Digital Logic Design

4190.201.001

2010 Spring Semester

10. Sequential Logic Technology

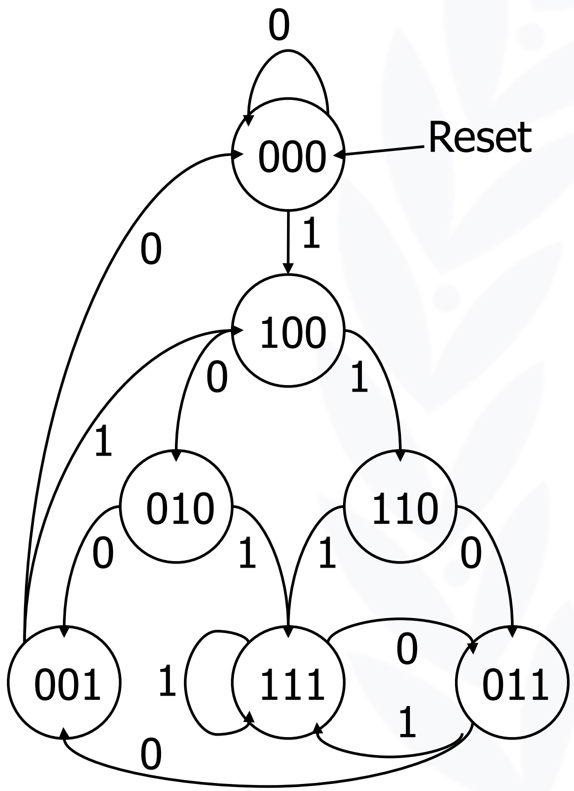
Naehyuck Chang
Dept. of EECS/CSE
Seoul National University
naehyuck@snu.ac.kr

Sequential logic implementation

- Implementation
  - Random logic gates and FFs
  - Programmable logic devices (PAL with FFs)
- Design procedure
  - State diagrams
    - Design
    - Verification (branch condition)
    - Reduction (implicant chart or raw matching)
  - State transition table
  - State assignment
    - Tight encoding for random logic
    - One-hot for FPGA
    - Output-based for PLD
  - Next state functions
  - Input synchronization

Median filter FSM

- Remove single 0s between two 1s (output = NS3)



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

Embedded Low-Power Laboratory

Median filter FSM (cont'd)

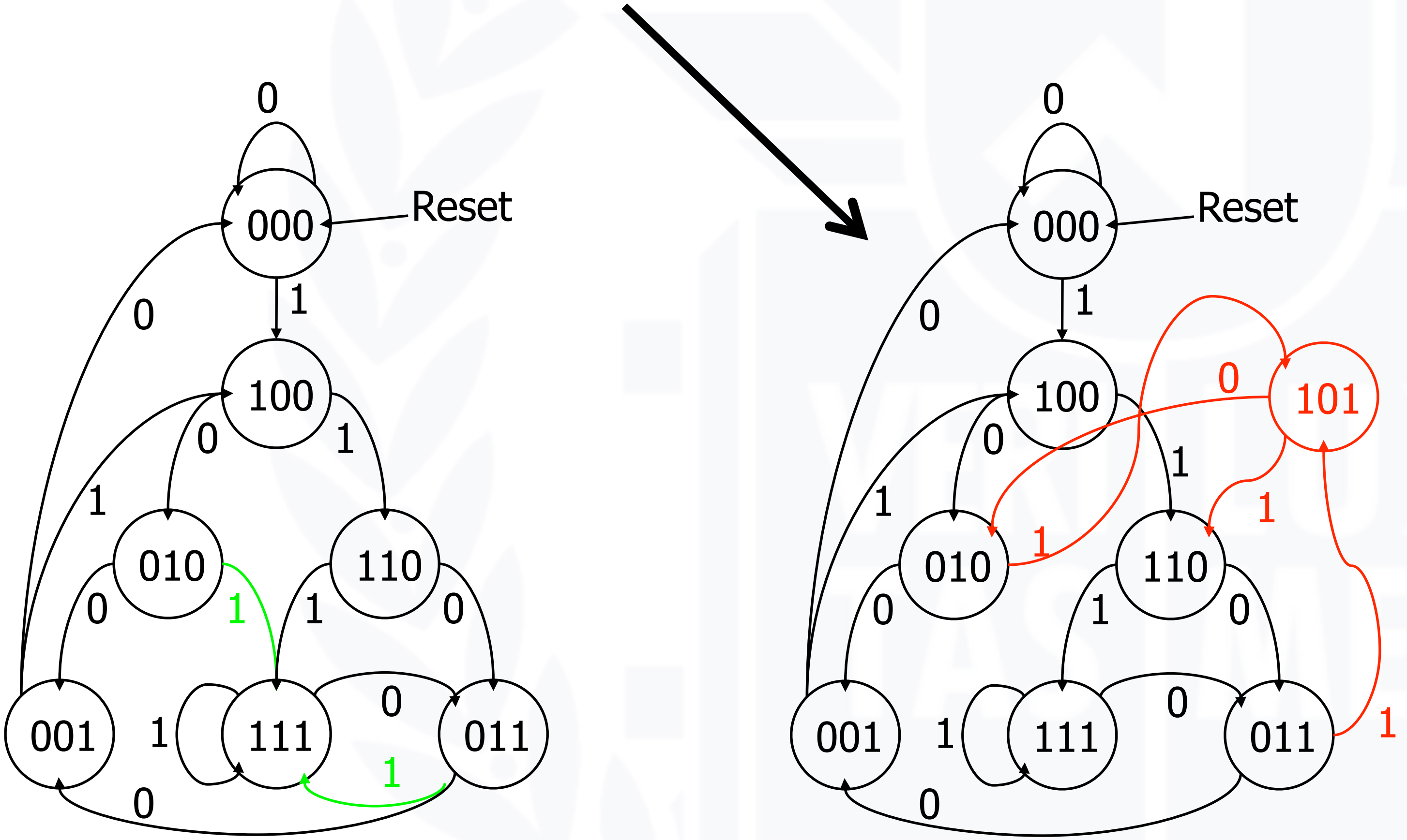
- Realized using the standard procedure and individual FFs and gates

