      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。

//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  
 default: begin{o1,o2,err}<=3'b000;NS <= IDLE; end  
   
 endcase  
 end  
endmodule  
  
//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;  
   
 always @ (posedge clk or negedge nrst)//sequential state transition  
 if (!nrst)  
 CS <= IDLE;  
 else  
 CS <=NS;  
   
 always @ (CS or i1 or i2) //combinational condition judgment  
 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  
 default: begin  
 IDLE\_out;  
 NS = IDLE;  
 end  
 endcase  
 end  
   
 task IDLE\_out;  
 {o1,o2,err} = 3'b000; //output task  
 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  
  
//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 state3 ( 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;  
   
 always @ (posedge clk or negedge nrst) //1st always block, sequential state transition  
 if (!nrst)  
 CS <= IDLE;  
 else  
 CS <=NS;  
   
 always @ (nrst or CS or i1 or i2) //2nd always block, combinational condition judgment  
 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  
 default: NS = IDLE;  
 endcase  
 end  
   
 always @ (posedge clk or negedge nrst) //3rd always block, the sequential FSM output  
 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