第6章　如何写好状态机

节选自《Verilog 设计与验证》 作者：吴继华、王诚

状态机是逻辑设计的重要内容，状态机的设计水平直接反应工程师的逻辑功底，所以许多公司的硬件和逻辑工程师面试中，状态机设计几乎是必选题目。本章在引入状态机设计思想的基础上，重点讨论如何写好状态机。

本章主要内容如下：

- 状态机的基本概念；
- 如何写好状态机；
- 使用 Synplify Pro 分析 FSM。

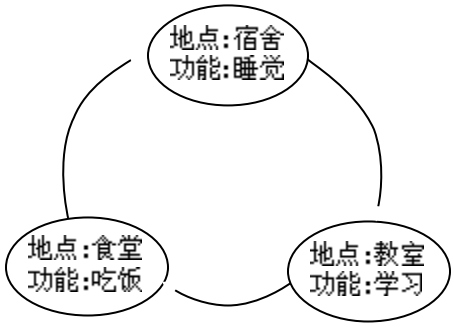
6.1　状态机的基本概念

本节的重点在于帮助读者理解状态机的基本概念和应用场合。

6.1.1　状态机是一种思想方法

相信大多数工科学生在学习数字电路时都学习过状态机的基本概念，了解一些使用状态机描述时序电路的基本方法。但是，笔者希望大家能扩展思维，认识到状态机不仅仅是一种时序电路设计工具，它更是一种思想方法。

我们先看下面一个简单的例子。在大学生活中，某学生的在校的学习生活可以简单地概括为宿舍、教室、食堂之间的周而复始，用图 6-1 就可以形象地表现出来。这里画这张图，并不是要讨论这个学生是否是一个“乖乖”类型学生，请大家注意，如果将图中的“地点”认为是“状态”，将“功能”认为是状态的“输出”，这张图就是一张标准的状态转移图，也就是说，我们用状态机的方式清晰地描述了这个学生的在校生活方式。



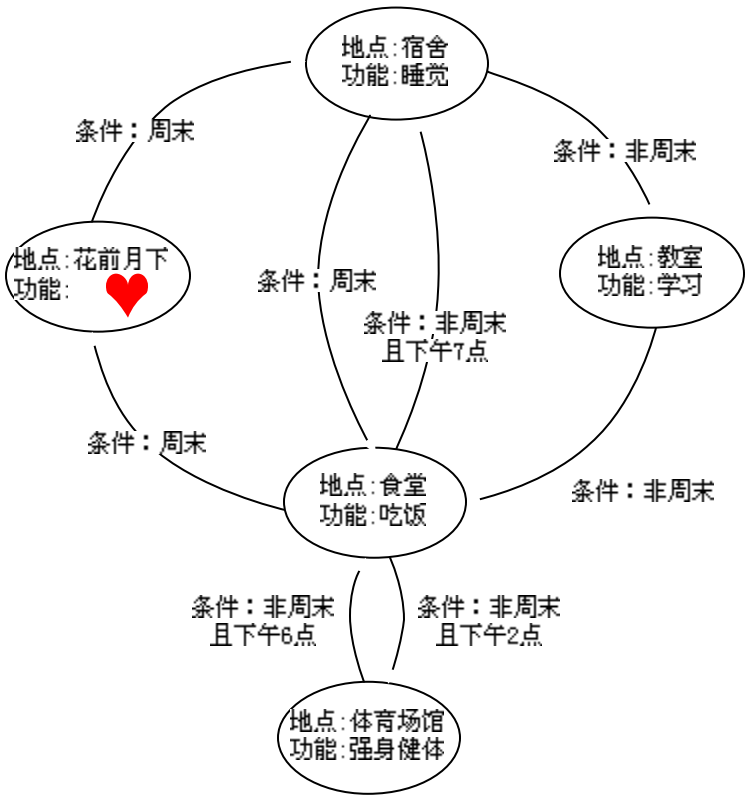


图6-2　另一位学生在校生活状态转移图

同样如果将图中的“ 地点 ”认为是“ 状态 ”，将“ 功能 ”认为是状态的“ 输出 ”，将“ 条件 ”认为是状态转移的“ 输入条件 ”，图 6-2 也是一张标准的状态转移图，通过状态机的方式我们再次清晰地描述另一个学生的在校生活方式。

事实上使用状态机方式，我们可以细致入微地描述任何一个学生的在校生活方式。大家通过前面两个简单举例已经发现状态机特别适合描述那些有发生有先后顺序，或者有逻辑规律的事情——其实这就是状态机的本质。状态机的本质就是对具有逻辑顺序或时序规律事件的一种描述方法。这个论断的最重要的两个词就是“ 逻辑顺序 ”和“ 时序规律 ”，这两点

就是状态机所要描述的核心和强项，换言之，所有具有逻辑顺序和时序规律的事情都适合用状态机描述。

很多初学者不知道何时应用状态机。这里介绍两种应用思路：第一种思路，从状态变量入手。如果一个电路具有时序规律或者逻辑顺序，我们就可以自然而然地规划出状态，从这些状态入手，分析每个状态的输入，状态转移和输出，从而完成电路功能；第二种思路是首先明确电路的输出的关系，这些输出相当于状态的输出，回溯规划每个状态，和状态转移条件与状态输入。无论那种思路，使用状态机的目的都是要控制某部分电路，完成某种具有逻辑顺序或时序规律的电路设计。

其实对于逻辑电路而言，小到一个简单的时序逻辑，大到复杂的微处理器，都适合用状态机方法进行描述。请读者打开思路，不要仅仅局限于时序逻辑，发现电路的内在规律，确认电路的“状态变量”，大胆使用状态机描述电路模型。由于状态机不仅仅是一种电路描述工具，它更是一种思想方法，而且状态机的 HDL 语言表达方式比较规范，有章可循，所以很多有经验的设计者习惯用状态机思想进行逻辑设计，对各种复杂设计都套用状态机的设计理念，从而提高设计的效率和稳定性。

6.1.2 状态机基本要素与分类

状态机的基本要素有 3 个，其实我们在第一节的举例中都有涉及，只是没有点明，它们是：状态、输出和输入。

- 状态：也叫状态变量。在逻辑设计中，使用状态划分逻辑顺序和时序规律。比如：设计伪随机码发生器时，可以用移位寄存器序列作为状态；在设计电机控制电路时，可以以电机的不同转速作为状态；在设计通信系统时，可以用信令的状态作为状态变量等。
- 输出：输出指在某一个状态时特定发生的事件。如设计电机控制电路中，如果电机转速过高，则输出为转速过高报警，也可以伴随减速指令或降温措施等。
- 输入：指状态机中进入每个状态的条件，有的状态机没有输入条件，其中的状态转移较为简单，有的状态机有输入条件，当某个输入条件存在时才能转移到相应的状态。

根据状态机的输出是否与输入条件相关，可将状态机分为两大类：摩尔（Moore）型状态机和米勒（Mealy）型状态机。

- 摩尔状态机：摩尔状态机的输出仅仅依赖于当前状态，而与输入条件无关。例如图 6-1 所示的例子，将图中的“地点”认为是“状态”，将“功能”认为是状态的“输出”，则每个输出仅仅与状态相关，所以它是一个摩尔型状态机。
- 米勒型状态机：米勒型状态机的输出不仅依赖于当前状态，而且取决于该状态的输入条件。例如图 6-2 所示的例子，将图中的“地点”认为是“状态”，将“功能”认为是状态的“输出”，将“条件”认为是状态转移的“输入条件”，大家可以发现，该学生到达什么地方，做什么事情都是由当前状态和输

入条件共同决定，所以它是一个米勒型状态机。

根据状态机的数量是否为有限个，可将状态机分为有限状态机（Finite State Machine，FSM）和无限状态机（Infinite State Machine，ISM）。逻辑设计中一般所涉及的状态都是有限的，所以以后我们所说的状态机都指有限状态机，用 FSM 表示。

6.1.3 状态机的基本描述方式

逻辑设计中，状态机的基本描述方式有 3 种，分别是：状态转移图，状态转移列表，HDL 语言描述。

- **状态转移图**

  状态转移图是状态机描述的最自然的方式。如本章第一节图 6-1，6-2 都使用了状态转移图这一描述方式。状态转移图经常在设计规划阶段定义逻辑功能时使用，也可以在分析代码中状态机时使用，通过图形化的方式非常有助于理解设计意图。

  另外值得一提的是目前有一些 EDA 工具支持状态转移图作为逻辑设计的输入，例如在 StateCAD。在该工具中设计者只要画出状态转移图就可以了，StateCAD 能自动将状态转移图翻译成 HDL 语言代码，而且翻译出来的代码规范、可读性较好、可综合、易维护。StateCAD 还能能自动检测状态机的完备性和正确性，对状态转移图中的冗余状态、自锁状态、歧义转移条件和不完备状态机等隐含错误都会报警，并协助设计者更正错误。最后 StateCAD 会自动生成设计的测试激励，并调用仿真程序，验证状态机的正确性，这个测试激励甚至可在后仿真中使用。总之，StateCAD 提供了状态机的输入、翻译、检测、优化和测试等一条龙的服务，使状态机的设计变得安全、可靠、快速、便捷。这类自动转换状态转移图为 HDL 源代码的工具对设计、分析一些规模较小的状态机非常有效，但是由于自动反应的代码过于程式化，效率不是最高，所以对于较大规模的逻辑设计，一般还是推荐使用 HDL 语言之间描述。

