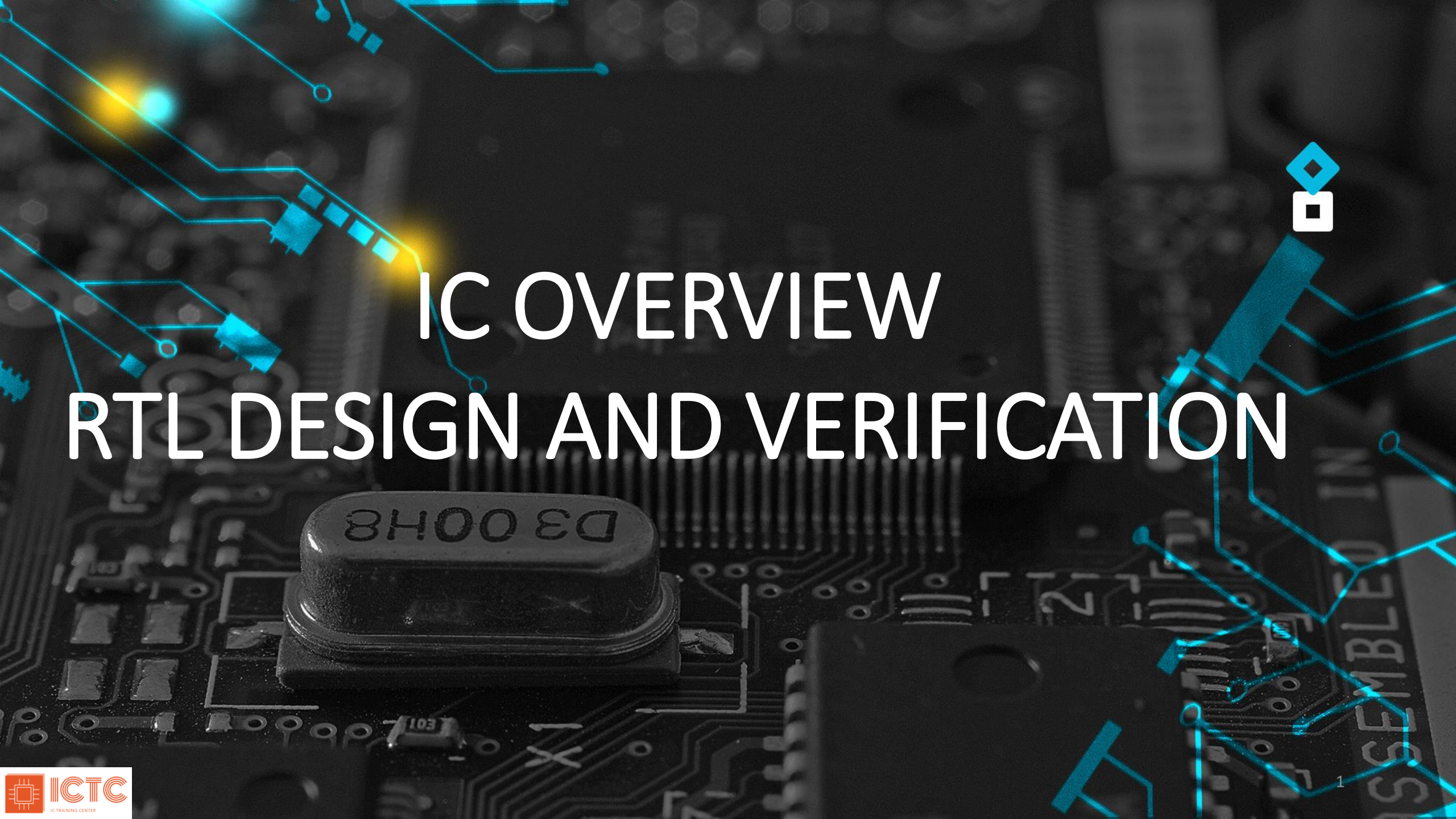
