Ch 7. Finite State Machines

Finite State Machines

- Sequential circuits
  - primitive sequential elements
  - 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

Counters

- Sequential logic circuit that proceed through a well defined sequence of states
  - 3-bit binary up-counter
    - 000, 001, 010, 011, 100, 101, 110, 111; and return to 000
  - Decade counter
    - 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001; and return to 0000 (binary-coded decimal)
  - Gray-code counter
    - Only a single bit of the counter changes at a time to avoid circuit hazard
    - 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000 and repeat

Counters (cont'd)

- Ring counters
  - Shift registers can also be used as a kind of primitive counter
  - Uses minimal hardware for its implementation
  - Not efficient state encoding

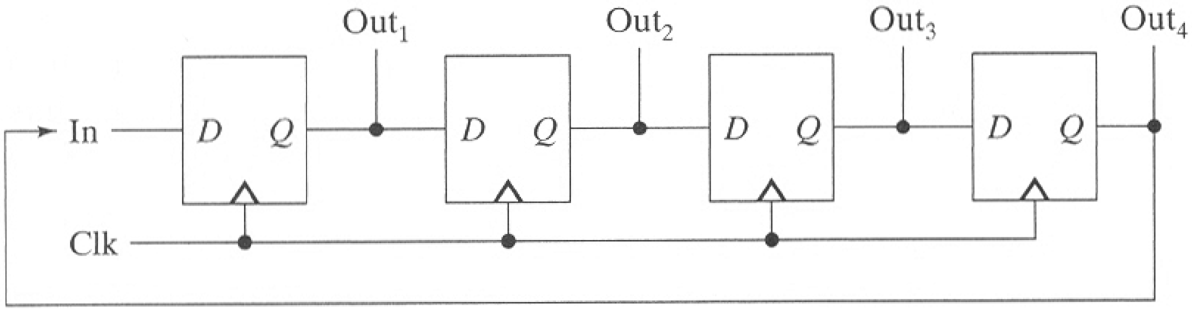


Figure 7.1 Four-bit simple ring counter.

Counters (cont'd)

- Johnson counter (aka Mobius counter)
  - Requires only one inverter more than the basic ring counter
  - Can sequence through twice as many states

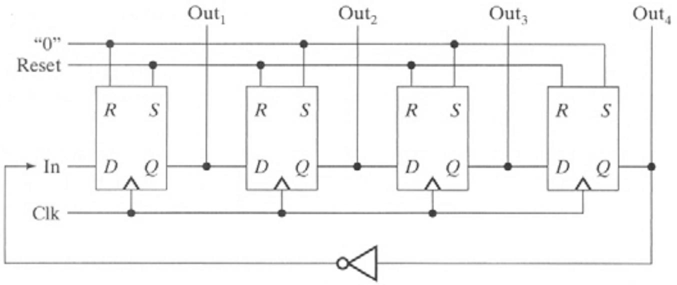


Figure 7.2 **Four-bit Johnson counter.**

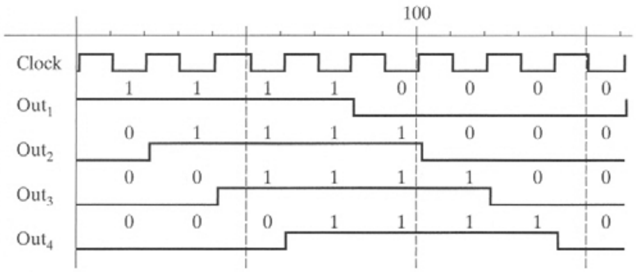
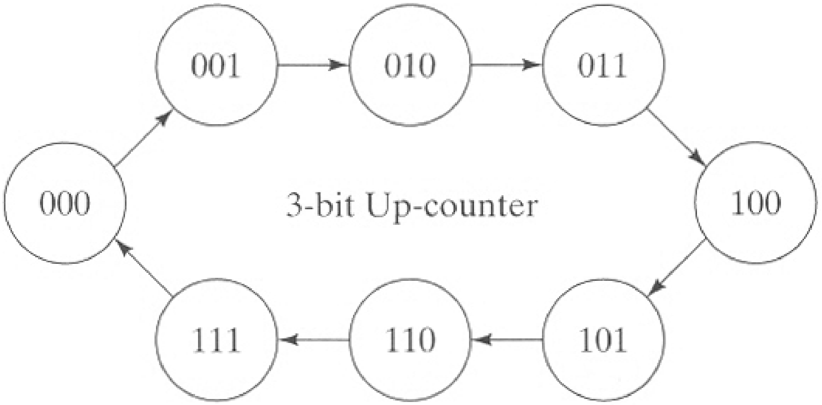
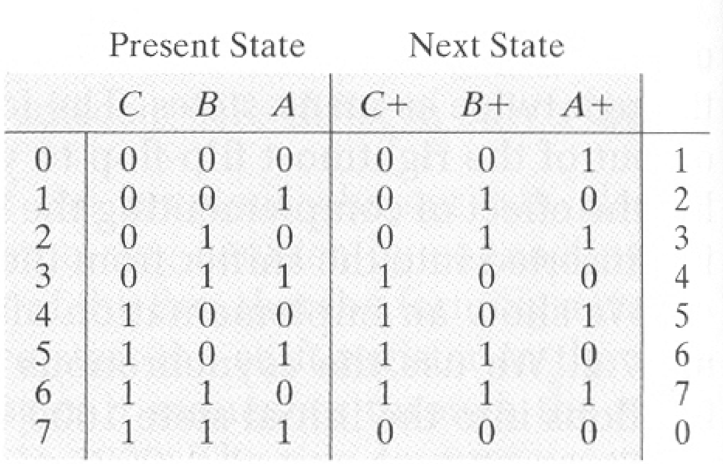


Figure 7.3 **Timing waveforms for 4-bit Johnson counter.**

Counter Design Procedure

- 3-bit binary up-counter
  - Describe state transition diagram and table



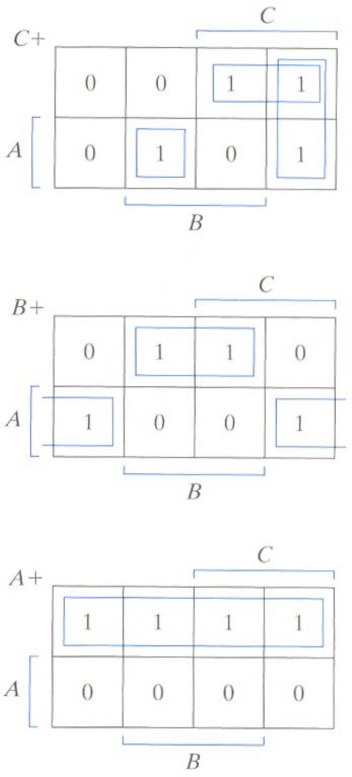


	Present State			Next State			
	C	B	A	$C+$	$B+$	$A+$	
0	0	0	0	0	0	1	1
1	0	0	1	0	1	0	2
2	0	1	0	0	1	1	3
3	0	1	1	1	0	0	4
4	1	0	0	1	0	1	5
5	1	0	1	1	1	0	6
6	1	1	0	1	1	1	7
7	1	1	1	0	0	0	0

Figure 7.4 State transition diagram and table for a 3-bit binary up-counter.

Counter Design Procedure (cont'd)

