when rom_addr = "001" and ce = '0' else
            "10000011" when rom_addr = "010" and ce = '0' else
            "10000100" when rom_addr = "011" and ce = '0' else
            "10000101" when rom_addr = "100" and ce = '0' else
            "10000110" when rom_addr = "101" and ce = '0' else
            "10000111" when rom_addr = "110" and ce = '0' else
            "10001000" when rom_addr = "111" and ce = '0' else
```

"ZZZZZZZZ";

end a;

四、程序说明

1. 本程序用条件信号赋值语句进行描述。ROM 中的数据通过条件赋值设置。ROM 中数据的读出条件是 ce 有效,同时数据地址有效。

2. 本程序设计的 ROM,实际上是一个带有使能的 3 线-8 线译码电路(与常规 3 线-8 线译码器不同)。

3. 设计的电路在工作频率较高时,会出现冒险,应当注意克服。有冒险的电路仿真图如图 7-20 所示。

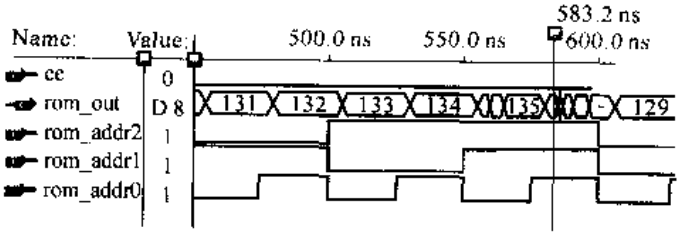


图 2-20　工作频率 f = 20 MHz 时电路出现冒险

4. 电路正常工作时的仿真图如图 2-21 所示。

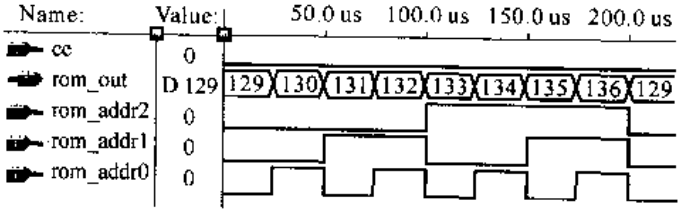


图 2-21　电路的仿真图

实例 2-21　并行奇校验发生器

一、设计任务

有一些数字电路的结构具有重复性,且重复部件较多,如果每一部分都要描述,工作量太大。利用 VHDL 语言的"复制功能"将设计中的重复劳动得以简化,可以提高工作效率。

本例具体任务是,设计一个具有重复结构的 4 位并行奇校验发生器。

二、算法设计

用循环语句进行描述。

三、VHDL 源程序

1. 文件名:parity_loop1.vhd; parity_loop2.vhd

2. 端口图

4 位并行奇校验发生器端口图如图 2-22 所示。

源程序 1 描述的 4 位并行奇校验发生器逻辑电路图如图 2-23 所示,用于编制 VHDL 程序。

图 2-22 4 位并行奇校验发生器端口图

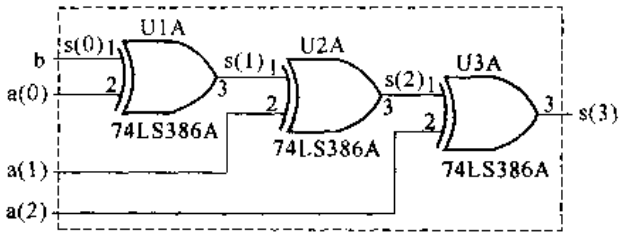


图 2-23 4 位并行奇校验发生器逻辑电路图

3. 源程序

源程序 1

```
library ieee;
use ieee.std_logic_1164.all;

entity parity_loop1 is
  port(a: in std_logic_vector(0 to 2);
       b: in std_logic;
       y: out std_logic);
end ;
architecture a of parity_loop1 is
signal s :std_logic_vector(0 to 3);

begin
  process(a)
    begin
        s(0) <= b ;
      for i in 0 to 2 loop
        s(i+1) <= s(i) xor a(i);
      end loop;
    y <= s(3);
  end process;
end a;
```

源程序 2

```
library ieee;
use ieee.std_logic_1164.all;

entity parity_loop2 is
  port(a: in std_logic_vector(0 to 3);
       y: out std_logic);
end ;
```

```
architecture a of parity_loop2 is
begin
  process(a)
    variable ls : std_logic ;
    begin
        ls := '0';
    for i in 0 to 3 loop
        ls := ls xor a(i);
    end loop;
    y <= ls ;
end process;
end ;
```

四、程序说明

1. 源程序 1 使用循环语句 for … loop 对电路进行描述。数据对象使用信号类型。图 2-23 的虚线框表示芯片，虚线框内的逻辑图表示要设计的电路。图中的每一个异或门都可以表示为：s(i + 1) <= s(i) xor a(i)，只要规定变量 i 的变化范围即可。由逻辑电路图得知，i 的范围是0 ~ 2，由于输出规定为 s(3)，所以在结构体中定义信号 s 为 4 位矢量。为了使 s(0)有驱动源，设定输入信号中的 b 作为 s(0)的驱动源，这样 s(0)就可以运用在循环语句中了。注意，s 只能在结构体内定义，而不能在设计实体中定义。若需设计扩容，只需对循环语句 for … loop 中的上限进行修改即可。例如，要设计 10 位并行奇校验发生器，相应修改的程序段如下(注意黑斜体字部分)：

```
a: in std_logic_vector(0 to 9);
……
signal s :std_logic_vector(0 to 10);
……
for i in 0 to 9 loop
```

2. 源程序 2 亦使用循环语句 for … loop 对电路进行描述。其特点是使用变量数据对象。设计程序的关键一步是给变量 ls 赋予初值，以便使用“ ls := ls xor a(i)”来描述电路。因为在 i =0 时，只有一个输入信号接到异或门上，为了使通式成立，必须令 ls = 0，这样就能保证每个异或门均接入两个信号。设计扩容方法请参照并使用程序说明 1 所述方法。源程序 2 电路端口图与源程序 1 电路端口图相类似，不再另行给出。

3. 图 2-24、图 2-25 是源程序 2 的仿真波形图。下面结合这两个仿真图对仿真信号的工作频率选择作简要说明。在程序仿真时，如果仿真信号工作频率超出器件容许的范围，电路逻辑功能就有可能遭到破坏。在图 2-24 中由于仿真信号的最高频率为 100 MHz，致使奇校验功能丧失(奇校验功能是：若有奇数个“1”同时输入奇校验器，输出应为“1”，而在图 2-24 中，输出却为“0”)。为了克服这个缺点，在仿真时使用的测量信号频率应在器件支持的范围内。图 2-25 中的仿真信号频率符合规定，所以仿真图显示的电路逻辑功能正确。

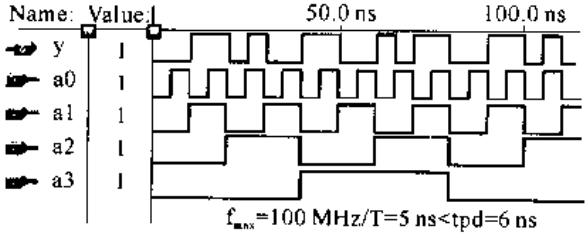


图 2-24　仿真图显示奇校验功能不正确

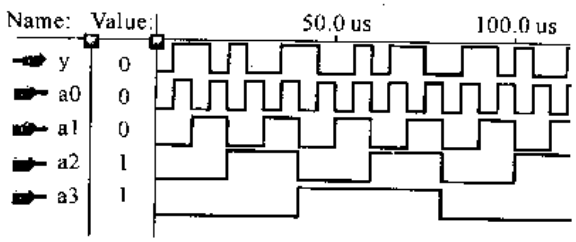


图 2-25　仿真图显示奇校验功能正确

第三章　用 VHDL 设计时序电路

第一节　时序电路的基本要素

触发器、时钟信号是构成时序电路的基本要素。本节将学习对这两个基本要素的 VHDL 描述。触发器有时钟控制型与非时钟控制型之分，本节主要介绍时钟控制型触发器。

实例 3-1　对触发器及时钟信号的 VHDL 描述(1)

一、设计任务

设计一个 D 触发器。

二、算法设计

用 IF 语句进行描述。

三、VHDL 源程序

1. 文件名：dff1.vhd

2. 端口图

D 触发器端口图如图 3-1 所示。

图 3-1　D 触发器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity dff1 is
  port( d,clk : in std_logic;
        q : out std_logic );
end ;
architecture a of dff1 is
signal q_temp: std_logic;
begin
p1:process(clk)
begin
  if clk'event and clk = '1' then
    q_temp <= d;
  end if ;
    q <= q_temp;
end process ;
end;
```

四、源程序说明

1. “if clk'event and clk = '1' then”语句是对触发时钟进行描述时，用得最多的一种语句。if

语句中,用信号属性 event 来判断时钟触发事件是否发生,用 clk = '1'规定触发时钟的上沿为有效触发沿。语句"if clk'event and clk = '1' then"的含义是:若有触发时钟信号产生且当触发时钟的上沿出现时,容许触发器读入输入数据。如果上述事件没有发生,则不容许触发器读入输入数据,而读入输入数据的规则由后续语句规定。

2. 若规定触发器是下沿触发,则触发时钟描述语句为"if clk'event and clk = '0' then"。

3. 图 3-2 是 D 触发器的仿真波形图。由图可以看出,触发器的默认开机状态值为'0',在第一个触发脉冲的上沿出现时刻,电路系统把数据'1'写入触发器。在第二个触发脉冲的上沿出现时刻,电路系统仍把数据'1'写入触发器,但由于触发器原状态为'1',写入数据仍为'1',所以触发器保持'1'状态不变。在第三个触发脉冲的上沿出现时刻,输入数据变为'0',电路系统把数据'0'写入触发器。以后尽管有触发脉冲的上沿出现,但是输入值为'0'不变,所以触发器保持'0'状态。

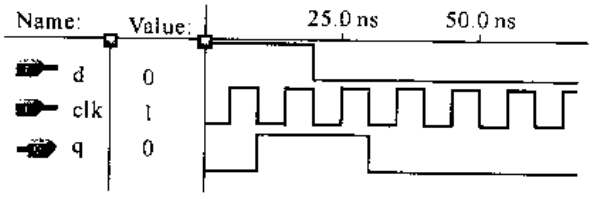


图 3-2 D 触发器的仿真波形图

4. 进行触发器设计时,应当使用触发器的直接输出 q_temp 进行设计,而不要用芯片的输出 q 进行设计。另外设置一条信号赋值语句把触发器输出信号传给芯片输出 q。

实例 3-2 对触发器及时钟信号的 VHDL 描述(2)

一、设计任务

设计一个 T 触发器。

二、算法设计

T 触发器的状态方程为:$q = t\bar{q} + \bar{t}q$。用 if 语句对其进行描述。

三、VHDL 源程序

1. 文件名:tff1.vhd

2. 端口图

T 触发器端口图如图 3-3 所示。

图 3-3 T 触发器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity tff1 is
  port(t,clk:in std_logic;
       f:out std_logic);
end ;
```

```
architecture a of tff1 is
signal q_temp :std_logic;
begin
process(clk)
begin
if falling_edge(clk) then
  if t = '1' then
    q_temp <= not q_temp;
  else
    q_temp <= q_temp;
  end if;
end if ;
end process;
q <= q_temp ;
end ;
```

四、源程序说明

1. "if falling_edge(clk) then"语句是对触发时钟进行描述的一种语句,但不太常用。clk 使用的数据类型应是标准逻辑类型。"falling"表示触发器是下沿触发。

2. 若规定触发器是上沿触发,则触发时钟描述语句为"if rising_edge(clk) then"。

3. 图 3-4 是 T 触发器的仿真波形图。由图可以看出,触发器的默认开机状态值为'0',在第一个触发脉冲的下沿出现时刻,触发器状态由'0'变为'1'。在第二个触发脉冲的下沿出现时刻,触发器状态由'1'变为'0'……在第五个触发脉冲的下沿出现时刻,由于现态为'0',触发器经触发后,次态变为'1'。之后,由于 t = '0', 触发脉冲触发失效。以后尽管有触发脉冲的下沿出现,但是输出状态保持'1'不变。t = '1'时,T 触发器具有 2 分频功能。

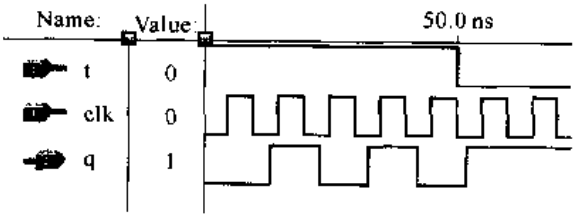


图 3-4　T 触发器的仿真波形图

实例 3-3　对触发器及时钟信号的 VHDL 描述(3)

一、设计任务

设计一个 JK 触发器。

二、算法设计

JK 触发器的状态方程是:$q\_temp <= j\overline{q\_temp} + \overline{k}q\_temp$。用 VHDL 逻辑表达式对 JK 触发器进行描述。

三、VHDL 源程序

1. 文件名:jkff1.vhd

2. 端口图

JK 触发器端口图如图 3-5 所示。

图 3-5 JK 触发器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity jkff1 is
  port( j,k,clk: in std_logic ;
        q: out std_logic);
end ;
architecture a of jkff1 is
signal q_temp: std_logic ;
begin
p1:process(clk)
begin
wait until( clk'event and clk = '1') ;
  q_temp <= (j and(not q_temp))or((not k)and q_temp) ;
end process ;
  q <= q_temp;
end;
```

四、源程序说明

1. 用 wait 语句引入 JK 触发器。程序执行到 wait 语句,等到有一个时钟上沿后,执行程序规定的操作:

```
q_temp <= (j and(not q_temp))or((not k)and q_temp);
```

2. 注意:wait 语句必须放在进程语句的首部和尾部,在本程序中 wait 语句放在进程语句的首部。

3. 信号赋值语句"q <= q_temp"应放到进程之外,这样可使"q <= q_temp"描述用组合电路实现,否则"q <= q_temp"描述会被综合成时序电路。

实例 3-4 对触发器及时钟信号的 VHDL 描述(4)

一、设计任务

设计一个 RS 触发器。

二、算法设计

RS 触发器的状态方程是:$q=\overline{(s\ (\overline{rq})}$。用 VHDL 逻辑表达式对 RS 触发器进行描述。

三、VHDL 源程序

1. 文件名:rsff1.vhd

2. 端口图

RS 触发器端口图如图 3-6 所示。

图 3-6　RS 触发器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity rsff1 is
  port( r,s,clk: in std_logic;
        q: out std_logic );
end ;
architecture a of rsff1 is
signal q_temp: std_logic;
begin
p1:process(clk)
begin
wait until( clk'event and clk = '1');
    q_temp = not(s and (not(r and q_temp))) ;
end process ;
    q <= q_temp;
end;
```

四、源程序说明

1. 用 wait 语句引入 RS 触发器。程序执行到 wait 语句，等到有一个时钟上沿后，执行程序规定的操作：

q_temp = not(s and (not(r and q_temp)))

2. 注意：wait 语句必须放在进程语句的首部和尾部，在本程序中 wait 语句放在进程语句的首部。为保证电路可靠工作，不允许 r、s 同时为 0。

3. 信号赋值语句“q <= q_temp”应放到进程之外，这样可使“q <= q_temp”描述用组合电路实现，否则“q <= q_temp”描述会被综合成时序电路。

实例 3-5　对触发器及时钟信号的 VHDL 描述(5)

一、设计任务

引入 ALTERA 库中模块 dffe。

二、算法设计

用 VHDL 库命令调入模块 dffe。

三、VHDL 源程序

1. 文件名：dff2.vhd

2. 端口图

dff2 触发器的端口图如图 3-7 所示。

图 3-7　dff2 触发器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
library altera;
use altera.maxplus2.all;

entity dff2 is
  port(d,clk,clrn,prn,ena: in std_logic;
                    q: out std_logic );
end ;
architecture a of dff2 is
begin
u: dffe
   port map(d => d,clk => clk,clrn => clrn,prn => prn,ena => ena,q => q);
end ;
```

四、源程序说明

1. 用 VHDL 库命令"library altera;use altera.max plus2.all;"语句引入 dffe 模块。
2. 注意:程序中需设置元件例化语句。

实例 3-6　对触发器及时钟信号的 VHDL 描述(6)

一、设计任务

设计一个具有同步输入数据端 d、时钟端 clk、异步清零端 clrn、异步置位端 prn、触发使能端 ena 和输出端 q 的 D 触发器。

二、算法设计

用 if 语句描述 D 触发器。

三、VHDL 源程序

1. 文件名:dffe2.vhd
2. 端口图

dffe2 触发器的端口图如图 3-8 所示。

图 3-8　dffe2 触发器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity dffe2 is
  port(d,clk,clrn,prn,ena: in std_logic;
                    q: out std_logic );
end ;

architecture a of dffe2 is
signal q_temp:std_logic;
begin
```

```
p1:process(clk)
    begin
if     (clrn = '0' and prn = '1') then
        q_temp <= '0' ;        --异步清零。
elsif  (clrn = '1' and prn = '0') then
        q_temp <= '1' ;        --异步置位。
elsif  (clrn = '0' and prn = '0') then
        null ;          --异步保持。
elsif clk'event and clk = '1' then
  if ena = '1' then
     q_temp <= d ;        --触发使能,同步输入数据。
  else
     q_temp <= q_temp;        --触发不使能,触发器状态保持。
  end if;
end if ;
     q <= q_temp ;
end process;
end a ;
```

四、源程序说明

1. 本例描述的 D 触发器，容许 clrn = '0'和 prn = '0' 同时存在，以实现异步保持功能。有的 D 触发器不容许 clrn = '0'和 prn = '0' 同时存在，否则输出状态不确定。

2. 本例描述涉及“优先权操作”描述，用顺序语句描述“优先权操作”是很方便的，优先权高的操作先描述，优先权低的操作后描述。本例异步清零操作优先权最高，所以最先描述。

3. 本例描述的 D 触发器的仿真波形图如图 3-9 所示。由图可见，开机时输出 q 的状态为'0'，之后，由于 clrn = '0'， prn = '0'，电路实现异步保持功能，输出 q 保持'0'状态。之后，由于 clrn = '0' 和 prn = '1'，电路实现异步清零功能，输出 q 仍为'0'状态。之后，由于 clrn = '1'和 prn = '0'，电路实现异步置位功能，输出 q 由'0'状态改变为'1'状态。之后，由于 clrn = '1'， prn = '1'，ena = '1'，在时钟有效沿出现时，电路实现同步输入数据功能 q = d，因 d = '1'，所以输出 q 仍为'1'状态。之后，clrn = '1'， prn = '1'，ena = '0'，电路呈现触发不使能，尽管此时 d = '0'，但输出

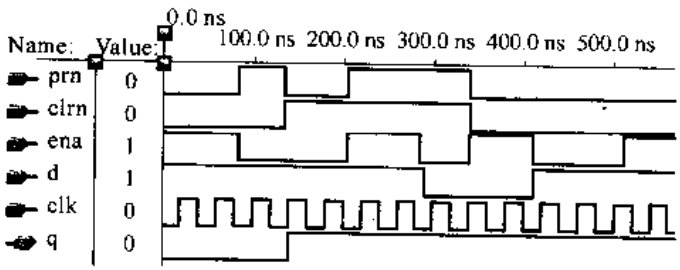


图 3-9　D 触发器的仿真波形图

q 仍保持'1'状态。之后，由于 clrn = '0'，prn = '0'，电路又进入异步保持功能，输出 q 仍保持'1'状态。

实例 3-7 锁 存 器

一、设计任务

设计一个锁存器。

二、算法设计

用条件涵盖不完整的 if 语句实现锁存器。

三、VHDL 源程序

1. 文件名：latch.vhd

2. 端口图

锁存器的端口图如图 3-10 所示。

图 3-10 锁存器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity latch is
   port( kz,din: in std_logic;
              q: out std_logic );
end ;
architecture a of latch is
signal q_temp: std_logic;
begin
p1:process(kz,din)
begin
   if kz = '1' then
      q_temp <= din;
   end if ;
      q <= q_temp;
end process ;
end;
```

四、源程序说明

1. 用条件涵盖不完整的 if 语句可以实现锁存器。这个结果提示我们在用 if 条件语句进行设计时，要明确设计的任务是什么，若设计的不是锁存器，则必须使条件涵盖完整，以避免错误引入锁存器。

2. 若本例描述的锁存器中的控制信号 kz 是一个异步信号，而且输入信号较高，就有可能在输出形成冒险干扰，应注意克服。

3. 图 3-11 是锁存器正常工作时的仿真波形图。图 3-12 是锁存器有毛刺干扰时的仿真波形图。

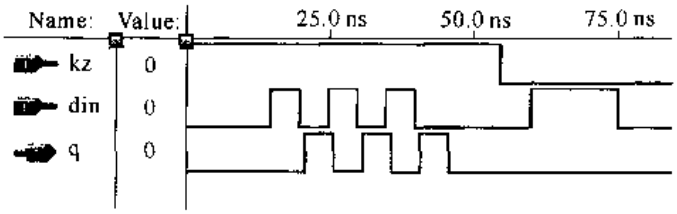


图 3-11　锁存器正常工作时的仿真波形图

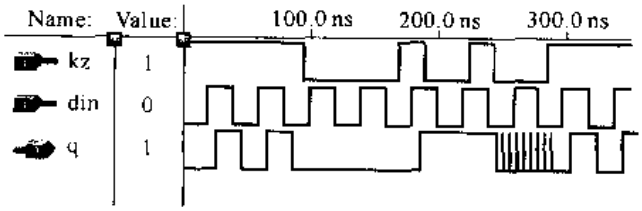


图 3-12　锁存器有毛刺干扰时的仿真波形图

第二节　移　存　器

实例 3-8　串入串出移存器(1)

一、设计任务

设计一个 4 位串入串出移存器。

二、算法设计

用 if 语句描述 4 位串入串出移存器。

三、VHDL 源程序

1. 文件名：shift4_1.vhd

2. 端口图

4 位串入串出移存器端口图如图 3-13 所示。

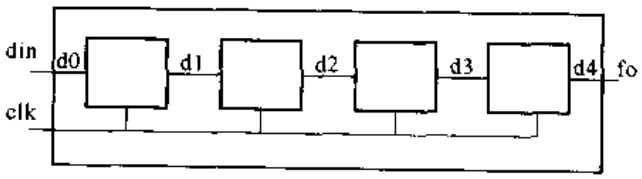


图 3-13　4 位串入串出移存器端口图(1)

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity shift4_1 is
  port (din,clk:in std_logic;
        fo:out std_logic);
```

```
end ;

architecture a of shift4_1 is
signal d0,d1,d2,d3,d4:std_logic;
begin
  process (clk)
    begin
      if (clk'event and clk = '1') then
        d0 <= din;
        d1 <= d0;
        d2 <= d1;
        d3 <= d2;
        d4 <= d3;
        fo <= d4;
      end if;
  end process;
end ;
```

四、源程序说明

1. 一般多位移存器可以用多个 D 触发器串联组成。串行数据先写入移存器的第一级,以后存入移存器的数据沿各级移存器进行移位运动,最终由最后一级输出端口输出。

2. 移存器的关键描述语句是移位操作。因为我们使用的 MAX + plus Ⅱ 开发软件不支持直接用移位操作符对移位进行描述,故改用其他方法进行描述。推而广之,当使用的某种 VHDL 开发软件不支持 IEEE 支持的运算时,应当先改变描述方法,以适应开发软件的特殊要求,以便能通过该软件的编译、仿真,而不必先考虑替换 VHDL 开发软件。

实例 3-9 串入串出移存器(2)

一、设计任务

设计一个 4 位串入串出移存器。

二、算法设计

用 if 语句及数组赋值语句描述 4 位串入串出移存器。

三、VHDL 源程序

1. 文件名:shift4_2.vhd

2. 端口图

4 位串入串出移存器端口图如图 3-14 所示。

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity shift4_2 is
  port (din,clk:in std_logic;
```

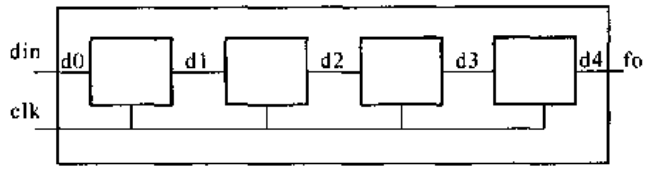


图 3-14　4 位串入串出移存器端口图(2)

```
        fo :out std_logic);
end ;

architecture a of shift4_2 is
signal d:std_logic_vector(4 downto 0);
begin
  process (clk)
    begin
if (clk'event and clk = '1') then
  d(0) <= din;
  d(4 downto 1) <= d(3 downto 0);        --数组赋值。
  fo <= d(4);
end if;
end process;
end ;
```

四、源程序说明

本例采用 if 语句及数组赋值描述多位移存器的移位操作,这样可使描述简单一些。

实例 3-10　串入串出移存器(3)

一、设计任务

设计一个 4 位串入串出移存器。

二、算法设计

用 if 语句及循环语句描述 4 位串入串出移存器。

三、VHDL 源程序

1. 文件名:shift4_3.vhd

2. 端口图

4 位串入串出移存器端口图如图 3-15 所示。

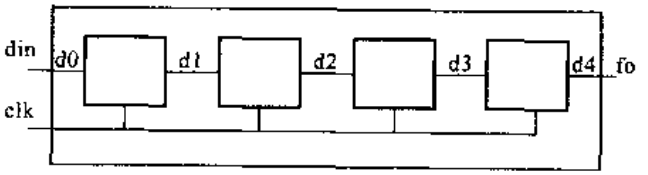


图 3-15　4 位串入串出移存器端口图(3)

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
entity shift4_3 is
  port (din,clk:in std_logic;
        fo:out std_logic);
end ;

architecture a of shift4_3 is
signal d:std_logic_vector(4 downto 0);
begin
  process (clk)
    begin
if (clk'event and clk = '1') then
  d(0) <= din;
loop1:
for i in 0 to 3 loop
  d(i + 1) <= d(i);
end loop;
  fo <= d(4);
end if;
end process;
end ;
```

四、源程序说明

本例采用 if 语句及循环语句描述多位移存器的移位操作,这样可使描述简单一些。

实例 3-11 串入串出双向移存器

一、设计任务

设计一个 4 位串入串出双向移存器。

二、算法设计

用 if 语句及循环语句描述 4 位串入串出双向移存器。

三、VHDL 源程序

1. 文件名:shift4_4.vhd

2. 端口图

4 位串入串出双向移存器端口图如图 3-16 所示。

图 3-16 4 位串入串出双向移存器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity shift4_4 is
```

```
    port (dr,dl,clk:in std_logic;
                kz:in bit;
              fr,fl:out std_logic);
end ;
architecture a of shift4_4 is
signal q:std_logic_vector(3 downto 0);
begin
   process (clk)
      begin
if (clk'event and clk = '1') then
   if kz = '1' then
      q(0) <= dr;
      loop1:
      for i in 0 to 2 loop
      q(i + 1) <= q(i);
      end loop;
   else
      q(3) <= dl;
      loop2:
      for i in 1 to 3 loop
      q(i - 1) <= q(i);
      end loop ;
   end if;
end if;
      fr <= q(3);
      fl <= q(0);
end process;
end ;
```

四、源程序说明

1. 当 kz = '1'时，移存器执行右移功能，右移输入信号为 dr。当 kz = '0'时，移存器执行左移功能，左移输入信号为 dl。

2. 本例采用 if 语句及两个循环语句描述多位移存器的双向移位操作，这样可使描述简单一些。

实例 3-12　串入并出移存器

一、设计任务

设计一个 5 位串入并出移存器。

二、算法设计

用 if 语句描述 5 位串入并出移存器。

三、VHDL 源程序

1. 文件名:shift4_5.vhd

2. 端口图

5 位串入并出移存器端口图如图 3-17 所示。

图 3-17 5 位串入并出移存器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity shift4_5 is
  port(clk,shift,cr: in std_logic;
                y:out std_logic_vector(4 downto 0) );
end ;

architecture a of shift4_5 is
signal temp_date:std_logic_vector(5 downto 0);
begin
  process(clk)
    begin
      if (clk'event and clk = '1') then
        if (cr = '0') then
          temp_date <= (others => '0') ;        --表示各位均赋'0'。
        elsif temp_date(0) = '0' then
          temp_date <= shift & "01111";
        else
          temp_date <= shift & temp_date(5 downto 1) ;
        end if;
      end if ;
  end process ;

  process
    begin
      if temp_date(0) = '0' then
        y <= temp_date(5 downto 1);
      else
        y <= "00000";
      end if;
  end process ;
end ;
```

四、源程序说明

1. 在程序中,cr 为清零信号,shift 为串行输入信号,clk 为移存时钟,y 为 5 位并出信号。

2. 串行输入信号每 5 位为一组数据,该信号一位一位的按顺序移入移存器。由于数据为

5 位，设置 6 个寄存器构成串入并出移存器，其中 5 个寄存器用于移位、寄存串入的数据，另一个作为标志位用于记录 5 位数据是否全部移进移存器，一旦移存器侦测到 5 位数据全部进入，则使 5 位数据立即并行输出。而串行输入数据在移进移存器的过程中，使移存器的并行输出信号保持为特定值。（本例特定值为“00000”，亦可选高阻输出。）

3. 本程序设计的关键步骤是设置标志位 temp_date(0)及设置标志信号“01111”，在并行输出状态的下一状态执行：“temp_date <= shift & "01111";”操作。串行输入数据在移进移存器的几个状态期间，标志位始终为'1'状态，当 5 位数据全部进入移存器后，标志位改变为'0'状态，即电路系统自动侦测到一组串行输入数据已经全部移入移存器。

4. 5 位串入并出移存器的仿真波形图如图 3-18 所示。在图中串入数据为"11010"，并出数据亦为"11010"。

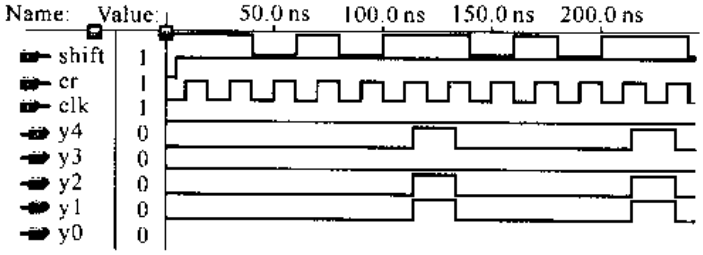


图 3-18　5 位串入并出移存器的仿真波形图

实例 3-13　并入串出移存器

一、设计任务

设计一个 4 位并入串出移存器。

二、算法设计

用 if 语句描述 4 位并入串出移存器。

三、VHDL 源程序

1. 文件名：shift4_6.vhd

2. 端口图

4 位并入串出移存器端口图如图 3-19 所示。

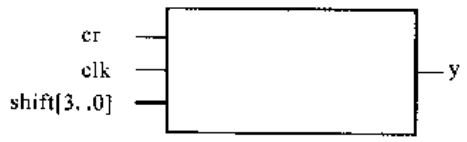


图 3-19　4 位并入串出移存器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity shift4_6 is
  port(clk,cr:in std_logic;
       shift:in std_logic_vector(3 downto 0);
         y:out std_logic);
end ;

architecture a of shift4_6 is
```

```
signal k:std_logic_vector(3 downto 0);
signal q:std_logic_vector(1 downto 0);

begin
p1:process(clk)
   begin
      if (clk'event and clk = '1') then
         q <= q + 1;
      end if ;
   end process;

p2:process(clk)
begin
  if cr = '0' then       --在 cr = '0'条件下,若无操作规定,则 K 维持初始值"0000"。
  elsif (clk'event and clk = '1') then
    if q > "00" then
    k(3 downto 1) <= k(2 downto 0);
    elsif q = "00" then
    k <= shift;
    end if;
  end if ;
    y <= k(3);
end process ;
end ;
```

