Lecture 17

- Logistics
  - HW6 due today, HW7 out last Monday --- due on Monday
  - Midterm review Tuesday 4:15pm Room TBA
- Last lecture
  - Introduction to finite state machines
  - Counters as finite state machines
- Today
  - Finish counter design
  - More complex finite-state machines
  - Introduction of Moore and Mealy machines

Review: Counter design procedure

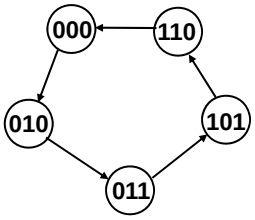
1. Draw a state diagram
2. Draw a state-transition table
3. Encode the next-state functions
   - Minimize the logic using k-maps
4. Implement the design

Example: A 5-state counter

Example: A 5-state counter

- Counter repeats 5 states in sequence
  - Sequence is 000, 010, 011, 101, 110, 000 (not binary)

Step 1: State diagram



Step 2: State transition table

Assume D flip-flops

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	–	–	–

5-state counter (con't)

Step 3: Encode the next state functions

C+

A \ C	0	0	0	X
A	X	1	X	1

B

C+ = A

B+

A \ C	1	1	0	X
A	X	0	X	1

B

B+ = B' + A'C'

A+

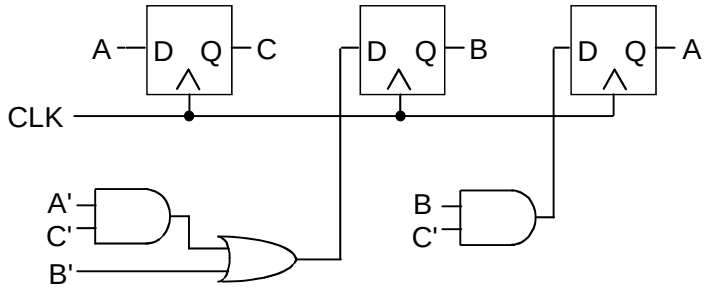
A \ C	0	1	0	X
A	X	1	X	0

B

A+ = BC'

5-state counter (con't)

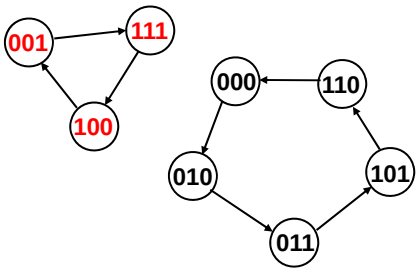
Step 4: Implement the design



5-state counter (con't)

- Is our design robust?
  - What if the counter starts in a 111 state?

Does our counter get stuck in invalid states???



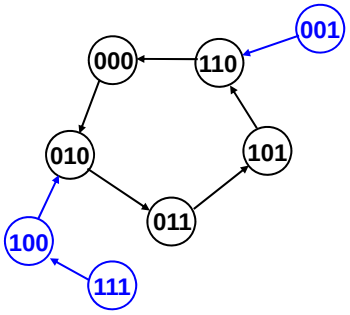
5-state counter (con't)

- Back-annotate our design to check it

Fill in state transition table

Draw state diagram

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



A+ = BC'

B+ = B' + A'C'

C+ = A

Self-starting counters

- Invalid states should always transition to valid states
  - Assures startup
  - Assures bit-error tolerance
- Design your counters to be self-starting
  - Draw all states in the state diagram
  - Fill in the entire state-transition table
  - May limit your ability to exploit don't cares
    - Choose startup transitions that minimize the logic

Finite state machines: more than counters

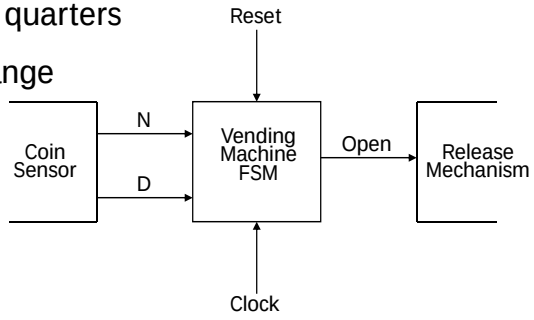
- FSM: A system that visits a finite number of logically distinct states
- Counters are simple FSMs
  - Outputs and states are identical
  - Visit states in a fixed sequence
- FSMs are typically more complex than counters
  - Outputs can depend on current state and on inputs
  - State sequencing depends on current state and on inputs

