EDA技术及其应用

第6章 实用状态机设计技术

6.1 有限状态机设计初步

6.1.1 为什么要使用状态机

1、状态机克服了纯硬件数字系统顺序方式控制不灵活的缺点。
2、由于状态机的结构相对简单，设计方案相对固定，特别是可以定义符号化枚举类型的状态，这一切都为VHDL综合器尽可能发挥其强大的优化功能提供了有利条件。
3、状态机容易构成性能良好的同步时序逻辑模块，这对于对付大规模逻辑电路设计中令人深感棘手的竞争冒险现象无疑是一个上佳的选择。此外为了消除电路中的毛刺现象，在状态机设计中有更多的设计方案可供选择。
4、与VHDL的其他描述方式相比，状态机的VHDL表述丰富多样、程序层次分明，结构清晰，易读易懂；在排错、修改和模块移植方面也有其独到的好处。
5、在高速运算和控制方面，状态机更有其巨大的优势。
6、高可靠性。

6.1 有限状态机设计初步

6.1.2 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 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 BOOLEAN IS (FALSE，TRUE) ;
 TYPE my _logic IS ( '1' , 'Z' , 'U' , '0' ) ;
 SIGNAL s1 : my _logic ;
   s1 <= 'Z' ;
```

6.1 有限状态机设计初步

6.1.3 一般有限状态机的结构

1. 说明部分

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

6.1 有限状态机设计初步

6.1.3 一般有限状态机的结构

2. 主控时序进程

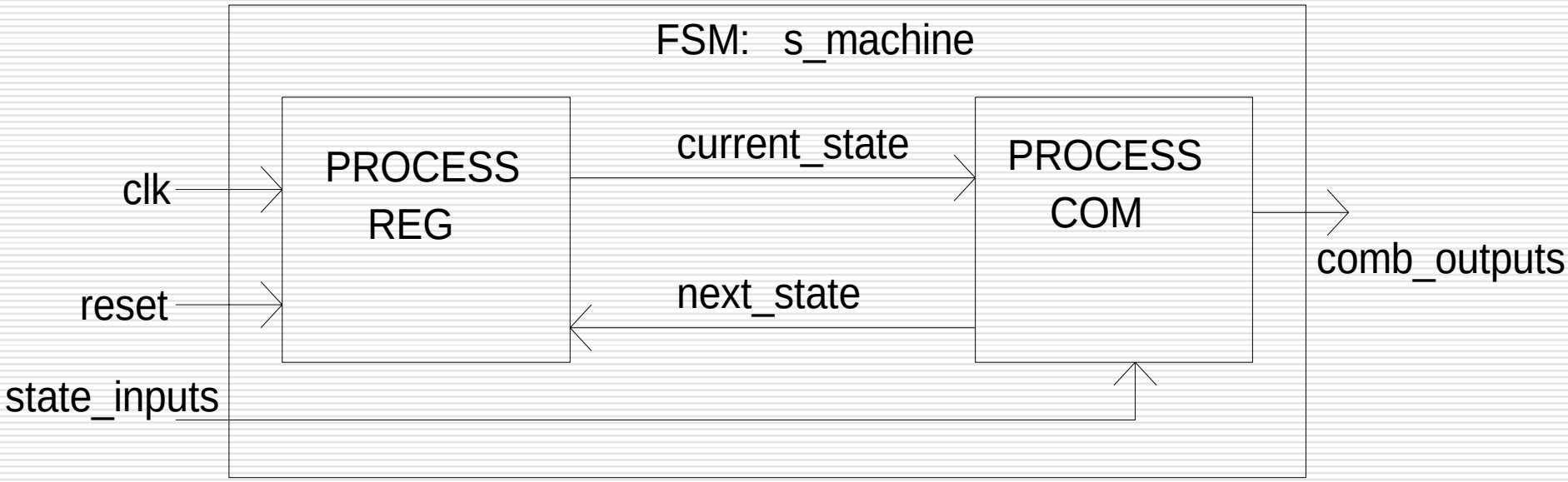

图6-1 一般状态机结构图

6.1 有限状态机设计初步

6.1.3 一般有限状态机的结构

3. 主控组合进程

4. 辅助进程

【例6-1】

