现代计算机组成原理

潘 明　潘松 编著

科学出版社

第 3 章

VHDL 深入

3.1 数据对象

3.1.1 常数

CONSTANT 常数名：数据类型 **:=** 表达式 ；

```
CONSTANT  FBT : STD_LOGIC_VECTOR := "010110" ;
                                          -- 标准位矢类型
CONSTANT  DATAIN : INTEGER := 15 ;        -- 整数类型
```

3.1 数据对象

3.1.2 变量

VARIABLE 变量名 **:** 数据类型 **:=** 初始值 **;**

```
VARIABLE  a : INTEGER RANGE 0 TO 15 ;
```

--变量a定义为常数，取值范围是0到5

```
VARIABLE  d : STD_LOGIC := '1' ;
```

--变量a定义为标准逻辑位数据类型，初始值是1

3.1 数据对象

3.1.2 变量

目标变量名 := 表达式 ;

VARIABLE x，y : INTEGER RANGE 15 DOWNTO 0 ;--分别定义变量**x**和**y**为整数类型

VARIABLE a，b : STD_LOGIC_VECTOR(7 DOWNTO 0) ;

x := 11 ;

y := 2 + x ; -- 运算表达式赋值，y 也是实数变量

a := b --b向a赋值

a(0 TO 5) := b(2 TO 7) ;

3.1 数据对象

3.1.3 信号

SIGNAL 信号名： 数据类型 **:=** 初始值 ；

目标信号名 <= 表达式 **AFTER** 时间量;

```
    SIGNAL a，b，c，y，z: INTEGER ；
   ...
PROCESS (a，b，c)
BEGIN
  y <= a + b ;
  z <= c – a ;
  y <= b ;
    END PROCESS ;
```

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

表3-1 信号与变量赋值语句功能的比较

	信号SIGNAL	变量VARIABLE
基本用法	用于作为电路中的信号连线	用于作为进程中局部数据存储单元
适用范围	在整个结构体内的任何地方都能适用	只能在所定义的进程中使用
行为特性	在进程的最后才对信号赋值	立即赋值

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

【例3-1】

```
ARCHITECTURE bhv OF DFF3 IS
 BEGIN
 PROCESS (CLK)
  VARIABLE QQ : STD_LOGIC ;
  BEGIN
   IF  CLK'EVENT AND CLK = '1'  THEN QQ := D1 ;
   END IF;
END PROCESS ;
     Q1 <= QQ;
END ;
```

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

【例3-2】

```
ARCHITECTURE bhv OF DFF3 IS
    SIGNAL QQ : STD_LOGIC ;
   BEGIN
   PROCESS (CLK)
    BEGIN
     IF  CLK'EVENT AND CLK = '1'  THEN  QQ <= D1 ;
     END IF;
  END PROCESS ;
       Q1 <= QQ;
 END ;
```

3.1.4 进程中的信号赋值与变量赋值

【例3-3】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY DFF3 IS
PORT ( CLK,D1 : IN STD_LOGIC ;
             Q1  : OUT STD_LOGIC  ) ;
END ;
 ARCHITECTURE bhv OF DFF3 IS
   SIGNAL A,B : STD_LOGIC ;
  BEGIN
  PROCESS (CLK) BEGIN
    IF  CLK'EVENT AND CLK = '1'  THEN
      A <= D1 ; B <= A ; Q1 <= B ;
END IF;
  END PROCESS ;
END ;
```

3.1.4 进程中的信号赋值与变量赋值

【例3-4】

```
LIBRARY IEEE ;
 USE IEEE.STD_LOGIC_1164.ALL ;
 ENTITY DFF3 IS
 PORT ( CLK,D1 : IN STD_LOGIC ;
                Q1 : OUT STD_LOGIC  ) ;
 END ;
  ARCHITECTURE bhv OF DFF3 IS
   BEGIN
   PROCESS (CLK)
    VARIABLE A,B : STD_LOGIC ;
    BEGIN
     IF  CLK'EVENT AND CLK = '1'  THEN
       A := D1 ; B := A ;  Q1 <= B ;
     END IF;
  END PROCESS ;
    END ;
```

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

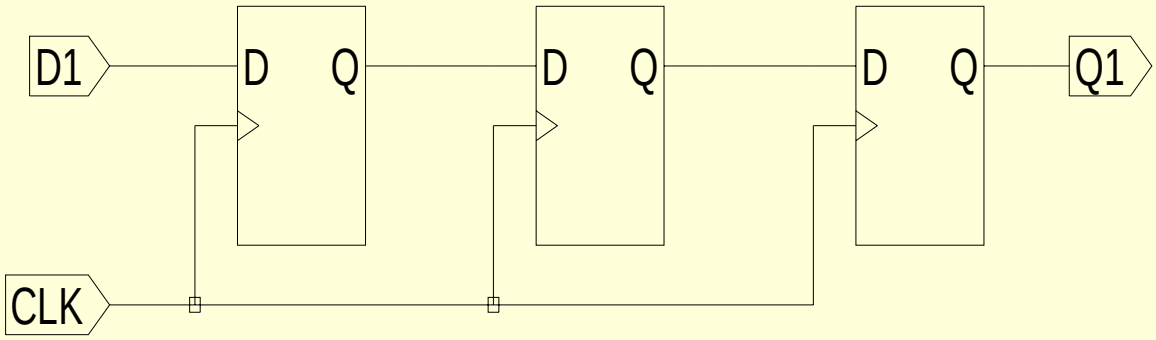


图3-1 例3-3的RTL电路

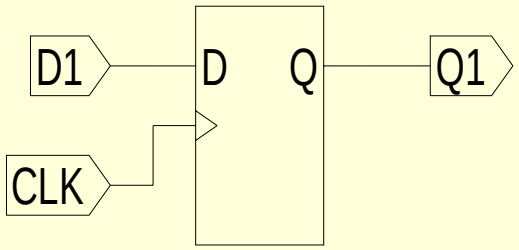


图3-2 D触发器电路

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

【例3-5】

```
SIGNAL  in1, in2, e1,  ... :  STD_LOGIC ;
...
PROCESS(in1, in2,  . . .)
VARIABLE c1, . . . : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
  BEGIN
    IF in1 = '1' THEN ...                  -- 第 1 行
        e1 <= "1010" ;                       -- 第 2 行
             ...
    IF in2 = '0' THEN . . .                -- 第 15+n 行
             ...
        c1 := "0011" ;                       -- 第 30+m 行
             ...
    END IF;
END PROCESS;
```

【例3-6】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux4 IS
PORT (i0, i1, i2, i3, a, b : IN STD_LOGIC;
      q : OUT STD_LOGIC);
END mux4;
ARCHITECTURE body_mux4 OF mux4 IS
signal muxval : integer range 7 downto 0;
BEGIN
process(i0,i1,i2,i3,a,b)
begin
                        muxval <= 0;
if (a = '1') then   muxval <= muxval + 1; end if;
if (b = '1') then   muxval <= muxval + 2; end if;
case muxval is
     when 0 => q <= i0;
     when 1 => q <= i1;
     when 2 => q <= i2;
     when 3 => q <= i3;
     when others => null;
end case;
end process;
END body_mux4;
```

【例3-7】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux4 IS
PORT (i0, i1, i2, i3, a, b : IN STD_LOGIC;
      q : OUT STD_LOGIC);
END mux4;
ARCHITECTURE body_mux4 OF mux4 IS
BEGIN
process(i0,i1,i2,i3,a,b)
variable muxval : integer range 7 downto 0;
begin
                        muxval := 0;
if (a = '1') then   muxval := muxval + 1;  end if;
if (b = '1') then   muxval := muxval + 2;  end if;
case muxval is
     when 0 => q <= i0;
     when 1 => q <= i1;
     when 2 => q <= i2;
     when 3 => q <= i3;
     when others => null;
end case;
end process;
END body_mux4;
```

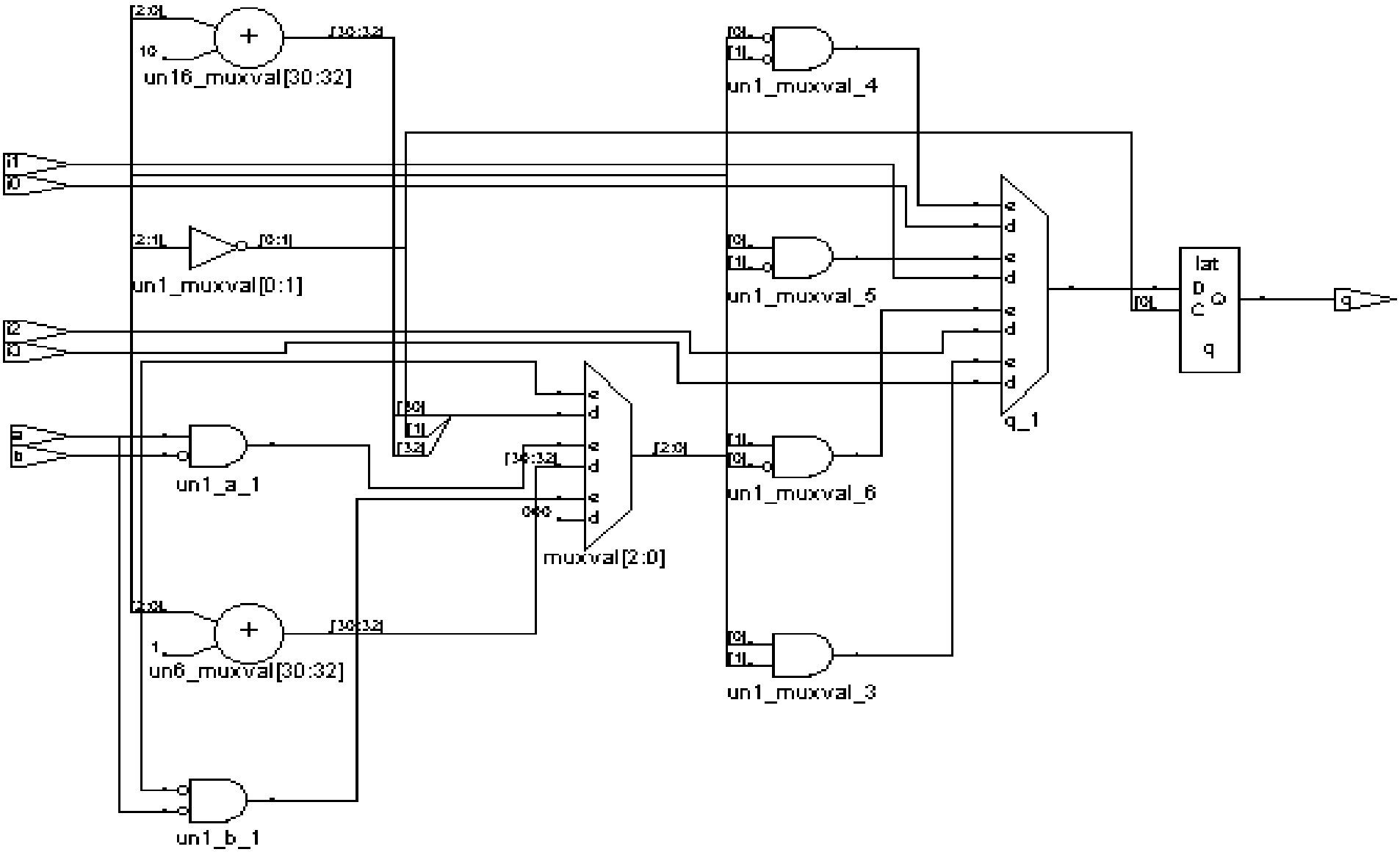


图3-3 例3-6的RTL电路

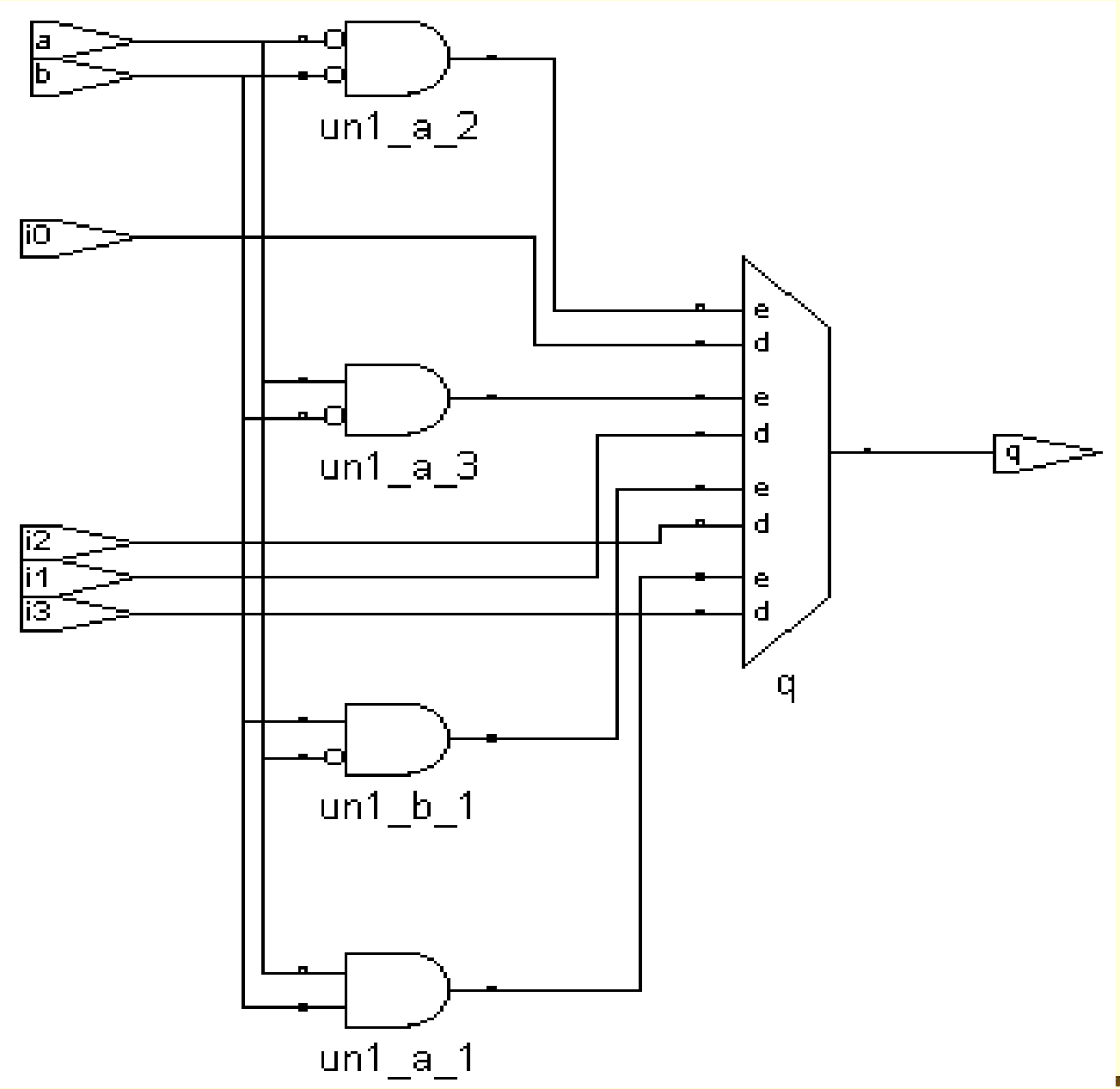


图3-4 例3-7的RTL电路

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

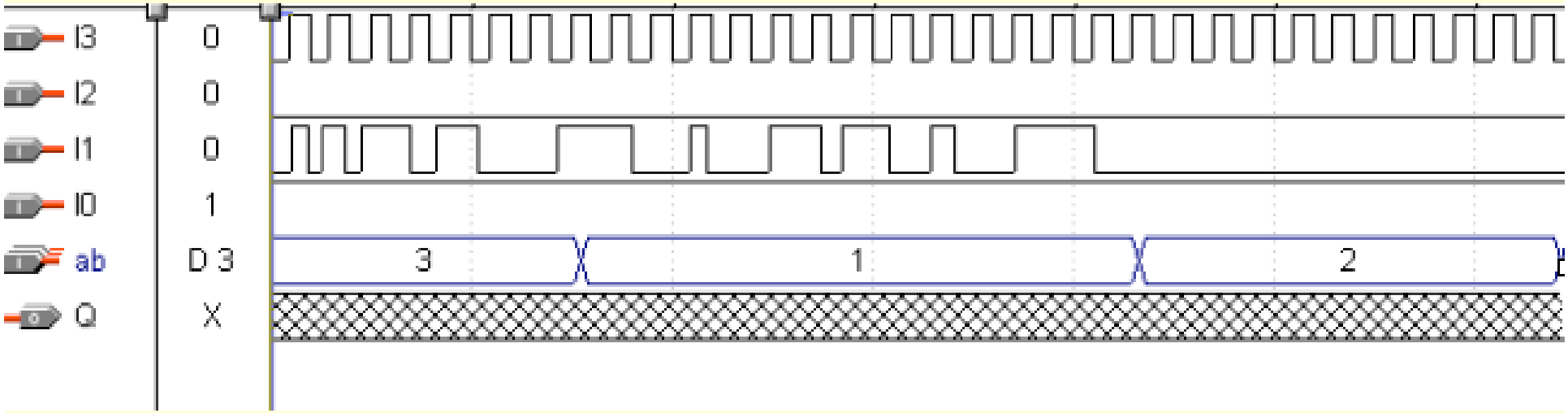


图3-5 例3-6中错误的工作时序

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

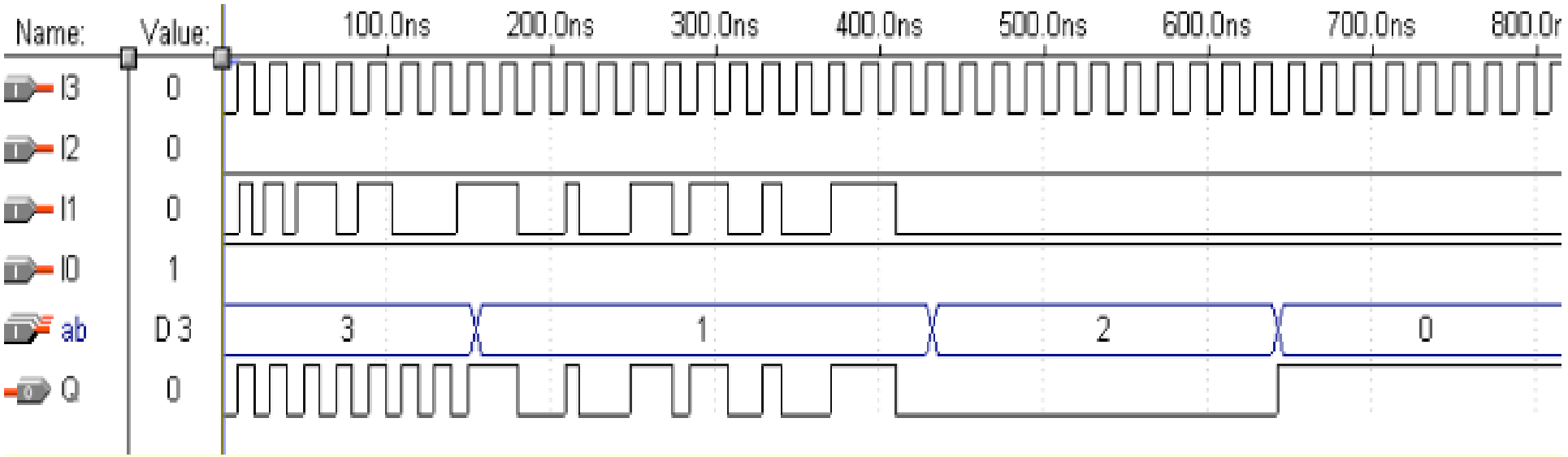


图3-6 例3-7中正确的工作时序

【例3-8】

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SHIFT IS
     PORT (CLK,C0 :  IN STD_LOGIC;  --时钟和进位输入
        MD    :  IN STD_LOGIC_VECTOR(2 DOWNTO 0); --移位模式控制字
        D     :  IN STD_LOGIC_VECTOR(7 DOWNTO 0); --待加载移位的数据
        QB    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); --移位数据输出
        CN          : OUT STD_LOGIC);                    --进位输出
END ENTITY;
ARCHITECTURE BEHAV OF SHIFT IS
          SIGNAL REG  : STD_LOGIC_VECTOR(7 DOWNTO 0);
      SIGNAL   CY : STD_LOGIC ;
 BEGIN
PROCESS (CLK,MD,C0)
   BEGIN
    IF CLK'EVENT AND CLK = '1' THEN
      CASE MD IS
       WHEN "001" =>    REG(0) <= C0 ;
REG(7 DOWNTO 1) <= REG(6 DOWNTO 0); CY <= REG(7);--带进位循环左移
```

