

---

# Computer Science 141

## Computing Hardware

Fall 2009

Harvard University

Instructor: Prof. David Brooks

[dbrooks@eecs.harvard.edu](mailto:dbrooks@eecs.harvard.edu)

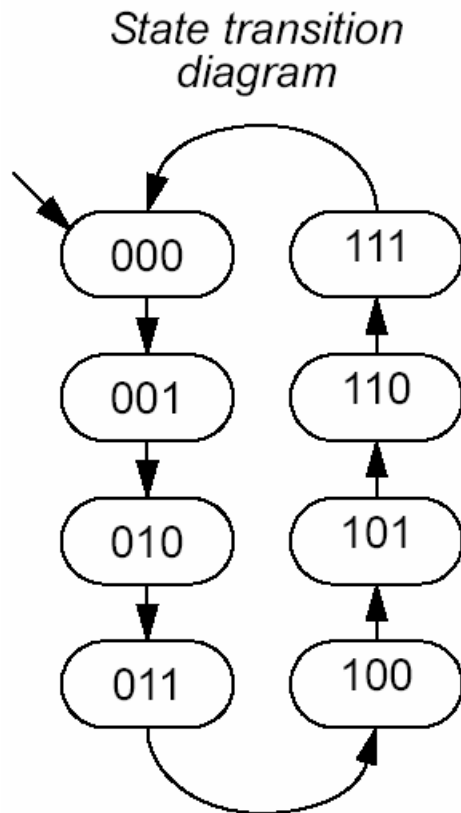
# Counters

---

- sequential logic circuits that proceed through:
  - a well-defined sequence of states
- types
  - up-counter vs. down-counter
  - binary vs. decade vs. Gray code vs. ring vs. Johnson
- simplest possible finite-state machines (FSMs)
  - single input (typically clock saying “count!”)
  - outputs simply the current state

# design example: 3-bit binary up-counter

- Step 1: describe operation (graphically and in tabular form)



*State transition table*

Present State			Next State		
S2	S1	S0	S2 <sup>+</sup>	S1 <sup>+</sup>	S0 <sup>+</sup>
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

# 3-bit binary up-counter example (continued)

---

- Step 2: choose a flip-flop type

Excitation Tables

Q	Q <sup>+</sup>	R	S	J	K	D	T
0	0	X	0	0	X	0	0
0	1	0	1	1	X	1	1
1	0	1	0	X	1	0	1
1	1	0	X	X	0	1	0

# 3-bit binary up-counter example (continued)

- Step 3: remap next state into required FF inputs

			Present State			Next State			Remapped Next State		
			S2	S1	S0	S2 <sup>+</sup>	S1 <sup>+</sup>	S0 <sup>+</sup>	Ts2	Ts1	Ts0
			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