使用 Synplify Pro 的 RTL 视图配合 FSM Viewer 可以将源代码中描述的 FSM 用状态转移图显示出来，使用图形化的界面帮助用户分析理解状态机。关于使用 FSM Viewer 分析状态机的方法在本章 6.3 节有详细介绍。

- **状态转移列表**

  状态转移列表是用列表的方式描述状态机，是数字逻辑电路常用的设计方法之一，经常被用于对状态化简，对于可编程逻辑设计，由于可用逻辑资源比较丰富，而且状态编码要考虑设计的稳定性，安全性等因素，所以并不经常使用状态转移列表优化状态。

- **HDL 语言描述状态机**

  使用 HDL 语言描述状态机是本章讨论的重点，使用 HDL 语言描述状态机

有一定的灵活性，但是决不是天马行空，而是有章可循的。通过一些规范的描述方法，可以使 HDL 语言描述的状态机更安全、稳定、高效、易于维护。

6.2 如何写好状态机

本节重点讨论可综合的状态机描述的一些基本规范，即如何在 RTL 级描述安全、高效的 FSM。

6.2.1 什么是 RTL 级好的 FSM 描述

首先介绍好的 RTL 级 FSM 的评判标准。其实评判 FSM 的标准很多，这里我们拣选最重要的几个方面讨论一下。好的 RTL 级 FSM 的评判标准如下：

- **FSM 要安全，稳定性高。**
  所谓 FSM 安全是指 FSM 不会进入死循环，特别是不会进入非预知的状态，而且由于某些扰动进入非设计状态，也能很快的恢复到正常的状态循环中来。这里面有两层含义，第一：要求该 FSM 的综合实现结果无毛刺等异常扰动；第二：要求状态机要完备，即使收到异常扰动进入非设计状态，也能很快恢复到正常状态。
- **FSM 速度快，满足设计的频率要求。**
  任何 RTL 设计都应该满足设计的频率要求。
- **FSM 面积小，满足设计的面积要求。**
  同理任何 RTL 设计都应该满足设计的面积要求。
- **FSM 设计要清晰易懂、易维护。**
  不规范的 FSM 写法很难让其他人解读，甚至过一段时间后设计者也发现很难维护。

需要说明的是以上所列的各项标准，特别是前 3 项标准绝不是割裂的，它们直接有紧密的内在联系。如果读者读过本工作室的其他书籍，应该记得其中花了相当长的篇幅论述 FPGA/CPLD 设计评判的两个基本标准：面积和速度。这里“面积”是指一个设计所消耗 FPGA/CPLD 的逻辑资源数量；“速度”指设计在芯片上稳定运行所能够达到的最高频率。两者是对立统一的矛盾体，要求一个设计同时具备设计面积最小，运行频率最高，这是不现实的。科学的设计目标应该是：在满足设计时序要求（包含对设计最高频率的要求）的前提下，占用最小的芯片面积，或者在所规定的面积下，使设计的时序余量更大，频率更高。

另外，如果要求 FSM 安全，则很多时候需要使用“full case”的编码方式，即将状态转移变量的所有向量组合情况都在 FSM 中有相应的处理，这经常势必意味着要多花更多的设计资源，有时也会影响 FSM 的频率。

所以，各条标准要综合考虑，根据设计的要求进行权衡。但是如果各条评判标准发生冲突时，请按照标准的罗列顺序考虑，前文标准的罗列顺序是根据这些标准在设计中的重要性排列的，也就是说第一条“FSM 要安全，稳定性高”的优先级最高，最重要；第四条

“FSM 设计要清晰易懂、易维护”的优先级最低，是相对次要的标准。

6.2.2　RTL 级状态机描述常用语法

本书第 2、3 章论述了 Verilog 的基本语法和常用关键字，其中在 RTL 级设计可综合的 FSM 相关的常用关键字如下：

- **wire 、reg 等**

  对 wire 、reg 等变量、向量定义不加累述，需要补充的是状态编码时（也就是用某种编码描述各个状态）一般都要使用 reg 寄存器型向量。

- **parameter**

  用于描述状态名称，增强源代码可读性，简化描述。

  例：某状态机使用初始值为“0”的独热码（one-hot）编码方式定义的 4bit 宽度的状态变量 NS（代表 Next State，下一状态）和 CS（代表 Current State，当前状态），且状态机包含 5 个具体状态 IDLE（空闲状态）、S1（工作状态 1）、S2（工作状态 2）、S3（工作状态 3）、ERROR（告警状态），则代码如下：

```
reg [3:0] NS,CS;
parameter [3:0]   //one hot with zero initial
IDLE   = 3'b0000,
S1     = 3'b0001,
S2     = 3'b0010,
S3     = 3'b0100,
ERROR  = 3'b1000;
```

- **always**

  在 FSM 设计中有 3 种 always 的使用方法，第 1 种用法是根据主时钟沿，完成同步时序的状态迁移。

  例：某状态机从当前状态 CS 迁移到下一个状态 NS 可以如下表述：

```
//sequential state transition
always @ (posedge clk or negedge nrst)
     if (!nrst)
        CS <= IDLE;
     else
        CS <=NS;
```

  always 的第 2 种用法是根据信号敏感表，完成组合逻辑的输出。

  always 的第 3 种用法是根据时钟沿，完成同步时序逻辑的输出。

- **case/endcase**

  case/endcase 是 FSM 描述中最重要的语法关键字，这里我们要详细讨论一下。case/endcase 的基本语法结构如下：

```
case (case_expression)
```

```
case_item1 : case_item_statement1;
case_item2 : case_item_statement2;
case_item3 : case_item_statement3;
case_item4 : case_item_statement4;
default : case_item_statement5;
endcase
```

其中，case_expression **就是** case **的判断条件表达式，在** FSM **描述中，它一般为当前状态寄存器；每个** case_item **是** case **语句的分支列表，在** FSM **描述中，它一般为** FSM **中的所有状态的罗列，从中还可以分析出状态的编码方式；**case_item_statement **为进入每个** case_item **的对应操作，在** FSM **中，即为每个状态对应的状态转移或者输出，如果** case_item_statement **包含的操作不只一条，可以用** **begin/end** **嵌套多条操作；****default** **是个可选的关键字，用以指明当所列的所有** case_item **与** case_expression **都不匹配时的操作，在** FSM **设计中，为了提高设计的安全性，排除所设计的** FSM **进入死循环，一般要求加上** **default** **关键字来描述** FSM **所需状态的补集状态下的操作。另外** Verilog **还支持** **casex** **和** **casez** **等不同关键字，但是由于综合器对这两个关键字的支持情况略有差异，所以笔者建议初学者使用完整的** **case** **结构而不使用** **casex** **或** **casez**。

例：某 FSM **的状态转移用** case/endcase **结构描述如下：**

```
case (CS)
    IDLE:    begin
                 IDLE_out;
                 if (~i1)          NS = IDLE;
                 if (i1 && i2)      NS = S1;
                 if (i1 && ~i2)    NS = ERROR;
             end
    S1:      begin
                 S1_out;
                 if (~i2)          NS = S1;
                 if (i2 && i1)      NS = S2;
                 if (i2 && (~i1))   NS = ERROR;
             end
    S2:      begin
                 S2_out;
                 if (i2)           NS = S2;
     if (~i2 && i1)    NS = IDLE;
                 if (~i2 && (~i1))  NS = ERROR;
             end
    ERROR:   begin
```

```
                    ERROR_out;
                    if (i1)          NS = ERROR;
                    if (~i1)         NS = IDLE;
                end
        default:  begin
                    Default_out;
                    NS = ERROR;
                end
    endcase
```

Verilog 的 case 结构虽然与 C 等高级语言的 case 结构虽然形式相似，但是本质不同。Verilog 的 case 结构对应并行判断的硬件结构，而且当 case_expression 与任意一个 case_item 匹配后，将忽略对其它 case_item 的判断，执行完匹配的 case_item_statement 后直接跳出 case 结构。

- **task/endtask**

  task/endtask 在描述状态机是主要用途是将不同状态对应的输出用 task/endtask 封装，增强了代码的可维护性和可读性。

  例：某状态机的 IDLE 状态的输出可以用 task/endtask 封装为“IDEL_out”任务：

```
task IDLE_out;
    begin
    {w_o1,w_o2,w_err} = 3'b000;
    end
```

endtask 当然描述状态机时也会使用到其它一些常用的 RTL 级语法，如 if/else，assign 等等，它们的功能和一般 RTL 描述方法一致，这里不在叙述。