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

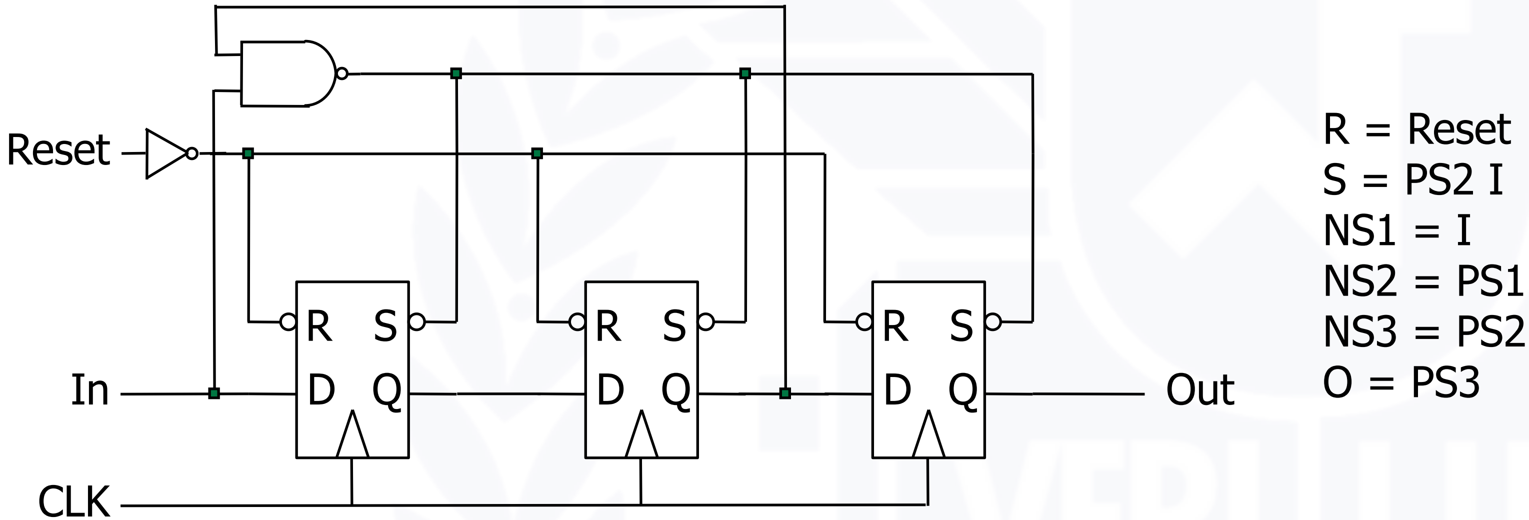
Median filter FSM (cont'd)

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



Median filter FSM (cont'd)

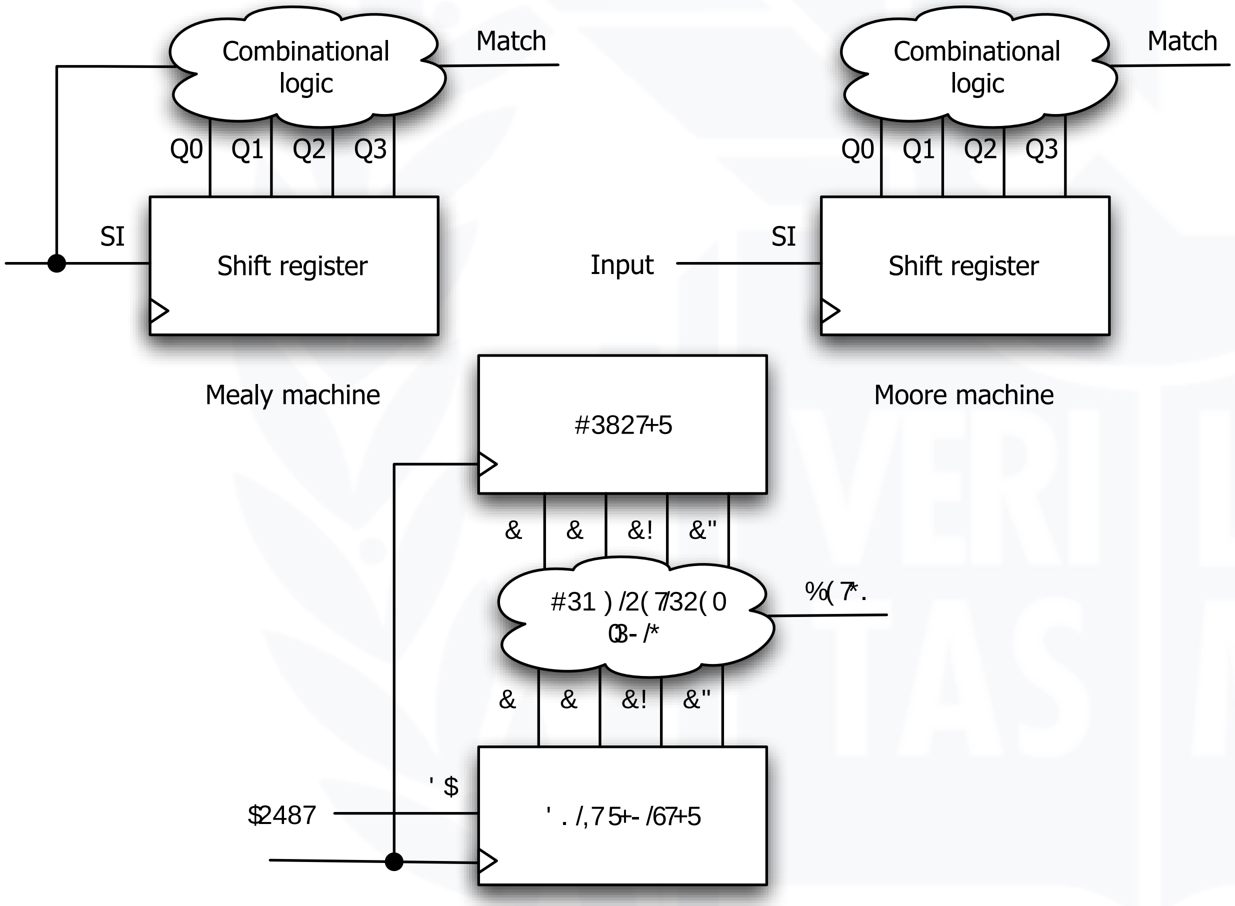
- An alternate implementation with S/R FFs
  - Personally I do not recommend this!



