SMJE 3173
Digital System Design

Introduction to Finite State Machine Design

- Finite State Machine Concept
- Basic Design Procedure
- JASM (Just Another State Machine) Example
- Parity Checker Example
- Counter with Enable Example
- Complex Counter Example

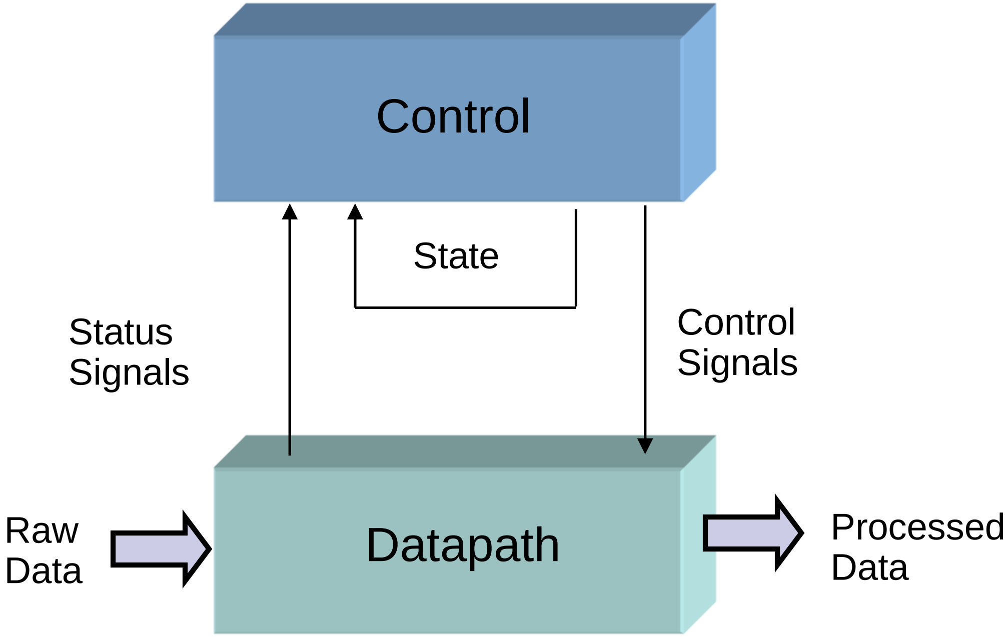
Finite State Machines

- **State**: collection of state variables containing all information from past needed to predict future behavior
- **Finite state machines** (FSMs): circuits that can be in only a fixed number of possible states
- A counter state transition diagram represents simple finite state machines.
  - State = output
  - No choice of sequence
- More generally, in FSM:
  - Next State = function of input and present state
  - Outputs = function of input and present state
  - More complex behavior than counters.
- Finite state machines perform decision-making logic

Concept of the State Machine

Computer Hardware = Datapath + Control

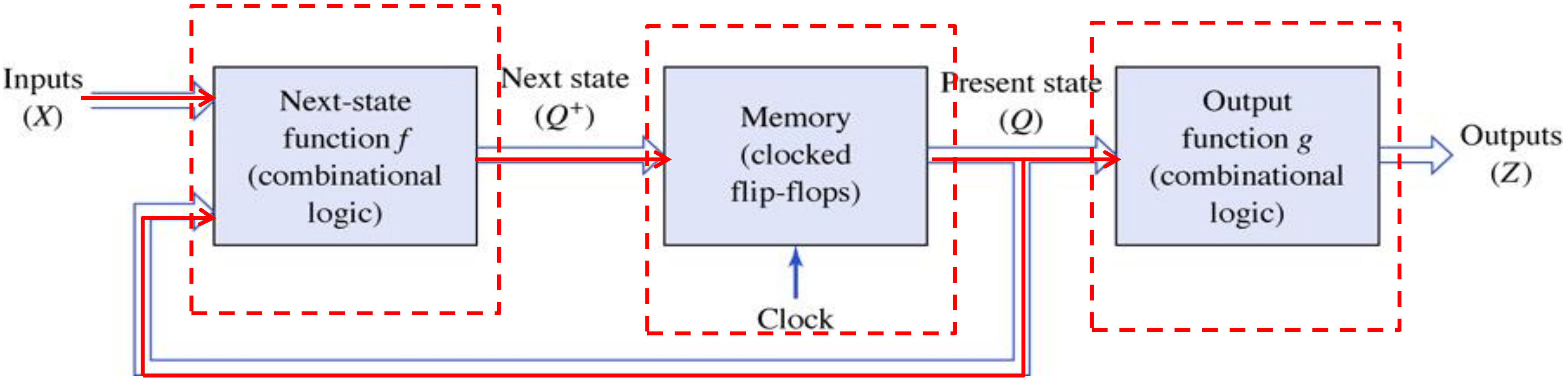
- FSM generating sequences of control signals
- Instructs datapath what to do next
- Registers
- Combinational Functional Units (e.g., ALU)
- Busses



State Machine Structure

- **State memory**:
  - ***n*** FFs to store current states. All FFs are connected to a common clock signal (i.e. synchronous)
- **Next-state logic**:
  - determine the next state when state changes occur
- **Output logic**:
  - determines the output as a function of current state and input
- There are three models for Finite State Machine (FSM)
  - Moore model
  - Mealy model
  - Synchronous Mealy model
- What are the differences between all these three models?

Moore Machine



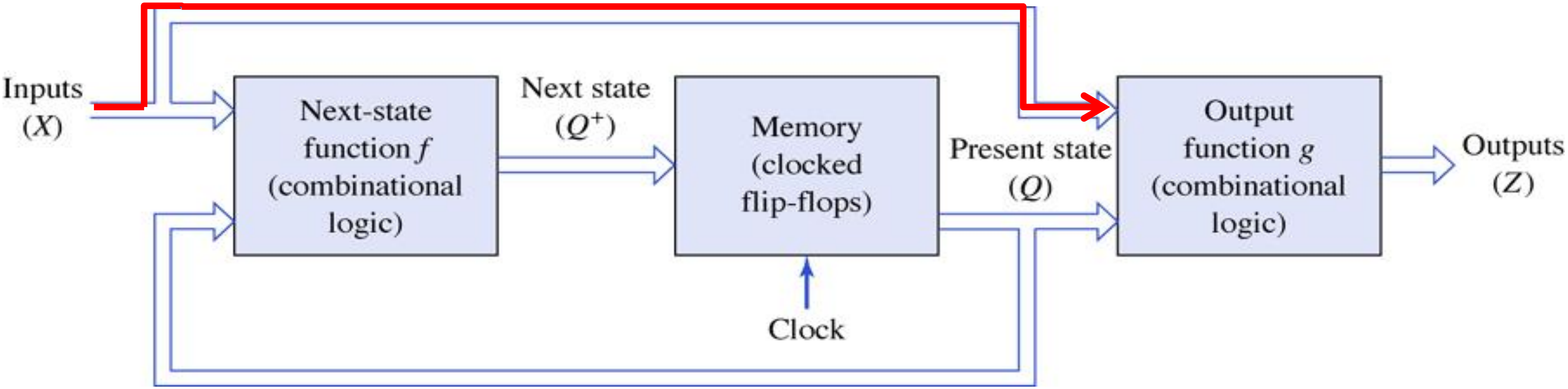
Moore Machine

Outputs are function solely of the current state (present state)

Outputs change synchronously with state changes

Outputs depend on current/present state only

Mealy Machine



Mealy Machine

Outputs depend on state AND inputs

Asynchronous signals: Input change causes an immediate output change (i.e. output change depends on present state and input to the circuit)

Moore vs Mealy

- **Moore**:
  - Generally **more** states required to solve a given problem
  - Easier to understand
  - Synchronous output (changes only with a clock pulse) -- The output is delayed in a Moore machine. Output does not occur until the next state change
  - Typically take more gates
  - Generally easier clocked (generally able to clock faster)
  - Easier to simulate using Quartus II

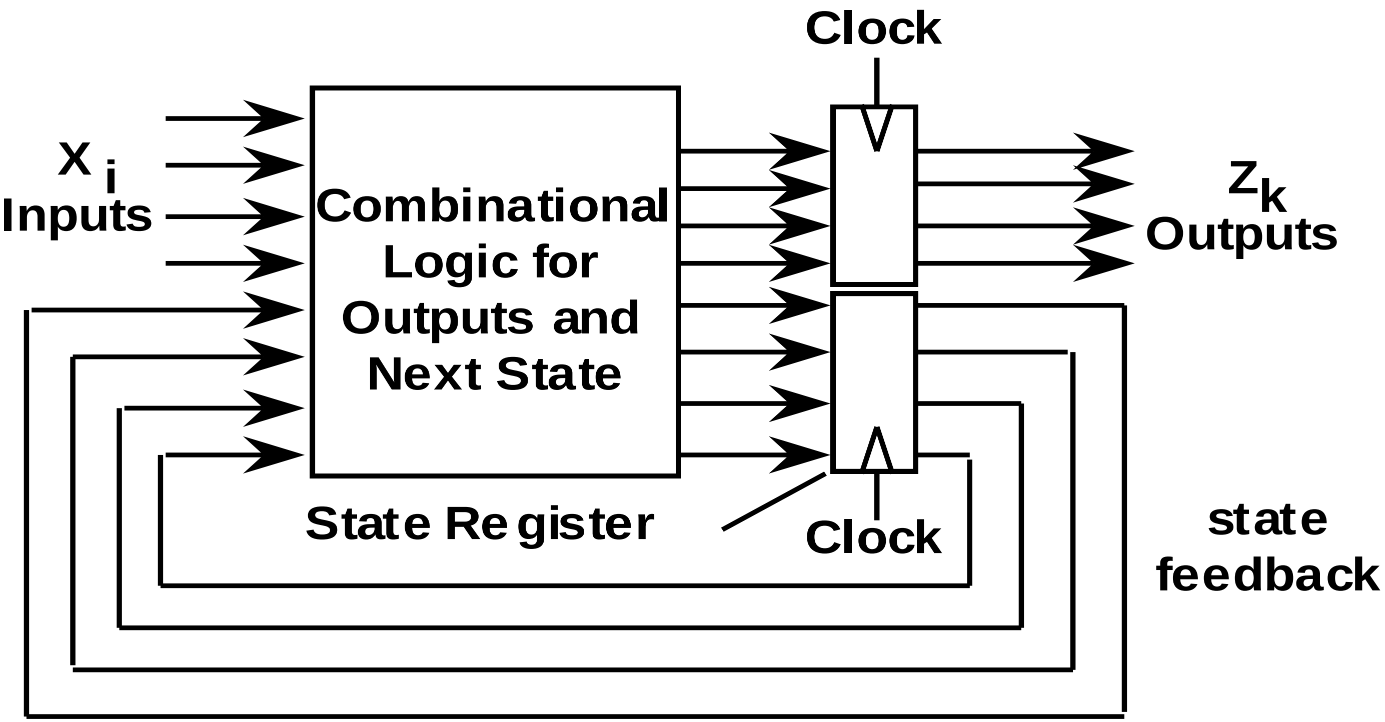
- **Mealy**:
  - Generally same or less states required
  - Slightly more complex to analyze
  - Asynchronous output (output can change any time an input changes) may lead to false outputs due to output changing after state changes
  - Generally requires less logic