四、源程序说明

1. 在程序中 cr 为清零信号,shift 为并行输入信号,clk 为移存时钟,y 为一位串出信号。

2. 程序中设置一个模值等于 4 的控制计数器,用其 4 个状态实现并入串出控制。控制计数器 q 处于"00"状态时,4 位并入数据写入移存器,同时,串出 1 位数据;处于状态"01"、"10"、"11"时,并入数据移 1 位,其余 3 位数据串行移出。

3. 4 位并入串出移存器的仿真波形图如图 3-20 所示。图中显示:输入的并行数据为"1110",该数据从左到右一位一位地移出。利用本并入串出移存器构成波形发生器是很方便的。

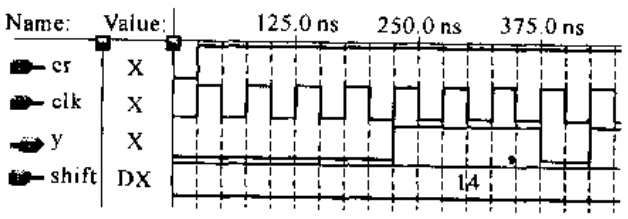


图 3-20 4 位并入串出移存器的仿真波形图

第三节　计　数　器

实例 3-14　一热态位编码计数器

一、设计任务

设计一个一热态位(one-hot)编码的四进计数器。该计数器类型也可称为“循环出 1(或 0)计数器”或“循环计数器”。其特点是对每一个计数状态采用一个触发器。

二、算法设计

用 if 语句描述一热态位编码四进计数器。

三、VHDL 源程序

1. 文件名:hot_count.vhd

2. 端口图

一热态位编码四进计数器端口图如图 3-21 所示。

图 3-21　一热态位编码四进计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity hot_count is
  port(clk:in std_logic ;
       y:out std_logic_vector(3 downto 0));
end ;

architecture shift_1 of hot_count is
signal q: std_logic_vector(3 downto 0);
begin
process (clk)
begin
  if (clk'event and clk = '1') then
    if (q = "1000" or q = "0100" or q = "0010" or q = "0001") then
      q <= q(0) & q(3) & q(2) & q(1);
    else
      q <= "1000";
    end if;
  end if;
end process;
      y <= q ;
end ;
```

四、源程序说明

1. 本程序具有自启动功能。一旦电路进入非法状态之后,电路会自动恢复合法状态。

2. 一热态位编码四进计数器的仿真波形图如图 3-22 所示。由图可以看出，开机时电路进入“0000”非法状态，之后电路会自动跳出该非法状态而恢复合法状态“1000”。

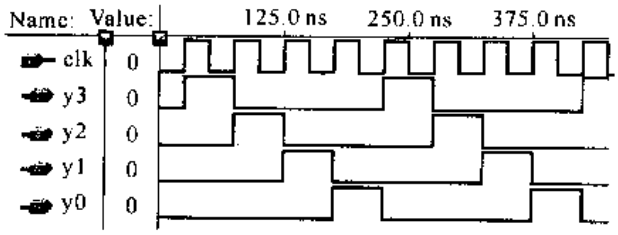


图 3-22 一热态位编码四进计数器仿真波形图

实例 3-15 4 位移存器型扭环计数器

一、设计任务

设计一个 4 位移存器型扭环计数器。4 位移存器型扭环计数器用 4 个触发器组成。4 个触发器可以组成一个八进移存器型扭环计数器。八进移存器型计数器的编码方案有 2 组。按移存顺序排列，一组是：15、14、12、8、0、1、3、7，这 8 个码按顺序形成格雷码（亦称循环码）；另一组是：13、10、4、9、2、5、11、6，这 8 个码按顺序形成非格雷码。由于格雷码的相邻代码之间只有一位发生变化，用其作为计数器的状态编码可靠性好，因此，设计扭环计数器时应当用格雷码编码，而把非格雷码作为非法状态码。

二、算法设计

用 if 语句描述 4 位移存器型扭环计数器。

三、VHDL 源程序

1. 文件名：shift_count.vhd

2. 端口图

4 位移存器型扭环计数器端口图如图 3-23 所示。

图 3-23 4 位移存器型扭环计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_count is
  port(clr,clk: in std_logic ;
       y : out std_logic_vector(3 downto 0));
end ;

architecture shift_1 of shift_count is
signal q:std_logic_vector(3 downto 0);
signal d0:std_logic;

begin
process (clk,clr)
begin
```

```
    if (clr = '0') then
    q <= "1111" ;
    elsif (clk'event and clk = '1') then
          q(0) <= d0;
          q(3 downto 1) <= q(2 downto 0);
    end if;
    if (q = "1111" or q = "1110" or q = "1100" or q = "1000"
          or q = "0000" or q = "0001" or q = "0011" or q = "0111") then
            d0 <= not(q(3));
    else
        if q = "1010" or q = "0101" then
          d0 <= not q(3);
        else
          d0 <= q(3) ;
        end if;
    end if;
end process;
        y <= q ;
end ;
```

四、源程序说明

1. 设计移存器型扭环计数器的关键是描述 q(0)级的输入信号 d0。根据扭环定义，d0 来自最后一级的反相端，但是应当考虑自启动问题。即考虑一旦电路进入非法状态，电路能够自动回到正确状态，本程序设计了电路自启动功能。

2. 4 位移存器型扭环计数器的仿真波形图如图 3-24 所示。为了检查设计电路的自启动功能，预置开机计数器的状态为“0101”，相应的仿真波形图如图 3-25 所示。图中显示，一开机计数器先进入非法状态“0101”、“1011”，之后即进入合法状态“0111”，电路自启动功能得到证明。

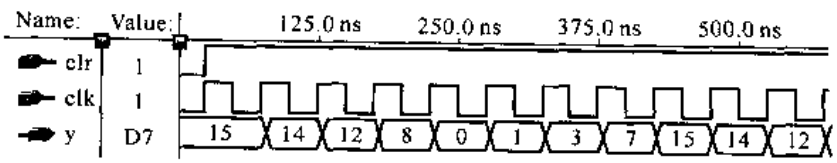


图 3-24　4 位移存器型扭环计数器的仿真波形图(1)

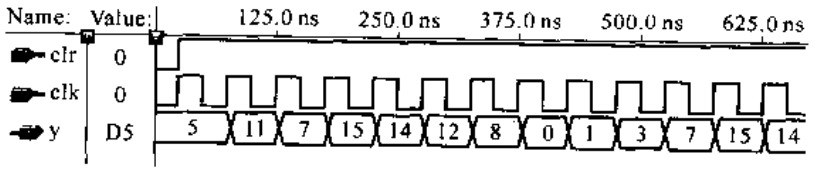


图 3-25　4 位移存器型扭环计数器的仿真波形图(2)

实例 3-16 七进移存器型计数器

一、设计任务

设计一个 4 位移存器型计数器。其编码方案为:0、1、3、15、14、12、8。

二、算法设计

用 if 语句描述七进移存器型计数器。

三、VHDL 源程序

1. 文件名:shift_count2.vhd

2. 端口图

七进移存器型计数器端口图如图 3-26 所示。

图 3-26 七进移存器型计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_count2 is
  port(clk: in std_logic ;
       y: out std_logic_vector(3 downto 0));
end ;

architecture shift_1 of shift_count2 is
signal q:std_logic_vector(3 downto 0);
signal d0,load:std_logic;

begin
process (clk)
begin
  if (clk'event and clk = '1') then
    if load = '1' then
      q(0) <= d0;
      q(3 downto 1) <= q(2 downto 0);
    else
      q <= "1111";
    end if;
  end if;
  if (q = "1111" or q = "1110" or q = "1100" or q = "1000" ) then
            d0 <= '0';
          load <= '1';
  elsif ( q = "0000" or q = "0001" ) then
            d0 <= '1' ;
          load <= '1';
  else
```

```
            load <= '0';
    end if;
end process;
        y <= q ;
end ;
```

四、源程序说明

1. 设计移存器型计数器的关键是描述 q(0)级的输入信号 d0 及各状态的连接。按移存规律状态 3 之后应为状态 7,而现在要求是 15,程序是这样处理的:电路系统检测出状态 3 后,等待下一个时钟来到,计数器只进行装载状态 15 的操作,而不进行移位操作。为了使计数器具有装载数据功能,电路中设置了自动装载控制端。

2. 七进移存器型计数器的仿真波形图如图 3-27 所示。

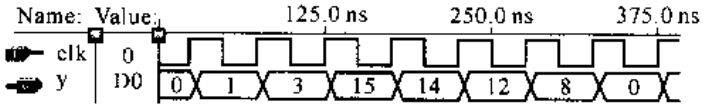


图 3-27　七进移存器型计数器的仿真波形图

实例 3-17　二进制($M=16$)计数器

一、设计任务

设计一个二进制($M=16$)计数器。一般把计数器的模值 $M=2^n$、状态编码为自然二进制数的计数器简称为二进制计数器。

二、算法设计

用 if 语句描述二进制($M=16$)计数器。

三、VHDL 源程序

1. 文件名:count16.vhd

2. 端口图

二进制($M=16$)计数器端口图如图 3-28 所示。

图 3-28　二进制($M=16$)计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count16 is
  port(clk: in bit ;
       oc: out bit;
        y: out integer range 0 to 15);
end ;
architecture a of count16 is
  signal q :integer range 0 to 15;
```

```
    begin
  p1: process (clk)
    begin
    if (clk'event and clk = '1') then
      q <= q + 1;
    end if ;
    if q = 15 then
      oc <= '0';
    else
      oc <= '1';
    end if;
    y <= q;
  end process ;
  end ;
```

四、源程序说明

1. 程序中 oc 是计数器进位输出端。当 M 值较大时，oc 的描述方法可参照实例 3-18，以避免进位脉冲宽度太小。

2. 用整数数据类型设计二进制计数器是很方便的。若要 $M = 256$，只要把整数数据范围改为：integer range 0 to 255。

3. 二进制计数器的仿真波形图如图 3-29 所示。

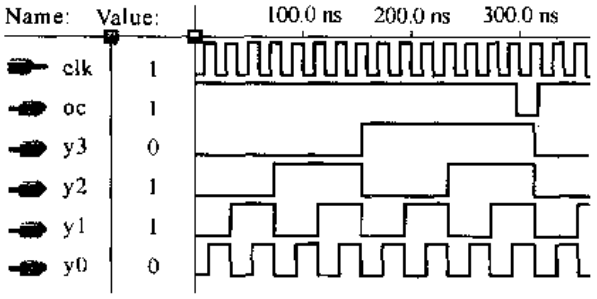


图 3-29 二进制($M = 16$)计数器的仿真波形图

实例 3-18 60 进计数器

一、设计任务

设计一个 60 进计数器。

二、算法设计

用 if 语句描述 60 进计数器。

三、VHDL 源程序

1. 文件名：count60.vhd

2. 端口图

60 进计数器端口图如图 3-30 所示。

图 3-30　60 进计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count60 is
  port(clk: in bit ;
       oc: out bit;
        y: out integer range 0 to 59);
end ;
architecture a of count60 is
  signal q :integer range 0 to 59;
  begin
p1: process (clk)
  begin
  if (clk'event and clk = '1') then
    if q = 59 then
      q <= 0;
    else
      q <= q + 1;
    end if ;
  end if;
    if q < 30 then
      oc <= '0';
    else
      oc <= '1';
    end if;
    y <= q;
end process ;
end ;
```

四、源程序说明

1. 程序中 oc 是计数器进位输出端。当 M 值较大时，对 oc 的描述请参照本例进行，以避免进位脉冲宽度太小。本例的 oc 是一个方波，其宽度等于 30 个时钟周期。线标之处计数器值为 59("111011")。

2. 60 进计数器的仿真波形图如图 3-31 所示。

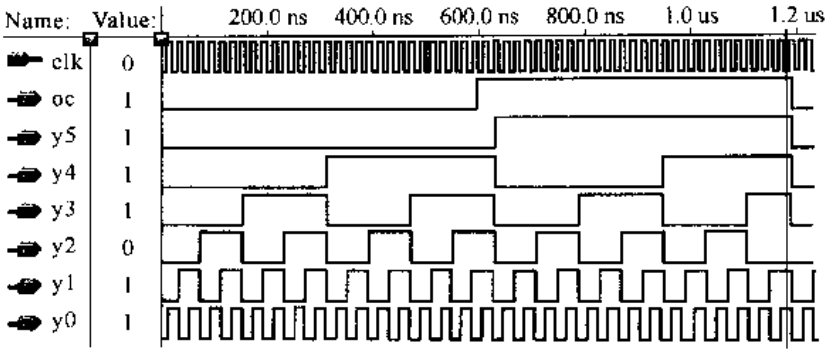


图 3-31 60 进计数器的仿真波形图

实例 3-19 BCD 码 60 进异步计数器

一、设计任务

设计一个 BCD 码 60 进异步计数器。个位显示 0～9，十位显示 0～5，均用 4 位二进制数表示。

二、算法设计

个位计数器的模 $M=10$，十位计数器的模 $M=6$。用 if 语句描述该计数器。

三、VHDL 源程序

1. 文件名：count6x10.vhd

2. 端口图

BCD 码 60 进异步计数器端口图如图 3-32 所示。

图 3-32 BCD 码 60 进异步计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count6x10 is
  port(clk: in std_logic;
       oc: out std_logic;
    y0,y1: out std_logic_vector(3 downto 0));
end ;
architecture a of count6x10 is
  signal q : std_logic_vector(3 downto 0);
```

```
    signal k : std_logic_vector(3 downto 0);
    signal j9,j60:std_logic;
    begin
p1: process (clk)        --进程 p1 描述个位计数器。
    begin
    if (clk'event and clk = '1') then
       if q = "1001" then
          q <= "0000" ;
       else
          q <= q + 1;
       end if ;
    end if;
    y0 <= q;
    end process ;
p2: process (j9)         --进程 p2 描述十位计数器。使用个位计数器的 q(3)作时钟。
    begin
    j9 <= q(3);
    if (j9'event and j9 = '0') then
       if k = "0101" then
          k <= "0000";
       else
          k <= k + 1 ;
       end if ;
    end if;
    y1 <= k ;
    if q = "1001" and k = "0101" then
       j60 <= '0';
    else
       j60 <= '1';
    end if ;
    oc <= j60 ;        --BCD 码 60 进计数器的进位输出。
end process ;
end ;
```

四、源程序说明

1. 两级计数器的模 $M = M1 \times M0$。本程序描述的计数器是一个异步计数器，个位计数器的时钟和十位计数器的时钟是不相同的，频率相差 10 倍。正式使用时往往还需要设置一个公共清零端。

2. BCD 码 60 进异步计数器的仿真波形图如图 3-33 所示，其局部放大图如图 3-34 所示。

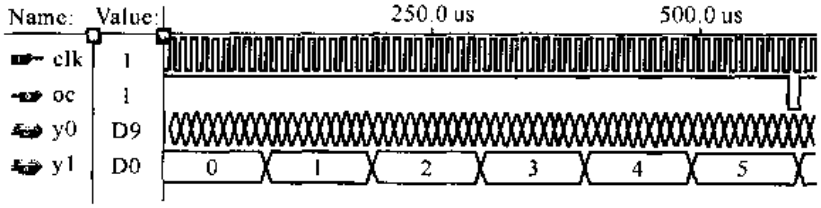


图 3-33 BCD 码 60 进异步计数器的仿真波形图

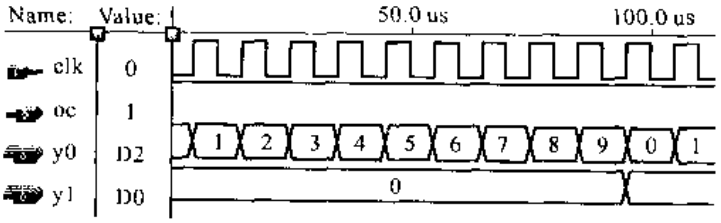


图 3-34 BCD 码 60 进异步计数器的仿真波形局部放大图

实例 3-20 BCD 码 60 进同步计数器

一、设计任务

设计一个 BCD 码 60 进同步计数器。个位显示 0～9，十位显示 0～5，均用 4 位二进制数表示。

二、算法设计

个位计数器的模 $M=10$，十位计数器的模 $M=6$。用 if 语句描述该计数器。

三、VHDL 源程序

1. 文件名：count6x10tb.vhd

2. 端口图

BCD 码 60 进同步计数器端口图如图 3-35 所示。

图 3-35 BCD 码 60 进同步计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count6x10tb is
  port(clk: in std_logic;
       clr: in std_logic;
        oc: out std_logic;
    y0,y1: out std_logic_vector(3 downto 0));
end ;
architecture a of count6x10tb is
  signal q : std_logic_vector(3 downto 0);
  signal k : std_logic_vector(3 downto 0);
  signal j9,j60:std_logic;
```

```
  begin
p1: process (clk)
  begin
  if clr = '0' then
      q <= "0000";        --清零。
  elsif (clk'event and clk = '1') then
      if q = "1001" then
        q <= "0000" ;
      else
      q <= q + 1;
      end if ;
  end if;
  y0 <= q;
  end process ;
p2: process (clk)
    begin
    if clr = '0' then
      k <= "0000";        --清零。
    elsif (clk'event and clk = '1') then
      if q = "1001" then      --见源程序说明 1。
        if k = "0101" then
          k <= "0000";
        else
         k <= k + 1;
        end if ;
      else
        k <= k ;
      end if ;
    end if;
    y1 <= k ;
    if q = "1001" and k = "0101" then
      j60 <= '0';
    else
      j60 <= '1';
    end if ;
    oc <= j60 ;
  end process ;
end ;
```

四、源程序说明

1. 两级计数器的模 $M = M1 \times M0$。本程序描述的计数器是一个同步计数器，由于十位计数器和个位计数器使用同一个时钟，所以程序中使用 q = "1001"条件对十位计数器的时钟作用时刻进行控制。

2. BCD 码 60 进计数器的仿真波形图如图 3-36 所示，其局部放大图如图 3-37 所示。

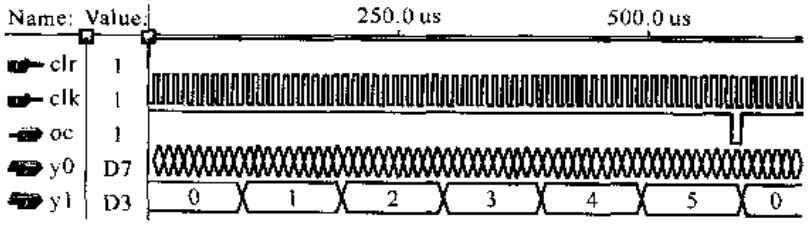


图 3-36 BCD 码 60 进同步计数器的仿真波形图

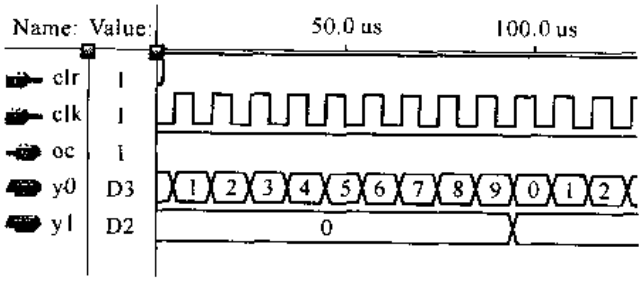


图 3-37 BCD 码 60 进同步计数器的仿真波形局部放大图

实例 3-21 用 VHDL 描述中小规模集成电路 74LS169

一、设计任务

用 VHDL 描述中小规模集成电路 74LS169。

二、算法设计

用 if 语句描述 74LS169 计数器。

三、VHDL 源程序

1. 文件名：altera169.vhd

2. 端口图

74LS169 计数器端口图如图 3-38 所示。

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
entity altera169 is
  port(
       d : in integer range 0 to 15;
     clk : in std_logic;
    entn : in std_logic;
```

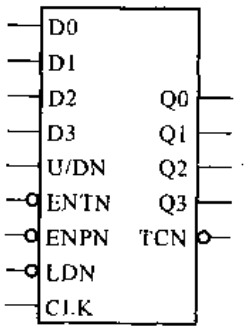


图 3-38 74LS169 计数器端口图

```
    enpn : in std_logic;
   u_dn : in std_logic;
    ldn : in std_logic;
    tcn : out std_logic;
      q : out integer range 0 to 15);
end ;

architecture a of altera169 is
  signal qi :integer range 0 to 15;
  signal ocin:std  logic;

begin
  process (clk, enpn,entn)
    begin

      if (clk'event and clk = '1' ) then
        if ldn = '0' then
          qi <= d;
        else
          if (enpn = '0' and entn = '0') then
            if( u_dn = '1' ) then
              qi <= qi + 1;
            else
              qi <= qi - 1;
            end if;
          else
            qi <= qi;
          end if;
        end if;

    if entn = '0' then

      if ( u_dn = '0' and qi = 0 )or ( u_dn = '1' and qi = 15 ) then
        ocin <= '0';
      else
        ocin <= '1';
      end if;

    else
        ocin <= '1';
    end if;
    q <= qi;
```

```
    tcn <= ocin;
  end process;
  end ;
```

四、源程序说明

1. Altera169 计数器仿真波形图如图 3-39 ~ 3-41 所示。

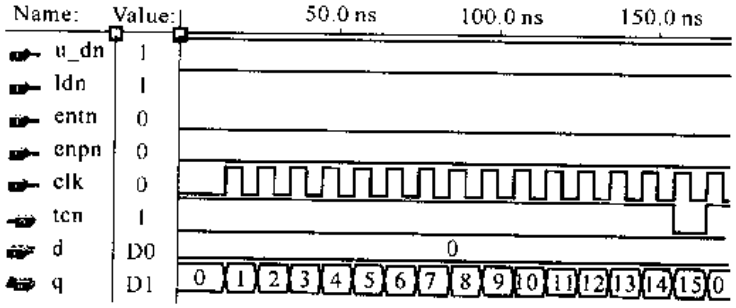


图 3-39 Altera169 计数器执行加计数操作

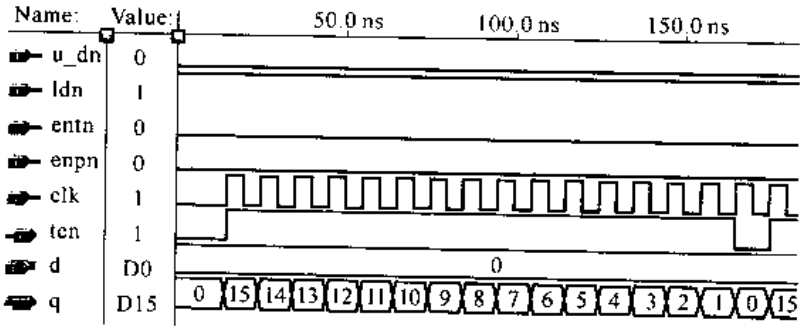


图 3-40 Altera169 计数器执行减计数操作

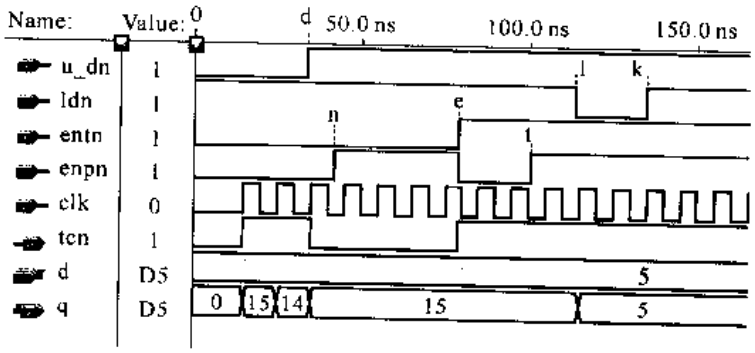


图 3-41 Altera169 计数器执行综合操作

2. 图 3-41 是综合仿真图，对该图说明如下：

(1) 0 ~ d 时间段计数器执行减计数操作。

(2) 从 d 时刻开始计数器执行加计数操作，同时输出状态为“15”，进位/借位信号变为低电平。

(3) n ~ e 时间段，由于计数器操作模式控制信号 enpn = '1'，entn = '0'，计数器进入保持状态，输出保持“15”状态，进位/借位信号保持'0'状态。

(4) 从 e 开始时间段计数器操作模式控制信号 entn = '1',电路功能规定只要 entn = '1',进位/借位信号即被置成'1'状态。

(5) l ~ k 时间段,并行预置控制信号 ldn = '0', 预置数 5 被载入计数器。图示说明设计电路符合集成电路 74LS169 的功能要求。

实例 3-22　模值可变计数器

一、设计任务

设计一种模值可变计数器,其模值变化范围为 2 ~ 16。

二、算法设计

用 if 语句描述。

三、VHDL 源程序

1. 文件名:variable_m.vhd

2. 端口图

模值可变计数器端口图如图 3-42 所示。

图 3-42　模值可变计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity variable_m is
  port(clk,cr: in std_logic;
    i:in integer range 0 to 14;
    y:out integer range 0 to 15);
end ;
architecture a of variable_m is
  signal fpq:integer range 0 to 15;
  signal  k:integer range 0 to 14;
  begin
p1: process (clk)
  begin
  k <= i;
  if cr = '0' then
    fpq <= 0 ;
  elsif (clk'event and clk = '1') then
    if fpq = 15 - k then
      fpq <= 0 ;
    else
      fpq <= fpq + 1;
    end if ;
```

```
    end if ;
  y <= fpq;
end process p1 ;
end ;
```

四、源程序说明

1. 本设计的关键是计数最大值的描述，只要写出其表达式“ fpq = 15 – k”，并找到 k 的来源，设计即可完成。本程序中的变量 k 由端口人工输入，或者自动输入。

2. 用此类电路组成可变分频系数的分频器，可以得到一系列频率信号。

3. 模值可变计数器的仿真波形图如图 3-43 所示。图中，$k = 6$，$M = 10$，改变 k 的值，即可改变计数器的模值。

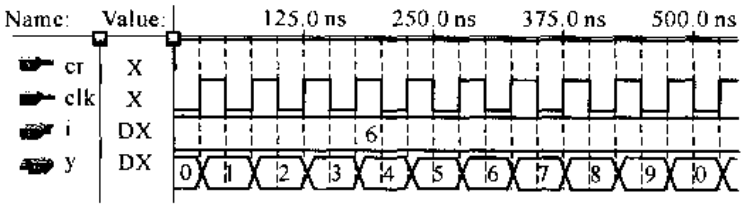


图 3-43 模值可变计数器的仿真波形图

实例 3-23 指定起始状态的计数器

一、设计任务

设计一种指定起始(初始化)状态的计数器。

二、算法设计

用 if 语句描述指定起始状态的计数器。

三、VHDL 源程序

1. 文件名：kj_ql.vhd

2. 端口图

指定起始状态的计数器端口图如图 3-44 所示。

图 3-44 指定起始状态的计数器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity kj_ql is
  port(clk: in std_logic;
       f: out std_logic_vector(0 to 2));
end ;

architecture a of kj_ql is
  signal fpq:std_logic_vector(0 to 2);
  signal clk2,kz:std_logic;
```

```
    begin
p1: process (clk)
    begin
    if (clk'event and clk = '1') then
        kz <= '1';
    end if;
end process;
p2: process (clk)
    begin
    if (clk'event and clk = '1') then
        clk2 <= not clk2;
    end if;
end process;
process (clk2)
    begin
    if kz = '0' then
      fpq <= "001";
    elsif (clk2'event and clk2 = '1') then
      if fpq = "0110" then
        fpq <= "001";
      else
        fpq <= fpq + 1;
      end if ;
    end if ;
      f <= fpq;
end process ;
end ;
```

四、源程序说明

1. 本设计要求计数器有指定的起始状态，为了实现这个要求，需设立一个预置控制信号。本程序设置 kz 作为控制信号，开机时 kz 值为 0，用 kz = '0'控制预置数；另外用时钟信号将 kz 置为'1'，使计数器能正常工作。

