EDA 技术实用教程

第 8 章

状态机设计

8.1 一般有限状态机设计

8.1.1 数据类型定义语句

TYPE语句的用法如下：

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);
```

8.1 一般有限状态机设计

8.1.1 数据类型定义语句

```
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' ;
```

```
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 ;
```

8.1 一般有限状态机设计

8.1.2 为什么要使用状态机

- 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
- 状态机可以定义符号化枚举类型的状态
- 状态机容易构成性能良好的同步时序逻辑模块
- 状态机的VHDL表述丰富多样、程序层次分明，易读易懂
- 在高速运算和控制方面，状态机更有其巨大的优势
- 高可靠性

8.1 一般有限状态机设计

8.1.3 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;
 …
```

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

2. 主控时序进程

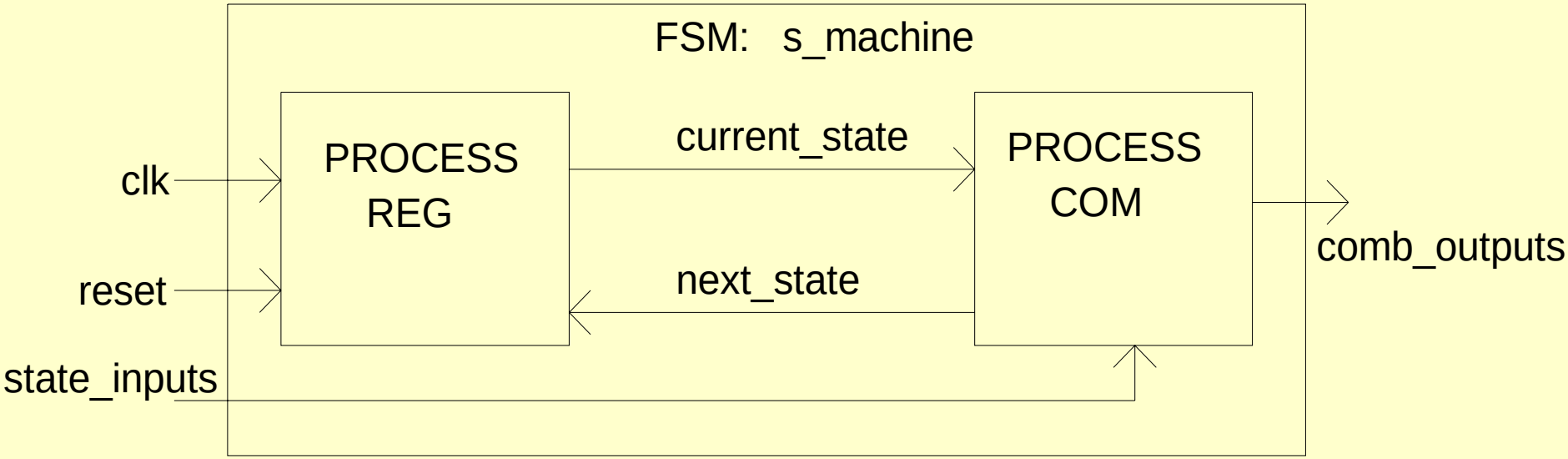


图8-1 一般状态机结构框图

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

3. 主控组合进程

4. 辅助进程

【例8-1】

```
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
```

（接下页）

```
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;
```

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

