

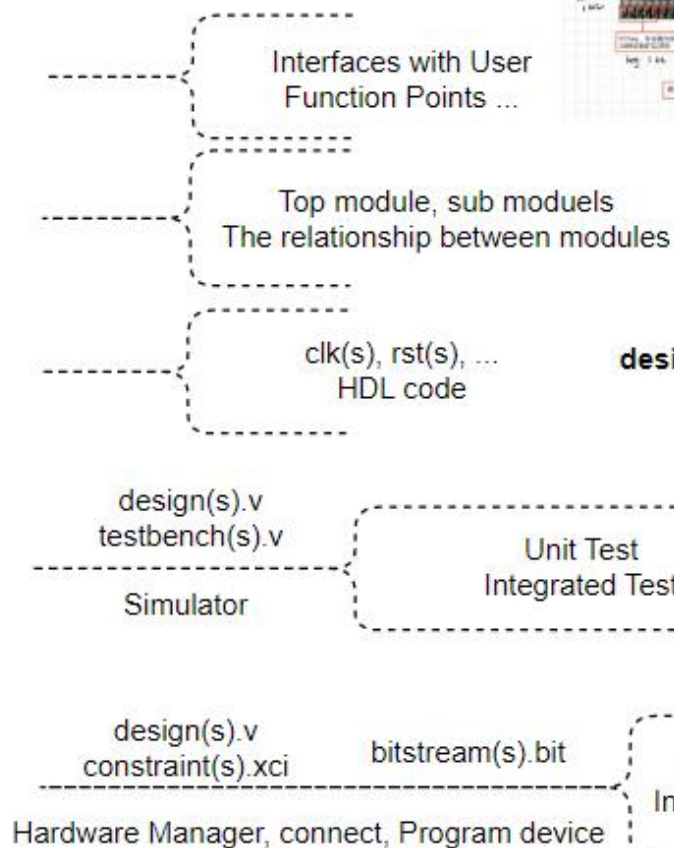
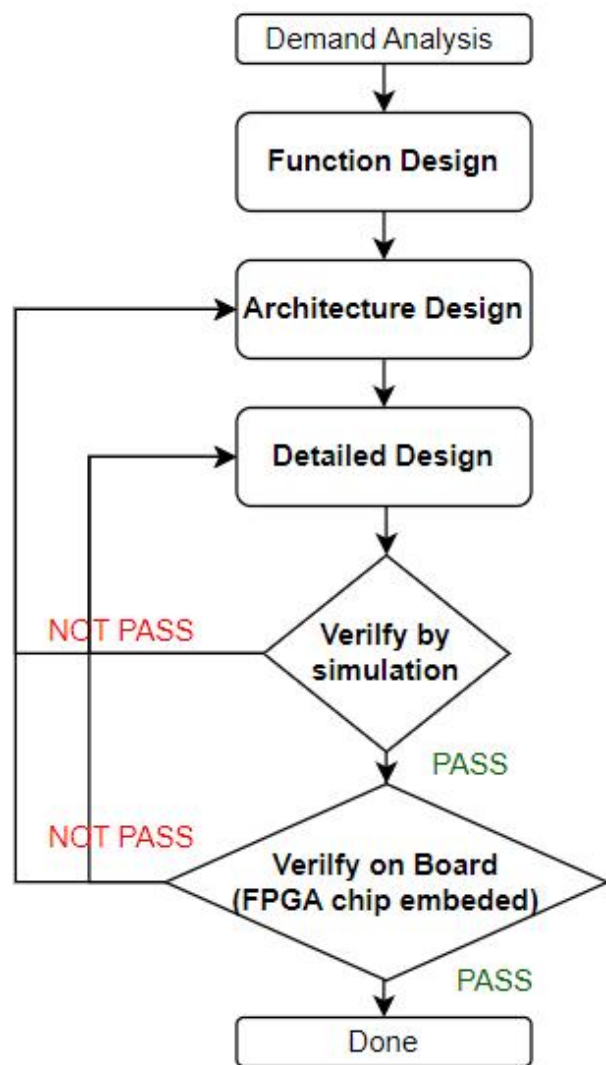
A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark blue background, resembling a circuit board or a neural network.

