第 5 章 状态机和简单数字系统设计

5.1 状态机实验

有限状态机(Finite State Machine,FSM)简称状态机,是一个在有限个状态之间进行转换和动作的计算模型。有限状态机含有一个起始状态、一个输入列表(列表中包含了所有可能的输入信号序列)、一个状态转移函数和一个输出端,状态机在工作时由状态转移函数根据当前状态和输入信号来确定下一个状态和输出。状态机一般都是从起始状态开始,根据输入信号由状态转移函数决定状态机的下一个状态。

有限状态机是数字电路系统中一种十分重要的电路模块,是一种输出取决于过去输入和当前输入的时序逻辑电路,是组合逻辑电路和时序逻辑电路的一种组合。其中组合逻辑分为两个部分,一部分是用于产生有限状态机下一个状态的次态逻辑,另一部分是用于产生输出信号的输出逻辑。除了输入和输出外,状态机还有一组具有“记忆”功能的寄存器,这些寄存器的功能是记忆有限状态机的内部状态,常被称为状态寄存器。

本实验的目的是学习状态机的工作原理,了解状态机的编码方式,并学习简单状态机的设计。

5.1.1 有限状态机

在实际应用中,有限状态机被分为两种:Moore 型有限状态机和 Mealy 型有限状态机。

Moore 型有限状态机的输出信号只与有限状态机的当前状态有关,与输入信号的当前值无关,输入信号的当前值只会影响到状态机的次态,不会影响到状态机当前的输出。即 Moore 型有限状态机的输出信号是直接由状态寄存器译码得到。Moore 型有限状态机在时钟 CLK 信号有效后再经过一段时间的延迟,输出达到稳定值。即使在这个时钟周期内输入信号发生变化,输出也会在这个完整的时钟周期内保持稳定值而不再变化。输入对输出的影响要到下一个时钟周期才能反映出来。Moore 有限状态机最重要的特点就是将输入与输出信号隔离开来。

Mealy 状态机与 Moore 有限状态机不同,Mealy 有限状态机的输出不仅与状态机的当前状态有关,而且与输入信号的当前值也有关。Mealy 有限状态机的输出直接受输入

信号的当前值影响，而输入信号可能在一个时钟周期内任意时刻发生变化，这使得 Mealy 有限状态机对输入的响应发生在当前时钟周期，比 Moore 有限状态机对输入信号的响应要早一个周期。因此，输入信号的噪声可能影响到输出的信号。

5.1.2 简单状态机 FSM

本节通过设计一个实际的状态机来了解状态机的工作过程和设计方法。

请设计一个区别两种特定时序的有限状态机 FSM：该有限状态机有一个输入 w 和一个输出 z。当 w 是 4 个连续的“0”或者 4 个连续的“1”时，输出 $z=1$，否则 $z=0$。时序允许重叠，若 w 是连续的 5 个“1”时，则在第 4 个和第 5 个时钟之后，z 均为“1”。图 5-1 是这个有限状态机的时序图。

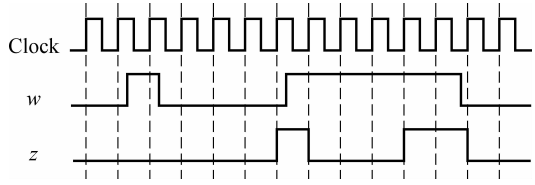

图 5-1 FSM 的时序图

这个状态机的状态图如图 5-2 所示。

用 Verilog 代码实现如图 5-2 所示的状态机。此状态机有 9 个状态，至少需要 4 个状态寄存器按照二进制编码形式实现这个 FSM，如表 5-1 所示。

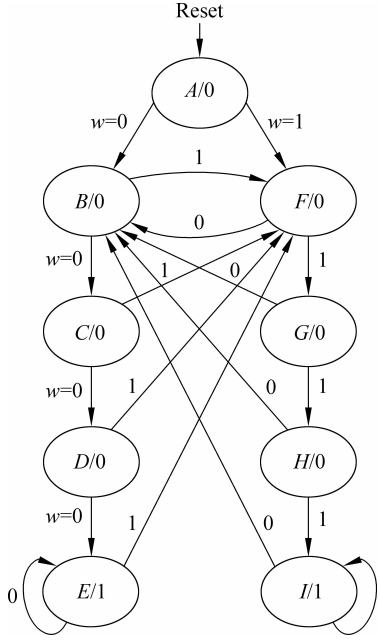

图 5-2 FSM 的状态图

表 5-1 FSM 的二进制编码

状态	y_3	y_2	y_1	y_0
A	0	0	0	0
B	0	0	0	1
C	0	0	1	0
D	0	0	1	1
E	0	1	0	0
F	0	1	0	1
G	0	1	1	0
H	0	1	1	1
I	1	0	0	0

