

SEE 3243/4243

FSM Modelling & Systematic Realization II

Week 10

- Vending Machine Example
- Finite String Pattern Recognizer
- Traffic Light Controller
- Digital Combination Lock



Finite State Machine Word Problems

- Mapping English Language Description to Formal Specifications
- This Week we'll cover applications of FSM as controller:
 - Two part design: Data Path + Controller
- Four Case Studies:
 - Vending Machine
 - Finite String Pattern Recognizer
 - Traffic Light Controller
 - Digital Combination Lock



Review of Design Steps

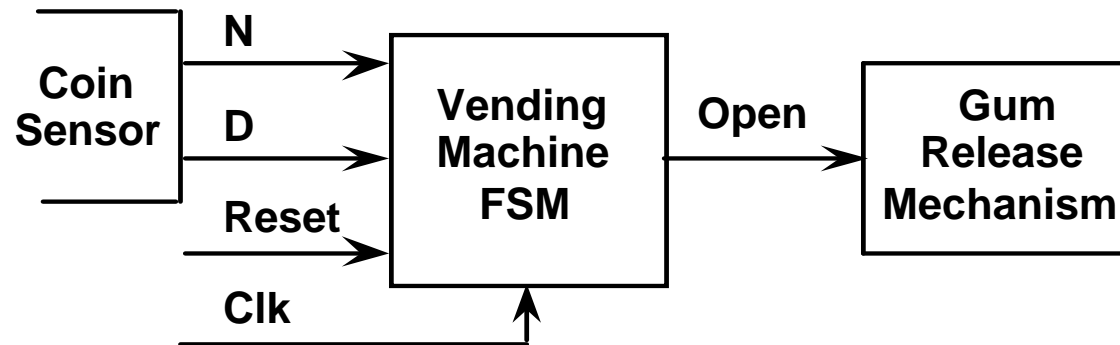
- Obtain specification of the desired circuit.
- Create a state diagram from specification.
- Create a state table from state diagram.
- Perform state minimization.
- Perform state assignment.
- Derive the next-state logic expressions.
- Implement circuit described by logic.

Example: Vending Machine FSM

- General Machine Concept:
 - deliver package of gum after 15 cents deposited
 - single coin slot for dimes, nickels
 - no change

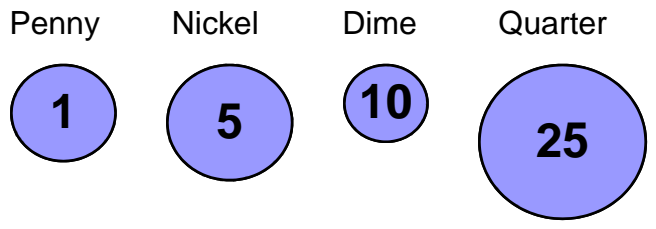
Step 1. Understand the problem: Draw a picture!

Block Diagram



Vending Machine Example

Step 2. Map into more suitable abstract representation

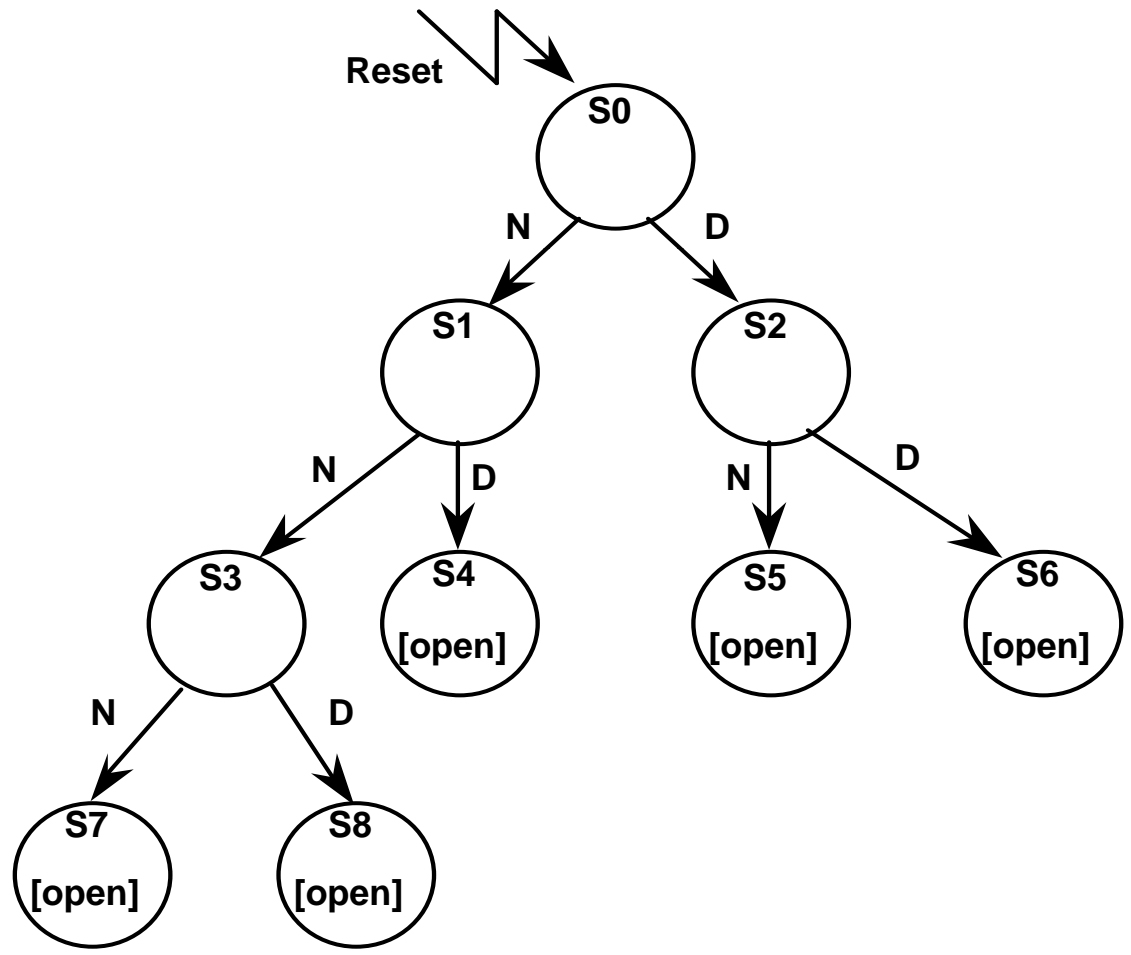


Tabulate typical input sequences
three nickels
nickel, dime
dime, nickel
two dimes
two nickels, dime

Draw state diagram:

