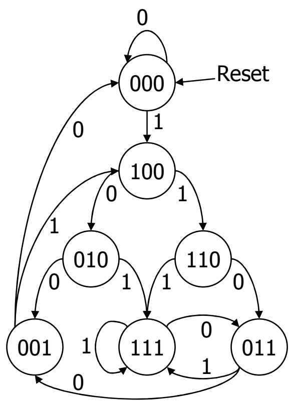
Sequential logic implementation

- Implementation
  - random logic gates and FFs
  - programmable logic devices (PAL with FFs)
- Design procedure
  - state diagrams
  - state transition table
  - state assignment
  - next state functions

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

Median filter FSM (cont'd)

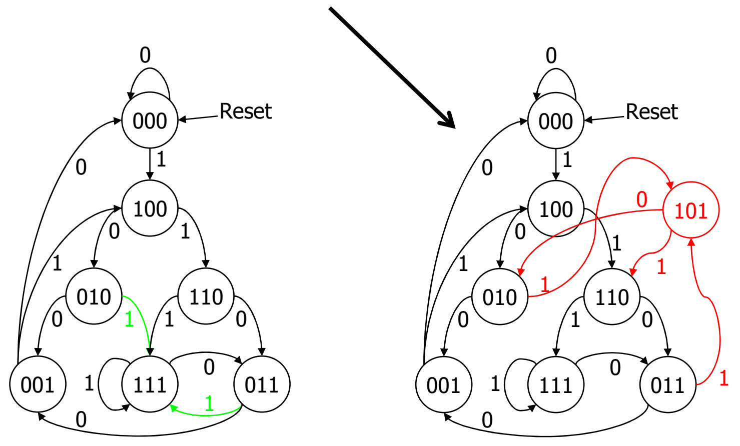
- Realized using the standard procedure and individual FFs and gates

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

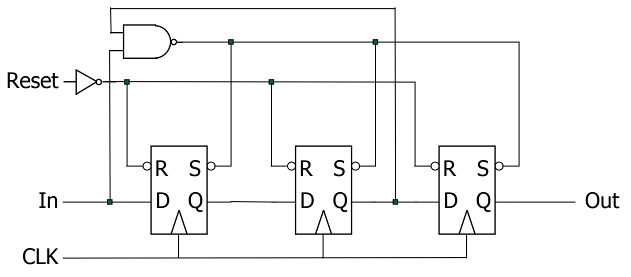
Median filter FSM (cont'd)

- 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

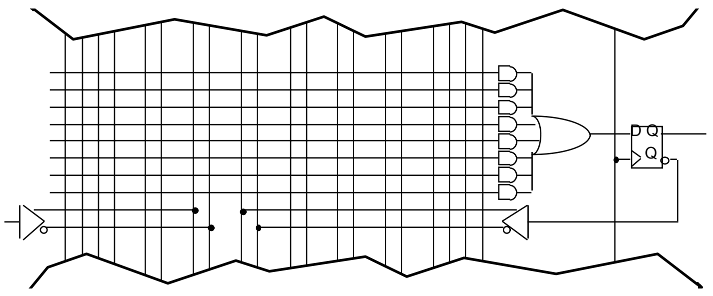


R = Reset
S = PS2 I
NS1 = I
NS2 = PS1
NS3 = PS2
O = PS3

- 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)