建立一个 Verilog 文件，用 iSW[0]作为 FSM 低电平有效同步复位端，用 iSW[1]作为输入 w，用 iKEY[0]作为手动的时钟输入，用 oLEDG[0]作为输出 z，用 oLEDR[3]～oLEDR[0]显示 9 个状态，完成该状态机的具体设计。

Quartus Ⅱ里面自带了用 HDL 语言实现的状态机模板，在 Quartus Ⅱ的文件编辑窗口单击右键选择“insert template”，参考 template 可以提高状态机设计效率，如图 5-3 和图 5-4 所示。程序清单 5.1 即为此状态机的 Verilog HDI 代码。

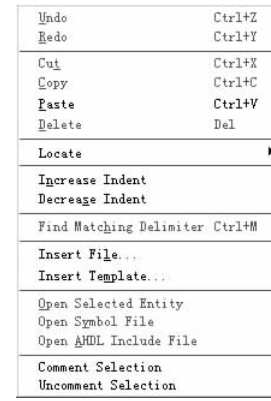

图 5-3　选择 Quartus Ⅱ设计模板

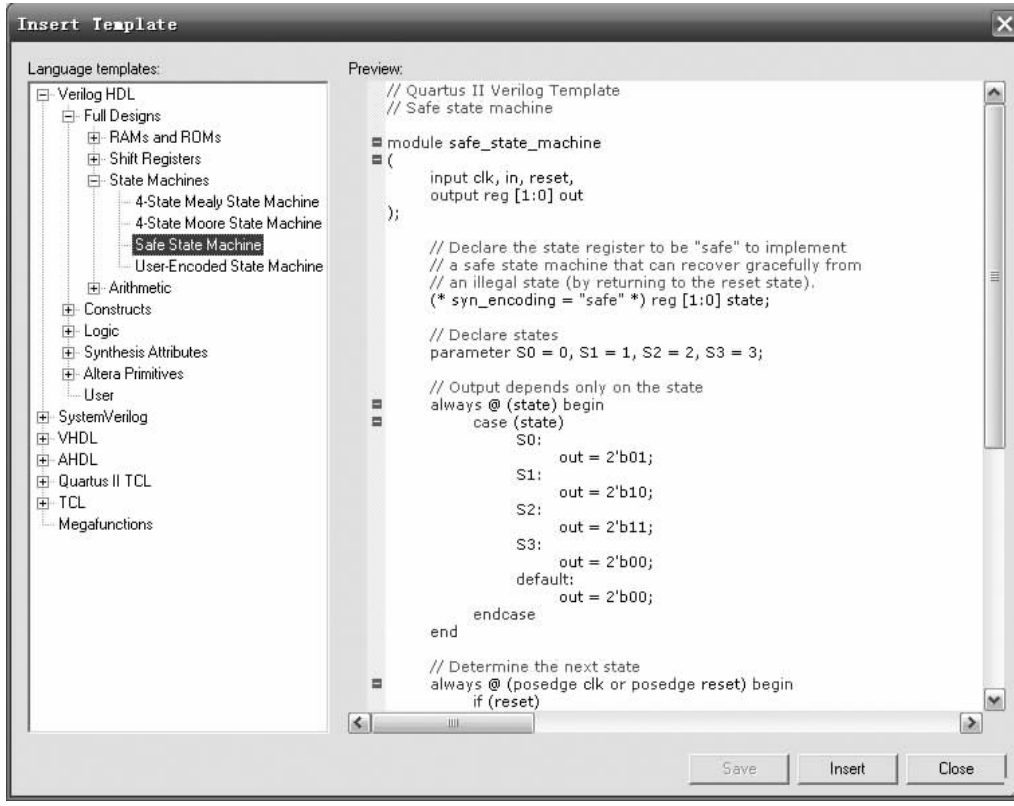

图 5-4　模板代码

程序清单 5.1　区别两种输入状态的状态机。

```
module FSM_bin
(
    input    clk, in, reset,
    output reg out
);
```

```
    //Declare the state register to be "safe" to implement
    //a safe state machine that can recover gracefully from
    //an illegal state (by returning to the reset state).
    (* syn_encoding="safe" *) reg [3:0] state;
    //Declare states
    parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5, S6=6, S7=7, S8=8;
    //Output depends only on the state
    always @(state) begin
        case (state)
            S0: out=1'b0;
            S1: out=1'b0;
            S2: out=1'b0;
            S3: out=1'b0;
            S4: out=1'b1;
            S5: out=1'b0;
            S6: out=1'b0;
            S7: out=1'b0;
            S8: out=1'b1;
            default: out=1'bx;
        endcase
    end
    //Determine the next state
    always @(posedge clk) begin
        if (reset)        state<=S0;
        else
            case (state)
                S0: if (in)    state<=S5;    else    state<=S1;
                S1: if (in)    state<=S5;    else    state<=S2;
                S2: if (in)    state<=S5;    else    state<=S3;
