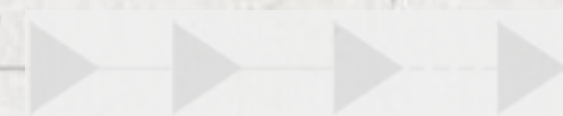


芯动力——硬件加速设计方法

第三章 同步电路与跨时钟域电路设计(7)

邸志雄@西南交通大学

zxdi@home.swjtu.edu.cn

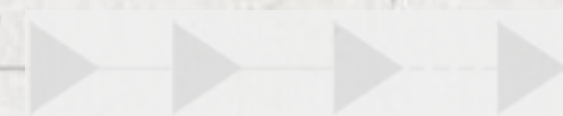



如何写好状态机

两段式状态机描述方法

两段式FSM 的核心:

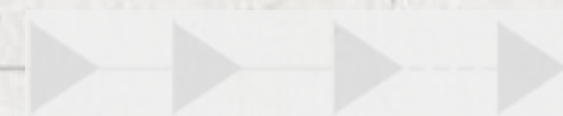
- 一个 always 模块采用同步时序描述状态转移
- 另一个模块采用组合逻辑判断状态转移条件，描述状态转移规律。





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

同步时序描述状态转移

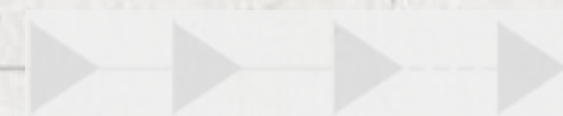


组合逻辑判断状态转移条件

```
//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
    end
```

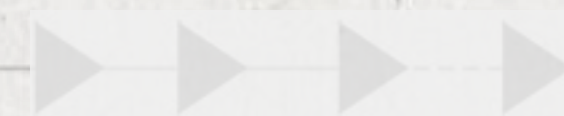

组合逻辑判断状态转移条件

```
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
```

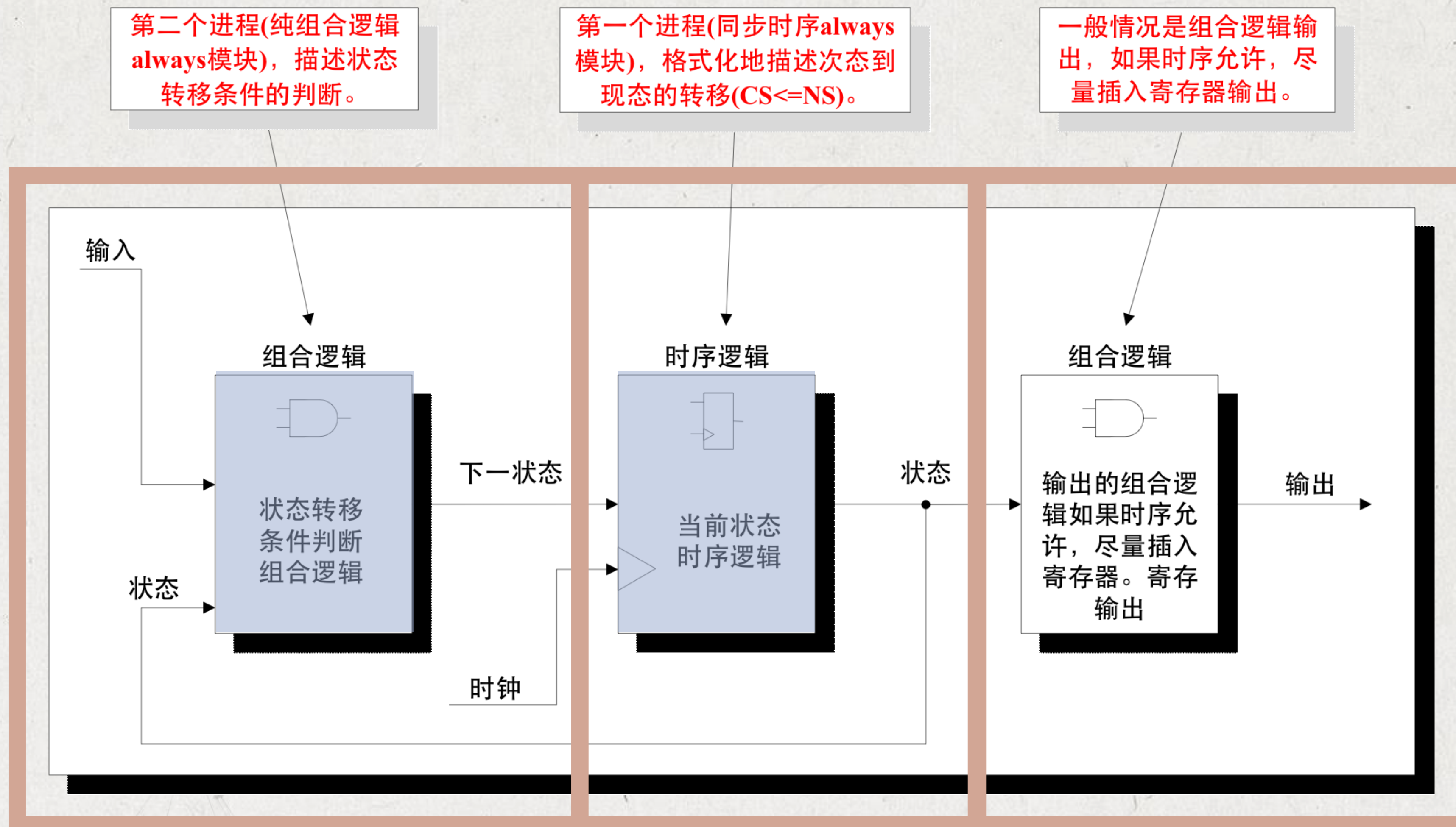



```
//output task
task IDLE_out;
    {o1,o2,err} = 3' b000;
endtask
task S1_out;
    {o1,o2,err} = 3' b100;
endtask
task S2_out;
    {o1,o2,err} = 3' b010;
endtask
task ERROR_out;
    {o1,o2,err} = 3' b111;
endtask
```