Conclusion: Must know both, but learn Moore first

Synchronous Mealy

- Mealy model tend to has glitches in the output.
  - This is due to the asynchronous nature of the Mealy machine.
- Glitches are undesirable in real hardware controllers.
  - But because Mealy machines encode control in fewer states, saving on state register flip-flops, it is still desirable to use them.
- This leads to alternative synchronous design styles for Mealy machines.
- Simply stated, the way to construct a synchronous Mealy machine is to break the direct connection between inputs and outputs by introducing storage elements.

Synchronous Mealy Machine

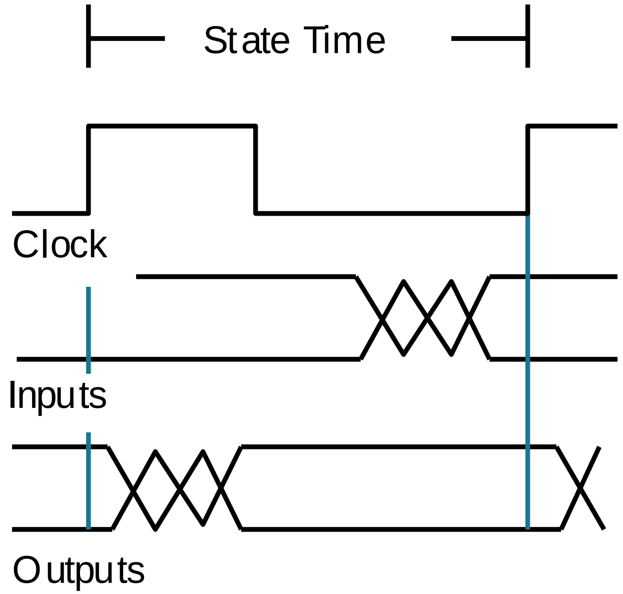


Combination of best ideas of Moore and Mealy:
Less logic + synchronous output

latched state AND outputs

avoids glitchy outputs!

State Machine Timing



- State Time:
  - □ Time between clocking events.
  - □ Within the whole clock period
- Clocking event:
  - □ inputs sampled
  - □ outputs, next state computed
- After propagation delay
  - □ outputs stable
  - □ next state entered
- Moore vs Mealy:
  - □ Asynchronous signals take effect immediately
  - □ Synchronous signals take effect at the next clocking event
- Immediate Outputs affect datapath immediately
- Delayed Outputs take effect on next clock edge
  - □ Important for synchronous Mealy
- For set-up/hold time considerations:
  - □ Inputs should be stable before clocking event

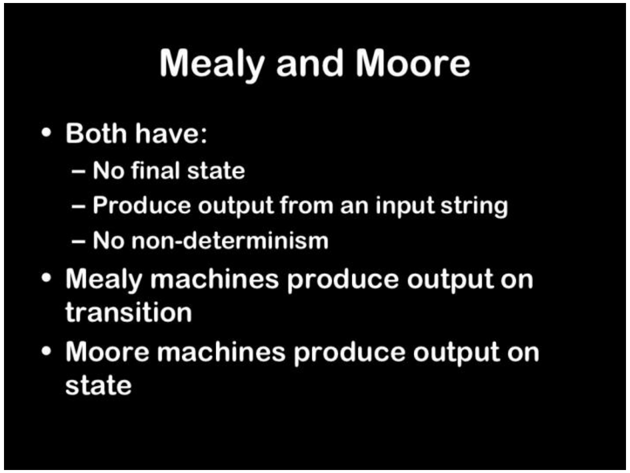
Basic Design Approach

- Eight-step process
  1. Understand the statement of the Specification
  2. Draw a state diagram
  3. Convert state diagram to state table
  4. Optionally, perform state minimization
  5. Perform state assignment
  6. Obtain next state and output equations
  7. Optionally, choose a flip flop type other than DFF and derive the flip flop input maps or tables.
  8. Implement (Draw circuit realization, enter design & verify)

Example 1: Sequence detector for 01 or 10 (Moore Machine)

Mealy and Moore

- Moore: outputs depend on current state only (Output does not react immediately to input change)
- Mealy: outputs depend on current state and inputs



Mealy and Moore

- Both have:
  - No final state
  - Produce output from an input string
  - No non-determinism
- Mealy machines produce output on transition
- Moore machines produce output on state

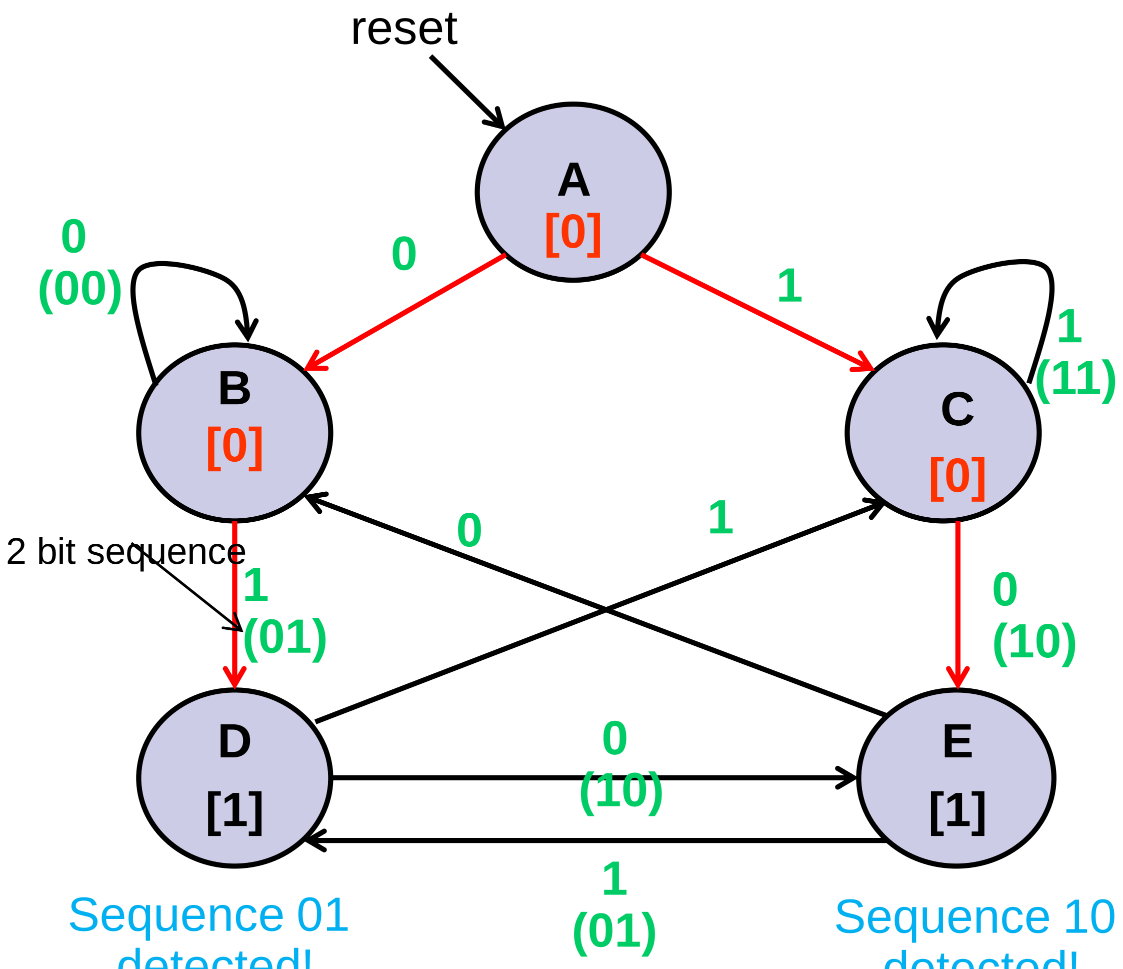
Understand the problem specification

Clock cycle :	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
input	0	0	0	1	1	1	0	0	0	0	0
Output:	0	0	0	0	1	0	0	1	0	0	0

Example 1: Sequence detector for 01 or 10 (Moore Machine)

1) Draw the complete state diagram:

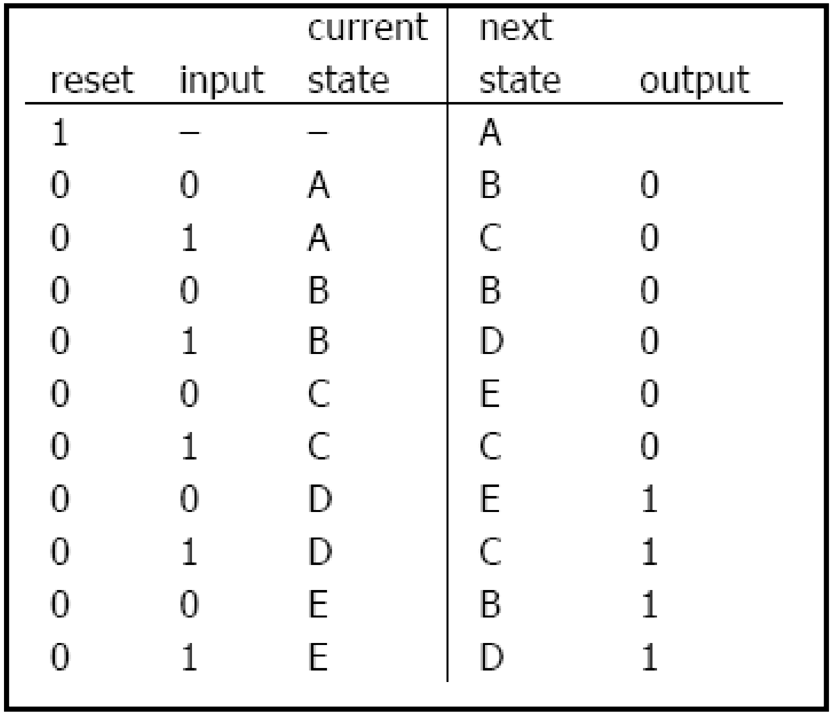
Start with the expected sequence first



Sequence 01 detected!