DIGITAL DESIGN

LAB11 FSM & PROJECT RELATED

2022 FALL TERM @ CSE . SUSTECH

HOW TO DEVELOP A CIRCUIT PROJECT



电路的结构设计:

不要把所有的代码放在一个module里面
从用户看: 读入输入 (拨码/按键防抖, 外接键盘?), 输出 (LED灯, 七段数码显示管), 核心 (输入转输出, 洗衣机按阶段分模块, 小车uart。按功能或流程去划分)。顶层模块需要。各个子模块相互连接 (确定输入输出端口), 之后连接顶层, 要注意各个子模块之间的联系 (高/低电平有效? 在写的时候要规定好) 架构之后就分工下去。详细设计: 用verilog代码去实现。

(功能) 验证: 两种方式 (仿真看波形, 上板需要设计和约束文件, 谁拿开发板要协调好), 做好单元测试再做集成测试, 子模块之间



两次测试结果不一样

一定不用忽略multiple driver: 同一个信号有多个源头

时序逻辑的verilog语法规则:

1.三种建模方式的选择

- 1) 不使用 数据流建模 (assign)
- 2) 可以使用结构化建模, 使用基本器件(通过例化来生成对应的电路组件) 来完成电路搭建
- 3) 更多时候使用行为级建模

2.行为级建模来表示 组合逻辑 和 时序逻辑的区别

1) 敏感列表的变化

注意敏感列表中不能混用 信号和时钟跳变沿

`always@*`

`always@ (posedge clk)`

2) 赋值语句的变化

`always@*`

`a=b; // the type of a MUST be REG, using blocking assignment =`

`always@ (posedge clk)`

`a<=b;`

`// the type of a MUST be REG, using NON blocking assignment <=`

FSM : FINITE STATE MACHINES

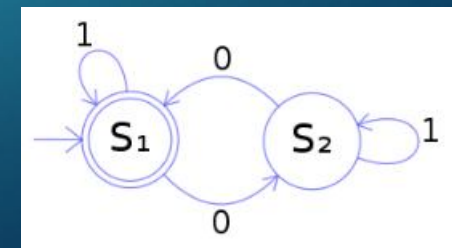
An FSM is defined by a list of its states, its initial state, and the conditions for each transition. Finite state machines are of two types – deterministic finite state machines and non-deterministic finite state machines.^[1] A deterministic finite-state machine can be constructed equivalent to any non-deterministic one.

When describing a FSM, the key is to clearly describe several elements of the state machine :

- how to make state transition
- the conditions of state transition
- what is the output of each state.

饮水机的童锁：状态的迁移

Generally speaking, the state transition part is a synchronous sequential circuit after the state machine is implemented, and the judgment of the state transition condition is combinational logic.



Double circle means
initial state

https://en.wikipedia.org/wiki/Finite-state_machine

3 WAYS ON FSM

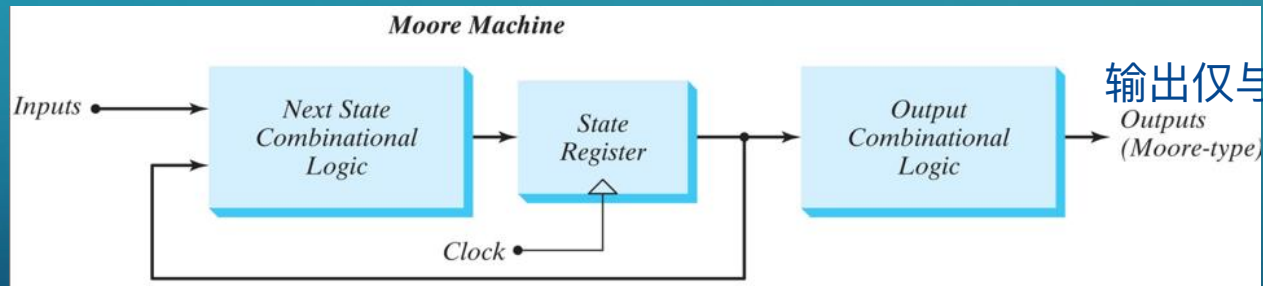
- (1) **One-stage**: stage: 一个always语句块 The whole FSM is written into **one always block**, which describes not only the state transition, but also the input and output of the state. 注意敏感列表中不能混用信号和时钟跳变沿 (**NOT suggested**)
- (2) **Two-stages**: **two always blocks** are used to describe the state machine, one of which uses **sequential logic** to describe the **state transition**; the other uses **combinational logic** to judge the condition of state transition, to describe **the rules of state transition and output**;
- (3) **Three-stages**: One always module uses **sequential logic** to describe **state transition**, One always uses **combination logic** to judge state transition conditions and describe state transition rules, and the Other always block describes state **output** (which can either be output of combination circuit or the output of sequential circuit).

