咋写Verilog

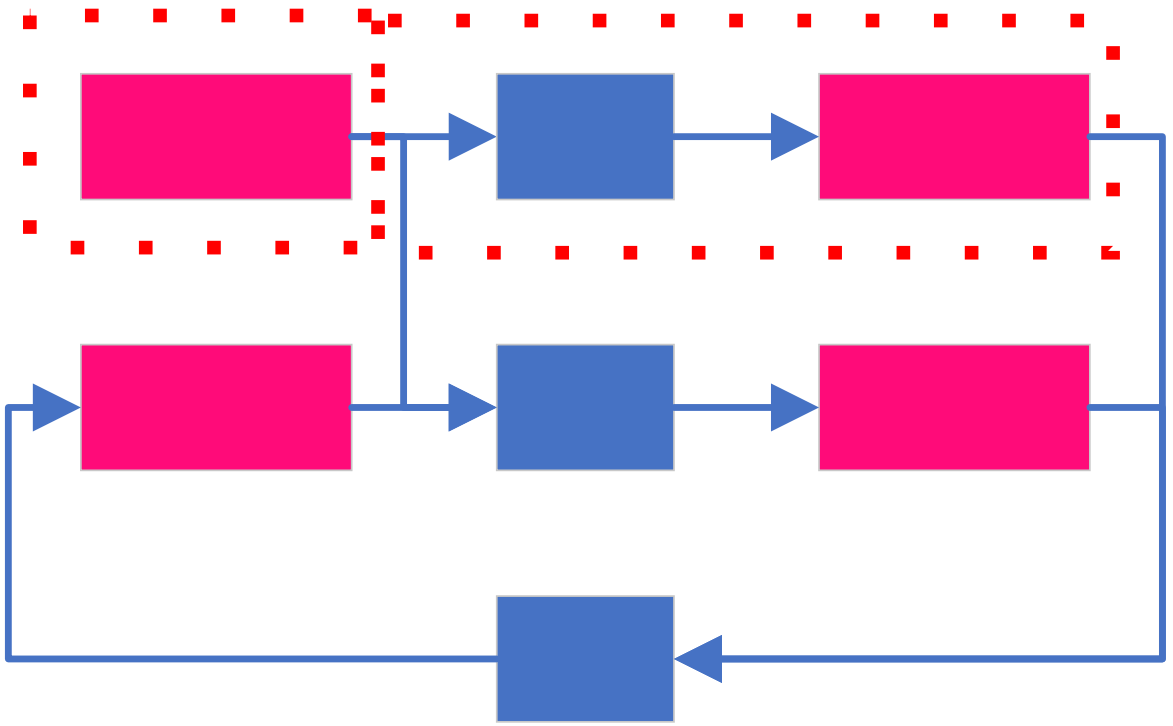
许嘉帆

软件&硬件

- 软件
  - 写文章
  - 执行顺序有先后
- 硬件（时序设计）
  - 画图
  - 执行顺序无先后
  - 代码块的顺序不重要，重要的是每块代码要写清楚

逻辑电路和时序电路

- 逻辑电路
  - 类似软件的逻辑
  - 赋值有先后顺序
- 测试电路
  - 类似软件逻辑，有先后顺序
- 时序电路
  - 主要由Reg和逻辑电路组成
  - 编写以always块为单位
  - 每个块编写一个(或多个)reg的前驱电路



reg变量和wire变量的区别

- reg能够在always语句块中进行赋值，wire不能
- assign和wire只是对变量的重命名
- reg变量不都会变成register
  - 通常在设计声明时就已经确认这个变量是啥类型，一般在真reg的变量后添加_r来做一个标记，比如:state_r
  - 真reg变量的赋值全部都使用阻塞赋值，假reg变量的赋值全部都使用非阻塞赋值
- 事实上，大家可以把所有实体对应变量分为register和wire，其中register的声明一定是reg，而wire的命名可能是wire，也可能是reg。

