CH-9: Sequential Logic Implementation

Contemporary Logic Design

YONSEI UNIVERSITY

Fall 2016

Sequential Logic Implementation

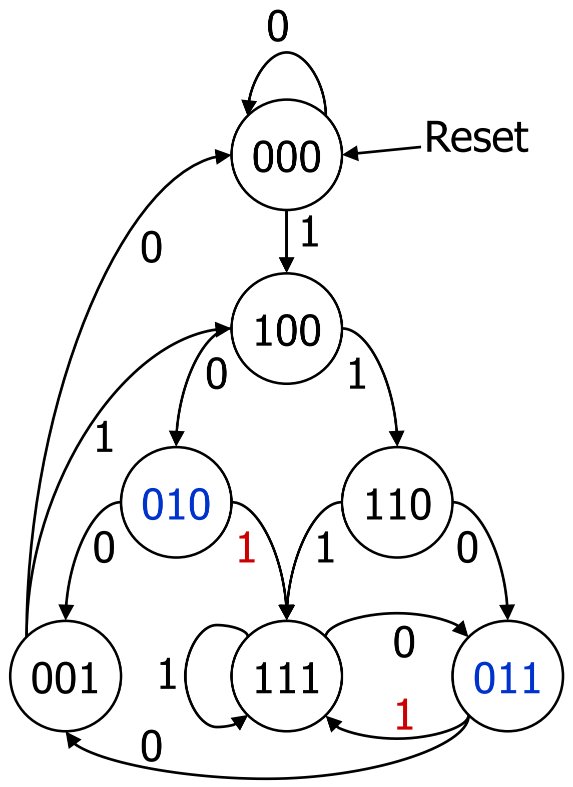
- Implementation
  - Random logic gates and FFs
  - Programmable logic devices (PAL with FFs)
- Design procedure
  - *Derive state diagrams*
  - *Obtain state transition table*
  - *Do state assignment*
  - *Obtain next state functions*

Advantage of FFs with Set/Reset

- Logic for FSM with a reset input
  - $NS_i = Reset' (PSa\ ICa + PSb\ ICb + \ldots)$
  - *NSi*: next state bit, *PSa*: present state
  - *ICa*: input condition
  - If use a FF with reset input, the Eq becomes simple as
    $NS_i = PSa\ ICa + PSb\ ICb + \ldots$
- EX: median filter
  - To show the advantage of FFs with set/reset inputs

Median Filter FSM

- Remove single 0 between any two 1s (output = NS3)
- Input: 0③,1②, 0① (101 → 111)



I	PS1	PS2	PS3	NS1	NS2	NS3
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	X	X	X
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	0	1	X	X	X
1	1	1	0	1	1	1
1	1	1	1	1	1	1

State 101 cannot occur

Median Filter FSM

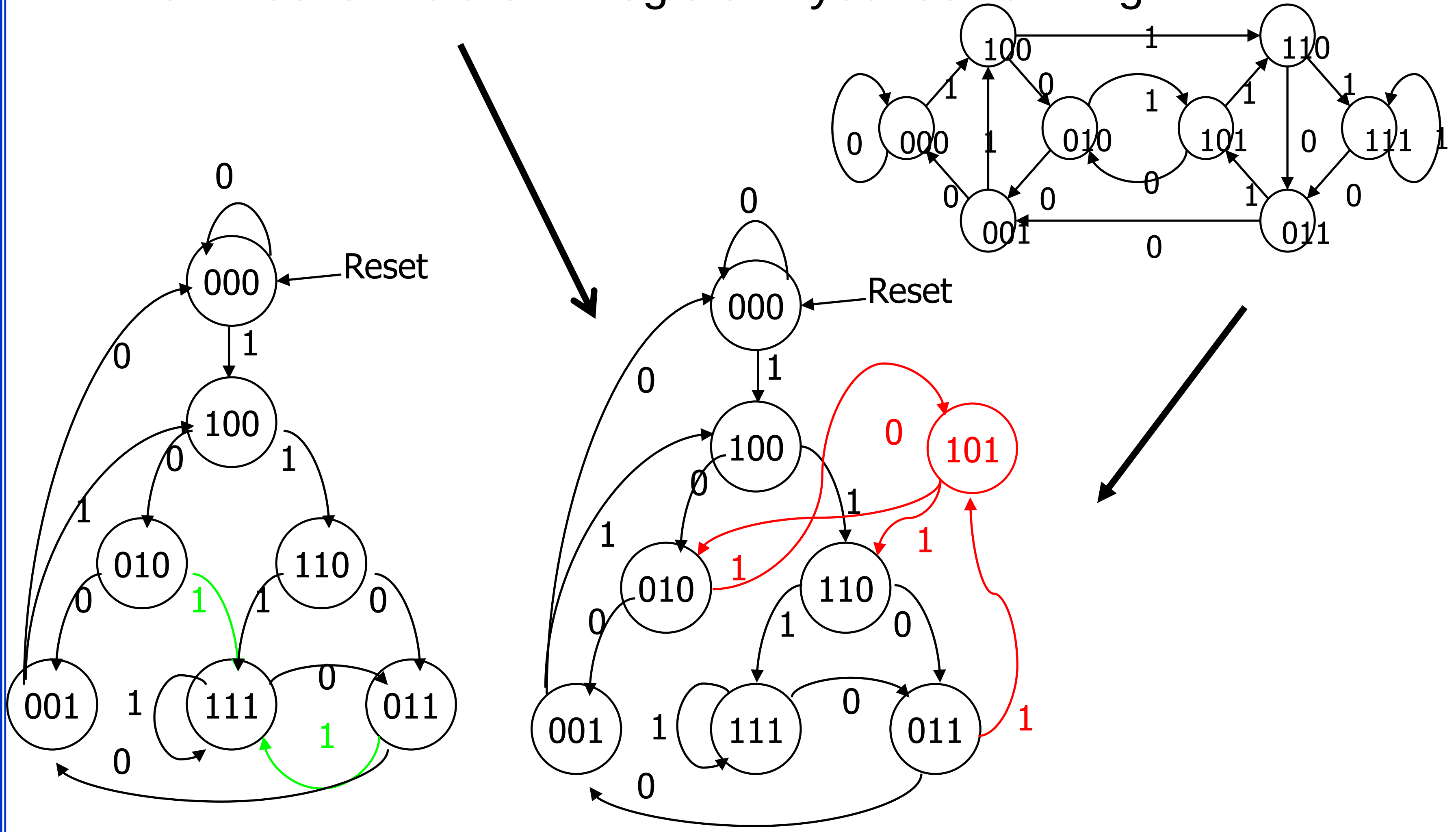
- Realized using the standard procedure from K-maps, and individual FFs and gates

I	PS1	PS2	PS3	NS1	NS2	NS3
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	X	X	X
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	0	1	X	X	X
1	1	1	0	1	1	1
1	1	1	1	1	1	1