2. 指定起始状态的计数器的仿真波形图如图 3-45 所示。图中，计数器的起始状态设计为"001"，计数器的模为 6。

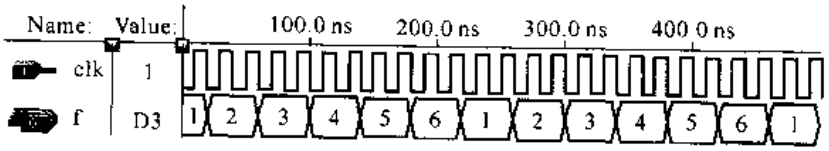


图 3-45　指定起始状态计数器的仿真波形图

第四节 状态机应用

实例 3-24 序列信号发生器

一、设计任务

设计一种能按规定输出 8 位"0"、"1"序列的信号发生器。

二、算法设计

欲完成设计任务,考虑用状态机结构进行设计,case 语句是描述状态机结构的最重要的语句。本序列信号发生器用一个有指定状态编码的、模值等于 8 的 8 位计数器组成。

三、VHDL 源程序

1. 文件名:count_xl.vhd

2. 端口图

序列信号发生器端口图如图 3-46 所示。

图 3-46 序列信号发生器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count_xl is
  port( clr,clk: in std_logic;
    y : out std_logic_vector (7 downto 0));
end ;

architecture a of count_xl is

  constant s0 :std_logic_vector (7 downto 0): ="10000000";
  constant s1 :std_logic_vector (7 downto 0): ="11000000";
  constant s2 :std_logic_vector (7 downto 0): ="11100000";
  constant s3 :std_logic_vector (7 downto 0): ="00010000";
  constant s4 :std_logic_vector (7 downto 0): ="11111000";
  constant s5 :std_logic_vector (7 downto 0): ="00000011";
  constant s6 :std_logic_vector (7 downto 0): ="11111011";
  constant s7 :std_logic_vector (7 downto 0): ="00000001";
  signal q :std_logic_vector (7 downto 0);
  begin
  p1: process (clk,clr)
    begin
    if (clr='0') then
      q<=s0;
    elsif (clk'event and clk='1') then
```

```
        case q is
                when s0  => q <= s1;
                when s1  => q <= s2;
                when s2  => q <= s3;
                when s3  => q <= s4;
                when s4  => q <= s5;
                when s5  => q <= s6;
                when s6  => q <= s7;
                when s7  => q <= s0;
                when others  => q <= s0;
        end case;
        end if;
        end process p1;
    y <= q;
end ;
```

四、源程序说明

1. 设计一个 8 位 $M=8$ 的计数器。计数器的每一位可以输出位长为 8 的 0、1 周期序列信号。为了得到规定的序列，则要按序列要求规定各个状态的编码。这实质是设计一个有指定编码状态的计数器。

2. 指定状态编码计数器的仿真波形图如图 3-47 所示。由图可见，计数器的每一位都可以输出一定序列的波形。例如，用 y7 作为序列输出信号，其输出序列是："11101010"。

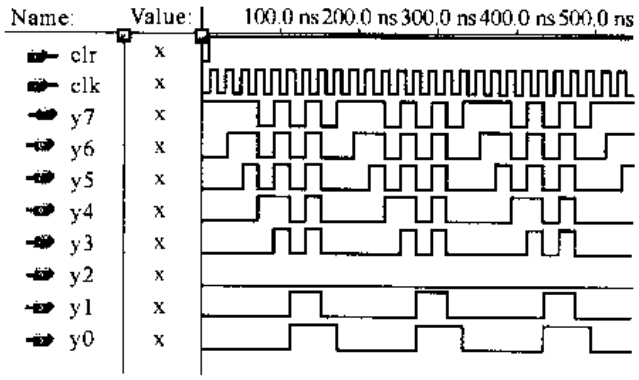


图 3-47　序列信号发生器的仿真波形图

实例 3-25　告　警　器

一、设计任务

设计一种告警器，告警时间在每天 6 点钟、23 点钟，告警信号为声音，声音持续 1 分钟。

二、算法设计

用仿状态机结构设计本告警器。考虑用 1 分钟作为时钟信号，则 1 天有 1 440 分钟(24 × 60)，设置的状态机有 1 440 个状态，6 点钟是第 360 个状态，23 点钟是第 1 380 个状态。时间达

到这两个状态,输出一个有效控制信号,同时控制声音源输出音频信号。

三、VHDL 源程序

1. 文件名:count24×60.vhd

2. 端口图

告警器端口图如图 3-48 所示。

图 3-48 告警器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity count24×60 is
  port(clk,clk1k: in bit ;
       speak: out bit);
end ;

architecture a of count24×60 is
  signal q :integer range 0 to 1439;
  signal oc: bit;

  begin
process (clk)        --clk 的单位为 1 分钟。
  begin

    if (clk'event and clk = '1') then
      if q = 1439 then
        q <= 0 ;
    else
        q <= q + 1;
    end if ;
  end if ;

  if (q = 360 or q = 1380) then        --按规定状态进行控制。
    oc <= '1';
  else
    oc <= '0';
  end if;

  speak <= (oc and clk1k);
end process ;
end ;
```

四、源程序说明

1. 由于在 1440 个状态中只有 2 个状态有输出,如果把所有状态都写出来,程序太长。为了减少文字量,采用本程序写法,而不采用状态机结构描述方法。

2. 程序中，oc 是输出信号，它通过一个与门控制声音源 clk1k(1 kHz 音频)是否输出信号 speak。若音频信号 speak 有输出，则将其外接到扬声器中播出告警声音。

实例 3-26　楼梯照明灯控制器

一、设计任务

设计一种楼梯照明灯控制器，当声音传感器探测到有人上楼或者下楼时，立即向照明灯控制器发出需要开灯信号，控制器相应输出开灯信号，控制照明灯亮；若楼层无人，则开灯信号无效，灯灭。假设有 3 层楼，每层安装一个声音传感器用以探测有无行人(本设计不包括声音传感器)。

二、算法设计

用状态机结构设计楼梯照明灯控制器。

三、VHDL 源程序

1. 文件名：asm.vhd

2. 端口图

楼梯照明灯控制器端口图如图 3-49 所示。

图 3-49　楼梯照明灯控制器端口图

3. 源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity asm is
  port(call1,call2,call3: in std_logic;
                clr,clk: in std_logic;
          kg1,kg2,kg3: out std_logic);
end ;
architecture a of asm is
type states is (s0,s1,s2);        --对状态机的状态进行声明。
signal q:std_logic_vector (0 to 2);
signal state:states;

begin
p1: process (clk,clr)
begin
  if (clr='0') then
    state<=s0 ;
  elsif (clk'event and clk='1') then
       case state is        --用 case 语句描述状态机最方便。
         when s0 =>
           state<=s1;
         when s1 =>
           state<=s2;
```

```
            when s2 =>
              state <= s0;
        end case;
      end if;
    end process p1;

    p2: process(call1, call2, call3)
    begin
      if cr = '0' then
        kg1 <= '0';
        kg2 <= '0';
        kg3 <= '0';
      else
        case state is
        when s0 =>
          if call1 = '1' then
            kg1 <= '1'; kg2 <= '0'; kg3 <= '0';
          else
            kg1 <= '0'; kg2 <= '0'; kg3 <= '0';
          end if;

        when s1 =>
          if call2 = '1' then
            kg2 <= '1'; kg1 <= '0'; kg3 <= '0';
          else
            kg2 <= '0'; kg1 <= '0'; kg3 <= '0';
          end if;

        when s2 =>
          if call3 = '1' then
            kg3 <= '1'; kg1 <= '0'; kg2 <= '0';
          else
            kg3 <= '0'; kg1 <= '0'; kg2 <= '0';
          end if;
       end case;
      end if;
    end process p2;
    end;
```

四、源程序说明

1. 程序所用时钟的频率应较高，以便能快速扫描多个楼层。

2. 本程序仿真波形图如图 3-50 所示。由图可见，设计要求已经实现。

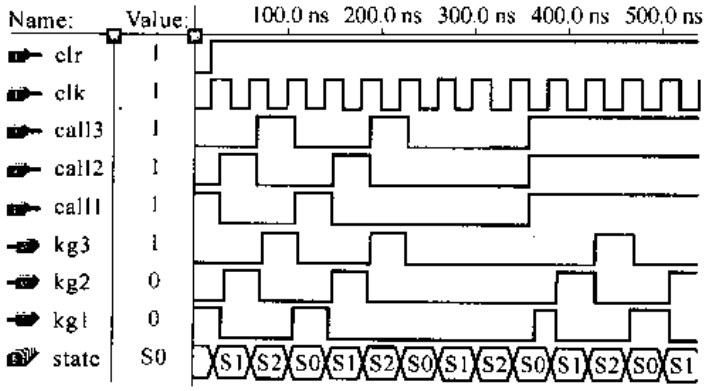


图 3-50　楼梯照明灯控制器仿真波形图

实例 3-27　循环彩灯控制器

一、设计任务

设计一个循环彩灯控制器，该控制器控制红、绿、黄三个发光管循环发亮。要求红发光管亮 2 秒，绿发光管亮 3 秒，黄发光管亮 1 秒。

二、算法设计

用状态机结构设计循环彩灯控制器。

三、VHDL 源程序

1．文件名：asm_led.vhd

2．端口图

循环彩灯控制器端口图如图 3-51 所示。

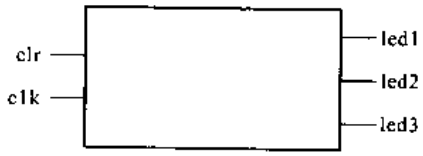


图 3-51　循环彩灯控制器端口图

3．源程序

```
library ieee;
use ieee.std_logic_1164.all;

entity asm_led is
  port(clr,clk:in std_logic;
    led1,led2,led3:out std_logic);
end ;
architecture a of asm_led is
type states is (s0,s1,s2,s3,s4,s5);
signal q:std_logic_vector (0 to 2);
signal state :states;

begin
p1: process (clk,clr)
begin
  if (clr='0') then
    state<=s0 ;
  elsif (clk'event and clk='1') then
```

```
        case state is
          when s0 =>
            state <= s1;
          when s1 =>
            state <= s2;
          when s2 =>
            state <= s3;
          when s3 =>
            state <= s4;
          when s4 =>
            state <= s5;
          when s5 =>
            state <= s0;
        end case;
  end if;
  end process p1;
  p2: process
  begin
    if cr = '0' then
      led1 <= '0'; led2 <= '0'; led3 <= '0';
    else
     case state is
          when s0 =>
            led1 <= '1'; led2 <= '0'; led3 <= '0';
          when s1 =>
            led1 <= '0'; led2 <= '1'; led3 <= '0';
          when s2 =>
            led1 <= '0'; led2 <= '1'; led3 <= '0';
          when s3 =>
            led1 <= '0'; led2 <= '0'; led3 <= '1';
          when s4 =>
            led1 <= '0'; led2 <= '0'; led3 <= '1';
          when s5 =>
            led1 <= '0'; led2 <= '0'; led3 <= '1';
     end case;
    end if;
  end process p2;
  end ;
  end ;
```

四、源程序说明

1．程序所用时钟的频率为 1 Hz，以便能快速扫描多个楼层。

2．本程序仿真波形图如图 3-52 所示。图中 led2 代表红发光管，led3 代表绿发光管，led1 代表黄发光管。由图可见，设计要求已经实现。

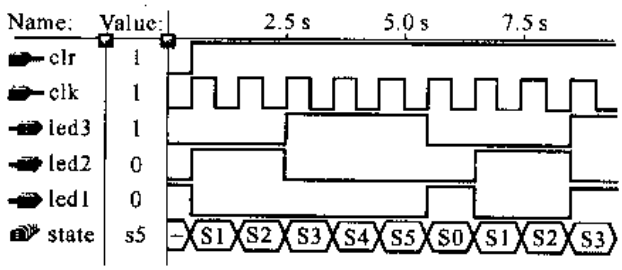


图 3-52　循环彩灯控制器仿真波形图

第四章　用电路图输入方法设计数字电路

用电路图输入方法设计数字电路，是 EDA 设计的一种重要方法。学会电路图输入方法，对提高使用开发商提供的或者他人提供的图形式电路模块，特别是使用逻辑功能丰富的图形式宏功能器件的能力，以及提高各种层次化设计水平都是有益的。本章通过几个实例的讲解，期望使读者能领会并掌握使用 MAX + plus Ⅱ 软件的图形输入方法进行电路设计的要领，从而提高电路设计水平。

实例 4-1　TFFE 触发器

通过了解 TFFE 触发器，掌握图形输入方法中电路引脚端的建立及使用引脚端的缺省参数的方法。

一、逻辑电路图

1. 文件名：tffe.gdf

2. 电路图

TFFE 触发器逻辑电路图如图 4-1 所示。

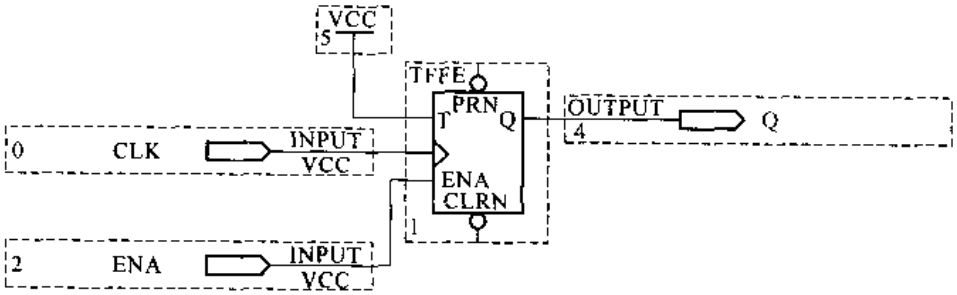


图 4-1　TFFE 触发器逻辑电路图

二、电路图输入文件说明

1. 利用图形输入方法进行数字电路设计的主要流程是：

算法设计→构建电路原理图→编译→仿真→器件下载→下载芯片测量

在图 4-1 中 PRN、CLRN 引脚端子被设计者置为悬空状态，综合编译时，软件系统按默认参数自动为这两个引脚端配置高电平，以便使触发器能正常工作。使器件正常工作是建立缺省参数的一个重要原则。

2. 电路的信号输入、输出、测试引脚端必须接引脚端元件，如图中的 ENA、CLK、Q 端所示的箭头块元件，即是引脚端元件。引脚端元件有单向输入、输出以及双向输入输出三种类型（电路图 4-1 中没有双向引脚）。

3. TFFE 触发器仿真波形图如图 4-2 所示。由图可见，TFFE 触发器具有二分频功能。

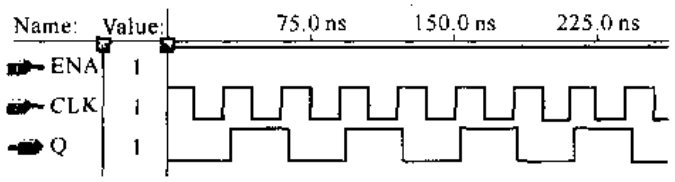


图 4-2　TFFE 触发器仿真波形图

实例 4-2　冒险电路

设计一个电路，用于研究组合电路可能产生的冒险现象。通过研究冒险电路，熟悉电路外接引脚的连接工具 inputc 和 outputc 元件的使用。

一、逻辑电路图

1. 文件名：hazad.gdf

2. 电路图

研究组合电路冒险现象的逻辑电路图如图 4-3 所示。

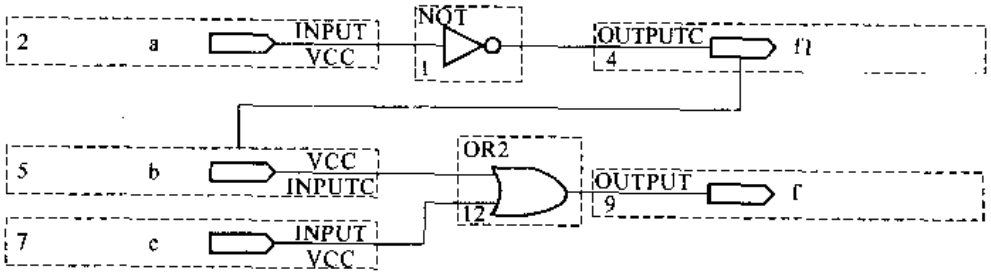


图 4-3　研究组合电路冒险现象的逻辑电路图

二、电路图输入文件说明

1. 若输出 f 和输入 a 满足逻辑表达式：$f = a + \bar{a}$，则电路存在冒险，冒险对逻辑电路产生的破坏程度随输入信号 a 的频率升高而升高。本实例设计了一个电路，用于研究组合逻辑电路可能产生的冒险现象。用 MAX + plus Ⅱ 软件实施实验电路的关键是应用元件 inputc 和 outputc。在图 4-3 中，$b = \bar{a}$，令 a = c，电路即可实现冒险函数。注意，电路冒险的起因是信号 a 和信号 $\bar{a}$ 不能同时达到或门的输入端。如果不用 inputc 和 outputc，而是把反相器的输出直接连到或门的 b 引脚，软件综合结果 a 和信号 $\bar{a}$ 之间没有时延，致使冒险现象观察不到。inputc 和 outputc 的功能是：连接外引脚，仅用于仿真。在编译综合时被连接的两个子电路是被各自独立编译综合的，反相器输出 f1 与输入信号 a 之间有时延。若把 f1 引回到或门的输入端 b，即可实现冒险函数。这里需要着重指出的是，用 inputc 和 outputc 进行连接，仅仅是为了进行仿真的需要，系统进行综合配置时，在物理上并没有把反相器输出和或门的输入端 b 连接在一起。

2. 本程序仿真波形图如图 4-4、4-5 所示。由图 4-4 可见，当输入信号周期为 40 ns，电路中产生的冒险现象很明显。由图 4-5 可见，当输入信号周期为 40 us，电路中产生的冒险现象（毛刺）的宽度就变得很窄（实际毛刺宽度并没有变小，而是由于屏幕显示的时间长了，相对来看，毛刺宽度变小了），尽管冒险依然存在，但对有效信号的威胁程度降低了。

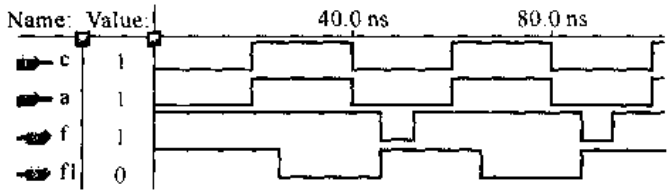


图 4-4 电路出现明显冒险现象的仿真波形图

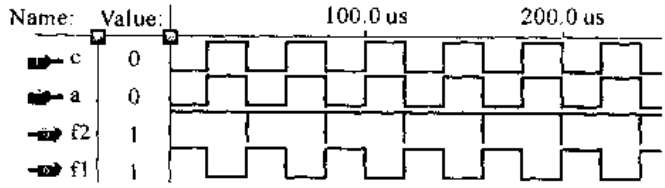


图 4-5 电路出现的冒险现象不太明显的仿真波形图

实例 4-3 74169 计数器的应用

通过了解 74169 计数器，掌握图形输入方法中引脚连线的操作方法。

一、逻辑电路图

1. 文件名：a_74169.gdf

2. 电路图

a_74169 计数器逻辑电路图如图 4-6 所示。

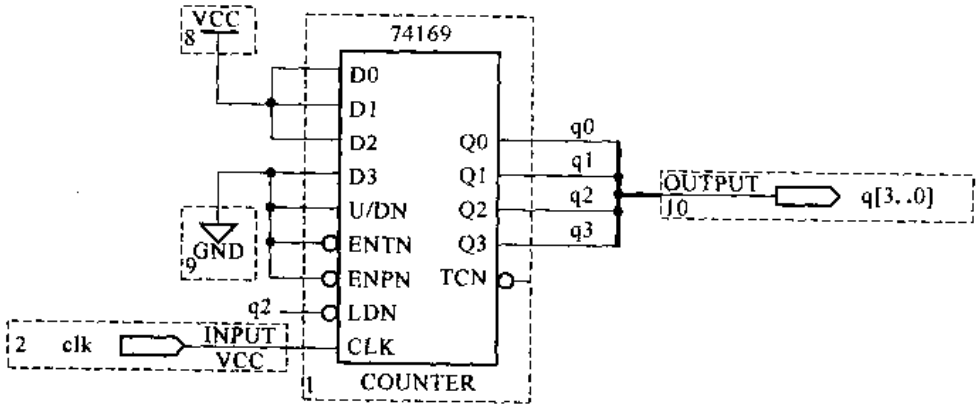


图 4-6 a_74169 计数器逻辑电路图

二、电路图输入文件说明

1. 在图 4-6 中，计数器输出引脚端用总线连接。注意连接方式：细线代表一位线，粗线代表多位线，称为 bus(总线)。细线连到粗线上，称为并位操作。并位后，由粗线引出的信号是一个多位信号。对多位信号，除了需要命名以外，还需要标明位宽。位宽的标明方法如图中 q[3..0]所示。注意，无论多少位，标明位宽的方括号中句点的个数都是 2 个。

2. 两个节点进行连接时，可以用相同信号名命名两个节点，即可实现文本方式连接。在图 4-6 中如要把 LDN 端和 q2 端相接，只要把 q2 标注在 LDN 端上，即实现 LDN 端和 q2 端的相接。换言之，同名节点端总是相接的。

3. a_74169 计数器的仿真波形图如图 4-7 所示。

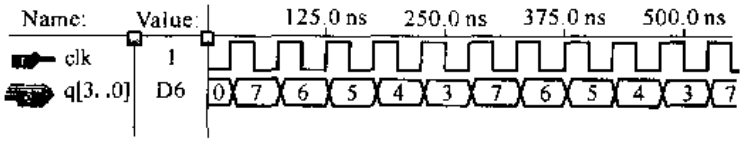


图 4-7　a_74169 计数器仿真波形图

实例 4-4　与门阵列控制器

通过了解与门阵列控制器，掌握图形输入方法中引入阵列元件的方法。

一、逻辑电路图

1. 文件名：and_array.gdf

2. 电路图

与门阵列控制器逻辑电路图如图 4-8 所示，其展平后的逻辑图如图 4-9 所示。

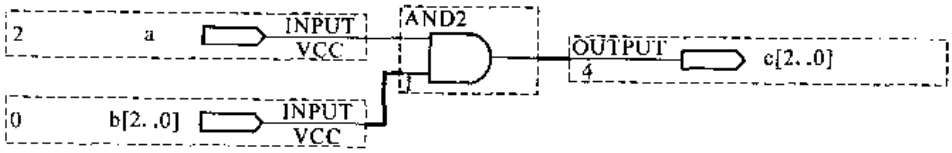


图 4-8　与门阵列控制器逻辑电路图

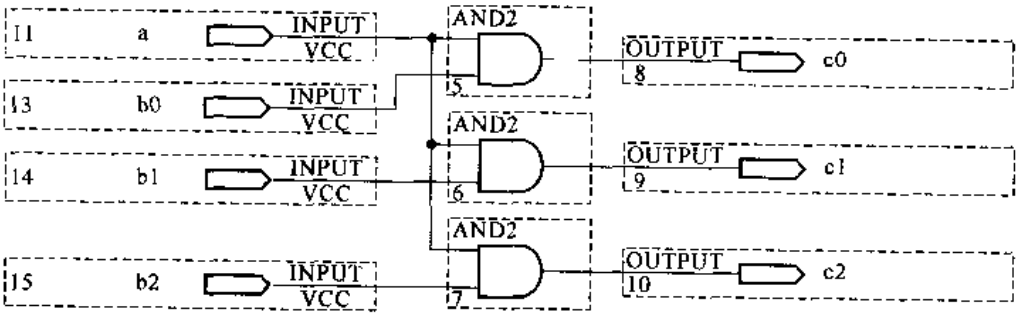


图 4-9　与门阵列控制器展平后的逻辑电路图

二、电路图输入文件说明

1. 在图 4-8 中，2 条总线分别连到与门的输入端 b 和输出端 c，总线标明为 3 位，说明总线要与同类 3 个与门的相应引脚相接，例如接在输出端的总线的各个位线要和 3 个与门输出端相接。图中，a 是一位线，表明它是公共线。图 4-8 中用总线引入多个与门，由类型相同的多个与门组成与门阵列。

2. 图 4-9 所示电路是图 4-8 所示电路展平后的逻辑电路图。显见，与门阵列画法较图 4-9 所示电路画法简易。

3. 与阵列控制器仿真波形图如图 4-10 所示。由图可见,a 信号是与阵列控制器的控制信号。a = '1'时,电路执行" c <= b;"操作。

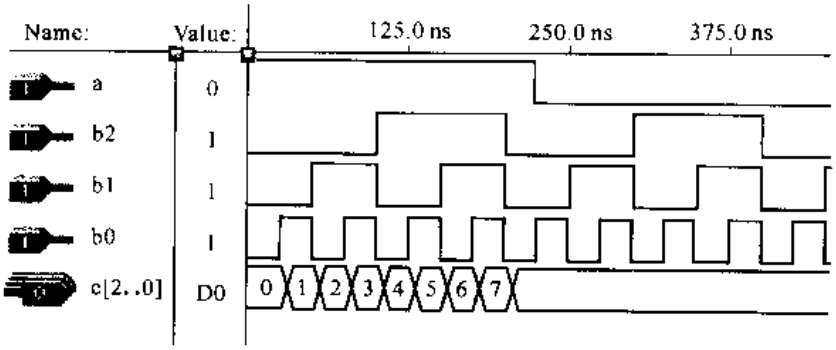


图 4-10 与阵列控制器仿真波形图

实例 4-5 参数型宏功能与门

通过了解参数型宏功能与门,掌握图形输入方法中 2 维总线的表示方法。

一、逻辑电路图

1. 文件名:lpm_and.gdf

2. 电路图

LPM_AND 参数型宏功能与门逻辑电路图如图 4-11 所示。

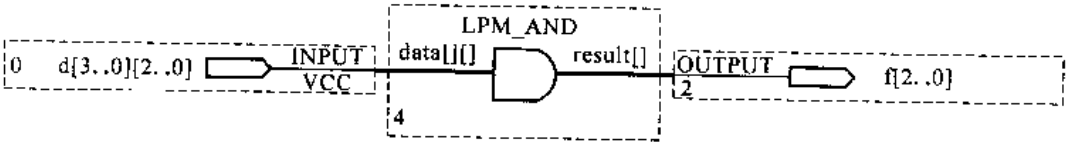


图 4-11 LPM_AND 参数型宏功能与门逻辑电路图

二、电路图输入文件说明

1. 在图 4-11 中,2 条总线分别连到一个与门的输入端和输出端。标识符 d[3..0][2..0] 的含义是:第一个方括号内的参数称为 LPM_SIZE,即说明一个与门的输入信号有多少个,本例定义一个与门的输入信号有 4 个;第二个方括号内的参数称为 LPM_WIDTH,即说明一个 LPM_AND 参数型宏功能与门内有多少个与门,本例定义一个 LPM_AND 参数型宏功能与门内有 3 个与门。

2. 图 4-12 所示电路是图 4-11 所示电路展平后的逻辑电路图。显见采用图 4-11 所示画法可使电路画法简洁。

3. LPM_AND 参数型宏功能与门中的一个与门的仿真波形图如图 4-13 所示。

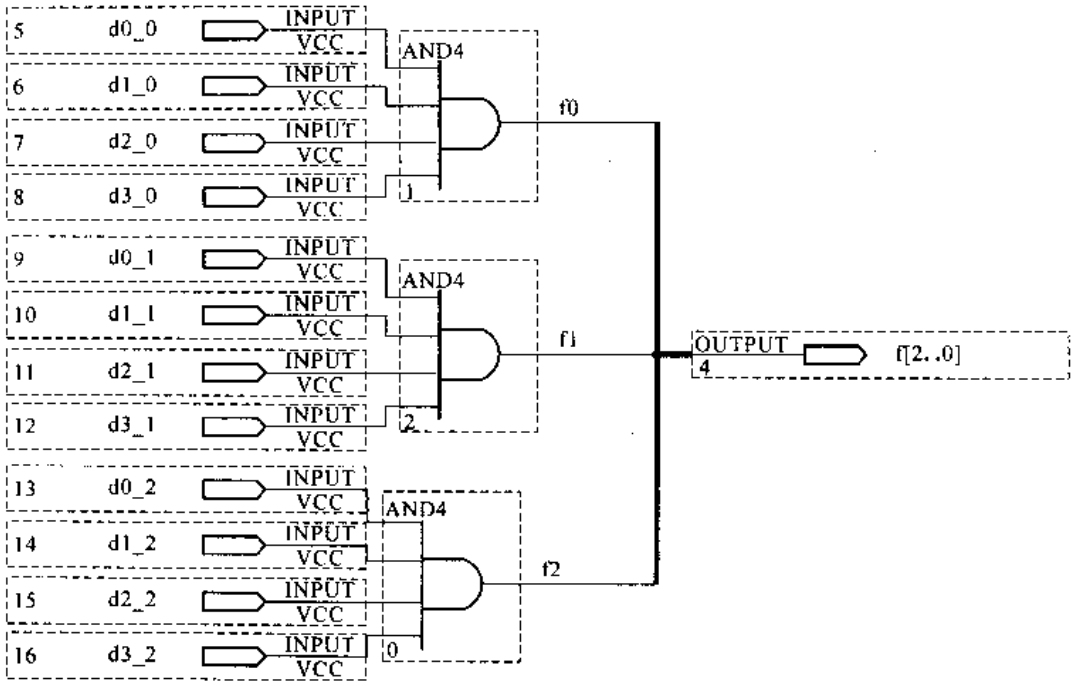


图 4-12　lpm_and 参数型宏功能与门展平后的逻辑电路图

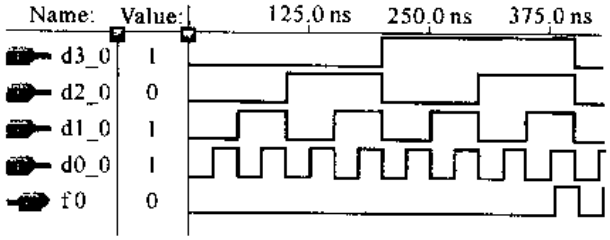


图 4-13　lpm_and 参数型宏功能与门中的一个与门的仿真波形图

实例 4-6　序列发生器

通过了解序列发生器，掌握图形输入方法与文本输入方法的层次设计方法。

一、逻辑电路图

1. 文件名：xl.gdf

2. 电路图

序列发生器逻辑电路图如图 4-14 所示。

二、电路图输入文件说明

1. 利用图形输入和文本(VHDL 代码)输入进行数字电路设计主要流程是：

算法设计→编辑 VHDL 代码生成自制器件，同时生成元件图符→
用自制器件和标准模块构建逻辑电路原理图→编译→仿真→
器件下载→下载芯片测量

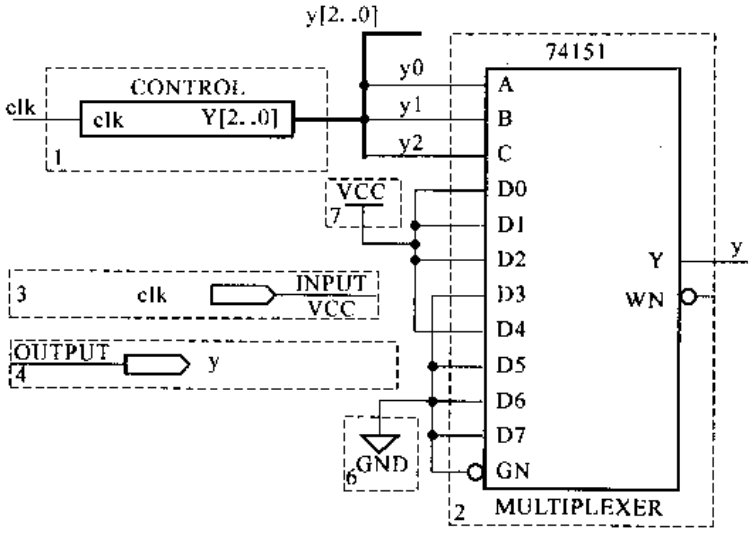


图 4-14 序列发生器逻辑电路图

2. 在图 4-14 中,自制器件 control 的电路图符功能由 VHDL 程序 control.vhd 定义,标准模块 74151(8 选 1 选择器)由图形文件定义,利用这两个器件合成序列发生器。由 control 器件输出端 y 供给 74151 需要的选择控制信号(A、B、C)。选择器 74151 的数据输入端并接入"11101000"固定数据,此数据在地址信号的连续选择下,从选择器串口 y 顺序输出,生成序列码"11101000"。

3. 文件 xl.gdf 和文件 control.vhd 应处在同一个目录中,否则编译 xl.gdf 时会出错。

4. 注意,control 的输出 y 应被命名并标注信号宽度,同时选择控制信号 A、B、C 端口也要明确引入哪一个信号(用位索引号码标示,图 4-14 示为 y0、y1、y3)。另外还要注意序列发生器 clk 输入端、选择器串口 y 端的连接方法。

5. xl.gdf 相对 control、74151 是上层设计,而 control、74151 是 xl.gdf 的下层设计。如果设计进一步扩大,xl.gdf 亦可能变为一个下层设计。

6. control.vhd 源文件

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY control IS
  PORT(clk: IN BIT ;
       y: OUT INTEGER RANGE 0 TO 7);
END ;

ARCHITECTURE a OF control IS
  SIGNAL q: INTEGER RANGE 0 TO 99;
  SIGNAL j: INTEGER RANGE 0 TO 15;
  SIGNAL clk1:BIT ;

  BEGIN
```

```
p1: PROCESS (clk)
  BEGIN
  IF (clk'event AND clk = '1') THEN        --进程 p1 描述 100 分频器的生成。
    IF q = 99 THEN
      q <= 0 ;
    ELSE
      q <= q + 1;
    END IF;
    IF q < 50 THEN
      clk1 <= '0';       --clk1 作为 74151 的选择控制计数器 j 的触发时钟。
    ELSE
      clk1 <= '1';
    END IF;
  END IF ;
END PROCESS ;
p2: PROCESS(clk1)        --进程 p2 描述标准器件 74151 的选择控制信号 y 的生成。
  BEGIN
  IF (clk1'event AND clk1 = '1') THEN
    IF j = 7 THEN
      j <= 0;
    ELSE
        j <= j + 1;
    END IF;
  END IF ;
    y <= j;
END PROCESS;
END ;
```

