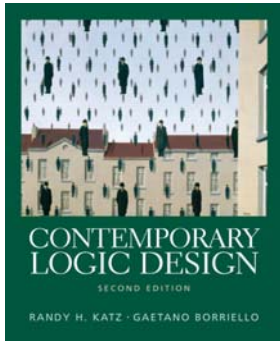


CSE 370 Spring 2006

Introduction to Digital Design

Lecture 21: Sequential Logic Technologies



Last Lecture

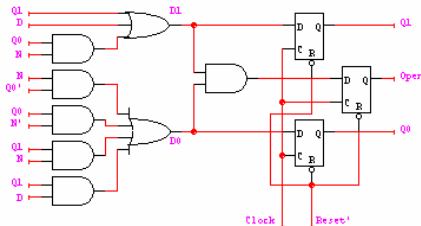
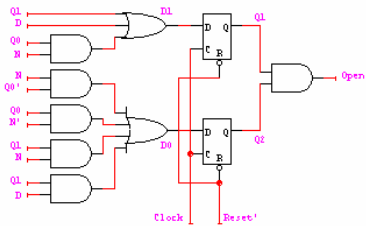
- Moore and Mealy Machines

Today

- Sequential logic technologies

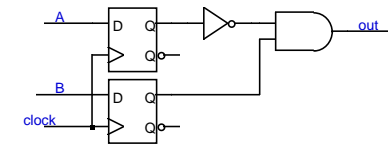
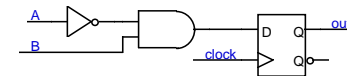
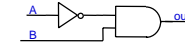
Vending machine: Moore to synchron. Mealy

- $OPEN = Q_1Q_0$ creates a combinational delay after Q_1 and Q_0 change in Moore implementation
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay
- $OPEN.d = (Q_1 + D + Q_0N)(Q_0'N + Q_0N' + Q_1N + Q_1D)$
 $= Q_1Q_0N' + Q_1N + Q_1D + Q_0'ND + Q_0N'D$
- Implementation now looks like a synchronous Mealy machine
 - it is common for programmable devices to have FF at end of logic



Mealy and Moore examples

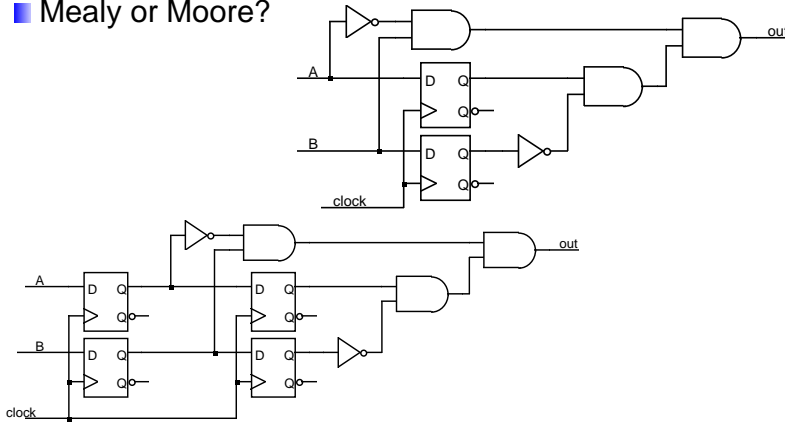
- Recognize A,B = 0,1
 - Mealy or Moore?



Mealy and Moore examples (cont'd)

- Recognize A,B = 1,0 then 0,1

Mealy or Moore?

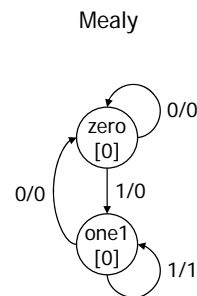
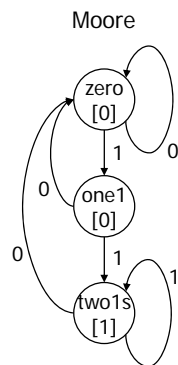


