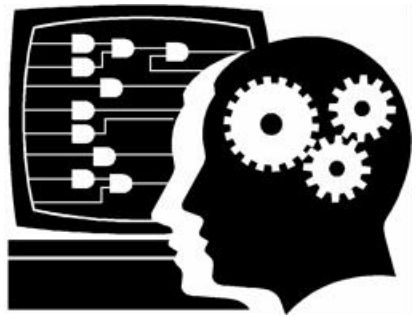
数字系统设计与Verilog HDL

（第6版）



第8章 Verilog有限状态机设计

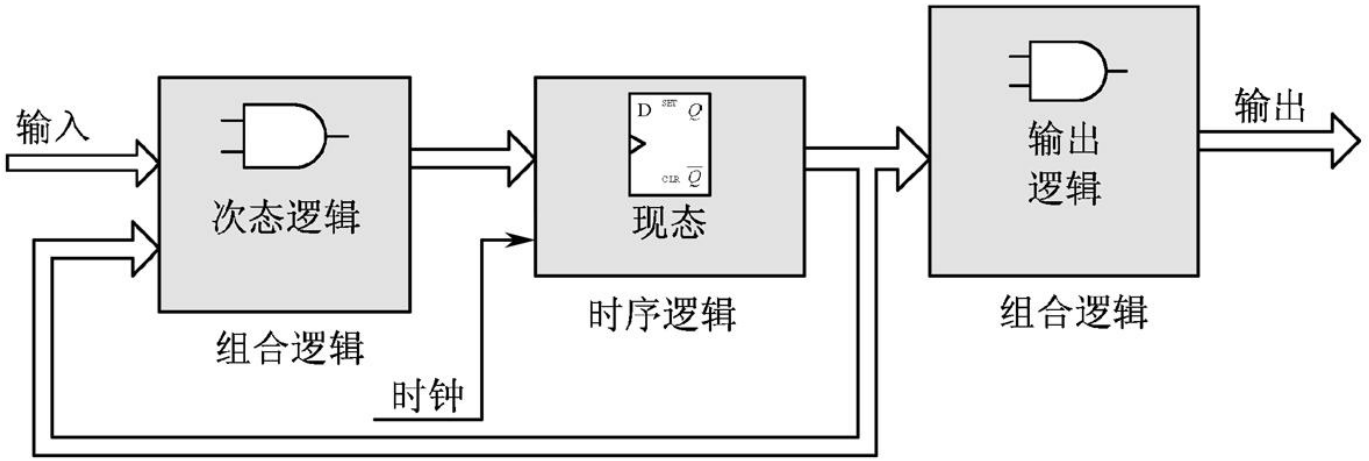
8.1 有限状态机

8.2 有限状态机的Verilog描述

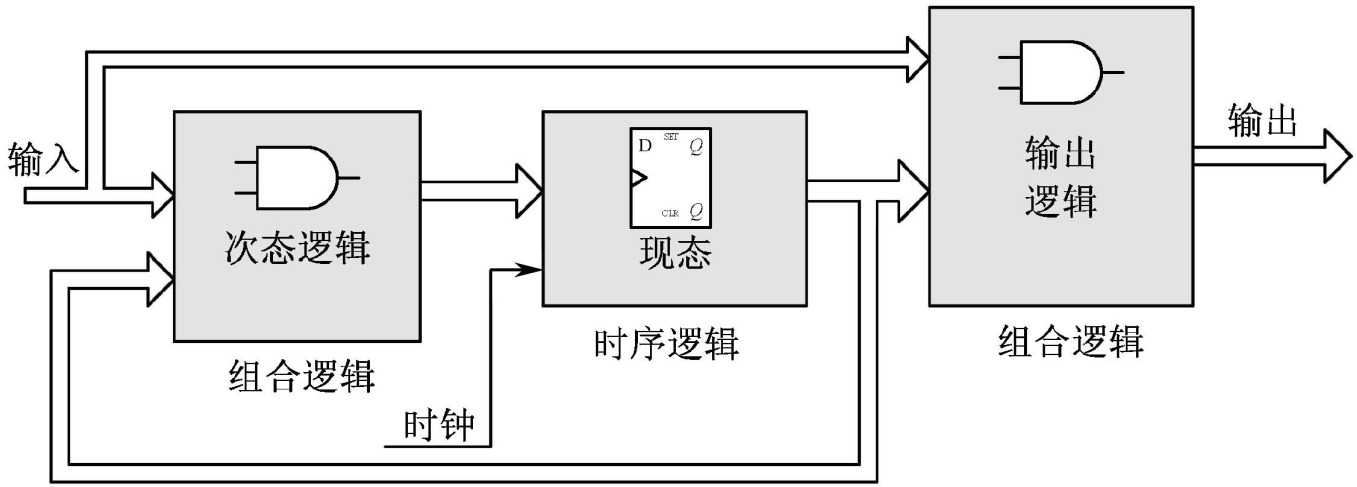
8.3 状态编码

8.4 有限状态机设计要点

8.1 有限状态机



摩尔型（Moore）状态机



米里型（Mealy）状态机

用状态机设计模5计数器

```
module fsm(clk,clr,z,qout);
input clk,clr; output reg z; output reg[2:0] qout;
always @(posedge clk or posedge clr) //此过程定义状态转换
begin if(clr) qout<=0;                //异步复位
      else  case(qout)
      3'b000: qout<=3'b001;
      3'b001: qout<=3'b010;
      3'b010: qout<=3'b011;
      3'b011: qout<=3'b100;
      3'b100: qout<=3'b000;
      default: qout<=3'b000;   /*default语句*/
      endcase
end
always @(qout)                 /*此过程产生输出逻辑*/
begin  case(qout)
      3'b100: z=1'b1;
      default:z=1'b0;
endcase
end
endmodule
```

8.2 有限状态机的几种描述方式

