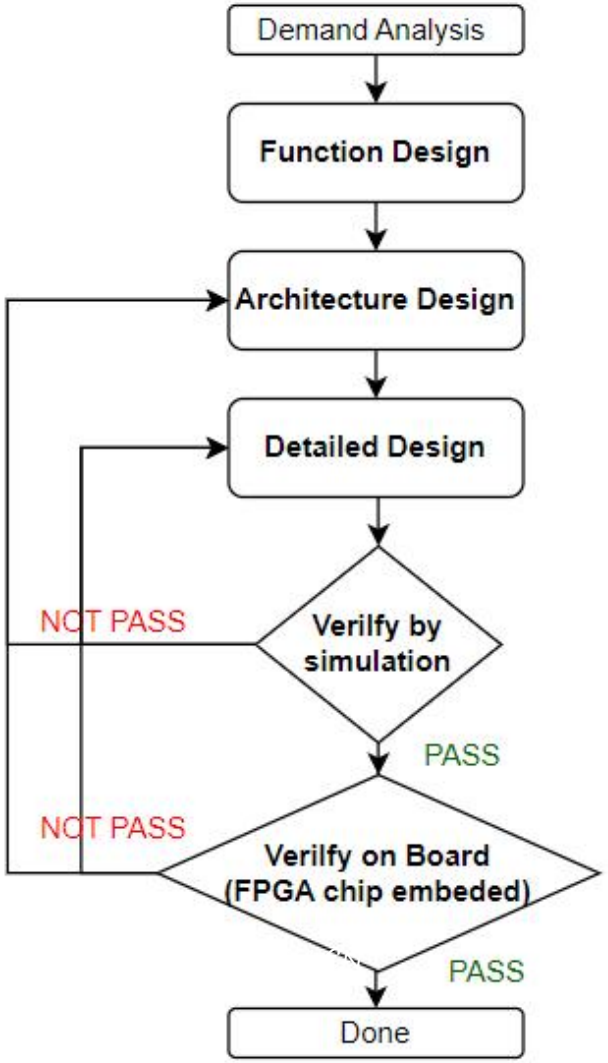
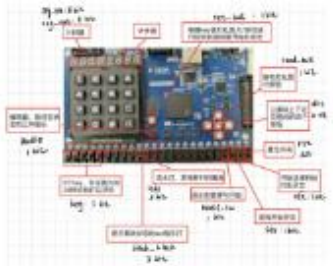
DIGITAL DESIGN

LAB11 FSM & PROJECT RELATED

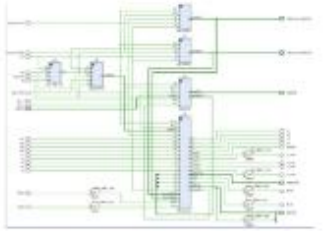
WANGW6@SUSTECH.EDU.CN

HOW TO DEVELOP A CIRCUIT PROJECT





Interfaces with User
Function Points ...



Top module, sub moduels
The relationship between modules

clk(s), rst(s), ...
HDL code

design(s).v ...