NS1 = Reset' (I)
NS2 = Reset' (PS1 + PS2 I)
NS3 = Reset' PS2
O = PS3

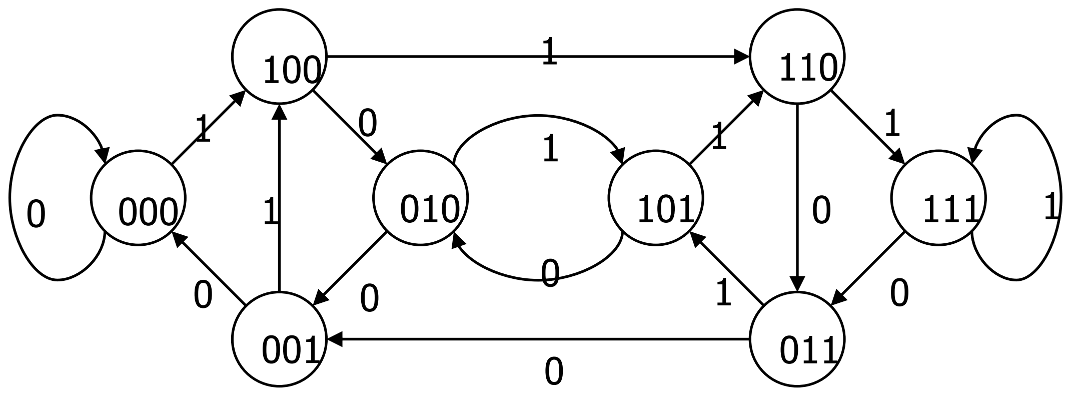
Median Filter FSM

- But it looks like a shift register if you look at it right

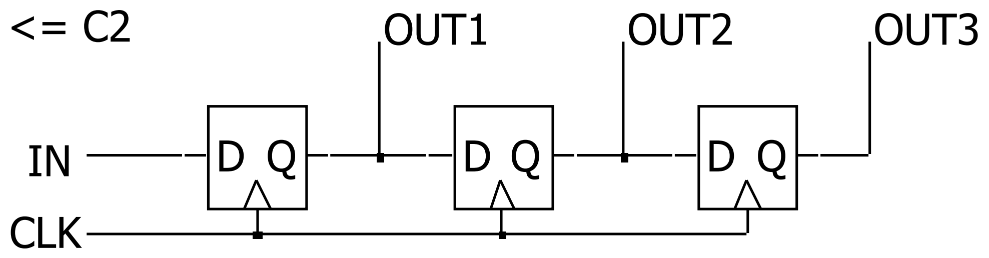


Median Filter FSM

- Shift register designed in Chap 7



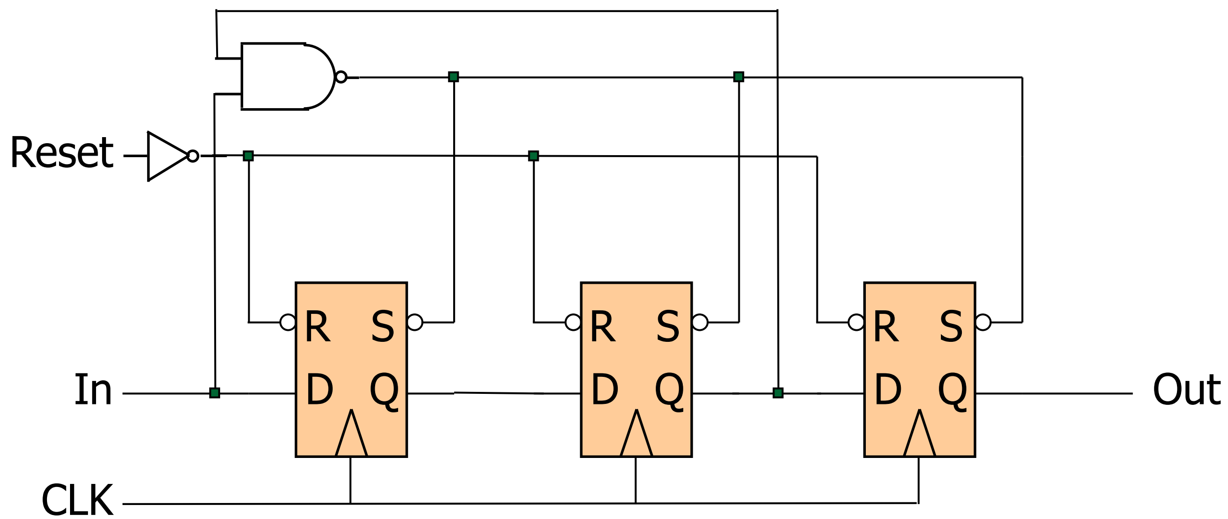
N1 <= In
N2 <= C1
N3 <= C2



Only NS2 is different: NS2 = Reset' (PS1 + PS2 I)

Median Filter FSM

- An alternate implementation with S/R FFs

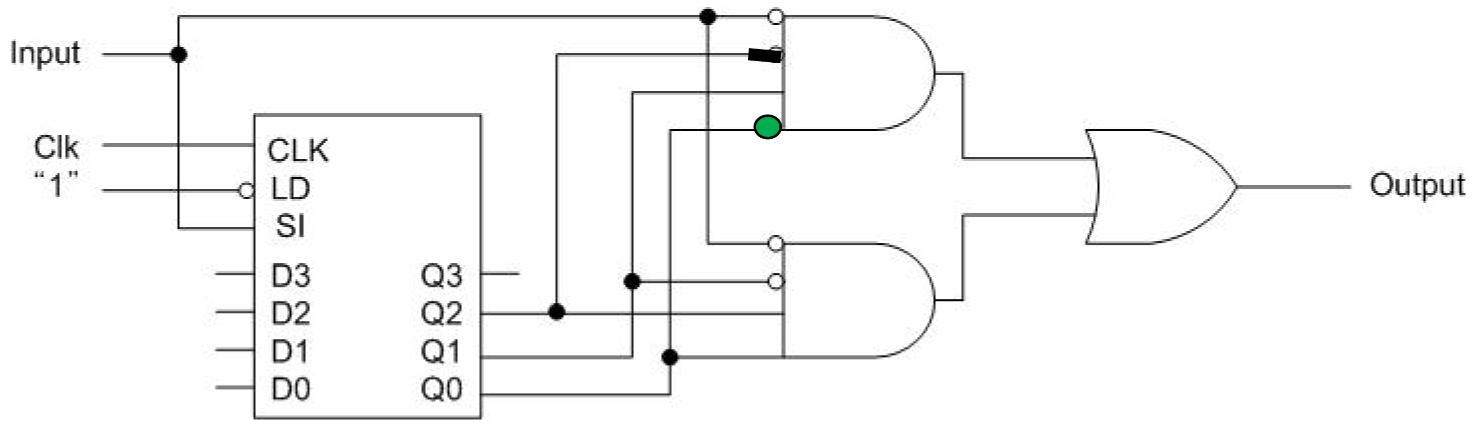


R = Reset
S = PS2 I
NS1 = I
NS2 = PS1
NS3 = PS2
O = PS3

- The set input (S) does the median filter function by making the next state 111 whenever the input is 1 and PS2 is 1 (1 input to state x1x)

String Recognizer using a Shift Register

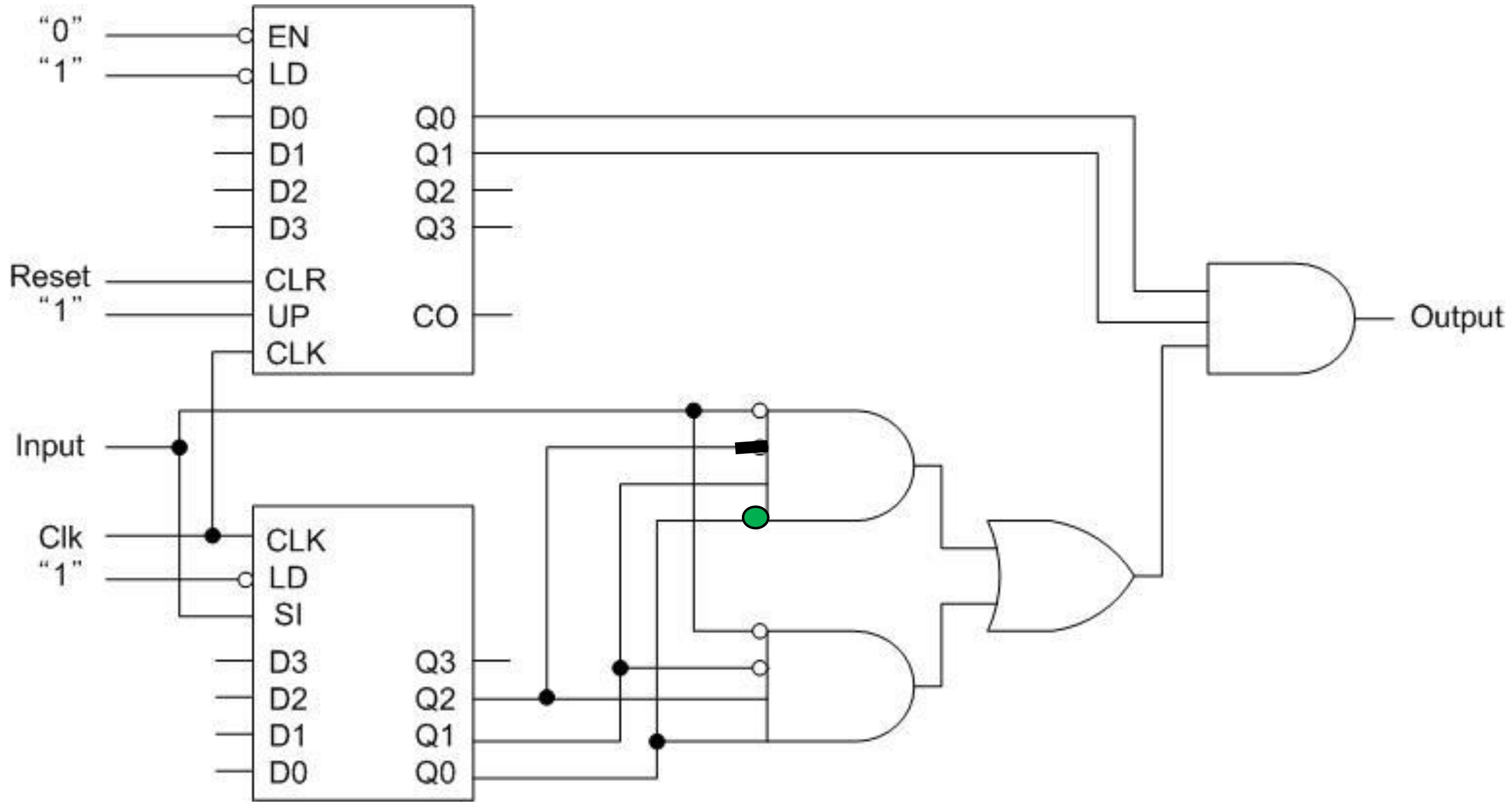
- Initial realization of the 4-bit string recognizer using a shift register: 1 if string 0110 or 1010 is detected



  - Problems : shift-register implementation looks for the two patterns anywhere in the input stream and <u>not just in 4-bit</u> groups
  - Need to make sure that output is only asserted <u>on 4-bit boundaries</u>

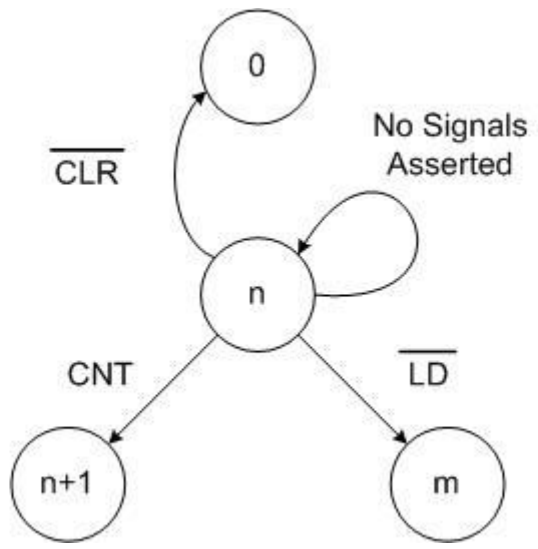
String Recognizer using a Shift Register