- 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)

FSM implementation with a shift register

- String recognizer
  - Good candidate for a shift register implementation



FSM implementation with a counter

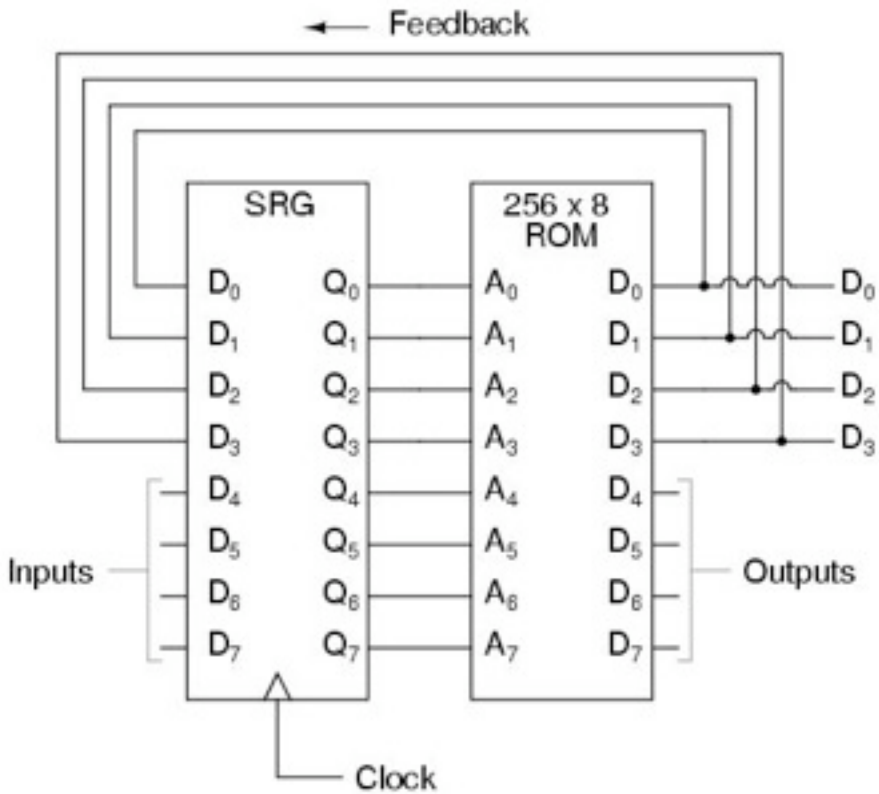
- Three functions of a counter
  - Count
  - Reset
  - Jump
- State machine implementation with a counter
  - Next state function
    - Count (CNT), Reset (R) and Load (LD)
- Sequencer

Input/Current State								Next State				Output							
I0	I1	I2	I3	Q0	Q1	Q2	Q3	Q0	Q1	Q2	Q3	LD*	R*	EN*	A	B	C	D	E

Embedded Low-Power Laboratory

FSM implementation with a ROM

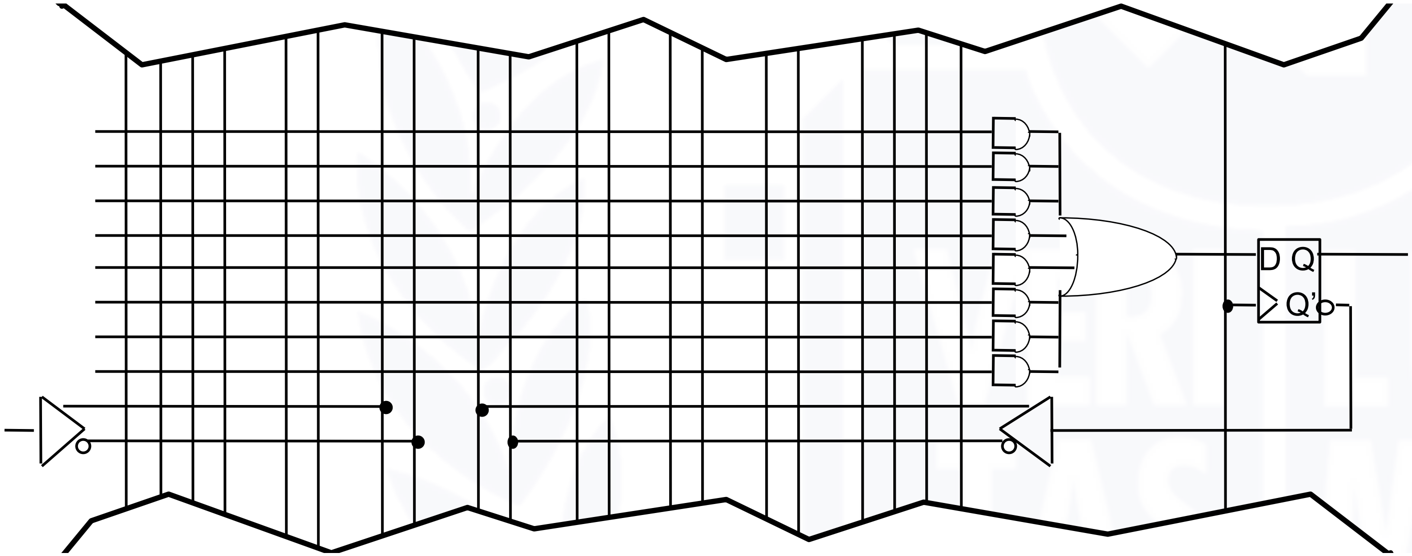
- PS + FSM input
  - Address input
- NS
  - Data output
- Input synchronization is applied here
- Both Moore and Mealy machines can be implemented
- Advantage and disadvantages?
  - Same to the combinational logic implementation with a ROM