design(s).v
testbench(s).v

Simulator

Unit Test
Integrated Test

design(s).v
constraint(s).xci

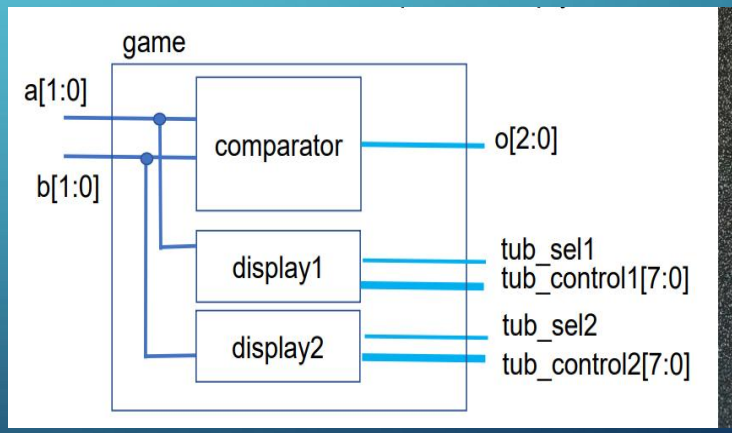
bitstream(s).bit

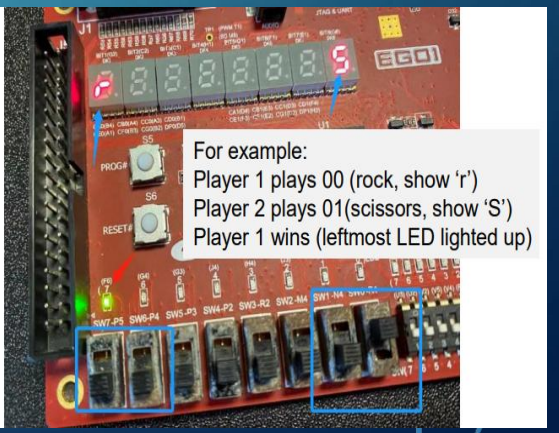
Hardware Manager, connect, Program device

Unit Test
Integrated Test

CODE SPECIFICATIONS IN PROJECT

- **Same Naming rules**: Maintain consistency in naming module, ports, datas
  - option1: using CamelCase ONLY in project; option2: using snake_case ONLY in project
- **Avoiding Devil's Numbers**: define parameter and using it instead of using numbers in code
  - using parameter (Using the example on page 7,8,11,12 as a reference)
- **Structal design**
  - Parallel work
  - Self testing and integration
  - Rock-Paper-Scissors game (v2.0) in lab6





CODE SPECIFICATIONS IN SENSITIVE LIST OF ALWAYS

examples	YES/ NO	NOTES
always @ * always@(a,b) always@(a or b)	YES	combinational logic
always @(posedge clk) always @(negedge clk)	YES	Sequential logic
always @(posedge clk, posedge rst) always @(negedge clk, posedge rst)	YES	Sequential logic rst high level valid, using “if(rst)” in this always block
always @(posedge clk, negedge rst) always @(negedge clk, negedge rst)	YES	Sequential logic rst low level valid, using “if(!rst)” in this always block
always @(posedge clk, negedge clk)	NO	using a higher frequency clock instead
always @(posedge clk, a,b) always @(negedge clk, a,b)	NO	using a sequential logic always and a combinational logic always instead

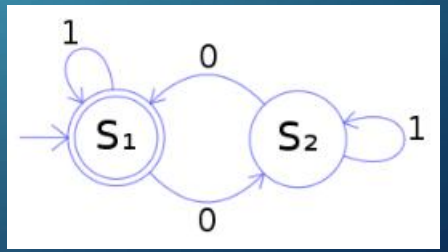
WANGWG@SUSTECH.EDU.CN

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.



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.