组合逻辑



两段式状态机描述方法

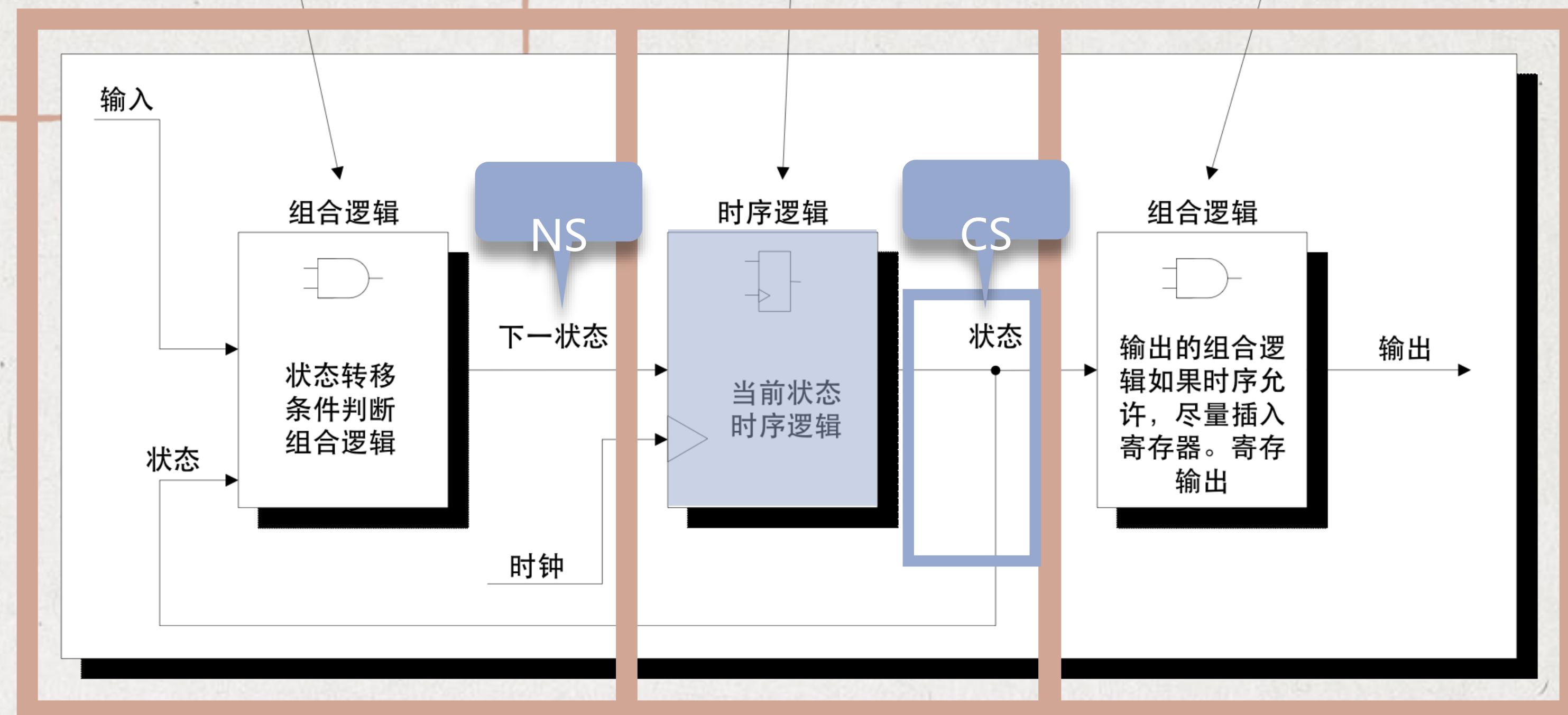



```
//sequential state transition
always @ (posedge clk or negedge nrst)
    if (!nrst)
        CS <= IDLE;
    else
        CS <= NS;
```

第二个进程(纯组合逻辑always模块), 描述状态转移条件的判断。

第一个进程(同步时序always模块), 格式化地描述次态到现态的转移(CS<=NS)。

一般是组合逻辑输出, 如果时序允许, 尽量插入寄存器输出。




```
//combinational condition judgment  
always @ (CS or i1 or i2)
```

