Digital Logic Design

4190.201

2014 Spring Semester

8. Finite State Machines

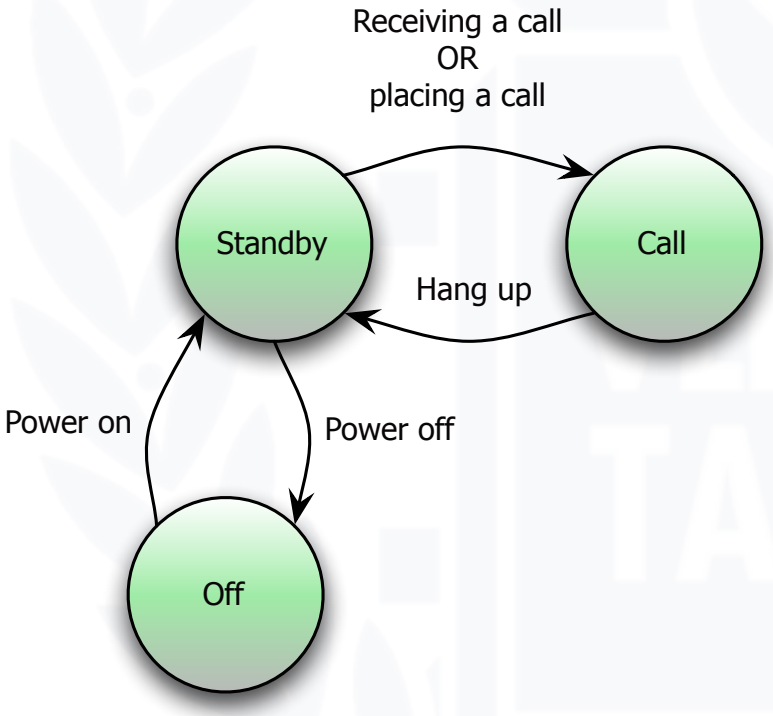
Naehyuck Chang
Dept. of CSE
Seoul National University
naehyuck@snu.ac.kr

Finite state machines

- Sequential circuits include
  - Primitive sequential elements (latches and FFs)
  - Combinational logic
- Models for representing sequential circuits
  - Finite-state machines (Moore and Mealy)
- Basic sequential circuits revisited
  - Shift registers
  - Counters
- Design procedure
  - State diagrams
  - State transition table
  - Next state functions
- Hardware description languages

What is state machine?

- A state is a set of values measured at different parts of the circuit
  - RF amplifier is on/off, LCD is on/off, loudspeaker is on/off, etc.
- A state machine is a digital device that traverses through a predetermined sequence of states in an orderly fashion
- A synchronous state machine distinguishes state by the clock.

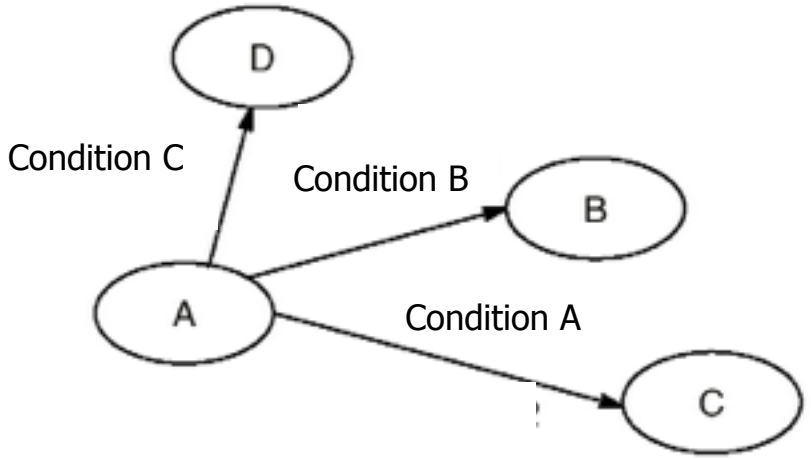


Finite state machines

- Asynchronous state machine
  - State transition occurs when an event is occur
  - Concept is natural
  - Hard and expensive to implement
  - Only one event occurs at a time
- Synchronous state machine
  - State transition occurs when a clock transition occurs
  - Concept is artificial
  - Easy and efficient to implement
  - More than two event may occur during a clock period
    - Appropriate priority is given
- Finite state machine
  - Number of states is finite

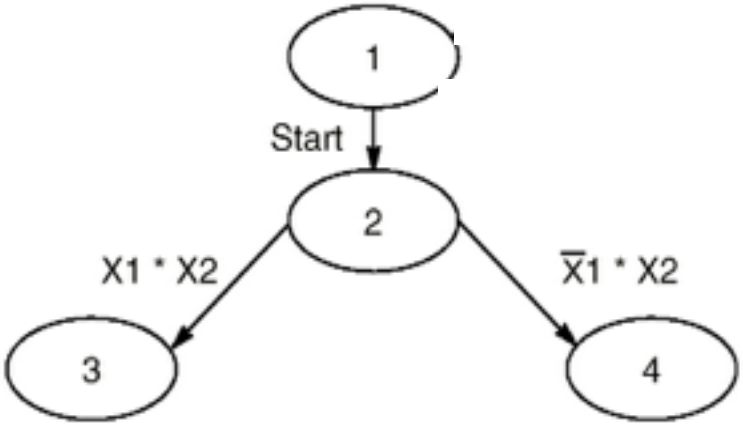
Finite state machines

- Branch condition
  - No overlapped condition
  - No ambiguity



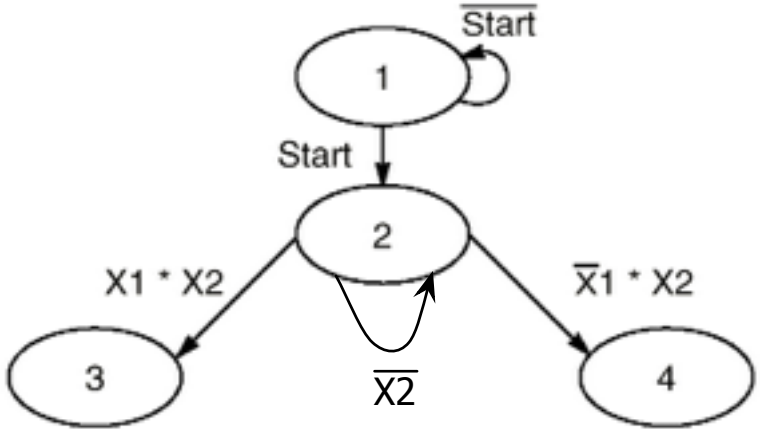
Finite state machines

- Asynchronous state diagram
  - State machine remains forever in State 1 unless Start becomes active.



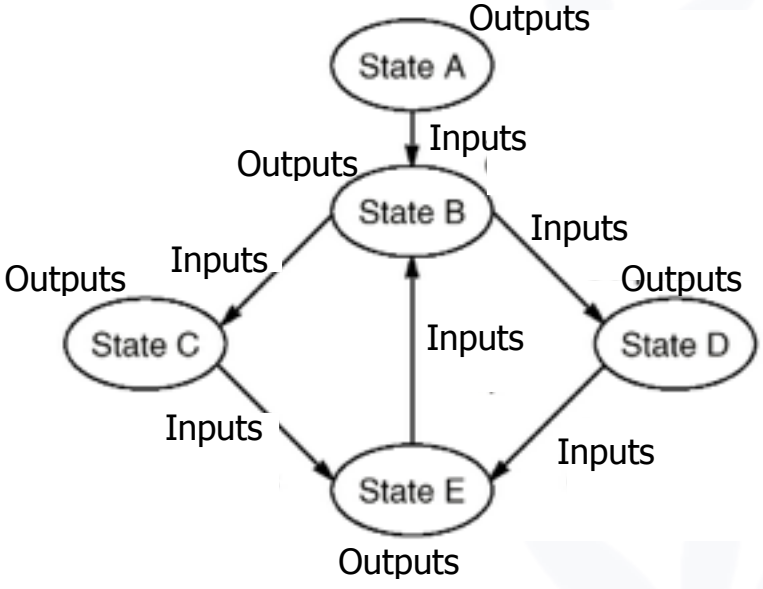
Finite state machines

- Synchronous state machine
  - State transition has to be made in every clock cycle
  - The sum of branch conditions must to be 1
- We will deal with synchronous FSMs only

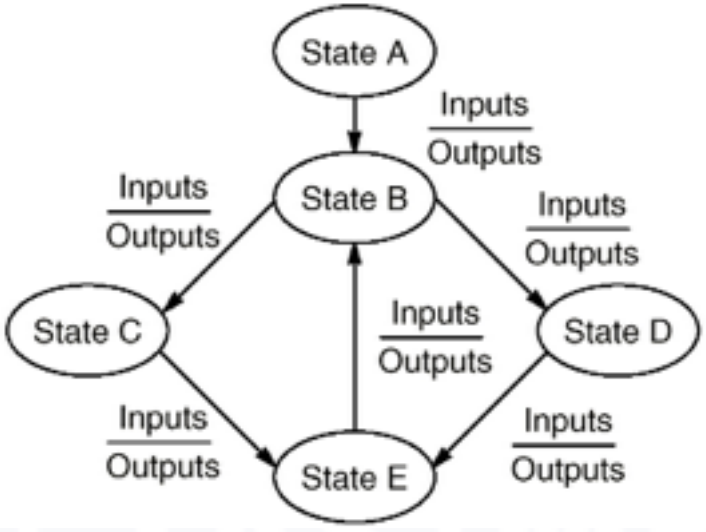


Finite state machines

- Mealy model
  - Output is a function of the state and the input
- Moore model output
  - Output is a function of the state only



