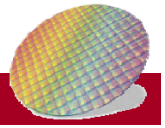
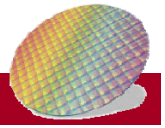
FSM Design



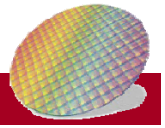
FSM Design procedure

- Step 1: Understand the problem
- Step 2: Obtain an abstract representation of FSM
- Step 3:Perform state minimization
- Step 4: Perfrom assignment
- Step 5: Choose flip-flop types for implementing the FSM's state
  - D-FF or JK-FF or others
- Step 6: Implement the FSM



The vending machine problem

- Design a vending machine
- The vending machine delivers a package of gun after it has received 15 cents in coins. The machine has a single coin slot that accepts nickels (5 cents) and dimes (10 cents), one coin at a time.
- A mechanical sensor indicates to the control whether a dime or a nickel has been inserted into the coin slot. The controller's output causes a single package of gum to be released down a chute to the customer.
- 中文解釋: 設計一台自動販賣機. 當你投入15cents 之後, 口香糖就會掉下來
- 自動販賣機只有一個硬幣投入孔, 一次只能投入一個 5cent 或是10 cent的硬幣
- 在美國5cent 的硬幣叫 nickel,在美國 10 cent 的硬幣叫 dime,



Basic Design Approach

Example: **Vending Machine FSM**

General Machine Concept:

deliver package of gum after 15 cents deposited

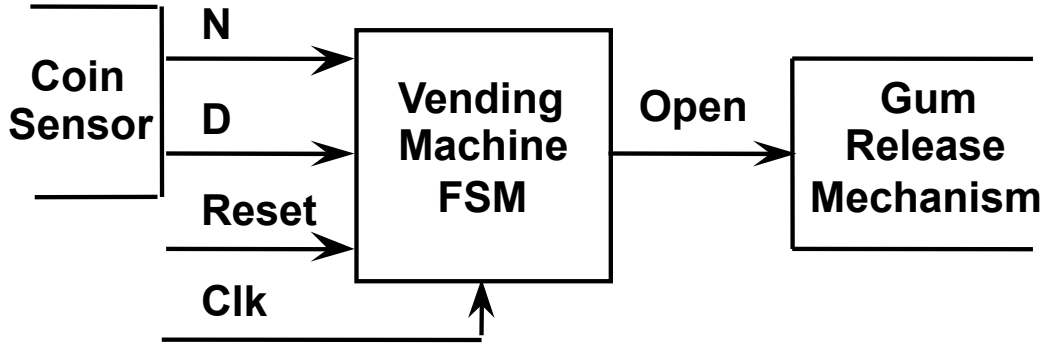
single coin slot for dimes (10 cents), nickels (5 cents)

no change

Step 1. ***Understand the problem:***

Draw a picture!

