第3章

用VHDL程序实现常用逻辑电路

本章通过用硬件描述语言 VHDL 实现的设计举例，介绍 EDA 技术在组合逻辑电路、时序逻辑电路、存储器设计中的应用，同时介绍 VHDL 程序的状态机设计。

3.1 组合逻辑电路设计

3.1.1 基本门电路

【例 3.1】 2 输入与门的行为描述。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY xx1 IS
  PORT(a,b: IN STD_ LOGIC;
       y: OUT STD_LOGIC);
END xx1;
ARCHITECTURE AND2PP OF xx1 IS
BEGIN
  y <= a AND B;
END AND2PP;
```

【例 3.2】 2 输入与门的寄存器传输级描述。

```
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  xx1  IS
PORT(a,b: IN  STD_ LOGIC;
         y: OUT STD_LOGIC);
END xx1;
ARCHITECTURE AND2PP OF xx1 IS
BEGIN
  PROCESS(a,b);
  VARIABLE COM: STD_LOGIC_VECTOR(1 DOWNTO 0);
```

```
    BEGIN
    COM := a & b;
    CASE COM IS
        WHEN"00" = > y < = '0';
        WHEN"01" = > y < = '0';
        WHEN"10" = > y < = '0';
        WHEN"11" = > y < = '1';
        WHEN OTHERS = > y < = 'X';
    END CASE;
END AND2PP;
```

【例 3.3】 2 输入异或门电路。

```
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
  PORT(a,b: IN STD_IOGIC;
  y: out std_logic);
END xor2;
ARCHITECTURE XOR_BEHAVE OF XOR2 IS
begin
  y < = a xor b;
END XOR_BEHAVE;
```

3.1.2　译码器

【例 3.4】 实现 74LS1383-8 译码器(输出低电平有效)。

3-8 译码器 74LS138 的输出有效电平为低电平,译码器的使能控制输入端 g1、g2a、g2b 有效时,当 3 线数据输入端 cba=000 时,y[7..0]=11111110(即 y[0]=0);当 cba=001 时,y[7..0]=11111101(即 y[1]=0);以此类推。图 3.1 表示的是 3-8 译码器端口图。

用 VHDL 描述的 3-8 译码器 74LS138 源程序如下:

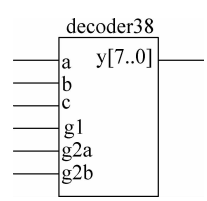

图 3.1　3-8 译码器端口图

```
library ieee;
use ieee.std_logic_1164.all;
entity decoder38 is
port(a,b,c,g1,g2a,g2b: in std_logic;
           y: out std_logic_vector(7 downto 0));
end decoder38;
architecture behave38 OF decoder38 is
signal indata: std_logic_vector(2 downto 0);
begin
  indata < = c&b&a;
  process(indata,g1,g2a,g2b)
     begin
       if(g1 = '1' and g2a = '0' and g2b = '0') then
       case indata 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 behave38;
```

【例 3.5】 分别以 4 种方法描述一个输出高电平有效的 3-8 译码器。

```
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED_ALL;
ENTITY  DECODER IS
    PORT(INP: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
             OUTP: OUT BIT_VECTOR (7 DOWNTO 0));
END DECODER;
```

方法 1：使用 PROCESS 语句。

```
ARCHITECTURE  ART1 OF DECODER IS
  BEGIN
  PROCESS(INP)
BEGIN
    OUTP <= (OTHERS =>'0') ;
    OUTP(CONV_INTEGER(INP))<= '1';
  END PROCESS;
END ART2;
```

方法 2：使用 WHEN ELSE 语句(条件赋值语句,并行的)。

```
ARCHITECTURE ART2 OF DECODER IS
BEGIN
  OUTP(0)<= '1' WHEN INP = "000" ELSE "0";
  OUTP(1)<= '1' WHEN INP = "001" ELSE "0";
  OUTP(2)<= '1' WHEN INP = "010" ELSE "0";
  OUTP(3)<= '1' WHEN INP = "011" ELSE "0";
  OUTP(4)<= '1' WHEN INP = "100" ELSE "0";
  OUTP(5)<= '1' WHEN INP = "101" ELSE "0";
  OUTP(6)<= '1' WHEN INP = "110" ELSE "0";
  OUTP(7)<= '1' WHEN INP = "111" ELSE "0";
END ART2;
```

方法 3：使用 CASE_WHEN 语句(转向控制语句,顺序的)。

```
ARCHITECTURE ART3 OF DECODER IS
   BEGIN
   PROCESS(INP)
   CASE INP IS
```

```
    WHEN "000" => OUTP <= "00000001";
    WHEN "001" => OUTP <= "00000010";
    WHEN "010" => OUTP <= "00000100";
    WHEN "011" => OUTP <= "00001000";
    WHEN "100" => OUTP <= "00010000";
    WHEN "101" => OUTP <= "00100000";
    WHEN "110" => OUTP <= "01000000";
    WHEN "111" => OUTP <= "10000000";
    WHEN OTHERS => OUTP <= "XXXXXXXX";
  END CASE;
END ART3;
```

方法 4：使用 SLL 逻辑运算符(使用逻辑左移运算符)。

```
ARCHITECTURE ART4 OF DECODER IS
   BEGIN
   OUTP <= "00000001" SLL (CONV_INTEGER(INP));
END ART4;
```

【例 3.6】 二进制-十进制译码器。

二进制-十进制码是指用 4 位二进制码来表示 1 位十进制数中的 0～9 这 10 个数码，简称 BCD 码。二进制-十进制译码器是实现 8421-BCD 码至十进制译码的电路。真值表见表 3.1，输出为高电平有效。

表 3.1　二进制-十进制译码器真值表

输　入				输　出									
d	c	b	a	y0	y1	y2	y3	y4	y5	y6	y7	y8	y9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

源程序如下：

```
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  DECODER 8421_10  IS
  PORT (A,B,C,D: IN STD_LOGIC;
       Y: OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END DECODER 8421_10;
ARCHITECTURE RTL OF DECODER 8421_10  IS
SIGNAL COMB: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
```

```
  COMB <= D&C&B&A;
  PROCESS(COMB)
  BEGIN
    CASE COMB IS
      WHEN "0000" => Y <= "0000000001";
      WHEN "0001" => Y <= "0000000010";
      WHEN "0010" => Y <= "0000000100";
      WHEN "0011" => Y <= "0000001000";
      WHEN "0100" => Y <= "0000010000";
      WHEN "0101" => Y <= "0000100000";
      WHEN "0110" => Y <= "0001000000";
      WHEN "0111" => Y <= "0010000000";
      WHEN "1000" => Y <= "0100000000";
      WHEN "1001" => Y <= "1000000000";
      WHEN OTHERS => Y <= "XXXXXXXXXX";
    END CASE;
  END PROCESS;
END RTL;
```

3.1.3 编码器

【例 3.7】 8-3 线优先编码器，输入信号为 A、B、C、D、E、F、G 和 H，输出信号为 OUT0、OUT1 和 OUT2。输入信号中 A 的优先级别最低，以此类推，H 的优先级别最高。下面用 3 种方法设计 8-3 线优先编码器。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  ENCODER IS
    PORT (A,B,C,D,E,F,G,H: IN STD_LOGIC;
            OUT0,OUT1,OUT2: OUT STD_LOGIC);
END ENCODER;
```

方法 1：使用条件赋值语句。

```
ARCHITECTURE ART1 OF ENCODER IS
SIGNAL OUTS: STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
       OUTS (2 DOWNTO 0)<=   "111" WHEN H = '1' ELSE
                             "110" WHEN G = '1' ELSE
                             "101" WHEN F = '1' ELSE
                             "100" WHEN E = '1' ELSE
                             "011" WHEN D = '1' ELSE
                             "010" WHEN C = '1' ELSE
                             "001" WHEN B = '1' ELSE
                             "000" WHEN A = '1' ELSE
                             "XXX";
     OUT0 <= OUTS(0);
     OUT1 <= OUTS(1);
     OUT2 <= OUTS(2);
END ART1;
```

方法 2：使用 LOOP 语句。

```
ARCHITECTURE ART2 OF ENCODER IS
BEGIN
PROCESS(A,B,C,D,E,F,G,H)
VARIABLE INPUTS: STD_LOGIC_VECTOR(7 DOWNTO 0);
VARIABLE I: INTEGER;
BEGIN
   INPUT := (H,G,F,E,D,C,B,A);
   I := 7;
   WHILE I >= 0 AND INPUTS(I) = '1' LOOP
        I := I - 1;
   END LOOP;
   (OUT2,OUT1,OUT0)<= CONV_STD_LOGIC_VECTOR(I,3);
  END PROCESS;
```

方法 3：使用 IF 语句。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  ENCODER IS
  PORT(IN1: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       OUT1: OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END ENCODER;
ARCHITECTURE  ART3 OF ENCODER IS
BEGIN
PROCESS(IN1)
BEGIN
   IF    IN1(7) = '1' THEN OUT1 <= "111";
   ELSIF IN1(6) = '1' THEN OUT1 <= "110";
   ELSIF IN1(5) = '1' THEN OUT1 <= "101";
   ELSIF IN1(4) = '1' THEN OUT1 <= "100";
   ELSIF IN1(3) = '1' THEN OUT1 <= "011";
   ELSIF IN1(2) = '1' THEN OUT1 <= "010";
   ELSIF IN1(1) = '1' THEN OUT1 <= "001";
   ELSIF IN1(0) = '1' THEN OUT1 <= "000";
   ELSE OUT1 <= "XXX";
   END IF ;
   END PROCESS;
END ART3;
```

3.1.4　7 段码译码器

【例 3.8】 只有 7 段码的输出驱动 7 段码显示器，才能显示正常的数字。图 3.2 是共阴极数码管显示器电路示意图，在这里要设计一个共阴极 7 段码显示驱动程序，其源程序如下：

```
Library ieee;
use ieee.std_logic_1164.all;
```

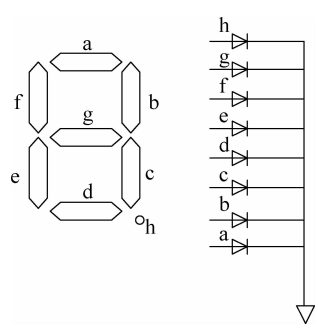

图 3.2　共阴极数码管显示器电路示意图