Sequence 10 detected!

2) Derive state table



reset	input	current state	next state	output
1	–	–	A	
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

a.k.a. symbolic state table, state name as used in state diagram

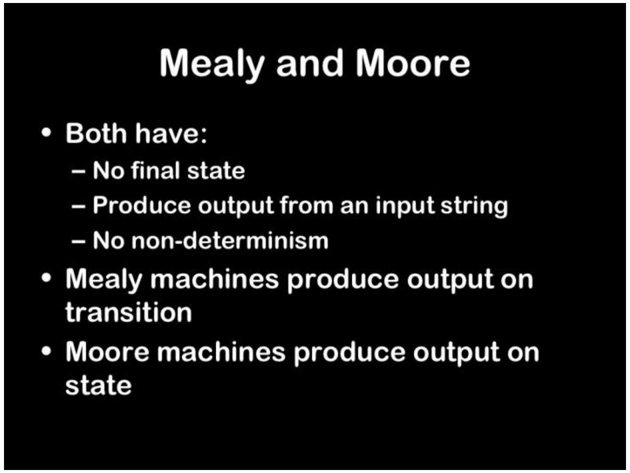
- Step 4: Perform state assignment:
  - Use “simple” binary encoding:
    - State ‘A’ >> S0 = 000
    - State ‘B’ >> S1 = 001
    - State ‘C’ >> S2 = 010
    - State ‘D’ >> S3 = 011
    - State ‘E’ >> S4 = 100

Present State	Input	Next State	Output
$PS_2PS_1PS_0$	IN	$NS_2NS_1NS_0$	OUT
000 (A)	0	001 (B)	0
	1	010 (C)	
001 (B)	0	001 (B)	0
	1	011 (D)	
010 (C)	0	100 (E)	0
	1	010 (C)	
011 (D)	0	100 (D)	1
	1	010 (C)	
100 (E)	0	001 (B)	1
	1	011 (D)	
101	x	x	x
110	x	x	x
111	x	x	x

Example 1: Sequence detector for 01 or 10 (Mealy Machine)

Mealy and Moore

- Moore: outputs depend on current state only (Output does not react immediately to input change)
- Mealy: outputs depend on current state and inputs



Mealy and Moore

- Both have:
  - No final state
  - Produce output from an input string
  - No non-determinism
- Mealy machines produce output on transition
- Moore machines produce output on state

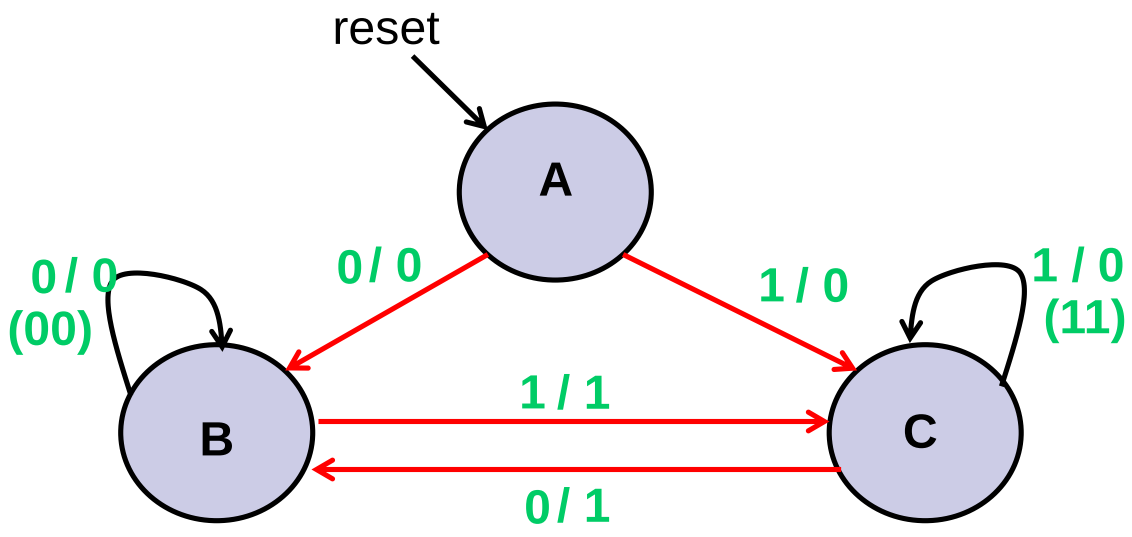
Understand the problem specification

Clock cycle :	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
input	0	0	0	1	1	1	0	0	0	0	0
Output:	0	0	0	1	1	0	1	1	0	0	0

Example 1: Sequence detector for 01 or 10 (Mealy Machine)

1) Draw the complete state diagram:

Start with the expected sequence first

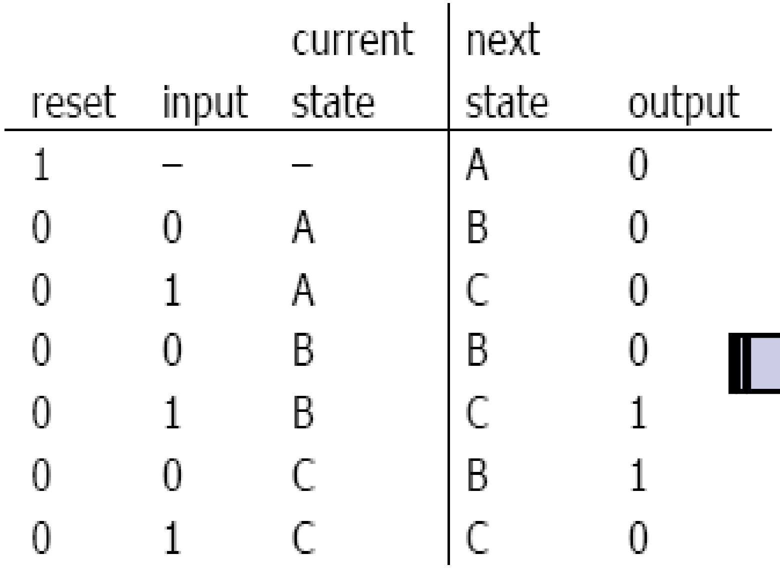


2) Derive state table

		current	next	
reset	input	state	state	output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

Sequence 10
detected!

- Step 4: Perform state assignment:
  - Use “simple” binary encoding:
    - State ‘A’ >> S0 = 00
    - State ‘B’ >> S1 = 01
    - State ‘C’ >> S2 = 10



reset	input	current state	next state	output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

Encoded State Table

Present State	Input	Next State	Output
PS_1PS_0	IN	NS_1NS_0	OUT
00 (A)	0	01 (B)	0
	1	10 (C)	0
01 (B)	0	01 (B)	0
	1	10 (C)	1
10 (C)	0	01 (B)	1
	1	10 (C)	0

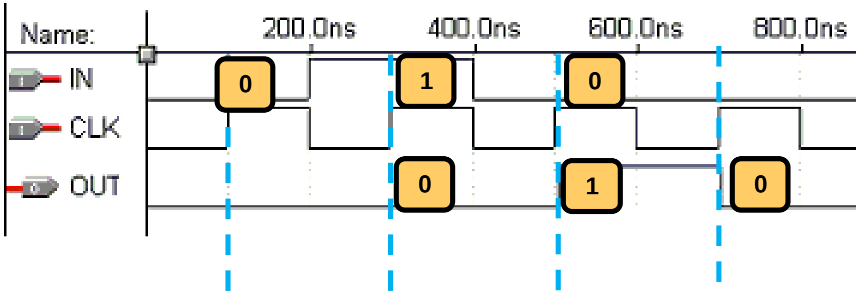
Example 2: JASM (Just Another State Machine)

- The specification:
  - □ An idle system is activated when an input, **IN** is given. Then, an output, **OUT** is produced after one interval time or cycle later. Next, the system will be back to the idle state, waiting for the next triggering input **IN**.

- Step 1: Understand the specs.
  - □ Get a sample input/output relationship. More may be needed later.
    - Sample input/output relationship:

      IN : 001001110

      OUT : 000100101



  - □ Draw a simple block diagram and the steps involve.