Block Diagram



Vending Machine Example

Step 2. ***Map into more suitable abstract representation***

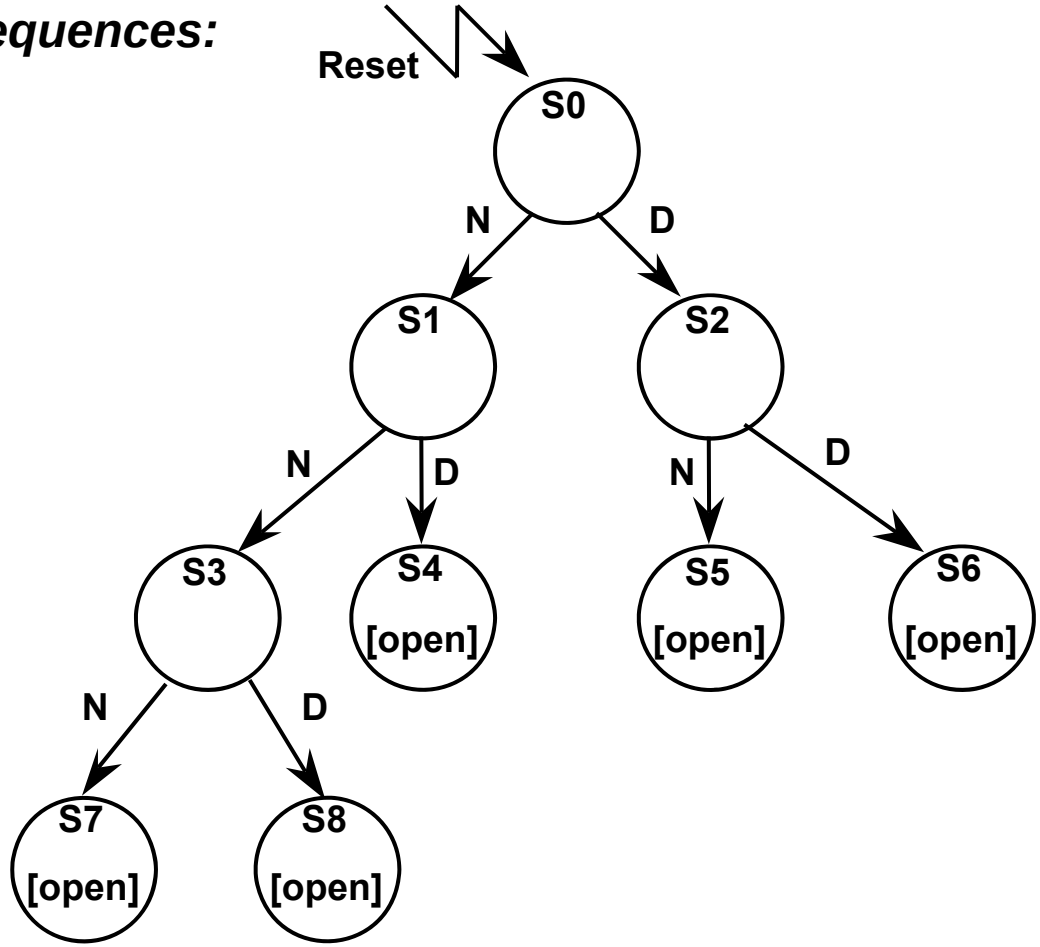
Tabulate typical input sequences:

three nickels
nickel, dime
dime, nickel
two dimes
two nickels, dime

Draw state diagram:

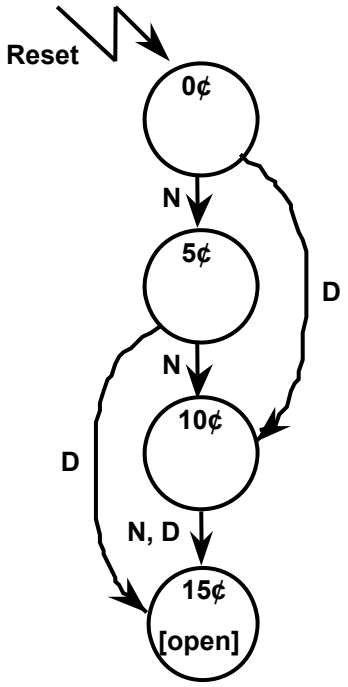
Inputs: N, D, reset

Output: open



Vending Machine Example

Step 3: State Minimization



reuse states whenever possible

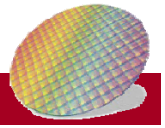
Present State	Inputs D	Inputs N	Next State	Output Open
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	X	X
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	X	X
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	X	X
15¢	X	X	15¢	1

Symbolic State Table

Vending Machine Example

Step 4: State Encoding

Present State Q_1	Q_0	Inputs D	N	Next State D_1	D_0	Output Open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	X	X	X
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	X	X	X
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	X	X	X
1	1	0	0	1	1	1
		0	1	1	1	1
		1	0	1	1	1
		1	1	X	X	X