Embedded Low-Power Laboratory

FSM implementation using PALs

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



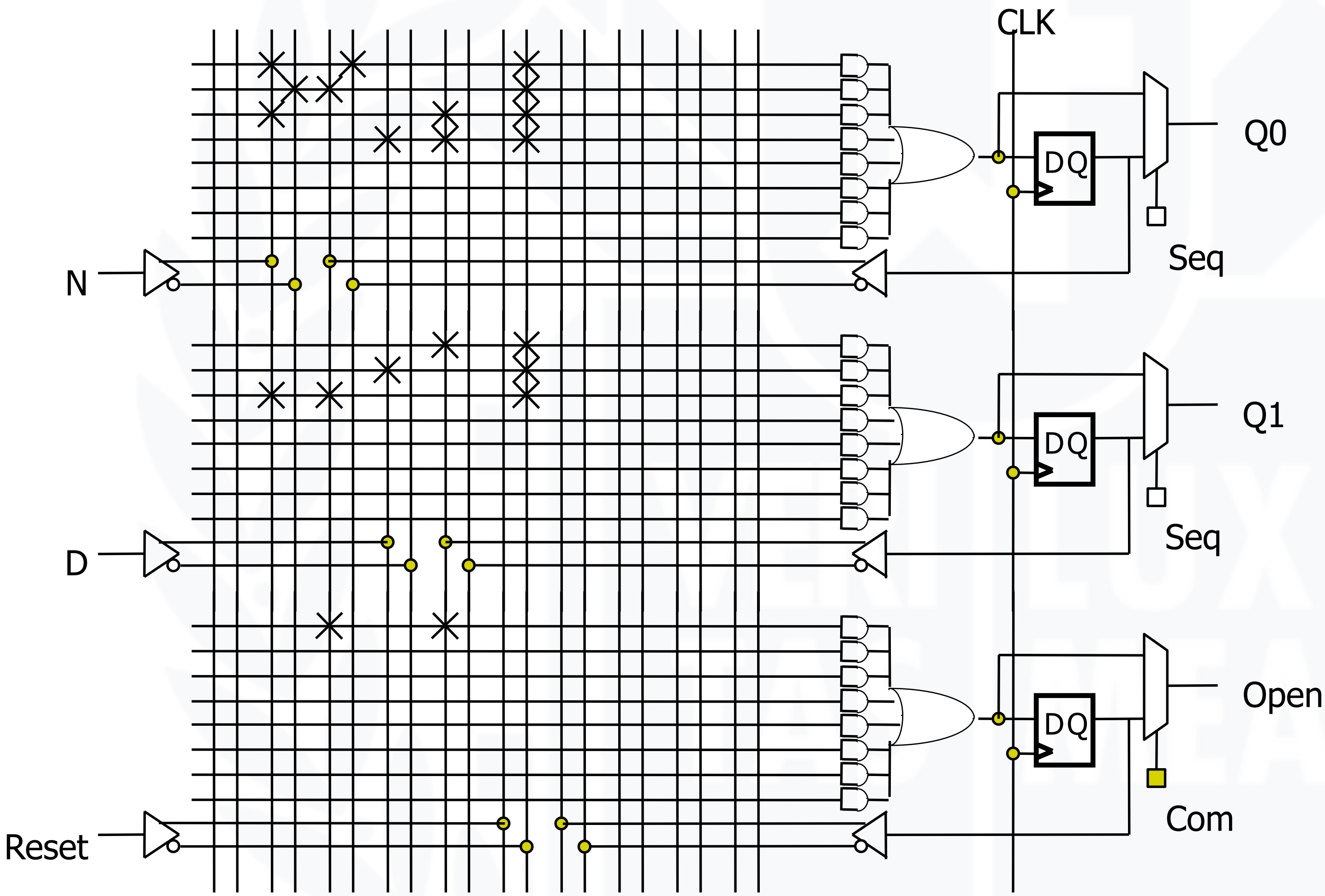
Embedded Low-Power Laboratory

Vending machine example (Moore PLD mapping)

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

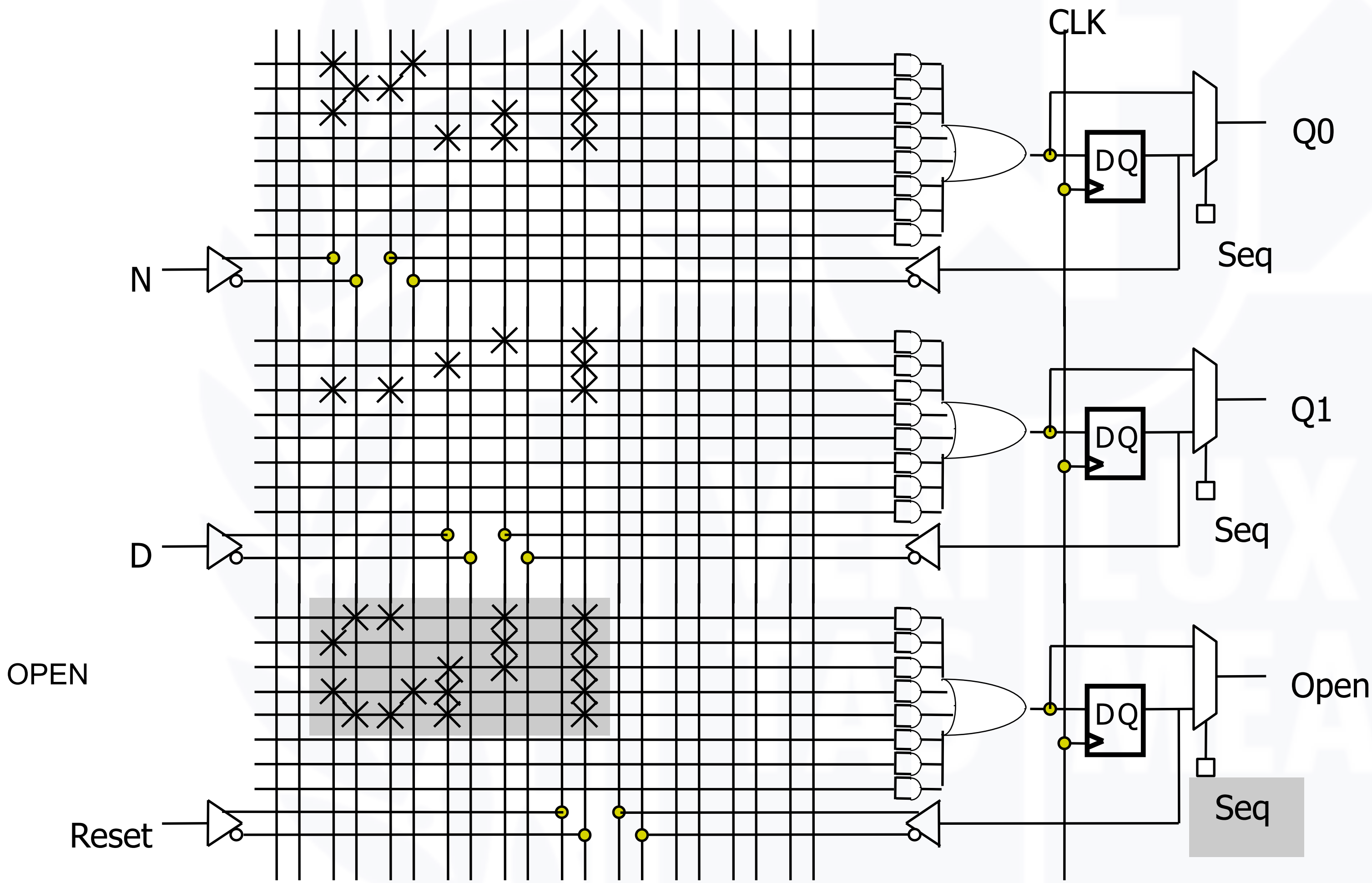
D1 = reset'(Q1 + D + Q0N)

