Sequential logic implementation

- Finite-state machines
  - Moore
  - Mealy
  - Synchronous Mealy

- 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)
Comparison of Mealy and Moore machines

- **Mealy machines tend to have less states**
  - different outputs on arcs \((n^2)\) rather than states \((n)\)
- **Moore machines are safer to use**
  - outputs change at clock edge (always one cycle later)
  - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback
- **Mealy machines react faster to inputs**
  - react in same cycle – don’t need to wait for clock
  - in Moore machines, more logic may be necessary to decode state into outputs – more gate delays after

Comparison of Mealy and Moore machines (cont’d)

- **Moore**
  - combinational logic for next state
  - state feedback
  - logic for outputs
  - outputs

- **Mealy**
  - combinational logic for next state
  - state feedback
  - logic for outputs
  - outputs

- **Synchronous Mealy**
  - combinational logic for next state
  - state feedback
  - logic for outputs
  - outputs
**Example: vending machine**

- Release item after 15 cents are deposited
- Single coin slot for dimes, nickels
- No change

![State diagram of a vending machine]

**Example: vending machine (cont’d)**

- Suitable abstract representation
  - tabulate typical input sequences:
    - 3 nickels
    - nickel, dime
    - dime, nickel
    - two dimes
  - draw state diagram:
    - inputs: N, D, reset
    - output: open chute
  - assumptions:
    - assume N and D asserted for one cycle
    - each state has a self loop for N = D = 0 (no coin)
Example: vending machine (cont’d)

Minimize number of states - reuse states whenever possible

<table>
<thead>
<tr>
<th>Present state</th>
<th>Inputs</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

symbolic state table

Example: vending machine (cont’d)

Uniquely encode states

<table>
<thead>
<tr>
<th>Present state</th>
<th>Inputs</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0</td>
<td>D N</td>
<td>D1 D0</td>
<td>open</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>- - -</td>
<td>-</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>1 0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>- - -</td>
<td>-</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>- - -</td>
<td>-</td>
</tr>
</tbody>
</table>

Autumn 2000  CSE370 - VIII - Sequential Logic Technology
Example: Moore implementation

- Mapping to logic

\[ D_1 = Q_1 + D + Q_0 N \]
\[ D_0 = Q_0' N + Q_0 N' + Q_1 N + Q_1 D \]
\[ OPEN = Q_1 Q_0 \]

Example: vending machine (cont’d)

- One-hot encoding

<table>
<thead>
<tr>
<th>present state</th>
<th>inputs</th>
<th>next state output</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3 D2 D1 D0</td>
<td>D N</td>
<td>D3 D2 D1 D0 open</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 0</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0 0</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>1 1 - - -</td>
<td>- -</td>
<td>- - - - -</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 0</td>
<td>0 1 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 0</td>
<td>1 0 0 0 0</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 0</td>
<td>1 0 0 0 0</td>
</tr>
<tr>
<td>1 1 - - -</td>
<td>- -</td>
<td>- - - - -</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>- -</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>
Equivalent Mealy and Moore state diagrams

Moore machine
- Outputs associated with state

Mealy machine
- Outputs associated with transitions

Vending machine example (Moore PLD mapping)

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

D0 = reset(Q0'N' + Q0N' + Q1N + Q1D)
D1 = reset(Q1 + D + Q0N)
OPEN = reset(Q1Q0 + Q1N + Q1D + Q0D)
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 = reset'(Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = reset'(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

Vending machine: Mealy to synch. Mealy

- OPEN = reset'(Q1Q0 + Q1N + Q1D + Q0D)
- OPEN = reset'(Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = reset'(Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D)
Vending machine (synch. Mealy PLD mapping)

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

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

Example: traffic light controller (cont')

Tabulation of inputs and outputs

<table>
<thead>
<tr>
<th>Inputs description</th>
<th>Outputs description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>HG, HY, HR assert green/yellow/red highway lights</td>
</tr>
<tr>
<td>C</td>
<td>FG, FY, FR assert green/yellow/red highway lights</td>
</tr>
<tr>
<td>TS</td>
<td>ST start timing a short or long interval</td>
</tr>
<tr>
<td>TL</td>
<td>long time interval expired</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

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></td>
<td>C   TL TS   ST</td>
<td>H</td>
<td>G</td>
</tr>
<tr>
<td>0</td>
<td>0   0   1   0</td>
<td>HG</td>
<td>HG</td>
</tr>
<tr>
<td>1</td>
<td>0   0   1   0</td>
<td>HG</td>
<td>HG</td>
</tr>
<tr>
<td>1</td>
<td>1   0   0   1</td>
<td>HY</td>
<td>HY</td>
</tr>
<tr>
<td>1</td>
<td>1   0   1   0</td>
<td>HY</td>
<td>FG</td>
</tr>
<tr>
<td>1</td>
<td>0   1   0   0</td>
<td>FY</td>
<td>FY</td>
</tr>
<tr>
<td>0</td>
<td>1   1   0   0</td>
<td>FY</td>
<td>FY</td>
</tr>
<tr>
<td>0</td>
<td>0   0   0   0</td>
<td>HG</td>
<td>HG</td>
</tr>
<tr>
<td>0</td>
<td>1   1   1   1</td>
<td>HG</td>
<td>FY</td>
</tr>
<tr>
<td></td>
<td>SA1: HG = 00</td>
<td>HY = 01</td>
<td>FG = 11</td>
</tr>
<tr>
<td></td>
<td>SA2: HG = 00</td>
<td>HY = 10</td>
<td>FG = 01</td>
</tr>
<tr>
<td></td>
<td>SA3: HG = 001</td>
<td>HY = 0010</td>
<td>FG = 0100</td>
</tr>
</tbody>
</table>
Logic for different state assignments

**SA1**

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

**SA2**

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

**SA3**

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

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)