（接下页）

（接上页）

```
            WHEN "010" =>           REG(0) <= REG(7);
   REG(7 DOWNTO 1) <= REG(6 DOWNTO 0);              --自循环左移
            WHEN "011" =>           REG(7) <= REG(0);
   REG(6 DOWNTO 0) <= REG(7 DOWNTO 1);              --自循环右移
            WHEN "100" =>           REG(7) <= C0 ;
   REG(6 DOWNTO 0) <= REG(7 DOWNTO 1); CY <= REG(0); --带进位循环右移
            WHEN "101" =>  REG(7 DOWNTO 0) <=   D(7 DOWNTO 0);    --加载待
移数
         WHEN OTHERS => REG <= REG ;  CY <= CY ;             --保持
       END CASE;
     END IF;
    END PROCESS;
          QB(7 DOWNTO 0) <= REG(7 DOWNTO 0); CN <= CY;        --移位后输出
   END BEHAV;
```

3.1 数据对象

3.1.4 进程中的信号赋值与变量赋值

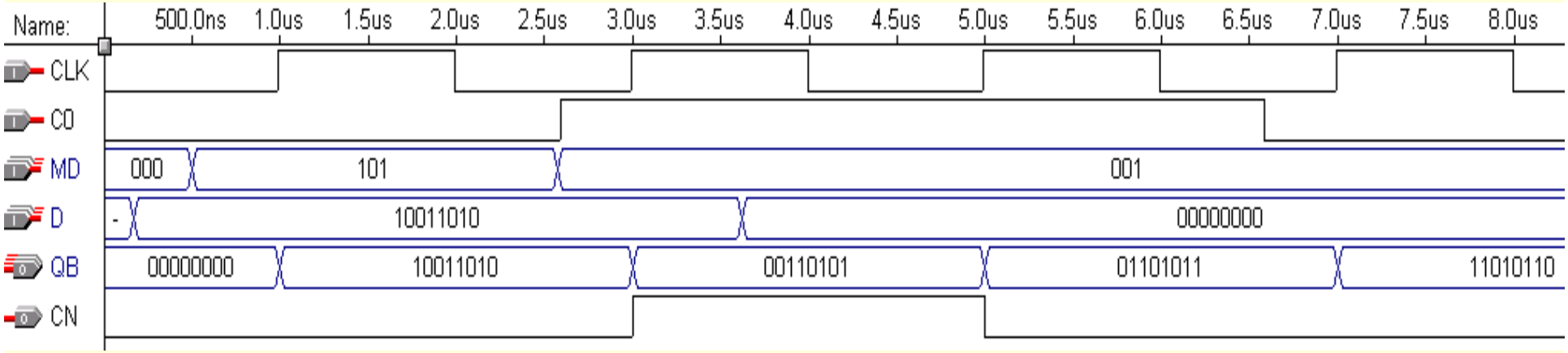


图3-7 例3-8中带进位循环左移仿真波形（MD="001"）

3.2 IF语句概述

(1) IF 条件句 Then
顺序语句
END IF ;

(2) IF 条件句 Then
顺序语句
ELSE
顺序语句
END IF ;

(3) IF 条件句 Then
IF 条件句 Then
...
END IF
END IF

(4) IF 条件句 Then
顺序语句
ELSIF 条件句 Then
顺序语句
...
ELSE
顺序语句
END IF

3.2 IF语句概述

【例3-9】

```
  LIBRARY IEEE;
  USE IEEE.STD_LOGIC_1164.ALL;
  ENTITY control_stmts IS
PORT (a, b, c: IN BOOLEAN;
        output: OUT BOOLEAN);
  END control_stmts;
  ARCHITECTURE example OF control_stmts IS
  BEGIN
    PROCESS (a, b, c)
      VARIABLE n: BOOLEAN;
    BEGIN
      IF  a  THEN   n := b;   ELSE   n := c;
      END IF;
      output <= n;
    END PROCESS;
  END example;
```

【例3-10】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY coder IS
  PORT (    din : IN STD_LOGIC_VECTOR(0 TO 7);
          output : OUT STD_LOGIC_VECTOR(0 TO 2)  );
END coder;
ARCHITECTURE behav OF coder IS
  SIGNAL SINT : STD_LOGIC_VECTOR(4 DOWNTO 0);
  BEGIN
    PROCESS (din)
  BEGIN
       IF (din(7)='0') THEN  output <= "000" ;
    ELSIF (din(6)='0') THEN  output <= "100" ;
    ELSIF (din(5)='0') THEN  output <= "010" ;
    ELSIF (din(4)='0') THEN  output <= "110" ;
    ELSIF (din(3)='0') THEN  output <= "001" ;
    ELSIF (din(2)='0') THEN  output <= "101" ;
    ELSIF (din(1)='0') THEN  output <= "011" ;
                          ELSE  output <= "111" ;
  END IF ;
    END PROCESS ;
   END behav;
```

3.2 IF语句概述

表3-2 8线-3线优先编码器真值表

输入								输出		
din0	din1	din2	din3	din4	din5	din6	din7	output0	output1	output2
x	x	x	x	x	x	x	0	0	0	0
x	x	x	x	x	x	0	1	1	0	0
x	x	x	x	x	0	1	1	0	1	0
x	x	x	x	0	1	1	1	1	1	0
x	x	x	0	1	1	1	1	0	0	1
x	x	0	1	1	1	1	1	1	0	1
x	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	1	1

注：表中的“x”为任意，类似VHDL中的“—”值。

3.3 进程语句归纳

3.3.1 进程语句格式

[进程标号:] PROCESS [(敏感信号参数表)] [IS]

[进程说明部分]

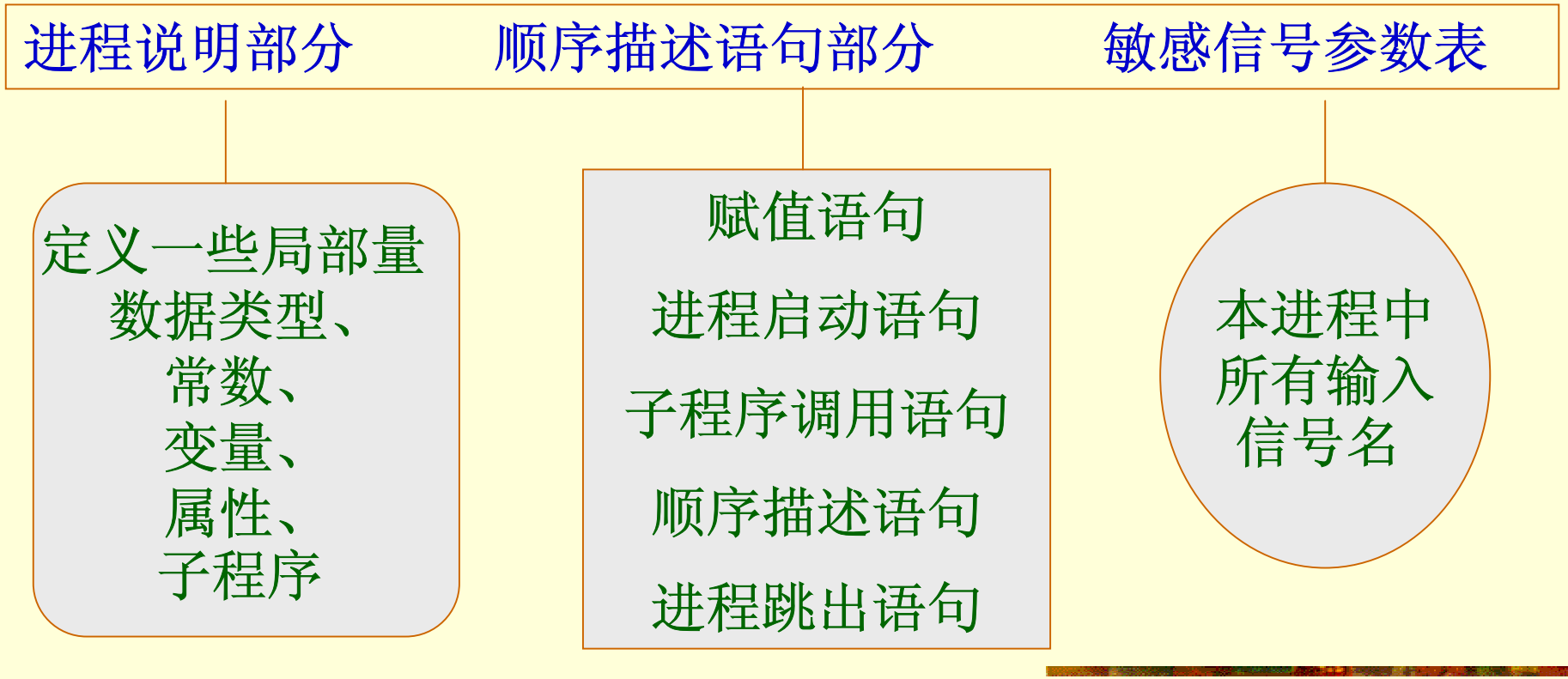
BEGIN

顺序描述语句

END PROCESS [进程标号];

3.3 进程语句归纳

3.3.2 进程结构组成



3.3 进程语句归纳

3.3.3 进程要点

设计者要从三个方面去判断它的功能和执行情况：

- 基于**CPU**的纯软件的**VHDL**行为仿真运行方式；
- 基于**VHDL**综合器的综合结果所可能实现的运行方式；
- 基于最终实现的硬件电路的运行方式。

3.3 进程语句归纳

3.3.3 进程要点

注意以下几方面的问题：

1. **PROCESS为一无限循环语句**
2. **PROCESS中的顺序语句具有明显的顺序/并行运行双重性**

```
PROCESS(abc)
  BEGIN
   CASE abc IS
    WHEN "0000" => so<="010" ;
    WHEN "0001" => so<="111" ;
    WHEN "0010" => so<="101" ;
    . . .
    WHEN "1110" => so<="100" ;
    WHEN "1111" => so<="000" ;
    WHEN OTHERS => NULL ;
   END CASE;
END PROCESS;
```

3.3 进程语句归纳

3.3.3 进程要点

注意以下几方面的问题：

1. **PROCESS**为一无限循环语句

2. **PROCESS**中的顺序语句具有明显的顺序/并行运行双重性

3. 进程必须由敏感信号的变化来启动

4. 进程语句本身是并行语句

【例3-11】

```
ENTITY mul IS
PORT (a, b, c, selx, sely  : IN BIT;
                  data_out  : OUT BIT  );
END mul;
ARCHITECTURE ex OF mul IS
  SIGNAL  temp : BIT;
BEGIN
p_a : PROCESS (a, b, selx)
      BEGIN
        IF (selx = '0') THEN  temp <= a;  ELSE    temp <= b;
      END IF;
  END PROCESS p_a;
   p_b:  PROCESS(temp, c, sely)
      BEGIN
    IF (sely = '0')  THEN  data_out <= temp; ELSE   data_out <= c;
    END IF;
  END PROCESS p_b;
END ex;
```

3.4 并行语句例解

```
data1 <=  a AND b ;
data2 <=  c ;
```

【例3-12】

```
ARCHITECTURE dataflow OF mux IS
SIGNAL seiect :  INTEGER RANGE 15 DOWNTO 0;
BEGIN
Select <= 0 WHEN s0='0' AND s1='0' ELSE
1 WHEN s0='1' AND s1='0' ELSE
2 WHEN s0='0' AND s1='1' ELSE
3 ;
x <= a WHEN select=0  ELSE
b WHEN select=1  ELSE
c WHEN select=2  ELSE
d ;
. . .
```

3.5 仿真延时

3.5.1 固有延时

固有延时（**Inertial Delay**）也称为惯性延时，是任何电子器件都存在的一种延时特性。

```
z <= x XOR y AFTER 5ns ;

z <= x XOR y ;

B <=  A AFTER 20ns ; --固有延时模型
```

3.5 仿真延时

3.5.2 传输延时

另一种延时模型是传输模型
（**Transport Delay**）