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

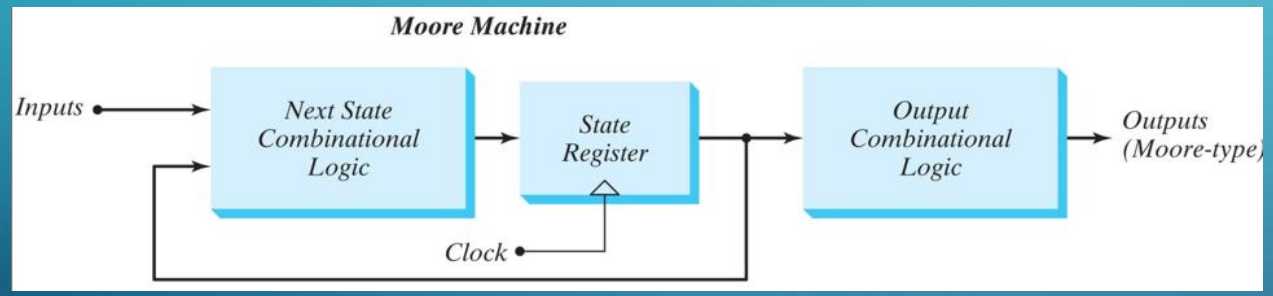
3 WAYS ON FSM

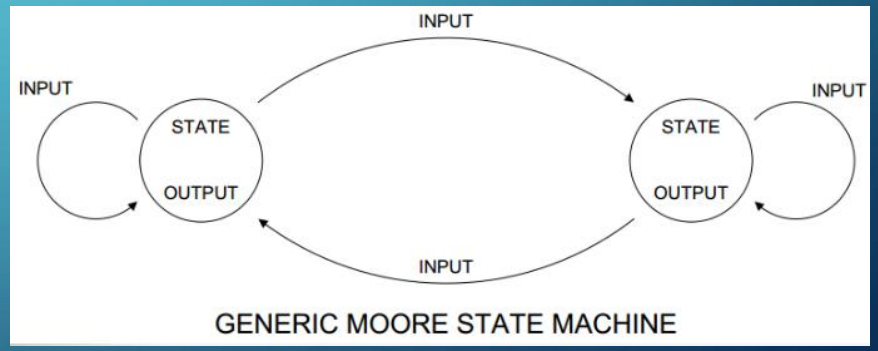
- (1) **One-stage**: The whole FSM is written into one always block, which describes not only the state transition, but also the input and output of the state. （NOT suggested）
- (2) **Two-stages**: two always blocks are used to describe the state machine, one of which uses sequential logic logic to describe 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.

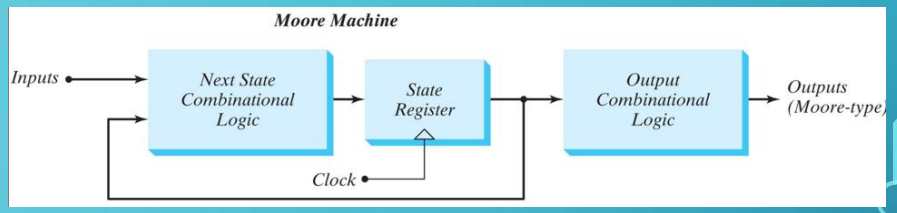
MOORE MODE

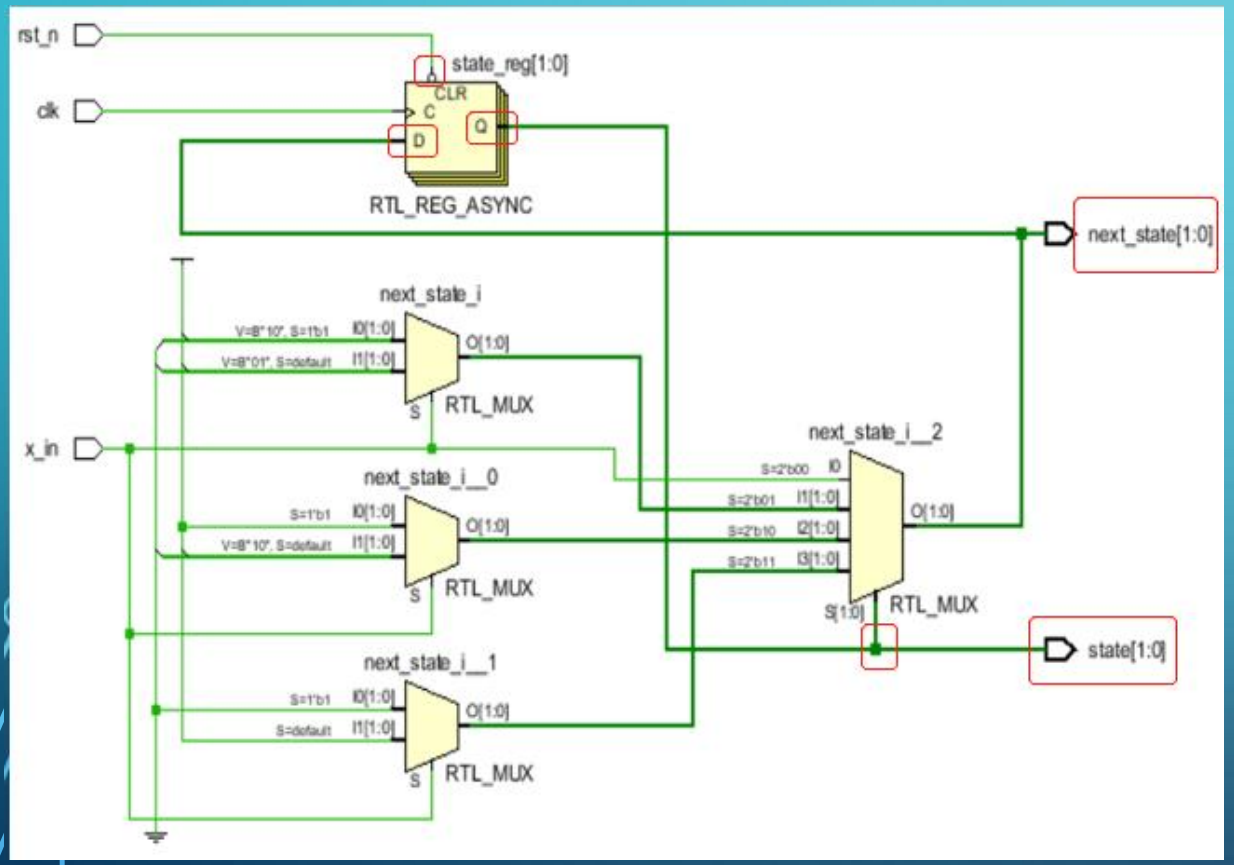
- Outputs are functions of 'present state' ONLY
- Outputs are synchronized with clock
- Output is the state of the circuit, Relatively simple