- 3-bit binary up-counter
  - Design combinational logic through K-map methods

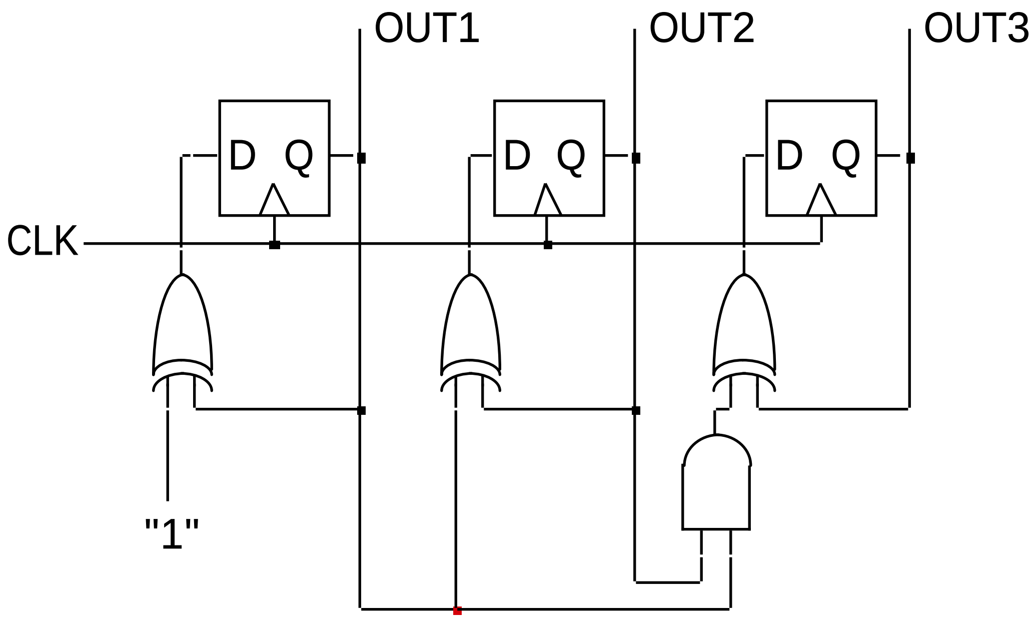


C+			C	C
	0	0	1	1
A	0	1	0	1
		B	B	

B+			C	C
	0	1	1	0
A	1	0	0	1
		B	B	

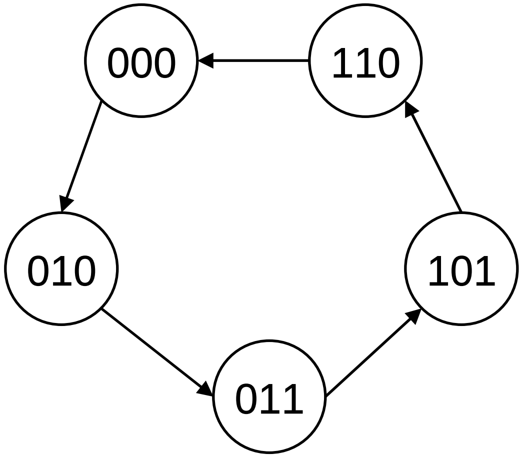
A+			C	C
	1	1	1	1
A	0	0	0	0
		B	B	

Figure 7.5 K-maps for up-counter flip-flops.



More complex counter example

- 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

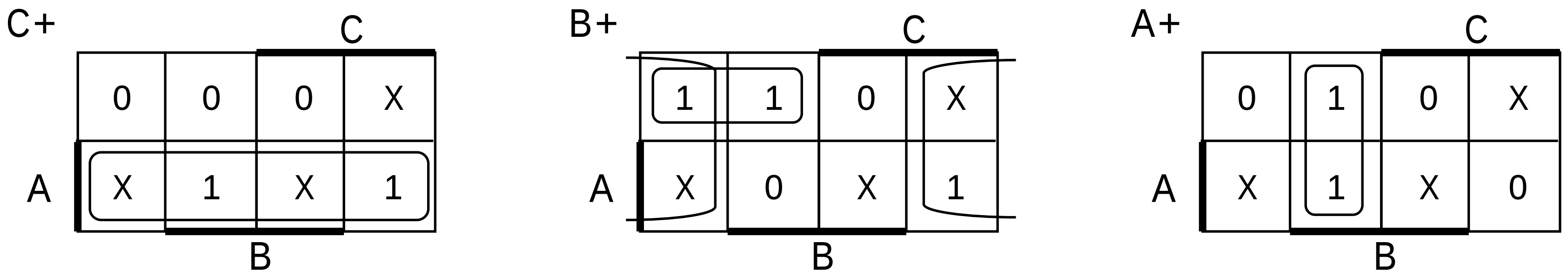


Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	–	–	–
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	–	–	–
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	–	–	–

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

More complex counter example (cont'd)

- Step 3: K-maps for next state functions



C+

	0	0	0	X
A	X	1	X	1

B+

	1	1	0	X
A	X	0	X	1

A+

	0	1	0	X
A	X	1	X	0

C+ <= A

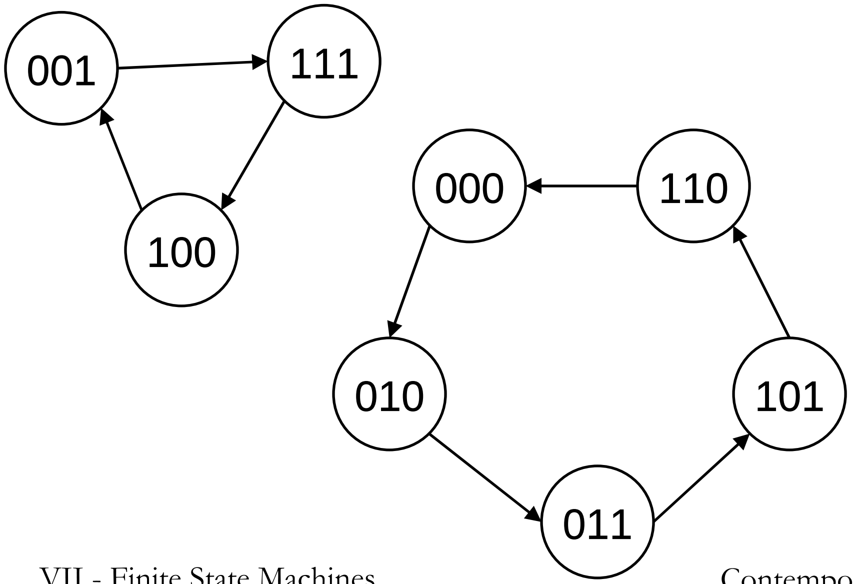
B+ <= B' + A'C'

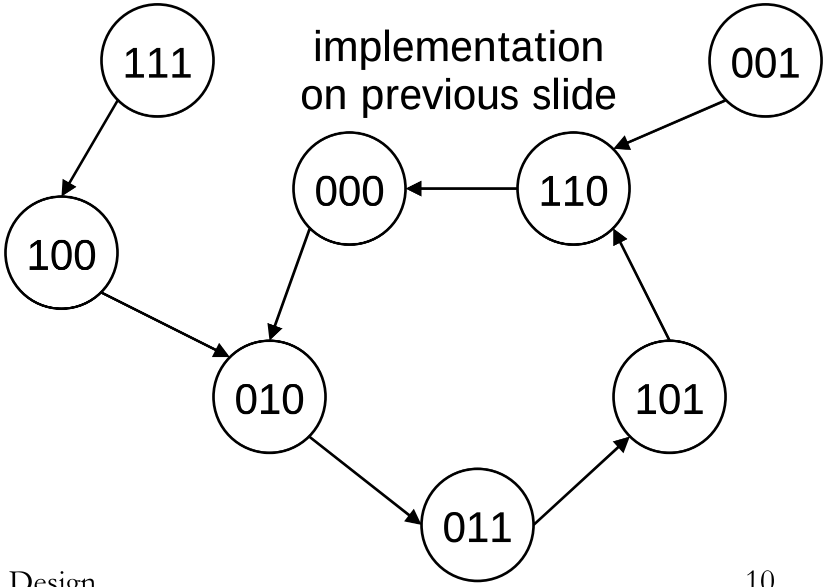
A+ <= BC'

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

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





implementation on previous slide

Self-starting counters (cont'd)