- Corrected realization of the 4-bit string recognizer using a shift register and counter



String Recognizer using a Shift Register

- State transitions of a counter-based finite state machine
  - CNT, LD, & Clear signals
  - Design a sequential circuit with counters



BCD-to-excess-3 Code Converter

- BCD and excess-3 code

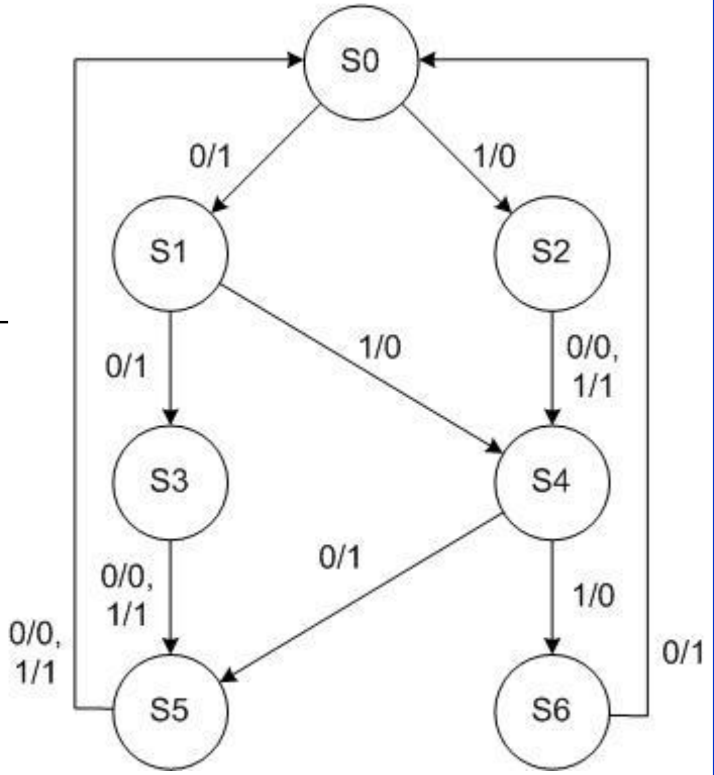
BCD	Excess-3 Code
0000	0011
0001	0100
0010	0101
0011	0110
0100	0111
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100

- Excess-3 code: by adding 11_2 to the BCD number
- Design to accept a bit serial BCD number
  - single input X & single output Z

BCD-to-excess-3 Code Converter

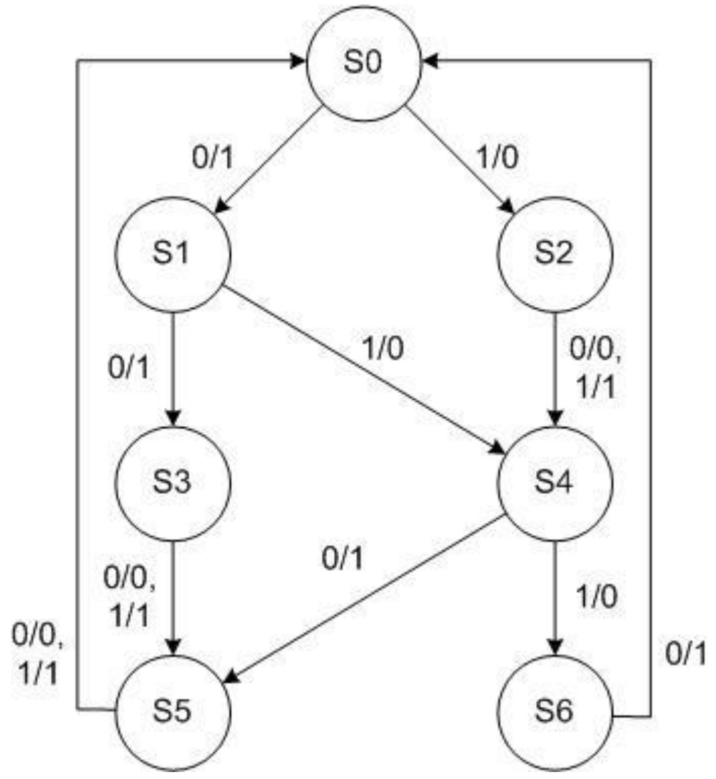
- Symbolic state transition table

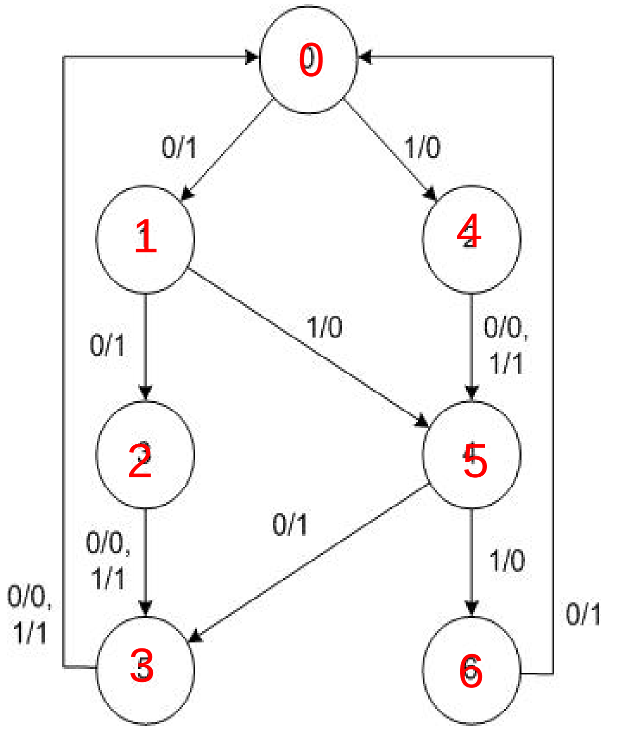
	Next State		Output	
Present State	X=0	X=1	X=0	X=1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	-	1	-



BCD-to-excess-3 Code Converter

- Symbolic state transition diagram

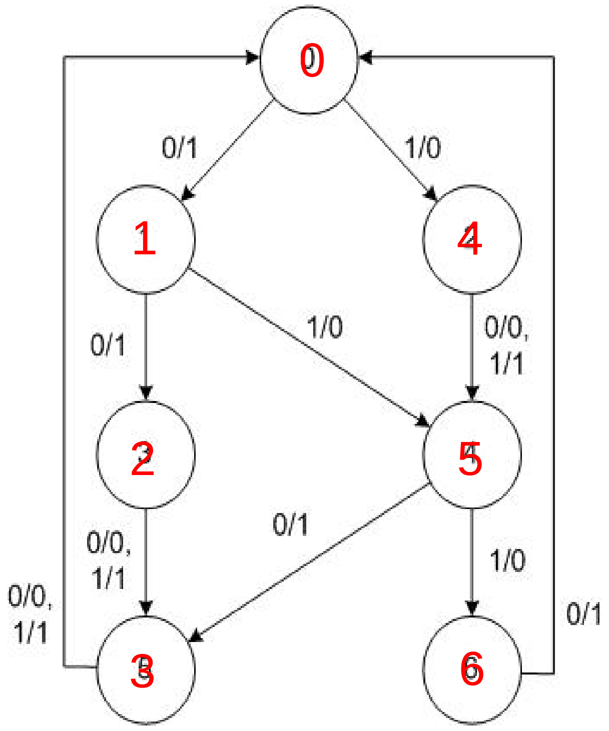




Encoded State Diagram

BCD-to-excess-3 Code Converter

- Encoded state transition diagram
  - There are 3 jumps (LD)
  - State 0 → 4, 1 → 5, 5 → 3
  - Need to specify directly next state bits as parallel inputs with LD’ asserted
- Consider some cases in SD
  - State0 & in=0: FSM goes to state 1 with output 1 (CLR’ & LD’ unasserted)
  - State0 & in=1: FSM goes to state 4 with output 0→ require jump LD’ asserted & value to be loaded is 100 (4) on C,B,A



Encoded State Diagram

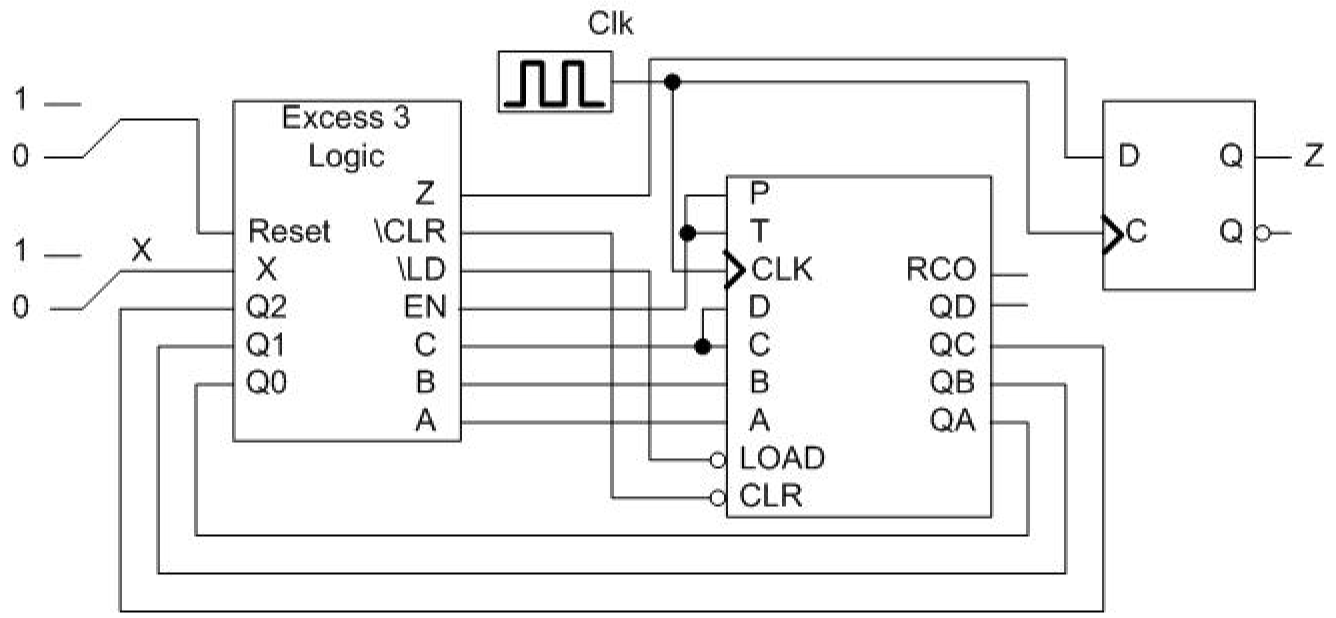
BCD-to-excess-3 Code Converter

- Transition table for code converter implemented with a counter-based finite state machine