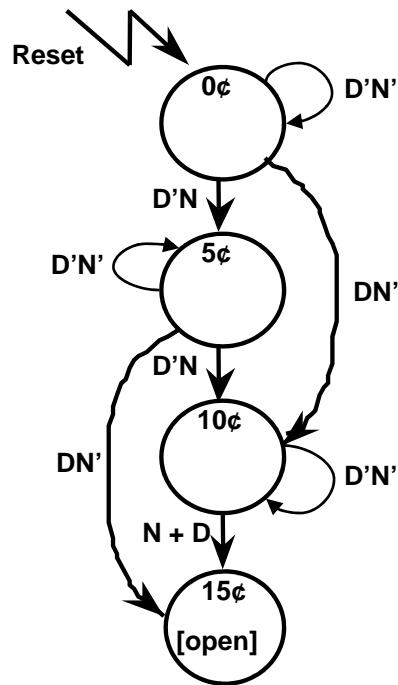
Inputs: N, D, reset

Output: open



Vending Machine Controller

Step 3: State Minimization: reuse states whenever possible



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

We can assume X for Next State actually

Symbolic State Table

Vending Machine Controller

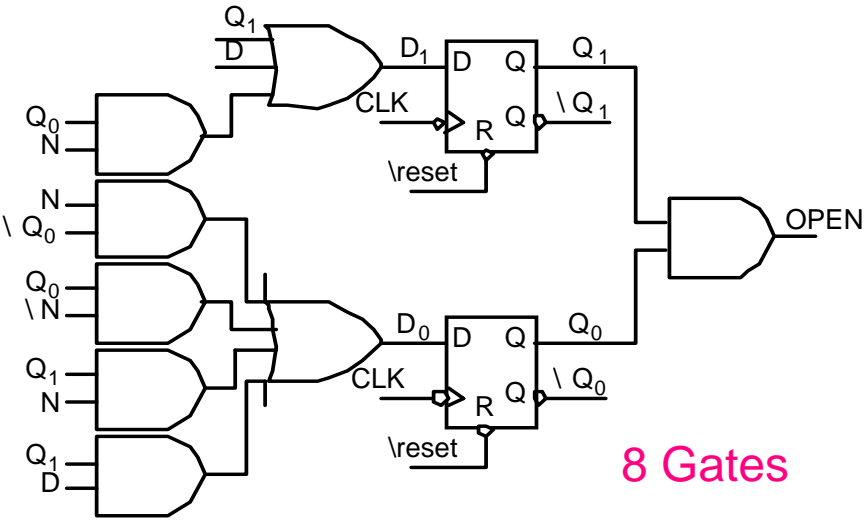
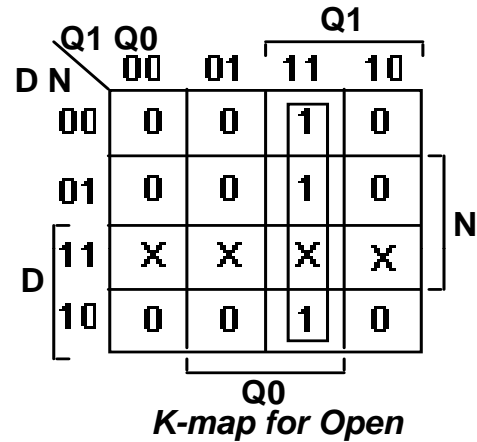
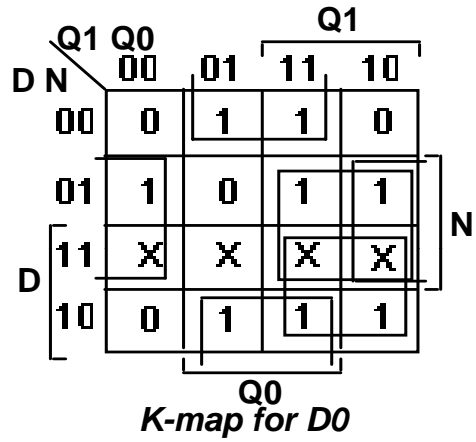
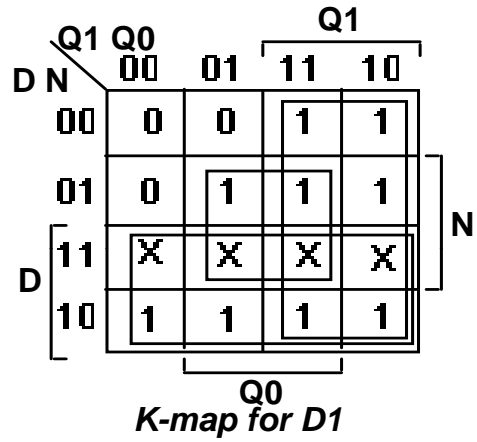
Step 4: State Encoding: “simple” binary easiest to understand

Present State		Inputs		Next State		Output
Q ₁	Q ₀	D	N	D ₁	D ₀	OPEN
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	X	X	0
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	X	X	0
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	1	1	0
1	1	X	X	1	1	1

Encoded State Table

Vending Machine Example

Step 5. Choose FFs for implementation: D FF easiest to use



Step 6. Implementation

$$D1 = Q1 + D + Q_0 \cdot N$$

$$D0 = N \cdot Q_0' + Q_0 \cdot N' + Q_1 \cdot N + Q_1 \cdot D$$

$$OPEN = Q_1 \cdot Q_0$$

8 Gates

Vending Machine Controller

Step 5. Choosing FF for Implementation: Use JKFF for kicks

- In FPGA/VLSI, DFF is most efficient.
- JKFF must be constructed using DFF plus a few gates.
- Run away from JKFF if possible!
- Different story when using discrete chips (BUT WHO DOES?)

Excitation Table			
Q	Q+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Present State		Inputs		Next State		J ₁	K ₁	J ₀	K ₀
Q ₁	Q ₀	D	N	Q ₁ ⁺	Q ₀ ⁺				
0	0	0	0	0	0	0	X	0	X
		0	1	0	1	0	X	1	X
		1	0	1	0	1	X	0	X
		1	1	X	X	X	X	X	X
0	1	0	0	0	1	0	X	X	0
		0	1	1	0	1	X	X	1
		1	0	1	1	1	X	X	0
		1	1	X	X	X	X	X	X
1	0	0	0	1	0	X	0	0	X
		0	1	1	1	X	0	1	X
		1	0	1	1	X	0	1	X
		1	1	X	X	X	X	X	X
1	1	X	X	1	1	X	0	X	0

