5 时序逻辑电路建模

5.1状态机的基本概念

5.2基于Verilog HDL的状态机描述方法

5.3状态机设计中的关键技术

5.4状态机设计举例

5.1 状态机的基本概念

5.1.1 状态机的基本结构及类型

5.1.2 状态机的状态图表示法

5.1.3 状态机的设计步骤

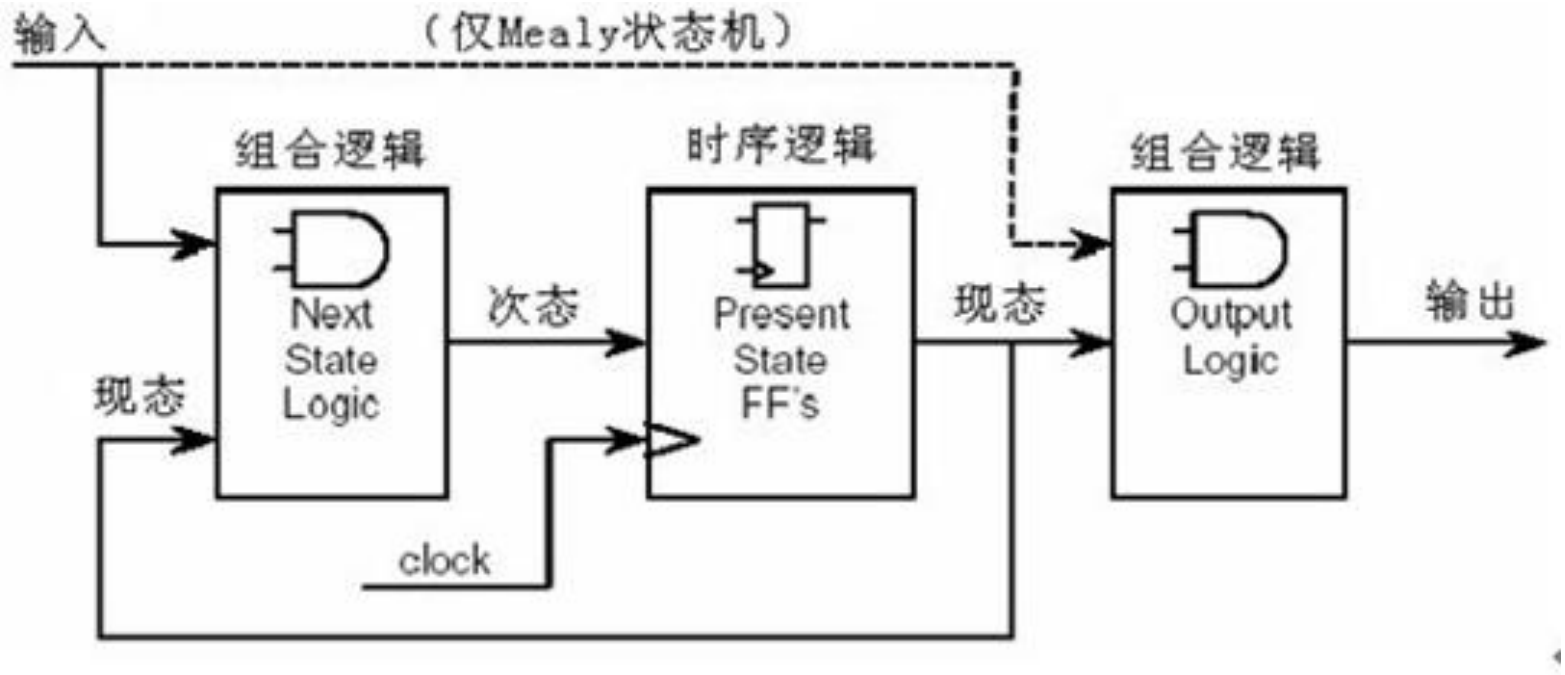
5.1.1 状态机的基本结构及类型

有限状态机的标准模型如图5.1.1，它主要由三部分组成：

(1)下一状态的逻辑电路，

(2) 存储状态机当前状态的时序逻辑电路，

(3) 输出组合逻辑电路。



5.1.1状态机的基本结构及类型

一般情况下，状态触发器的数量是有限的，其状态数也是有限的，故称为有限状态机（Finite State Machine，简称为FSM）。状态机中所有触发器的时钟输入端被连接到一个公共时钟脉冲源上，其状态的转换是在同一时钟源的同一脉冲边沿同步进行的，所以它也被称作时钟同步状态机。

一般来说，状态机的基本操作主要有以下两种：

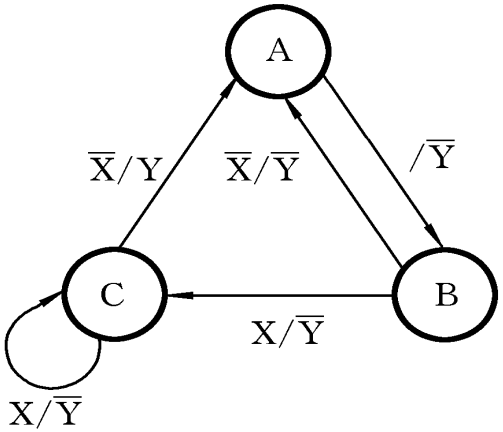
(1) 状态机的内部状态转换。

(2) 产生输出信号序。

根据电路的输出信号是否与电路的输入有关，可以将状态机分为两种类型：一类是米利型（Mealy）状态机，电路的输出信号不仅与电路当前的状态有关，还与电路的输入有关；另一类是穆尔型（Moore）状态机，电路输出仅仅取决于各触发器的状态，而不受电路当时的输入信号影响或没有输入变量。

5.1.2 状态机的状态图表示法

图5.1.2是米利型状态图的一个例子。在状态图中，每个状态用一个圆圈表示，圆圈内有指示状态的符号。用带箭头的方向线指示状态转换的方向，当方向线的起点和终点都在同一个圆圈上时，则表示状态不变。



注：X/Y 表示输入/输出

A、B、C 是符号，表示不同的状态

图 5.1.2　**米利型**状态图

一般来说，状态机中的状态转移有两种方式：无条件转移和有条件转移。在图4.1.2中，从状态A转移到状态B为无条件转移，其它状态之间的转移都是有条件要求的。

5.1.2状态机的状态图表示法

需要强调指出，米利型状态图中，输出值的表示方法容易引起读者的误解。当状态机处于所在的状态，并且在所示的输入的作用下，就会产生输出值，并非在状态机转移到下一状态时才出现输出。由于穆尔型电路的输出只依赖于状态机的当前状态，其状态图的表示方法略有不同，通常将输出变量写在表示状态的圆圈内部，图5.1.3是穆尔型状态图表示的例子。

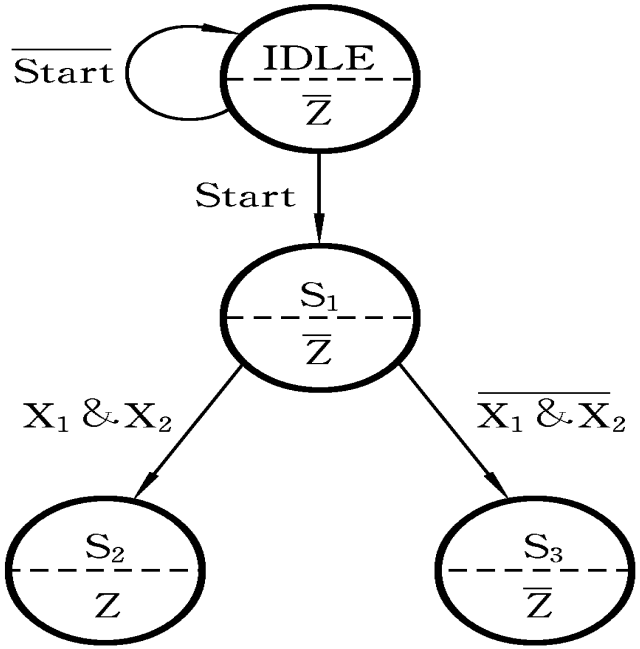


图 5.1.3　穆尔型状态图

5.1.3状态机的设计步骤

一般来说，状态机的设计步骤如下所示：

1.依据具体的设计原则，确定采用穆尔型状态机还是米利型状态机。

2.分析设计要求，列出状态机的所有状态，并对每一个状态进行状态编码。

3.根据状态转移关系和输出函数，画出状态图。

4.根据所画的状态图，采用硬件描述语言对状态机进行描述。

第3步是最困难也是最有创造性的一步。状态图直观地反映了状态机各个状态之间的转换关系以及转换条件，但要求设计的状态个数不能太多。状态个数较多，采用状态表的方法列出状态机的转移条件。输出信号较多，可采用输出逻辑真值表进行表示。