Inputs/Current State				Next State			Outputs						
X	Q2	Q1	Q0	Q2+	Q1+	Q0+	Z	$\overline{CLR}$	$\overline{LD}$	EN	C	B	A
0	0	0	0	0	0	1	1	1	1	1	X	X	X
0	0	0	1	0	1	0	1	1	1	1	X	X	X
0	0	1	0	0	1	1	0	1	1	1	X	X	X
0	0	1	1	0	0	0	0	0	X	X	X	X	X
0	1	0	0	1	0	1	0	1	1	1	X	X	X
0	1	0	1	0	1	1	1	1	0	X	0	1	1
0	1	1	0	0	0	0	1	0	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X	X	X	X
1	0	0	0	1	0	0	0	1	0	X	1	0	0
1	0	0	1	1	0	1	0	1	0	X	1	0	1
1	0	1	0	0	1	1	1	1	1	1	X	X	X
1	0	1	1	0	0	0	1	0	X	X	X	X	X
1	1	0	0	1	0	1	1	1	1	1	X	X	X
1	1	0	1	1	1	0	0	1	1	1	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

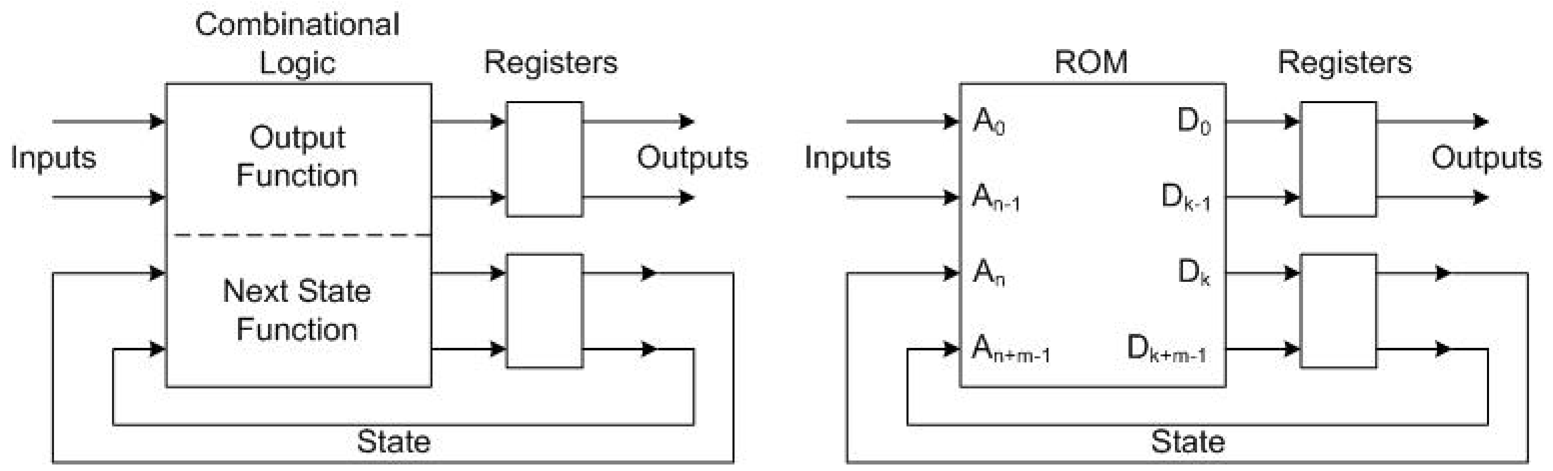
BCD-to-excess-3 Code Converter

- Counter-based implementation of code converter



BCD-to-excess-3 Code Converter

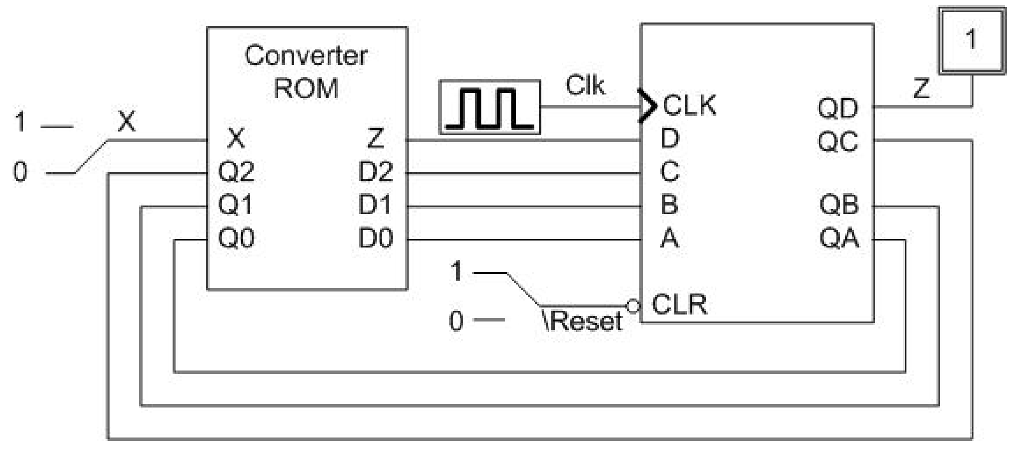
- ROM implementation of a synchronous Mealy finite state machine

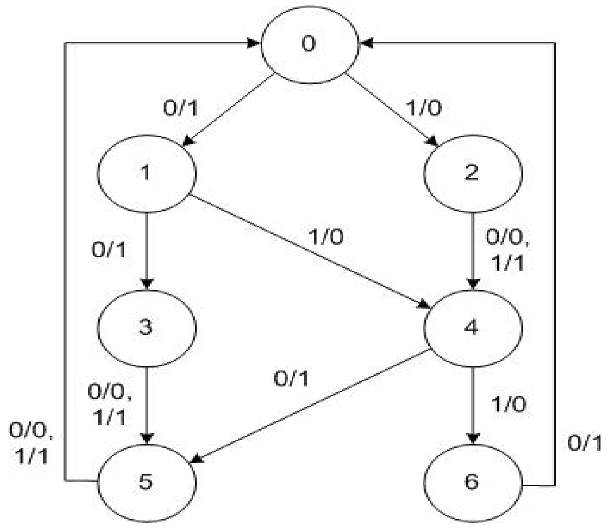


  - ROM or PAL/PLA is a convenient way to implement the combinational logic of a FSM.

BCD-to-excess-3 Code Converter

- Excess-3 synchronous Mealy ROM/FF-based implementation

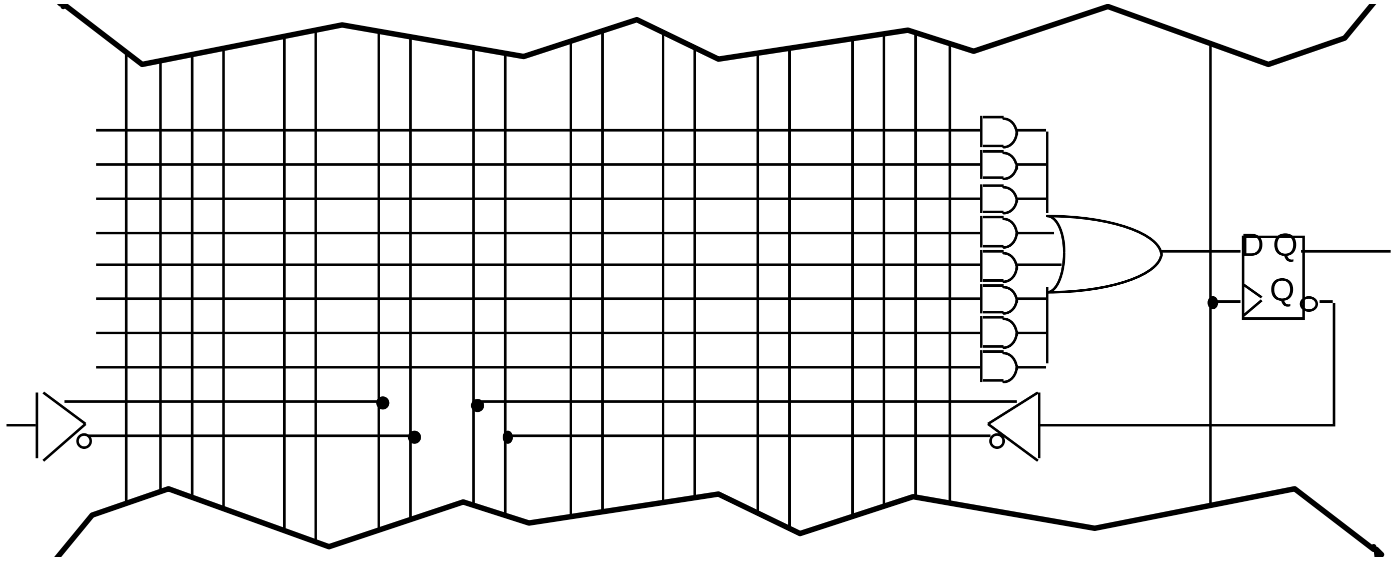




ROM Address				ROM Outputs			
X	Q2	Q1	Q0	Z	D2	D1	D0
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	1
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1
0	1	0	0	1	1	0	1
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	X	X	X	X
1	0	0	0	0	0	1	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	0	1	1	0
1	1	0	1	1	0	0	0
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

