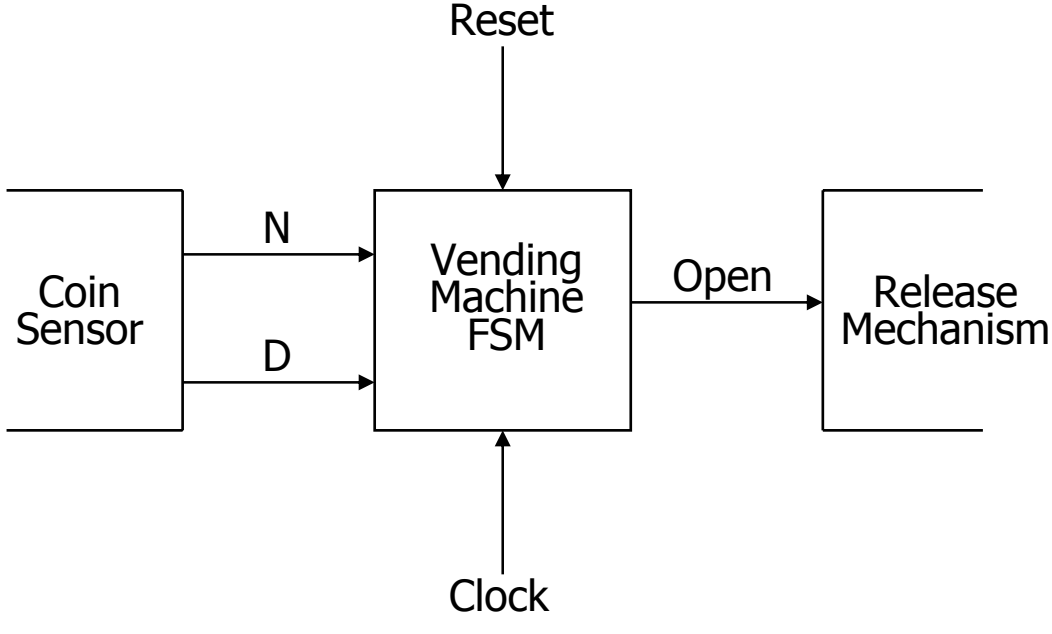
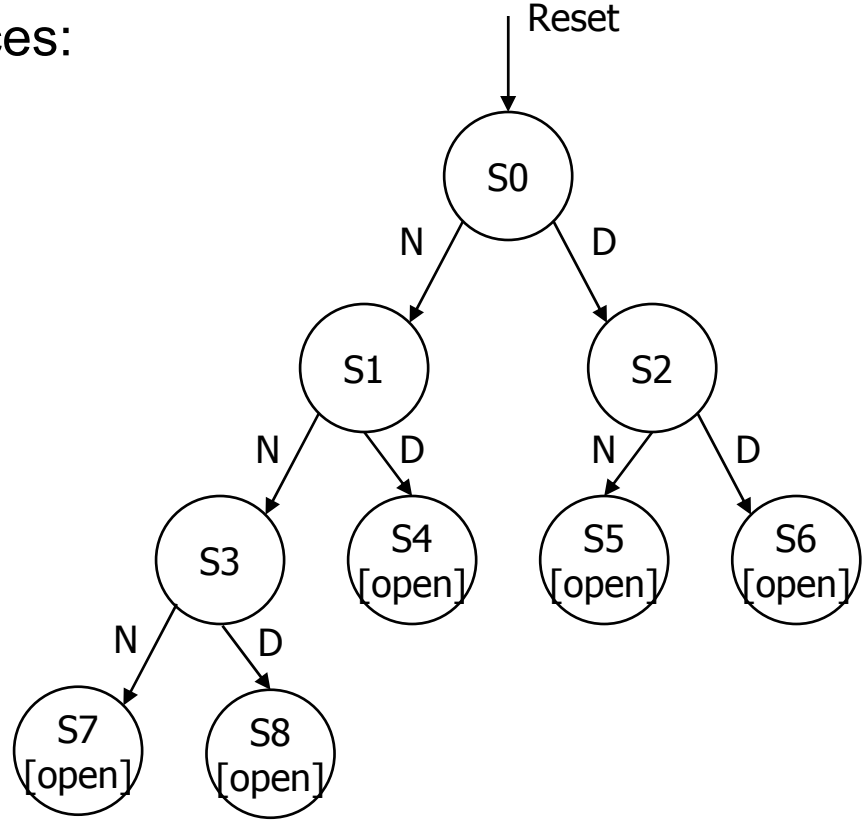
Example: vending machine

- Release item after 15 cents are deposited
- Single coin slot for dimes, nickels
- No change



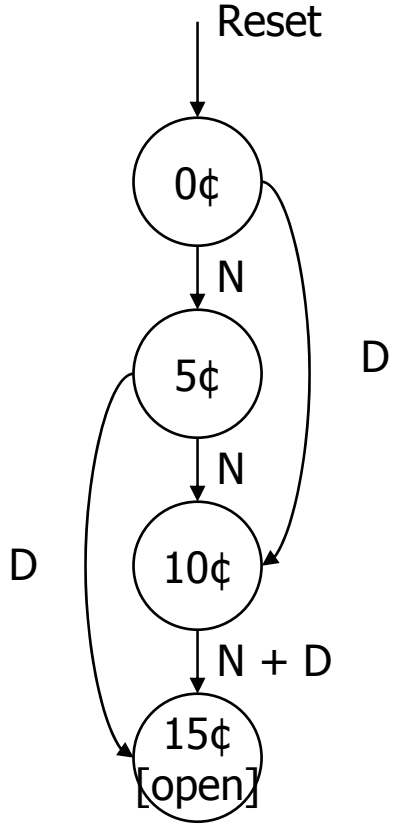
Example: vending machine (cont'd)

- Suitable abstract representation
  - tabulate typical input sequences:
    - 3 nickels
    - nickel, dime
    - dime, nickel
    - two dimes
  - draw state diagram:
    - inputs: N, D, reset
    - output: open chute
  - assumptions:
    - assume N and D asserted for one cycle
    - each state has a self loop for N = D = 0 (no coin)



Example: vending machine (cont'd)