6.2.3 推荐的状态机描述方法

状态机描述时关键是要描述清楚前面提到的几个状态机的要素，即如何进行状态转移；每个状态的输出是什么；状态转移是否和输入条件相关等。具体描述时方法各种各样，有的设计者习惯将整个状态机写到 1 个 always 模块里面，在该模块中即描述状态转移，又描述状态的输入和输出，这种写法一般被称为一段式 FSM 描述方法；还有一种写法是将用 2 个 always 模块，其中一个 always 模块采用同步时序描述状态转移；另一个模块采用组合逻辑判断状态转移条件，描述状态转移规律，这种写法被称为两段式 FSM 描述方法；还有一种写法是在两段式描述方法基础上发展出来的，这种写法使用 3 个 always 模块，一个 always 模块采用同步时序描述状态转移；第二个采用组合逻辑判断状态转移条件，描述状态转移规律；第三个 always 模块使用同步时序电路描述每个状态的输出，这种写法本书称为三段式写法。

一般而言，推荐的 FSM 描述方法是后两种，即两段式和三段式 FSM 描述方法。其原因为：FSM 和其他设计一样，最好使用同步时序方式设计，以提高设计的稳定性，消除毛刺。状态机实现后，一般来说，状态转移部分是同步时序电路而状态的转移条件的判断是组合逻辑。两段式之所以比一段式编码合理，就在于两段式编码将同步时序和组合逻辑分别放到不同的 always 程序块中实现。这样做的好处不仅仅是便于阅读、理解、维护，更重要的是利于综合器优化代码，利于用户添加合适的时序约束条件，利于布局布线器实现设计。而一段式 FSM 描述不利于时序约束、功能更改、调试等，而且不能很好的表示米勒 FSM 的输出，容易写出 Latches，导致逻辑功能错误。

在一般两段式描述中，为了便于描述当前状态的输出，很多设计者习惯将当前状态的输出用组合逻辑实现。但是这种组合逻辑仍然有产生毛刺的可能性，而且不利于约束，不利于综合器和布局布线器实现高性能的设计。因此如果设计运行额外的一个时钟节拍的插入（latency），则要求尽量对状态机的输出用寄存器寄存一拍。但是很多实际情况不允许插入一个寄存节拍，此时则可以通过三段式描述方法进行解决。三段式与两段式相比，关键在于根据状态转移规律，在上一状态根据输入条件判断出当前状态的输出，从而在不插入额外时钟节拍的前提下，实现了寄存器输出。

为了便于理解，我们通过一个实例讨论这三种不同的写法。

【例1-1】 使用不同的 FSM 描述风格描述状态机，参考示例详见本书附带光盘的“Example-6-1”目录。

在这个范例中我们将用一段式、两段式、三段式分别描述图 6-3 所示的状态机。这里我们选用了一个非常典型的米勒型状态机，共有 4 个状态：IDEL，S1，S2，ERROR；输入信号为时钟 clk，低电平异步复位信号 nrst，输入信号 i1，i2，输出信号为 o1，o2 和 err，状态关系如图 6-2 所示。状态的输出如下：

IDLE　　状态的输出为：　{o1,o2,err} = 3'b000;
S1　　状态的输出为：　{o1,o2,err} = 3'b100;
S2　　状态的输出为：　{o1,o2,err} = 3'b010;
ERROR　　状态的输出为：　{o1,o2,err} = 3'b111。

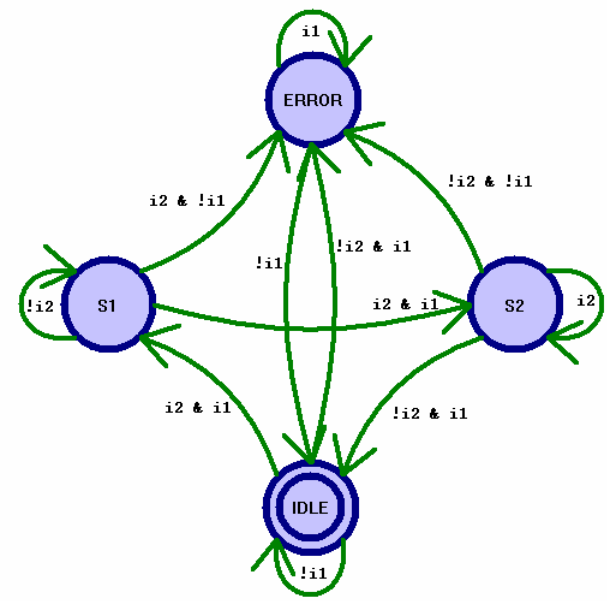


图6-3　例子的状态转移图

6.2.3.1　一段式状态机描述方法（应该避免的写法）

该例的一段式描述代码如下：

```
//1-paragraph method to describe FSM
//Describe state transition, state output, input condition in 1 always block
module state1 ( nrst,clk,
                i1,i2,
                o1,o2,
                err
               );

input          nrst,clk;
input          i1,i2;
output         o1,o2,err;
reg            o1,o2,err;
reg    [2:0]   NS; //NextState
parameter [2:0]      //one hot with zero idle
      IDLE   = 3'b000,
      S1     = 3'b001,
      S2     = 3'b010,
      ERROR  = 3'b100;

//1 always block to describe state transition, state output, input condition
```

```
always @ (posedge clk or negedge nrst)
 if (!nrst)
    begin
       NS         <= IDLE;
      {o1,o2,err} <= 3'b000;
    end
 else
    begin
       NS         <=  3'bx;
      {o1,o2,err} <=  3'b000;
      case (NS)
        IDLE:  begin
                 if (~i1)        begin{o1,o2,err}<=3'b000;NS  <= IDLE; end
                  if (i1 && i2)    begin{o1,o2,err}<=3'b100;NS <= S1;   end
                 if (i1 && ~i2)   begin{o1,o2,err}<=3'b111;NS <= ERROR;end
               end
        S1:    begin
                 if (~i2)         begin{o1,o2,err}<=3'b100;NS <= S1;   end
                  if (i2 && i1)    begin{o1,o2,err}<=3'b010;NS <= S2;   end
                  if (i2 && (~i1)) begin{o1,o2,err}<=3'b111;NS <= ERROR;end
               end
        S2:    begin
                 if (i2)          begin{o1,o2,err}<=3'b010;NS <= S2;   end
                if (~i2 && i1)   begin{o1,o2,err}<=3'b000;NS <= IDLE; end
                 if (~i2 && (~i1))begin{o1,o2,err}<=3'b111;NS <= ERROR;end
               end
        ERROR: begin
                 if (i1)         begin{o1,o2,err}<=3'b111;NS <= ERROR;end
                  if (~i1)        begin{o1,o2,err}<=3'b000;NS <= IDLE; end
               end
      endcase
   end
endmodule
```

如前面介绍，一段式写法就是将状态的同步转移，状态输出和状态的输入条件都写在一个 always 模块中，一段式写法可以概括为图 6-4 **描述的结构。**

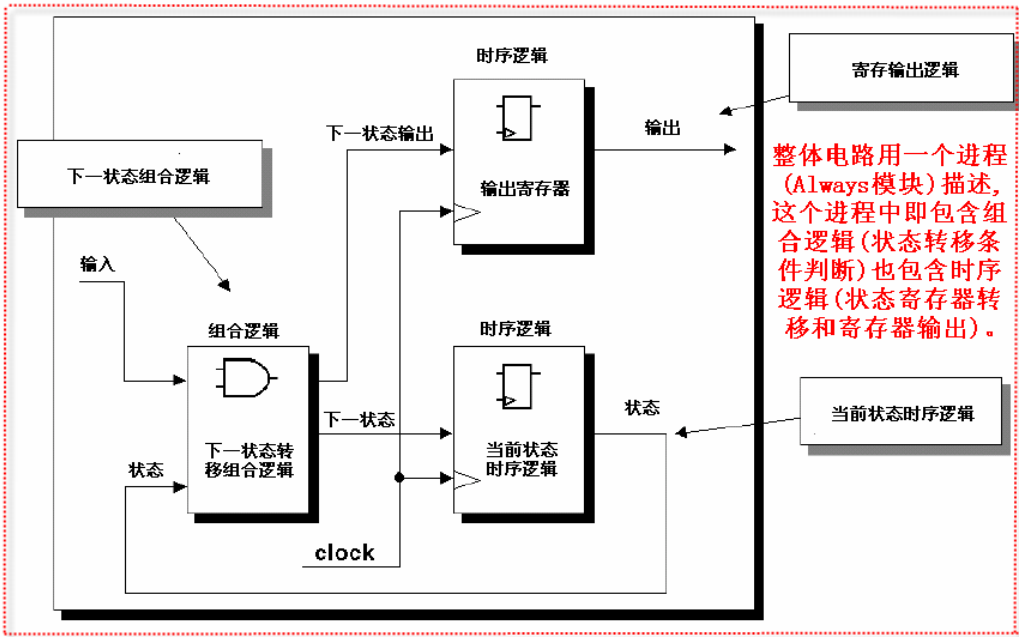


图6-4　一段式 FSM 描述结构图

一段式描述方法将状态转移判断的组合逻辑和状态寄存器转移的时序逻辑混写在同一个 always 模块中，不符合将时序和组合逻辑分开描述的 Coding Style（代码风格），而且在描述当前状态时要考虑下个状态的输出，整个代码不清晰，不利于维护修改，并且不利于附加约束，不利于综合器和布局布线器对设计的优化。