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)

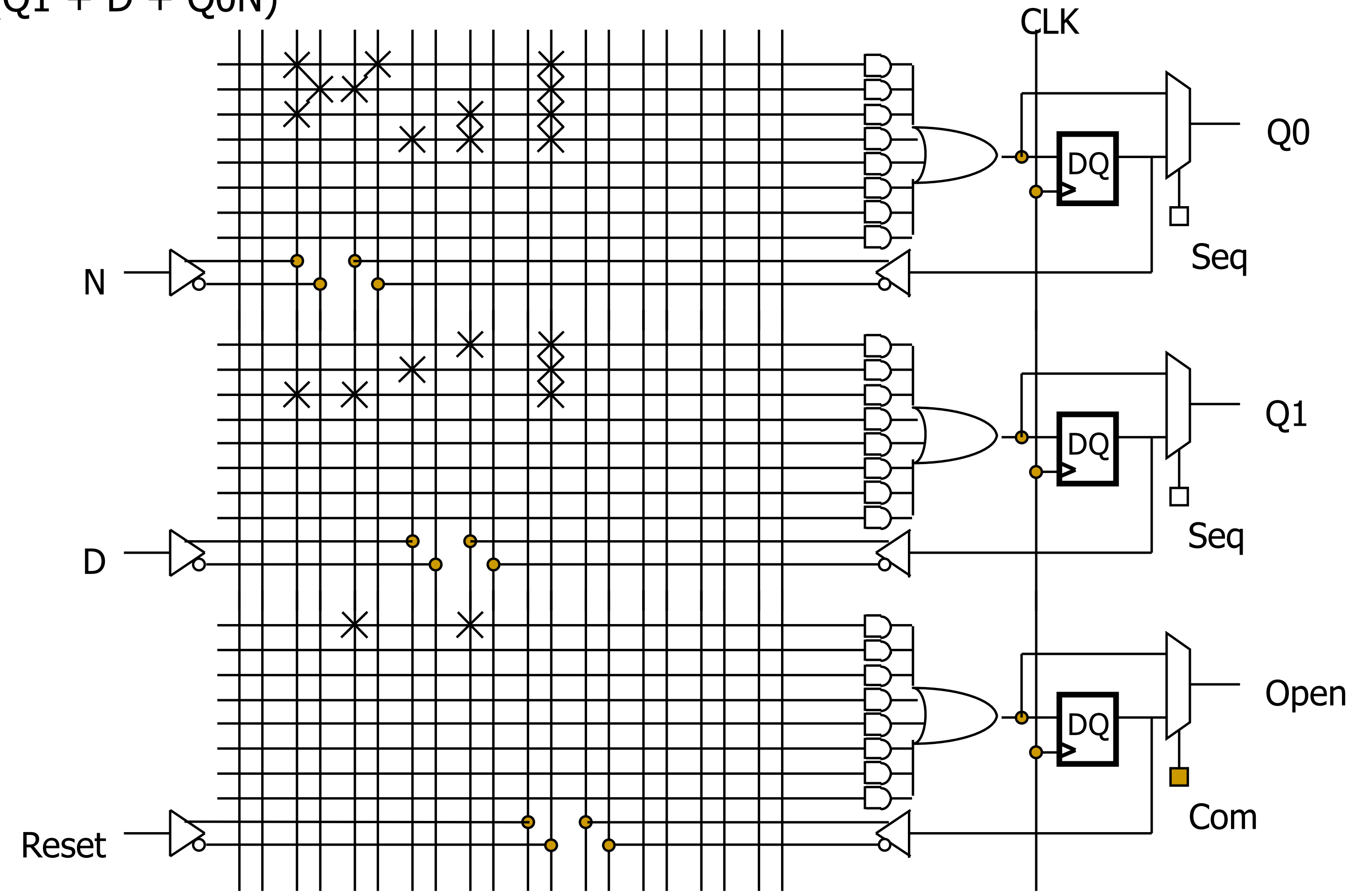


Vending Machine EX(Moore PLD mapping)

D0 = reset'(Q0'N + Q0N' + Q1N + Q1D)

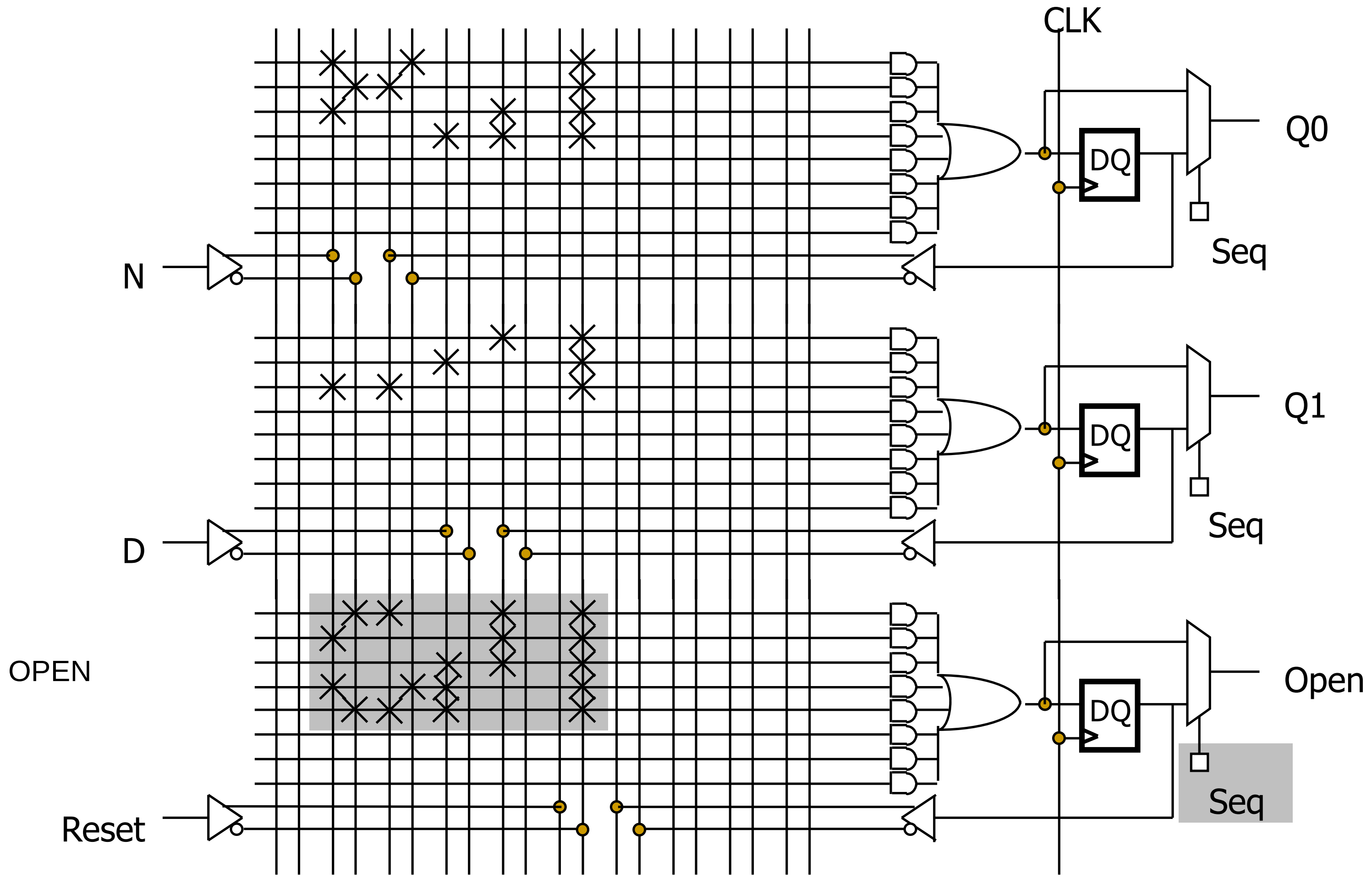
D1 = reset'(Q1 + D + Q0N)

OPEN = Q1Q0



Vending Machine (synch. Mealy PLD)