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

C+

			C	C
	0	0	0	0
A	1	1	1	1
		B	B	

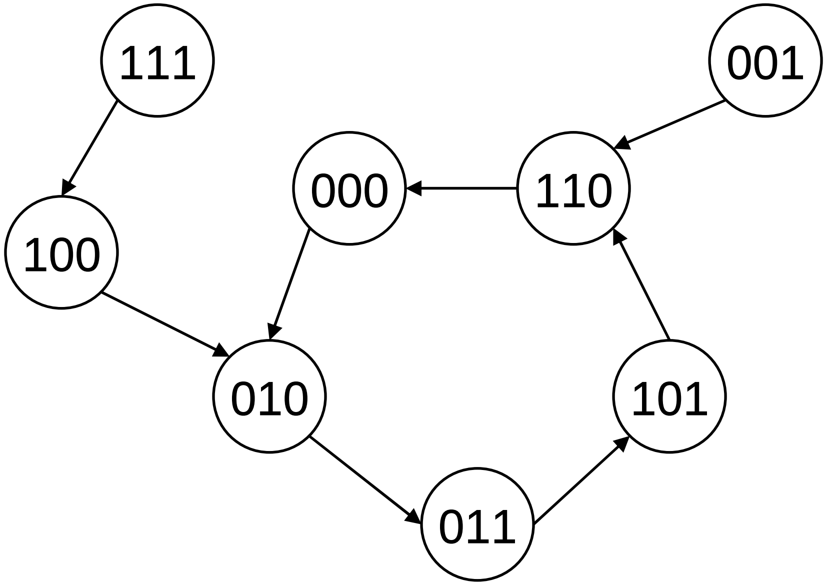
B+

			C	C
	1	1	0	1
A	1	0	0	1
		B	B	

A+

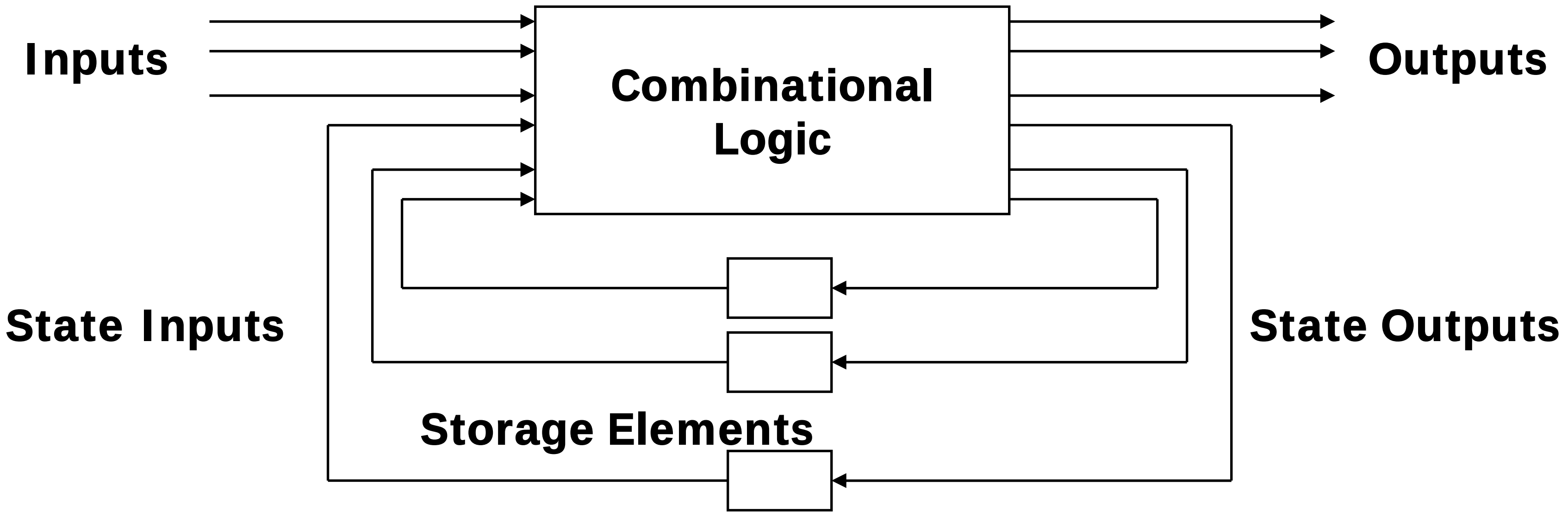
			C	C
	0	1	0	0
A	0	1	0	0
		B	B	

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



Abstraction of state elements

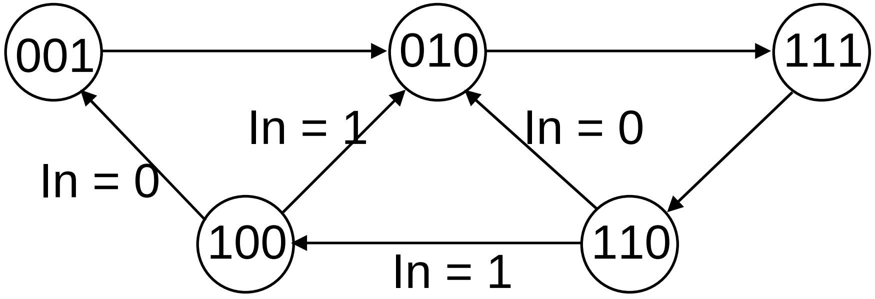
- Divide circuit into combinational logic and state
- Localize the feedback loops and make it easy to break cycles
- Implementation of storage elements leads to various forms of sequential logic



Finite state machine representations

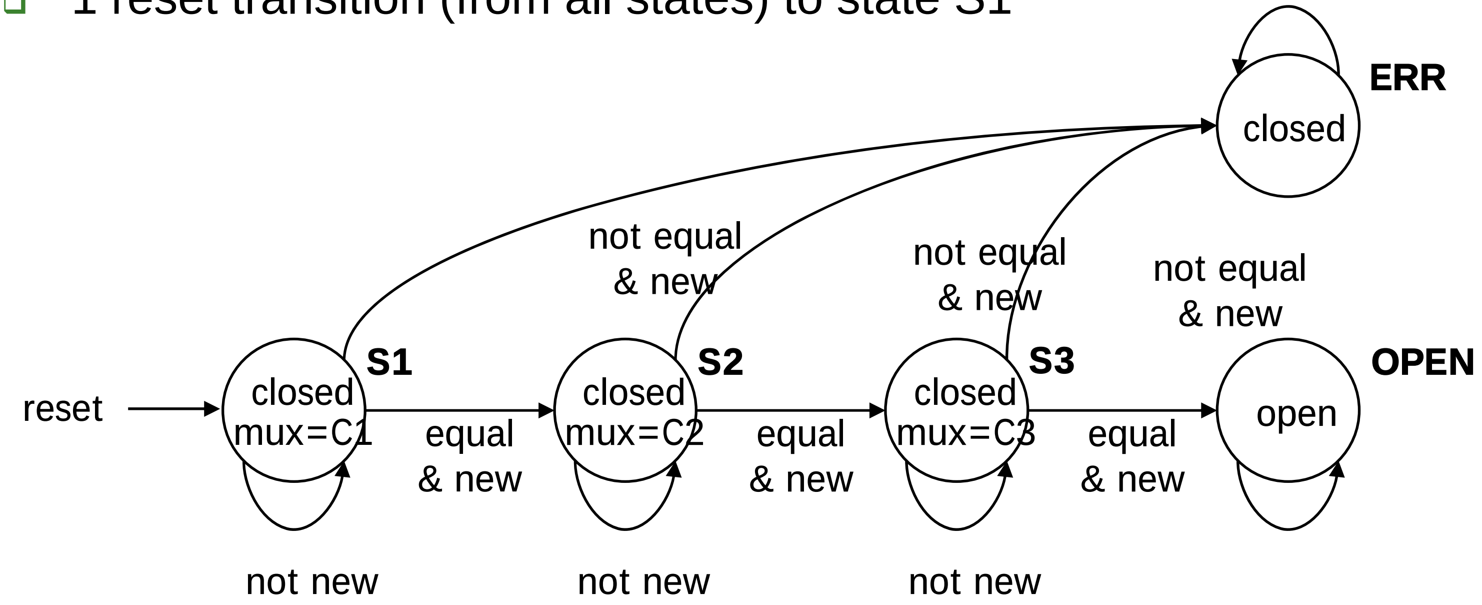
- States: determined by possible values in sequential storage elements
- Transitions: change of state
- Clock: controls when state can change by controlling storage elements

- Sequential logic
  - sequences through a series of states
  - based on sequence of values on input signals
  - clock period defines elements of sequence



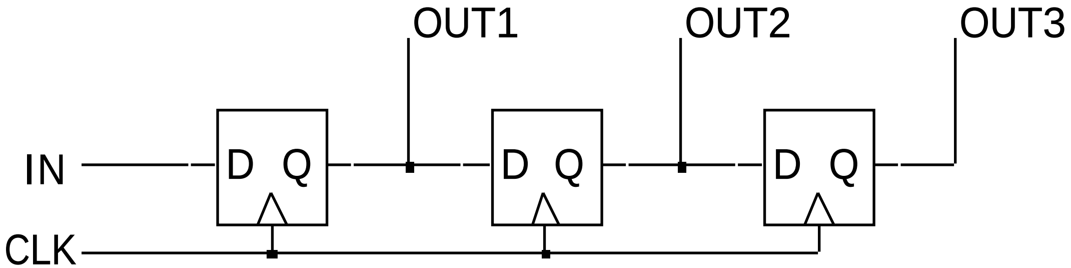
Example finite state machine diagram

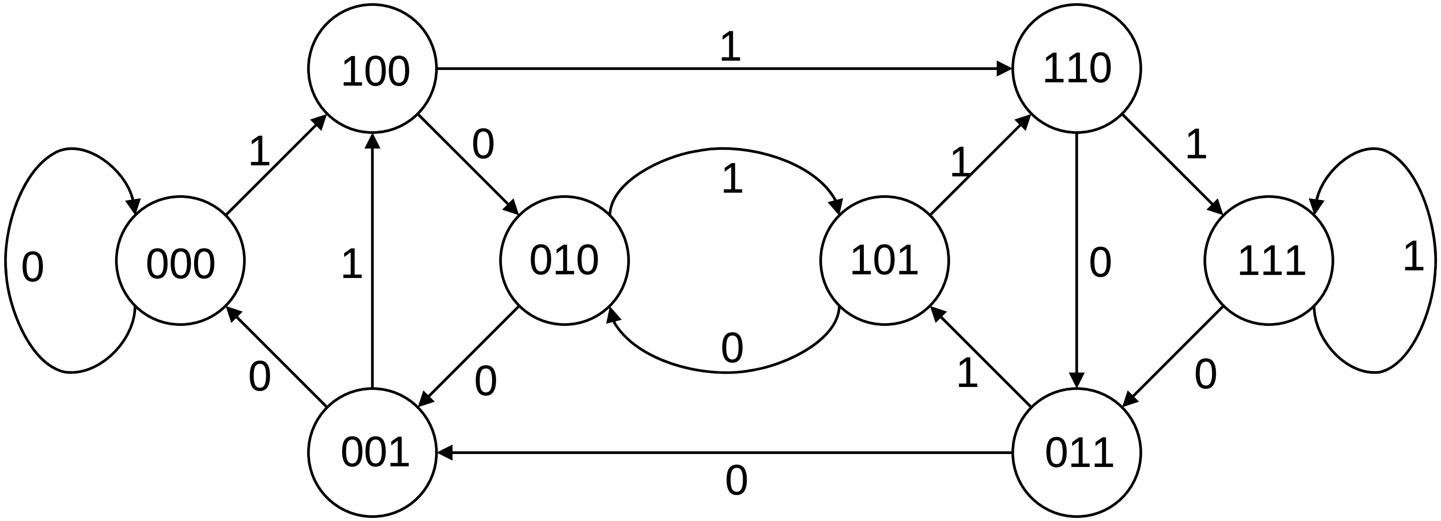
- Combination lock from introduction to course
  - 5 states
  - 5 self-transitions
  - 6 other transitions between states
  - 1 reset transition (from all states) to state S1



Can any sequential system be represented with a state diagram?