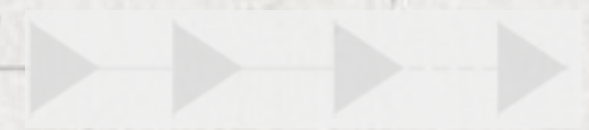
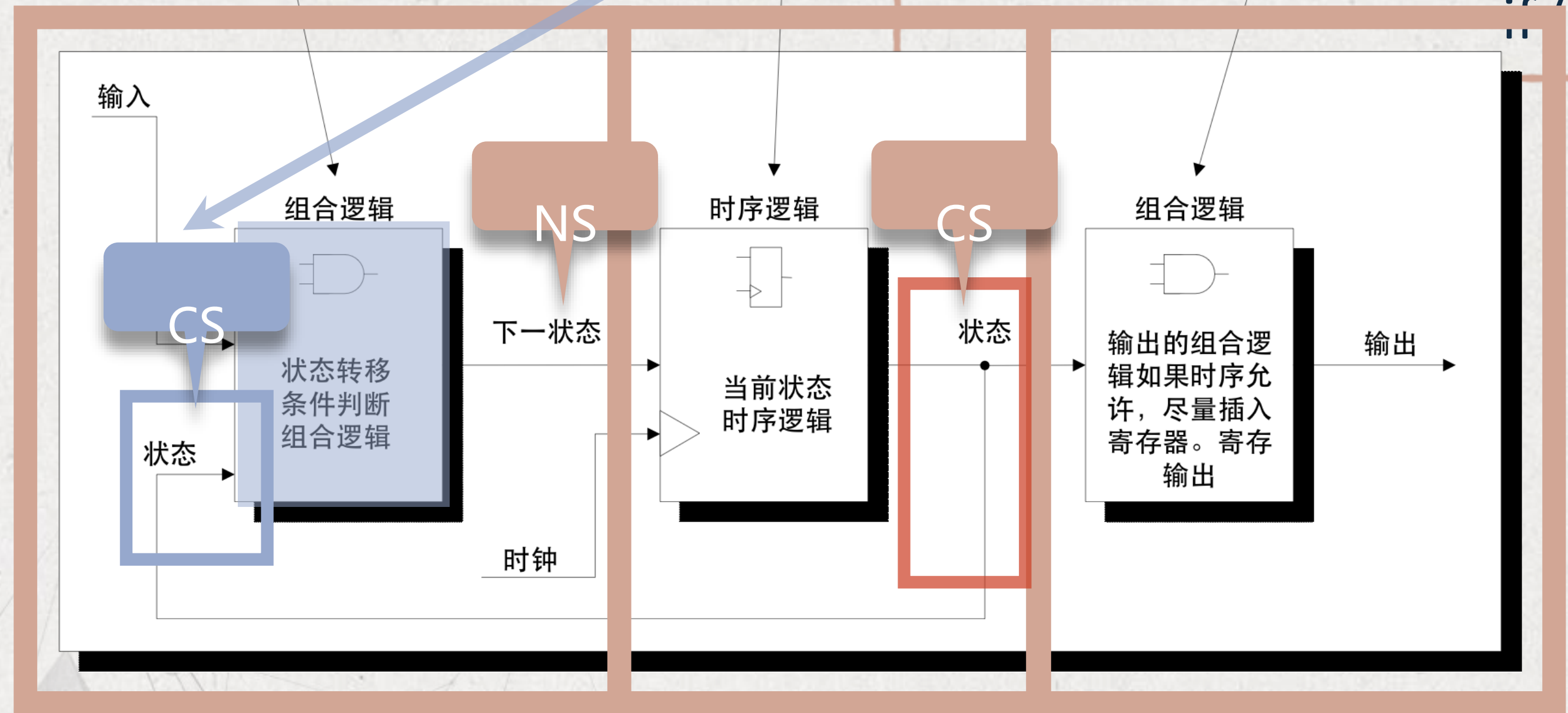
```
begin  
    NS = 3' bx;  
    ERROR_out;  
    case (CS);  
        IDLE: begin
```

```
            IDLE_out;  
            if (~i1)  
                NS = IDLE;  
            if (i1 && i2)  
                NS = S1;
```

第二个进程(纯组合逻辑
always模块), 描述状态
转移条件的判断。

第一个进程(同步时序always
模块), 格式化地描述次态到
现态的转移(CS<=NS)。

一般情况是组合逻辑输
出, 如果时序允许, 尽
量插入寄存器输出。




```
//combinational condition judgment
always @ (CS or i1 or i2)
```

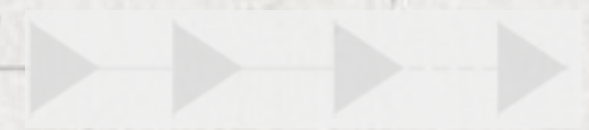
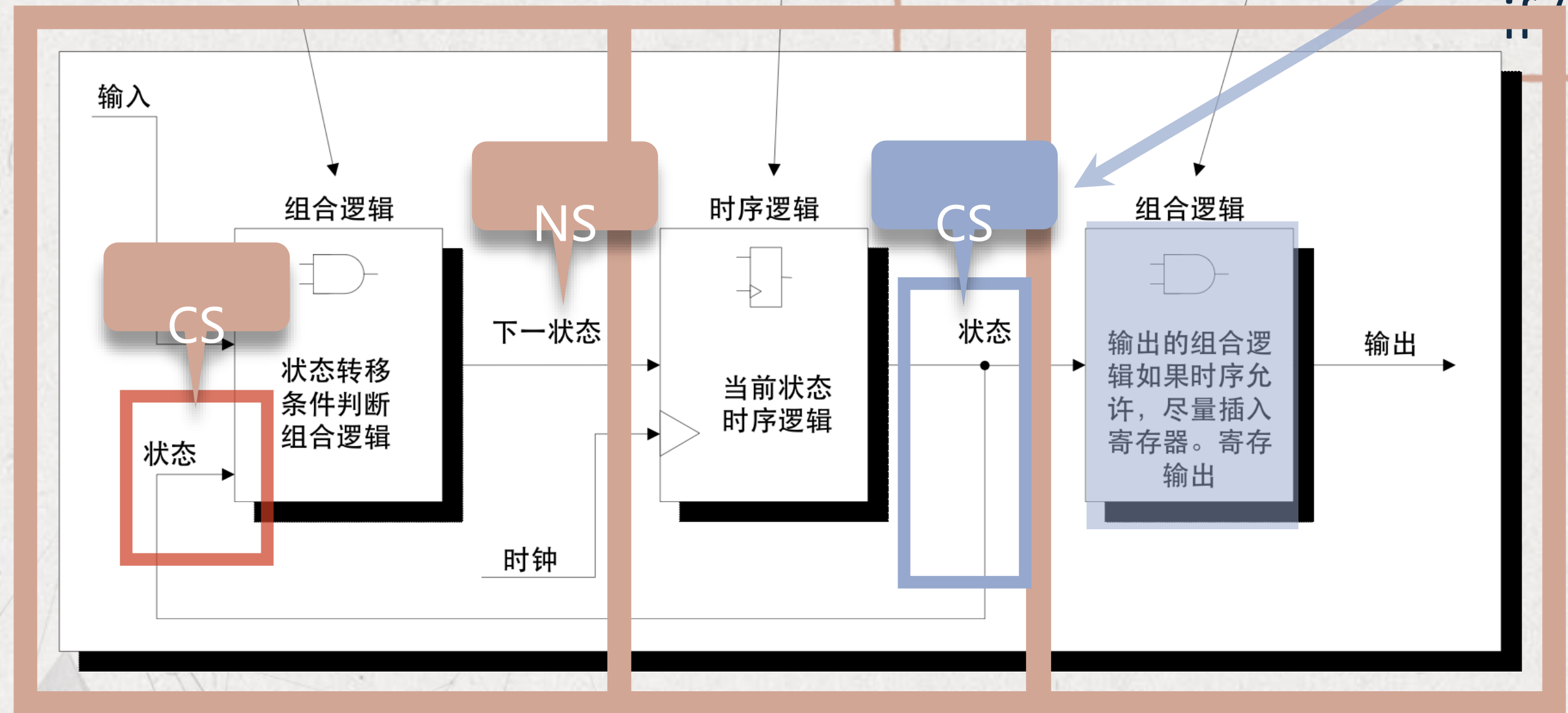
```
begin
    NS = 3' bx;
    ERROR_out;
    case (CS);
        IDLE: begin
```

```
            IDLE_out;
            if (~i1)
                NS = IDLE;
            if (i1 && i2)
                NS = S1;
```