7. 序列发生器仿真波形图如图 4-15 所示。图中由于 clk 信号的频率比序列信号的频率高几百倍，因此 clk 信号被压缩为辨不清波形细节的黑带。

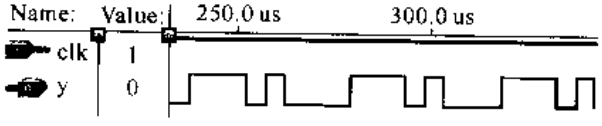


图 4-15　序列发生器仿真波形图

第五章　资源调用与特色电路

第一节　资源调用

利用 VHDL 的公共资源库和程序包，可提高电路设计的效率并降低设计成本。本节通过库、包调用实例，介绍如何调用 VHDL 库的公共资源。

实例 5-1　不用调库令调用自制器件

一、VHDL 源程序

1. 调用库器件的文件名：call_fpq.vhd

2. 调用库器件的源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY call_fpq IS
PORT(clk:IN STD_LOGIC;
     y:BUFFER STD_LOGIC);          --由于二分频器使用端口反馈，端口必须定义
                                   --BUFFER 类型。
END;

ARCHITECTURE a OF call_fpq IS
  COMPONENT fpq IS       --说明调用的器件。
    PORT(clk:IN STD_LOGIC;
         y:BUFFER STD_LOGIC);
  END COMPONENT;

BEGIN
u1: fpq PORT MAP(clk,y);
END;
```

3. 被调用器件的源文件名：fpq.vhd

4. 被调用器件的源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY fpq IS
  PORT(clk:IN STD_LOGIC;
       y:BUFFER STD_LOGIC);          --由于二分频器使用端口反馈，端口必须定义
                                     --BUFFER 类型。
END;
```

```
ARCHITECTURE a OF fpq IS
SIGNAL q : STD_LOGIC;
BEGIN
p:PROCESS(clk)
  BEGIN
   IF (clk'event AND clk = '1') THEN
    q <= NOT q;
   END IF;
    y <= q;
END PROCESS;
END;
```

二、源程序说明

1. 用 fpq.vhd 描述一个二分频器，该分频器作为自制器件供其他程序调用。

2. 在程序 call_fpq.vhd 中，用 component 语句调用自制器件 fpq。

3. 若自制器件的描述文件 fpq.vhd 和调用文件 call_fpq.vhd 处于同一个目录中，则在 MAX + plus Ⅱ 系统中，无需设置库调用令，只需用 component 语句及其例化语句即可调用自制器件。

4. 调用文件 call_fpq.vhd 中的端口名称可以与自制器件 fpq 的端口名称相同。

实例 5-2　使用调库、调包令调用程序包中的自制器件

一、VHDL 源程序

1. 包程序文件名：myuse.vhd

2. 包源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE myuse IS
  COMPONENT component_and IS        --将自制器件存于 myuse 包中。
  PORT(d1,d2: IN STD_LOGIC;
           f: OUT STD_LOGIC);
  END COMPONENT;

END ;
```

3. 调用库器件的文件名：call_component_and.vhd

4. 调用库器件的源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.myuse.component_and; --调用工作库(即当前目录)中的器件 component_and。

ENTITY call_component_and IS
PORT(k1,k2,k3,k4: IN STD_LOGIC;
```

```
                f1: OUT STD_LOGIC);
END;

ARCHITECTURE a OF call_component_and IS
SIGNAL m1,m2:STD_LOGIC;

BEGIN
u1: component_and
      PORT MAP (k1,k2,m1);
u2: component_and
      PORT MAP (k3,k4,m2);
u3: component_and
      PORT MAP (m1,m2,f1);
END;
```

5. 被调用器件的文件名:component_and.vhd

6. 被调用器件的源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY component_and IS        --生成被调用器件2输入端与门。
GENERIC (tr: time);
        PORT(d1,d2: IN STD_LOGIC;
                   f: OUT STD_LOGIC);
        END;

ARCHITECTURE a OF component_and IS
BEGIN
      f<=d1 AND d2;
END;
```

二、源程序说明

1. 用 component_and 描述一个 2 输入端与门,该与门作为自制器件供其他程序调用。

2. 在程序 call_component_and.vhd 中,用 USE work.myuse.component_and 库语句调用自制器件 component_and。库语句只调用 myuse 软件包中的 component_and 器件。

3. 自制器件的描述文件 component_and.vhd、调用文件 call_component_and.vhd 和程序包文件 myuse.vhd 应处于同一个目录中。在使用包语句前应先编译 myuse.vhd。若使用库语句,则在 call_component_and.vhd 文件中,无需设置 component 调用元件语句。

实例 5-3 使用调库、调包令调用程序包中定义的函数

一、VHDL 源程序

1. 包程序文件名: myuse01.vhd

2. 包源程序

```
LIBRARY IEEE ;
```

```
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE myuse01 IS
  FUNCTION max(j:BIT_VECTOR; k:BIT_VECTOR) RETURN BIT_VECTOR; --函数声明。
END myuse01;

PACKAGE BODY myuse01 IS      --用包体具体定义一个函数。
FUNCTION max(j:BIT_VECTOR(2 downto 0);
             k:BIT_VECTOR(2 downto 0))
      RETURN BIT_VECTOR IS

VARIABLE tmp:BIT_VECTOR (2 downto 0);
BEGIN
  IF(j > k) THEN
   tmp:= j;
  ELSE
   tmp:= k;
  END IF;
RETURN tmp;
END ;
END ;
```

3. 调用库器件的文件名:call_function.vhd

4. 调用库器件的源程序

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.myuse01.all;

ENTITY call_function IS
  PORT( data1,data2,data3:IN BIT;
        dat1,dat2,dat3:IN BIT;
                  set:IN BIT;
    dataout:OUT BIT_VECTOR(2 downto 0));
END;

ARCHITECTURE maxdet OF call_function IS
SIGNAL j1,k1: BIT_VECTOR(2 downto 0);
BEGIN
  j1 <= data1 & data2 & data3;
  k1 <= dat1 & dat2 & dat3;
PROCESS
  BEGIN
    IF (set = '1') THEN
```

```
        dataout <= "000";
      ELSE
        dataout <= max(j1,k1);        --将 j1、k1 值代入函数表达式中,得到返回值。
      END IF;
    END PROCESS;
  END;
```

二、源程序说明

1. 在程序包 myuse01 中定义一个求两个数中的最大数的函数 max(j,k),供调用赋值。

2. 在程序 call_function.vhd 中,用 USE work.myuse01.all 库语句调用函数 max(j,k)。

3. 由于函数 max(j,k)已在包 myuse01 中定义,所以不需要另外建立描述函数 max(j,k)的文件。

4. 本例仿真波形图如图 5-1 所示。图中,输入两个数 5、7,输出大数 7。当然在应用中两个数是自动输入、自动比较、自动得出比较结果的。

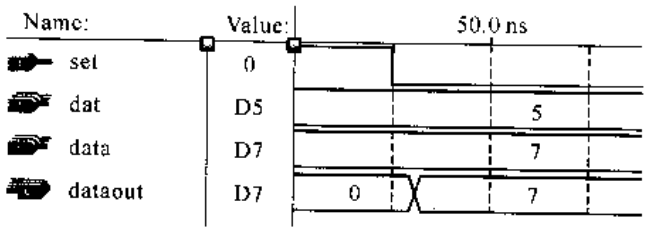


图 5-1 求两个数中的最大数的仿真图

5. 在程序包中可以定义函数、子程序,也可以由用户自行定义子数据类型。下面给出在程序包中定义一个子数据类型的例子。程序中定义的子数据类型是一个整数,其范围是 0~15。包文件名称是 sy04.vhd。

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

PACKAGE SY04 IS
SUBTYPE int_15 IS integer RANGE 0 TO 15;
END SY04;
```

实例 5-4 使用调库、调包命令调用软件包中定义的子程序

一、VHDL 源程序

1. 包程序文件名:myuse02.vhd

2. 包源程序

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE myuse02 IS
  PROCEDURE jfq (din1: in integer range 0 to 31;      --子程序声明。
```

```
                din2: in integer range 0 to 31;
                dout: out integer range 0 to 31);
END myuse02;

PACKAGE BODY myuse02 IS      --用包体具体定义一个子程序。
  PROCEDURE jfq (din1,din2:IN INTEGER RANGE 0 TO 31;
      VARIABLE dout :OUT INTEGER RANGE 0 TO 31) IS
  BEGIN
    dout: = din1 + din2;
  END jfq;
END;
```

3. 调用库器件的文件名:call_procedure.vhd

4. 调用库器件的源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.myuse02.all;

ENTITY call_procedure IS
    PORT (d1 : IN INTEGER RANGE 0 TO 31;
          d2 : IN INTEGER RANGE 0 TO 31;
        fout :OUT INTEGER RANGE 0 TO 31);
END;
ARCHITECTURE a OF call_procedure IS
  BEGIN
PROCESS (d1,d2)
  VARIABLE fo:INTEGER RANGE 0 TO 31;
  begin
    jfq(d1,d2,fo);
    fout <= fo;
END PROCESS;
END;
```

二、源程序说明

1. 在程序包 myuse02 中定义一个求两个数加法运算的子程序,供调用。

2. 在程序 call_procedure.vhd 中,用 USE work.myuse02.all 库语句调用子程序 jfq(din1, din2,dout)。

3. 由于子程序 jfq(din1,din2,dout)已在包 myuse02 中定义,不需要另外建立描述子程序 jfq(din1,din2,dout)的文件了。

4. 本例仿真波形图如图 5-2 所示。图中,输入两个加数为 5、7,和为 12。注意,本例和的最大计数值是 31。

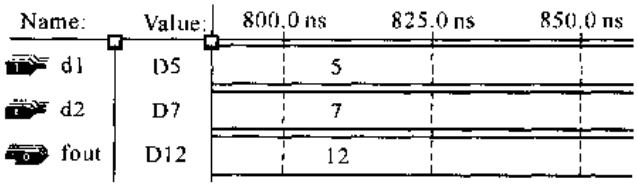


图 5-2 求两个数和运算的仿真图

实例 5-5 为支持不同数据类型进行运算,调用库系统程序包

一、VHDL 源程序

1. 文件名:function3.vhd

2. 源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;      --见源程序说明 1。

ENTITY function3 IS
    PORT (din1,din2 : IN STD_LOGIC_VECTOR(0 TO 3);
          dout : OUT STD_LOGIC_VECTOR(0 TO 3));
END;
ARCHITECTURE a OF function3 IS

  FUNCTION ls_xj(d1,d2:IN STD_LOGIC_VECTOR(0 TO 3))
                    RETURN STD_LOGIC_VECTOR IS
  VARIABLE temp :STD_LOGIC_VECTOR(0 TO 3) ;
  BEGIN
    temp:=d1 + d2;
    RETURN temp;
  END;

  BEGIN
    dout <= ls_xj (din1,din2);
END;
```

二、源程序说明

1. 为了支持标准矢量数据类型的两个数进行加法运算,需要调用程序包 STD_LOGIC_UNSIGNED.ALL给予支持。

2. 文件在结构体内定义了两个标准矢量数据类型的数进行相加的函数 xj(d1,d2),同时调用它。

3. 函数既可以在程序包中定义,也可以在结构体内定义。

实例 5-6　在结构体内定义一个子程序(过程)

一、VHDL 源程序

1. 文件名:procedure2.vhd

2. 源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY procedure2 IS
    PORT (din1 : IN INTEGER RANGE 0 TO 31;
          din2 : IN INTEGER RANGE 0 TO 31;
          dout : OUT INTEGER RANGE 0 TO 31);
END ;
ARCHITECTURE a OF procedure2 IS
  PROCEDURE jfq (d1,d2:IN INTEGER RANGE 0 TO 31;
      VARIABLE fout :OUT INTEGER RANG 0 TO 31) IS
  BEGIN
    fout: = d1 + d2;
  END;

BEGIN
  PROCESS
  VARIABLE fo:INTEGER RANG 0 TO 31;
    BEGIN
    jfq(din1,din2,fo);      --通过调用过程得到 fo 值,但是过程不等于 fo 值。
    dout <= fo;
END PROCESS;
END;
```

二、源程序说明

1. 在结构体内定义一个子程序:

PROCEDURE jfq(d1,d2,fout)

2. 在结构体内调用子程序:jfq(din1,din2,fo),其形参 d1,d2,fout 被实参 din1,din2,fo 替换。不能用 jfq(din1,din2,fo)给 dout 赋值。

实例 5-7　调用 Altera 公司的库元件 DFF(D 触发器)和 74151b(选择器)

一、VHDL 源程序

1. 文件名: components.vhd

2. 源程序

```
LIBRARY altera;
```

```
USE altera.maxplus2.all;        -- 调用 Altera 元件库。

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY components IS
    PORT
    (din,clk,crn,setn: IN STD_LOGIC;
                   qo: OUT STD_LOGIC;

         a, b, c, gn: IN STD_LOGIC;
                   d: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
              y, wn: OUT STD_LOGIC);
END ;

ARCHITECTURE a OF components IS
BEGIN
  dff1 : DFF
  PORT MAP(d => din,q => qo,clk => clock,clrn => crn,prn => setn);

  mux: a_74151b
  PORT MAP (c, b, a, d, gn, y, wn);
END a;
```

二、源程序说明

1. 使用下述调库令：

```
LIBRARY altera;
USE altera.maxplus2.ALL;
```

调用 Altera 公司的库元件 dff、74151b。

2. 由于 74151b 宏功能元件的书写方法不符合 VHDL 语言对标识符的规定，因此在 74151b 的字首加 a_ 前缀，使改变后的 a_74151b 符合 VHDL 语言对标识符的规定，这样就可以在 VHDL 语言描述中使用 Altera 公司的库元件 74151b。

第二节　特色电路

实例 5-8　计数器型防抖动电路(1)

在数字电路设计中经常采用机械开关作为控制开关，但是机械开关有其固有的缺点，如容易引入抖动干扰。为了消除抖动干扰，需要使用防抖动电路。

一、VHDL 源程序

1. 文件名：fd_dmc.vhd

2. 源程序

```
LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY fd_dmc IS
    PORT(clk,key:IN STD_LOGIC;
         cpo:OUT STD_LOGIC);
END ;
ARCHITECTURE a OF fd_dmc IS
  SIGNAL cp:STD_LOGIC;
  SIGNAL jsq:integer range 0 to 3 ;
  BEGIN

d: PROCESS (clk)
   BEGIN
IF (clk'event and clk = '1') THEN
    IF key = '1' THEN
      IF jsq = 3 THEN
        jsq <= jsq;
      ELSE
        jsq <= jsq + 1;
      END IF ;
      IF jsq = 1 THEN
          cp <= '1';
      ELSE
          cp <= '0';
      END IF;
    ELSE
        jsq <= 0;
    END IF;
END IF;
cpo <= cp;
END PROCESS;
END;
```

二、源程序说明

1. 工作原理

本例描述的防抖动电路属于计数器型防抖动电路。其工作原理是，设置一个模值为 4 的控制计数器，在人工按键 key = '1'时，执行加 1 计数；key = '0'时，计数器进入状态 0。计数器只在状态 1 有输出。计数器进入状态 3，处于保持状态。总之，按键一次，计数器只有一个单脉冲输出。本防抖动电路亦可以作为无抖动单脉冲发生器使用。

2. 防抖动原理

按键 key 是产生抖动的根源，按照设计，只有按键持续时间大于 3 个时钟周期，计数器输

出才可能产生有效正跳变,输出一个单脉冲。由于机械开关抖动产生的毛刺宽度小于 3 个时钟周期,因而毛刺作用不可能使计数器有输出,防抖动目的得以实现。

3. 本实例的仿真波形图如图 5-3、图 5-4 所示。

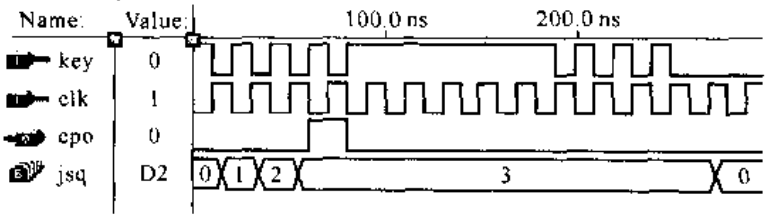


图 5-3 防抖动电路仿真波形图(1)

由图 5-3 可见,按键持续时间大于 3 个时钟周期,计数器输出一个单脉冲,其宽度等于 1 个时钟周期。窄脉冲用作模拟抖动干扰,由图可见,抖动不能干扰正常的单脉冲输出。

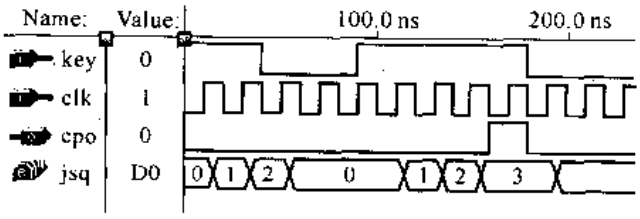


图 5-4 防抖动电路仿真波形图(2)

由图 5-4 可见,按键持续时间小于 3 个时钟周期计数器无输出;按键持续时间大于 3 个时钟周期,计数器只输出一个单脉冲,防抖动目的得到实现。长时间按键时,由于按键持续时间大于 3 个时钟周期,此时,计数器亦只输出一个单脉冲。

实例 5-9 计数器型防抖动电路(2)

一、VHDL 源程序

1. 文件名:fd_dmc.vhd

2. 源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY fd_dmc IS
    PORT(key,clk: IN STD_LOGIC;
         cpo:OUT STD_LOGIC);
END ;
ARCHITECTURE fd OF fd_dmc IS
  SIGNAL qen,cp,cr:STD_LOGIC;
  SIGNAL jsq:integer range 0 to 15;
  BEGIN
d1: PROCESS (clk)
```

```
    BEGIN
  IF qen = '0' THEN
     jsq <= 0;
  ELSIF (clk'event and clk = '1') THEN
    IF jsq = 7 THEN
      jsq <= 0 ;
    ELSE
      jsq <= jsq + 1;
    END IF;
    IF jsq = 0 THEN
     cp <= '1';
    ELSE
     cp <= '0';
    END IF;
  END IF;
  cpo <= cp;
  IF jsq < 7 THEN
    cr <= '1';
  ELSE
     cr <= '0';
  END IF;
  END PROCESS d1 ;
  d2: PROCESS (clk)
   BEGIN
    IF (clk'event and clk = '0') THEN
      IF cr = '1' THEN
        IF key = '1' THEN
          qen <= '1';
        ELSE
          qen <= qen;
        END IF;
      ELSE
        qen <= '0';
    END IF;
    END IF;
  END PROCESS d2;
  END;
```

二、源程序说明

1. 工作原理

本例描述的防抖动电路属于计数器型防抖动电路，控制计数器工作一个循环周期(8 个状态)，且仅在一个状态有输出。电路设计了连锁控制设施。在计数器处于状态 0 时，此时若有按键操作，则计数器进入状态 1，同时输出单脉冲(其宽度等于时钟周期)。计数器处于其他状态，都没有单脉冲输出。计数器处于状态 7 时，cr 由'1'变为'0'，控制 qen = '0'，导致计数器退出状态 7，进入状态 0，同时 cr 恢复'1'状态(状态 0 ~ 6，cr = '1')。计数器能否保持状态 0，取决于人工按键操作，若按键 key = '1'，控制 qen = '1'(计数器能正常工作)，key = '0'，计数器状态保持。显见计数器处于状态 0 时，人工不按键，则计数器保持状态 0。

2. 防抖动原理

设计数器处于状态 0 时，出现一次有效按键(key = '1')，则有 qen = '1'，计数器从状态 0 开始计数。若计数器处于状态 0 时，key = '0'(释放键)，而 qen = '1'保持不变，则计数器继续进行计数，直到状态 0 结束。计数器从状态 0 开始计数后，按键的'0'、'1'抖动不会改变 qen = '1'的值，于是抖动对计数器的干扰就被排除了。

3. 按键持续时间

如果按键时间超过计数器的 1 个而小于 2 个循环周期，则会输出 2 个单脉冲。若持续按键，则会间隔输出单脉冲。按键时间小于计数器的 1 个循环周期，计数器只能输出一个单脉冲。

4. 仿真波形图

本实例的仿真波形图如图 5-5 所示。由图可见，在 125 ns 区域，尽管按键 key 抖动，但不影响单脉冲正常输出。375 ns 区域，由于一个计数循环没有结束，尽管有按键作用，也不能改变正常计数状态。计数器进入状态 0，由于没有按键操作，状态 0 延续了一段时间。在 500 ns 区域，又有按键操作，计数器重新开始计数，又输出一个单脉冲。

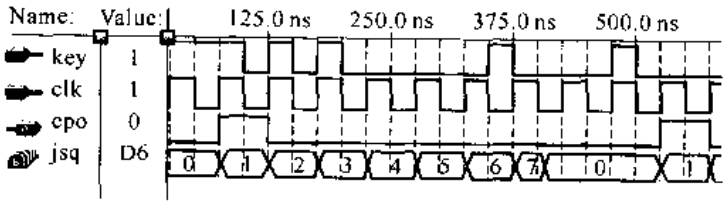


图 5-5 防抖动电路仿真波形图

实例 5-10 采样型防抖动微分电路

一、VHDL 源程序

1. 文件名：fd_rs.vhd

2. 源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY fd_rs IS
  PORT(clk,key: IN STD_LOGIC;
```

```
        dmc: OUT STD_LOGIC);
END ;
ARCHITECTURE a OF fd_rs IS
  SIGNAL r,s,qr,qs,d1,d2,q2,d3,d4,q4,cp:STD_LOGIC;
  BEGIN
k1: PROCESS(clk)
    BEGIN
  IF (clk'event AND clk = '1') THEN
      d1 <= key;      --存入输入数据。
      d2 <= d1 ;
      q2 <= d2 ;
      d3 <= qr ;
      d4 <= d3 ;
      q4 <= d4 ;
    END IF;
  r <= (NOT d2 ) AND (NOT q2);      --采样处理。
  s <= d2 AND q2;
  qr <= r NOR qs;      --构成 RS 触发器。
  qs <= s NOR qr;
  cp <= d4 AND (NOT q4);      --微分处理。
  dmc <= cp;
END PROCESS ;      --output
END;
```

二、源程序说明

1. 工作原理

本例描述的防抖动微分电路属于采样型防抖动电路。其原理结合图 5-6 说明。由机械开关引起的有抖动的脉冲数据输入采样型防抖动微分电路的第一级电路，即进入采样型防抖动电路。在此子电路中，令采样时钟周期大于欲滤去的窄脉冲宽度，经过采样，只有宽脉冲被识别，窄脉冲被滤去。

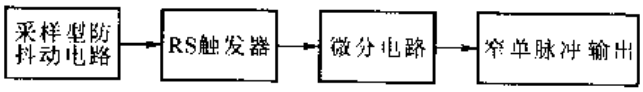


图 5-6　采样型防抖动微分电路框图

为了提高电路的防抖动能力，将采样防抖动电路输出信号输入 RS 触发器，进一步滤去可能由采样防抖动电路遗漏下来的相临"01"串，RS 触发器的输出只能是连续"1"或者连续"0"的宽脉冲(随输入脉冲信号的宽度不同而不同)。

一些数字系统要求使用与时钟同宽度的信号作同步控制信号，为了满足这类要求，再将

RS 触发器输出的宽脉冲输入微分电路(即宽脉冲变窄脉冲电路)进行处理,最终由微分电路输出需要的窄脉冲。

需要指出的是,若数字电路可以使用宽脉冲,则无需对其进行微分处理。

2. 仿真波形图

本实例的仿真波形图如图 5-7 所示。由图可见,输入的窄脉冲被滤去,输入信号中大约大于 2 个时钟周期宽度的宽脉冲才能被识别(识别标志是输出一个单脉冲)。

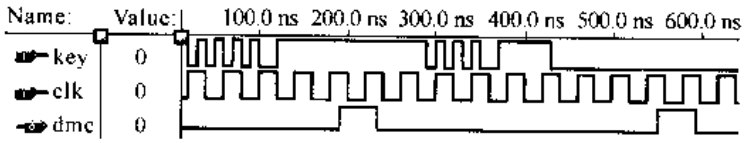


图 5-7 采样型防抖动微分电路仿真波形图

实例 5-11 积分分频器

分频器有两类,一类是脉冲波形均匀分布的分频器,即常规分频器;另一类是脉冲波形不均匀分布的分频器,即积分分频器。下面通过设计一个电路介绍积分分频器的基本概念。

设输入频率为 10 kHz 脉冲信号,要求得到 3 kHz 脉冲信号,且不容许有计算误差,要求设计一个实现电路。

分析:设用"1010101010"代表 5 kHz 信号中的一段信号。在同样长的时间内设法得到另一个脉冲串"1010100000"信号,设其'0'、'1'宽度与 5 kHz 信号的'0'、'1'宽度相同,显见,若能得到"0101010000"信号,则该信号的频率就是 3 kHz。用一个有 10 位输入端的选择器及其地址计数器构成分频器,地址计数器的时钟频率选为 10 kHz。

一、VHDL 源程序

1. 文件名:mux10.vhd

2. 源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY mux10 IS
    PORT(clk:IN STD_LOGIC;
         f:out STD_LOGIC);

END ;

ARCHITECTURE key OF mux10 IS
  SIGNAL q:STD_LOGIC_vector(3 downto 0);
  SIGNAL d:STD_LOGIC_vector(9 downto 0);

  BEGIN
P1: PROCESS(clk)        --进程描述选择器的地址计数器。
  BEGIN
```

```
  IF (clk'event and clk = '1') THEN
   IF q = "1001" THEN
      q <= "0000";
   ELSE
      q <= q + 1;
   END IF;
  END IF;
END PROCESS;
P2:PROCESS        --进程描述 10 选 1 选择器。
BEGIN
 IF q = "0000"     THEN
       f <= d(0);
 ELSIF q = "0001" THEN
       f <= d(1);
 ELSIF q = "0010" THEN
       f <= d(2);
 ELSIF q = "0011" THEN
       f <= d(3);
 ELSIF q = "0100" THEN
       f <= d(4);
 ELSIF q = "0101" THEN
       f <= d(5);
 ELSIF q = "0110" THEN
       f <= d(6);
 ELSIF q = "0111" THEN
       f <= d(7);
 ELSIF q = "1000" THEN
       f <= d(8);
 ELSIF q = "1001" THEN
       f <= d(9);
END IF;
END PROCESS;
P3:PROCESS        --进程设置规定的 0、1 脉冲串。
BEGIN
 d(0) <= '0';
 d(1) <= '0';
 d(2) <= '0';
 d(3) <= '0';
 d(4) <= '0';
```

```
    d(5) <= '1';
    d(6) <= '0';
    d(7) <= '1';
    d(8) <= '0';
    d(9) <= '1';
  END PROCESS;
  END ;
```