```
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;
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;
      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;
```

6.1 有限状态机设计初步

6.1.3 一般有限状态机的结构

4. 辅助进程

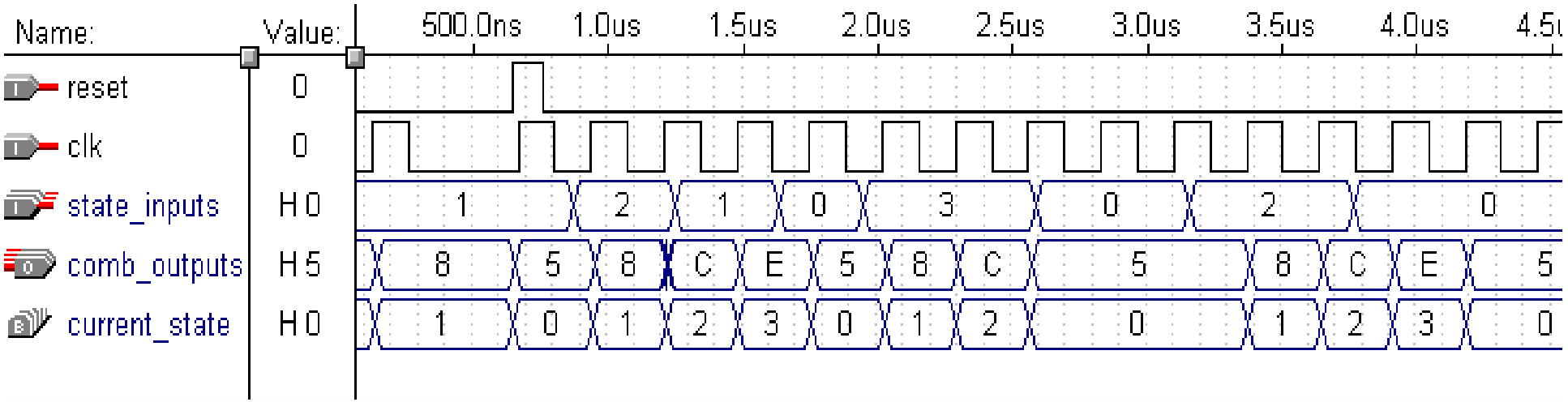

图6-2 例6-1的工作时序

6.1 有限状态机设计初步

6.1.3 一般有限状态机的结构

4. 辅助进程

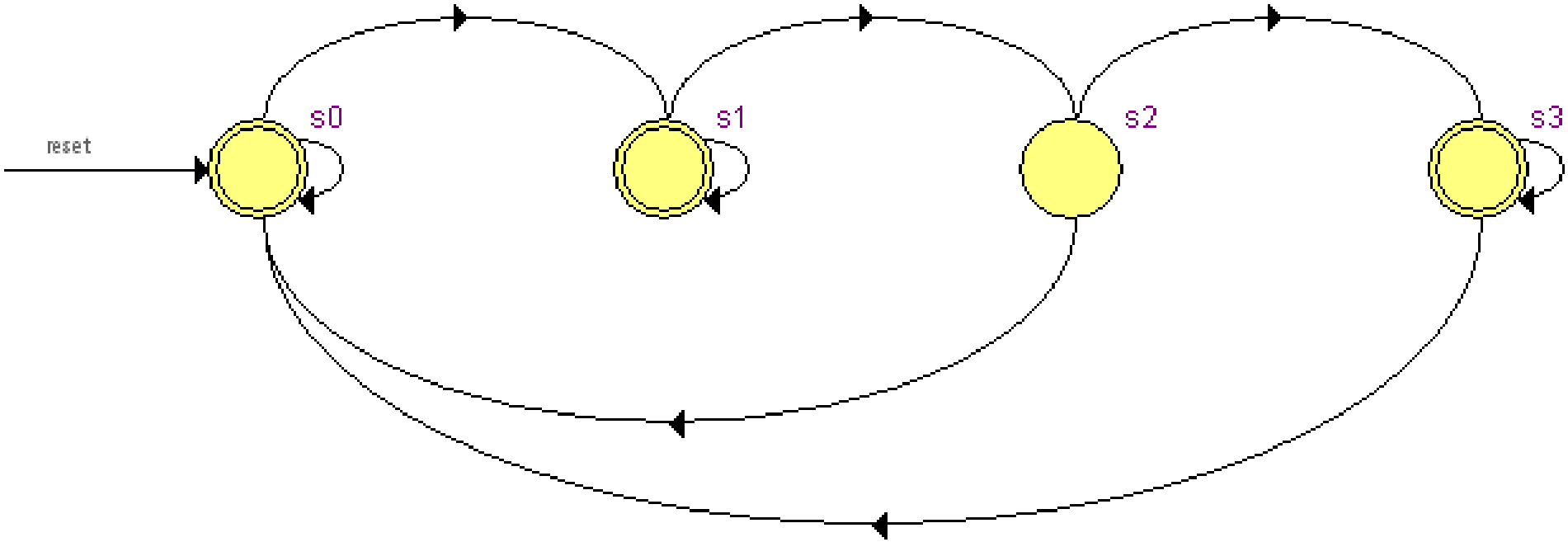

图6-3 通过State Machine Viewer观察到的例6-1的状态图

6.2 Moore型有限状态机设计

6.2.1 多进程有限状态机

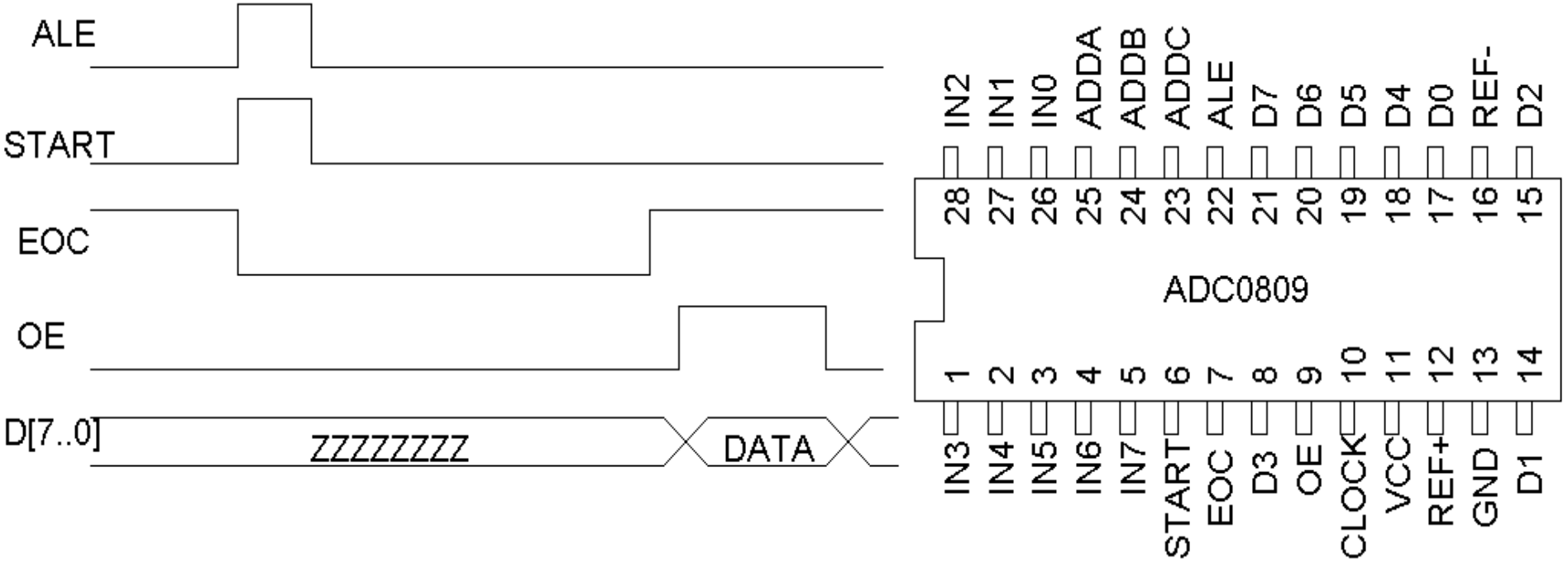

图6-4 ADC0809工作时序

6.2 Moore型有限状态机设计

6.2.1 多进程有限状态机

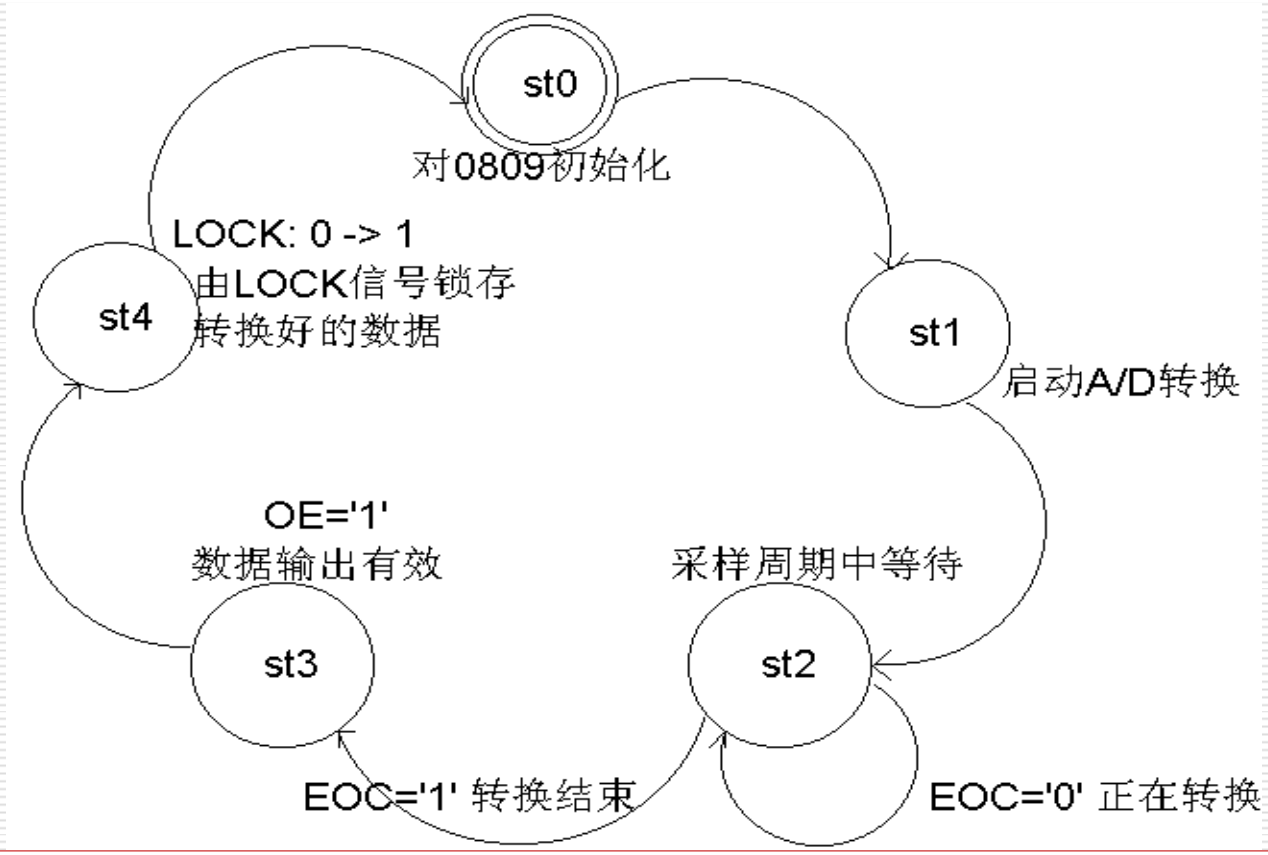

图6-5 控制ADC0809采样状态图

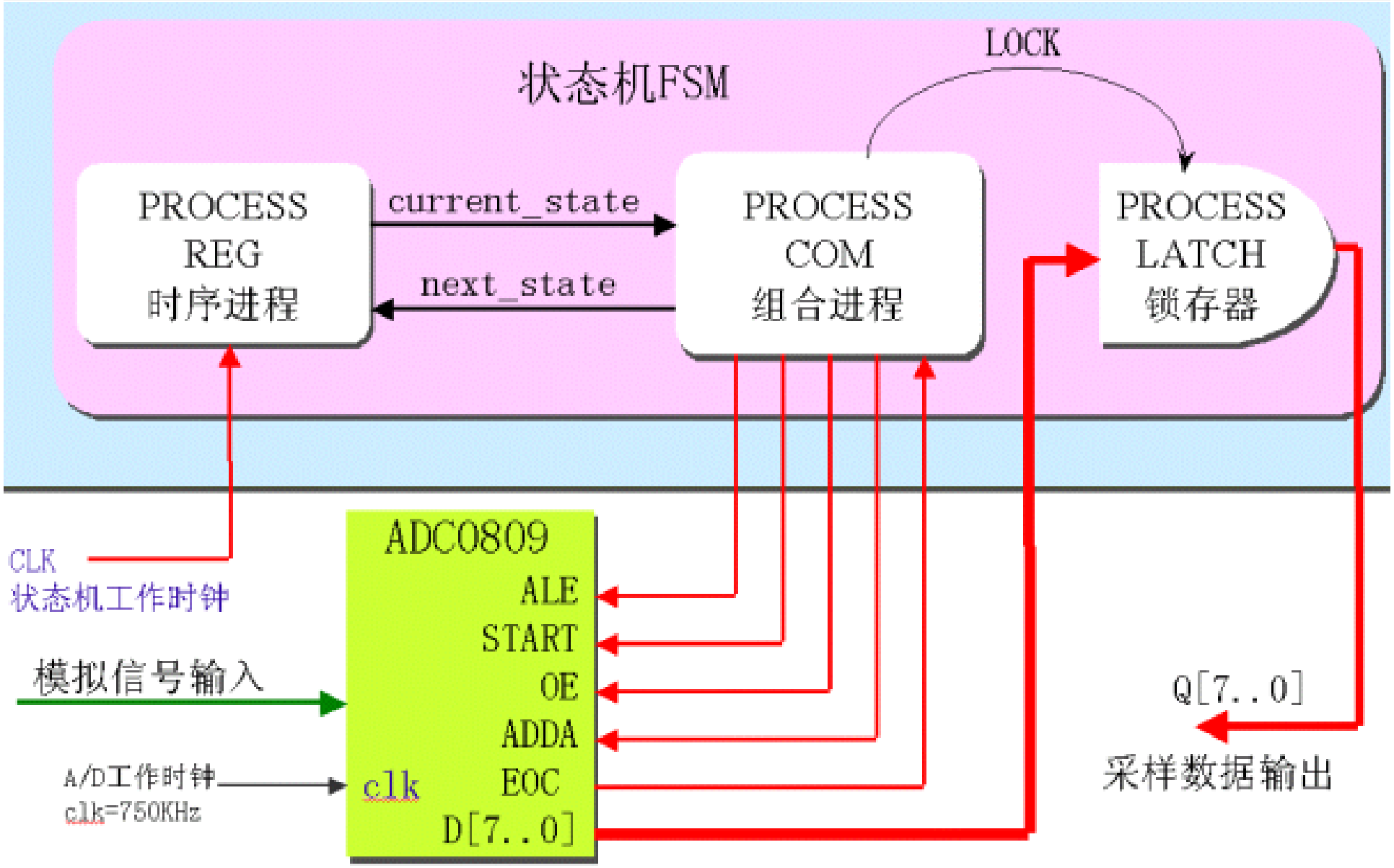

图6-6 采样状态机结构框图

【例6-2】

```
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 ;
 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 ;
```

【例6-3】