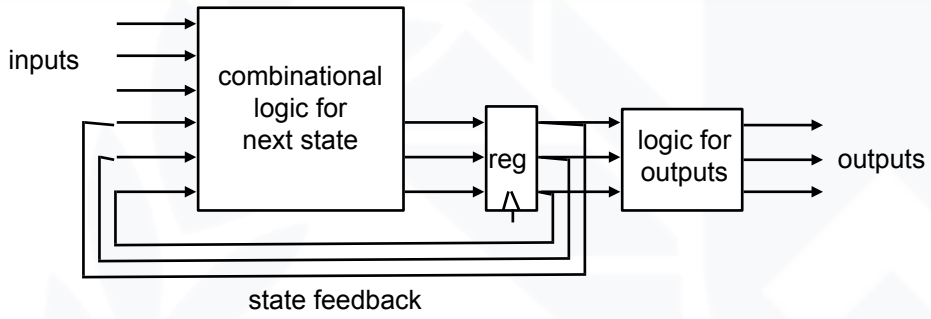
Moore model



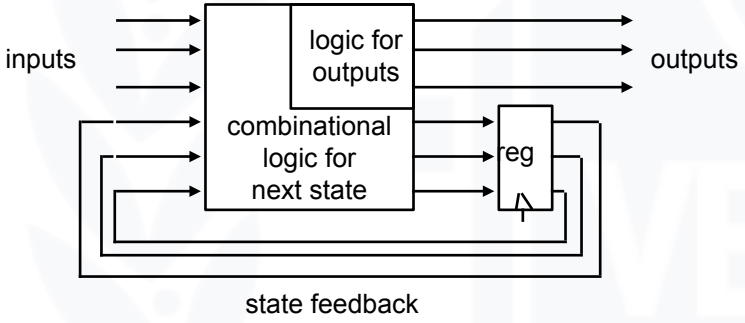
Mealy model

Comparison of Mealy and Moore machines

- Moore

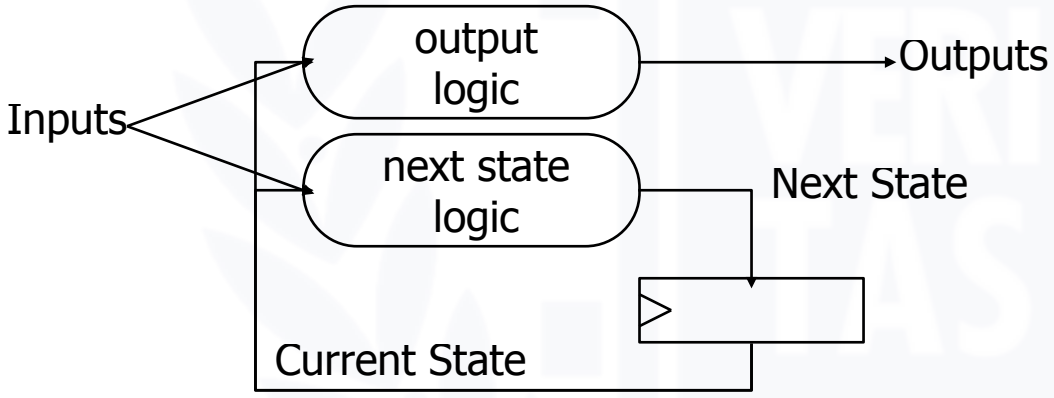


- Mealy



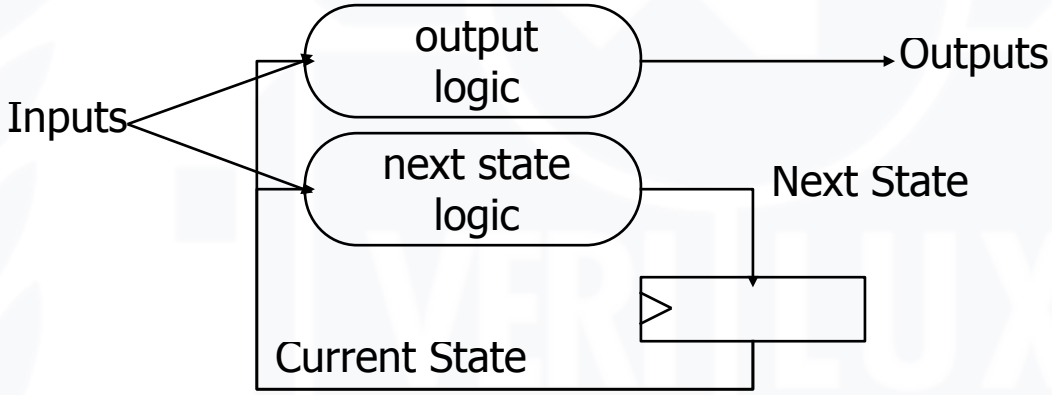
General state machine model

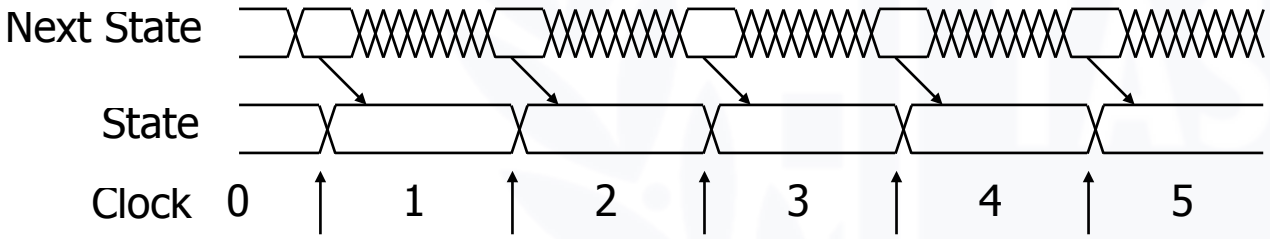
- Values stored in registers represent the state of the circuit
- Combinational logic computes:
  - Next state
    - Function of current state and inputs
  - Outputs
    - Function of current state and inputs (Mealy machine)
    - Function of current state only (Moore machine)



State machine model (cont'd)

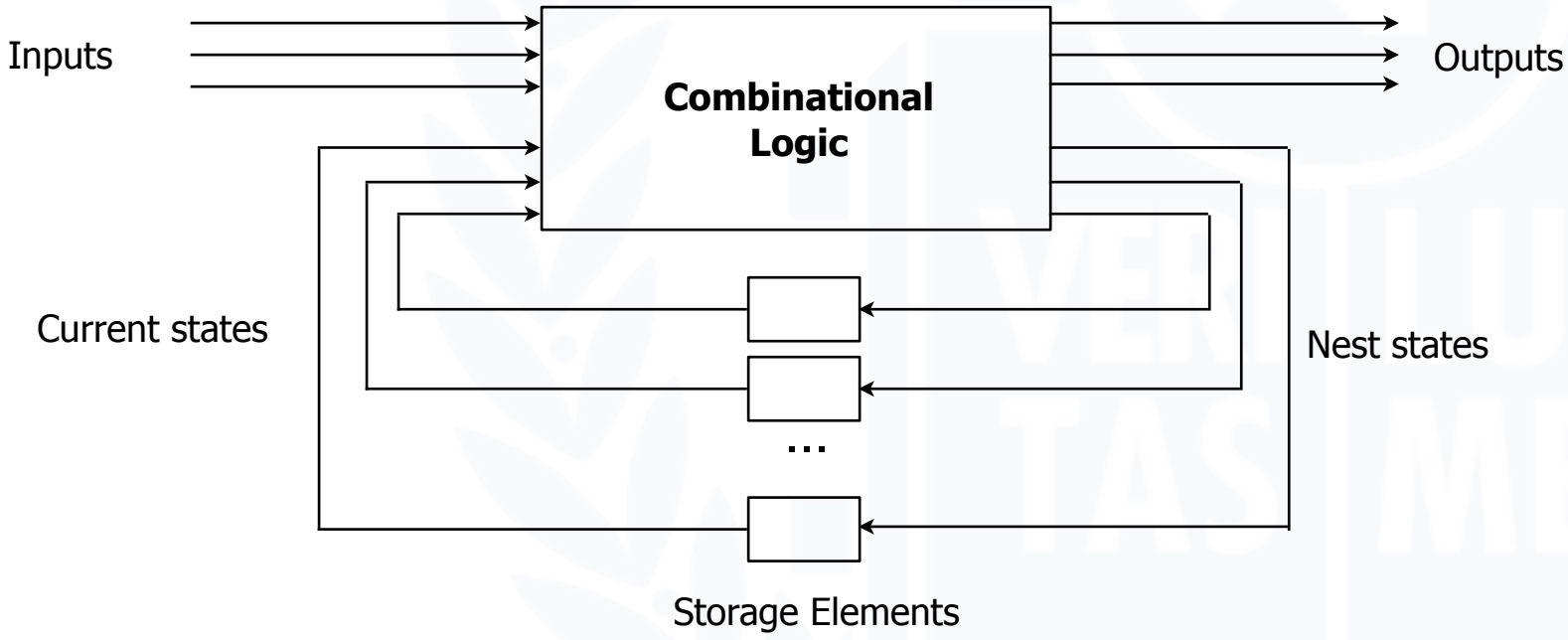
- States: S_1, S_2, ..., S_k
- Inputs: I_1, I_2, ..., I_m
- Outputs: O_1, O_2, ..., O_n
- Transition function: $F_s(S_i, I_j)$
- Output function: $F_o(S_i)$ or $F_o(S_i, I_j)$





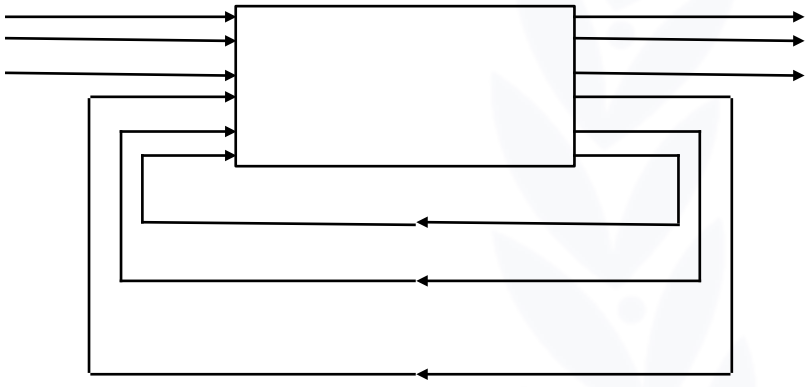
Abstraction of state elements

- Divide circuit into combinational logic and state
- Feedback loops is isolated by FFs
  - Only the current state outputs affect the combination logic

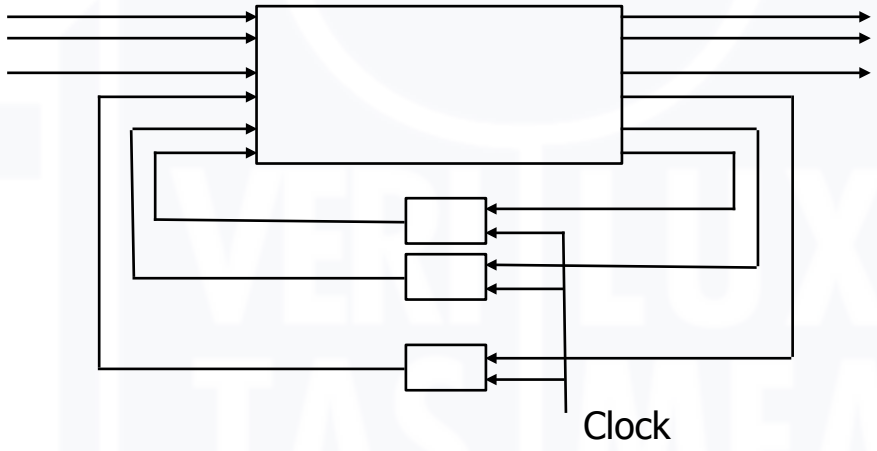


Forms of sequential logic

- Asynchronous sequential logic – state changes occur whenever state inputs change (elements may be simple wires or delay elements)
- Synchronous sequential logic – state changes occur in lock step across all storage elements (using a periodic waveform - the clock)



Asynchronous FSM



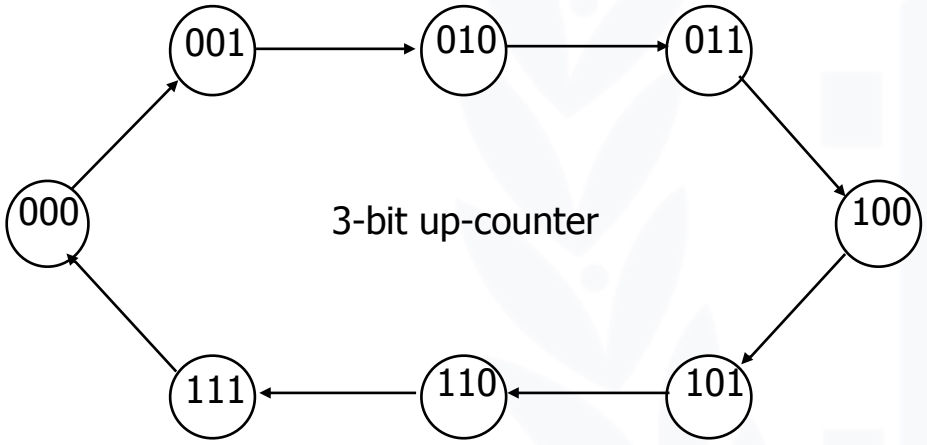
Synchronous FSM

FSM design procedure

- Start with counters
  - Simple because output is just state
  - Simple because no choice of next state based on input
- State diagram to state transition table
  - Tabular form of state diagram
  - Like a truth-table
- State encoding
  - Decide on representation of states
  - For counters it is simple: just its value
- Implementation
  - Flip-flop for each state bit
  - Combinational logic based on encoding

FSM design procedure: state diagram

- State transition table
  - Tabular form of state diagram
  - Like a truth-table (specify output for all input combinations)
  - Encoding of states: easy for counters – just use value



Present state		Next state	
0	000	001	1
1	001	010	2
2	010	011	3
3	011	100	4
4	100	101	5
5	101	110	6
6	110	111	7
7	111	000	0

FSM design procedure: state transition table

- Format of state transition table

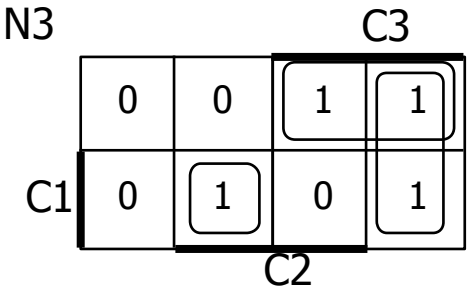
Present State	Inputs	Next State	Outputs Generated
S0 – Sn	I0 – Im	S0 – Sn	O0 – Op

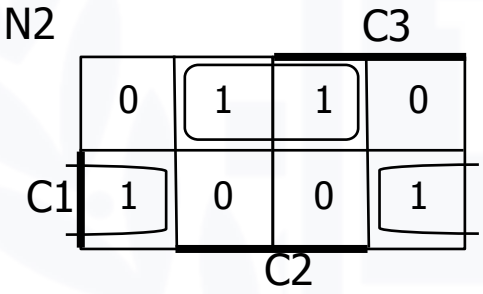
FSM design procedure: next state forming logic

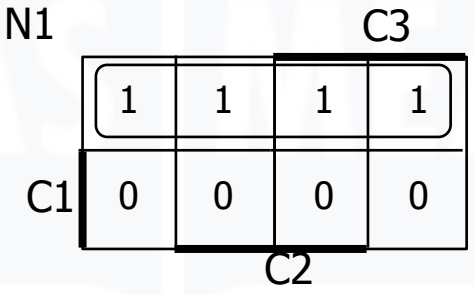
- D flip-flop for each state bit
- Combinational logic based on encoding

C3	C2	C1	N3	N2	N1
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

N1 = C1′
N2 = C1C2′ + C1′C2
= C1 xor C2
N3 = C1C2C3′ + C1′C3 + C2′C3
= (C1C2)C3′ + (C1′ + C2′)C3
= (C1C2)C3′ + (C1C2)′C3
= (C1C2) xor C3

