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

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)
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
Vending machine example (Moore PLD mapping)

\[
\begin{align*}
D_0 &= \text{reset}'(Q_0' + Q_0' + Q_1 + Q_1'P) \\
D_1 &= \text{reset}'(Q_1 + P + Q_0'P') \\
\text{OPEN} &= Q_1Q_0
\end{align*}
\]

Vending machine (synch. Mealy PLD mapping)

\[
\begin{align*}
\text{OPEN} &= \text{reset}'(Q_1Q_0' + Q_1 + Q_1D + Q_0'ND + Q_0'P')
\end{align*}
\]
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))
    position <= position;
  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
```

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

```
  farm road
    car sensors
    highway
```

Winter 2005  CSE370 - VIII - Sequential Logic Technology
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>assert green/yellow/red highway lights</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**

```
  HG  (TL+C)'
     |       |
     | Reset |

  TS'  TL+C / ST  TS / ST

  TS'  TL+ C' / ST

  FG  (TL+C)'
```

Winter 2005  CSE370 - VIII - Sequential Logic Technology
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)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Present State</th>
<th>Next State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C TL TS</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>0 - -</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>1 1 -</td>
<td>HG</td>
<td>HY</td>
<td>1 Green Red</td>
</tr>
<tr>
<td>- - 0</td>
<td>HY</td>
<td>FG</td>
<td>1 Yellow Red</td>
</tr>
<tr>
<td>1 0 -</td>
<td>FG</td>
<td>FG</td>
<td>0 Red Green</td>
</tr>
<tr>
<td>0 - -</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>- 1 -</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>- - 0</td>
<td>FY</td>
<td>FY</td>
<td>0 Red Yellow</td>
</tr>
<tr>
<td>- - 1</td>
<td>FY</td>
<td>HG</td>
<td>1 Red Yellow</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
  F1 = PS1'

- 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
  F1 = 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
  F1 = PS1 + PS0
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)