HDLs and Sequential Logic

- 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

```

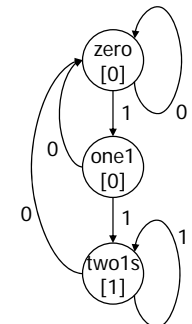
module reduce (clk, reset, in, out);
    input clk, reset, in;
    output out;

    parameter zero = 2'b00;
    parameter one1 = 2'b01;
    parameter twos1 = 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 = twos;
else next_state = zero;
end
twos:
// we've seen at least 2 ones
begin
if (in) next_state = twos;
else next_state = zero;
end
endcase
endmodule

```

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;
twos: out = 1;
endcase
endmodule

```

Mealy Verilog FSM

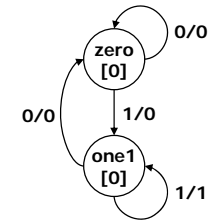
```

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
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
end
endcase
endmodule

```

Finite state machines summary

- Models for representing sequential circuits
 - abstraction of sequential elements
 - finite state machines and their state diagrams
 - inputs/outputs
 - Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
 - deriving state diagram
 - deriving state transition table
 - determining next state and output functions
 - implementing combinational logic
- Hardware description languages

Sequential logic implementation

Implementation

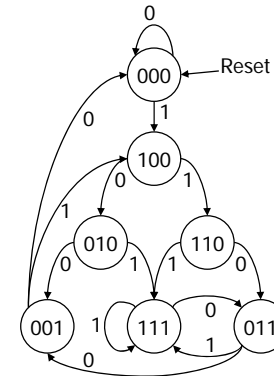
- random logic gates and FFs
- programmable logic devices (PAL with FFs)

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)

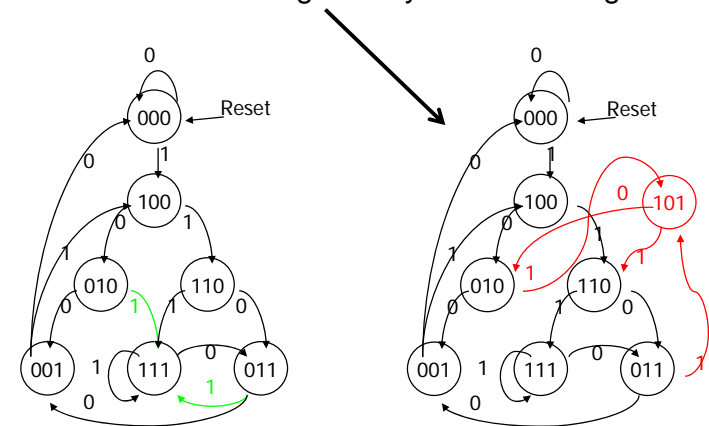
Realized using the standard procedure and individual FFs and gates

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