```
entity decl7s is
  port (a: in std_logic_vector(3 downto 0);
        led7s: out std_logic_vector(7 downto 0));
end decl7s;
architecture behave of decl7s is
begin
  process(a)
  begin
   case a is
      when "0000" => led7s <= "0111111";
      when "0001" => led7s <= "0000110";
      when "0010" => led7s <= "1011011";
      when "0011" => led7s <= "1001111";
      when "0100" => led7s <= "1100110";
      when "0101" => led7s <= "1101101";
      when "0110" => led7s <= "1111101";
      when "0111" => led7s <= "0000111";
      when "1000" => led7s <= "1111111";
      when "1001" => led7s <= "1101111";
      when "1010" => led7s <= "1110111";
      when "1011" => led7s <= "1111100";
      when "1100" => led7s <= "0111001";
      when "1101" => led7s <= "1011110";
      when "1110" => led7s <= "1111001";
      when "1111" => led7s <= "1110001";
      when others => null;
    end case;
  end process;
end behave;
```

3.1.5 数据选择器

数据选择是指经过选择，把多路数据中的某一路数据传送到公共数据线上，实现数据选择功能的逻辑电路称为数据选择器。它的作用相当于多个输入的单刀多掷开关。通常，选择器有 2^N 个输入信号，1 个输出信号，同时还有 N 条数据选择线。常用的数据选择器有 4 选 1、8 选 1 和 16 选 1 等类型。下面以 8 选 1 数据选择器为例，介绍数据选择器的 VHDL 设计，真值表见表 3.2，其工作原理为：根据数据选择输入端 s2、s1、s0 的不同组合，将 A[7..0] 相应的输入信号传到输出端 y。

表 3.2　8 选 1 数据选择器真值表

s2	s1	s0	y
0	0	0	A[0]
0	0	1	A[1]
0	1	0	A[2]
0	1	1	A[3]
1	0	0	A[4]
1	0	1	A[5]
1	1	0	A[6]
1	1	1	A[7]

【例 3.9】 用 VHDL 描述的 8 选 1 数据选择器的源程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux81 IS
  PORT(s2,s1,s0:IN STD_LOGIC;
       A: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       y: OUT STD_LOGIC);
END mux81;
ARCHITECTURE rtl OF mux81 IS
  SIGNAL s: STD_LOGIC_VECTOR(2 DOWNTO 0);
  BEGIN
  s <= s2&s1&s0;
  PROCESS(s2,s1,s0)
  BEGIN
    CASE s IS
    WHEN "000" => y <= A(0);
    WHEN "001" => y <= A(1);
    WHEN "010" => y <= A(2);
    WHEN "011" => y <= A(3);
    WHEN "100" => y <= A(4);
    WHEN "101" => y <= A(5);
    WHEN "110" => y <= A(6);
    WHEN "111" => y <= A(7);
    WHEN OTHERS => y<= 'X';
    END CASE;
  END PROCESS;
END rtl;
```

8 选 1 数据选择器电路的仿真波形如图 3.3 所示。从图中可以看出，当 s＝000、A＝10011110 时，y＝A(0)＝0，以此类推，结果与理论值符合。

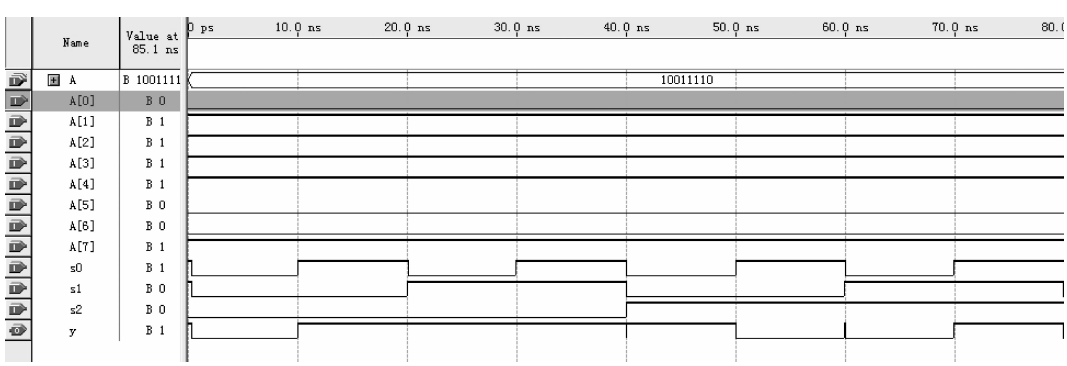

图 3.3　8 选 1 数据选择器电路的仿真波形

3.1.6　数值比较器

数值比较器可以比较两个二进制是否相等，下面是一个 4 位比较器的 VHDL 描述。有两个 4 位二进制数，分别是 A 和 B，输出为 EQ，当 A＝B 时，EQ＝1，否则 EQ＝0。

【例 3.10】 4 位比较器的 VHDL 描述。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY COMPARE IS
  PORT(A,B:IN STD_LOGIC_VECTOR(4 DOWNTO 0);
       EQ:OUT STD_LOGIC);
END COMPARE;
ARCHITECTURE ART OF COMPARE IS
BEGIN
  EQ<='1' WHEN  A=B ELSE '0';
END ART;
```

3.1.7 算术运算电路

1. 加法器

加法器有全加器和半加器之分,全加器可以用两个半加器构成,因此下面先以半加器为例加以说明。

半加器有两个二进制一位的输入端 a 和 b,一位的和输出端 s 及一位的进位输出端 co。半加器的真值表见表 3.3。

表 3.3 半加器的真值表

二进制输入		和 输 出	进位输出
b	a	s	co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

【例 3.11】 半加器。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY half_adder IS
    PORT (a,b: IN STD_LOGIC;
          s,co: OUT STD_LOGIC);
END half_adder;
ARCHITECTURE half1 OF half_adder  IS
SIGNAL  c,d: STD_LOGIC;
BEGIN
  c<=a OR b;
  d<=a NAND b;
  co<=NOT d;
  s<=c and d;
END half1;
```

用两个半加器可以构成一个全加器，全加器的电路如图 3.4 所示。基于半加器的描述，若采用 COMPONENT 语句和 PORT MAP 语句就很容易编写出描述全加器的程序。

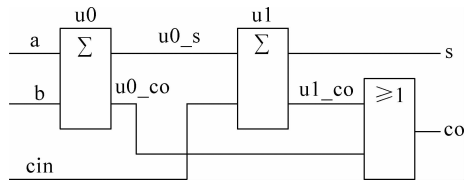

图 3.4　用两个半加器构成的全加器

【例 3.12】 利用 COMPONENT 语句编写的全加器的程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY full_adder IS
    PORT (a,b,cin: IN  STD_LOGIC;
          s,co: OUT  STD_LOGIC);
END full_adder;
ARCHITECTURE full1 OF full_adder IS
COMPONENT half_adder
    PORT (a,b: IN  STD _LOGIC;
          s,co: OUT STD _LOGIC);
END COMPONENT;
SIGNAL u0_co,u0_s,u1_co: STD _LOGIC;
BEGIN
  u0: half_adder port map(a,b,u0_s,u0_co);
  u1: half_adder port map(u0_s,cin,s,u1_co);
  co <= u0_co or u1_co;
END full1;
```

2. 乘法器

【例 3.13】 4 位乘法器的设计。其中 a[3..0]和 b[3..0]是被乘数和乘数，其 q[7..0]是乘积输出端。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY v_mult IS
    PORT (a,b: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
          q: OUT  STD_LOGIC_VECTOR(7 DOWNTO 0));
END v_mult;
ARCHITECTURE  rtl OF v_mult IS
BEGIN
  q <= a * b;
END rtl;
```

4 位乘法器电路的仿真波形如图 3.5 所示。从图中可以看出，当 a＝0010、b＝0001 时，乘法结果 q＝00000010，以此类推，结果与理论值符合。

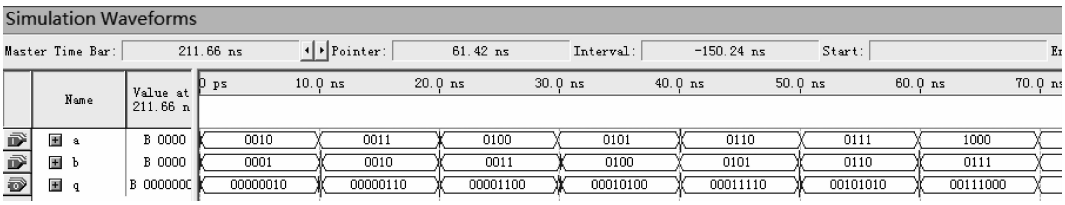

图 3.5　4 位乘法器电路的仿真波形

3.1.8　三态门及总线缓冲器

三态门和数据缓冲器是接口电路和总线驱动电路中经常用到的器件，涉及三态门或三态总线缓冲器的状态，除了正常的高低状态外，要有一个高阻态，在 VHDL 语言中规定了高阻态常用大写字母 Z 来表示。

1. 三态门电路描述

三态门就是在普通门电路的基础上增加控制电路，使它除了具有高电平和低电平两个状态外，还具有高阻态，这里输出端相当于开路。

【例 3.14】　三态门的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY TRIGATE IS
    PORT (EN,DIN:IN  STD_LOGIC;
          DOUT:OUT  STD_LOGIC);
END TRIGATE;
ARCHITECTURE  ART OF TRIGATE IS
  BEGIN
  PROCESS(EN,DIN)
  BEGIN
   IF EN = '1' THEN DOUT <= DIN;
    ELSE  DOUT <= 'Z';
   END IF;
  END PROCESS;
END ART;
```

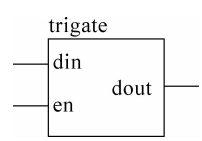

图 3.6　三态门的逻辑示意图

图 3.6 为三态门的逻辑示意图，图 3.7 为三态门的时序仿真图。

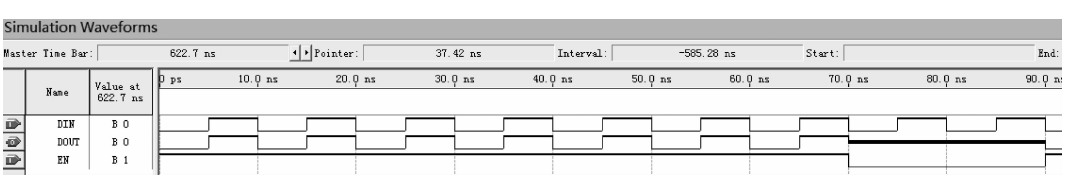

图 3.7　三态门的时序仿真图

2. 单向总线缓冲器

单向总线缓冲器通常由三态门组成，用来驱动地址总线和控制总线。一个 8 位单向总

线缓冲器由 8 个三态门构成，具有 8 路输入、8 路输出和一个使能信号 EN。

【例 3.15】 单向总线缓冲器的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  TRIBUF8 IS
    PORT (DIN: IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
          EN: IN STD_LOGIC;
         DOUT: OUT  STD_LOGIC_VECTOR(7 DOWNTO 0));
END TRIBUF8;
ARCHITECTURE  ART OF TRIBUF8 IS
  BEGIN
  PROCESS(EN,DIN)
  BEGIN
   IF EN = '1' THEN DOUT <= DIN;
    ELSE  DOUT <= "ZZZZZZZZ";
   END IF;
  END PROCESS;
END ART;
```