- Minimize number of states - reuse states whenever possible



present state	inputs D	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
	0	1	15¢	0
	1	0	15¢	0
	1	1	–	–
15¢	–	–	15¢	1

symbolic state table

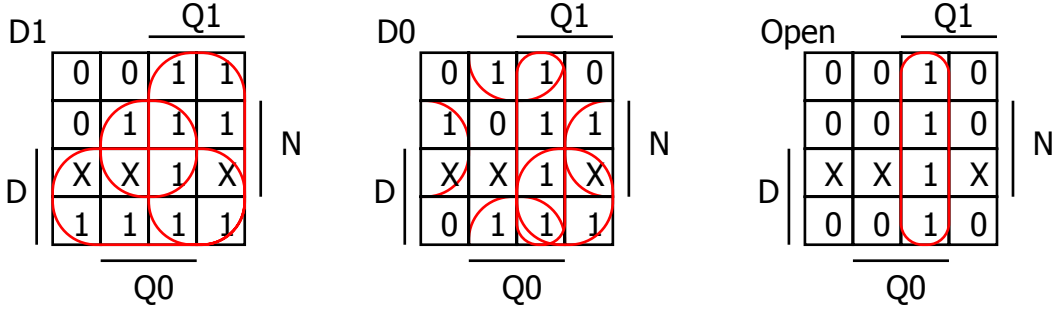
Example: vending machine (cont'd)

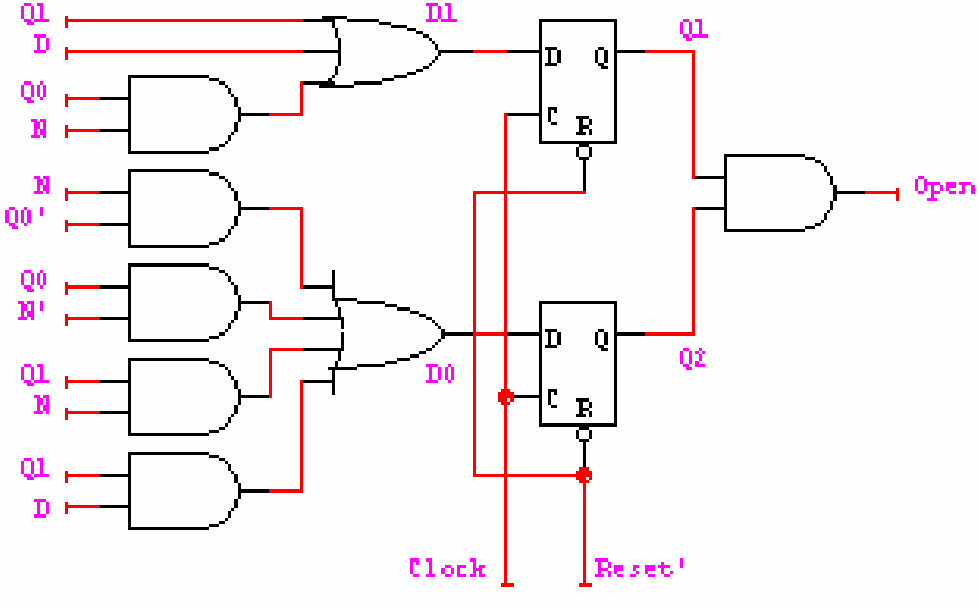
- Uniquely encode states

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

Example: Moore implementation

- Mapping to logic





D1 = Q1 + D + Q0 N

D0 = Q0′ N + Q0 N′ + Q1 N + Q1 D

OPEN = Q1 Q0

Example: vending machine (cont'd)

- One-hot encoding

present state				inputs		next state				output
Q3	Q2	Q1	Q0	D	N	D3	D2	D1	D0	open
0	0	0	1	0	0	0	0	0	1	0
				0	1	0	0	1	0	0
				1	0	0	1	0	0	0
				1	1	-	-	-	-	-
0	0	1	0	0	0	0	0	1	0	0
				0	1	0	1	0	0	0
				1	0	1	0	0	0	0
				1	1	-	-	-	-	-
0	1	0	0	0	0	0	1	0	0	0
				0	1	1	0	0	0	0
				1	0	1	0	0	0	0
				1	1	-	-	-	-	-
1	0	0	0	-	-	1	0	0	0	1

D0 = Q0 D' N'

D1 = Q0 N + Q1 D' N'

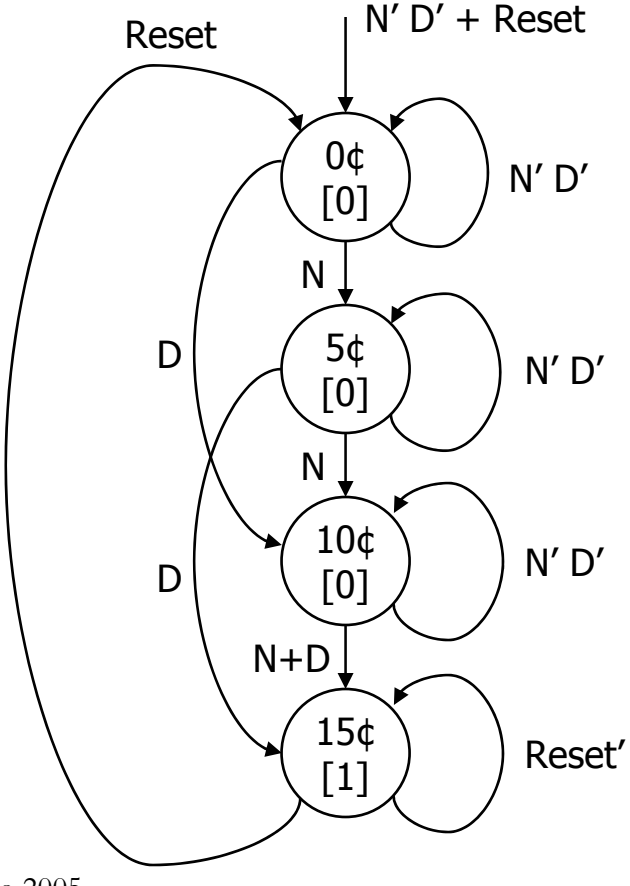
D2 = Q0 D + Q1 N + Q2 D' N'