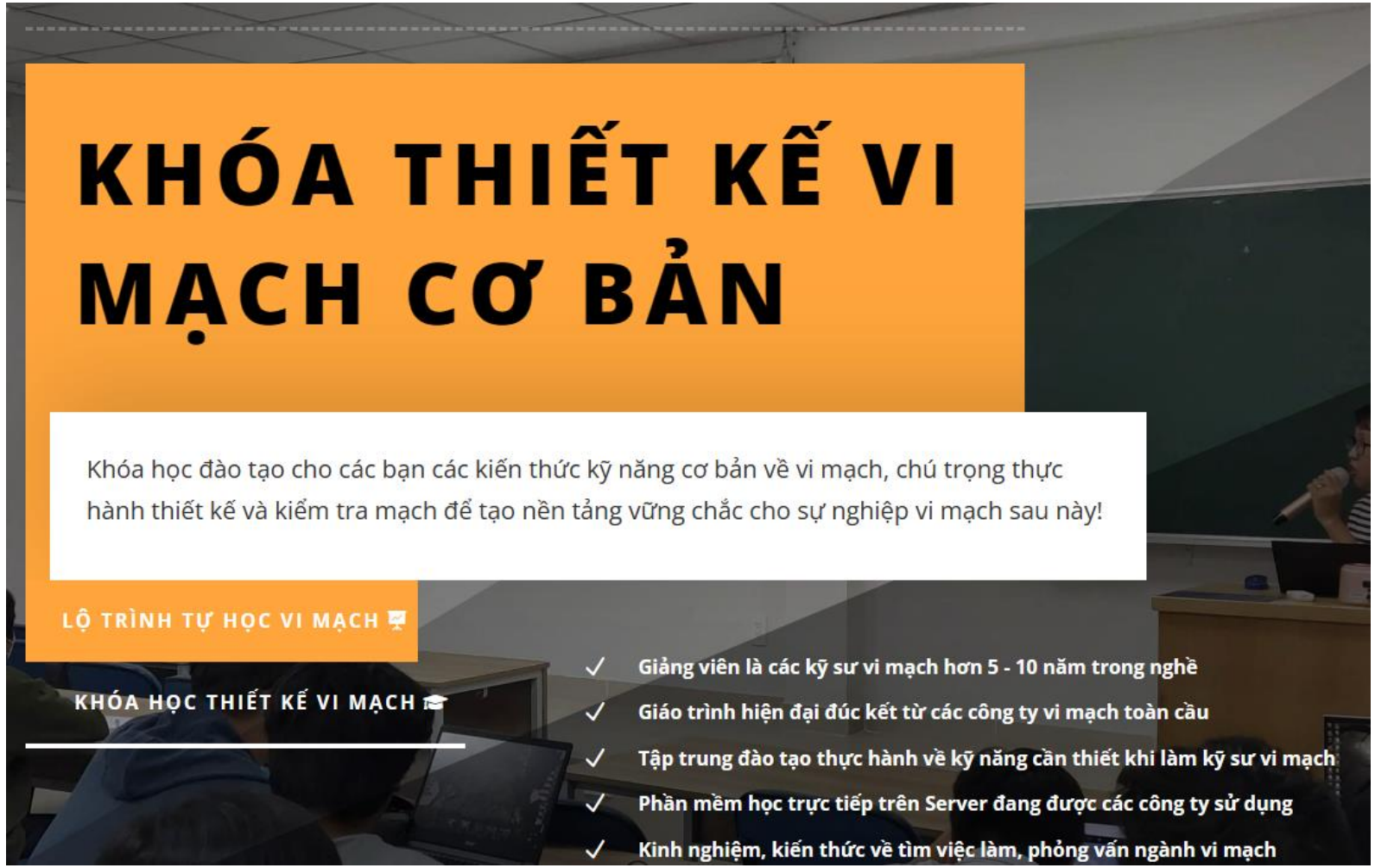