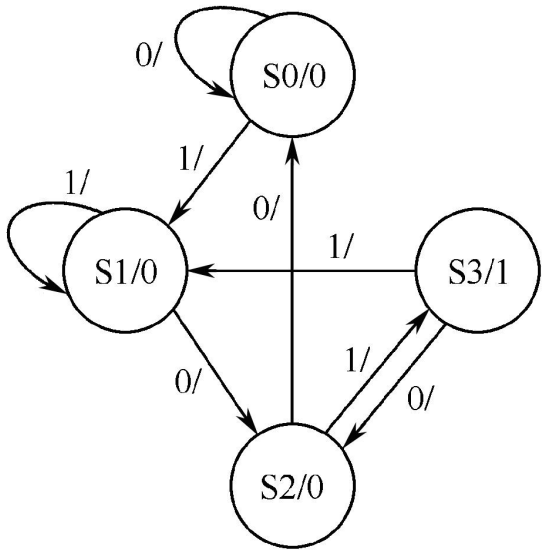
（1）用三个过程描述：即现态（CS）、次态（NS）、输出逻辑（OL）各用一个always过程描述。

（2）双过程描述（CS+NS、OL双过程描述）：使用两个always过程来描述有限状态机，一个过程描述现态和次态时序逻辑（CS+NS）；另一个过程描述输出逻辑（OL）。

（3）双过程描述（CS、NS+OL双过程描述）：一个过程用来描述现态（CS）；另一个过程描述次态和输出逻辑（NS+OL）。

（4）单过程描述：在单过程描述方式中，将状态机的现态、次态和输出逻辑（CS+NS+OL）放在一个always过程中进行描述。

```
module fsm1_seq101(clk,clr,x,z);
input clk,clr,x; output reg z; reg[1:0] state,next_state;
parameter   S0=2'b00,S1=2'b01,S2=2'b11,S3=2'b10;
      /*状态编码，采用格雷（Gray）编码方式*/
always @(posedge clk or posedge clr) /*该过程定义当前状态*/
begin if(clr) state<=S0;          //异步复位，s0为起始状态
      else state<=next_state;
end
always @(state or x)              /*该过程定义次态*/
begin
case (state)
   S0:begin if(x) next_state<=S1;
   else       next_state<=S0; end
   S1:begin  if(x)        next_state<=S1;
   else       next_state<=S2; end
```