FSM design

- Counter-design procedure
  1. State diagram
  2. State-transition table
  3. Next-state logic minimization
  4. Implement the design
- FSM-design procedure
  1. State diagram
  2. state-transition table
  3. State minimization
  4. State encoding
  5. Next-state logic minimization
  6. Implement the design

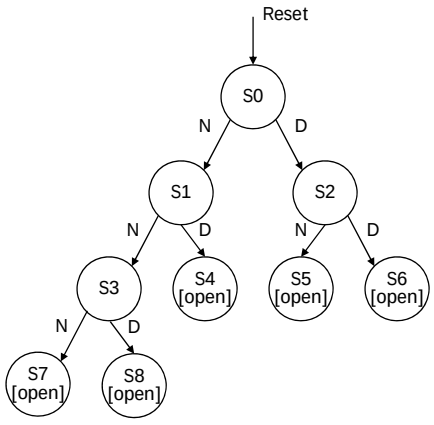
Example: A vending machine

- 15 cents for a cup of coffee
- Doesn't take pennies or quarters
- Doesn't provide any change



- FSM-design procedure
  1. State diagram
  2. state-transition table
  3. State minimization
  4. State encoding
  5. Next-state logic minimization
  6. Implement the design

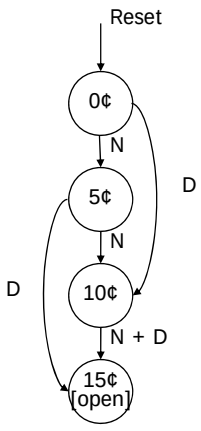
A vending machine: state diagram



A vending machine: State transition table

present state	inputs D	N	next state	output open
S0	0	0	S0	0
	0	1	S1	0
	1	0	S2	0
	1	1	--	–
S1	0	0	S1	0
	0	1	S3	0
	1	0	S4	1
	1	1	–	–
S2	0	0	S2	0
	0	1	S5	1
	1	0	S6	1
	1	1	–	–
S3	0	0	S3	0
	0	1	S7	1
	1	0	S8	1
	1	1	–	–
S4	–	–	S4	1
S5	–	–	S5	1
S6	–	–	S6	1
S7	–	–	S7	1
S8	–	–	S8	1

A vending machine: State minimization



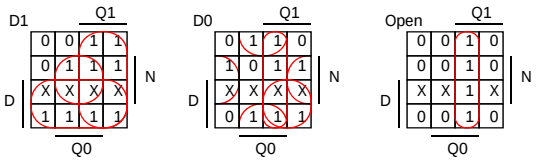
present state	inputs D	N	next state	output open
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	–	–
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	–	–
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	–	–
15¢	–	–	15¢	1

symbolic state table

A vending machine: State encoding

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

A vending machine: Logic minimization

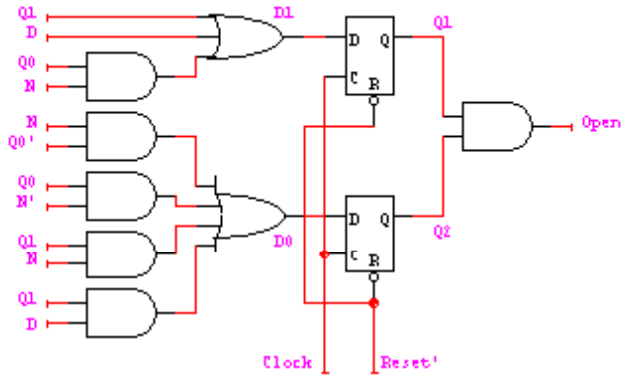


D1 = Q1 + D + Q0 N

D0 = Q0' N + Q0 N' + Q1 N + Q1 D

OPEN = Q1 Q0

A vending machine: Implementation



Generalized FSM model

- State variables (state vector) holds circuit state
  - Stored in registers
- Combinational logic computes next state and outputs
  - Next state is a function of current state and inputs
  - Outputs are functions of
    - Current state (Moore machine)
    - Current state and inputs (Mealy machine)