IC OVERVIEW
RTL DESIGN AND VERIFICATION

COURSE INTRODUCTION

[Khóa Học Thiết Kế Vi Mạch Cơ Bản - Trung Tâm Đào Tạo Thiết Kế Vi Mạch ICTC]



COURSE INTRODUCTION

SUMMARY

HOMEWORK

QUESTION

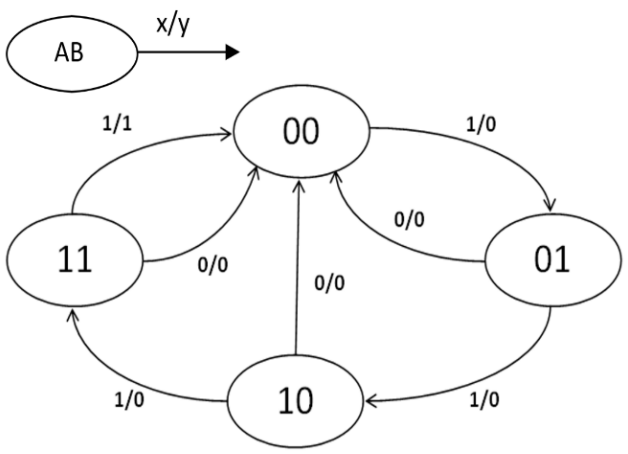
SELF-LEARNING

Session 14: Finite State Machine - FSM

FSM OVERVIEW

- FSM stands for Finite-State Machine.
- FSM is used to describe:
  - The behavior of a digital system with a finite number of states.
  - Transitions between these states.
  - Actions taken in response to these transitions.
- The FSM has advantages:
  - It makes it easy to control the operating sequence.
  - It simplifies the debugging process for the design.

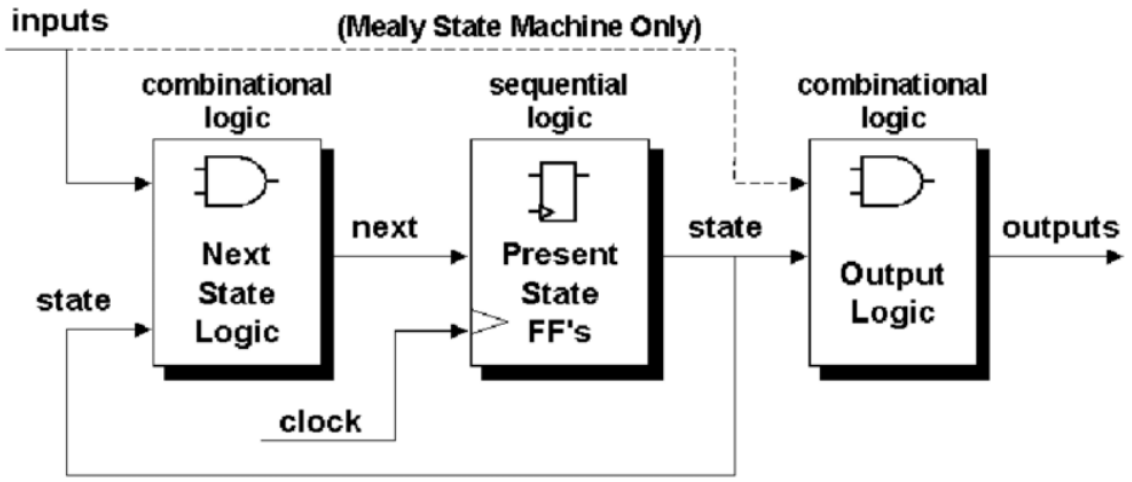
FSM is crucial for modeling and controlling sequential logic in digital circuits.