```
B <= TRANSPORT A AFTER 20 ns;--
```
传输延时模型

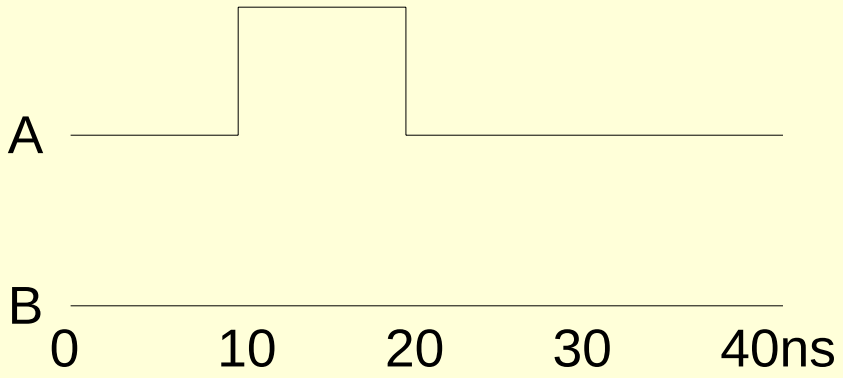


图3-8 固有延时输入输出波形

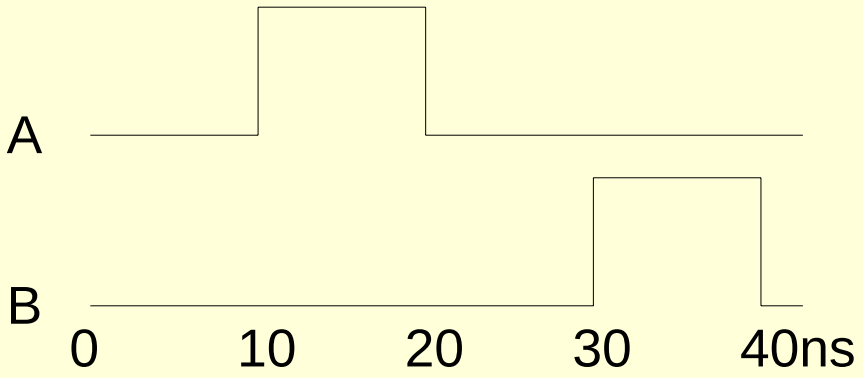


图3-9传输延时输入输出波形

3.5 仿真延时

3.5.3 仿真 δ

VHDL仿真器和综合器将自动为系统中的信号赋值配置一足够小而又能满足逻辑排序的延时量，即仿真软件的最小分辨时间，这个延时量就称为仿真δ（Simulation Delta），或称 δ 延时，从而使并行语句和顺序语句中的并列赋值逻辑得以正确执行。

3.6 有限状态机

3.6.1 数据类型定义语句

TYPE 数据类型名 IS 数据类型定义 OF 基本数据类型；

或 TYPE 数据类型名 IS 数据类型定义；

```
TYPE st1 IS ARRAY ( 0 TO 15 ) OF STD_LOGIC ;
TYPE week IS (sun, mon, tue, wed, thu, fri, sat) ;
```

```
TYPE m_state IS  ( st0, st1, st2, st3, st4, st5 ) ;
SIGNAL present_state, next_state  :  m_state  ;
```

```
TYPE my_logic IS ( '1' , 'Z' , 'U' , '0' ) ;
SIGNAL s1 : my_logic ;
s1 <= 'Z' ;
```

SUBTYPE 子类型名 IS 基本数据类型 RANGE 约束范围;

```
SUBTYPE digits IS INTEGER RANGE 0 to 9 ;
```

3.6 有限状态机

3.6.2 一般有限状态机的设计

1. 说明部分

```
ARCHITECTURE ...IS
  TYPE FSM_ST IS (s0, s1, s2, s3);
  SIGNAL current_state, next_state: FSM_ST;
```

3.6 有限状态机

3.6.2 一般有限状态机的设计

2. 主控时序进程

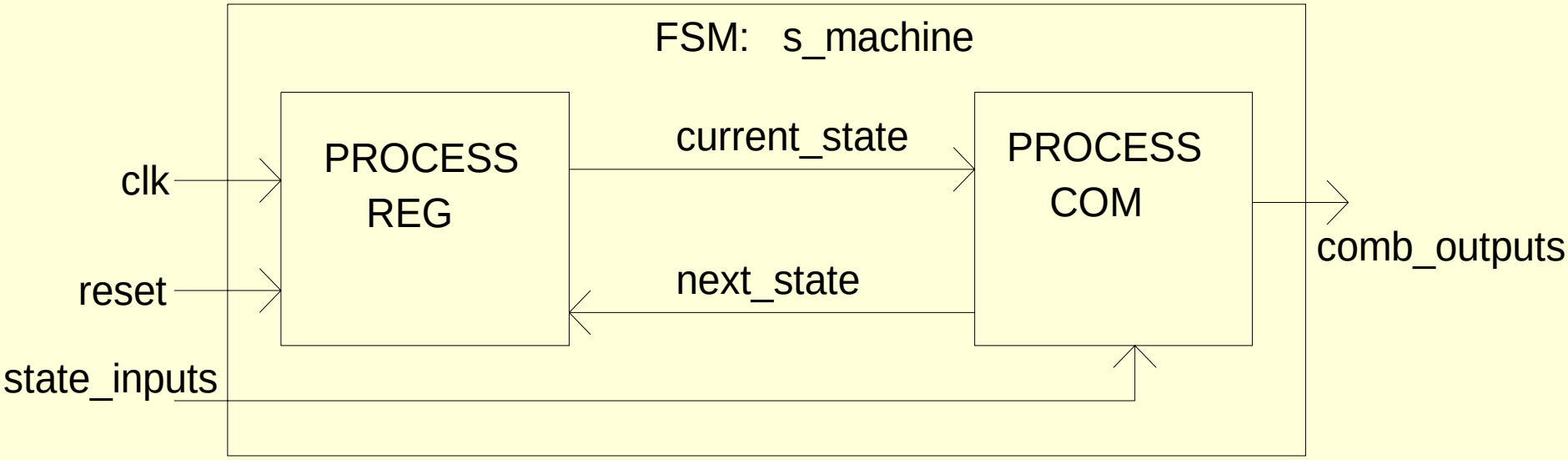


图3-10 一般状态机结构框图

3.6 有限状态机

3.6.2 一般有限状态机的设计

3. 主控组合进程

4. 辅助进程

【例3-13】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY s_machine IS
  PORT ( clk,reset    : IN STD_LOGIC;
         state_inputs : IN STD_LOGIC_VECTOR (0 TO 1);
         comb_outputs : OUT INTEGER RANGE 0 TO 15 );
END s_machine;
```

接下页

接上页

```
ARCHITECTURE behv OF s_machine IS
  TYPE FSM_ST IS (s0, s1, s2, s3);  --数据类型定义，状态符号化
  SIGNAL current_state, next_state: FSM_ST;--将现态和次态定义为新的数据类型
BEGIN
 REG: PROCESS (reset,clk)                --主控时序进程
  BEGIN
    IF reset = '1' THEN   current_state <= s0;--检测异步复位信号
    ELSIF clk='1' AND clk'EVENT THEN
      current_state <= next_state;
    END IF;
  END PROCESS;
COM:PROCESS(current_state, state_Inputs)     --主控组合进程
  BEGIN
    CASE current_state IS
      WHEN s0 => comb_outputs<= 5;
        IF state_inputs = "00" THEN  next_state<=s0;
          ELSE  next_state<=s1;
        END IF;
```

接下页

接上页

```
WHEN s1 =>  comb_outputs<= 8;
      IF state_inputs = "00" THEN  next_state<=s1;
      ELSE  next_state<=s2;
      END IF;
    WHEN s2 =>   comb_outputs<= 12;
      IF state_inputs = "11" THEN  next_state <= s0;
      ELSE  next_state <= s3;
      END IF;
    WHEN s3 =>  comb_outputs <= 14;
      IF state_inputs = "11" THEN  next_state <= s3;
      ELSE  next_state <= s0;
      END IF;
  END case;
 END PROCESS;
END behv;
```