MOORE MODE WITH 2-STAGES

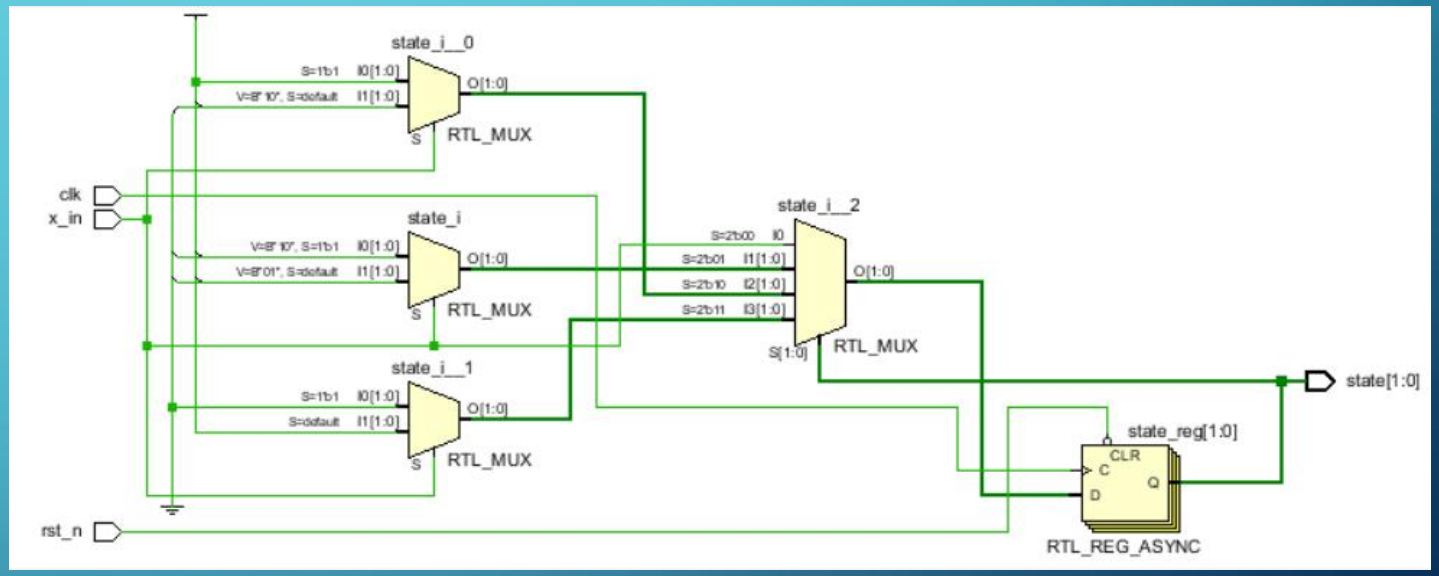


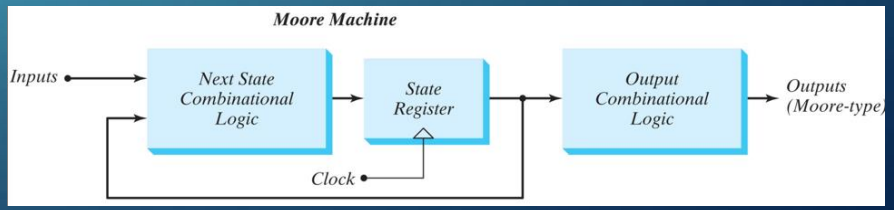



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module moore_2b(input clk,rst_n,x_in,output[1:0] state,next_state);
reg [1:0] state,next_state;
parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
always @(posedge clk,negedge rst_n) begin
    if(~rst_n)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