OPEN = Q1Q0



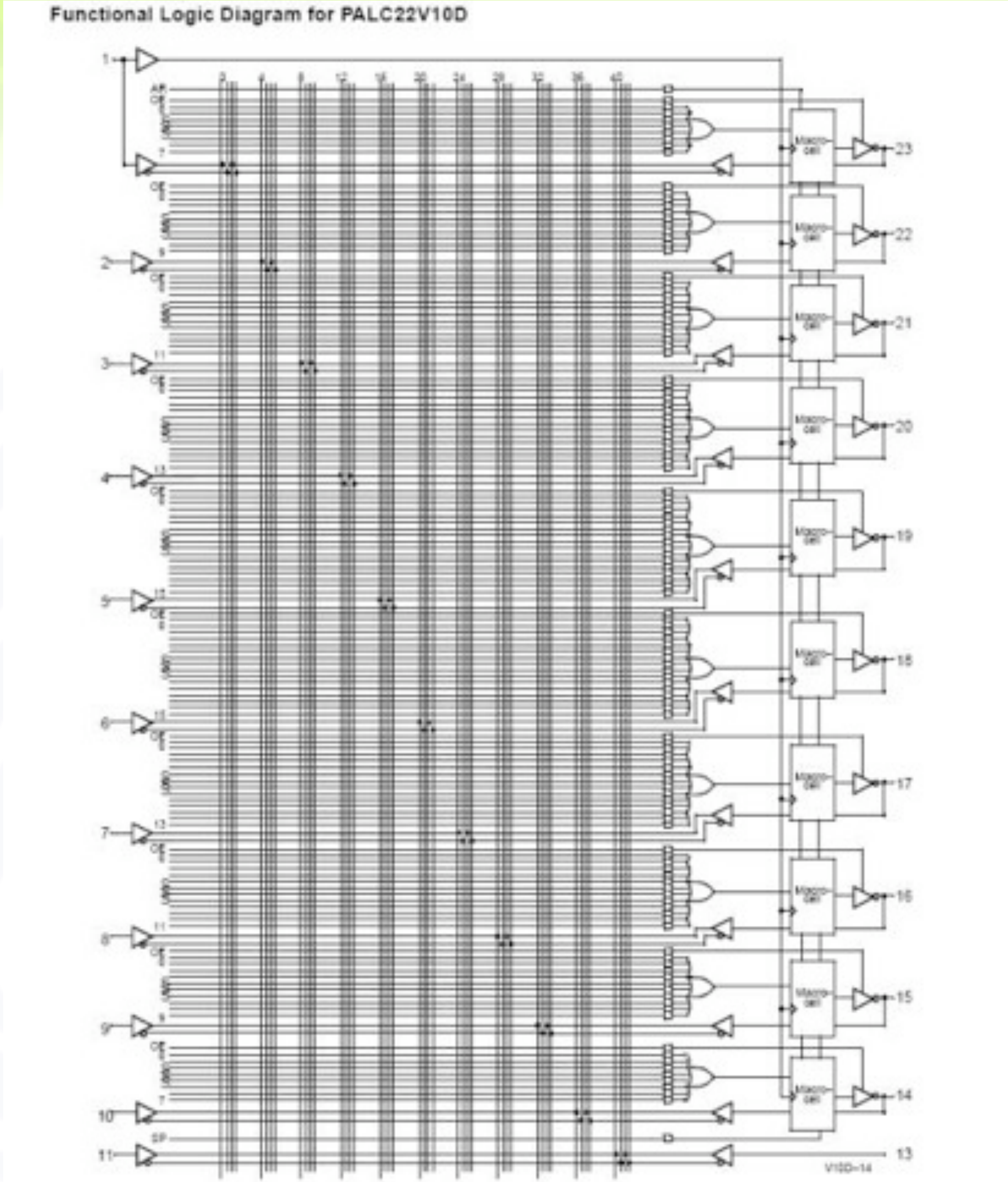
Vending machine (synch. Mealy PLD mapping)

