Ch 9. Sequential Logic Technologies

Overview

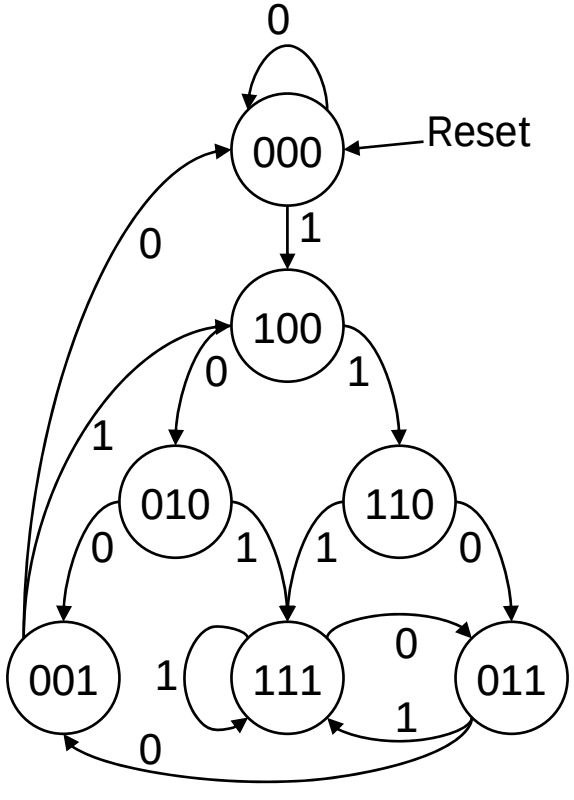
- Basic Sequential Logic Components
- FSM Design with Counters
- FSM Design with Programmable Logic
- FSM Design with More Sophisticated Programmable Logic
- Case Study: Traffic Light Controller

Sequential logic implementation

- Implementation
  - discrete logic gates and flip-flops
  - ROMs or PALs/PLAs
  - CPLDs or FPGAs
- Design procedure
  - state diagram
  - state transition table
  - state assignment
  - next state and output functions

Median filter FSM

- Remove single 0s between two 1s (output = PS3)



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

Median filter FSM (cont'd)

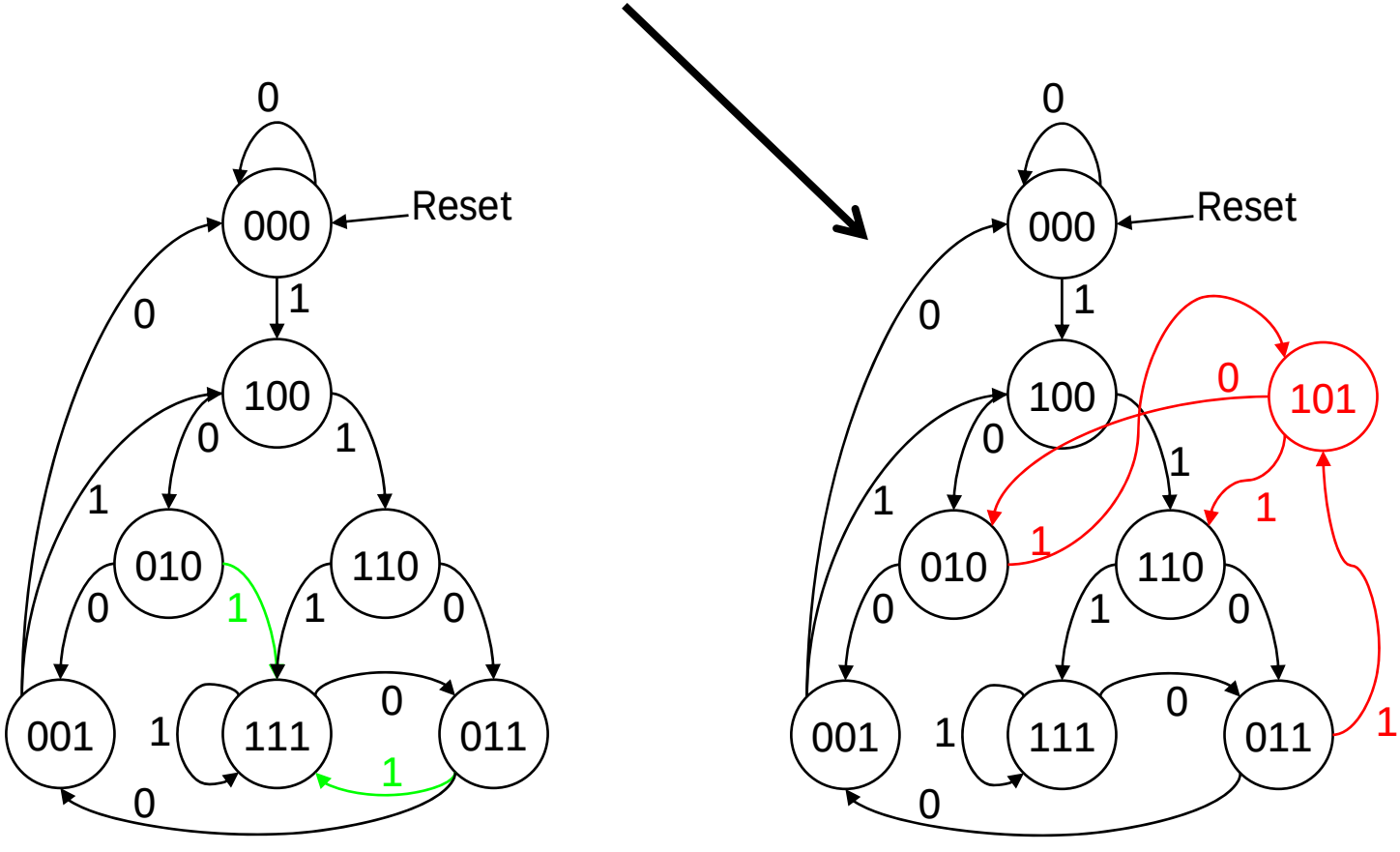
- Realized using the standard procedure and individual FFs and gates

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

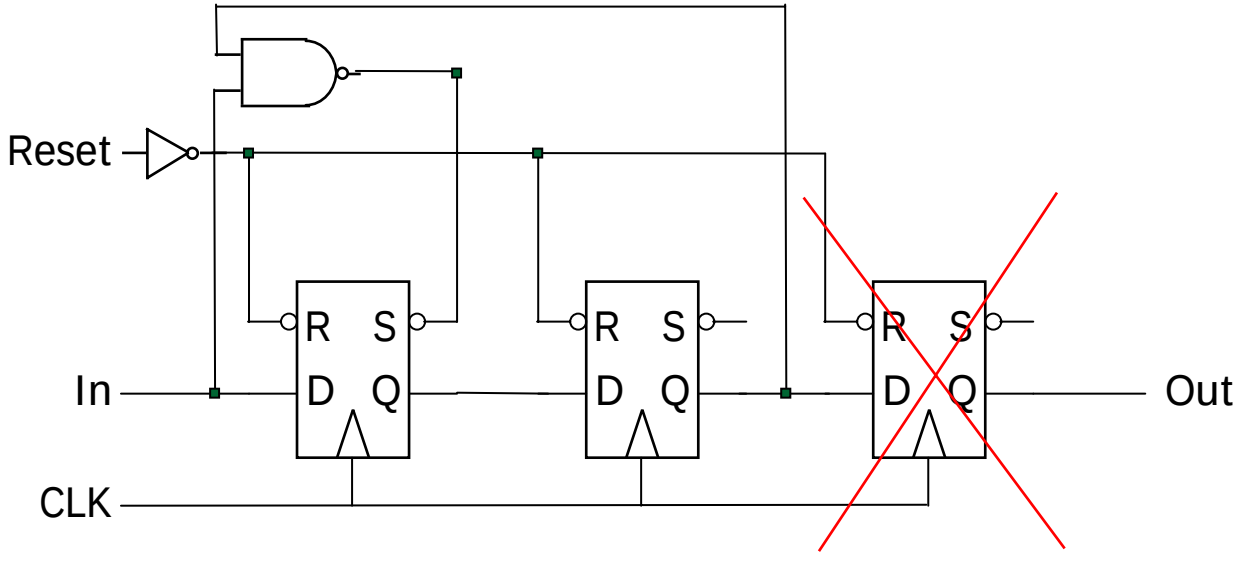
Median filter FSM (cont'd)

- 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

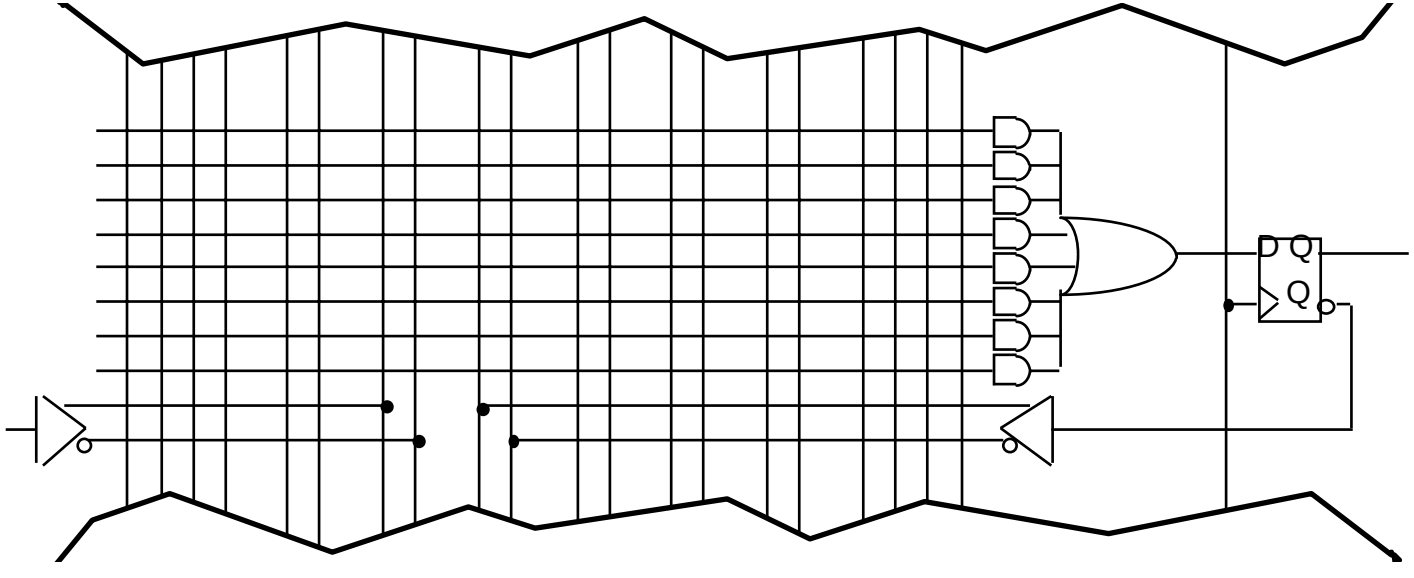


R = Reset
S = PS2 I
NS1 = I
NS2 = PS1
NS3 = PS2
O = PS3

- 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)