4. 辅助进程

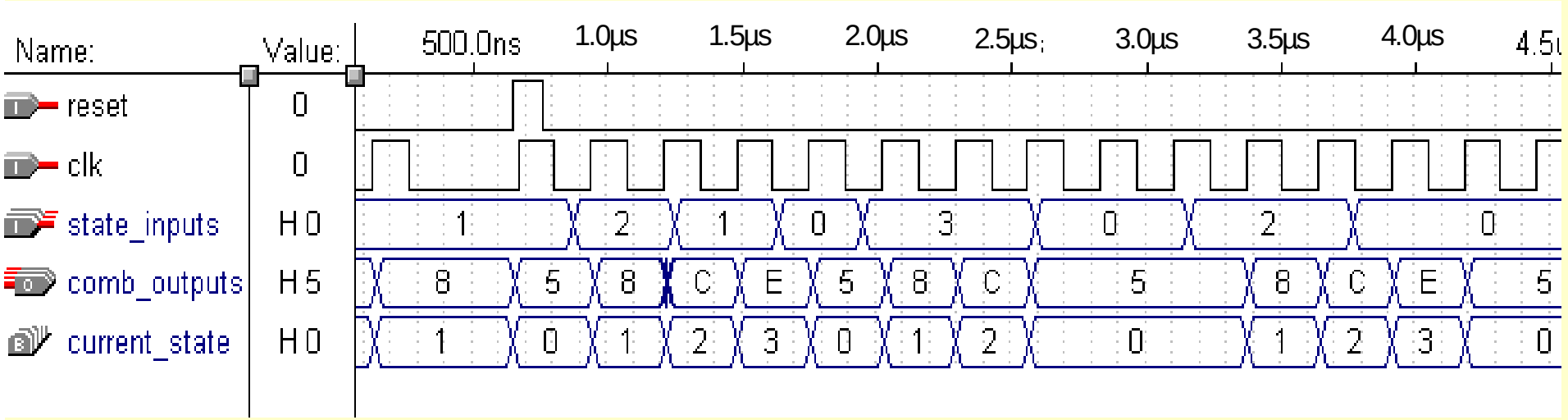


图8-2a 例8-1状态机的工作时序

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

4. 辅助进程

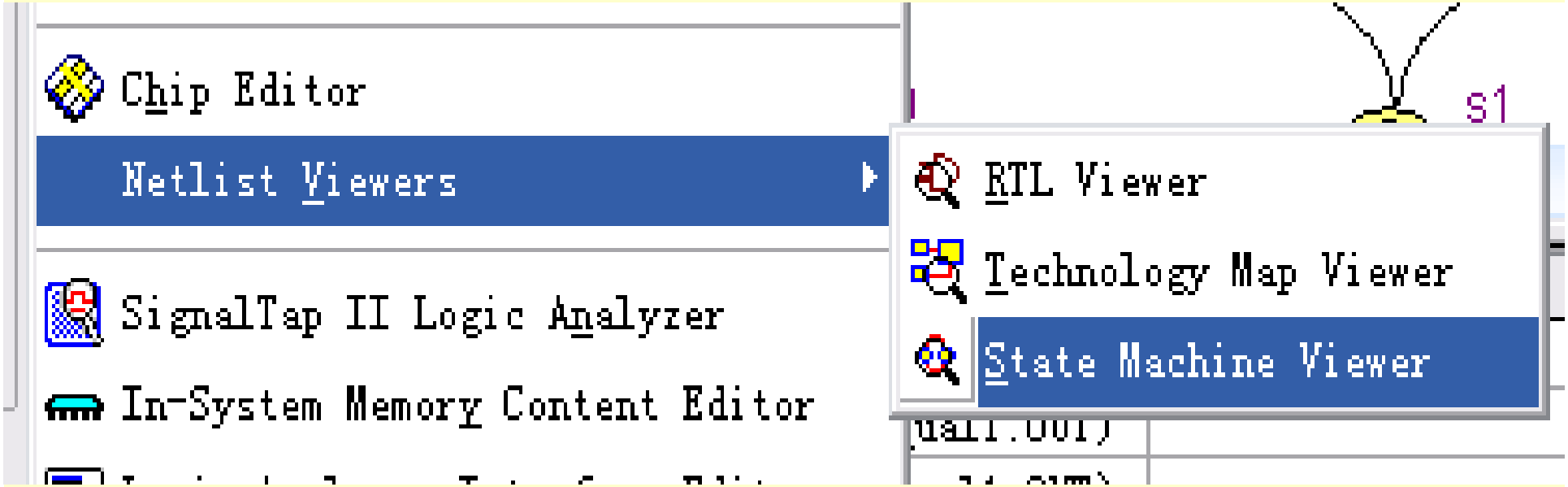


图8-2b 打开QuartusII状态图观察器

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

4. 辅助进程

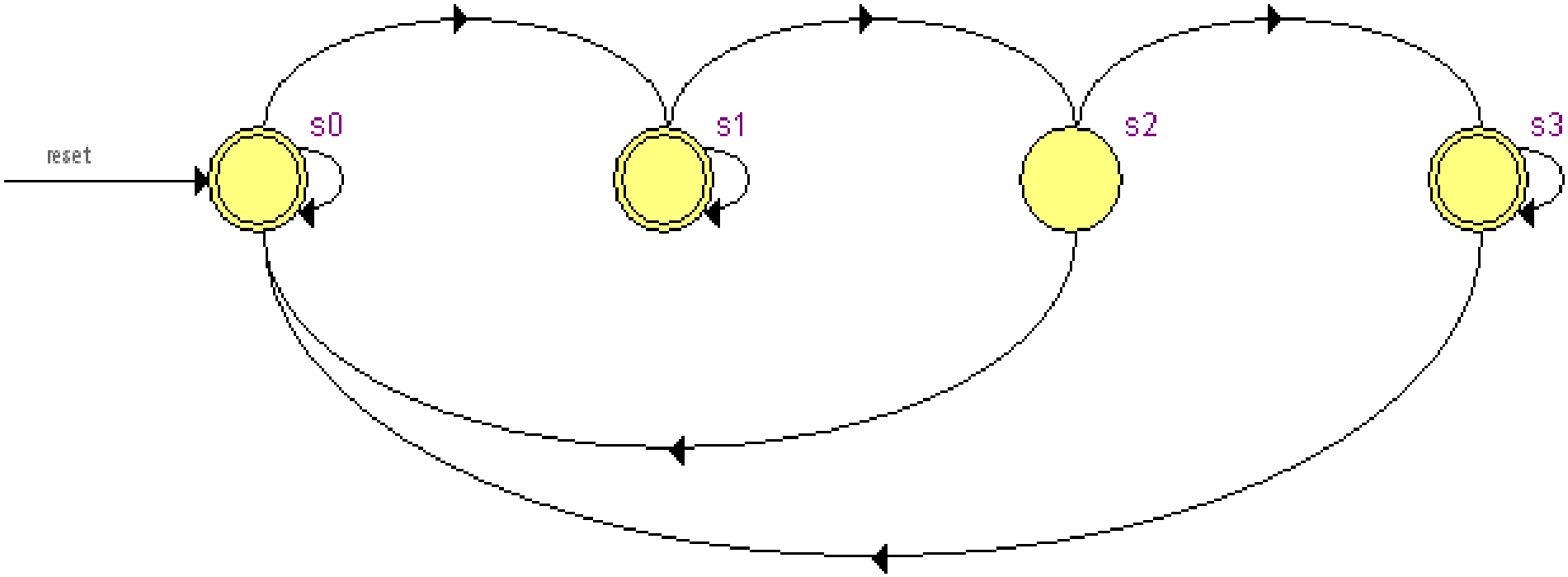


图8-2c 例8-1的状态图

8.2 Moore型有限状态机设

8.2.1 多进程有限状态机

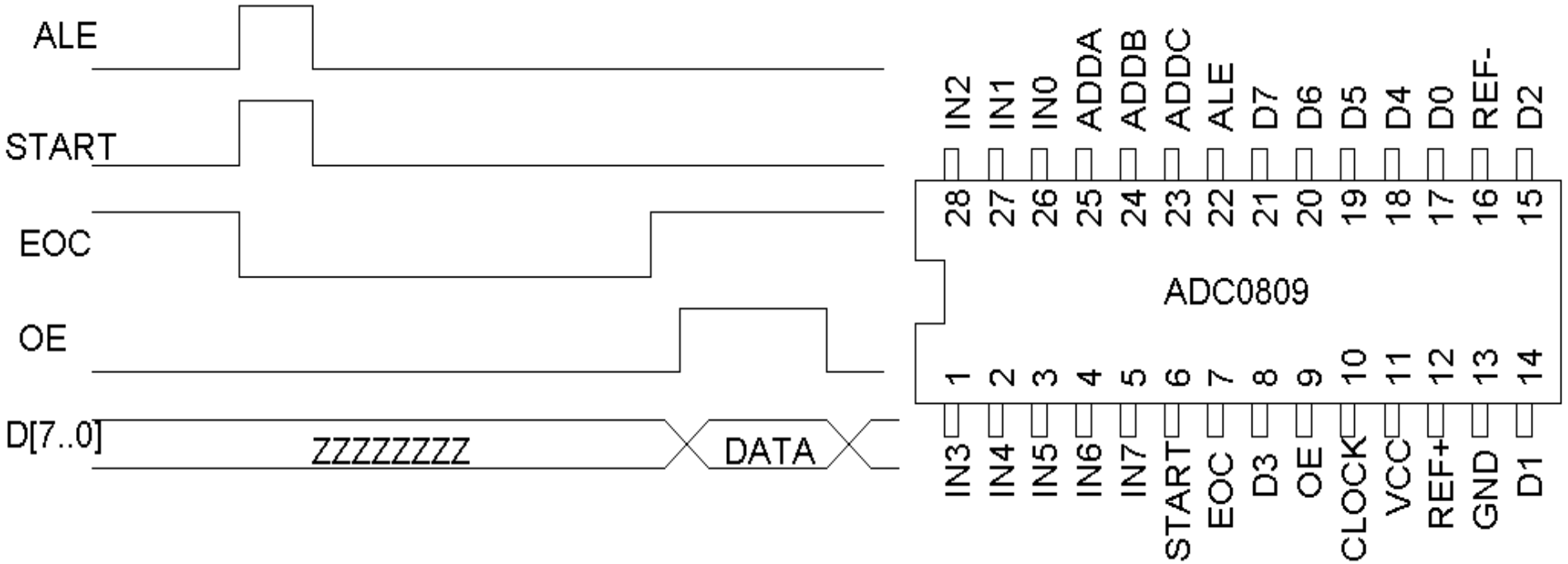


图8-3 ADC0809工作时序

8.2 Moore型有限状态机设

8.2.1 多进程有限状态机

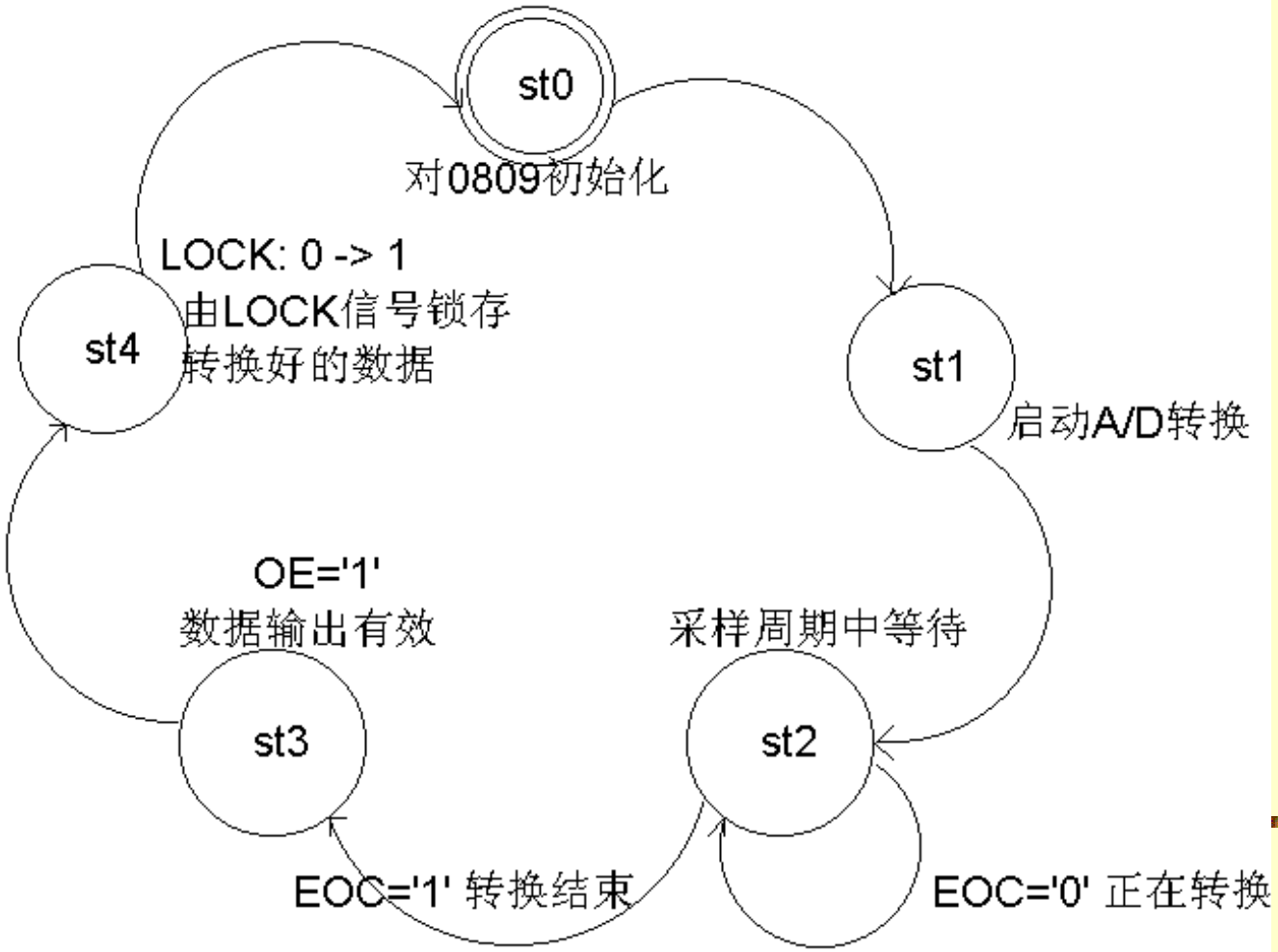


图8-4 控制ADC0809采样状态图

8.2.1 多进程有限状态机

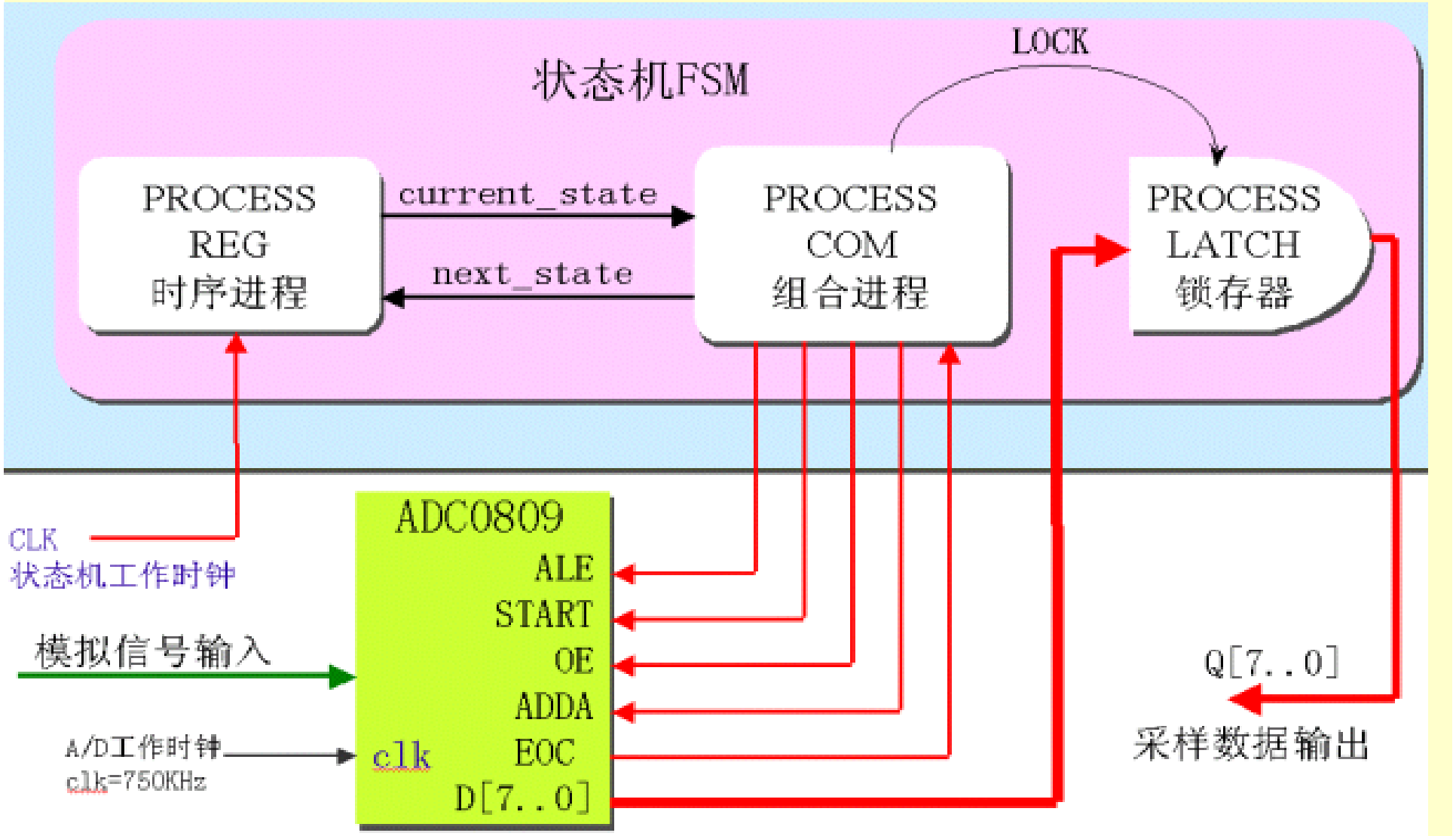


图8-5 采样状态机结构框图

【例8-2】

```
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
```

（接下页）

```
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8-3】

```
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 ;
```

8.2 Moore型有限状态机设

8.2.1 多进程有限状态机

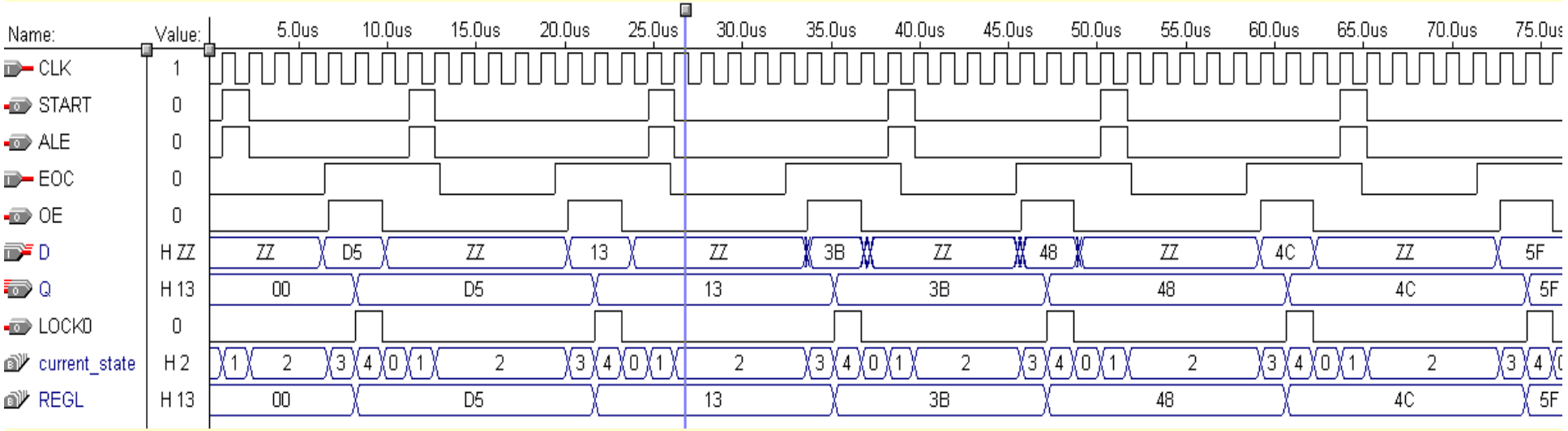


图8-6 ADC0809采样状态机工作时序

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

【例8-4】