Remapped encoded state transition table

Vending Machine Example

Step 6. Implementation

	Q1	Q0	Q1		
D N	00	01	11	10	
00	0	0	X	X	N
01	0	1	X	X	
11	X	X	X	X	
10	1	1	X	X	
		Q0			

K-map for J1

	Q1	Q0	Q1		
D N	00	01	11	10	
00	X	X	0	0	N
01	X	X	0	0	
11	X	X	X	X	
10	X	X	0	0	
		Q0			

K-map for K1

	Q1	Q0	Q1		
D N	00	01	11	10	
00	0	X	X	0	N
01	1	X	X	1	
11	X	X	X	X	
10	0	X	X	1	
		Q0			

K-map for J0

	Q1	Q0	Q1		
D N	00	01	11	10	
00	X	0	0	X	N
01	X	1	0	X	
11	X	X	X	X	
10	X	0	0	X	
		Q0			

K-map for K0

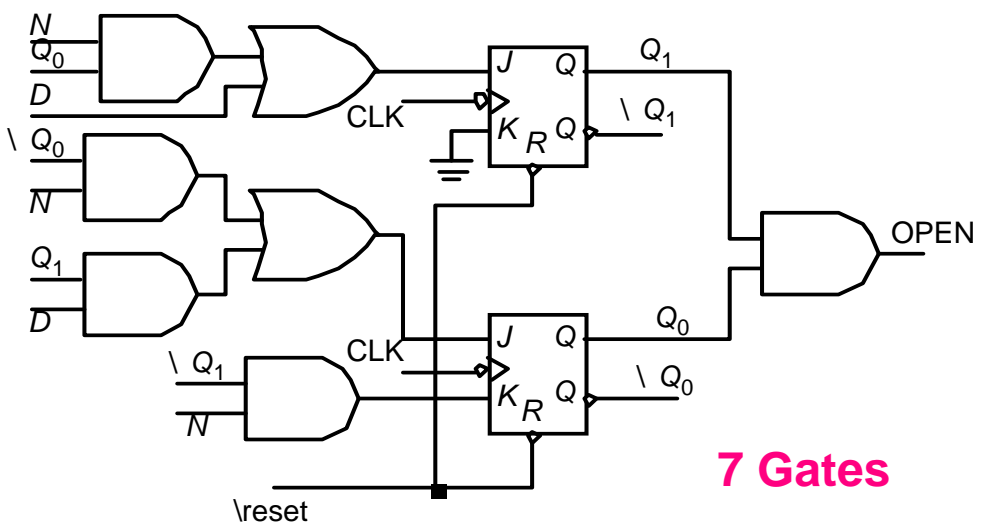
$$J1 = D + Q_0 \cdot N$$

$$K1 = 0$$

$$K0 = Q_0' \cdot N + Q_1 \cdot D$$

$$K1 = Q_1' \cdot N$$

$$OPEN = Q_1 \cdot Q_0$$

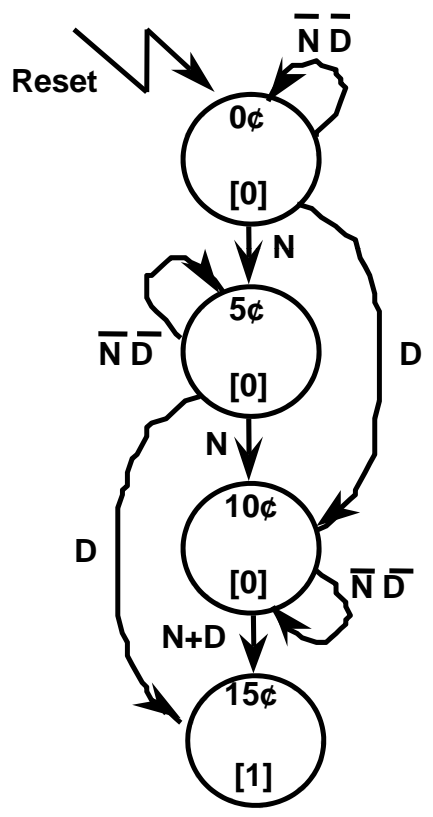


7 Gates

Moore & Mealy State Diagram Equivalents

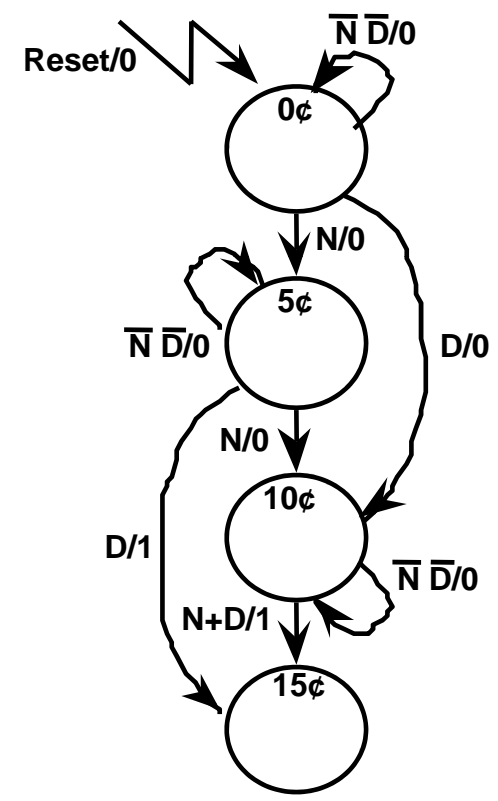
Back to Vending Machine Example

Moore Machine



Outputs are associated with State

Mealy Machine



Outputs are associated with Transitions

Finite String Pattern Recognizer

- A finite string recognizer has one input (X) and one output (Z). The output is asserted whenever the input sequence ...010... has been observed, as long as the sequence 100 has never been seen.

- **Step 1. Understanding the problem statement**

- Sample input/output behavior:

X: 00101010010...

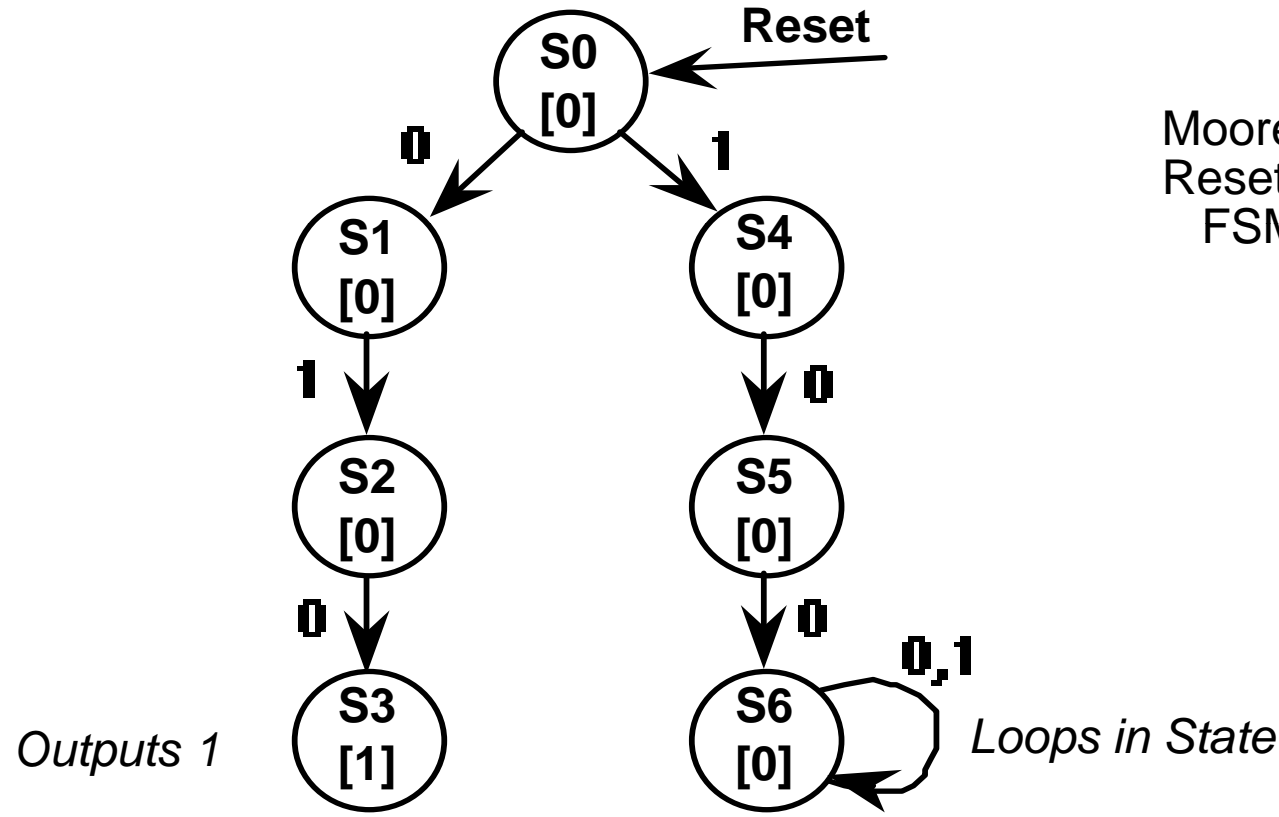
Z: 00010101000...