二、源程序说明

1. 本实例的仿真波形图如图 5-8 所示。图中,clk 的频率为 10 kHz,在 1 ms 时间段,有 10 个周期时钟脉冲,而分频器输出有 3 个周期脉冲,显见若用频率计测量(测量 1 s 内含有的脉冲个数),分频器输出信号 f 的频率应是 3 kHz。设计任务得到实现。

2. 在图 5-8 中,分频器输出信号 f 的波形分布是不均匀的,其特点是信号频率和 0、1 周期脉冲的个数相对应,3 个 0、1 周期脉冲表示 3 kHz。若信号 f 是以图中"0"、"1"宽度均匀分布,则 f 的频率是 5 kHz,含有 5 个 0、1 周期脉冲。由图 5-8 可以看出信号 f 的最高频率数呈现的规律就好像是所有 0、1 周期脉冲个数叠加的结果,这就是该分频器被称为"积分"分频器的来由。设计规律请读者结合本例自行总结。

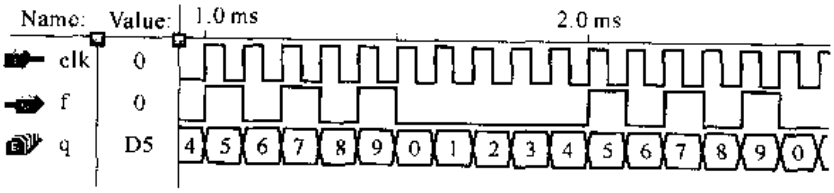


图 5-8 积分分频器仿真波形图

实例 5-12 4×4 键盘输入电路

由 4×4 键盘输入数据,是设计数字电路常用的有效手段。由于键盘按键是一种机械开关,所以设计其控制电路时,一个任务是正确译码;另一个任务是设计防抖动电路。

一、VHDL 源程序

1. 文件名:key_scan.vhd

2. 源程序

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY key_scan IS
    PORT(clk0,clk2,cr:IN STD_LOGIC;
                col:in STD_LOGIC_VECTOR (3 downto 0);
 a1,b1,c1,d1,e1,f1,g1:OUT STD_LOGIC;
                row:out STD_LOGIC_VECTOR (3 downto 0);
               p78,:OUT STD_LOGIC);
```

```
END ;

ARCHITECTURE a OF key_scan IS
  SIGNAL line : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL bcdmin,bcdmo: STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL clr : STD_LOGIC;
  BEGIN

p78 <= '1';

p1: PROCESS (clk0) - - p1 进程描述一个循环码计数器,计数器输出作键盘行扫描信号。
  BEGIN

  IF ( clr = '0' ) THEN
    line <= "1000";
  ELSIF (clk0'event and clk0 = '1') THEN
    line(3 downto 0) <= line(2 downto 0) & line(3);      - - xhm
  END IF;
END PROCESS p1;
  row(3) <= line(3);      - - 行扫描信号输出到端口,以驱动键盘行线。
  row(2) <= line(2);
  row(1) <= line(1);
  row(0) <= line(0);

p2: PROCESS      - - p2 进程描述键盘输入信号的译码过程。
  BEGIN
IF( line = "1000" and col = "1000")      THEN
  bcdmin <= "1111";
ELSIF( line = "1000" and col = "0100" ) THEN
  bcdmin <= "0001";
ELSIF( line = "1000" and col = "0010") THEN
  bcdmin <= "0010";
ELSIF(line = "1000" and col = "0001") THEN
  bcdmin <= "0011";
ELSIF( line = "0100" and col = "1000") THEN
  bcdmin <= "0100";
ELSIF( line = "0100" and col = "0100") THEN
  bcdmin <= "0101";
ELSIF( line = "0100" and col = "0010") THEN
  bcdmin <= "0110";
ELSIF( line = "0100" and col = "0001") THEN
  bcdmin <= "0111";
```

```
ELSIF(line = "0010" and col = "1000")   THEN
  bcdmin <= "1000";
ELSIF( line = "0010" and col = "0100")   THEN
  bcdmin <= "1001";
ELSE
  bcdmin <= "0000";
END IF;
END PROCESS p2;

p3: PROCESS (clk2)          --p3 进程描述把按键信号的译码输出寄存到寄存器。
  BEGIN
  IF(clk2'event AND clk2 = '1') THEN
    IF(col(0) OR col(1) OR col(2) OR col(3)) = '1' THEN
      bcdmo <= bcdmin;      --关键步骤,若有按键,第一次出现的正确译码被寄存
                            --器存储。
    ELSE
      bcdmo <= bcdmo;       --如按键有抖动,则寄存器中数据保持原状态,已写入数
                            --据不受干扰。
    END IF;
  END IF;
END PROCESS p3;

p4:PROCESS
BEGIN

  IF(bcdmo = "1111") THEN--将寄存器输出的 4 位码转换为驱动七段数码管的代码。
     a1 <= '1';
     b1 <= '1';
     c1 <= '1';
     d1 <= '1';
     e1 <= '1';
     f1 <= '1';
     g1 <= '0';

  ELSIF(bcdmo = "0001" ) THEN
     a1 <= '0';
     b1 <= '1';
     c1 <= '1';
     d1 <= '0';
     e1 <= '0';
     f1 <= '0';
     g1 <= '0';
```

```
ELSIF(bcdmo = "0010") THEN
    a1 <= '1';
    b1 <= '1';
    c1 <= '0';
    d1 <= '1';
    e1 <= '1';
    f1 <= '0';
    g1 <= '1';
ELSIF(bcdmo = "0011") THEN
    a1 <= '1';
    b1 <= '1';
    c1 <= '1';
    d1 <= '1';
    e1 <= '0';
    f1 <= '0';
    g1 <= '1';
ELSIF( bcdmo = "0100") THEN
    a1 <= '0';
    b1 <= '1';
    c1 <= '1';
    d1 <= '0';
    e1 <= '0';
    f1 <= '1';
    g1 <= '1';
ELSIF( bcdmo = "0101") THEN
    a1 <= '1';
    b1 <= '0';
    c1 <= '1';
    d1 <= '1';
    e1 <= '0';
    f1 <= '1';
    g1 <= '1';
ELSIF( bcdmo = "0110") THEN
    a1 <= '1';
    b1 <= '0';
    c1 <= '1';
    d1 <= '1';
    e1 <= '1';
```

```
        f1 <= '1';
        g1 <= '1';
    ELSIF( bcdmo = "0111") THEN
        a1 <= '1';
        b1 <= '1';
        c1 <= '1';
        d1 <= '0';
        e1 <= '0';
        f1 <= '0';
        g1 <= '0';
    ELSIF( bcdmo = "1000") THEN
        a1 <= '1';
        b1 <= '1';
        c1 <= '1';
        d1 <= '1';
        e1 <= '1';
        f1 <= '1';
        g1 <= '1';
    ELSIF(bcdmo = "1001") THEN
        a1 <= '1';
        b1 <= '1';
        c1 <= '1';
        d1 <= '1';
        e1 <= '0';
        f1 <= '1';
        g1 <= '1';
    ELSE
        a1 <= '0';
        b1 <= '0';
        c1 <= '0';
        d1 <= '0';
        e1 <= '0';
        f1 <= '0';
        g1 <= '0';
END IF;
END PROCESS;
p5: PROCESS(cr)        --用一个端口按键开关控制扫描计数器的初始化。
```

```
BEGIN
  IF (cr'event and cr = '1') THEN
    clr <= NOT clr;
  END IF;
END PROCESS p5;

END;
```

二、源程序说明

1. 本例描述一个键盘扫描电路，使用者利用 4×4 键盘输入 BCD 码数据。输入数据前，应对电路清零，做电路的初始化工作。描述的电路外接一个七段数码管，按键时相应数据就显示在数码管上。

2. 结合本例框图对键盘描述电路在工作状态作进一步介绍。图 5-9 是本例的内外结构框图。虚线内部是芯片内的电路。键盘的行(row)、列(col)数据输入译码器，译码器输出传入寄存器存储。寄存器输出传入 4/7 线译码器译码，其输出驱动数码管显示数。用循环码计数器(xhm)产生 4 位扫描信号，用以驱动键盘的行线。本例使用的键盘是一种无极性压力变阻开关，不按键相当断路；按键，开关呈现低阻。设行线输出高电平，按键，则高电平输入列端口。注意，为了保证不按键时，列端口是低电平，电路使用时，需要在地和 4 个列端口之间分别接一个电阻。图中的 C 是程序中的控制信号，C = COL(0) + COL(1) + COL(2) + COL(3)。其他符号请参阅程序。

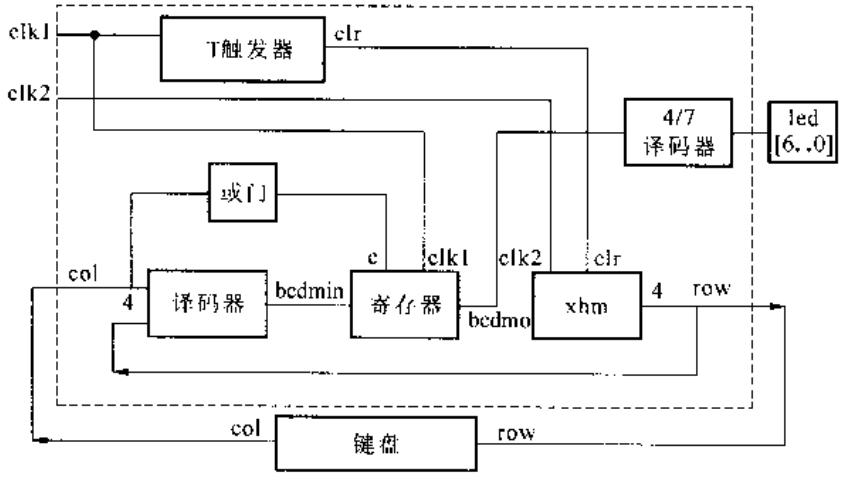


图 5-9　键盘扫描电路总体框图

3. 图 5-10 是本例仿真波形图。图中，row 信号是循环码"1000"、"0100"、"0010"、"0001"，列信号 col 置为 8("1000")，按程序描述，row = "1000" 和 col = "1000"，输出数据应为 0。图中显示七段译码的值为 126，即是十进制数 0。显见键盘输入正确。其他值对应情况请读者结合程序自行分析。

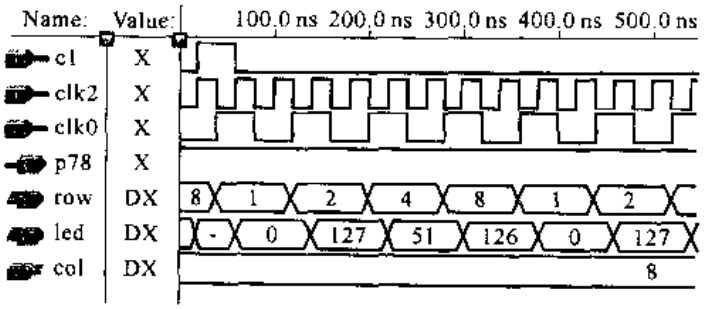


图 5-10 键盘扫描电路仿真波形图

实例 5-13 串行偶校验器

为检验传输代码可能产生的一位差错，常用奇偶校验电路。一般来说，人们对并行奇偶校验电路较熟，而对串行奇偶校验电路比较生疏。本例介绍串行偶校验电路。

一、VHDL 源程序

1. 文件名：parity.vhd

2. 源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity parity is
  port(clk,din,clr: in bit ;
                   q: out bit);

end ;
architecture a of parity is
  signal k :bit;
  begin
p1: process (clk)
    begin
    if clr='0' then
        k<='1';
    elsif (clk'event and clk='1') then
        k<=k xor din;
    end if;
        q<=k;
end process ;
end ;
```

二、源程序说明

1. “异或”门是构成奇偶校验器的主要器件，可以作为通用模块使用。

2. 串行偶校验器在工作前，其输出应预置为'1'；若是奇校验，校验器的输出应预置为'0'。串行偶校验器端口图如图 5-11 所示。

图 5-11　串行偶校验器端口图

3. 要求被校验序列与时钟同步。

4. 本程序仿真波形图如图 5-12 所示。

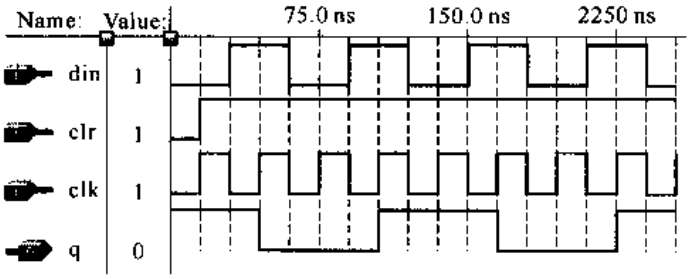


图 5-12　串行偶校验器仿真波形图

第六章　数字系统课题

前面几章已详细地介绍了基本数字电路的 VHDL 描述，为进一步掌握 VHDL 的综合应用，本章提供几个数字系统综合课题供读者进行分析、练习、修改与扩展。为了提高分析、实践效果，本章各例只提供简单的课题总体框图与注释作为提示。

课题 6-1　带数字显示的秒表

一、设计任务及要求

1. 设计一块用数码管显示的秒表。
2. 能够准确地计时并显示。
3. 开机显示 00.00.00。
4. 用户可随时清零、暂停、计时。
5. 最大计时 59 分钟，最小精确到 0.01 秒。

二、可选器件

EPM7128S、共阴极七段数码管、发光二级管、按键开关、电阻、电容。

三、设计总体框图

数字显示的秒表总体框图如图 6-1 所示。

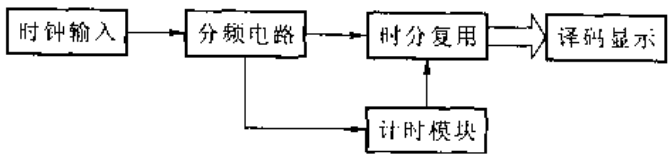


图 6-1　带数字显示的秒表总体框图

四、源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity watch is
port(sel:out std_logic_vector(6 downto 1);        --片选信号。
seg:out std_logic_vector(7 downto 0);        --数码管的输出信号。
beginstop:in std_logic;        --开始、停止计数。
reset:in std_logic;        --复位。
cp2:in std_logic);
end watch;
architecture behave of watch is
```

```
signal num1:std_logic_vector(3 downto 0);
signal num2:std_logic_vector(3 downto 0);
signal num3:std_logic_vector(3 downto 0);
signal num4:std_logic_vector(3 downto 0);
signal num5:std_logic_vector(3 downto 0);
signal num6:std_logic_vector(3 downto 0);
signal num:std_logic_vector(3 downto 0);
signal numlet:std_logic_vector(2 downto 0);
signal count:std_logic_vector(17 downto 1);
signal selsig:std_logic_vector(6 downto 1);
signal segsig:std_logic_vector(7 downto 0);
signal cp1: std_logic;
signal cp3: std_logic;
  begin
    process(cp2)        --分频。
      begin
        if (cp2'event and cp2 = '1')then
          if (count = "11000011010011111")then
            count <= "00000000000000000"; cp1 <= not cp1;
          else count <= count + 1;
          end if;
        end if;
          cp3 <= count(10);
end process;
process (cp1)        --计数控制。
  begin
    if reset = '1' then num1(3 downto 0) <= "0000";
                        num2(3 downto 0) <= "0000";
                        num3(3 downto 0) <= "0000";
                        num4(3 downto 0) <= "0000";
                        num5(3 downto 0) <= "0000";
                        num6(3 downto 0) <= "0000";
      else if cp1'event and cp1 = '1' then
    if beginstop = '1'  then num1 <= num1 + 1;
    if num1(3 downto 0) = "1001" then
      num1(3 downto 0) <= "0000"; num2 <= num2 + 1;
    if num2(3 downto 0) = "1001" then
      num2(3 downto 0) <= "0000"; num3 <= num3 + 1;
    if num3(3 downto 0) = "1001" then
```

```
        num3(3 downto 0) <= "0000"; num4 <= num4 + 1;
      if num4(3 downto 0) = "0101" then
        num4(3 downto 0) <= "0000"; num5 <= num5 + 1;
      if num5(3 downto 0) = "1001" then
        num5(3 downto 0) <= "0000"; num6 <= num6 + 1;
      if num6(3 downto 0) = "0101" then
        num6(3 downto 0) <= "0000";
      end if;end if;end if;end if;end if; end if;end if;
      end if;end if;
      end process;
  process (cp3)       - - 显示控制。
    begin
      if (cp3'event and cp3 = '1') then
        if (numlet(2 downto 0) = "000") then
          num <= num1;selsig(6 downto 1) <= "111110"; end if;
        if (numlet(2 downto 0) = "001") then
          num <= num2;selsig(6 downto 1) <= "111101"; end if;
        if (numlet(2 downto 0) = "010") then
          num <= num3;selsig(6 downto 1) <= "111011"; end if;
        if (numlet(2 downto 0) = "011") then
          num <= num4;selsig(6 downto 1) <= "110111"; end if;
        if (numlet(2 downto 0) = "100") then
          num <= num5;selsig(6 downto 1) <= "101111"; end if;
          numlet(2 downto 0) <= numlet(2 downto 0) + 1;
        if (numlet(2 downto 0) = "101") then
          numlet(2 downto 0) <= "000";num <= num6;
       selsig(6 downto 1) <= "011111";end if;
  end if;
  if (num(3 downto 0) = "0000") then
    segsig(7 downto 0) <= "01111111"; end if;
  if (num(3 downto 0) = "0001") then
    segsig(7 downto 0) <= "00001101"; end if;
  if (num(3 downto 0) = "0010") then
    segsig(7 downto 0) <= "10110111"; end if;
  if (num(3 downto 0) = "0011") then
  segsig(7 downto 0) <= "10011111"; end if;
  if (num(3 downto 0) = "0100") then
    segsig(7 downto 0) <= "11001101"; end if;
  if (num(3 downto 0) = "0101") then
```

```
    segsig(7 downto 0) <= "11011011"; end if;
  if (num(3 downto 0) = "0110") then
    segsig(7 downto 0) <= "11111011"; end if;
  if (num(3 downto 0) = "0111") then
    segsig(7 downto 0) <= "00001111"; end if;
  if (num(3 downto 0) = "1000") then
    segsig(7 downto 0) <= "11111111"; end if;
  if (num(3 downto 0) = "1001") then
    segsig(7 downto 0) <= "11011111"; end if;
  end process;
    sel <= selsig;
  seg(7 downto 0) <= segsig(7 downto 0);
  end behave;
```

五、程序说明

1. 此程序包括分频、计数控制、显示控制部分。

2. 开机显示 00.00.00,用户可随时计时、暂停、清零,最大计时可到 59 分 59.99 秒。

3. 计数时钟为 100 Hz。

4. 采用时分复用的方法控制 4 个数码管的显示,可节省资源。

课题 6-2　8×8 发光点阵逐点扫描显示装置

一、设计任务及要求

1. 使用 8×8 矩阵显示屏设计一个扫描控制电路。

2. 光点从屏左上角像素点开始逐点扫描,终止于右下角像素点,然后周而复始地重复运行下去。

3. 扫描一帧所需时间为 13 s。

二、可选器件

EPM7128S、8×8 矩阵显示屏、电阻。

三、设计总体框图

8×8 发光点阵逐点扫描显示装置总体框图如图 6-2 所示。

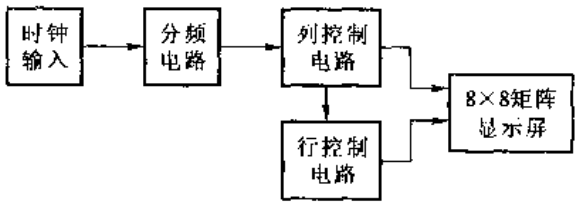


图 6-2　8×8 发光点阵逐点扫描显示装置总体框图

四、源程序

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_signed.all;
entity zlx2 is
    port(
      clk : in std_logic;
       q: out std_logic_vector(21 downto 0);      --分频电路输出。
       t: out std_logic_vector(0 to 7);           --列选通。
       y: out std_logic_vector(0 to 7));          --行选通。
end zlx2;
architecture a of zlx2 is
  signal b: std_logic_vector(21 downto 0 ):="0000000000000000000000";
  signal a: std_logic_vector(0 to 2);
  signal e: std_logic_vector(0 to 2);
  signal c: std_logic_vector(0 to 7);
  signal d: std_logic_vector(0 to 7);
begin
process (clk)      --分频电路。
begin
if (clk'event and clk = '1') then
b(21 downto 0) <= b(21 downto 0) + 1;
  end if;
  q <= b;          --输出扫描频率。
end process;
process            --以下两个进程控制行、列选通线实现点扫描。
begin
a(0) <= b(16);
a(1) <= b(17);
a(2) <= b(18);
        if   a="000" then c <="10000000";
        elsif a="001" then c <="01000000";
        elsif a="010" then c <="00100000";
        elsif a="011" then c <="00010000";
        elsif a="100" then c <="00001000";
        elsif a="101" then c <="00000100";
        elsif a="110" then c <="00000010";
        elsif a="111" then c <="00000001";
end if;
  y <= c;
end process;
process
```

```
begin
e(0) <= b(19);
e(1) <= b(20);
e(2) <= b(21);
        if    e = "000" then d <= "01111111";
        elsif e = "001" then d <= "10111111";
        elsif e = "010" then d <= "11011111";
        elsif e = "011" then d <= "11101111";
        elsif e = "100" then d <= "11110111";
        elsif e = "101" then d <= "11111011";
        elsif e = "110" then d <= "11111101";
        elsif e = "111" then d <= "11111110";
end if;
    t <= d;
end process;
end a;
```

五、程序说明

1. 程序主要由分频电路和扫描控制电路组成。
2. 扫描控制电路可用 3 线-8 线译码器的设计思路来实现。

课题 6-3　彩灯闪烁装置

一、设计任务及要求

1. 使用 8×8 矩阵显示屏设计一个彩灯闪烁装置。
2. 第一帧以 1 个光点为 1 个像素点从屏左上角开始逐点扫描，终止于右下角。
   第二帧以 2 个光点为 1 个像素点从屏左上角开始逐点扫描，终止于右下角。
   第三帧重复第一帧，第四帧重复第二帧，周而复始地重复运行下去。

二、可选器件

EPM7128S、8×8 矩阵显示屏、电阻。

三、设计总体框图

彩灯闪烁装置总体框图如图 6-3 所示。

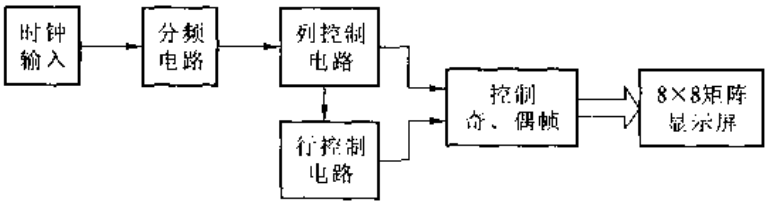


图 6-3　彩灯闪烁装置总体框图

四、源程序

```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;          --for + operation。

entity scan2 is
  port(cp1: in std_logic;          --输入时钟 20MHz。
       ho: out std_logic_vector(7 downto 0);          --行控制信号。
       vo: out std_logic_vector(7 downto 0));          --列控制信号。
end scan2;

architecture scanpixel of scan2 is
signal count1:std_logic_vector(2 downto 0):="000";          --扫描哪一个点的计数器。
signal count2:std_logic_vector(3 downto 0):="0000";          --扫描哪一行的计数器。
signal v:std_logic_vector(7 downto 0):="10000000";
signal h:std_logic_vector(7 downto 0):="10000000";          --输出信号。
signal cp2:std_logic;
signal cp:std_logic;
signal cnt:std_logic_vector(15 downto 0):="0000000000000000";
signal cnt2:std_logic_vector(5 downto 0):="000000";          --分频用的计数器。
begin

process(cp1)
begin
  if(cp1'event and cp1='1') then
    cnt<=cnt+1;
  end if;
  cp2<=cnt(15);
end process;

process(cp2)
begin
  if(cp2'event and cp2='1') then
    cnt2<=cnt2+1;
  end if;
  cp<=cnt2(5);
end process;

process(cp)
begin
  if(cp'event and cp='1') then
    case count2(2 downto 0) is          --根据 count2 输出行控制信号。
      when "000"=> h<="10000000";
      when "001"=> h<="01000000";
```

```
        when "010" => h <= "00100000";
        when "011" => h <= "00010000";
        when "100" => h <= "00001000";
        when "101" => h <= "00000100";
        when "110" => h <= "00000010";
        when "111" => h <= "00000001";
        when others => h <= "00000000";
      end case;
      if count2 <= "0111" then      --如果扫描的是前8行,则逐点扫描。
        count1 <= count1 + 1;       --扫描下一点。
        case count1(2 downto 0) is      --根据count1输出列控制信号。
          when "000" => v <= "01111111";
          when "001" => v <= "10111111";
          when "010" => v <= "11011111";
          when "011" => v <= "11101111";
          when "100" => v <= "11110111";
          when "101" => v <= "11111011";
          when "110" => v <= "11111101";
          when "111" => v <= "11111110";
          when others => v <= "11111111";
        end case;
      if count1 = "111" then count2 <= count2 + 1;end if; --扫描完一行,行计数器加1。
      else                                              --若扫描的是后8行,则2点扫。
        if count1 = "010" then count1 <= "111"; else count1 <= count1 + 1; end if;
        --使count1清零,准备下次循环。
        case count1(1 downto 0) is      --根据count1输出列控制信号。
          when "00" => v <= "00111111";
          when "01" => v <= "11001111";
          when "10" => v <= "11110011";
          when "11" => v <= "11111100";
          when others => v <= "11111111";
        end case;
        if count1(1 downto 0) = "11" then count2 <= count2 + 1; end if;
        --扫描完一行,行计数器加1。
      end if;
    end if;
    ho <= h;
    vo <= v;
  end process;
```

```
end scanpixel;
```

五、程序说明

此程序主要由分频电路,行、列控制电路组成。

课题 6-4 抢 答 器

一、设计任务及要求

设计一个 2 人抢答器。要求如下:

1. 两人抢答,先抢为有效,用发光二级管显示是否抢到优先答题权。
2. 每人 2 位计分显示,答错了不加分,答对了可加 10 分、20 分、30 分。
3. 每题结束后,裁判按复位,可重新抢答下一题。
4. 累计加分可由裁判随时清除。

二、可选器件

EPM7128S、共阴极七段数码管、发光二级管、按键开关、电阻、电容。

三、设计总体框图

抢答器总体框图如图 6-4 所示。

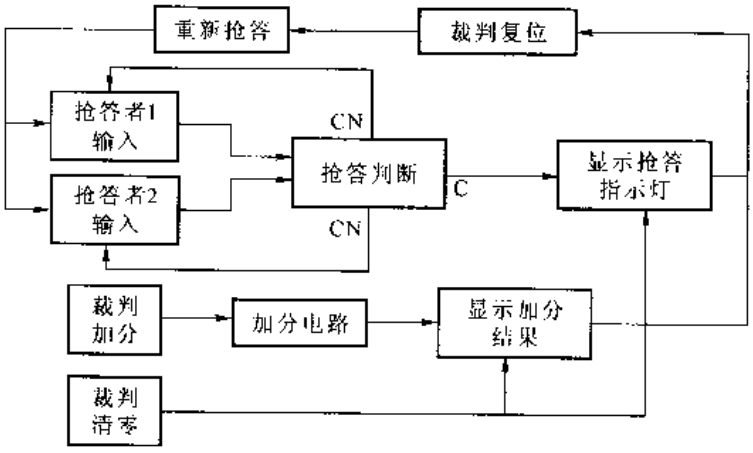


图 6-4 抢答器总体框图

四、源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity qd is
    port(i1,i2 : in bit;        --抢答器两输入端。
      reset : in bit;           --抢答器复位端。
      g10,g20,g30 : in bit;     --加分输入端 10 分、20 分、30 分。
      light1,light2: out bit := '0';    --抢答标志灯。
      t11 : out std_logic_vector(6 downto 0): = "0111111";    --"t11、t12",显示第一个
                                                              --人得分。
      t12 : out std_logic_vector(6 downto 0): = "0111111";
```