t			t+1		
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	0	0	1

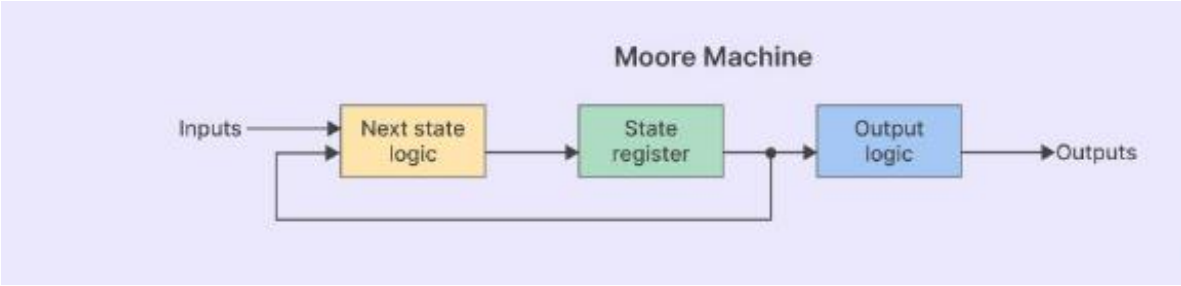
FSM COMPONENTS

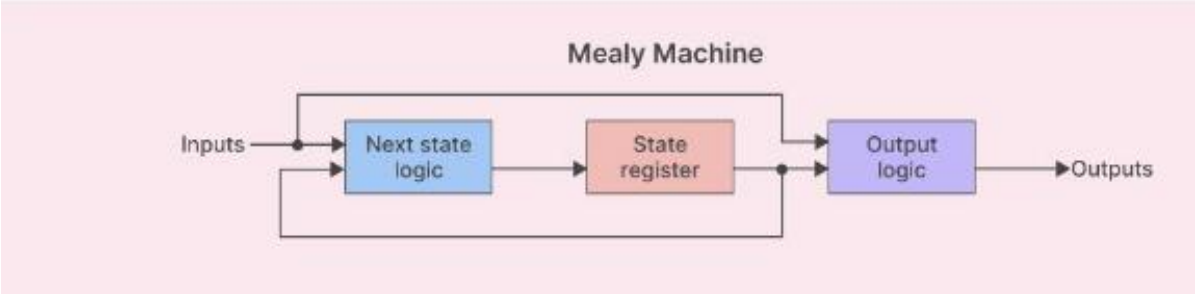
- An FSM consists of 3 fundamental components:
  - Next State Logic: This is a combinational circuit that depends on the inputs of the FSM and the current state from the state memory. It determines the next state based on the current state and inputs.
  - State memory: This component stores the current state of the FSM. It can be implemented using flip-flops or latches and receives inputs from the next state logic.
  - Output logic: this is another combinational circuit that generates outputs corresponding to the current state. In the case of a Mealy FSM, this block can also take inputs directly from the FSM's inputs, influencing the outputs based on both the current state and inputs.



FSM CLASSIFICATION

- FSM is divided into 2 types:
  - Moore FSM is a type with a circuit that generates outputs that do not directly depend on the FSM inputs
  - Mealy FSM is a type with a circuit that generates outputs directly depending on the FSM inputs.