- Shift register
  - input value shown on transition arcs
  - output values shown within state node



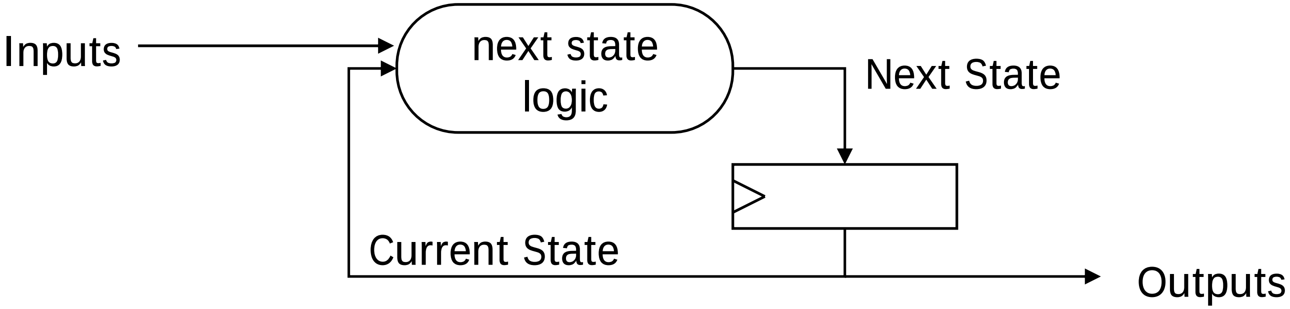


FSM design procedure

- Step 1. Understand the problem
  - Describe a finite state machine in an unambiguous manner
- Step 2. Obtain an abstract representation of the FSM
  - State diagram
- Step 3. Perform state minimization
  - Certain paths through the state machine can be eliminated
- Step 4. Perform state assignment
  - Counter: the state and the output are identical
  - General FSM: good state encoding often leads to a simpler implementation
- Step 5. Implement the finite state machine
  - Using Boolean equations or K-maps

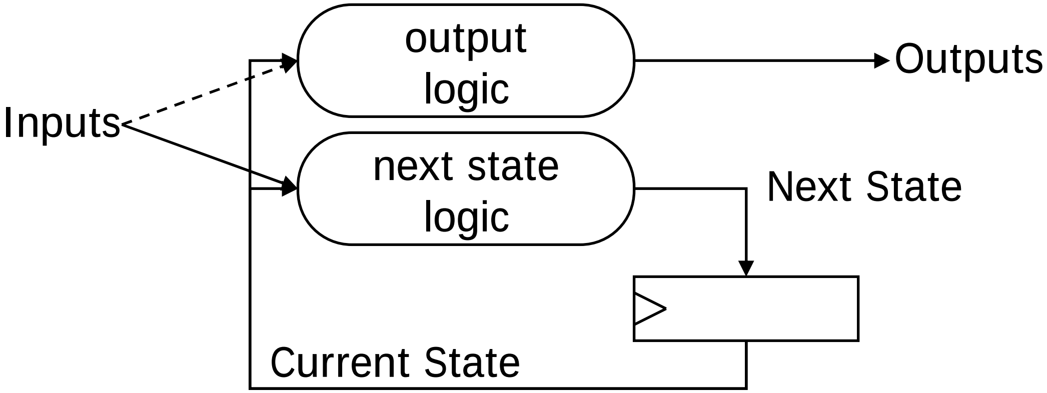
Counter/shift-register model

- Values stored in registers (flip-flops) represent the state of the circuit
- Next state is function of current state and inputs
- Outputs are the state
- Combinational logic implements the function for next state



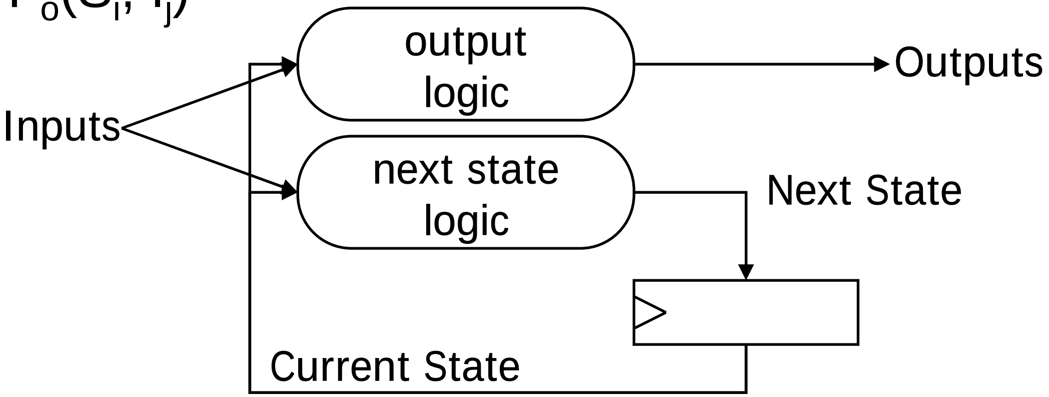
General state machine model

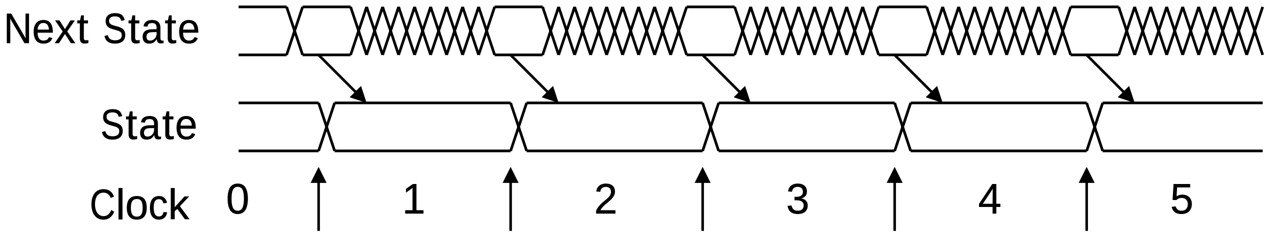
- Values stored in registers represent the state of the circuit
- Next state is function of current state and inputs
- Outputs are
  - function of current state and inputs (Mealy machine)
  - function of current state only (Moore machine)
- Combinational logic implements the functions for next state and outputs



State machine model (cont’d)

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



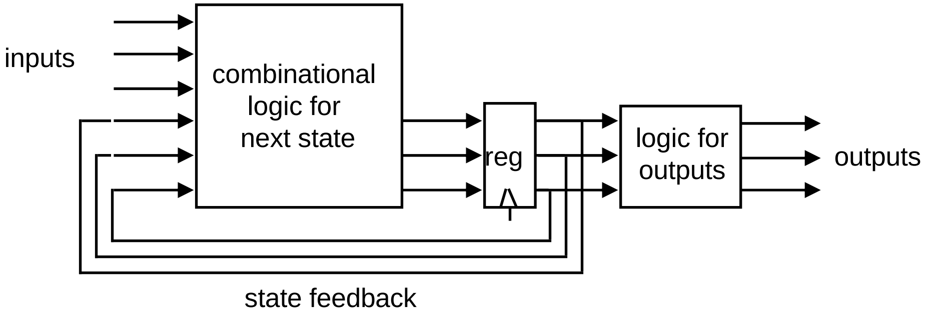


Comparison of Mealy and Moore machines

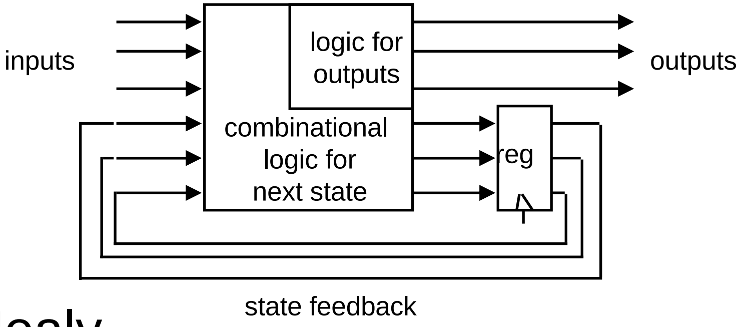
- Mealy machines tend to have less states
    - different outputs on arcs (n^2) rather than states (n)
- Moore machines are safer to use
    - outputs change at clock edge (always one cycle later)
    - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when 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

Comparison of Mealy and Moore machines (cont'd)