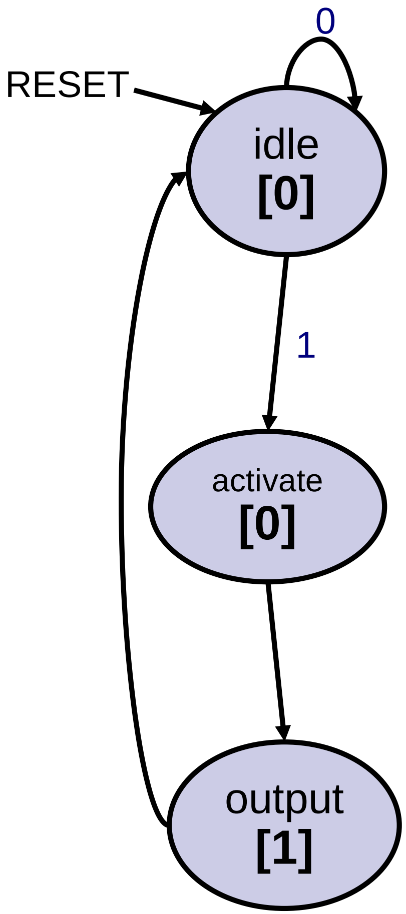
    1) Activate the system from idle state upon receiving an input; i.e. IN=1)

    2) One cycle later produced an output; i.e. OUT=1

    3) Then return the system to idle state

JASM State Transition Diagram (Moore)

Step 2: Draw state transition diagram



Step 3: Derive symbolic state transition table

Present State	Input	Next State	Output	Comments
	IN		OUT	
idle	0	idle	0	Remain in idle state if input does not change
	1	activate		Go to next state (activate state) if input is 1
activate	0	output	0	Go to next state (output state) no matter what the input is.
	1	output		
output	0	idle	1	Return to idle state no matter what the input is. Output is high in this state.
	1	idle		

JASM Encoded State Table (Moore)

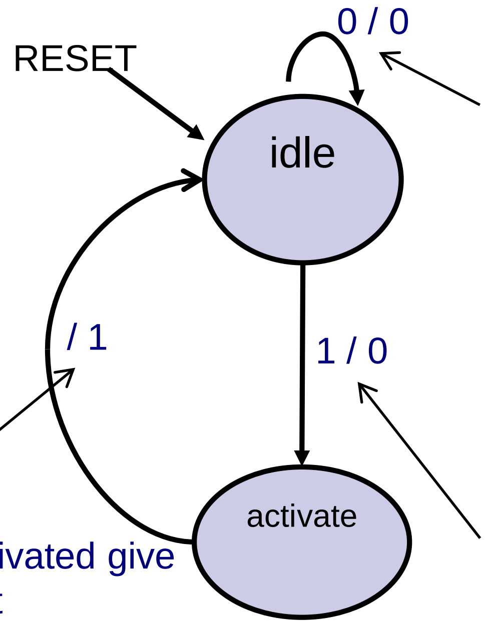
- Step 4: Perform state assignment:
  - □ **Use of "simple" binary encoding gives us: S0 = 00, S1 = 01 and S2 = 10.**
  - □ Must also add in code 11 to take care of don't cares.
  - □ Here, if we somehow get to state 11, next state & output are don't cares.

Present State	**Input**	**Next State**	**Output**
PS_1PS_0	**IN**	**NS_1NS_0**	**OUT**
00	0	00	0
	1	01	
01	0	10	0
	1	10	
10	0	00	1
	1	00	
11	0	11	X
	1	11	

JASM State Transition Diagram (Mealy)

Step 2: Draw state transition diagram

- Step 3: get symbolic state table.



If no input received, remain in idle state

If input is received move to activate (next) state i.e. Output of 0 triggered by input of 1

Once activated give an output regradless of the input condition. i.e.Output of 1 triggered unconditionally

Present State	Input	Next State	Output
	IN		OUT
idle	0	idle	0
	1	activate	0
activate	0	idle	1
	1	activate	1

→ idle

JASM Symbolic & Encoded State Table (Mealy)

refer Step 3: symbolic state table.

Present State	Input	Next State	Output
	IN		OUT
idle	0	idle	0
	1	activate	0
activate	0	idle	1
	1	idle	1

- Step 4: minimize state transition (SKIP).

Step 5: Perform state assignment

Simple binary encoding of states

Idle : S0 = 00

Activate : S1 = 01

Get encoded state table.

Present State	Input	Next State	Output
PS	IN	NS	OUT
0	0	0	0
	1	1	
1	0	0	1
	1	0	

Example 3: Coin-operated turnstile

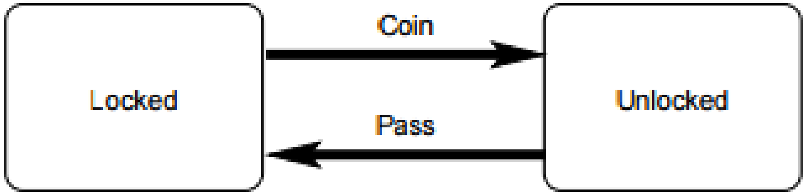
Problem description

The gate is initially 'locked" and will not rotate to allow a passenger to pass through.

When a coin or token is deposited into the receiver, the gate is "unlocked" but does not turn until the person actually pushes on it and passes through the turnstile, at which time the gate locks until the next coin is deposited

From the description, it should be clear that the turnstile is always in exactly 1 of 2 possible states: **Locked** or **Unlocked**.

The system can be modelled graphically, using rounded rectangles to represent states and arrows to represent transitions between states



Simple State Machine Model of a Turnstile

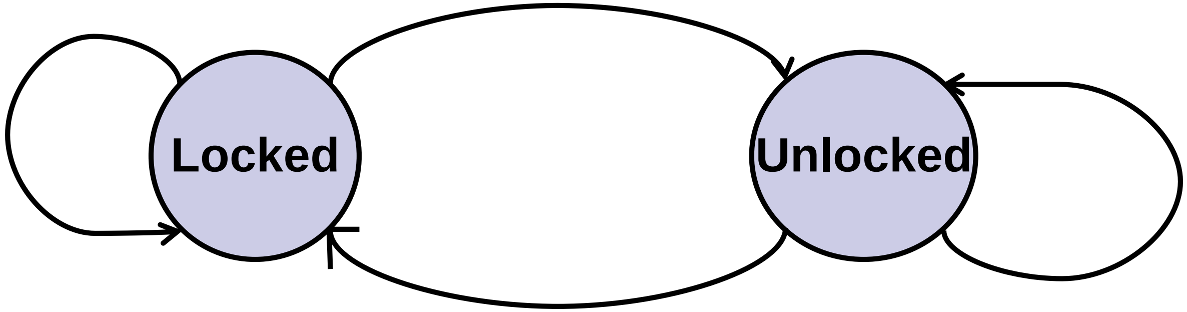
Defining Problem Specification

However there are two inputs that affect its transition from one state to the other: putting a coin in the slot (***coin***) and pushing the arm (***push***).

-In the locked state, pushing on the arm has no effect; no matter how many times the input ***push*** is given, it stays in the locked state.

-Putting a coin in – that is, giving the machine a ***coin*** input – shifts the state from ***Locked*** to ***Unlocked***. In the unlocked state, putting additional coins in has no effect; that is, giving additional ***coin*** inputs does not change the state.

-However, a customer pushing through the arms, giving a ***push*** input, shifts the state back to ***Locked***.



If coin inserted, move from locked to unlocked state

Coin=1
Push = x

Remains unlocked even if extra coin is inserted with no push

Coin=0
Push = 1

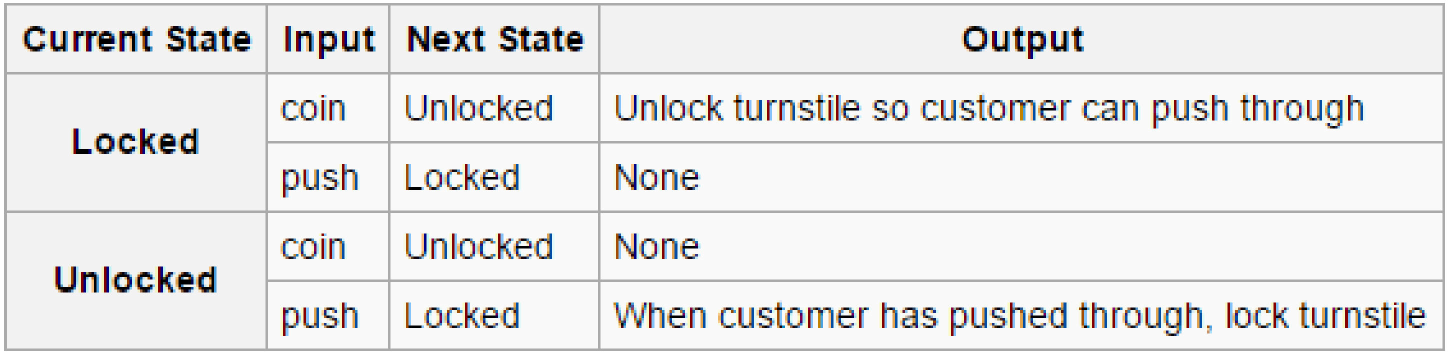
Coin=1
Push = 0

Remains locked if no coin even if its pushed

Push =1
Coin = x

If push, move from unlocked to locked state

Symbolic state transition table



Current State	Input	Next State	Output
Locked	coin	Unlocked	Unlock turnstile so customer can push through
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When customer has pushed through, lock turnstile

Perform state assignment

Simple binary encoding of states

Locked : S0 = 00, Unlocked : S1 = 01

Get encoded state table.

Present State	Input	Next State	Output
PS	IN	NS	OUT
S0 (Locked)	Coin	S0 (Unlocked)	unlocked
	Push	S1 (Locked)	No action
S1 (Unlocked)	Coin	S1 (UnLocked)	No action
	Push	S1 (Locked)	locked

Example 2: Bit Sequence Detector (BSD)

- The specification:
    - □ An input is used to detect a sequence or a series of inputs, 110. When the specific sequence is detected, an output high is produced for a cycle. Then, the system will continue detecting the next sequence.
- Motivation
    - □ The sequence detector circuit has a practical application in code encoding and decoding such as Huffman codes

- Step 1: Understand the specs.
    - □ Get a sample input/output relationship.
        - Sample input/output relationship:

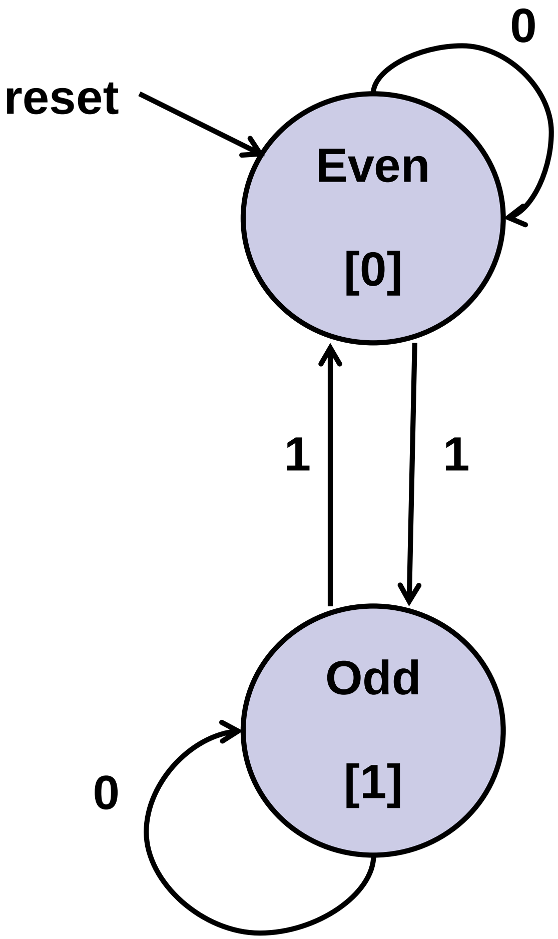
          IN : 1100011011110…

          OUT : 0010000100001…

Example 4: Odd Parity Checker

- **The specification:**
  - □ Assert output whenever an input bit stream has odd # of 1's. Output is the same as the state
- **Step 1: Understand the specs.**
  - □ Get a sample input/output relationship. (result : 1 = detect || 0= not detected)
    - Input bit #1: **0** ←
      Output bit #1: 0 >> result = 0 because the number of 1’s detected is 0 (even)
    - Input bit#2 : 0**1**←
      Output bit #2 : 1 >> result = 1 because the number of 1’s detected is 1 (odd)
    - Input bit#3: 01**1** ←
      Output : 0 >> result = 0 because the number of 1’s detected is 2 (even)
    - Input bit #4:011**0** ←
      Output : 0 >> result = 0 because the number of 1’s detected is 2 (even)
    - Input bit#5: 0110**1** ←
      Output : 1 >> result = 1 because the number of 1’s detected is 3 (odd)
    - Input : 01101**0** ←
      Output : 1 >> result = 1 because the number of 1’s detected is 3 (odd)

Step 2 : Obtain state diagram

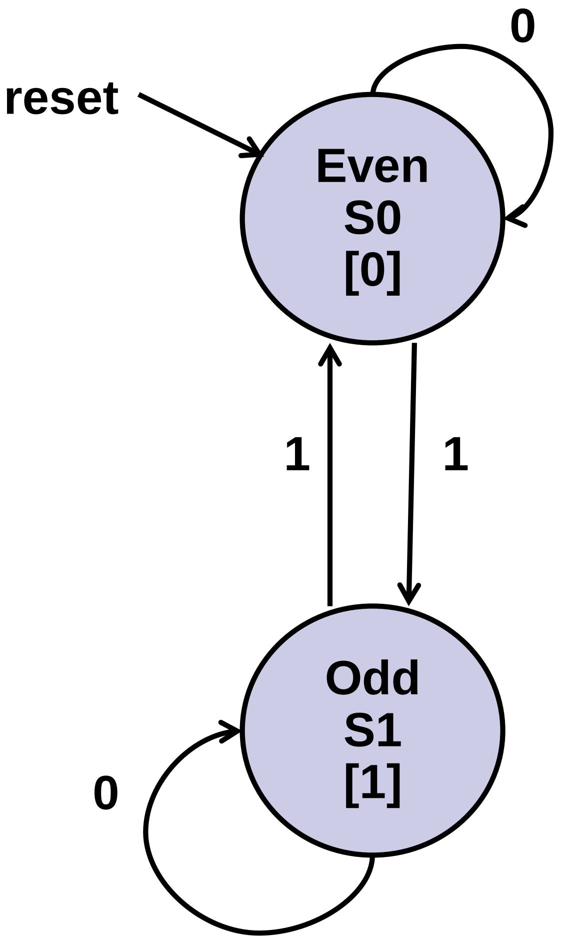


- circuit is in one of two states.
- transition on each cycle with each new input, over exactly one arc (edge).
- Output depends on which state the circuit is in.

Step 3 : Obtain state table

Present State	i/p	Next State	o/p
Even (S0)	0	Even (S0)	0
Even (S0)	1	Odd (S1)	0
Odd (S1)	0	Odd (S1)	1
Odd (S1)	1	Even (S0)	1

Step 4 : Obtain encoded state diagram



- Encode : S0 → 0
- Encode : S1 → 1

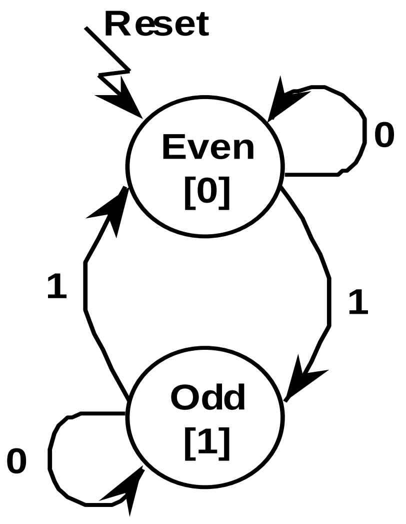
Present State	i/p	Next State	o/p
0 (S0)	0	0 (S0)	0
0 (S0)	1	1 (S1)	0
1 (S1)	0	1 (S1)	1
1(S1)	1	0 (S0)	1

Step 6: Next state & Output Equations

NS = PS ⊕ INPUT;   OUTPUT = PS

Odd Parity Checker

- Steps 2,3,5: State Diagram, symbolic state table, & encoded state table



State Diagram

Present State	Input	Next State	Output
Even	0	Even	0
Even	1	Odd	0
Odd	0	Odd	1
Odd	1	Even	1

Symbolic State Transition Table

Present State	Input	Next State	Output
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

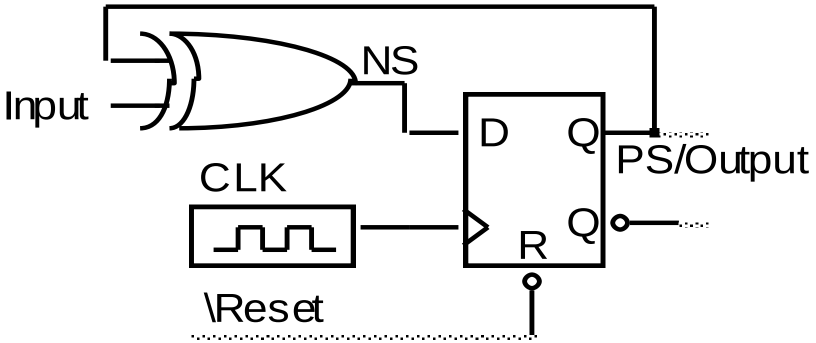
Encoded State Transition Table

Step 6: Next state & Output Equations

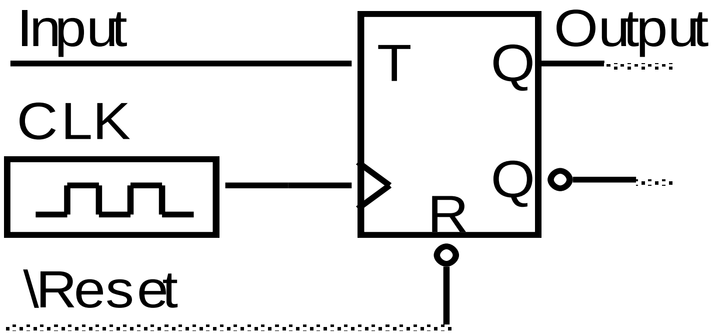
NS = PS xor INPUT;   OUTPUT = PS

Odd Parity Checker

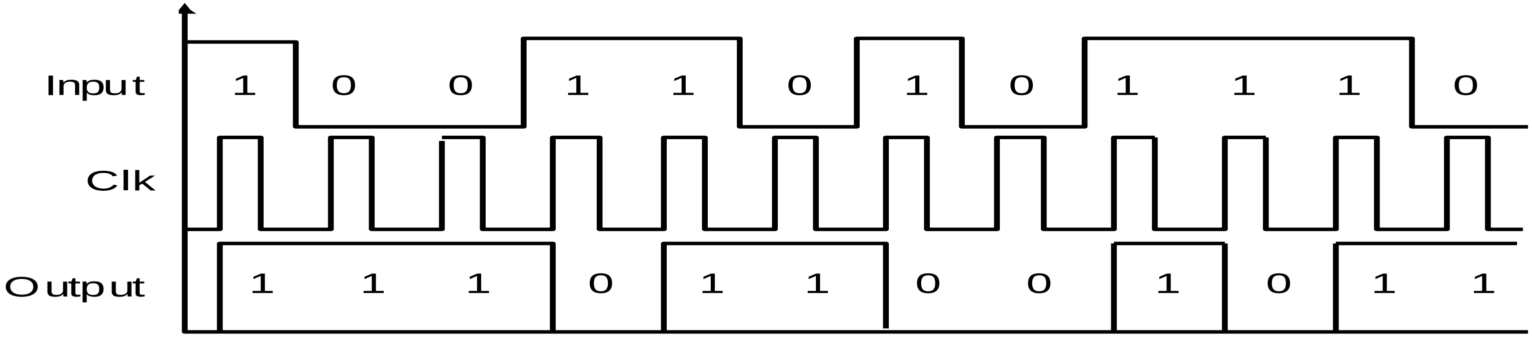
Steps 7 & 8: Implementation (DFF & TFF)



D FF Implementation



T FF Implementation



Timing Behavior: Input 1 0 0 1 1 0 1 0 1 1 1 0

Example 4: Dual-Mode Counter

Skip !!

- A sync. 3 bit counter has a mode control M. When M = 0, the counter counts up in the binary sequence. When M = 1, the counter advances through the Gray code sequence.