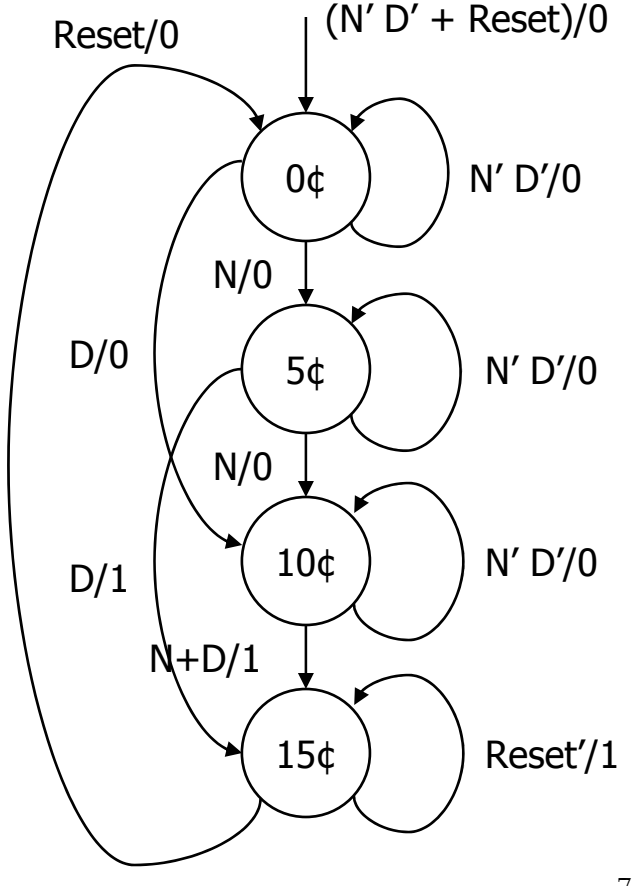
D3 = Q1 D + Q2 D + Q2 N + Q3

OPEN = Q3

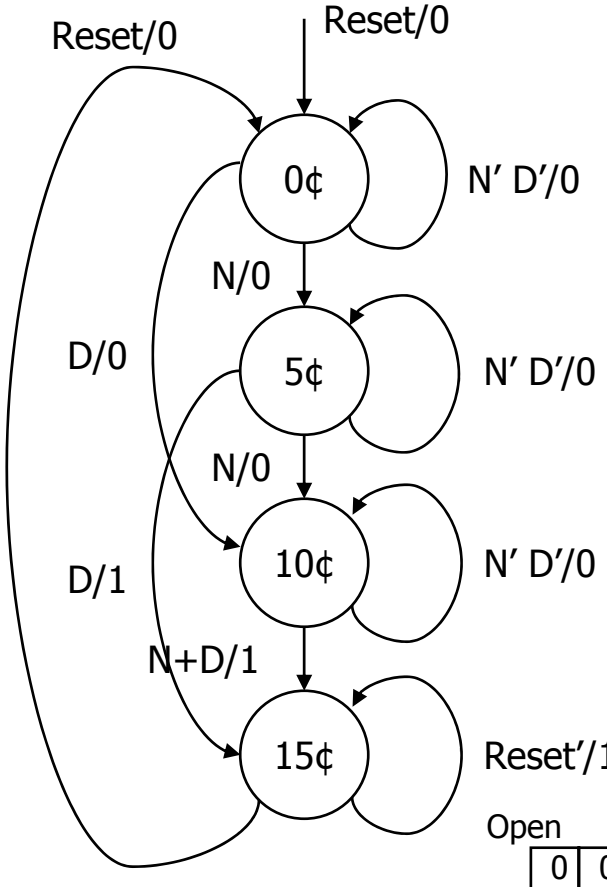
Equivalent Mealy and Moore state diagrams

- Moore machine
  - outputs associated with state
- Mealy machine
  - outputs associated with transitions

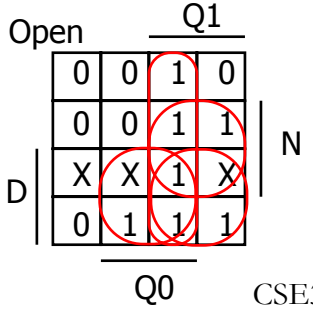




Example: Mealy implementation



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	1
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	1
		1	0	1	1	1
		1	1	–	–	–
1	1	–	–	1	1	1



D0 = Q0′N + Q0N′ + Q1N + Q1D

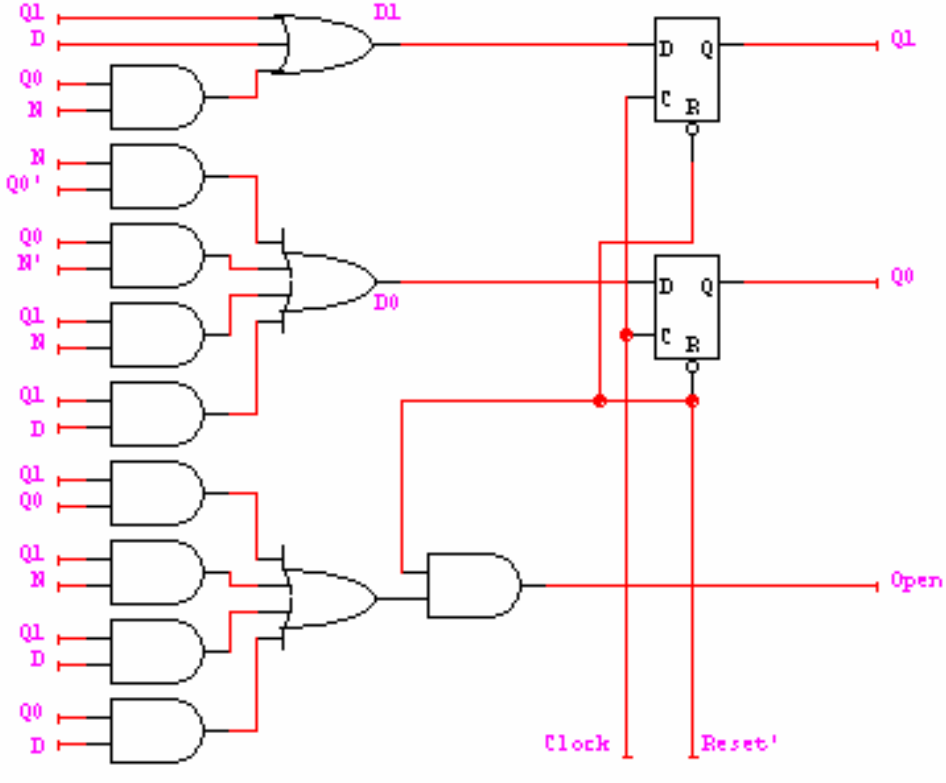
D1 = Q1 + D + Q0N

OPEN = Q1Q0 + Q1N + Q1D + Q0D

Example: Mealy implementation

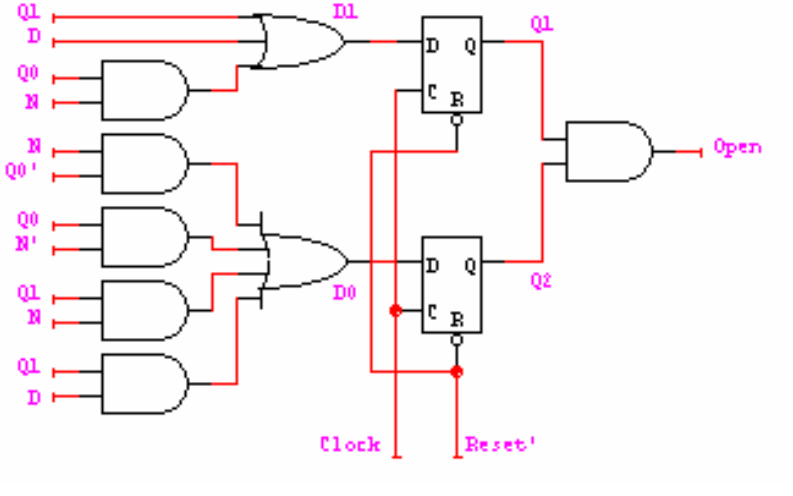
D0 = Q0'N + Q0N' + Q1N + Q1D
D1 = Q1 + D + Q0N
OPEN = Q1Q0 + Q1N + Q1D + Q0D