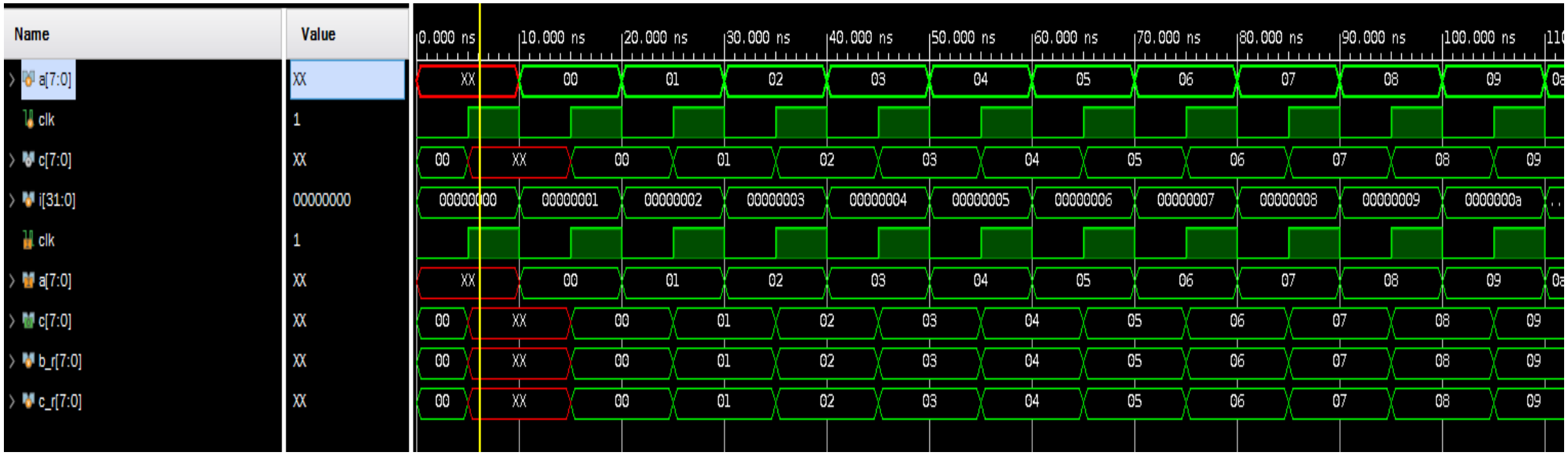
重点

- 一个Reg变量的赋值不能出现在2个always块中
- 一个always语句块中不要出现2种赋值
- 一个Reg变量不要出现2种赋值
- 对于reg类型最后变成wire的变量，需要对出现的所有情况考虑，再赋值，不允许出现某个if下面没有对其赋值。
- always @(posedge clk) 块下不要出现 =
- 一个always要么用于描述时序电路，要么用于描述逻辑电路
  - always @(posedge clk) begin ... end
  - always @(*) begin ... end