5.2 基于Verilog HDL的状态机描述方法

5.2.1 状态图的建立过程

5.2.2 状态图描述方法

5.2基于Verilog HDL的状态机描述方法

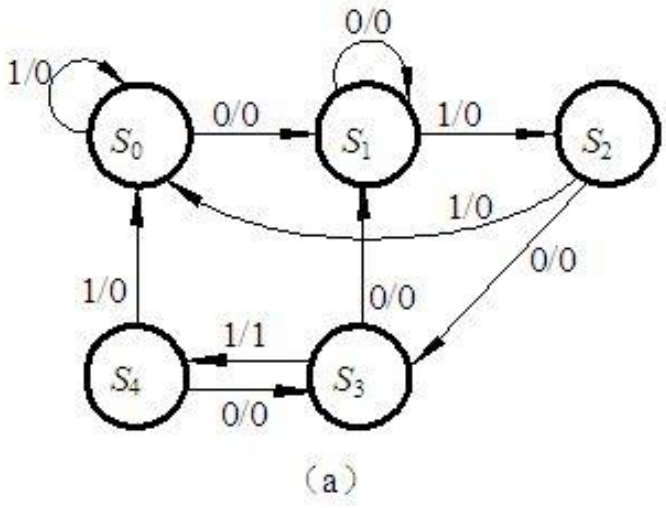
5.2.1 状态图的建立过程

设计一个序列检测器电路。功能是检测出串行输入数据*Sin*中的4位二进制序列0101（自左至右输入），当检测到该序列时，输出*Out*=1；没有检测到该序列时，输出*Out*=0。（注意考虑序列重叠的可能性，如010101，相当于出现两个0101序列）。

解：首先，确定采用米利型状态机设计该电路。因为该电路在连续收到信号0101时，输出为1，其他情况下输出为0，所以采用米利型状态机。

其次，确定状态机的状态图。根据设计要求，该电路至少应有四个状态，分别用S_1、S_2、S_3、S_4表示。若假设电路的初始状态用S_0表示，则可用五个状态来描述该电路。根据分析，可以画出图(a)所示的原始状态图。

5.2.1 状态图的建立过程



观察该图可以看出，S_2、S_4为等价状态，可用S_2代替S_4，于是得到简化状态图，如图(b)所示。

然后，根据上面的状态图给出该状态机的输出逻辑。该状态机只有一个输出变量*Out*，其输出逻辑非常简单，直接标注在状态图中了。若输出变量较多，则可以列出输出逻辑真值表。

最后，就可以使用硬件描述语言对状态图进行描述了。

5.2.2 推荐的状态图描述方法

利用Verilog HDL语言描述状态图主要包含四部分内容：

1，利用参数定义语句parameter描述状态机中各个状态的名称，并指定状态编码。例如，对序列检测器的状态分配可以使用最简单的自然二进制码，其描述如下：

```
parameter  S0=2'b00, S1=2'b01, S2 = 2'b10, S3 = 2'b11;
```

或者，

```
parameter [1:0] S0=2'b00, S1=2'b01, S2 = 2'b10, S3 = 2'b11;
```

2，用时序的always 块描述状态触发器实现的状态存储。

3，使用敏感表和case语句(也可以采用if-else等价语句)描述的状态转换逻辑。

4，描述状态机的输出逻辑。

描述状态图的方法多种多样，下面介绍几种：

5.2.2 推荐的状态图描述方法

1. 单个always块描述状态机的方法（应该避免的写法）

用一个always块对该例的状态机进行描述，其代码如下：

```
module Detector1 ( Sin, CP, nCR, Out) ;
        input Sin, CP, nCR; //声明输入变量
        output Out ; //声明输出变量
        reg Out;
        reg [ 1 : 0 ] State; //声明两个状态触发器变量state[1]和state[0]，记忆电路现态
//The state labels and their assignments
        parameter [1:0] S0=2'b00, S1=2'b01, S2 = 2'b10, S3 = 2'b11;
always @(posedge CP or negedge nCR)
begin
if (~nCR)
```

5.2.2 推荐的状态图描述方法

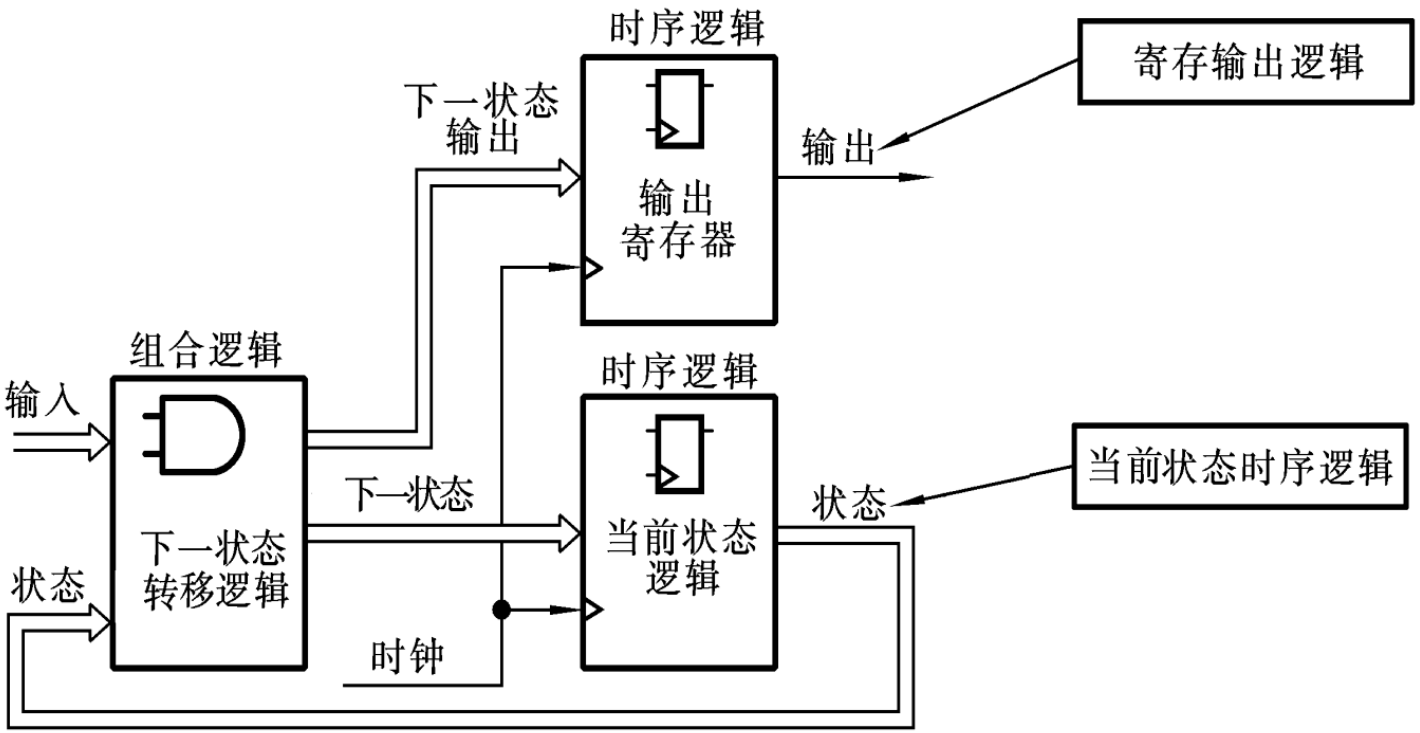
```
state <= S0;  //在nCR跳变为0时，异步清零
Else
case(State)
        S0: begin Out =1'b0; State = (Sin==1)? S0 : S1; end
        S1: begin Out = 1'b0; State = (Sin==1)? S2 : S1; end
        S2: begin Out = 1'b0; State = (Sin==1)? S0 : S3; end

  S3: if (Sin==1)  begin Out =1'b1; State = S2; end
                   else  begin Out =1'b0; State = S1; end
      endcase
end
Endmodule
```

严格地说，对序列检测器电路用单个always块的描述方法所描述的逻辑存在着一个隐含的错误，即输出信号Out的描述

5.2.2 推荐的状态图描述方法

存在着一个隐含的错误，即输出信号Out的描述存在错误。因此，单个always块描述状态机的写法仅仅适用于穆尔型状态机。单个always块写法的电路结构框图可以用图4.2.2进行概括。



5.2.2 推荐的状态图描述方法

2. 两个always块描述状态机的方法（推荐写法）

用两个always块对该例的状态机进行描述，其代码如下：

```
module  Detector2 ( Sin, CP, nCR, Out) ;
        input Sin, CP, nCR; //定义输入变量
        output Out ; //定义输出变量
        reg Out;
        reg [ 1 : 0 ] Current_state, Next_state;
parameter  S0=2'b00, S1=2'b01, S2 = 2'b10, S3 = 2'b11;
//状态转换，时序逻辑
always @(posedge CP or negedge nCR )
        begin
        if (~nCR)
                Current_state <= S0;  //异步清零
        else
```

5.2.2 推荐的状态图描述方法

```
Current_state <= next_state; //在CP上升沿触发器状态翻转
        end
//下一状态产生和输出信号，组合逻辑
always @( Current_state or Sin)
        begin
        Next_state =2'bxx;
Out=1'b 0;
case(Current_state )
          S0: begin Out =1'b0; Next_state = (Sin==1)? S0 : S1;
end
          S1: begin Out =1'b0; Next_state = (Sin==1)? S2 : S1;
end
           S2: begin Out =1'b0; Next_state = (Sin==1)? S0 : S3;
end
```