```
        t21 : out std_logic_vector(6 downto 0): = "0111111"; - -"t21、t22"显示第二个人得分。
        t22 : out std_logic_vector(6 downto 0): = "0111111";
        clk : in bit;        - -时钟输入端。
        cong: in bit         - -清零控制端。
        );
end qd;
architecture stru of qd is
signal cs1: integer range 0 to 9;
signal cs2: integer range 0 to 9;
signal a,b: bit := '0';
signal l1,l2: bit := '0';
begin
process(clk)
begin
if clk'event and clk = '1' then
  if (cong = '1') then
  if (reset = '1') then
      if (i1 = '0' and a = '0' ) then l1 <= '1';a <= '1';        - -抢答。
      elsif (i2 = '0' and a = '0' ) then l2 <= '1';a <= '1';
    end if;
      if (g10 = '0' and l1 = '1' and b = '0') then cs1 <= cs1 + 1;b <= '1';
     elsif (g20 = '0' and l1 = '1' and b = '0') then cs1 <= cs1 + 2;b <= '1';
     elsif (g30 = '0' and l1 = '1' and b = '0') then cs1 <= cs1 + 3;b <= '1';
     end if;        - -完成第一人的加分。
      if  (g10 = '0' and l2 = '1' and b = '0') then cs2 <= cs2 + 1;b <= '1';
     elsif (g20 = '0' and l2 = '1' and b = '0') then cs2 <= cs2 + 2;b <= '1';
     elsif (g30 = '0' and l2 = '1' and b = '0') then cs2 <= cs2 + 3;b <= '1';
     end if;        - -完成第二人的加分。
      if (cs1 = 0 ) then t11 <= "0111111";        - -显示。
     elsif (cs1 = 1 ) then t11 <= "0000110";
     elsif (cs1 = 2 ) then t11 <= "1011011";
     elsif (cs1 = 3 ) then t11 <= "1001111";
     elsif (cs1 = 4 ) then t11 <= "1100110";
     elsif (cs1 = 5 ) then t11 <= "1101101";
     elsif (cs1 = 6 ) then t11 <= "1111101";
     elsif (cs1 = 7 ) then t11 <= "0000111";
     elsif (cs1 = 8 ) then t11 <= "1111111";
     elsif (cs1 = 9 ) then t11 <= "1101111";
```

```
        end if;
          if (cs2 = 0 ) then t21 <= "0111111";
        elsif (cs2 = 1 ) then t21 <= "0000110";
        elsif (cs2 = 2 ) then t21 <= "1011011";
        elsif (cs2 = 3 ) then t21 <= "1001111";
        elsif (cs2 = 4 ) then t21 <= "1100110";
        elsif (cs2 = 5 ) then t21 <= "1101101";
        elsif (cs2 = 6 ) then t21 <= "1111101";
        elsif (cs2 = 7 ) then t21 <= "0000111";
        elsif (cs2 = 8 ) then t21 <= "1111111";
        elsif (cs2 = 9 ) then t21 <= "1101111";
        end if;
    else l1 <= '0'; l2 <= '0'; a <= '0'; b <= '0';
    end if;
    else cs1 <= 0; cs2 <= 0;
         l1 <= '0'; l2 <= '0';
    end if;
  light1 <= l1;
  light2 <= l2;
  t12 <= "0111111";
  t22 <= "0111111";
  end if;
  end process;
  end stru;
```

五、程序说明

1. 此程序主要由 3 部分组成，即抢答、加分、显示。

2. 当一人抢到优先答题权，发光二极管亮，另一人再按按键无效。答题结束后，裁判按复位键，方可再次抢答。

3. 每人有 2 个数码管显示累加计分情况，分数分为 3 档，用按键来区别。

课题 6-5 密 码 锁

一、设计任务及要求

设计一个 2 位的密码锁。要求如下：

1. 开锁代码为 2 位十进制并行码。

2. 当输入的密码与锁内的密码一致时，绿灯亮，开锁；当输入的密码与锁内的密码不一致时，红灯亮，不能开锁。

3. 密码可由用户自行设置。

二、可选器件

EPM7128S、共阴极七段数码管、发光二级管、按键开关、电阻、电容。

三、设计总体框图

密码锁总体框图如图 6-5 所示。

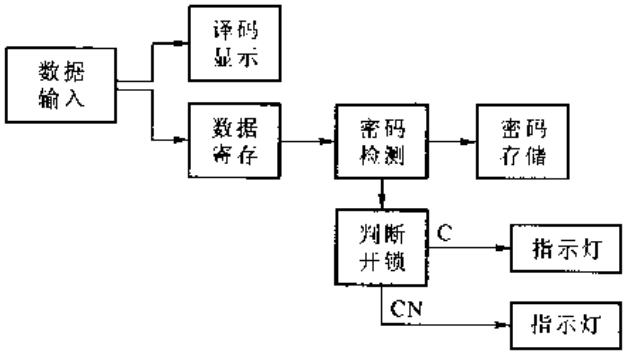


图 6-5　密码锁总体框图

四、源程序

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity lock is
port(a: in std_logic_vector(9 downto 0);        --十位输入端。
     b: in std_logic_vector(9 downto 0);        --个位输入端。
en,clk: in std_logic;          --en为密码确认开关。
   en1: in std_logic;          --en1为密码检验开关。
 c ,d: out std_logic_vector(6 downto 0):="0111111"; --输出七段数码管的控制。
 k,m: out std_logic);          --k绿灯,m红灯。
end lock;
architecture behave of lock is
signal e : std_logic_vector(3 downto 0);
signal f : std_logic_vector(3 downto 0);
signal g : std_logic_vector(3 downto 0);
signal h : std_logic_vector(3 downto 0);
signal count1,count2: std_logic := '0';
signal s : std_logic := '0';
begin
process(a,e)          --十位的数据转换。
begin
if (a(9 downto 0) = "0000000000")
then c(6 downto 0) <= "0111111";
elsif (a(0) = '1' ) then
e <= "0000"; c(6 downto 0) <= "0111111";
elsif (a(1) = '1' ) then
```

```
e <= "0001"; c(6 downto 0) <= "0000110";
elsif (a(2) = '1' ) then
e <= "0010"; c(6 downto 0) <= "1011011";
elsif (a(3) = '1' ) then
e <= "0011"; c(6 downto 0) <= "1001111";
elsif (a(4) = '1' ) then
e <= "0100"; c(6 downto 0) <= "1100110";
elsif (a(5) = '1' ) then
e <= "0101"; c(6 downto 0) <= "1101101";
elsif (a(6) = '1' ) then
e <= "0110"; c(6 downto 0) <= "1111100";
elsif (a(7) = '1' ) then
e <= "0111"; c(6 downto 0) <= "0000111";
elsif (a(8) = '1' ) then
e <= "1000"; c(6 downto 0) <= "1111111";
elsif (a(9) = '1' ) then
e <= "1001"; c(6 downto 0) <= "1100111";
end if;
end process;
process(b,f)        --个位的数据转换。
begin
if (b(9 downto 0) = "0000000000") then
d(6 downto 0) <= "0111111";
elsif (b(0) = '1' ) then
f <= "0000"; d(6 downto 0) <= "0111111";
elsif (b(1) = '1' ) then
f <= "0001"; d(6 downto 0) <= "0000110";
elsif (b(2) = '1' ) then
f <= "0010"; d(6 downto 0) <= "1011011";
elsif (b(3) = '1' ) then
f <= "0011"; d(6 downto 0) <= "1001111";
elsif (b(4) = '1' ) then
f <= "0100"; d(6 downto 0) <= "1100110";
elsif (b(5) = '1' ) then
f <= "0101"; d(6 downto 0) <= "1101101";
elsif (b(6) = '1' ) then
f <= "0110"; d(6 downto 0) <= "1111100";
elsif (b(7) = '1' ) then
f <= "0111"; d(6 downto 0) <= "0000111";
```

```
elsif (b(8) = '1' ) then
f <= "1000"; d(6 downto 0) <= "1111111";
elsif (b(9) = '1' ) then
f <= "1001"; d(6 downto 0) <= "1100111";
end if;
end process;
process(clk)        --判断密码是否正确,可否开锁。
begin
if (s = '1') and (en = '1') and (count1 = '0') then
g(3 downto 0) <= e(3 downto 0);
h(3 downto 0) <= f(3 downto 0);
count1 <= not(count1);
end if;
if ( en1 = '1') and( count2 = '0' ) then
if (e(3 downto 0) = g(3 downto 0) and f(3 downto 0) = h(3 downto 0)) then
k <= '1';
s <= '1';
else
k <= '0';
m <= '1';
end if;
count2 <= not(count2);
end if;
if ( en = '0') and (en1 = '0') then
count1 <= '0';
count2 <= '0';
k <= '0'; s <= '0'; m <= '0';
end if;
if ( en = '0' and s = '1' ) then
count1 <= '0';
end if;
end process;
end behave;
```

五、程序说明

1. 此程序由解码、设码、数码管显示等部分组成。

2. 此程序是一个并行密码锁,用户开锁密码为 00。当使用开锁密码后,指示灯亮(绿灯),表示锁被打开。用户可自行设置密码。

3. 用户可用 2 个 DIP 开关(1 ~ 10)设置 0 ~ 99 的 2 位十进制数的密码。

课题 6-6 数字频率计

一、设计任务及要求

设计一个 4 位十进制数字显示的数字式频率计。要求如下：

1. 4 位十进制数字显示的数字式频率计，其频率测量范围为 10 ~ 9 999 kHz，测量单位为 kHz。

2. 要求量程能够自动转换。（即测几十到几百千赫兹(kHz)时，有小数点显示，前者显示小数点后 2 位，后者显示小数点后 1 位。）

3. 当输入的信号小于 10 kHz 时，输出显示全 0；当输入的信号大于 9999 kHz 时，输出显示全 H。

二、可选器件

EPM7128S、共阴极七段数码管、按键开关、电阻、电容。

三、设计总体框图

数字频率计总体框图如图 6-6 所示。

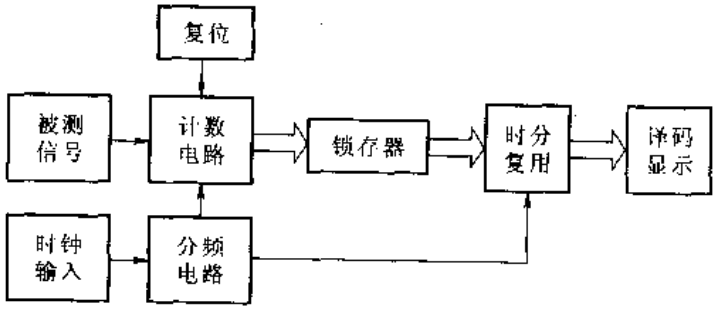


图 6-6 数字频率计总体框图

四、源程序

上层模块

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity total is
port (cp_20m:in std_logic;          --20 MHz 时钟信号。
      enable: in std_logic;         --开关信号。
       input: in std_logic;         --输入被测信号。
       reset: in std_logic;         --复位信号。
         sel: out std_logic_vector( 3 downto 0);      --输出片选信号。
        show: out std_logic_vector(6 downto 0);       --输出七位数码管显示。
     decimal: out std_logic_vector(2 downto 0));      --输出小数点。
end total;
architecture content of total is
signal cp_1,cp_2,cp_3:std_logic;
```

```
signal reset_1:std_logic;
signal overflow_1,low_1:std_logic;
signal play0_1,play1_1,play2_1,play3_1:integer range 0 to 9;
signal overlatch_1,lowlatch_1:std_logic;
signal p0latch_1,p1latch_1,p2latch_1,p3latch_1:integer range 0 to 9;
signal decimal_1:std_logic_vector(2 downto 0);
component dividefre4 is
port(cp_20m:in std_logic;
        cp1:out std_logic;
        cp2:out std_logic;
        cp3:out std_logic );
end component;

component debounce is
port(key:in std_logic;
     cp:in std_logic;
    imp:out std_logic);
end component;

component fretest is
port (enable,cp3,input,reset :in std_logic;
      overflow,low:out std_logic;
      play0,play1,play2,play3: out integer range 0 to 9;
      decimal: out std_logic_vector (2 downto 0)
      );
end component;

component frelatch is
port ( reset: in std_logic;
       cp3:in std_logic;
       overflow:in std_logic;
       low:in std_logic;
       play0,play1,play2,play3: in integer range 0 to 9;
       decimal: in std_logic_vector(2 downto 0);
       overlatch: out std_logic;
       lowlatch: out std_logic;
       p0latch,p1latch,p2latch,p3latch: out integer range 0 to 9;
       delatch:out std_logic_vector(2 downto 0)
       );
end component;

component display is
```

```
port( cp1:in std_logic;
      low:in std_logic;
      overflow:in std_logic;
      p0,p1,p2,p3:in integer range 0 to 9;
      show:out std_logic_vector(6 downto 0);
      sel: out std_logic_vector(3 downto 0)
      );
end component;

begin
u1: dividefre4 port map(cp_20m,cp_1,cp_2,cp_3);
u2: debounce port map(reset,cp_2,reset_1);
u3: fretest port map(enable,cp_3,input,reset_1,overflow_1,low_1,
                     play0_1,play1_1,play2_1,play3_1,decimal_1);
u4: frelatch port map(reset_1,cp_3,overflow_1,low_1,play0_1,play1_1,
                      play2_1,play3_1,decimal_1,overlatch_1,lowlatch_1,
                      p0latch_1,p1latch_1,p2latch_1,p3latch_1,decimal);
u5: display port map (cp_1, lowlatch_1,overlatch_1,p0latch_1,p1latch_1,
                      p2latch_1,p3latch_1,show,sel);
end content;
```

下层模块

防抖电路

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity debounce is
  port(key,cp: in std_logic;
       imp: out std_logic);
end debounce;

architecture base of debounce is
  signal q1,q2 : std_logic;
begin
    process (cp)
    begin
        if cp'event and cp='1' then
          q2 <= q1;
          q1 <= key;
        end if;
```

```
    end process;
    imp <= q1 and not q2;
end base;
```

显示模块

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity display is
port( cp1:in std_logic;          --200 Hz 信号。
      low:in std_logic;          --输入小于 10 kHz 的信号。
      overflow:in std_logic;     --输入大于 9 999 kHz 的信号。
      p0,p1,p2,p3:in integer range 0 to 9;     --输入信号。
      show:out std_logic_vector(6 downto 0);   --七位数码管输出信号。
      sel: out std_logic_vector(3 downto 0)    --片选信号。
      );
end display;

architecture behavior of display is
signal count: integer range 0 to 3;
signal sel_1: std_logic_vector(3 downto 0);
begin
  process(cp1)
    begin
      if(cp1'event and cp1 = '1') then
        if count = 3 then count <= 0;
          else
          count <= count + 1;
        end if;
      end if;
  end process;

process(count)
begin
case count is
    when 0 => sel_1 <= "1110";
    when 1 => sel_1 <= "1101";
    when 2 => sel_1 <= "1011";
    when 3 => sel_1 <= "0111";
end case;
end process;
```

```
process(low,overflow)
begin
    if(low = '1') then        --输入小于 10 kHz 的信号时,输出全'0'。
  show <= "1111110";
    elsif (overflow = '1') then        --输入大于 9 999 kHz 的信号时,输出全'H'。
show <= "0110111";
    elsif (sel_1(0) = '0') then
  case p0 is
    when 0 => show <= "1111110";
    when 1 => show <= "0110000";
    when 2 => show <= "1101101";
    when 3 => show <= "1111001";
    when 4 => show <= "0110011";
    when 5 => show <= "1011011";
    when 6 => show <= "0011111";
    when 7 => show <= "1110000";
    when 8 => show <= "1111111";
    when 9 => show <= "1110011";
  end case;
  elsif (sel_1(1) = '0') then
  case p1 is
    when 0 => show <= "1111110";
    when 1 => show <= "0110000";
    when 2 => show <= "1101101";
    when 3 => show <= "1111001";
    when 4 => show <= "0110011";
    when 5 => show <= "1011011";
    when 6 => show <= "0011111";
    when 7 => show <= "1110000";
    when 8 => show <= "1111111";
    when 9 => show <= "1110011";
end case;
elsif (sel_1(2) = '0') then
  case p2 is
    when 0 => show <= "1111110";
    when 1 => show <= "0110000";
    when 2 => show <= "1101101";
    when 3 => show <= "1111001";
```

```
        when 4 =>  show <= "0110011";
        when 5 =>  show <= "1011011";
        when 6 =>  show <= "0011111";
        when 7 =>  show <= "1110000";
        when 8 =>  show <= "1111111";
        when 9 =>  show <= "1110011";
    end case;
    elsif (sel_1(3) = '0') then
    case p3 is
        when 0 =>  show <= "1111110";
        when 1 =>  show <= "0110000";
        when 2 =>  show <= "1101101";
        when 3 =>  show <= "1111001";
        when 4 =>  show <= "0110011";
        when 5 =>  show <= "1011011";
        when 6 =>  show <= "0011111";
        when 7 =>  show <= "1110000";
        when 8 =>  show <= "1111111";
        when 9 =>  show <= "1110011";
        end case;
        end if;
end process;
sel <= sel_1;
end behavior;
```

分频模块

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity dividefre4 is
port (cp_20m:in std_logic;
      cp1:out std_logic;        --200 Hz 片选信号。
      cp2:out std_logic;        --25 Hz 防抖动电路周期信号。
      cp3:out std_logic);       --5 Hz 闸门信号。
end dividefre4;
architecture behavior of dividefre4 is
signal tout : integer range 0 to 50000;
signal tout1: integer range 0 to 7;
signal tout2: integer range 0 to 39;
```

```
signal cp_1: std_logic;
signal cp_2:std_logic;
signal cp_3:std_logic;
signal cp: std_logic;
begin
  process(cp_20m)        --50000 分频,分到 400 Hz。
  begin
    if (cp_20m'event and cp_20m = '1') then
      if tout = 49999 then
        tout <= 0;
      else tout <= tout + 1;
      end if;
      if tout = 24999 then
        cp <= '0';
      else cp <= '1';
      end if;
    end if;
  end process;
process(cp)          --200 Hz。
  begin
    if (cp'event and cp = '1') then
      cp_1 <= not cp_1;
      end if;
  end process;
process(cp_1)
  begin
    if (cp_1'event and cp_1 = '1') then
      if tout1 = 7 then tout1 <= 0;
        else tout1 <= tout1 + 1;
      end if;
      if tout1 = 3 then
        cp_2 <= '1';
    elsif tout1 = 7 then cp_2 <= '0';
    end if;
    if tout2 = 39 then tout2 <= 0;
    else tout2 <= tout2 + 1;
    end if;
    if tout2 = 39 then
```

```
            cp_3 <= '1';
          elsif tout2 = 19 then cp_3 <= '0';
          end if;
        end if;
    end process;
cp1 <= cp_1; cp2 <= cp_2; cp3 <= cp_3;
end behavior;
```

锁存器模块

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity frelatch is
port ( reset: in std_logic;        --复位信号。
       cp3:in std_logic;          --闸门信号。
       overflow:in std_logic;        --各项输入。
       low:in std_logic;
       play0,play1,play2,play3: in integer range 0 to 9;
       decimal: in std_logic_vector(2 downto 0);
       overlatch: out std_logic;        --锁存后的各项输出。
       lowlatch: out std_logic;
       p0latch,p1latch,p2latch,p3latch: out integer range 0 to 9;
       delatch:out std_logic_vector(2 downto 0)
       );
end frelatch;

architecture behavior of frelatch is
begin
  process(cp3,reset)
  begin
  if reset = '1' then        --reset 为'1'时置零。
    overlatch <= '0';lowlatch <= '0';p0latch <= 0;p1latch <= 0;p2latch <= 0;p3latch <= 0;
      delatch <= decimal;
      elsif cp3'event and cp3 = '0' then
--当闸门处于下降沿时,将此时计数模块的各项输出值锁存。
overlatch <= overflow;
      lowlatch <= low;
      p0latch <= play0;p1latch <= play1;p2latch <= play2;p3latch <= play3;
      delatch <= decimal;
    end if;
```

```
end process;
end behavior;
```

计数模块

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity fretest is
port (enable:in std_logic;        --开关信号,'1'有效。
        cp3:in std_logic;         --闸门信号。
      input:in std_logic;         --被测信号。
      reset:in std_logic;         --复位信号,'0'有效。
   overflow:out std_logic;        --输入大于 9 999 kHz 信号的输出。
        low:out std_logic;        --输入小于 10 kHz 信号的输出。
          play0,play1,play2,play3: out integer range 0 to 9;   --4 位 BCD 显示结果输出。
          decimal: out std_logic_vector(2 downto 0) );         --小数点输出。

end fretest;

architecture behavior of fretest is
signal r0_1,r1_1,r2_1,r3_1,r4_1,r5_1:integer range 0 to 9;
begin
  process(input,enable,reset)
    begin
    if enable = '0' then null;
    elsif (input'event and input = '1') then
      if reset = '1' then       --复位信号为'1'时,输出为全 0。
overflow <= '0'; r0_1 <= 0;r1_1 <= 0; r2_1 <= 0; r3_1 <= 0; r4_1 <= 0; r5_1 <= 0;
      elsif cp3 = '0' then      --当闸门处于低电平'0'时,输出为全 0 。
      overflow <= '0';r0_1 <= 0;r1_1 <= 0;r2_1 <= 0;r3_1 <= 0;r4_1 <= 0;r5_1 <= 0;
    else
      r0_1 <= r0_1 + 1;
        if r0_1 = 9 then r1_1 <= r1_1 + 1;r0_1 <= 0;
          if (r1_1 = 9) then r2_1 <= r2_1 + 1;r1_1 <= 0;
            if (r2_1 = 9 ) then r3_1 <= r3_1 + 1;r2_1 <= 0;
              if (r3_1 = 9 ) then r4_1 <= r4_1 + 1;r3_1 <= 0;
                if (r4_1 = 9 ) then r5_1 <= r5_1 + 1;r4_1 <= 0;
                  if (r5_1 = 9) then
                --当计到第六位仍不够时,溢出为'1'。
r5_1 <= 0;overflow <= '1';
                  end if;
```

```
                end if;
              end if;
            end if;
          end if;
        end if;
      if ( r5_1 = 0 and r4_1 = 0 and r3_1 = 0) then
          low <= '1';
        else low <= '0';
      end if;
    end if;
    end if;
  end process;
process(r5_1,r4_1)
  begin
    if r5_1 = 0 and r4_1 = 0 then        --输出为###.##kHz。
      play0 <= r0_1; play1 <= r1_1; play2 <= r2_1;play3 <= r3_1;
      decimal <= "100";
    elsif r5_1 = 0 and r4_1 > 0 then        --输出为####.#kHz。
      play0 <= r1_1; play1 <= r2_1; play2 <= r3_1;play3 <= r4_1;
      decimal <= "010";
else      --输出为####kHz。
  play0 <= r2_1; play1 <= r3_1; play2 <= r4_1; play3 <= r5_1;
    decimal <= "000";
  end if;
end process;

end behavior;
```

五、程序说明

1. 此程序由1个上层模块将5个下层模块连接在一起而组成,5个下层模块分别是分频模块、防抖电路、计数模块、锁存器模块和显示模块。

2. 此程序将时钟分到5Hz,形成一个固定的0.2 s的闸门时间,被测信号通过0.1 s的闸门进入计数器进行计数,而在0.1 s的低电平内不计数,将计数器清零,以便下一次计数。

3. 为满足题意,被测信号为几十千赫兹(kHz)时,显示##.##kHz;
被测信号为几百千赫兹(kHz)时,显示###.#kHz;
被测信号为几千千赫兹(kHz)时,显示####kHz。

4. 采用时分复用的方法控制4个数码管的显示。

附　录

附录 1　VHDL、MAX + plus Ⅱ 知识问答

1. 学习 VHDL 应注意什么问题?

学习 VHDL 应注意以下问题:

(1) VHDL 的版本号、关键字要求、所用软件的支持范围。

(2) 自动选项与人工选项的区别。

(3) VHDL 仿真与 VHDL 综合仿真的区别。如果仅限于学习 VHDL 语法,那么需要一个 VHDL 仿真支持好的软件,这个软件最好能支持 VHDL 全集。如果需要用 VHDL 开发可编程逻辑器件,那么需要一个支持所使用的可编程逻辑器件的 VHDL 综合、仿真软件。

(4) 学习和应用 VHDL 要有一定的数字电路基本知识,并且要对可编程逻辑器件有一定的了解。

2. VHDL 对推动电子设计自动化(EDA)有什么样的作用?

VHDL 是一种新语言,在某些方面,类似于其他流行语言,如 C 语言和 Pascal 语言,但其功能完全不同。C 语言和 Pascal 语言对应于一个 CPU,即一个时序机,在某一时刻只有一条指令在运行;而 VHDL 适应于硬件的一般结构(硬件的结构大部分是并行的)。实际结果表明,使用 VHDL,其性能总是比常规 CPU 的程序设计语言在性能上要高几个数量级。这就是说,在使用软件语言的地方,VHDL 作为一个实现的语言会用得越来越多。可以预言,将来大部分小型微控制器将用 VHDL 语言取代 CPU 程序设计语言和机器码进行设计。

20 世纪 80 年代后期,由美国国防部开发的 VHDL(VHSI Hardware Description Language)在 1987 年 12 月被 IEEE 标准化(定为 IEEE 1076—1987 标准;1993 年进一步修订,定为 ANSI/IEEE 1076—1993 标准)。1996 年 IEEE 又将电路合成的标准程序与规格加入到 VHDL 中,构成 IEEE 1076.3 标准,这个标准是 VHDL 的综合标准。IEEE 制定的这些 VHDL 国际标准为电子设计自动化(EDA)的普及和推广奠定了坚实的基础。

3. 用 VHDL 进行数字电路设计涉及什么样的器件?

用 VHDL 进行数字电路设计涉及到可编程逻辑器件。VHDL 的综合,离不开可编程逻辑器件。用 VHDL 设计一个简单、不太复杂的数字电路时,设计者可以对可编程逻辑器件的知识了解不多;但是,要用 VHDL 设计一个复杂的、高性能的数字电路系统,就要求设计者对使用的可编程逻辑器件有较好的了解。

在学习使用可编程逻辑器件时,应着重了解以下几个方面的内容:

(1) 器件各管脚的功能及结构特点。

(2) 器件安全工作需要的条件,例如工作电压、能承受的最大功率、环境温度等。

(3) 器件的动态特性,例如器件固有时延、工作频率、负载电容限制等。

(4) 开发器件需要哪些硬件、软件资源。

(5) 器件的价格功能比如何。

至于那些涉及半导体方面的理论知识,对电路应用、开发者来说,只需进行一般的了解,而不必作深入研究。

4. 什么叫 VHDL 综合?

从 VHDL 代码到某一工艺技术实现的转换称为综合。综合所产生的硬件应当与已经模拟过的 VHDL 代

码有相同的行为(功能),这一点是最重要的。

综合过程:首先对要实现的逻辑进行优化,以便硬件平台(例如 PLD 器件)能优化实现设计的逻辑功能;然后自动分配硬件资源。(硬件平台优化是指占用资源(LAB、CELL)少、逻辑功能可靠等等。)

5. 什么叫 VHDL 仿真?

VHDL 仿真不涉及器件结构,一般在综合仿真前进行。VHDL 仿真的目的:一是为检查 VHDL 描述的逻辑功能是否能实现;二是为进行 VHDL 综合仿真做准备。VHDL 仿真采用功能(functional simulation)仿真。VHDL 仿真软件性能越高,能仿真的 VHDL 描述就越多。有一点需要指出的是,由于受到技术制约,能仿真 VHDL 全集的仿真软件很难见到。因此,寻找到一个性能优良的仿真软件无论对 VHDL 的初学者,还是有一定电路开发经验的科技工作者,都是一件令人兴奋的事。

6. 什么叫 VHDL 综合仿真?

VHDL 综合涉及器件结构,综合仿真是针对某个选定的器件进行的。综合仿真应采用时序仿真,即考虑器件时延存在的仿真,只有进行了综合仿真,计算机设计的实用性、可靠性才能得到有效的检验。

7. 综合是自动实现的吗?

综合是由软件自动实现的。综合工作,如器件选择、逻辑化简、器件内的布局、连线都可以由软件自动实现。一些性能优良的综合软件还能提供人工介入的命令选项。

8. MAX + plus Ⅱ 有哪些综合方式?

(1) 快速(fast)方式

快速方式只追求器件的工作速度,不追求资源最小。

(2) 所见即所得方式

所见即所得(The WYSIWYG: What You See Is What You Get)方式在综合时注意尽可能不改变设计者的逻辑设计(不增加,不化简),忠实地把设计者的逻辑设计分配到选择的器件中(例如,为解决冒险,设计了冗余逻辑,用缺省综合方式就可能把冗余逻辑消除,从而使解决冒险的措施失效。而用所见即所得综合方式,就可能把解决冒险的冗余逻辑分配进器件中。)

选择 The WYSIWYG 方式时,相关的选项应选择缺省(default)方式。

(3) 缺省综合方式(NORMAL)

缺省综合方式是厂商预先规定的综合方式,该方式能满足绝大部分逻辑设计的综合需要。初学者首先应当选择缺省综合方式。

9. 任何 VHDL 源程序都能被一种 VHDL 开发软件综合吗?