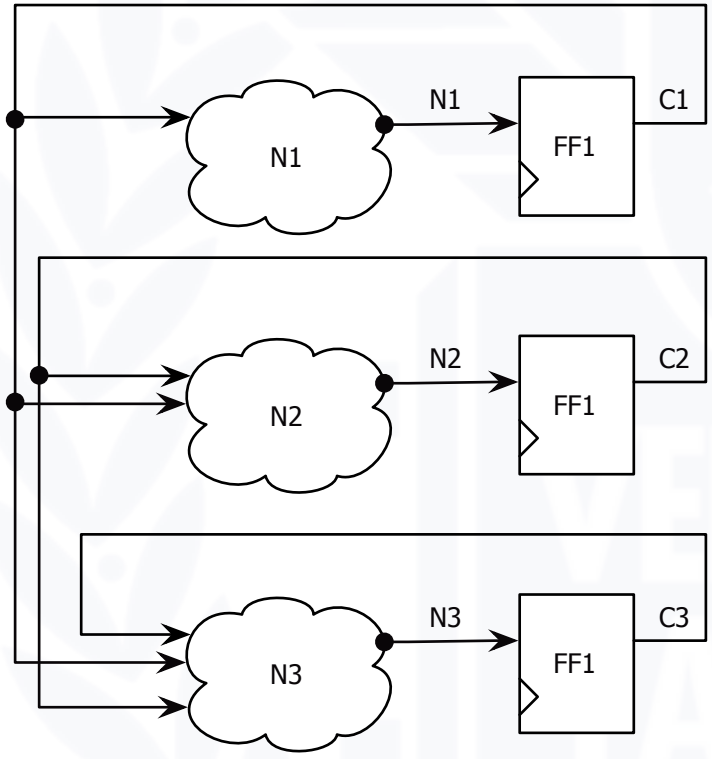






FSM design procedure: implementation

- Implementation



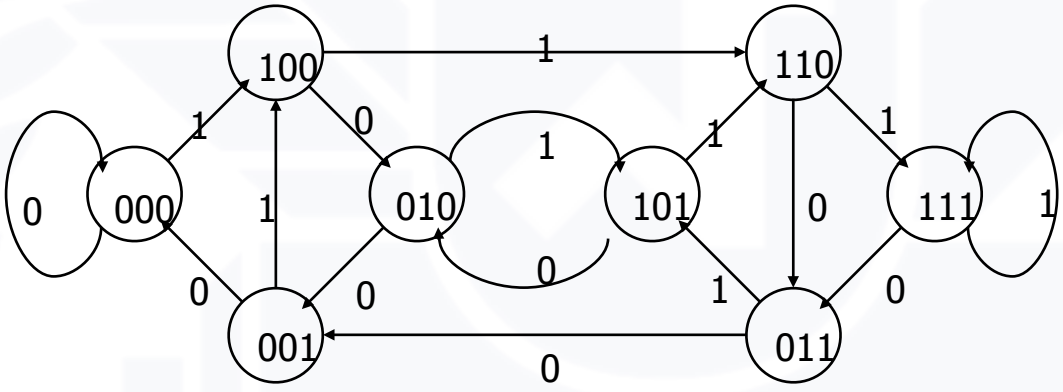
Quiz

- Design a clock-enable DFF with a primitive DFF
  - Draw a state transition diagram
  - Draw a state transition table
  - Derive the next state and output forming logics
  - Draw a schematic diagram

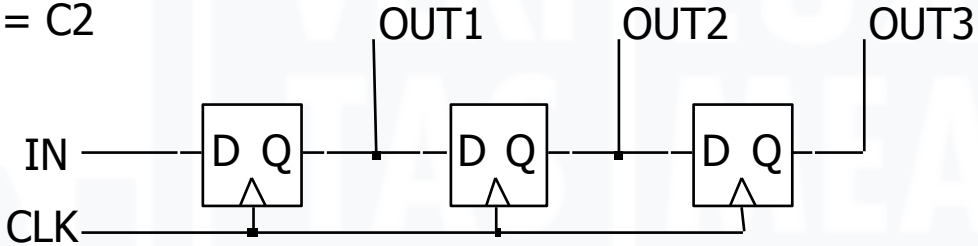
More FSM design examples

- Shift register
  - Input determines next state

In	C1	C2	C3	N1	N2	N3
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1

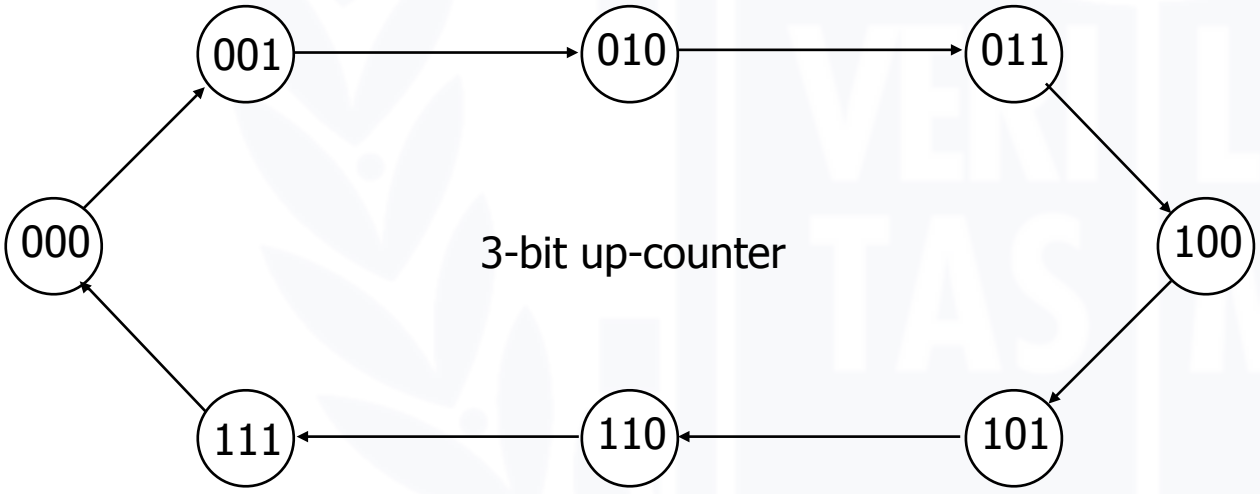


N1 = In
N2 = C1
N3 = C2



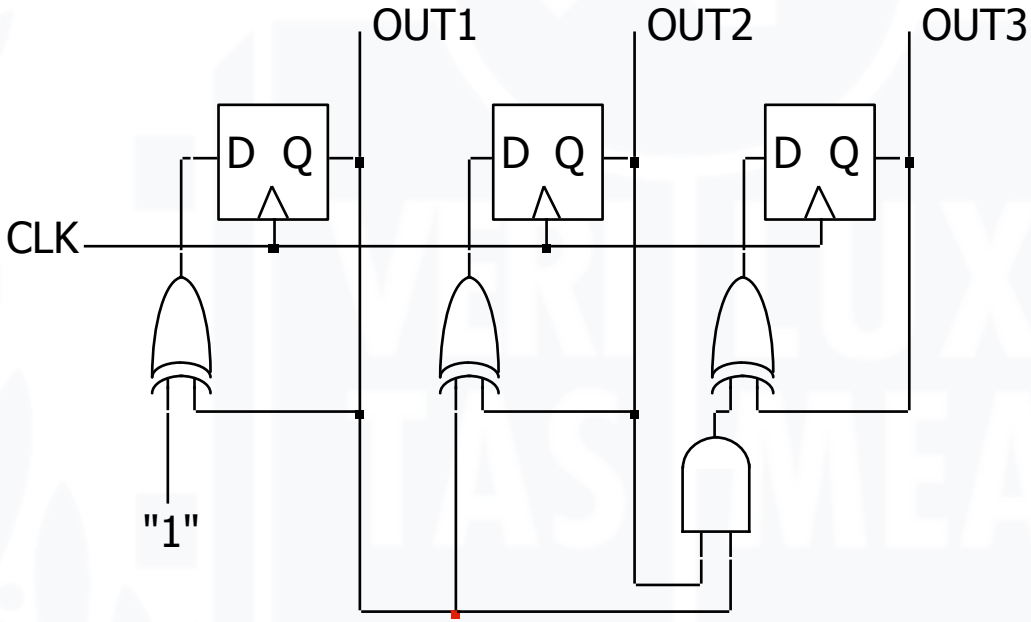
More FSM design examples

- Counters
  - proceed through well-defined sequence of states in response to enable
- Many types of counters: binary, BCD, Gray-code
  - 3-bit up-counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...
  - 3-bit down-counter: 111, 110, 101, 100, 011, 010, 001, 000, 111, ...



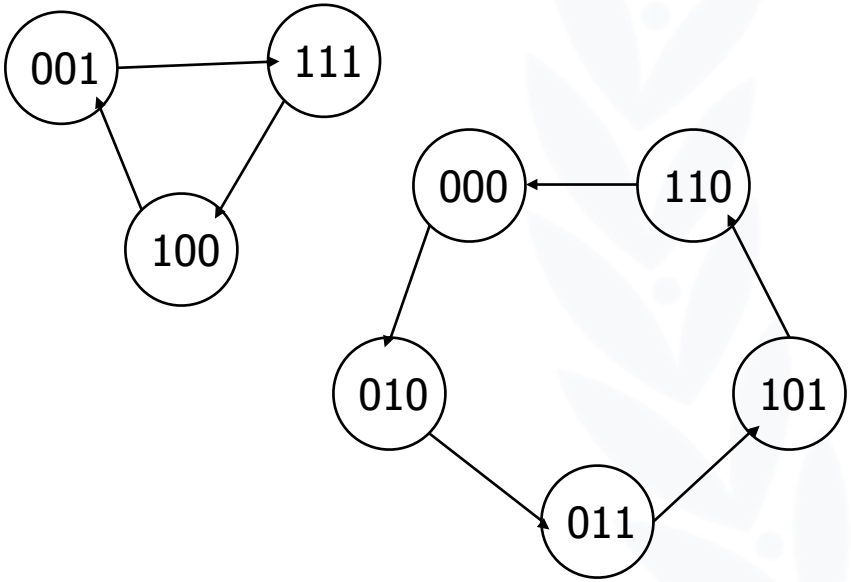
More FSM design examples

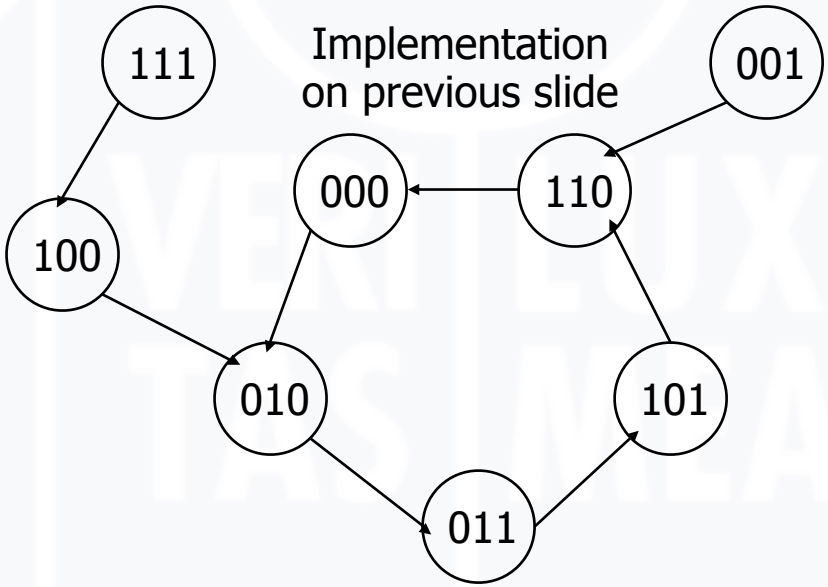
- Counter
  - 3 flip-flops to hold state
  - Logic to compute next state
  - Clock signal controls when flip-flop memory can change
    - Wait long enough for combinational logic to compute new value
    - Don't wait too long as that is low performance



More FSM design examples

- Self-starting counters
- Start-up states
  - At power-up, counter may be in an unused or invalid state
  - Designer must guarantee that it (eventually) enters a valid state
- Self-starting solution
  - Design counter so that invalid states eventually transition to a valid state
  - May limit exploitation of don't cares