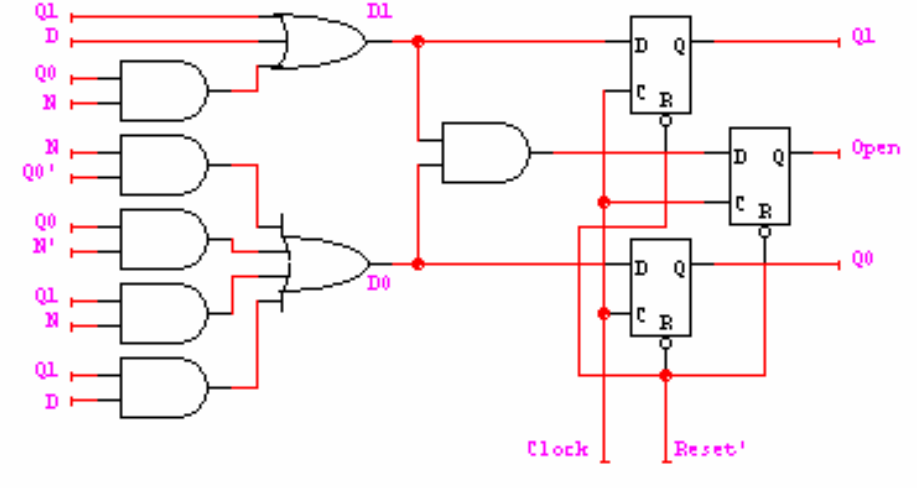
make sure OPEN is 0 when reset
– by adding AND gate



Vending machine: Moore to synch. Mealy

- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 change in Moore implementation
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D
- Implementation now looks like a synchronous Mealy machine
  - it is common for programmable devices to have FF at end of logic



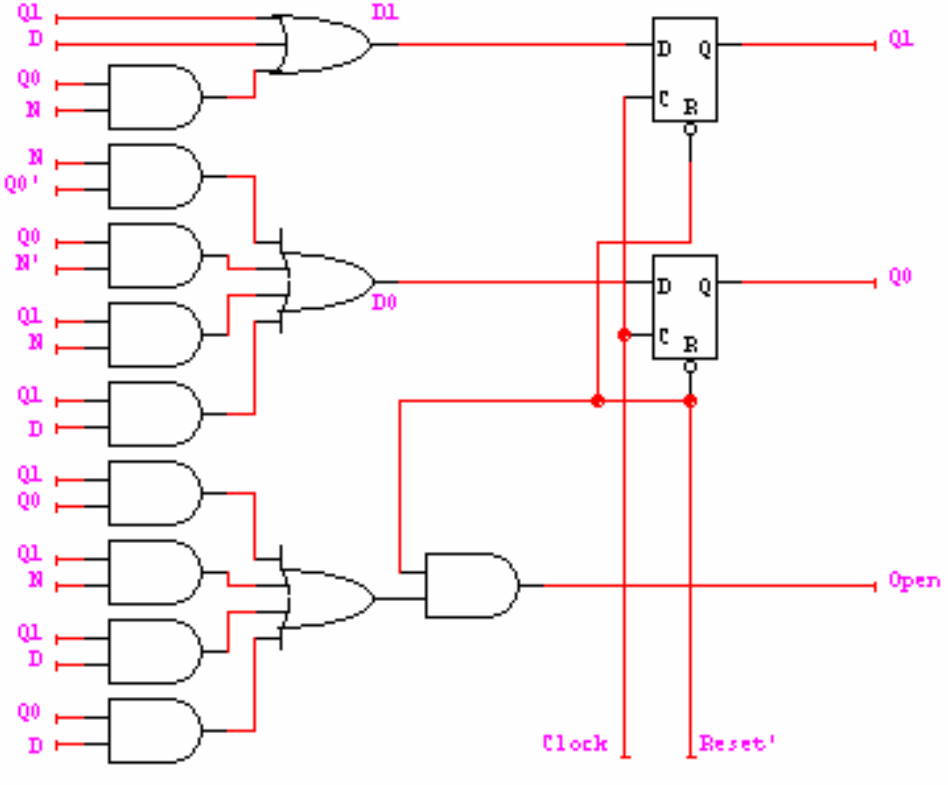


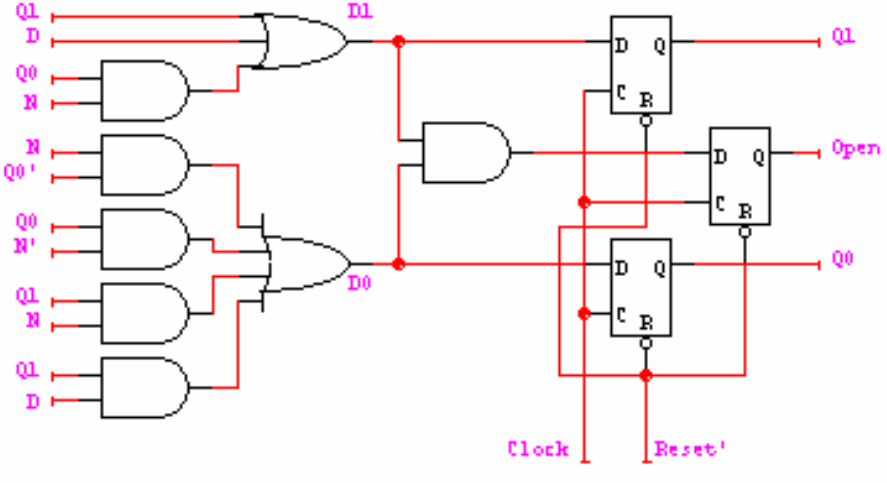
Vending machine: Mealy to synch. Mealy

- OPEN.d = Q1Q0 + Q1N + Q1D + Q0D
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D