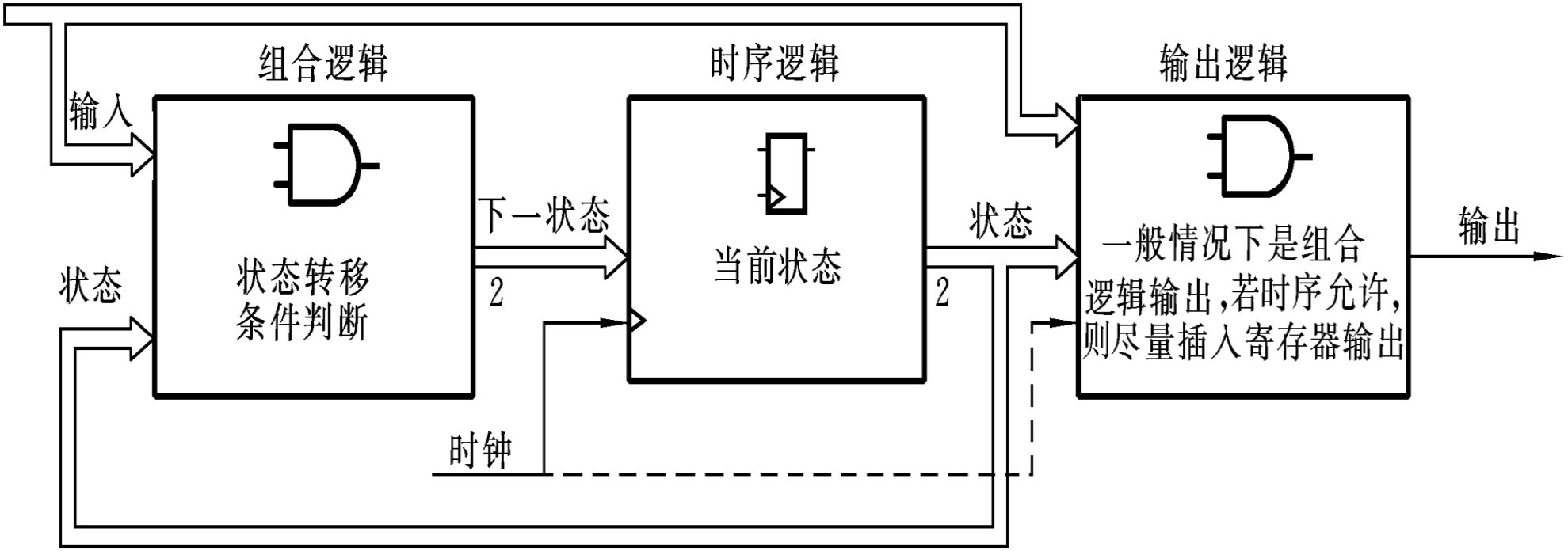
5.2.2 推荐的状态图描述方法

```
S3: if (Sin==1)
                begin Out =1'b1; Next_state = S2; end
                 else
                        begin Out =1'b0; Next_state = S1; end

        endcase
        end
endmodule
```

用两个always块描述状态机的写法是值得推荐的方法之一，两个always块写法的电路结构框图可以用图4.2.3进行概括。

5.2.2 推荐的状态图描述方法



5.2.2 推荐的状态图描述方法

另外，还可以使用三个always块分别描述，即第一个always模块采用同步时序逻辑方式描述状态转移（中间方框），第二个always模块采用组合逻辑方式描述状态转移规律（第一个方框），第三个always模块描述电路的输出信号，在时序允许的情况下，通常让输出信号经过一个寄存器再输出，保证输出信号中没有毛刺。

用三个always块对该例的状态机进行描述，其代码如下：

```
module  Detector3 ( Sin, CP, nCR, Out) ;
        input Sin, CP, nCR; //定义输入变量
        output Out ; //定义输出变量
        reg Out;
        reg [ 1 : 0 ] Current_state, Next_state;
        parameter  S0=2'b00, S1=2'b01, S2 = 2'b10, S3 = 2'b11; //状态转换，时序逻辑
```

5.2.2 推荐的状态图描述方法

```
always @(posedge CP or negedge nCR )
begin
        if (~nCR)
                Current_state <= S0;  //异步清零
        else
                Current_state <= next_state; //在CP上升沿触发
器状态翻转
end
//下一状态产生，组合逻辑
always @( Current_state or Sin)
        begin
Next_state =2'bxx;
case(Current_state )
S0: begin Next_state = (Sin==1)? S0 : S1; end
```

5.2.2 推荐的状态图描述方法

```
S1: begin Next_state = (Sin==1)? S2 : S1; end
          S2: begin Next_state = (Sin==1)? S0 : S3; end
 S3: if (Sin==1)
                  begin Next_state = S2; end
                   else
                           begin Next_state = S1; end
        endcase
        end
//输出: 让输出信号经过一个寄存器再输出，可以消除Out信号中的毛刺
always @(posedge CP or negedge nCR )
begin
        if (~nCR)  Out=1'b 0;
else begin
```

5.2.2 推荐的状态图描述方法

```
Out=1'b 0;
case(Current_state )
            S0, S1, S2:  Out =1'b0;
    S3: if (Sin==1)  Out =1'b1;
                    else  Out =1'b0;
        endcase
        end
end
endmodule
```

5.3 状态机设计中的关键技术

5.3.1状态编码

在使用Verilog HDL描述状态机时，通常用参数定义语句**parameter**指定状态编码。状态编码方案一般有三种：自然二进制编码、格雷（Gray）编码和one-hot编码。对应于图4.2.1（b）所示的状态图的各种编码方案如表4.3.1所示。

表 4.3.1 有限状态机的编码方案

二进制编码	格雷编码	one-hot 编码
S0=2′b00	S0=2′b 00	S0=4′b 0001
S1=2′b 01	S1=2′b 01	S1=4′b 0010
S2=2′b 10	S2=2′b 11	S2=4′b 0100
S3=2′b 11	S3=2′b 10	S3=4′b 1000

5.3.1状态编码

由表可知，格雷码的特点是当前状态改变时，状态向量中仅一位发生变化，因此当系统的状态变化是基于异步的输入信号时，格雷编码能够避免进入错误的状态。而one-hot编码的特点是：状态数等于触发器的数目，冗余的触发器带来的好处是译码电路的简单化，因此它的速度非常快，此外由于FPGA器件内部触发器的数量是固定的且比较丰富，所以one-hot编码非常适合于FPGA设计。不管使用哪种编码，状态机中的各个状态都应该使用符号常量，而不应该直接使用编码数值，赋予各状态有意义的名字对于设计的验证和代码的可读性都是有益的。

5.3.2如何消除输出端产生的毛刺

前面介绍的普通状态机由组合逻辑电路决定电路的输出当组合逻辑较大时，若状态触发器的值发生变化或者输入信号发生变化，由于各信号在组合逻辑内部经过的路径不一样，就容易在输出端产生毛刺。