Q	Q <sup>+</sup>	T
0	0	0
0	1	1
1	0	1
1	1	0

# 3-bit binary up-counter example (continued)

- Step 4: use K-maps to derive simplified logic eqns for FF inputs

		S2 S1					
		00	01	11	10		
S0	0	0	0	0	0		
	1	0	1	1	0		

$$Ts2 = S1 \cdot S0$$

		S2 S1					
		00	01	11	10		
S0	0	0	0	0	0		
	1	1	1	1	1		

$$Ts1 = S0$$

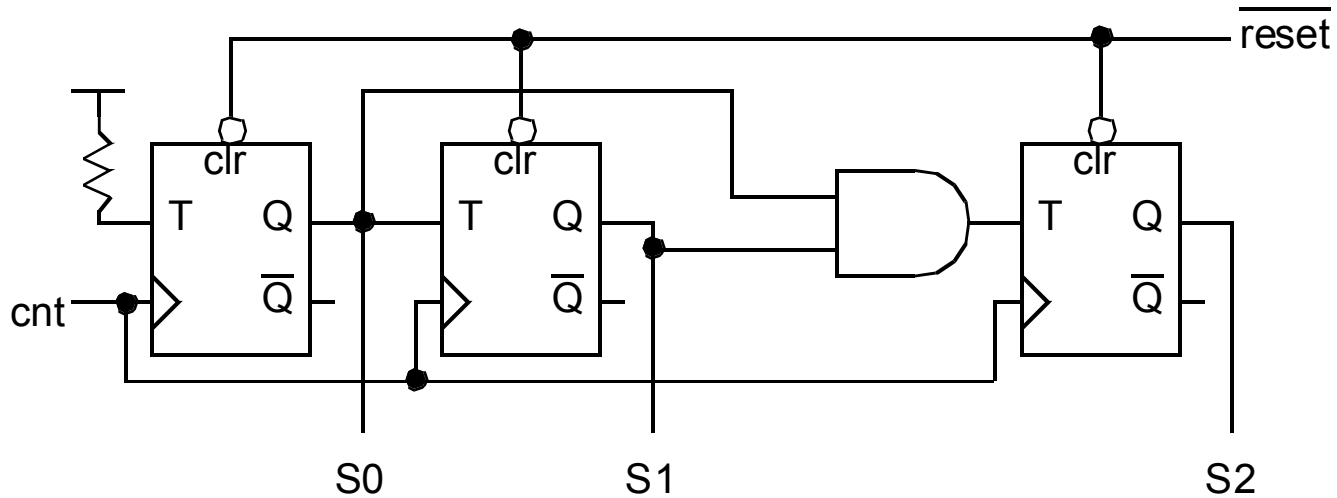
		S2 S1					
		00	01	11	10		
S0	0	1	1	1	1		
	1	1	1	1	1		

$$Ts0 = 1$$

# 3-bit binary up-counter example (continued)

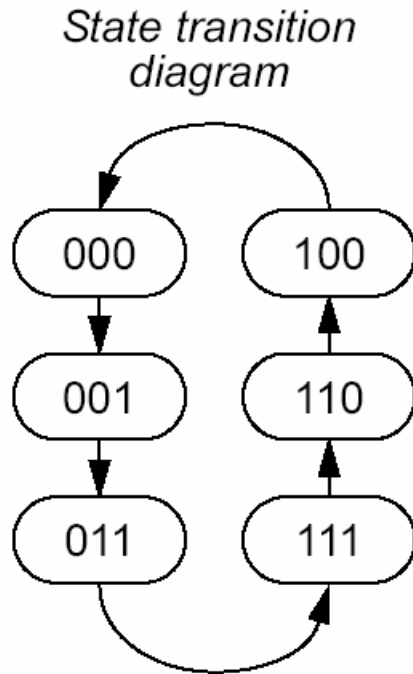
- Step 5: wire it up!

Asynchronous reset  
used for initialization



# another example: 3-bit Johnson counter

- Step 1: describe operation



*State transition table*

Present State			Next State		
S2	S1	S0	S2 <sup>+</sup>	S1 <sup>+</sup>	S0 <sup>+</sup>
0	0	0	0	0	1
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

# another example: 3-bit Johnson counter

---

- Step 2: choose a flip-flop type

Excitation Tables

Q	Q <sup>+</sup>	R	S	J	K	D	T
0	0	X	0	0	X	0	0
0	1	0	1	1	X	1	1
1	0	1	0	X	1	0	1
1	1	0	X	X	0	1	0

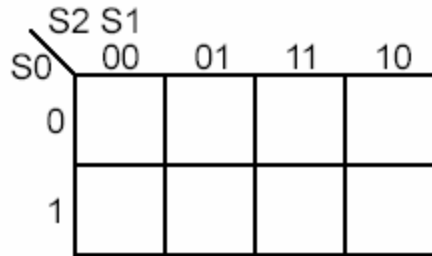
# another example: 3-bit Johnson counter

- Step 3: remap next state into required J-K FF inputs

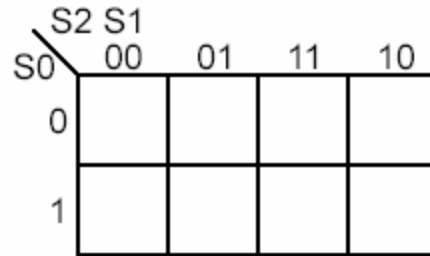
Excitation Table				Present State			Next State			Remapped Next State					
Q	Q <sup>+</sup>	J	K	S2	S1	S0	S2 <sup>+</sup>	S1 <sup>+</sup>	S0 <sup>+</sup>	J2	K2	J1	K1	J0	K0
0	0	0	X	0	0	0	0	0	1	0	X	0	X	1	X
0	1	1	X	0	0	1	0	1	1						
0	1	1	X	0	1	0	X	X	X						
1	0	X	1	0	1	1	1	1	1						
1	1	X	0	1	0	0	0	0	0						
				1	0	1	X	X	X						
				1	1	0	1	0	0						
				1	1	1	1	1	0						