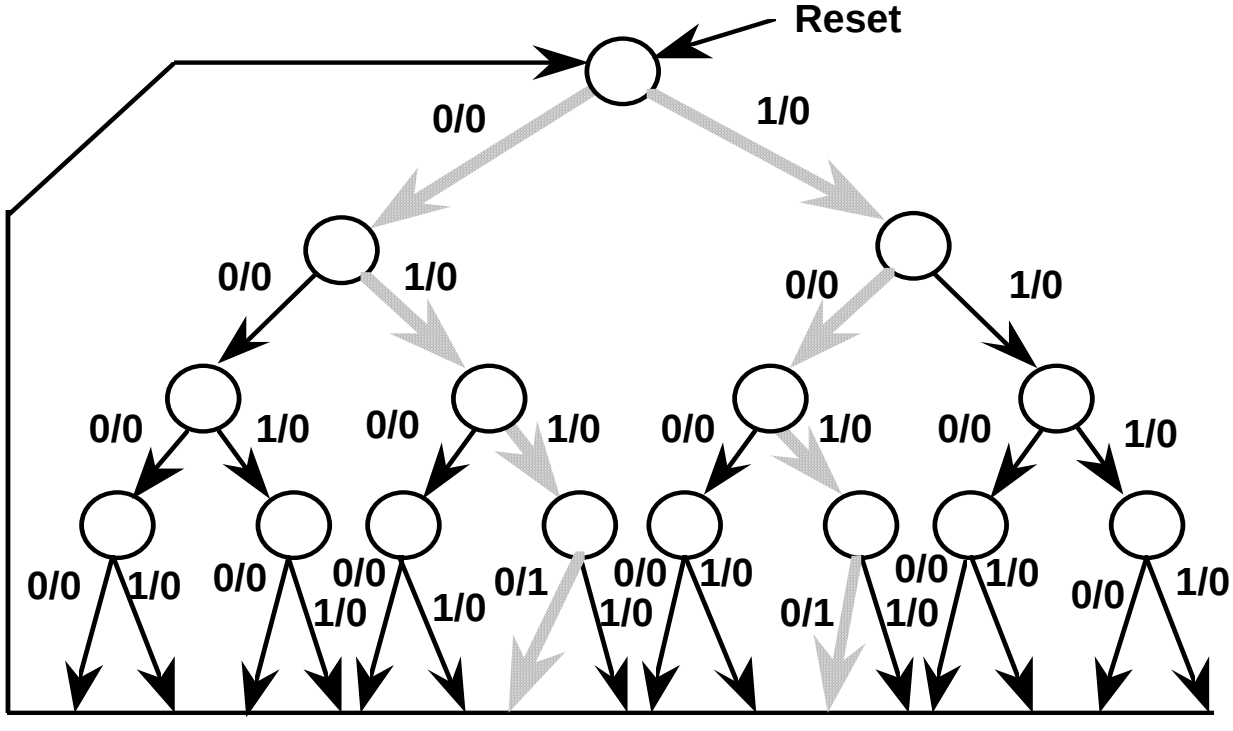
Implementation using PALs

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

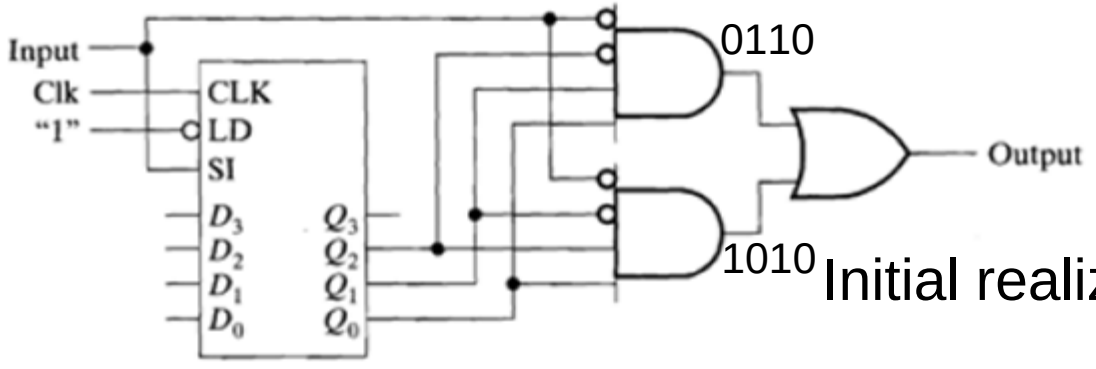


Using a Shift Register

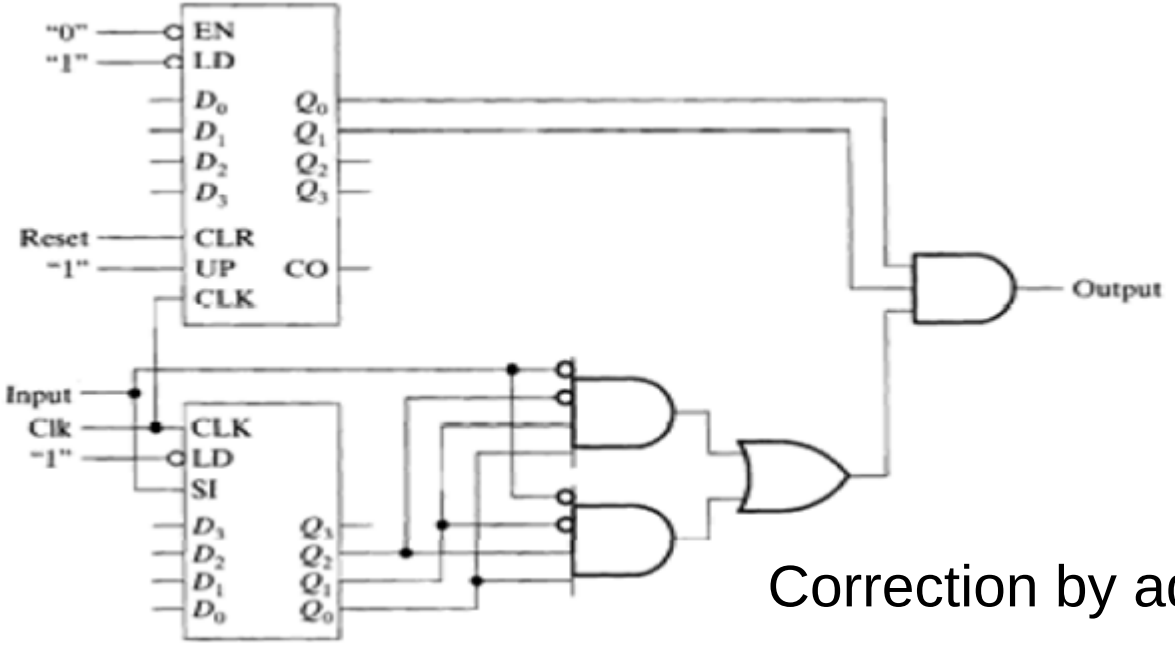
- 4-bit string (0110, 1010) recognizer



Using a Shift Register



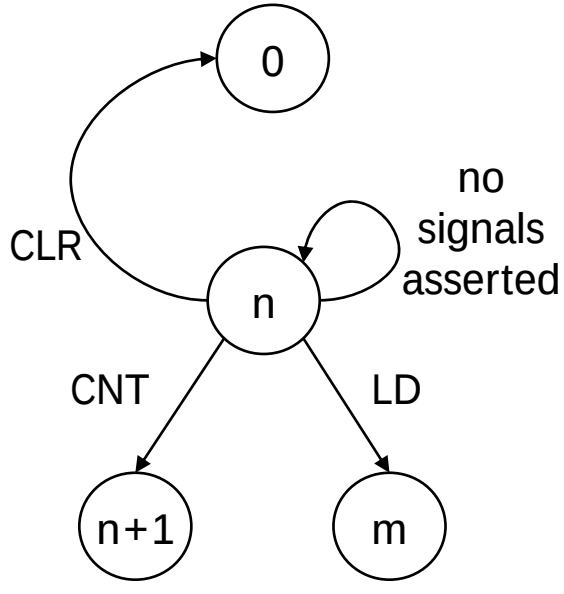
Initial realization



Correction by adding a counter

FSM Design with Counters

- Synchronous counters: CLR, LD, CNT
- Four kinds of transition for each state:
  - To State 0 (CLR)
  - To next state in sequence (CNT)
  - To arbitrary next state (LD)
  - Loop in current state



- Careful state assignment is needed to reflect basic sequencing of the counter

BCD to Excess 3 Serial Converter

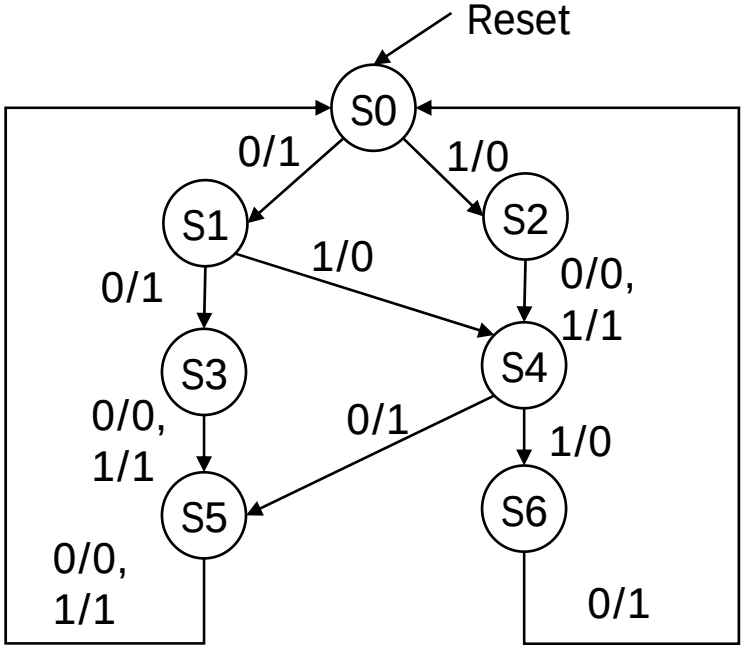
- Conversion Process

  Bits are presented in bit serial fashion starting with the least significant bit

  Single input X, single output Z

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

BCD to Excess 3 Serial Converter (Cont'd)



State Diagram

		Next State		Output	
Present State		**X=0**	**X=1**	**X=0**	**X=1**
S0	000 (0)	**S1**	**S2**	**1**	**0**
S1	001 (1)	**S3**	**S4**	**1**	**0**
S2	100 (4)	**S4**	**S4**	**0**	**1**
S3	010 (2)	**S5**	**S5**	**0**	**1**
S4	101 (5)	**S5**	**S6**	**1**	**0**
S5	011 (3)	**S0**	**S0**	**0**	**1**
S6	110 (6)	**S0**	**--**	**1**	**--**

State Transition Table

Note the sequential nature of the state assignment

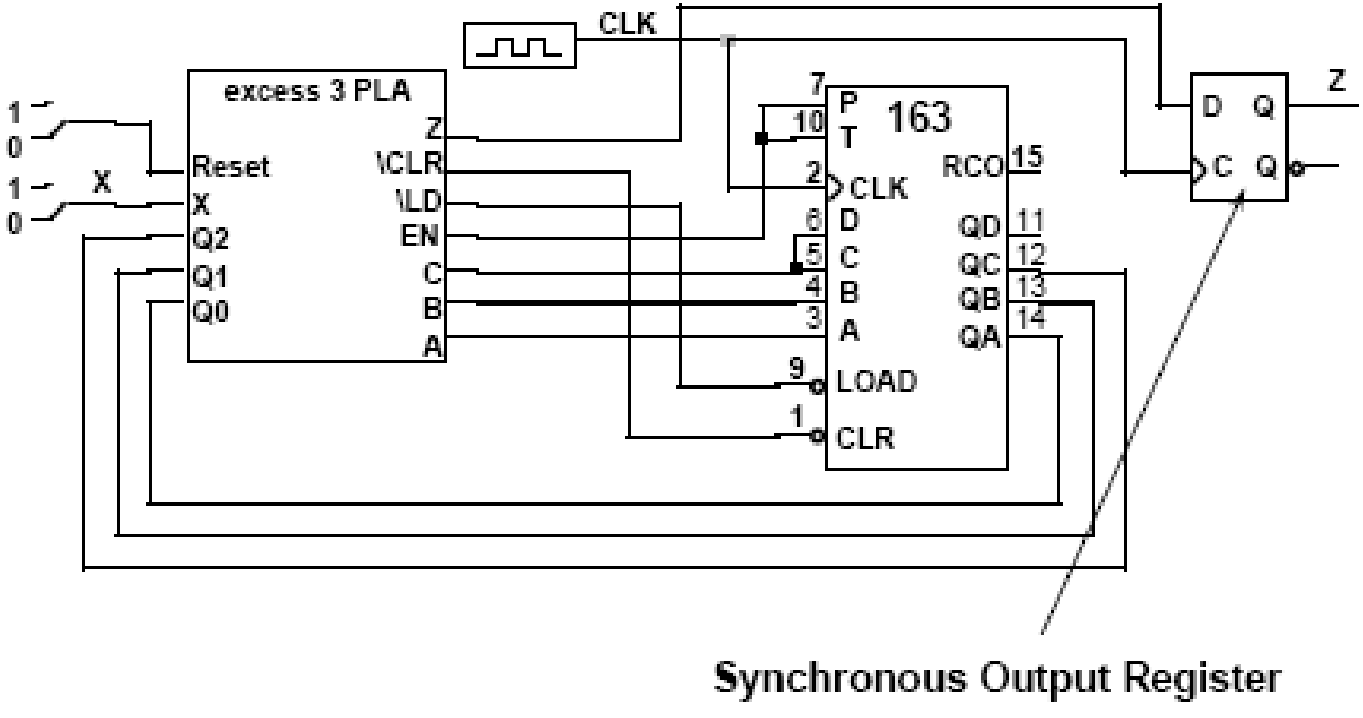
Serial Converter: Transition Table

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

CLR signal dominates LD which dominates Count

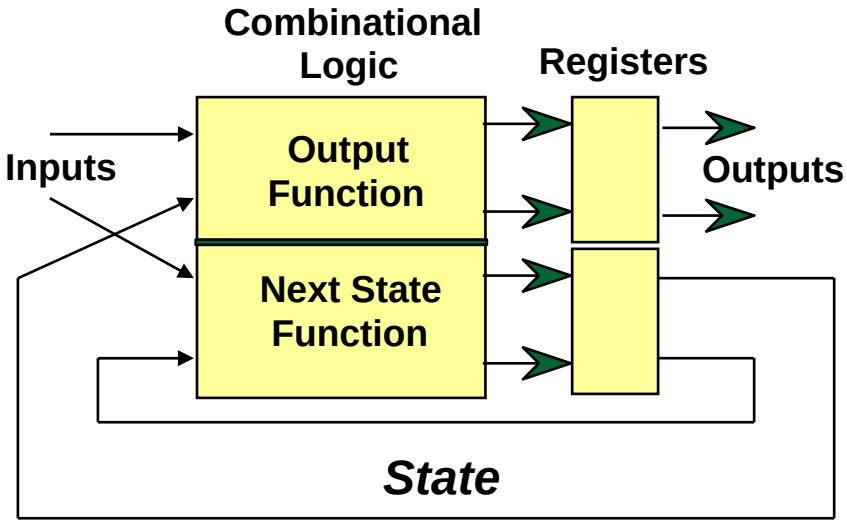
Serial Converter (Cont'd)