图 3.8 为 8 位单向总线缓冲器的时序仿真图。

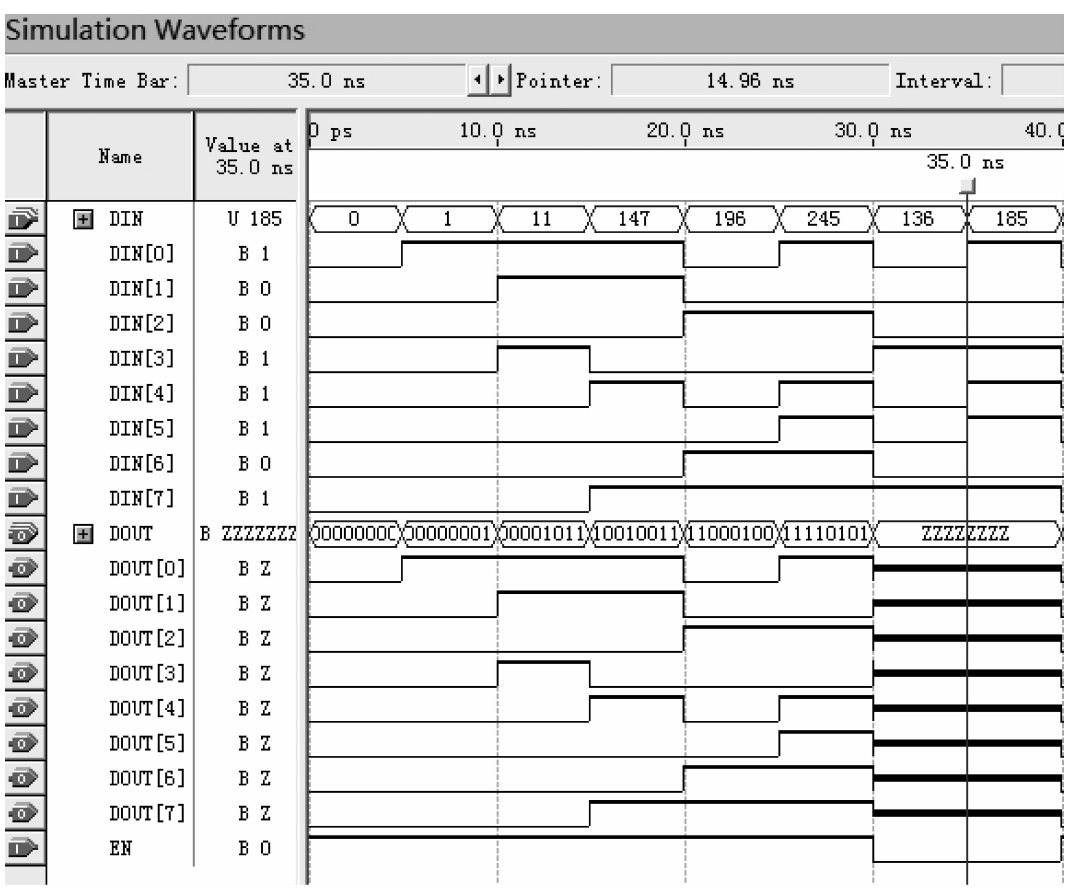

图 3.8　8 位单向总线缓冲器的时序仿真图

3. 双向总线缓冲器

双向总线缓冲器用于数据总线的驱动和缓冲，典型的双向总线缓冲器如图 3.9 所示。图中的双向总线缓冲器有两个数据输出输入端 A 和 B，一个方向控制端 DIR 和一个选通端 EN。EN＝0 时双向缓冲器选通，若 DIR＝0，则 A＝B，反之则 B＝A。

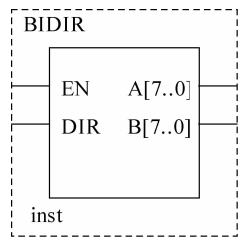

图 3.9　双向总线缓冲器

【例 3.16】 双向总线缓冲器的 VHDL 源程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY BIDIR IS
  PORT (A,B:INOUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        EN,DIR:IN STD_LOGIC);
END BIDIR ;
ARCHITECTURE ART OF BIDIR IS
  SIGNAL AOUT,BOUT:STD_LOGIC_VECTOR(7 DOWNTO 0);
  BEGIN
  PROCESS(A,EN,DIR)
  BEGIN
  IF((EN = '0')AND(DIR = '1')) THEN BOUT <= A;
    ELSE BOUT <= "ZZZZZZZZ";
  END IF;
   B <= BOUT;
  END PROCESS;
  PROCESS(B,EN,DIR)
  BEGIN
  IF((EN = '0')AND(DIR = '1')) THEN AOUT <= B;
    ELSE AOUT <= "ZZZZZZZZ";
  END IF;
   A <= AOUT;
  END PROCESS;
END ART;
```

3.2　时序逻辑电路设计

时序逻辑电路是数字电路系统中常用基本逻辑电路，它由组合逻辑电路和存储电路两部分组成，存储电路由触发器构成，是时序逻辑电路不可缺少的部分。电路结构决定了时序逻辑电路的特点，即任一时刻的输出信号不仅取决于当时的输入信号，还取决于电路的原来状态。由于时序电路具有“记忆”功能，因此在数字系统设计中应用广泛。

时序逻辑电路的重要标志是具有时钟脉冲 clock，在时钟脉冲的上升沿和下降沿的控制下，时序逻辑电路的状态才发生变化。

3.2.1　触发器

1. D 触发器

【例 3.17】 基本 D 触发器的设计。上升沿时，保证 Q＝D。上升沿触发的 D 触发器的逻辑功能表见表 3.4。图 3.10 为 D 触发器的逻辑示意图，图 3.11 为 D 触发器的时序仿真图。

表 3.4　D 触发器的逻辑功能表

输　　入	输　　出	
D	CLK	Q
X	0	保持
X	1	保持
0	↑	0
1	↑	1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY DCFQ IS
    PORT(D,CLK:IN STD_LOGIC;
                Q:OUT STD_LOGIC);
END DCFQ;
ARCHITECTURE ART OF DCFQ IS
   BEGIN
   PROCESS(CLK)
   BEGIN
   IF (CLK'EVENT AND CLK = '1') THEN      -- 时钟上升沿触发
       Q <= D;
  END IF;
  END PROCESS;
END ART;
```

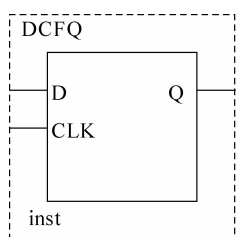

图 3.10　D 触发器的逻辑示意图

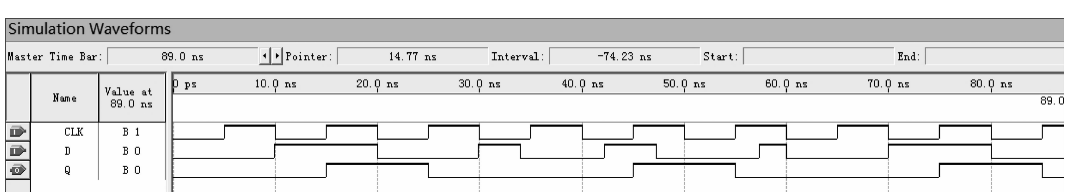

图 3.11　D 触发器的时序仿真图

【例 3.18】　异步置位/复位 D 触发器的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY YFCFQ IS
    PORT(CLK,D,CLR,PSET:IN STD_LOGIC;
       Q:OUT STD_LOGIC);
END YFCFQ;
```

```
ARCHITECTURE RTL OF YFCFQ IS
BEGIN
  PROCESS(CLK,CLR,PSET)
  BEGIN
  IF(PSET = '0')THEN
      Q <= '1';
    ELSIF (CLR = '0') THEN
     Q <= '0';
     ELSIF (CLK'EVENT AND CLK  = '1')THEN
     Q <=  D;
    END IF;
   END PROCESS;
END RTL;
```

图 3.12 为异步置位/复位 D 触发器的逻辑示意图，图 3.13 为异步置位/复位 D 触发器的时序仿真图。

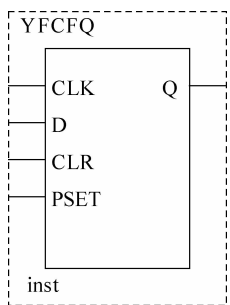

图 3.12　异步置位/复位 D 触发器的逻辑示意图

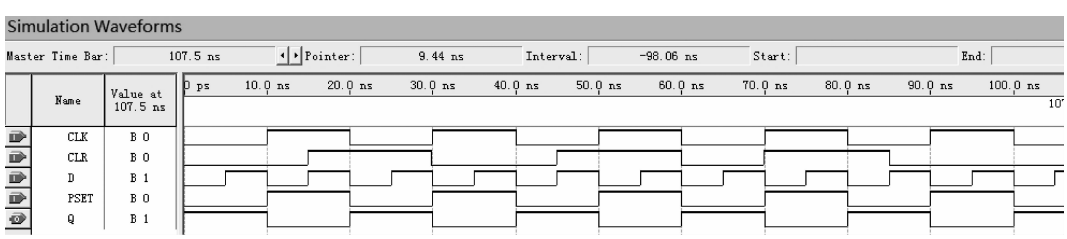

图 3.13　异步置位/复位 D 触发器的时序仿真图

【例 3.19】 同步复位 D 触发器的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY TFCFQ IS
    PORT(CLK,D,CLR:IN STD_LOGIC;
       Q:OUT STD_LOGIC);
END TFCFQ;
ARCHITECTURE RTL OF TFCFQ IS
BEGIN
  PROCESS(CLK)
  BEGIN
  IF(CLK'EVENT AND CLK  = '1')THEN
```

```
    IF (CLR = '1') THEN
     Q <= '0';
     ELSE
      Q <= D;
      END IF;
      END IF;
  END PROCESS;
END RTL;
```