Generally speaking, the recommended FSM description method is the latter two. This is because: FSM, like other designs, is best designed in a synchronous sequential manner to improve the stability of the design and eliminate burrs.

MOORE MODE

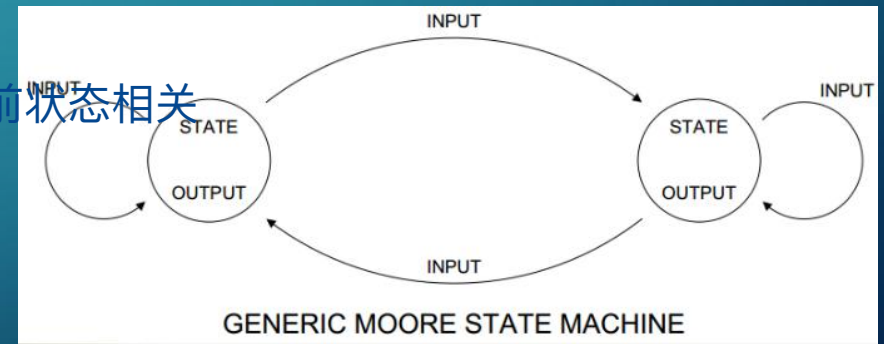
- Outputs are functions of 'present state' ONLY
- Outputs are synchronized with clock
- Output is the state of the circuit, Relatively simple

摩尔模式

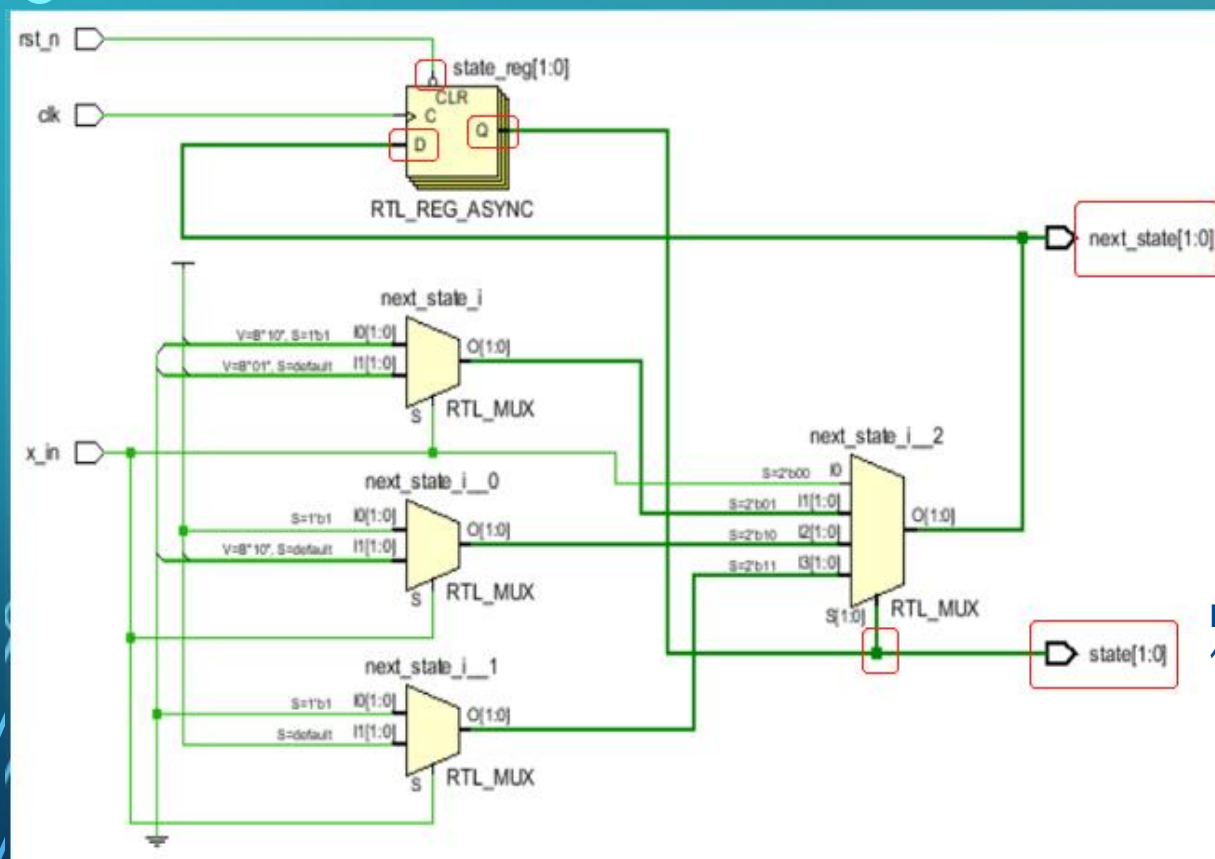
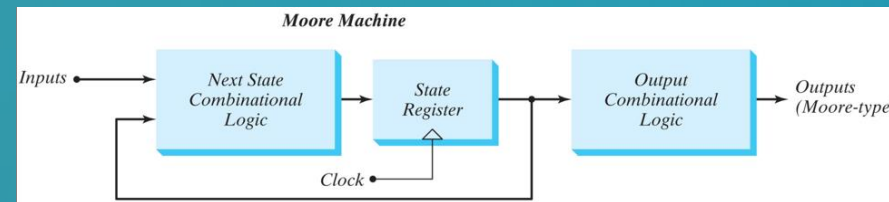