# another example: 3-bit Johnson counter

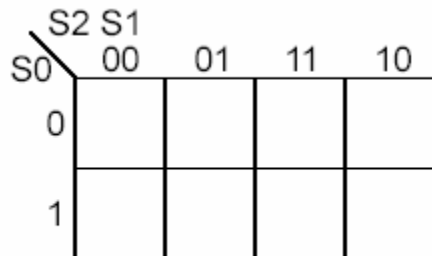
- Step 4: use K-maps to derive simplified logic eqns for FF inputs



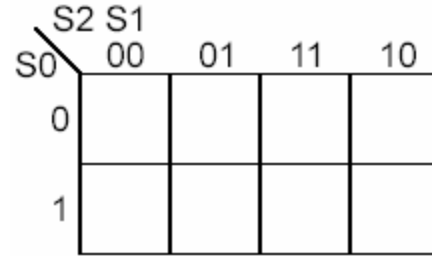
J2 =



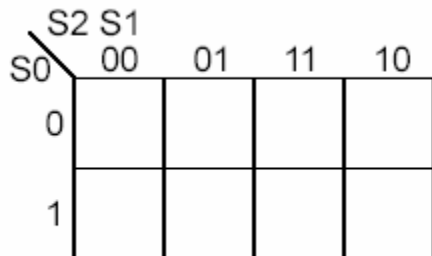
K2 =



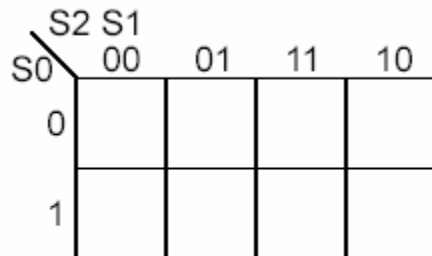
J1 =



K1 =



J0 =



K0 =

•Step 5: Wire it up!

## another example: 3-bit Johnson counter

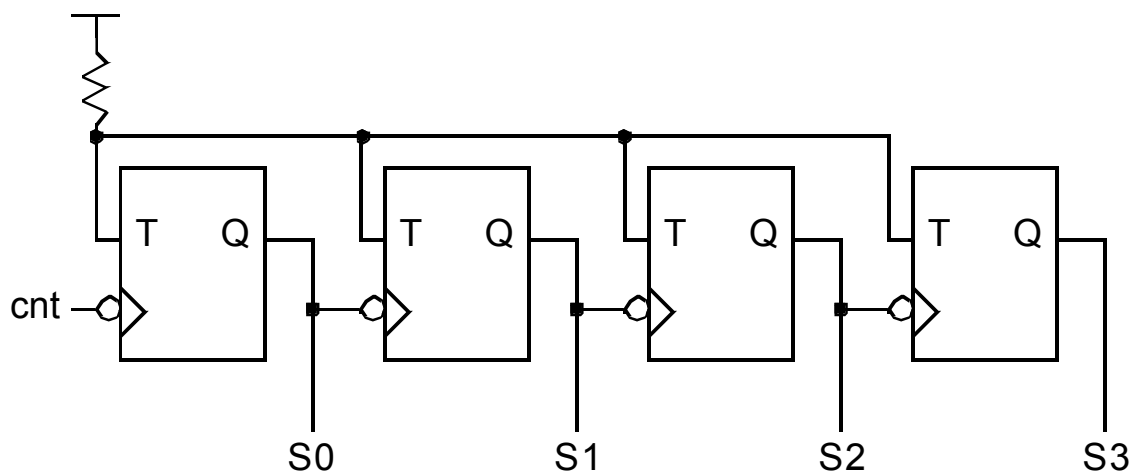
---

- Step 5: wire it up

# Asynchronous-ity in counters

---

- state transitions only through synchronous inputs
- asynchronous inputs for power-on reset/preset only!
- example no-no: ripple counter



# Cascaded Counters

---

- Say you have a pre-designed 4-bit counter  
How do you build a 8-bit (or bigger) counter?
- Solution: 4-bit counters have RCO (*ripple carry out*) pin  
where  
$$\text{RCO} = S_3 \cdot S_2 \cdot S_1 \cdot S_0$$
  
Feed this to enable pin of next significant 4-bit counter.