图 3.14 为同步复位 D 触发器的逻辑示意图，图 3.15 为同步复位 D 触发器的时序仿真图。

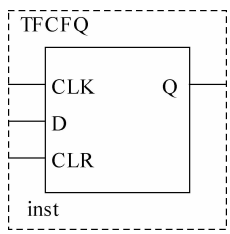

图 3.14　同步复位 D 触发器的逻辑示意图

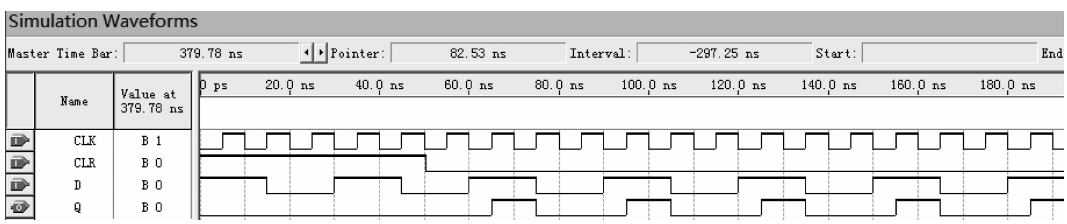

图 3.15　同步复位 D 触发器的时序仿真图

2. RS 触发器

【例 3.20】 RS 触发器的设计。RS 触发器的逻辑功能表见表 3.5。

表 3.5　RS 触发器的逻辑功能表

R	S	Q^{n+1}
1	0	1
0	1	0
0	0	Q^n
1	1	不定

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY RSCFQ IS
    PORT(R,S,CLK: IN STD_LOGIC;
        Q,QB: BUFFER  STD_LOGIC);
END RSCFQ;
ARCHITECTURE  ART  OF  RSCFQ IS
   SIGNAL  Q_S,QB_S: STD_LOGIC;
   BEGIN
   PROCESS(CLK,R,S)
```

```
    BEGIN
IF (CLK'EVENT AND CLK = '1')THEN
    IF(S = '1' AND R = '0') THEN
         Q_S < = '0';
    QB_S < = '1';
    ELSIF (S = '0' AND R = '1') THEN
      Q_S < = '1';
      QB_S < = '0';
    ELSIF (S = '0' AND R = '0') THEN
      Q_S < = Q_S;
      QB_S < = QB_S;
      END IF;
    END IF ;
      Q < = Q_S;
      QB < = QB_S;
    END PROCESS;
END ART;
```

3. JK 触发器

【例 3.21】 JK 触发器的设计。JK 触发器的逻辑功能表见表 3.6。

表 3.6 JK 触发器的逻辑功能表

J	K	CLK	Q^{n+1}
0	0		Q^n
0	1		0
1	0		1
1	1		$\overline{Q}^n$

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY JKCFQ IS
     PORT(J,K,CLK: IN STD_LOGIC;
            Q,QB: BUFFER STD_LOGIC);
END JKCFQ;
ARCHITECTURE ART OF JKCFQ IS
    SIGNAL Q_S,QB_S: STD_LOGIC;
    BEGIN
    PROCESS(CLK,J,K)
    BEGIN
       IF (CLK'EVENT AND CLK = '1')THEN
           IF(J = '0' AND K = '1') THEN
      Q_S < = '0';
      QB_S < = '1';
    ELSIF (J = '1' AND K = '0') THEN
      Q_S < = '1';
      QB_S < = '0';
    ELSIF (J = '1' AND K = '1') THEN
       Q_S < = NOT Q_S;
       QB_S < = NOT QB_S;
```

```
    END IF;
   END IF ;
   Q <= Q_S;
   QB <= QB_S;
 END PROCESS;
END ART;
```

4. T 触发器

设计一个不带置/复位的 T 触发器，数据输入端为 T，时钟输入端为 CLK，两个反相的输出端 Q，QB。原理：当 T＝0 时，T 触发器保持前一状态的值；当 T＝1 时，T 触发器状态在时钟边沿的作用下发生翻转。

【例 3.22】 T 触发器的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY TCFQ IS
      PORT(T,CLK:IN STD_LOGIC;
                 Q,QB:BUFFER STD_LOGIC);
END TCFQ;
ARCHITECTURE ART OF TCFQ IS
 SIGNAL BUF:STD_LOGIC;
 BEGIN
   PROCESS(CLK)
   BEGIN
       IF (CLK'EVENT AND CLK = '1')THEN
       IF(T = '1') THEN
       BUF <= NOT BUF;
       ELSE BUF <= BUF;
       END IF;
       END IF;
   END PROCESS;
  Q <= BUF;
 QB <= NOT BUF;
END ART;
```

图 3.16 为 T 触发器的时序仿真图。

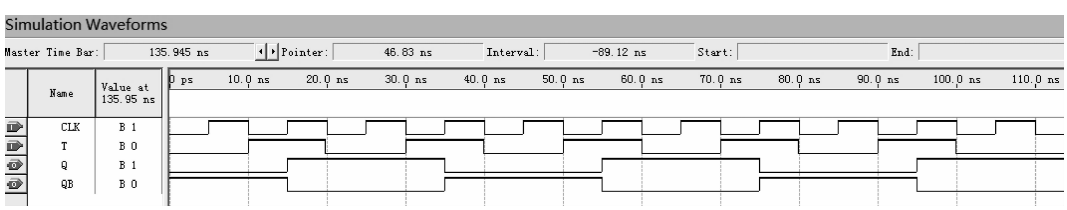

图 3.16　T 触发器的时序仿真图

3.2.2　锁存器

锁存器是一种用来暂时保存数据的逻辑电路，下面以 8 位锁存器 74LS373 为例，介绍锁存器的设计方法。74LS373 的逻辑符号如图 3.17 所示，功能表见表 3.7。其逻辑功能

为：当三态控制端口的信号有效(OE＝0)并且数据控制端口的信号也有效(G＝1)时，锁存器把输入端口的 8 位数据送到输出端口上去；当三态控制端口的信号有效(OE＝0)而数据控制端口的信号无效(G＝0)时，锁存器的输出端口将保持前一个状态；当三态控制端口的信号无效(OE＝1)时，锁存器的输出端口将处于高阻状态。

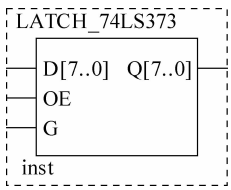

图 3.17　74LS373 的逻辑符号

表 3.7　锁存器 74LS373 的功能表

OE	G	D	Q
0	1	0	0
0	0	X	保持
1	X	X	高阻

【例 3.23】 8 位锁存器 74LS373 的源程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY LATCH_74LS373 IS
  PORT(D:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       OE,G: IN STD_LOGIC;
       Q:INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END LATCH_74LS373;
ARCHITECTURE RTL OF LATCH_74LS373 IS
 BEGIN
  PROCESS(OE,G)
    BEGIN
     IF (OE = '0')THEN
       IF (G = '1') THEN
          Q <= D;
       ELSE Q <= Q;
       END IF;
     ELSE
       Q <= "ZZZZZZZZ";
     END IF;
  END PROCESS;
END RTL;
```

8 位锁存器 74LS373 电路仿真波形如图 3.18 所示。从图中可以看出，当三态控制端口的信号有效(OE＝0)，并且数据控制端口的信号也有效(G＝1)时，锁存器把输入端口的 8 位数据 D 送到输出端口 Q；当三态控制端口的信号有效(OE＝0)而数据控制端口的信号无效(G＝0)时，锁存器的输出端口将保持前一个状态；当三态控制端口的信号无效(OE＝1)时，锁存器的输出端口将处于高阻状态，结果与理论值符合。

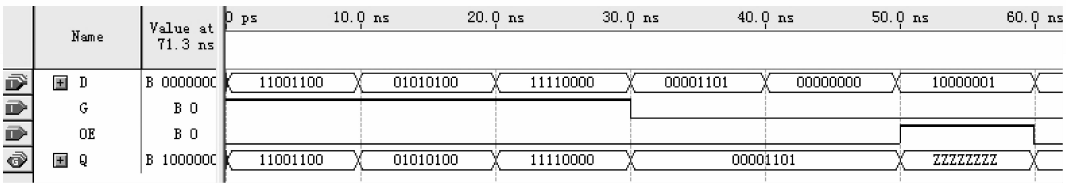

图 3.18　8 位锁存器 74LS373 的仿真波形

3.2.3　寄存器和移位寄存器

1. 寄存器的设计

寄存器是数字系统中用来存储二进制数据的逻辑电路。1 个触发器可存储 1 位二进制数据，存储 n 位二进制数据的寄存器需要用 n 个触发器组成。寄存器与锁存器具有类似的功能，两者的区别在于寄存器是同步时钟控制，而锁存器是电位信号控制。带使能端的 8 位寄存器的逻辑符号如图 3.19 所示，功能表见表 3.8。

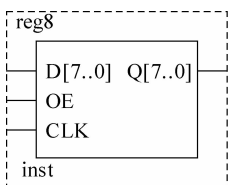

图 3.19　带使能端的 8 位寄存器的逻辑符号

表 3.8　带使能端的 8 位寄存器的功能表

输　　入			输　出
OE	CP	D	Q
0	↑	0	0
0	↑	1	1
0	0	X	保持
1	X	X	高阻

【例 3.24】　带使能端的 8 位寄存器的源程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY reg8 IS
  PORT(D:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       OE: IN STD_LOGIC;
       CLK:IN STD_LOGIC;
       Q:INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END reg8 ;
ARCHITECTURE RTL OF reg8   IS
 BEGIN
  PROCESS(OE,CLK)
    BEGIN
     IF (OE = '0')THEN
       IF (CLK'EVENT AND CLK = '1') THEN
```

```
            Q <= D;
          ELSE Q <= Q;
          END IF;
        ELSE
          Q <= "ZZZZZZZZ";
        END IF;
    END PROCESS;
END RTL;
```

带使能端的8位寄存器的电路仿真波形如图3.20所示。当使能端OE=0,并且时钟信号CLK上升沿到来时,寄存器把输入端口的8位数据D送到输出端口Q;当OE=1时,寄存器的输出端口将处于高阻状态。

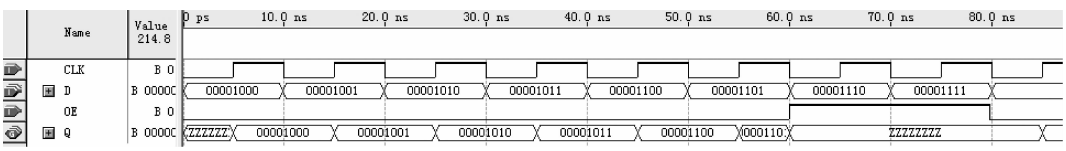

图3.20 带使能端的8位寄存器的电路仿真波形

2. 移位寄存器的设计

移位寄存器除了具有寄存数码的功能外,还具有移位功能,即在移位脉冲作用下,能够把寄存器中的数依次向右或向左移,它是一个同步时序逻辑电路。可预加载循环移位寄存器逻辑符号如图3.21所示。

【例3.25】 可预加载循环移位寄存器的源程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY YWJCQ IS
 PORT(D:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      LOAD,CLK:IN STD_LOGIC;
      QS: BUFFER STD_LOGIC;
      Q:BUFFER STD_LOGIC_VECTOR(7 DOWNTO 0));
END YWJCQ;
ARCHITECTURE RTL OF YWJCQ IS
 BEGIN
  PROCESS(CLK,LOAD,D)
   BEGIN
    IF (CLK'EVENT AND CLK = '1') THEN
      IF LOAD = '1' THEN
       Q <= D;
       QS <= '0';
      ELSE
       QS <= Q(7);
       Q(7 DOWNTO 1)<= Q(6 DOWNTO 0);
       Q(0)<= QS;
      END IF;
    END IF;
  END PROCESS;
END RTL;
```