```
COM1: PROCESS (current _state ,EOC)  BEGIN
  CASE current _state IS
  WHEN st0=> next _state <= st1;
  WHEN st1=> next _state <= st2;
  WHEN st2=> IF (EOC='1') THEN next _state <= st3;
                ELSE next _state <= st2; END IF ;
  WHEN st3=> next _state <= st4;--开启OE
  WHEN st4=> next _state <= st0;
  WHEN OTHERS => next _state <= st0;
  END CASE ;
 END PROCESS COM1 ;
 COM2: PROCESS (current _state)  BEGIN
  CASE current _state IS
  WHEN st0=>ALE<='0';START<='0';LOCK<='0';OE<='0' ;
  WHEN st1=>ALE<='1';START<='1';LOCK<='0';OE<='0' ;
  WHEN st2=>ALE<='0';START<='0';LOCK<='0';OE<='0' ;
  WHEN st3=>ALE<='0';START<='0';LOCK<='0';OE<='1' ;
  WHEN st4=>ALE<='0';START<='0';LOCK<='1';OE<='1' ;
  WHEN OTHERS => ALE<='0';START<='0';LOCK<='0';
  END CASE ;
 END PROCESS COM2 ;
```

6.2 Moore型有限状态机设计

6.2.1 多进程有限状态机

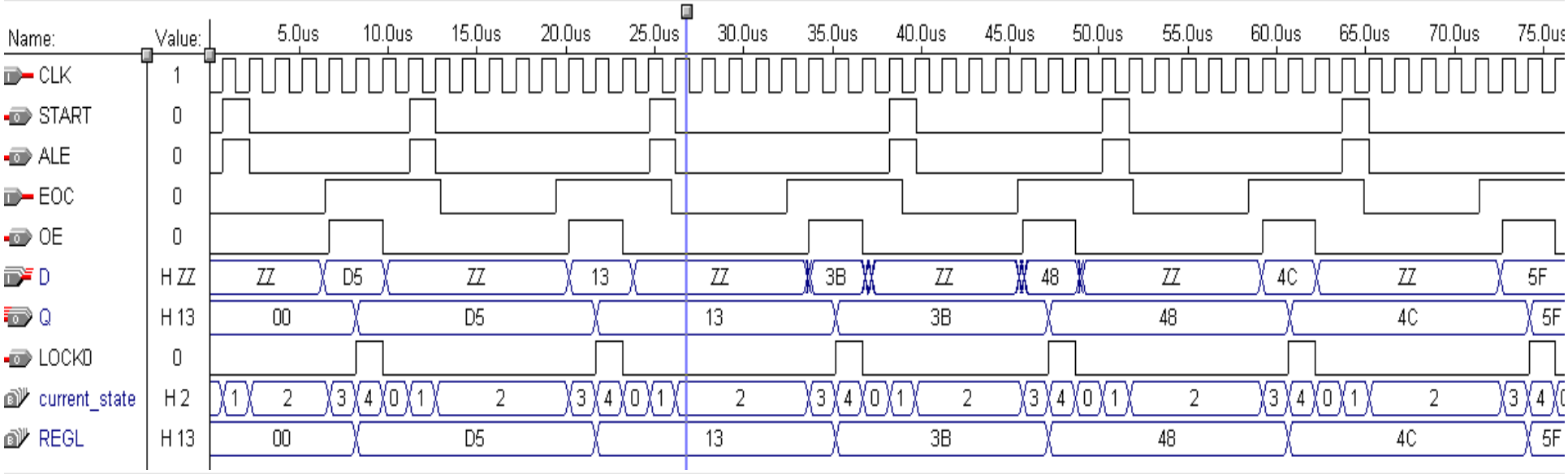

图6-7 ADC0809采样状态机工作时序

6.2.2 单进程Moore型有限状态机

【例6-4】

```
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)
     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;
```

6.2 Moore型有限状态机设计

6.2.2 单进程Moore型有限状态机

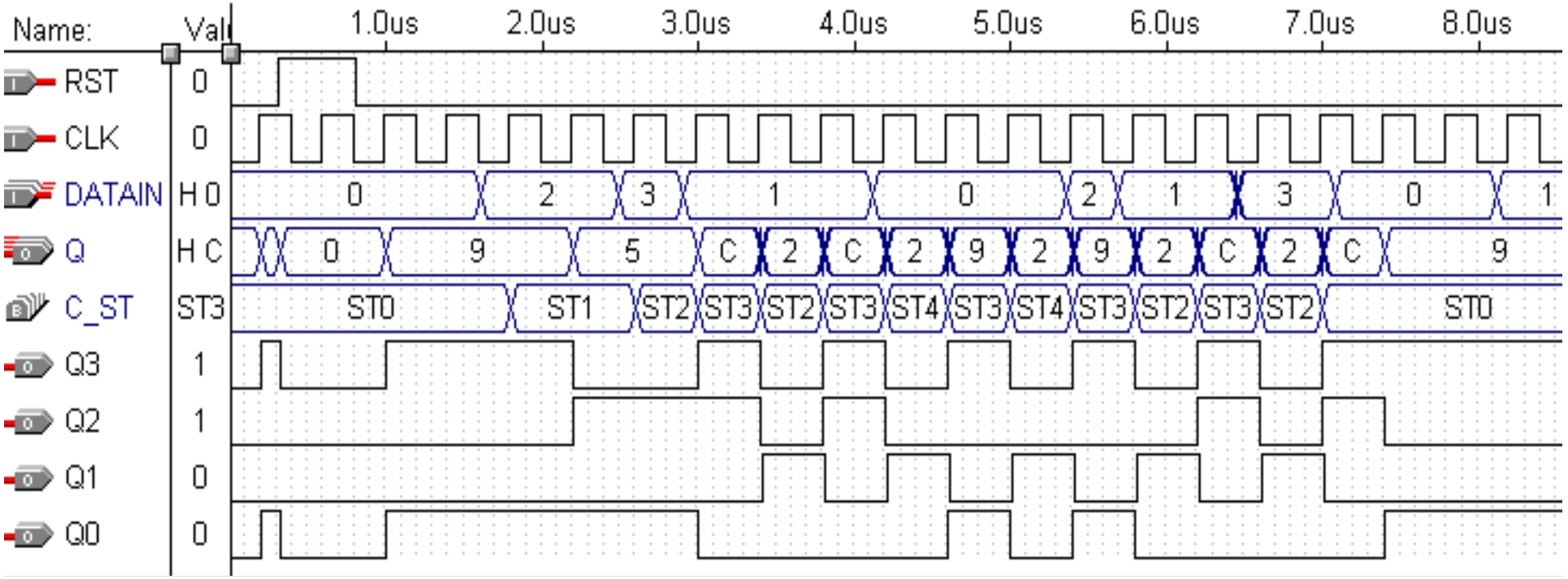

图6-8 例6-4单进程状态机工作时序

6.2 Moore型有限状态机设计

6.2.2 单进程Moore型有限状态机

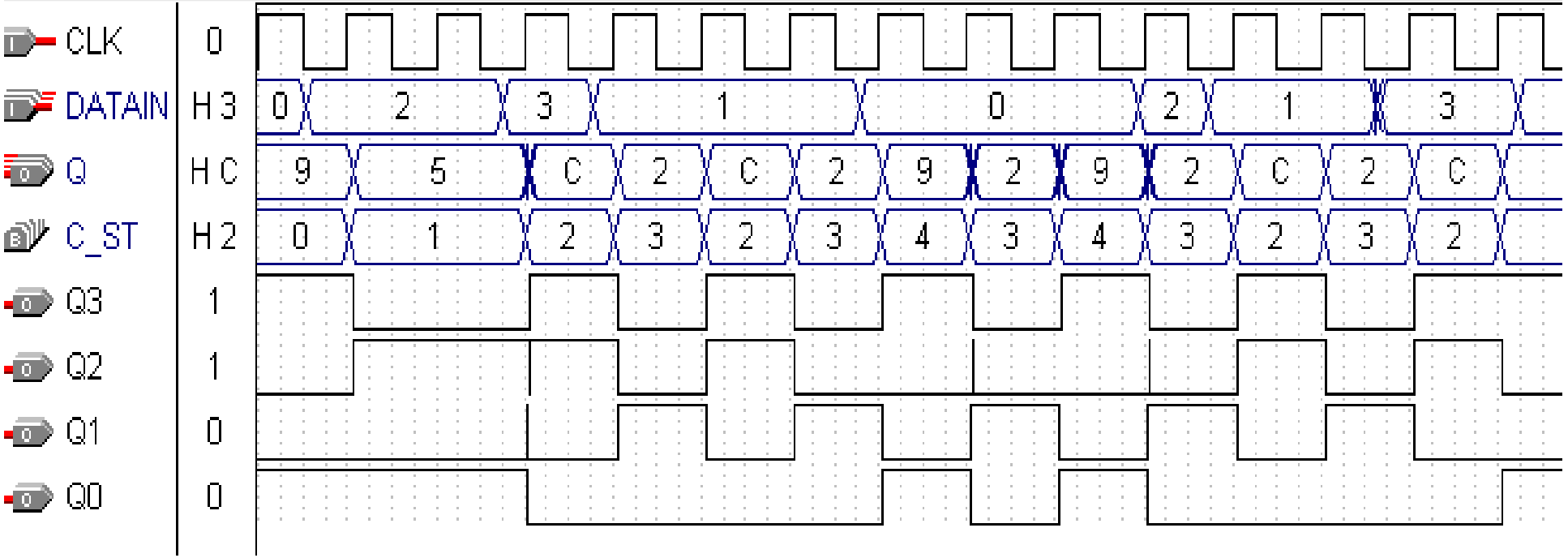

图6-9 对应于例6-4的二进程状态机工作时序图

6.3 Mealy型有限状态机设计

【例6-5】

```
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)

