Sequential Optimizations and Tradeoffs
State Encoding

- **Encoding**: Assigning a unique bit representation to each state
- Different encodings may optimize size, or tradeoff size and performance
- Consider 3-Cycle Laser Timer...
  - Example 3.7's encoding: 15 gate inputs
  - Try alternative encoding
    - \( x = s_1 + s_0 \)
    - \( n_1 = s_0 \)
    - \( n_0 = s_1 s' + s_1 s_0 \)
    - Only 8 gate inputs

### One-hot Encoding
- One bit per state – a bit being ‘1’ corresponds to a particular state
- Alternative to minimum bit-width encoding in previous example
- For \( A, B, C, D; D = 0001, B = 0010, C = 0100, D = 1000 \)
- Example: FSM that outputs 0, 1, 1, 1
  - Equations if one-hot encoding:
    - \( n_3 = s_2 \)
    - \( n_2 = s_1 \)
    - \( n_1 = s_0 \)
    - \( x = s_3 + s_2 + s_1 \)
  - Fewer gates and only one level of logic – less delay than two levels, so faster clock frequency
Sequential Optimizations and Tradeoffs

One-Hot Encoding Example: Three-Cycles-High Laser Timer

- Four states - Use four-bit one-hot encoding
  - State table leads to equations:
    - \( x = s_3 + s_2 + s_1 \)
    - \( n_3 = s_2 \)
    - \( n_2 = s_1 \)
    - \( n_1 = s_0 \cdot b \)
    - \( n_0 = s_0 \cdot b' + s_3 \)
  - Smaller
    - \( 3+0+0+2+(2+2) = 9 \) gate inputs
  - Faster
    - Critical path: \( n_0 = s_0 \cdot b' + s_3 \)
      - Previously: \( n_0 = s_1 \cdot s_0' + s_1 s_0' \)
      - 2-input AND slightly faster than 3-input AND

Output encoding: Encoding method where the state encoding is same as the output values

- Possible if enough outputs, all states with unique output values

Sequential Optimizations and Tradeoffs

State Encoding: Output Encoding

- Output encoding: Encoding method where the state encoding is same as the output values
  - Possible if enough outputs, all states with unique output values

Sequential Optimizations and Tradeoffs

Output Encoding Example: Sequence Generator

- Generate sequence 0001, 0011, 1110, 1000, repeat
  - FSM shown
- Use output values as state encoding
- Derive equations for next state
  - \( n_3 = s_1 + s_2; n_2 = s_1; n_1 = s_1 \cdot s_0; n_0 = s_1 \cdot s_0 + s_3 s_2' \)

Sequential Optimizations and Tradeoffs

Moore vs. Mealy FSMs

- FSM implementation architecture
  - State register and logic
    - Next state logic - function of present state and FSM inputs
    - Output logic
      - 1f function of present state only - Moore FSM
      - 1f function of present state and FSM inputs - Mealy FSM

Graphically: show outputs with arcs, not with states
Sequential Optimizations and Tradeoffs

Moore vs. Mealy FSMs: Mealy FSMs May Have Fewer States

Moore

- Inputs: enough (bit)
- Outputs: d, clear (bit)
- Initial state: Init
- Transitions:
  - enough\' → Wait
  - Wait → Disp
  - Disp → Init

Mealy

- Inputs: enough (bit)
- Outputs: d, clear (bit)
- Initial state: Init
- Transitions:
  - enough\' → enough
  - enough → d
  - clear

Soda dispenser example: Initialize, wait until enough, dispense
- Moore: 3 states; Mealy: 2 states

Sequential Optimizations and Tradeoffs

Moore vs. Mealy Tradeoff

- Mealy outputs change mid-cycle if input changes
  - Note earlier soda dispenser example
- Represent a type of tradeoff

Mealy vs. Moore Example: Beeping Wristwatch

- Button b
  - Sequences mux select lines s1s0 through 00, 01, 10, and 11
    - Each value displays different internal register
    - Each unique button press should cause 1-cycle beep, with p=1 being beep
  - Must wait for button to be released (b) and pushed again (b') before sequencing
    - Note that Moore requires unique state to pulse p, while Mealy pulses p on arc
    - Tradeoff: Moore's pulse on p may not last one full cycle

Mealy

- Inputs: s1, s0, p
- Outputs: s1, s0, p
- States: S2, S4, S6, S8
- Transitions:
  - s1s0=00, p=0
  - s1s0=01, p=0
  - s1s0=10, p=0
  - s1s0=11, p=0

Moore

- Inputs: s1, s0, p
- Outputs: s1, s0, p
- States: S2, S4, S6, S8
- Transitions:
  - s1s0=00, p=0
  - s1s0=01, p=0
  - s1s0=10, p=0
  - s1s0=11, p=0
Sequential Optimizations and Tradeoffs
Implementing a Mealy FSM

- Straightforward
  - Convert to state table
  - Derive equations for each output
  - Key difference from Moore: External outputs (d, clear) may have different value in same state, depending on input values

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>0</td>
</tr>
<tr>
<td>enough</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>s0</td>
<td>0</td>
</tr>
<tr>
<td>enough</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
</tr>
<tr>
<td>s0</td>
<td>1</td>
</tr>
<tr>
<td>enough</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>s1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
</tr>
</tbody>
</table>

Sequential Optimizations and Tradeoffs
Mealy and Moore can be combined

- Final note on Mealy/Moore
  - May be combined in same FSM

Combined Moore/Mealy FSM for beeping wristwatch example