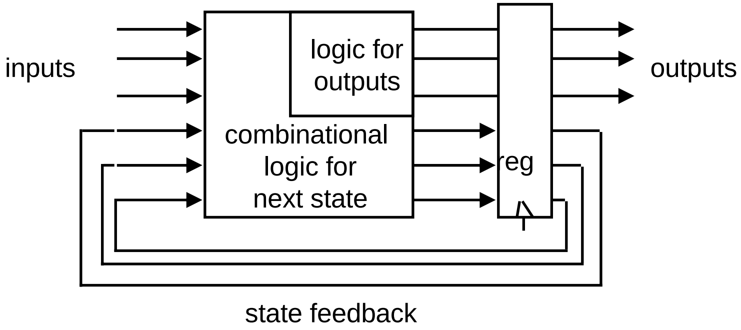
- Moore



- Mealy

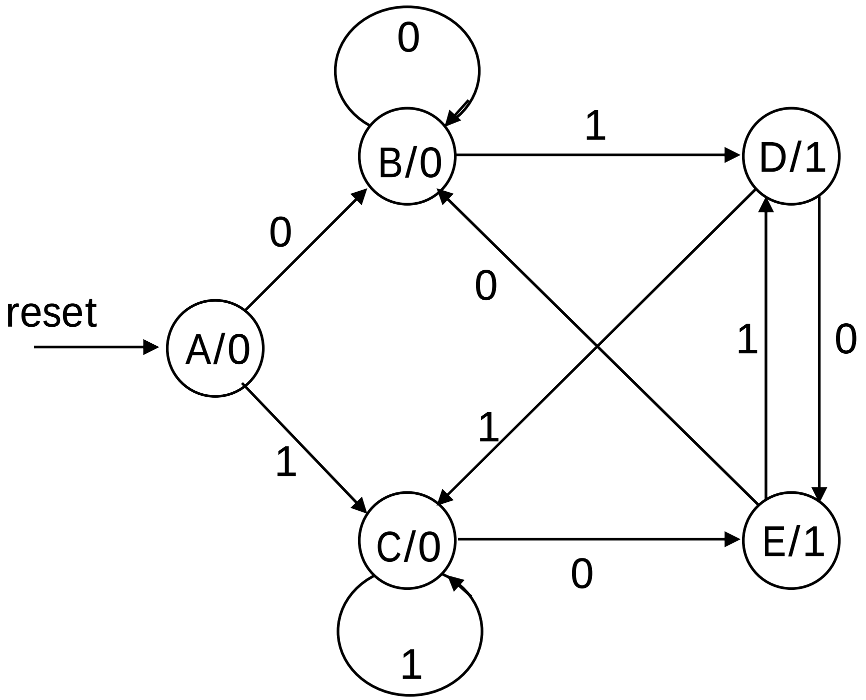


- Synchronous Mealy



Specifying outputs for a Moore machine

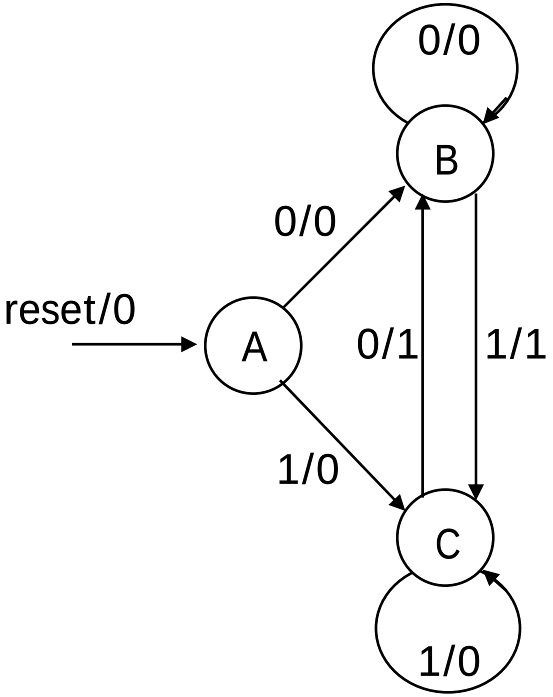
- Output is only function of state
  - specify in state bubble in state diagram
  - example: sequence detector for 01 or 10



		current	next	
reset	input	state	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

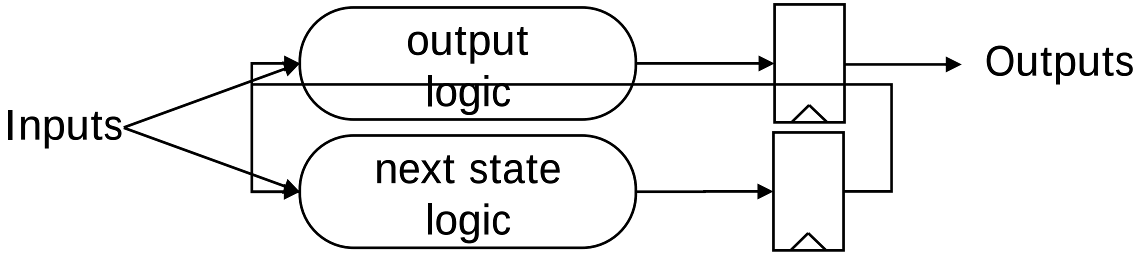
- Output is function of state and inputs
  - specify output on transition arc between states
  - example: sequence detector for 01 or 10



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)

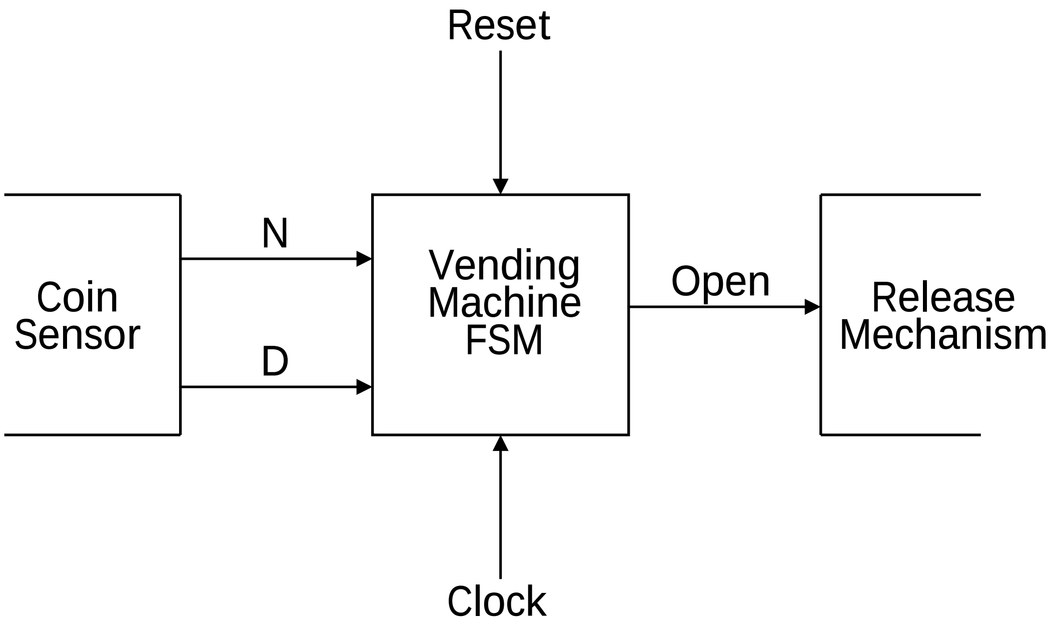
- Synchronous (or registered) Mealy machine
  - registered state AND outputs
  - avoids ‘glitchy’ outputs
  - easy to implement in PLDs
- Moore machine with no output decoding
  - outputs computed on transition to next state rather than after entering
  - view outputs as expanded state vector