6.3 Mealy型有限状态机设计

```
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;
```

6.3 Mealy型有限状态机设计

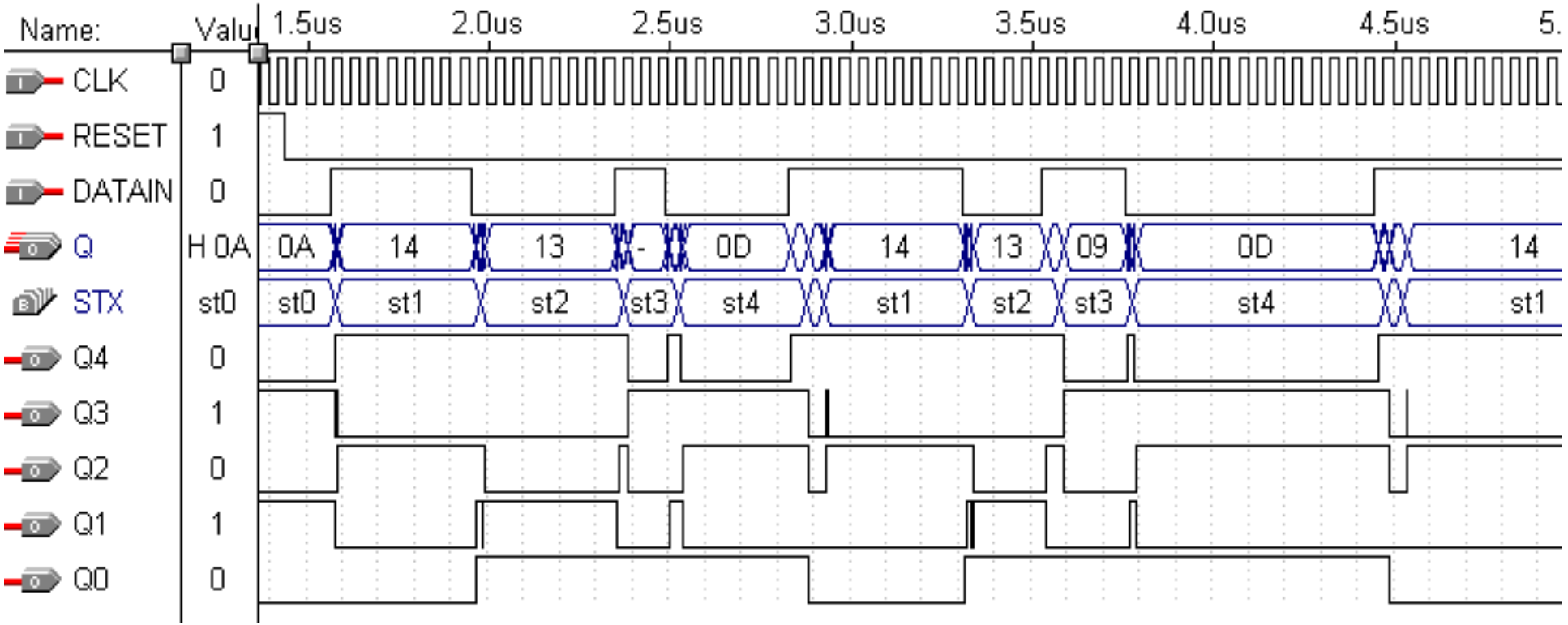

图6-10 例6-5状态机工作时序图

【例6-6】

```
LIBRARY IEEE;  --MEALY FSM
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MEALY2 IS
  PORT ( CLK ,DATAIN,RESET  : IN STD_LOGIC;
       Q : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END MEALY2;
ARCHITECTURE behav OF MEALY2 IS
   TYPE states IS (st0, st1, st2, st3,st4);
   SIGNAL STX : states  ;
   SIGNAL Q1 : STD_LOGIC_VECTOR(4 DOWNTO 0);
 BEGIN
 COMREG : PROCESS(CLK,RESET)   --决定转换状态的进程
   BEGIN
     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;
       END CASE ;
     END IF;
   END PROCESS COMREG ;
COM1: PROCESS(STX,DATAIN,CLK)  --输出控制信号的进程
    VARIABLE Q2 : STD_LOGIC_VECTOR(4 DOWNTO 0);
  BEGIN
  CASE STX IS
  WHEN st0=> IF DATAIN='1' THEN Q2 :="10000"; ELSE Q2:="01010"; END IF;
  WHEN st1=> IF DATAIN='0' THEN Q2 :="10111"; ELSE Q2:="10100"; END IF;
  WHEN st2=> IF DATAIN='1' THEN Q2 :="10101"; ELSE Q2:="10011"; END IF;
  WHEN st3=> IF DATAIN='0' THEN Q2 :="11011"; ELSE Q2:="01001"; END IF;
  WHEN st4=> IF DATAIN='1' THEN Q2 :="11101"; ELSE Q2:="01101"; END IF;
  WHEN OTHERS =>  Q2:="00000" ;
  END CASE ;
  IF CLK'EVENT AND CLK = '1' THEN   Q1<=Q2;  END IF;
   END PROCESS COM1 ;
 Q <= Q1 ;
END behav;
```

6.3 Mealy型有限状态机设计

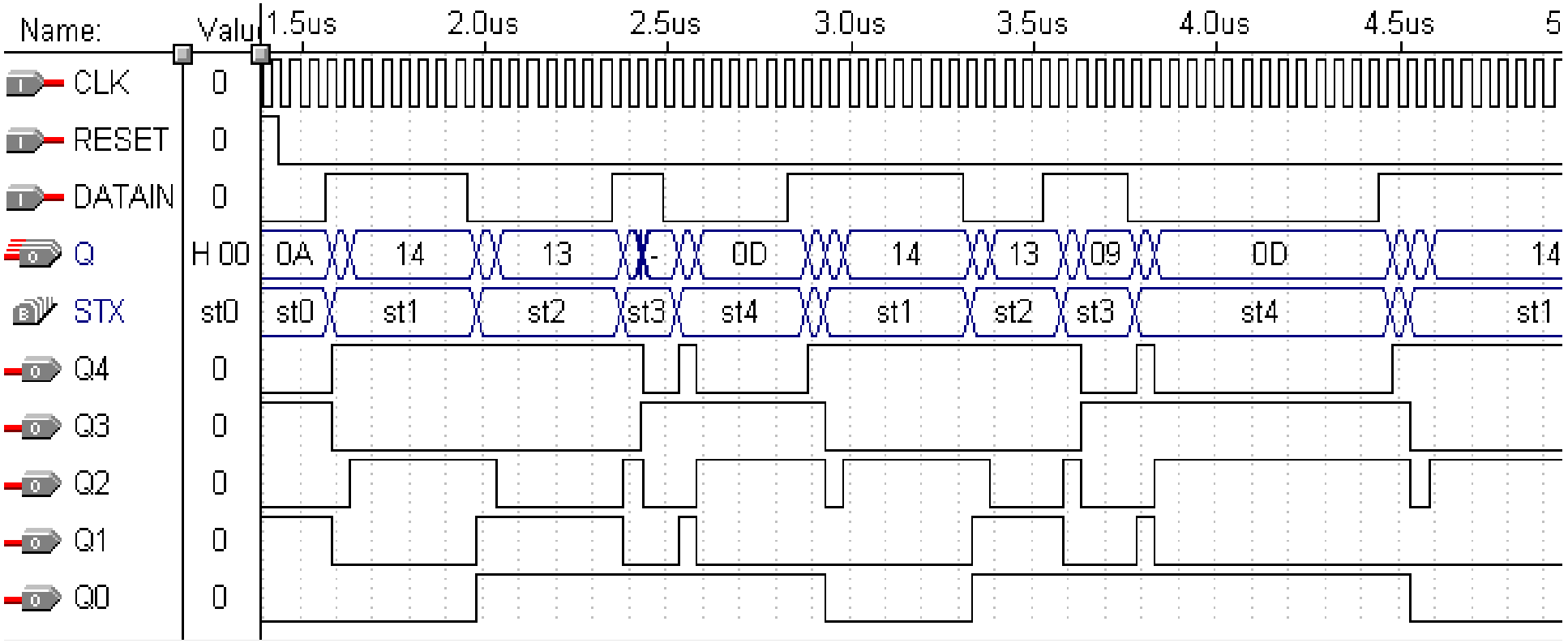

图6-11 例6-6状态机工作时序图

6.4 状态编码

6.4.1 状态位直接输出型编码

START=current_state(4)； ALE=current_state(3)；
OE=current_state(2)； LOCK=current_state(1)；

表6-1 控制信号状态编码表

状态	状态编码					
	START	ALE	OE	LOCK	B	功能说明
ST0	0	0	0	0	0	初始态
ST1	1	1	0	0	0	启动转换
ST2	0	0	0	0	1	若测得EOC=1时，转下一状态ST3
ST3	0	0	1	0	0	输出转换好的数据
ST4	0	0	1	1	0	利用LOCK的上升沿将转换好的数据锁存

