Vending machine: Moore to synch. Mealy
- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 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 = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'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?

![Vending machine diagram](image1)
![Mealy and Moore examples](image2)
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

```verilog
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  parameter zero  = 2'b00;
  parameter one1  = 2'b01;
  parameter two1s = 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;
endmodule
```
Moore Verilog FSM (cont’d)

```verilog
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 = two1s;
      else    next_state = zero;
    end
  two1s: // we've seen at least 2 ones
    begin
      if (in) next_state = two1s;
      else    next_state = zero;
    end
endcase
```

always @(in or state)
  begin
    if (in) next_state = one1;
    else    next_state = zero;
  end

```

Mealy Verilog FSM

```verilog
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
        begin
          if (in) begin
            next_state = one; out = 1;
          end else begin
            next_state = zero; out = 0;
          end
        end
    endcase
endmodule
```

Synchronous Mealy Machine

```verilog
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
          begin
            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)

Median filter FSM (cont’d)

- Realized using the standard procedure and individual FFs and gates

<table>
<thead>
<tr>
<th></th>
<th>PS1</th>
<th>PS2</th>
<th>PS3</th>
<th>NS1</th>
<th>NS2</th>
<th>NS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>PS1</th>
<th>PS2</th>
<th>PS3</th>
<th>NS1</th>
<th>NS2</th>
<th>NS3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

![Median filter FSM diagram]

- 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)

\[ D_0 = \text{reset}'(Q_0N + Q_0N' + Q_1N + Q_1D) \]
\[ D_1 = \text{reset}'(Q_1 + D + Q_0N) \]
\[ \text{OPEN} = Q_1Q_0 \]

Vending machine (synch. Mealy PLD mapping)

\[ \text{OPEN} = \text{reset}'(Q_1Q_0N' + Q_1N + Q_1D + Q_0'ND + Q_0N'D) \]
**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**

```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;

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

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

**Example: traffic light controller (cont’)**

- Tabulation of unique states – some light configurations imply others

<table>
<thead>
<tr>
<th>state</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HG</td>
<td>highway green (farm road red)</td>
</tr>
<tr>
<td>HY</td>
<td>highway yellow (farm road red)</td>
</tr>
<tr>
<td>FG</td>
<td>farm road green (highway red)</td>
</tr>
<tr>
<td>FY</td>
<td>farm road yellow (highway red)</td>
</tr>
</tbody>
</table>

**Example: traffic light controller (cont’)**

- State diagram
Example: traffic light controller (cont’)

- Generate state table with symbolic states
- Consider state assignments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Present State</th>
<th>Next State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>TG</td>
<td>TG</td>
<td>ST</td>
</tr>
<tr>
<td>0</td>
<td>HG</td>
<td>HG</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>HG</td>
<td>HY</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>HY</td>
<td>HY</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>HY</td>
<td>FG</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>FG</td>
<td>FY</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>FG</td>
<td>FY</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>FG</td>
<td>FY</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>FY</td>
<td>FY</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>FY</td>
<td>FY</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>FG</td>
<td>FY</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>FY</td>
<td>FY</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>HG</td>
<td>HG</td>
<td>0</td>
</tr>
</tbody>
</table>

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