More FSM design examples

- Re-deriving state transition table from don't care assignment

C+

A \ C	0	0	0	0
	1	1	1	1

B

B+

A \ C	1	1	0	1
	1	0	0	1

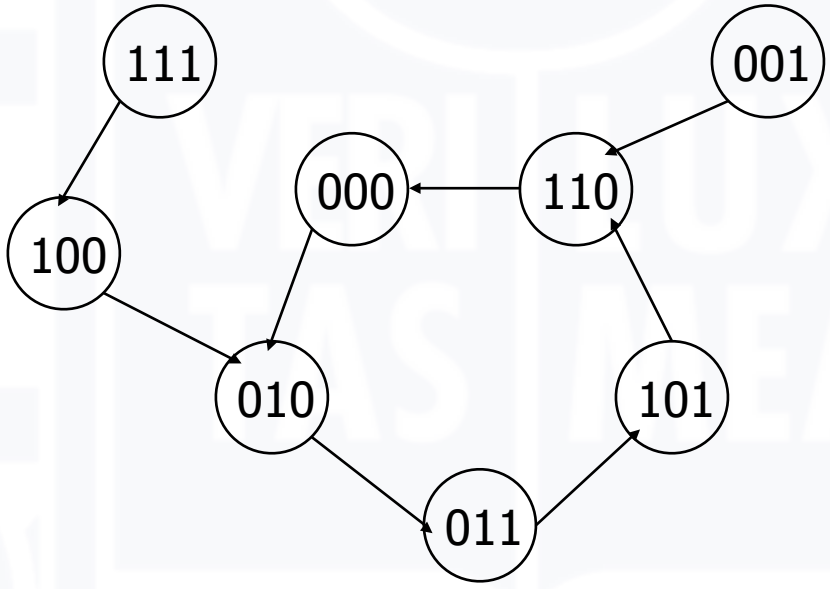
B

A+

A \ C	0	1	0	0
	0	1	0	0

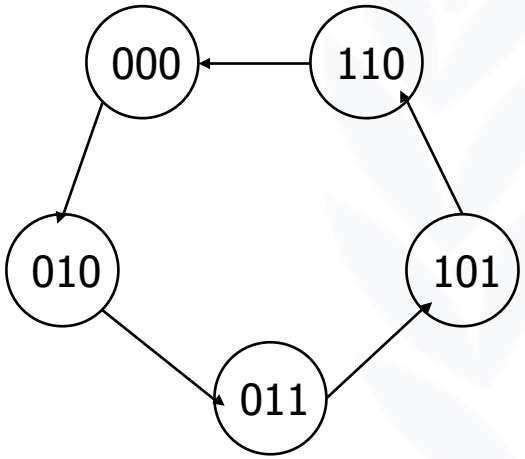
B

Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	1	0	0



More FSM design examples

- Complex counter
  - repeats 5 states in sequence
  - not a binary number representation
- Step 1: derive the state transition diagram
  - count sequence: 000, 010, 011, 101, 110
- Step 2: derive the state transition table from the state transition diagram

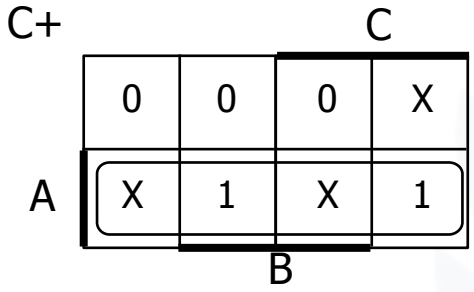


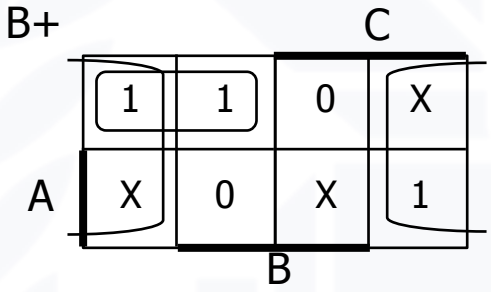
Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	–	–	–
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	–	–	–
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	–	–	–

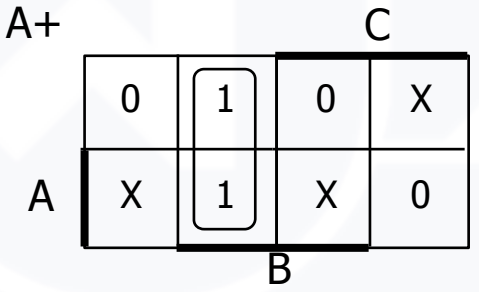
Note the don't care conditions that arise from the unused state codes

More FSM design examples

- Step 3: K-maps for next state functions







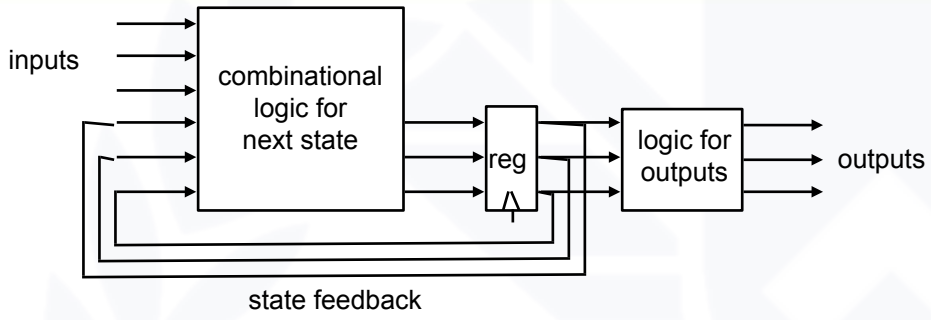
C+ <= A

B+ <= B′ + A′C′

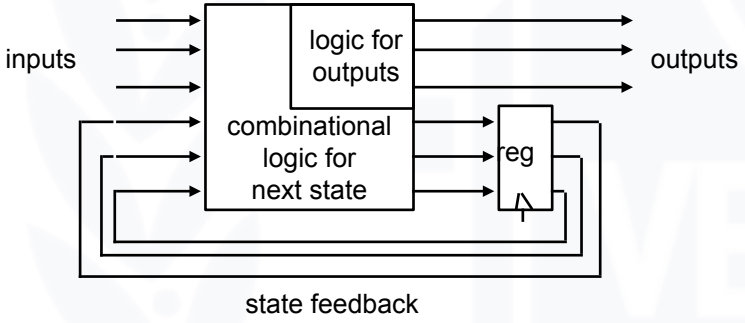
A+ <= BC′

Comparison of Mealy and Moore machines

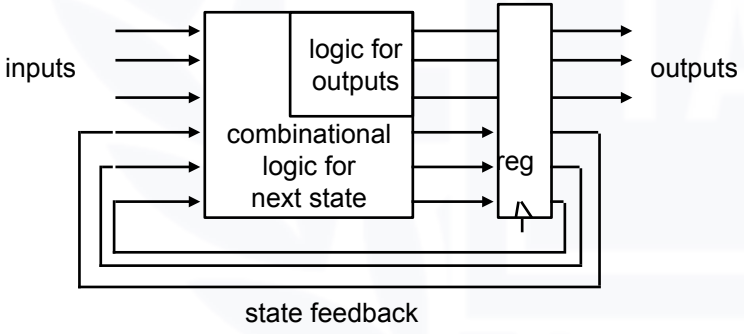
- Moore



- Mealy

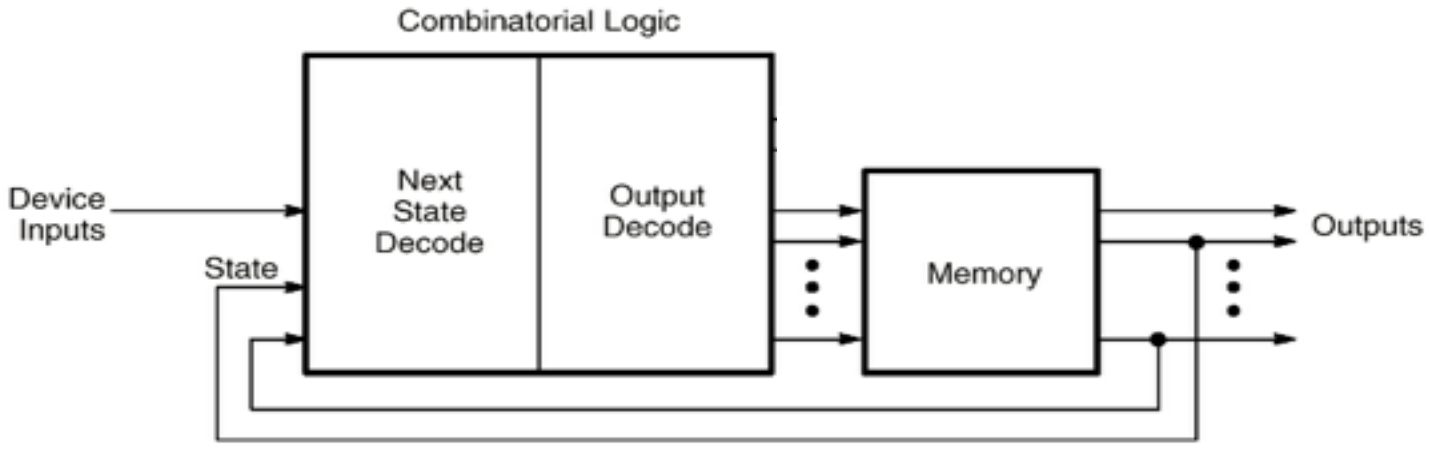


- Synchronous Mealy



Comparison of Mealy and Moore machines

- Basic model
  - Suitable for a single PAL implementation

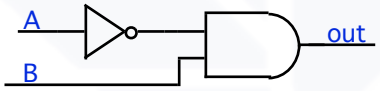


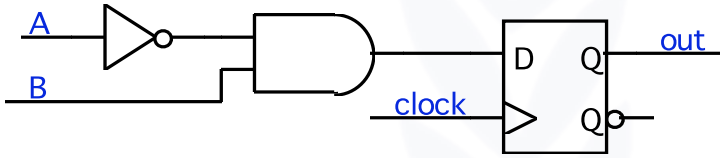
Comparison of Mealy and Moore machines

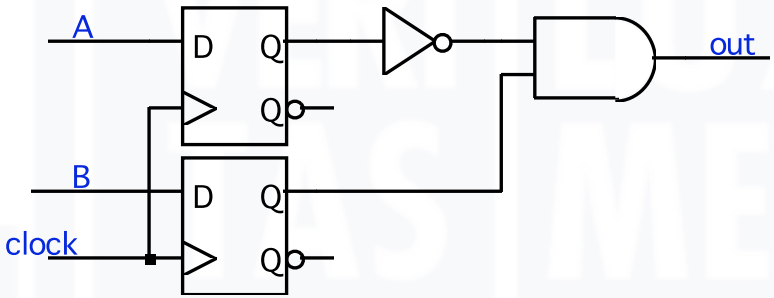
- Mealy machines tend to have less states
  - Different outputs on arcs (n2) rather than states (n)
- Moore machines are safer to use
  - Outputs change at clock edge (always one cycle later)
  - In Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback may occur if one isn't careful
- Mealy machines react faster to inputs
  - React in same cycle – don't need to wait for clock
  - In Moore machines, more logic may be necessary to decode state into outputs – more gate delays after clock edge

Mealy and Moore examples

- Recognize A,B = 0,1
  - Mealy or Moore?

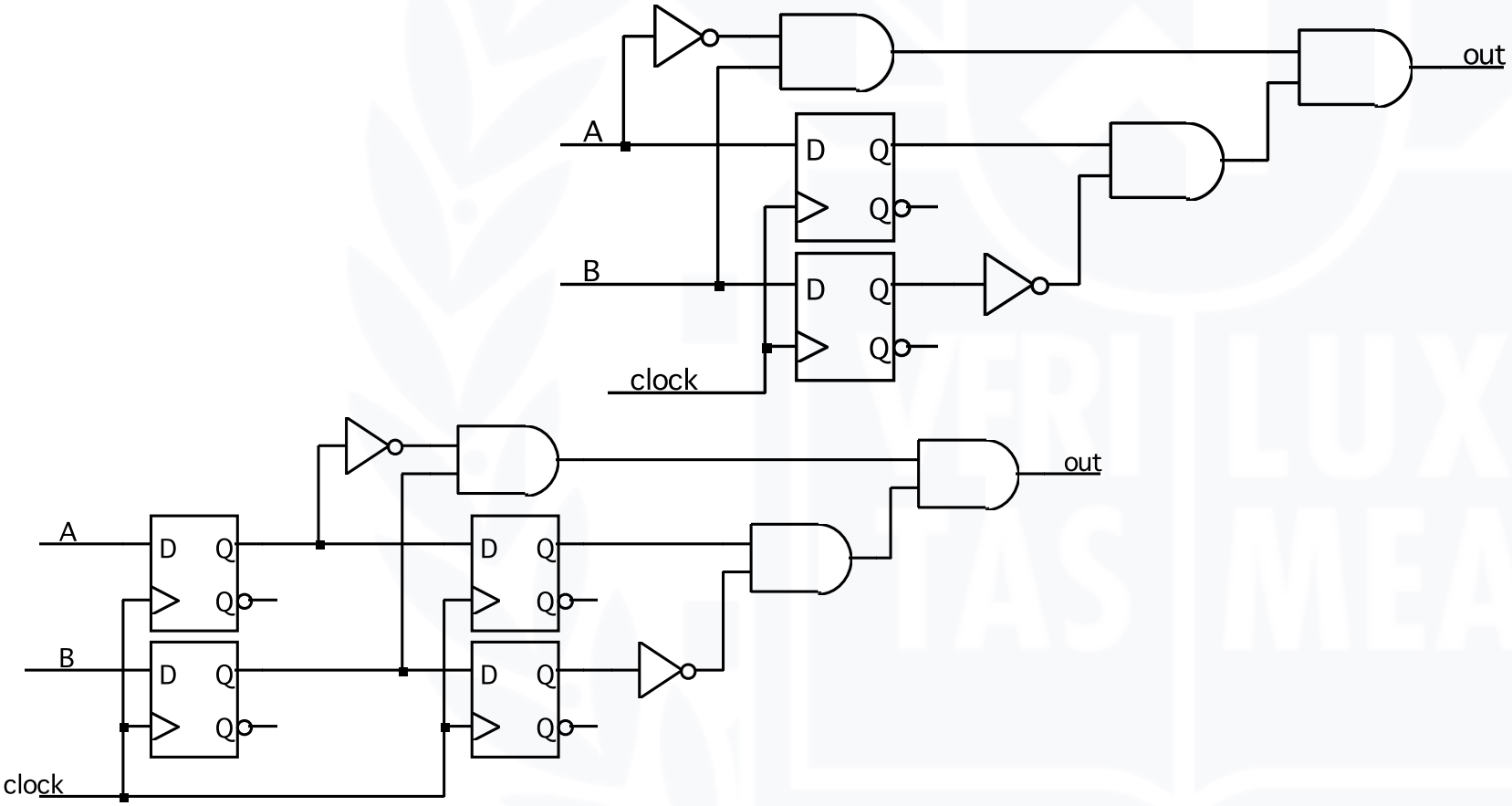






Mealy and Moore examples (cont'd)

- Recognize A,B = 1,0 then 0,1
  - Mealy or Moore?