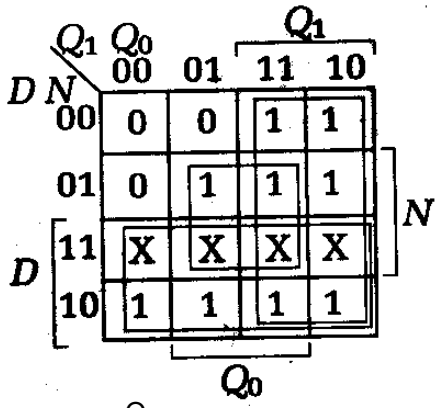


Vending Machine Example

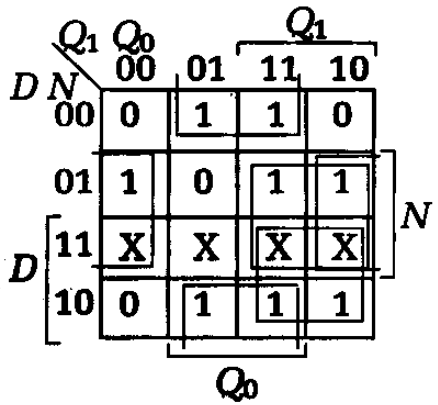
Step 5. ***Choose FFs for implementation***

DFF easiest to use

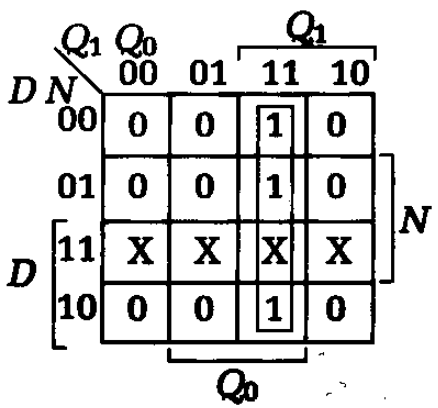
D1

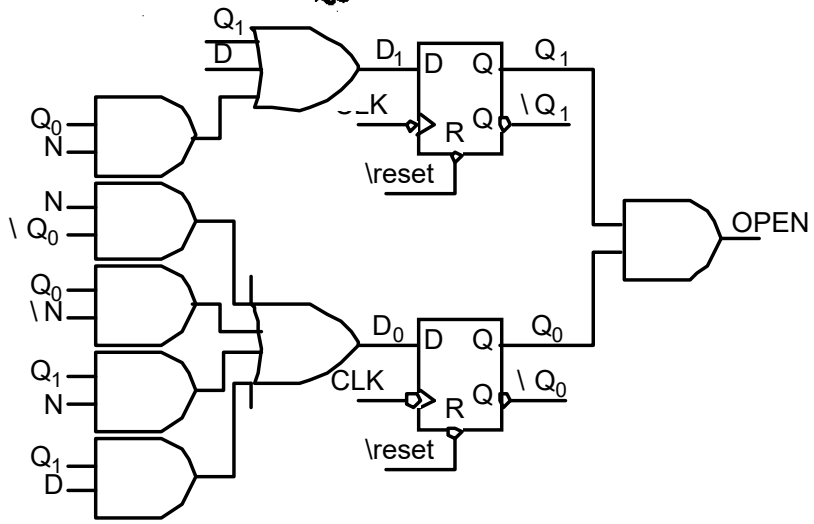


D0



Open





D1 = Q1 + D + Q0 N

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

OPEN = Q1 $\overline{Q0}$

Vending Machine Example

Step 5. ***Choosing FF for Implementation***

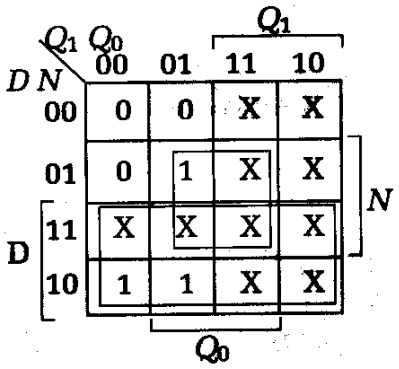
J-K FF

Present State Q_1	Q_0	Inputs D	N	Next State D_1	D_0	J_1	K_1	J_0	K_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	0	0	1	1	X	0	X	0
		0	1	1	1	X	0	X	0
		1	0	1	1	X	0	X	0
		1	1	X	X	X	X	X	X