- Counter-based implementation of code converter

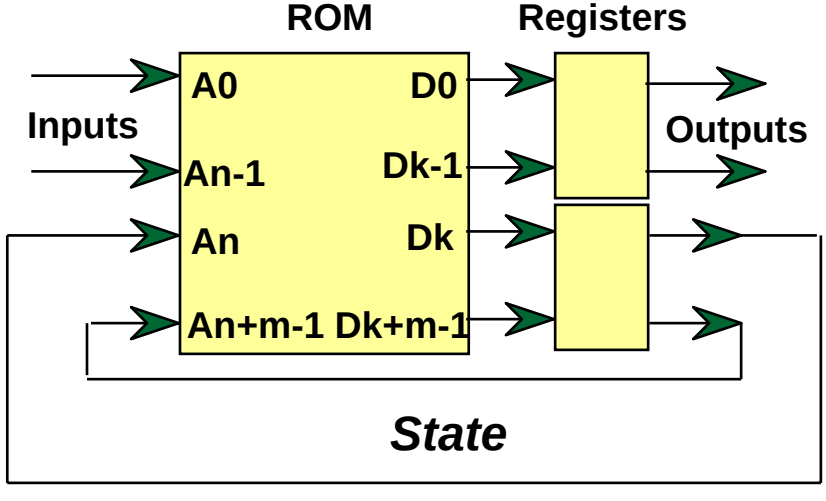


When the state diagram has fewer out-of-sequence jumps, a counter based implementation can be very effective

Rom-Based Design



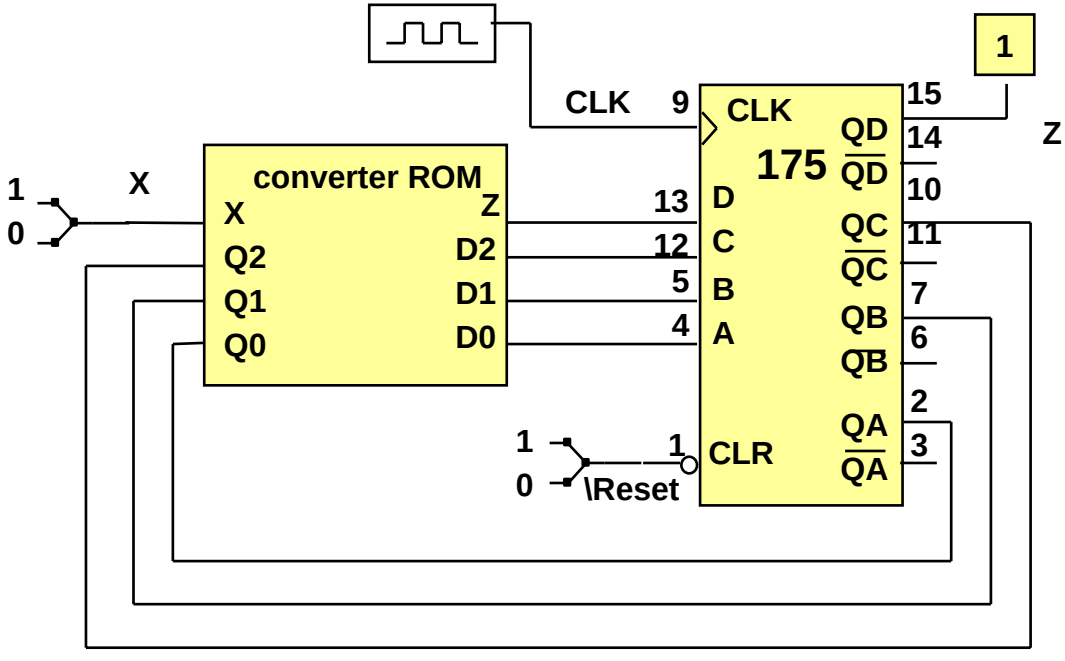
- Block Diagram for Synchronous Mealy Machine



- ROM-based Realization
  - Inputs & Current State form the address
  - ROM data bits form the Outputs & Next State

Rom-Based Design (Cont'd)

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

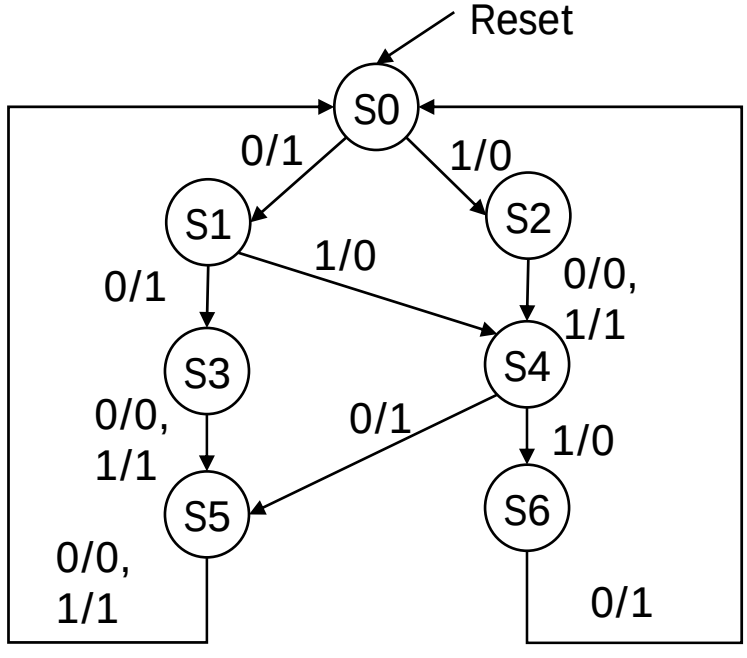


Excess-3 synchronous Mealy ROM-based implementation

PLA-Based Design

- State assignment with NOVA

S0 = 000
S1 = 001
S2 = 011
S3 = 110
S4 = 100
S5 = 111
S6 = 101



PLA-Based Design (Cont'd)

```
.i 4
.o 4
.ilb x q2 q1 q0
.ob d2 d1 d0 z
.p 16
0 000 001 1
1 000 011 0
0 001 110 1
1 001 100 0
0 011 100 0
1 011 100 1
0 110 111 0
1 110 111 1
0 100 111 1
1 100 101 0
0 111 000 0
1 111 000 1
0 101 000 1
1 101 --- -
0 010 --- -
1 010 --- -
.e
```

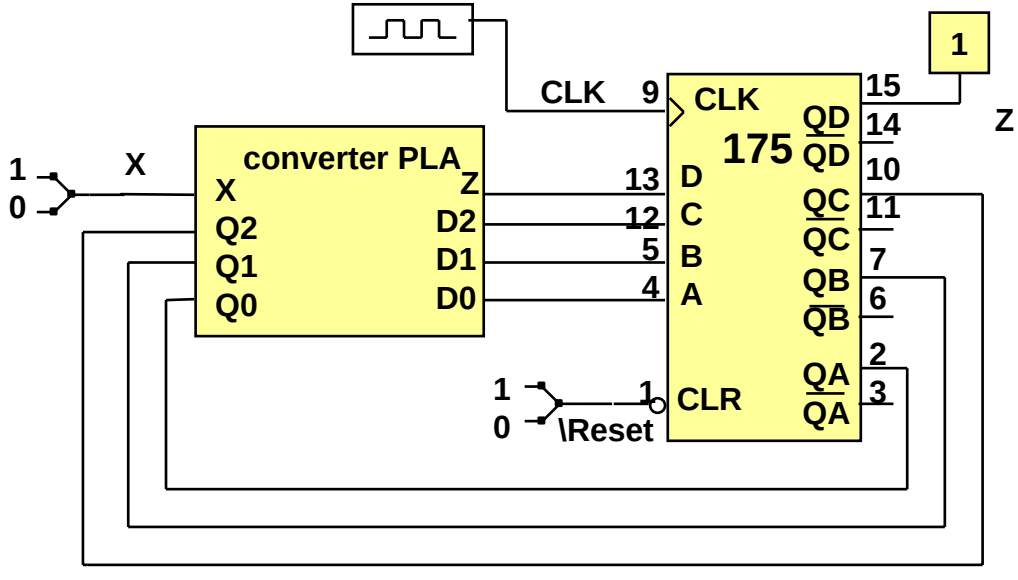
Espresso Inputs

```
.i 4
.o 4
.ilb x q2 q1 q0
.ob d2 d1 d0 z
.p 9
0001 0100
10-0 0100
01-0 0100
1-1- 0001
-0-1 1000
0-0- 0001
-1-0 1000
--10 0100
---0 0010
.e
```

Espresso Outputs

$Q2^+ = Q2'Q0 + Q2Q0'$
$Q1^+ = X'Q2'Q1'Q0 + XQ2'Q0'$
$\quad + X'Q2Q0' + Q1Q0'$
$Q0^+ = Q0'$
$Z = XQ1 + X'Q1'$

9 product term implementation



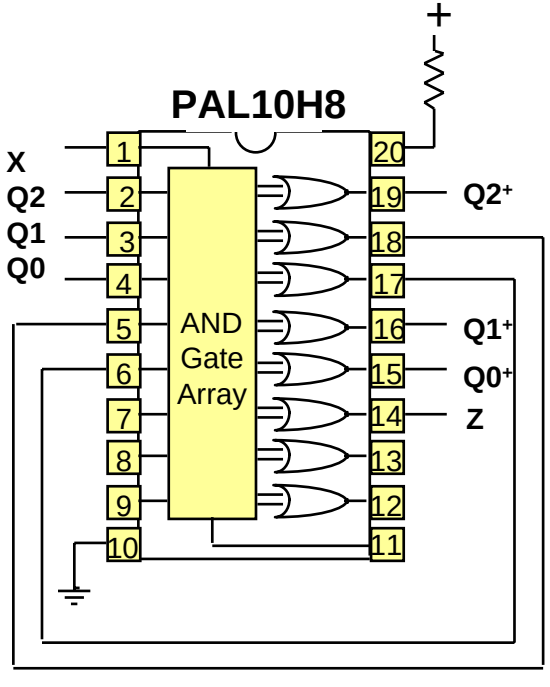
PAL-Based Design (Cont'd)

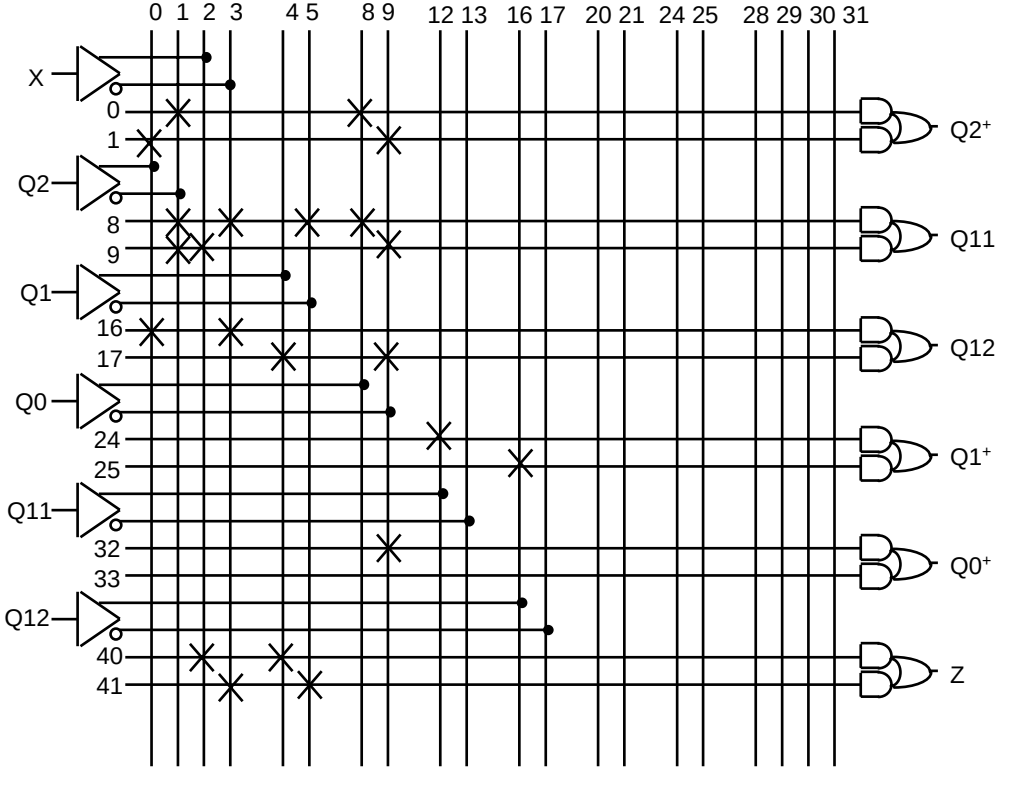
- PAL10H8: 10 inputs, 8 outputs, 2 product terms per OR gate

Q2+ = Q2'Q0 + Q2Q0'
Q1+ = X'Q2'Q1'Q0 + XQ2'Q0' + X'Q2Q0' + Q1Q0'
Q0+ = Q0'
Z = XQ1 + X'Q1'