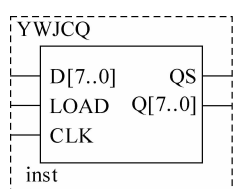

图3.21 可预加载循环移位寄存器逻辑符号

可预加载循环移位寄存器的波形如图 3.22 所示。从图中可以看出，当时钟信号 CLK 的上升沿到来时，如果加载信号有效即 LOAD＝1 时，寄存器把输入端口的 8 位数据 D＝00100011 送到输出端口 Q；LOAD＝0 时，在时钟信号 CLK 上升沿到来时，Q 低 7 位向左移一位，最低位 Q(0)等于 Q 的前一次状态 Q(7)，在移动脉冲的作用下，Q 的移动以此类推，结果正确。

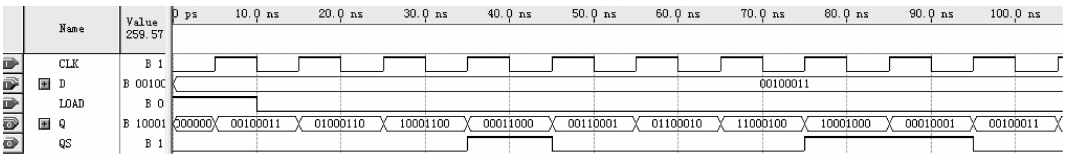

图 3.22　可预加载循环移位寄存器的仿真波形

3.2.4　计数器

计数器一般可分为同步计数器和异步计数器。同步计数器与异步计数器的不同，主要体现在对各级时钟脉冲的描述上。同步计数器指在时钟脉冲(计数脉冲)的控制下，构成计数器的各触发器状态同时发生变化。异步计数器又称为行波计数器，它的低位计数器的输出作为高位计数器的时钟信号。异步计数器采用行波计数，使计数延迟增加，计算器工作频率较低。

1. 同步计数器

【例 3.26】　一个模为 60，具有异步复位、同步置数功能的 8421BCD 码计数器。

```
LIBRARY IEEE;
  USE IEEE.STD_LOGIC_1164.ALL;
  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
  ENTITY CNTM60 IS
    PORT(CI: IN STD_LOGIC;
         NRESET: IN STD_LOGIC;
         LOAD: IN STD_LOGIC;
         D: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         CLK: IN STD_LOGIC;
         CO: OUT STD_LOGIC;
QH: BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);
         QL: BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0));
END CNTM60;
ARCHITECTURE ART OF CNTM60 IS
   BEGIN
   CO<= '1'WHEN(QH = "0101"AND QL = "1001"AND CI = '1')ELSE'0';
                                                    -- 进位输出的产生
PROCESS(CLK,NRESET)
BEGIN
   IF(NRESET = '0')THEN                             -- 异步复位
      QH<= "0000";
      QL<= "0000";
ELSIF(CLK'EVENT AND CLK = '1')THEN                  -- 同步置数
    IF(LOAD = '1')THEN
```

```
    QH <= D(7 DOWNTO 4)
    QL <= D(3 DOWNTO 0);
ELSIF(CI = '1')THEN                                    -- 模 60 的实现
    IF(QL = 9)THEN
        QL <= "0000";
         IF(QH = 5)THEN
              QH <= "0000";
         ELSE                                          -- 计数功能的实现
               QH <= QH + 1;
         END IF
     ELSE
         QL <= QL + 1;
     END IF;
    END IF;                                            -- END IF LOAD
  END PROCESS;
END ART;
```

2. 异步计数器

用 VHDL 语言描述异步计数器，与上述同步计数器不同之处主要表现在对各级时钟的描述上。

【例 3.27】 由 8 个触发器构成的异步计数器，采用元件例化的方式生成。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY DIFFR IS
 PORT(CLK,CLR,D:IN STD_LOGIC;
      Q,QB:OUT STD_LOGIC);
END DIFFR;
ARCHITECTURE ART1 OF DIFFR IS
SIGNAL Q_IN:STD_LOGIC;
 BEGIN
  Q <= Q_IN;
  QB <= NOT Q_IN;
 PROCESS(CLK,CLR)
  BEGIN
   IF(CLR = '1')THEN
     Q_IN <= '0';
   ELSIF (CLK'EVENT AND CLK = '1') THEN
     Q_IN <= D;
   END IF;
  END PROCESS;
 END ART1;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY RPLCOUNT IS
 PORT(CLK,CLR:IN STD_LOGIC;
      COUNT:OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END RPLCOUNT;
ARCHITECTURE ART2 OF  RPLCOUNT IS
  SIGNAL COUNT_IN:STD_LOGIC_VECTOR(8 DOWNTO 0);
```

```
  COMPONENT DIFFR
   PORT(CLK,CLR,D:IN STD_LOGIC;
        Q,QB:OUT STD_LOGIC);
  END COMPONENT;
  BEGIN
  COUNT_IN(0)<=CLK;
  GEN1:FOR I IN 0 TO 7 GENERATE
    U:DIFFR PORT MAP(CLK=>COUNT_IN(I),CLR=>CLR,
     D=>COUNT_IN(I+1),Q=>COUNT_IN(I),
     QB=>COUNT_IN(I+1));
      END GENERATE;
END ART2;
```

3. 可逆计数器

8位二进制加减计数器的元件符号如图3.23所示，CLR是复位控制输入端；ENA是使能控制输入端；LOAD是预置控制输入端；D[7..0]是8位并行数据输入端；UPDOWN是加减控制输入端，当UPDOWN＝0时，计数器作加法操作，UPDOWN＝1时，计数器作减法操作；COUNT是进/借位输出端。

【例3.28】 用VHDL描述的8位二进制加减计数器源程序。

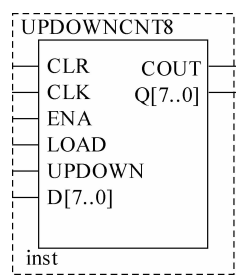

图3.23　8位二进制加减计数器的元件符号

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY UPDOWNCNT8 IS
  PORT(CLR,CLK,ENA,LOAD,UPDOWN:IN STD_LOGIC;
       D:IN INTEGER RANGE 0 TO 255;
       COUT: OUT STD_LOGIC;
      Q:BUFFER INTEGER RANGE 0 TO 255);
END UPDOWNCNT8;
ARCHITECTURE ONE OF UPDOWNCNT8 IS
 BEGIN
   PROCESS(CLR,CLK,ENA,LOAD,UPDOWN,D)
   BEGIN
    IF CLR='0' THEN Q<=0;
    ELSIF CLK'EVENT AND CLK='1' THEN
      IF LOAD='1' THEN Q<=D;
      ELSIF ENA='1' THEN
          IF UPDOWN='0' THEN Q<=Q+1;
              IF Q=255 THEN COUT<='1';
              END IF;
              ELSE Q<=Q-1;
              IF Q=0 THEN COUT<='0';
              END IF;
          END IF;
       END IF;
     END IF;
   END PROCESS;
END ONE;
```

3.2.5 分频器

分频器就是对较高频率的信号进行分频，得到较低频率的信号。

【例 3.29】 利用计数器实现 2-4 分频器。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DIV IS
   PORT(RESET,CLK:IN STD_LOGIC;
        CLK_2,CLK_4:OUT STD_LOGIC);
END DIV;
ARCHITECTURE RTL OF DIV IS
SIGNAL COUNT:STD_LOGIC_VECTOR(1 DOWNTO 0);
 BEGIN
    PROCESS(RESET,CLK)
     BEGIN
       IF (RESET = '0') THEN COUNT <= "00";          -- 异步复位
       ELSIF(CLK'EVENT AND CLK = '1') THEN
        COUNT <= COUNT + 1;
       ELSE NULL;       -- 注意当使用 IF 条件语句判断时钟沿有效时,后面不能有 ELSE 分支,如果
                        -- 必须写,就只能写 ELSE NULL
       END IF;
    END PROCESS;
    CLK_2 <= COUNT(0);
    CLK_4 <= COUNT(1);
END RTL;
```

分频器的本质是计数器，在本例中之所以设定两位的中间信号 COUNT 计数，是因为根据要求要设计 2 分频和 4 分频，就相当于设计一个计数器，计数周期是 2 和 4，可计数范围为 0～1 和 0～3。图 3.24 为 2-4 分频器的逻辑示意图，图 3.25 为 2-4 分频器的时序仿真图。

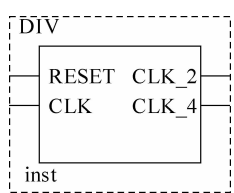

图 3.24 2-4 分频器的逻辑示意图

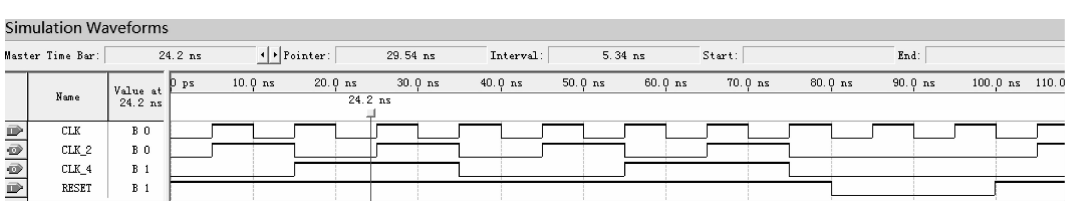

图 3.25 2-4 分频器的时序仿真图

3.2.6　序列发生器和检测器

1. 序列信号发生器

在数字信号的传输和数字系统的测试中，有时需要用到一组特定的串行数字信号，产生序列信号的电路称为序列信号发生器。

要求信号发生器周期性地循环输出“01111110”序列信号，其本质与计数器一样，设计核心仍然是一个计数器。因为按一个循环规律变化。所以只要设定一个一定长度的计数器，确定在计数器计数过程中对应的状态与要求序列状态一致，就可以利用这种方式实现任意序列信号发生器。

【例 3.30】 “01111110”序列发生器。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SENQGEN IS
 PORT(CLK,CLR,CLOCK:IN STD_LOGIC;
      ZO:OUT STD_LOGIC);
END SENQGEN;
ARCHITECTURE ART OF SENQGEN IS
SIGNAL COUNT:STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL Z:STD_LOGIC := '0';
 BEGIN
  PROCESS(CLK,CLR)                              -- 3 位计数器,产生 8 个状态
  BEGIN
    IF(CLK'EVENT AND CLK = '1')THEN
      IF(CLR = '0' OR COUNT = "111")THEN
        COUNT <= "000";
      ELSE
        COUNT <= COUNT + 1;
      END IF;
    END IF;
  END PROCESS;
  PROCESS(COUNT)                                -- 根据计数器状态产生对应的输出状态
  BEGIN
   CASE COUNT IS
     WHEN"000" => Z <= '0';
     WHEN"001" => Z <= '1';
     WHEN"010" => Z <= '1';
     WHEN"011" => Z <= '1';
     WHEN"100" => Z <= '1';
     WHEN"101" => Z <= '1';
     WHEN"110" => Z <= '1';
     WHEN OTHERS => Z <= '0';
    END CASE;
   END PROCESS;
  PROCESS(CLOCK,Z)                              -- 消除毛刺的锁存器
  BEGIN
    IF(CLOCK'EVENT AND CLOCK = '1')THEN
      ZO <= Z;
     END IF;
  END PROCESS;
END ART;
```