Current State

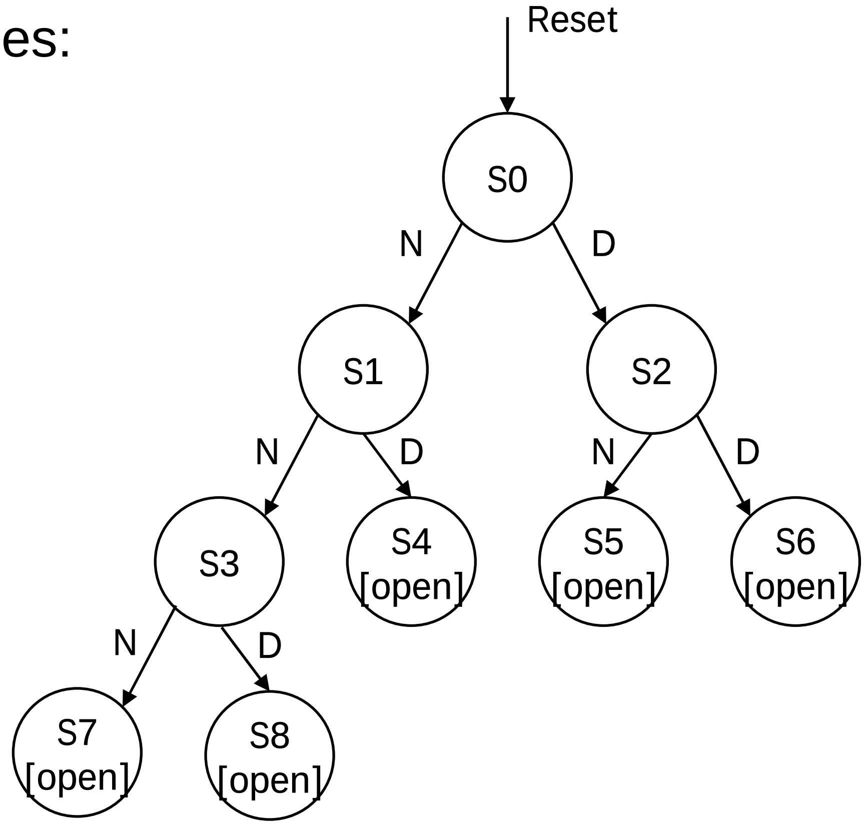
Example: vending machine

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



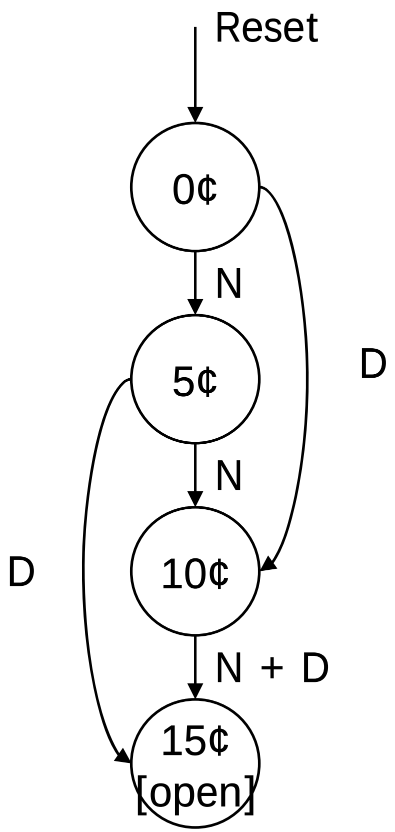
Example: vending machine (cont'd)

- 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)



Example: vending machine (cont'd)

- 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

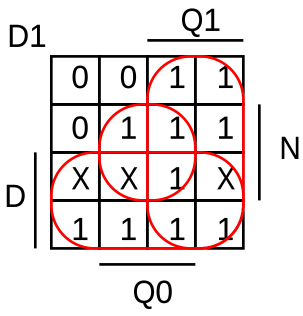
Example: vending machine (cont'd)

- Uniquely encode states

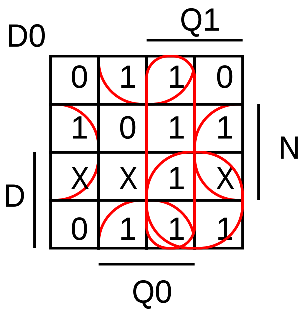
present state		inputs		next state		output
Q1	Q0	D	N	D1	D0	open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	–	–	–
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	–	–	–
1	1	–	–	1	1	1