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



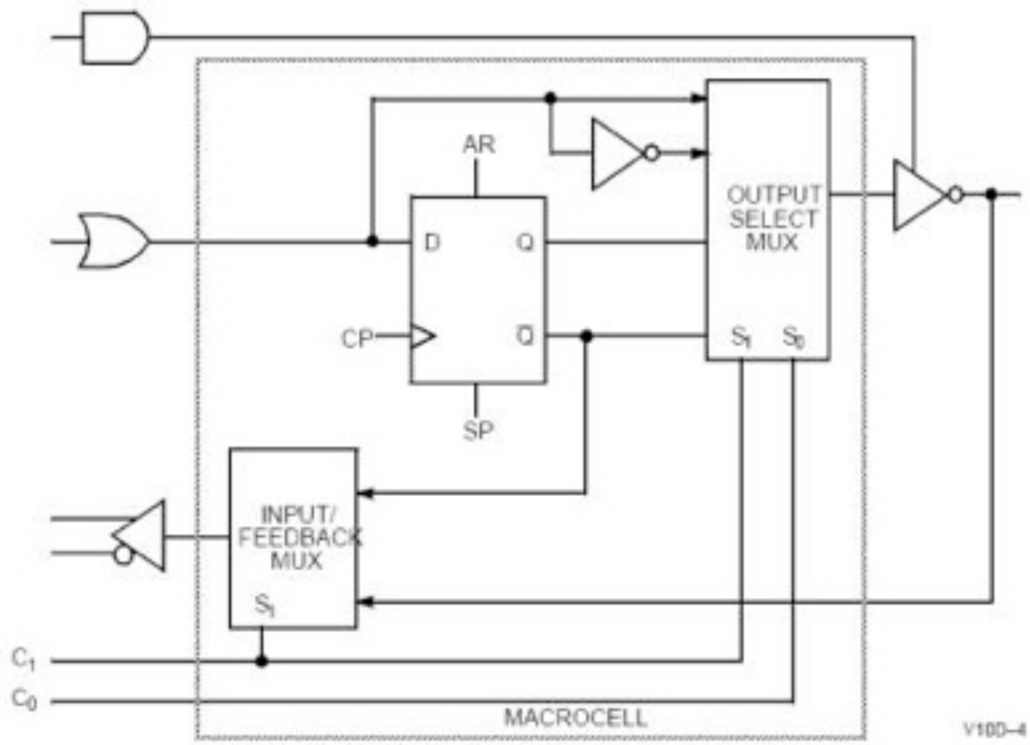
22V10 PAL

- 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

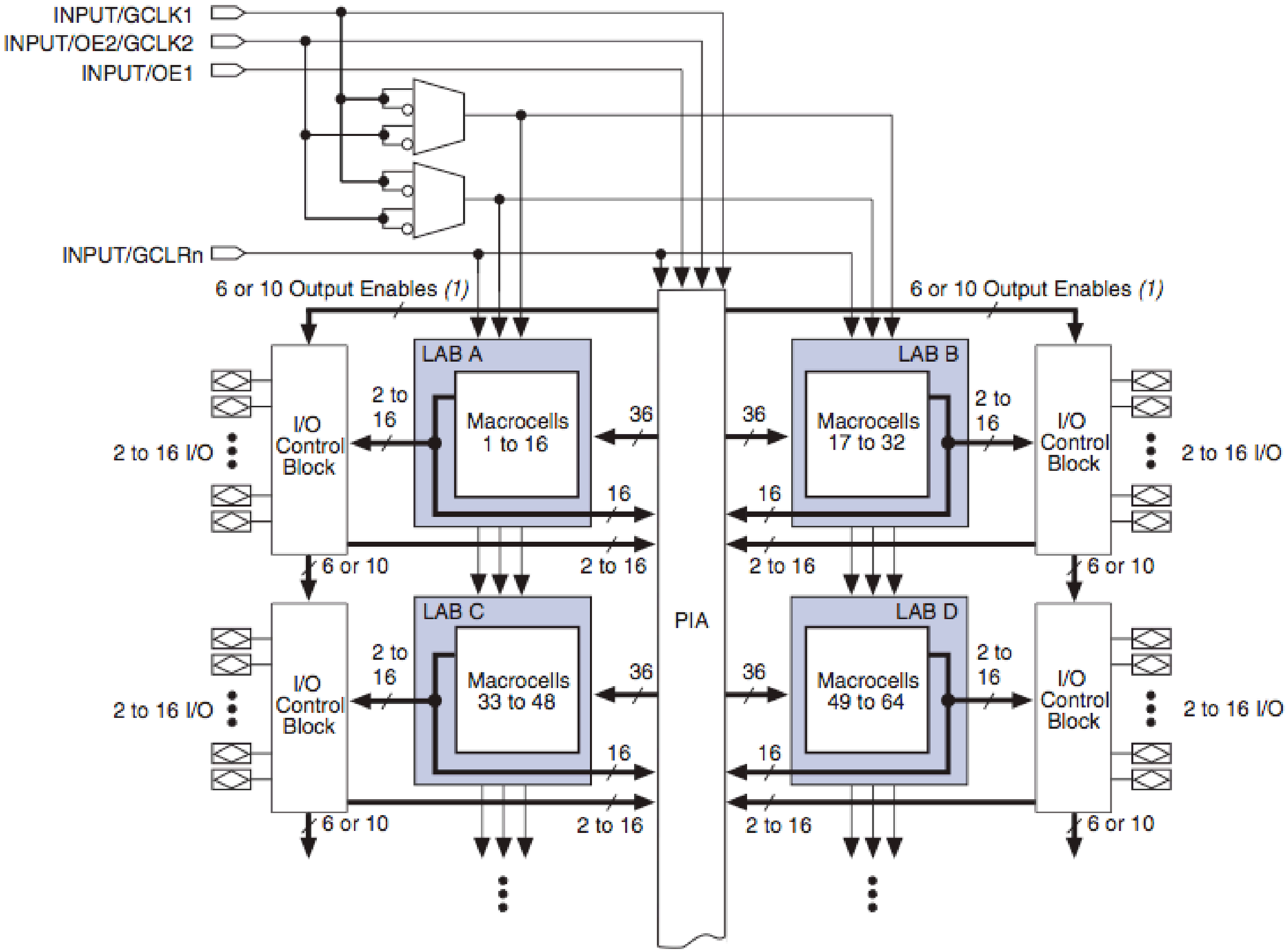
- Sequential logic element + output/input selection