$NS1 = \text{Reset}' (I)$
 $NS2 = \text{Reset}' (PS1 + PS2 I)$
 $NS3 = \text{Reset}' PS2$
 $O = PS3$

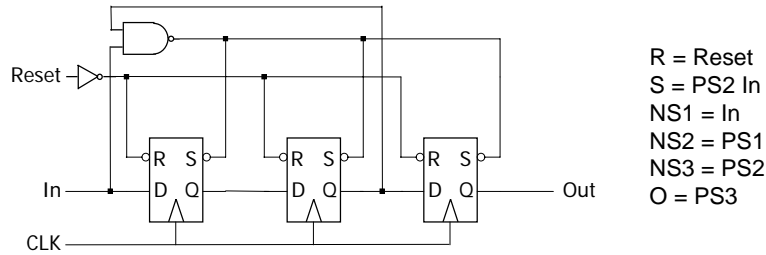
Median filter FSM (cont'd)

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



- 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)

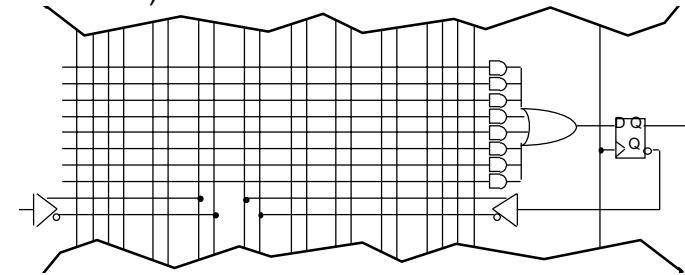
Implementation using PALs

- Programmable logic building block for sequential logic

■ macro-cell: FF + logic

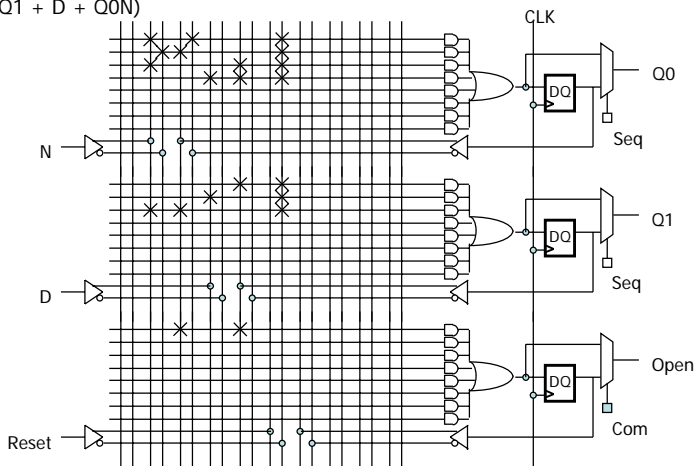
■ D-FF

■ two-level logic capability like PAL (e.g., 8 product terms)



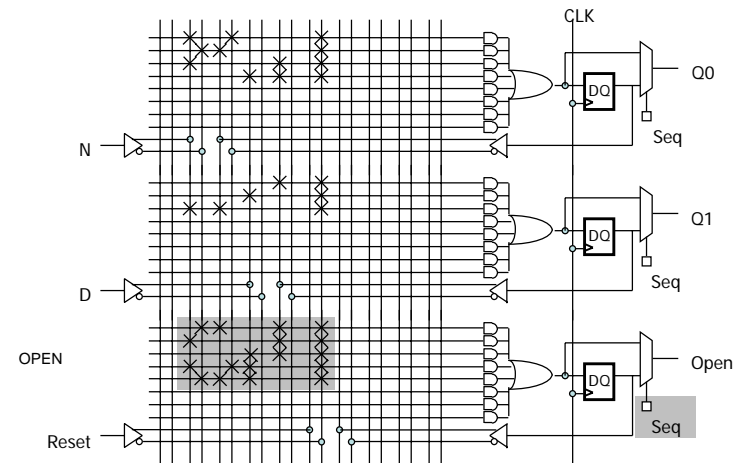
Vending machine example (Moore PLD mapping)

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



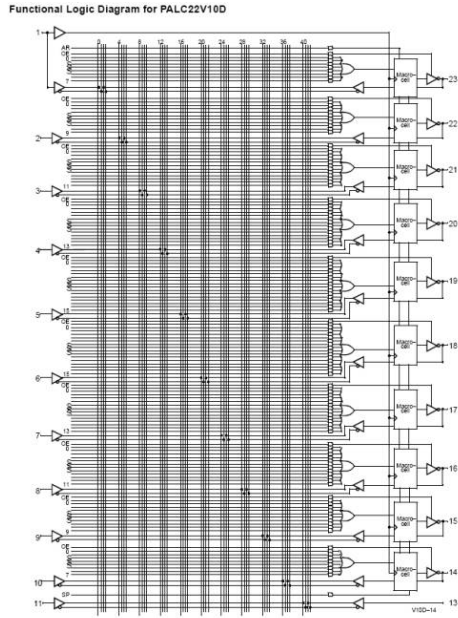
Vending machine (synch. Mealy PLD mapping)

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



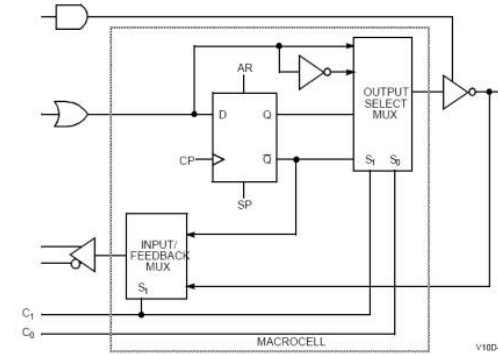
22V10 PAL

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



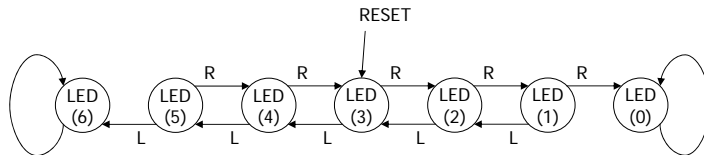
22V10 PAL Macro Cell

- Sequential logic element + output/input selection



Light Game FSM

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



Light Game FSM Verilog

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

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

combinational logic

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

sequential logic

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

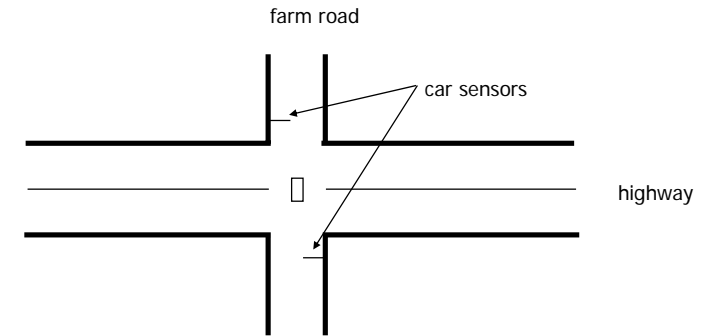
```
endmodule
```

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

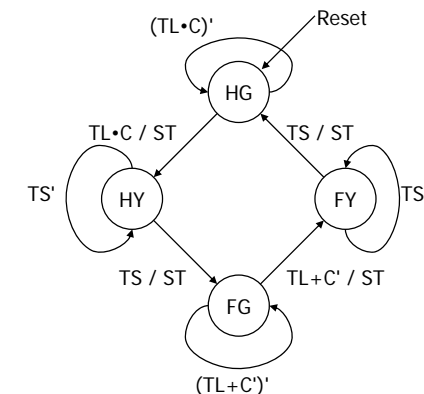
