Finite State Machines

• Example: Edge Detector
  Bit are received one at a time (one per cycle), such as: 000111010

Design a circuit that asserts its output for one cycle when the input bit stream changes from 0 to 1.

Try two different solutions.

State Transition Diagram Solution A

Solution A, circuit derivation

Solution B
Output depends non only on $PS$ but also on input, $IN$

Edge detector timing diagrams

• Solution A: output follows the clock
• Solution B: output changes with input rising edge and is asynchronous wrt the clock.
### FSM Comparison

**Solution A**
- Moore Machine
  - output function only of PS
  - maybe more state
  - synchronous outputs
    - no glitching
    - one cycle "delay"
    - full cycle of stable output

**Solution B**
- Mealy Machine
  - output function of both PS & input
  - maybe fewer states
  - asynchronous outputs
    - If input glitches, so does output
    - output immediately available
    - output may not be stable long enough to be useful.

### FSM Recap

FSM Recap

**Moore Machine**
- Input value/output only of current state
- Synchronous outputs
  - No glitches
  - One cycle "delay"
  - Full cycle of stable output

**Mealy Machine**
- Input value/output also depends on input
- Asynchronous outputs
  - Glitching possible
  - Output immediately available
  - May not be stable long enough to be useful:

### FSM Optimization

**State Reduction**
- Motivation: lower cost
  - Fewer flip-flops in one-hot implementations
  - Possibly fewer flip-flops in encoded implementations
  - More don't cares in NS logic
  - Fewer gates in NS logic
- Simplifier to design with extra states then reduce later.

### State Reduction
- State Reduction is based on:
  - Two states are equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state.
  - If two states are equivalent, one can be eliminated without affecting the behavior of the FSM.
- Several algorithms exist:
  - Row matching method.
  - Implication table method.
- "Row Matching" is based on the state-transition table:
  - Two states are equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state.
  - If two states are equivalent, one can be eliminated without affecting the behavior of the FSM.
  - Several algorithms exist:
    - Row matching method.
    - Implication table method.
- Note: This algorithm is slightly different than the book.

### Row Matching Example

**State Transition Table**

<table>
<thead>
<tr>
<th>NS</th>
<th>PS</th>
<th>x=x0</th>
<th>x=x1</th>
<th>x=x0</th>
<th>x=x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Reduced State Transition Diagram**

<table>
<thead>
<tr>
<th>NS</th>
<th>PS</th>
<th>x=x0</th>
<th>x=x1</th>
<th>x=x0</th>
<th>x=x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>f</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Row Matching Example**

<table>
<thead>
<tr>
<th>NS</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
</tr>
<tr>
<td>g</td>
<td>a</td>
</tr>
</tbody>
</table>

**Reduced State Transition Diagram**

<table>
<thead>
<tr>
<th>NS</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
</tr>
<tr>
<td>g</td>
<td>a</td>
</tr>
</tbody>
</table>
State Reduction

- The "row matching" method is not guaranteed to result in the optimal solution in all cases, because it only looks at pairs of states.
- For example:

State Assignment (from Katz)

- In encoded (non-one-hot) FSMs, the choice of binary encodings for the states has an influence on the number of logic gates (or LUTs) needed to compute the next state and outputs.
- For \( n \) states, at least \( s \) bits are needed for a binary encoding.
  \[ s = \lceil \log_2 n \rceil \]
- \( 2^s \) different encodings exist.
- We will look at several "by-hand" heuristic methods for choosing good assignments.
- Some CAD tools will make assignments automatically.

State Assignment (from Katz)

- In encoded (non-one-hot) FSMs, the choice of binary encodings for the states has an influence on the number of logic gates (or LUTs) needed to compute the next state and outputs.
- For \( n \) states, at least \( s \) bits are needed for a binary encoding.
  \[ s = \lceil \log_2 n \rceil \]
- \( 2^s \) different encodings exist.
- We will look at several "by-hand" heuristic methods for choosing good assignments.
- Some CAD tools will make assignments automatically.

**Example**

Examples: 3-bit Sequence Detector

- Reset State = 00
- Highest Priority Adjacency
  \[ Q0 \]
  \[ 0 1 \]
  \[ 0 0 \]

Paper and Pencil Methods

- Reset State = 00
- Highest Priority Adjacency
  \[ Q0 \]
  \[ 0 1 \]
  \[ 0 0 \]
**State Assignment Example**

Another Example: 4 bit String Recognizer

Highest Priority: 2(S1, S2) (S3', S4'), (S7', S10')

Medium Priority: (S1, S2), 2(S3', S4'), (S7', S10')

Lowest Priority: 0/0: (S0, S1, S2, S3', S4', S7')
1/0: (S0, S1, S2, S3', S4', S7')

**State Assignment**

Effect of Adjacencies on Next State Map

First encoding exhibits a better clustering of 1's in the next state map