Open.d	Q1			
D	0	0	1	0
	0	0	1	1
	1	0	1	1
	0	1	1	1
		Q0		N

Open.d	Q1			
D	0	0	1	0
	0	0	1	1
	X	X	1	X
	0	1	1	1
		Q0		N



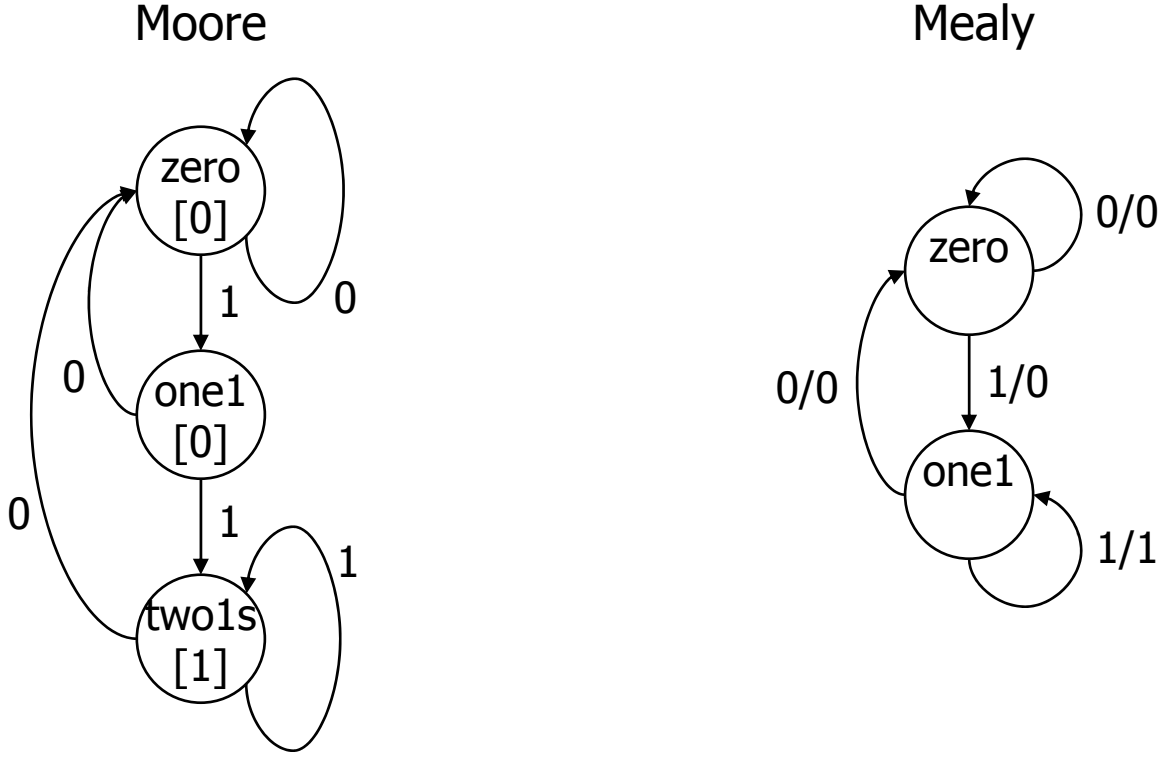


Hardware Description Languages and Sequential Logic

- Flip-flops
  - representation of clocks - timing of state changes
  - asynchronous vs. synchronous
- FSMs
  - structural view (FFs separate from combinational logic)
  - behavioral view (synthesis of sequencers – not in this course)
- Data-paths = data computation (e.g., ALUs, comparators) + registers
  - use of arithmetic/logical operators
  - control of storage elements

Example: reduce-1-string-by-1

- Remove one 1 from every string of 1s on the input

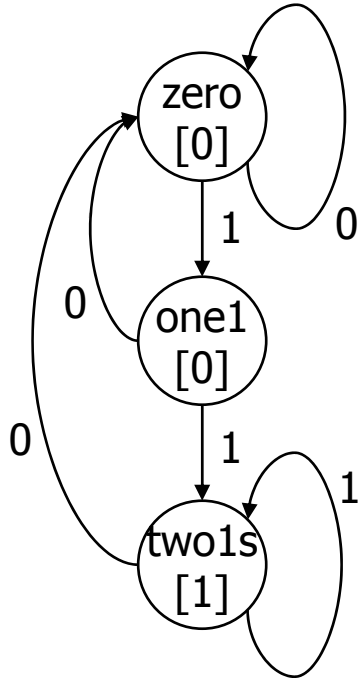


Verilog FSM - Reduce 1s example

- Moore machine

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;

  parameter zero  = 2'b00;
  parameter one1  = 2'b01;
  parameter two1s = 2'b10;

  reg out;
  reg [2:1] state;        // state variables
  reg [2:1] next_state;

  always @(posedge clk)
    if (reset) state = zero;
    else       state = next_state;
```

state assignment
(easy to change,
if in one place)



Moore Verilog FSM (cont'd)

```
always @(in or state)

  case (state)
    zero:
   // last input was a zero
    begin
      if (in) next_state = one1;
      else    next_state = zero;
    end
     one1:
   // we've seen one 1
    begin
      if (in) next_state = two1s;
      else    next_state = zero;
    end
     two1s:
   // we've seen at least 2 ones
    begin
      if (in) next_state = two1s;
      else    next_state = zero;
    end
  endcase
```

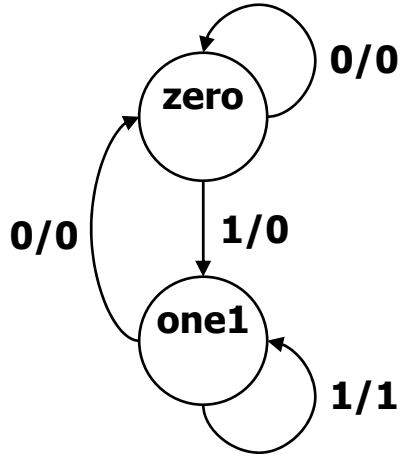
crucial to include all signals that are input to state determination

note that output depends only on state

```
always @(state)
  case (state)
    zero: out = 0;
    one1: out = 0;
   two1s: out = 1;
  endcase

endmodule
```

Mealy Verilog FSM

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables
  reg next_state;

  always @(posedge clk)
    if (reset) state = zero;
    else       state = next_state;

  always @(in or state)
    case (state)
      zero:              // last input was a zero
     begin
       out = 0;
       if (in) next_state = one;
       else    next_state = zero;
     end
      one:               // we've seen one 1
     if (in) begin
        next_state = one; out = 1;
     end else begin
        next_state = zero; out = 0;
     end
    endcase
endmodule
```