写一个综合用的 VHDL 代码是不容易的,不同的综合工具只能综合 VHDL 的一部分,也就是说,可使用的综合工具支持的 VHDL 只是 IEEE VHDL 标准的一个子集。当前比较流行的综合开发软件工具为 Synopsys、Autologic(Mentor Graphics)和 ViewLogic、MAX + plus Ⅱ。ViewLogic 是一个基于 PC 或工作站的系统,而 Synopsys 和 Autologic 都是基于工作站的系统,MAX + plus Ⅱ 是一个基于 PC 的系统。当编写的 VHDL 源程序不能为 MAX + plus Ⅱ 综合时,应该按 MAX + plus Ⅱ 软件要求改写源程序,以便能通过综合;否则,应寻找其他综合软件去执行所希望的综合。

VHDL 的特点是覆盖各个层次,包括行为描述、数据流描述和结构描述,因而它给电子电路设计者提供了很大的自由度。也正是这个原因,使得设计者在初学用 VHDL 进行硬件描述遇到困难时,往往会觉得无从下手。另外,并不是所有的 VHDL 语句都可以被某一个综合器(即开发软件)接受。设计者在描述自己的设计时必须按照 EDA 工具(使用的开发软件)的要求进行描述。软件的综合能力与其库中器件数量有密切关系,因此,设计者除了要学会 VHDL 语言的语法之外,还必须学会对不同硬件的具体描述方法,当然这需要经验的积累。

10. MAX + plus Ⅱ 能做些什么事?

利用 MAX + plus Ⅱ,可根据系统的行为和功能要求,自上而下地逐层完成相应的描述、综合、优化、仿真与验证,直到生成器件。上述设计过程除了系统行为和功能描述(用 VHDL 描述)以外,其余所有的设计过程几乎都可以用计算机自动地完成。由于 VHDL 不是专为 MAX + plus Ⅱ 软件编写的,因而 VHDL 的语法规则,不可能在 MAX + plus Ⅱ 软件环境中全部得到实现。但可以预言,MAX + plus Ⅱ 软件对 IEEE 的 VHDL 标准的支持,随着可编程器件及信息技术的发展会越来越多。

11. 在使用中小规模集成电路进行数字电路设计时,如何利用 MAX + plus Ⅱ 软件进行辅助设计?

辅助设计是指辅助完成电路原理图的设计,主要设计流程如下所述。

(1) 应用图形输入法进行设计

① 进入 MAX + plus Ⅱ 软件的图形编辑器。

② 选择中小规模集成电路芯片图符。

③ 按电路图构造原则画电路原理图,形成电路原理图文件(*.gdf)。

④ 对设计文件(*.gdf)进行编译、仿真,确定最终电路方案。

若仿真没有通过,则重新修改、建立电路原理图文件(*.gdf)),再编译、仿真,直到仿真通过。

由于在电路图设计时,需要选择器件,所以设计的成效往往有赖于对器件的熟悉程度。

(2) 应用文本(VHDL)输入法进行设计

① 可用任何一种文本编辑器建立 VHDL 文本文件(*.VHD)。用 VHDL 建立电路模型时应尽可能用行为描述语句进行描述,以减小对器件的依赖程度。

② 打开 MAX + plus Ⅱ 软件的编译窗口,对 VHDL 文本文件(*.VHD)进行编译、仿真。若仿真没有通过,则重新修改、建立 VHDL 文本文件(*.VHD),再编译、仿真,直到仿真通过。

③ 若仿真通过,查阅 *.rpt 文件得到电路逻辑方程,依据逻辑方程选择中小规模集成电路。由于 MAX + plus Ⅱ 软件提供的逻辑方程主要涉及非门、与门、或门、异或门、三态门,要使用其他中小规模集成电路,往往要对得到的逻辑方程进行转化,这一步有时是很繁琐的。需要指出的是,如设计任务复杂,不应使用中小规模集成电路进行设计,而应使用可编程逻辑器件进行设计。

如果要用 VHDL 文本法辅助设计用中小规模集成电路搭建的电路,最好使用这样的开发软件——一种能把 VHDL 文本设计转化为可视逻辑图的软件,用这种软件进行辅助设计,效果要好得多。

④ 选择中小规模集成电路的原则:

· 依据仿真后得到的延时报告选择器件的时间参数。

· 依据逻辑方程选择功能相同、结构相近的器件。

⑤ 选定中小规模集成电路器件后,再利用图形输入法建立电路模型、综合、仿真,直到仿真通过,确定最终电路方案。

附录 2 集成电路

一、我国 TTL 集成电路型号命名规则

1. CT0000 系列

1997年以后,我国生产的 TTL 集成电路型号与国际 54/74 系列 TTL 电路系列完全一致,并采用了统一型号,即 CT0000 系列。

例: $\frac{C}{⓪}\ \frac{T}{①}\ \frac{4020}{②}\ \frac{C}{③}\ \frac{J}{④}$ (正常书写:CT4020CJ)

说明:

⓪位置符号 C 表示中国制造。

①位置符号 T 表示器件类型。

②位置的数字表示系列品种代号，其中：

——首数字含义如下：

1 为标准系列，同国际 54/74 系列；

2 为高速系列，同国际 54/74 系列；

3 为肖特基系列，同国际 54S/74S 系列；

4 为低功耗肖特基系列，同国际 54LS/74LS 系列。

——其他部分表示品种代号，同国际一致。

③表示工作温度范围。

④表示封装形式。

2. 命名规则

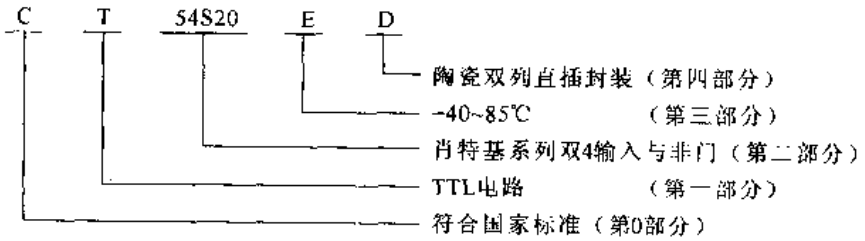
器件的型号由 5 部分组成，其中各个组成部分的符号及意义如附表 2-1 所示。

附表 2-1

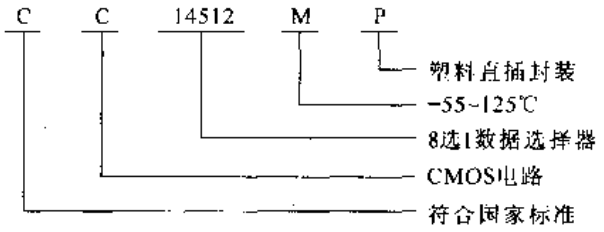
第 0 部分		第一部分		第二部分	第三部分		第四部分	
用字母表示器件符合国家标准		用字母表示器件类型		用字母、阿拉伯数字表示器件系列和品种代号	用字母表示器件的工作温度范围		用字母表示器件封装形式	
符号	意义	符号	意义		符号	意义	符号	意义
C	中国制造	T	TTL		C	0～70℃	W	陶瓷扁平
		H	HTL		E	-40～85℃	B	塑料扁平
		E	ECL		R	-55～85℃	F	全密封扁平
		C	CMOS		M	-55～125℃	D	陶瓷直插
		F	线性放大器				P	塑料直插
		D	音响、电视电路				J	黑陶瓷直插
		W	稳压器				K	金属菱形
		J	接口电路				M	金属圆型
		B	非线性电路					
		M	存储器					
		μ	微型机电路					

3. 示例

- 肖特基 TTL 双输入与非门：



• CMOS 8 选 1 数据选择器(3S)：



二、部分国际公司 TTL 集成电路型号命名规则

1.(美国)德克萨斯(TEXAS)公司

例：SN ① 74 ② LS ③ 74 ④ J ⑤

说明：

①表示德克萨斯公司标准电路。

②表示工作温度范围。

——54 系列：－55～125℃；

——74 系列：0～70℃。

③表示类型。其中：

——ALS：先进的低功耗肖特基类型；

——AS：先进的肖特基类型；

——〈空白〉：标准类型；

——H：高速类型；

——L：低功耗类型；

——LS：低功耗肖特基类型；

——S：肖特基类型。

④表示品种代号。

⑤表示封装形式。其中：

——J：陶瓷双列直插；

——N：塑料双列直插；

——T：金属扁平；

——W：陶瓷扁平。

2.(美国)摩托罗拉(MOTOROLA)公司

例：MC ① 74 ② 194 ③ P ④

说明：

①表示摩托罗拉公司封装的集成电路。

②表示工作温度范围。其中：

——4、20、30、40、72、74、83 系列：0～75℃；

——5、21、31、43、82、54、93 系列：－55～125℃。

③表示品种代号。

④表示封装形式。其中：

——F：陶瓷扁平；

——L：陶瓷双列直插；

——P：塑料双列直插。

（注：LS-TTL的型号同德克萨斯公司一致，如：SN74LS194J。）

3.（美国）国家半导体（NATIONAL SEMICONDUCTOR）公司

例：$\underset{①}{\underline{DM}}$ $\underset{②}{\underline{74}}$ $\underset{③}{\underline{LS}}$ $\underset{④}{\underline{161}}$ $\underset{⑤}{\underline{N}}$

说明：

①表示国家半导体公司单片数字电路。

②表示工作温度。其中：

——74、80、81、82、85、87、88系列：0～70℃；

——54、70、71、72、75、77、78、93、96系列：－55～125℃；

——83，96系列：0～75℃。

③表示类型。其中：

——〈空白〉：标准类型；

——H：高速类型；

——L：低功耗类型；

——LS：低功耗肖特基类型；

——S：肖特基类型。

④表示品种代号。

⑤表示封装形式。

——D：玻璃-金属双列直插；

——F：玻璃-金属扁平；

——J：低温陶瓷双列直插；

——N：塑料双列直插；

——W：低温陶瓷扁平。

4.（日本）日立（HITACHI）公司

例：$\underset{①}{\underline{HD}}$ $\underset{②}{\underline{74}}$ $\underset{③}{\underline{LS}}$ $\underset{④}{\underline{191}}$ $\underset{⑤}{\underline{P}}$

说明：

①表示日立公司数字集成电路。

②表示工作温度范围。其中：

——74系列：－20～75℃。

③表示类型。其中：

——〈空白〉：标准系列；

——LS：低功耗肖特基系列；

——S：肖特基系列。

④表示品种代号。

⑤表示封装形式。其中：

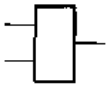
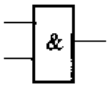
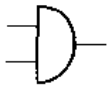
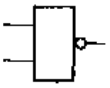
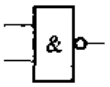
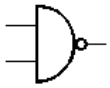
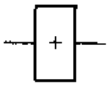
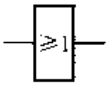
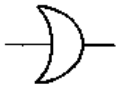
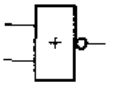
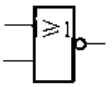
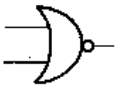
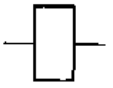
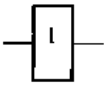
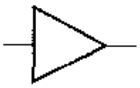
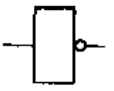
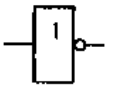
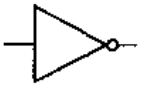
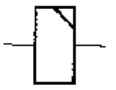
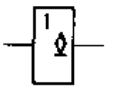
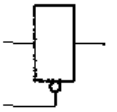
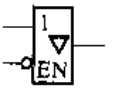
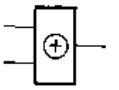
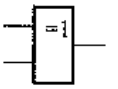
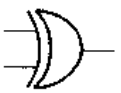
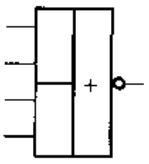
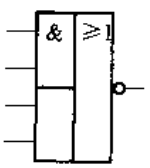
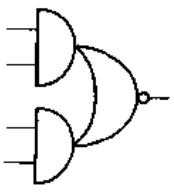
——〈空白〉：玻璃-陶瓷双列直插；

——P：塑料双列直插。

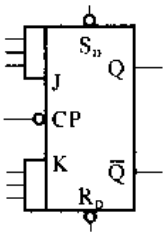
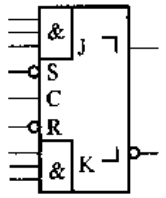
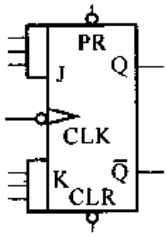
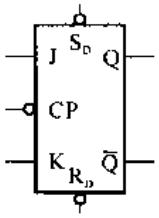
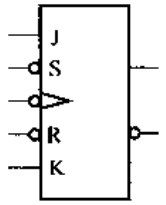
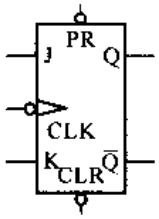
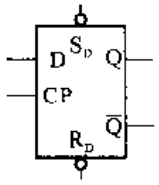
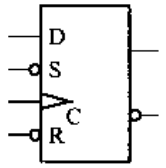
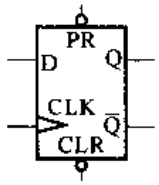
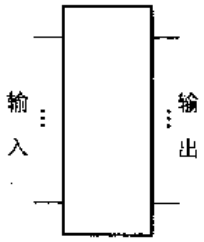
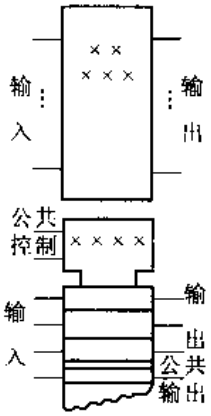
三、部分逻辑图形符号对照表

部分逻辑图形符号对照表如附表 2-2 所示。

附表 2-2 部分逻辑图形符号对照表

器件名称	原部标(SJ)符号	国标(GB)符号	IEEE符号
与门			
与非门			
或门			
或非门			
缓冲门			
反相器			
集电极开路输出门			
三态输出门			
异或门			
与或非门			

续表

器件名称	原部标(SJ)符号	国标(GB)符号	IEEE符号
(主从) J-K触发器 (T1072)			
(下降沿) J-K触发器 (T112)			
(上升沿) D触发器 (T4074)			
中规模集成 功能电路			与SJ符号相似

常用总限定符及其含义：

& ——与，只有所有输入都为1，输出才为1；

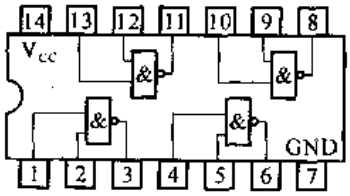
≥1——或，只要有1个以上输入为1，输出才为1；

=1——异或，只有两个输入之一为1，输出才为1；

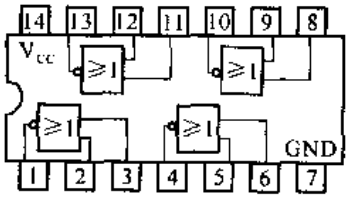
1——缓冲，只有输入为1，输出才为1。

四、TTL 集成电路管脚排列图

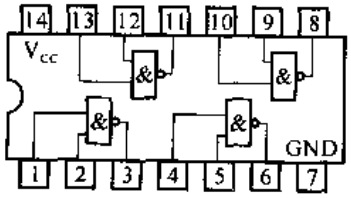
1. 逻辑门



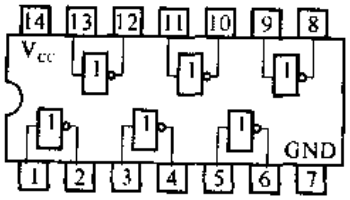
74LS00 2 输入四与非门



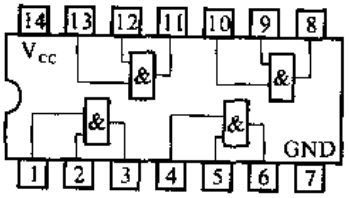
74LS03 2 输入四或非门



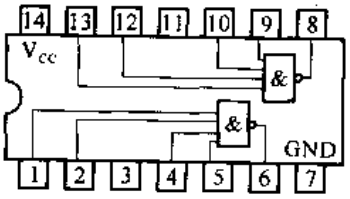
74LS03 2 输入四与非门(OC)



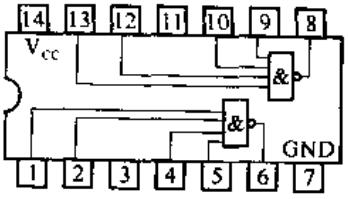
74LS04 六反相器



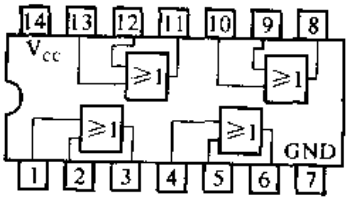
74LS08 2 输入四与门



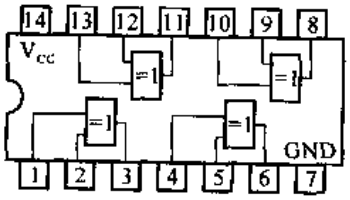
74LS20 4 输入二与非门



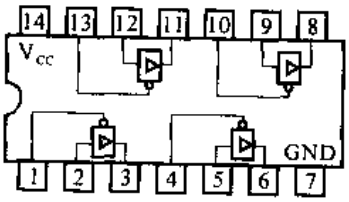
74LS22 4 输入二与非门(OC)



74LS32 2 输入四或门

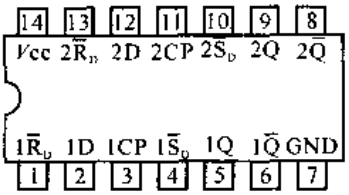


74LS86、74LS386 2 输入四异或门

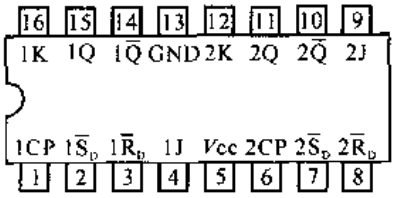


74LS125 4总线缓冲门

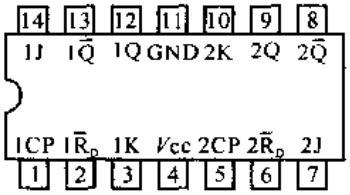
2. 触发器与锁存器



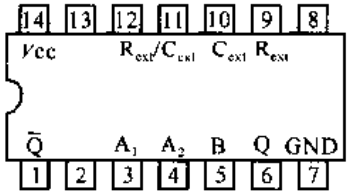
74LS74 双D触发器



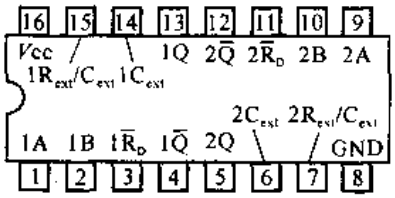
74LS76 双J-K触发器



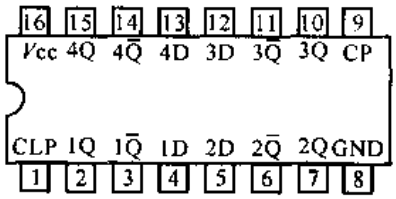
74LS73 双J-K触发器



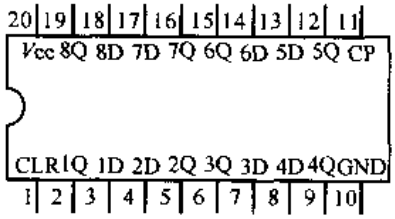
74LS121 单稳多谐振荡器



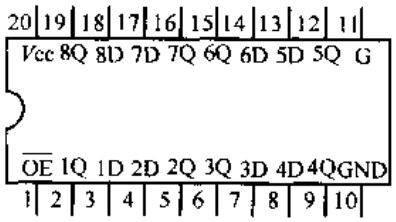
74LS123 可重触发双单稳多谐振荡器



74LS175 4D触发器

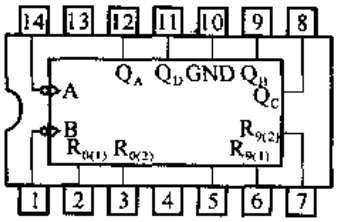


74LS273 8D触发器

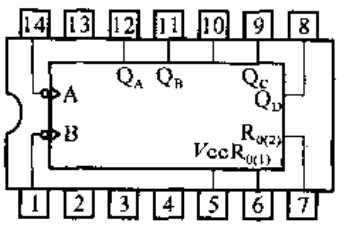


74LS373 8D锁存器

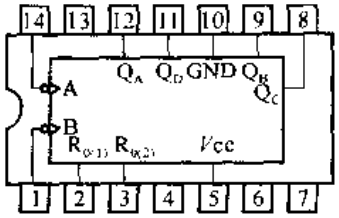
3. 计数器、译码器、数据选择器



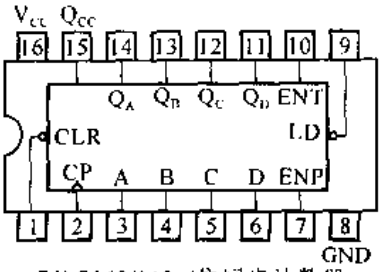
74LS90 二、五、十进制计数器



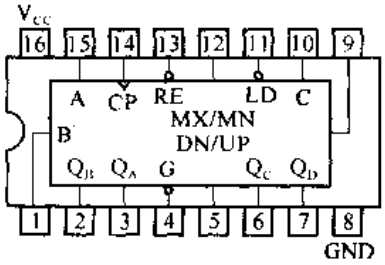
74LS92 12分步计数器



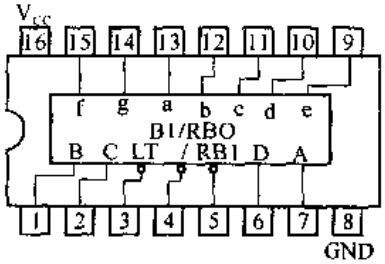
74LS93 4位二进制计数器



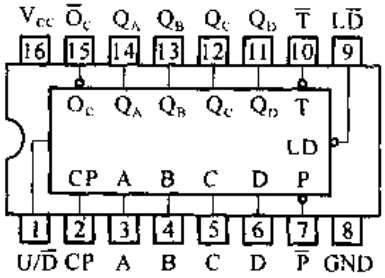
74LS160/163 4位同步计数器



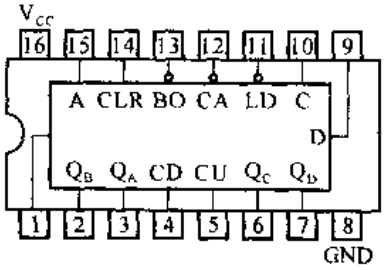
74LS190/191同步可逆计数



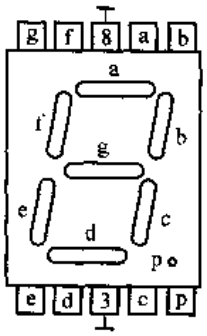
74LS48 BCD-7段译码器/驱动器



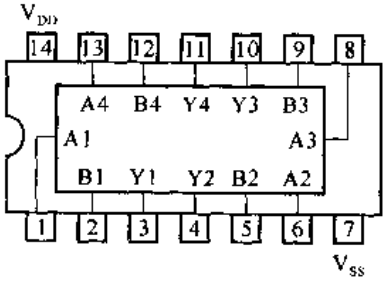
74LS169 十进制加减计数器



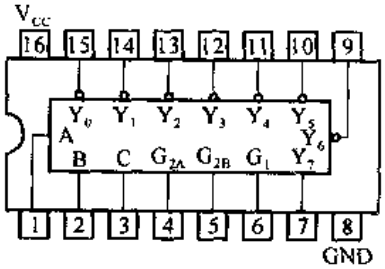
74LS192/193同步可逆双时钟计数



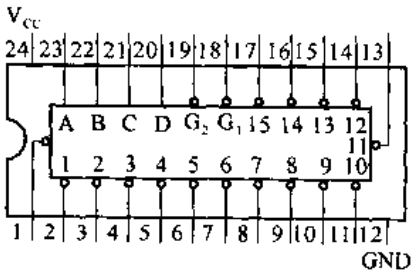
BS202 数码显示器



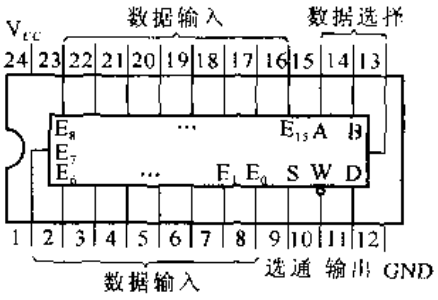
CD4011B 2输入四与非门



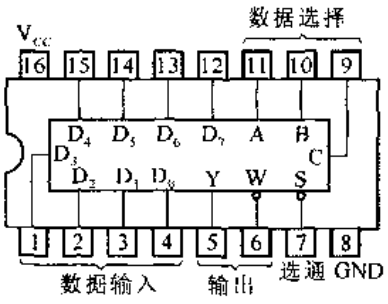
74LS138 3线-8线译码器



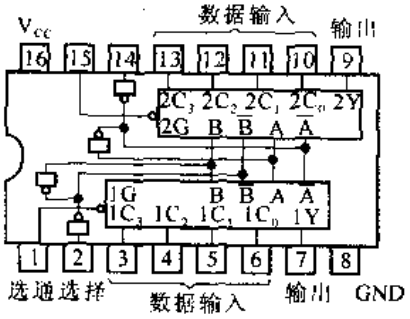
74LS154 4线-16线译码器



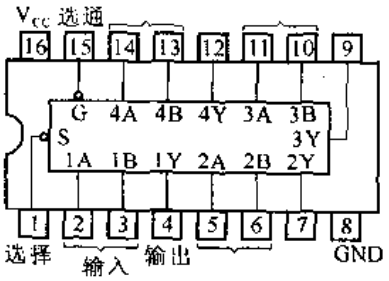
74LS150 16选1数据选择器



74LS151 8选1数据选择器

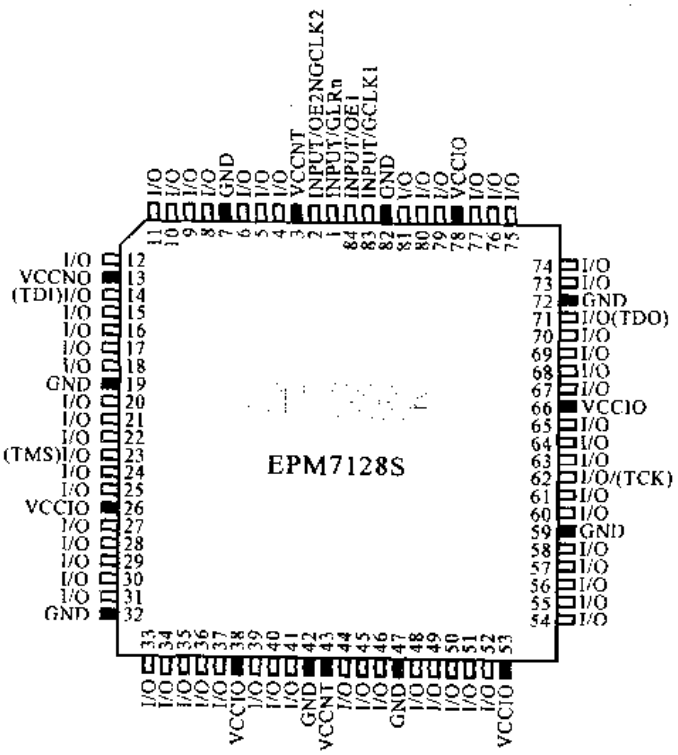


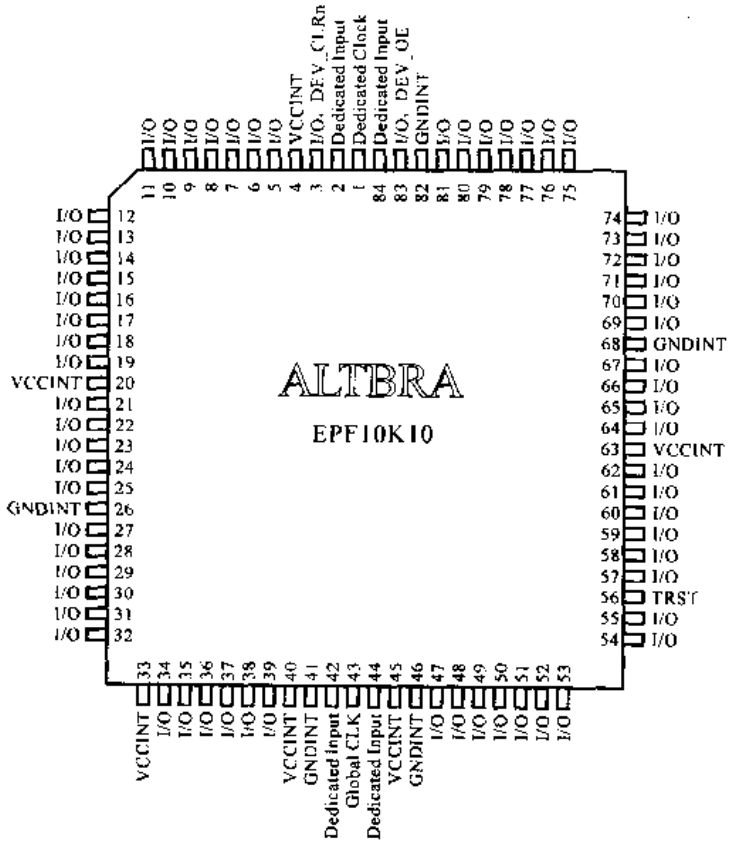
74LS153 双4选1数据选择器



74LS157 4位2选1数据选择器

4. 可编程器件管脚排列图





附录 3　集成电路主要性能参数

一、TTL 与非门的主要性能参数

参数名称	单位	测试条件		CT54/74	CT54H/74H	CT54S/74S	CT54LS/74LS
输入高电平 $V_{IH(min)}$	V			2	2	2	2
输入低电平 $V_{IL(max)}$	V			0.8	0.8	0.8	0.8
输出高电平 $V_{OH(min)}$	V	$V_{CC(max)}$ $I_{OH(max)}$		2.4	2.4	2.7	2.7
输出低电平 $V_{OL(max)}$	V	$V_{CC(min)}$ $I_{OL(max)}$		0.4	0.4	0.5	0.5
输入高电平电流 $I_{IH(max)}$	μA	$V_{CC(max)}$	$V_{IH}=2.4\,V$	40	50		
			$V_{IH}=2.7\,V$			50	20
输入低电平电流 $I_{IL(max)}$	mA		$V_{IL}=0.4\,V$	-1.6			-0.4
			$V_{IL}=0.5\,V$		-2	-2	
输出高电平电流 $I_{OH(max)}$	mA			0.4	0.5	1.0	0.4
输出低电平电流 $I_{OL(max)}$	mA			-16	-20	-20	-4