“101”序列检测器的Verilog描述（三个过程）

```
S2:begin
   if(x) next_state<=S3;
   else next_state<=S0; end
S 3 : b e g i n
          i   f   (   x   )
       next_state<=S1;
   else      next_state<=S2;
   end
default:     next_state<=S0;
       /*default语句*/
   endcase
end
```

```
always @(state)
       /*该过程产生输出逻辑*/
begin  case(state)
       S3: z=1'b1;
       default:z=1'b0;
endcase
end
endmodule
```

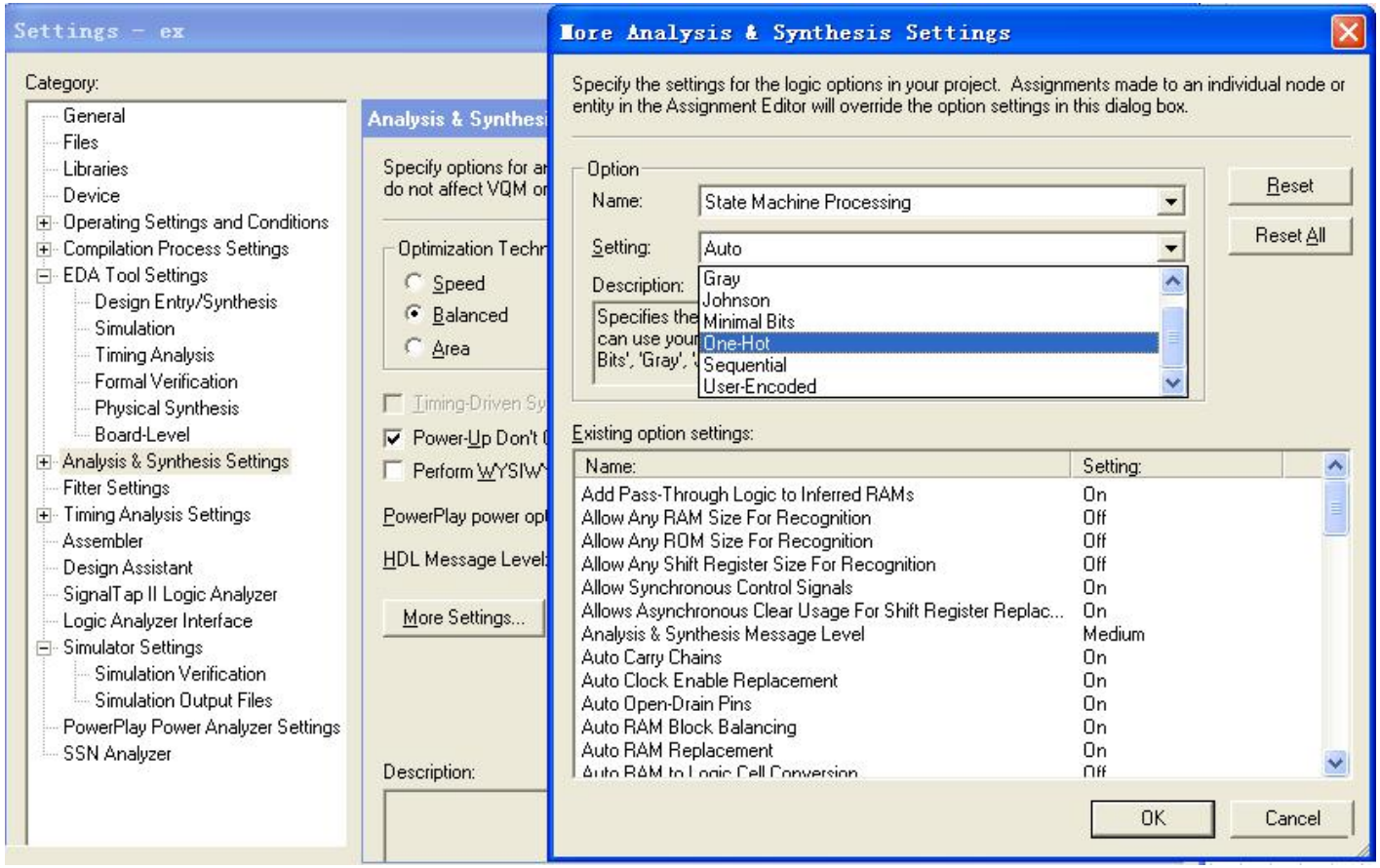
“101”序列检测器（单过程描述）

```
module fsm4_seq101(clk,clr,x,z);
input clk,clr,x; output reg z; reg[1:0] state;
parameter     S0=2'b00,S1=2'b01,S2=2'b11,S3=2'b10;
/*状态编码，采用格雷（Gray）编码方式*/
always @(posedge clk or posedge clr)
Begin     if(clr) state<=S0;          //异步复位，s0为起始状态
   else case(state)
   S0:begin   if(x) begin state<=S1; z=1'b0;end
       else begin state<=S0; z=1'b0;end
              end
   S1:begin   if(x) begin state<=S1; z=1'b0;end
              else begin state<=S2; z=1'b0;end
               end
   S2:begin   if(x) begin state<=S3; z=1'b0;end
              else begin state<=S0; z=1'b0;end
              end
   S3:begin  if(x) begin state<=S1; z=1'b1;end
              else begin state<=S2; z=1'b1;end
              end
   default:begin  state<=S0; z=1'b0;end   /*default语句*/
   endcase  end
endmodule
```