第二个进程(纯组合逻辑always模块), 描述状态转移条件的判断。

第一个进程(同步时序always模块), 格式化地描述次态到现态的转移(CS<=NS)。

一般是组合逻辑输出, 如果时序允许, 尽量插入寄存器输出。



两段式 FSM 描述方法

弱点

- 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。

两段式 FSM 描述方法

FSM 的组合逻辑输出



寄存器寄存一个节拍

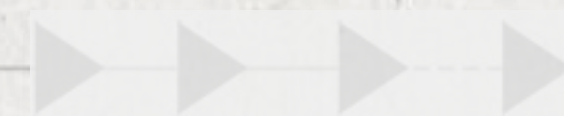
有效地消除毛刺

- 如果时序允许插入一个额外的时钟节拍

三段式 FSM 描述方法

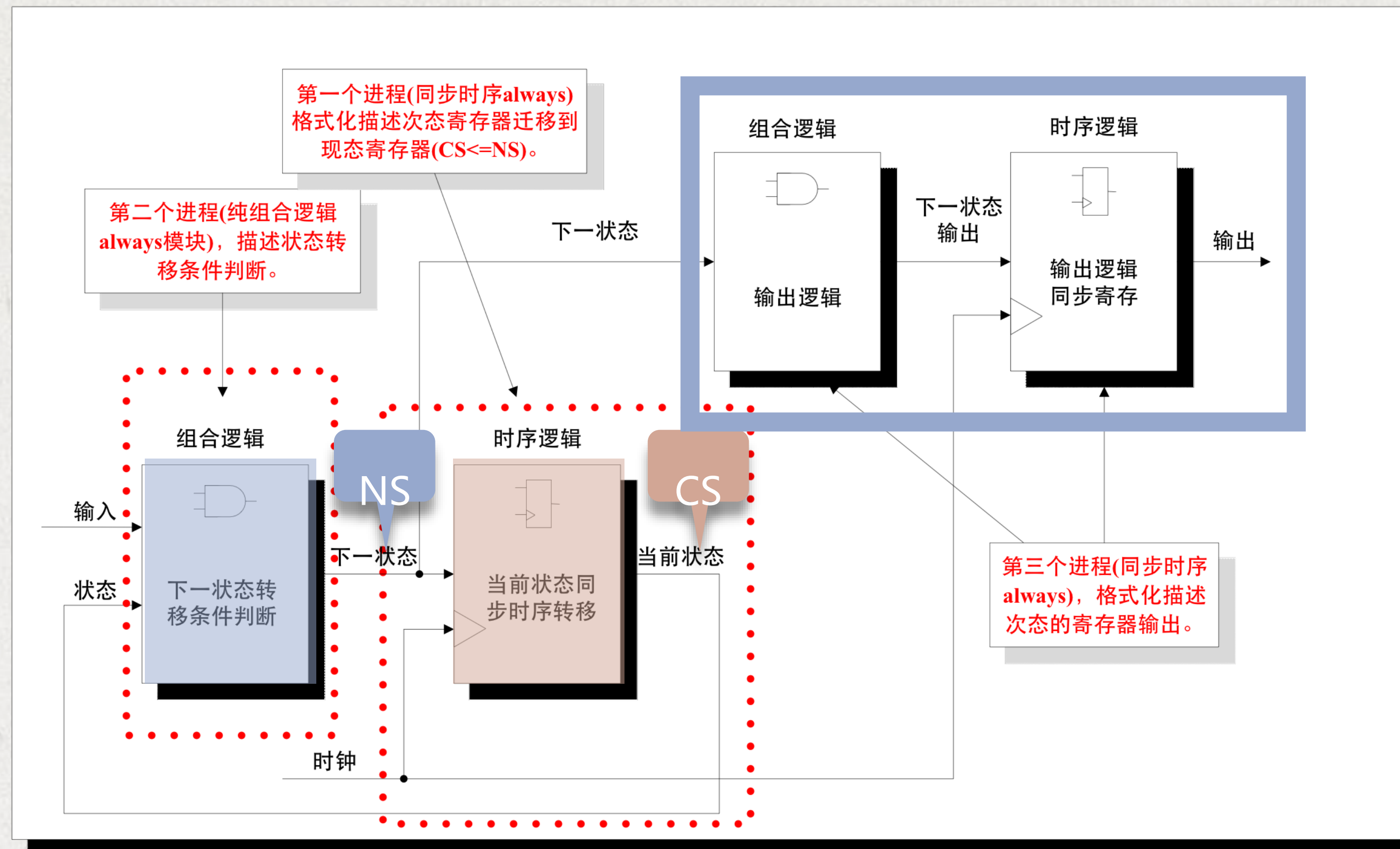
- 设计并不允许额外的节拍插入 (Latency)