X: 11011010010...

Z: 00000001000...

Finite String Pattern Recognizer

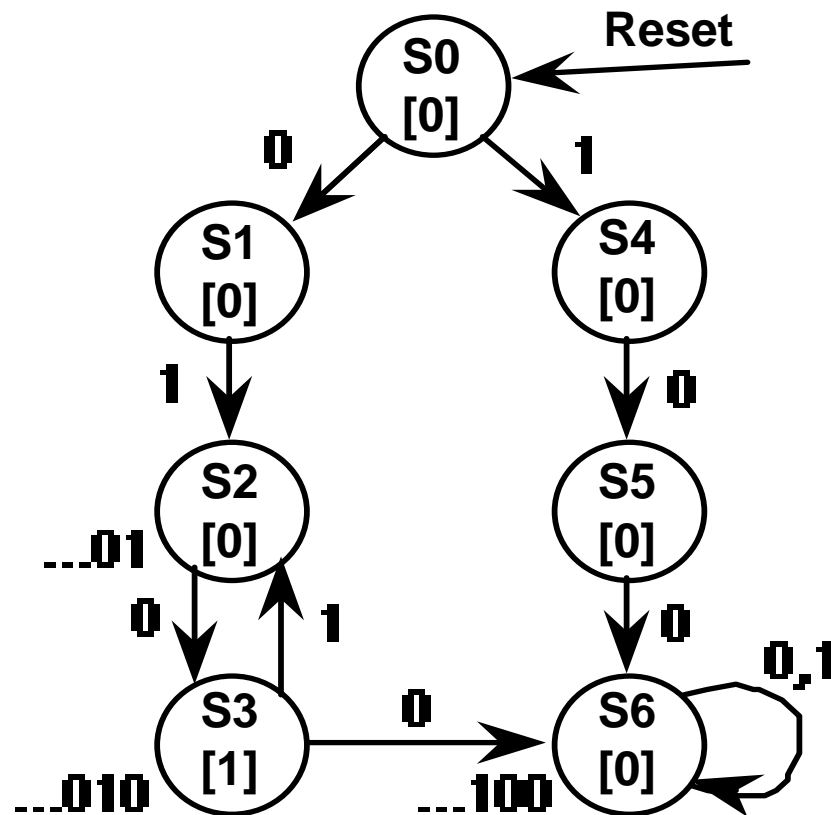
Step 2. Draw State Diagrams/ASM Charts for the strings that must be recognized. i.e., 010 and 100.



Moore State Diagram
Reset signal places
FSM in S0

Finite String Pattern Recognizer

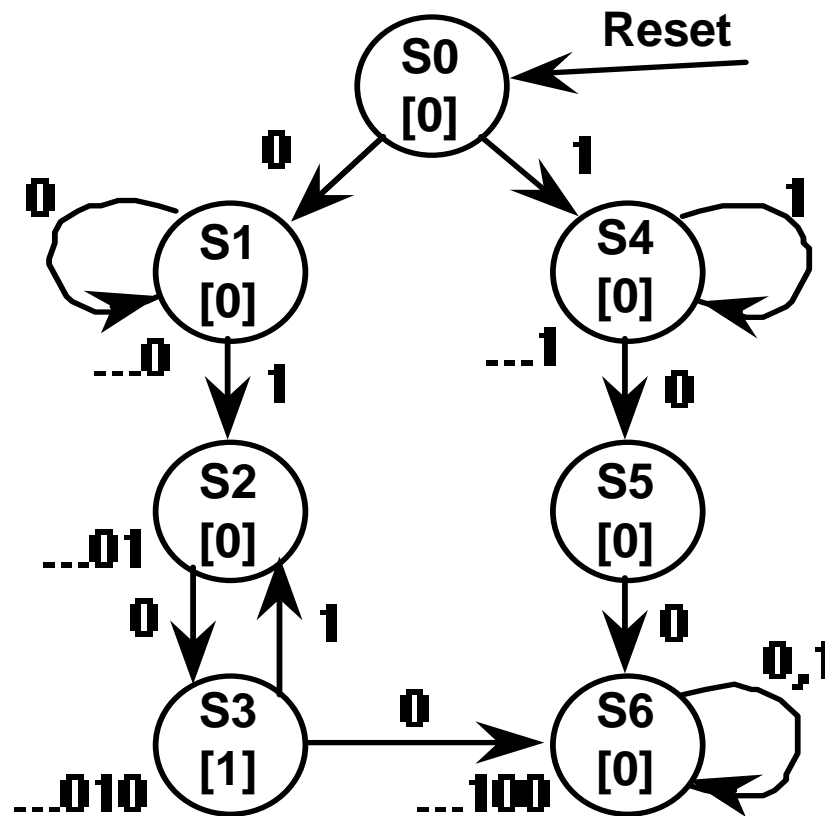
Exit conditions from state S3: have recognized ...010
 if next input is 0 then have ...0100!
 if next input is 1 then have ...0101 = ...01 (state S2)



Finite String Pattern Recognizer

Exit conditions from S1: recognizes strings of form ...0 (no 1 seen)
loop back to S1 if input is 0

Exit conditions from S4: recognizes strings of form ...1 (no 0 seen)
loop back to S4 if input is 1