另外，这种描述相对于两段式描述比较冗长。本例为了便于初学者掌握，选择了一个非常简单的米勒型状态机，不能很好的反应一段式比较冗长的缺点，但是如果状态机相对复杂些，一般来说，一段式代码长度会比两段式冗长大约 80%到 150%左右。

所以一段式 FSM 描述是不推荐的 FSM 描述方式，请读者一定要避免。

6.2.3.2　两段式状态机描述方法（推荐写法）

为了使 FSM 描述清晰简介，易于维护，易于附加时序约束，使综合器和布局布线器更好的优化设计，推荐使用两段式 FSM 描述方法。

本例的两段式描述代码如下：

```
//2-paragraph method to describe FSM
//Describe sequential state transition in 1 sequential always block
//State transition conditions in the other combinational always block
//Package state output by task. Then register the output
module state2 ( nrst,clk,
                i1,i2,
                o1,o2,
                err
               );
```

```
input         nrst,clk;
input         i1,i2;
output        o1,o2,err;
reg           o1,o2,err;
reg    [2:0]   NS,CS;
parameter [2:0]      //one hot with zero idle
     IDLE   = 3'b000,
     S1     = 3’b001,
     S2     = 3’b010,
     ERROR  = 3’b100;
//sequential state transition
always @ (posedge clk or negedge nrst)
     if (!nrst)
        CS <= IDLE;
     else
        CS <=NS;
//combinational condition judgment
always @ (CS or i1 or i2)
         begin
              NS = 3'bx;
              ERROR_out;
              case (CS)
                  IDLE:     begin
                                IDLE_out;
                                if (~i1)          NS = IDLE;
                                if (i1 && i2)      NS = S1;
                                if (i1 && ~i2)     NS = ERROR;
                            end
                  S1:       begin
                                S1_out;
                                if (~i2)          NS = S1;
                                if (i2 && i1)      NS = S2;
                                if (i2 && (~i1))   NS = ERROR;
                            end
                  S2:       begin
                                S2_out;
                                if (i2)           NS = S2;
                   if (~i2 && i1)     NS = IDLE;
                                if (~i2 && (~i1))  NS = ERROR;
```

```
                    end
            ERROR:   begin
                        ERROR_out;
                        if (i1)          NS = ERROR;
                        if (~i1)         NS = IDLE;
                    end
        endcase
    end
//output task
task IDLE_out;
    {o1,o2,err} = 3'b000;
endtask
task S1_out;
    {o1,o2,err} = 3'b100;
endtask
task S2_out;
    {o1,o2,err} = 3'b010;
endtask
task ERROR_out;
    {o1,o2,err} = 3'b111;
endtask
endmodule
```

两段式写法是推荐的 FSM 描述方法之一，在此我们仔细讨论一下代码结构。两段式 FSM 的核心就是：一个 always 模块采用同步时序描述状态转移；另一个模块采用组合逻辑判断状态转移条件，描述状态转移规律。两段式写法可以概括为图 6-5 描述的结构。

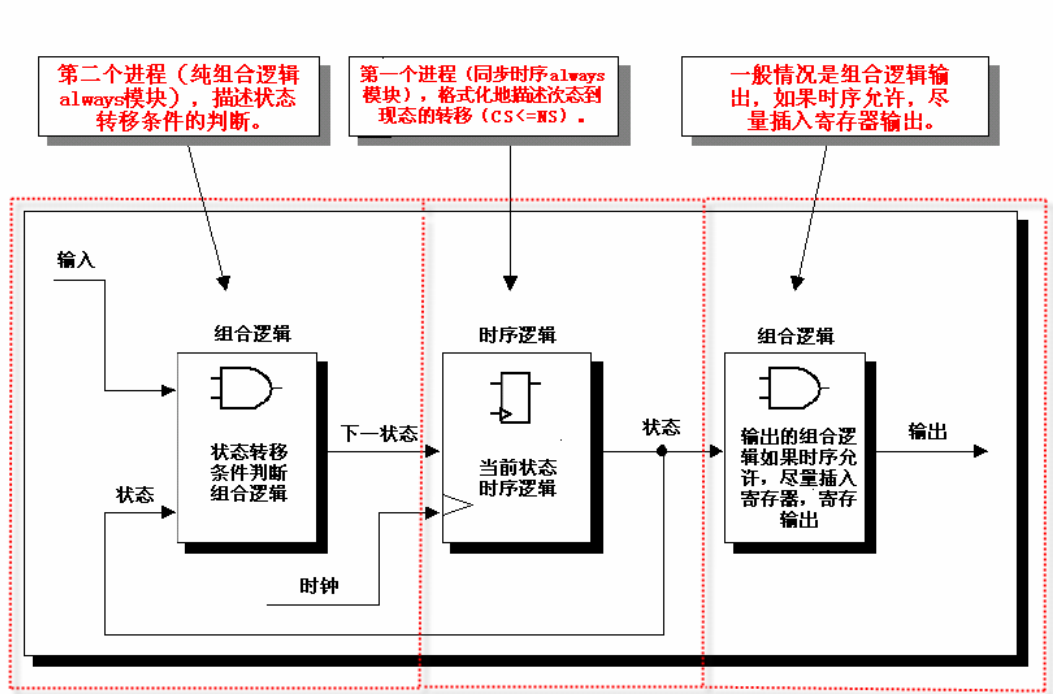


图6-5　两段式 FSM 描述结构图

本例中，同步时序描述状态转移的 always 模块代码如下：

```
always @ (posedge clk or negedge nrst)
     if (!nrst)
        CS <= IDLE;
     else
        CS <=NS;
```

其实这是一种程式化的描述结构，无论具体到何种 FSM 设计，都可以定义两个状态寄存器“CS”和“NS”，分别代表当前状态和下一状态，然后根据所需的复位方式（同步复位或异步复位），在时钟沿到达时将 NS 赋给 CS。需要注意的是这个同步时序模块的赋值要采用非阻塞赋值“<=”。

本例中，另一个采用组合逻辑判断状态转移条件的 always 模块代码如下：

```
//combinational condition judgment
always @ (nrst or CS or i1 or i2)
         begin
              NS = 3'bx;
              ERROR_out;
              case (CS)
                   IDLE:     begin
                                  IDLE_out;
                                  if (~i1)          NS = IDLE;
                                  if (i1 && i2)      NS = S1;
```

```
                    if (i1 && ~i2)     NS = ERROR;
                end
        S1:      begin
                    S1_out;
                    if (~i2)          NS = S1;
                    if (i2 && i1)      NS = S2;
                    if (i2 && (~i1))   NS = ERROR;
                end
        S2:      begin
                    S2_out;
                    if (i2)           NS = S2;
         if (~i2 && i1)     NS = IDLE;
                    if (~i2 && (~i1))  NS = ERROR;
                end
        ERROR:    begin
                    ERROR_out;
                    if (i1)           NS = ERROR;
                    if (~i1)          NS = IDLE;
                end
    endcase
end
```

这个使用组合逻辑判断状态转移条件的 always 模块也可以看成格式化的书写结构。其中 always 的敏感列表为当前状态“CS”，复位信号和输入条件（如果是米勒状态机，则必须有输入条件；如果是摩尔状态机，一般敏感表和后续逻辑判定没有输入），请大家注意电平敏感表必须列完整。本例中这段电平敏感列表为：

```
always @ (nrst or CS or i1 or i2)
```

一般来说，在这个组合 always 敏感表下先写一个默认的下一状态“NS”的描述，然后根据实际的状态转移条件由内部的 case 或者 if...else 条件判断确定正确的转移。如本例中下面这段代码，

```
……
begin
    NS = ERROR;
    ERROR_out;
    case (CS)
……
```

推荐在敏感表下的默认状态为不定状态 X，这样描述的好处有两个：第一在仿真时可以很好的考察所设计的 FSM 的完备性，如果所设计的 FSM 不完备，则会进入任意状态，仿真很容易发现；第二个好处是综合器对不定态 X 的处理是“Don’t Care”，即任何没有定义的状态寄存器向量都会被忽略。这里赋值不定态的效果和使用 casez 或 casex 替代 case 的效果

非常相似。

在每个 case 模块的内部的结构也非常相似，都是先描述当前状态的组合逻辑输出，然后根据输入条件（米勒 FSM）判定下一个状态。

该组合逻辑模块中所有的赋值推荐采用阻塞赋值“ = ”。

请大家注意，虽然下一状态寄存器 NS 为寄存器类型，但是在两段式 FSM 的判断状态转移条件的 always 模块中，实际上对应的真实硬件电路是纯组合逻辑电路。

对于每个输出，一般用组合逻辑描述，比较简便的方法是用 task/endtask 将输出封装起来，这样做的好处不仅仅是写法简单，而且利于复用共同的输出。例如本例中 S1 状态的输出被封装为 S1_out，在组合逻辑 always 模块中直接调用即可。

```
task S1_out;
    {o1,o2,err} = 3'b100;
endtask
```