【例6-7】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AD0809 IS
...   PORT (D  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       CLK ,EOC : IN STD_LOGIC;
      ALE, START, OE, ADDA   : OUT STD_LOGIC;
     c _state  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
       Q  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END AD0809;
ARCHITECTURE behav OF AD0809 IS
SIGNAL  current _state, next _state: STD_LOGIC_VECTOR(4 DOWNTO 0 );
 CONSTANT st0 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00000" ;
 CONSTANT st1 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "11000" ;
 CONSTANT st2 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00001" ;
 CONSTANT st3 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00100" ;
 CONSTANT st4 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00110" ;
 SIGNAL REGL   : STD_LOGIC_VECTOR(7 DOWNTO 0);
 SIGNAL LOCK   : STD_LOGIC;
BEGIN
ADDA <= '1'; Q <= REGL;  START<=current_state(4); ALE<=current_state(3);
  OE<=current_state(2); LOCK<=current_state(1);c_state <=current _state;
 COM: PROCESS (current _state, EOC)  BEGIN  --规定各状态转换方式
  CASE current _state IS
  WHEN st0=> next _state <= st1; --0809初始化
  WHEN st1=> next _state <= st2; --启动采样
  WHEN st2=> IF (EOC='1') THEN next _state <= st3; --EOC=1表明转换结束
                    ELSE next _state <= st2;          --转换未结束，继续等待
   END IF ;
  WHEN st3=> next _state <= st4;--开启OE,输出转换好的数据
  WHEN st4=> 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;
```

6.4 状态编码

6.4.1 状态位直接输出型编码

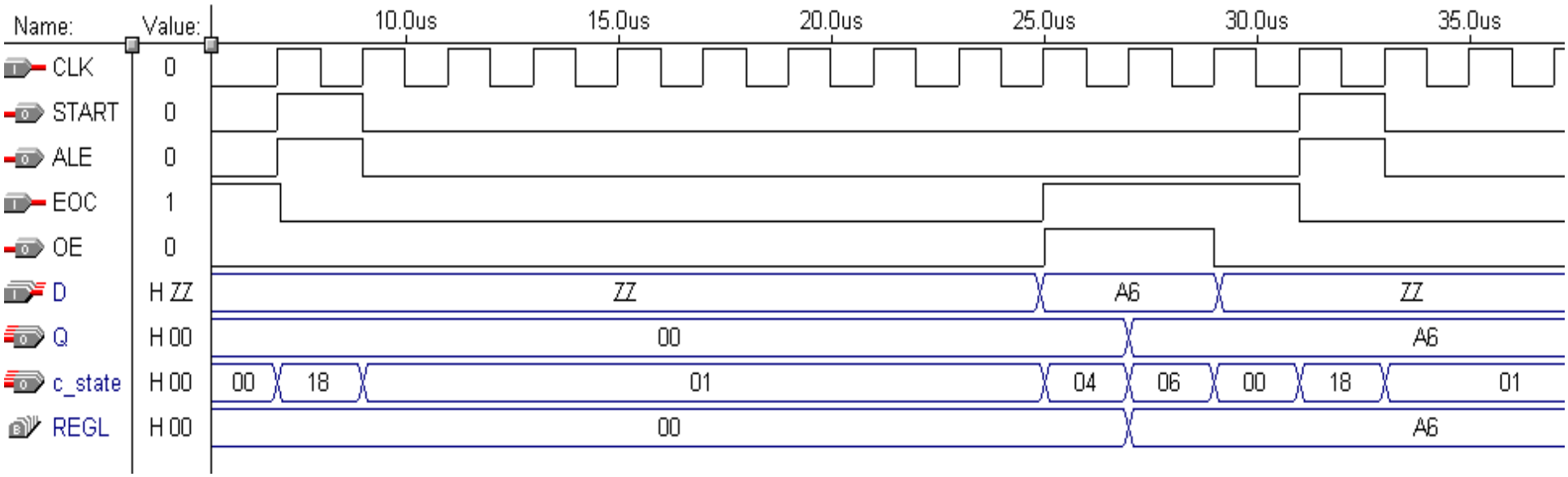

图6-12 例6-7状态机工作时序图

6.4 状态编码

6.4.2 顺序编码

表6-2 编码方式

状 态	顺序编码	一位热码编码
STATE0	000	100000
STATE1	001	010000
STATE2	010	001000
STATE3	011	000100
STATE4	100	000010
STATE5	101	000001

6.4 状态编码

6.4.2 顺序编码

【例6-8】

```
SIGNAL CRURRENT_STATE,NEXT_STATE: STD_LOGIC_VECTOR(2
DOWNTO 0 );
CONSTANT ST0 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000" ;
CONSTANT ST1 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "001" ;
CONSTANT ST2 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "010" ;
CONSTANT ST3 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011" ;
CONSTANT ST4 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "100" ;
```

6.4 状态编码

6.4.3 一位热码编码

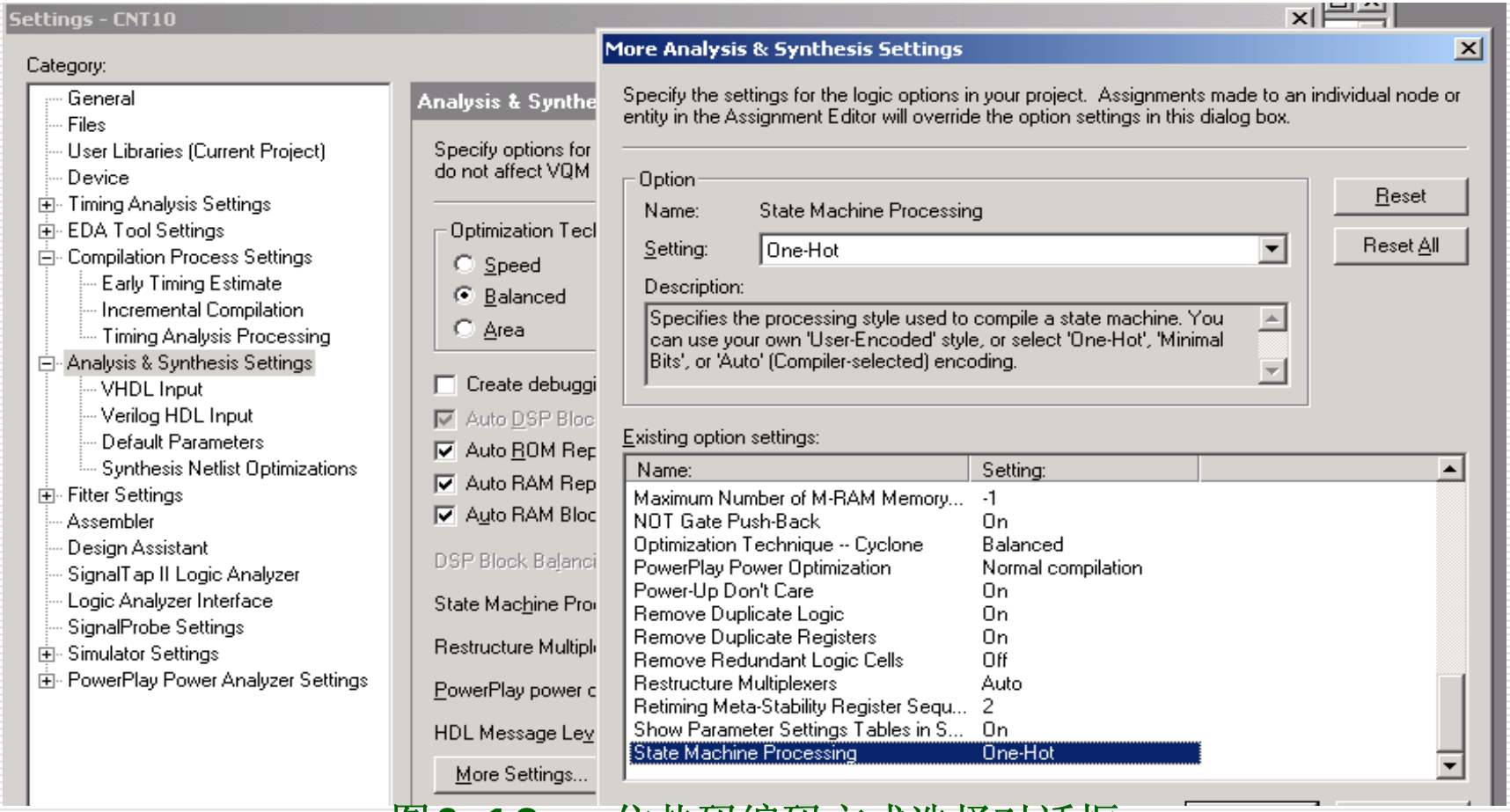

图6-13 一位热码编码方式选择对话框

6.5 非法状态处理

表6-3 剩余状态

状 态	st0	St1	St2	St3	St4	st_ilg1	st_ilg2	st_ilg3
顺序编码	000	001	010	011	100	101	110	111

```
WHEN st_ilg1 =>  next_state <= st0;
WHEN st_ilg2 =>  next_state <= st0;
```

6.5 非法状态处理

【例6-9】

```
TYPE states IS (st0, st1, st2, st3，st4，st_ilg1，st_ilg2 ，st_ilg3);
SIGNAL current_state, next_state: states;
...
COM：PROCESS(current_state, state_Inputs)   -- 组合逻辑进程
BEGIN
  CASE current_state IS                    -- 确定当前状态的状态值
    ...
    WHEN OTHERS =>   next_state <= st0;
    END case;
```

6.5 非法状态处理

【例6-10】

```
alarm <= (st0 AND (st1 OR st2 OR st3 OR st4 OR st5)) OR
        (st1 AND (st0 OR st2 OR st3 OR st4 OR st5)) OR
        (st2 AND (st0 OR st1 OR st3 OR st4 OR st5)) OR
        (st3 AND (st0 OR st1 OR st2 OR st4 OR st5)) OR
        (st4 AND (st0 OR st1 OR st2 OR st3 OR st5)) OR
        (st5 AND (st0 OR st1 OR st2 OR st3 OR st4)) ;
```

习 题

6-1. 仿照例**6-1**，将例**6-4**用两个进程，即一个时序进程，一个组合进程表达出来。

6-2. 为确保例**6-5**的状态机输出信号没有毛刺，试用例**6-4**的方式构成一个单进程状态，使输出信号得到可靠锁存，在相同输入信号条件下，给出两程序的仿真波形。

6-3. 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**6-11**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**”。

【例6-11】

```
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
 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