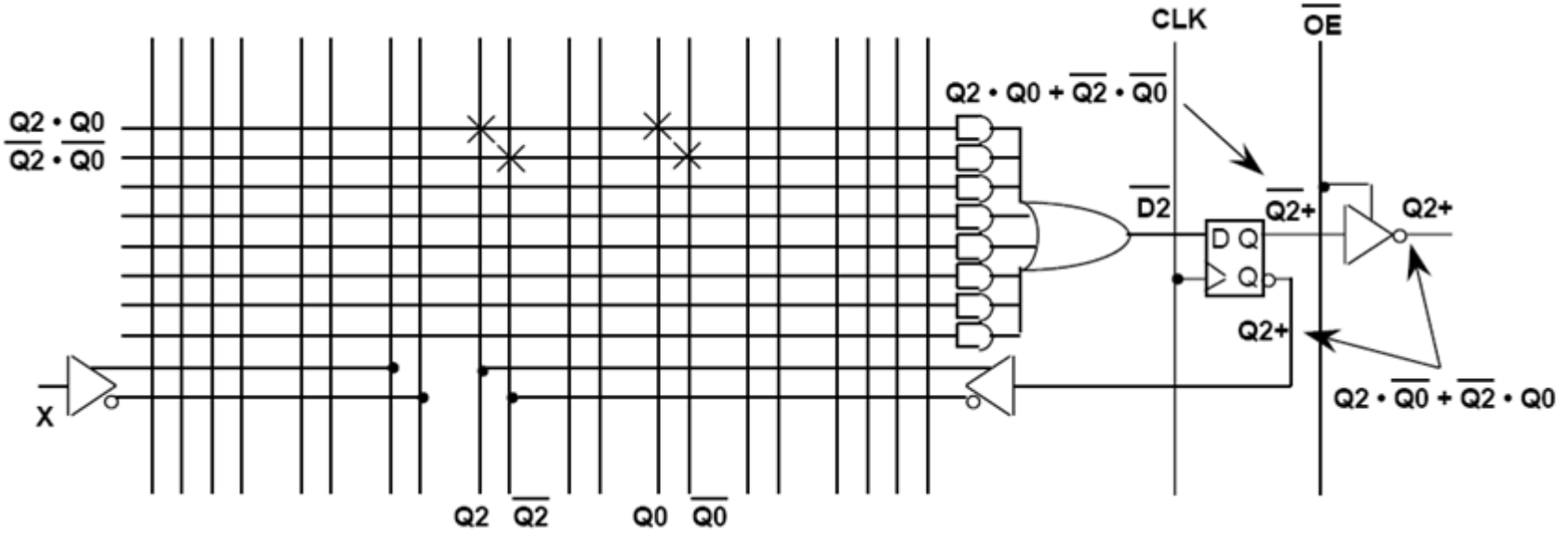
→

Q1+ = Q11+Q12
Q11 = X'Q2'Q1'Q0 + XQ2'Q0'
Q12 = X'Q2Q0' + Q1Q0'





Alternative PAL Architectures



D2’ = Q2Q0 + Q2’Q0’

D1’ = X’Q2’Q1’Q0’ + XQ2 + XQ0 + Q2Q0 + Q1Q0

D0’ = Q0

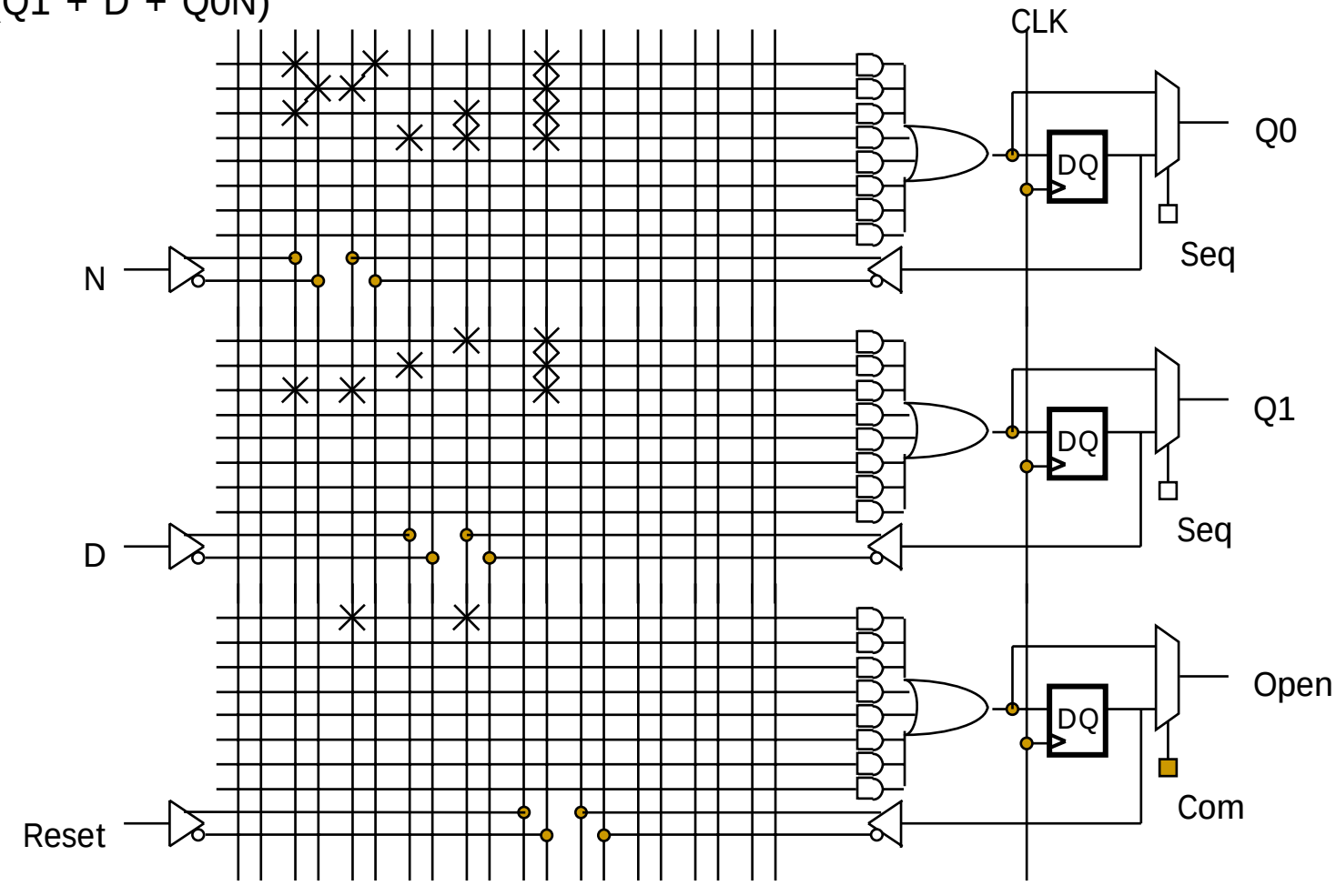
Z’ = XQ1’ + X’Q1

Vending machine example (Moore PLD mapping)

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

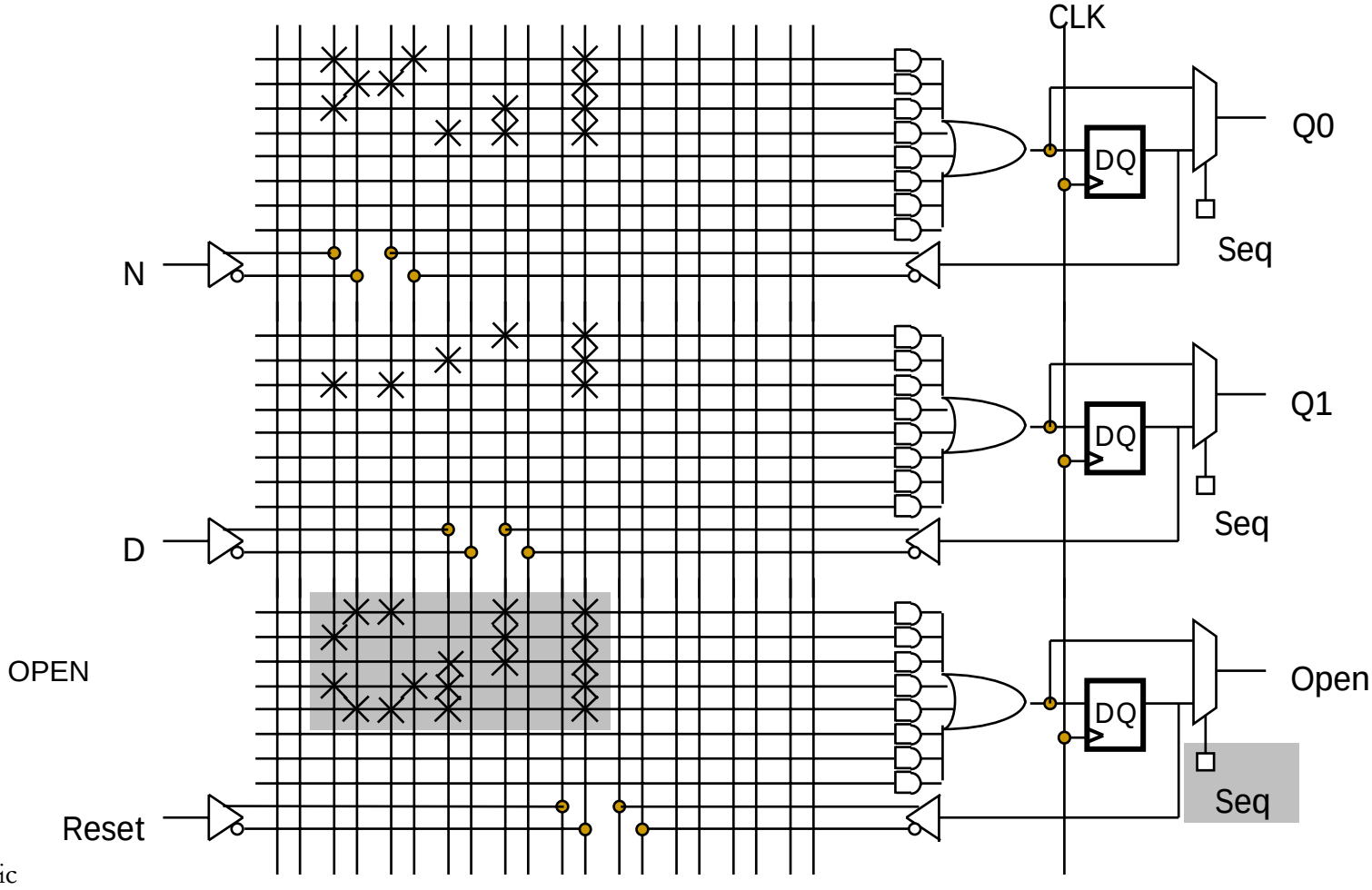
D1 = reset'(Q1 + D + Q0N)

OPEN = Q1Q0



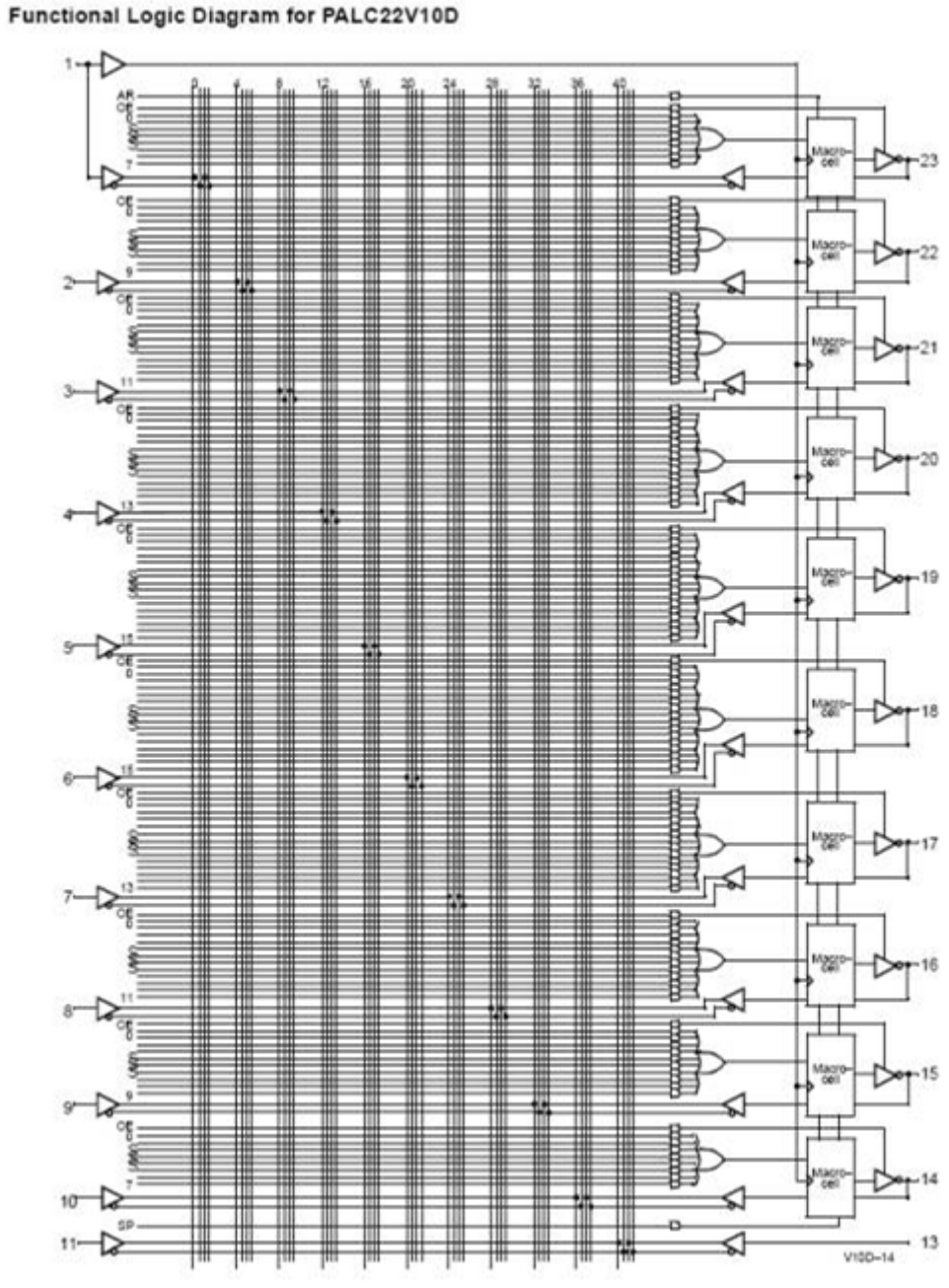
Vending machine (synch. Mealy PLD mapping)