# General Finite State Machine (FSM) design

---

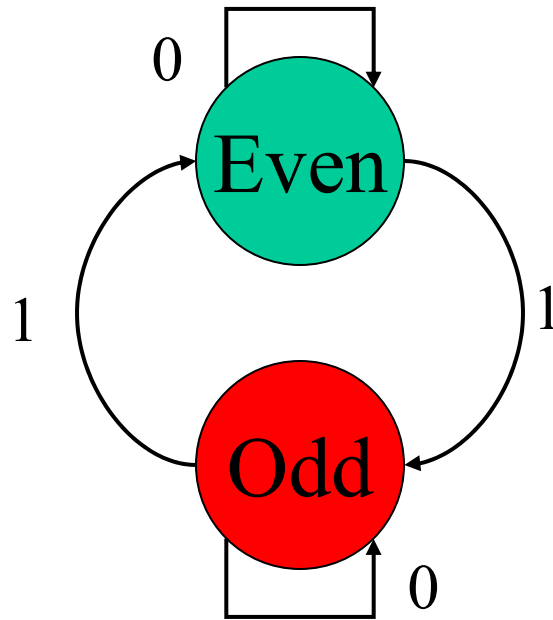
- FSM = sequential logic circuit which can be implemented in a fixed number of states
- basically extend design technique learned for counters

# Mapping Input History

---

- Parity Checkers: Simplest example of general class of *error detecting codes*
- How to build a odd/even parity checker?
  - Odd parity: asserts output when a serial input stream contains an odd number of 1's
  - Even parity: asserts output when it's seen an even number of 1's
- Clearly, output depends on entire input history!

# Building the FSM

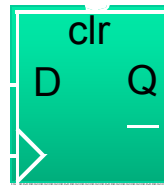


State Diagram

Present State	Input	Next State	Output
Even (0)	0	Even (0)	0
Even (0)	1	Odd (1)	0
Odd (1)	0	Odd (1)	1
Odd (1)	1	Even (0)	1

Next State = Present State XOR Input

Output = Present State



# Basic design approach

---

1. understand the problem
  - make assumptions to complete design specification
  - Identify inputs and outputs
  - draw a block diagram of FSM
  - enumerate possible inputs sequences and system states
2. obtain abstract representation of FSM
  - draw state (transition) diagram
  - or alternative state machine representation
3. perform state minimization (new step)
4. perform state assignment
  - encode states
  - build state transition table
5. choose FF type
  - remap next state into required FF inputs
6. implement
  - simplify next state and output logic equations
  - wire the circuit together

# Design example: simple vending machine

---

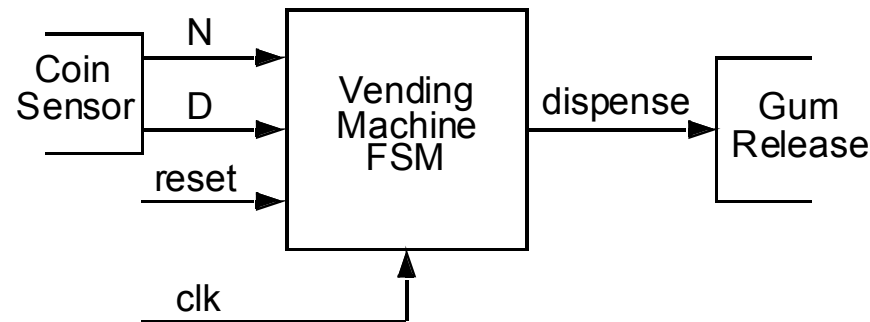
## Step 1: understand the problem

- problem description:
  - vending machine dispenses gum given 15¢ or more
  - machine accepts nickels or dimes (sensor determines coin)
  - machine provides no change
- assumptions:
  - other coins automatically returned
  - external reset applied after gum dispensed
- inputs/outputs:
  - nickel inserted; dime inserted; reset; clock
  - dispense gum