输出仅与当前状态相关

只有时钟跳变时当前状态才会变化



MOORE MODE WITH 2-STAGES



状态即输出

Next主要用于观察

```
timescale 1ns / 1ps
////////////////////////////////////// 复位信号 ////////////////////////////////////////
module moore_2b(input clk, rst_n, x_in, output [1:0] state, next_state);
    reg [1:0] state, next_state;
    parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
    always @(posedge clk, negedge rst_n) begin
        if (~rst_n)
            state <= S0;
        else
            state <= next_state;
    end
    always @(state, x_in) begin
        case(state)
            S0: if(x_in) next_state = S1; else next_state = S0;
            S1: if(x_in) next_state = S2; else next_state = S1;
            S2: if(x_in) next_state = S3; else next_state = S2;
            S3: if(x_in) next_state = S0; else next_state = S3;
        endcase
    end
end
endmodule
```

两个条件有一个满足就执行下面的语句

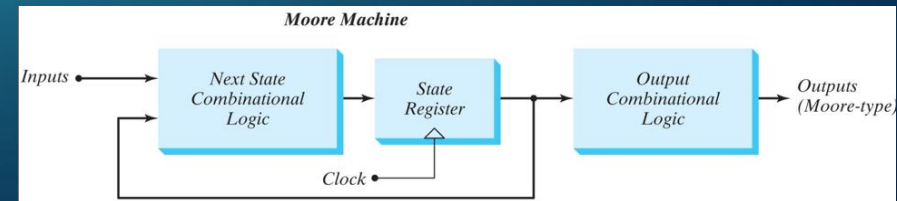
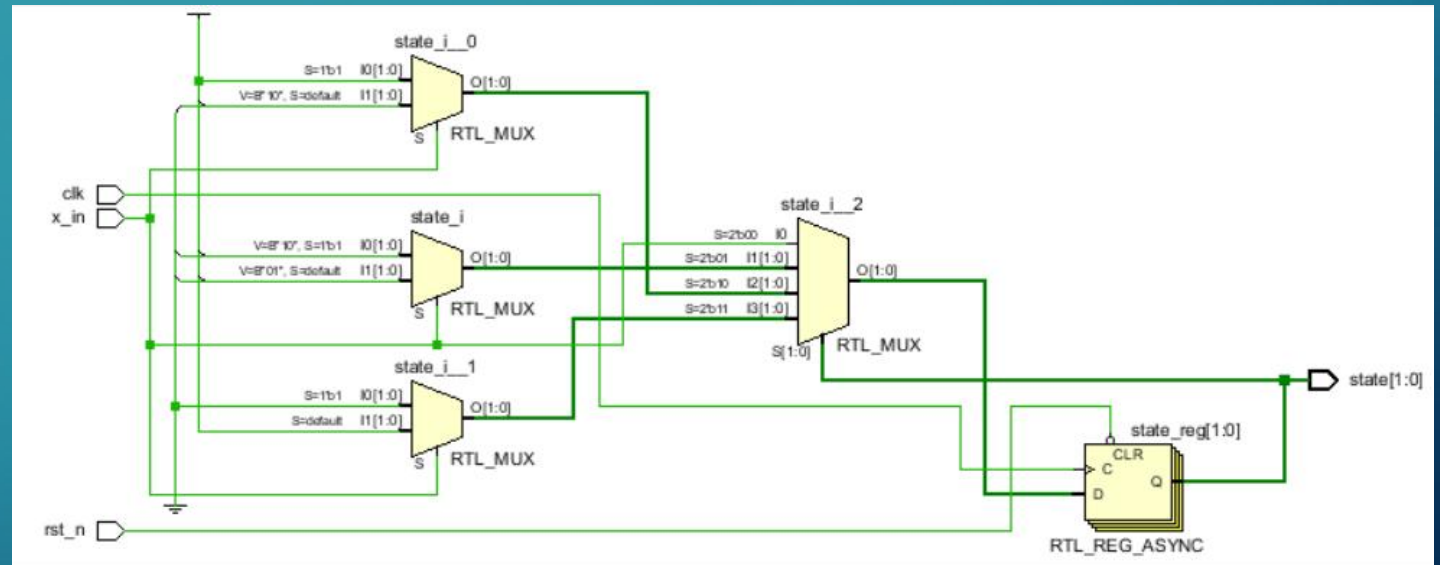
state何时更新