                S3: if (in)    state<=S5;    else    state<=S4;
                S4: if (in)    state<=S5;    else    state<=S4;
                S5: if (in)    state<=S6;    else    state<=S1;
                S6: if (in)    state<=S7;    else    state<=S1;
                S7: if (in)    state<=S8;    else    state<=S1;
                S8: if (in)    state<=S8;    else    state<=S1;
            endcase
    end
endmodule
```

编译工程，用 Netlist Viewers 工具查看代码生成的状态图，如图 5-5 所示。如果编译

结果不能产生如图 5-5 所示的状态图，可能是用户编写的代码不符合 Quartus Ⅱ编译器要求的状态机代码风格，重新采用 Quartus Ⅱ编译器能够识别的方式编写状态机，就可以产生状态图。

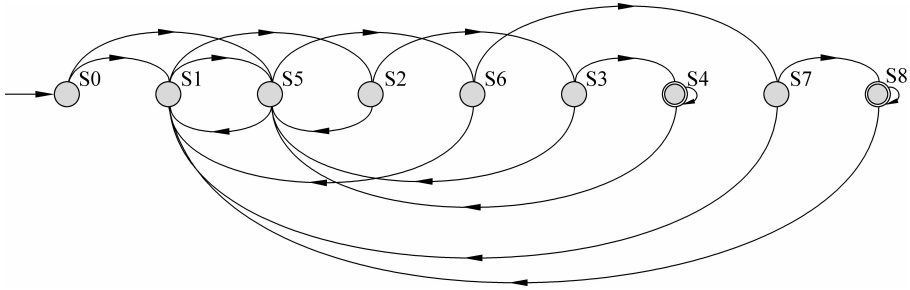

图 5-5　生成状态图

为此状态机编写测试代码，如图 5-6 所示。图 5-7 是此状态机的功能仿真结果。

```
`timescale 10 ns/ 1 ps
module FSM_bin_vlg_tst();
// constants
// test vector input registers
reg clk;
reg in;
reg reset;
// wires
wire out;

FSM_bin i1 (
// port map - connection between master ports and signals/registers
   .clk(clk),
   .in(in),
   .out(out),
   .reset(reset)
);
initial
begin
clk = 0; in = 0; reset = 1;
#15 in = 1; reset = 0;
#20 in = 0;
#40 in = 1;
#150 reset = 1;
#20 in = 0;
#50;
$stop;
end
always
#10 clk = ~clk;
endmodule
```

图 5-6　测试代码

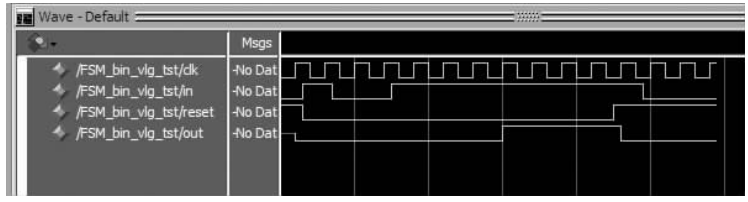

图 5-7　功能仿真

用 RTL Viewer 工具查看代码生成的门级电路，然后用 Technology Map Viewer 工具查看状态机在 FPGA 中的实现。对电路进行仿真并下载到 DE2-70 开发板上，验证其

功能。

5.1.3 状态机的编码方式

上一节例子中的状态机采用顺序二进制编码 binary 方式，即将状态机的状态依次编码为顺序的二进制数，用顺序二进制数编码可使状态向量的位数最少。如本例中只需要4位二进制数来编码。节省了保存状态向量的逻辑资源。但是在输出时要对状态向量进行解码以产生输出(某个状态有特定的输出，因此输出时要对此状态进行解码，满足状态编号时才可输出特定值)，这个解码过程往往需要许多组合逻辑。