6-4. 根据图**6-14(a)**所示的状态图，分别按照图**6-14(b)**和图**6-14(c)**写出对应结构的**VHDL**状态机。

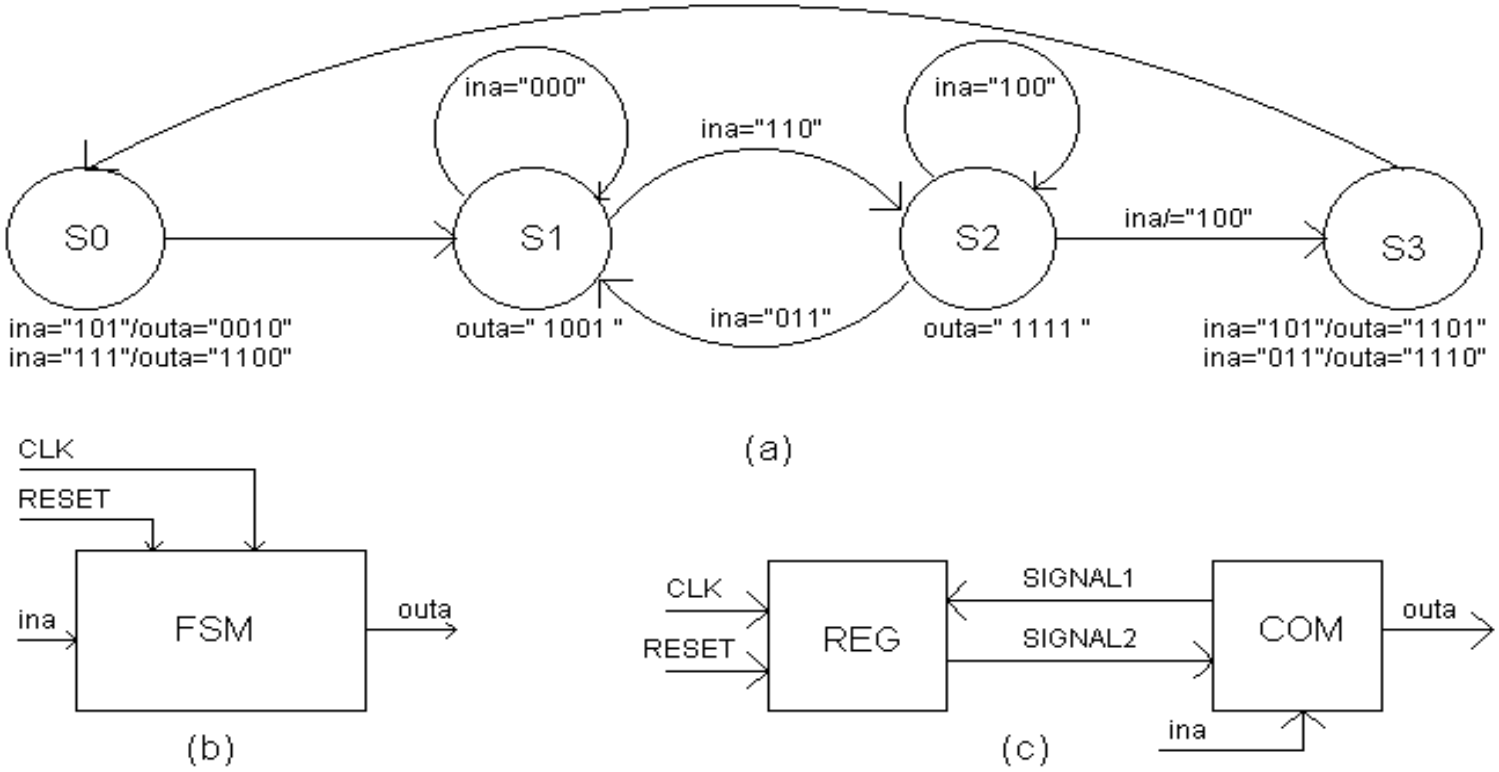

图**6-14** 习题**6-4**状态图

习 题

6-5. 在不改变原代码功能的条件下用两种方法改写例**6-2**，使其输出的控制信号**(ALE、START、OE、LOCK)**没有毛刺。

方法**1**：将输出信号锁存后输出；

方法**2**：使用状态码直接输出型状态机，并比较这三种状态机的特点。

实验与实践

6-1. 序列检测器设计

6-2. ADC0809采样控制电路实现

6-3. 基于0809的数据采集电路和简易存储示波器设计

实验与实践

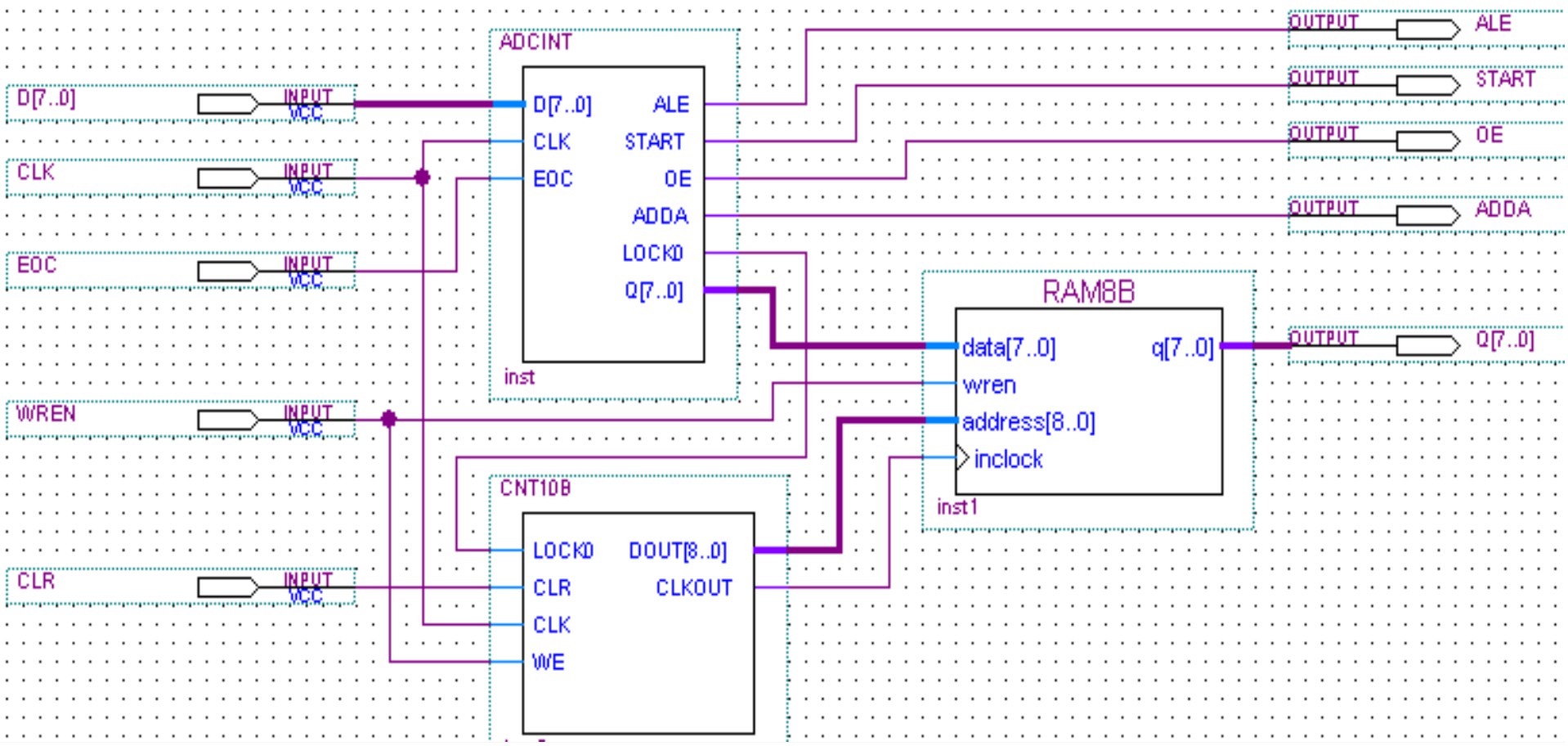

图6-15 ADC0809采样电路系统：RSV.bdf

【例6-12】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CNT10B IS
    PORT (LOCK0,CLR : IN STD_LOGIC;
            CLK : IN STD_LOGIC;
             WE : IN STD_LOGIC;
           DOUT : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
         CLKOUT : OUT STD_LOGIC );
   END CNT10B;
ARCHITECTURE behav OF CNT10B IS
    SIGNAL CQI  :  STD_LOGIC_VECTOR(8 DOWNTO 0);
    SIGNAL CLK0 :  STD_LOGIC;
BEGIN
CLK0 <= LOCK0 WHEN WE='1' ELSE   CLK;
 PROCESS(CLK0,CLR,CQI)
  BEGIN
   IF CLR = '1' THEN  CQI <= "000000000";
   ELSIF CLK0'EVENT AND CLK0 = '1' THEN  CQI <= CQI + 1; END IF;