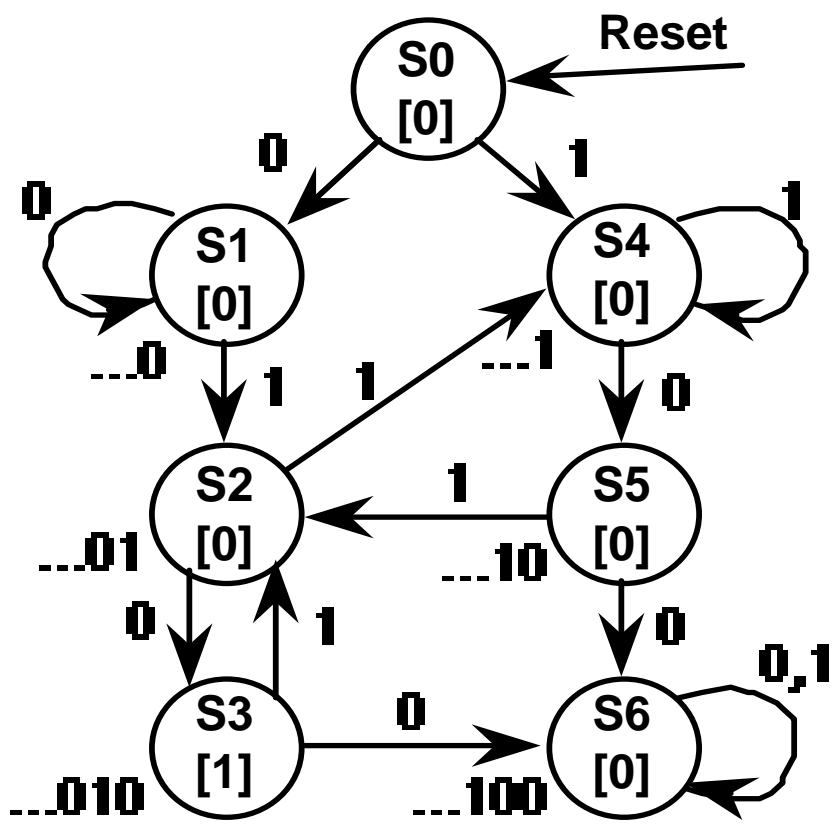


Finite String Pattern Recognizer

S2, S5 with incomplete transitions

S2 = ...01; If next input is 1, then string could be prefix of (01)1(00)
S4 handles just this case!

S5 = ...10; If next input is 1, then string could be prefix of (10)1(0)
S2 handles just this case!



Final State Diagram



Finite String Pattern Recognizer

■ Review of Process:

- Write down sample inputs and outputs to understand specification
- Write down sequences of states and transitions for the sequences to be recognized
- Add missing transitions; reuse states as much as possible
- Verify I/O behavior of your state diagram to insure it functions like the specification

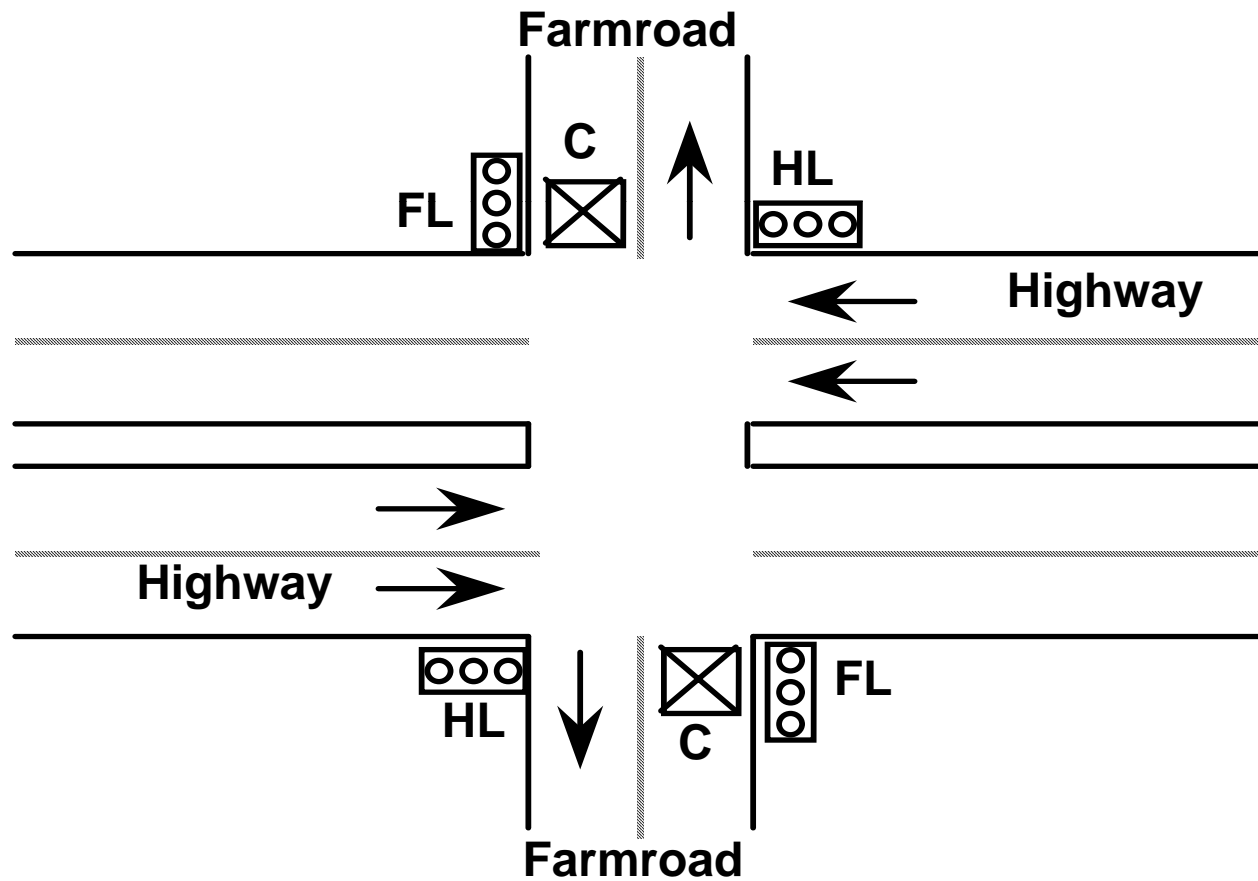


Traffic Light Controller

- A busy highway is intersected by a little used farmroad. Detectors C sense the presence of cars waiting on the farmroad. With no car on farmroad, light remain green in highway direction. If vehicle on farmroad, highway lights go from Green to Yellow to Red, allowing the farmroad lights to become green. These stay green only as long as a farmroad car is detected but never longer than a set interval.
- When these are met, farm lights transition from Green to Yellow to Red, allowing highway to return to green. Even if farmroad vehicles are waiting, highway gets at least a set interval as green.
- Assume you have an interval timer that generates a short time pulse (TS) and a long time pulse (TL) in response to a set (ST) signal. TS is to be used for timing yellow lights and TL for green lights.