Example: Moore implementation

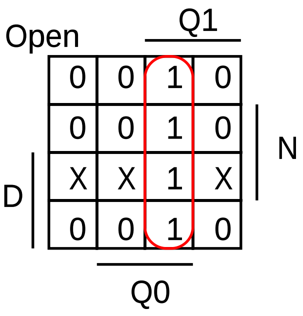
- Mapping to logic



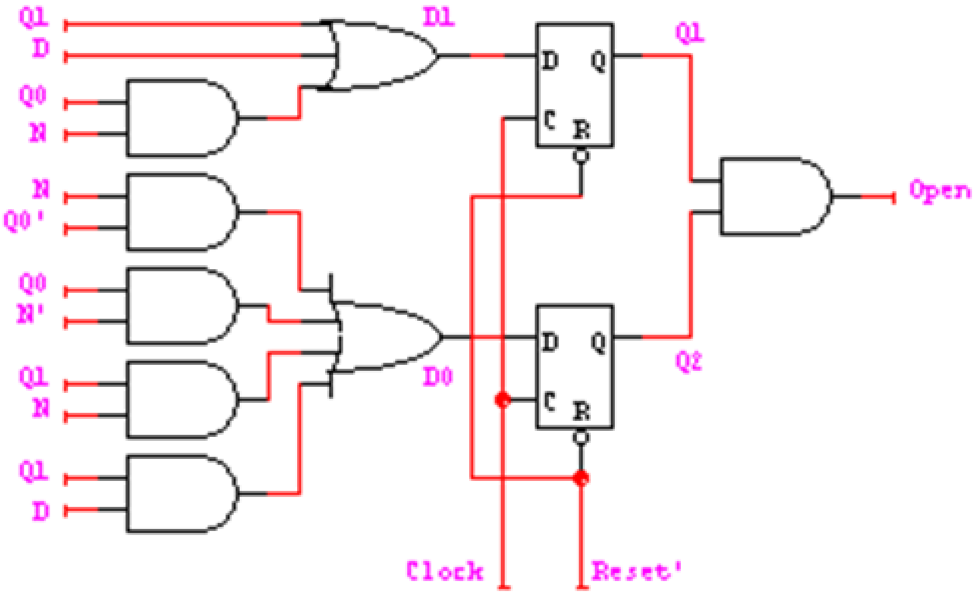
0	0	1	1
0	1	1	1
X	X	1	X
1	1	1	1



0	1	1	0
1	0	1	1
X	X	1	X
0	1	1	1



0	0	1	0
0	0	1	0
X	X	1	X
0	0	1	0



D0 = Q0’ N + Q0 N’ + Q1 N + Q1 D

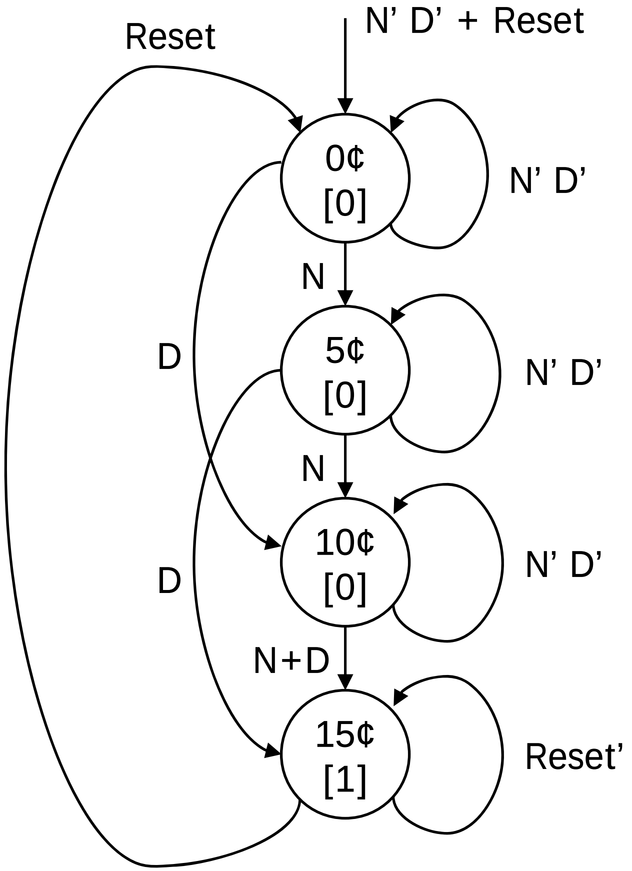
D1 = Q1 + D + Q0 N

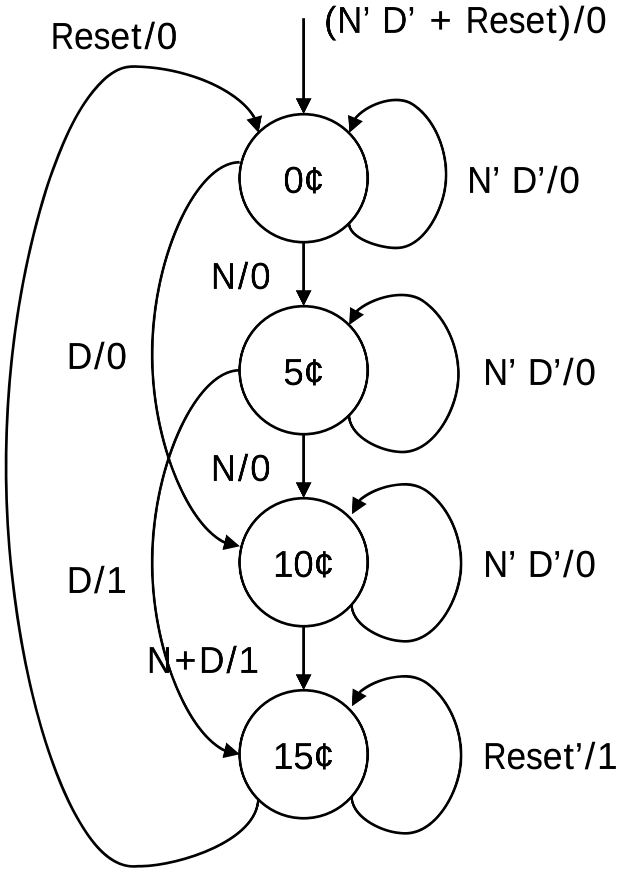
OPEN = Q1 Q0

Equivalent Mealy and Moore state diagrams

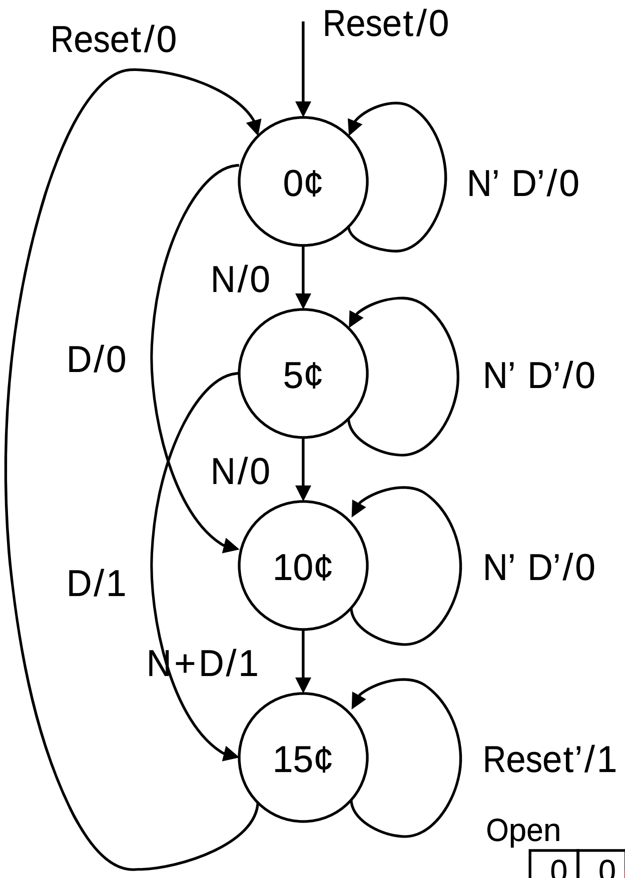
- Moore machine
  - outputs associated with state

- Mealy machine
  - outputs associated with transitions





Example: Mealy implementation



present state		inputs		next state		output
Q1	Q0	D	N	D1	D0	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

Open

			Q1	Q1	
	0	0	1	0	
	0	0	1	1	N
D	X	X	1	X	N
D	0	1	1	1	
		Q0	Q0		

D0 = Q0’N + Q0N’ + Q1N + Q1D

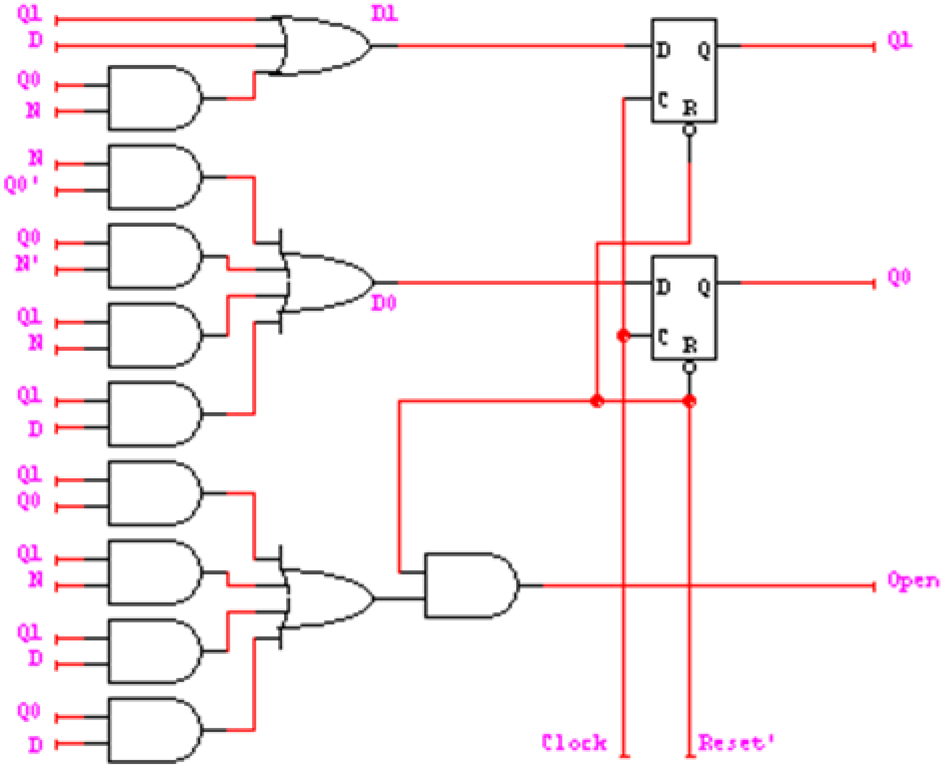
D1 = Q1 + D + Q0N

OPEN = Q1Q0 + Q1N + Q1D + Q0D

Example: Mealy implementation

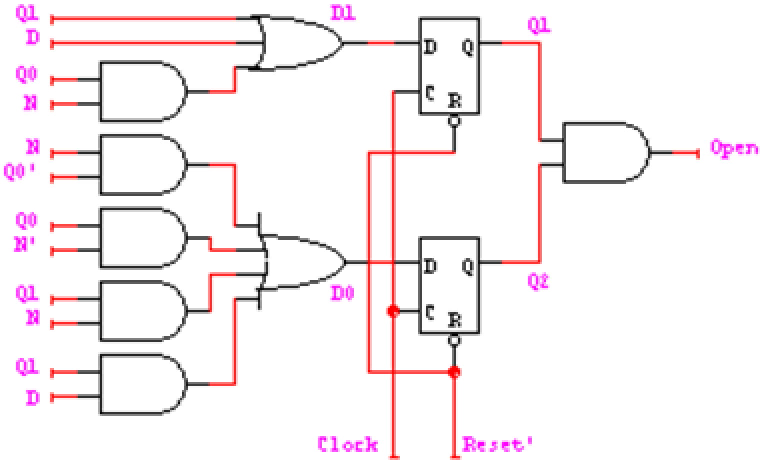
D0 = Q0’N + Q0N’ + Q1N + Q1D
D1 = Q1 + D + Q0N
OPEN = Q1Q0 + Q1N + Q1D + Q0D

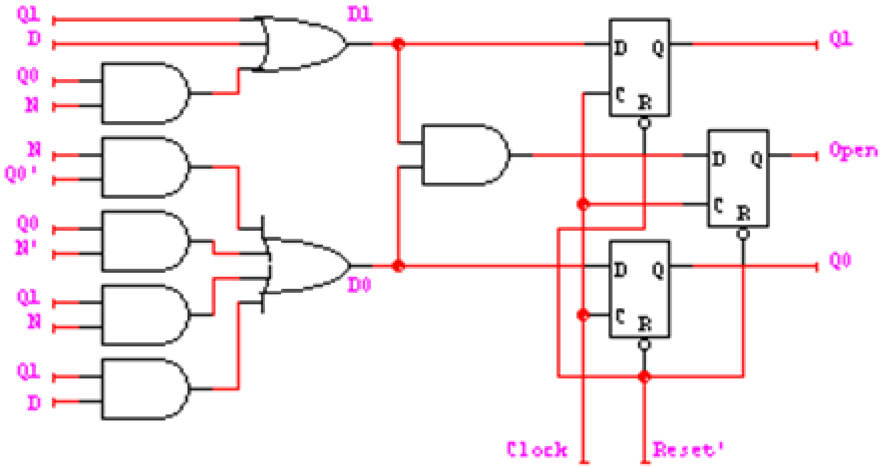
make sure OPEN is 0 when reset
– by adding AND gate



Vending machine: Moore to synch. Mealy

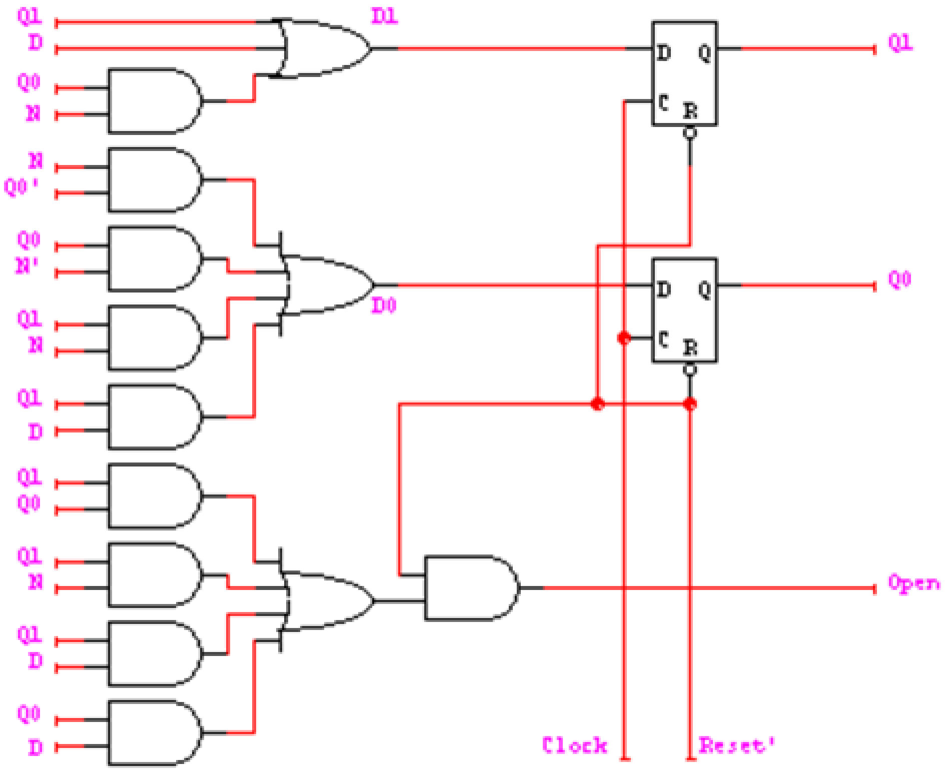
- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 change in Moore implementation
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D
- Implementation 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