State assignment

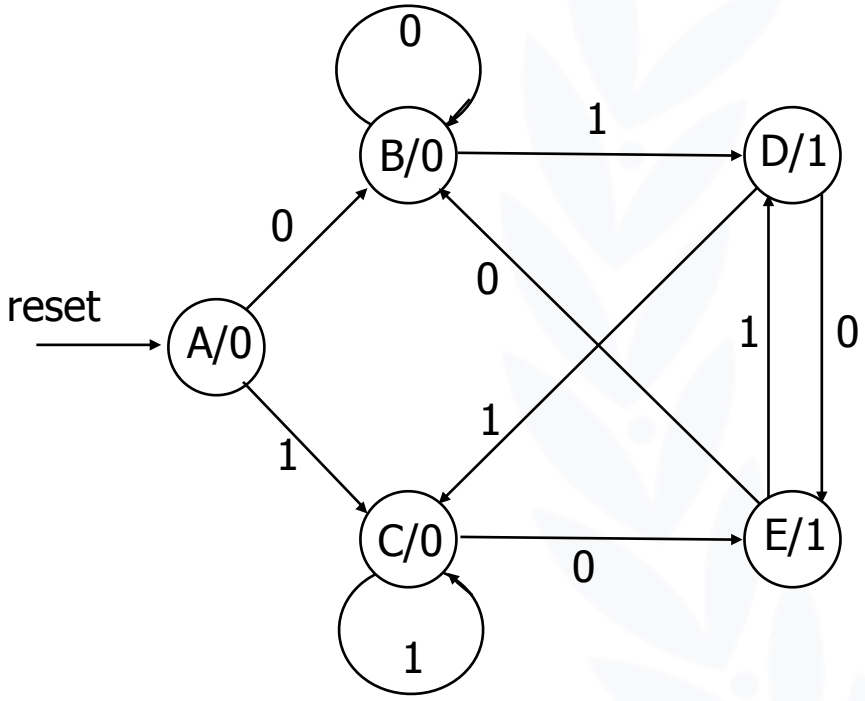
- State encoding
  - Identify each state by a unique name
- Non-redundant encoding
  - Binary encoding
  - Gray code encoding
- Redundant encoding
  - One-hot encoding
  - BCD encoding
  - Basic machine encoding

One-hot encoding

- Use of n-bit code for n states
  - S1: 0000000001
  - S2: 0000000010
  - S3: 0000000100
- Suitable for FPGA implementation
  - Use more FFs
  - Make the combination logic simpler
    - Less number of inputs
    - Even complicated FSM has limited number of previous states

Specifying outputs for a Moore machine

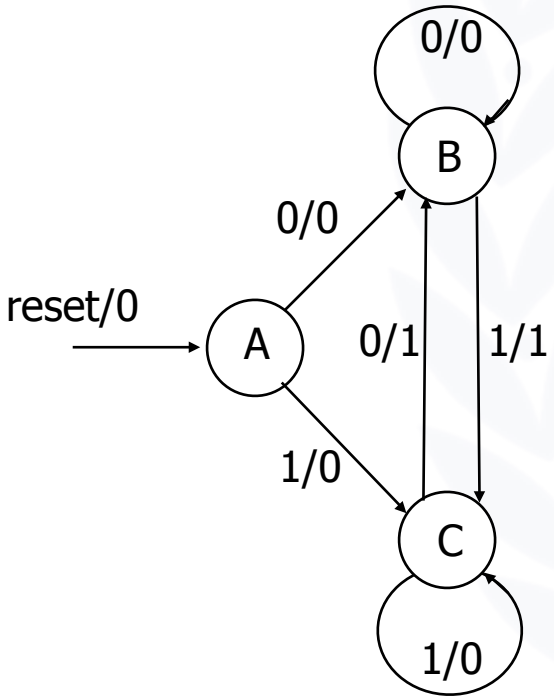
- Output is only function of state
  - Specify in state bubble in state diagram
  - Example: sequence detector for 01 or 10



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

Specifying outputs for a Mealy machine

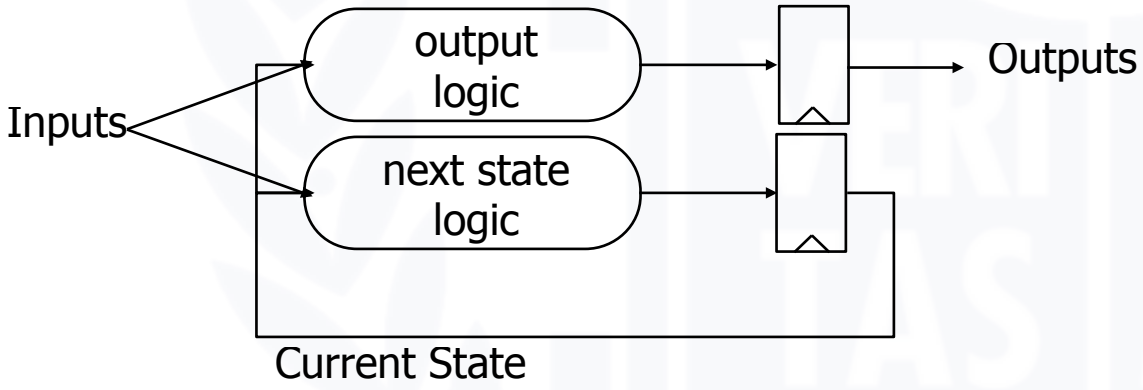
- Output is function of state and inputs
  - Specify output on transition arc between states
  - Example: sequence detector for 01 or 10



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

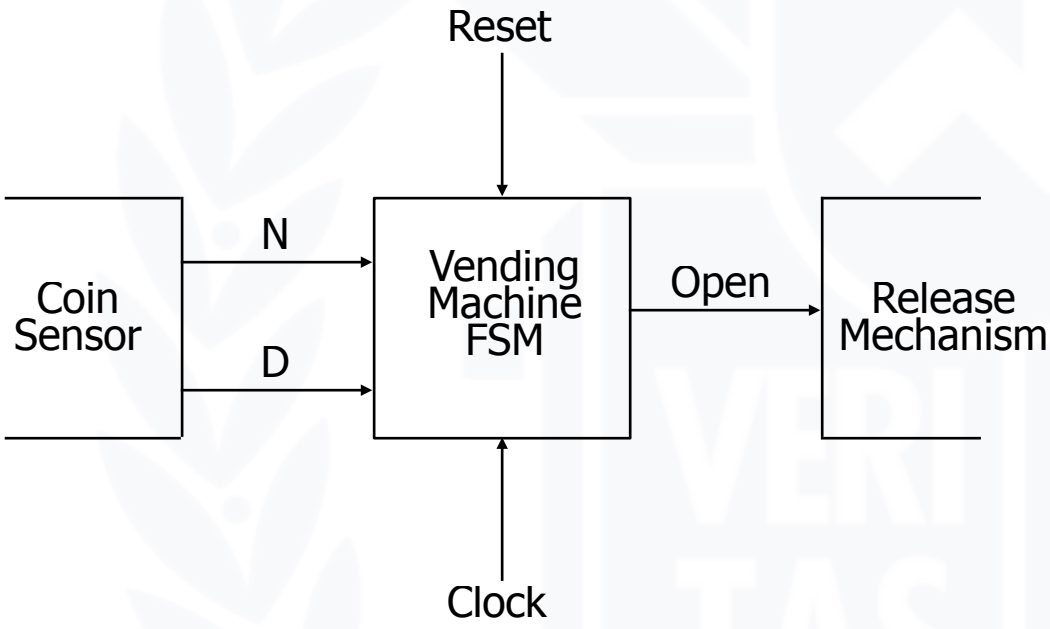
Registered Mealy machine (really Moore)

- Synchronous (or registered) Mealy machine
  - Registered state AND outputs
  - Avoids 'glitchy' outputs
  - Easy to implement in PLDs
- Moore machine with no output decoding
  - Outputs computed on transition to next state rather than after entering
  - View outputs as expanded state vector



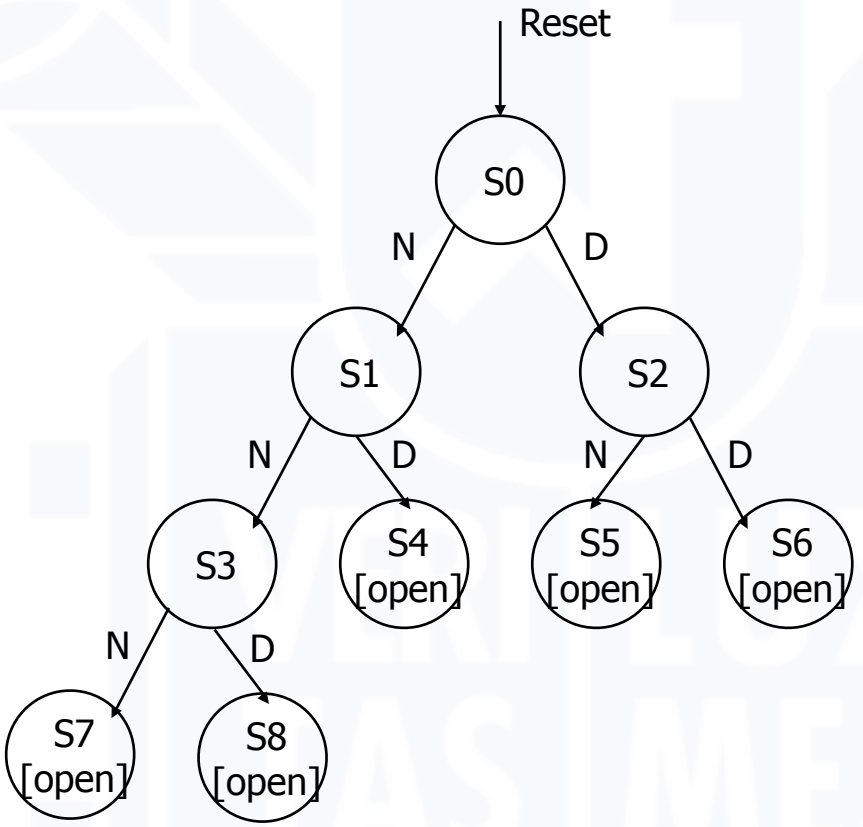
More complex example: vending machine

- Release item after 15 cents are deposited
- Single coin slot for dimes, nickels
- No change



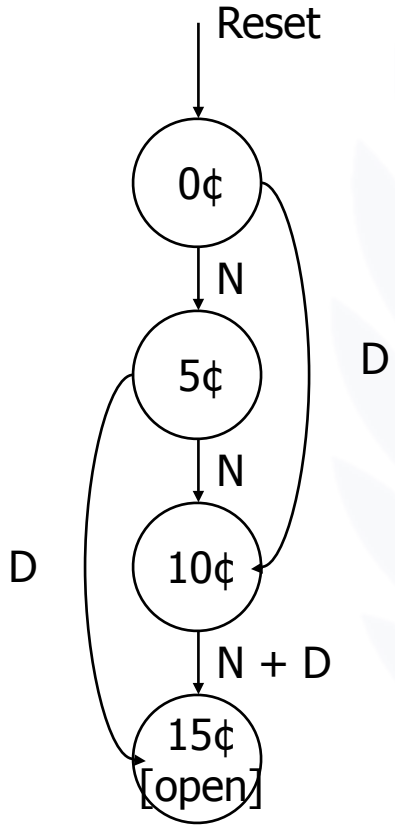
More complex example: vending machine

- Suitable abstract representation
  - Tabulate typical input sequences:
    - 3 nickels
    - Nickel, dime
    - Dime, nickel
    - Two dimes
  - Draw state diagram:
    - Inputs: N, D, reset
    - Output: open chute
  - Assumptions:
    - Assume N and D asserted for one cycle
    - Each state has a self loop for N = D = 0 (no coin)