8.3 状态编码

常用的编码方式

- 顺序编码
- 格雷编码
- 约翰逊编码
- 一位热码



一位热码编码选择对话框（Quartus II）

状态编码的定义

在Verilog语言中，有两种方式可用于定义状态编码，分别用parameter和'define语句实现，比如要为state0、state1、state2、state3四个状态定义码字为：00、01、11、10，可采用下面两种方式。

方式1：用parameter参数定义

```
parameter
state1=2'b00,state2=2'b01,state3=2'b11,state4=2'b10;
……
case(state)
state1: …;                    //调用
state2: …;
……
```

状态编码的定义

状态编码的定义方式2：用'define语句定义

```
'define state1   2'b00                    //不要加分号“;”
'define state2   2'b01
'define state3   2'b11
'define state4   2'b10
   case(state)
'state1:     …;                          //调用，不要漏掉符号“'”
'state2:     …;
……
```

要注意两种方式定义与调用时的区别，一般情况下，更倾向于采用方式1来定义状态编码。一般使用case、casez和casex语句来描述状态之间的转换，用case语句表述比用if-else语句更清晰明了。

8.4 有限状态机设计要点

1．起始状态的选择：

起始状态是指电路复位后所处的状态，选择一个合理的起始状态将使整个系统简洁、高效。多数EDA软件会自动为基于状态机的设计选择一个最佳的起始状态。

2．有限状态机的同步复位

3．有限状态机的异步复位

多余状态的处理

一般有如下两种处理多余状态的方法：

（1）在case语句中用default分支决定如果进入无效状态所采取的措施；

（2）编写必要的Verilog源代码明确定义进入无效状态所采取的行为。

习 题 8

8.1 设计一个“111”串行数据检测器。要求是：当检测到连续3个或3个以上的“1”时输出为1，其他输入情况下输出为0。

8.2 设计一个“1001”串行数据检测器。其输入、输出如下所示：

输入x：000 101 010 010 011 101 001 110 101

输出z：000 000 000 010 010 000 001 000 000

8.3 编写一个8路彩灯控制程序，要求彩灯有以下3种演示花型。

（1）8路彩灯同时亮灭；

（2）从左至右逐个亮（每次只有1路亮）；

（3）8路彩灯每次4路灯亮，4路灯灭，且亮灭相间，交替亮灭；

在演示过程中，只有当一种花型演示完毕才能转向其他演示花型。

习 题 8

8.4 用状态机设计一个交通灯控制器，设计要求：A路和B路，每路都有红、黄、绿三种灯，持续时间为：红灯45 s，黄灯5 s，绿灯40 s。A路和B路灯的状态转换是：