Traffic Light Controller

Picture of Highway/Farmroad Intersection:



Traffic Light Controller

- Tabulation of Inputs and Outputs:

Input Signal

reset

C

TS

TL

Description

place FSM in initial state

detect vehicle on farmroad

short time interval expired

long time interval expired

Output Signal

HG, HY, HR

FG, FY, FR

ST

Description

assert green/yellow/red highway lights

assert green/yellow/red farmroad lights

start timing a short or long interval

- Tabulation of Unique States: Some light configuration imply others

State

S0

S1

S2

S3

Description

Highway green (farmroad red)

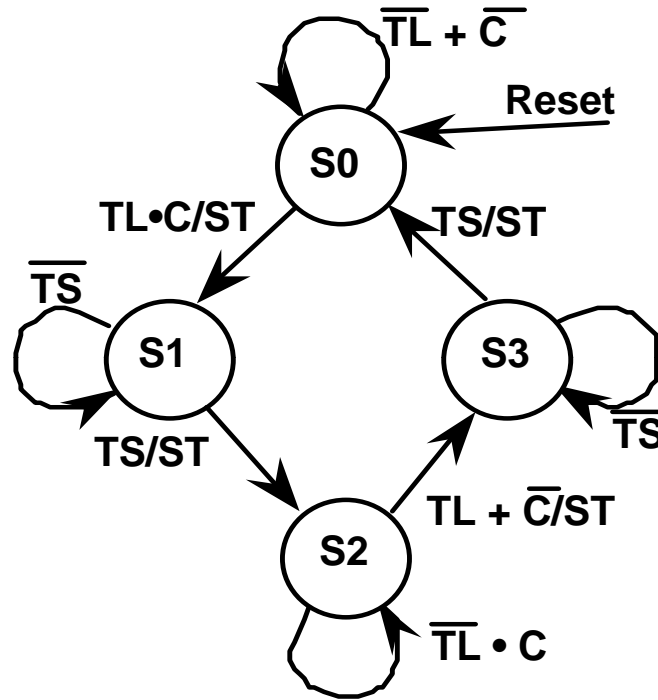
Highway yellow (farmroad red)

Farmroad green (highway red)

Farmroad yellow (highway red)

Traffic Light Controller

State diagram:



S0: HG

S1: HY

S2: FG

S3: FY

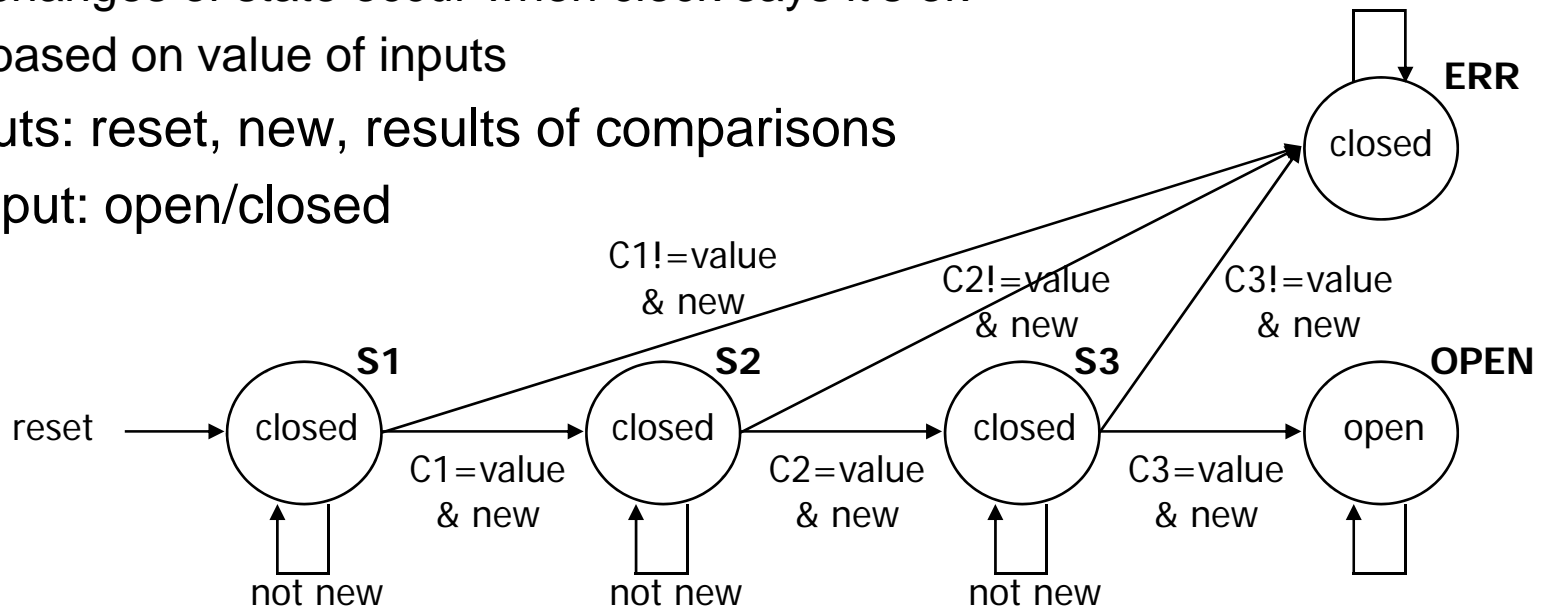
Door combination lock

- General specs:
 - punch in 3 values in sequence and the door opens; if there is an error the lock must be reset; once the door opens the lock must be reset
- Inputs:
 - sequence of input values, reset
- Outputs:
 - door open/close
- Memory:
 - must remember combination or always have it available as an input

Door combination lock: initial STD

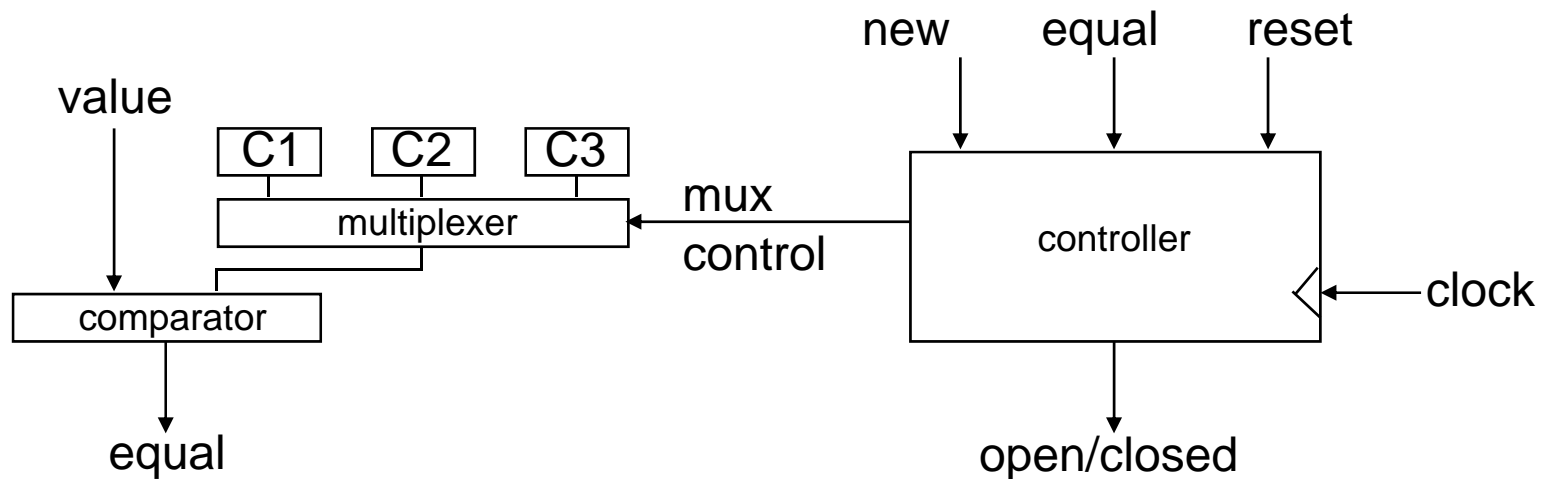
State diagram

- States: 5 states
 - represent point in execution of machine
 - each state has outputs
- Transitions: 6 from state to state, 5 self transitions, 1 global
 - changes of state occur when clock says it's ok
 - based on value of inputs
- Inputs: reset, new, results of comparisons
- Output: open/closed