下面介绍两种常用消除毛刺的方法：

1. 具有流水线输出的Mealy状态机

为了消除毛刺，可以在普通Mealy的输出逻辑后加一组输出寄存器，将寄存器的输出值作为输出向量，这种Mealy状态机的等效方框如图5.3.1所示。

5.3.2如何消除输出端产生的毛刺

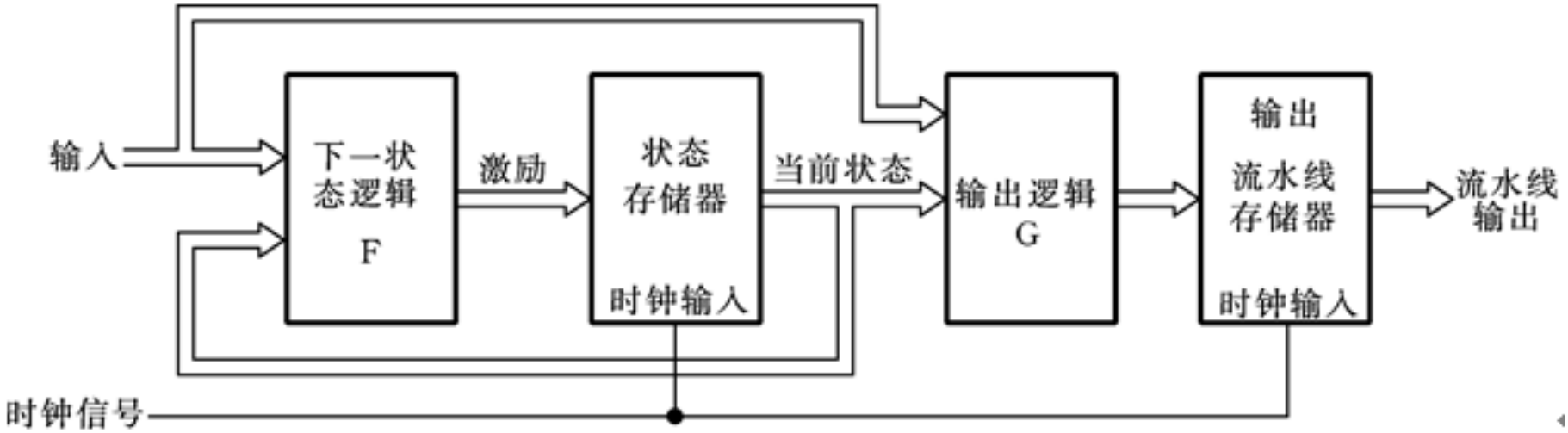
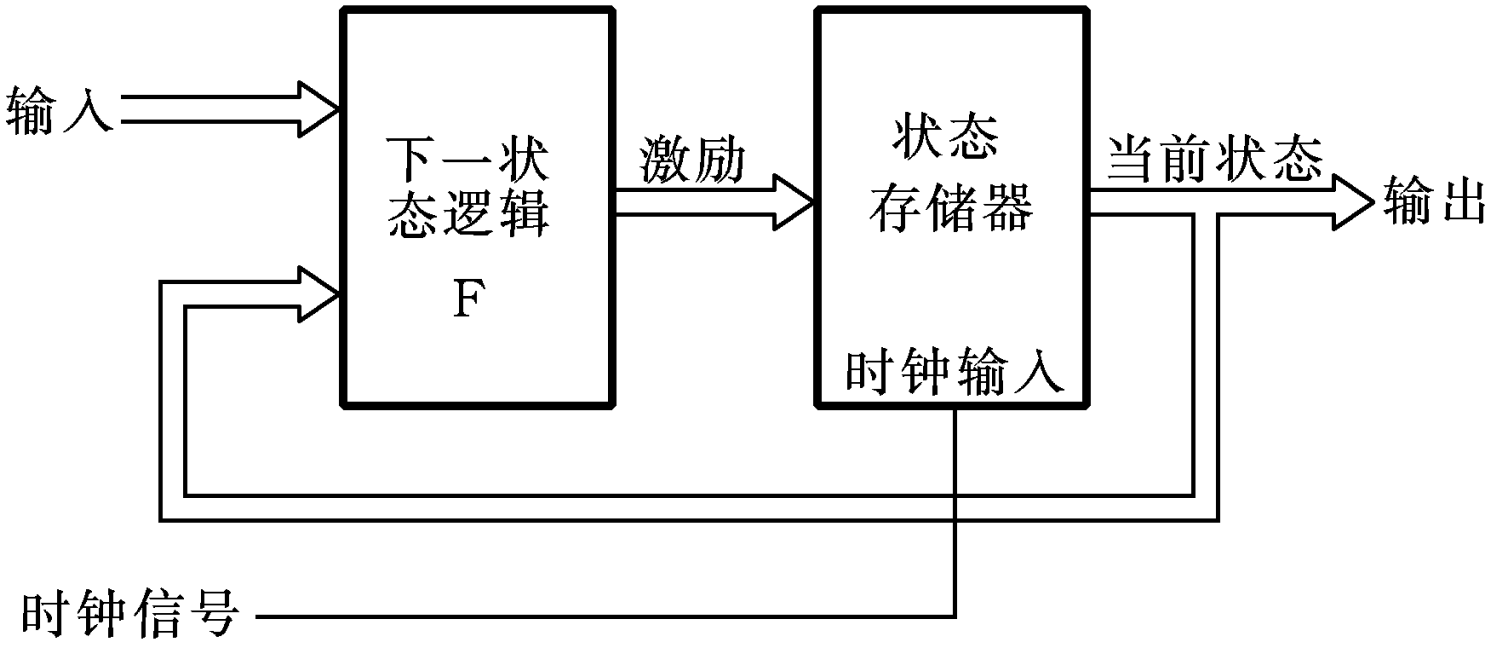


图5.3.1 具有流水线输出的Mealy 状态机

5.3.2如何消除输出端产生的毛刺

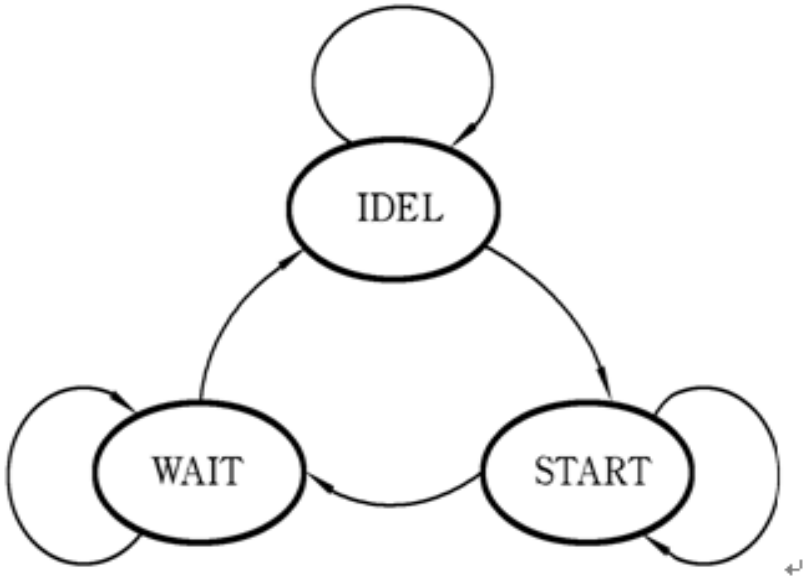
2. 在状态位里编码输出的Moore状态机

这种方法的指导思想是将状态寄存器和输出向量统一进行编码，即将状态位本身作为输出信号，其等效状态框图如下所示。



5.3.2如何消除输出端产生的毛刺

下面以图5.3.3所示的状态图说明在状态位里编码输出的方法。图中，状态机共有三个状态：IDLE，START和WAIT，输入信号为：input_1，input_2，input_3，input_4。这些输入信号的不同逻辑组合就构成了状态之间跳转的条件。



状态转换条件如下：

当前状态 IDEL:

{output_1,output_2}=2'b00

if (input1 && input2): IDLE→START

else: IDLE→IDLE;

当前状态 START:

{output_1,output_2}=2'b01

if (input3): START→WAIT

else: START→START;

当前状态 WAIT:

{output_1,output_2}=2'b11

if (input4): WAIT→IDLE

else: WAIT→WAIT;

5.3.2如何消除输出端产生的毛刺

该状态机需要控制两个输出信号：output_1和output_2。可以采用4bits的状态编码，其中高两位表示当前的状态，末尾两位控制output_1和output_2的输出。IDLE状态编码为4'b0000，START状态编码为4'b0101，WAIT状态编码为4'b1011。

参考程序如下所示：

```
module  FSM1 (nRST, CP, input_1, input_2, input_3, input_4,
output_1, output_2) ;
        input input_1, input_2, input_3, input_4; //定义输入变
量
        input nRST, CP;
output output_1, output_2 ; //定义输出变量
        wire output_1, output_2;
        reg [ 3 : 0 ] Current_state, Next_state;
```

5.3.2如何消除输出端产生的毛刺

```
//状态参量的定义，根据前面所述的全译码状态编码
parameter [ 3 : 0 ] IDLE = 4'b0000,  START = 4'b0101, WAIT = 4'b1011;
//The first always block, sequential state transition
always @(posedge CP or negedge nRST)
if(!nRST)  //当系统复位时，状态寄存器置为IDLE
        Current_state <= IDLE;
    else  //状态寄存器进行状态存储，将下一状态存储到状态寄存器成为当前状态
        Current_state <= Next_state;
// The second always block, combinational condition judgment
always @(input_1 or input_2 or input_3 or input_4)
    case(Current_state) //根据当前状态和状态转换条件进行译码
```

5.3.2如何消除输出端产生的毛刺

```
 IDLE:
        if(input_1 && input_2)  Next_state = START;
        else  Next_state = IDLE;
      START:
        if(input_3)  Next_state = WAIT;
        else  Next_state = START;
      WAIT:
         if(input_4)  Next_state = IDLE;
         else Next_state          = WAIT;
      default:  Next_state = IDLE;
endcase
//状态机的输出逻辑
assign  output_1 = Current_state[0];
assign  output_2 = Current_state[1];
endmodule
```

5.3.3如何使用One-hot编码方案设计状态机

对状态机的各个状态赋予一组特定的二进制数称为状态编码。比较常用的有自然二进制码、格雷码和One-hot编码。自然二进制码和格雷码的编码方案使用的触发器较少，其编码效率较高，但负责根据当前状态和状态转换条件进行译码的组合电路会比较复杂，其逻辑规模也较大，使得次态逻辑在传输过程中需要经过多级逻辑，从而影响电路的工作速度。

One-hot编码方案使用n位状态触发器表示具有n个状态的状态机，每个状态与一个独立的触发器相对应，并且在任何时刻其中只有一个触发器有效（其值为1）。虽然这种方案会使用较多的触发器，但它的编码方式非常简单，可有效地简化组合电路，并换得工作可靠性和工作速度的提高。在大规模可编程逻辑器件如FPGA中，触发器数量较多而门逻辑相对较少，One-hot编码方案有时反而更有利于提高器件资源的利用率。

5.3.3如何使用One-hot编码方案设计状态机

2. 两个always块描述状态机的方法（推荐写法）

用两个always块对该例的状态机进行描述，其代码如下：

```
module  Detector2 ( Sin, CP, nCR, Out) ;
        input Sin, CP, nCR; //定义输入变量
        output Out ; //定义输出变量
        reg Out;
        reg [ 1 : 0 ] Current_state, Next_state;
parameter  S0=2'b00, S1=2'b01, S2 = 2'b10, S3 = 2'b11;
//状态转换，时序逻辑
always @(posedge CP or negedge nCR )
        begin
        if (~nCR)
                Current_state <= S0;  //异步清零
        else
```

5.3.3如何使用One-hot编码方案设计状态机

定义当前状态向量state为一个5-bits向量，末尾的两位表示状态机输出，state[2]为1表示状态IDLE，state[3]为1表示状态START，state[4]为1表示状态WAIT。