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



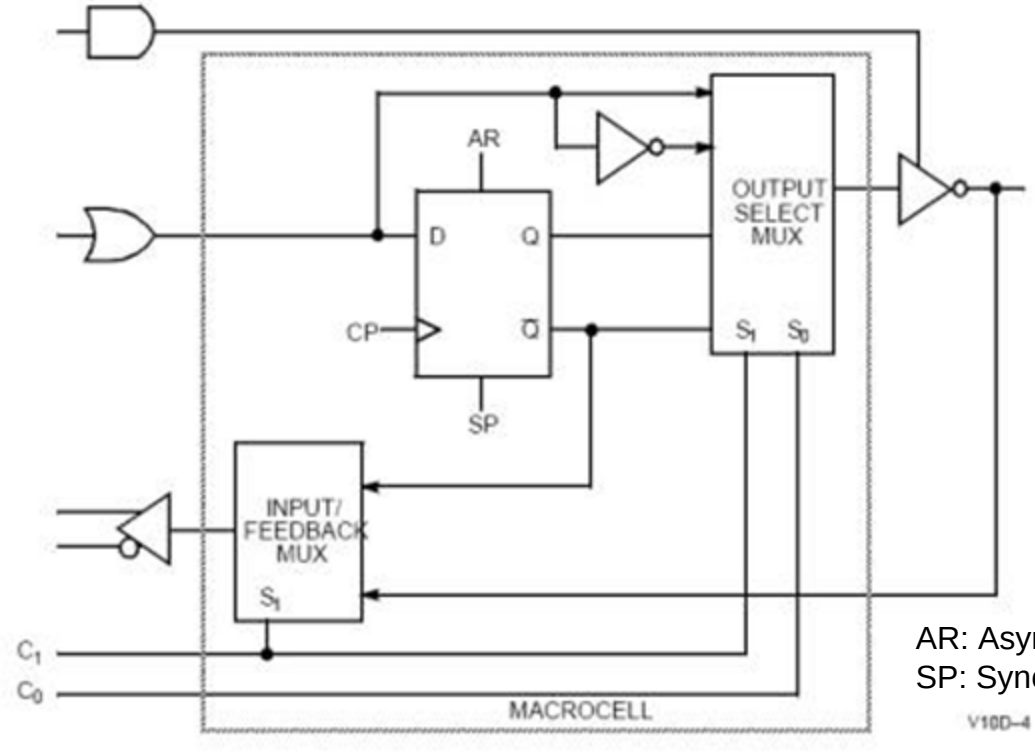
22V10 PAL

- 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

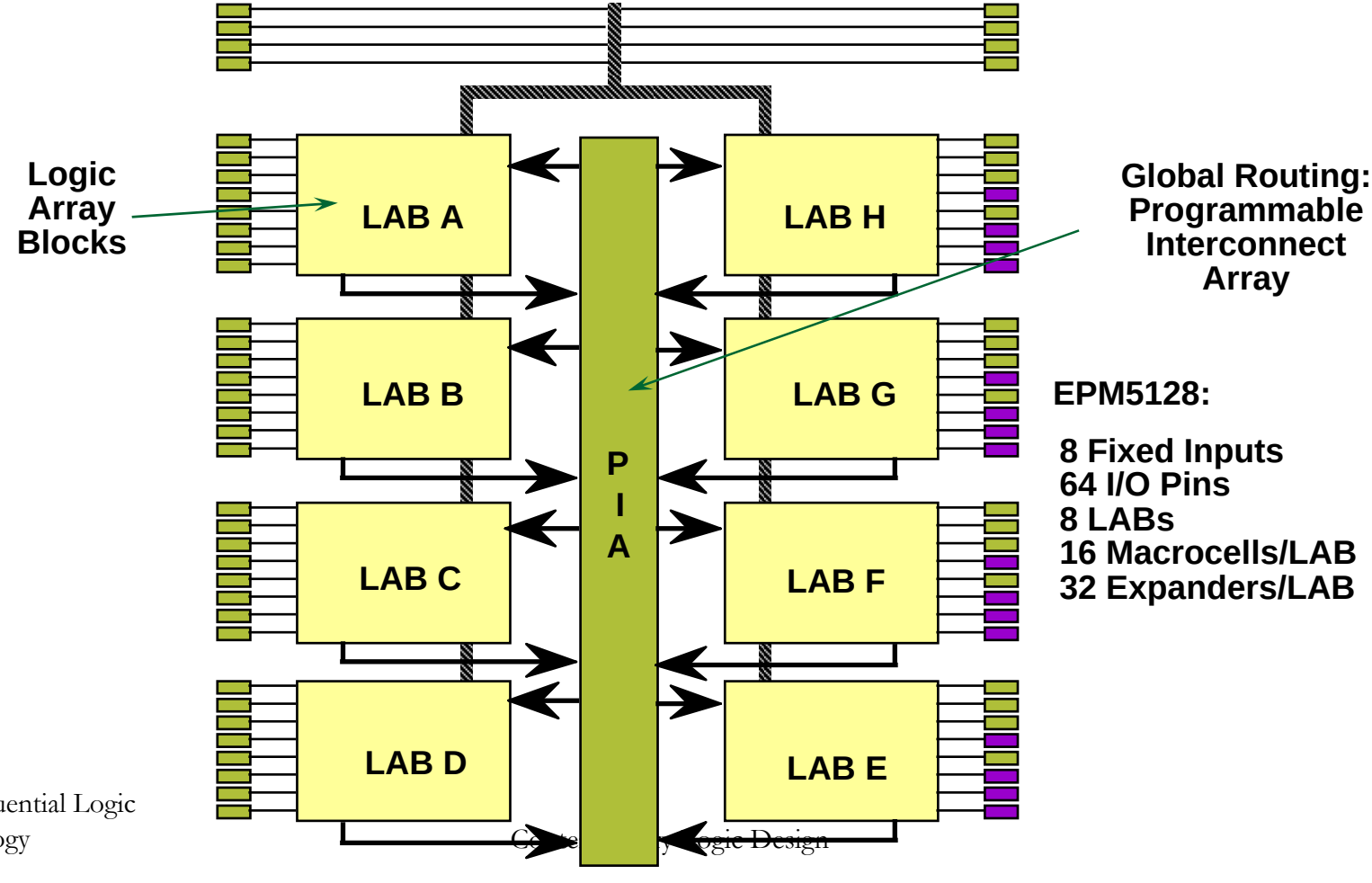
- Sequential logic element + output/input selection



AR: Asynchronous Reset
SP: Synchronous Preset

CPLD (Complex Programmable Logic Device)

- Altera MAX (Multiple Array Matrix) Family
  - Array of EPLD + Programmable Interconnect Array
  - EPROM technology



FPGA (Field Programmable Gate Array)

- Array of programmable logic-blocks with programmable interconnects
- Actel Programmable Gate Array
  - Multiplexer-based logic blocks
  - Anti-fuse technology for interconnects
- Xilinx Logic Cell Array
  - SRAM-based logic blocks and interconnects

FPGA (Cont'd)

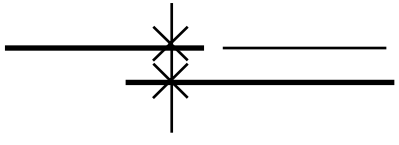
- Actel Programmable Gate Arrays

Rows of programmable logic building blocks

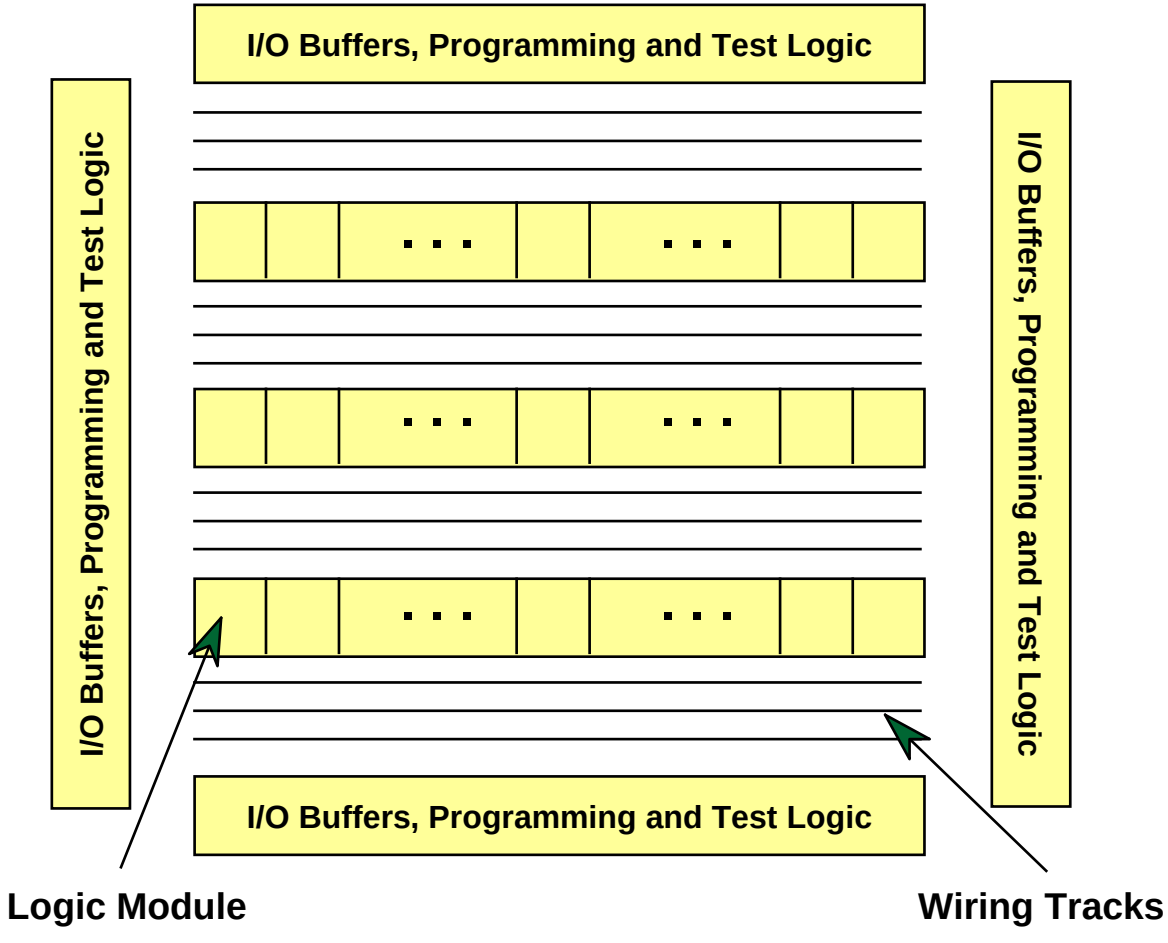
+

rows of interconnect

Anti-fuse Technology: Program Once



Use Anti-fuses to build up long wiring runs from short segments

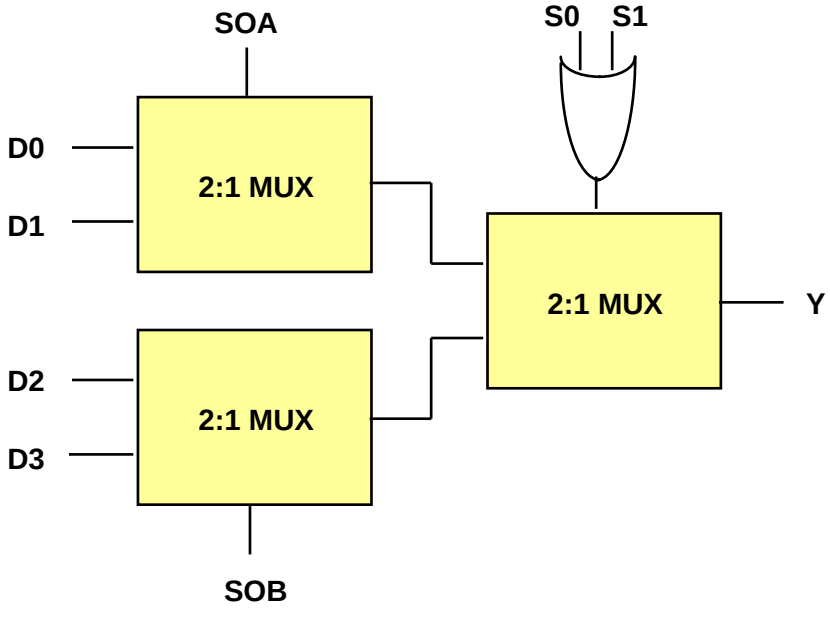


8 input, single output combinational logic blocks

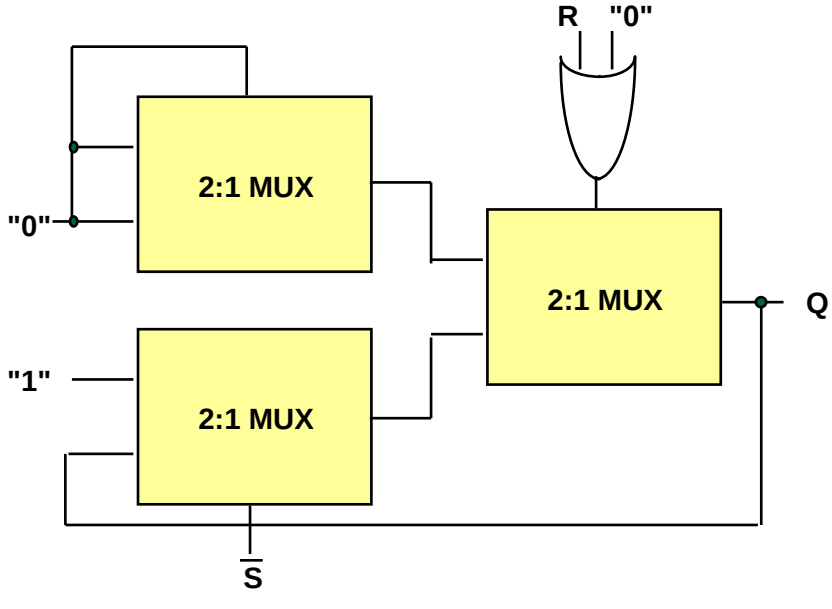
FFs constructed from discrete cross coupled gates