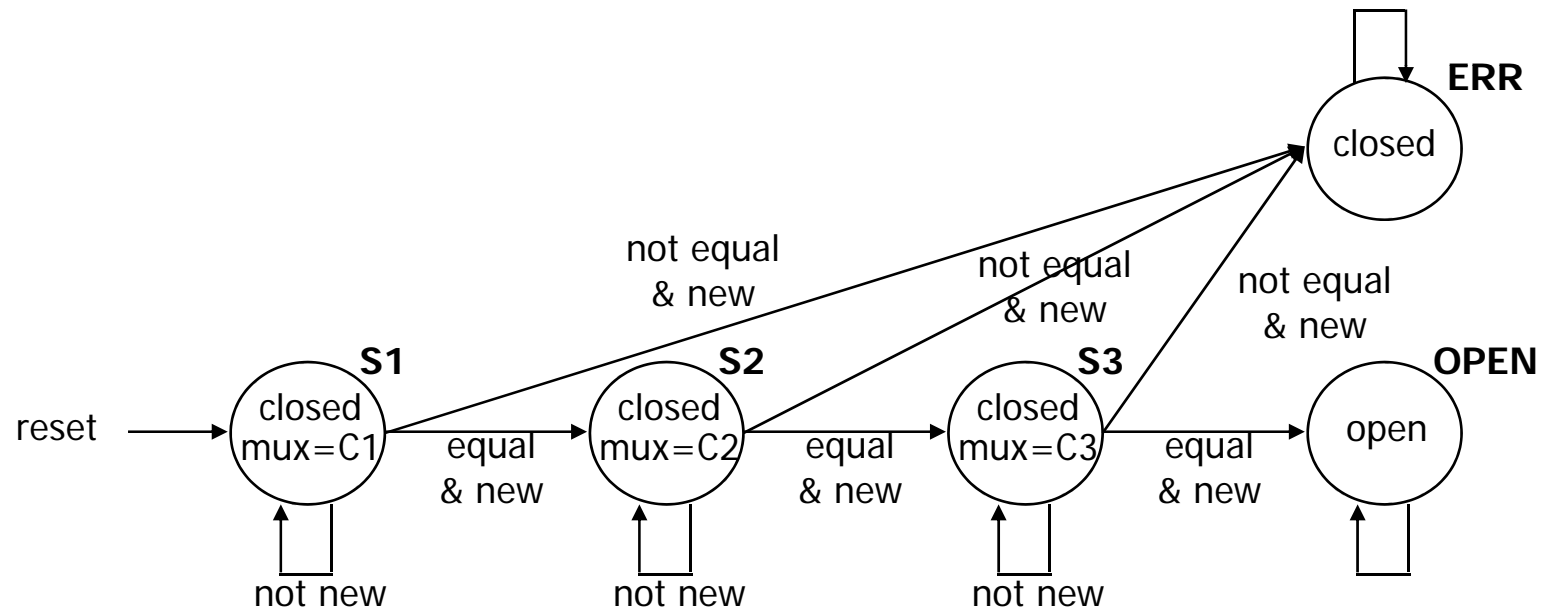
Door combination lock : data-path vs. control

- Internal structure
 - data-path
 - storage for combination
 - comparators
 - control
 - finite-state machine controller
 - control for data-path
 - state changes controlled by clock



Door combination lock: Final STD

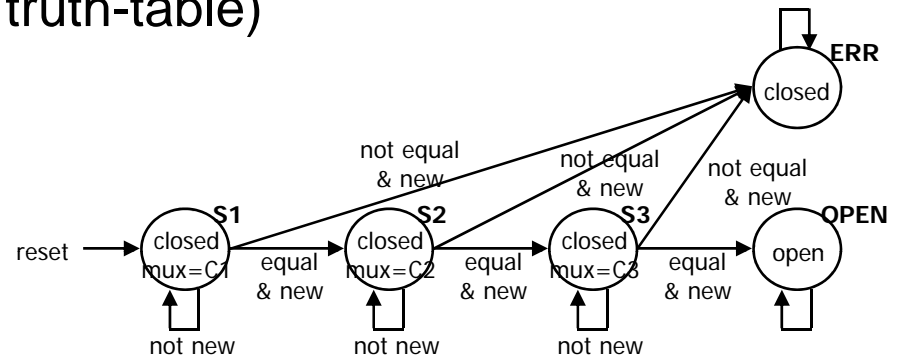
- Finite-state machine
 - refine state diagram to include internal structure



Door combination lock: STT

- Finite-state machine

- generate state table (much like a truth-table)



reset	new	equal	state	next state	mux	open/closed
1	-	-	-	S1	C1	closed
0	0	-	S1	S1	C1	closed
0	1	0	S1	ERR	-	closed
0	1	1	S1	S2	C2	closed
0	0	-	S2	S2	C2	closed
0	1	0	S2	ERR	-	closed
0	1	1	S2	S3	C3	closed
0	0	-	S3	S3	C3	closed
0	1	0	S3	ERR	-	closed
0	1	1	S3	OPEN	-	open
0	-	-	OPEN	OPEN	-	open
0	-	-	ERR	ERR	-	closed



Door combination lock: encoding

■ Encode state table

- state can be: S1, S2, S3, OPEN, or ERR
 - needs at least 3 bits to encode: 000, 001, 010, 011, 100
 - and as many as 5: 00001, 00010, 00100, 01000, 10000
 - choose 4 bits: 0001, 0010, 0100, 1000, 0000
- output mux can be: C1, C2, or C3
 - needs 2 to 3 bits to encode
 - choose 3 bits: 001, 010, 100
- output open/closed can be: open or closed
 - needs 1 or 2 bits to encode
 - choose 1 bits: 1, 0

Door combination lock: encoding

■ Encode state table

- state can be: S1, S2, S3, OPEN, or ERR
 - choose 4 bits: 0001, 0010, 0100, 1000, 0000
- output mux can be: C1, C2, or C3
 - choose 3 bits: 001, 010, 100
- output open/closed can be: open or closed
 - choose 1 bits: 1, 0

reset	new	equal	state	next state	mux	open/closed
1	-	-	-	0001	001	0
0	0	-	0001	0001	001	0
0	1	0	0001	0000	-	0
0	1	1	0001	0010	010	0
0	0	-	0010	0010	010	0
0	1	0	0010	0000	-	0
0	1	1	0010	0100	100	0
0	0	-	0100	0100	100	0
0	1	0	0100	0000	-	0
0	1	1	0100	1000	-	1
0	-	-	1000	1000	-	1
0	-	-	0000	0000	-	0