下面是基于One-Hot编码方式的状态机实现代码：

```
module  FSM2 (nRST, CP, input_1, input_2, input_3, input_4,
output_1, output_2) ;
        input input_1, input_2, input_3, input_4; //定义输入变量
        input nRST, CP;
output output_1, output_2 ; //定义输出变量
        wire output_1, output_2;
        reg [ 4 : 0 ] state, Next_state;
parameter [ 4 : 0 ] IDLE  = 5'b001_00, //状态参量，末尾两位表示对应的输出
```

5.3.3如何使用One-hot编码方案设计状态机

```
 START = 5'b010_01,
WAIT  = 5'b100_11;
parameter [2:0] IDLE_POS  = 3'd2, //状态对应在state中的表示位置,One-hot
START_POS = 3'd3, // START POSition
WAIT_POS  = 3'd4;
always @(posedge CP or negedge nRST) //状态存储
if(!nRST)  state <= IDLE;
else  state <= Next_state;
always @(input_1 or input_2 or input_3 or input_4)       //状态转移逻辑
begin
    Next_state = IDLE; //设置初态
    case(1'b1) //One-Hot编码实现状态转移时,每次取state的一
```

5.3.3如何使用One-hot编码方案设计状态机

位与1比较

```
state[IDLE_POS]:
  if(input_1 && input_2)
      Next_state = START;
  else  Next_state = IDLE;
state[START_POS]:
  if(input_3)
      Next_state = WAIT;
  else  Next_state = START;
state[WAIT_POS]:
   if(input_4)
```

5.3.3如何使用One-hot编码方案设计状态机

```
 Next_state = IDLE;
          else Next_state = WAIT;
       default: Next_state = IDLE;
endcase
end
//状态机的输出逻辑
assign  output_1 = state[0];
assign  output_2 = state[1];
endmodule
```

One-hot编码特点：指定各个状态在状态编码中的表示位，采用参量定义方式指定One-hot状态编码；使用always语句描述状态寄存器的状态存储；使用敏感表和case语句描述状态转换逻辑，在case语句中只采用一位寄存器比较方式；使用assign语句描述状态编码控制的状态机输出。

5.4状态机设计举例

5.4.1汽车尾灯控制电路设计

本节重点介绍构造状态图的两种方法：一是试探法，二是基于算法状态机构造状态图的方法。

例5.4.1 汽车尾灯发出的信号主要是给后面行驶汽车的司机看的，通常汽车驾驶室有刹车开关（HAZ）、左转弯开关(LEFT)和右转弯开关(RIGHT)，司机通过操作这3个开关给出车辆的行驶状态。假设在汽车尾部左、右两侧各有3个指示灯，分别用LA、LB、LC、RA、RB、RC表示，如图5.4.1所示。这些灯的亮、灭规律如下：

（1）汽车正常行驶时，尾部两侧的6个灯全部熄灭。

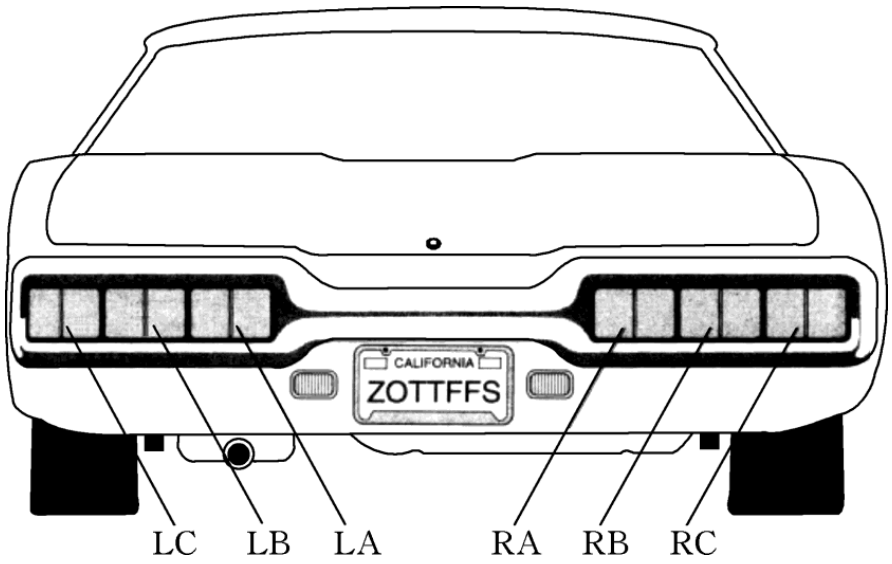
（2）刹车时，汽车尾灯工作在告警状态，所有6个灯按一定频率闪烁。

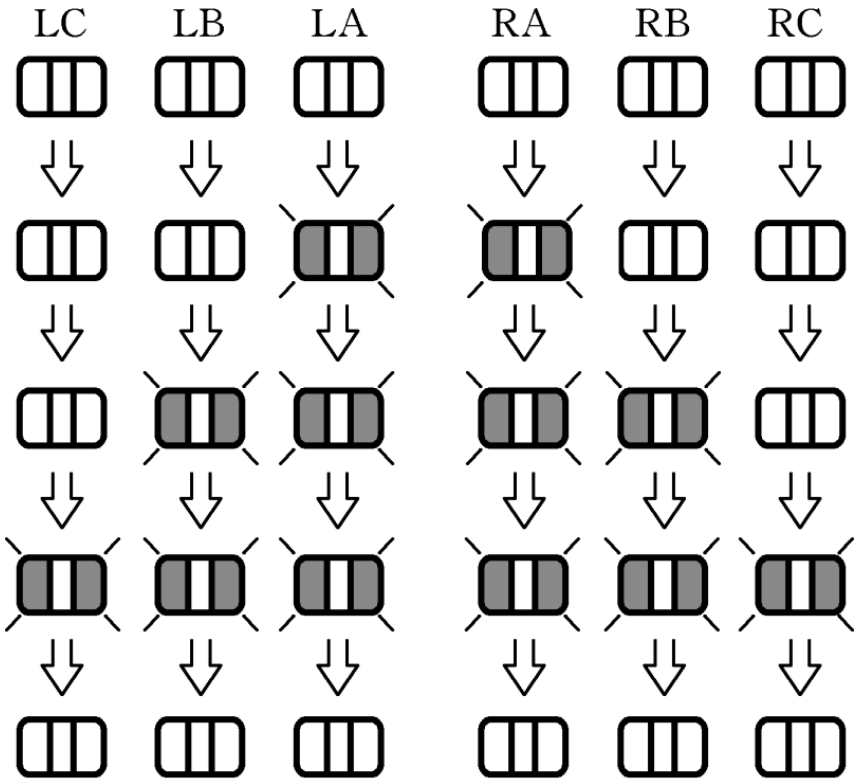
（3）左转弯时，左侧3个灯轮流顺序点亮，其规律如图5.4.2（a）所示，右侧灯全灭。

4.4.1汽车尾灯控制电路设计

（4）右转弯时，右侧3个灯轮流顺序点亮，其规律如图5.4.2（b）所示，左侧灯全灭。

假设电路的输入时钟信号为CP，CP的频率对于汽车尾灯所要求的闪烁频率。试根据上述要求设计出一个时钟同步的状态机来控制汽车的尾灯。





（a）　（b）

5.4.1汽车尾灯控制电路设计

（1）画出原始状态图。选择Moore机设计该电路，则尾灯的亮、灭直接由状态译码就可以得到。由设计要求可知：汽车左转弯时，右边的灯不亮而左边的灯依次循环点亮，即0个、1个、2个或3个灯亮，分别用L0、L1、L2、L3表示，状态机在4个状态中循环。同理，汽车右转弯时，状态机也会在4个状态中循环，即左边灯不亮而右边的灯有0个、1个、2个或3个灯亮，分别用R0、R1、R2、R3表示。由于L0和R0都表示6个灯不亮，所以合起来用IDLE表示。可得原始的状态图就画出来了，如图5.4.3所示。

分析一下图5.4.3，就会发现一个没有考虑到的实际问题，即如果多个输入同时有效，状态机如何工作呢？图5.4.4解决了多个输入同时有效的问题，并将LEFT和RIGHT同时有效的情况处理成告警状态。经过改进且具有这一特性的状态图如图5.4.5所示。