# Design example: simple vending machine

---

- block diagram:



- possible input sequences:
  - N, N, N
  - N, N, D
  - N, D
  - D, N
  - D, D
- output: asserted only after reaching 15¢ or greater

# Design example: simple vending machine

---

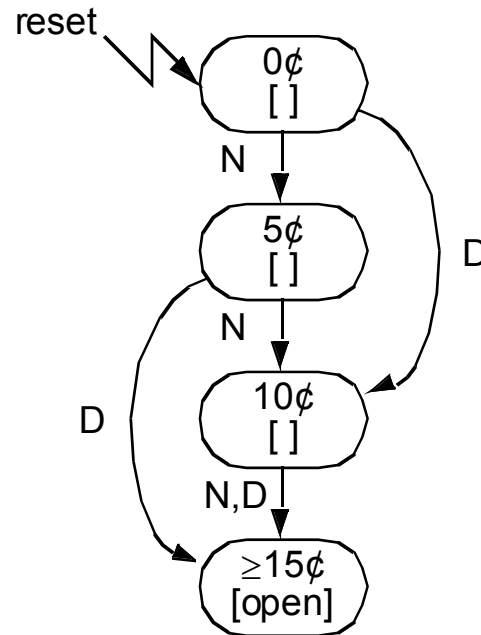
Step 2: obtain abstract representation of FSM

- state diagram — based on tree of input options

# Design example: simple vending machine

---

minimized state diagram (intuitive approach to Step 3)



- alternative state machine representations
  - algorithm state machines – like flowcharts w/ rigorous timing
  - hardware description languages (Verilog, etc)

# Design example: simple vending machine

Step 4: perform state assignment

Present State	Inputs		Next State	Output Dispense
	D	N		
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	X	X
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	≥15¢	0
	1	1	X	X
10¢	0	0	10¢	0
	0	1	≥15¢	0
	1	0	≥15¢	0
	1	1	X	X
≥15¢	X	X	≥15¢	1

4 states requires 2 bits

0¢ ⇒ 00

5¢ ⇒ 01

10¢ ⇒ 10

≥15¢ ⇒ 11

# Design example: simple vending machine

---

## Step 5: choose FF type

- Choose J-K FF  $\Rightarrow$  remap next state into FF inputs

Present State		Inputs		Next State						
Q1	Q0	D	N	Q1 <sup>+</sup>	Q0 <sup>+</sup>	J1	K1	J0	K0	
0	0	0	0	0	0	0	0	X	0	X
		0	1	0	1	1	0	X	1	X
		1	0	1	0	0	1	X	0	X
		1	1	X	X	X	X	X	X	X
0	1	0	0	0	0	1	0	X	X	1
		0	1	1	0	0	1	X	X	1
		1	0	1	1	1	1	X	X	0
		1	1	X	X	X	X	X	X	X
1	0	0	0	1	1	0	X	0	0	X
		0	1	1	1	1	X	0	1	X
		1	0	1	1	1	X	0	1	X
		1	1	X	X	X	X	X	X	X
1	1	X	X	1	1	X	0	X	0	

---

# Step 6: implement

- simplify logic equations for FF inputs and circuit output

D N		Q1Q0			
		00	01	11	10
00	0	0	X	X	
01	0	1	X	X	
11	X	X	X	X	
10	1	1	X	X	

$$J1 = D + Q0 \cdot N$$

D N		Q1Q0			
		00	01	11	10
00	X	X	0	0	
01	X	X	0	0	
11	X	X	X	X	
10	X	X	0	0	

$$K1 = 0$$

D N		Q1Q0			
		00	01	11	10
00	0	X	X	0	
01	1	X	X	1	
11	X	X	X	X	
10	0	X	X	1	

$$J0 = N + Q1 \cdot D$$

D N		Q1Q0			
		00	01	11	10
00	X	0	0	X	
01	X	1	0	X	
11	X	X	X	X	
10	X	0	0	X	

$$K0 = \overline{Q1} \cdot N$$

D N		Q1Q0			
		00	01	11	10
00	0	0	1	0	
01	0	0	1	0	
11	X	X	X	X	
10	0	0	1	0	

$$\text{Dispense} = Q1 \cdot Q0$$

# Gate Implementation

---