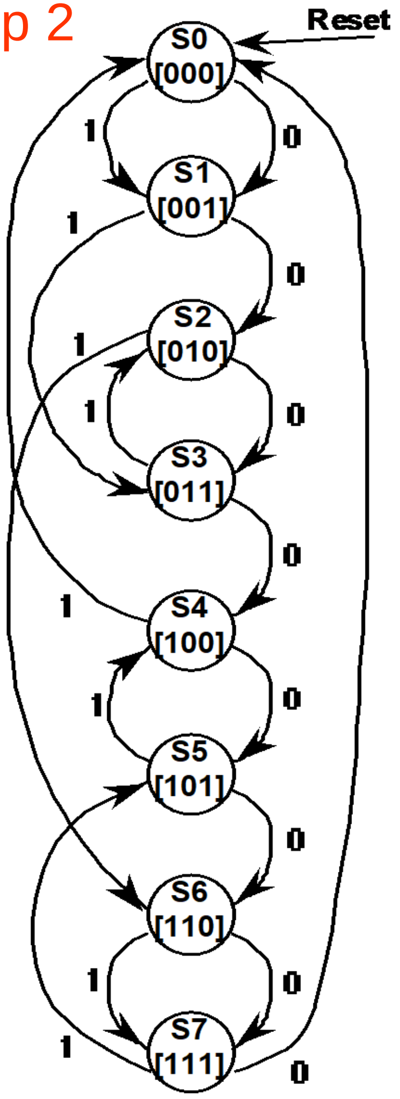
Step 1 : Understanding the problem specification

- List possible sequences to understand the problem.
  - □ Binary: 000, 001, 010, 011, 100, 101, 110, 111
  - □ Gray: 000, 001, 011, 010, 110, 111, 101, 100

Dual-Mode Counter

Skip !!

Step 2



One state for each output combination

Add appropriate arcs for the mode control

State assign:

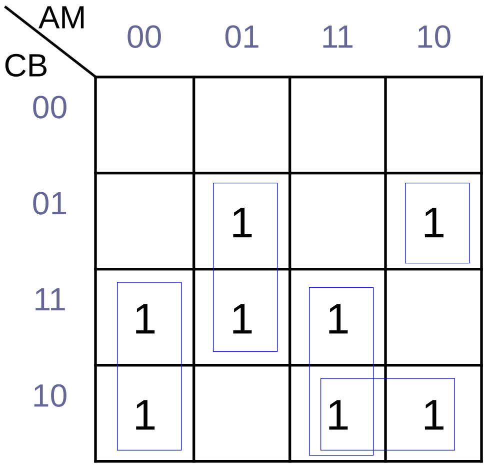
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

Step 3,5

Present State			Input	Next State		
C	B	A	M	DC	DB	DA
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	0
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	1	1	0	1

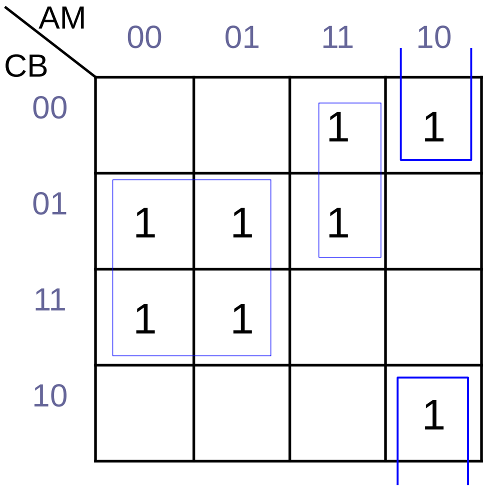
Dual-Mode Counter

Skip !!



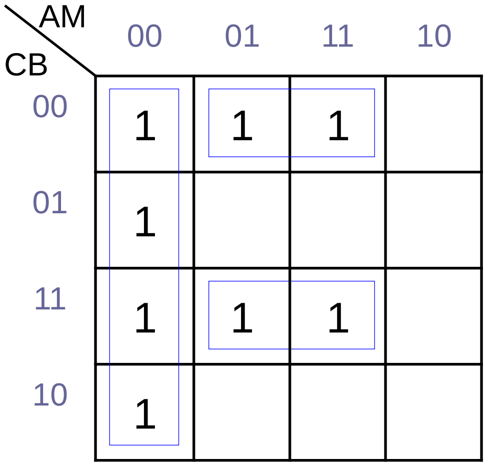
CB \ AM	00	01	11	10
00				
01		1		1
11	1	1	1	
10	1		1	1

DC = CA’M’ + BA’M + CAM + CB’A + C’BAM’



CB \ AM	00	01	11	10
00			1	1
01	1	1	1	
11	1	1		
10				1

DB = BA’ + C’AM + B’AM’



CB \ AM	00	01	11	10
00	1	1	1	
01	1			
11	1	1	1	
10	1			

DA = A’M’ + C’B’M + CBM

Dual-mode Counter Circuit

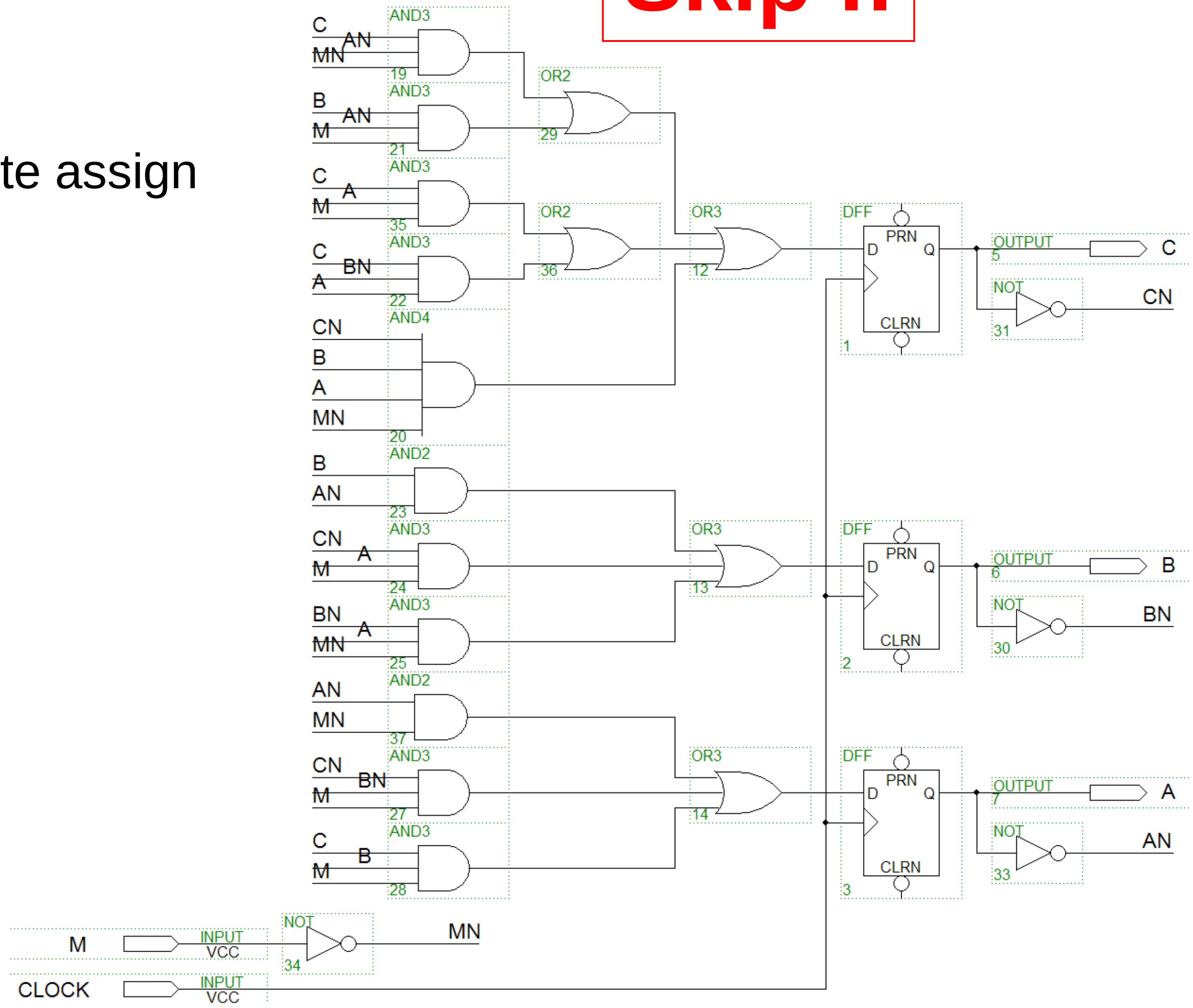
Skip !!

- Exercise:

  obtain circuit using state assign below and TFFs

State assign:

S0	000
S1	001
S2	011
S3	010
S4	110
S5	111
S6	101
S7	100



Verilog Construct of FSM -Moore Model

Skip !!

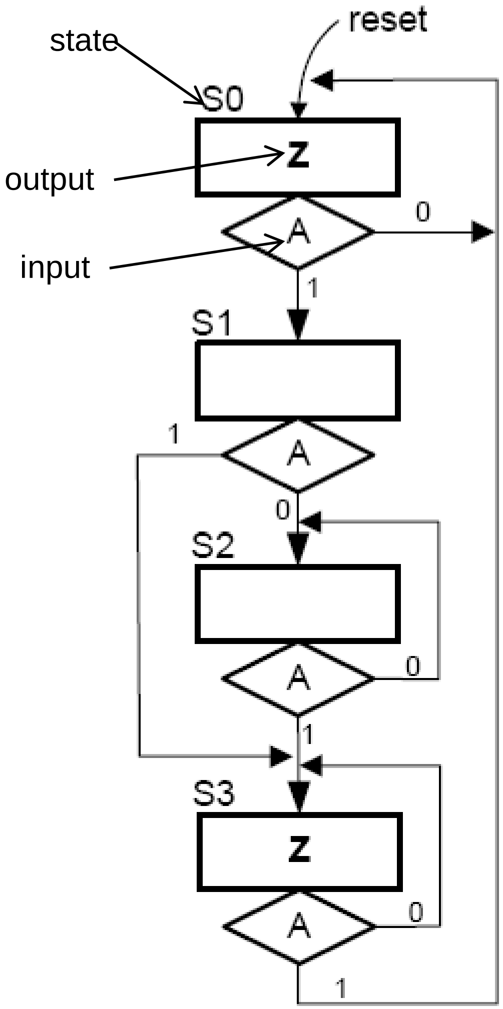
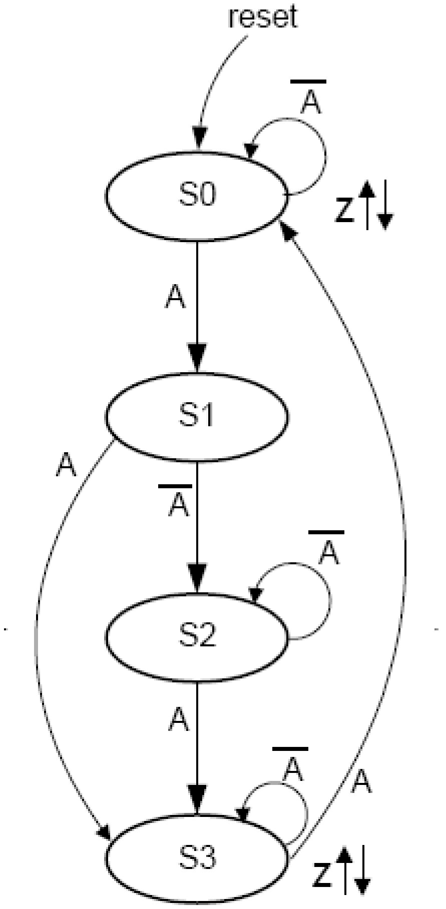


Fig. 5-2a ASM-Chart of FSM1



Hardware construct of the FSM

1) The FSM has 4 states (S0 – S3) implying that the states variable will be 2 (00 -11).