5.4.1汽车尾灯控制电路设计

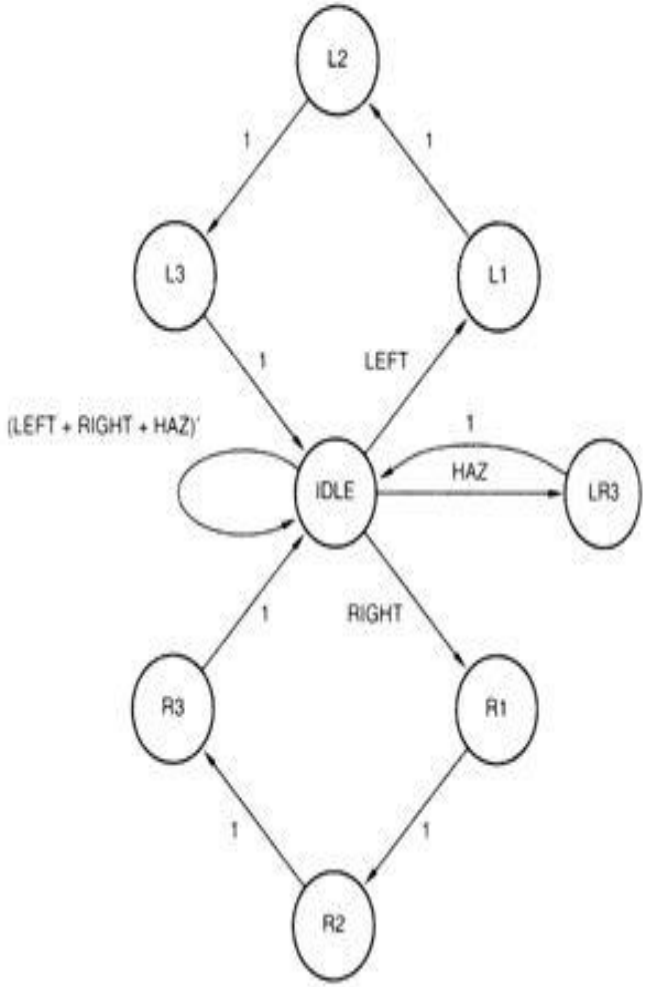


图4.4.3 有错误的汽车尾灯状态图

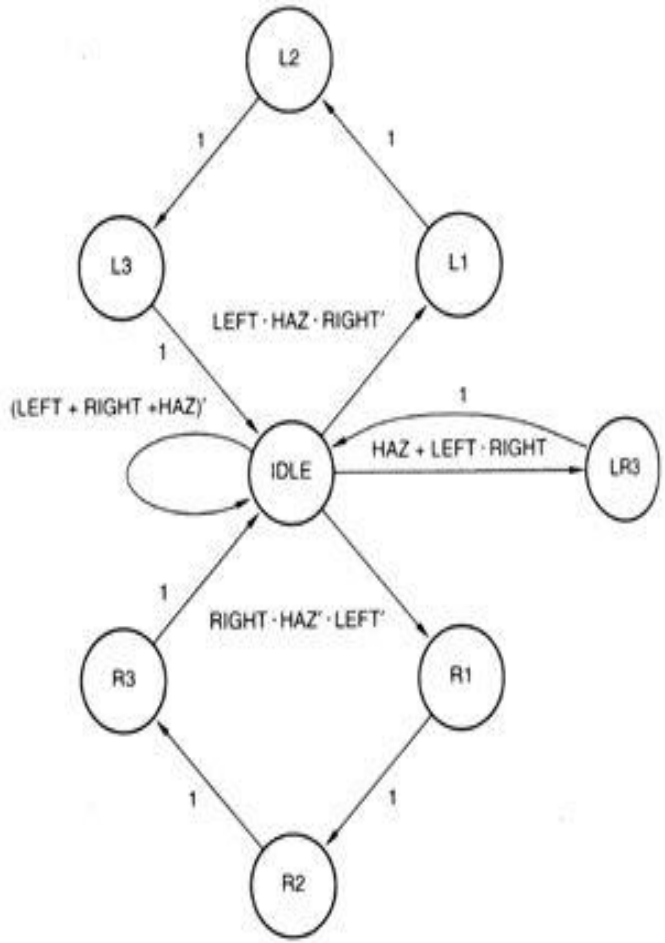


图4.4.4正确的汽车尾灯状态图

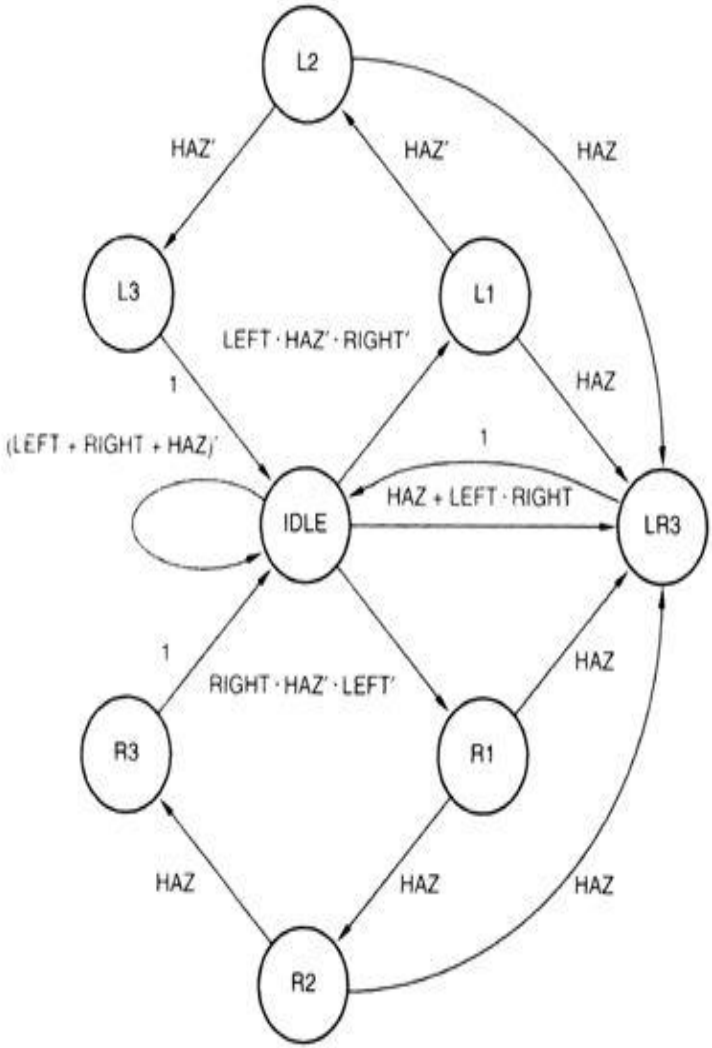


图4.4.5 改进后的汽车尾灯状态图

5.4.1汽车尾灯控制电路设计

（2）列出电路的输出。由于电路的输出信号较多，不便于写在状态图中，所以单独列出输出逻辑真值表，如表5.4.1所示。

表 5.4.1 汽车尾灯输出逻辑真值表

状 态	LC	LB	LA	RA	RB	RC
IDLE	0	0	0	0	0	0
L_1	0	0	1	0	0	0
L_2	0	1	1	0	0	0
L_3	1	1	1	0	0	0
R_1	0	0	0	1	0	0
R_2	0	0	0	1	1	0
R_3	0	0	0	1	1	1
LR_3	1	1	1	1	1	1

5.4.1汽车尾灯控制电路设计

（3）选择一种编码方案，对上述状态图进行状态分配，然后用Verilog描述状态图和输出逻辑。

状态图完成后，必须进行完备性和互斥性的检查。

①完备性的检查方法是：对于每一个状态，将所有脱离这一状态的条件表达式进行逻辑或运算，如果结果为1就是完备的。否则不完备，也就是说状态图进入某状态后，却不能跳出该状态。

②互斥性的检查方法是：对于每一个状态，将所有脱离这一状态的条件表达式找出来，然后任意两个表达式进行逻辑与运算，如果结果为0就是互斥的。也就是要保证在任何时候不会同时激活两个脱离状态的转换，即从一个状态跳到两个状态。

5.4.2应用算法状态机设计十字路口交通灯控制电路

算法状态机ASM（Algorithmic state Machine）图是描述数字系统控制算法的流程图。应用ASM图设计数字系统，可以很容易将语言描述的设计问题变成时序流程图的描述，根据时序流程图就可以得到电路的状态图和输出函数，从而得出相应的硬件电路。

1. ASM图的状态框、判断框和输出框，ASM图中有三种基本的符号，即状态框、判断框和输出框。数字系统控制序列中的状态用状态框表示，如图4.4.6（a）所示。图4.4.6（b）为状态框实例。图4.4.6中的箭头表示系统状态的流向。

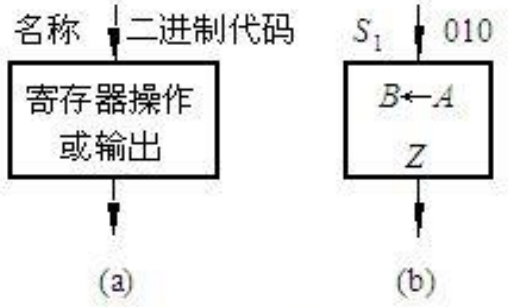


图 4.4.6　状态框与实例

(a) 状态框　(b) 状态框实例

5.4.2应用算法状态机设计十字路口交通灯控制电路

判断框表示状态变量对控制器工作的影响，如图4.4.7所示：

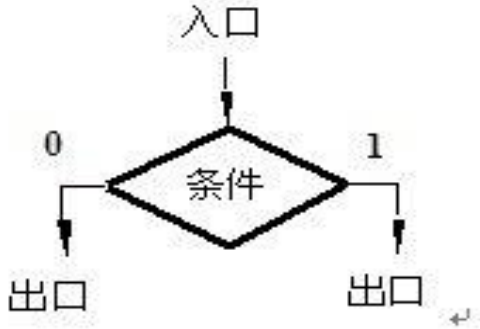


图 4.4.7　判断框

条件输出框如下图所示，条件框的入口必定与判断框的输出相连。

(a)

5.4.2应用算法状态机设计十字路口交通灯控制电路

2. ASM图中各种逻辑框之间的时间关系

从表面上来看ASM图与程序流程图很相似，但实际上有很大的差异。程序流程图只表示事件发生的先后顺序，没有时间概念，而ASM图则不同，它表示事件的精确时间间隔顺序。在ASM图中每一个状态框表示一个时钟周期内的系统状态，状态框和与之相连的判断框，条件输出框所规定的操作，都是在一个共同的时钟周期实现的，同时系统的控制器从现在状态（现态）转移到下一个状态（次态）。因此，可以很容易将图4.4.9（a）所示的ASM图，转换成状态图，如图4.4.9（b）所示，其中E和F为状态转换条件。与ASM图不同，状态图无法表示寄存器操作。