图 3.26 为“01111110”序列发生器的逻辑示意图，图 3.27 为“01111110”序列发生器的时序仿真图。

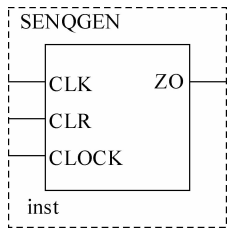

图 3.26 “01111110”序列发生器的逻辑示意图

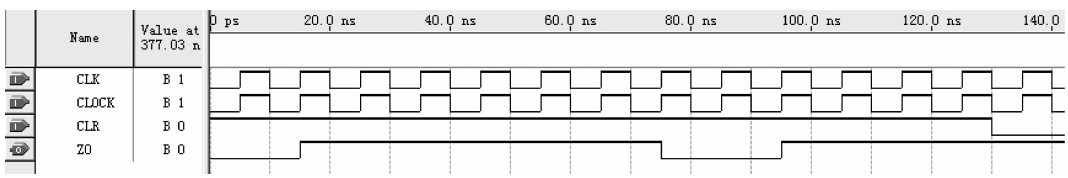

图 3.27 “01111110”序列发生器的时序仿真图

2. 序列信号检测器

检测 8 位串行输入序列信号，可以采用对输入的每一位串行数据进行检测。如果检测到某一位跟输入相对应时，就检测下一位。如果不一致，就回到起点重新检测，具体可以利用 case 语句来实现。

【例 3.31】 “01111110”序列信号检测器的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY DETECT IS
 PORT(DATAIN:IN STD_LOGIC;
      CLK:IN STD_LOGIC;
      Q:OUT STD_LOGIC);
END DETECT;
ARCHITECTURE ART OF DETECT IS
 TYPE STATETYPE IS(S0,S1,S2,S3,S4,S5,S6,S7,S8);    --定义了一个新的枚举类型
 BEGIN
  PROCESS
  VARIABLE PRESENT_STATE:STATETYPE;
   BEGIN
     Q<='0';
     CASE PRESENT_STATE IS
     WHEN S0=>
          IF DATAIN='0' THEN PRESENT_STATE:=S1;
          ELSE PRESENT_STATE:=S0;
          END IF;
     WHEN S1=>
          IF DATAIN='1' THEN PRESENT_STATE:=S2;
          ELSE PRESENT_STATE:=S1;
          END IF;
     WHEN S2=>
```

```
            IF DATAIN = '1' THEN PRESENT_STATE := S3;
            ELSE PRESENT_STATE := S1;
            END IF;
        WHEN S3 = >
            IF DATAIN = '1' THEN PRESENT_STATE := S4;
            ELSE PRESENT_STATE := S1;
            END IF;
        WHEN S4 = >
            IF DATAIN = '1' THEN PRESENT_STATE := S5;
            ELSE PRESENT_STATE := S1;
            END IF;
        WHEN S5 = >
            IF DATAIN = '1' THEN PRESENT_STATE := S6;
            ELSE PRESENT_STATE := S1;
            END IF;
        WHEN S6 = >
            IF DATAIN = '1' THEN PRESENT_STATE := S7;
            ELSE PRESENT_STATE := S1;
            END IF;
        WHEN S7 = >
            IF DATAIN = '0' THEN PRESENT_STATE := S8;
            Q < = '1';
            ELSE PRESENT_STATE := S0;
            END IF;
        WHEN S8 = >
            IF DATAIN = '0' THEN PRESENT_STATE := S1;
            ELSE PRESENT_STATE := S2;
            END IF;
        END CASE;
        WAIT UNTIL CLK = '1';
    END PROCESS;
END ART;
```

图 3.28 为“01111110”序列检测器的逻辑示意图，图 3.29 为“01111110”序列检测器的时序仿真图。

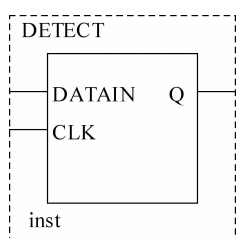

图 3.28　“01111110”序列检测器的逻辑示意图

图 3.29　“01111110”序列检测器的时序仿真图

上述采用了逐步判断进行检测的设计方式，设计思路比较清晰，但要检测长序列信号的话，如16位、32位或更多，就要对每一位输入信号进行判断，导致程序书写太长，变得复杂，中间判断过程较多也会容易出错。另外，当检测序列长度发生变化时，整个设计要做出较大改动。因此可以考虑一种较简单的设计方式，如例3.32。该例设计思想是将输入信号在时钟脉冲的控制下逐位移入一组寄存器里，然后判断寄存器内的值是否与原序列信号的值相一致。移位操作时重复过程，只需要做一次描述即可。

【例3.32】 简洁的序列信号检测器的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY DETECT IS
  PORT(DATAIN,CLK:IN STD_LOGIC;
       Q:OUT STD_LOGIC);
END DETECT;
ARCHITECTURE ART OF DETECT IS
SIGNAL REG:STD_LOGIC_VECTOR(7 DOWNTO 0);
 BEGIN
  PROCESS(CLK)
  BEGIN
   IF(CLK'EVENT AND CLK = '1') THEN
     REG(0)<= DATAIN;
     REG(7 DOWNTO 1)<= REG(6 DOWNTO 0);
   END IF;
   IF REG = "01111110" THEN
       Q<= '1';
   ELSE
      Q<= '0';
   END IF;
  END PROCESS;
END ART;
```

图3.30为简洁序列检测器的逻辑示意图，图3.31为简洁序列检测器的时序仿真图。

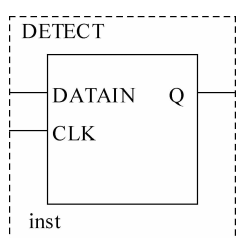

图3.30 简洁序列检测器的逻辑示意图

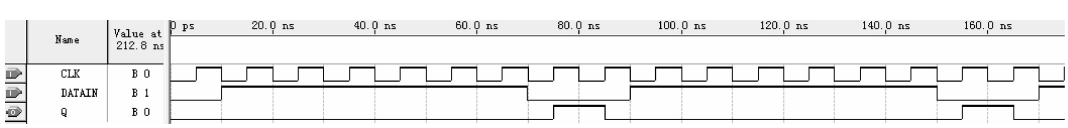

图3.31 简洁序列检测器的时序仿真图

3.3　存储器设计

半导体存储器的种类很多，从功能上可以分为只读存储器（Read Only Memory，ROM）和随机存储器（Random Access Memory，RAM）两大类。ROM 和 RAM 属于通用大规模器件，一般不需要自行设计，特别是采用 PLD 进行设计时。但是在数字系统中，有时也需要设计一些小型的存储器件，用于特定的用途：临时存放数据、构成查表运算等。此类器件的特点为地址与存储内容直接对应，设计时将输入地址作为给出输出内容的条件。