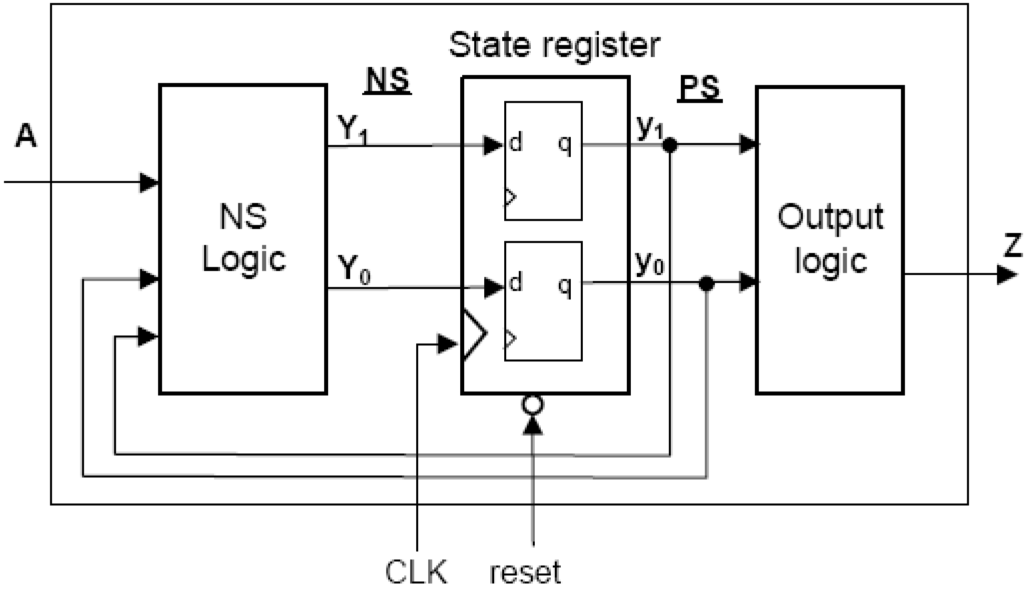
2) Two (2) F/F is required to be form as state register.

3) The machine has 1 i/p (A) and 1 o/p (Z)

4) The machine is a Moore FSM since output Z only depends on the present state.

Style A: Verilog Construct of FSM -Moore Model

Skip !!

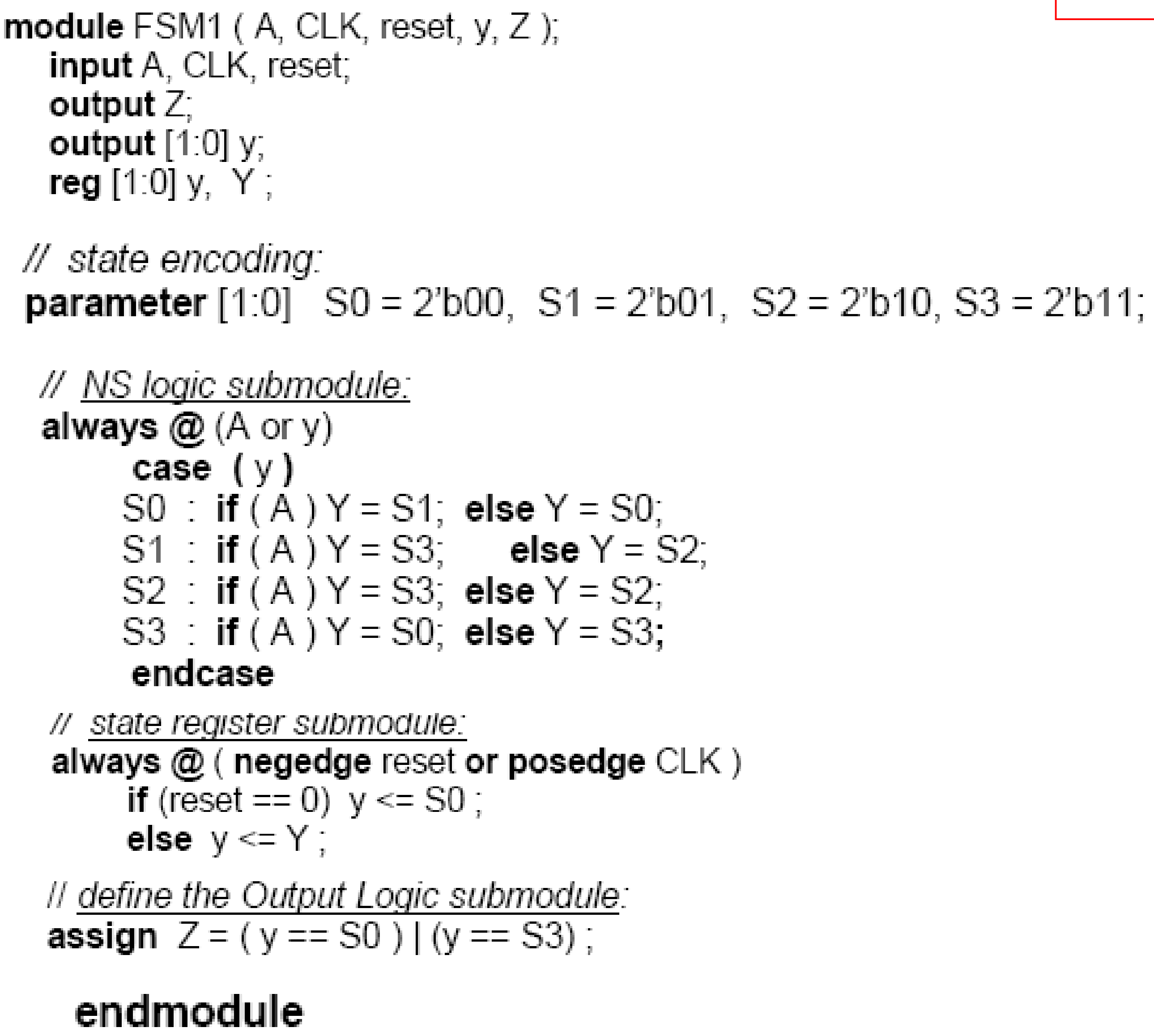


- There are three separate coding segments:

  one for the *NS logic submodule* (via an **always** block),
  one for the *state register* (via an **always** block), and
  one for the *output logic submodule* (via an **assign** statement).

Style A: Verilog Construct of FSM -Moore Model

Skip !!

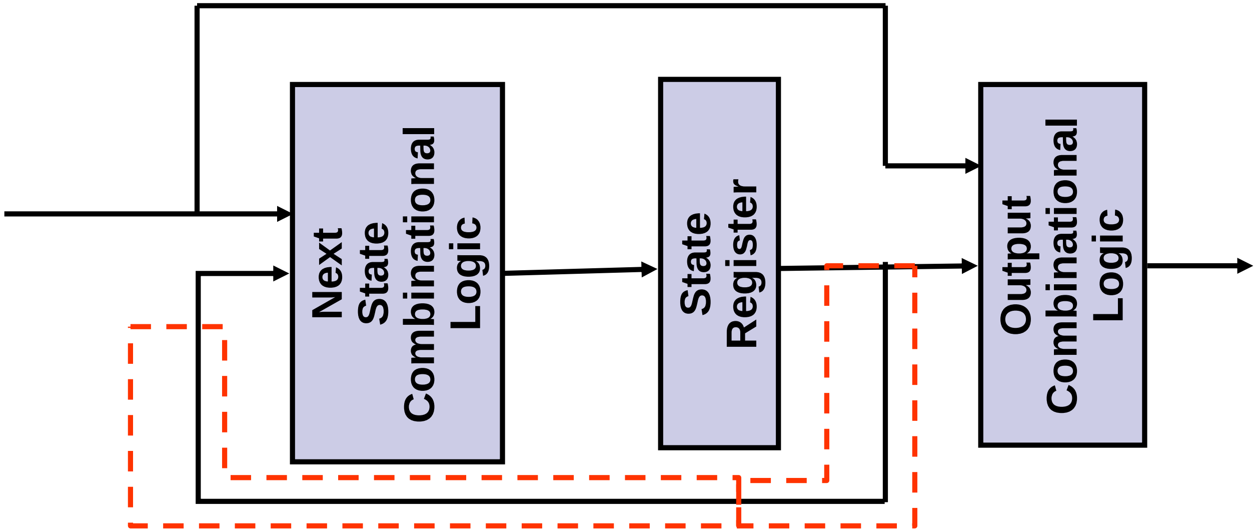


```
module FSM1 ( A, CLK, reset, y, Z );
    input A, CLK, reset;
    output Z;
    output [1:0] y;
    reg [1:0] y,  Y ;

  //  state encoding:
  parameter [1:0]   S0 = 2'b00,  S1 = 2'b01,  S2 = 2'b10, S3 = 2'b11;

   //  NS logic submodule:
   always @ (A or y)
           case  ( y )
          S0  :  if ( A ) Y = S1;  else Y = S0;
          S1  :  if ( A ) Y = S3;      else Y = S2;
          S2  :  if ( A ) Y = S3;  else Y = S2;
          S3  :  if ( A ) Y = S0;  else Y = S3;
           endcase
    //  state register submodule:
    always @ ( negedge reset or posedge CLK )
           if (reset == 0)  y <= S0 ;
           else  y <= Y ;

    // define the Output Logic submodule:
    assign  Z = ( y == S0 ) | (y == S3) ;

      endmodule
```

JASM Using Mealy Model

Skip !!