3.6 有限状态机

3.6.2 一般有限状态机的设计

4. 辅助进程

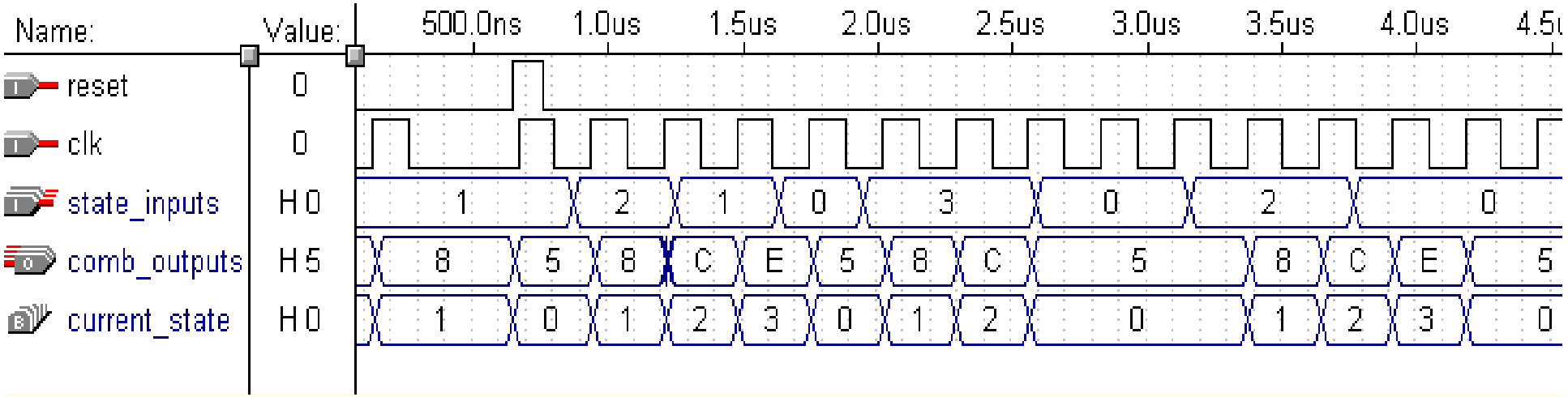


图3-11 例3-13状态机的工作时序

3.6 有限状态机

3.6.2 一般有限状态机的设计

4. 辅助进程

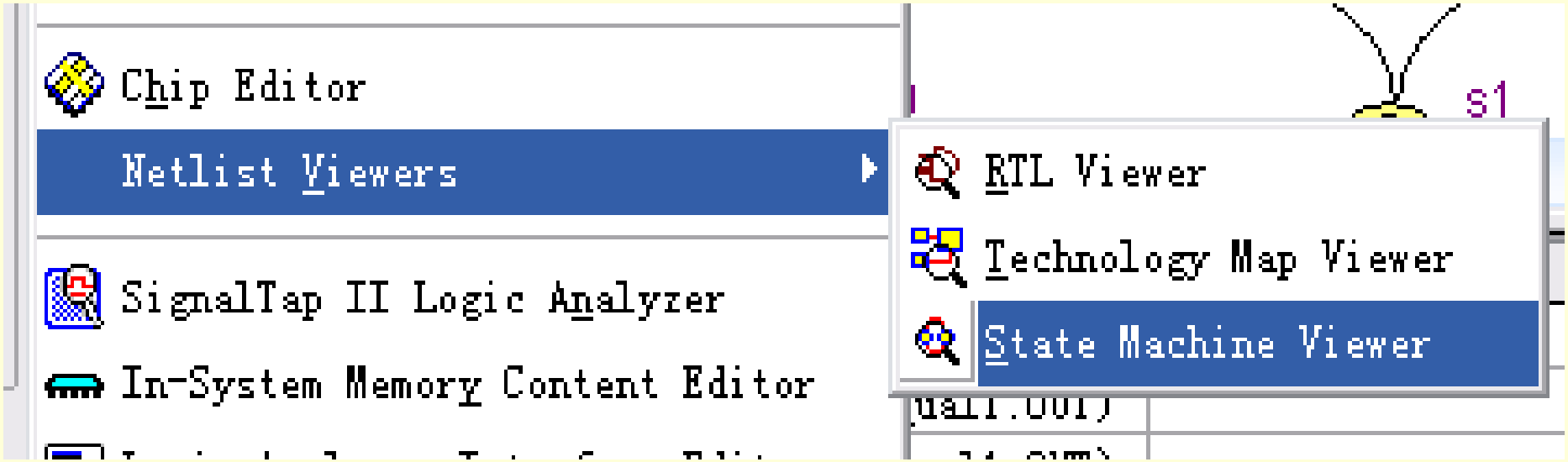


图3-12 打开QuartusII状态图观察器

3.6 有限状态机

3.6.2 一般有限状态机的设计

4. 辅助进程

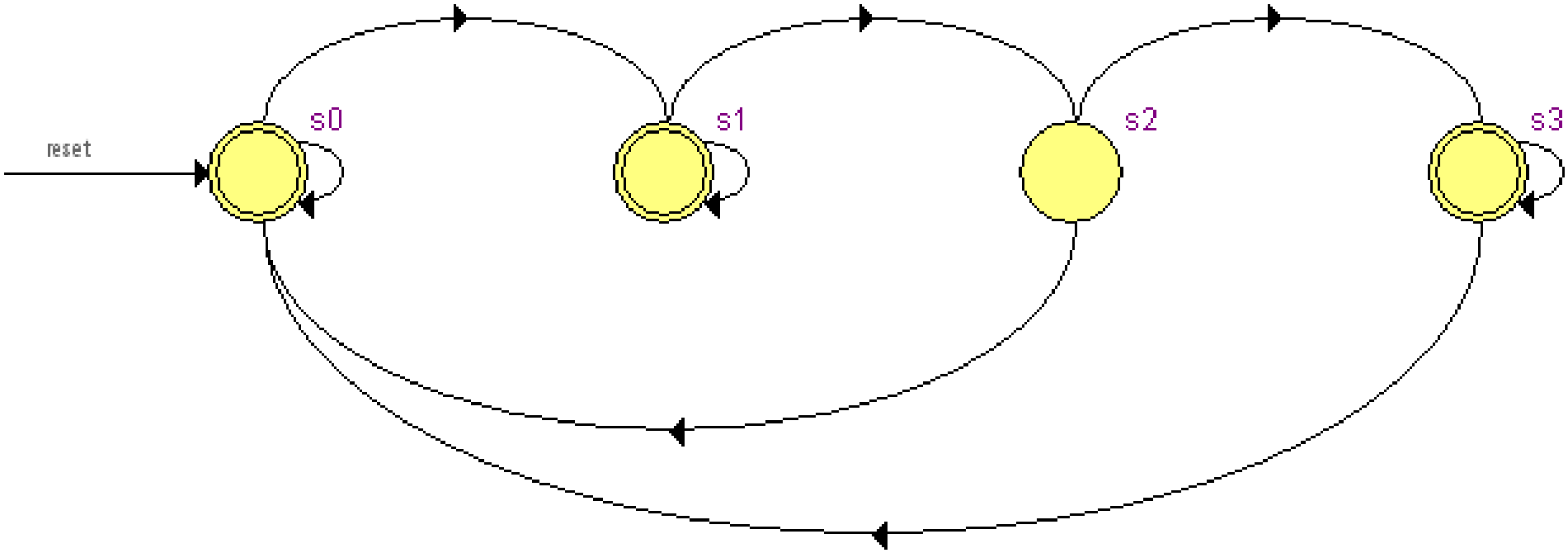


图3-13 例3-13的状态图

3.6 有限状态机

3.6.3 Moore型状态机

1. 多进程Moore型有限状态机

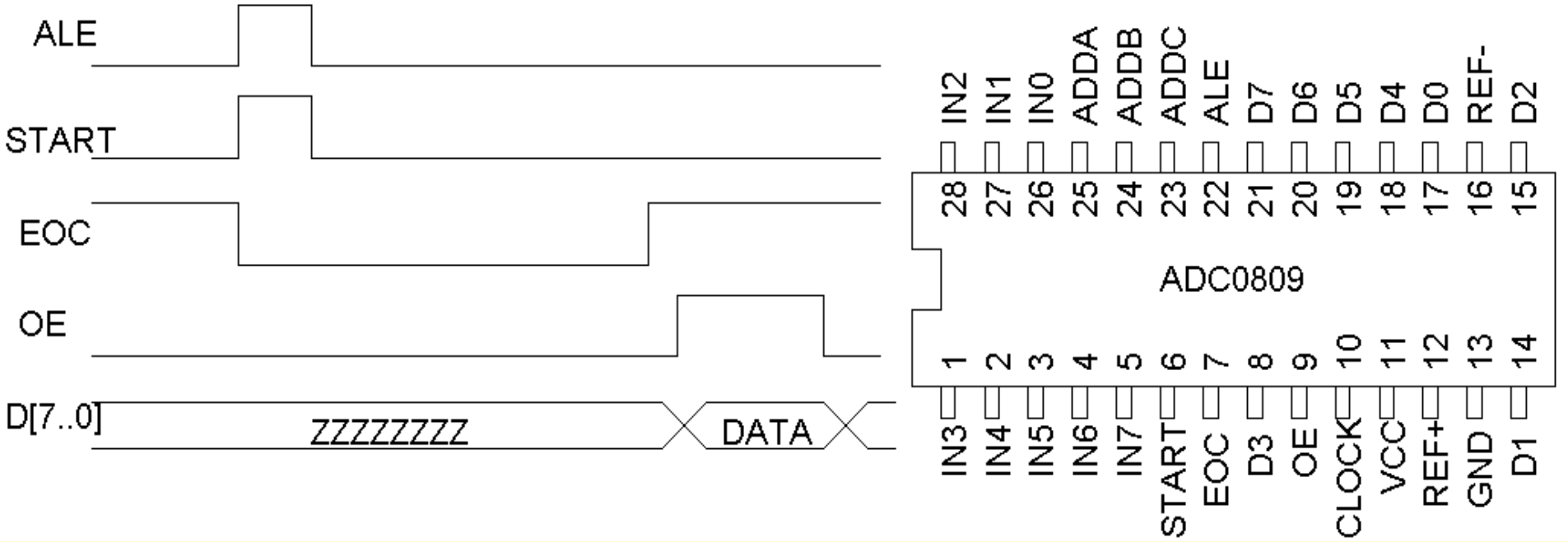


图3-14 ADC0809工作时序

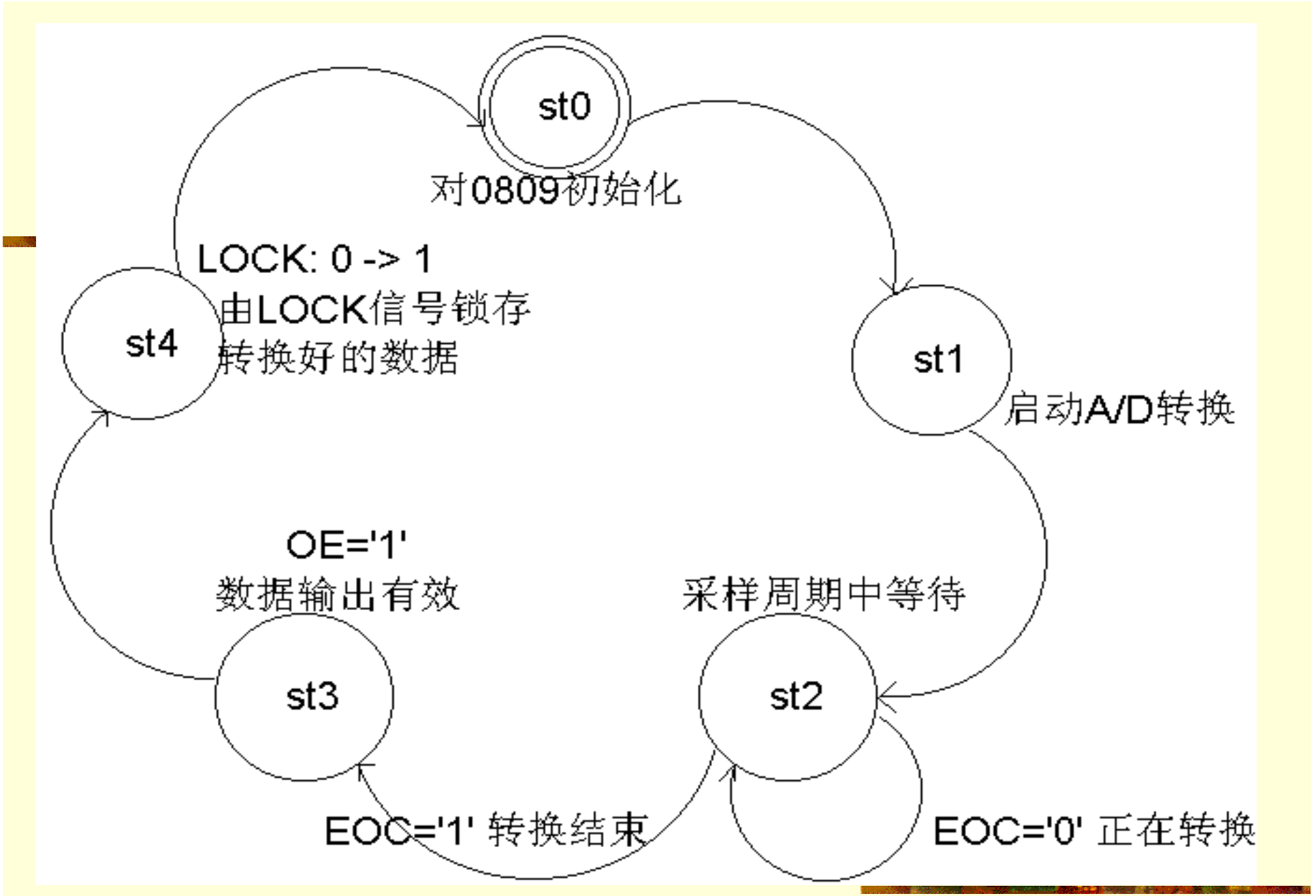


图3-15 控制ADC0809采样状态图

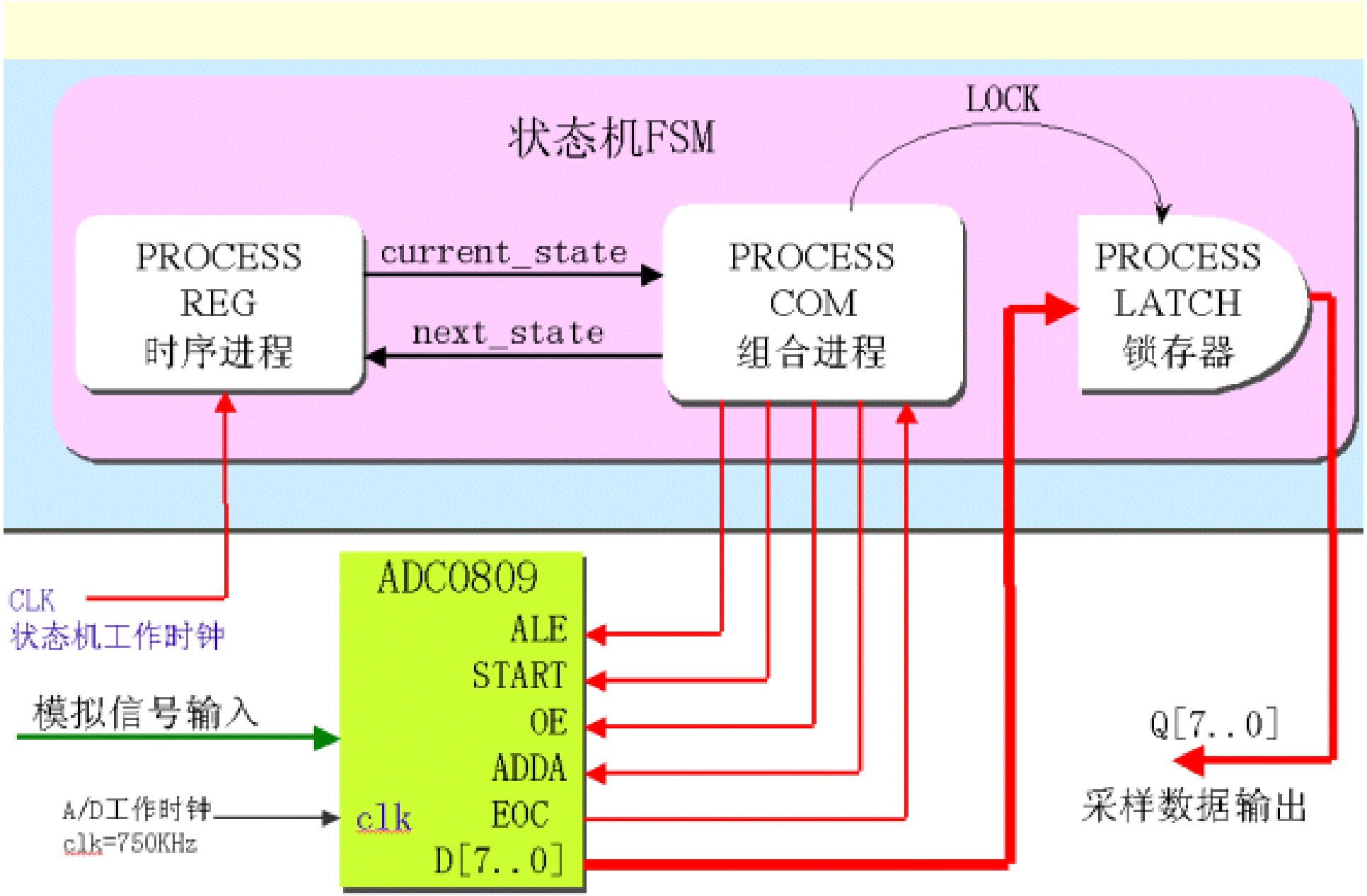


图3-16 采样状态机结构框图

【例3-14】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ADCINT IS
   PORT(D  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);  --来自0809转换好的8位数据
CLK  : IN STD_LOGIC;                    --状态机工作时钟
EOC  : IN STD_LOGIC;                 --转换状态指示，低电平表示正在转换
ALE  : OUT STD_LOGIC;                 --8个模拟信号通道地址锁存信号
START  : OUT STD_LOGIC;                 --转换开始信号
OE  : OUT STD_LOGIC;                 --数据输出3态控制信号
ADDA  : OUT STD_LOGIC;                 --信号通道最低位控制信号
LOCK0  : OUT STD_LOGIC;                 --观察数据锁存时钟
Q  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --8位数据输出
END ADCINT;
ARCHITECTURE behav OF ADCINT IS
TYPE states IS (st0, st1, st2, st3,st4) ; --定义各状态子类型
  SIGNAL current_state, next_state: states :=st0 ;
  SIGNAL REGL              : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL LOCK              : STD_LOGIC; -- 转换后数据输出锁存时钟信号
 BEGIN
ADDA <= '1';--当ADDA<='0'，模拟信号进入通道IN0；当ADDA<='1'，则进入通道
IN1
Q <= REGL; LOCK0 <= LOCK ;
```

接下页

```
COM: PROCESS(current_state,EOC)  BEGIN  --规定各状态转换方式
   CASE current_state IS
   WHEN st0=>ALE<='0';START<='0';LOCK<='0';OE<='0';
       next_state <= st1; --0809初始化
   WHEN st1=>ALE<='1';START<='1';LOCK<='0';OE<='0';
next_state <= st2; --启动采样
   WHEN st2=> ALE<='0';START<='0';LOCK<='0';OE<='0';
    IF (EOC='1') THEN next_state <= st3; --EOC=1表明转换结束
          ELSE next_state <= st2;  END IF ;   --转换未结束，继续等待
   WHEN st3=> ALE<='0';START<='0';LOCK<='0';OE<='1';
next_state <= st4;--开启OE,输出转换好的数据
   WHEN st4=> ALE<='0';START<='0';LOCK<='1';OE<='1'; next_state <= st0;
   WHEN OTHERS => next_state <= st0;
   END CASE ;
 END PROCESS COM ;
  REG: PROCESS (CLK)
    BEGIN
    IF (CLK'EVENT AND CLK='1') THEN current_state<=next_state; END IF;
   END PROCESS REG ;      -- 由信号current_state将当前状态值带出此进程:REG
  LATCH1: PROCESS (LOCK) -- 此进程中，在LOCK的上升沿，将转换好的数据锁入
       BEGIN
       IF LOCK='1' AND LOCK'EVENT THEN   REGL <= D ; END IF;
      END PROCESS LATCH1 ;
  END behav;
```

3.6 有限状态机

3.6.3 Moore型状态机

1. 多进程Moore型有限状态机

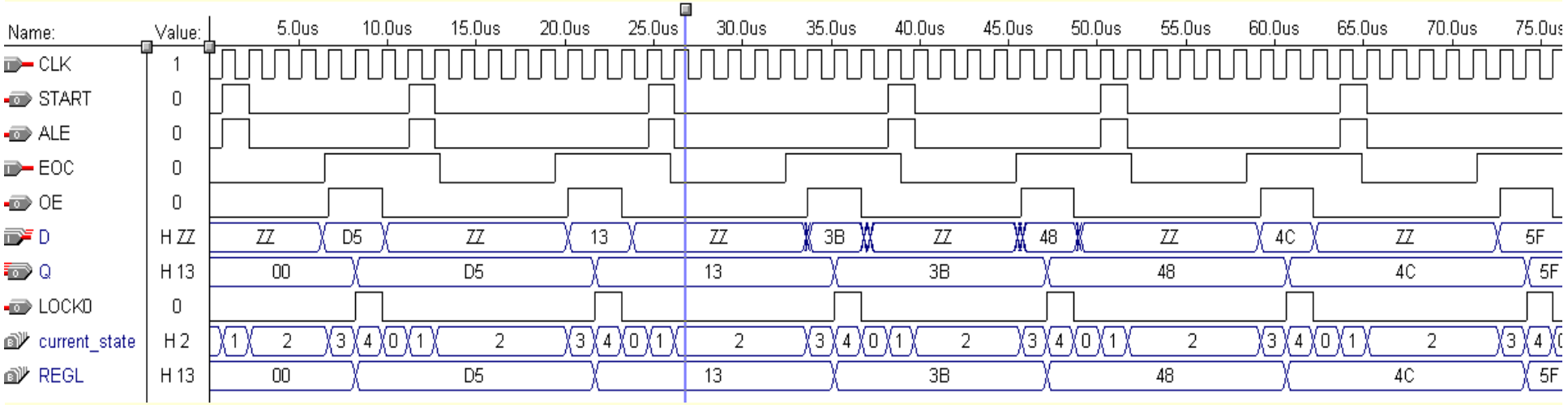


图3-17 ADC0809采样状态机工作时序

3.6 有限状态机

3.6.3 Moore型状态机

2. 单进程Moore型有限状态机

【例3-15】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MOORE1 IS
  PORT (DATAIN  :IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        CLK,RST : IN STD_LOGIC;
              Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END MOORE1;
ARCHITECTURE behav OF MOORE1 IS
  TYPE ST_TYPE IS (ST0, ST1, ST2, ST3,ST4);
   SIGNAL C_ST : ST_TYPE ;
    BEGIN
    PROCESS(CLK,RST)
```

（接下页）

```
BEGIN
    IF RST ='1' THEN  C_ST <= ST0 ; Q<= "0000" ;
      ELSIF CLK'EVENT AND CLK='1' THEN
        CASE C_ST IS
         WHEN ST0 => IF DATAIN ="10" THEN C_ST <= ST1 ;
                ELSE C_ST <= ST0 ; END IF;
                Q <= "1001" ;
         WHEN ST1 => IF DATAIN ="11" THEN C_ST <= ST2 ;
                ELSE C_ST <= ST1 ;END IF;
                Q <= "0101" ;
         WHEN ST2 => IF DATAIN ="01" THEN C_ST <= ST3 ;
                ELSE C_ST <= ST0 ;END IF;
                Q <= "1100" ;
         WHEN ST3 => IF DATAIN ="00" THEN C_ST <= ST4 ;
                ELSE C_ST <= ST2 ;END IF;
                Q <= "0010" ;
         WHEN ST4 => IF DATAIN ="11" THEN C_ST <= ST0 ;
                ELSE C_ST <= ST3 ;END IF;
                  Q <= "1001" ;
         WHEN OTHERS => C_ST <= ST0;
       END CASE;
    END IF;
  END PROCESS;
END behav;
```