```
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
```

（接下页）

```
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;
```

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

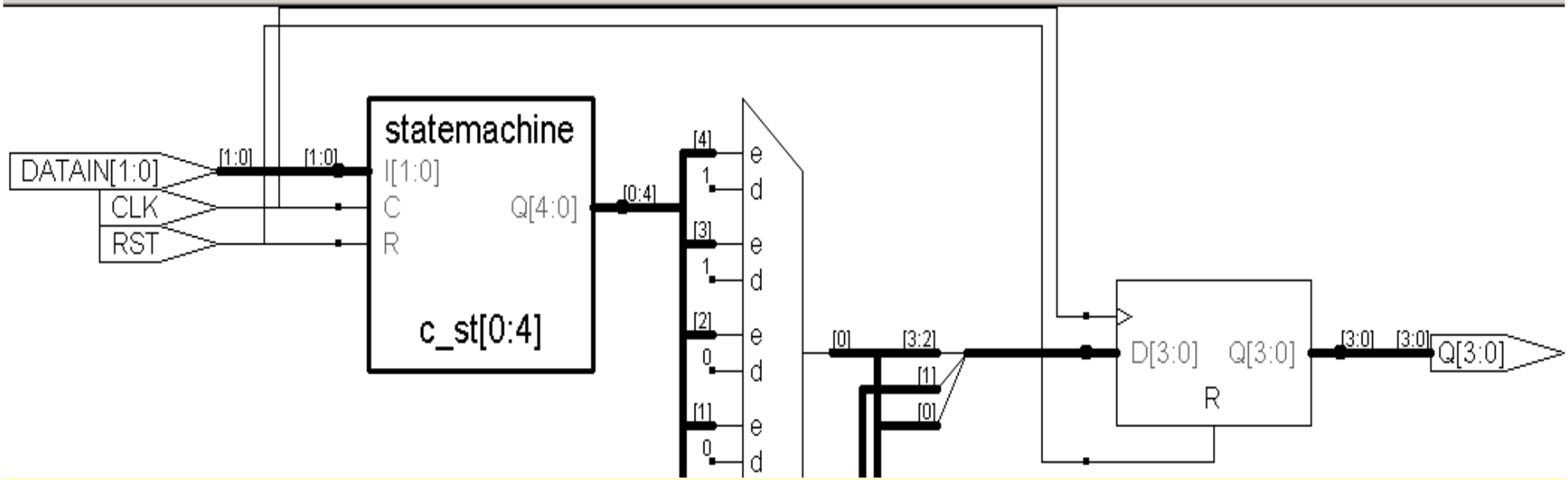


图8-7 例8-4状态机综合后的部分主要RTL电路模块（Synplify综合）

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

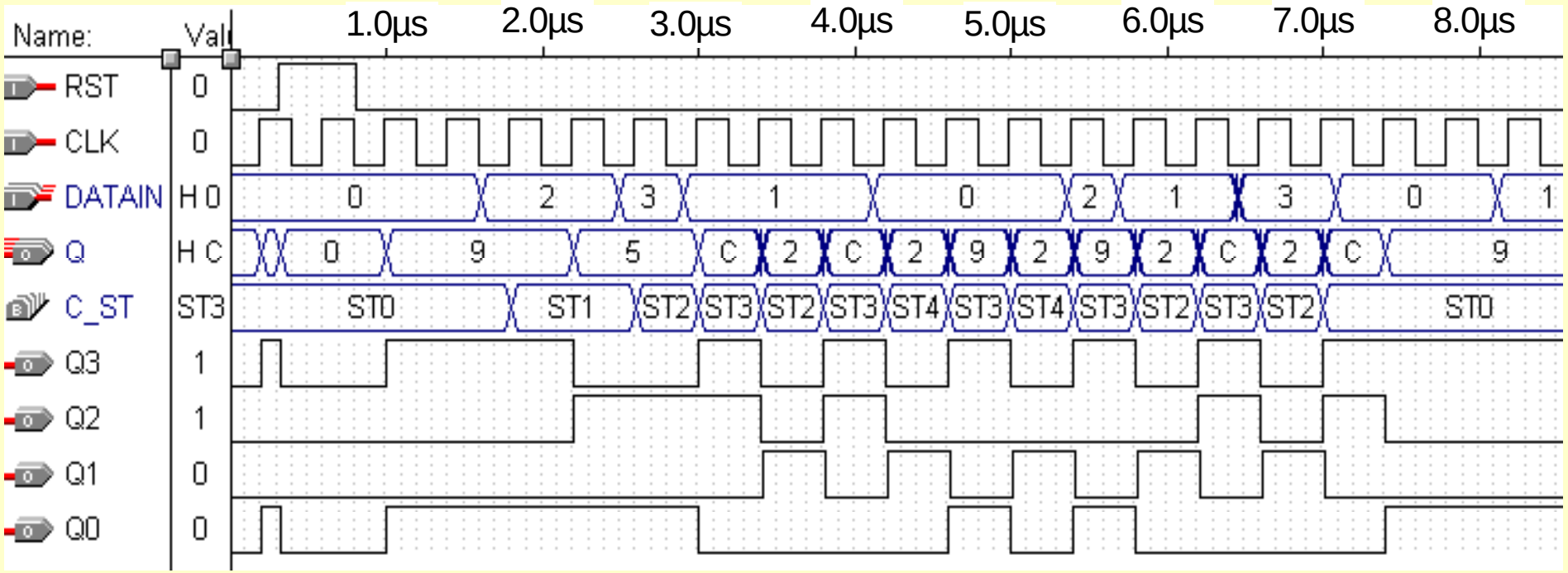


图8-8 例8-4单进程状态机工作时序

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

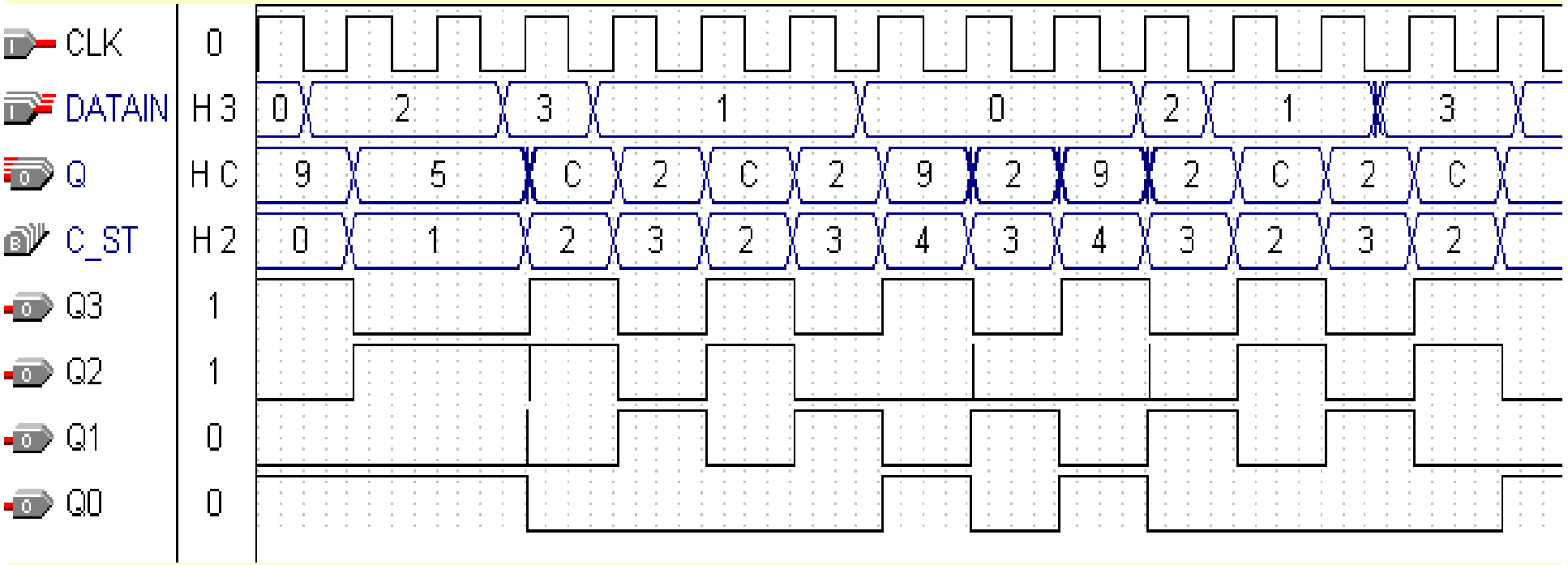


图8-9 对应于例8-4的二进程状态机工作时序图

8.3 Mealy型有限状态机设计

【例8-5】

```
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;
```

(接下页)

```
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) 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;
```

8.3 Mealy型有限状态机设计

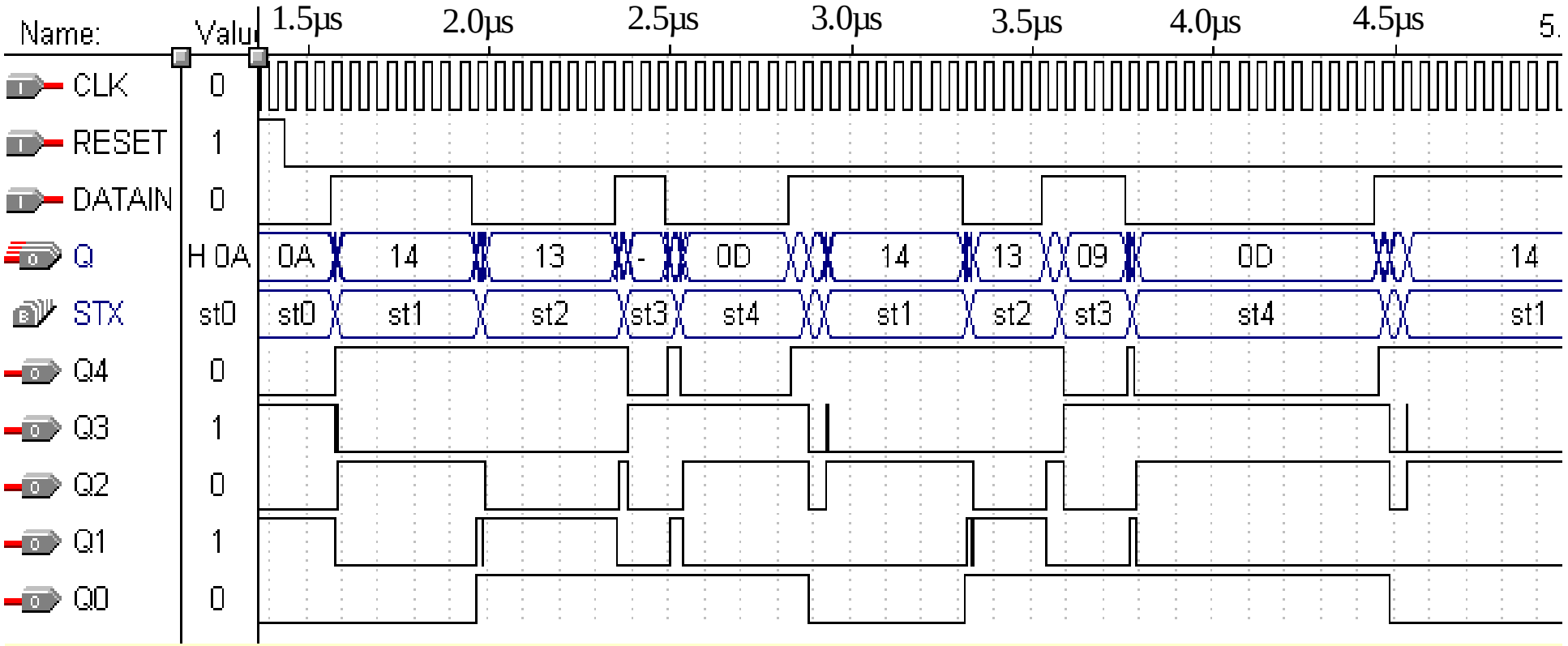


图8-10 例8-5状态机工作时序图

8.3 Mealy型有限状态机设计

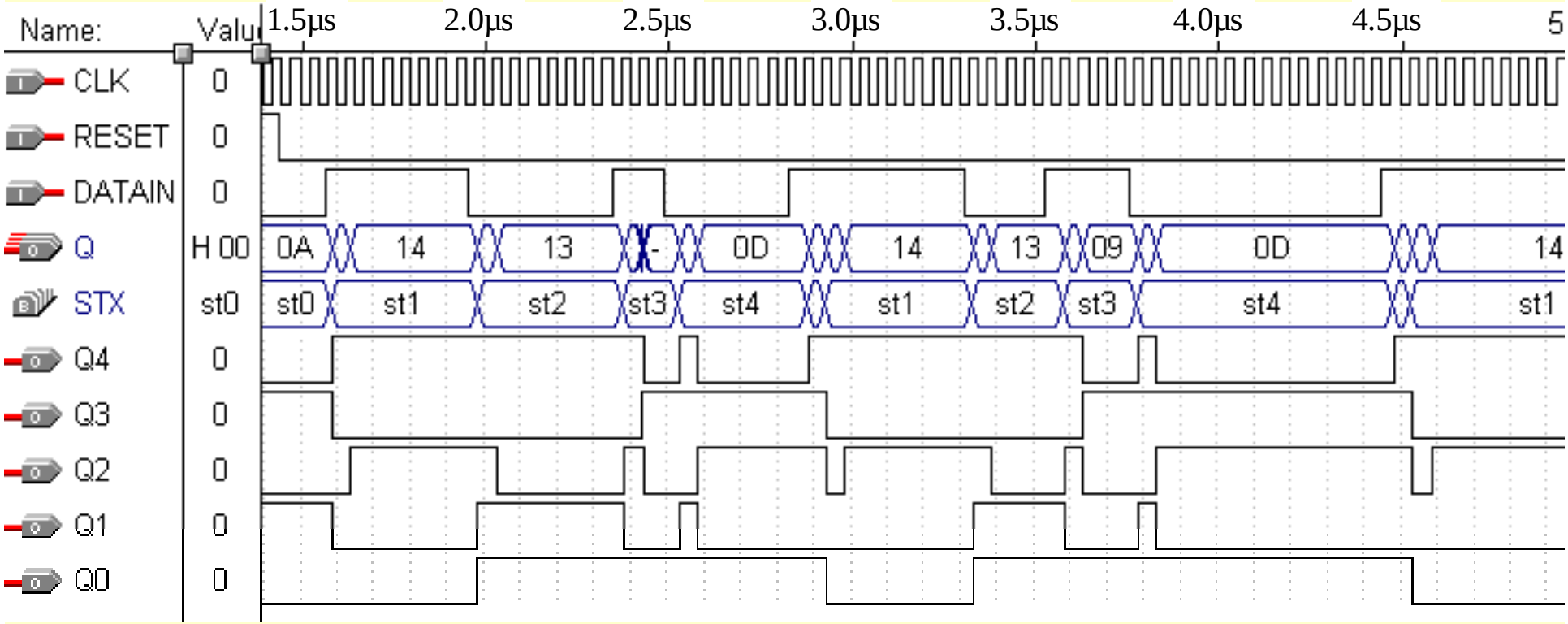


图8-11 例8-6状态机工作时序图

8.3 Mealy型有限状态机设计

【例8-6】

```
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
```

（接下页）

```
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;
```

8.4 状态编码

8.4.1 状态位直接输出型编码

表8-1 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

8.4 状态编码

【例8-7】

```
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 REGL   : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL LOCK   : STD_LOGIC;
```

（接下页）