inputs	description	outputs	description
reset	place FSM in initial state	HG, HY, HR	assert green/yellow/red highway lights
C	detect vehicle on the farm road	FG, FY, FR	assert green/yellow/red highway lights
TS	short time interval expired	ST	start timing a short or long interval
TL	long time interval expired		

- 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			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)

Logic for different state assignments

- SA1

$$NS1 = C \cdot TL \cdot PS1 \cdot PS0 + TS \cdot PS1 \cdot PS0 + TS \cdot PS1 \cdot PS0' + C' \cdot PS1 \cdot PS0 + TL \cdot PS1 \cdot PS0$$

$$NS0 = C \cdot TL \cdot PS1 \cdot PS0' + C \cdot TL' \cdot PS1 \cdot PS0 + PS1 \cdot PS0$$

$$ST = C \cdot TL \cdot PS1 \cdot PS0' + TS \cdot PS1 \cdot PS0 + TS \cdot PS1 \cdot PS0' + C' \cdot PS1 \cdot PS0 + TL \cdot PS1 \cdot PS0$$

$$H1 = PS1 \qquad H0 = PS1 \cdot PS0$$

$$F1 = PS1' \qquad F0 = PS1 \cdot PS0'$$
- SA2

$$NS1 = C \cdot TL \cdot PS1' + TS \cdot PS1 + C' \cdot PS1 \cdot PS0$$

$$NS0 = TS \cdot PS1 \cdot PS0' + PS1 \cdot PS0 + TS \cdot PS1 \cdot PS0$$

$$ST = C \cdot TL \cdot PS1' + C' \cdot PS1 \cdot PS0 + TS \cdot PS1$$

$$H1 = PS0 \qquad H0 = PS1 \cdot PS0'$$

$$F1 = PS0' \qquad F0 = PS1 \cdot PS0$$
- SA3

$$NS3 = C' \cdot PS2 + TL \cdot PS2 + TS \cdot PS3$$

$$NS1 = C \cdot TL \cdot PS0 + TS \cdot PS1$$

$$NS2 = TS \cdot PS1 + C \cdot TL \cdot PS2$$

$$NS0 = C' \cdot PS0 + TL \cdot PS0 + TS \cdot PS3$$

$$ST = C \cdot TL \cdot PS0 + TS \cdot PS1 + C' \cdot PS2 + TL \cdot PS2 + TS \cdot PS3$$

$$H1 = PS3 + PS2 \qquad H0 = PS1$$

$$F1 = PS1 + PS0 \qquad F0 = PS3$$