另外，当芯片受到辐射或者其他干扰时，可能会造成状态机跳转失常，甚至跳转到无效的编码状态而出现死机。例如，状态机因为异常跳转到某状态，而此状态需要等待输入，并作出应答，此时因为状态运转不正常，不会出现输入，状态机就会进入死等状态。

one-hot 编码也是状态机设计中常用的编码，在 one-hot 编码中，对于任何给定的状态，其状态向量中只有 1 位是“1”，其他所有位的状态都为“0”，n 个状态就需要 n 位的状态向量，所以 one-hot 编码最长。one-hot 编码对于状态的判断非常方便，如果某位为“1”就是某状态，如果该位为“0”则不是此状态。因此，判断状态输出时非常简单，只要一、两个简单的“与门”或者“或门”即可。

one-hot 编码的状态机从一个状态到另一个状态的状态跳转速度非常快，而顺序二进制编码从一个状态跳转到另外一个状态则需要较多次跳转，并且随着状态的增加，速度急剧下降。

在芯片受到干扰时，one-hot 编码一般只能跳转到无效状态(如果跳到另一有效状态必须是当前为“1”的变为“0”，同时另外一位由“0”变为“1”，这种可能性很小)，因此 one-hot 编码的状态机稳定性高。

格雷码(gray-code)也是状态机设计中常用一种编码方式，它的优点是 gray-code 状态机在发生状态跳转时，状态向量只有 1 位发生变化。

一般而言，顺序二进制编码和 gray-code 的状态机使用了最少的触发器，较多的组合逻辑，适用于提供更多的组合逻辑的 CPLD 芯片。对于具有更多触发器资源的 FPGA，用 one-hot 编码实现状态机则更加有效。所以，CPLD 多使用 gray-code，而 FPGA 多使用 one-hot 编码。另一方面，对于小型设计使用 gray-code 和 binary 编码更有效，而大型状态机设计使用 one-hot 更高效。

在表 5-2 中，状态机有 9 个状态，我们可以用 9 个触发器来实现这个状态机的电路，这个状态机的电路就是从输入到每一个状态触发器输出端的逻辑表达式的实现电路。这 9 个状态触发器用 y_8 y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0 表示。该状态机的 one-hot 编码如表 5-2 所示。

5.1.4 实验内容

建立一个 Verilog 文件，调用 9 个触发器来实现这个 FSM，用 iSW[0]作为 FSM 低电平有效同步复位端，用 iSW[1]作为输入 w，用 iKEY[0]作为手动的时钟输入，用 oLEDG[0]作为输出 z，用 oLEDR[8]～oLEDR[0]显示 9 个触发器的状态。

表 5-2　FSM 的 one-hot 编码

状态	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

编译工程，用 Netlist Viewers 工具查看代码生成的状态图，然后用 Technology Map Viewer 工具查看状态机在 FPGA 中的实现。对电路进行仿真并下载到 DE2-70 开发板上，验证其功能。

请完成该设计，并与上一个二进制编码的 FSM 进行比较。

5.2　雷鸟车尾灯控制器*

5.2.1　实验目的

大部分的控制系统都可以描述为一个有限状态机，复杂的计算机系统其实也是一个有限状态机系统，本次实验通过一个实例，进一步了解状态机的工作过程和设计方法。

5.2.2　实验内容

1965 年生产的福特雷鸟汽车的车尾每边有三个灯，这些灯轮流地按顺序亮起，表示车子的转向。当驾驶员发出将要向左(LEFT)转的指令时，车尾左侧的三个灯按顺序分别亮 0、1 和 2 三个灯。当驾驶员发出将要向右(RIGHT)转的指令时，车尾右侧的三个灯按顺序分别亮 0、1 和 2 三个灯。另外，还有一个应急闪烁输入(HAZ)，它要求车尾灯工作在告警状态，这时所有 6 个灯协调地闪烁。

5.2.3　问题分析

本设计有三个输入端：LEFT、RIGHT 和 HAZ，告警状态优先级最高，有告警信号时首先进入告警状态。汽车在空闲状态时，输入信号要求汽车进入的工作状态如表 5-3 所示。另外，还需要一个单独运行的时钟信号，该信号的频率等于这些灯所要求的闪烁频率。

汽车有 4 个工作状态：空闲状态、左转弯状态、右转弯状态和告警状态，汽车的左转弯状态和右转弯状态又分为 4 个状态：空闲及 0、1 和 2 三个灯亮。因此，本设计用一个

* 本例选自《数字设计原理与实践》第四版中的 7.5 节。

时钟同步 Moore 状态机来实现，这个状态机有 8 个状态，每个状态对应的输出如表 5-4 所示。

表 5-3　汽车工作状态表

HAZ	LEFT	RIGHT	汽车工作状态
0	0	0	空闲
0	1	0	左转弯
0	0	1	右转弯
0	1	1	告警
1	×	×	告警

表 5-4　工作状态对应的输出

状　态	输　出
IDLE	空闲状态，6 个灯全灭
L_1	左边 1 灯亮
L_2	左边 2 灯亮
L_3	左边 3 灯亮
R_1	右边 1 灯亮
R_2	右边 2 灯亮
R_3	右边 3 灯亮
ERR	告警状态，6 个灯全亮