```
   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;
```

8.4 状态编码

8.4.1 状态位直接输出型编码

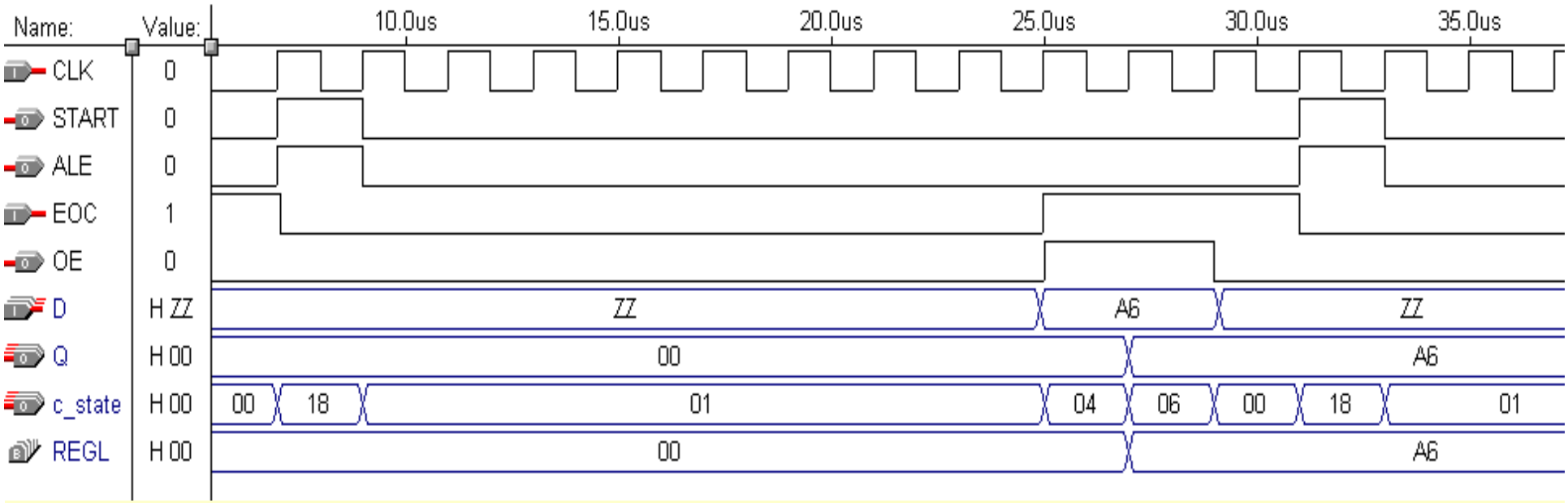


图8-12 例8-7状态机工作时序图

8.4 状态编码

8.4.2 顺序编码

表8-2 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**

8.4 状态编码

8.4.2 顺序编码

【例8-8】

```
...
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" ;
...
```

8.4.3 一位热码编码

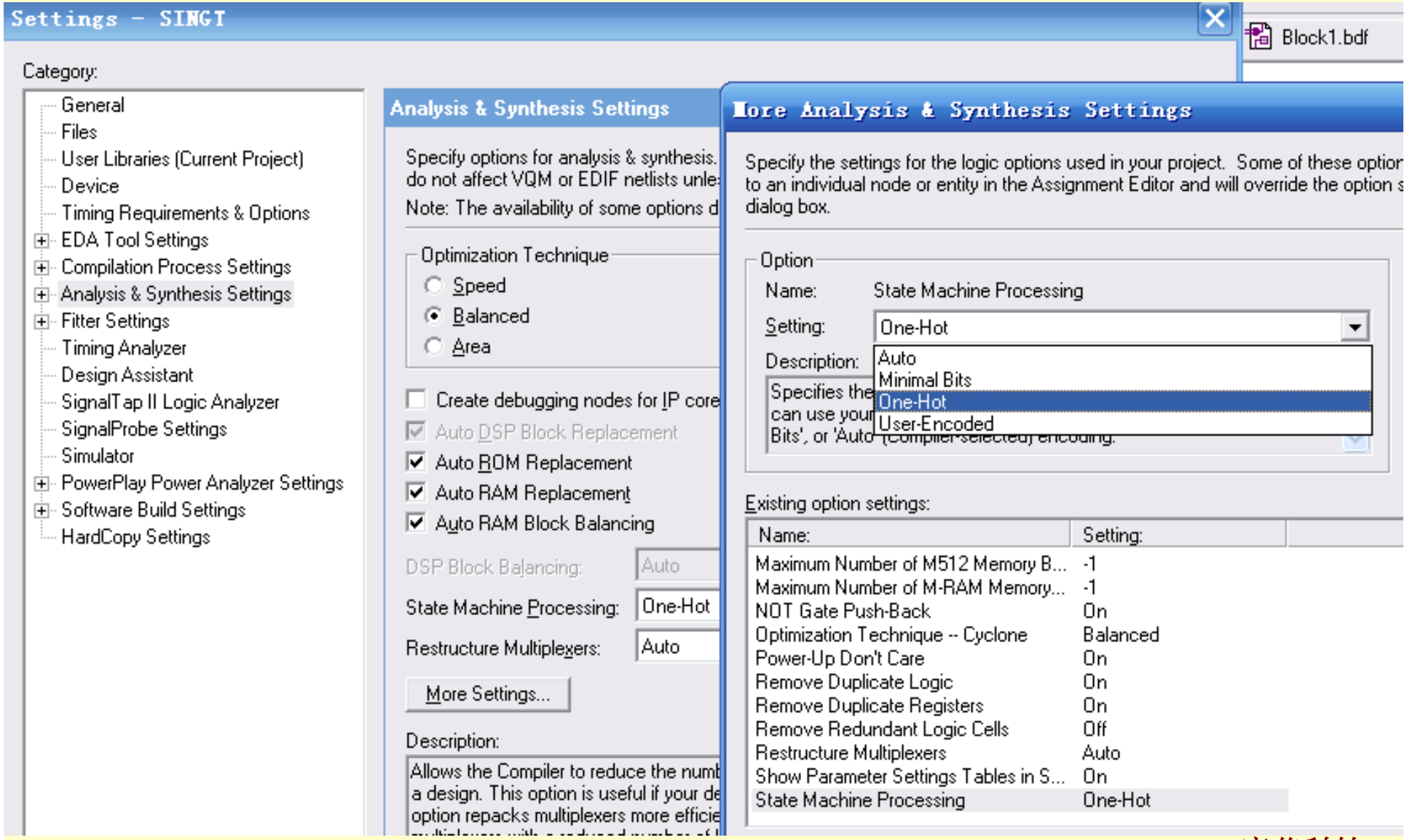


图8-13 一位热码编码方式选择对话框

8.5 非法状态处理

表8-3 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

(1) 在语句中对每一个非法状态都作出明确的状态转换指示，如在原来的CASE语句中增加诸如以下语句：

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

(2) 如例8-9那样，利用OTHERS语句中对未提到的状态作统一处理。

8.5 非法状态处理

【例8-9】

```
  ...
  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;
```

8.5 非法状态处理

【例8-10】

```
...
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

8-1. 仿照例**8-1**，将例**8-4**用两个进程，即一个时序进程，一个组合进程表达出来。

8-2. 为确保例**8-5**的状态机输出信号没有毛刺，试用例**8-4**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。

8-3. 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**8-11**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**”。