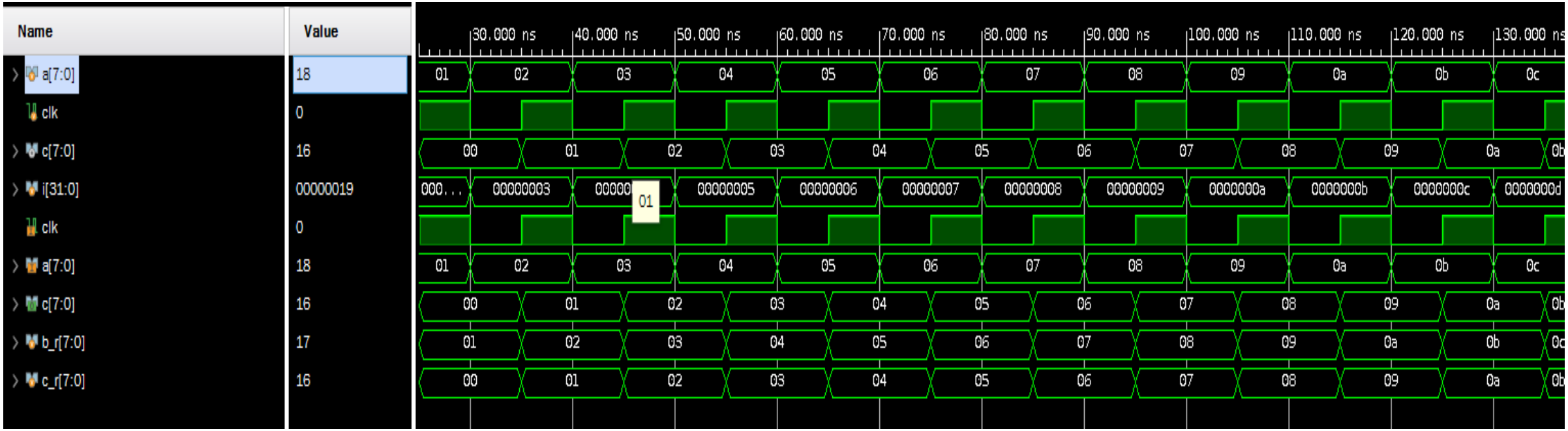
阻塞赋值 = 和非阻塞赋值 <=

- 阻塞赋值有先后
- 非阻塞赋值在时钟上升沿事件结束后统一赋值

```
input [7:0]a;
reg [7:0]b; reg [7:0]c;
always @(posedge clk) begin
        b = a;
        c = b;
end
```



```
input [7:0]a;
reg [7:0]b; reg [7:0]c;
always @(posedge clk) begin
        b <= a;
        c <= b;
end
```



always @(posedge clk) 不要出现 =

- https://blog.csdn.net/weixin_38679924/article/details/102524574

Verilog中的for循环

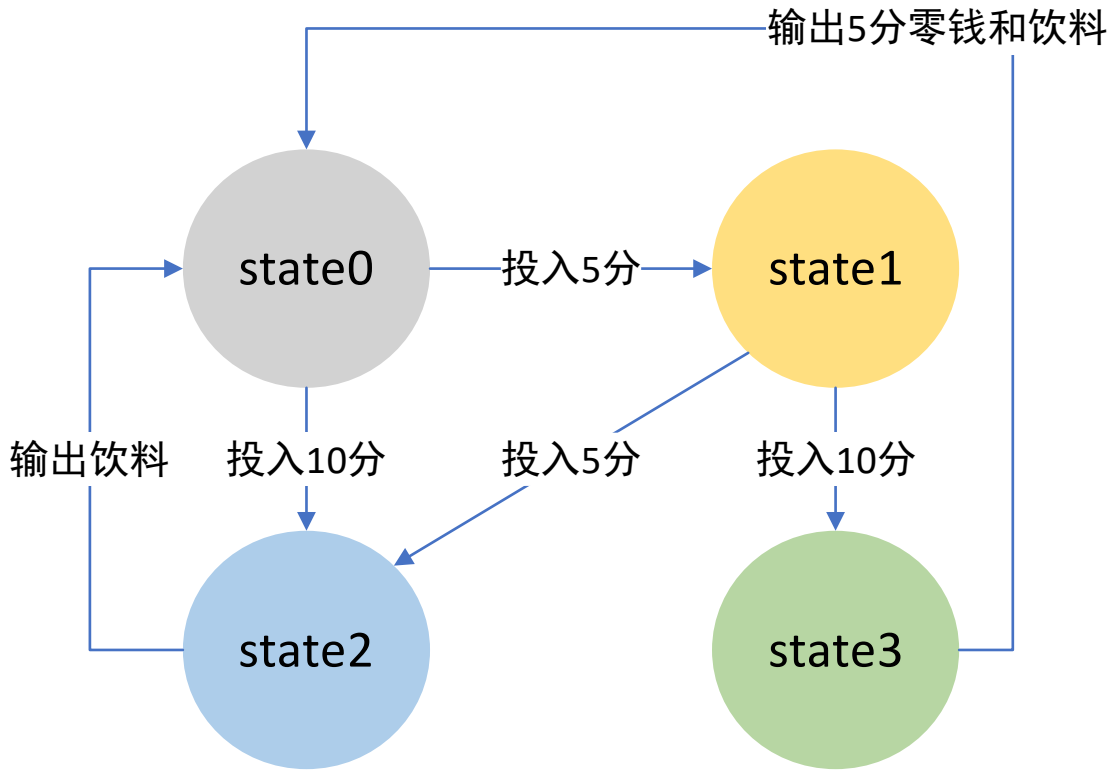
- 设计阶段的for循环只是为了简化很多条类似的语句，不要将其和软件代码中的for循环搞混
  - 前导0