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
```



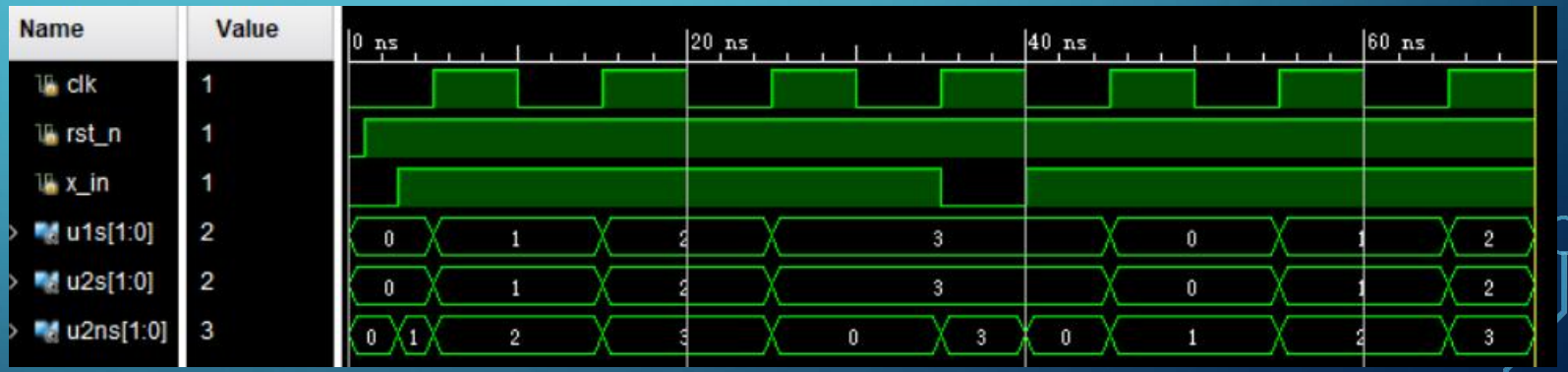


SIMULATION ON 1-STAGE & 2-STAGES OF MOORE