习 题

【例8-11】

```
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 ;
```

（接下页）

习 题

```
    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8-11的代码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：根据例8-11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。

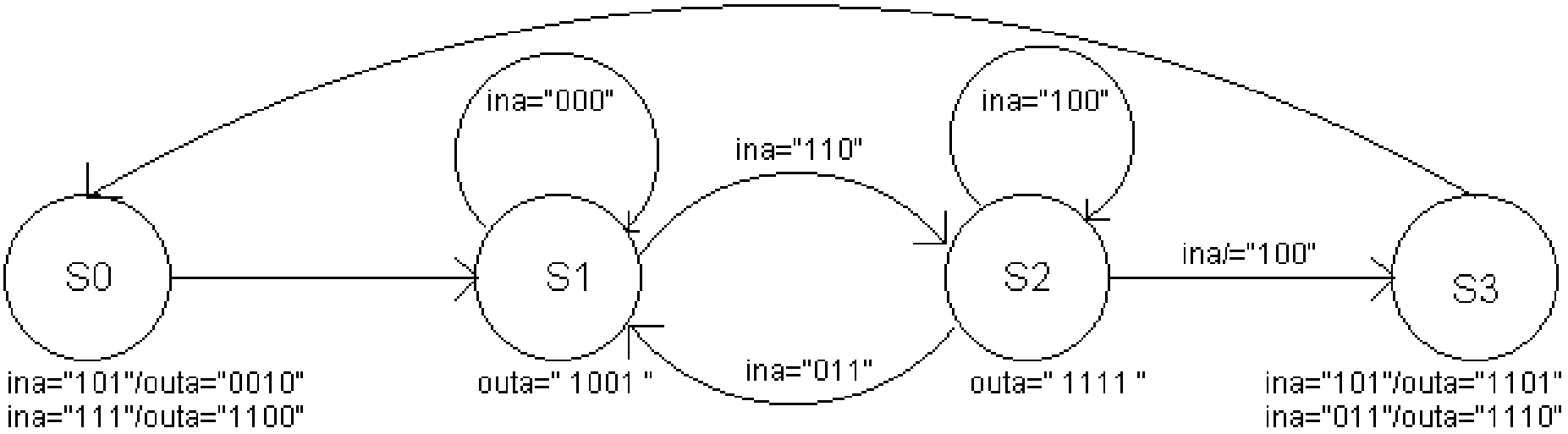
提示：对于D<=“11100101”，电路需分别不间断记忆：初始状态、1、11、111 、 1110 、11100、111001、1110010、11100101 共9种状态。

习 题

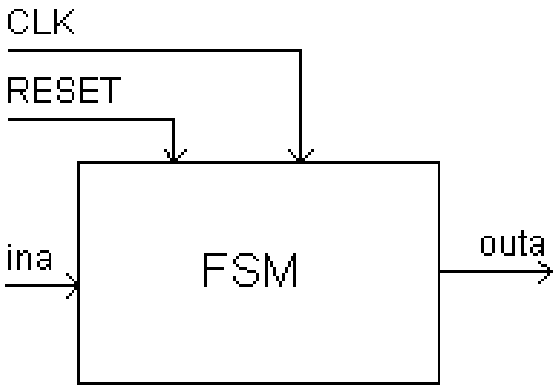
8-4. 根据图8-14(a)所示的状态图，分别按照图8-4(b)和图8-14(c)写出对应结构的VHDL状态机。

8-5. 在不改变原代码功能的条件下用两种方法改写例8-2，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。

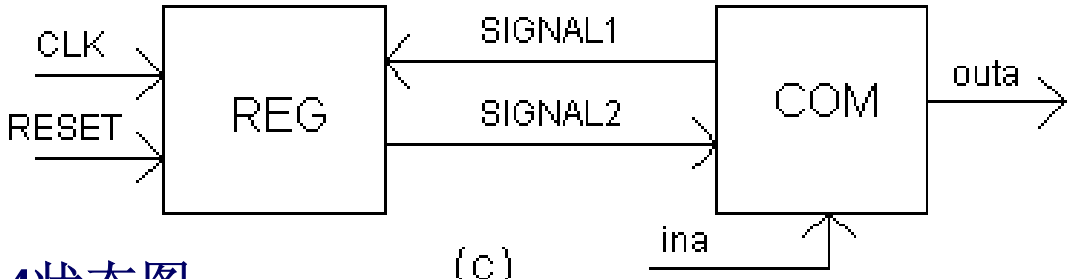
习 题



(a)



(b)



(c)

图8-14 习题8-4状态图

实 验 与 设 计

8-1. 序列检测器设计

(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**8-3**中作了说明。

(3) 实验内容1：仔细完成习题**8-3**的全部内容，利用**QuartusII**对例**8-11**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。

建议选择电路模式**No.8**（附录图**9**），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“**1100101**”；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“**1100101**”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**。

实 验 与 设 计

8-1. 序列检测器设计

(4) 实验内容2：根据习题8-3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。

(6) 实验报告：根据以上的实验内容写出实验报告，包括设计原理、程序设计、程序分析、仿真分析、硬件测试和详细实验过程。

实 验 与 设 计

8-2. ADC0809采样控制电路实现

(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：ADC0809的采样控制原理已在8.2.1节中作了详细说明(实验程序用例8-2)。

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8-3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。实验过程。

实 验 与 设 计

8-2. ADC0809采样控制电路实现

(3) 实验内容：利用**QuartusII**对例**8-2**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**8-2**电路对**ADC0809**的控制功能。

测试步骤：建议选择电路模式**No.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)**。

(4) 实验思考题：在不改变原代码功能的条件下将例**8-2**表达成用状态码直接输出型的状态机。

(5) 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

8-3. 数据采集电路和简易存储示波器设计

(1) 实验目的：掌握**LPM RAM**模块**VHDL**元件定制、调用和使用方法；熟悉**A/D**和**D/A**与**FPGA**接口电路设计；了解**HDL**文本描述与原理图混合设计方法。

(2) 实验原理：主要内容参考本章和**7.2**节。本设计项目是利用**FPGA**直接控制**0809**对模拟信号进行采样，然后将转换好的**8**位二进制数据迅速存储到存储器中，在完成对模拟信号一个或数个周期的采样后，由外部电路系统**(**如单片机**)**将存储器中的采样数据读出处理。**1.** 元件**ADCINT**。**ADCINT**是控制**0809**的采样状态机，其**VHDL**描述以及其输入输出信号的含义与例**8-2**完全相同，工作方式可参考**8.2.1**节；。

(3) 实验内容1：设**ADDA='1'**；即模拟信号来自**0809**的**IN1**口（可用实验系统右下角的电位器产生被测模拟信号）完成此项设计，给出仿真波形及其分析，将设计结果在**Cyclone**中硬件实现，用**QuartusII**的在系统**RAM/ROM**数据编辑器了解采入**RAM**中的数据。

(4) 实验内容2：优化设计。仿真设计电路图**8-15**，检查此项设计得**START**信号是否有毛刺，如果有，改进**ADCINT**的设计（也可用其他方法），排除**START**的毛刺。

实 验 与 设 计

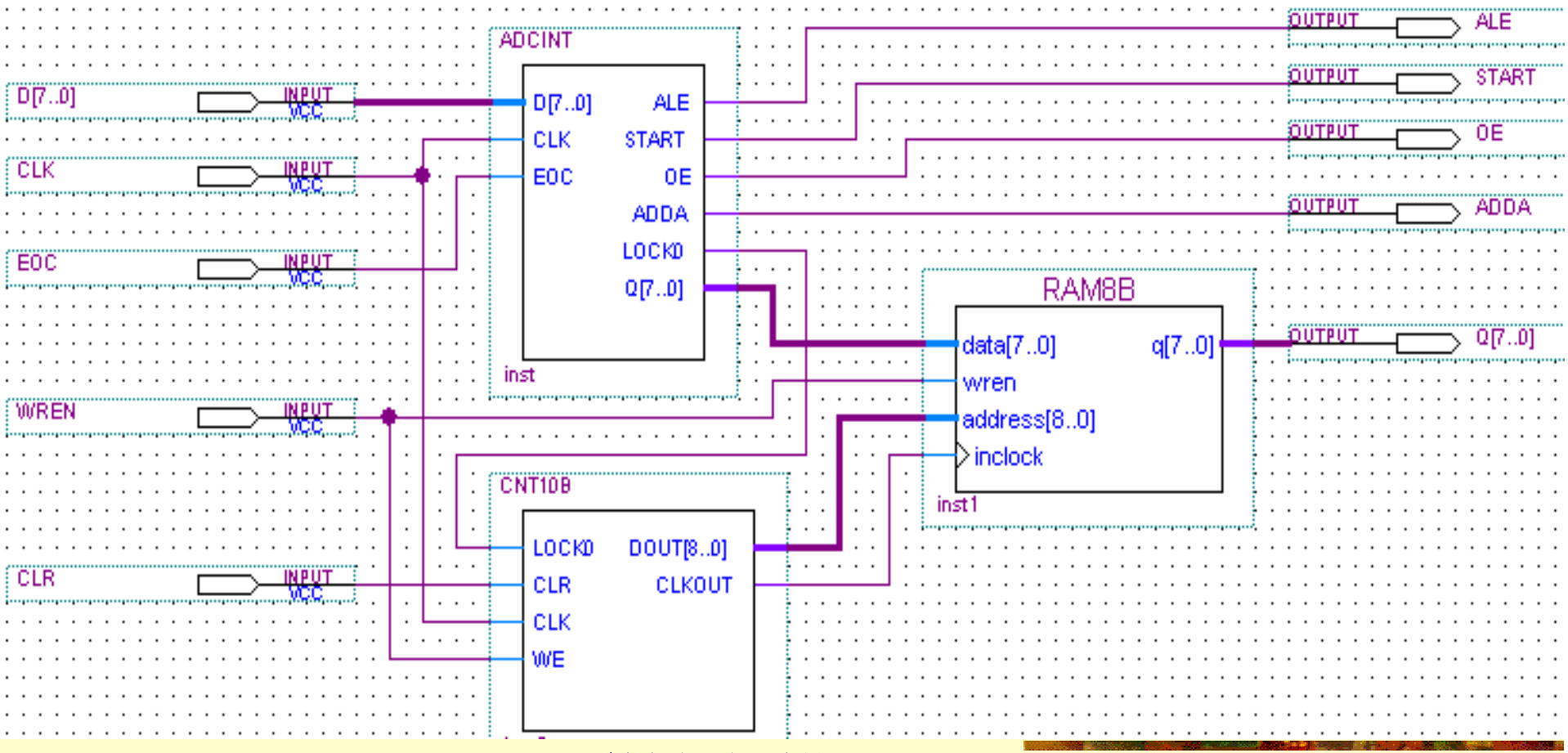


图8-15 ADC0809采样电路系统：RSV.bdf

实 验 与 设 计

8-3. 数据采集电路和简易存储示波器设计

(5) 实验内容3：对电路图**8-15**完成设计和仿真后锁定引脚，进行硬件测试。参考实验**8-2**和**7-1**对**0809**和**0832**的引脚锁定：元件**“ADCINT”**引脚锁定参考实验**8-2**。**WE**用键**1**控制；为了实验方便，**CLK**接**clock0**，频率先选择**64Hz**（选择较慢的采样时钟），作状态机工作时钟。硬件实验中，建议选择电路模式**No.5**，打开**+/-12V**电源，首先使**WE=’1’**，即键**1**置高电平，允许采样，由于这时的程序中设置**ADDA <= '1'**，模拟信号来自**AIN1**，即可通过调协实验板上的电位器（此时的模拟信号是手动产生的），将转换好的数据采入**RAM**中；然后按键**1**，使**WE=’0’**，**clock0**的频率选择**16384Hz**（选择较高时钟），即能从示波器中看见被存于**RAM**中的数据（可以首先通过**QuartusII**的**RAM**在系统读写器观察已采入**RAM**中的数据）。

实 验 与 设 计

8-3. 数据采集电路和简易存储示波器设计

(6) 实验内容4：程序中设置**ADDA <= '0'**，模拟信号将由**AIN0**进入，即**AIN0**的输入信号来自外部信号源的模拟连续信号。外部模拟信号可来自实验箱，方法如下：

首先打开**+/-12V**电源，将**GW48**主系统板右侧的**“JL11”**跳线座短路**“L_F”**端；跳线座**“JP18”**的**“INPUT”**端与系统右下角的时钟**64Hz**相接；并用一插线将插座**“JP17”**的**“OUTPUT”**端与实验箱最左侧的**“JL10”**坐的**“AIN0”**端相接，这样就将**64Hz**待采样的模拟信号接入了**0809**的**IN0**端（注意，这时例**8-2/12**程序中设置**ADDA <= '0'**）。试调节**“JP18”**上方的电位器，使得主系统右侧的**“WAVE OUT”**端输出正常信号波形（用示波器监视，峰值调在**4V**以下）。注意，如果要将采入（用**CLK=64Hz**采样）**RAM**中的数据扫描显示到示波器上观察，必须用高频率时钟才行（**clock0**接**16384Hz**）。

可以使键**1**高电平是对模拟信号采样，低电平时示波器显示已存入**RAM**的波形数据。

实 验 与 设 计

8-3. 数据采集电路和简易存储示波器设计

(6) 实验内容5：仅按照以上方法，会发现示波器显示的波形并不理想，原因是从**RAM**中扫出的数据都不是一个完整的波形周期。试设计一个状态机，结合被锁入**RAM**中的某些数据，改进元件**CNT10B**，使之存入**RAM**中的数据和通过**D/A**在示波器上扫出的数据都是一个或数个完整波形数据。

(7) 实验内容6：在图**8-15**的电路中增加一个锯齿波发生器，扫描时钟与地址发生器的时钟一致。锯齿波数据通过另一个**D/A**输出，控制示波器的**X**端（不用示波器内的锯齿波信号），而**Y**端由原来的**D/A**给出**RAM**中的采样信息，由此完成一个比较完整的存储示波器的显示控制。

8-3. 数据采集电路和简易存储示波器设计

【例8-12】

```
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 设 计

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

(1) 实验目的：学习较复杂状态机的设计。

(2) 实验原理：图8-19是一个用比较器LM311和DAC0832构成的8位A/D转换器的电路框图。其工作原理是：当被测模拟信号电压vi接于LM311的“+”输入端时，由FPGA产生自小到大的搜索数据加于DAC0832后，LM311的“-”端将得到一个比较电压vc；当vc<vi时，LM311的“1”脚输出高电平’1’，而当vc>vi时，LM311输出低电平。在LM311输出由’1’到’0’的转折点处，FPGA输向0832数据必定与待测信号电压vi成正比。由此数即可算得vi的大小。