3.6 有限状态机

3.6.3 Moore型状态机

2. 单进程Moore型有限状态机

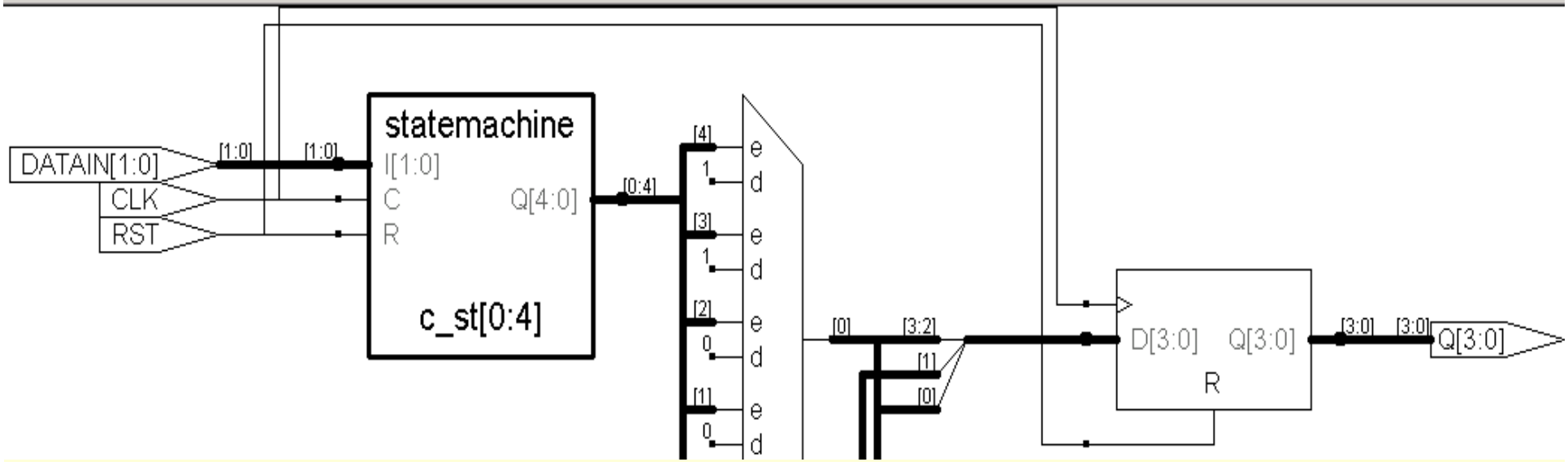


图3-18 例3-15状态机综合后的部分主要RTL电路模块（Synplify综合）

3.6 有限状态机

3.6.3 Moore型状态机

2. 单进程Moore型有限状态机

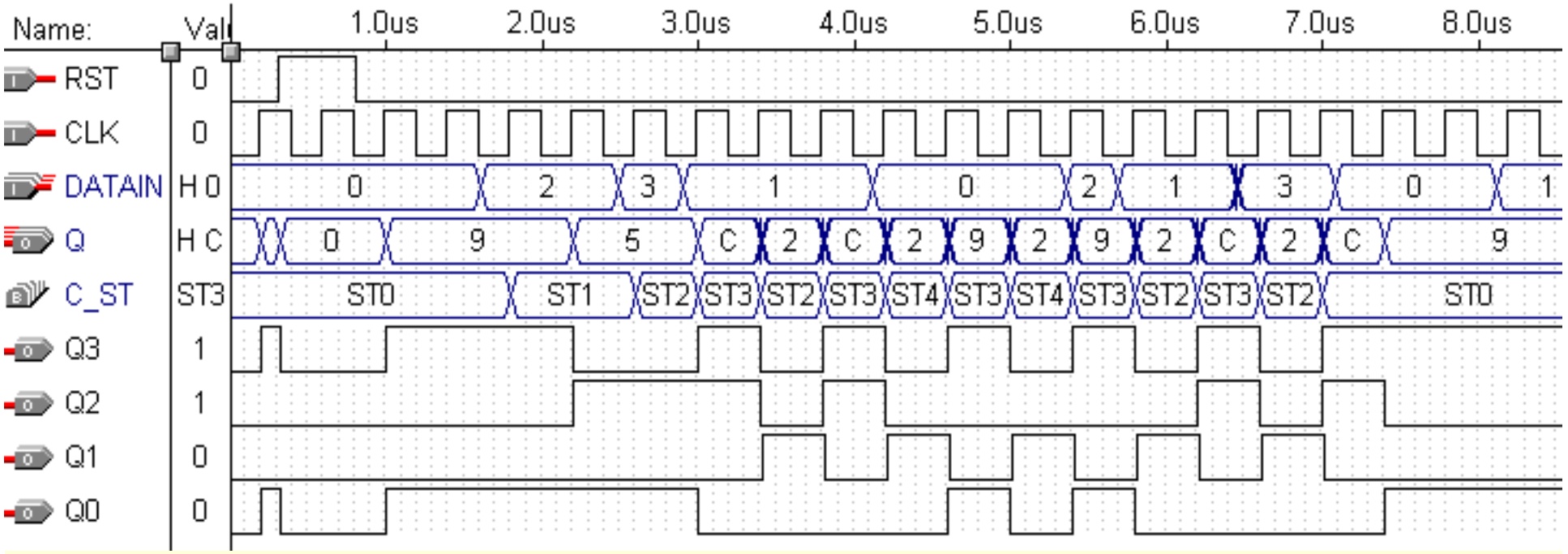


图3-19 例3-15单进程状态机工作时序

3.6 有限状态机

3.6.3 Moore型状态机

2. 单进程Moore型有限状态机

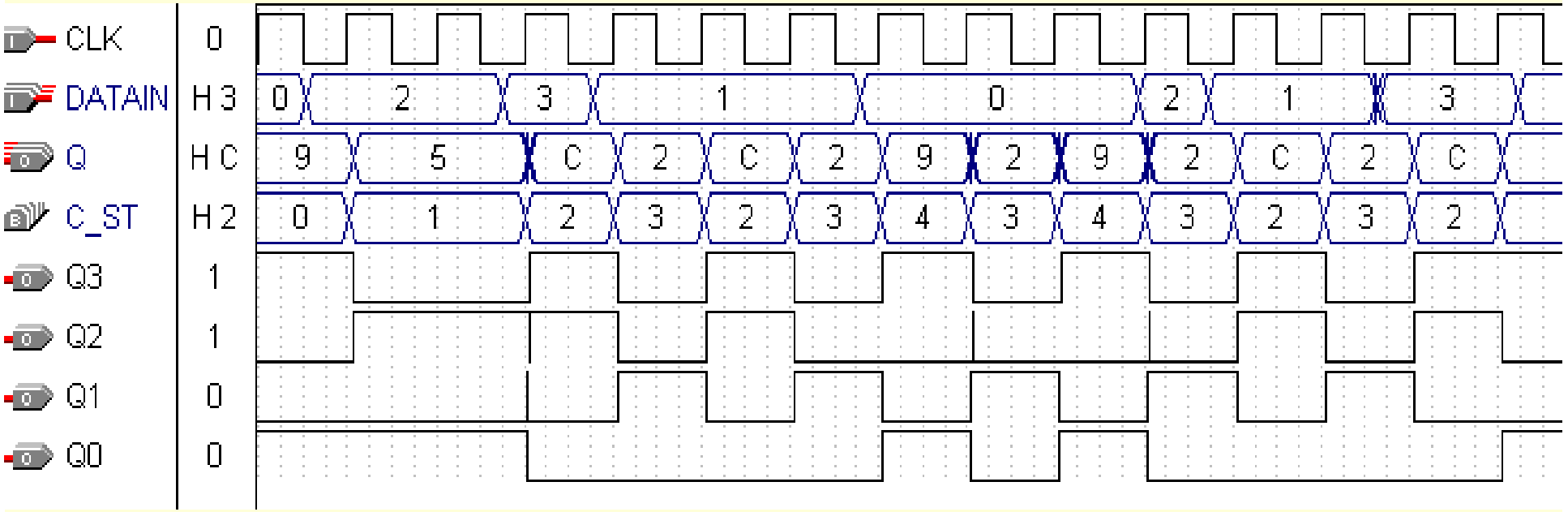


图3-20 对应于例3-15的二进程状态机工作时序图

3.6 有限状态机

3.6.4 Mealy型状态机

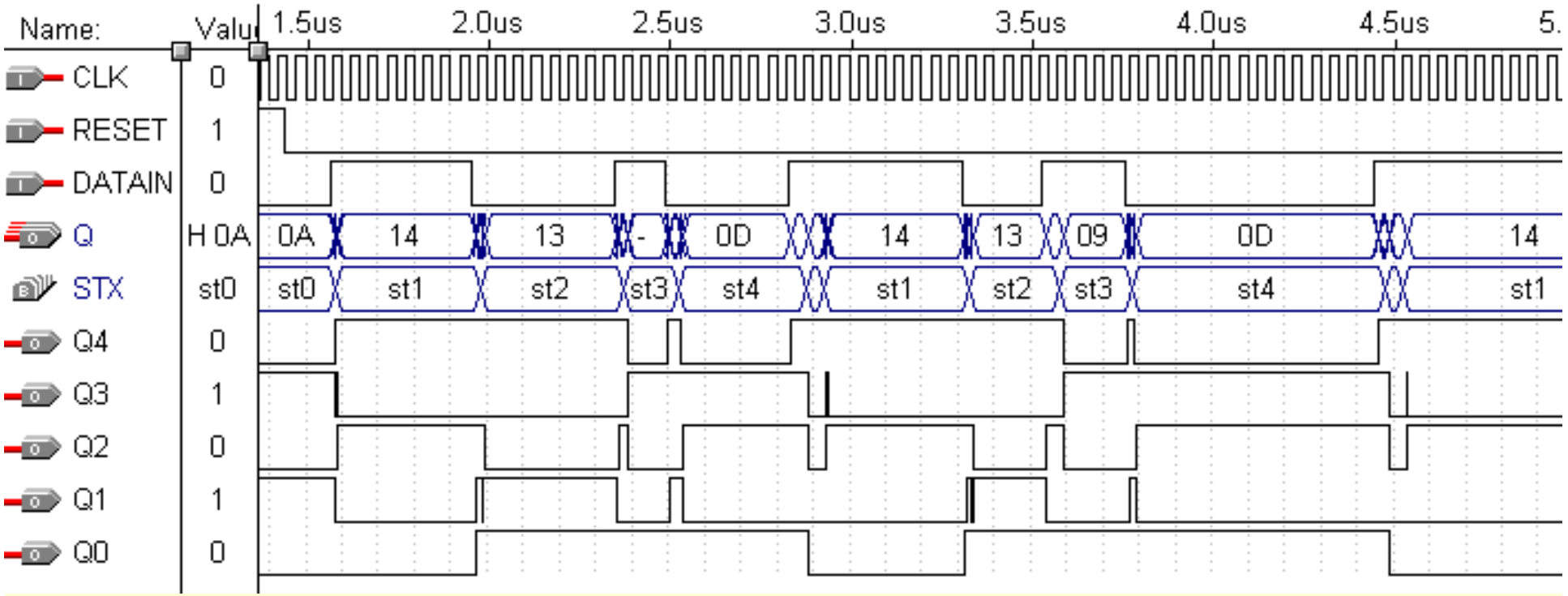


图3-21 例3-16状态机工作时序图

【例3-16】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MEALY1 IS
PORT ( CLK ,DATAIN,RESET  : IN STD_LOGIC;
       Q : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END MEALY1;
ARCHITECTURE behav OF MEALY1 IS
 TYPE states IS (st0, st1, st2, st3,st4);
 SIGNAL STX : states  ;
BEGIN
 COMREG : PROCESS(CLK,RESET)  BEGIN --决定转换状态的进程
   IF RESET ='1' THEN    STX <= ST0;
   ELSIF CLK'EVENT AND CLK = '1' THEN    CASE STX IS
      WHEN st0 => IF DATAIN = '1' THEN  STX <= st1; END IF;
      WHEN st1 => IF DATAIN = '0' THEN  STX <= st2; END IF;
      WHEN st2 => IF DATAIN = '1' THEN  STX <= st3; END IF;
      WHEN st3=>  IF DATAIN = '0' THEN  STX <= st4; END IF;
      WHEN st4=>  IF DATAIN = '1' THEN  STX <= st0; END IF;
      WHEN OTHERS => STX <= st0;
```

（接下页）

```
END CASE ;
  END IF;
 END PROCESS COMREG ;
COM1: PROCESS(STX,DATAIN) BEGIN --输出控制信号的进程
  CASE STX IS
    WHEN st0 => IF DATAIN = '1' THEN Q <= "10000" ;
                ELSE Q<="01010" ; END IF ;
   WHEN st1 => IF DATAIN = '0' THEN Q <= "10111" ;
                ELSE Q<="10100" ; END IF ;
   WHEN st2 => IF DATAIN = '1' THEN Q <= "10101" ;
                ELSE Q<="10011" ; END IF ;
   WHEN st3=>  IF DATAIN = '0' THEN Q <= "11011" ;
                ELSE Q<="01001" ; END IF ;
   WHEN st4=>  IF DATAIN = '1' THEN Q <= "11101" ;
                ELSE Q<="01101" ; END IF ;
   WHEN OTHERS =>  Q<="00000" ;
  END CASE ;
  END PROCESS COM1 ;
END behav;
```

3.7 双向和三态电路信号赋值

3.7.1 三态门设计

【例3-17】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY tri_s IS
    port (     enable : IN STD_LOGIC;
             datain : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            dataout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END tri_s ;
ARCHITECTURE bhv OF tri_s IS
BEGIN
PROCESS(enable,datain)
   BEGIN
   IF enable = '1' THEN dataout <= datain ;
     ELSE dataout <="ZZZZZZZZ" ;  END IF ;
END PROCESS;
END bhv;
```

3.7 双向和三态电路信号赋值

3.7.1 三态门设计

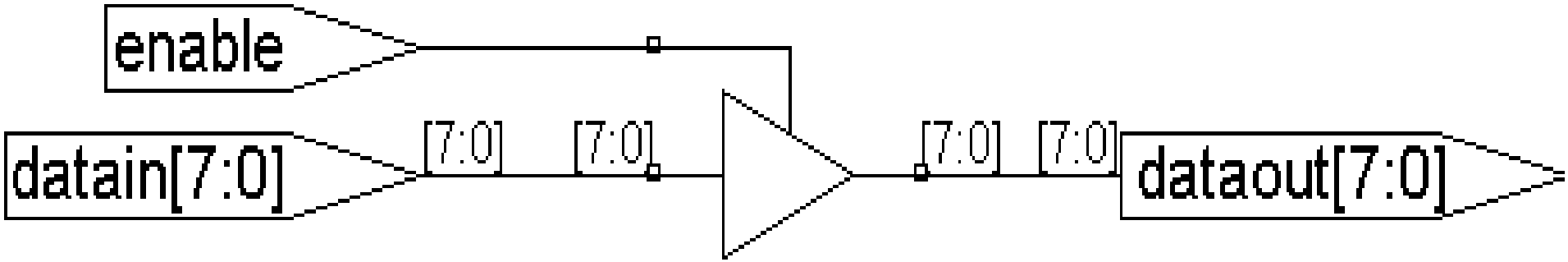


图3-22 8位3态控制门电路（Synplify综合）

3.7 双向和三态电路信号赋值

3.7.2 双向端口设计

【例3-18】

```
library ieee;
use ieee.std_logic_1164.all;
entity tri_state is
port (control : in std_logic;
      in1: in std_logic_vector(7 downto 0);
       q : inout std_logic_vector(7 downto 0);
       x : out std_logic_vector(7 downto 0));
end tri_state;
architecture body_tri of tri_state is
begin
process(control,q,in1)
begin
if (control = '0') then   x <= q  ;
else      q <= in1; x<="ZZZZZZZZ" ;  end if;
end process;
end body_tri;
```

3.7 双向和三态电路信号赋值

3.7.2 双向端口设计

【例3-19】

(以上部分同上例)

```
process(control,q,in1)
begin
if (control='0') then  x <= q ; q <= "ZZZZZZZZ";
                  else  q <= in1; x <="ZZZZZZZZ";   end if;
end process;
end body_tri;
```