FPGA (Cont'd)

- Actel Logic Module



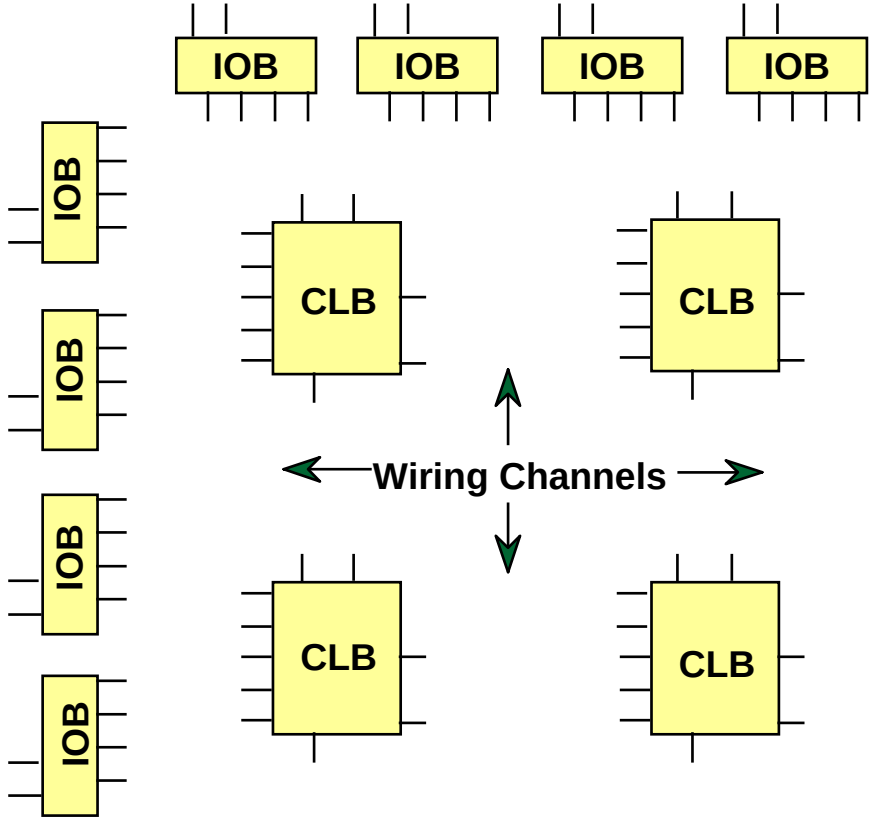
Basic Module is a Modified 4:1 Multiplexer



Example:
Implementation of $\overline{S}$-$\overline{R}$ Latch

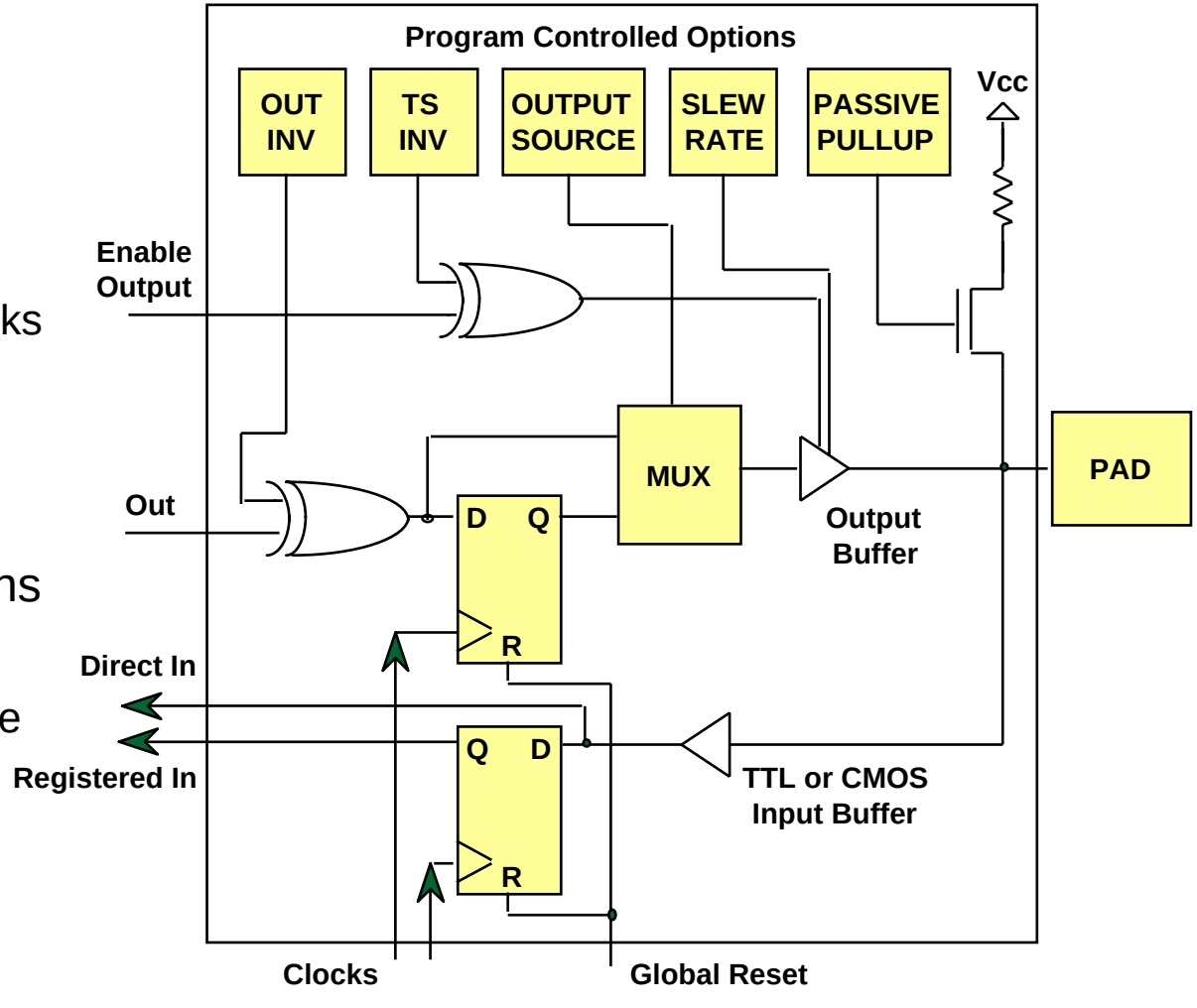
FPGA (Cont'd)

- Xilinx Logic Cell Arrays
  - CMOS Static RAM Technology: programmable on the fly!
  - All personality elements connected into serial shift register
  - Shift in string of 1's and 0's on power up
  - General Chip Architecture:
    - Logic Blocks (CLBs)
    - IO Blocks (IOBs)
    - Wiring Channels



FPGA (Cont'd)

- IO block
  - Inputs:
    - Tri-state enable
    - bit to output
    - input, output clocks
  - Outputs:
    - input bit
  - Internal FFs for input & output paths
  - Fast/Slow outputs
    - 5 ns vs. 30 ns rise
  - Pull-up used with unused IOBs



FPGA (Cont'd)

- Xilinx LCA Architecture
  - Configurable Logic Block (CLB)

2 4-bit address SRAMs

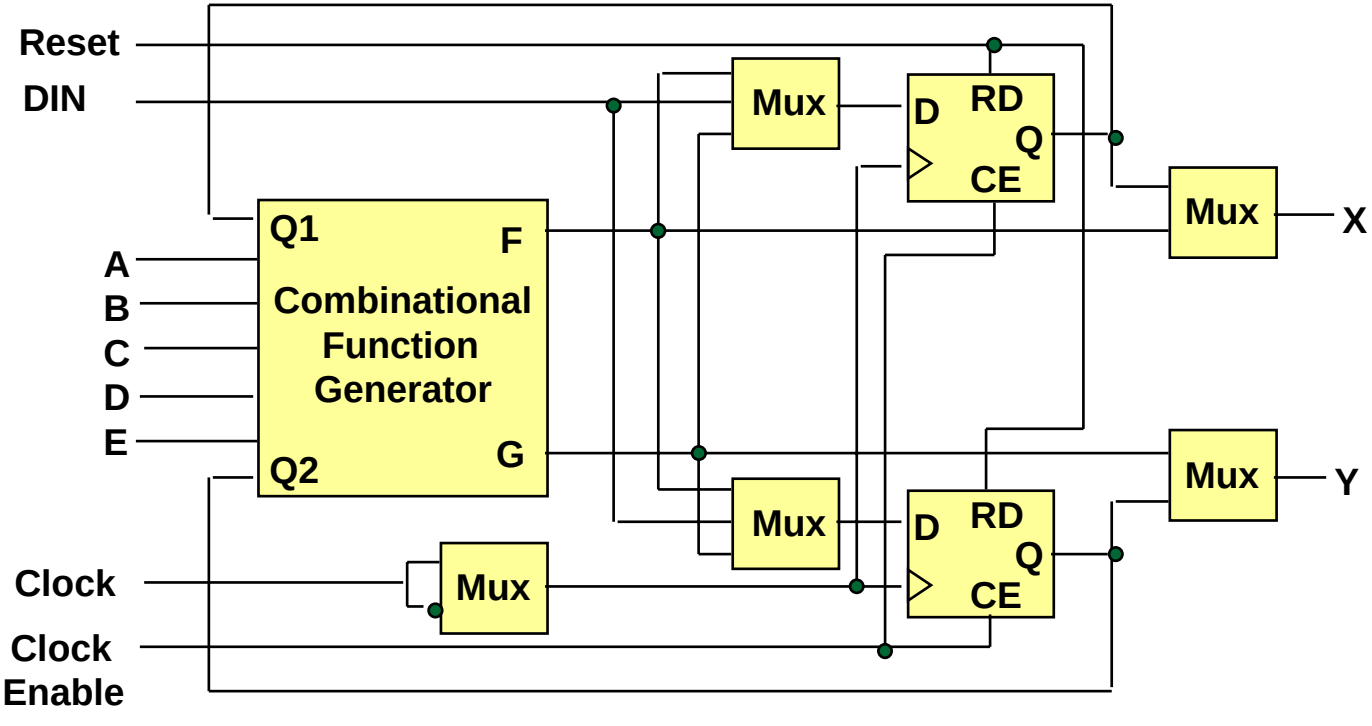
2 FFs

Any function of 5 Variables

Global Reset

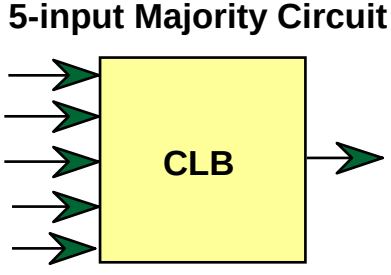
Clock, Clock Enb

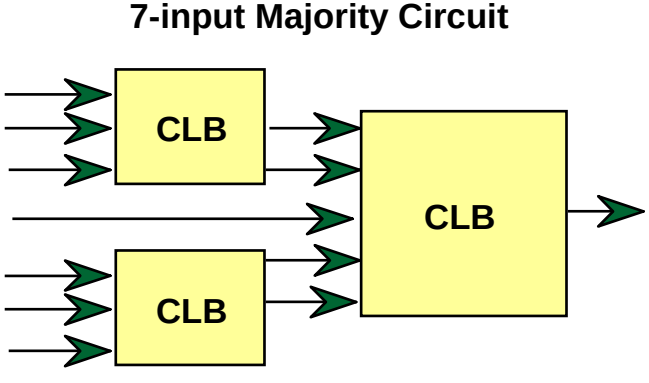
Independent DIN

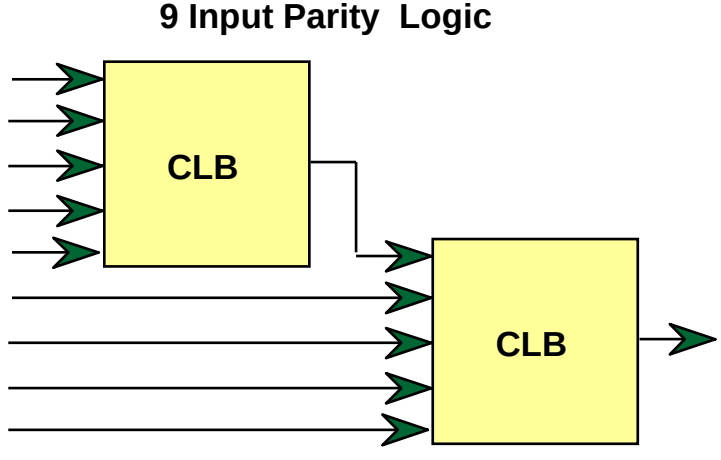


FPGA (Cont'd)

- n-input Majority Circuit
  - Assert 1 whenever n/2 or greater inputs are 1
- n-input Parity Functions
  - 5 input = 1 CLB, 2 Levels of CLBs yield up to 25 inputs!