状态转换约定：在进入左转弯状态时，状态机在 IDLE、L_1、L_2 和 L_3 4 个状态间循环，此时只有在 IDLE 状态可以接收任何其他信号输入，并选择进入相应状态。在 L_1 和 L_2 状态只能接收告警状态输入，如果有告警信号，结束左转弯循环，进入告警状态。进入 L_3 状态后不接受任何信号输入，直接进入 IDLE 状态。右转弯状态类似左转弯状态。告警状态在 ERR 状态和 IDLE 状态循环转换。没有任何信号输入时，状态机保持在 IDLE 状态。

状态图如图 5-8 所示。

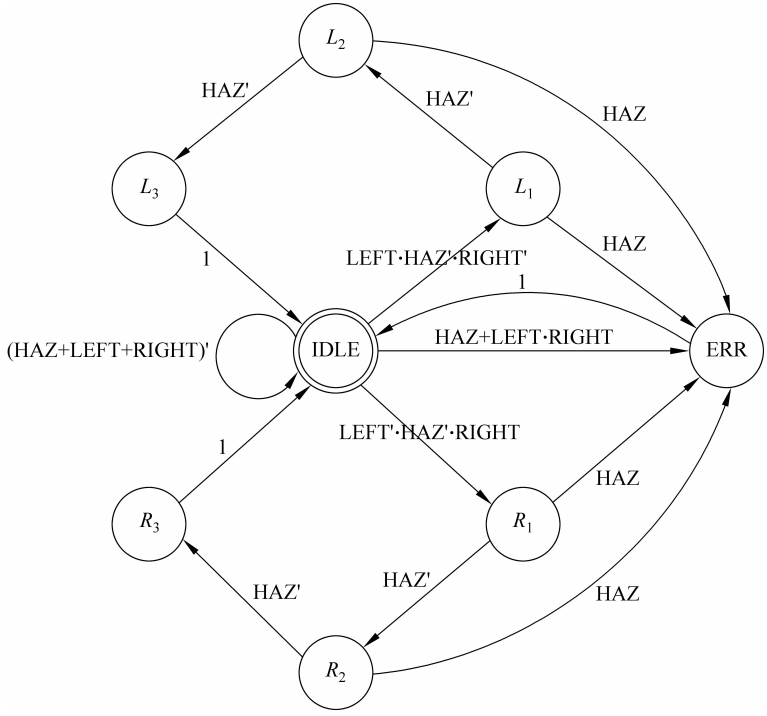

图 5-8　雷鸟车尾灯状态图

请根据上述分析，设计出雷鸟汽车尾灯控制器。并利用 FPGA 开发板资源，验证你设计的正确性。

5.3 交通控制灯实验

5.3.1 实验目的

这是一个综合性的实验，目的是复习巩固以前学过的方法和知识，请读者通过自己的努力设计一个相对复杂的、有实际用途的数字控制系统，学习应用已经学到的知识，了解“我们生活在数字世界中”这一概念。

5.3.2 实验内容

要求设计一个交通信号灯控制电路，主要适用于在两条干道（一条主干道一条支干道）汇合点形成的十字交叉路口，路口设计两组红绿灯分别对两个方向上的交通运行状态进行管理。实验要求如下：

（1）能显示十字路口东西、南北两个方向的红、黄、绿的指示状态。用两组红、黄、绿三色灯作为两个方向的红、黄、绿灯。（注：本实验板上没有黄灯，请想办法用其他方式代替。）

（2）能实现正常的倒计时功能。用两组数码管作为东西和南北方向的倒计时显示，主干道每次放行（绿灯）60 秒，支干道每次放行（绿灯）45 秒，在每次由绿灯变成红灯的转换过程中，要亮黄灯 5 秒作为过渡。

（3）能实现总体清零功能。按下该键后，系统实现总清零，计数器由初始状态计数，对应状态的指示灯亮。

（4）可变换亮灯时间（选做）。由于夜间 21:00 点至 05:00 点之间路上车辆很少，主干道和支干道相比放行时间太长，因此一些交通路口设置了不同时间段的放行时间，请修改你的设计，优化交通灯功能。这里的时间控制最好采用自动的方式，而不是等待外界输入信号而改变的方式，因为设计者不可能在每个夜间人工改变路灯的放行时间。

请根据所学实验方法，完成该设计。