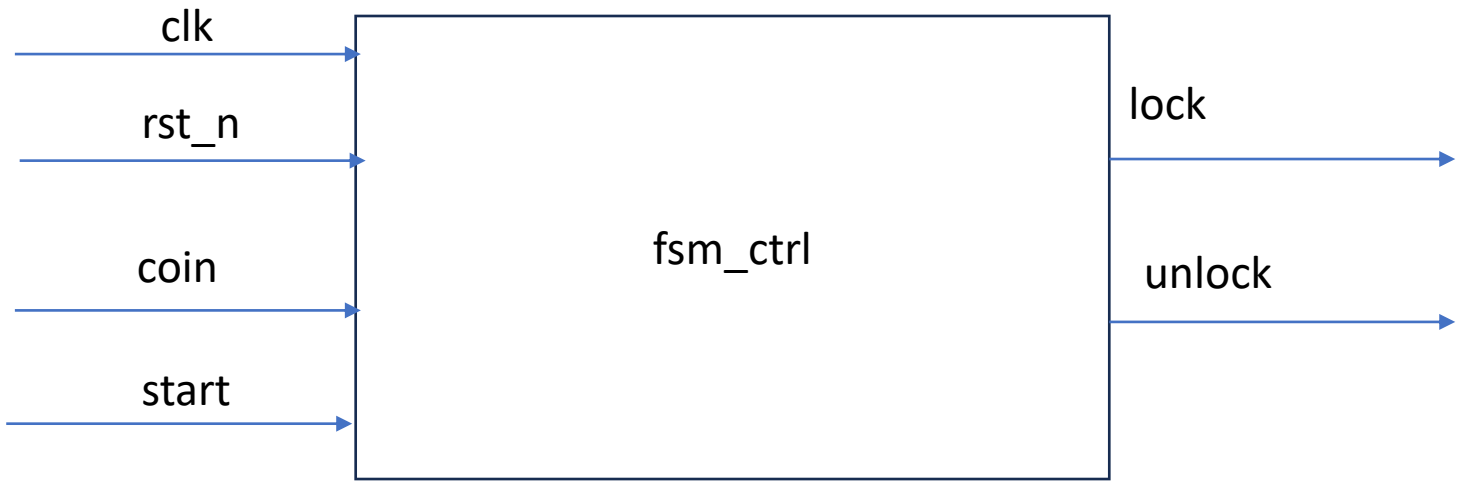
FSM CLASSIFICATION

Comparison

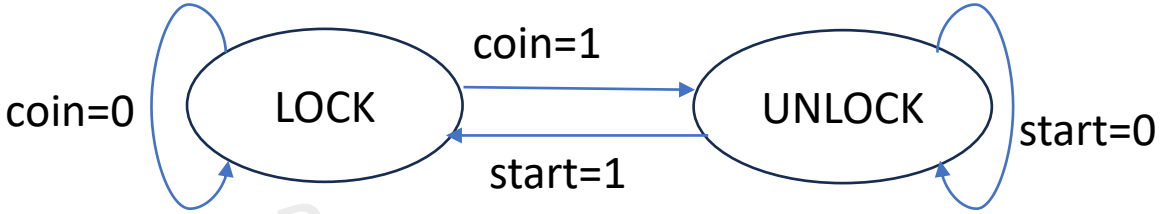
Num	Mealy FSM	Moore FSM
Output	Outputs are determined by both the current state and current inputs.	Outputs are determined by current state.
Complexity	More complex because outputs depend on both states and inputs, but this can lead to fewer states than an equivalent Moore machine.	More simple because the outputs are state-dependent and do not change with inputs within the same state. Moore FSM can have more states than Mealy FSM.
Timing	Worse. Outputs can change in response to input changes even within the same clock cycle. Critical path is from input to output.	Better. Outputs change only at state transitions (output of Flip-Flop)
Use cases	Suitable for applications where immediate reaction to input changes is critical.	Suitable for applications where stable outputs are required, and the reaction time to input changes is not critical.