next state何时更新

```
always@ (POSedge clk, negedge rst _n) //异步复位
if(!rst_n)
X<= 50;
else
    x<=
always@ (POSedge clk)
if(!rst_n) // 同步复位
x<= S0,
else
X<=
```


MOORE MODE WITH 1-STAGE (NOT SUGGESTED)

```
`timescale 1ns / 1ps
///////////////////////////////////////////////////
module moore_1b(input clk,rst_n,x_in,output[1:0] state):
  reg [1:0] state;
  parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
  always @(posedge clk,negedge rst_n) begin
    if(~rst_n)
      state <= S0;
    else
      case(state)
        S0: if(x_in) state <= S1; else state <= S0;
        S1: if(x_in) state <= S2; else state <= S1;
        S2: if(x_in) state <= S3; else state <= S2;
        S3: if(x_in) state <= S0; else state <= S3;
      endcase
  end
endmodule
```

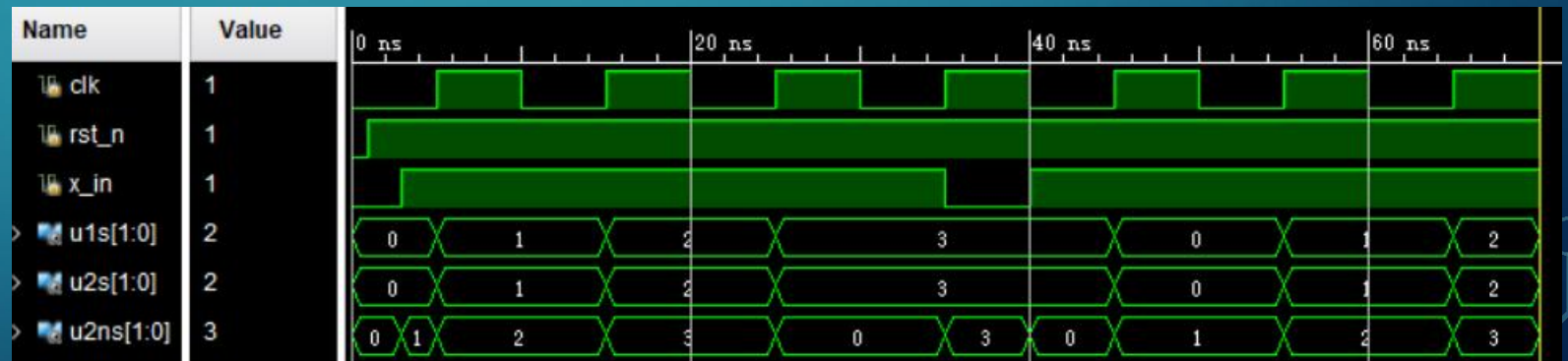


SIMULATION ON 1-STAGE & 2-STAGES OF MOORE