图4.4.9（c）给出了ASM图的各种操作及状态转换的时间图。

5.4.2应用算法状态机设计十字路口交通灯控制电路

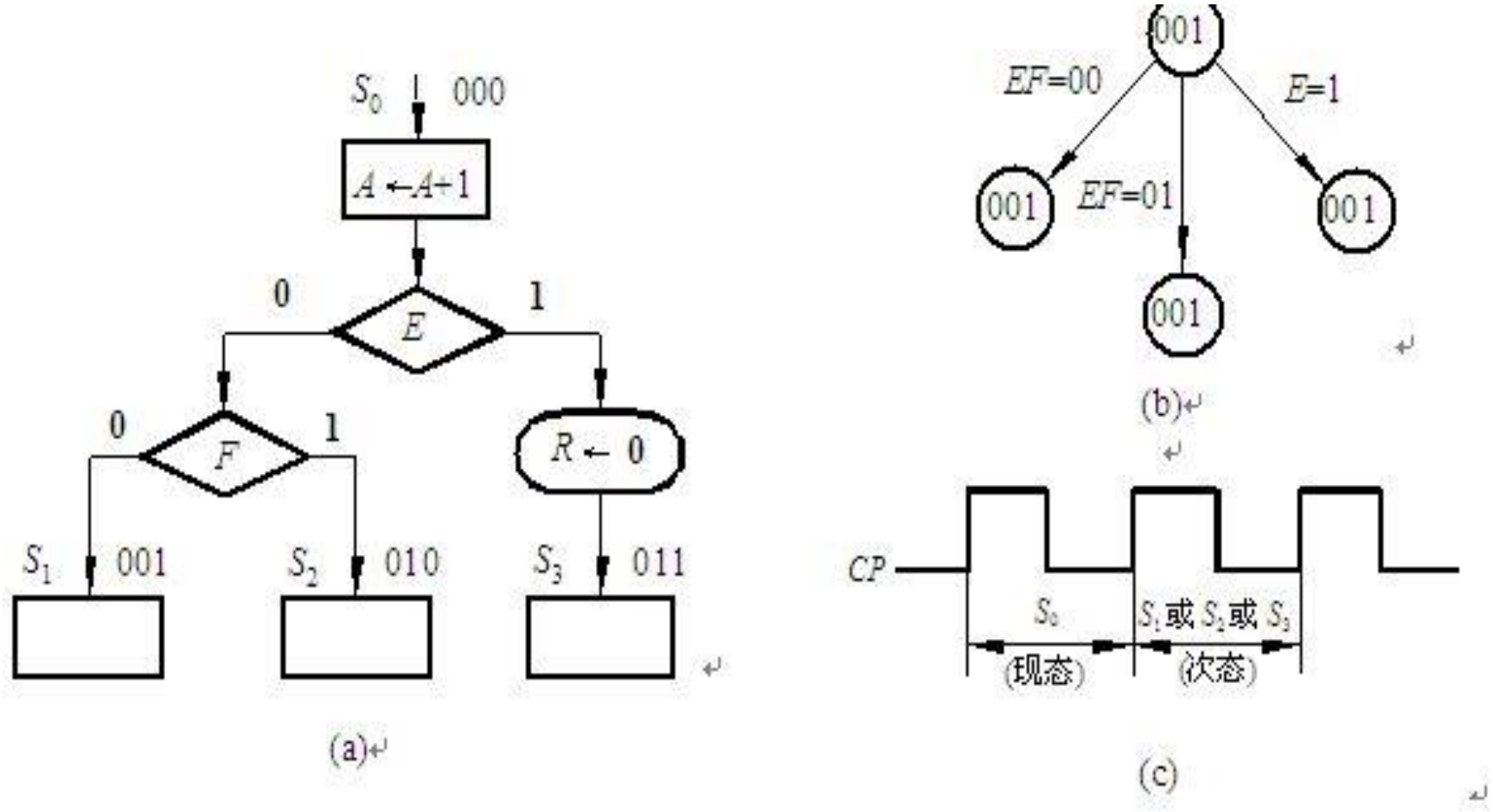


图 4.4.9 ASM 图、状态图及状态的转换关系

(a) ASM 图 (b) 状态图 (c) 状态转换时间图

5.4.2应用算法状态机设计十字路口交通灯控制电路

3. 十字路口交通灯控制电路设计举例

例4.4.2 图4.4.10表示位于主干道和支干道的十字路口交通灯系统，支干道两边安装有传感器S，试设计一个主干道和支干道十字路口的交通灯控制电路，其技术要求如下：（1）一般情况下，保持主干道畅通，主干道绿灯亮、支干道红灯亮，并且主干道绿灯亮的时间不得少于60秒。

（2）主干道车辆通行时间已经达到60秒，且支干道有车时，则主干道红灯亮、支干道绿灯亮，但支干道绿灯亮的时间不得超过30秒。

（3）每次主干道或支干道绿灯变红灯时，黄灯先亮5秒钟。

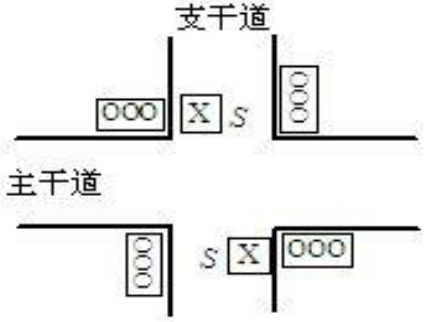


图 4.4.10　交通灯示意图

5.4.2应用算法状态机设计十字路口交通灯控制电路

（1）明确系统的功能，进行逻辑抽象

（2）确定系统方案并画出ASM图

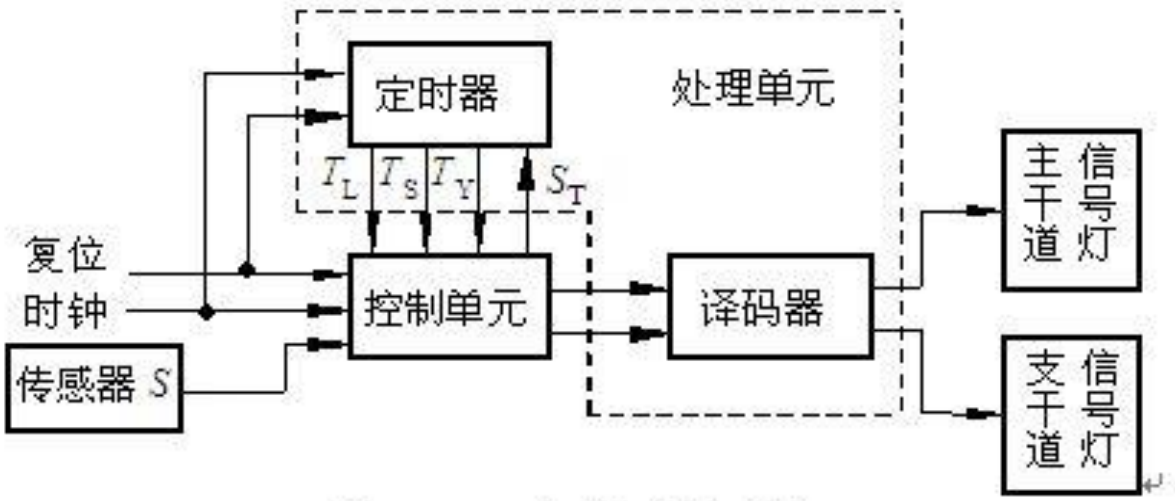


图 4.4.11　交通灯系统框图

交通灯控制单元的控制过程分为四个阶段，对应的输出有四种状态，分别用S_0, S_1, S_2和S_3表示：

S_0状态：主干道绿灯亮支干道红灯亮，此时若支干道有车等待通过，而且主干道绿灯已亮足规定的时间间隔T_L，控制器发出状态转换信号S_T，输出从状态S_0转换到S_1。

5.4.2应用算法状态机设计十字路口交通灯控制电路

S1状态：主干道黄灯亮，支干道红灯亮，进入此状态，黄灯亮足规定的时间间隔TY时，控制器发出状态转换信号ST，输出从状态S1转换到S2。

S2状态：支干道绿灯亮，主干道红灯亮，若此时支干道继续有车，则继续保持此状态，但支干道绿灯亮的时间不得超过TS时间间隔，否则控制单元发出状态转换信号ST，使输出转换到S3状态。

S3状态：支干道黄灯亮，主干道红灯亮，此时状态与S1状态持续的时间间隔相同，均为TY，时间到时，控制器发出ST信号，输出从状态S3回到S0状态。

对上述S0、S1、S2和S3四种状态按照格雷码进行编码分别为00，01，11和10，由此得到交通灯控制单元的ASM图如图4.4.12所示。依此类推得出4.4.12所示的ASM的图。