 END PROCESS;
     DOUT <= CQI; CLKOUT <= CLK0;
END behav;
```

实验与实践

6-4．基于5510/5651的数字存储示波器设计

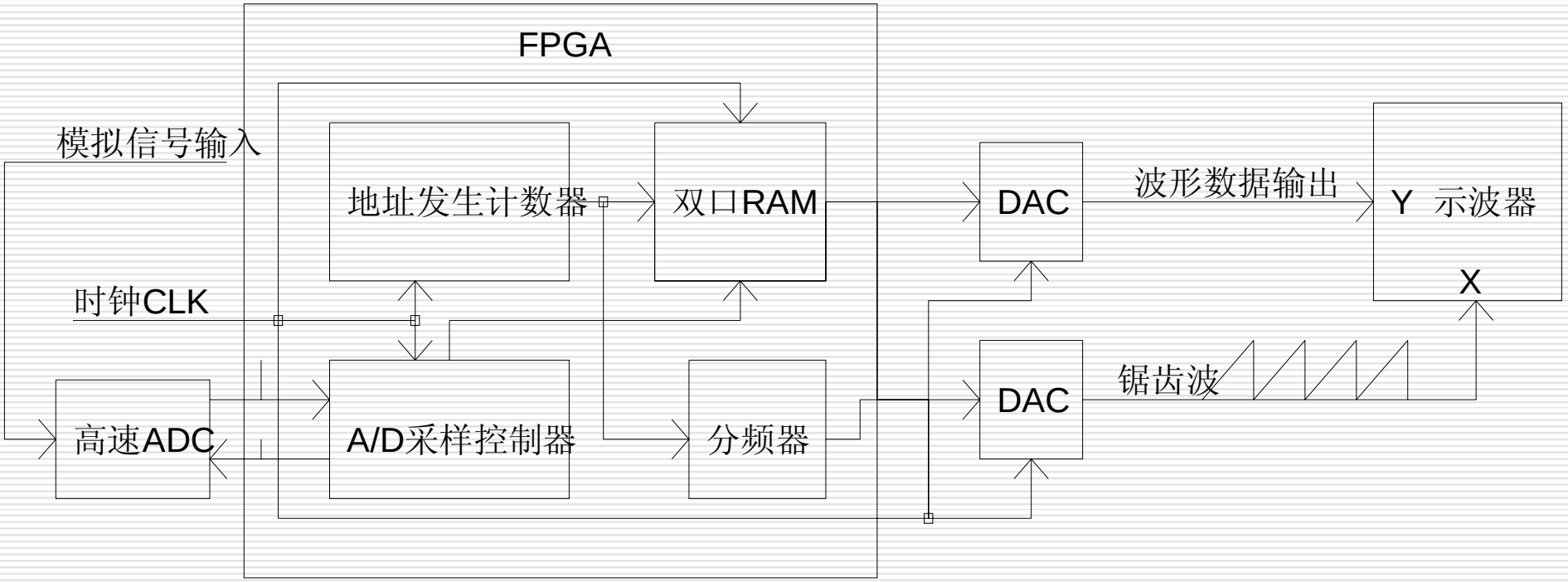

图6-16 存储示波器结构简图

实验与实践

6-4．基于5510/5651的数字存储示波器设计

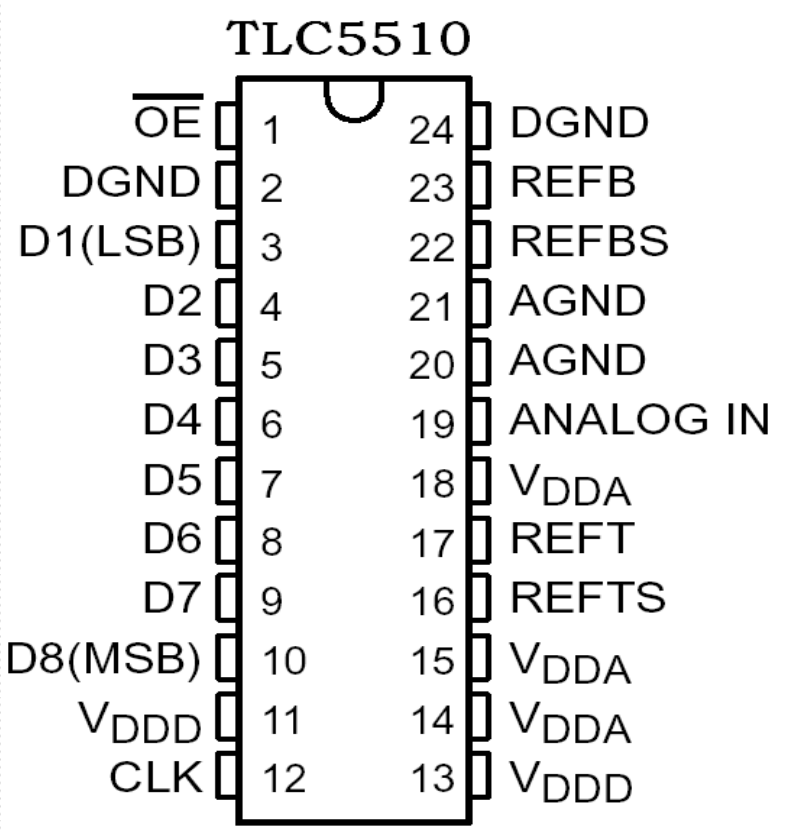

图6-17 TLC5510引脚图

实验与实践

6-4．基于5510/5651的数字存储示波器设计

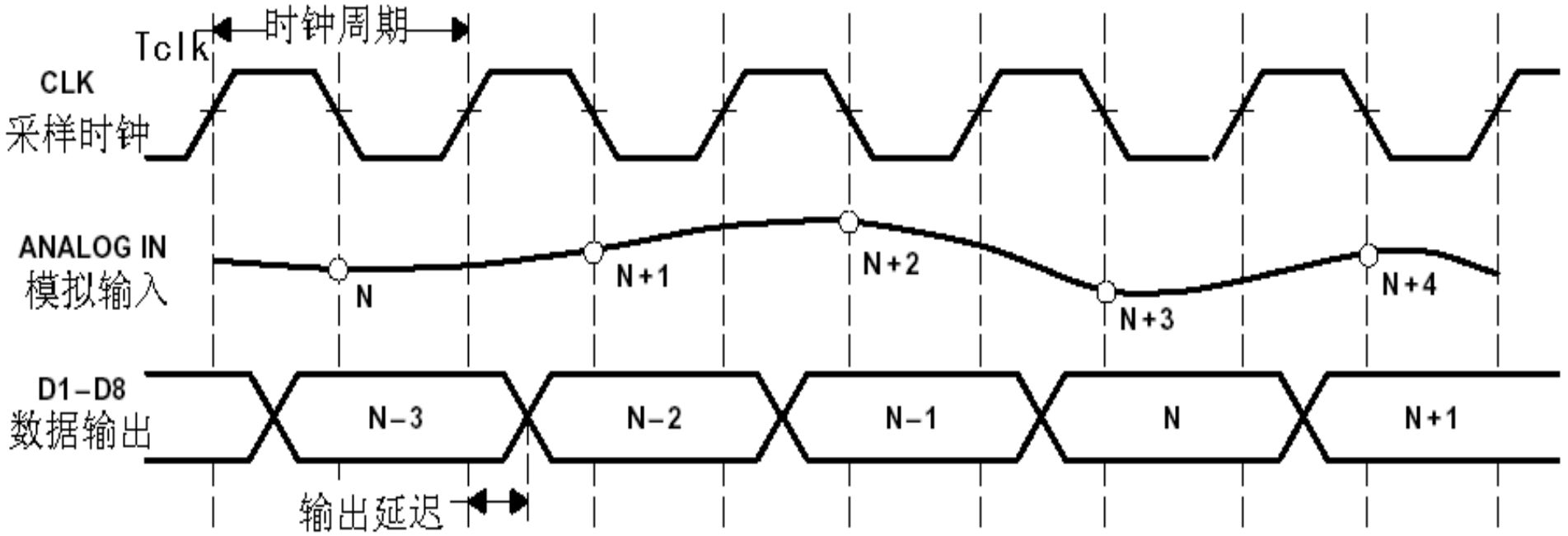

图6-18 TLC5510采样时序图

实验与实践

6-5. VGA彩条信号显示控制器设计

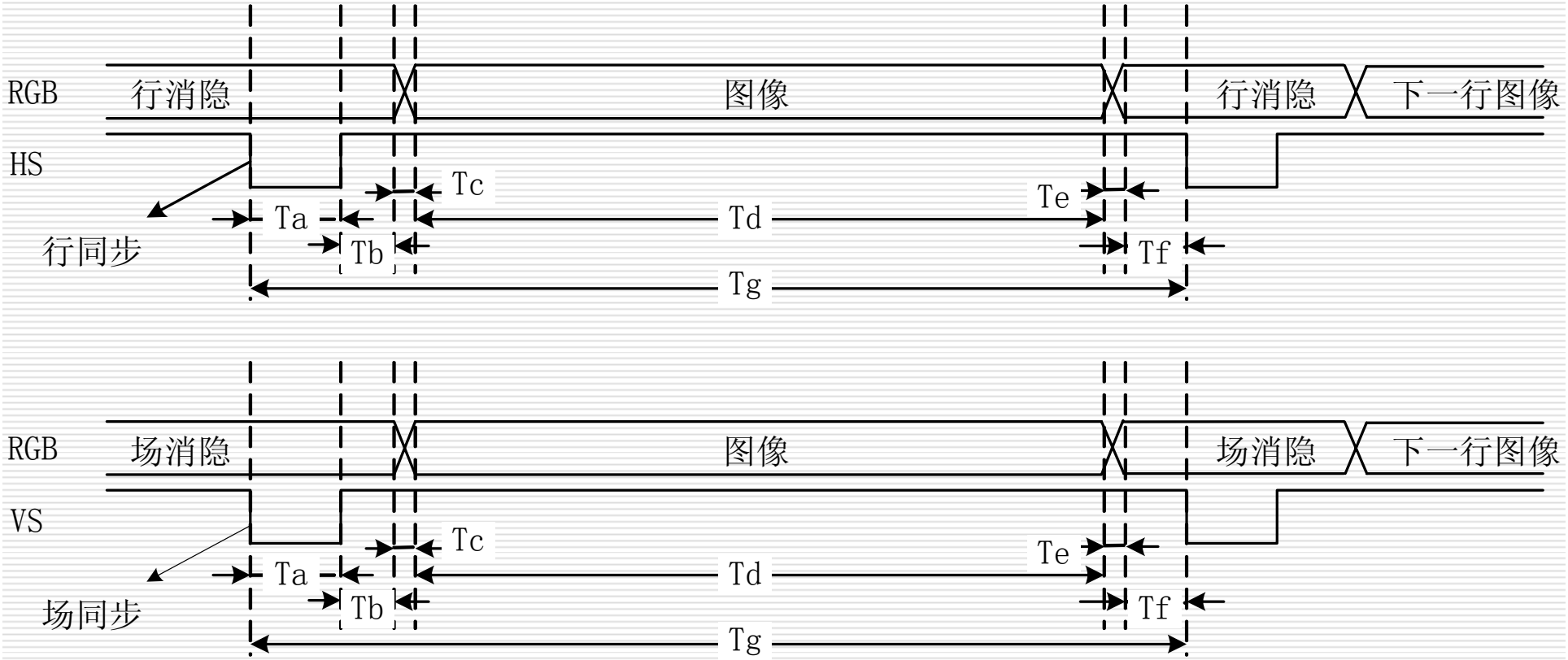

图6-19 VGA行扫描、场扫描时序示意图

实验与实践

6-5. VGA彩条信号显示控制器设计

表6-4 行扫描时序要求：(单位：像素，即输出一个像素Pixel的时间间隔)

		行同步头			行图像		行周期
对应位置	Tf	Ta	Tb	Tc	Td	Te	Tg
时间(Pixels)	8	96	40	8	640	8	800

实验与实践

6-5. VGA彩条信号显示控制器设计

表6-5 场扫描时序要求：(单元：行，即输出一行Line的时间间隔)

		行同步头			行图像	行周期	
对应位置	Tf	Ta	Tb	Tc	Td	Te	Tg
时间(Lines)	2	2	25	8	480	8	525

实验与实践

6-5. VGA彩条信号显示控制器设计

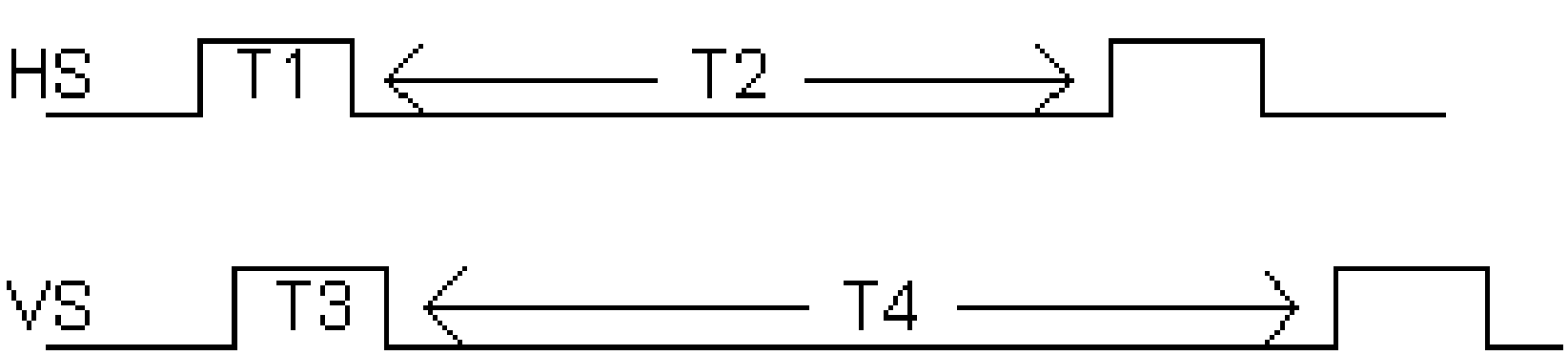

图6-20 HS和VS的时序图

实验与实践

6-5. VGA彩条信号显示控制器设计

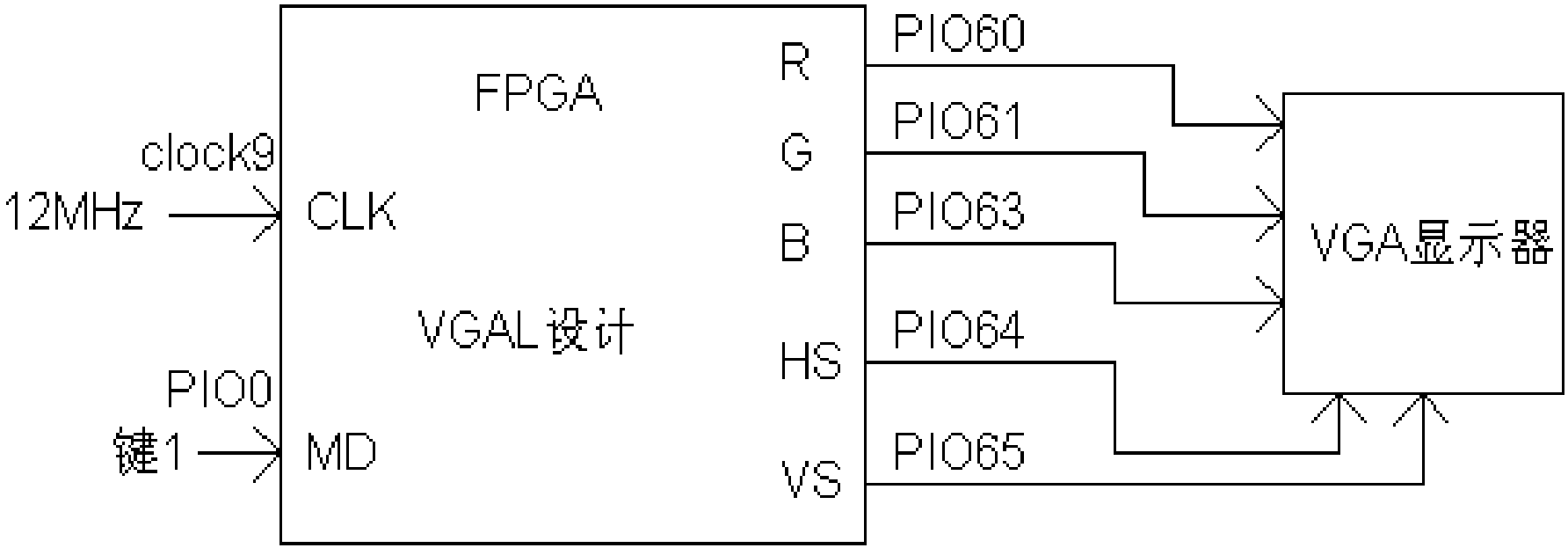

图6-20 HS和VS的时序图

实验与实践

6-5. VGA彩条信号显示控制器设计

表6-6 颜色编码

颜色	黑	蓝	红	品	绿	青	黄	白
R	0	0	0	0	1	1	1	1
G	0	0	1	1	0	0	1	1
B	0	1	0	1	0	1	0	1

【例6-13】