组合逻辑容易产生毛刺，因此如果时序允许，请尽量对组合逻辑的输出插入一个寄存器节拍，这样可以很好的保证输出信号的稳定性。

6.2.3.3 三段式状态机描述方法（推荐写法）

两段式 FSM 描述方法虽然有很多好处，但是它有一个明显的弱点就是其输出一般使用组合逻辑描述，而组合逻辑易产生毛刺等不稳定因素，并且在 FPGA/CPLD 等逻辑器件中过多的组合逻辑会影响实现的速率（这点与 ASIC 设计不同）。所以在上节我们特别提到了在两段式 FSM 描述方法中，如果时序允许插入一个额外的时钟节拍，则尽量在在后级电路对 FSM 的组合逻辑输出用寄存器寄存一个节拍，则可以有效地消除毛刺。但是很多情况下，设计并不允许额外的节拍插入（Latency），此时，解决之道就是采用 3 段式 FSM 描述方法。三段式描述方法与两段式描述方法相比，关键在于使用同步时序逻辑寄存 FSM 的输出。

本例的三段式描述代码如下：

```
//3-paragraph method to describe FSM
//Describe sequential state transition in the 1st sequential always block
//State transition conditions in the 2nd combinational always block
//Describe the FSM out in the 3rd sequential always block
    module state2 ( nrst,clk,
                    i1,i2,
                    o1,o2,
                    err
                   );
    input          nrst,clk;
    input          i1,i2;
    output         o1,o2,err;
```

```
reg          o1,o2,err;
reg   [2:0]   NS,CS;
parameter [2:0]     //one hot with zero idle
     IDLE   = 3'b000,
     S1     = 3'b001,
     S2     = 3'b010,
     ERROR  = 3'b100;
//1st always block, sequential state transition
always @ (posedge clk or negedge nrst)
     if (!nrst)
        CS <= IDLE;
     else
        CS <=NS;
//2nd always block, combinational condition judgment
always @ (nrst or CS or i1 or i2)
         begin
              NS = 3'bx;
              case (CS)
                   IDLE:     begin
                                 if (~i1)           NS = IDLE;
                                  if (i1 && i2)     NS = S1;
                                  if (i1 && ~i2)    NS = ERROR;
                             end
                   S1:       begin
                                 if (~i2)           NS = S1;
                                 if (i2 && i1)      NS = S2;
                                  if (i2 && (~i1))  NS = ERROR;
                             end
                   S2:       begin
                                 if (i2)            NS = S2;
                                if (~i2 && i1)      NS = IDLE;
                                  if (~i2 && (~i1)) NS = ERROR;
                             end
                   ERROR:    begin
                                 if (i1)           NS = ERROR;
                                  if (~i1)          NS = IDLE;
                             end
              endcase
         end
```

```
//3rd always block, the sequential FSM output
always @ (posedge clk or negedge nrst)
 if (!nrst)
     {o1,o2,err} <= 3'b000;
 else
    begin
       {o1,o2,err} <=  3'b000;
       case (NS)
          IDLE:  {o1,o2,err}<=3'b000;
          S1:    {o1,o2,err}<=3'b100;
          S2:    {o1,o2,err}<=3'b010;
          ERROR: {o1,o2,err}<=3'b111;
       endcase
    end
endmodule
```

三段式写法可以概括为图 6-6 描述的结构。

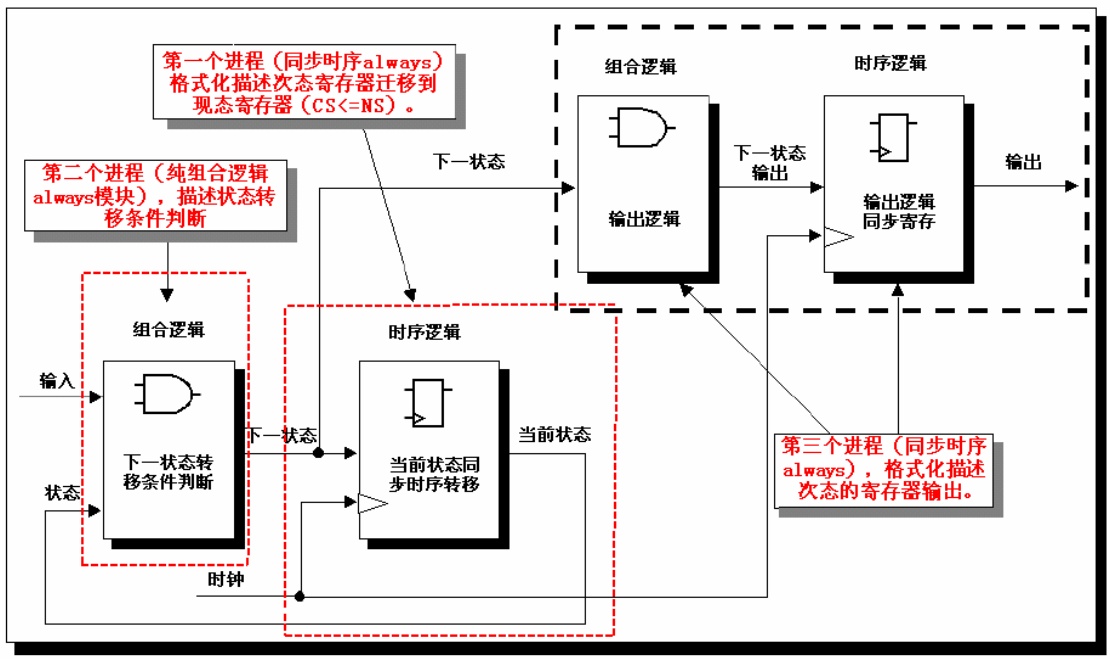


图6-6　三段式 FSM 描述结构图

对比一下上节两段式 FSM 的描述，读者可以清晰发现三段式与两段式 FSM 描述的最大区别在于两段式采用了组合逻辑输出，而三段式巧妙地根据下一状态的判断，用同步时序逻辑寄存 FSM 的输出。本例中就是下面一段代码，

```
always @ (posedge clk or negedge nrst)
 if (!nrst)
     {o1,o2,err} <= 3'b000;
```

```
else
   begin
      {o1,o2,err} <=  3'b000;
      case (NS)
         IDLE:  {o1,o2,err}<=3'b000;
         S1:    {o1,o2,err}<=3'b100;
         S2:    {o1,o2,err}<=3'b010;
         ERROR: {o1,o2,err}<=3'b111;
      endcase
   end
```

有的读者可能会问，一段式写法也是用寄存器同步了 FSM 的输出，为什么前面介绍一段式的输出代码容易混淆，不利于维护呢？请大家对比一下 6.2.3.1 小节的这段一段式输出的代码，

```
……
case (NS)
 IDLE:  begin
        if (~i1)        begin{o1,o2,err}<=3'b000;NS  <= IDLE; end
        if (i1 && i2)    begin{o1,o2,err}<=3'b100;NS <= S1;   end
        if (i1 && ~i2)   begin{o1,o2,err}<=3'b111;NS <= ERROR;end
       end
……
```

通过对比，可以清晰地看到：使用一段式建模 FSM 的寄存器输出的时候，必须要综合考虑现态在何种状态转移条件下会进入哪些次态，然后在每个现态的 case 分支下分别描述每个次态的输出，这显然不符合思维习惯；而三段式建模描述 FSM 的状态机输出时，只需指定 case 敏感表为次态寄存器，然后直接在每个次态的 case 分支中描述该状态的输出即可，根本不用考虑状态转移条件。本例的 FSM 很简单，如果设计的 FSM 相对复杂，三段式的描述优势就会凸显出来。

另一方面，三段式描述方法与两段式描述相比，虽然代码结构复杂了一些，但是换来的优势是使 FSM 做到了同步寄存器输出，消除了组合逻辑输出的不稳定与毛刺的隐患，而且更利于时序路径分组，一般来说在 FPGA/CPLD 等可编程逻辑器件上的综合与布局布线效果更佳。

请读者注意，在三段式 FSM 描述方法中，判断状态转移的 always 模块的 case 语句判断的条件是当前状态“CS”而在同步时序 FSM 输出的 always 模块的 case 语句判断的条件是下一状态“NS”。

6.2.3.4　三种描述方法与状态机建模问题的引申

可以说合理的状态机描述与状态机的建模技巧是本章的重中之重。这里需要引申讨论几个问题。

- **n 段式描述方法和 always 语法块的个数**
  通过学习，大家知道标准的一段式、两段式、三段式 FSM 描述方法分别使用了 1、2、3 个 always 语法块。但是请读者注意这个命题的反命题不成立，不能说一段 FSM 的描述中，使用了 n 个 always 语法块，就是 n 段式描述方法。这是因为我们特指的一段式、两段式、三段式 FSM 描述方法中每个 always 语法块都有固定的描述内容和格式化的结构，其实也就是通过这些特定的描述内容和格式化的结构，确立了 3 种 FSM 建模方式。例如两段式写法中，第一个 always 模块格式化地使用同步时序电路描述次态寄存器到现态寄存器的转移；而第二个 always 模块格式化地使用纯组合逻辑描述状态转移条件。也就是说两段式描述对应的建模方式的硬件电路就是图 6-5 所示的电路结构。其实站在语法角度上，我们总可以将一个 always 模块拆分成多个 always 模块，或者反之将多个 always 模块合并为一个 always 模块。所以请读者注意，n 段式 FSM 描述方法强调的是一种建模思路，绝不是简单的 always 语法块个数。