```
`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
```

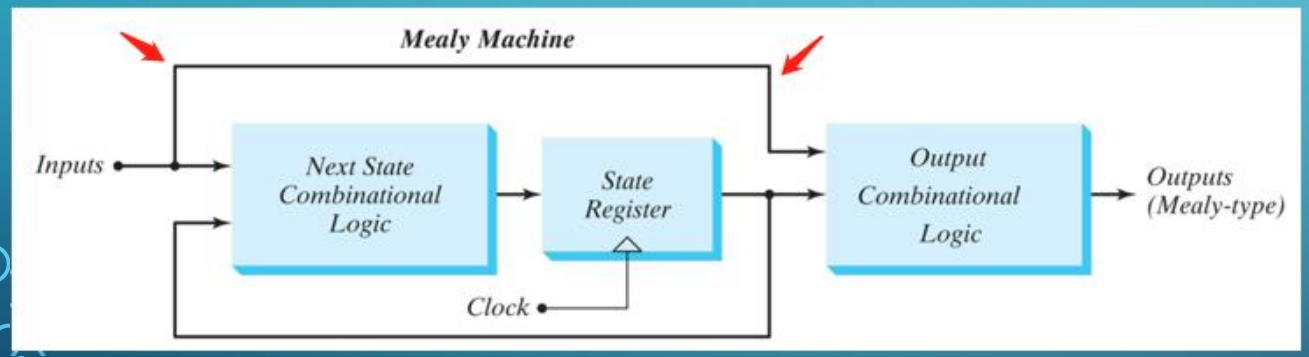
Here the behaviors of circuits implemented by one stage or two stage on moore FSM are same, but two stage is clearer than one stage.

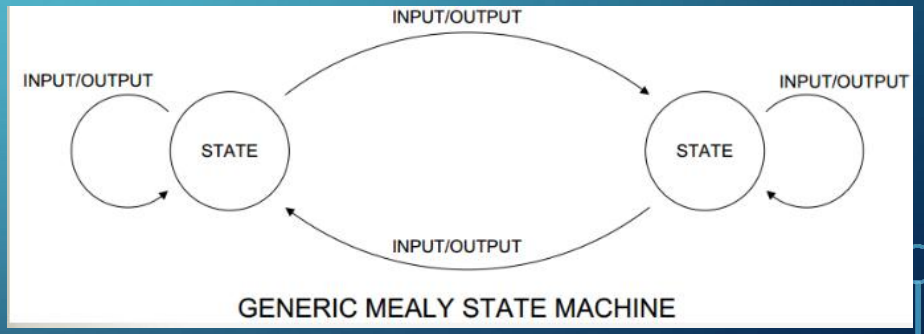


WANGW6@SUSTECH.EDU.CN

MEALY MODE

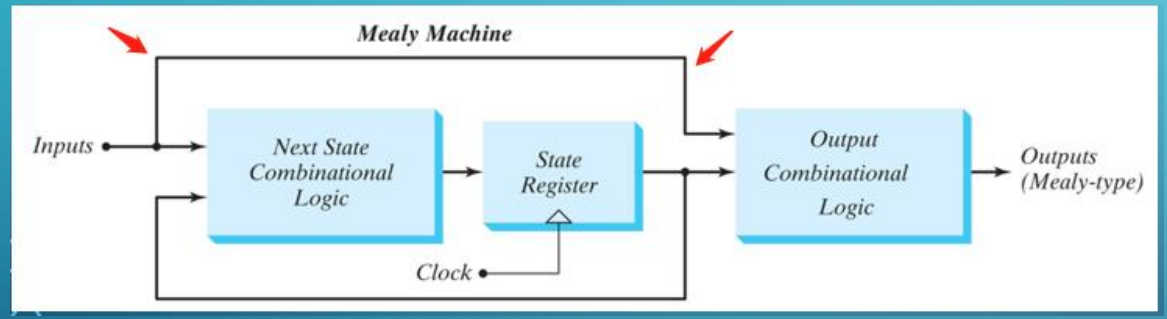
- Outputs are functions of both 'present state' and 'inputs'
- Outputs may change if inputs change
- Output is not the state of the circuit, Relatively complex





MEALY MODE WITH TWO-STAGES

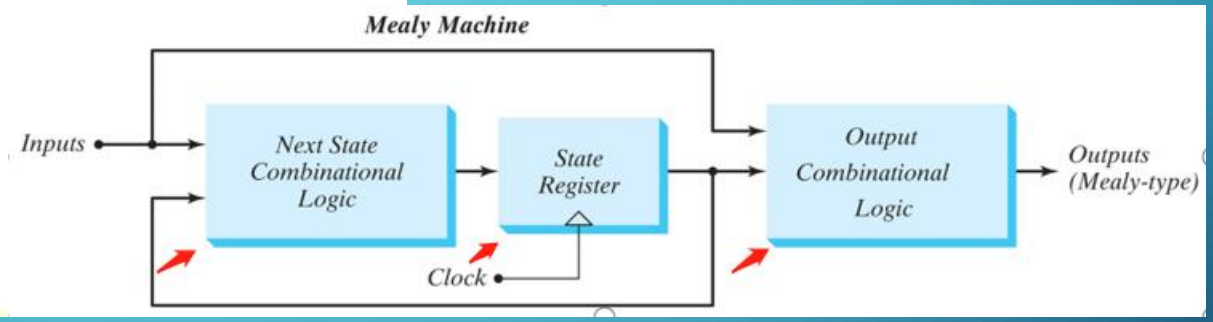
The 'next state' and 'output' are both determined in the combinational logic.
The 'state' is updated in the sequential logic.