- Any system can be defined as Moore machine or Mealy machine but the number of states and the transitions vary.
- The specifications (still remember?):
  - An idle system is activated when an input, **A** is given. Then, an output, **B** is produced after one interval time or cycle later. Next, the system will be back to the idle state, waiting for the next triggering input **A**.
- Step 1: Understand the specs.
  - Been there, done that!
  - Another view of Mealy Model. Notice: output = f(input,present state)



JASM Mealy State Transition Diagram

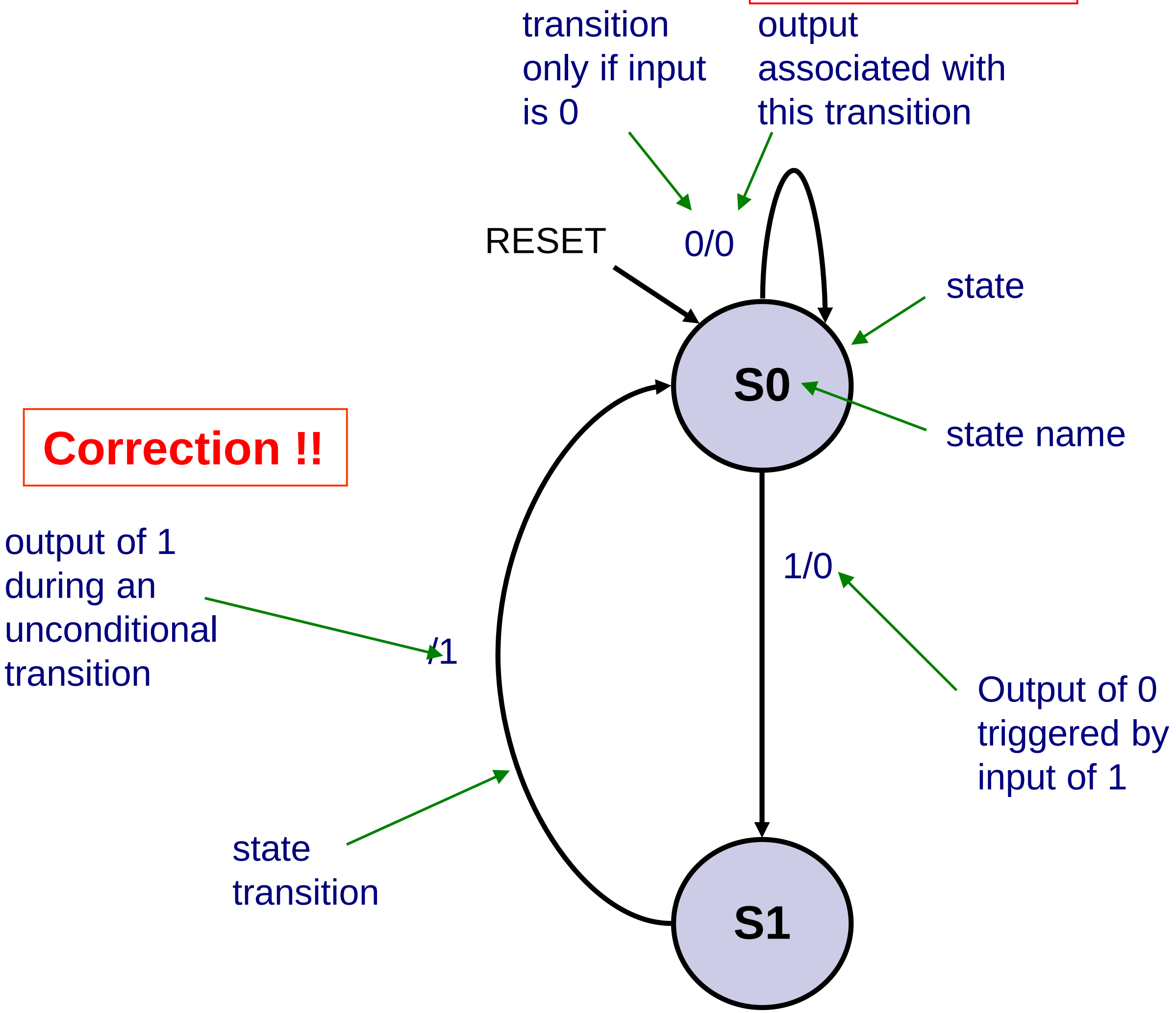
Skip !!

Step 2: Draw State diagram

Mealy state diagram is slightly different than Moore’s. Next state logic is defined by Input ~~outputs~~ too.

i.e. Outputs are associated with state transitions **(arcs)**

Correction !!


transition only if input is 0
output associated with this transition
RESET
0/0
state
S0
state name
output of 1 during an unconditional transition
/1
1/0
Output of 0 triggered by input of 1
state transition
S1

JASM Symbolic State Table

Skip !!

- Step 3: get symbolic state table.

Present State	Input	Next State	Output
	IN		OUT
S0	0	S0	0
	1	S1	0
S1	0	S0	1
	1	S0	1

Present State	Input	Next State	Output
PS	IN	NS	OUT
0	0	0	0
	1	1	
1	0	0	1
	1	0	

Step 5: Get encoded state table.

Get Logic Equations

Skip !!

- Step 6: Solve the next state & output equations.
- Step 7: Obtain circuit.

Present State	Input	Next State	Output
PS	IN	NS	OUT
0	0	0	0
	1	1	
1	0	0	1
	1	0	

IN \ PS	0	1
0	0	0
1	1	0

$NS = PS' \bullet IN$

IN \ PS	0	1
0	0	1
1	0	1

$OUT = PS$

- Exercise:

  Do step 7 – obtain circuit using TFFs.

Example 2: Bit Sequence Detector (BSD)

- The specification:
  - An input is used to detect a sequence or a series of inputs, 110. When the specific sequence is detected, an output high is produced for a cycle. Then, the system will continue detecting the next sequence inputs.

- Step 1: Understand the specs.
  - Get a sample input/output relationship.
    - Sample input/output relationship:

      IN : 1100011011110…

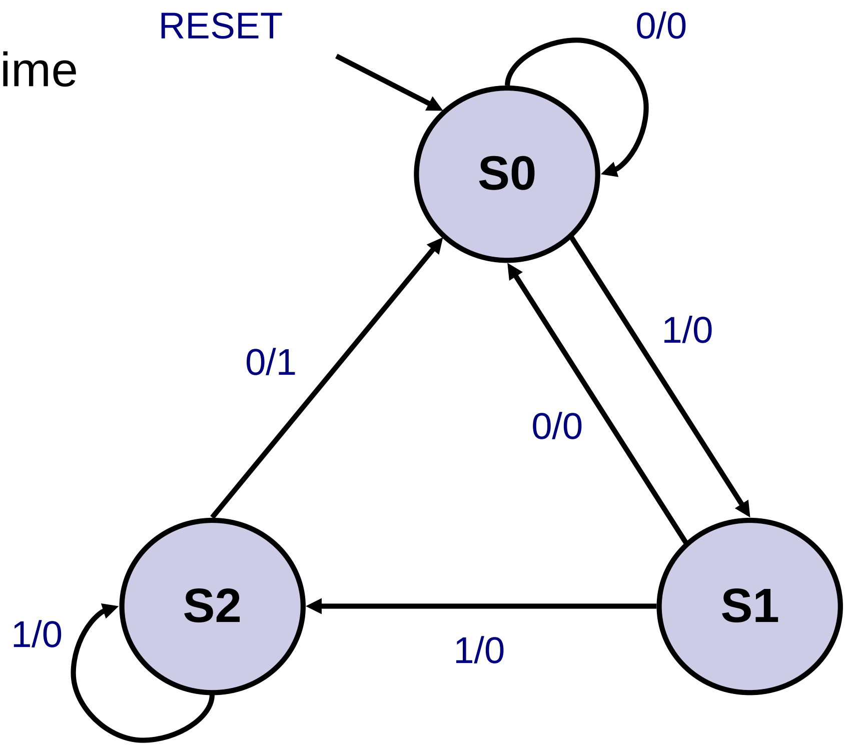
      OUT : 0010000100001…

110 BSD State Diagram

- Step 2: Get state diagram
  - □ Start with the expected sequence first
  - □ S0 means 0 bit found, S1 = 1 bit found, S2 = 2 bits found
  - □ If all three bits are detected (110 sequence completed) while in S2, reset (go to S0) while at the same time outputting a 1

IN : 1100011011110…

OUT : 0010000100001…



BSD Symbolic State Transition Table

Step 3: Symbolic state table

Present States	Input	Next States	Output	Comments
	IN		OUT	
S0	0	S0	0	Remain in idle state if start sequence is not detected
	1	S1	0	Go to next state if first bit is detected
S1	0	S0	0	Return to starting state if wrong sequence
	1	S2	0	Go to next state if second bit is detected
S2	0	S0	1	Complete sequence is detected, reset & output 1
	1	S2	0	Sequence is not completed yet, wait until ‘0’ appears

BSD Encoded State Table

- Step 5: Perform state assignment:
  - Use “simple” binary encoding:
    - S0 = 00
    - S1 = 01
    - S2 = 10

Present State		Input	Next State		Output
PS_1	PS_0	IN	NS_1	NS_0	OUT
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	1
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

BSD Next State & Output Equations

Step 6:

Present State		Input	Next State		Output
PS_1	PS_0	IN	NS_1	NS_0	OUT
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	1
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

- Simpler logic compared to Moore version!
- Exercise: Do step 7 – obtain circuit using JKFFs.

PS_1PS_0 IN	00	01	11	10
0	0	0	X	0
1	0	1	X	1

$$NS_1 = PS_1 \bullet IN + PS_0 \bullet IN$$

PS_1PS_0 IN	00	01	11	10
0	0	0	X	0
1	1	0	X	0

$$NS_0 = PS_1' \bullet PS_0' \bullet IN$$

PS_1PS_0 IN	00	01	11	10
0	0	0	X	1
1	0	0	X	0

$$OUT = PS_1 \bullet IN'$$