```
timescale 1ns / 1ps
////////////////////////////////////
module sim_moore_12();
reg clk,rst_n,x_in;
wire [1:0] u1s,u2s,u2ns;
moore_1b u1(clk,rst_n,x_in,u1s);
moore_2b u2(clk,rst_n,x_in,u2s,u2ns);
initial #70 $finish;
initial begin
clk = 1'b0;
rst_n=1'b0;
forever #5 clk=~clk;
end

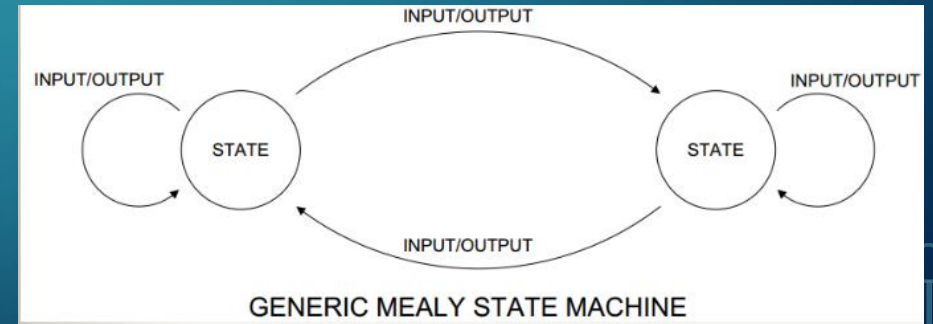
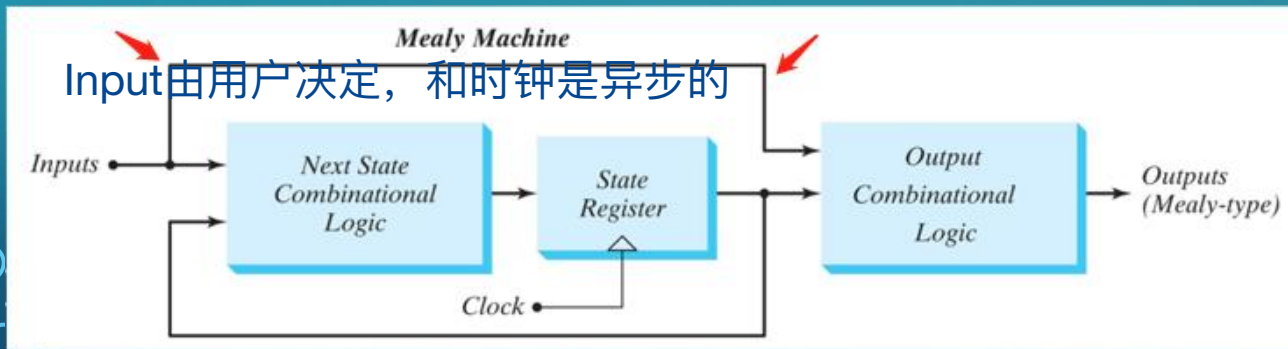
initial fork
x_in=1'b0;
#1 rst_n = 1'b1;
#3 x_in = 1'b1;
#35 x_in = 1'b0;
#40 x_in = 1'b1;
join
endmodule
```

Here the behaviors of circuits implemented by one stage or two stage on moore FSM are same, but two stage is clearer than one stage.



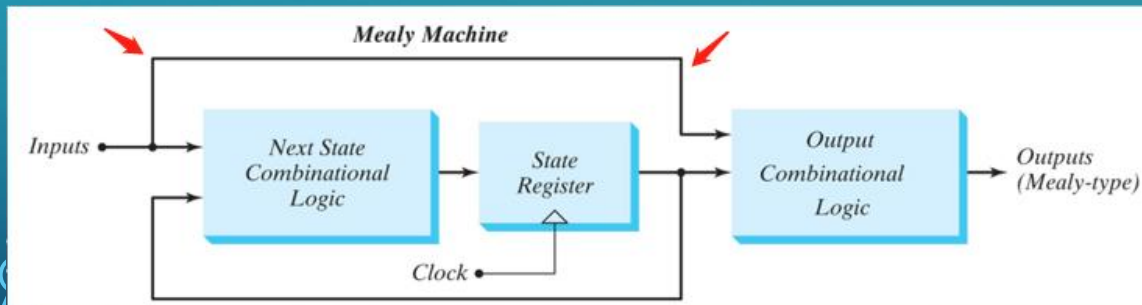
MEALY MODE

- Outputs are functions of both 'present state' and 'inputs'
- Outputs may change if inputs change
- Output is not the state of the circuit, Relatively complex