关键在于使用同步时序逻辑寄存 FSM 的输出



两段式 FSM 描述方法 • 组合逻辑输出

三段式 FSM 描述方法 • 同步时序逻辑寄存 FSM 的输出



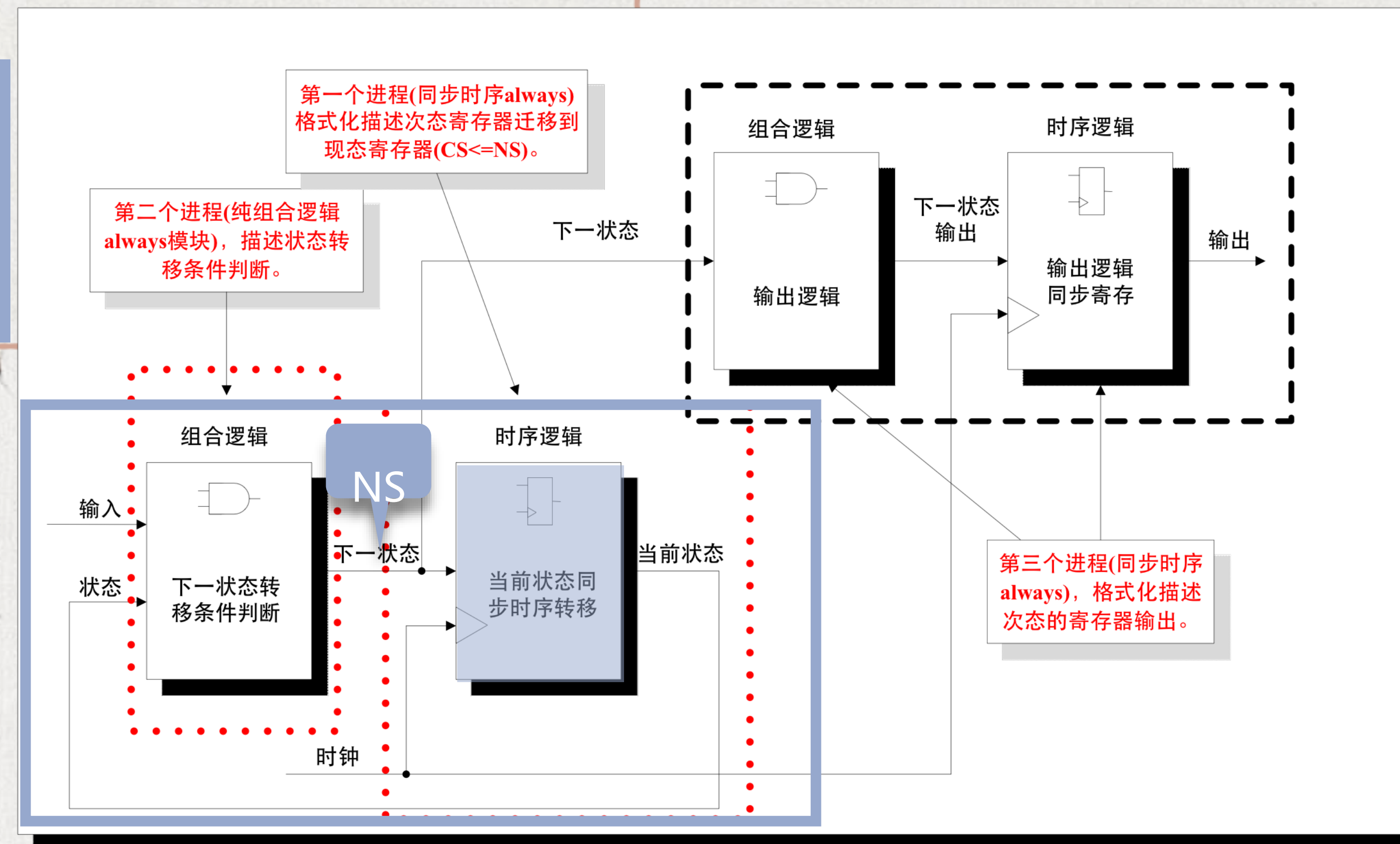

```
//1st always block, sequential state transition  
always @ (posedge clk or negedge nrst)
```

```
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)
```