More complex example: vending machine

- Minimize number of states - reuse states whenever possible



present state	inputs D	N	next state	output open
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	–	–
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	–	–
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	–	–
15¢	–	–	15¢	1

symbolic state table

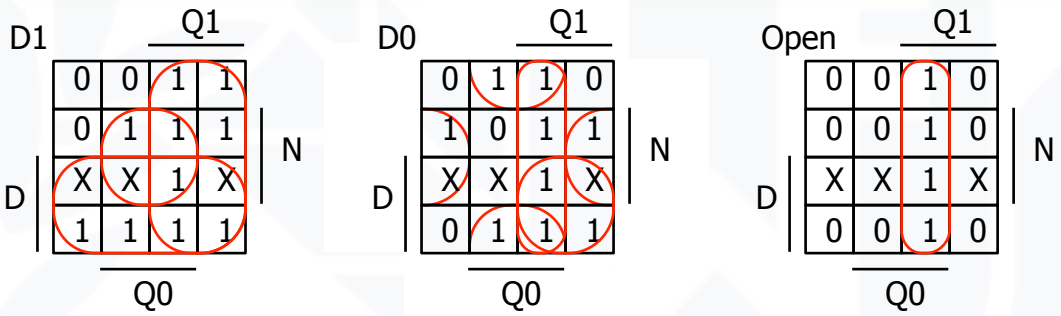
More complex example: vending machine

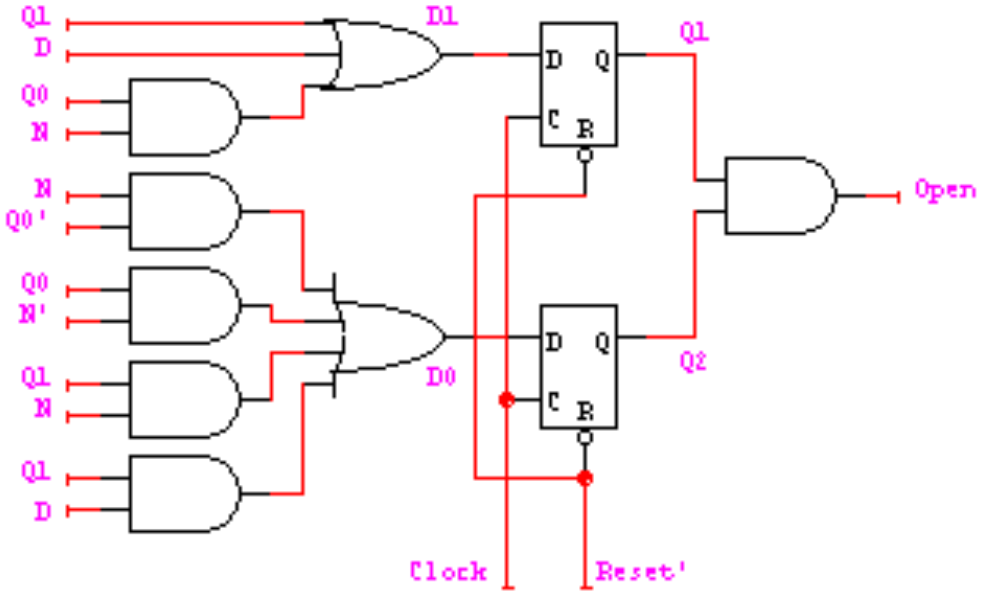
- Uniquely encode states

present state Q1	Q0	inputs D	N	next state D1	D0	output open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	–	–	–
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	–	–	–
1	1	–	–	1	1	1

More complex example: vending machine

- Mapping to logic





D1 = Q1 + D + Q0 N

D0 = Q0′ N + Q0 N′ + Q1 N + Q1 D

OPEN = Q1 Q0

More complex example: vending machine

- One-hot encoding

present state Q3	Q2	Q1	Q0	inputs D	N	next state D3	D2	D1	D0	output open
0	0	0	1	0	0	0	0	0	1	0
				0	1	0	0	1	0	0
				1	0	0	1	0	0	0
				1	1	-	-	-	-	-
0	0	1	0	0	0	0	0	1	0	0
				0	1	0	1	0	0	0
				1	0	1	0	0	0	0
				1	1	-	-	-	-	-
0	1	0	0	0	0	0	1	0	0	0
				0	1	1	0	0	0	0
				1	0	1	0	0	0	0
				1	1	-	-	-	-	-
1	0	0	0	-	-	1	0	0	0	1

D0 = Q0 D' N'

D1 = Q0 N + Q1 D' N'

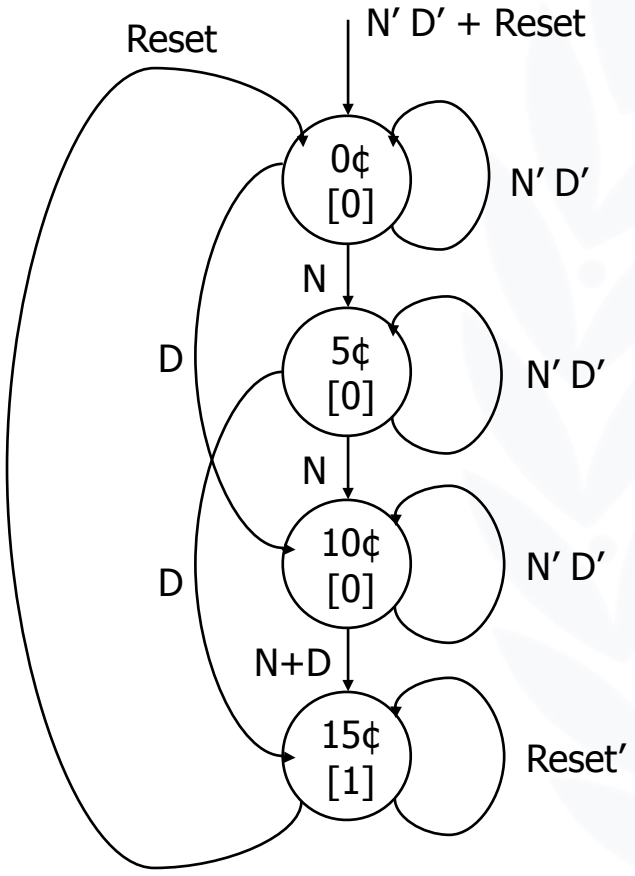
D2 = Q0 D + Q1 N + Q2 D' N'

D3 = Q1 D + Q2 D + Q2 N + Q3

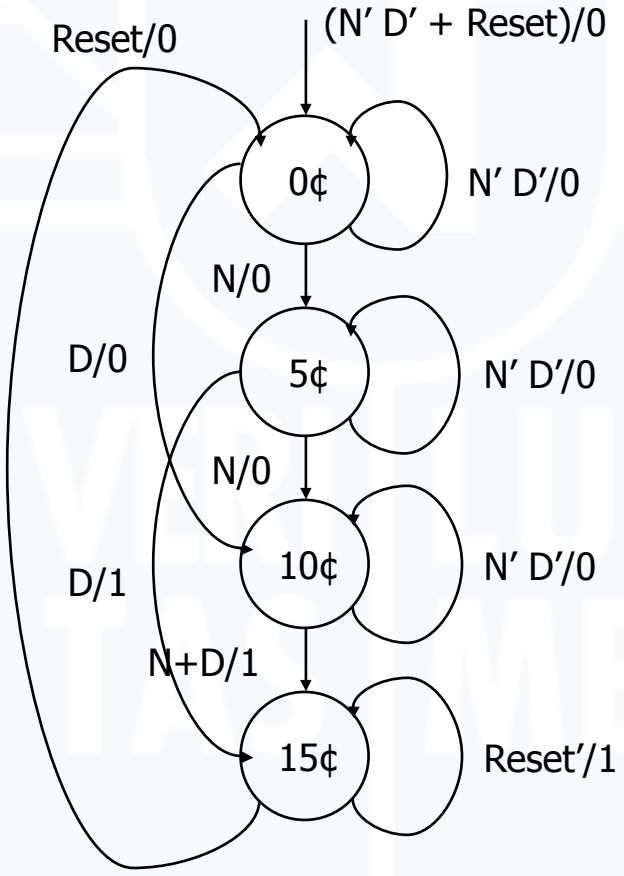
OPEN = Q3

More complex example: vending machine

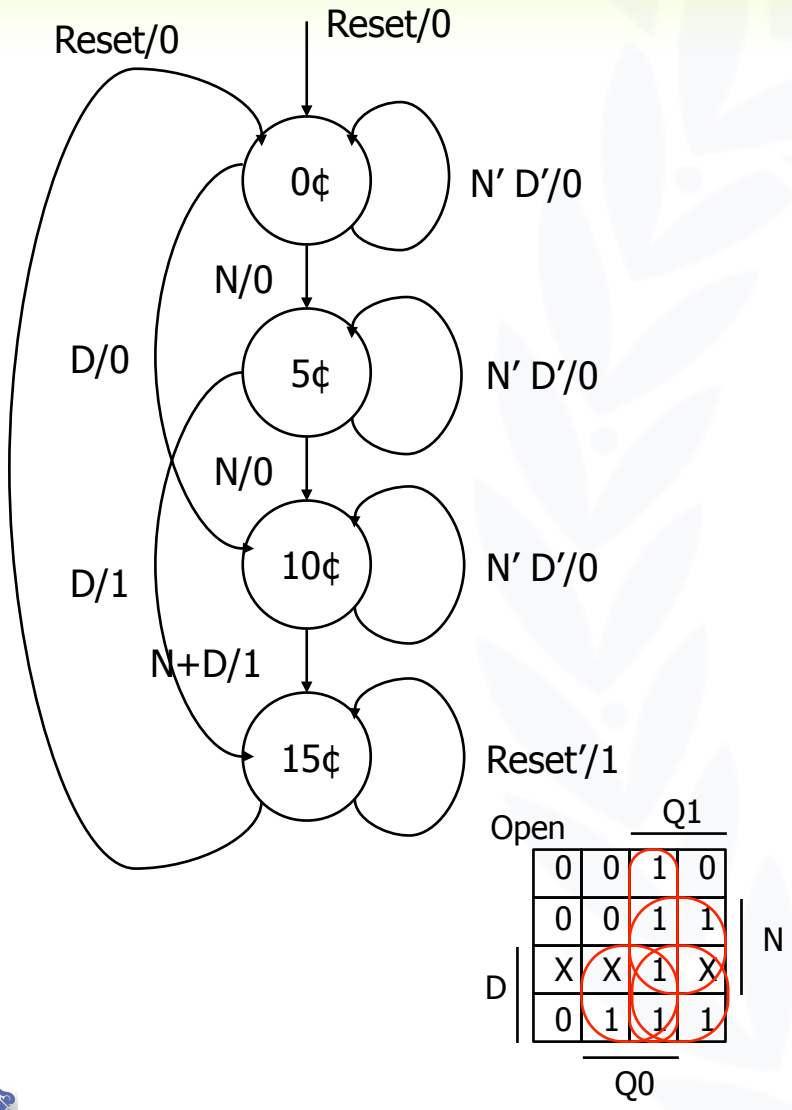
- Moore machine
  - Outputs associated with state



- Mealy machine
  - Outputs associated with transitions



Example: Mealy implementation



present state Q1	Q0	inputs D	N	next state D1	D0	output open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	–	–	–
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	1
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	1
		1	0	1	1	1
		1	1	–	–	–
1	1	–	–	1	1	1

