

## Lecture 18

---

### ◆ Logistics

- HW5 due today (with extra 10%)
- HW5 due Friday (20% off on Mon 10:29am, Sol'n posted 10:30am)
- HW6 out, due Wednesday
- My office hours canceled on Friday (I am out of town)
- Brian will cover lecture on Friday
- Midterm 2 covers materials up to Monday lecture & HW6

### ◆ Last lecture

- Registers/counters
- Design counters

### ◆ Today

- More counter designs
- Finite state machine design

## The "WHY" slide

---

### ◆ Finite State Machine (FSM)

- This is what we have been waiting for in this class. Using combinational and sequential logics, now you can design a lot of clever digital logic circuits for functional products. We will learn different steps you take to go from word problems to logic circuits. We first talk about a simplified version of FSM which is a counter.

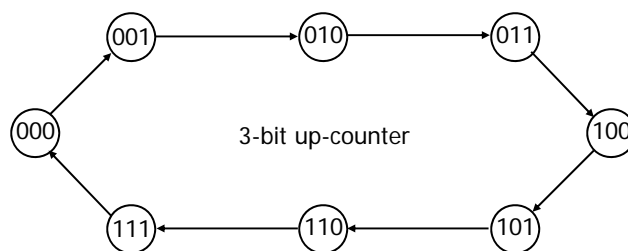
## Another 3-bit up counter: with T flip flops

---

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

## 1. Draw a state diagram

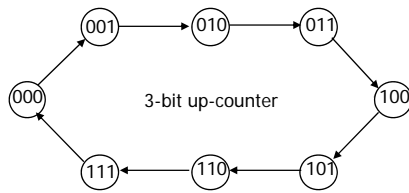
---



## 2. Draw a state-transition table

◆ Like a truth-table

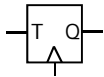
- State encoding is easy for counters → Use count value



	current 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

## 3. Encode the next state functions

T flip-flops



$T_1 := 1$   
 $T_2 := C_1$   
 $T_3 := C_1 C_2$

C3	C2	C1	N3	N2	N1	T3	T2	T1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

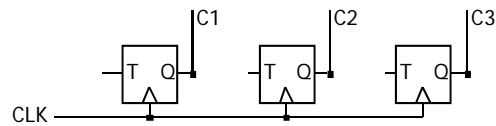
T1	C3			
	1	1	1	1
C1	1	1	1	1
	C2			

T2	C3			
	0	0	0	0
C1	1	1	1	1
	C2			

T3	C3			
	0	0	0	0
C1	0	1	1	0
	C2			

## 4. Implement the design

---

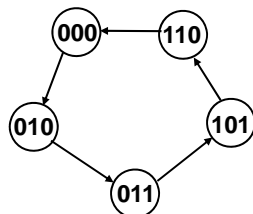


## One more counter example: A 5-state counter with D flip flops

---

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

[Step 1: State diagram](#)

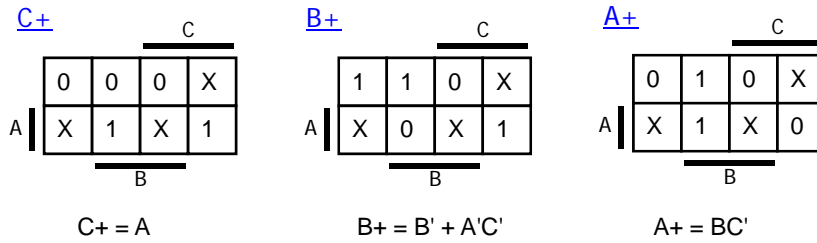


[Step 2: State transition table](#)  
Assume D flip-flops

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

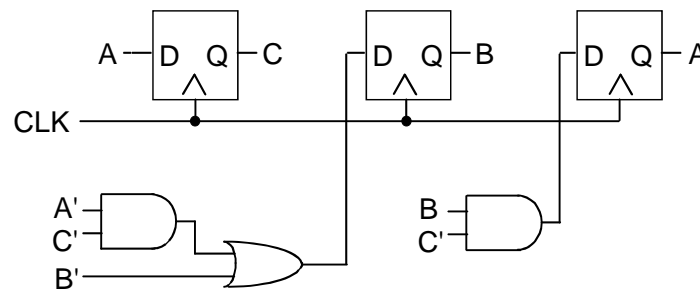
## 5-state counter (con't)

### Step 3: Encode the next state functions



## 5-state counter (con't)

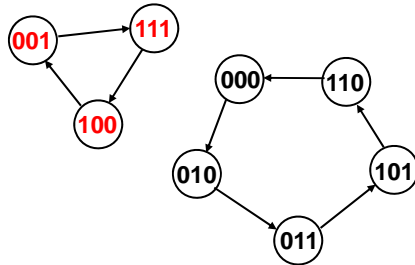
### Step 4: Implement the design



## 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](#)

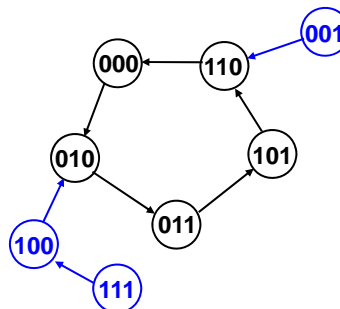
[Draw state diagram](#)

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

$$A+ = BC'$$

$$B+ = B' + A'C'$$

$$C+ = A$$



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

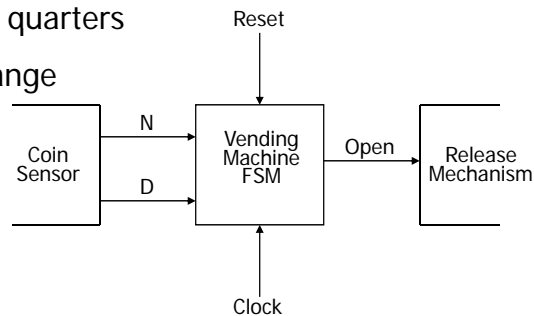
---

- 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

---

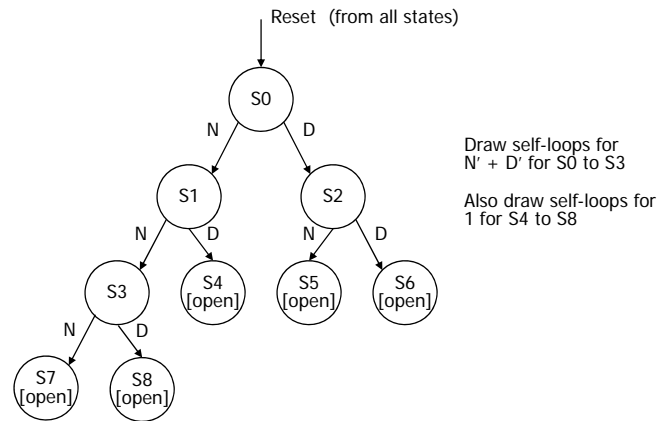
- ◆ 15 cents for a cup of coffee
- ◆ Doesn't take pennies or quarters
- ◆ Doesn't provide any change



- 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



## A vending machine: State transition table

present state	inputs		next state	output open
	D	N		
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