3.3.1　只读存储器 ROM

只读存储器在正常工作时可以从中读取数据，不能快速地修改或重新写入数据，适用于存储固定数据的场合，通常采用电路的固定结构来实现存储。

【例 3.33】　8×8 位 ROM 的设计。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ROM IS
  PORT(ADDR:IN INTEGER RANGE 0 TO 7;
       EN:IN STD_LOGIC;
       Q:OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ROM;
ARCHITECTURE ART OF ROM IS
 BEGIN
  PROCESS(EN,ADDR)
   BEGIN
    IF (EN = '1') THEN Q <= "ZZZZZZZZ";
    ELSE
    CASE ADDR IS
     WHEN 0 => Q <= "01000001";
     WHEN 1 => Q <= "01000010";
     WHEN 2 => Q <= "01000011";
     WHEN 3 => Q <= "01000100";
     WHEN 4 => Q <= "01000101";
     WHEN 5 => Q <= "01000110";
     WHEN 6 => Q <= "01000111";
     WHEN 7 => Q <= "01001000";
    END CASE;
    END IF;
   END PROCESS;
END ART;
```

由 VHDL 源代码生成的 8×8 位 ROM 的逻辑示意图如图 3.32 所示。其中，ADDR[2..0]是地址输入端，EN 是使能控制输入端，当 EN＝1 时，ROM 不能工作，输出 Q[7..0]为高阻态；当 EN＝0 时，ROM 工作，其输出的数据由输入地址决定。图 3.33 为 8×8 位 ROM 的时序仿真图。

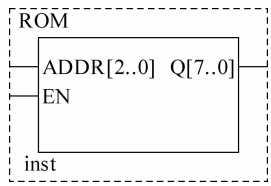

图 3.32　8×8 位 ROM 的逻辑示意图

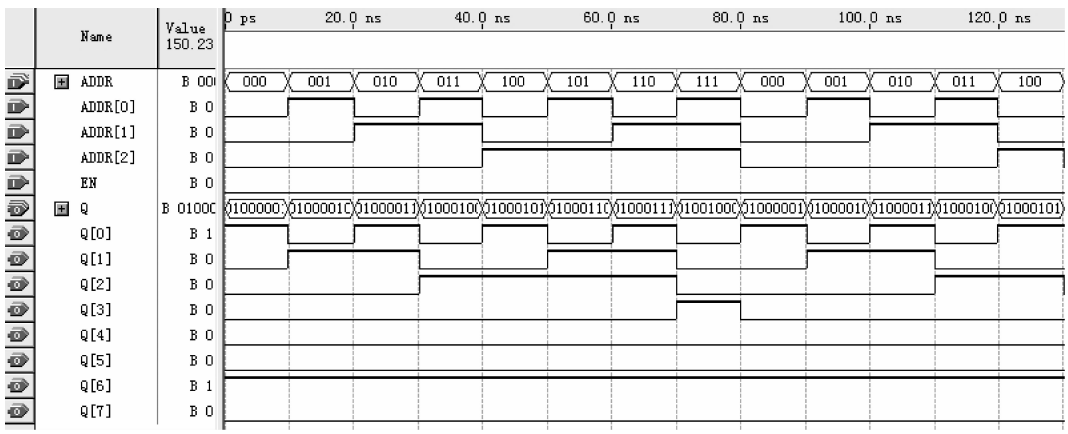

图 3.33　8×8 位 ROM 的时序仿真图

3.3.2　随机存储器 RAM

RAM 的用途是存储数据，其指标为存储容量和字长。RAM 的内部可以分为地址译码和存储单元两部分。

【例 3.34】　RAM 的设计。该电路界面端口由读写选通控制端(RD，WR)、片选端(CS)、时钟信号(CLK)、地址线(ADR)、数据输入线(DIN)和数据输出线(DOUT)组成。RAM 根据地址信号经由译码电路选择欲读写的存储单元。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY RAM IS
  GENERIC(WIDTH:INTEGER := 8;
          DEPTH:INTEGER := 16);
  PORT(RD,WR,CS,CLK:IN STD_LOGIC;                          -- 写读片选控制信号、时钟信号
       ADR:IN INTEGER RANGE 0 TO DEPTH - 1;                -- 地址信号
       DIN:IN STD_LOGIC_VECTOR(WIDTH - 1 DOWNTO 0);        -- 8 位输入信号
       DOUT:OUT STD_LOGIC_VECTOR(WIDTH - 1 DOWNTO 0));     -- 8 位输出信号
END RAM;
ARCHITECTURE ART OF RAM IS
TYPE MEMORY IS ARRAY(0 TO DEPTH - 1)OF STD_LOGIC_VECTOR(WIDTH - 1 DOWNTO 0);
SIGNAL RAM:MEMORY;
 BEGIN
```

```
WRITE:PROCESS(CS,WR,RD,DIN)                          -- 数据写入进程
        BEGIN
        IF(WR = '0' AND CS = '0' AND RD = '1')THEN
          IF CLK'EVENT AND CLK = '1' THEN
              RAM(ADR)<= DIN;
          END IF;
        END IF;
        END PROCESS;
READ:PROCESS(CS,WR,RD,ADR)                           -- 数据读出进程
        BEGIN
        IF(RD = '0' AND CS = '0' AND WR = '1')THEN
          DOUT <= RAM(ADR);
        ELSE
          DOUT <= (OTHERS =>'Z');
        END IF;
      END PROCESS;
END ART;
```

程序中有两个进程，一个是数据写入进程 WRITE，当 WR＝0 时，在满足条件(CS＝0 AND RD＝1)时，将外部 8 位数据 DIN 锁进指定地址 ADR 的 RAM 单元中。而当满足条件(RD＝0 AND CS＝0 AND WR＝1)时，此 RAM 将指定地址 ADR 的 RAM 单元中的数据向 DOUT 端口输出，否则该端口呈高阻态。图 3.34 为 RAM 存储器的时序仿真图。

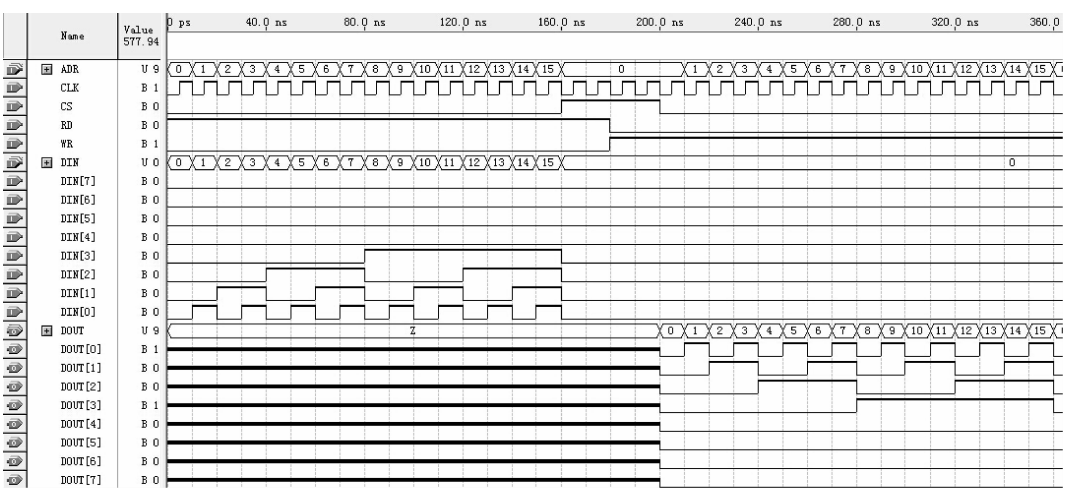

图 3.34　RAM 存储器的时序仿真图

3.4　状态机设计

状态机(State Machine)是一类很重要的时序电路，是很多逻辑电路的核心部件，是实现高效率、高可靠性逻辑控制的重要途径。状态机相当于一个控制器，它将一项功能的完成分解为若干步，每一步对应于二进制的一个状态，通过预先设计的顺序在各状态之间进行转换，状态转换的过程就是实现逻辑功能的过程。

状态机根据输出信号与当前状态以及输入信号的关系来分，可以分为摩尔(Moore)型和米立(Mealy)型。输出信号只和当前状态有关的状态机称为 Moore 型状态机，如图 3.35(a)所示。输出信号不仅和当前状态有关，而且也和输入信号有关的状态机称为 Mealy 型状态机，如图 3.35(b)所示。

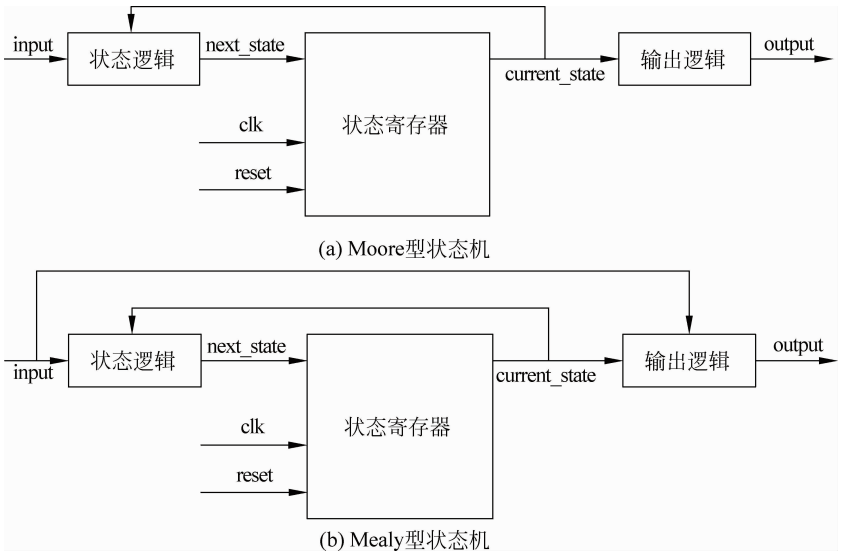

图 3.35　状态机的结构示意图

状态机的表达方式和功能不尽相同，但都有相对固定的语句和程序结构。状态机主要由 4 部分组成，但这 4 部分并非都是必需的。它们可以进行各种不同形式的变换，但基本构成思想是一样的。下面简要说明各部分的作用。

(1) 说明部分：定义枚举型数据类型 Type state is(s0,s1,s2,…)；定义现态信号 current_state 和次态信号 next_state。

(2) 主控时序进程：在时钟驱动下负责状态转换。只是机械地将代表次态信号的 next_state 中的内容送入现态信号的 current_state 中。

(3) 主控组合进程：根据外部输入的控制信号和当前状态确定下一状态的取向，以及确定当前对外的输出。

(4) 辅助进程：为了完成某种算法或为了输出设置的锁存器。

3.4.1　Moore 型状态机

【例 3.35】 完成双向步进电动机控制的 VHDL 设计。该控制电路的输入信号有 3 个：时钟信号 clk，复位信号 reset 和方向控制信号 dir。输出信号为 phase[3..0]，用来控制步进电动机的动作。当方向控制信号 dir 为 1 时，要求输出信号 phase[3..0]按照 0001、0010、0100、1000、0001 的顺序循环变化。当方向控制信号 dir 为 0 时，要求输出信号 phase[3..0]按照 1000、0100、0010、0001、1000 的顺序循环变化。

根据控制器的功能要求，画出状态转换图和状态与输出信号的关系，分别如图 3.36 和表 3.9 所示。

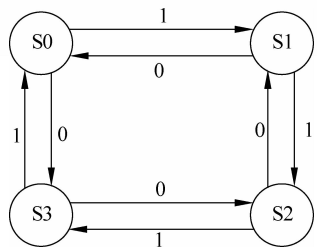

图 3.36　步进电动机控制器的状态转换图

表 3.9　步进电动机状态与输出信号的对应关系

状　态	输出信号 phase[3..0]
S0	0001
S1	0010
S2	0100
S3	1000

步进电动机控制器的 VHDL 程序如下。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY stepmotor IS
 PORT(clk,reset:IN STD_LOGIC;
       dir:in STD_LOGIC;
       phase:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END stepmotor;
ARCHITECTURE  ART OF stepmotor IS
TYPE states IS (S0,S1,S2,S3);
SIGNAL current_state:states;
 BEGIN
  PROCESS(clk)
   BEGIN
    IF clk'EVENT AND clk = '1' THEN
     IF reset = '1'  THEN
      current_state <= S0;
      ELSE
      CASE current_state IS
       WHEN S0 =>
          IF dir = '1' THEN
            current_state <= S1;
          ELSE
           current_state <= S3;
          END IF;
       WHEN S1 =>
          IF dir = '1' THEN
            current_state <= S2;
          ELSE
           current_state <= S0;
          END IF;
```