D0 = Q0′N + Q0N′ + Q1N + Q1D

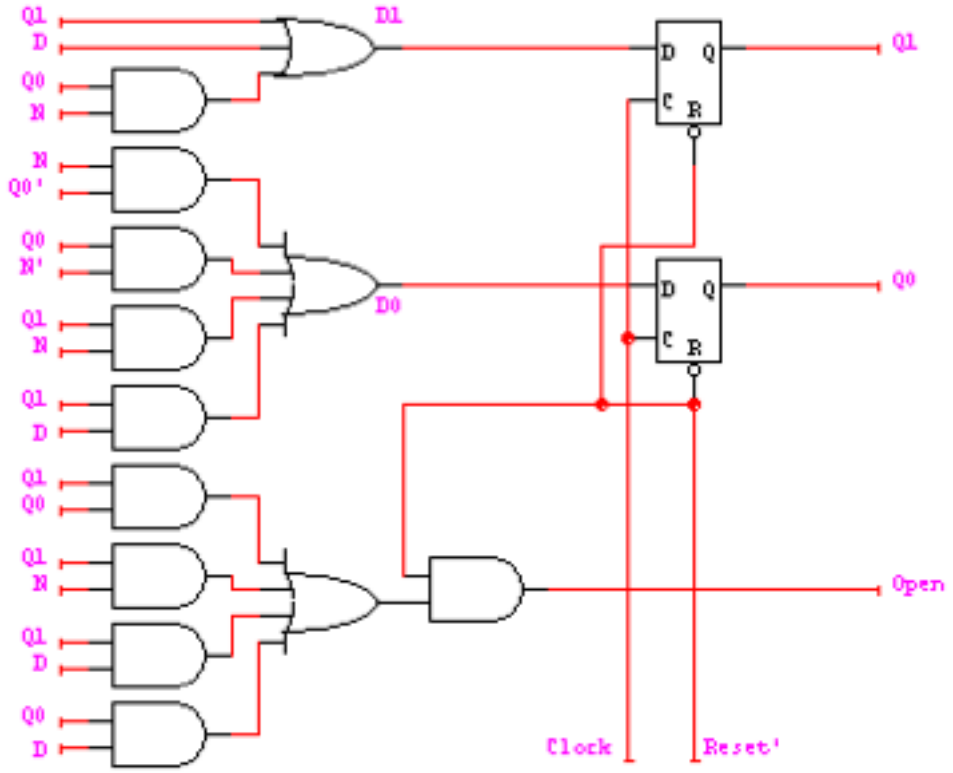
D1 = Q1 + D + Q0N

OPEN = Q1Q0 + Q1N + Q1D + Q0D

Example: Mealy implementation

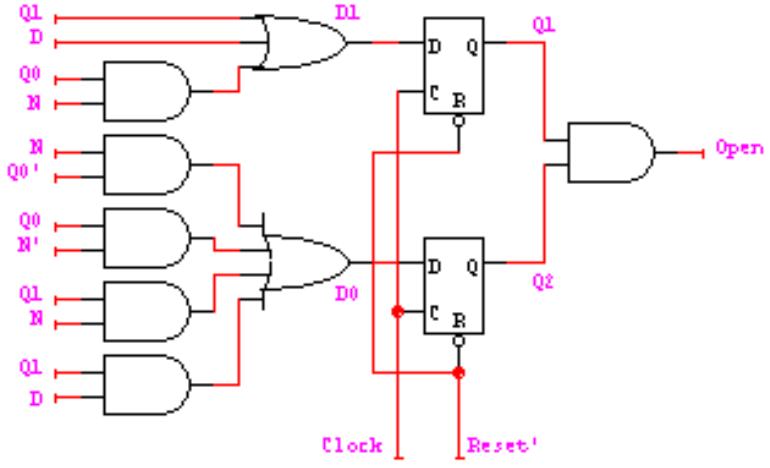
D0 = Q0'N + Q0N' + Q1N + Q1D
D1 = Q1 + D + Q0N
OPEN = Q1Q0 + Q1N + Q1D + Q0D

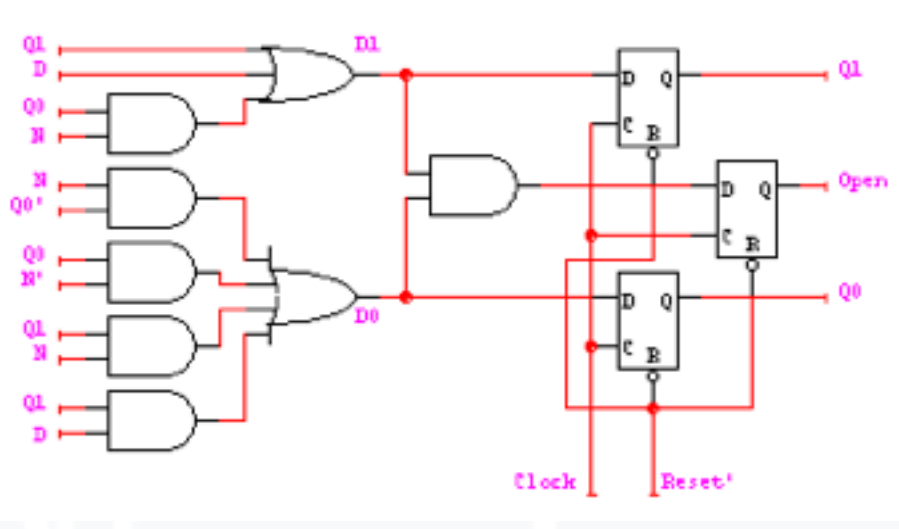
Make sure OPEN is 0 when reset
– by adding AND gate



Vending machine: Moore to synch. Mealy

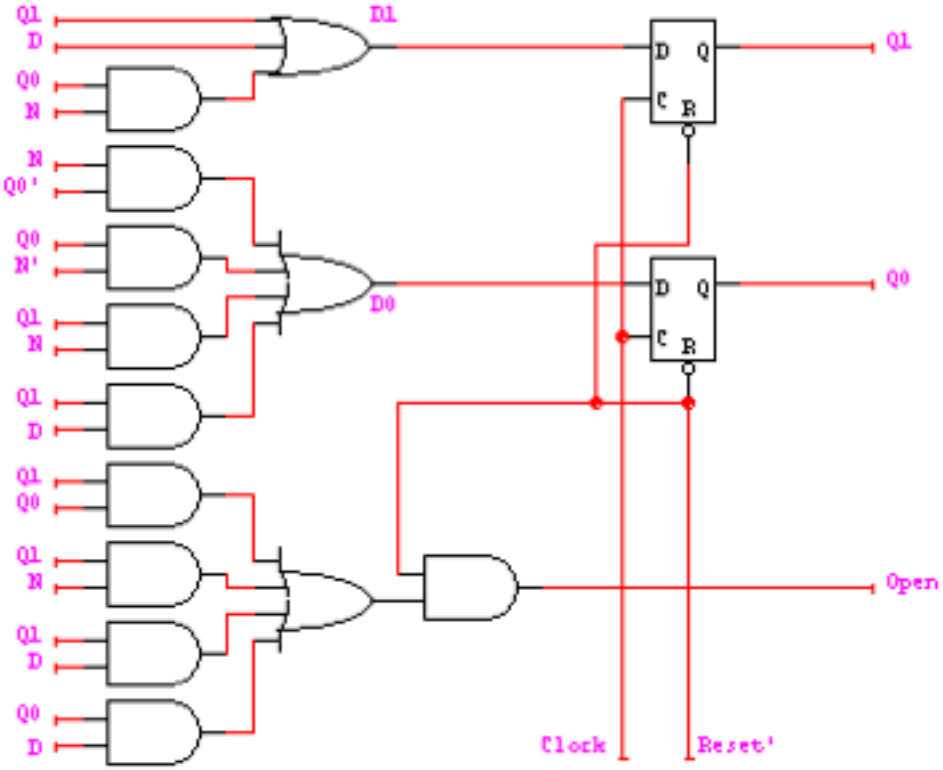
- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 change in Moore implementation
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D
- Implementation now looks like a synchronous Mealy machine
  - It is common for programmable devices to have FF at end of logic





Vending machine: Mealy to synch. Mealy

- OPEN.d = Q1Q0 + Q1N + Q1D + Q0D
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D

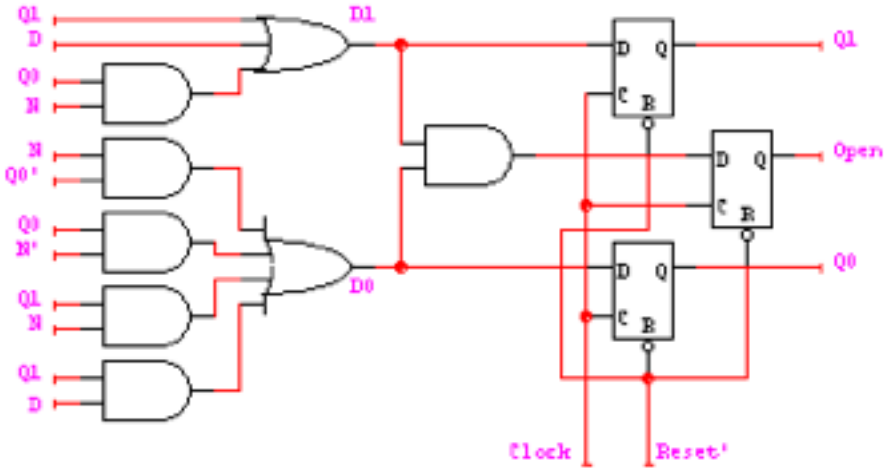


Open.d	Q1			
	0	0	1	0
	0	0	1	1
D	1	0	1	1
	0	1	1	1
		Q0		

N

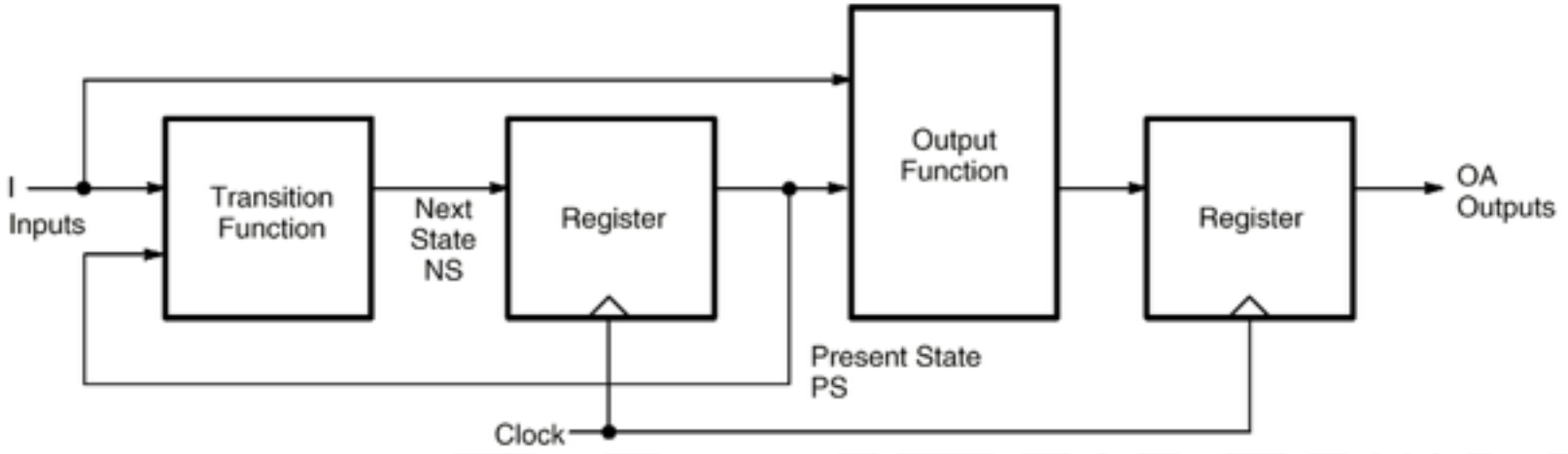
Open.d	Q1			
	0	0	1	0
	0	0	1	1
D	X	X	1	X
	0	1	1	1
		Q0		