续表

参数名称		单位	测试条件	CT54/74	CT54H/74H	CT54S/74S	CT54LS/74LS
电源电流	$I_{CCH(max)}$	mA		8	16.8	16	1.6
	$I_{CCL(max)}$	mA		22	40	36	4.4
传输延迟	$t_{PLH(max)}$	ns		22	10	4.5	15
	$t_{PHL(max)}$	ns		15	10	5	15

注：① 均为四2输入与非门；

② 未注明测试条件者，均为 $V_{CC}=5$ V 下的测量结果；

③ 环境温度均为 $T_A=25$℃；

④ 该表数据取自《中国集成电路大全——TTL电路》。

二、CMOS或非门CC4001的主要性能参数

参数名称	单位	测试条件			参数	
		V_O/V	V_I/V	V_{DD}/V	最大	最小
静态电源电流 I_{DD}	μA		0/5	5		0.25
			0/10	10		0.50
			0/15	15		1.00
输出低电平电流 I_{OL}	mA	0.4	5	5	-0.51	
		0.5	10	10	-1.30	
		1.5	15	15	-3.40	
输出高电平电流 I_{OH}	mA	4.6	0	5	0.5	
		9.5	0	10	1.3	
		13.5	0	15	3.4	
输入电流 I_I	μA		0/18	18		±0.1
输出低电平 V_{OL}	V		5	5		0.05
			10	10		0.05
			15	15		0.05
输出高电平 V_{OH}	V		0	5	4.95	
			0	10	9.95	
			0	15	14.95	
输入低电平 V_{IL}	V	4.5		5		1.5
		9		10		3
		13.5		15		4
输入高电平 V_{IH}	V	0.5		5	3.5	
		1		10	7	
		1.5		15	11	

续表

参数名称	单位	测试条件			参数	
		V_O/V	V_I/V	V_{DD}/V	最大	最小
传输延迟时间 t_{PHL}、t_{PLH}	ns			5		250
				10		120
				15		90
输入电容 C_I	pF					7.5

注：① CC4001 为四 2 输入或非门；

② t_{PHL}、t_{PLH}在 C_L = 50 pF、R_L = 200 kΩ 下测得；

③ 环境温度均为 T_A = 25℃；

① 该表数据选自《中国集成电路大全——CMOS 集成电路》。

三、TTL 集成触发器 CT5474/7474 等的主要性能参数

参数名称		单位	测试条件	CT5474/7474		CT54H74/74H74		CT54S74/74S74		CT54LS74/74LS74	
				最小	最大	最小	最大	最小	最大	最小	最大
输入高电平 V_{IH}		V		2		2		2		2	
输入低电平 V_{IL}		V			0.8		0.8		0.8		0.8
输入钳位电压 V_{IK}		V	V_{CC} = min I_{IK} = max		-1.5		-1.5		-1.2		-1.5
输出高电平 V_{OH}		V	V_{CC} = min V_{IL} = max V_{OH} = max	2.4		2.4		2.5		2.5	
输出低电平 V_{OL}		V	V_{CC} = min V_{IH} = 2 V I_{OL} = max		0.4		0.4		0.5		0.5
输出高电平电流 I_{OH}		mA			0.4		1		1		0.4
输出低电平电流 I_{OL}		mA			-16		-20		-20		-4
高电平输入电流 I_{IH}	D	μA	V_{CC} = max V_{IH} = 2.4 V		40		50		50		20
	$\overline{R}_D$				120		150		150		40
	$\overline{S}_D$				80		100		100		40
	CP				80		100		100		20
低电平输入电流 I_{IL}	D	mA	V_{CC} = max V_{IL} = 0.4 V		-1.6		-2		-2		-0.4
	$\overline{R}_D$				-3.2		-4		-6		-0.8
	$\overline{S}_D$				-1.6		-2		-4		-0.8
	CP				-3.2		-4		-4		-0.4
输出短路电流 I_{OS}		mA	V_{CC} = max	18	57	40	100	40	100	20	100
每个触发器电源电流 I_{CC}		mA	V_{CC} = max		15		25		25		4
最高时钟频率 f_{max}		MHz		15		35		75		25	
传输延迟时间	t_{PLH}	ns	CP→Q,$\overline{Q}$		25		15		9		25
	t_{PHL}				40		20		9		40

注：① CT5474/7474、CT54H74/74H74、CT54S74/74S74、CT54LS74/74LS74 均为双 D 触发器；

② 本表数据选自《中国集成电路大全——TTL 集成电路》。

四、MOS 集成触发器 CC4027 的主要性能参数

参数名称	单位	测试条件			参数	
		V_O/V	V_I/V	V_{DD}/V	最小	最大
静态电流 I_{DD}	μA		0/5	5		1
			0/10	10		2
			0/15	15		4
输出低电平电流 I_{OL}	mA	0.4	0/5	5	-0.51	
		0.5	0/10	10	-1.30	
		1.5	0/15	15	-3.40	
输出高电平电流 I_{OH}	mA	4.6	0/5	5	0.51	
		9.5	0/10	10	1.30	
		13.5	0/15	15	3.40	
输入电流 I_I	μA		0/18	18		±0.1
输出低电平 V_{OL}	V		0/5	5		0.05
			0/10	10		0.05
			0/15	15		0.05
输出高电平 V_{OH}	V		0/5	5	4.95	
			0/10	10	9.95	
			0/15	15	14.95	
输入低电平 V_{IL}	V	0.5/4.5		5		1.5
		1/9		10		3
		1.5/13.5		15		4
输入高电平 V_{IH}	V	0.5/4.5		5	3.5	
		1/9		10	7	
		1.5/13.5		15	11	
传输延迟 t_{PHL}、t_{PLH}	ns	CP→Q, $\overline{Q}$		5		300
				10		130
				15		90
最高输入时钟频率 f_{max}	MHz	C_L = 50 pF R_L = 200 kΩ		5	3.5	
				10	8	
				15	12	
脉冲宽度 t_W	ns	CP		5		140
				10		60
				15		40
输入电容 C_I	pF			7.5		

注：① CC4027 为双 J-K 触发器；

② 环境温度均为 T_A = 25℃；

③ 本表数据选自《中国集成电路大全——CMOS 集成电路》。

五、555 定时器 CC7555 的主要性能参数

参数名称		单位	测试条件	参数
电源电压 V_{DD}		V	$-40℃ \leqslant T_A \leqslant +80℃$	3 ~ 18
电源电流 I_{DD}		μA	$V_{DD} = 3$ V	60
			$V_{DD} = 18$ V	120
时间误差	初始精度		R_1、R_2 为 1 ~ 100 kΩ	≤5%
	温　漂	℃	$C = 0.1$ μF	50×10^{-6}
	随电压漂移	V	$5\text{ V} \leqslant V_{DD} \leqslant 15\text{ V}$	1.0%
阈值电压 V_{TH}		V	$5\text{ V} \leqslant V_{DD} \leqslant 15\text{ V}$	$\frac{2}{3}V_{DD}$
触发电压 V_{TR}		V	$5\text{ V} \leqslant V_{DD} \leqslant 15\text{ V}$	$\frac{1}{3}V_{DD}$
触发电流 I_{TR}		pA	$V_{DD} = 15$ V	50
复位电流 I_R		pA	$V_{DD} = 15$ V	100
复位电压 V_R		V	$5\text{ V} \leqslant V_{DD} \leqslant 15\text{ V}$	0.7
控制电压 V_{CO}		V	$5\text{ V} \leqslant V_{DD} \leqslant 15\text{ V}$	$\frac{2}{3}V_{DD}$
输出低电平 V_{OL}		V	$V_{DD} = 15$ V, $I_{OL} = -3.2$ mA	0.1
输出高电平 V_{OH}		V	$V_{DD} = 15$ V, $I_{OH} = 1$ mA	14.8
输出上升时间 t_r		ns	$R_L = 10$ MΩ, $C_L = 10$ pF	40
输出下降时间 t_f		ns	$R_L = 10$ MΩ, $C_L = 10$ pF	40
最高振荡频率 f_{max}		MHz	自激多谐振荡	≥500

注：本表参数选自《中国集成电路大全——CMOS 集成电路》。

附录 4　VHDL 术语汉英对照

包	package	库名	library name
包体	package body	库单元	library unit
包说明	package declartion	局部端口	local port
设计实体	design entity	组装	configuration
设计单元	design unit	组装说明	configuration declaration
设计库	design library	组装规定	configuration specification
设计重用	design reuse	组装语句	configuration statement
设计描述	design description	实体	entity
块	block	实体外貌	entity aspect
块语句	block statement	实体说明	entity declaration
块组装	block configuration	结构	architecture
库	library	结构体	architecture body
库子句	library clause	结构化设计	structured design

结构描述	structure description
端口	port
端口说明	port declaration
端口关联	port association
端口关联表	port association list
子(数据)类型	subtype
子(数据)类型说明	subtype declaration
文字	literal
文件对象	file object
文件(数据)类型	file type
记录元素	record element
记录(数据)类型	record type
对象	object
对象说明	object declaration
对象类别	object class
多维数组	multi-dimensionalarray
访问类型	visiting type
关系算符	relational operator
关联元素	association element
关联表	association list
形式类属	formal generic
形式参数	formal parameter
形式端口	formal port
局部类属	local generic
形参	formal parameter
位串	bit tstring
位向量	bit vector
表达式	expression
参数	parameter
非限定性数组(数据)类型	unconstrained array type
物理(数据)类型	physical type
物理描述	physical description
变量	variable
变量类	variable class
变量说明	variable declaration
变量赋值语句	variable assignment statement
转换函数	conversion function
枚举(数据)类型	enumeration type
限定性数组类型	constrained array type
选择信号赋值	selected signal assignment
信号	signal
信号说明	signal declaration
信号驱动源	signal drive source
信号类	signal class
信号赋值	signal assignment
标号	label
标识符	identifier
标识符表	identifier list
标量类型	scalar type
类属	Generic
类属关联	generic association
类属关联表	generic association list
数据类型	type
数据类型转换	type conversion
数据类型说明	type declaration
类型标记	type mark
预定义子数据类型	predefined subtype
预定义数据类型	predefined type
预定义运算符	predefined operator
浮点(数据)类型	float type
常量	constant
常量说明	constant declaration
常量类	constant class
敏感信号	sensitive signal
敏感信号表	sensitive signal list
敏感(性)	sensitivety
属性	attribute
属性名	attributename
属性说明	attribute declaration
属性指定	attribute specification
基类型	base type
数值类型	numeric type
数组	array
数组范围	array bounds
数组类型	array type
数组维数	array dimension
算术运算符	arithmetic operator
算符符号	operator symbol
整数文字	integer literal
整数(数据)类型	integer type
操作数	operand
操作算符	operator
IF 语句	IF statement
CASE 语句	CASE statement
EXIT 语句	EXIT statement

LOOP 语句	LOOP statement
NEXT 语句	NEXT statement
NULL 语句	NULL statement
RETUN 语句	RETURN statement
WAIT 语句	WAIT statement
子程序	subprogram
子程序规定	subprogram specification
子程序参量	subprogam parameter
子程序参量关联	subprogram pameter association
子程序形式参量	formal subprogram parameter
子程序说明	subprogram declaration
子程序调用	subprogram call
子程序体	subprogram body
串行语句	sequential statement
串行断言语句	sequential assert statement
串行操作	sequential operation
进程	process
进程语句	process statement
空	null
循环结构	loop constructs
循环语句	loop statement
嵌套	nest
元件	component
元件关联	component association
元件例化	component instantiations
元件例化语句	component instantiations statement
元件组装	component configuration
元件表	component table
元件说(声)明	component declaration
生成语句	generate statement
生成参数	generate parameter
并发	concurrent
并发过程调用	concurrent procedure call
并发性	concurrency
并发信号赋值语句	concurrent signal assignment statement
并发条件赋值语句	concurrent conditional assignment statement
并发语句	concurrent statement
并发断言语句	concurrent assertion statement
过程	procedure
过程调用	procedure call
过程调用语句	procedure call statement
过程规定	procedure specification
条件信号赋值	conditional signal assignment
位置关联	positional association
函数	function
函数调用	function call
函数说明	function declaration
函数定义	function specification
选择信号赋值	selected signal assignment
信号赋值语句	signal assignment statement
重载	overloading
重载操作符	overloading operator
顺序语句	sequential statement
ASIC(专用集成电路)	Application Specific Integrated Circuit
Δ延迟	delta delay
二元算符	binary operator
二值逻辑	binary valued logic
七值逻辑	seven valued logic
九值逻辑	nine valued logic
可见性	visibility
未连接端口	unconnected port
外部时钟信号	external clock signal
外部信号	external signal
输入激励	external input excitation
生存周期	lifetime intervals
对象断点	object breakpoint
对象说明	object declaration
对象类别	object class
传输	transfer
传输延迟	transport delay
延迟	delay
延迟模型	delay modal
关联	association
同步	synchronism
同步电路	synchronous circuit
同步模拟	synchronous simulation
多驱动源	multi drivers
多值模拟	multiple valueds simulation
多值逻辑	multiple valued logic
多重嵌套	iterative nest
多周期操作	multi-cycle operation
多层次模拟	multi-level simulation
多维数组	multi-dimensionalarray

字面量	literal
异步	asynchronous
异步事件	asynchronous event
异步时序电路	asynchronous sequential circuit
自顶向下	top-down
自底向上	bottom-up
行为	behavior
行为描述	behavioral description
行断点	line breakpoint
死锁	dead lock
动态行为	dynamic behaviour
动态冒险	dynamic hazard
自驱动模拟	self driven simulation
约束	constraint
优先级	dominance
初值	initial value
别名	alias
层次化设计	level design
层次化结构	level structure
时钟信号	clock signal
时序逻辑	sequential logic
完备性	completeness
驱动值	driving value
驱动源	driving source
错误程度	severity
事件	event
事件表	event table
事务	transaction
范围限制	range constraint
一元运算符	unary operator
单位延迟时间	unit delay time
组合逻辑	combinational logic
命名关联	named association
空事务处理	null transaction
规则结构	regular structure
实验模型	experiment model
波形	waveform
波形元素	waveform elements
抽象描述	abstract description
保留字	keyword
测试平台	test bench
测试向量	test vector
标准延迟	standard delay
标准单元/元件	standard element/ component
保护	guard
说明区	declarative regions
响应时间	reaction time
冒险	hazard
指定	specification
总线	bus
预定义库名	predefined library name
预定义属性	predefined attribute
递归结构	recursive structure
被保护的信号	guarde dsignal
被保护的赋值	guarded assignment
被保护的信号赋值	guarded signal assignment
扇入	fan in
扇入表	fan in table
扇出	fan out
扇出表	fan out table
浮点文字	float literal
竞争	Race
逻辑名	logical name
逻辑运算符	logical operator
逻辑模拟	logic simulation
接口元素	interface element
接口表	interface list
接口对象	interface object
常数延迟时间	constant delaytime
惯性延迟	inertial delay
描述范畴	description domain
混合模拟	hybrid simulation
断点	breakpoint
最大(小)延迟时间	maximal/Minimal delay time
硬件描述语言	hardware description language
源	source
源描述	source description
零延迟	zero delay
输出函数	output function
模式	mode
模块化设计	modularization design
模块化结构	modularization structure
模拟	simulation
模拟周期	Simulation cycle
模拟时钟	simulation clock
模拟层次	simulation level

模拟环境	simulation environment	默认(缺省)值	default value
模拟结构	Simulation structure	默认(缺省)的端口关联	default port association
模拟步长	simulation steplength	激励	Excitation
模拟器	simulator	激励产生器	Excitation generator
模拟模型	simulation model	激励响应	Excitation response
模拟波形	simulation waveform	激活	active
静态	Static	激活条件	activition condition
静态冒险	static hazard	操作并发性	operation concurrency
静态测试	static test		

附录 5 数字电路术语汉英对照

(以汉字笔画为序)

一 画

一热态位编码 One Hot Encoding

二 画

二进制 Binary System
二进制译码器 Binary Decoder
二-十进制 Binary-Coded Decimal
二-十进制译码器 Binary-Coded Decimal Decoder
二值编码 Binary Code
七段译码器 Seven-Segment Decoder
七段字符显示器 Seven-Segment Character Mode Display
十六进制 Hexadecimal System
十进制 Decimal System

三 画

下一状态 Next State
下降时间 Fall Time
与门 AND Gate
与非门 NAND Gate
与或非门 AND-OR-NOT Gate
门阵列 Gate Array(GA)
上升时间 Rise Time
干扰 Interference
工作速度 Operating Speed
小规模集成 Small Scale Integration(SSI)
大规模集成 Large Scale Integration(LSI)
三态门 Three-State Gate

四 画

双极型 CMOS Bipolar-CMOS(Bi-CMOS)
双积分型 A/D 转换器 Dual Integration A/D Converter
双极性变换器 Bipolar Converter
双向移位寄存器 Bidirectional Shift Register
计数器 Counter
计算机 Computer
计算机辅助设计 Computer Aided Design(CAD)
开关电路 Switching Circuit
反相器 Inverter
反变量 Complemented Variable
反演 Reversal Development
反熔丝 Anti-Fuse
分辨率 Resolution
分配器 Demultiplexer
比较器 Comparator
方波 Square Wave
互补 MOS Complementary Metal Oxide Semiconductor (CMOS)
中规模集成 Medium Scale Integration(MSI)
毛刺 Glitch
无关项 Don't Gare
无冗余覆盖 Irredundent Cover

内建自测试 Built In Self Test(BIST)
内建逻辑模块观察器 Built In Logic Block Observer (BILBO)
边界扫描设计 Boundary Scan Design
巨模块 Megablock

五 画

电擦除的可编程只读存储器 Electrically Erasable Programmable Read-Only Memory(EEPROM)
电子设计自动化 Electronic Design Automation(EDA)
电平相关扫描设计 Level-Sensitive Scan Design(LSSD)
电阻梯形网络 Resistor Ladder Network
可观察性 Observability
可控制性 Controllability
可测试性 Testability
可测试电路 Testable Circuit
可测试设计 Testable Design, Design For Test(DFT)
可编程逻辑阵列 Programmable Logic Array(PLA)
可编程阵列逻辑 Programmable Array Logic(PAL)
可编程只读存储器 Programmable Read-Only Memory (PROM)
可编程逻辑器件 Programmable Logic Device(PLD)
可编程连线 Programmable Interconnect(PI)
可擦除的可编程逻辑器件 Erasable Programmable Logic Device(EPLD)
可重构逻辑模块 Reconfigurable Logic Block(CLB)
只读存储器 Read-Only Memory(ROM)
主从触发器 Master-Slave Flip-Flop
边沿触发器 Edge-Triggered Flip-Flop
半加器 Half Adder
加法器 Adder
加/减计数器 Up-Down Counter
全加器 Full Adder
功能表 Function Table
卡诺图 Karnaugh Map
左移 Shift-Left
右移 Shift-Right
对偶 Duality
正逻辑 Positive Logic
失调误差 Offset Error
立方符号 Cubic Notation
本原蕴涵项 Prime Implicant
未完全规定的逻辑函数 Unspecified Logic Function
专用集成电路 Application Specific Integrated Circuit (ASIC)
写操作 Write Operation
发光二极管 Light Emitting Diode(LED)

六 画

字 Word
字线 Word Line
地址 Address
地址译码器 Address Decoder
并行语句 Parallel Statement
并行输入 Parallel Input
并联模/数转换器 Parallel A/D Converter
同步清 0 Synchronous Clear
同步计算器 Synchronous Counter
同步预置 Synchronous Preset
同步时序电路 Synchronous Sequential Circuit
异步计数器 Asynchronous Counter
异或 Exclusive-OR
约束 Constraint
约束项 Constraint Term
约束条件 Constraint Condition
权电阻 Weighted Resistance
权电阻网络 Weighted Resistance Network
存储器 Memory
先进先出存储器 First-In First-Out Memory(FIFO)
先进后出存储器 First-In Last-Out Memory(FILO)
先行进位加法器 Look-Ahead Carry Adder
存储矩阵 Memory Matrix
有权码 Weighted Code
优先编码器 Priority Encoder
自启动 Self-Starting
自检 Self Test
后沿 Trailing Edge
负逻辑 Negative Logic
快闪存储器 Flash Memory
动态险象 Dynamic Hazard
动态随机存储器 Dynamic Random Access Memory (DRAM)
在系统可编程 In-System Programmable(isp)
在电路可重构 In-Circuit Reconfiguration(icr)
米里型电路 Mealy Model Circuit

行为描述 Behaviour Description
传输延迟 Propagation Delay
设计实体 Design Entity
伪随机测试 Pseudo-Random Testing
伪穷举测试 Pseudo Exhaustive Testing
扫描设计 Scan Design
多输出函数 Multi-Output Function

七 画

状态表 State Table
状态图 State Diagram
状态分配 State Assignment
状态相容 State Compatible
状态化简 State Reduction
状态方程 State Equation
状态等价 State Equivalence
译码器 Decoder
时钟 Clock
时钟脉冲 Clock Pulse(CP)
时钟频率 Clock Frequency
时钟偏移 Clock Skew
时钟 R-S 触发器 Clocked R-S Flip-Flop
时间驱动 Time Drive
时间常数 Time Constant
时序电路 Sequential Circuit
时序波形图 Sequence Waveform
时序逻辑电路 Sequential Logic Circuit
时钟分配网络 Clock Distribution Network(CDN)
串行-并行变换 Serial-Parallel Transfer
进程语句 Process Statement
进位 Carry
位 Bit
位线 Bit Line
余 3 码 Excess-3 Code
肖特基 TTL Schottky TTL
延迟时间 Delay Time
扭环形计数器 Twisted Ring Counter
序列信号发生器 Sequencer
序列检测器 Sequence Detector
穷举测试 Exhaustive Testing
低压 CMOS Low-Voltage CMOS(LVC)
低压高速 CMOS Low-Voltage HCMOS(LV)
低压双极型 CMOS Low-Voltage BiCMOS(LVT)
低功耗 TTL Low-Power TTL
低功耗肖特基逻辑电路 Low Power Schottky Logic

八 画

或 OR
或门 OR Gate
或非 NOR
或非门 NOR Gate
非 NOT
非线性误差 Nonlinear Error
采样-保持电路 Sample-Hold Circuit
线与 Wired-AND
线性反馈移位寄存器 Linear Feedback Shift Register (LFSR)
直接预置 Direct Preset
直接复位 Direct Reset
函数分解 Function Decomposition
函数发生器 Function Generator
使能 Enable
组合逻辑 Combinational Logic
组合逻辑电路 Combinational Logic Circuit
组合逻辑函数 Combinational Logic Function
单极性变换器 Unipolar Converter
单输出函数 Single-Output Function
单位间距码 Unit Distance Code
和之积表达式 Product of Sum Expression
奇偶校验 Parity Check
奇偶校验码 Parity Check Code
奇偶校验/发生器 Odd-Even Check/Generator
定时 Timing
建立时间 Set-up Time
转换误差 Converting Error
环形计数器 Ring Counter
实质本原蕴涵项 Essential Prime Implicant
事件驱动 Event Drive
刷新 Refresh
现场可编程门阵列 Field Programmable Gate Array (FPGA)
现在状态 Present State
枚举类型 Enumeration Type
码组转换器 Code Converter
参考电压 Reference Voltage

九 画

总线 Bus
总线收发器 Bus Transceiver
显示器 Display
保持时间 Hold Time
前沿 Leading Edge
选通脉冲 Gate Pulse
险象 Hazard
标准和的积表达式 Standard(Canonical) Product of Sum Expression
标准积的和表达式 Standard(Canonical) Sum of Product Expression
标准单元阵列 Standard Cell Array(SCA)
相对精度 Relative Accuracy Error
故障模型 Model of Fault
美国标准信息交换码 American Standard Code for Information Interchange
查找表 Look Up Table(LUT)
顺序存储器 Sequential Access Memory(SAM)
顺序语句 Sequential Statement
脉冲分配器 Pulse Demultiplexer
结构体 Architecture Body
结构描述 Architecture Description
复位 Reset
复杂的可编程逻辑器件 Complex Programmable Logic Device(CPLD)

十 画

积之和表达式 Product of Sum Expression
积分器 Integrator
通用阵列逻辑 Generic Array Logic(GAL)
通用逻辑模块 Generic Logic Block(GLB)
通路敏化 Path Sensitization
逐次渐近型模/数转换器 Successive Approximation A/D Converter
莫尔型电路 Moore Model Circuit
特征方程 Characteristic Equation
特征分析 Signature Analysis
乘法 D/A 转换器 Multiplying D/A Converter
乘积项 Product Term
浮栅雪崩注入 MOS 管 Floating-Gate Avalanche-Injuction MOSEFT(FAMOS)
借位 Borrow
读出 Read
扇出 Fan Out
真值表 Truth Table
离散量 Discrete Quantity
格雷码 Gray Code
起始状态 Initial State
流程图 Flow Chart
竞争 Race
消抖动开关 Debounce Switch
原变量 Uncomplemented Variable
原始状态表 Primitive State Table
高密度可编程逻辑器件 High Density Programmable Logic Device(HDPLD)
高级低压 CMOS 工艺 Advanced Low-Voltage CMOS Technology(ALVC)
高速 CMOS 逻辑电路 High-Speed CMOS Logic(HC/HCT)
高级双极型 CMOS 工艺 Advanced BiCMOS Technology (ABT)
高级 CMOS 逻辑电路 Advanced CMOS Logic(AC)
高级高速 CMOS 逻辑电路 Advanced High-Speed CMOS Logic(AHC/AHCT)
高级低功耗肖特基逻辑电路 Advanced Low-Power Schottky Logic(ALS)
高级肖特基逻辑电路 Advanced Schottky Logic(AS)

十一画

逻辑 Logic
逻辑和 Logic Sum
逻辑积 Logic Product
逻辑图 Logic Diagram
逻辑电路 Logic Circuit
逻辑非门 Logic Inverter
逻辑运算 Logic Operation
逻辑设计 Logic Design
逻辑综合 Logic Synthesis
逻辑验证 Logic Verification
逻辑模拟 Logic Simulation
逻辑代数 Logic Algebra
逻辑电平 Logic Level
逻辑函数 Logic Function
逻辑单元 Logic Element(LE)

逻辑阵列块 Logic Array Block(LAB)
基数 Radix
基本 R-S 触发器 Simple R-S Flip-Flop
随机测试 Random Testing
随机存取存储器 Random Access Memory(RAM)
寄存器 Register
寄存器传输级 Register Transfer Level(RTL)
移位寄存器 Shift Register
移位寄存器用锁存器 Shift Register Latch(SRL)
敏感信号表 Sensitivity List
隐埋宏单元 Baried Macro Cell
控制单元 Control Unit
超大规模集成 Very Large Scale Integration(VLSI)
梯形网络 D/A 转换器 Ladder Network D/A Converter

十二画

最小项 Minterm
最小覆盖 Minimum Covering
最大项 Maxterm
集成电路 Integrated Circuit(IC)
集成极开路输出 Open-Collector Output
量化 Quantization
量化误差 Quantizing Error
编码状态表 Encoded State Table
编码 Encode
编码器 Encoder
晶体管-晶体管逻辑 Transistor-Transistor Logic(TTL)
紫外线擦除的可编程ROM Ultraviolet Erasable Programmable ROM (UEPROM)
锁存器 Latch

十三画

数字量 Digital Quantity
数字元件 Digital Device
数字电路 Digital Circuit
数字系统 Digital System
数字信号 Digital Signal
数字技术 Digital Technique
数据类型 Date Type
数据选择器 Multiplexer
数据处理单元 Data Processing Unit
数制 Number System
数码管 Nixie Light
数值比较器 Digital Comparator
数模转换 Digital to Analog Conversion
数模转换器 Digital to Analog Converter(DAC)
输入/输出模块 Input/Output Block(IOB)
输入/输出单元 Input/Output Cell(I/O C)
输出布线池 Output Routing Pool(ORP)
输出逻辑宏单元 Output Logic Macro Cell(OLMC)
输出方程 Output Equation
触发器 Filp-Flop(FF)
硬件描述语言 Hardware Description Language(HDL)
简单的可编程逻辑器件 Simple Programmable Logic Device(SPLD)
简化状态表 Reduced State Table

十四画

模拟量 Analog Quantity
模拟开关 Analog Switch
模数转换 Analog to Digital Conversion
模数转换器 Analog to Digital Converter(ADC)
静态险象 Static Hazard
静态随机存取存储器 Static Random Access Memory (SRAM)
熔丝 Fuse
算法设计 Algorithm Design
算法状态机 Algorithmic State Machine(ASM)
算法流程图 Algorithmic Flow Chart
算术逻辑单元 Arithmatic Logic Unit(ALU)
端口模式 Port Mode

十五画

蕴涵 Implicant
摩根定理 De Morgan's Theorem

十六画

激励表 Excitation Table
激励方程 Excitation Equation
噪声容限 Noise Margin
覆盖 Cover

参考文献

1 徐惠民,安德宁. 数字逻辑设计与 VHDL 描述. 北京:机械工业出版社,2002

2 ALTERA. max + plus Ⅱ Handbook

3 赵俊超,等. 集成电路设计 VHDL 教程. 北京:希望电子出版社,2002

4 林敏,方颖立. VHDL 数字系统设计与高层次综合. 北京:电子工业出版社,2002

5 曾繁泰,陈美金. VHDL 程序设计. 北京:清华大学出版社,2001

6 潘松,王国栋. VHDL 实用教程. 成都:电子科技大学出版社,2001

7 王毓银. 数字电路逻辑设计. 北京:高等教育出版社,2001

[General Information]
书名=数字电路 EDA 入门：VHDL 程序实例集
作者=张亦华　延明编著
页数=157
SS 号=11062962
出版日期=2003 年 03 月第 1 版

Powered by XiaoGuo’s publishing Studio

QQ:8204136

Website: www.mcuzone.com

2005