```
            WHEN S2 = >
             IF dir = '1' THEN
               current_state < = S3;
             ELSE
              current_state < = S1;
             END IF;
           WHEN S3 = >
             IF dir = '1' THEN
               current_state < = S0;
             ELSE
              current_state < = S2;
             END IF;
           WHEN OTHERS = >
              current_state < = S0;
          END CASE;
        END IF;
      END IF;
    END PROCESS;
    PROCESS(current_state)
     BEGIN
      CASE current_state IS
       WHEN S0 = > phase < = "0001";
       WHEN S1 = > phase < = "0010";
       WHEN S2 = > phase < = "0100";
       WHEN S3 = > phase < = "1000";
      END CASE;
     END PROCESS;
END ARCHITECTURE ART;
```

步进电动机控制器的仿真波形图如图 3.37 所示。

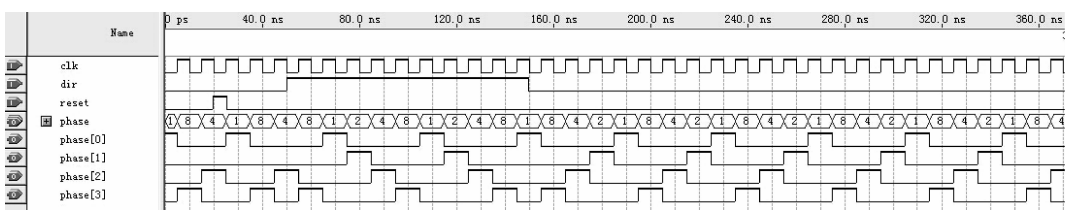

图 3.37　步进电动机控制器的仿真波形图

【例 3.36】 完成自动售货机的 VHDL 设计。要求：有两种硬币：1 元或 5 角，投入 1 元 5 角硬币输出货物，投入 2 元硬币输出货物并找 5 角零钱。

状态定义：S0 表示初态，S1 表示投入 5 角硬币，S2 表示投入 1 元硬币，S3 表示投入 1 元 5 角硬币，S4 表示投入 2 元硬币。

输入信号：state_input(0)表示投入 1 元硬币，state_input(1)表示投入 5 角硬币。输入信号为 1 表示投入硬币，输入信号为 0 表示未投入硬币。

输出信号：comb_outputs(0)表示输出货物，comb_outputs(1)表示找 5 角零钱。输出信号为 1 表示输出货物或找钱，输出信号为 0 表示不输出货物或不找钱。

根据设计要求分析，得到状态转换图如图 3.38 所示。其中状态为 S0、S1、S2、S3 和 S4；

输入为 state_input(0,1)；输出为 comb_outputs(0,1)；输出仅与状态有关，因此将输出写在状态圈内部。

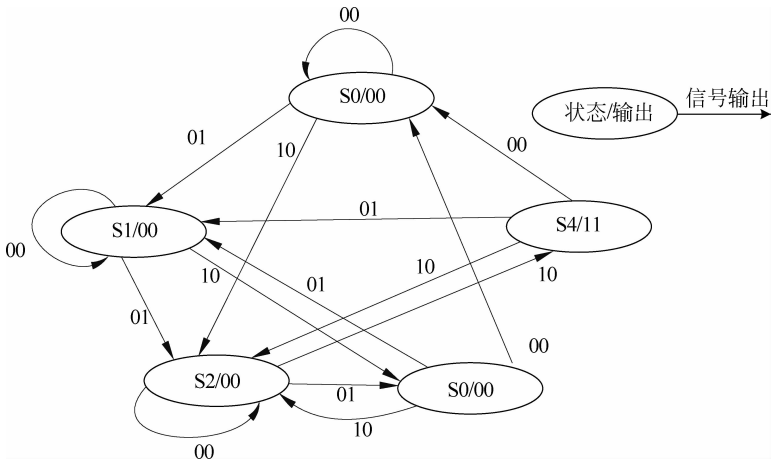

图 3.38　自动售货机的状态转换图

根据图 3.38 所示状态转换图设计的 VHDL 程序清单如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SHOUHUOJI IS
  PORT(clk,reset:IN STD_LOGIC;
       state_inputs:IN STD_LOGIC_VECTOR(0 TO 1);
       comb_outputs:OUT STD_LOGIC_VECTOR(0 TO 1));
END SHOUHUOJI;
ARCHITECTURE ART OF SHOUHUOJI IS
 TYPE fsm_st IS(S0,S1,S2,S3,S4);
 SIGNAL current_state,next_state:fsm_st;
 BEGIN
 reg:PROCESS(reset,clk)
  BEGIN
  IF reset = '1' THEN current_state <= S0;
  ELSIF rising_edge(clk) THEN
   current_state <= next_state;
  END IF;
  END PROCESS;
 corn:PROCESS(current_state,state_inputs)
   BEGIN
    CASE current_state IS
    WHEN S0 => comb_outputs <= "00";
       IF     state_inputs = "00" THEN next_state <= S0;
       ELSIF state_inputs = "01" THEN next_state <= S1;
       ELSIF state_inputs = "10" THEN next_state <= S2;
       END IF;
    WHEN S1 => comb_outputs <= "00";
       IF     state_inputs = "00" THEN next_state <= S1;
       ELSIF state_inputs = "01" THEN next_state <= S2;
```

```
        ELSIF state_inputs = "10" THEN next_state <= S3;
        END IF;
    WHEN S2 => comb_outputs <= "00";
        IF    state_inputs = "00" THEN next_state <= S2;
        ELSIF state_inputs = "01" THEN next_state <= S3;
        ELSIF state_inputs = "10" THEN next_state <= S4;
        END IF;
    WHEN S3 => comb_outputs <= "10";
        IF    state_inputs = "00" THEN next_state <= S0;
        ELSIF state_inputs = "01" THEN next_state <= S1;
        ELSIF state_inputs = "10" THEN next_state <= S2;
        END IF;
    WHEN S4 => comb_outputs <= "11";
        IF    state_inputs = "00" THEN next_state <= S0;
        ELSIF state_inputs = "01" THEN next_state <= S1;
        ELSIF state_inputs = "10" THEN next_state <= S2;
        END IF;
    END CASE;
  END PROCESS;
END ART;
```

3.4.2 Mealy 型状态机

与 Moore 型状态机不同，Mealy 型状态机的输出信号不仅与当前状态有关，而且与输入信号有关，因此输入信号可以直接影响输出信号，不依赖于时钟的同步，属于异步时序的概念。下面以一个具体的例子来讲述 Mealy 型状态机的设计过程，该状态机也是将状态寄存器和次态逻辑用一个进程描述，将输出逻辑用另一个进程描述。

【例 3.37】 Mealy 型状态机的 VHDL 程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY MEALY IS
 PORT(clk,rst: IN STD_LOGIC;
      id:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      y:OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END MEALY;
ARCHITECTURE ART OF MEALY IS
TYPE states IS(state0,state1,state2,state3,state4);
SIGNAL state:states;
SIGNAL y1:STD_LOGIC_VECTOR(1 DOWNTO 0);
 BEGIN
  PROCESS(clk,rst)
   BEGIN
    IF rst = '1' THEN
      state <= state0;
    ELSIF (clk'EVENT and clk = '1') THEN
      CASE state IS
        WHEN state0 =>
```

```
            IF id = x"3" THEN state <= state1;
            ELSE state <= state0;
            END IF;
          WHEN state1 =>
            IF id = x"1" THEN state <= state2;
            ELSE state <= state1;
            END IF;
          WHEN state2 =>
            IF id = x"7" THEN state <= state3;
            ELSE state <= state2;
            END IF;
          WHEN state3 =>
            IF id = x"B" THEN state <= state4;
            ELSE state <= state3;
            END IF;
          WHEN state4 =>
            IF id = x"5" THEN state <= state0;
            ELSE state <= state4;
            END IF;
          WHEN OTHERS => state <= state0;
        END CASE;
      END IF;
    END PROCESS;
  PROCESS(state,clk,id)
   VARIABLE tmp:STD_LOGIC_VECTOR(1 DOWNTO 0);
    BEGIN
     CASE state IS
      WHEN state0 => IF id = x"3" THEN tmp := "01";
                     ELSE  tmp := "00";
                     END IF;
      WHEN state1 => IF id = x"1" THEN tmp := "10";
                     ELSE  tmp := "00";
                     END IF;
      WHEN state2 => IF id = x"7" THEN tmp := "11";
                     ELSE  tmp := "00";
                     END IF;
      WHEN state3 => IF id = x"B" THEN tmp := "00";
                     ELSE  tmp := "00";
                     END IF;
      WHEN state4 => IF id = x"5" THEN tmp := "01";
                     ELSE  tmp := "00";
                     END IF;
      WHEN OTHERS => y <= "00";
     END CASE;
     IF clk'EVENT and clk = '1' THEN
      y1 <= tmp;
     END IF;
  END PROCESS;
  y <= y1;
END ART;
```

Mealy 状态机的仿真波形图如图 3.39 所示。

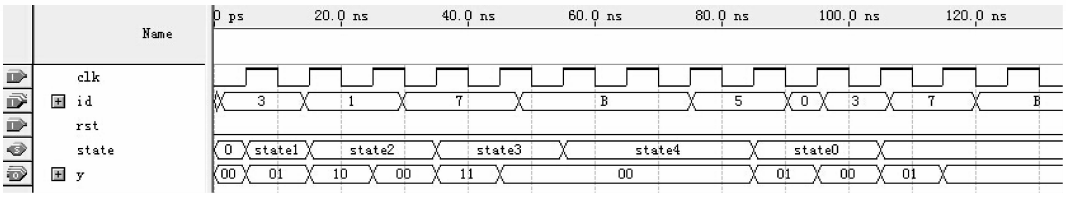

图 3.39　Mealy 状态机的仿真波形图

习题 3

1. 设计并实现一个 4 选 1 数据选择器。

2. 用组合逻辑设计一个 4 位二进制数乘法电路。

3. 设计一个 16 位加法器，要求由 4 个并行进位的 4 位加法器串联而成。

4. 设计一个带有异步清零端和异步置位端的十进制可逆计数器。

5. 什么叫状态机？状态机的基本结构是什么？状态机的种类有哪些？

6. 设计一个带同步预置功能的 16 位加减计数器。

7. 采用结构化描述，设计一个全减器。

8. 利用状态机的 VHDL 描述方法设计一个序列信号检测器，要求连续输入 4 个或 4 个以上的 1 时输出为 1，否则输出为 0。