3.7 双向和三态电路信号赋值

3.7.2 双向端口设计

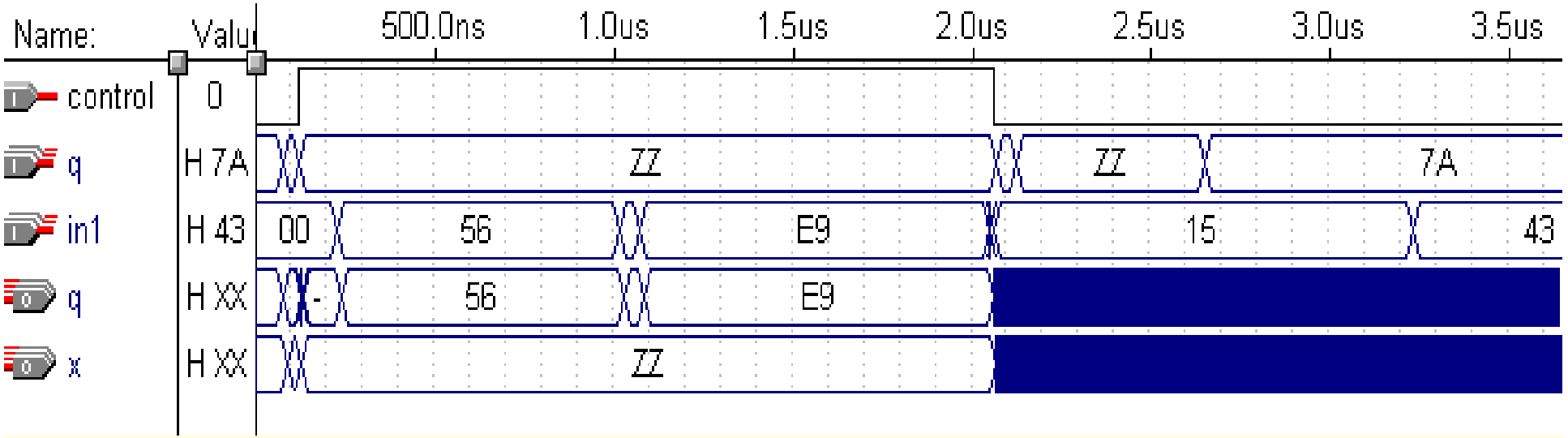


图3-23 例3-18的仿真波形图

3.7 双向和三态电路信号赋值

3.7.2 双向端口设计

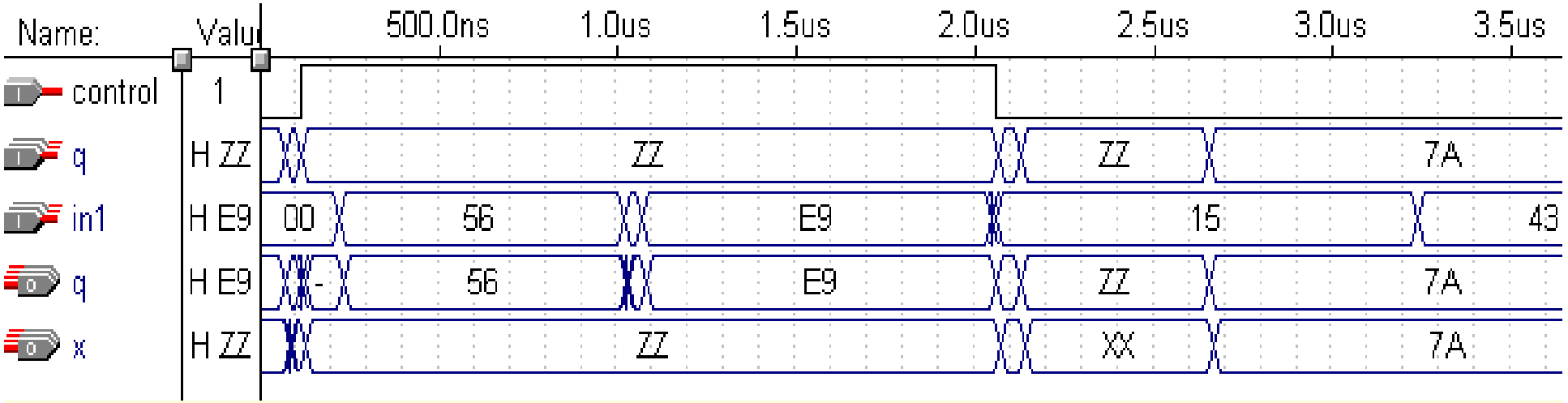


图3-24 例3-19的仿真波形图

3.7 双向和三态电路信号赋值

3.7.2 双向端口设计

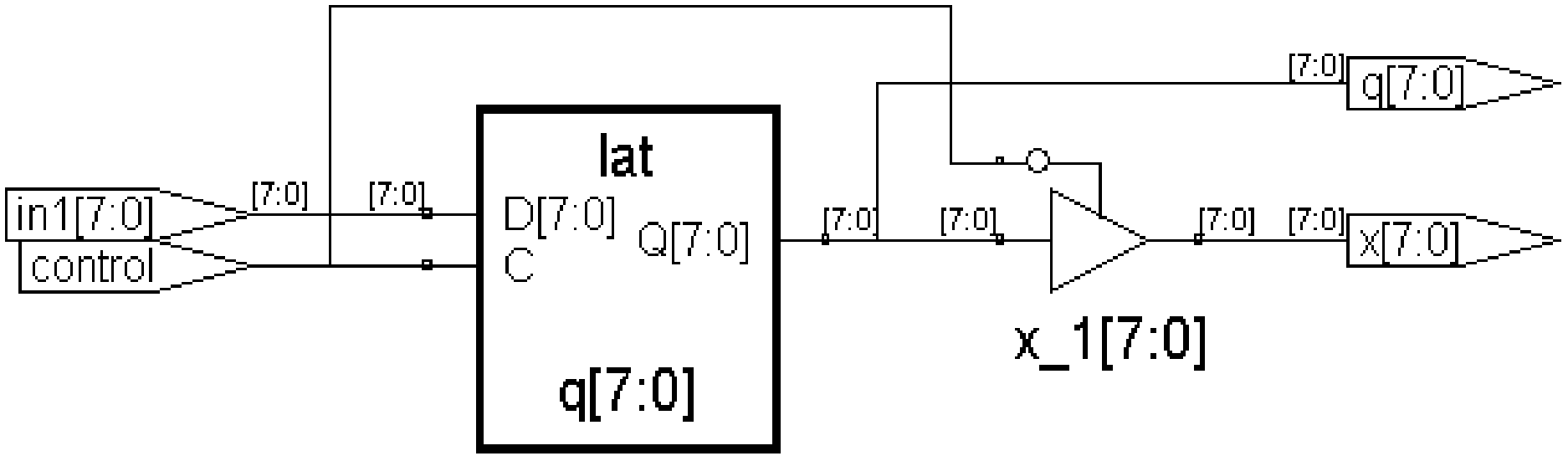


图3-25 例3-18的综合结果

3.7 双向和三态电路信号赋值

3.7.2 双向端口设计

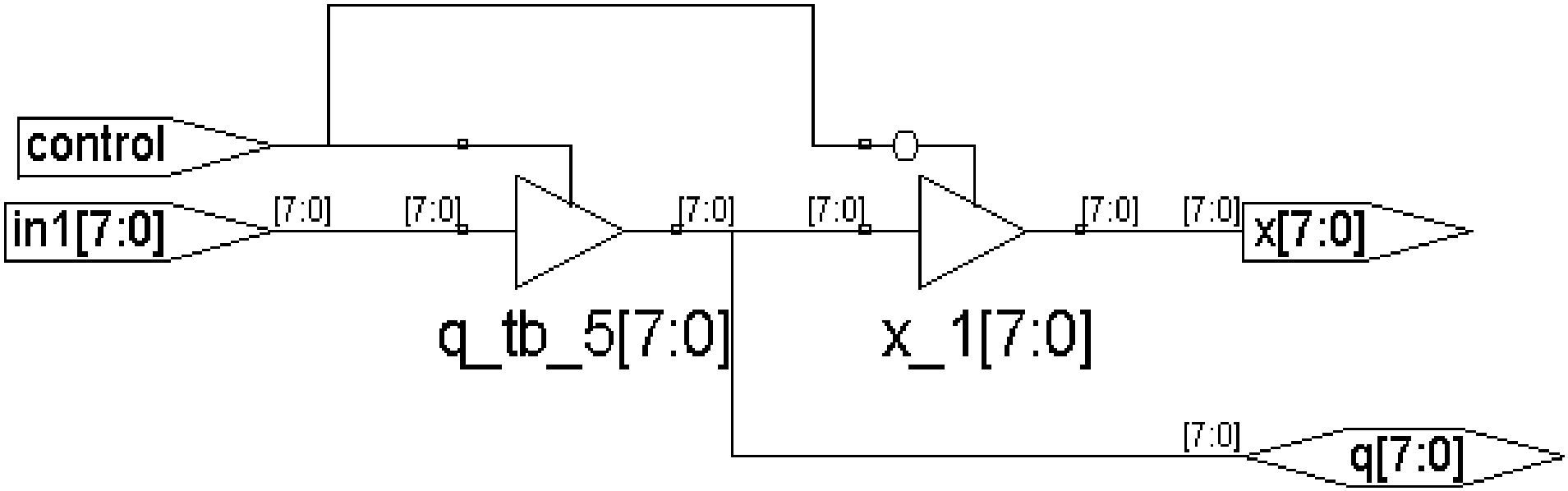


图3-26 例3-19的综合结果

3.7 双向和三态电路信号赋值

3.7.3 三态总线电路设计

【例3-20】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY tristate2 IS
    port ( input3, input2, input1, input0 :
                IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    enable : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    output : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END tristate2 ;
ARCHITECTURE multiple_drivers OF tristate2 IS
BEGIN
PROCESS(enable,input3, input2, input1, input0 )
   BEGIN
```

（接下页）

3.7 双向和三态电路信号赋值

3.7.3 三态总线电路设计

```
IF enable = "00" THEN output <= input3 ;
        ELSE output <=(OTHERS => 'Z');
    END IF ;
    IF enable = "01" THEN output <= input2 ;
        ELSE output <=(OTHERS => 'Z');
    END IF ;
    IF enable = "10" THEN output <= input1 ;
        ELSE output <=(OTHERS => 'Z');
    END IF ;
    IF enable = "11" THEN output <= input0 ;
        ELSE output <=(OTHERS => 'Z');
    END IF ;
END PROCESS;
END multiple_drivers;
```

3.7 双向和三态电路信号赋值

3.7.3 三态总线电路设计

【例3-21】

```
library ieee;
use ieee.std_logic_1164.all;
entity tri2 is
port (ctl : in  std_logic_vector(1 downto 0);
         datain1, datain2, datain3, datain4 :
              in std_logic_vector(7 downto 0);
        q : out std_logic_vector(7 downto 0)   );
end tri2;
architecture body_tri of tri2 is
begin
 q <= datain1   when ctl="00" else  (others =>'Z') ;
 q <= datain2   when ctl="01" else  (others =>'Z') ;
 q <= datain3   when ctl="10" else  (others =>'Z') ;
 q <= datain4   when ctl="11" else  (others =>'Z') ;
end body_tri;
```

3.7 双向和三态电路信号赋值

3.7.3 三态总线电路设计

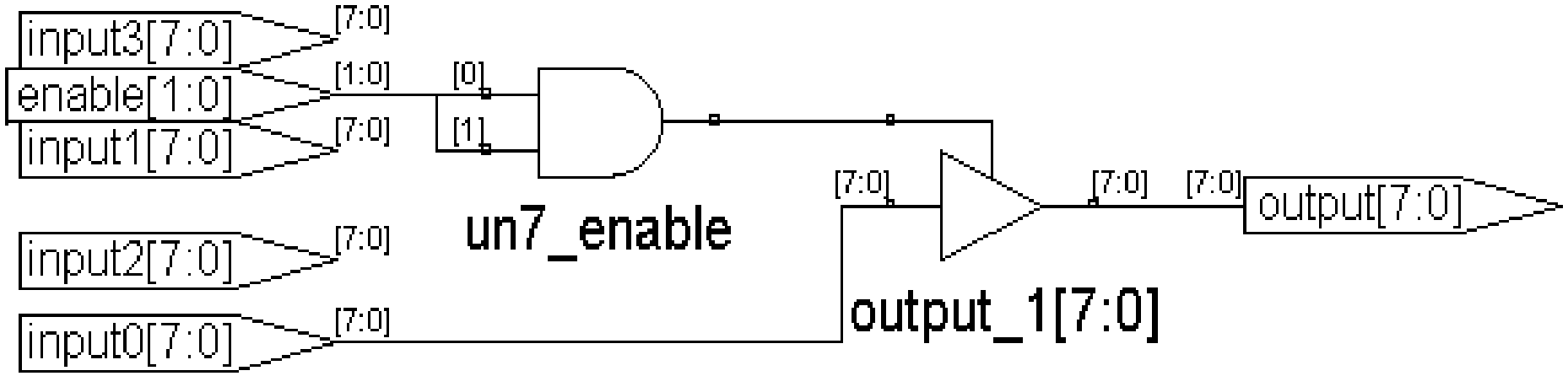


图3-27 例3-20错误的综合结果（Synplify综合结果）

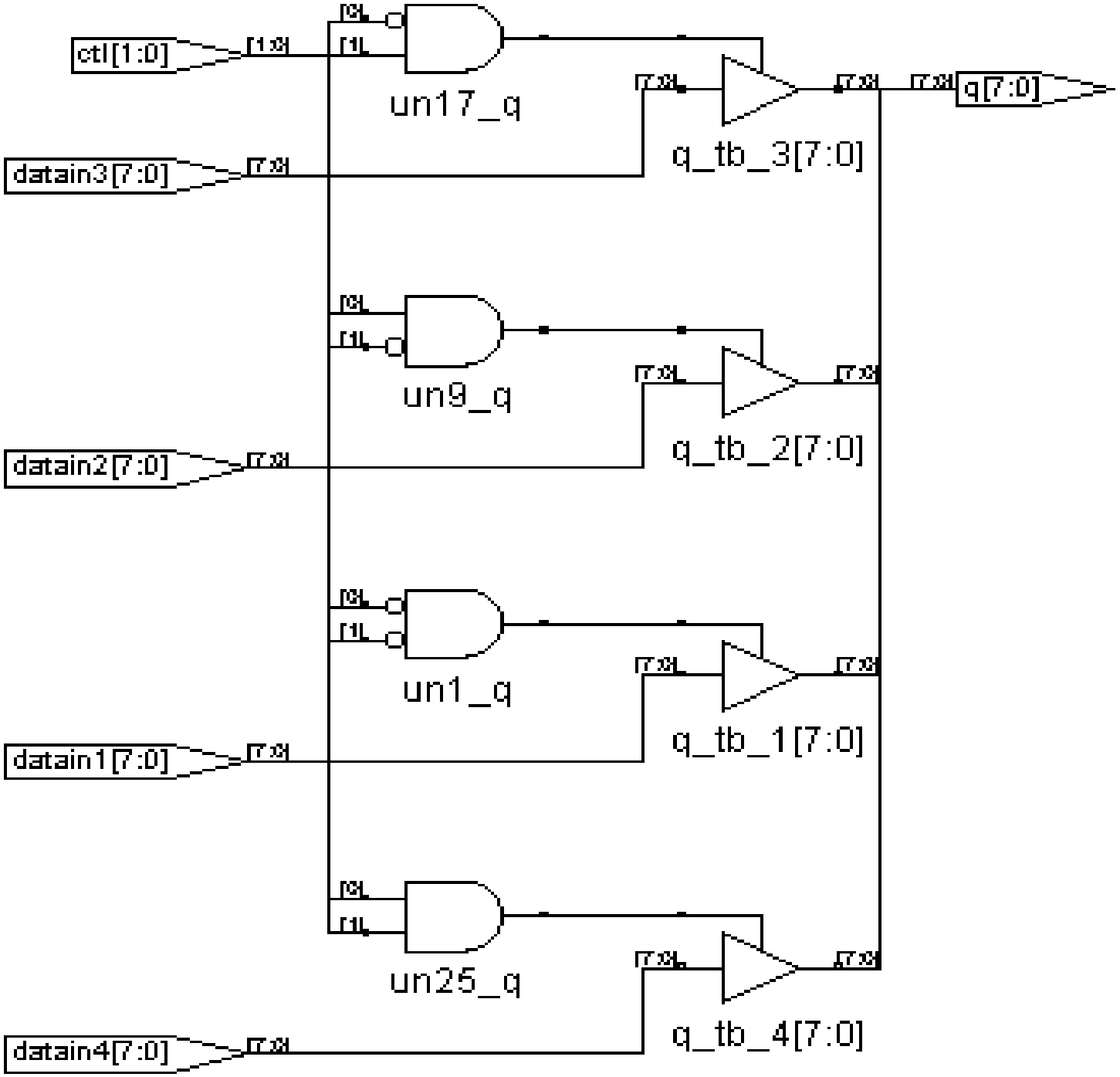


图3-28 例3-21正确的综合结果（Synplify综合结果）

3.8 LOOP语句与GENERIC语句

3.8.1 LOOP语句

（1） 单个LOOP语句。