```
module mearly_2b(
input clk,rst_n,x_in,
output reg [1:0] state, next_state,
output reg y );
parameter S0=2'B00, S1=2'B01, S2=2'B10, S3=3'B11;
always @(posedge clk, negedge rst_n) begin
    if(~rst_n)
        state <= S0;
    else
        state <= next_state;
end
always @ (state, x_in)
    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} = {S2,1'B0};
    endcase
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
```

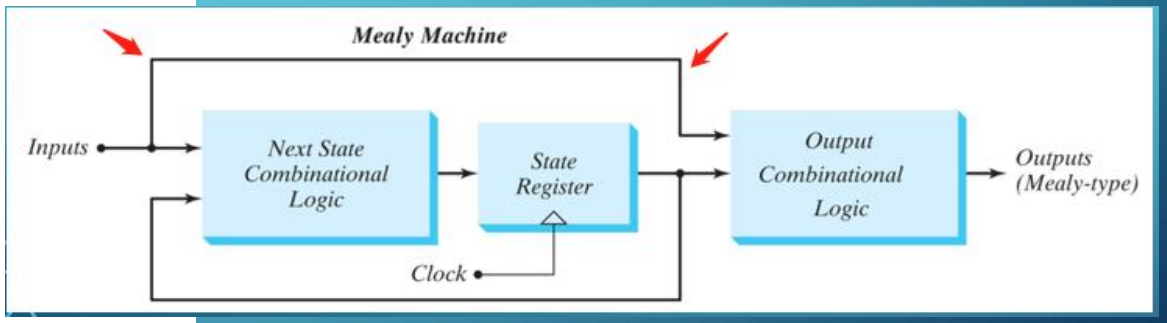
```
always @(posedge clk, negedge rst_n) begin
    if(~rst_n)
        state <= S0;
    else
        state <= next_state;
end
```

```
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
```

MEALY MODE WITH 1-STAGE(NOT SUGGESTED)

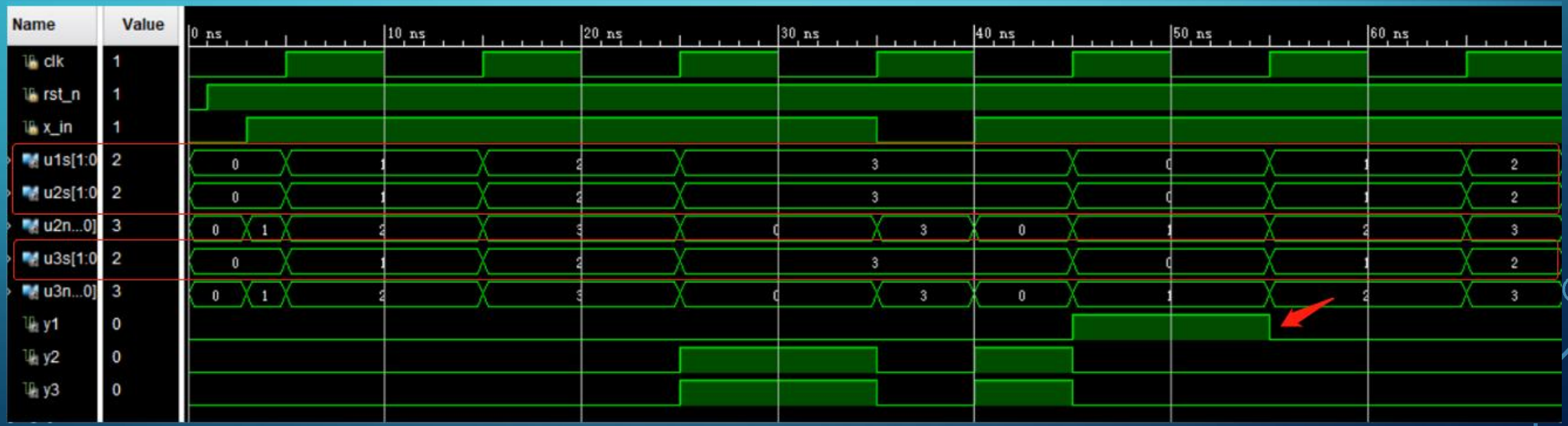
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module mealy_1b(input clk,rst_n,x_in,output[1:0] state,output y);
reg [1:0] state;
reg y;
parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
always @(posedge clk,negedge rst_n) begin
    if(~rst_n)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

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?



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.

WANGW6@SUSTECH.EDU.CN

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 EGO1 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 EGO1 board