(3) 实验内容1：例8-13是图8-16中FPGA的一个简单的示例性程序。

实 验 与 设 计

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

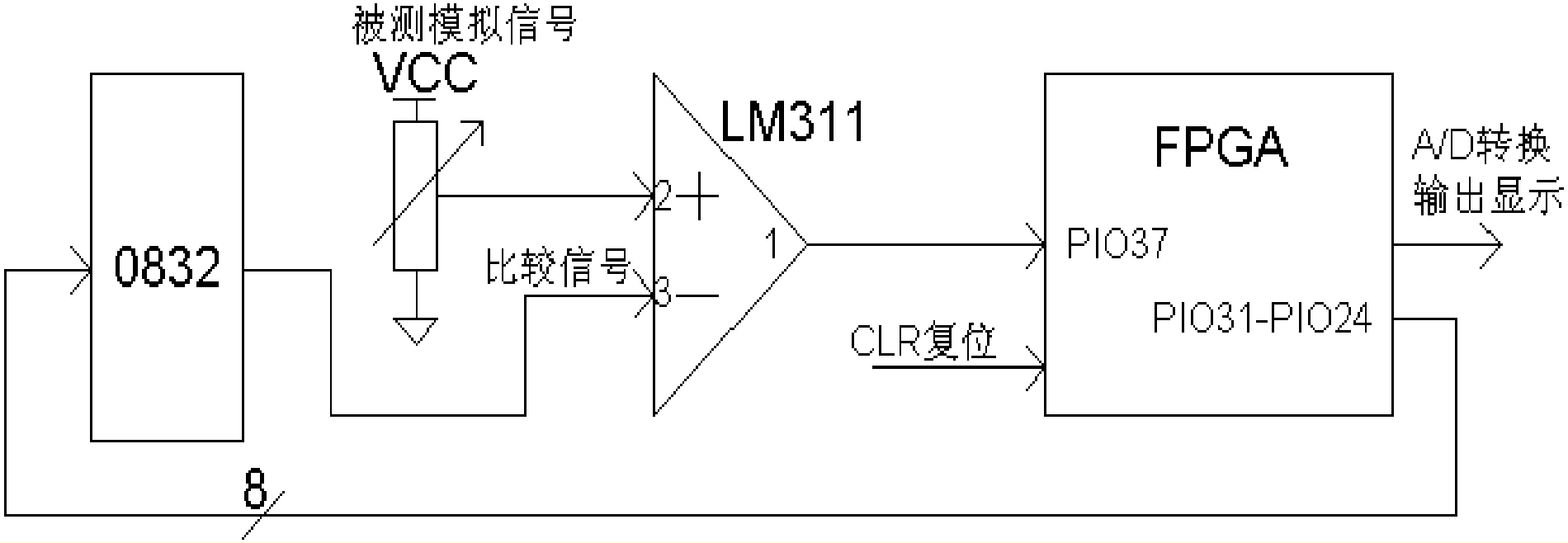


图8-16 比较器和D/A构成A/D电路框图。

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

【例8-13】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DAC2ADC IS
    PORT ( CLK     :  IN STD_LOGIC; --计数器时钟
           LM311   :  IN STD_LOGIC;  --LM311输出，由PIO37口进入FPGA
           CLR     :  IN STD_LOGIC;  --计数器复位
            DD     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;--输向0832的数据
         DISPDATA : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );--转换数据显示
END;
ARCHITECTURE DACC OF DAC2ADC IS
 SIGNAL CQI  : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
 BEGIN
 DD <= CQI ;
PROCESS(CLK, CLR, LM311)
    BEGIN
      IF CLR = '1' THEN   CQI <= "00000000";
       ELSIF CLK'EVENT AND CLK = '1' THEN
        IF LM311 = '1' THEN CQI <= CQI + 1;  END IF;--如果是高电平，继续搜索
            END IF;         --如果出现低电平，即可停止搜索，保存计数值于CQI中