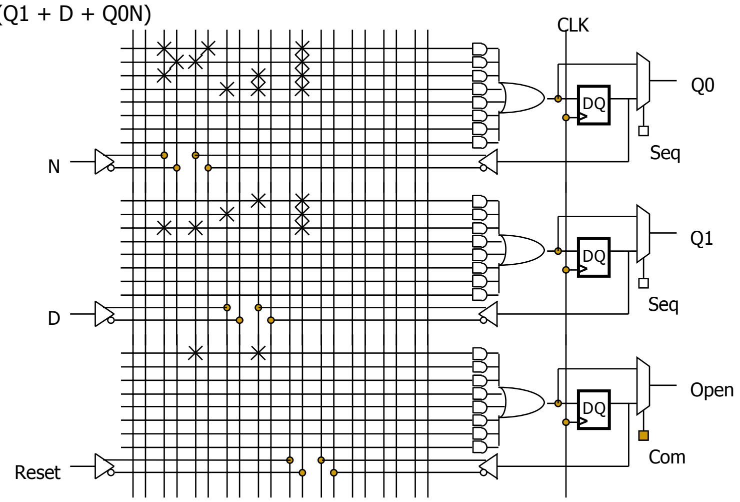
Implementation using PALs

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



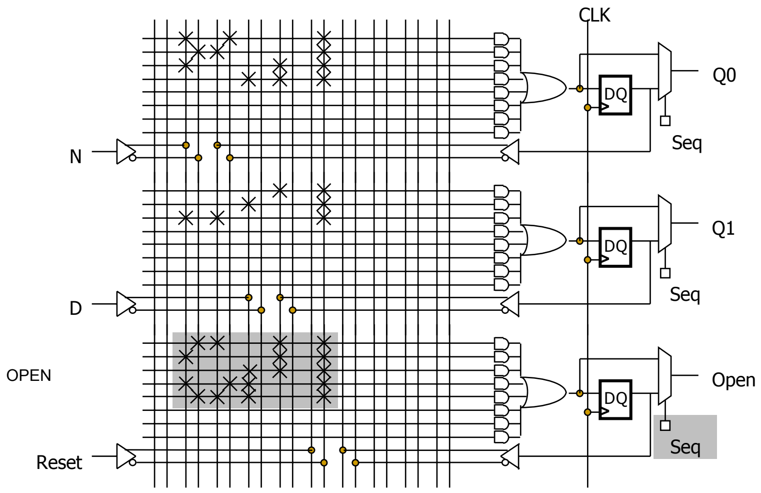
Vending machine example (Moore PLD mapping)

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



Vending machine (synch. Mealy PLD mapping)

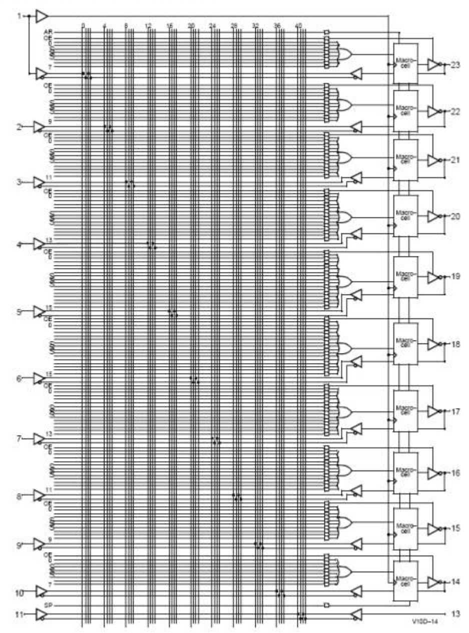
OPEN = reset'(Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D)



22V10 PAL

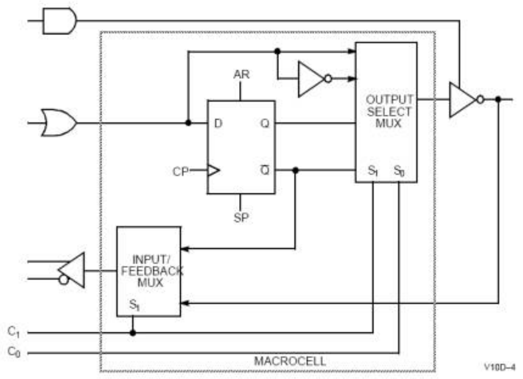
- Combinational logic elements (SoP)
- Sequential logic elements (D-FFs)
- Up to 10 outputs
- Up to 10 FFs
- Up to 22 inputs