Embedded Low-Power Laboratory

FSM implementation with an FPGA

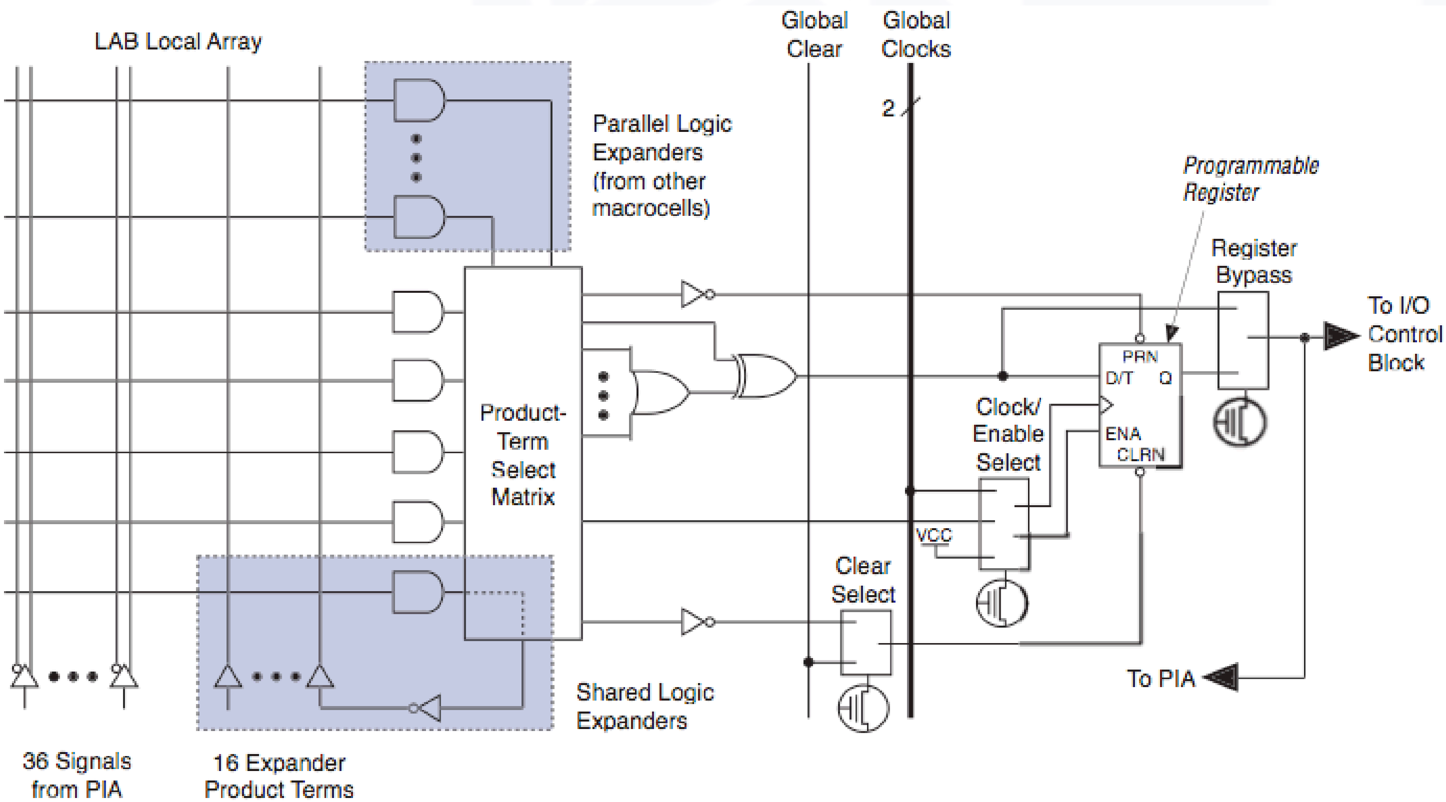
- Altera MAX 3000 CPLD architecture



Embedded Low-Power Laboratory

FSM implementation with an FPGA

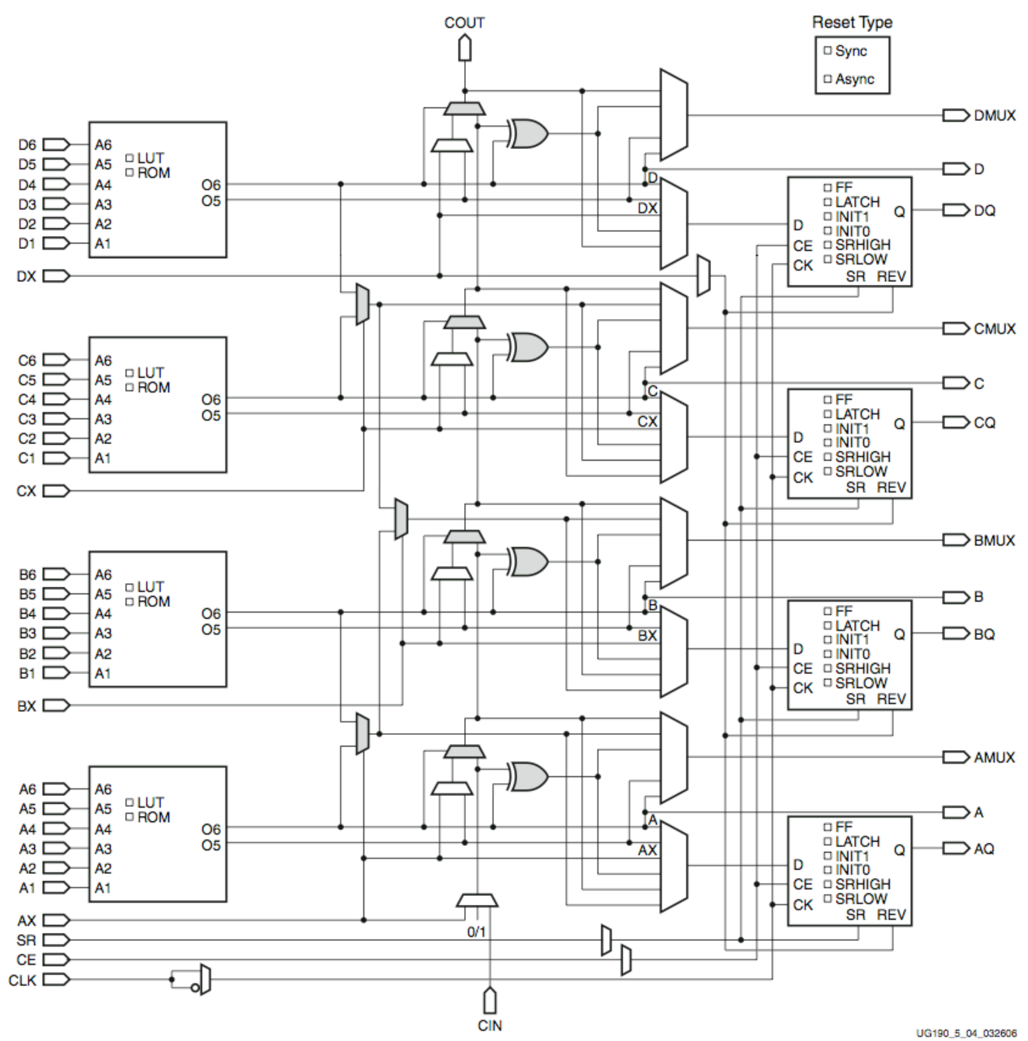
- Altera MAX 3000 CPLD macrocell



Embedded Low-Power Laboratory

FSM implementation with an FPGA

- Xilinx Vertex-5 slice

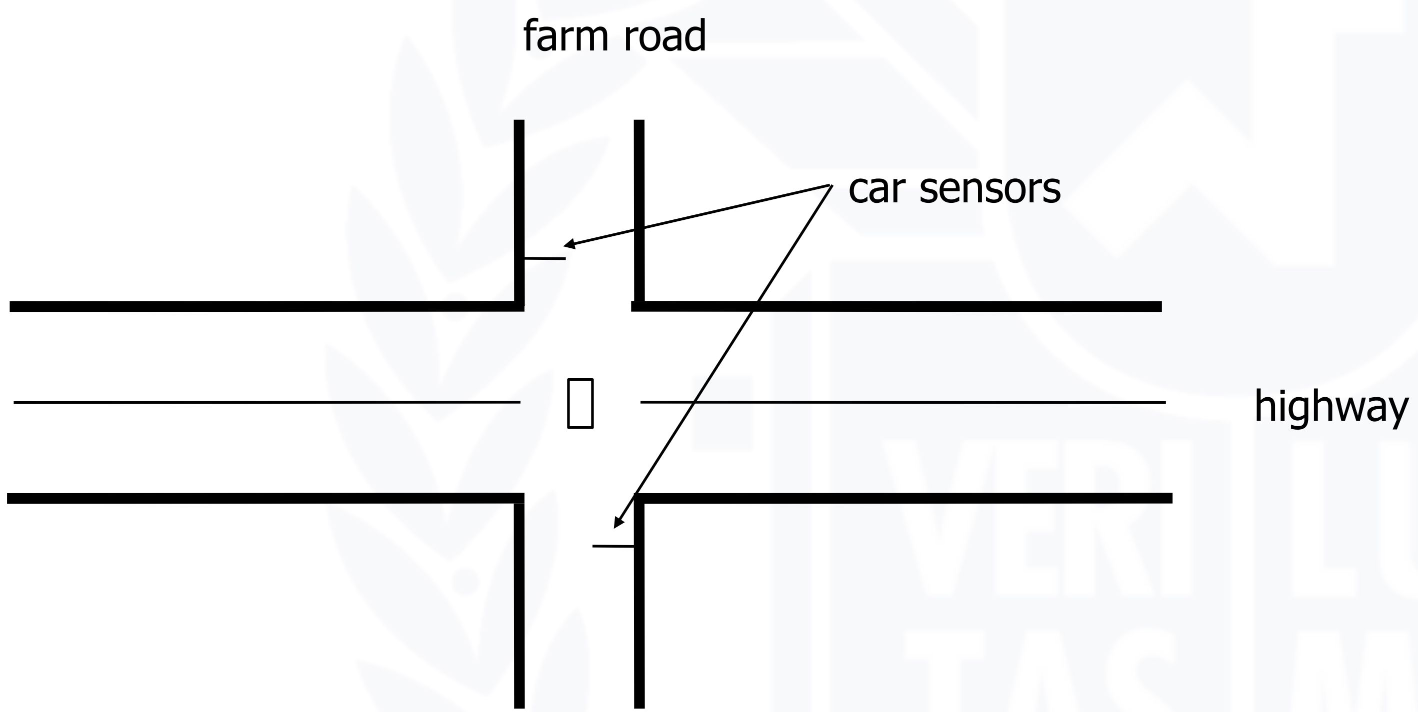


Example: traffic light controller

- 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 (cont’)

- Highway/farm road intersection



Friday, March 19, 2010

Example: traffic light controller (cont')

- Tabulation of inputs and outputs

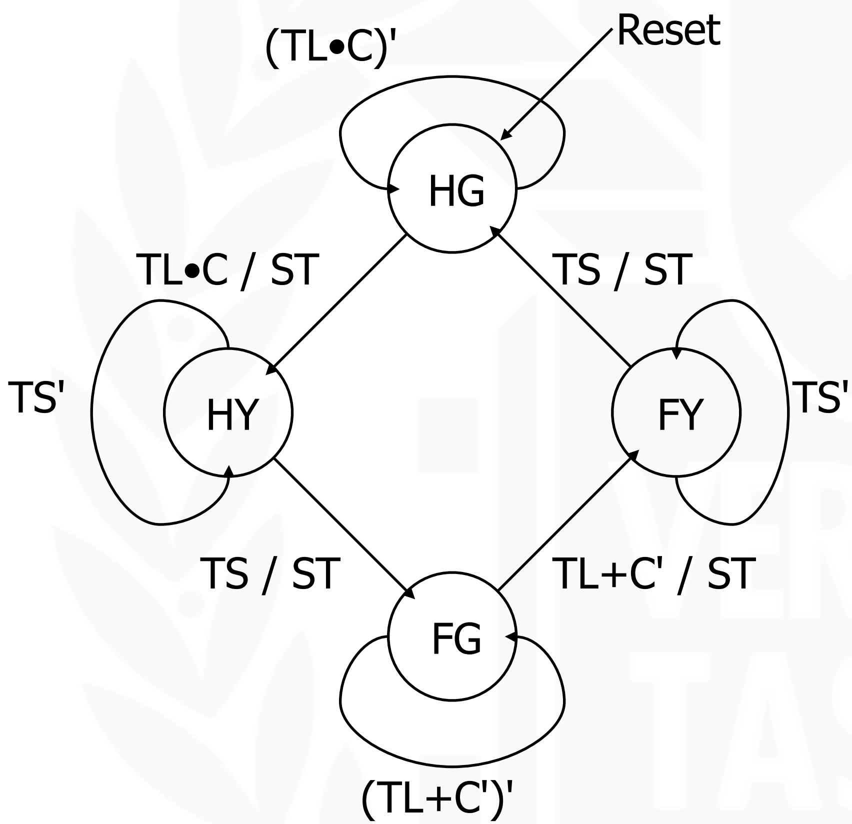
inputs	description	outputs	description
reset	place FSM in initial state	HG, HY, HR	assert green/yellow/red highway lights
C	detect vehicle on the farm road	FG, FY, FR	assert green/yellow/red highway lights
TS	short time interval expired	ST	start timing a short or long interval
TL	long time interval expired		

- 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 (cont’)

- State diagram



Embedded Low-Power Laboratory

Friday, March 19, 2010

Example: traffic light controller (cont')

- Generate state table with symbolic states
- Consider state assignments

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

Inputs C	TL	TS	Present State	Next State	Outputs 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)

Embedded Low-Power Laboratory

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 implementation 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)