5.4.2应用算法状态机设计十字路口交通灯控制电路

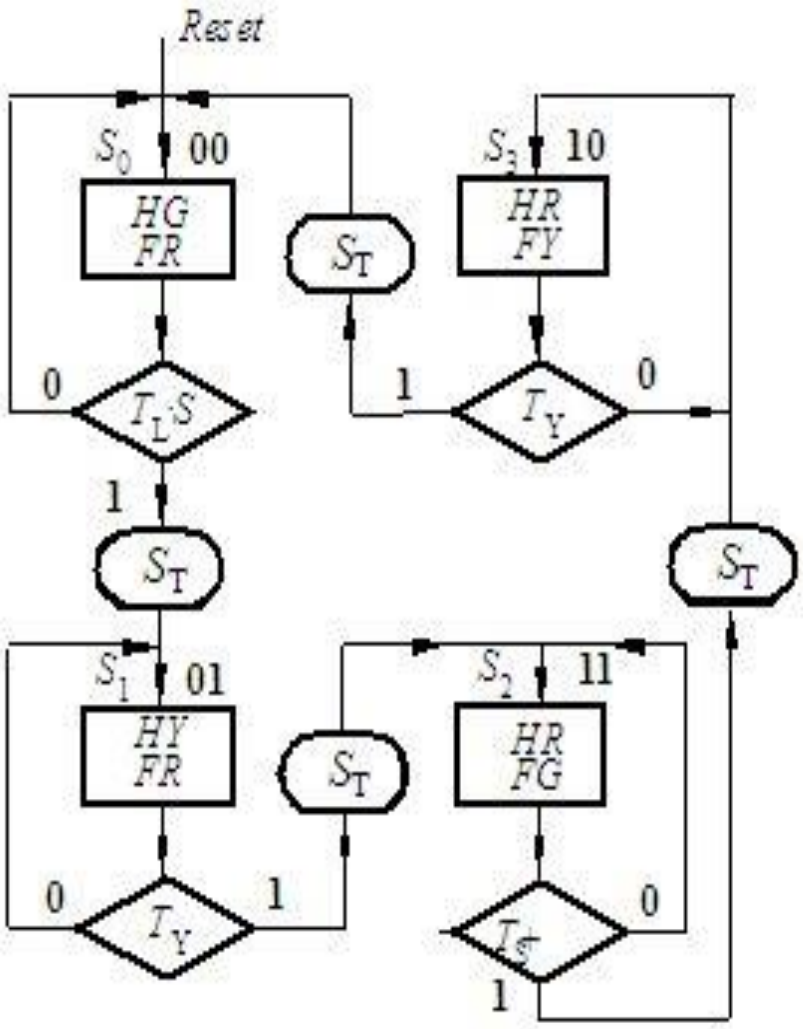


图 4.4.12　交通灯控制单元 ASM 图

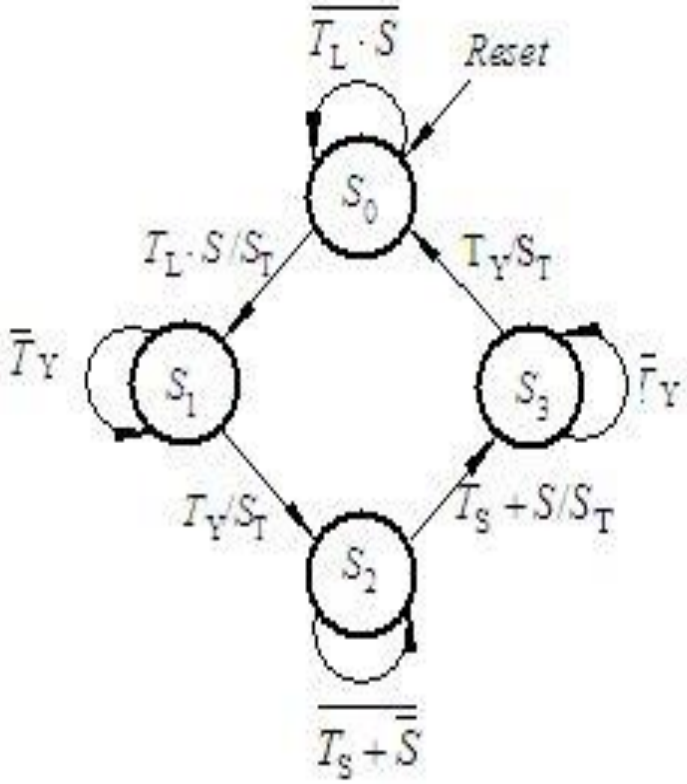


图 4.4.13　交通灯控制单元状态图

5.4.2应用算法状态机设计十字路口交通灯控制电路

（3）交通灯控制器各功能模块电路的框架设计

通过分析交通灯控制电路的要求可知，系统主要由传感器、时钟脉冲产生器、定时器、控制器及译码器构成，传感器S在有车辆通过时发出一个高电平信号。

①设计控制器

根据交通灯控制单元的ASM图，得出其状态图如图4.4.13所示。ASM图中的状态框与状态图中的状态相对应，判断框中的条件是状态转换的输入条件，条件输出框与控制单元状态转换的输出相对应。状态图是描述状态之间的转换。

②设计定时器

定时器由与系统秒脉冲同步的计数器构成，时钟脉冲上升沿到来时，在控制信号S_T作用下，计数器从零开始计数，并向控制器提供模M5、M30和M60信号，即T_Y、T_S和T_L定时时间信号。

5.4.2应用算法状态机设计十字路口交通灯控制电路

③设计译码器

当交通灯控制电路处于不同工作状态时，交通信号灯按一定的规律与之对应。

（4）用Verilog HDL描述交通灯控制电路

根据以上设计思路，可以写出交通灯控制电路的Verilog HDL代码如下：

```
//------------- controller.v -------------
// Traffic Signal Controller
//State definition     HighWay  Country
`define S0   2'b00  //GREENRED，采用宏定义方式给出状态编码
`define S1   2'b01  //YELLOW       RED，不建议采用此方法
`define S2   2'b11  //RED          GREEN
`define S3   2'b10  //RED          YELLOW
```

5.4.2应用算法状态机设计十字路口交通灯控制电路

```
module controller (CLK, S, RESET, HG, HY, HR, FG, FY, FR,
TimerH, TimerL);
//I/O ports
input CLK, S, RESET;  //if S=1, indicates that there is car on
the country road
output HG, HY, HR, FG, FY, FR; //declared output signals
are registers
reg    HG, HY, HR, FG, FY, FR;
output [3:0] TimerH;
output [3:0] TimerL;
reg    [3:0] TimerH, TimerL;
//Internal state variables
wire Tl, Ts, Ty; //timer output signals
reg St;          //state translate signal
reg [1:0] CurrentState, NextState;//FSM state register
```

5.4.2应用算法状态机设计十字路口交通灯控制电路

```
/*====== Descrition of the timer block ======*/
always @(posedge CLK or negedge RESET )
begin:counter
if (~RESET)  {TimerH, TimerL} = 8'b0;
else if (St) {TimerH, TimerL} = 8'b0;
else if ((TimerH == 5) & (TimerL == 9))
begin   {TimerH, TimerL} = {TimerH, TimerL}; end
else if (TimerL == 9)
begin TimerH = TimerH + 1;  TimerL = 0; end
else begin TimerH = TimerH; TimerL = TimerL + 1; end
end  // BCD counter
assign  Ty = (TimerH==0)&(TimerL==4);
assign  Ts = (TimerH==2)&(TimerL==9);
assign  Tl = (TimerH==5)&(TimerL==9);
```

5.4.2应用算法状态机设计十字路口交通灯控制电路

```
/*====== Descrition of the signal controller block ======*/
//FSM register:State change only at positive edge of clock
always @(posedge CLK or negedge RESET )
begin:statereg
if (~RESET)    //Signal controller starts in S0 state
          CurrentState <= `S0;
else  CurrentState <= NextState;
end //statereg
// FSM combinational block:state machine using case
statements
always @(S or CurrentState or Tl or Ts or Ty )
begin: fsm
          case(CurrentState)
                `S0: begin     //S0是用define定义的，在引用时
要加右撇号
```

5.4.2应用算法状态机设计十字路口交通灯控制电路

```
NextState = (Tl && S) ? `S1 :`S0;
                                    St = (Tl && S) ? 1:0;
                        end
            `S1: begin
                                    NextState = (Ty) ? `S2 :`S1;
                                    St = (Ty) ? 1:0;
                        end
            `S2: begin
                                    NextState = (Ts || ~S) ? `S3 :`S2;
                                    St = (Ts || ~S) ? 1:0;
                        end
            `S3: begin
                                    NextState = (Ty) ? `S0 :`S3;
                                    St = (Ty) ? 1:0;
```

5.4.2应用算法状态机设计十字路口交通灯控制电路

```
 end
            endcase
end  //fsm
/*====== Description of the decoder block ======*/
//Compute values of main signal and country signal
always @(CurrentState)
begin
          case (CurrentState)
                    `S0: begin
          {HG, HY, HR} = 3'b100; //Highway signal is green
          {FG, FY, FR} = 3'b001; //Country signal is red
End
`S1: begin
           {HG, HY, HR} = 3'b010; //Highway signal is yellow
```

5.4.2应用算法状态机设计十字路口交通灯控制电路

```
{FG, FY, FR} = 3'b001; //Country signal is red
                    end
                `S2: begin
        {HG, HY, HR} = 3'b001; //Highway signal is red
        {FG, FY, FR} = 3'b100; //Country signal is green
                        end
                `S3: begin
        {HG, HY, HR} = 3'b001; //Highway signal is red
        {FG, FY, FR} = 3'b010; //Country signal is yellow
                         end
          endcase
end
endmodule
```