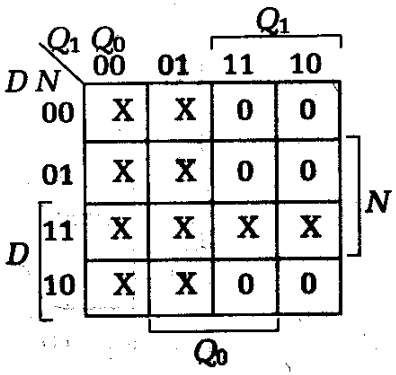
Remapped encoded state transition table

Vending Machine Example

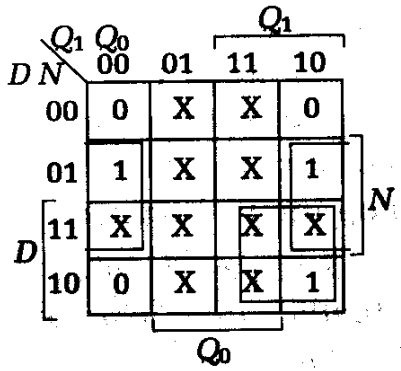
Step 6 Implementation:



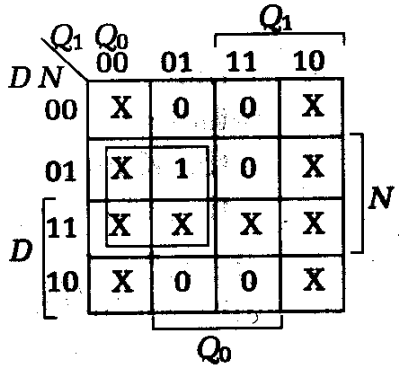
K-map for J_1



K-map for K_1



K-map for J_0



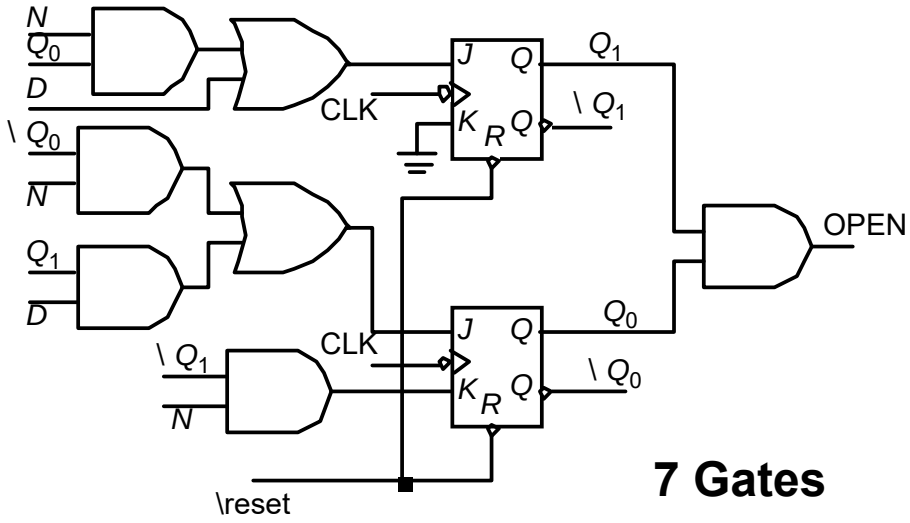
K-map for K_0

J1 = D + Q0 N

K1 = 0

J0 = $\overline{Q0}$ N + Q1 D

K0 = $\overline{Q1}$ N



7 Gates

Implement Vending Machine using Verilog code

```
module VM(clk, reset, N, D, open);
input clk, reset, N, D;
output open;
reg [1:0] state;

assign open = (state==2'd3);   // have 15 cent, open
always@(posedge clk) begin

  if(reset) state <= 0;
  else if (N||D) begin              // coin is coming
    case (state)
    0: state <= {D,N};              // if D = 1 go to state2, if N = 1 go to state1
    1: state <= state + {D,N};      // if D = 1 go to state3, if N = 1 go to state2
    2: state <= 2'd3;               // go to state3
    3:;                             // no operate
    endcase
  end
end
endmodule
```