（1）A红，B绿（持续时间40 s）；

（2）A红，B黄（持续时间5 s）；

（3）A绿，B红（持续时间40 s）；

（4）A黄，B红（持续时间5 s）。

实验与设计

8-1 流水灯控制器

实验要求：采用有限状态机设计彩灯控制器，控制LED灯实现预想的演示花型。

采用有限状态机设计一个彩灯控制器，要求控制18个LED灯实现如下的演示花型：

从两边往中间逐个亮；全灭；

从中间往两头逐个亮；全灭；

循环执行上述过程。

8-2 汽车尾灯控制器

实验要求：设计一个汽车尾灯控制电路。

功能：设计一个汽车尾灯控制电路。已知汽车左右两侧各有3个尾灯，要求控制尾灯按如下规则亮灭。

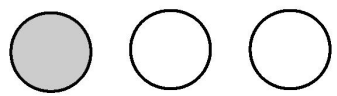
汽车沿直线行驶时，两侧的指示灯全灭；

右转弯时，左侧的指示灯全灭，右侧的指示灯按000，100，010，001，000循环顺序点亮；

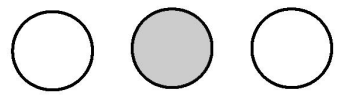
左转弯时，右侧的指示灯全灭，左侧的指示灯按与右侧同样的循环顺序点亮；

如果在直行时刹车，两侧的指示灯全亮；如果在转弯时刹车，转弯这一侧的指示灯按上述的循环顺序点亮，另一侧的指示灯全亮；

临时故障或紧急状态时，两侧的指示灯闪烁。



左



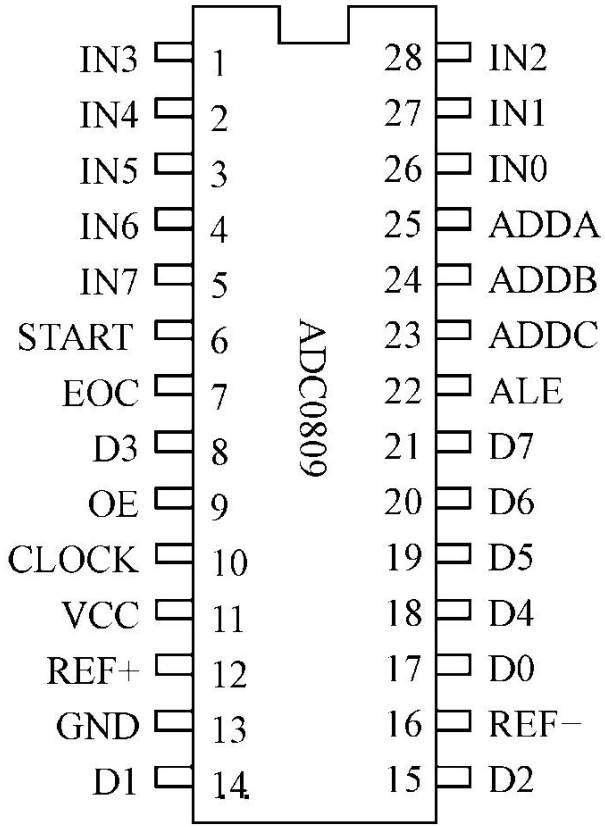
右

【例8.12】 汽车尾灯控制器。

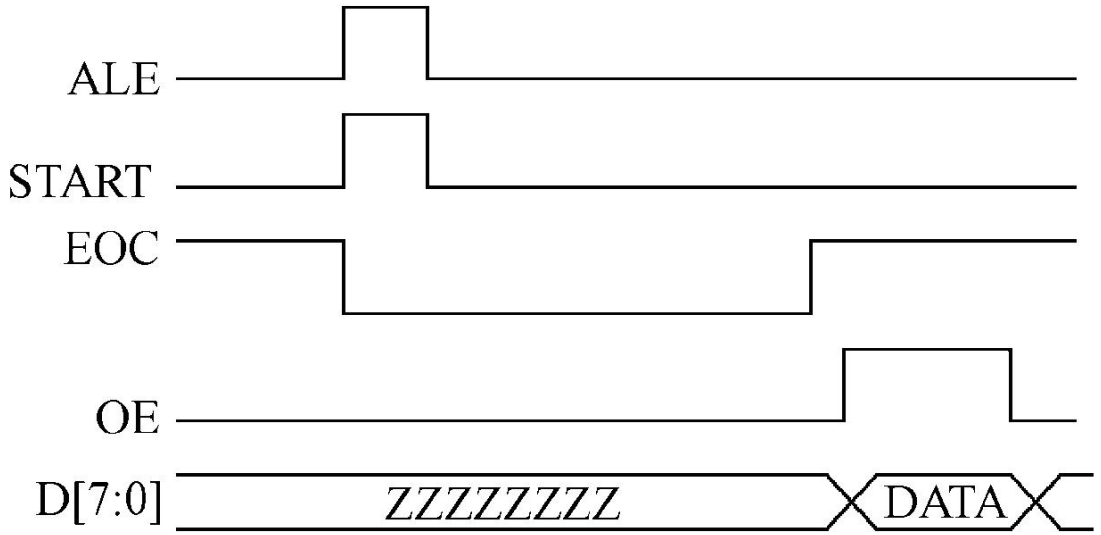
```
module backlight(clk50m,turnl,turnr,brake,fault,lightl,lightr);
(* chip_pin="Y2" *) input clk50m;                    //时钟
(* chip_pin="AB28" *) input turnl;                   //左转信号
(* chip_pin="AC28" *) input turnr;                   //右转信号
(* chip_pin="AC27" *) input brake;                   //刹车信号
(* chip_pin="AD27" *) input fault;                   //故障信号