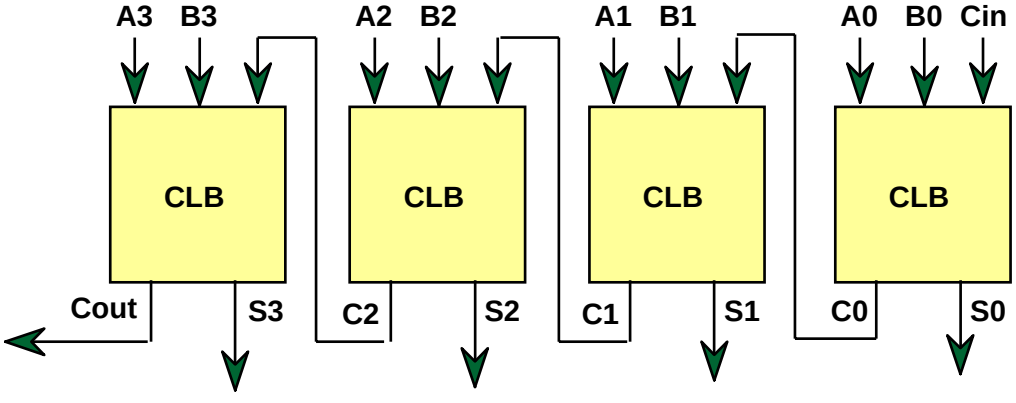




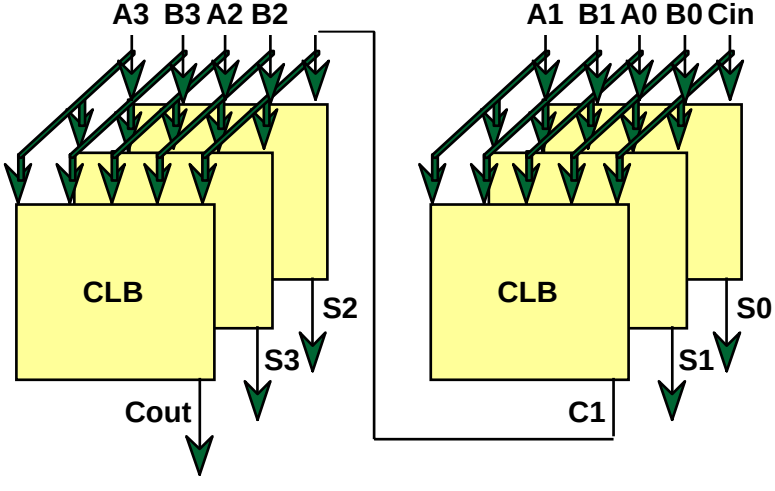


FPGA (Cont'd)

- 4-bit Binary Adder



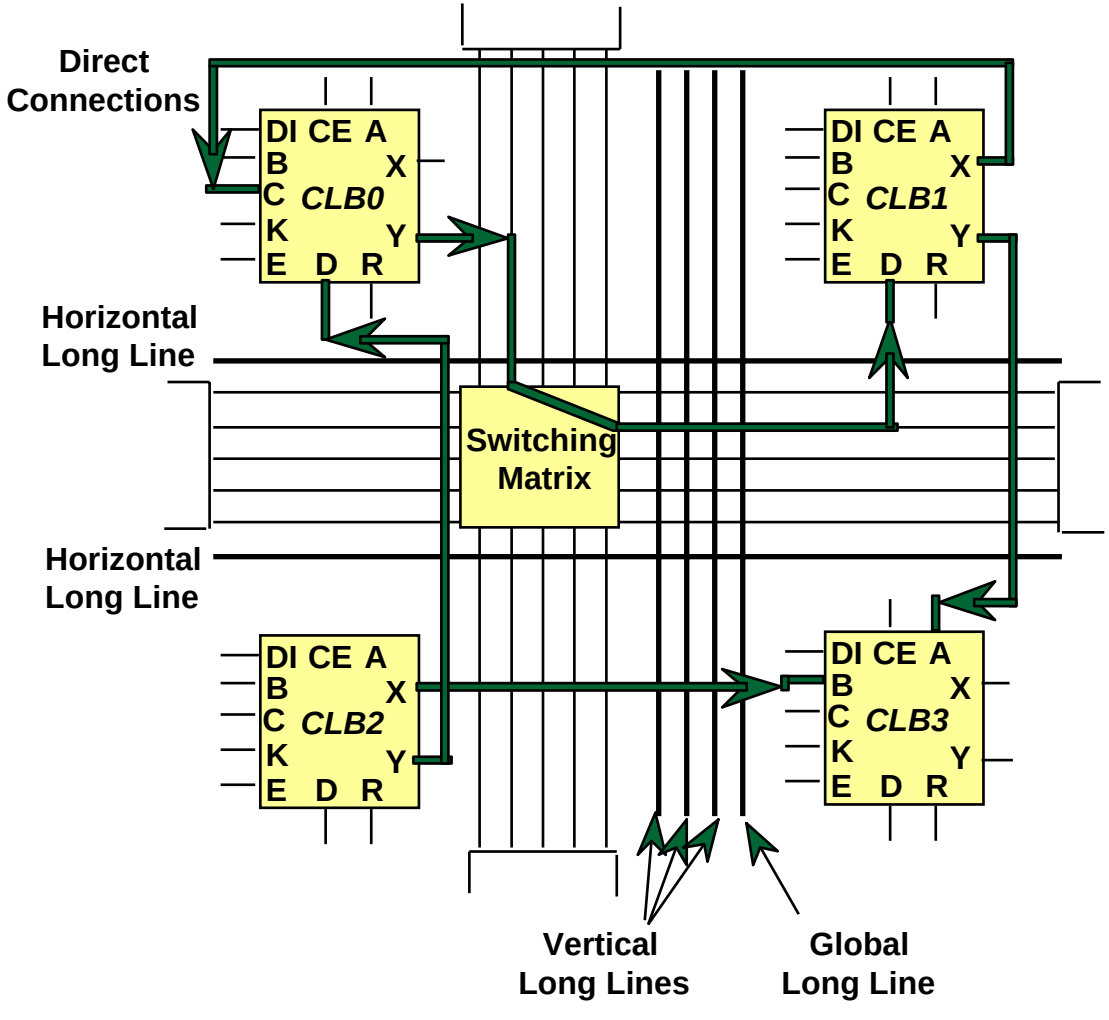
Full Adder, 4 CLB delays to final carry out



2 x Two-bit Adders (3 CLBs each) yields 2 CLBs to final carry out

FPGA (Cont'd)

- Interconnect
  - Direct Connections
  - Global Long Line
  - Horizontal/Vertical
  - Long Lines
  - Switching Matrix
  - Connections



FPGA (Cont'd)

- Xilinx Application Example (BCD to Excess 3 FSM)

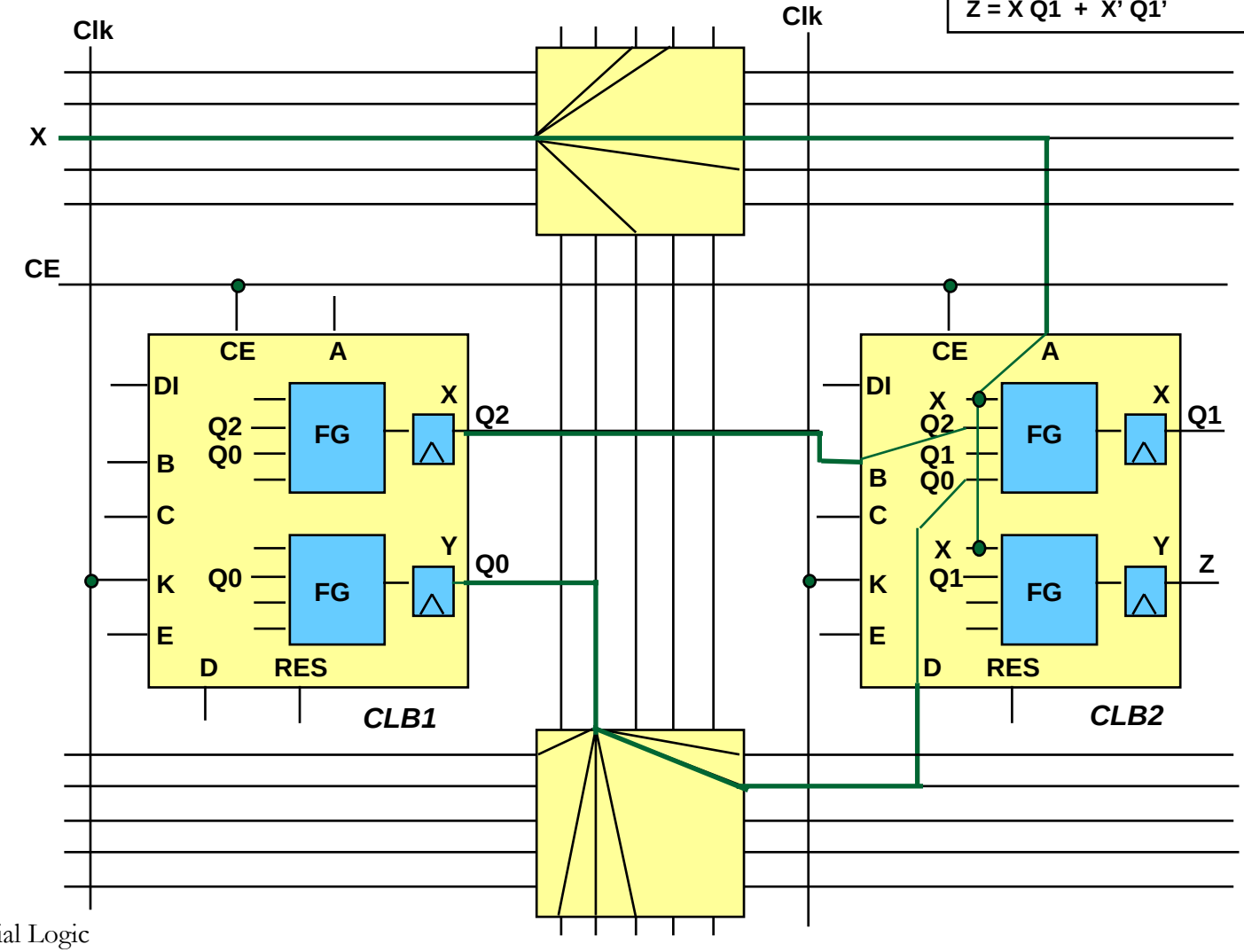
  Q2+ = Q2' Q0 + Q2 Q0'
  Q1+ = X' Q2' Q1' Q0 + X' Q2' Q0' + X Q2' Q0 + Q1' Q0
  Q0+ = Q0'
  Z = X Q1 + X' Q1'

  - Synchronous Mealy Machine
  - No function has more than 4 variables + 4 FFs implies 2 CLBs
  - Global Reset to be used
  - Place Q2+, Q0+ in one CLB and Q1, Z in second CLB
    --> maximize use of direct & general purpose interconnections

FPGA (Cont'd)

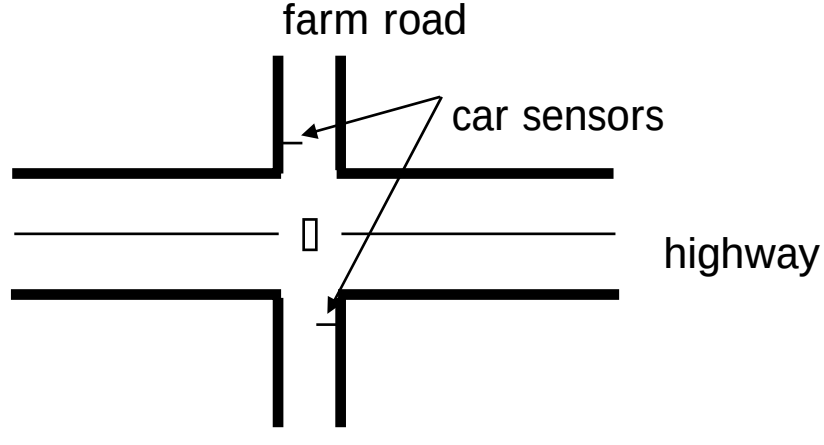
- Implementing the BCD to Excess 3 FSM

Q2+ = Q2' Q0 + Q2 Q0'
Q1+ = X' Q2' Q1' Q0 + X' Q2' Q0'
+ X Q2' Q0 + Q1' Q0
Q0+ = Q0'
Z = X Q1 + X' Q1'