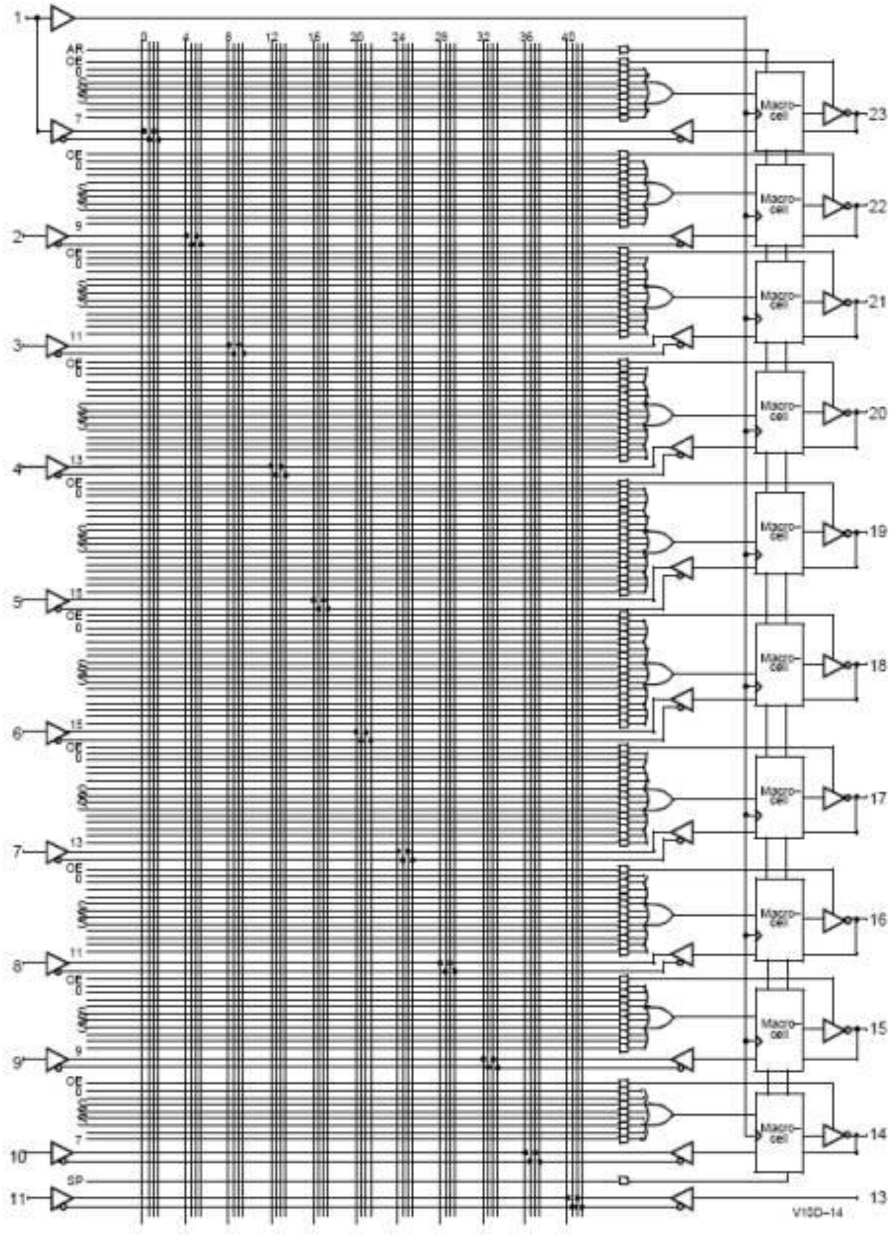
OPEN = reset'(Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D)



22V10 PAL

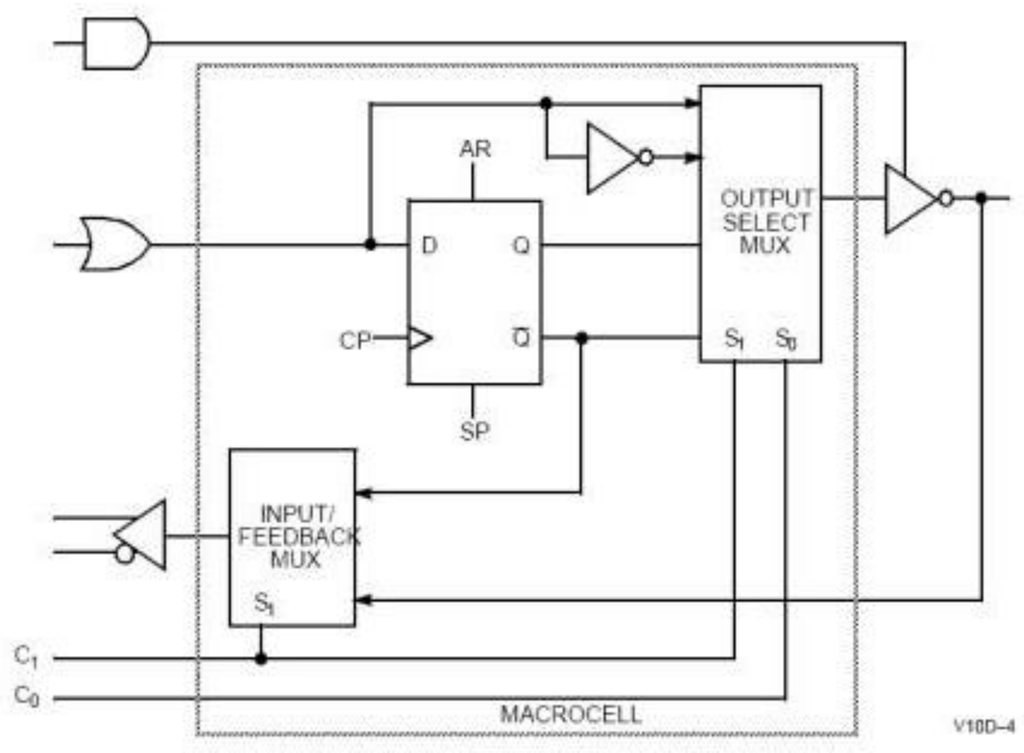
- Combinational logic elements (SoP)
- Sequential logic elements (D-FFs)
- Up to 10 outputs
- Up to 10 FFs
- Up to 22 inputs