- **FSM 的建模方式**
  这里我们反复强调 n 段式描述方法其实是 FSM 的三种建模方式。大家回顾一下状态转移图 6-3 描述的 FSM，在学习本节之前，大家可能会有产生各种不同的描述思路，通过本节的学习，希望读者能够自然而然地想到用图 6-5（对应两段式思路）和图 6-6（对应三段式思路）的结构建模。其实对于绝大多数 FSM，都可以采样图 6-4，或图 6-5，或图 6-6 的结构建模。一般来说，笔者推荐使用后两种结构建模。这是因为：两段式思路建模结构清晰，描述简洁，便于约束，而且如果允许输出逻辑允许插入一个节拍，就可以通过插入输出寄存器改善输出逻辑的时序并避免组合逻辑的毛刺；三段式思路建模结构清晰，格式化的结构，解决了不改变时序要求的前提下用寄存器做状态输出的问题。请大家仔细研究图 6-4、图 6-5、图 6-6 体会三种建模方式。

- **一段式建模和三段式建模的关系**
  这里我们引申比较一下 3 种 FSM 建模的关系。请读者比较图 6-4 与图 6-6，如果将图 6-4 的两部分组合逻辑合并起来，则三段式建模电路于一段式建模电路的结构完全一致了，如图 6-7 所示。反过来，大家可以看到三段式与一段式的最大区别在于：使用一段式建模 FSM 的寄存器输出的时候，必须要综合考虑现态在何种状态转移条件下会进入哪些次态，然后在每个现态的 case 分支下分别描述每个次态的输出，这显然不符合思维习惯；而三段式建模描述 FSM 的状态机输出时，只需指定 case 敏感表为次态寄存器，然后直接在每个次态的 case 分支中描述该状态的输出即可，根本不用考虑状态转移条件。对于简单的 FSM，三段式建模的寄存器输出的优势还不是十分明显，但是对于复杂一些的 FSM，三段式建模的优势就会十分显著。

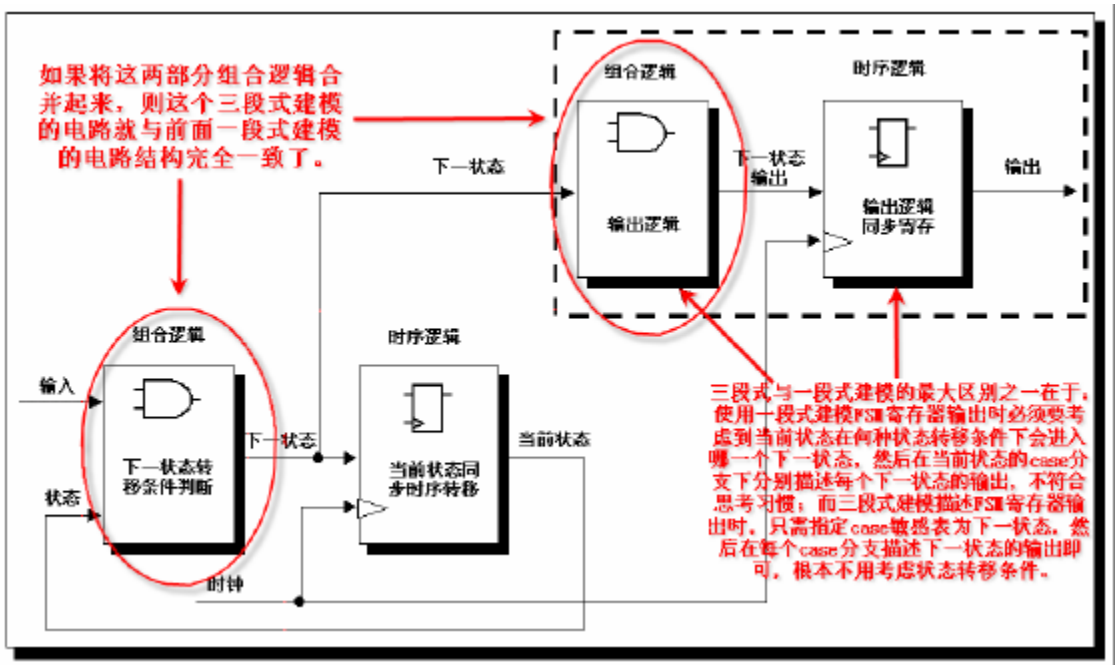
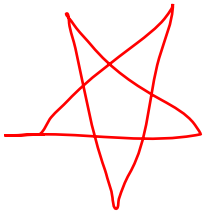


图6-7　三段式建模结构与一段式建模结构的关系图

- **两段式建模和三段式建模的关系**

从代码上看，三段式建模的前两段与两段式建模完全相同，仅仅多了一段寄存器 FSM 输出。一般来说，使用寄存器输出可以改善输出的时序条件，还能避免组合电路的毛刺，所以是更为推荐的描述方式。但是电路设计不是一成不变地，在某些情况下，两段式结构比三段式结构更有优势。请大家再分析一下图 6-5、图 6-6 的结构，细心的读者会发现，两段式用状态寄存器分割了两部分组合逻辑（状态转移条件组合逻辑和输出组合逻辑）；而三段式结构中，从输入到寄存器状态输出的路径上，要经过两部分组合逻辑（状态转移条件组合逻辑和输出组合逻辑），从时序上，这两部分组合逻辑完全可以看为一体。这样这条路径的组合逻辑就比较繁杂，该路径的时序相对紧张。也就是说，两段式建模中用状态寄存器分割了组合逻辑，而三段式将寄存器移到组合逻辑的最后端。如果寄存器前的组合逻辑过于复杂，势必会成为整个设计的关键路径，此时就不宜再使用三段式建模，而要使用两段式建模。解决两段式建模组合逻辑输出产生毛刺的方法是，额外的在 FSM 后级电路插入寄存器，调整时序，完成功能。



- **三种描述 FSM 方法的比较**

一般来说，三种 FSM 描述方法可以用表 6-1 进行比较。但是请读者注意，任何一种描述的优劣只是一般规律，而不是绝对性规律。例如一般来说不推荐一段式描述，但是如果 FSM 的结构十分简单，状态很少，状态转移条件和状态输出都十分简化，则使用一段式建模的效率很高。这些经验需要读者逐步积累，但是笔者在这里推荐的是一般性规律，请读者结合电路，体会这几种

FSM 建模方法。

表 6-1　　3 种 FSM 描述方法比较表

比较项目	一段式描述方法	两段式描述方法	三段式描述方法
推荐等级	不推荐	推荐	最优推荐
代码简洁程度（对于相对复杂的 FSM 而言）	冗长	最简洁	简洁
always 模块个数	1	2	3
是否利于时序约束	不利于	利于	利于
是否有组合逻辑输出	可以无组合逻辑输出	多数情况有组合逻辑输出	无组合逻辑输出
是否利于综合与布局布线	不利于	利于	利于
代码的可靠性与可维护度	低	高	最好
代码风格的规范性	低，任意度较大	格式化，规范	格式化，规范

6.2.4 状态机设计的其他技巧

本节讨论 FSM 设计的其他技巧。

- **FSM 的编码**

  Binary（二进制编码）、gray-code（格雷码）编码使用最少的触发器，较多的组合逻辑，而 one-hot（独热码）编码反之。one-hot 编码的最大优势在于状态比较时仅仅需要比较一个 bit，一定程度上从而简化了比较逻辑，减少了毛刺产生的概率。由于 CPLD 更多地提供组合逻辑资源，而 FPGA 更多地提供触发器资源，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，而大型状态机使用 one-hot 更高效。

  在代码中添加综合器的综合约束属性或者在图形界面下设置综合约束属性可以比较方便地改变状态的编码。需要注意的是：Synplicity、Synopsys、Exemplar 等综合工具关于 FSM 的综合约束属性的语法格式各不相同。

- **FSM 初始化状态**

  一个完备的状态机（健壮性强）应该具备初始化状态和默认状态。当芯片加电或者复位后，状态机应该能够自动将所有判断条件复位，并进入初始化状态。需要注明的一点是，大多数 FPGA 有 GSR（Global Set/Reset）信号，当 FPGA 加电后，GSR 信号拉高，对所有的寄存器、RAM 等单元复位/置位，这时配置于 FPGA 的逻辑并未生效，所以不能保证正确地进入初始化状态。所以使用 GSR 企图进入 FPGA 的初始化状态，常常会产生种种不必要的麻烦。一般的方法是采用异步复位信号，当然也可以使用同步复位，但是要注意同步复位逻辑的设计。解决这个问题的另一种方法是将默认的初始状态的编码设为全零，这样 GSR 复位后，状态机自动进入初始状态。如 6.2.3 节所示的编码方法

