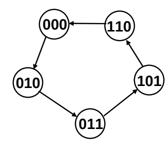
Lecture 18

- Logistics
  - HW6 due today
  - Midterm 2
    - Wednesday Feb 25
    - Review session Tuesday Feb 24, 4:30 in this room, EEB 037
    - Will cover materials up to today's lecture
- Last Lecture
  - Counter Finite State Machine
  - Timing
- Today
  - General Finite State Machine (FSM) design

One more counter example: A 5-state counter with D flip flops

- Counter repeats 5 states in sequence
  - Sequence is 000, 010, 011, 101, 110, 000

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

5-state counter (con't)

Step 3: Encode the next state functions

C+ (C over columns 3–4, B under columns 2–3, A on row 2)

0	0	0	X
X	1	X	1

C+ = A

B+

1	1	0	X
X	0	X	1

B+ = B' + A'C'

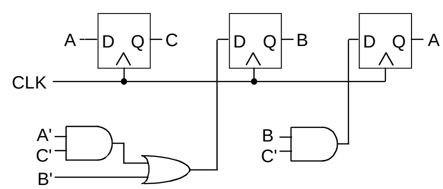
A+

0	1	0	X
X	1	X	0

A+ = BC'

5-state counter (con't)

Step 4: Implement the design

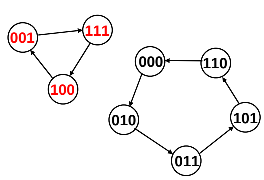


Recall that a D flip flop also produces Q' so A', B', and C' would all be available without any extra inverters

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

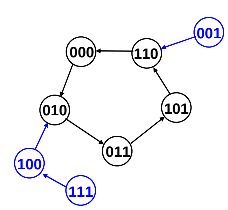
Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	1	0	0

A+ = BC'

B+ = B' + A'C'

C+ = A

Draw state diagram



The proper methodology is to ***design*** your counter to be self-starting

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

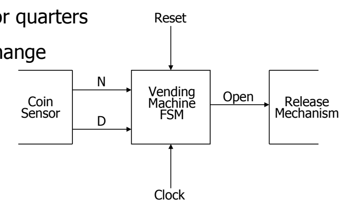
- FSM: A system that visits a finite number of logically distinct states
- Counters are simple FSMs
  - Outputs and states are identical
  - Visit states in a fixed sequence without inputs
- FSMs are typically more complex than counters
  - Outputs can depend on current state and on inputs
  - State 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

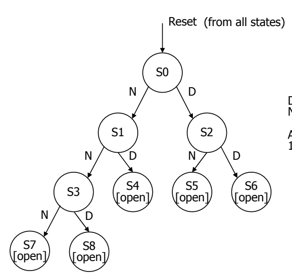
Example: A vending machine

- 15 cents for a cup of coffee (yeah, it's subsidized)
- 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: (conceptual) state diagram



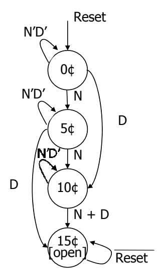
Draw self-loops for N' D' for S0 to S3

Also draw self-loops for 1 for S4 to S8

A vending machine: State transition table

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

A vending machine: State minimization



present state	inputs D	inputs 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	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, N, D, Q0)

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

(Q1, N, D, Q0)

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

Open (Q1, N, D, Q0)

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

D1 = Q1 + D + Q0 N

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

OPEN = Q1 Q0

A vending machine: Implementation