Case Study: 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

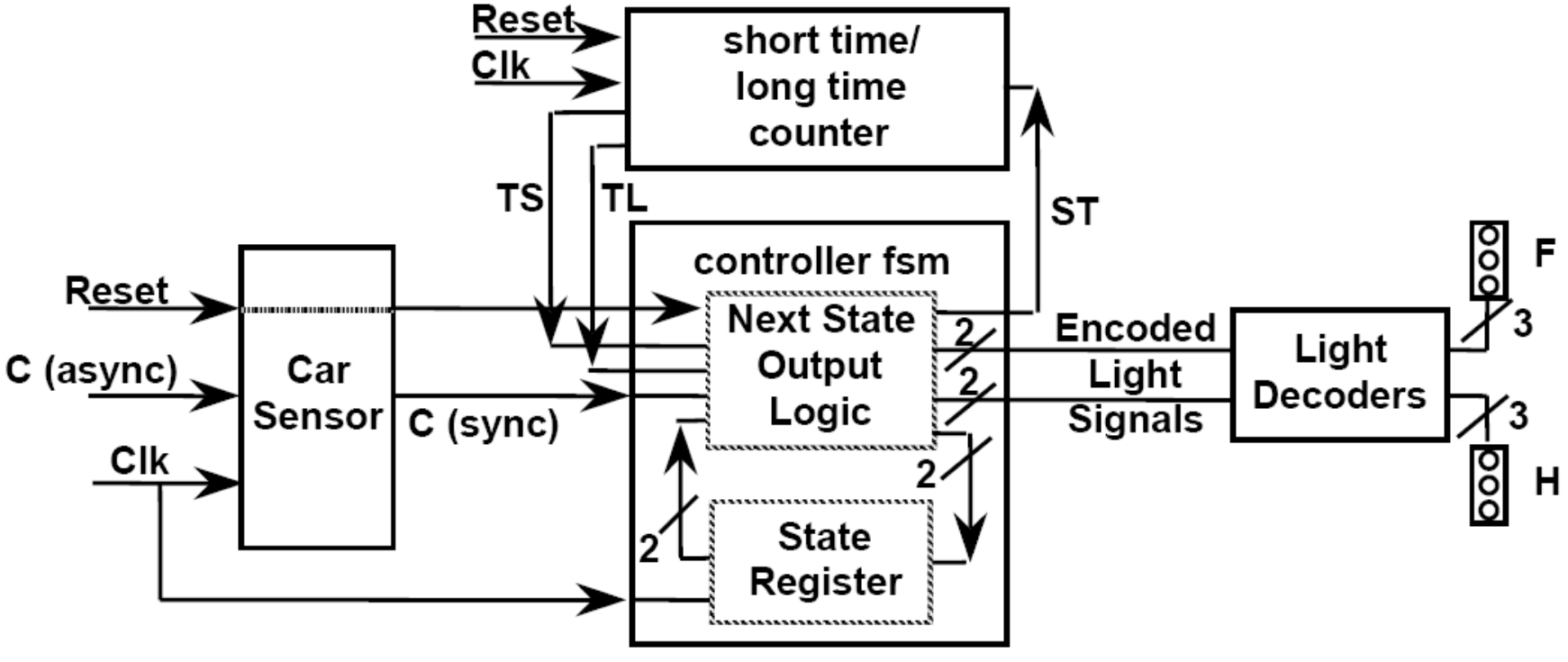


Traffic Light Controller (cont'd)

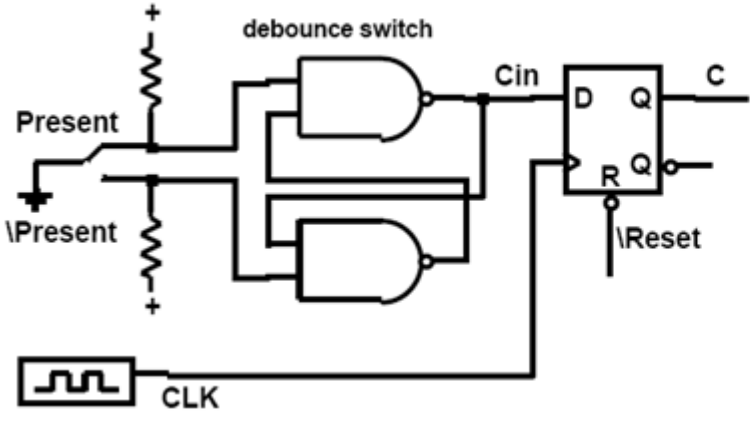
- Decomposition into primitive subsystems
  - Controller FSM: next state/output functions, state register
  - Short time/long time interval counter
  - Car Sensor
  - Output Decoders and Traffic Lights

Traffic Light Controller (cont'd)

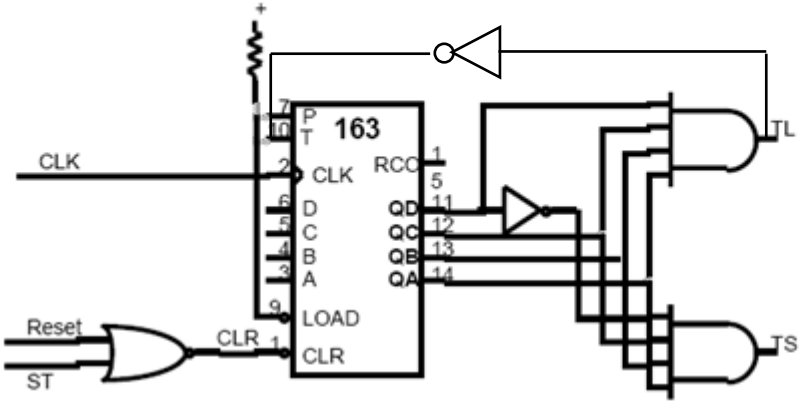
- Block diagram of complete traffic light system



Traffic Light Controller (cont'd)

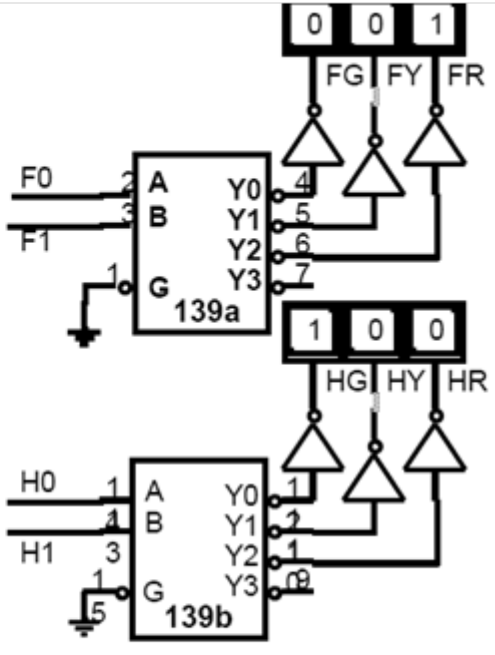


Car Detector



Interval Timer

Light Decoders



Traffic Light Controller (cont'd)

- 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)

Traffic Light Controller (cont'd)

- Next State/Output Logic

  State Assignment: HG=00, HY=10, FG=01, FY=11

  P1 = CTLQ1' + TS'Q1Q0' + C'Q1'Q0 + TS'Q1Q0

  P0 = TSQ1Q0' + Q1'Q0 + TS'Q1Q0

  ST = CTLQ1' + C'Q1'Q0 + TSQ1Q0' + TSQ1Q0

  H1 = TSQ1Q0 + Q1'Q0 + TS'Q1Q0

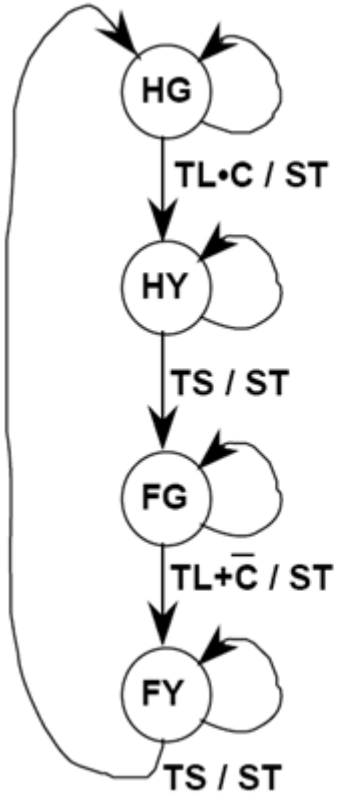
  H0 = TS'Q1Q0' + TSQ1Q0'

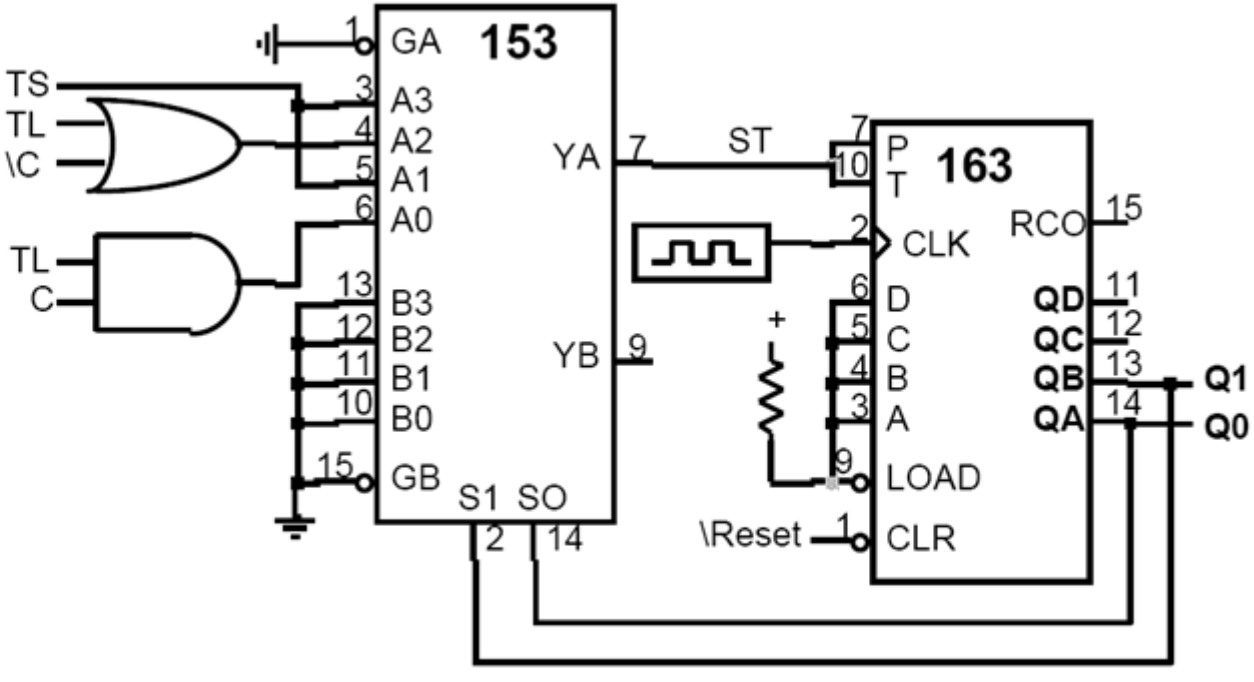
  F1 = Q0'

  F0 = TS'Q1Q0 + TSQ1Q0
- PAL/PLA Implementation
  - 5 inputs, 7 outputs, 8 product terms
  - PAL 22V10 – 12 inputs, 10 prog. IOs, 8 to 16 prod terms per OR
- ROM Implementation
  - 32 word by 8-bit ROM (256 bits)
  - Reset may double ROM size

Traffic Light Controller (cont'd)

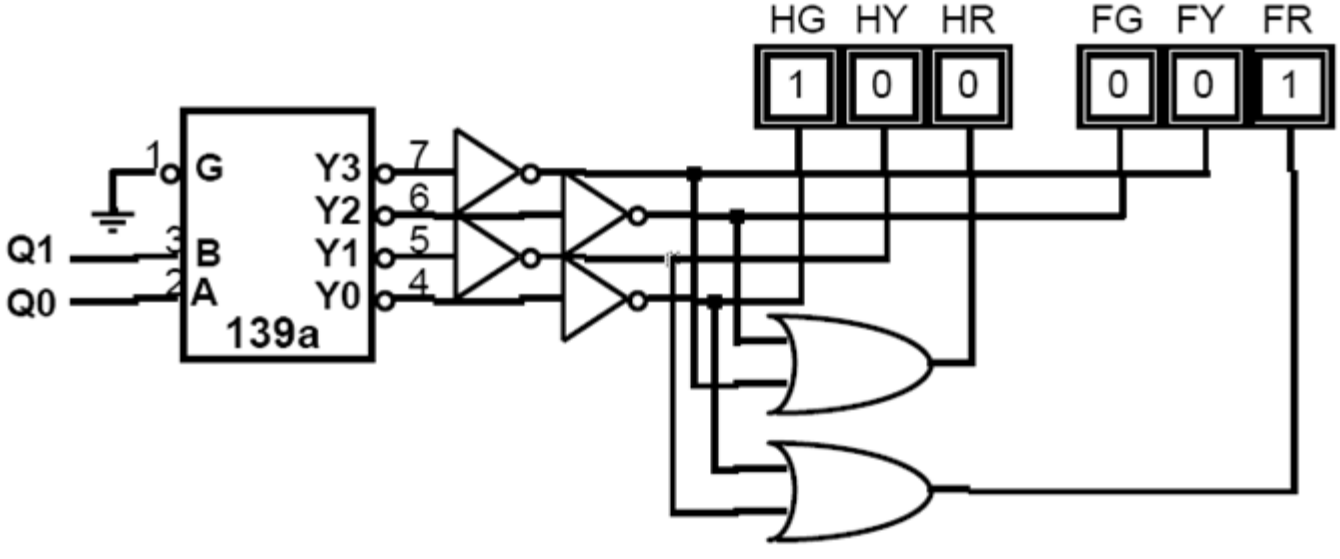
- Next State Logic
  - Counter-based Implementation
  - HG=00, HY=01, FG=10, FY=11





Traffic Light Controller (cont'd)

- Next State Logic
  - Counter-based Implementation
  - Dispense with direct output functions for the traffic lights



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 with FFs
  - PAL/PLA/ROM with FFs
  - CPLD/FPGA