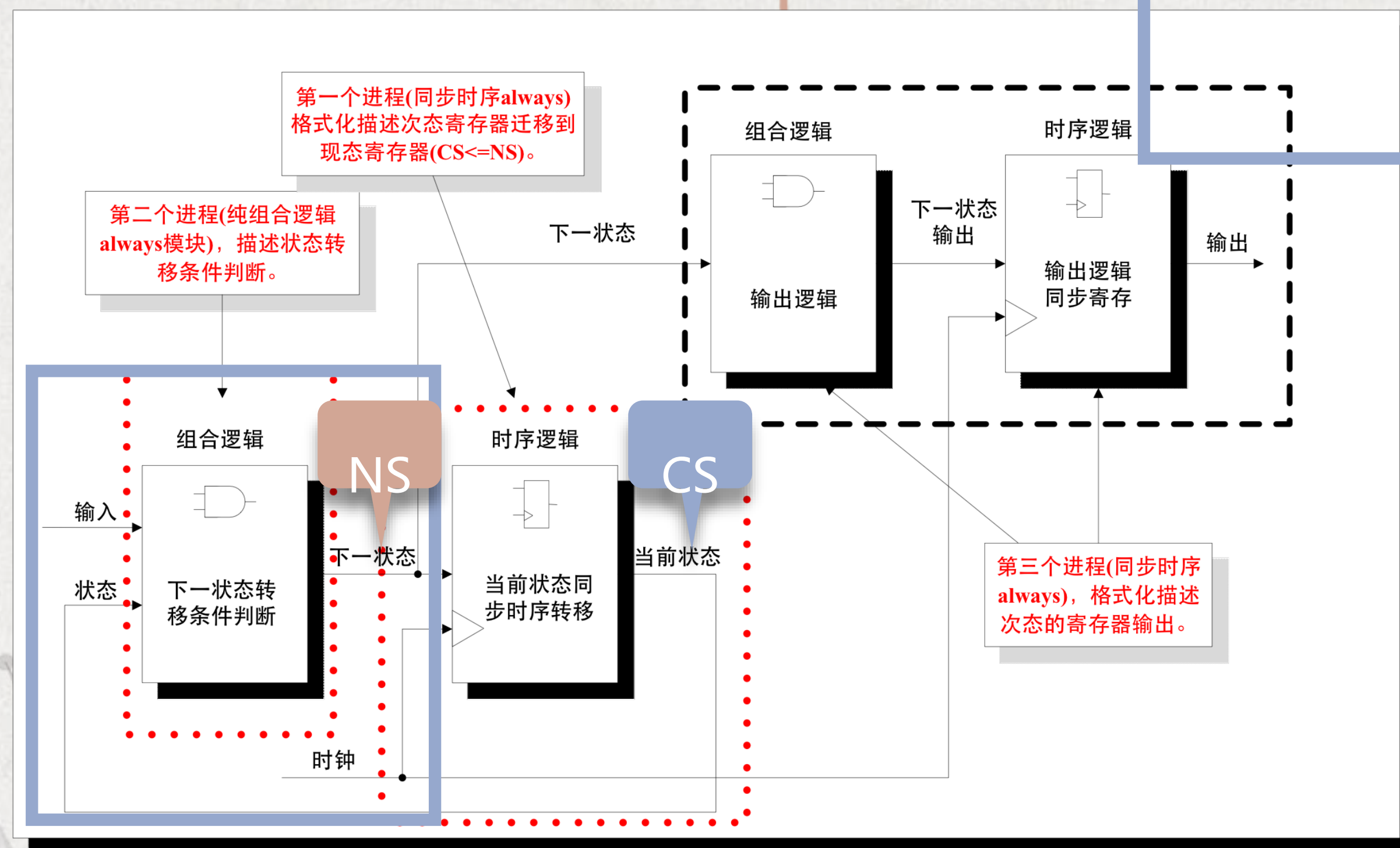
```
  IDLE: begin
```

```
    if {~i1}
```

```
      NS = IDLE;
```

```
    if {i1 && i2}
```

```
      NS = S1;
```




```
//3rd always block, the sequential PSM 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;
```

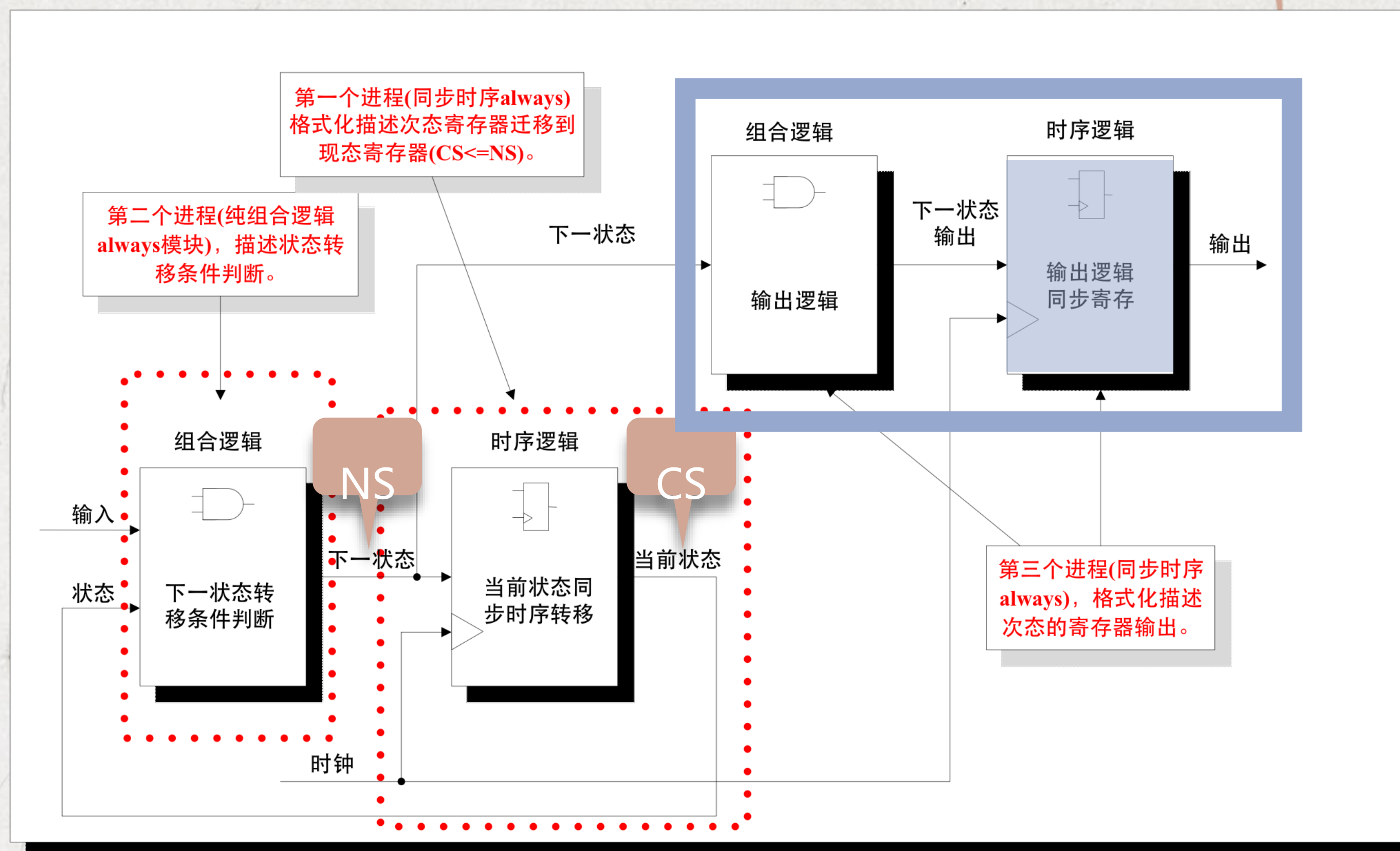
```
  S1: {o1,o2,err} <= 3' b100;
```

```
  S2: {o1,o2,err} <= 3' b010;
```

```
  ERROR: {o1,o2,err} <= 3' b111;
```

```
endcase
```

```
end
```