就是初始状态全零的 one－hot 编码方式。

- **FSM 状态编码定义**

  状态机的定义可以用 **parameter** 定义，但是不推荐使用`**define** 宏定义的方式，因为'**define** 宏定义在编译时自动替换整个设计中所定义的宏，而 **parameter** 仅仅定义模块内部的参数，定义的参数不会与模块外的其他状态机混淆。例如一个工程里面有两个 **module** 各包含一个 FSM，如果设计时都有 IDLE 这一名称的状态，如果使用'**define** 宏定义就会混淆起来，如果使用 **parameter** 则不会造成任何不良影响。

- **FSM 输出**

  如果使用 2 段式 FSM 描述 Mealy 状态机，输出逻辑可以用"？语句"描述，或者使用 case 语句判断转移条件与输入信号即可。如果输出条件比较复杂，而且多个状态共用某些输出，则建议使用 task/endtask 将输出封装起来，达到模块复用的目的。

- **阻塞和非阻塞赋值**

  为了避免不必要的竞争冒险，不论是做两段式还是三段式 FSM 描述时，必须遵循时序逻辑 **always** 模块使用非阻塞赋值“ <= ”，即当前状态向下一状态时序转移，和寄存 FSM 输出等时序 always 模块中都要使用非阻塞赋值；而组合逻辑 **always** 模块使用阻塞赋值“ = ”，即状态转移条件判断，组合逻辑输出等 always 模块中都要使用阻塞赋值。

- **FSM 的默认状态**

  完整的状态机应该包含一个默认（**default**）状态，当转移条件不满足，或者状态发生了突变时，要能保证逻辑不会陷入“ 死循环 ”。这是对状态机健壮性的一个重要要求，也就是常说的要具备“ 自恢复 ”功能。对应于编码就是对 **case** 和 **if…else** 语句要特别注意，尽量使用完备的条件判断语句。Verilog 中，使用 **case** 语句的时候要用 **default** 建立默认状态。读者可能注意到，在上节举例中的 **case** 语句中，我们没有写 **default** 默认状态，其实我们可以将其中一个状态不编码，指定其为 **default** 默认状态，则任何与所列状态机不匹配的状态都会转到 **default** 状态，从而增强了 FSM 的健壮性，另外我们也可以添加一个额外的 **default** 状态，这个一旦进入这个状态就会自动转到 IDLE 状态，从新启动状态机，这样做也增强了状态机的健壮性。

```
case (CS)
    IDLE:   begin
                IDLE_out;
                if (~i1)            NS = IDLE;
                 if (i1 && i2)      NS = S1;
                if (i1 && ~i2)      NS = ERROR;
             end
    S1:      begin
```

```
                    S1_out;
                    if (~i2)            NS = S1;
                    if (i2 && i1)       NS = S2;
                     if (i2 && (~i1))   NS = ERROR;
                end
        S2:     begin
                    S2_out;
                    if (i2)             NS = S2;
                    if (~i2 && i1)      NS = IDLE;
                     if (~i2 && (~i1))  NS = ERROR;
                end
        ERROR:  begin
                    ERROR_out;
                    if (i1)           NS = ERROR;
                    if (~i1)          NS = IDLE;
                end
        default: begin
                    IDLE_out;
                    NS = IDLE;
                end
    endcase
end
```

在 case 语句结构中增加 default 默认状态是推荐的代码风格。

- **Full Case 与 Parallel Case 综合属性**

所谓 Full Case 是指：FSM 的所有编码向量都可以与 case 结构的某个分支或 default 默认情况匹配起来。如果一个 FSM 的状态编码是 8bit，则对应的 256 个状态编码（全状态编码是 2^n 个）都可以与 case 的某个分支或者 default 映射起来。

所谓 Parallel Case 是指：在 case 结构中，每个 case 的判断条件表达式（即本章 6.2.2 小节描述的 case_expression），有且仅有唯一的 case 语句的分支（即本章 6.2.2 小节描述的每个 case_item）与之对应，即两者关系是一一对应关系。

目前知名综合器如 Synplify Pro、Precision RTL 和 Synopys 综合工具等都支持“ synthesis full_case”和“ synthesis parallel_case”这些综合约束属性，合理使用 Full Case 约束属性，可以增强设计的安全性；合理使用 Parallel Case

约束属性，可以改善状态机译码逻辑。但是设计者必须具体情况具体分析，对于有的设计，不当使用这两条语句，会占用大量逻辑资源，并恶化 FSM 的时序表现。

6.3 使用 Synplify Pro 分析 FSM

代码走读时分析 FSM 是一件比较耗时的事情，如果代码不符合本章 6.2 节所述的两段式或三段式 FSM 描述规范，走读他人代码则是一件异常痛苦的事情。这里笔者以 Synplify Pro 为例，介绍一下如何利于 EDA 工具分析 FSM，综合 FSM，提高 FSM 性能。

Synplify Pro 提供了 3 个有限状态机设计工具，FSM Compiler、FSM Explorer 和 FSM Viewer，灵活地使用这 3 个有限状态机工具分析、编译、优化 FSM，可使 FSM 的综合结果达到最优。下面逐一讨论它们的使用方法。

(1) 有限状态机编译器（FSM Compiler）。

一般的综合工具将 FSM 按照普通逻辑综合，而 Synplify Pro 与之不同。Synplify Pro 使用 FSM Compiler，先将 FSM 编译为类似状态转移图的连接图，然后对 FSM 重新编码、优化以达到更好的综合效果。

FSM Compiler 适应于有以下需求的场合：需要优化 FSM 设计，达到更好的综合效果；使用 FSM Viewer 调试状态机；使用 FSM Explorer 进一步优化有限状态机。

FSM Complier 的使用非常灵活，可以对整个设计的所有状态机都用 FSM Compiler 进行优化，也可以仅仅对指定的状态机进行优化。对整个设计使用 FSM Compiler 进行优化，只需在主界面重要综合优化参数中选择【FSM Compiler】选项，或者在综合优化参数设置时选中【FSM Compiler】选项即可。如果觉得设计其他 FSM 已经满意，而仅对某个 FSM 不满意时，可以在源代码或综合约束文件中手动添加综合属性，指定对单独状态机的编译与优化。FSM Compiler 综合属性如表 6-2 所示。

表 6-2　　FSM Compiler 综合属性用法

语法 / 语言	禁止优化所指定的 FSM	优化所指定的 FSM
Verilog	reg [3:0] curstate /* synthesis syn_state_machine=0 */ ;	reg [3:0] curstate /* synthesis syn_state_machine=1 */

(2) 有限状态机探测器（FSM Explorer）。

FSM Explorer 使用 FSM Compiler 的编译结果，鳞选不同的编码方式进行状态机编码试探，从而达到对 FSM 编码的最佳优化效果。与 FSM Compiler 相比，FSM Explorer 的优化效果往往更好，编译优化所花费的时间也更长。

对设计使用 FSM Explorer 的方法也有两种：第一种是对整个设计的所有 FSM 自动运用 FSM Explorer；第二种是对设计中特定的 FSM 使用 FSM Explorer。第一种方法可以在 Synplify Pro 主界面重要综合优化参数中选择 FSM Explorer 有效，或者在综合优化参数设置对话框选中【FSM Explorer】选项；第二种方法需要在源代码或者综合约束文件中添加使用 FSM Explorer 的综合属性声明。FSM Explorer 综合属性如表 6-3 所示。

(3) 有限状态机观察器（FSM Viewer）。

在 Synplify Pro 中除了可以使用 RTL 视图和结构视图观察、分析 FSM 外，还可以使用专用 FSM 观察器（FSM Viewer）分析 FSM。FSM Viewer 将源代码中描述的 FSM 根据 FSM Compiler 和 FSM Explorer 的编译优化结果，用状态转移图显示有限状态机。

表 6-3　　FSM Explorer 综合属性用法

语法 / 语言	手动指定 FSM 是否优化	手动指定 FSM 编码方式
Verilog	reg [3:0] curstate /* synthesis state_machine */ ;	reg [3:0] curstate /* synthesis syn_encoding “ gray” */

下面以 6.2.3 节举例的 FSM 为例，讲述 FSM Viewer 的使用方法。

【例1-2】 使用 FSM Viewer 分析有限状态机，参考示例详见本书附带光盘的“ Example-6-1 ”目录。

1. 启动 Synplify Pro，单击 Open Project... 打开“ Example-6-1\FSM\state2 ”目录下的“ state2.prj ”，单击 按钮启动 RTL 视图，选择状态机模块“ statemachine ”，单击 按钮进入状态机层次结构，或者单击鼠标右键，在弹出命令菜单中选择【View FSM】命令，如图 6-7 所示。

