课后实验二：状态机

预备知识

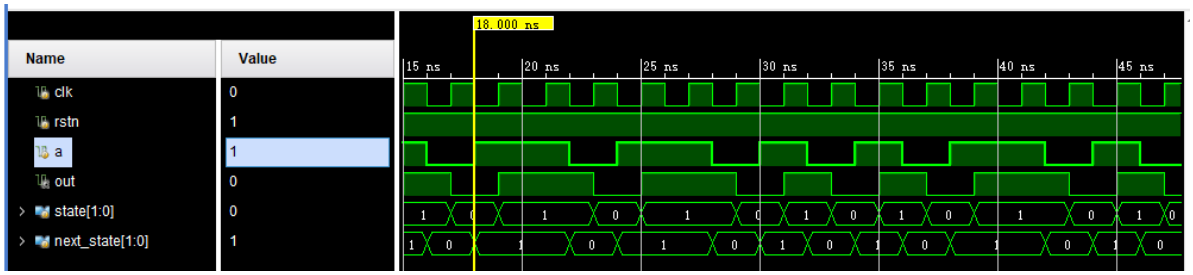
- 时序逻辑电路的verilog实现
  - clk信号的生成
  - 阻塞赋值与非阻塞赋值
  - always块的敏感信号的理解，上升沿和下降沿
- always块中相关语法
  - if else
  - case

实验重难点

- 状态机的verilog实现
  - 三段式状态机
    - 状态转移块
    - 状态获取块
    - 结果输出块

实验内容

- 阅读状态机的相关知识
- 阅读示例Moore状态机verilog代码实现
- 新建工程，将示例状态机模块和testbench加入后run simulation，观察波形
- 根据状态图编写三段式状态机



状态机知识点

状态机基础

有限状态机（Finite-State Machine，FSM），又成为有限状态自动机，简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。

有限状态机的组成

- 状态寄存器：记忆状态机当前所处的状态
- 产生下一状态的组合逻辑：根据输入信号或当前状态，确定下一状态
- 输出逻辑：由当前状态和输入信号，决定当前状态的输出

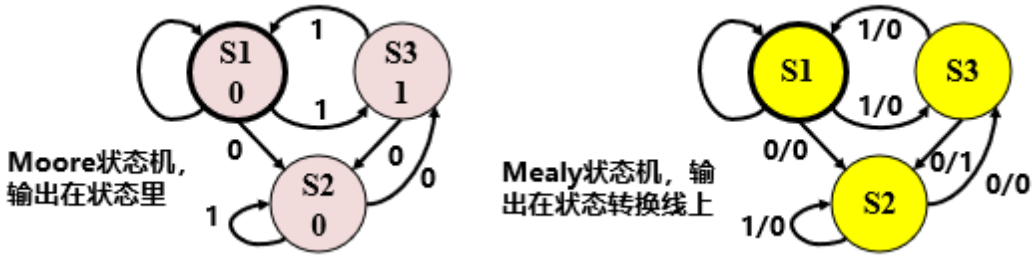
两类状态机

Moore状态机

状态机的输出仅仅依赖于当前状态，而与输入条件无关

Mealy状态机

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



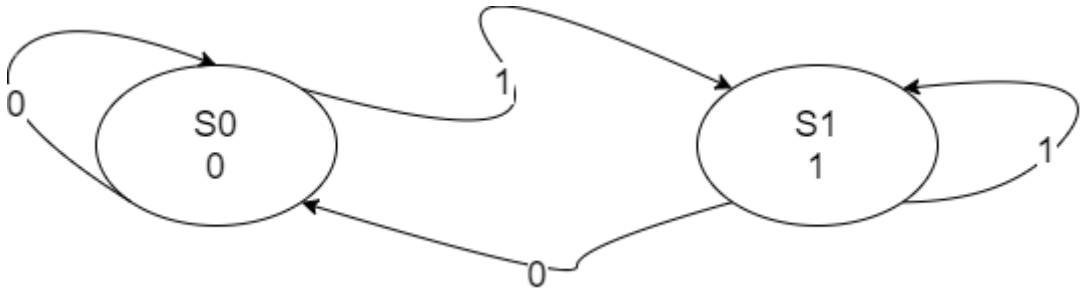
设计状态机（三段式）

- 一个时序逻辑always块用来描述当前状态转移（current_state），寄存器的复位和状态转移。
  - 时序逻辑采用非阻塞赋值（<=）
  - 时序逻辑的敏感信号需要是 信号的上升沿等（posedge、negedge）
- 一个组合逻辑always块用来描述下一个状态的求取（next_state）。
  - 组合逻辑采用阻塞赋值（=）
  - 组合逻辑的敏感信号需要包含always块中所有可能会发生变化的变量，简单写法为always@（*）
  - 组合逻辑电路需要处理所有可能的情况，否则会出现锁存器。解决的好办法：if else中的else处理或者case中的default处理
- 一个组合逻辑always块或者时序逻辑always块用来描述输出（output）。