Functional Logic Diagram for PALC22V10D



22V10 PAL Macro Cell

- Sequential logic element + output/input selection

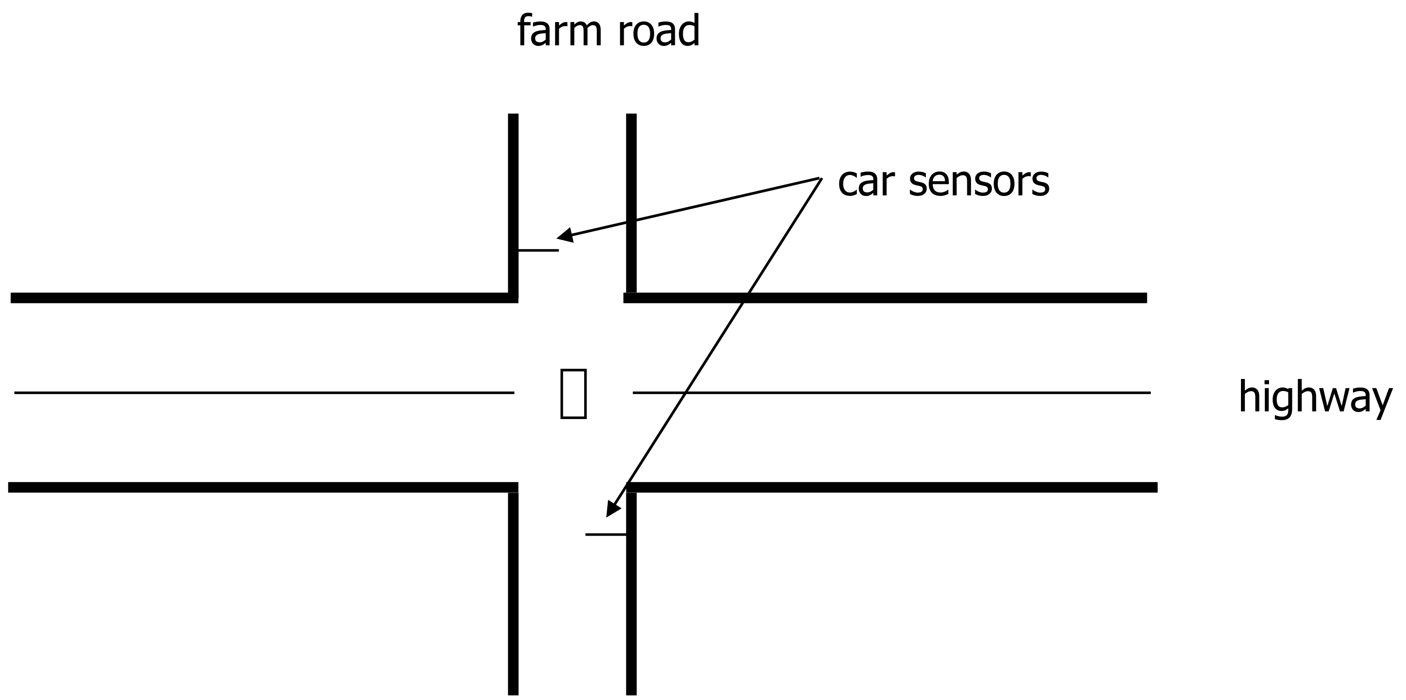


Example: Traffic Light Controller

- 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

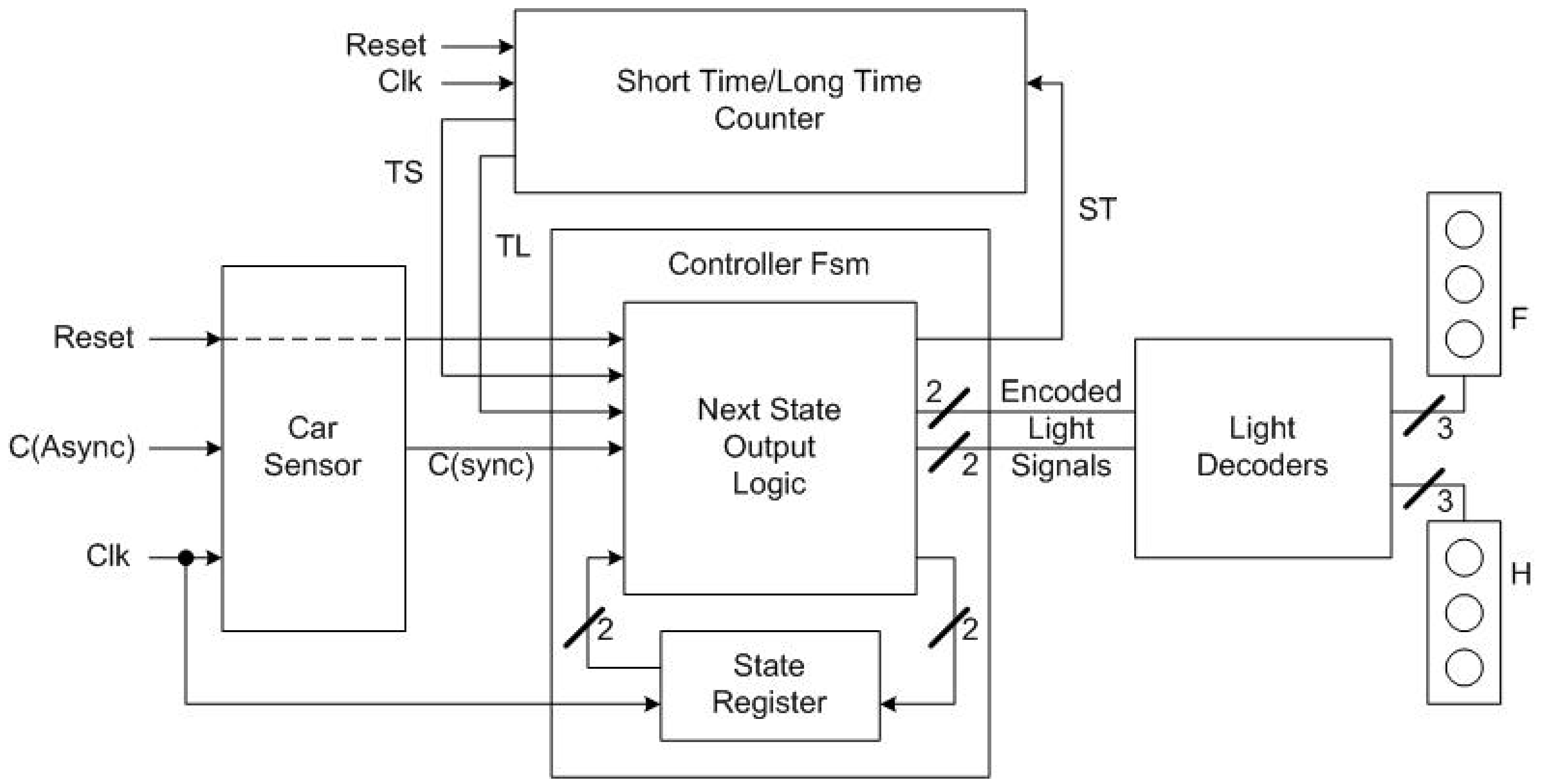
Example: Traffic Light Controller

- Highway/farm road intersection



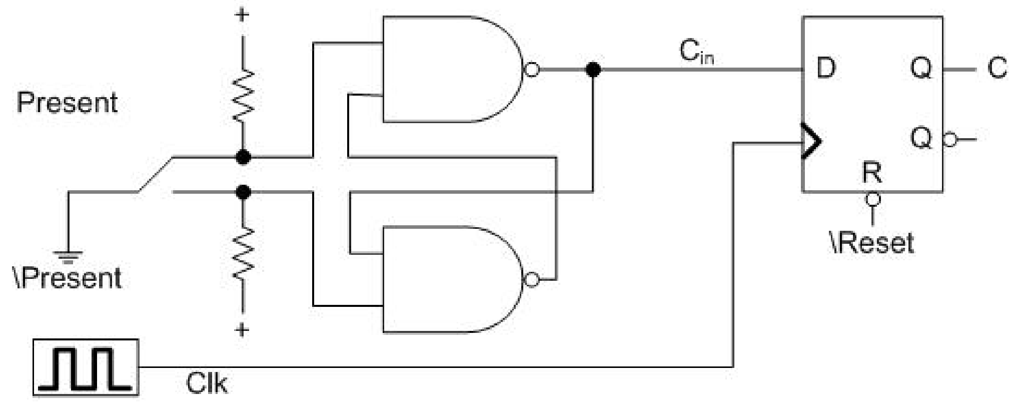
Example: Traffic Light Controller

- Block diagram of complete traffic light system

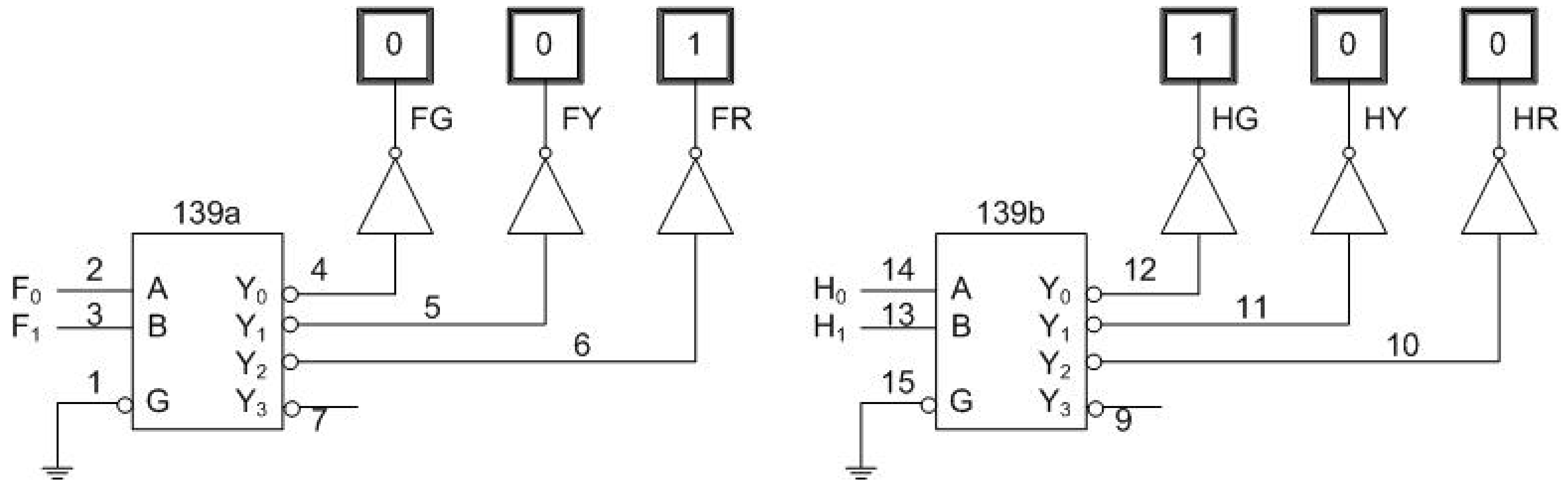


Example: Traffic Light Controller

- Car detector circuit

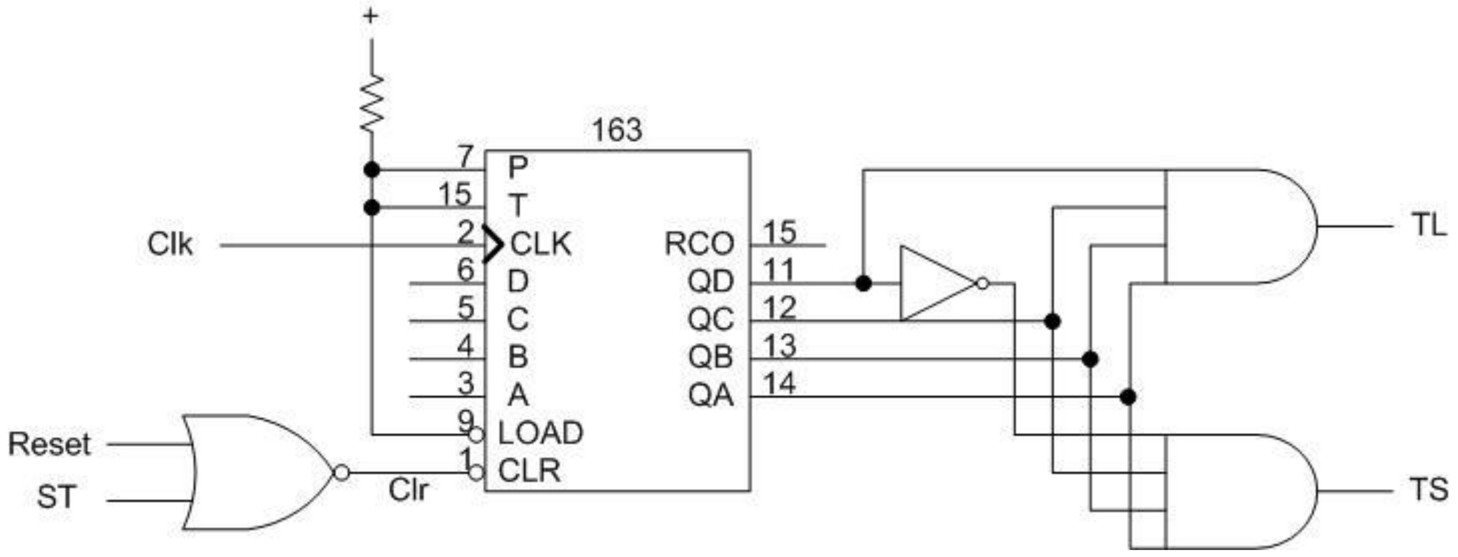


- Light decoder circuitry



Example: Traffic Light Controller

- Simple interval timing



Example: Traffic Light Controller

- Tabulation of inputs and outputs

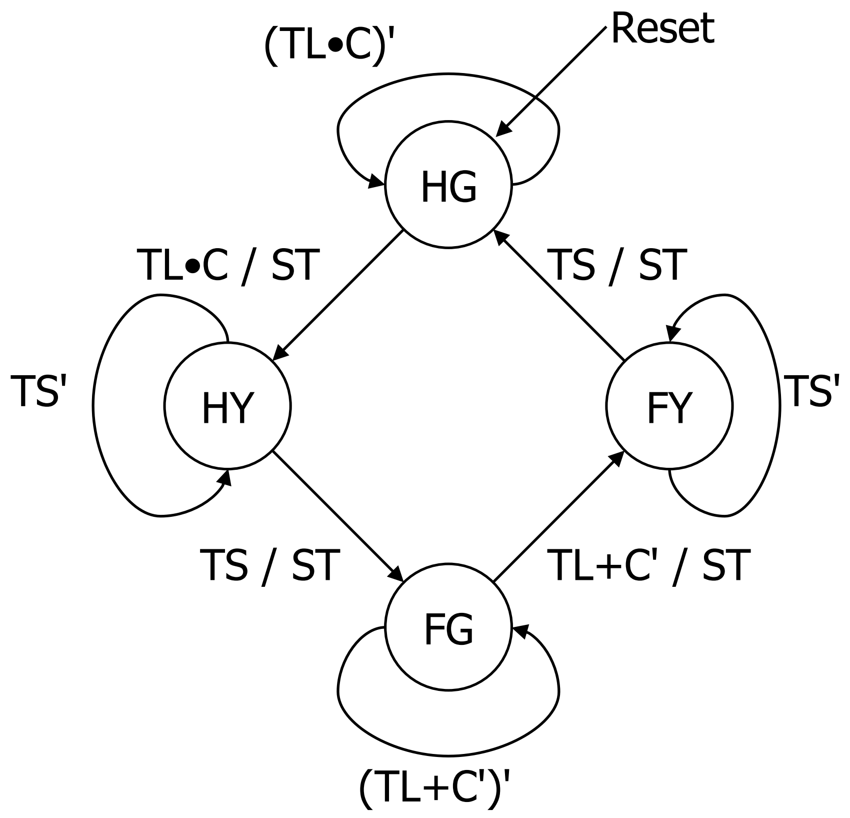
inputs	description	outputs	description
reset	place FSM in initial state	HG, HY, HR	assert green/yellow/red highway lights
C	detect vehicle on the farm road	FG, FY, FR	assert green/yellow/red highway lights
TS	short time interval expired	ST	start timing a short or long interval
TL	long time interval expired		

- Tabulation of unique states – some light configurations imply others

state	description
HG	highway green (farm road red)
HY	highway yellow (farm road red)
FG	farm road green (highway red)
FY	farm road yellow (highway red)

Example: Traffic Light Controller

- State diagram



Example: Traffic Light Controller

- *Generate state table* with symbolic states
- *Consider state assignments*

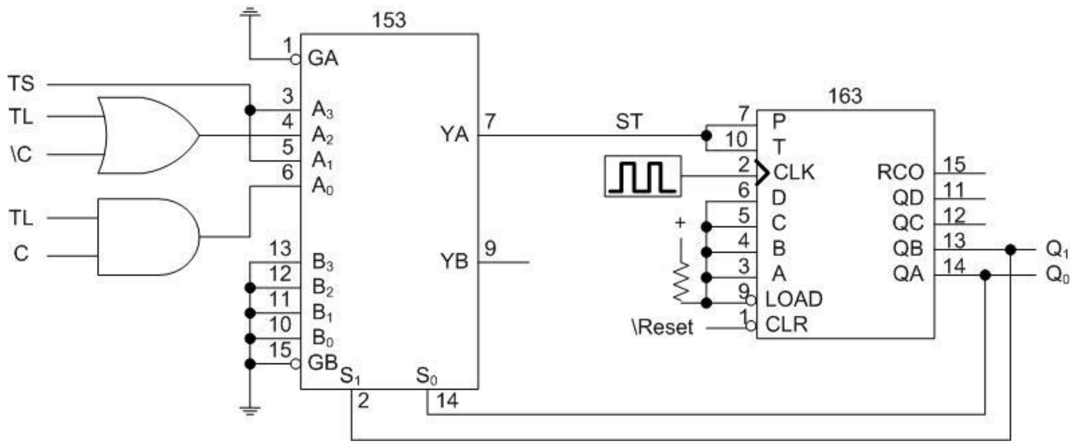
output encoding – similar problem
to state assignment
(Green = 00, Yellow = 01, Red = 10)

Inputs			Present State	Next State	Outputs		
C	TL	TS			ST	H	F
0	–	–	HG	HG	0	Green	Red