Synchronous Mealy Machine

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables

  always @(posedge clk)
    if (reset) state = zero;
    else
     case (state)
      zero:       // last input was a zero
     begin
       out = 0;
       if (in) state = one;
       else    state = zero;
     end
      one:        // we've seen one 1
     if (in) begin
        state = one; out = 1;
     end else begin
        state = zero; out = 0;
     end
    endcase
endmodule
```

State assignment

- Choose bit vectors to assign to each “symbolic” state
  - with n state bits for m states there are $2^n! / (2^n - m)!$
    [log n <= m <= 2^n]
  - 2^n codes possible for 1st state, 2^n–1 for 2nd, 2^n–2 for 3rd, …
  - huge number even for small values of n and m
    - intractable for state machines of any size
    - heuristics are necessary for practical solutions
  - optimize some metric for the combinational logic
    - size (amount of logic and number of FFs)
    - speed (depth of logic and fanout)
    - dependencies (decomposition)

State assignment strategies

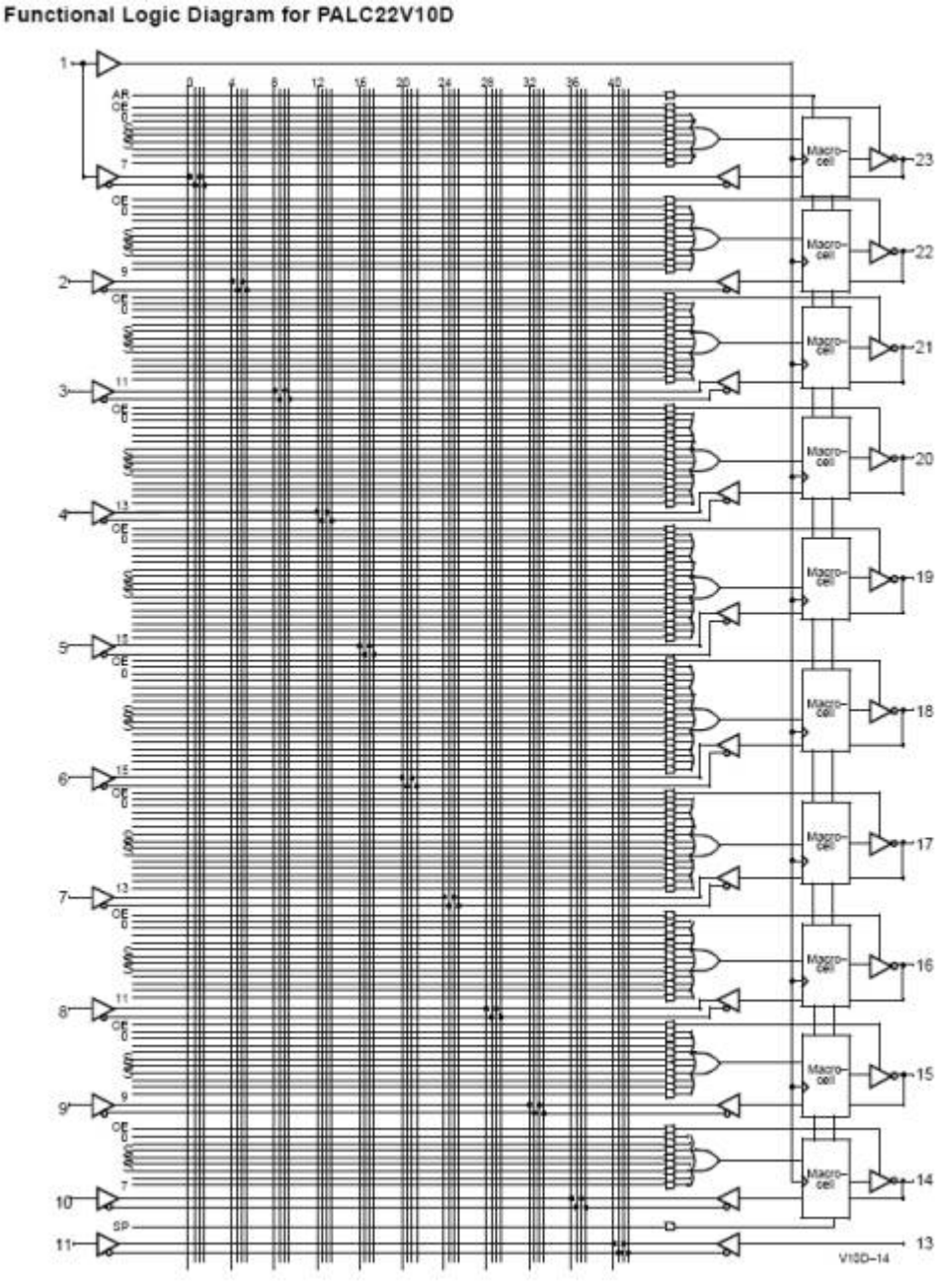
- Possible strategies
  - sequential – just number states as they appear in the state table
  - random – pick random codes
  - one-hot – use as many state bits as there are states (bit=1 –> state)
  - output – use outputs to help encode states
  - heuristic – rules of thumb that seem to work in most cases
- No guarantee of optimality – another intractable problem

Current state assignment approaches

- For tight encodings using close to the minimum number of state bits
  - best of 10 random seems to be adequate (averages as well as heuristics)
  - heuristic approaches are not even close to optimality
  - used in custom chip design
- One-hot encoding
  - easy for small state machines
  - generates small equations with easy to estimate complexity
  - common in FPGAs and other programmable logic
- Output-based encoding
  - ad hoc - no tools
  - most common approach taken by human designers
  - yields very small circuits for most FSMs
  - popular in PLDs

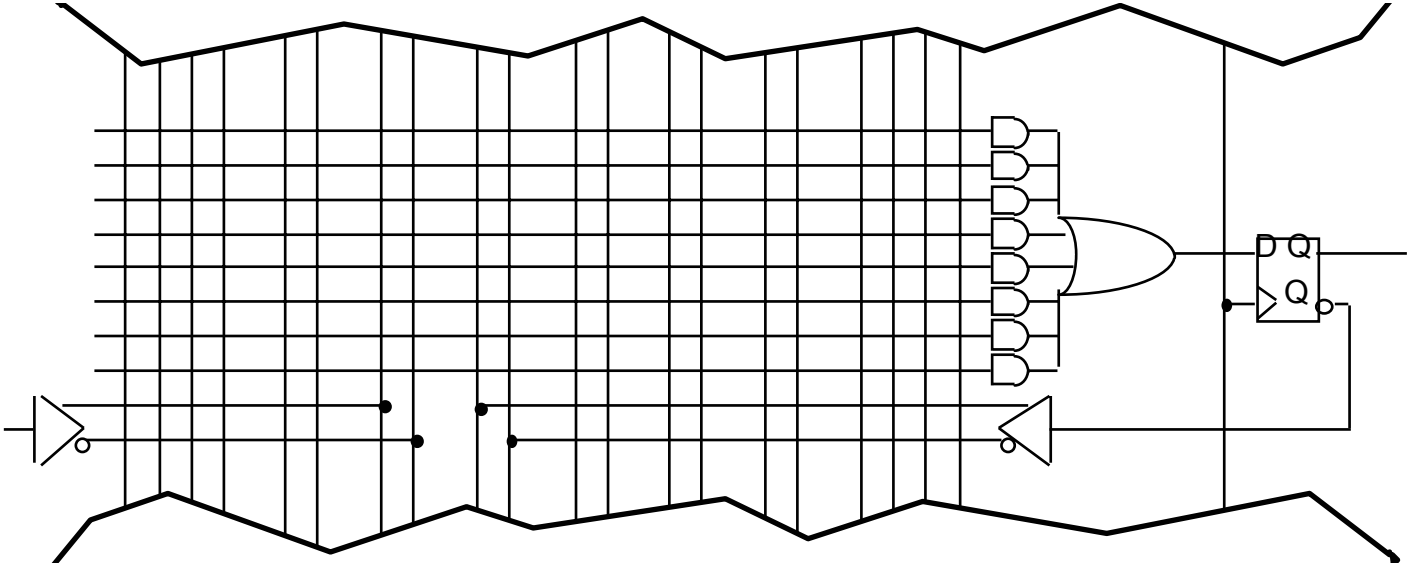