```
[ LOOP标号: ] LOOP
      顺序语句
END LOOP [ LOOP标号 ];
```

```
...
L2 : LOOP
     a := a+1;
     EXIT L2 WHEN a >10 ; -- 当a大于10时跳出循环
     END LOOP L2;
```

(2) FOR_LOOP语句。

```
[LOOP标号: ] FOR 循环变量, IN  循环次数范围  LOOP
          顺序语句
END LOOP [LOOP标号];
```

3.8 LOOP语句与GENERIC语句

3.8.1 LOOP语句

【例3-22】

```
 LIBRARY IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;
ENTITY p_check IS
 PORT (  a : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
          y : OUT STD_LOGIC );
END p_check;
ARCHITECTURE opt OF p_check IS
  SIGNAL tmp : STD_LOGIC ;
BEGIN
  PROCESS(a)
    BEGIN
     tmp <='0';
     FOR n IN 0 TO 7 LOOP
      tmp <= tmp XOR a(n);
     END LOOP ;
      y <= tmp;
     END PROCESS;
    END opt;
```

3.8 LOOP语句与GENERIC语句

3.8.1 LOOP语句

【例3-23】

```
    SIGNAL a, b, c : STD_LOGIC_VECTOR (1 TO 3);
...
FOR n IN 1 To 3 LOOP
a(n) <= b(n) AND c(n);
END LOOP;
```

```
a(1)<=b(1) AND c(1);
a(2)<=b(2) AND c(2);
a(3)<=b(3) AND c(3);
```

3.8 LOOP语句与GENERIC语句

3.8.2 GENERIC参数传递说明语句

参数传递说明语句的一般书写格式如下：

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

【例3-24】

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

【例3-25】

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

3.8 LOOP语句与GENERIC语句

3.8.3 参数传递映射语句

GENERIC MAP(类属表)

【例3-26】

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

【例3-27】

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

习 题

3-1. 什么是固有延时？什么是惯性延时？

3-2. δ是什么？在**VHDL**中，δ有什么用处？

3-3. 哪些情况下需要用到程序包**STD_LOGIC_UNSIGNED**？试举一例。

3-4. 说明信号和变量的功能特点，应用上的异同点。

3-5. 在**VHDL**设计中，给时序电路清**0**(复位)有两种方法，它们是什么？

3-6. 哪一种复位方法必须将复位信号放在敏感信号表中？给出这两种电路的**VHDL**描述。

3-7. 什么是重载函数？重载算符有何用处？如何调用重载算符函数？

习 题

3-8. 判断下面3个程序中是否有错误，若有则指出错误所在，并给出完整程序。

程序1：

```
Signal A, EN : std_logic;
Process (A, EN)
    Variable B : std_logic;
Begin
if EN = 1 then   B <= A;  end if;
end process;
```

程序2：

```
Architecture one of sample is
    variable a, b, c : integer;
begin
    c <= a + b;
end;
```

习 题

3-8. 判断下面3个程序中是否有错误，若有则指出错误所在，并给出完整程序。

程序3：

```
library ieee;
use ieee.std_logic_1164.all;
entity mux21 is
    port ( a, b : in std_logic; sel : in std_logic; c : out std_logic;);
end sam2;
architecture one of mux21 is
begin
if sel = '0' then   c := a; else    c := b;  end if;
end two;
```

习 题

3-9. 根据例2-23设计8位左移移位寄存器，给出时序仿真波形。

3-10 序列检测器可用于检测一组或多组由二进制码组成的脉冲序列信号，当序列检测器连续收到一组串行二进制码后，如果这组码与检测器中预先设置的码相同，则输出1，否则输出0。由于这种检测的关键在于正确码的收到必须是连续的，这就要求检测器必须记住前一次的正确码及正确序列，直到在连续的检测中所收到的每一位码都与预置数的对应码相同。在检测过程中，任何一位不相等都将回到初始状态重新开始检测。例3-28描述的电路完成对序列数“11100101”的检测，当这一串序列数高位在前(左移)串行进入检测器后，若此数与预置的密码数相同，则输出“A”，否则仍然输出“B”。

习 题

【例3-28】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SCHK IS
  PORT(DIN，CLK，CLR  : IN STD_LOGIC; --串行输入数据位/工作时钟/复位信号
       AB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)); --检测结果输出
END SCHK;
ARCHITECTURE behav OF SCHK IS
   SIGNAL Q : INTEGER RANGE 0 TO 8 ;
   SIGNAL D : STD_LOGIC_VECTOR(7 DOWNTO 0); --8位待检测预置数(密码
=E5H)
BEGIN
   D <= "11100101 " ; --8位待检测预置数
  PROCESS( CLK, CLR )
  BEGIN
  IF CLR = '1' THEN    Q <= 0 ;
  ELSIF  CLK'EVENT AND CLK='1' THEN  --时钟到来时，判断并处理当前输入的位
```

（接下页）

```
CASE Q IS
  WHEN 0=>  IF DIN = D(7) THEN Q <= 1 ; ELSE Q <= 0 ; END IF ;
  WHEN 1=>  IF DIN = D(6) THEN Q <= 2 ; ELSE Q <= 0 ; END IF ;
  WHEN 2=>  IF DIN = D(5) THEN Q <= 3 ; ELSE Q <= 0 ; END IF ;
  WHEN 3=>  IF DIN = D(4) THEN Q <= 4 ; ELSE Q <= 0 ; END IF ;
  WHEN 4=>  IF DIN = D(3) THEN Q <= 5 ; ELSE Q <= 0 ; END IF ;
  WHEN 5=>  IF DIN = D(2) THEN Q <= 6 ; ELSE Q <= 0 ; END IF ;
  WHEN 6=>  IF DIN = D(1) THEN Q <= 7 ; ELSE Q <= 0 ; END IF ;
  WHEN 7=>  IF DIN = D(0) THEN Q <= 8 ; ELSE Q <= 0 ; END IF ;
  WHEN OTHERS =>  Q <= 0 ;
      END CASE ;
    END IF ;
  END PROCESS ;
  PROCESS( Q )                          --检测结果判断输出
  BEGIN
     IF Q = 8  THEN  AB <= "1010" ;    --序列数检测正确，输出 “A”
     ELSE            AB <= "1011" ;    --序列数检测错误，输出 “B”
     END IF ;
  END PROCESS ;
END behav ;
```

习 题

要求1：说明例3-28表达的是什么类型的状态机，它的优点是什么？详述其功能和对序列数检测的逻辑过程。

要求2：根据上例，写出由两个主控进程构成的相同功能的符号化Moore型有限状态机，画出状态图，并给出其仿真测试波形。

要求3：将8位待检测预置数作为外部输入信号，即可以随时改变序列检测器中的比较数据。写出此程序的符号化单进程有限状态机。

提示：对于D<=“11100101”，电路需分别不间断记忆：初始状态、1、11、111 、 1110 、11100、111001、1110010、11100101 共9种状态。

实验与设计

实验3-1. 七段数码显示译码器设计

参考实验示例和实验课件：**/CMPUT_EXPMT/CH3_Expt/DEMO_31_DECL7S/** 和实验**3_1.ppt** 。

(1) 实验目的：学习7段数码显示译码器设计；学习**VHDL**的**CASE**语句应用及多层次设计方法。

(2) 实验原理：7段数码是纯组合电路，通常的小规模专用**IC**，如**74**或**4000**系列的器件只能作十进制**BCD**码译码，然而数字系统中的数据处理和运算都是2进制的，所以输出表达都是**16**进制的，为了满足**16**进制数的译码显示，最方便的方法就是利用译码程序在**FPGA/CPLD**中来实现。例**3-23**作为7段译码器，输出信号**LED7S**的7位分别接如图**3-23**数码管的7个段，高位在左，低位在右。例如当**LED7S**输出为“**1101101**”时，数码管的7个段：**g**、**f**、**e**、**d**、**c**、**b**、**a**分别接**1**、**1**、**0**、**1**、**1**、**0**、**1**；接有高电平的段发亮，于是数码管显示“**5**”。注意，这里没有考虑表示小数点的发光管，如果要考虑，需要增加段**h**，例**3-29**中的**LED7S:OUT STD_LOGIC_VECTOR(6 DOWNTO 0)**应改为 **... (7 DOWNTO 0)** 。

实验与设计

(3) 实验任务1：说明例3-29中各语句的含义，以及该例的整体功能。在QuartusII上对该例进行编辑、编译、综合、适配、仿真，给出其所有信号的时序仿真波形。

提示：用输入总线的方式给出输入信号仿真数据，仿真波形示例图如图3-29所示。

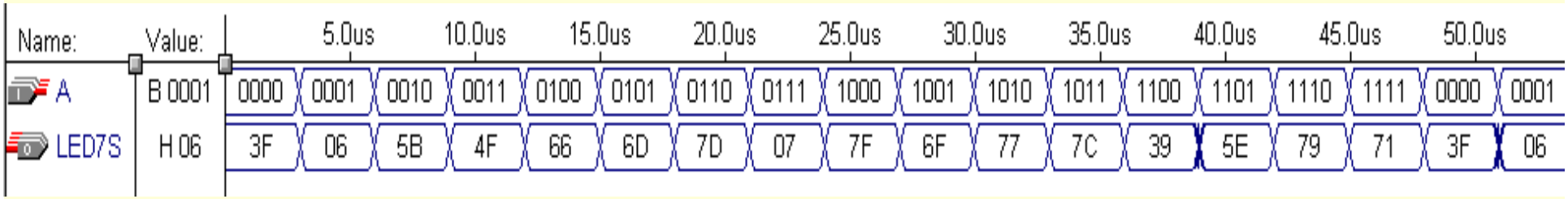


图3-29 7段译码器仿真波形

【例3-29】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY DECL7S IS
 PORT ( A  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
     LED7S : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)  ) ;
END ;
ARCHITECTURE one 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 ;
```

实验与设计

(4) 实验任务2

引脚锁定及硬件测试。建议选GW48系统的实验电路模式6，用数码8显示译码输出(PIO46-PIO40)，键8、键7、键6和键5四位控制输入，硬件验证译码器的工作性能。

(5) 实验任务3

用第2章介绍的例化语句，按图3-31的方式连接成顶层设计电路（用VHDL表述），图中的CNT4B是一个4位二进制加法计数器，可以由例2-22修改获得；模块DECL7S即为例3-29实体元件，重复以上实验过程。对于引脚锁定和实验，建议选电路模式6，用数码8显示译码输出，用键3作为时钟输入(每按2次键为1个时钟脉冲)，或直接接时钟信号clock0。

(6) 实验报告

根据以上的实验内容写出实验报告，包括程序设计、软件编译、仿真分析、硬件测试和实验过程；设计程序、程序分析报告、仿真波形图及其分析报告。

实验与设计

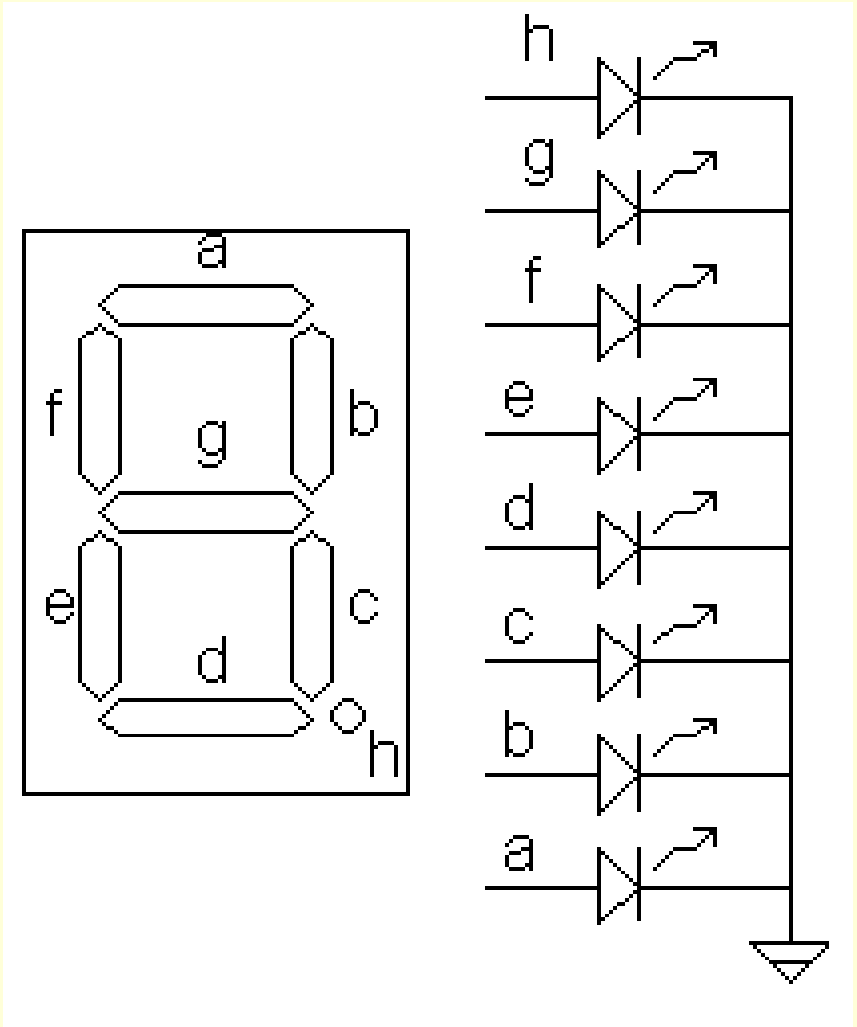


图3-30共阴数码管
及其电路

实验与设计

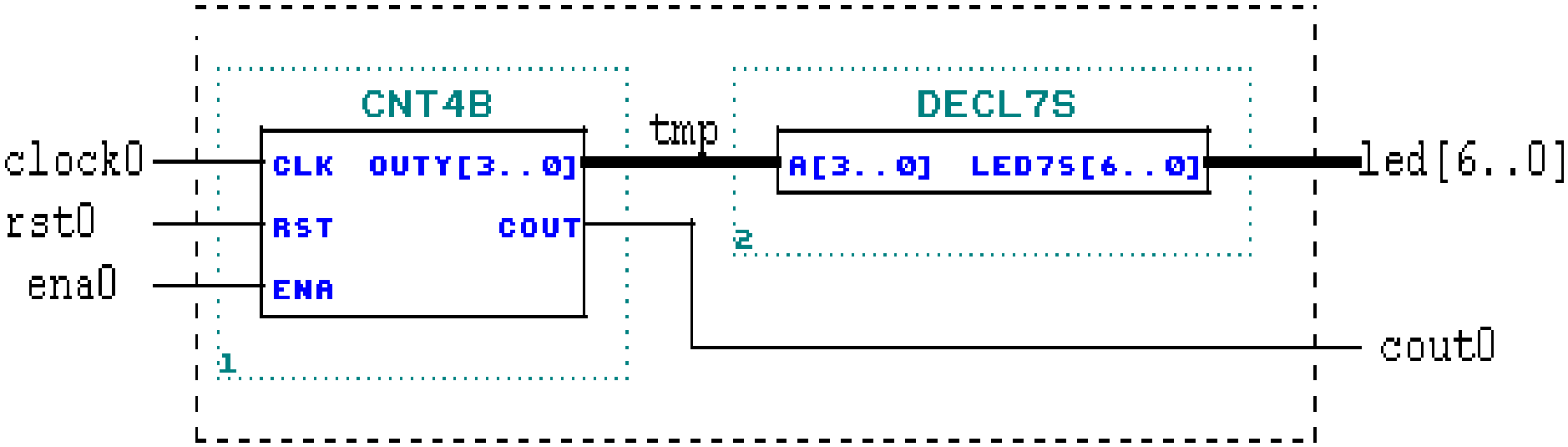


图3-31 计数器和译码器连接电路的顶层文件原理图

实验与设计

实验3-2. 数控分频器的设计

参考实验示例和实验课件：

/CMPUT_EXPMT/CH3_Expt/DEMO_32_DVF/ 和 实验**3_2.ppt** 。

(1) 实验目的

学习数控分频器的设计、分析和测试方法。

(2) 实验原理

数控分频器的功能就是当在输入端给定不同输入数据时，将对输入的时钟信号有不同的分频比，数控分频器就是用计数值可并行预置的加法计数器设计完成的，方法是将计数溢出位与预置数加载输入信号相接即可，详细设计程序如例**3-30**所示。

实验与设计

(3) 实验任务1

根据图3-32的波形，分析例3-30中的各语句功能、设计原理及逻辑功能，详述进程P_REG和P_DIV的作用，并画出该程序的RTL电路图。输入不同的CLK频率和预置值D，给出如图3-22的时序波形

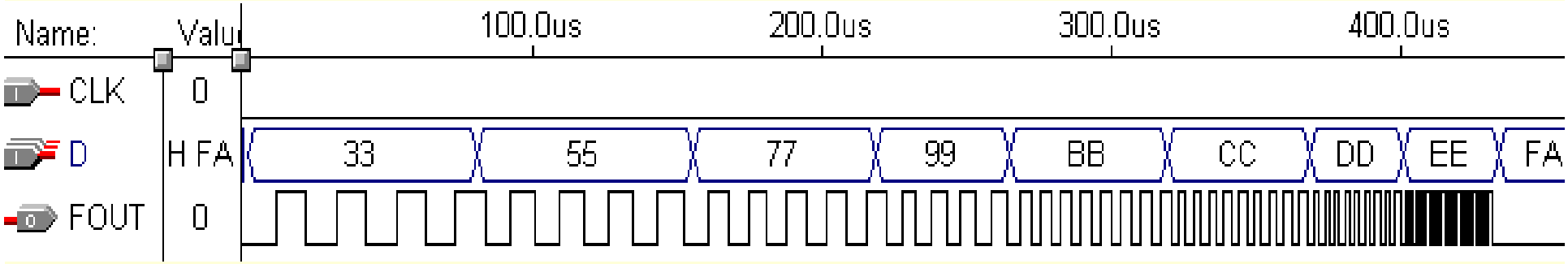


图3-32 当给出不同输入值D时，FOUT输出不同频率(CLK周期=50ns)

实验与设计

(4) 实验任务2

在实验系统上硬件验证例3-30的功能。可选实验电路模式1（参考附录图2）；键2/键1负责输入8位预置数D(PIO7-PIO0)；CLK由clock0输入，频率选65536Hz或更高(确保分频后落在音频范围)；输出FOUT接扬声器(SPKER)。编译下载后进行硬件测试：改变键2/键1的输入值，可听到不同音调的声音。

(5) 思考题

怎样利用2个例3-30给出的模块设计一个电路，使其输出方波的正负脉宽的宽度分别由可两个8位输入数据控制？

【例3-30】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DVF IS
   PORT (   CLK  : IN STD_LOGIC;
            D  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           FOUT : OUT STD_LOGIC  );
END;
ARCHITECTURE one OF DVF IS
   SIGNAL   FULL : STD_LOGIC;
BEGIN
  P_REG: PROCESS(CLK)
   VARIABLE CNT8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
   BEGIN
     IF CLK'EVENT AND CLK = '1' THEN
          IF CNT8 = "11111111" THEN
          CNT8 := D;     --当CNT8计数计满时，输入数据D被同步预置给计数器CNT8
           FULL <= '1'; --同时使溢出标志信号FULL输出为高电平
             ELSE   CNT8 := CNT8 + 1;  --否则继续作加1计数
                   FULL <= '0';       --且输出溢出标志信号FULL为低电平
          END IF;
      END IF;
    END PROCESS P_REG ;
   P_DIV: PROCESS(FULL)
    VARIABLE CNT2 : STD_LOGIC;
   BEGIN
   IF FULL'EVENT AND FULL = '1' THEN
     CNT2 := NOT CNT2; --如果溢出标志信号FULL为高电平，D触发器输出取反
        IF CNT2 = '1' THEN  FOUT <= '1'; ELSE FOUT <= '0';
       END IF;
   END IF;
    END PROCESS P_DIV ;
END;
```