代码结构复杂了一些

三段式描述方法

两段式描述

优势

- 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
- 更利于时序路径分组
- 在 FPGA/CPLD 等可编程逻辑器件上的综合与布局布线效果更佳

三种描述方法的引申

n 段式描述方法和 always 语法块的个数



n 段式描述方法和 always 语法块的个数

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

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

语法角度上

always 模块



always 模块

always 模块

always 模块

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建模方式

- n 段式描述方法其实是 FSM 的三种建模方式

两段式思路建模

结构清晰

描述简洁

便于约束

- 如果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；

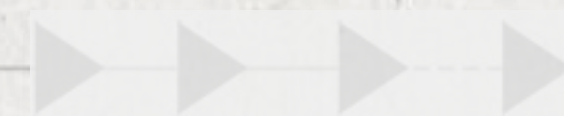
三段式思路建模

结构清晰

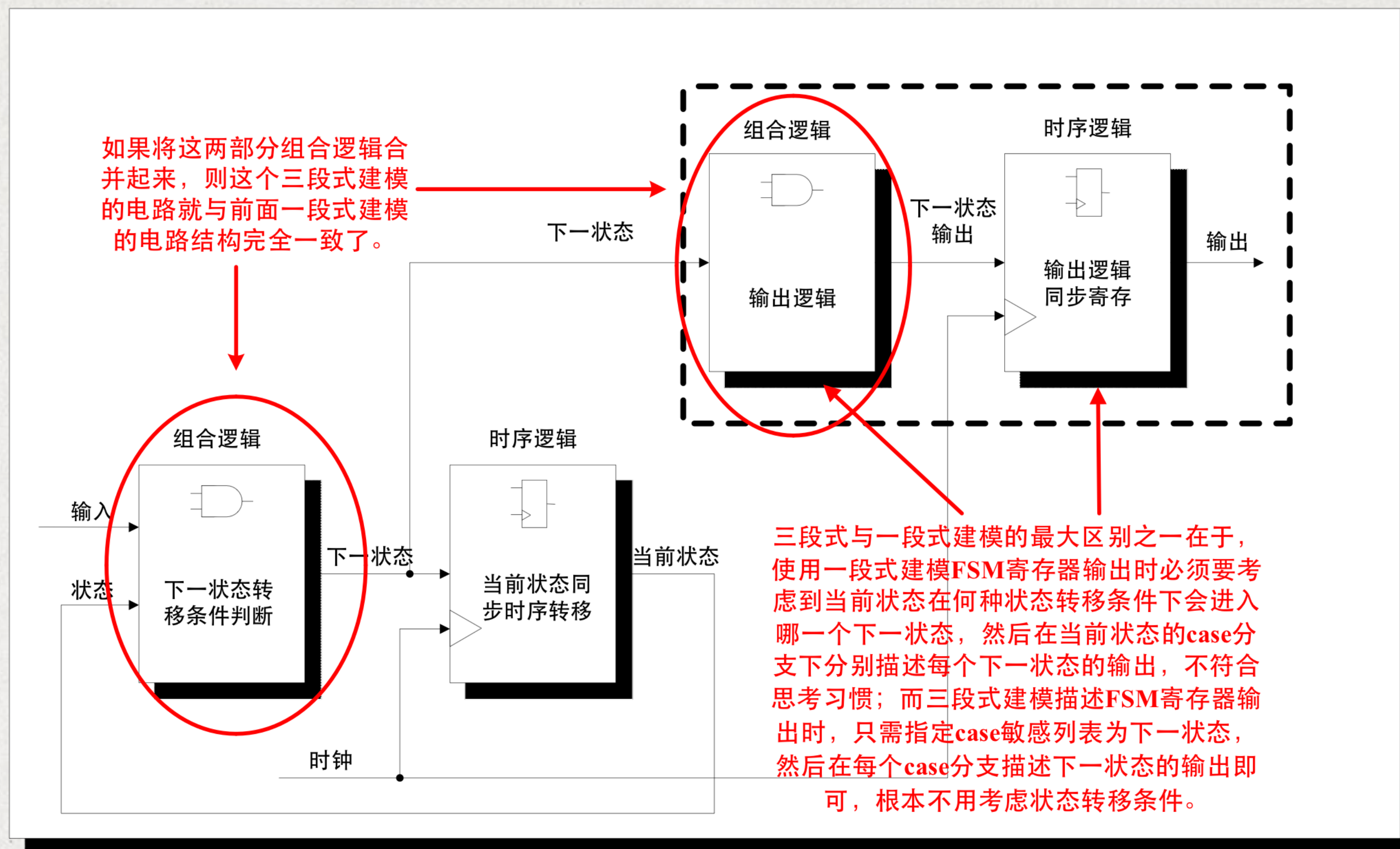
格式化的结构

- 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。

一段式思路建模



一段式建模和三段式建模的关系



一段式建模和二段式建模的关系

一段式建模

FSM 的寄存器输出

[不符合思维习惯]