State machines and PLDs

- Moore and synchronous Mealy most common
- Output-directed state assignment
  - All outputs already have FFs and are fed back in as input to logic array
  - Use these as part of the state register
  - Add only as many extra states bits as needed to make all state codes unique



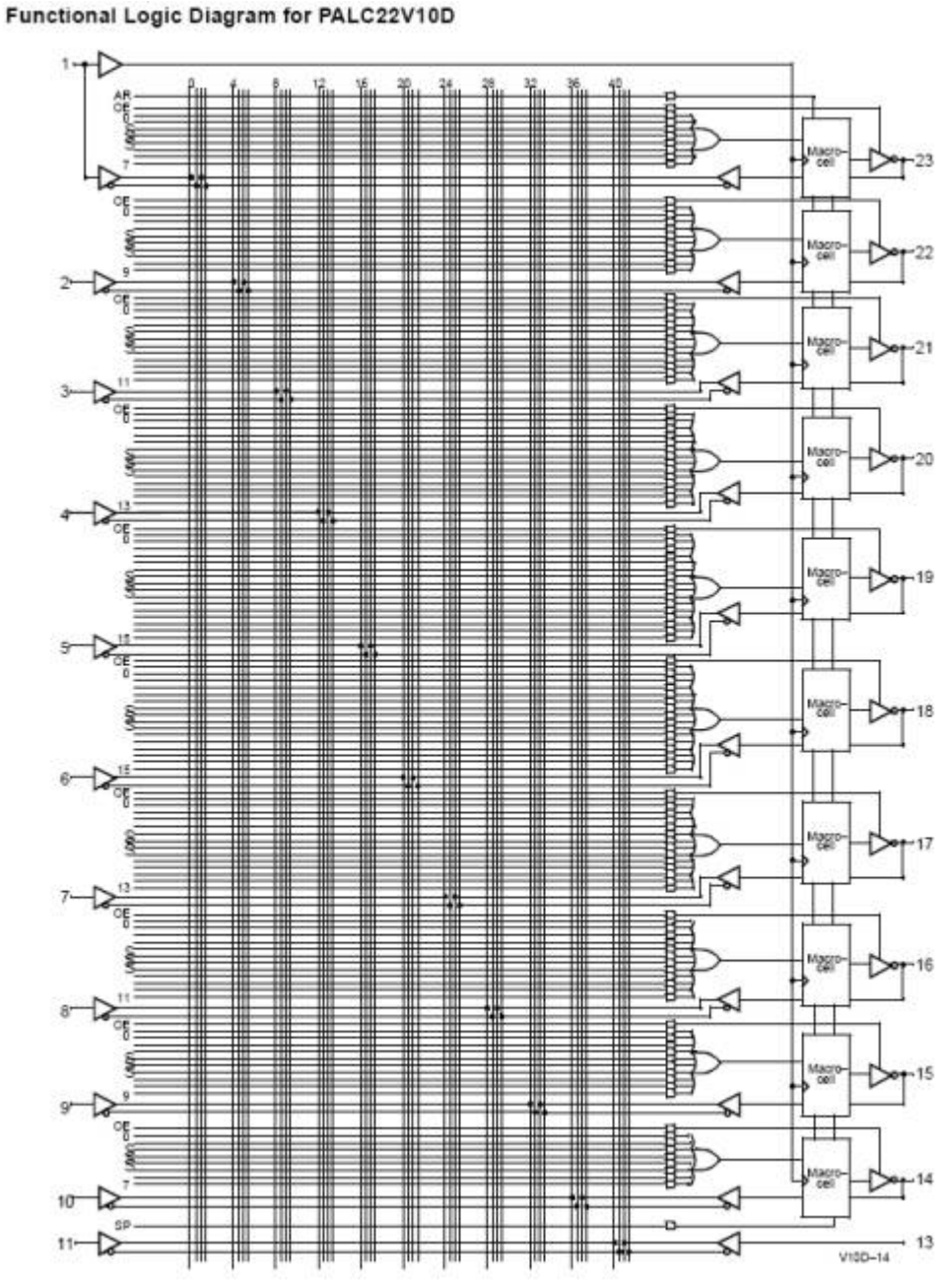
Implementation using PALs

- Programmable logic building block for sequential logic
  - macro-cell: FF + logic
    - D-FF
    - two-level logic capability like PAL (e.g., 8 product terms)



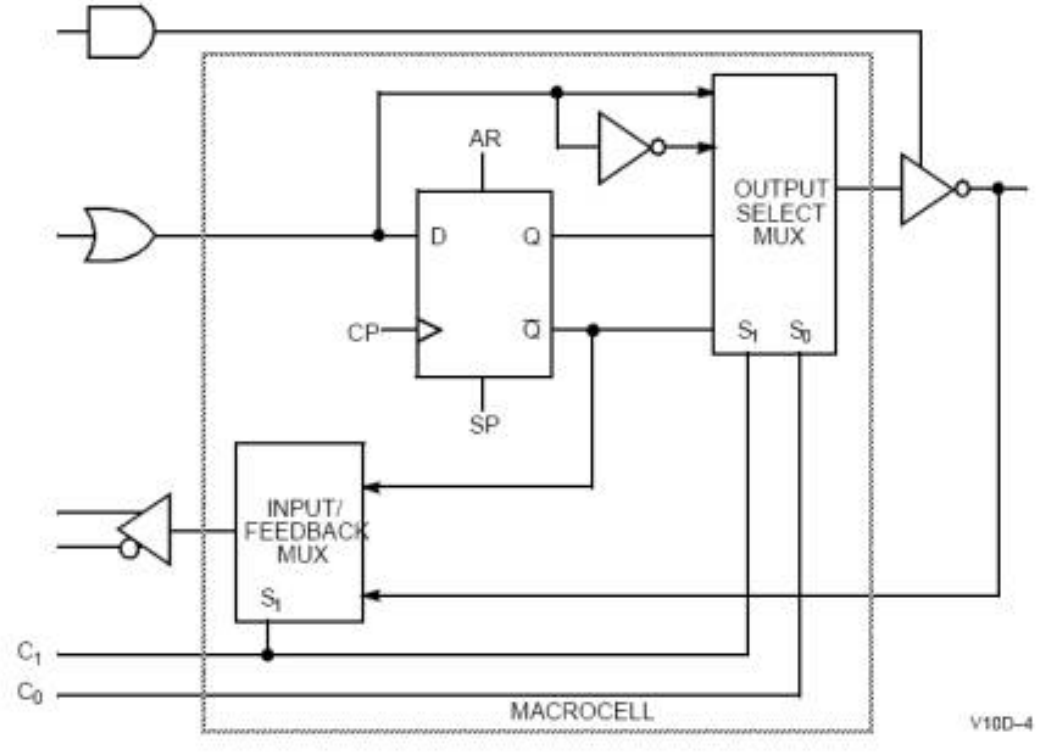
22V10 PAL

- Combinational logic elements (SoP)
- Sequential logic elements (D-FFs)
- Up to 10 outputs
- Up to 10 FFs
- Up to 22 inputs



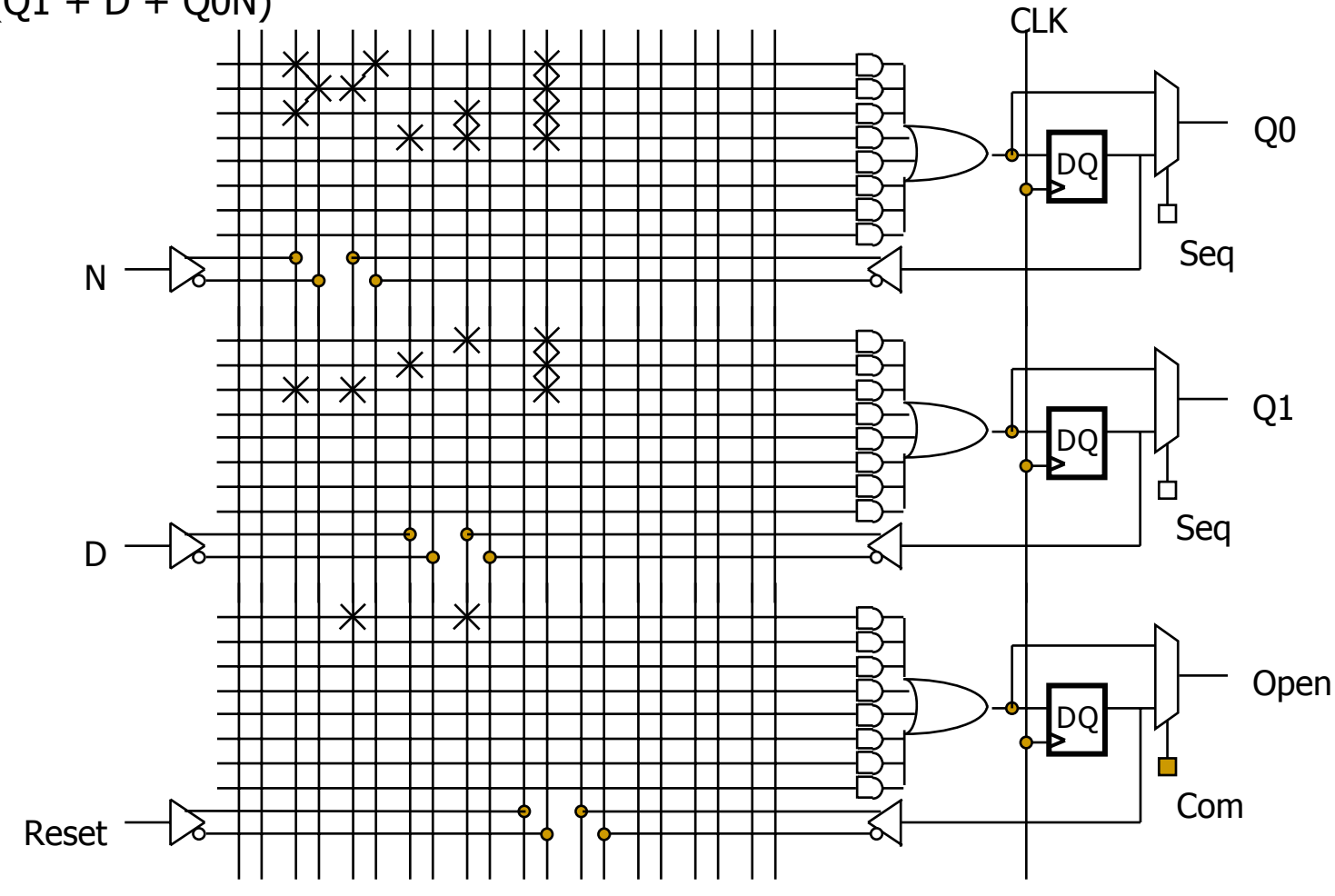
22V10 PAL Macro Cell

- Sequential logic element + output/input selection



Vending machine example (Moore PLD mapping)

D0 = reset'(Q0'N + Q0N' + Q1N + Q1D)
D1 = reset'(Q1 + D + Q0N)
OPEN = Q1Q0



Vending machine (synch. Mealy PLD mapping)

OPEN = reset'(Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D)