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>I</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>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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
Median filter FSM (cont’d)

- An alternate implementation with S/R FFs

![Diagram ofMedian filter FSM 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)

\[ D_0 = \text{reset}'(Q_0\overline{Q} + Q_0Q' + Q_1 + Q_1D) \]
\[ D_1 = \text{reset}'(Q_1 + D + Q_0) \]
\[ \text{OPEN} = Q_1Q_0 \]

Vending machine (synch. Mealy PLD mapping)

\[ \text{OPEN} = \text{reset}'(Q_1Q_0' + Q_1N + Q_1D + Q_0'ND + Q_0N'D) \]
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

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

![Diagram of a traffic light controller](attachment:image.png)
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>assert green/yellow/red highway lights</td>
</tr>
<tr>
<td>TS</td>
<td>short time interval expired</td>
<td>ST</td>
<td>start timing a short or long interval</td>
</tr>
<tr>
<td>TL</td>
<td>long time interval expired</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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

```
Reset
HG
(TL+C)'

TL+C / ST
TS
ST

TL+C / ST
(ST+C)'
FY
TS'

TL+C / ST
(ST+C)'
FG
TS'

(TL+C)'
```

Autumn 2003 CSE370 - VIII - Sequential Logic Technology
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>TL</td>
<td>TS</td>
<td>ST</td>
</tr>
<tr>
<td>0 0</td>
<td>HG</td>
<td>HG</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>HG</td>
<td>HY</td>
<td>1 1</td>
</tr>
<tr>
<td>1 0</td>
<td>FG</td>
<td>FG</td>
<td>0 FG</td>
</tr>
<tr>
<td>0 1</td>
<td>FG</td>
<td>FG</td>
<td>1 FG</td>
</tr>
<tr>
<td>1 0</td>
<td>FY</td>
<td>FY</td>
<td>0 FY</td>
</tr>
<tr>
<td>0 1</td>
<td>FY</td>
<td>HY</td>
<td>1 FY</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
  \[ \begin{align*}
  NS_1 &= C \cdot TL \cdot PS_1 \cdot PS_0 + TS \cdot PS_1 \cdot PS_0 + TS \cdot PS_1 \cdot PS_0' + C \cdot PS_1 \cdot PS_0 + TL \cdot PS_1 \cdot PS_0 \\
  NS_0 &= C \cdot TL \cdot PS_1 \cdot PS_0' + C \cdot TL \cdot PS_1 \cdot PS_0 + PS_1 \cdot PS_0 \\
  ST &= C \cdot TL \cdot PS_1 \cdot PS_0' + TS \cdot PS_1 \cdot PS_0 + TS \cdot PS_1 \cdot PS_0' + C \cdot PS_1 \cdot PS_0 + TL \cdot PS_1 \cdot PS_0 \\
  H_1 &= PS_1 \\
  F_1 &= PS_1' \\
  H_0 &= PS_1 \cdot PS_0' \\
  F_0 &= PS_1 \cdot PS_0'
  \end{align*} \]

- SA2
  \[ \begin{align*}
  NS_1 &= C \cdot TL \cdot PS_1 + TS \cdot PS_1 + C \cdot PS_1 \cdot PS_0 \\
  NS_0 &= TS \cdot PS_1 \cdot PS_0' + PS_1 \cdot PS_0 + TS \cdot PS_1 \cdot PS_0 \\
  ST &= C \cdot TL \cdot PS_1 \cdot PS_0 + C \cdot PS_1 \cdot PS_0 + TS \cdot PS_1 \\
  H_1 &= PS_0 \\
  F_1 &= PS_0' \\
  H_0 &= PS_1 \cdot PS_0' \\
  F_0 &= PS_1 \cdot PS_0
  \end{align*} \]

- SA3
  \[ \begin{align*}
  NS_3 &= C \cdot PS_2 + TL \cdot PS_2 + TS \cdot PS_3 \\
  NS_2 &= TS \cdot PS_1 + C \cdot TL \cdot PS_2 \\
  NS_1 &= C \cdot TL \cdot PS_0 + TS \cdot PS_1 \\
  NS_0 &= C \cdot PS_0 + TL \cdot PS_0 + TS \cdot PS_3 \\
  ST &= C \cdot TL \cdot PS_0 + TS \cdot PS_1 + C \cdot PS_2 + TL \cdot PS_2 + TS \cdot PS_3 \\
  H_1 &= PS_3 + PS_2 \\
  F_1 &= PS_1 + PS_0 \\
  H_0 &= PS_1 \\
  F_0 &= PS_3
  \end{align*} \]
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)