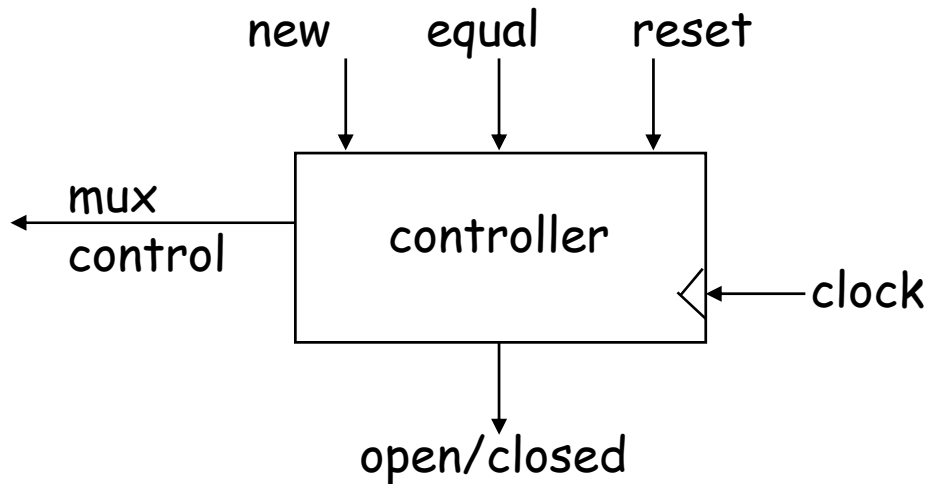
good choice of encoding!

mux is identical to last 3 bits of state

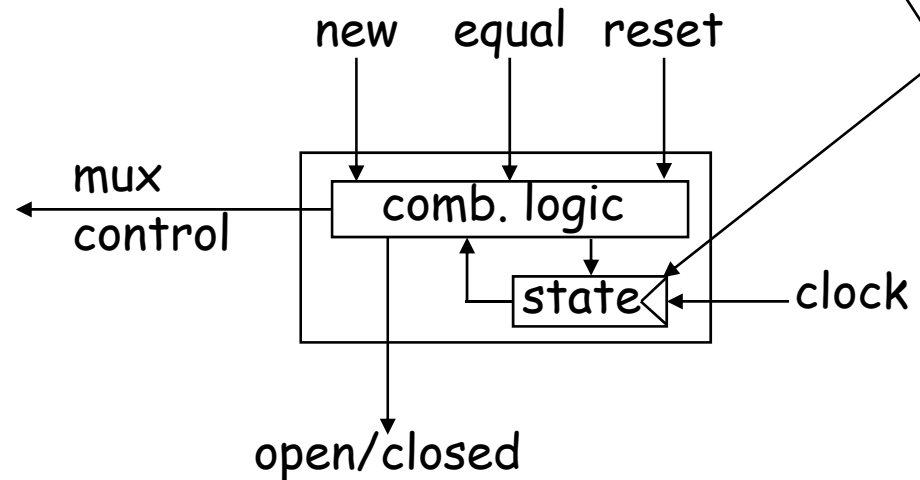
open/closed is identical to first bit of state

Door combination lock: controller implementation

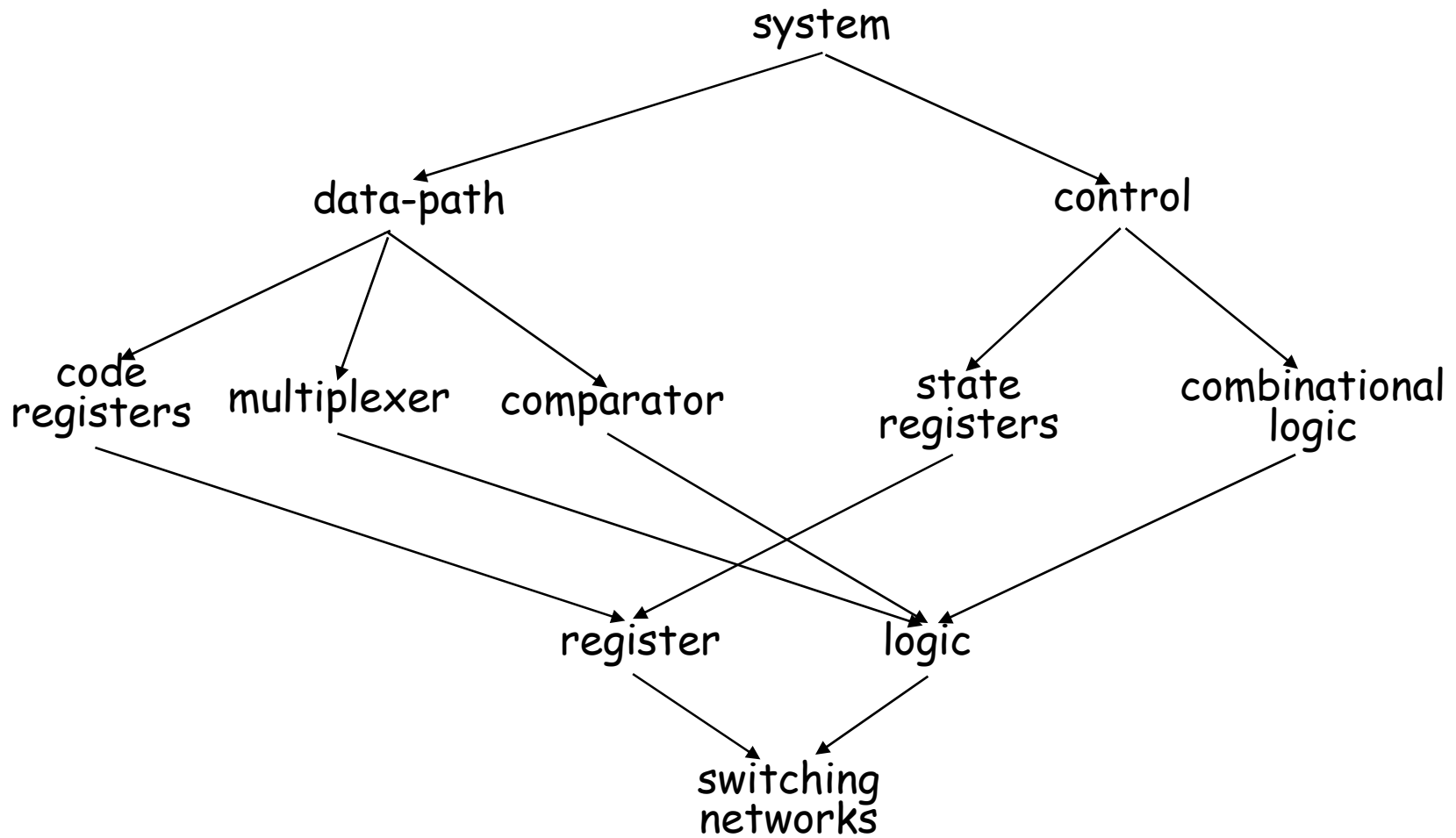
- Implementation of the controller



special circuit element, called a register, for remembering inputs when told to by clock



Design hierarchy





Chapter Review

- Word Problems
 - understand I/O behavior; draw diagrams
 - enumerate states for the "goal"; expand with error conditions
 - reuse states whenever possible
- First Two Steps of the Six Step Procedure for FSM Design
 - understanding the problem
 - abstract representation of the FSM