N



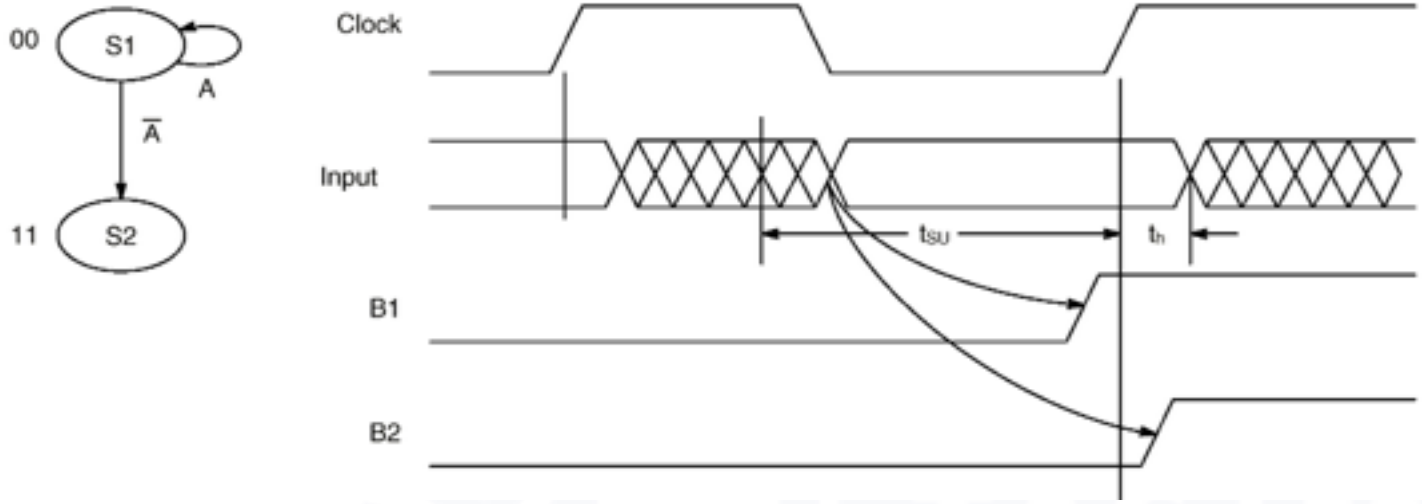
Output synchronization

- Prevent from glitch
- Increase output delay



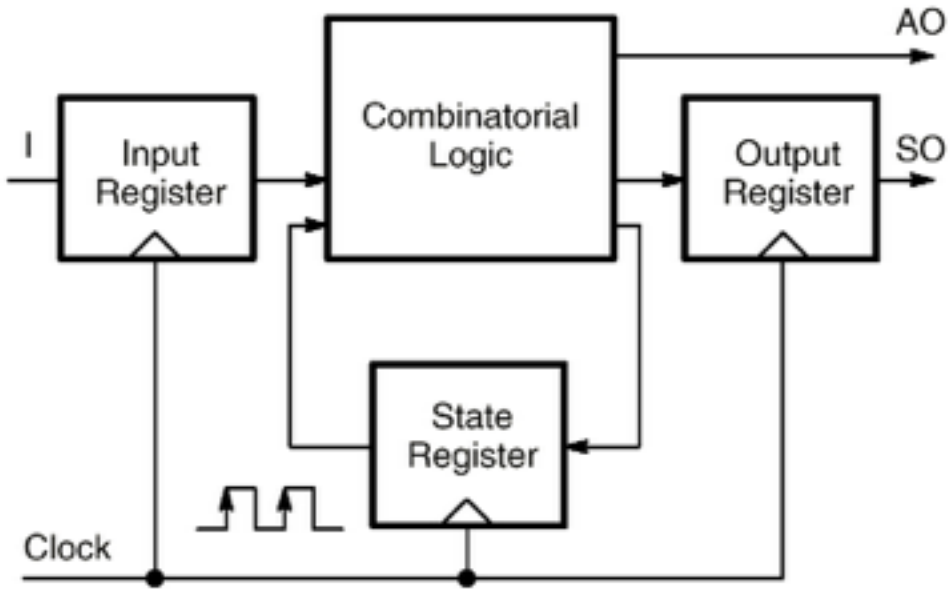
Input synchronization

- Prevent from timing fault
- Increase delay

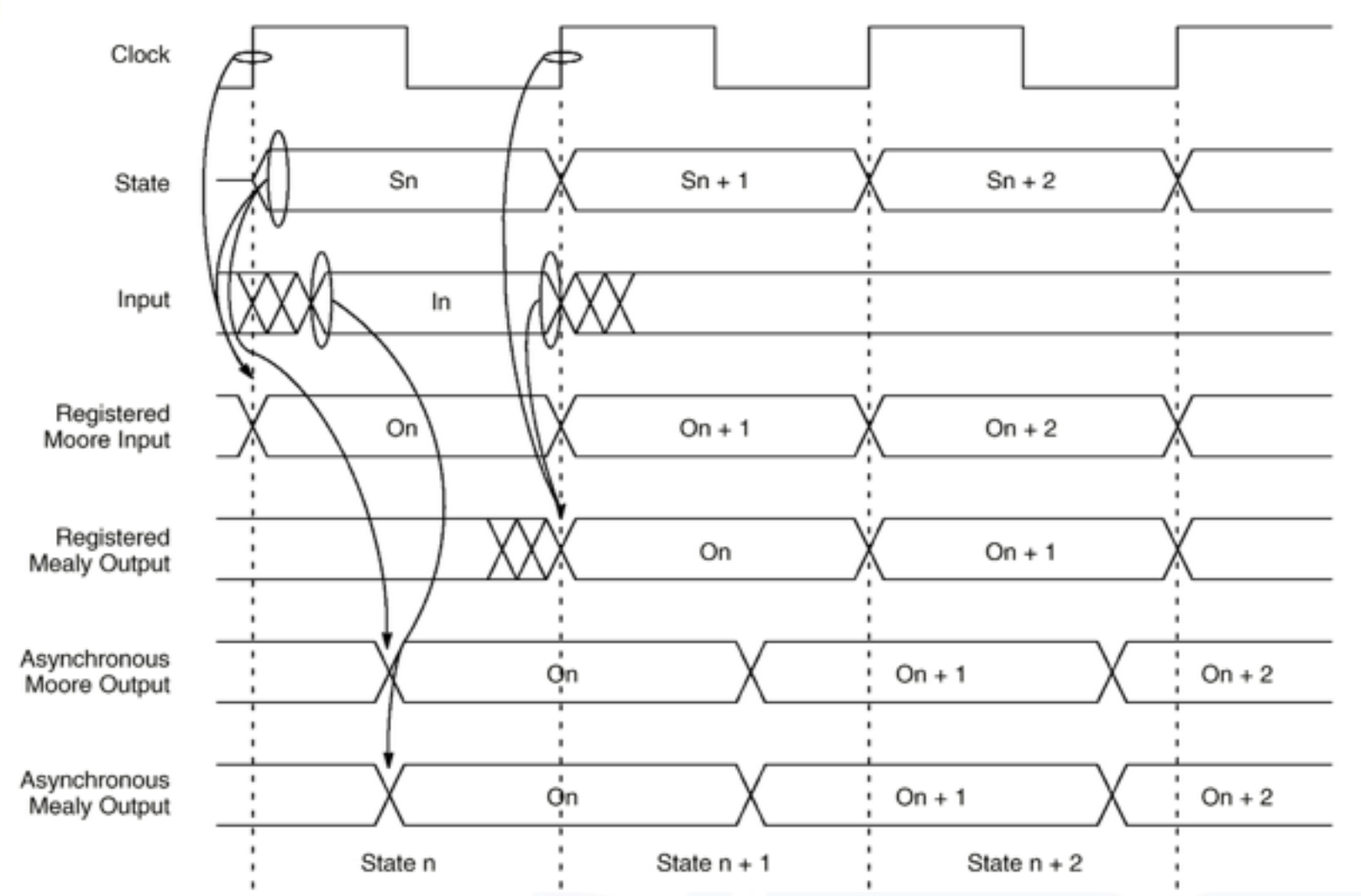


Generic synchronous state machine

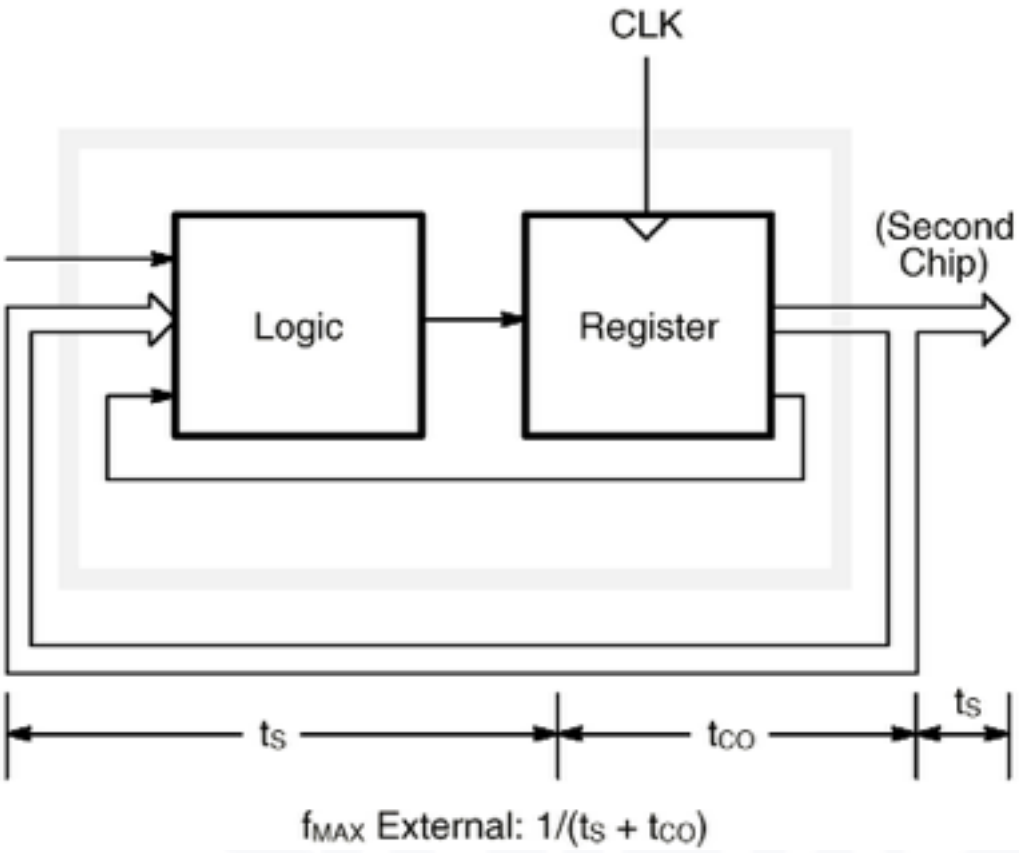
- I/O synchronization registers



Timing diagram



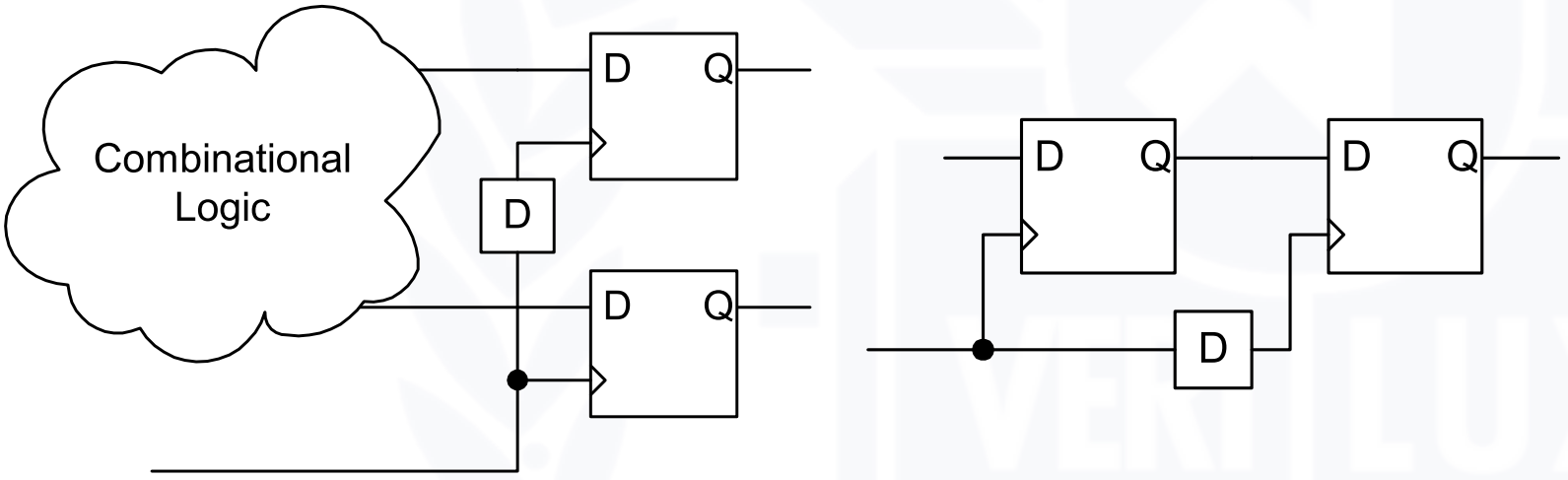
Maximum operating frequency



f_{MAX} External: $1/(t_S + t_{CO})$

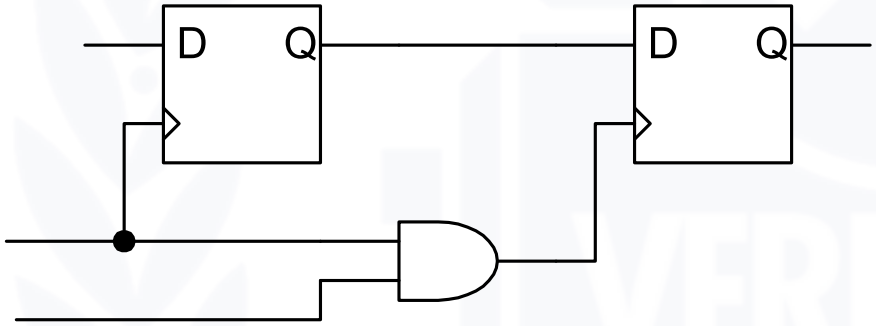
Clock skew

- Timing fault



Clock skew (2)

- Clock skew is caused by
  - Net delay
  - Artificial delay



Finite state machines summary

- Models for representing sequential circuits
  - Abstraction of sequential elements
  - Finite state machines and their state diagrams
  - Inputs/outputs
  - Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
  - Deriving state diagram
  - Deriving state transition table
  - Determining next state and output functions
  - Implementing combinational logic
- Hardware description languages