- 更多的for循环代码是用于编写测试代码时（类似软件）

代码鉴赏环节

单状态机设计

- https://www.educoder.net/tasks/lmy39o4sfeu8
- 设计一个自动饮料售卖机，饮料10分钱，硬币有5分和10分两种，并考虑找零。
  - a.画出fsm（有限状态机）

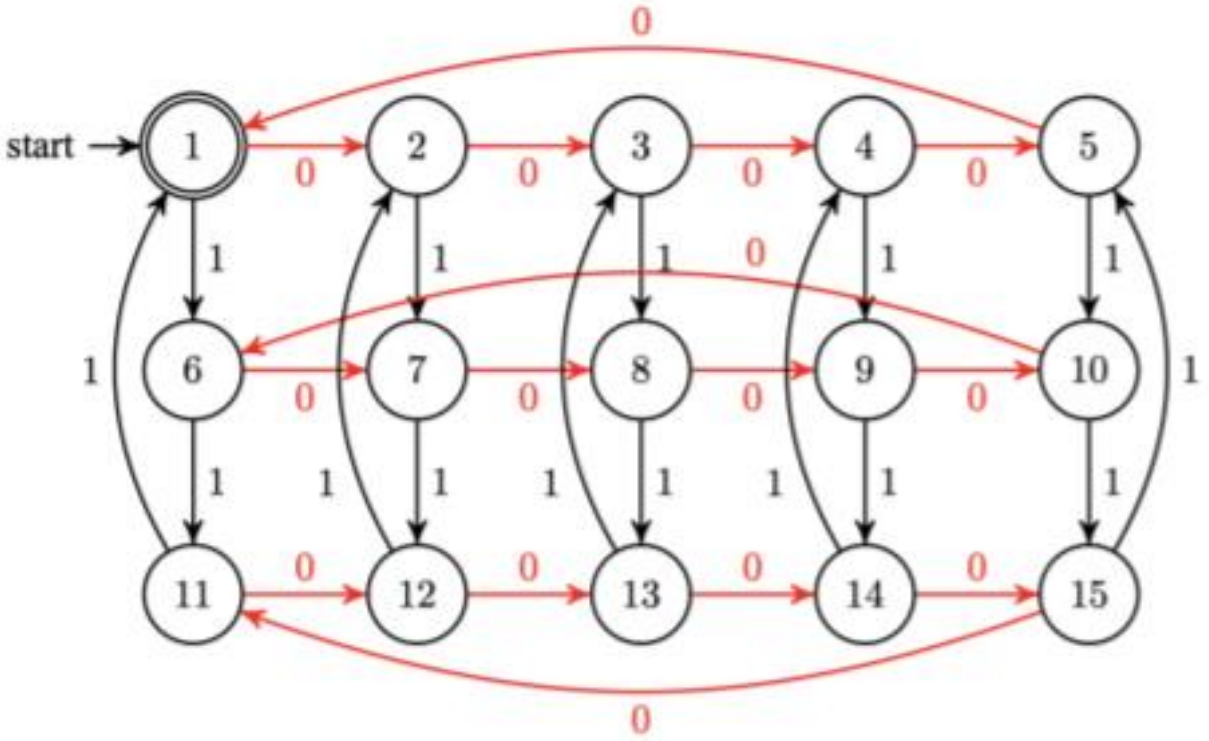


状态机代码模板

```
module xx(xxx);
    reg [2:0]state_r;
    always @(posedge clk) begin
        //这里用Case也可以。
        if (state_r == 0) begin
            if(xxx) begin
                        state_r <= ??
            end
        end
        else if(state_r == 1) begin
            if(xxx) begin
                    state_r <= ??
            end
        end
        ....
        else begin
            ...
        end
    end
endmodule
```

多标识的状态机

- 模块接受一个01串，判断当前已经输入的01串中，是否0的个数是5的倍数且1的个数是3的倍数。



我们使用俩个标识来标记这个状态机。