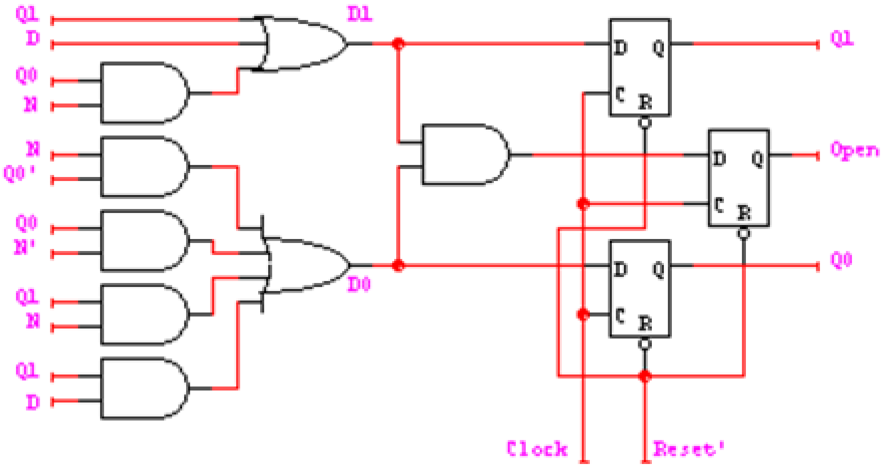
0	0	1	0
0	0	1	1
1	0	1	1
0	1	1	1

N D Q0

Open.d Q1

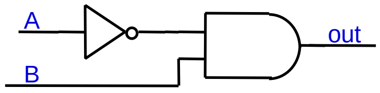
0	0	1	0
0	0	1	1
X	X	1	X
0	1	1	1

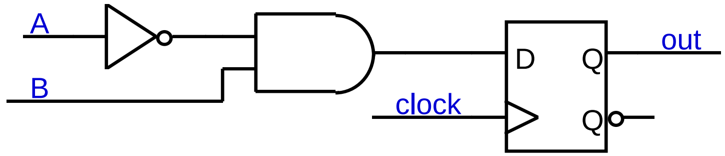
N D Q0

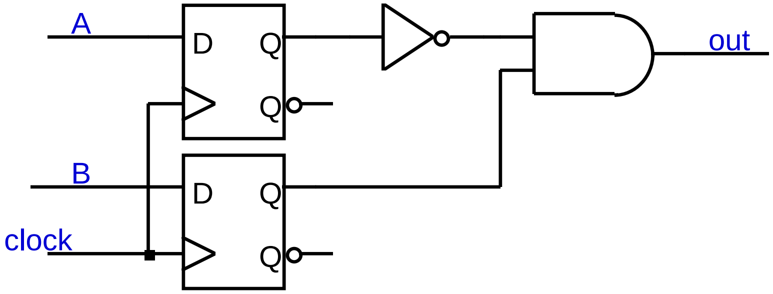


Mealy and Moore examples

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

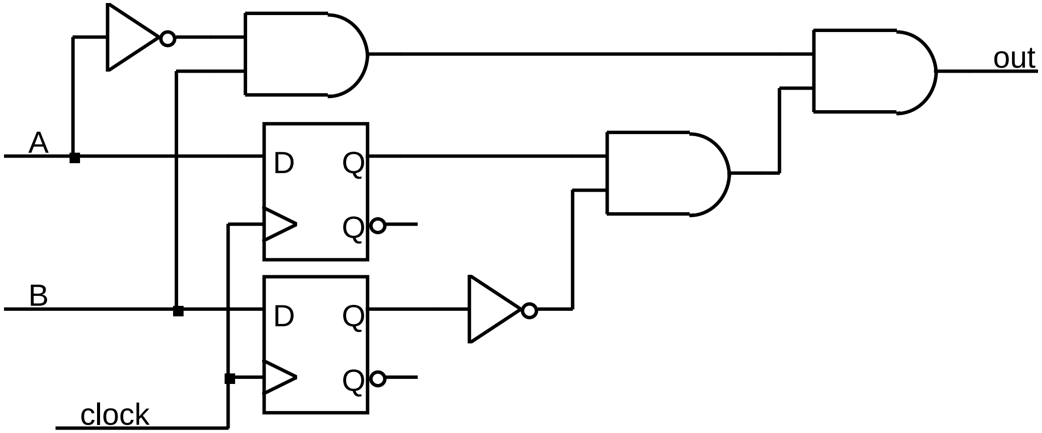


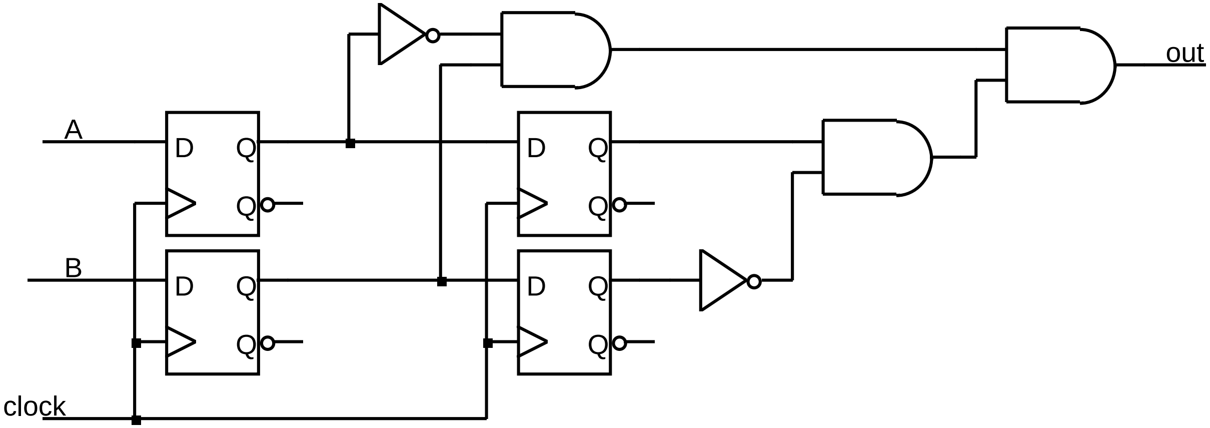




Mealy and Moore examples (cont'd)

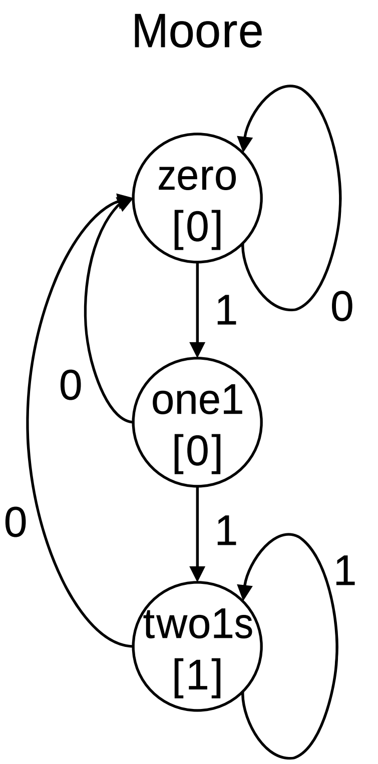
- Recognize A,B = 1,0 then 0,1
  - Mealy or Moore?

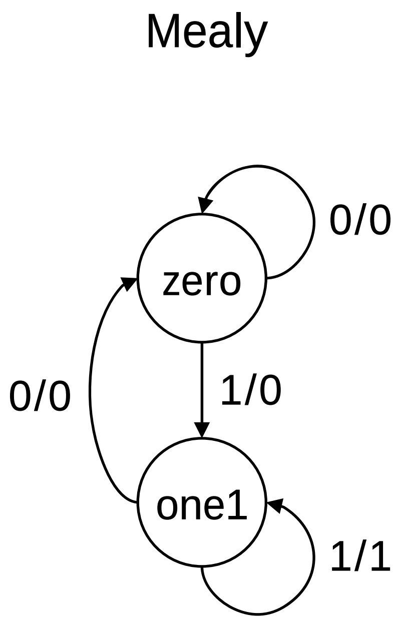




Example: reduce-1-string-by-1

- Remove one 1 from every string of 1s on the input





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