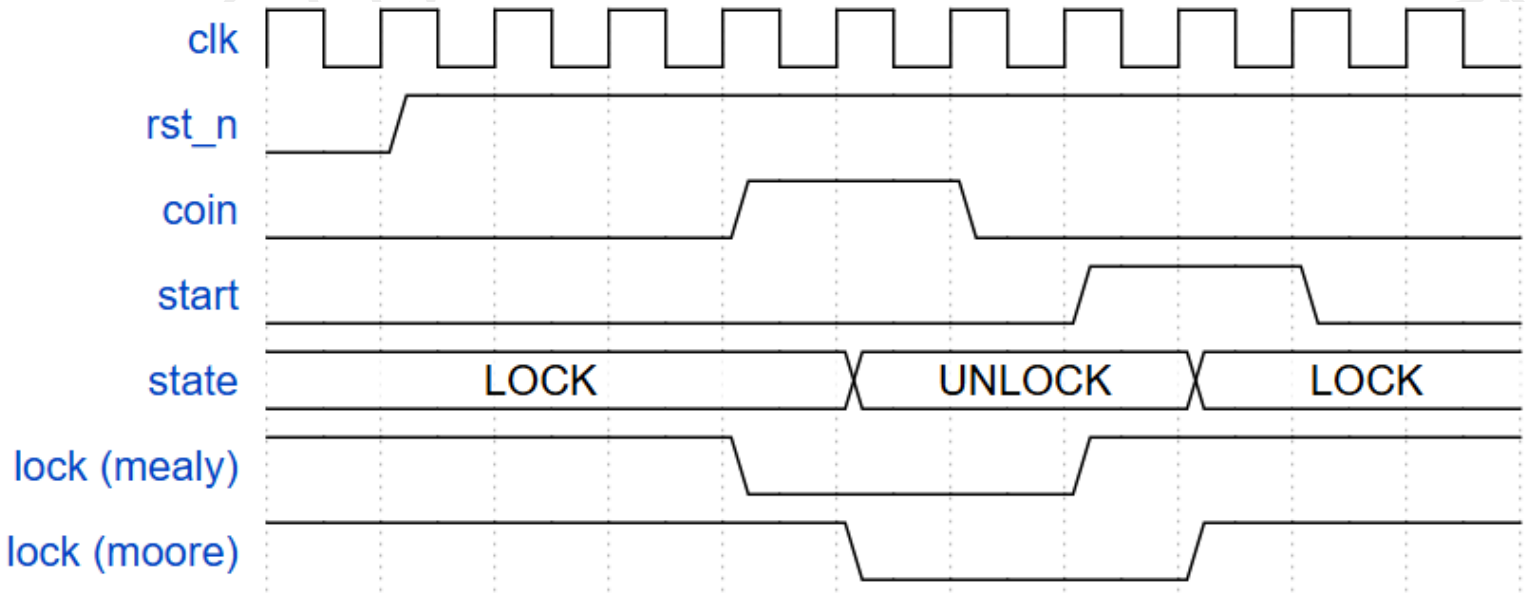
EXAMPLE OF FSM DESIGN

A game machine has 2 states: locked or unlocked. Normally, it is in the locked state; when a coin is inserted, it unlocks. When the start button is pressed, it locks again. Design FSM to control this game machine.



EXAMPLE OF FSM DESIGN





FSM TIMING DIAGRAM

Code wavedrom for FSM

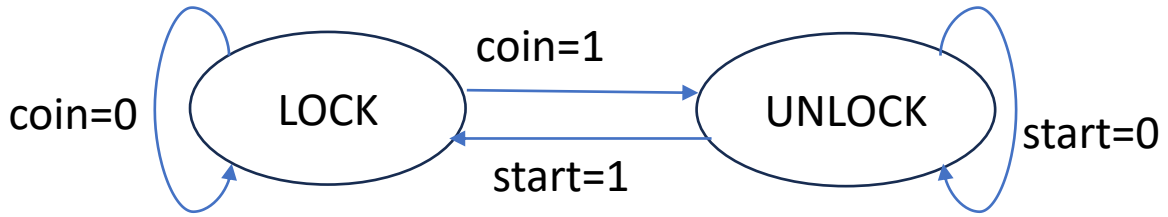
```
{signal: [
  {name: 'clk', wave: 'p.........'},
  {name: 'rst_n', wave: '01........'},
  {name: 'coin', wave: '0...1.0....'},
  {name: 'start', wave: '0......1.0.'},,
  {name: 'state', wave: '=....=..=..',data:["LOCK","UNLOCK","LOCK"]},
  {name: 'lock (mealy)', wave: '1...0..1...'},
  {name: 'lock (moore)', wave: '1....0..1..'},
]}
```

FSM DESIGN EXAMPLE

```
parameter LOCK = 1'b0;
parameter UNLOCK = 1'b1;

reg state;
reg nxt_state;

//next state logic
always @(*) begin
   case (state)
      LOCK: begin
         if( coin )
            nxt_state = UNLOCK;
         else
            nxt_state = state;
      end
      default: begin //UNLOCK
         if( start )
            nxt_state = LOCK;
         else
            nxt_state = state;
       end
   encase
end
```




```
//current state logic
always @(posedge clk or negedge rst_n) begin
   if( !rst_n)
      state <= LOCK;
   else
      state <= next_state;
end
```