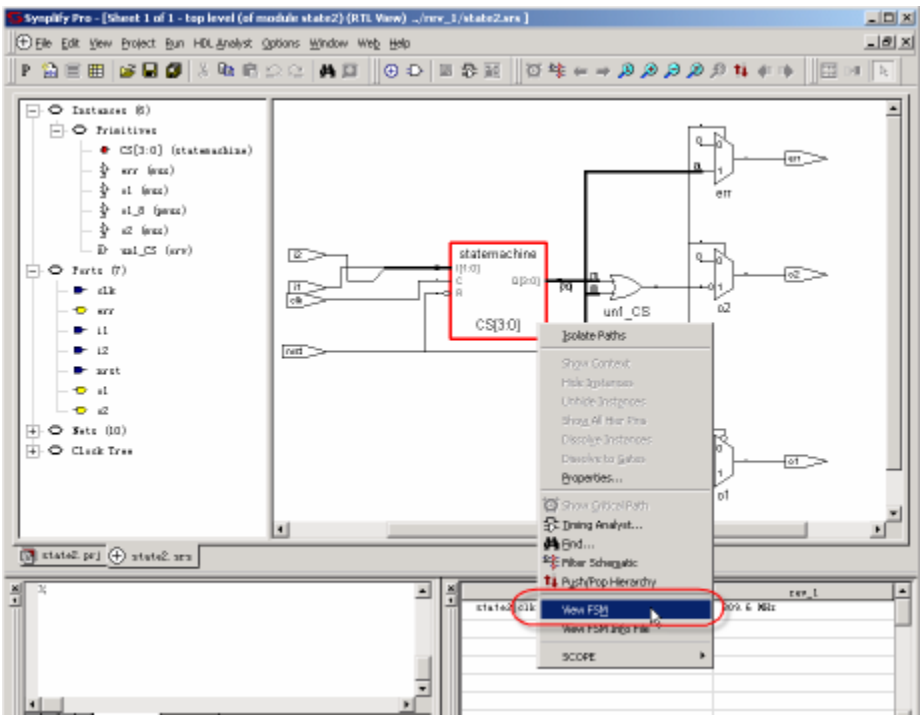


图6-8　启动 FSM Viewer 分析有限状态机

FSM Viewer 的主界面主要由状态转移图和 FSM 信息显示选项卡组成。状态转移图是源代码经过编译再现的状态机。FSM 信息显示包含转移条件（Transitions）、寄存器传输级状态编码（RTL Encodings）和映射后状态编码

（Mapped Encodings）等 3 个选项卡，如图 6-8 所示。

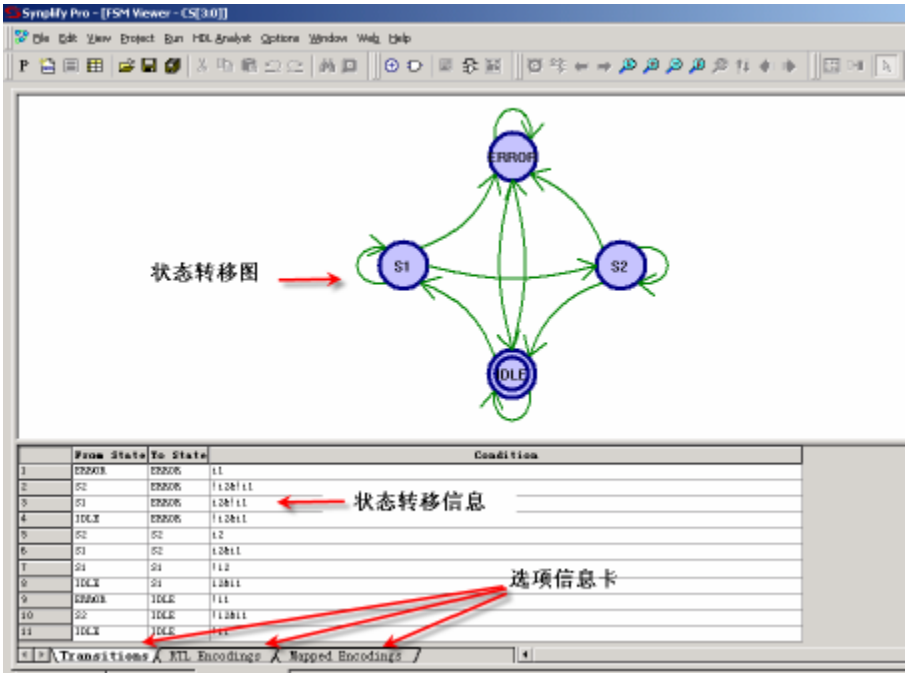


图6-9　FSM Viewer 主界面

选择某个状态，单击鼠标右键，在弹出的菜单中可以完成显示对象的选择和屏蔽，有利于理解状态之间关系，增加状态转移图的可读性，如图 6-9 所示。

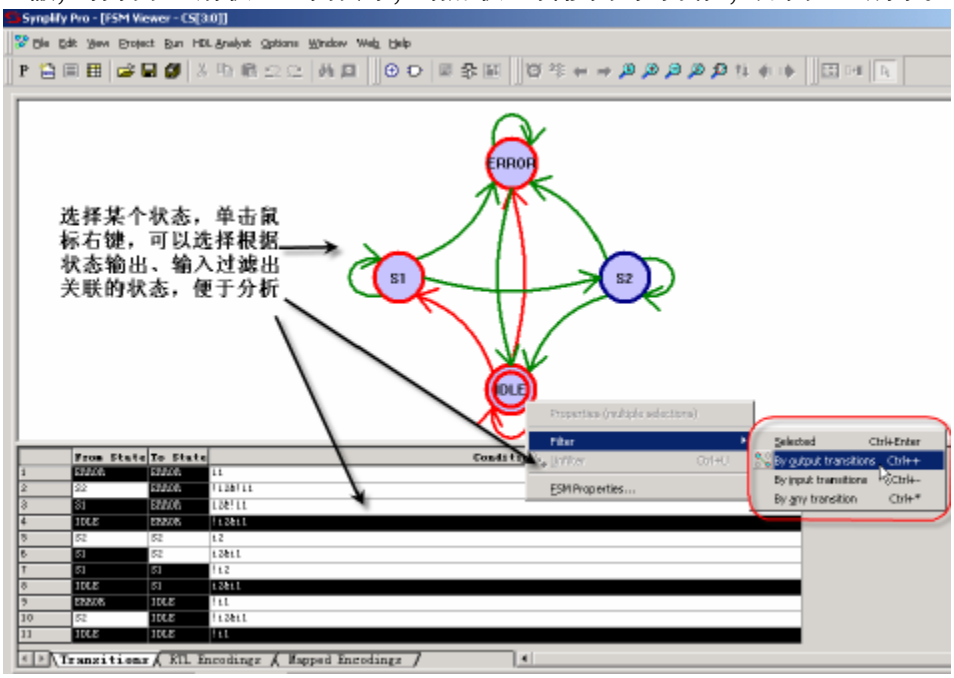


图6-10　在 FSM Viewer 中选择和屏蔽状态

6.4 小结

状态机不仅仅是一种时序电路设计工具，它更是一种思想方法。状态机的本质就是对具有逻辑顺序或时序规律事件的一种描述方法。这个论断的最重要的两个词就是“逻辑顺序”和“时序规律”，这两点就是状态机所要描述的核心和强项，换言之，所有具有逻辑顺序和时序规律的事情都适合用状态机描述。

根据 FSM 描述使用的 always 模块数和功能可以将 FSM 的描述分为 3 中写法：

- 不推荐使用一段式描述方法。因为一段式描述方法将状态转移判断的组合逻辑和状态寄存器转移的时序逻辑混写在同一个 always 模块中，不符合将时序和组合逻辑分开描述的 Coding Style（代码风格），而且在描述当前状态时要考虑下个状态的输出，整个代码不清晰，不利于维护修改，并且不利于附加约束，不利于综合器和布局布线器对设计的优化。
- 推荐使用两段式状态机描述方法。两段式描述方法用 2 个 always 模块，其中一个 always 模块采用同步时序描述状态转移；另一个模块采用组合逻辑判断状态转移条件，描述状态转移规律。这种方法使 FSM 描述清晰简介，易于维护，易于附加时序约束，使综合器和布局布线器更好的优化设计。
- 强烈推荐使用三段式描述方法。与一段式描述方法相比较，三段式 FSM 描述方法对 FSM 寄存器输出的描述只需判断下一状态，然后直接将下一状态的输出用寄存器输出即可，根本不用考虑状态转移条件（米勒状态机）。与两段式描述相比，三段式虽然代码结构复杂了一些，但是换来的优势是使 FSM 做到了同步寄存器输出，消除了组合逻辑输出的不稳定与毛刺的隐患，而且更利于时序路径分组，一般来说在 FPGA/CPLD 等可编程逻辑器件上的综合与布局布线效果更佳。

6.5 问题与思考

1. 简述状态机的本质和适应的逻辑设计场合。
2. 状态机的基本要素有那些？
3. 两段式、三段式 FSM 描述方法的基本结构如何？
4. FSM 描述何时使用阻塞赋值，何时使用非阻塞赋值？
5. 三段式 FSM 描述的两个 case 结构中判断表达式与当前状态和下一个状态寄存器的对应关系如何。