```
LIBRARY IEEE;   -- VGA显示器 彩条 发生器
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY COLOR IS
   PORT (     CLK, MD : IN STD_LOGIC;
      HS, VS, R, G, B : OUT STD_LOGIC  ); -- 行场同步/红，绿，兰
END COLOR;
ARCHITECTURE behav OF COLOR IS
   SIGNAL HS1,VS1,FCLK,CCLK     : STD_LOGIC;
   SIGNAL MMD : STD_LOGIC_VECTOR(1 DOWNTO 0);-- 方式选择
   SIGNAL FS : STD_LOGIC_VECTOR (3 DOWNTO 0);
   SIGNAL CC : STD_LOGIC_VECTOR(4 DOWNTO 0); --行同步/横彩条生成
   SIGNAL LL : STD_LOGIC_VECTOR(8 DOWNTO 0); --场同步/竖彩条生成

   SIGNAL GRBX : STD_LOGIC_VECTOR(3 DOWNTO 1);-- X横彩条
   SIGNAL GRBY : STD_LOGIC_VECTOR(3 DOWNTO 1);-- Y竖彩条
   SIGNAL GRBP : STD_LOGIC_VECTOR(3 DOWNTO 1);
   SIGNAL GRB  : STD_LOGIC_VECTOR(3 DOWNTO 1);
BEGIN
   GRB(2) <= (GRBP(2) XOR MD) AND HS1 AND VS1;
   GRB(3) <= (GRBP(3) XOR MD) AND HS1 AND VS1;
   GRB(1) <= (GRBP(1) XOR MD) AND HS1 AND VS1;
   PROCESS( MD )
```

(接下页)

```
BEGIN
    IF MD'EVENT AND MD = '0' THEN
      IF MMD = "10" THEN  MMD <= "00";
       ELSE   MMD <= MMD + 1; END IF;      --三种模式
    END IF;
  END PROCESS;
  PROCESS( MMD )
  BEGIN
    IF MMD = "00" THEN    GRBP <= GRBX;     -- 选择横彩条
    ELSIF MMD = "01" THEN  GRBP <= GRBY;    -- 选择竖彩条
    ELSIF MMD = "10" THEN  GRBP <= GRBX XOR GRBY; --产生棋盘格
     ELSE  GRBP <= "000";  END IF;
  END PROCESS;
  PROCESS( CLK )
  BEGIN
    IF CLK'EVENT AND CLK = '1' THEN -- 13MHz 13分频
       IF FS = 13 THEN FS <= "0000";
       ELSE FS <= (FS + 1); END IF;
    END IF;
  END PROCESS;
  FCLK <= FS(3); CCLK <= CC(4);
  PROCESS( FCLK )
  BEGIN
```

(接下页)

```
    IF FCLK'EVENT AND FCLK = '1' THEN
      IF CC = 29 THEN  CC <= "00000";
      ELSE  CC <= CC + 1; END IF;
   END IF;
END PROCESS;
PROCESS( CCLK )
BEGIN
   IF CCLK'EVENT AND CCLK = '0' THEN
      IF LL = 481 THEN  LL <= "000000000";
      ELSE  LL <= LL + 1; END IF;
   END IF;
END PROCESS;
PROCESS( CC,LL )
BEGIN
   IF CC > 23 THEN  HS1 <= '0';  --行同步
   ELSE HS1 <= '1'; END IF;
   IF LL > 479 THEN  VS1 <= '0'; --场同步
   ELSE  VS1 <= '1'; END IF;
END PROCESS;
PROCESS(CC, LL)
BEGIN
```

(接下页)

```
      IF CC < 3  THEN GRBX <= "111";  -- 横彩条
     ELSIF CC < 6  THEN GRBX <= "110";
     ELSIF CC < 9  THEN GRBX <= "101";
     ELSIF CC < 13 THEN GRBX <= "100";
     ELSIF CC < 15 THEN GRBX <= "011";
     ELSIF CC < 18 THEN GRBX <= "010";
     ELSIF CC < 21 THEN GRBX <= "001";
      ELSE GRBX <= "000";
        END IF;
        IF LL <  60 THEN GRBY <= "111";  -- 竖彩条
     ELSIF LL < 130 THEN GRBY <= "110";
     ELSIF LL < 180 THEN GRBY <= "101";
     ELSIF LL < 240 THEN GRBY <= "100";
     ELSIF LL < 300 THEN GRBY <= "011";
     ELSIF LL < 360 THEN GRBY <= "010";
     ELSIF LL < 420 THEN GRBY <= "001";
     ELSE GRBY <= "000";
        END IF;
   END PROCESS;
   HS <= HS1 ; VS <= VS1 ;R <= GRB(2) ;G <= GRB(3) ; B <=
GRB(1);
END behav;
```

实验与实践

6-5. VGA彩条信号显示控制器设计

表6-7 彩条信号发生器3种显示模式

1	横彩条	1：白黄青绿品红蓝黑	2：黑蓝红品绿青黄白
2	竖彩条	1：白黄青绿品红蓝黑	2：黑蓝红品绿青黄白
3	棋盘格	1：棋盘格显示模式1	2：棋盘格显示模式2