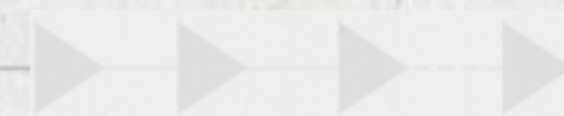
- 必须要综合考虑现态在何种状态转移条件下会进入哪些次态
- 然后在每个现态的 case 分支下分别描述每个次态的输出

二段式建模

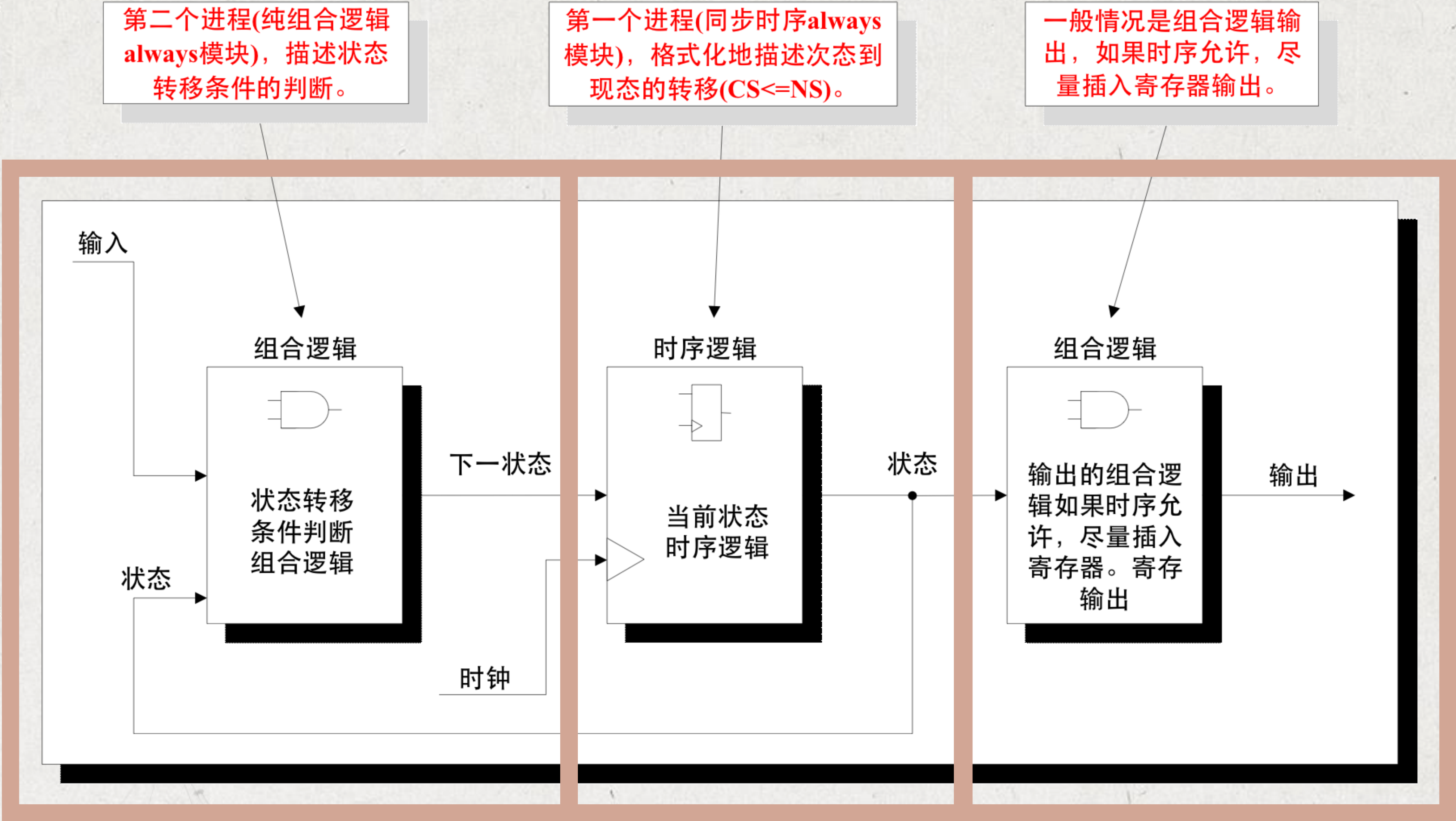
FSM 的状态机输出

- 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]



两段式建模和三段式建模的关系





两段式建模和三段式建模的关系

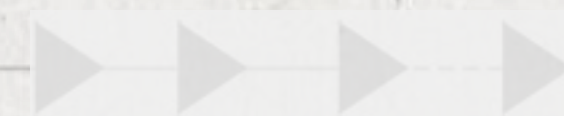
两段式建模

- 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。

三段式建模

寄存器 FSM 输出

- 可以改善输出的时序条件
- 还能避免组合电路的毛刺
- 是更为推荐的描述方式



两段式建模和三段式建模的关系

两段式建模

- 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

输出组合逻辑

电路时序路径较短，可以获得较高的性能;

三段式建模

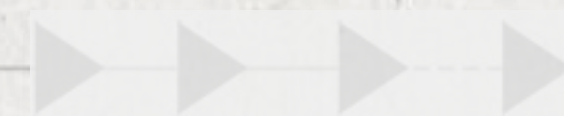
- 从输入到寄存器状态输出的路径上，要经过两部分组合逻辑

状态转移条件组合逻辑

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。

这条路径的组合逻辑就比较繁杂，该路径的时序相对紧张。



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