MEALY MODE WITH TWO-STAGES

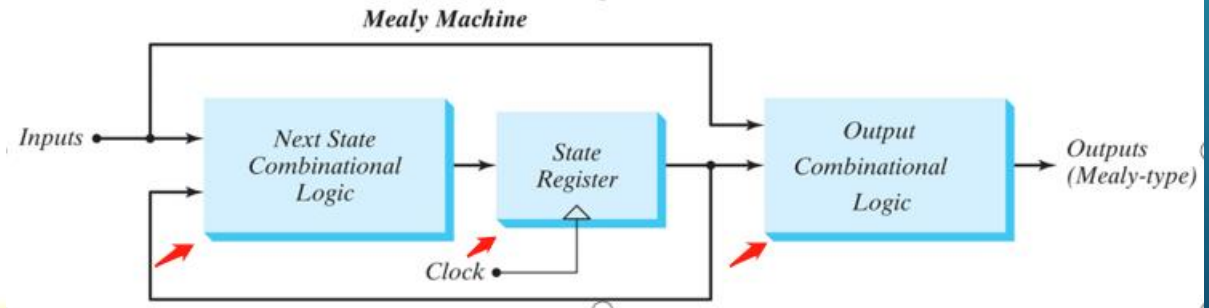
The 'next state' and 'output' are both determined in the combinational logic, the 'state' is updated in the sequential logic.



```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module mealy_2b(input clk,rst_n,x_in,output[1:0] state,next_state,output y);
  reg [1:0] state,next_state;
  reg y;
  parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
  always @(posedge clk,negedge rst_n) begin
    if(~rst_n)
    begin
      state <= S0;
      y <= 1'b0;
    end
    else
      state <= next_state;
  end
  always @(state,x_in) begin
    case(state)
    S0: if(x_in) {next_state,y} = {S1,1'b0}; else {next_state,y} = {S0,1'b0};
    S1: if(x_in) {next_state,y} = {S2,1'b0}; else {next_state,y} = {S1,1'b0};
    S2: if(x_in) {next_state,y} = {S3,1'b0}; else {next_state,y} = {S2,1'b0};
    S3: if(x_in) {next_state,y} = {S0,1'b1}; else {next_state,y} = {S3,1'b0};
    endcase
  end
end
endmodule
```


MEALY MODE WITH THREE-STAGES

```
module mealy_3b(input clk,rst_n,x_in,output[1:0] state,next_state,output y)
reg [1:0] state,next_state;
reg y;
parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
always @(posedge clk,negedge rst_n) begin...
always @(state,x_in) begin...
always @(state,x_in) begin...
endmodule
```



```
always @(state,x_in) begin
    case(state)
    S0: if(x_in) next_state = S1; else next_state = S0;
    S1: if(x_in) next_state = S2; else next_state = S1;
    S2: if(x_in) next_state = S3; else next_state = S2;
    S3: if(x_in) next_state = S0; else next_state = S3;
    endcase
end
```

```
always @(posedge clk,negedge rst_n) begin
    if(~rst_n)
    begin
        state <= S0;
        y <= 1'b0;
    end
    else
        state <= next_state;
    end
```

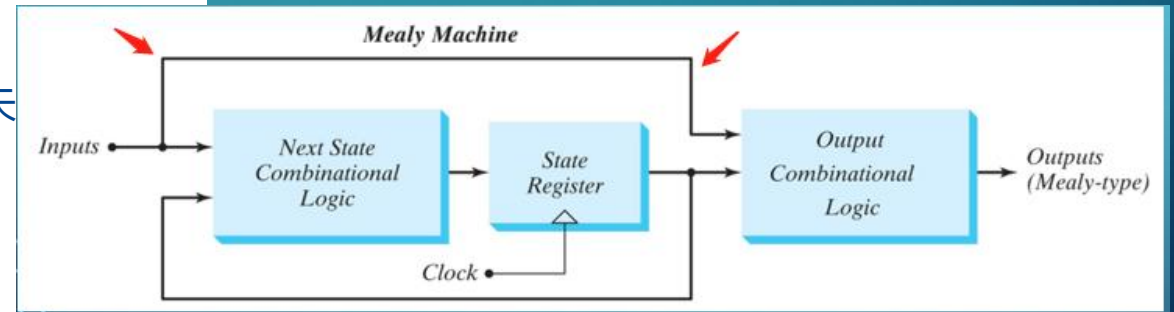
```
always @(state,x_in) begin
    case(state)
    S0,S1,S2: y=1'b0;
    S3: if(x_in) y=1'b1; else y=1'b0;
    endcase
end
```

可以嵌套

MEALY MODE WITH 1-STAGE(NOT SUGGESTED)

```
timescale 1ns / 1ps
////////////////////////////////////
module mealy_1b(input clk,rst_n,x_in,output[1:0] state,output y);
reg [1:0] state;
reg y;
parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
always @(posedge clk,negedge rst_n) begin
    if(~rst_n)
    begin
        state <= S0;
        y <= 1'b0;
    end
    else
    case(state)
    S0: if(x_in) {state,y} <= {S1,1'b0}; else {state,y} <= {S0,1'b0};
    S1: if(x_in) {state,y} <= {S2,1'b0}; else {state,y} <= {S1,1'b0};
    S2: if(x_in) {state,y} <= {S3,1'b0}; else {state,y} <= {S2,1'b0};
    S3: if(x_in) {state,y} <= {S0,1'b1}; else {state,y} <= {S3,1'b0};
    endcase
end
endmodule
```