详细实现见示例

示例状态机

Moore状态机



状态机代码

```
`timescale 1ns / 1ps

module fsm_2(
    input clk,
    input rstn,
    input a,
    output reg out
    );
```

```
    localparam S0 = 2'b0;//localparam关键字用来定义本模块内常量，通常定义状态值
    localparam S1 = 2'b1;

    //声明两个变量，state表示当前状态，next_state表示下一时刻将要更新的状态
    reg [1:0] state;
    reg [1:0] next_state;

    //时序逻辑块，表示当前状态的初始化或者更新状态
    //初始化发生在 置位信号的下降沿， 更新状态发生在 时钟信号的上升沿
    always@(negedge rstn or posedge clk)begin
        if (!rstn) begin//配合置位信号变为0的时候
            state <= S0;//初始化状态为S0
        end
        else begin
            state <= next_state;//当时钟上升沿到来时，state更新为上升沿到来前一刻
next_state的值
        end
    end

    //组合逻辑求取next_state的值，敏感信号为当前状态state和输入a
    always@(state or a)begin
        case(state)//case语句的判断条件
        S0:begin
            if (a == 1'b0)begin
                next_state = S0;//阻塞赋值
            end
            else begin
                next_state = S1;
            end
        end
        S1:begin
            next_state = a ? S1 : S0;//?:语法，类似C语言
        end
        default: next_state = S0;//加入default防止锁存器产生
        endcase
    end

    //组合逻辑生成输出值，这里注意需要是当前状态对应输出值，和课堂讲的状态机输出一致
    always @(state) begin
        case(state)
        S0:out = 1'b0;
        S1:out = 1'b1;
        default: out = 1'bx;//同样需要加入default来防止锁存器产生
        endcase
    end
endmodule
```

testbench

```
`timescale 1ns / 1ps

module test_fsm_2(

    );
    reg clk;
    reg rstn;
```

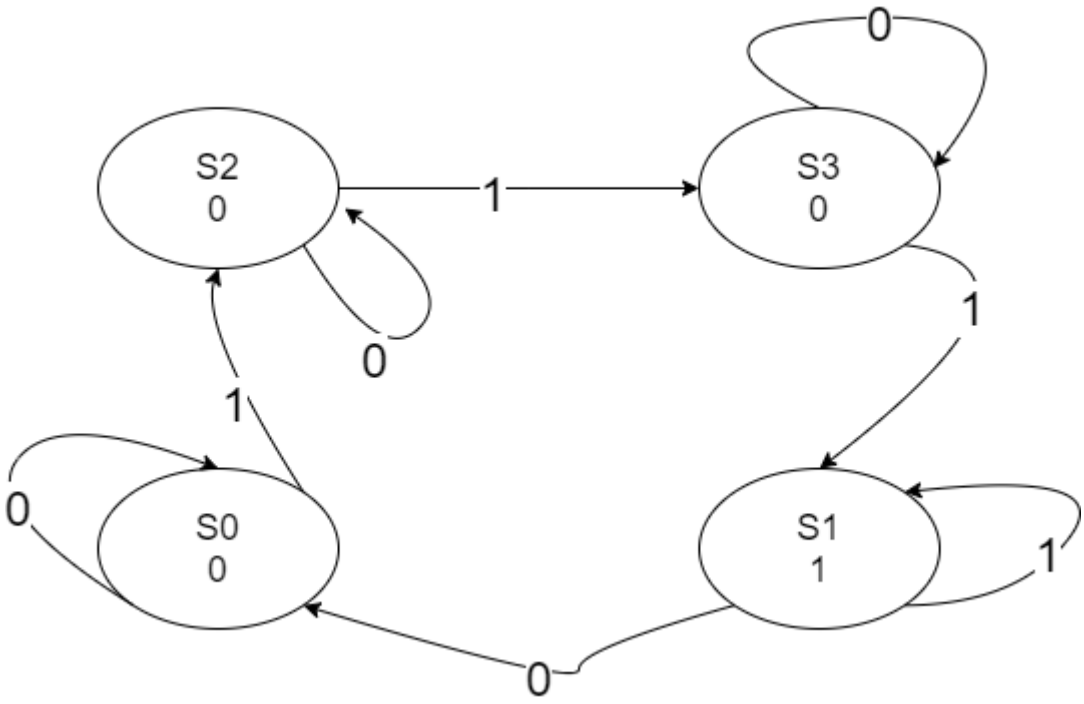
```
    reg a;
    wire out;

    //模块调用
    fsm_2 fsm_inst_0 (
        .clk(clk),
        .rstn(rstn),
        .a(a),
        .out(out)
    );

    //初始化 clk信号和置位信号
    initial begin
        clk = 0;
        rstn = 1;
        #0.1 rstn = 0;
        #1.1 rstn = 1;
    end

    //初始化 输入信号
    initial begin
        a = 0;
        #4 a = 1;
        #4 a = 0;
    end

    // always模块如果不加入#number的延迟会导致仿真错误，因为需要隔一定时间变换才能生成方波
    always begin
        #1 clk = ~clk;
    end

    // 周期性产生随机输入信号
    always begin
        #2 a = $random() % 2;
    end
endmodule
```

状态机练习题

按照三段式状态机的写法实现下面的状态机

状态机的端口

input	output
时钟信号 clk	输出信号 out
置位信号 rstn	
输入信号 a	



- 自行编写testbench，不需要验收。
- 一个正确的波形参考