(* chip_pin="E18,F18,F21" *)output[2:0] lightl; //左侧灯
(* chip_pin="E19,F19,G19" *)output[2:0] lightr; //右侧灯
reg[23:0] count;
wire clock;
reg[2:0] shift=3'b001;
reg flash=1'b0;
always@(posedge clk50m)
     begin if(count==12500000) count<=0; else count<=count+1;  end
   assign clock=count[23];
   always@(posedge clock)
begin  shift={shift[1:0],shift[2]};flash=~flash;  end
assign lightl=turnl?shift:brake?3'b111:fault?{3{flash}}:3'b000;
assign lightr=turnr?shift:brake?3'b111:fault?{3{flash}}:3'b000;
endmodule
```

8-3 状态机A/D采样控制电路

实验要求：采用状态机实现ADC0809控制电路，实现数据采集。



ADC0809引脚图



ADC0809工作时序

8-4 用状态机实现字符液晶显示控制

实验要求：采用状态机控制字符液晶实现字符的显示

基于DE2-115平台用FPGA控制字符液晶实现字符的显示。字符液晶由液晶显示器和专用的行、列驱动器、控制器及必要的连接件装配而成，可显示数字和英文字符。字符液晶本身具有字符发生器，显示容量大，功能丰富，一般最少可显示1行8个或l行16个字符，每个字符由5×7、5×8或5×11的一组像素点阵排列构成，字符间有一定的间隔，行与行间也有一定的间隔。

用状态机实现字符显示控制

H1602B液晶模块的读/写操作、屏幕和光标的操作都是通过指令编程来实现的，为了方便控制，采用状态机实现设计。在设计中设置8个状态，分别是起始状态CLEAR，设置CGRAM状态SETCGRAM，工作方式设置状态SETFUNCTION，显示方式设置状态SWITCHMODE，输入方式设置状态SETMODE，光标归位状态RETURNCURSOR，字符移位状态SHIFT，写RAM状态WRITERAM，状态编码采用One-Hot方式。