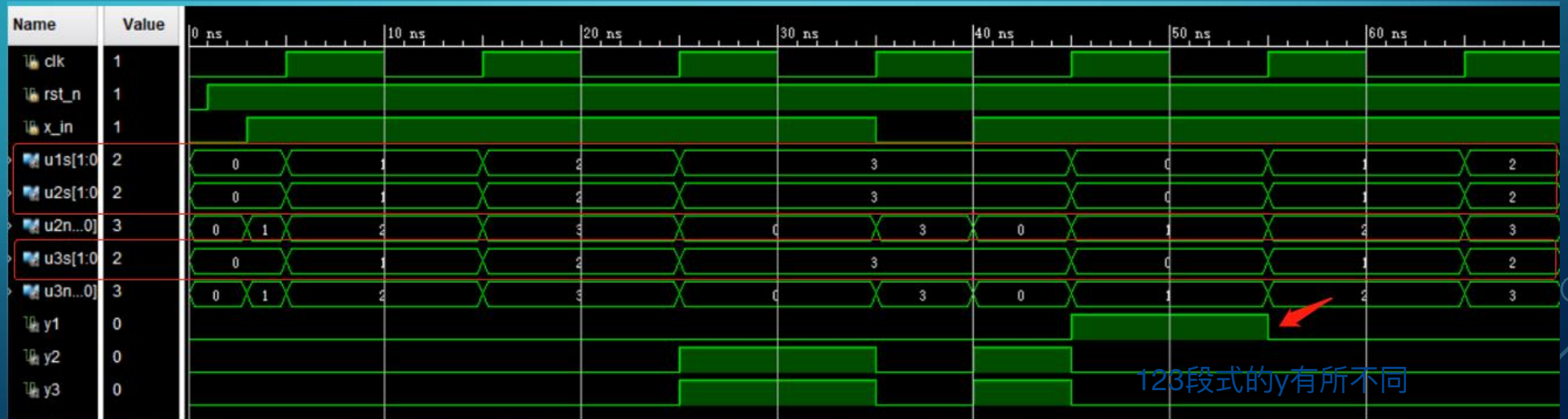
红色箭头部分会丢失



Is it ok to implement a mealy mode circuit by using 1-stage ?
Why?

SIMULATION ON ONE,TWO &THREE STAGE OF MEALY

Are the behaviors of circuits implemented by one stage, two stage and three stage on mealy FSM are same?
Which one(s) is(are) correct? Which one(s) is(are) wrong?



123段式的y有所不同

PRACTICE1 (TEAM WORK)

- Top Module and Teamwork:
 - Interfaces between the circuit and the user(Inputs and Outputs):
 - How many input and output port would be used, which input and output devices would be used in your project?
 - Top module and sub modules :
 - What are the sub modules in your project?
 - What's the relationship between sub modules, what's the relationship between sub modules and the top module?
- FSM:
 - How many states in your project, how does the state transmit from one to another? Draw the state transition diagram
 - What's the type of FSM for your project? Implement the FSM to describe the state transition and the output in verilog.

PRACTICE2

A circuit has 2 inputs (x_{in} (5bit-width), clk) and 1 output(y_{out}). The circuit get the state of x_{in} at every posedge of clk , If the total number of received 1 is a multiple of 5, then y_{out} is valid, otherwise y_{out} is invalid.

1. Do the design by using behavior modeling in verilog. Is this a moore mode or mealy mode? Try to implement the circuit by two-stages and three-stages respectively.
2. Build testbench and verify the function on this sequential circuit.
3. Try to implement the circuit on Minisys board

PRACTICE3

A sequential circuit consists of three D flip-flops A , B and C, an input x_{in} .

while $x_{in} : 0$, the state of the circuit remains unchanged;

while $x_{in} : 1$, the state of the circuit passes through 001, 010, 100, and then back to 001, so the cycle.

1. Do the design by using behavior modeling in verilog. Is this a moore mode or mealy mode? Try to implement the circuit by two-stages.
2. Build testbench and verify the function on this sequential circuit.
3. Try to implement the circuit on Minisys board