实验与设计

实验3-3. 8位16进制频率计设计

参考实验示例和实验课件：

/CMPUT_EXPMT/CH3_Expt/DEMO_33_FRQTEST/ 和 实验3_3.ppt 。

(1) 实验目的

设计**8**位**16**进制频率计，学习较复杂的数字系统设计方法。

(2) 实验原理

根据频率的定义和频率测量的基本原理，测定信号的频率必须有一个脉宽为**1**秒的输入信号脉冲计数允许的信号；**1**秒计数结束后，计数值被锁入锁存器，计数器清**0**，为下一测频计数周期作好准备。测频控制信号可以由一个独立的发生器来产生，即图**3-34**中的**FTCTRL**。根据测频原理，测频控制时序可以如图**3-33**所示。

实验与设计

设计要求是：**FTCTRL**的计数使能信号**CNT_EN**能产生一个**1**秒脉宽的周期信号，并对频率计中的**32**位二进制计数器**COUNTER32B**（图**3-34**）的**ENABL**使能端进行同步控制。当**CNT_EN**高电平时允许计数；低电平时停止计数，并保持其所计的脉冲数。在停止计数期间，首先需要一个锁存信号**LOAD**的上跳沿将计数器在前**1**秒钟的计数值锁存进锁存器**REG32B**中，并由外部的**16**进制**7**段译码器译出，显示计数值。设置锁存器的好处是数据显示稳定，不会由于周期性的清**0**信号而不断闪烁。锁存信号后，必须有一清**0**信号**RST_CNT**对计数器进行清零，为下**1**秒的计数操作作准备。

(3) 实验任务1

完成频率计的完整设计和硬件实现，并给出其测频时序波形及其分析。建议选实验电路模式**5**；**8**个数码管以**16**进制形式显示测频输出；待测频率输入**FIN**由**clock0**输入，频率可选**4Hz**、**256HZ**、**3Hz...50MHz**等；**1HZ**测频控制信号**CLK1HZ**可由**clock2**输入(用跳线选**1Hz**)。注意，这时**8**个数码管的测频显示值是**16**进制的。

实验与设计

(4) 实验任务2

参考例2-22，将频率计改为8位10进制频率计，注意此设计电路的计数器必须是8个4位的10进制计数器，而不是1个。此外注意在测频速度上给予优化。

(5) 实验报告：给出频率计设计的完整实验报告。

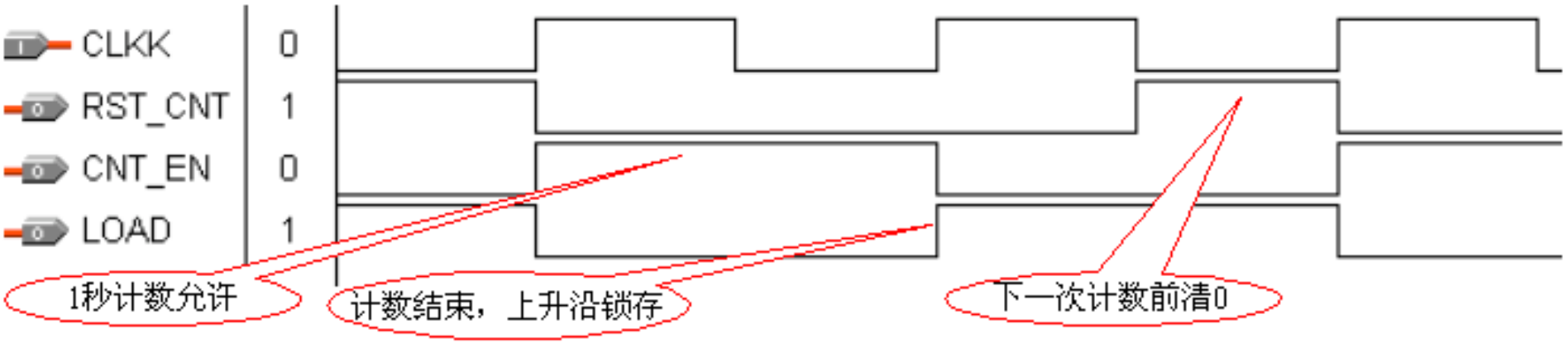


图3-33 频率计测频控制器FTCTRL测控时序图

实验与设计

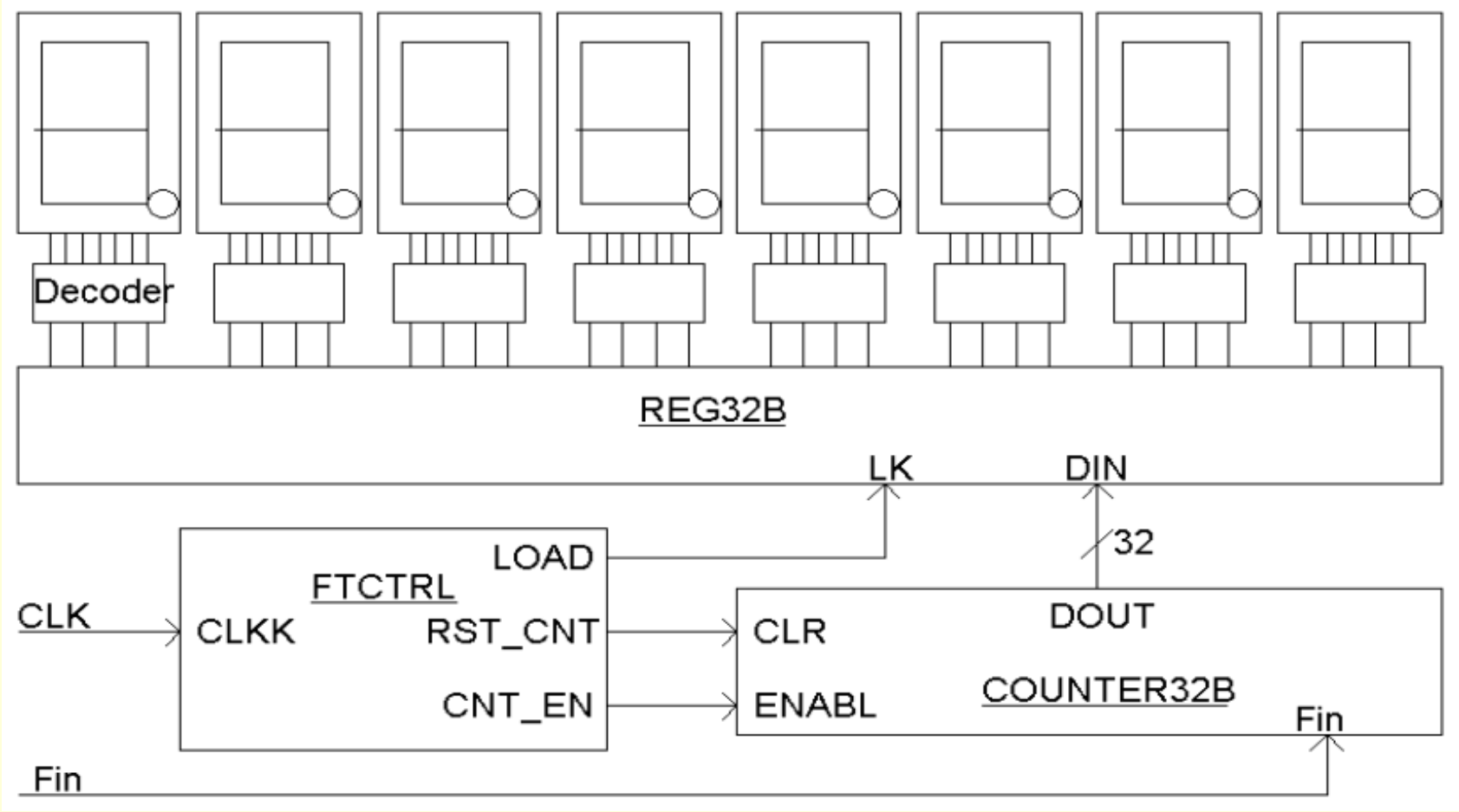


图3-34 频率计电路框图

实验与设计

实验3-4 ADC0809采样控制电路实现

参考实验示例和实验课件：/CMPUT_EXPMT/CH3_Expt/DEMO_34_ADCINT/和实验3_4.ppt 。

(1) 实验目的：学习用状态机对A/D转换器ADC0809的采样控制电路的实现。

(2) 实验原理：实验程序用例3-14。ADC0809是CMOS的8位A/D转换器，片内有8路模拟开关，可控制8个模拟量中的一个进入转换器中。转换时间约100μs，含锁存控制的8路多路开关，输出有三态缓冲器控制，单5V电源供电。

主要控制信号如图3-14所示：START是转换启动信号，高电平有效；ALE是3位通道选择地址(ADDC、ADDB、ADDA)信号的锁存信号。当模拟量送至某一输入端(如IN1或IN2等)，由3位地址信号选择，而地址信号由ALE锁存；EOC是转换情况状态信号，当启动转换约100μs后，EOC产生一个负脉冲，以示转换结束；在EOC的上升沿后，若使输出使能信号OE为高电平，则控制打开三态缓冲器，把转换好的8位数据结果输至数据总线，至此ADC0809的一次转换结束。

实验与设计

(3) 实验任务：利用QuartusII对例3-14进行文本编辑输入和仿真测试；给出仿真波形。最后进行引脚锁定并进行测试，硬件验证例3-14电路对ADC0809的控制功能。

测试步骤：建议选择电路模式5，由对应的电路图可见，ADC0809的转换时钟CLK已经事先接有750kHz的频率，引脚锁定为：START接PIO34，OE（ENABLE）接PIO35，EOC接PIO8，ALE接PIO33，状态机时钟CLK接clock0，ADDA接PIO32(ADDB和ADDC都接GND)，ADC0809的8位输出数据线接PIO23～PIO16，锁存输出Q显示于数码8/数码7(PIO47～PIO40)。

实验操作：将GW48系统拨码开关的4、6、7向下拨，其余向上，即使0809工作使能，及使FPGA能接受来自0809转换结束的信号。下载ADC0809中的ADCINT.sof到实验板的FPGA中；clock0的短路帽接可选12MHz、6MHz、65536Hz等频率；按动一次右侧的复位键；用螺丝刀旋转GW48系统精密电位器，以便为ADC0809提供变化的待测模拟信号（注意，这时必须在例3-14中赋值：ADDA <= '1'，这样就能通过实验系统左下的AIN1输入端与电位器相接，并将信号输入0809的IN1端）。这时数码管8和7将显示ADC0809采样的数字值（16进制），数据来自FPGA的输出。数码管2和1也将显示同样数据，此数据直接来自0809的数据口。实验结束后注意将拨码开关拨向默认：仅“4”向下。

(4) 实验报告：根据以上的实验要求、实验内容和实验思考题写出实验报告。

实验与设计

实验3-5 序列检测器设计

参考实验示例和实验课件：

/CMPUT_EXPMT/CH3_Expt/DEMO_35_SCHK/ 和实验**3_5.ppt** 。

(1) 实验目的：用状态机实现序列检测器的设计，了解一般状态机的设计与应用。

(2) 实验原理：序列检测器的工作原理已在习题**3-10**中作了说明。

(3) 实验任务**1**：仔细完成习题**3-10**的全部内容，利用**QuartusII**对例**3-28**进行文本编辑输入、仿真测试并给出仿真波形，了解控制信号的时序，最后进行引脚锁定并完成硬件测试实验。

实验与设计

建议选择电路模式8，用键7(PIO11)控制复位信号CLR；键6(PIO9)控制状态机工作时钟CLK；待检测串行序列数输入DIN接PIO10(左移，最高位在前)；指示输出AB接PIO39～PIO36(显示于数码管6)。下载后：①按实验板“系统复位”键；②用键2和键1输入2位十六进制待测序列数“11100101”；③按键7复位(平时数码6指示显“B”)；④按键6(CLK) 8次，这时若串行输入的8位二进制序列码(显示于数码2/1和发光管D8～D0)与预置码“11100101”相同，则数码6应从原来的B变成A，表示序列检测正确，否则仍为B。

(4) 实验任务2：根据习题3-10中的要求3提出的设计方案，重复以上实验内容(将8位待检测预置数由键4/键3作为外部输入，从而可随时改变检测密码)。

(5) 实验思考题：如果待检测预置数必须以右移方式进入序列检测器，写出该检测器的VHDL代码(两进程符号化有限状态机)，并提出测试该序列检测器的实验方案。