–	0	–	HG	HG	0	Green	Red
1	1	–	HG	HY	1	Green	Red
–	–	0	HY	HY	0	Yellow	Red
–	–	1	HY	FG	1	Yellow	Red
1	0	–	FG	FG	0	Red	Green
0	–	–	FG	FY	1	Red	Green
–	1	–	FG	FY	1	Red	Green
–	–	0	FY	FY	0	Red	Yellow
–	–	1	FY	HG	1	Red	Yellow

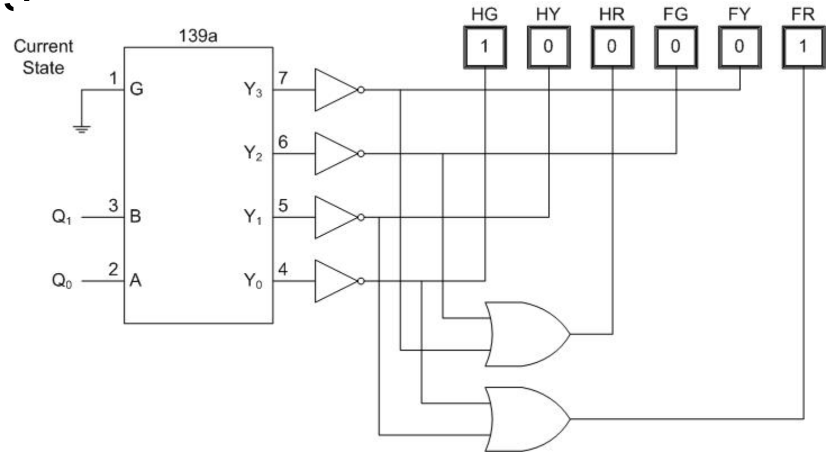
SA1:	HG = 00	HY = 01	FG = 11	FY = 10	
SA2:	HG = 00	HY = 10	FG = 01	FY = 11	
SA3:	HG = 0001	HY = 0010	FG = 0100	FY = 1000	(one-hot)

Example: Traffic Light Controller

- Counter-based implementation of the next-state function



- Traffic light decoder



Logic for Different State Assignments

- SA1

  NS1 = C•TL'•PS1•PS0 + TS•PS1'•PS0 + TS•PS1•PS0' + C'•PS1•PS0 + TL•PS1•PS0
  NS0 = C•TL•PS1'•PS0' + C•TL'•PS1•PS0 + PS1'•PS0

  ST = C•TL•PS1'•PS0' + TS•PS1'•PS0 + TS•PS1•PS0' + C'•PS1•PS0 + TL•PS1•PS0
  H1 = PS1　　　　H0 = PS1'•PS0
  F1 = PS1'　　　　F0 = PS1•PS0'

- SA2

  NS1 = C•TL•PS1' + TS'•PS1 + C'•PS1'•PS0
  NS0 = TS•PS1•PS0' + PS1'•PS0 + TS'•PS1•PS0

  ST = C•TL•PS1' + C'•PS1'•PS0 + TS•PS1
  H1 = PS0　　　　H0 = PS1•PS0'
  F1 = PS0'　　　　F0 = PS1•PS0

- SA3

  NS3 = C'•PS2 + TL•PS2 + TS'•PS3　　　　NS2 = TS•PS1 + C•TL'•PS2
  NS1 = C•TL•PS0 + TS'•PS1　　　　NS0 = C'•PS0 + TL'•PS0 + TS•PS3

  ST = C•TL•PS0 + TS•PS1 + C'•PS2 + TL•PS2 + TS•PS3
  H1 = PS3 + PS2　　　　H0 = PS1
  F1 = PS1 + PS0　　　　F0 = PS3

Sequential Logic Imp - Summary

- *Models* for representing sequential circuits
  - finite state machines and their state diagrams
  - Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
  - *deriving state diagram*
  - *deriving state transition table*
  - *assigning codes to states*
  - *determining next state and output functions*
  - *implementing combinational logic*
- Implementation technologies
  - random logic + FFs
  - PAL with FFs (programmable logic devices – PLDs)