    END PROCESS;
 DISPDATA <= CQI  WHEN LM311='0' ELSE "00000000" ;--将保存于CQI中的数输出
 END;
```

实 验 与 设 计

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

(3) 实验内容2： 例**8-14**的缺点有**2**个：**1**、无法自动搜索被测信号，每次测试都必须复位一次；**2**、由于每次搜索都是从**0**开始，从而**“A/D**转换**”**速度太慢。

试设计一个控制搜索的状态机，克服这两个缺点。且尽量提高“转换”速度，如安排一个特定的算法（如黄金分割法）进行快速搜索。

实 验 与 设 计

8-5. 通用异步收发器设计

(1) 实验目的：学习利用状态机设计通用异步RS232硬件通信模块，并根据图8-22设计信号采集、分析与通信模块。

(2) 实验原理：UART即Universal Asynchronous Receiver Transmitter通用异步收发器，是一种应用广泛的短距离串行传输接口。往往用于短距离、低速、低成本的微机与下位机的通讯中。8250、8251、NS16450等芯片都是常见的UART器件。这类芯片有些已经做得相当复杂，含有许多辅助的模块，比如FIFO。图8-17是常见的UART连接通信图。注意，图8-17中两边的TXD、RXD信号是交错的。TXD是UART发送端，为输出；RXD是UART接收端，为输入。在TXD、RXD信号线上的电平也不是普通的TTL 5V电平，而是RS232的接口电平。

实 验 与 设 计

8-5. 通用异步收发器设计

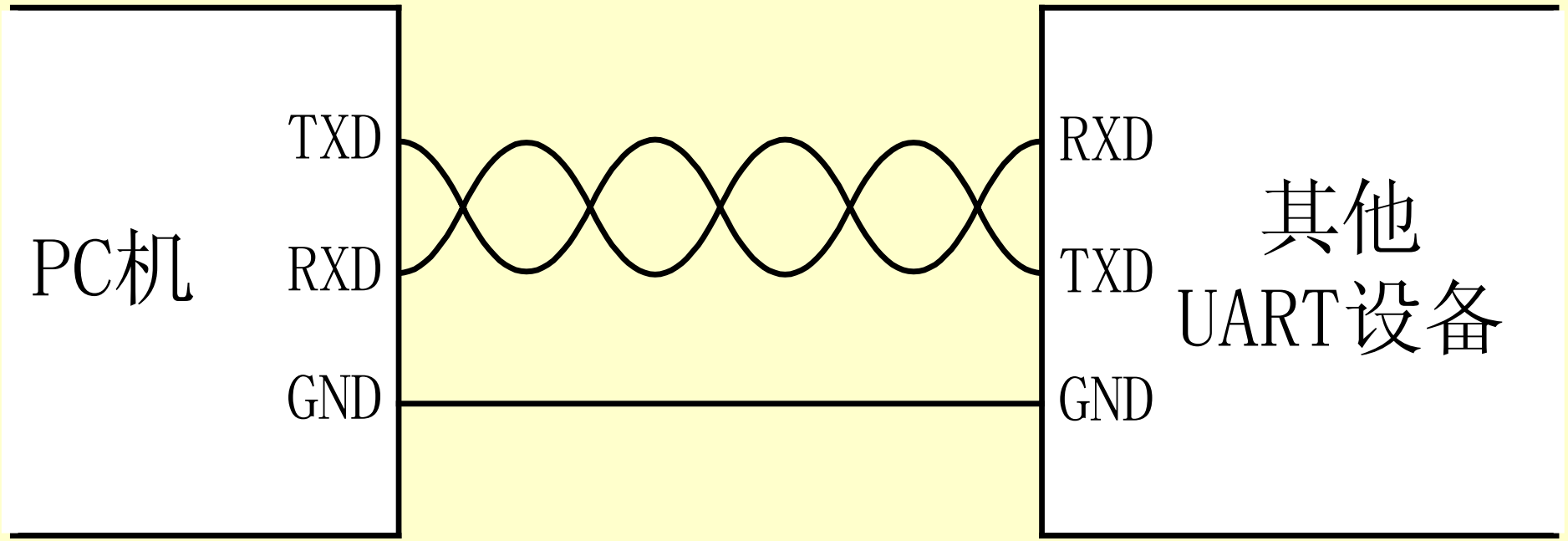


图8-17 UART三线连接通信示意

实 验 与 设 计

8-5. 通用异步收发器设计

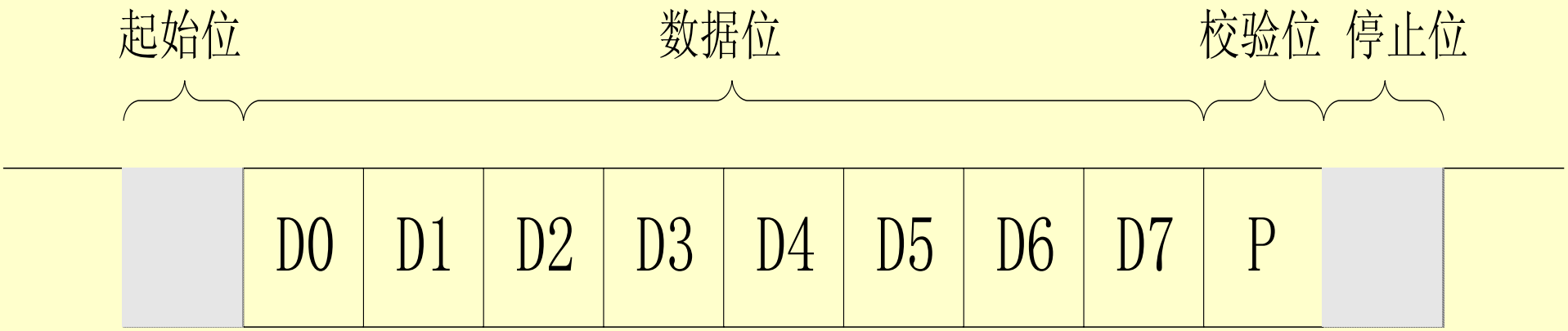


图8-18 基本UART帧格式

实 验 与 设 计

8-5. 通用异步收发器设计

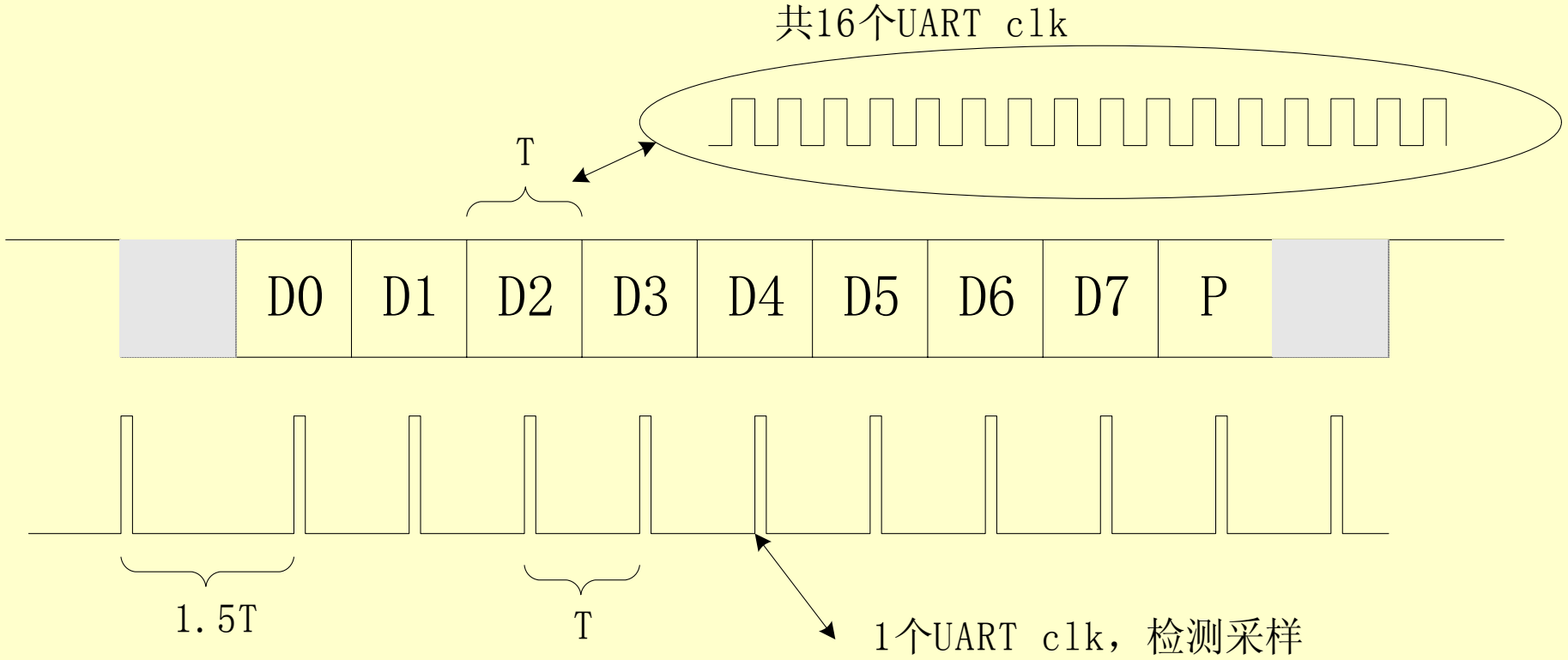


图8-19 基本UART帧时序