Functional Logic Diagram for PALC22V10D



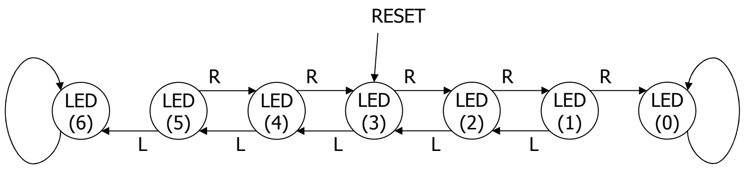
22V10 PAL Macro Cell

- Sequential logic element + output/input selection



Light Game FSM

- Tug of War game
  - 7 LEDs, 2 push buttons (L, R)


$$\begin{tikzcd}
 & & & \text{RESET} \arrow[d] & & & \\
\text{LED (6)} \arrow[loop left] & \text{LED (5)} \arrow[l, "L"'] \arrow[r, "R"] & \text{LED (4)} \arrow[l, shift left, "L"] \arrow[r, "R"] & \text{LED (3)} \arrow[l, shift left, "L"] \arrow[r, "R"] & \text{LED (2)} \arrow[l, shift left, "L"] \arrow[r, "R"] & \text{LED (1)} \arrow[l, shift left, "L"] \arrow[r, "R"] & \text{LED (0)} \arrow[loop right]
\end{tikzcd}$$

Light Game FSM Verilog

```
module Light_Game (LEDS, LPB, RPB, CLK, RESET);

   input LPB ;
   input RPB ;
   input CLK ;
   input RESET;
   output [6:0] LEDS ;

   reg [6:0] position;
   reg left;
   reg right;
```

combinational logic

```
    wire L, R;
    assign L = ~left && LPB;
    assign R = ~right && RPB;
    assign LEDS = position;
```

sequential logic

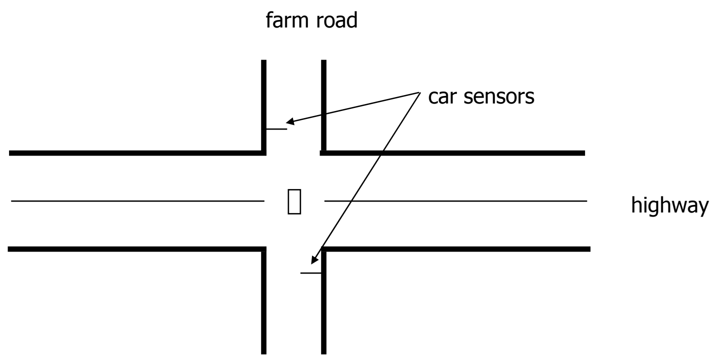
```
   always @(posedge CLK)
        begin
             left <= LPB;
             right <= RPB;
             if (RESET) position = 7'b0001000;
             else if ((position == 7'b0000001) || (position == 7'b1000000)) ;
             else if (L) position = position << 1;
             else if (R) position = position >> 1;
        end

endmodule
```

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



Example: traffic light controller (cont')

- Tabulation of inputs and outputs

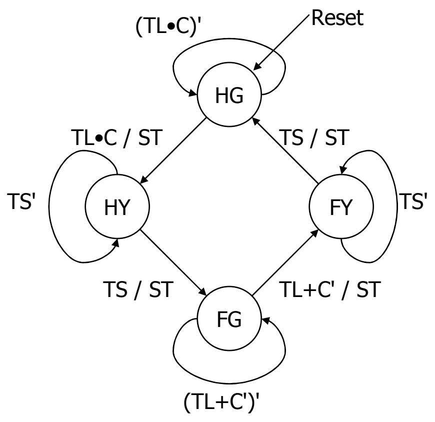
inputs	description	outputs	description
reset	place FSM in initial state	HG, HY, HR	assert green/yellow/red highway lights
C	detect vehicle on the farm road	FG, FY, FR	assert green/yellow/red highway lights
TS	short time interval expired	ST	start timing a short or long interval
TL	long time interval expired		

- 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



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)

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)