FSM PRACTICE

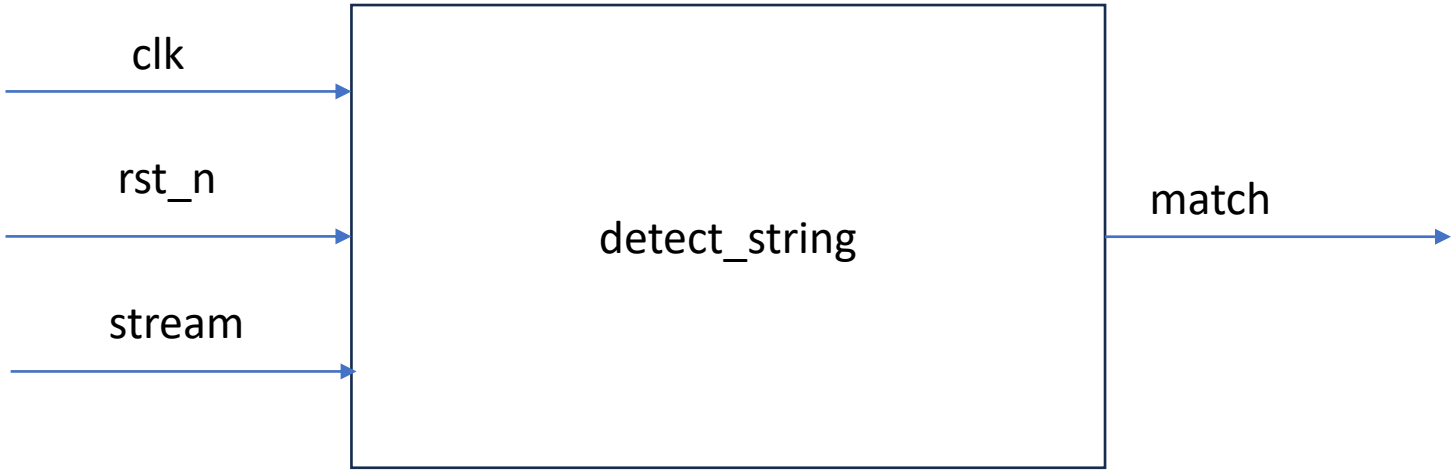
- Copy the tb provided in /ictc/student-data/share/teacher/14_ss14/fsm_ctrl
- Create fsm_ctrl.v for the above example
- Complete the design above. The design has parameter “MEALY_FSM”.
  - When MEALY_FSM is 1’b1, the output lock/unlock is Mealy FSM
  - When MEALY_FSM is 1’b0, the output lock/unlock is Moore FSM
- Run simulation
  - % make all : run simulation with Moore FSM
  - % make all MEALY_FSM=1 : run simulation with MEALY_FSM
- Check the result

FSM CLASSIFICATION

Every Moore machine can be converted to a Mealy machine and every Mealy machine can be converted to a Moore machine. Moore machine and Mealy machine are equivalent.

FSM PRACTICE

Design an FSM to detect a sequence of "1011" in a stream of bit.
Note: the left most bit come first.



Example:

*11001100**1011**00101010**1011**0**1011**111*

match match match match

Homework

Homework1:

- Finish the above practice. The output match is a Mealy FSM or Moore FSM using the parameter define MEALY_FSM same as previous example.
- Write simple testbench to test the design with the above given stream.

Homework2(*) – advanced level:

- Modify the testbench so that it can test with any given stream that has length define by parameter "**STR_LEN**".
- For example:
  - STR_LEN = 4 → the stream has 4 bits.
  - STR_LEN = 32 → the stream has 32 bits
- Note: stream is generated by testbench based on STR_LEN.
- **STR_LEN** also can be overriden by Makefile during execution