8-5. 通用异步收发器设计

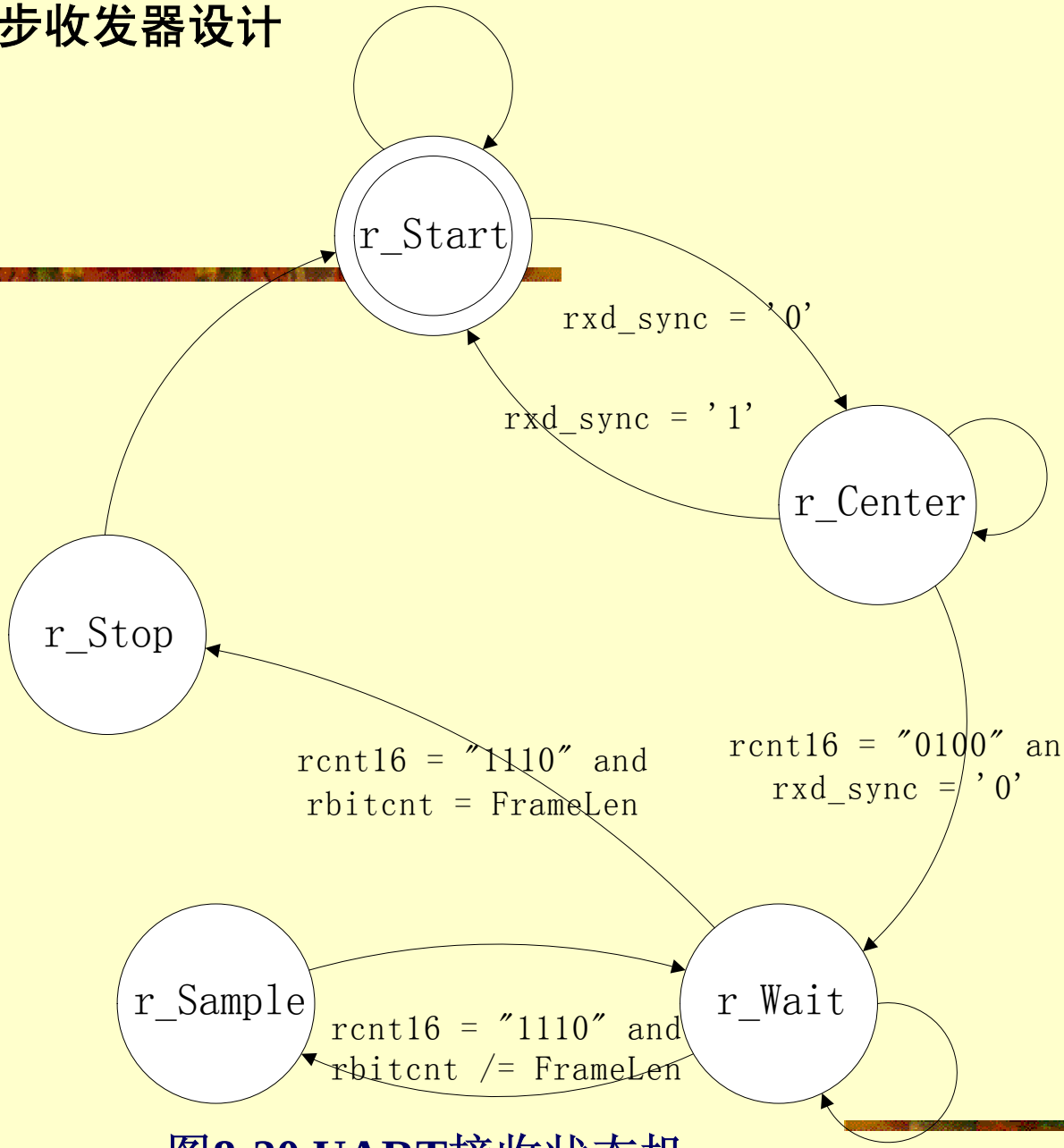


图8-20 UART接收状态机

8-5. 通用异步收发器设计

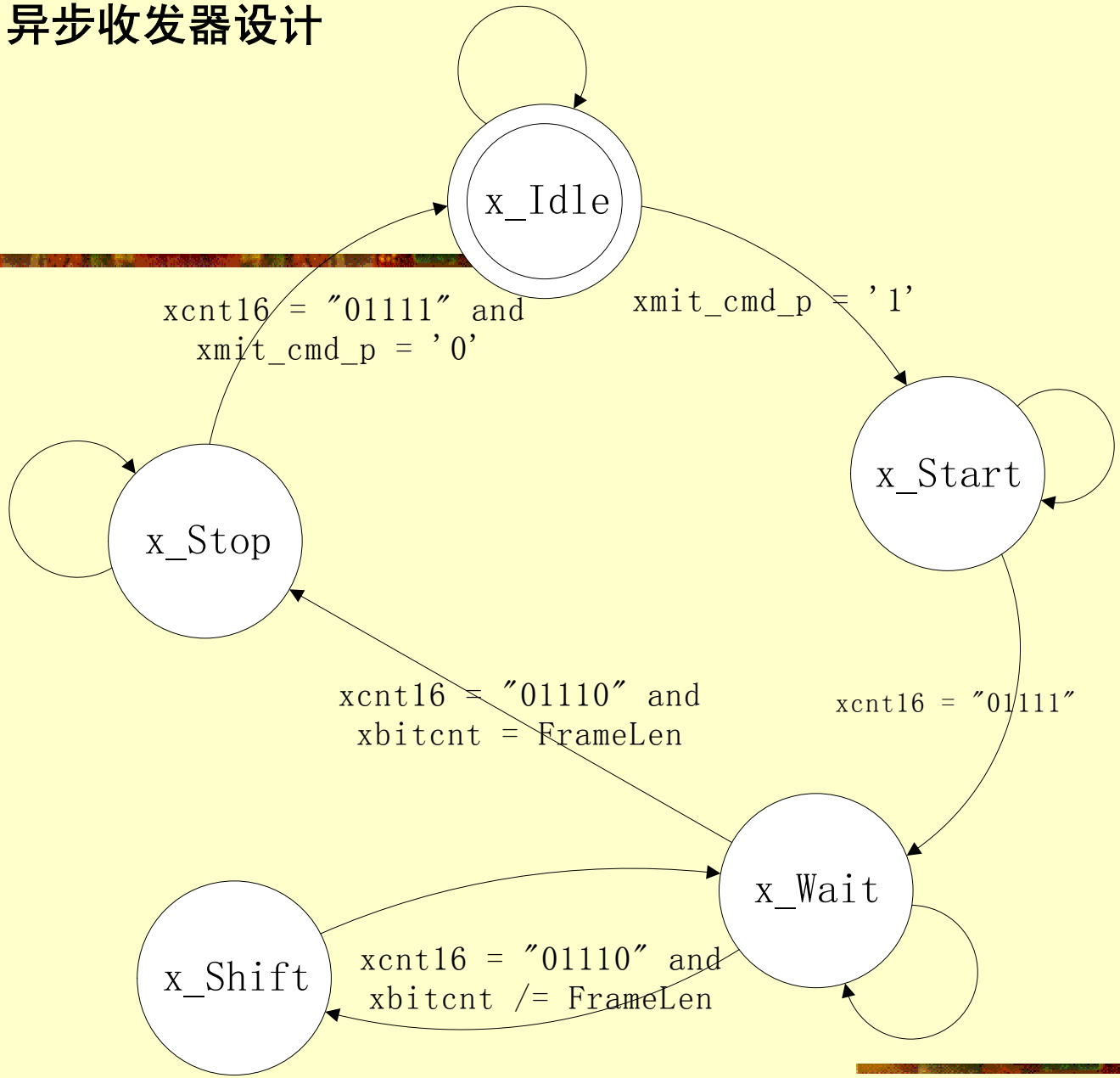


图8-21 UART发送状态机

实 验 与 设 计

8-5. 通用异步收发器设计

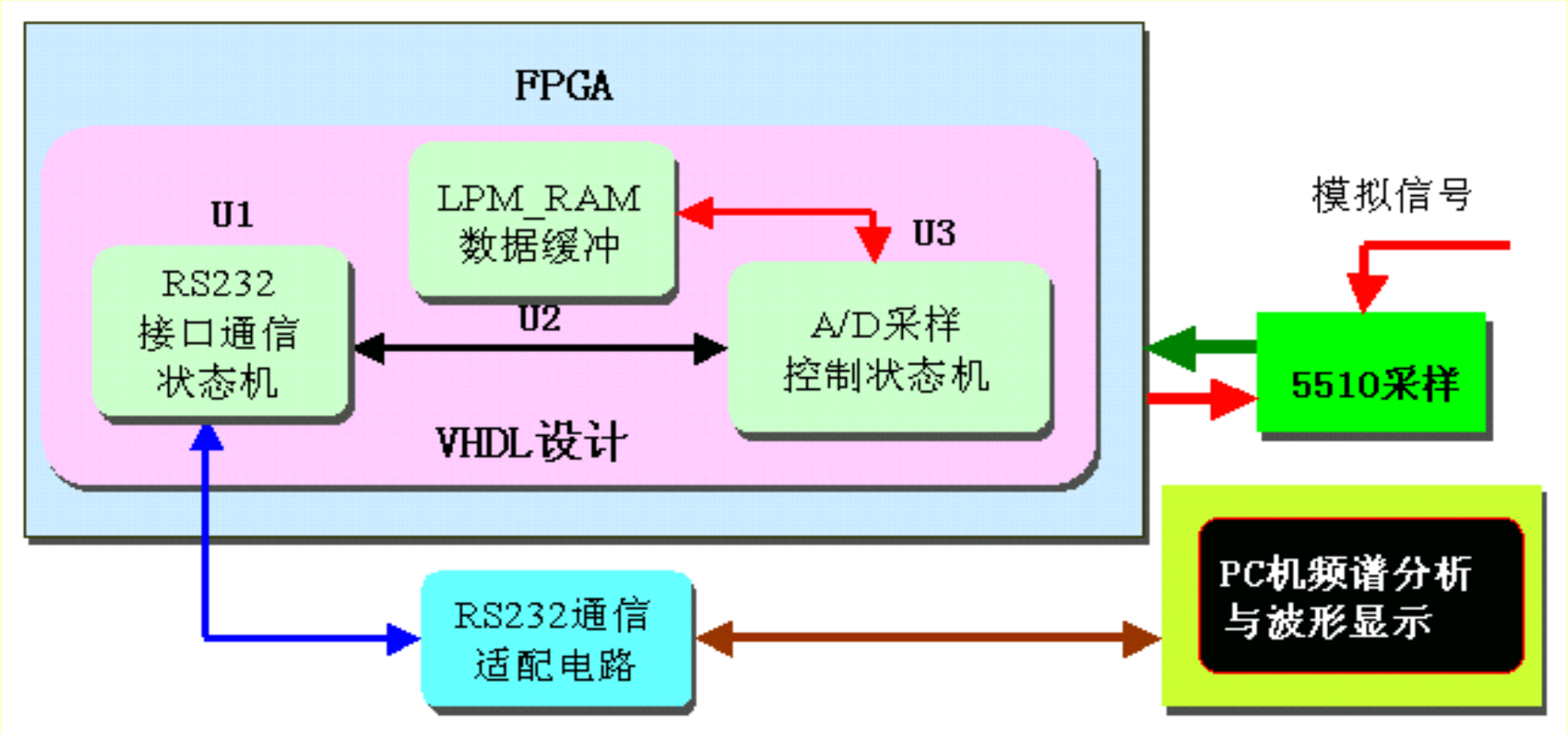


图8-22信号采集与频谱分析电路模块图

8-5. 通用异步收发器设计

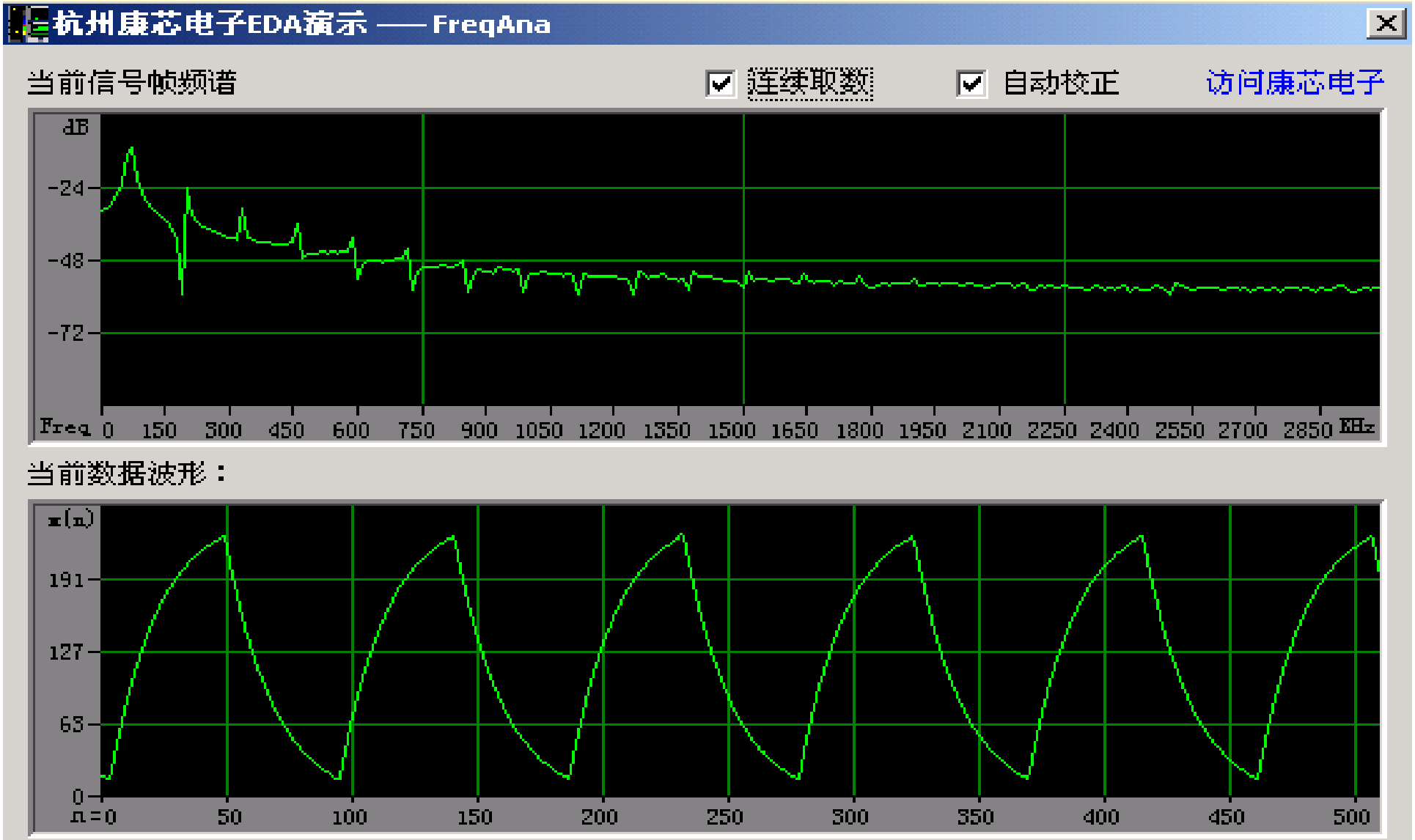


图8-23 采集的65536Hz模拟信号PC机显示

8-5. 通用异步收发器设计

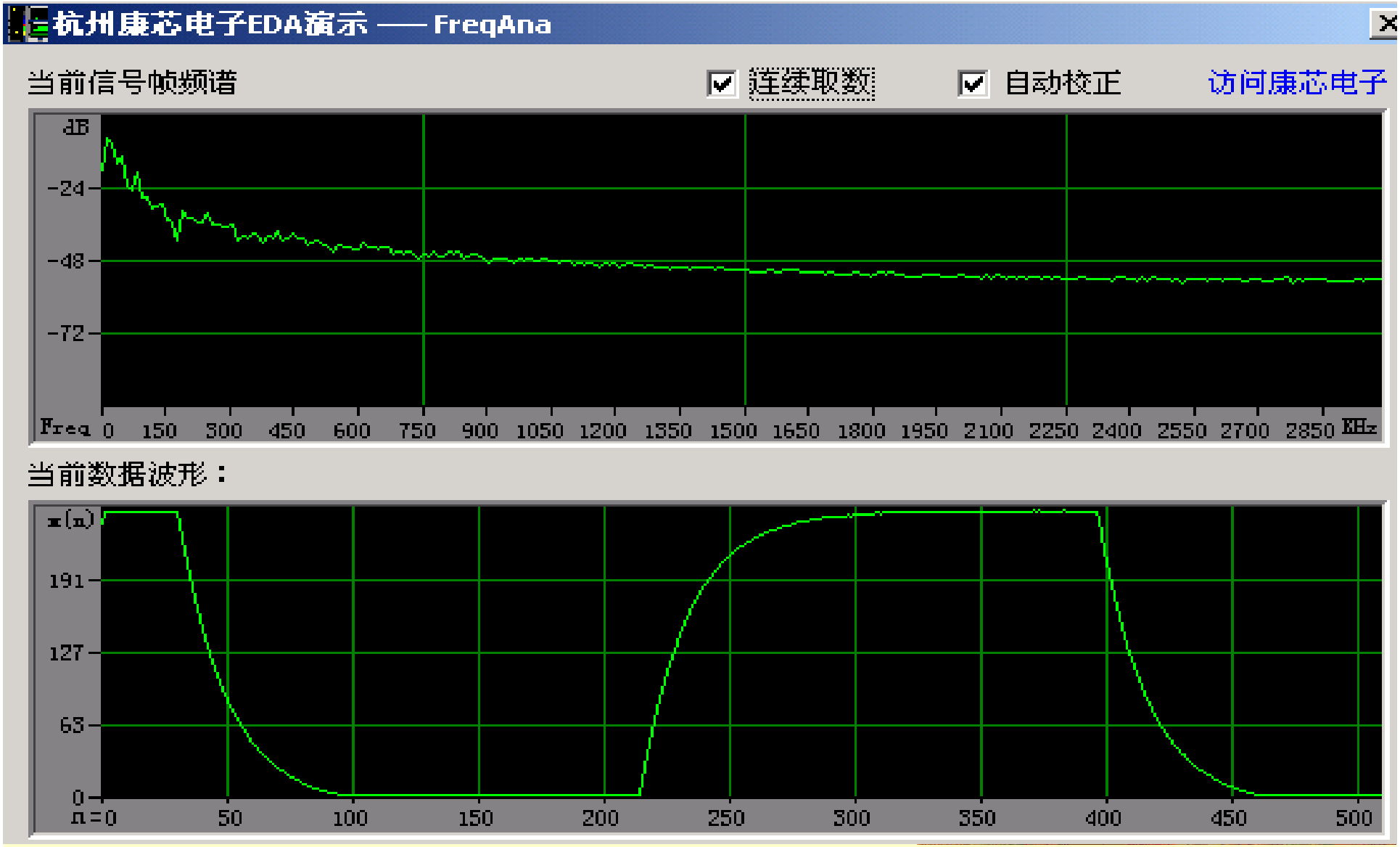


图8-24 采集的16384Hz模拟信号PC机显示

8-5. 通用异步收发器设计

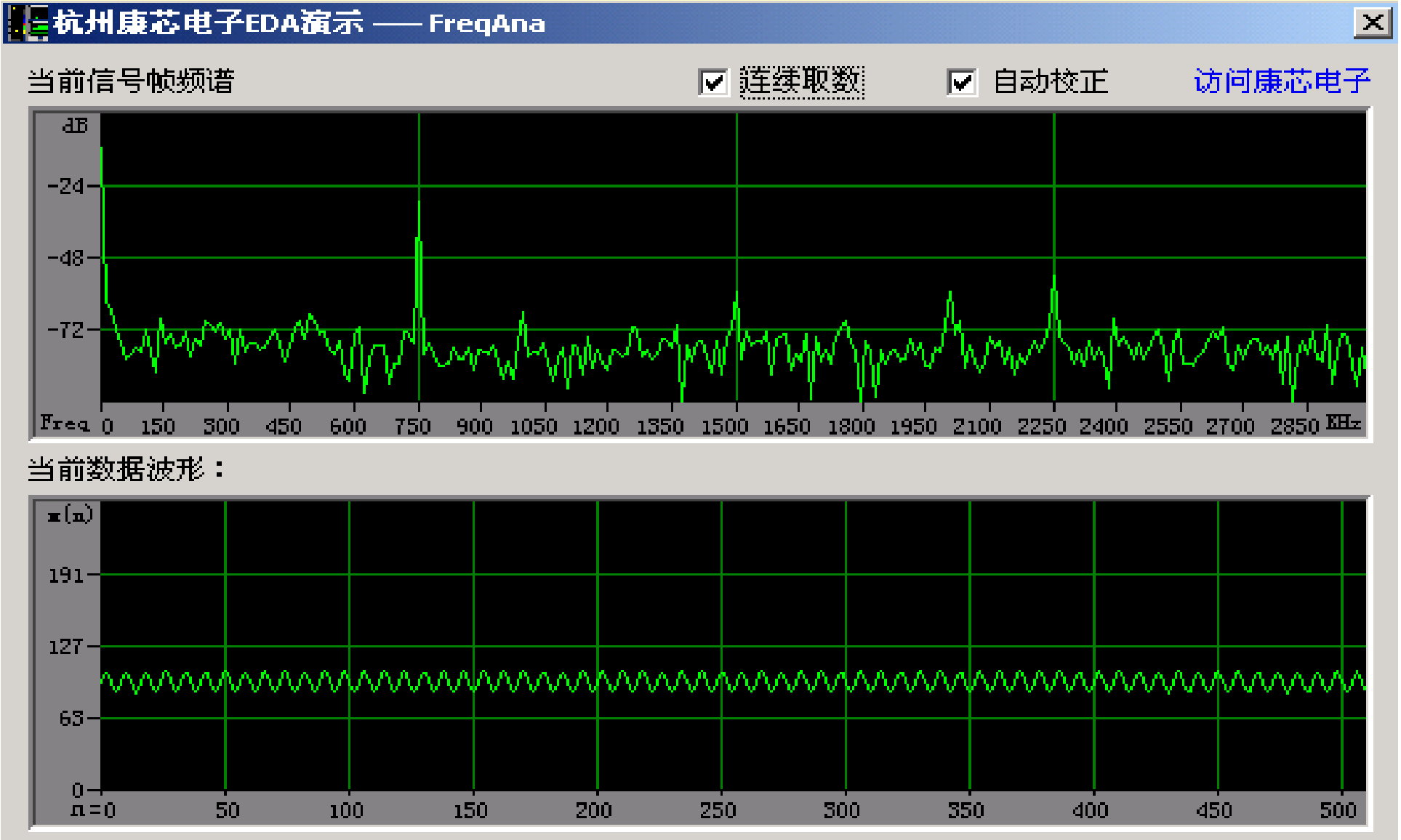


图8-23 采集的65536Hz模拟信号PC机显示

实 验 与 设 计

8-5. 通用异步收发器设计

(3) 实验内容：完成已有设计的验证性实验：将RS232通信线的一头接GW48系统，另一头接PC机的串行1口（COM1口）；将GW48系统右侧的开关向左拨“TO FPGA”，以便使FPGA直接与PC机的RS232通信接口相连；使clock0输入13MHz频率；下载SPECTREM中的ADSUART.SOF，到FPGA中；用模式键选模式“5”，再按一次右侧的复位键，按键1，键2，使其输出高电平；进入“README”目录，运行（双击）并安装分析软件：FASetup.exe ；将模拟信号输入ADDA适配板上的模拟信号输入端口“AIN”。

8-5. 通用异步收发器设计

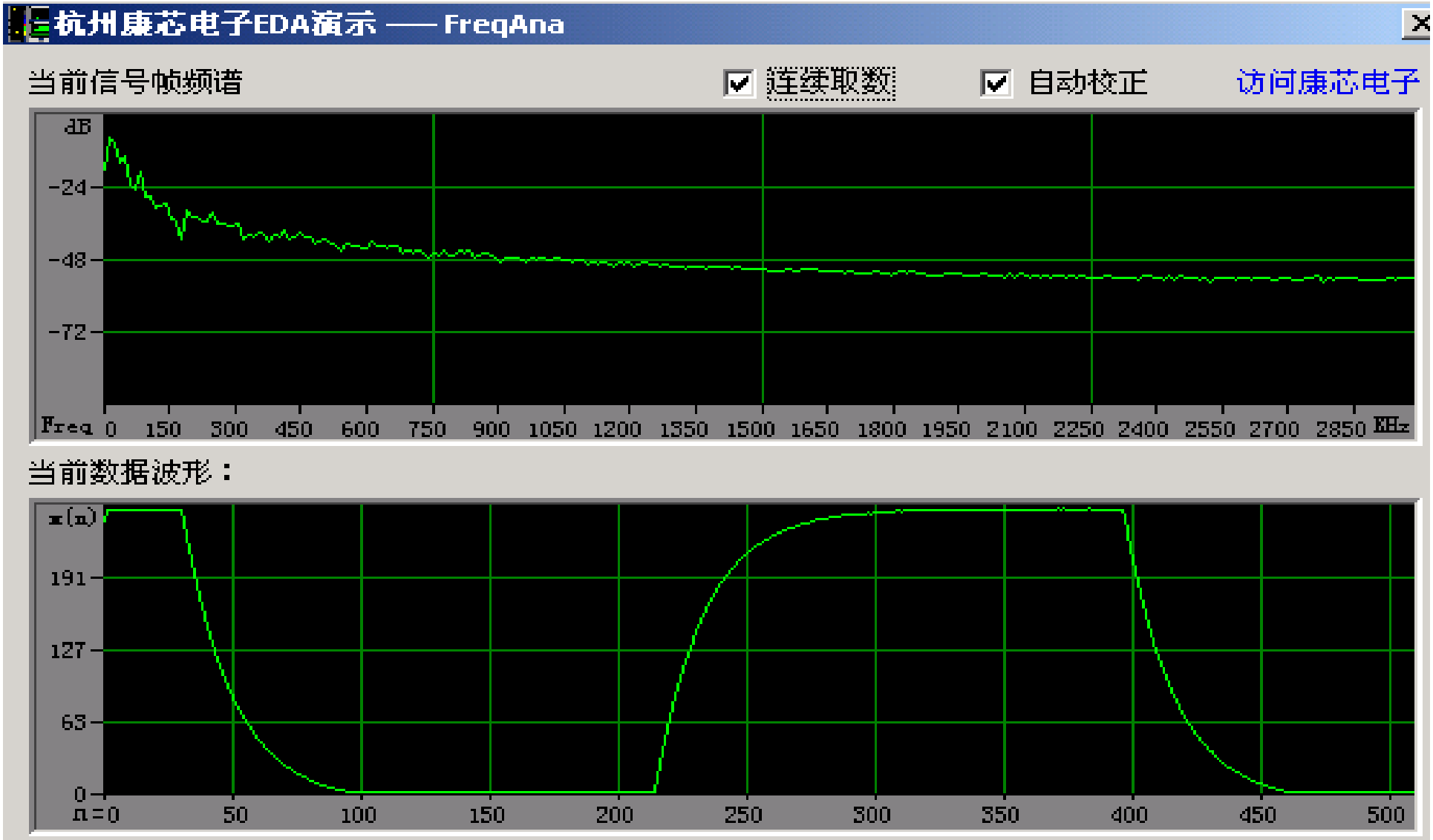


图8-24 采集的16384Hz模拟信号PC机显示

8-5. 通用异步收发器设计

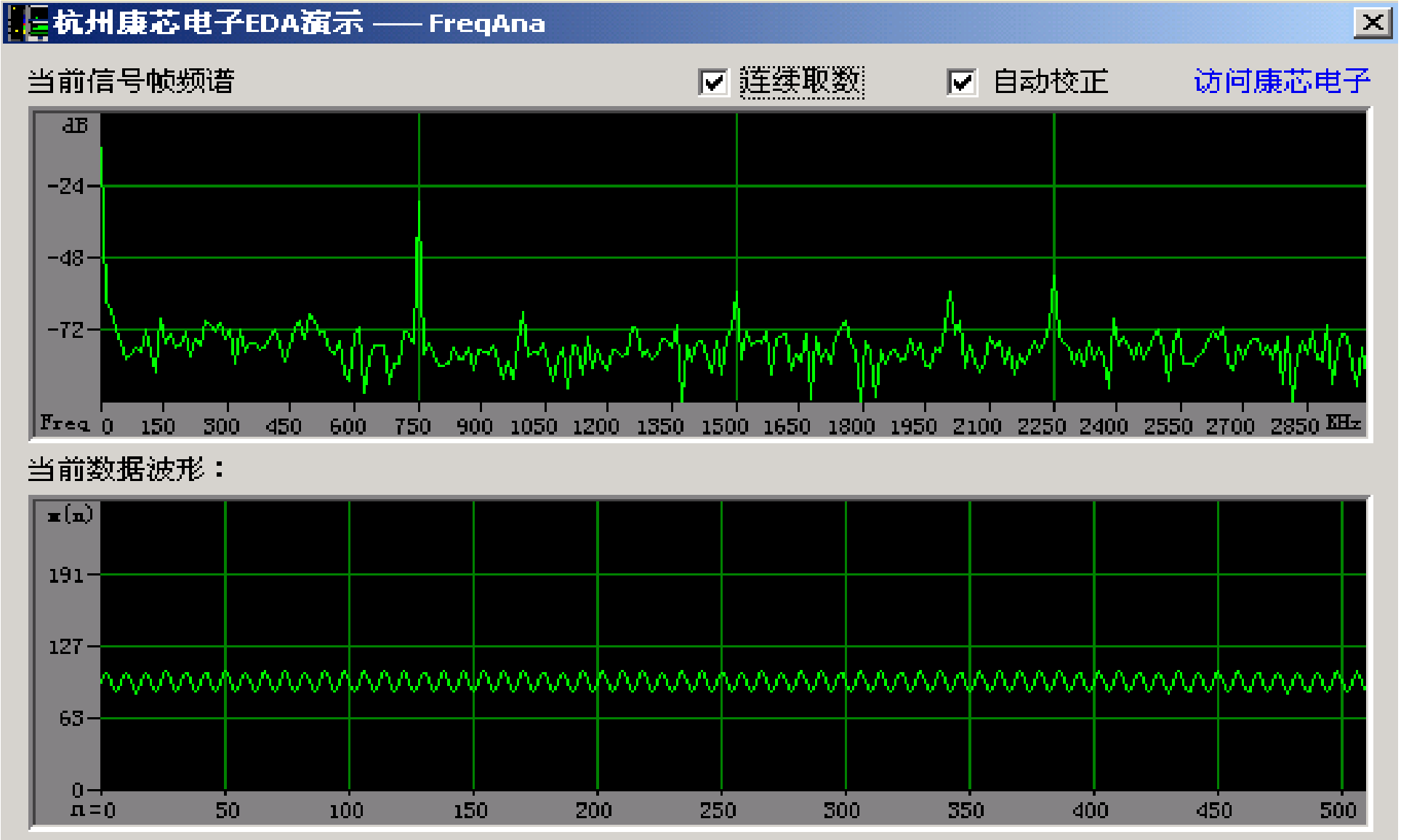


图8-25 采集的750KHz模拟信号PC机显示