Iterative Circuits:
Example #2
(Mealy-machine based)
Pattern Detector: Verbal Description

General Problem Statement: given a string X of input bits, which are expected to follow the repeated pattern “0011” (i.e. 0011001100111...), produce a corresponding string Z of output bits which flag any error locations with a ‘1’ bit (and all correct locations with a ‘0’ bit).

More Detailed Requirements: flagging errors
- the machine first expects a ‘0’ input (if it receives any ‘1’ inputs, stay in initial state and flag these erroneous inputs)
- after it receives its first ‘0’ bit, it outputs a ‘0’ (no error), and waits for a second bit
  - if second bit is ‘0’, no error (output ‘0’)
  - if second bit is ‘1’, has error (output ‘1’)
- ... in each case, absorb the second input and prepare for the next pair of ‘11’ bits
- if the machine receives any more ‘0’ inputs, stay in the current state and flag these erroneous inputs
- after it receives its first ‘1’ bit, it outputs a ‘0’ (no error), and waits for a second bit
  - if second bit is ‘1’, no error (output ‘0’)
  - if second bit is ‘0’, has error (output ‘1’)
- ... in each case, absorb the second input and prepare for the next pair of ‘00’ bits
... go to top and repeat

Iterative Circuit Synthesis: Mealy-based

FSM Block Diagram: bit-serial processing

Mealy State Diagram: bit-serial processing
# Iterative Circuit Synthesis: Mealy-based

### Symbolic State Table:

<table>
<thead>
<tr>
<th>PS input xi</th>
<th>NS output zi</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0 0</td>
<td>S1 0</td>
</tr>
<tr>
<td>S0 1</td>
<td>S0 1</td>
</tr>
<tr>
<td>S1 0</td>
<td>S2 0</td>
</tr>
<tr>
<td>S1 1</td>
<td>S2 1</td>
</tr>
<tr>
<td>S2 0</td>
<td>S2 1</td>
</tr>
<tr>
<td>S2 1</td>
<td>S3 0</td>
</tr>
<tr>
<td>S3 0</td>
<td>S0 1</td>
</tr>
<tr>
<td>S3 1</td>
<td>S0 0</td>
</tr>
</tbody>
</table>

### State Assignment (i.e. State Encoding):

<table>
<thead>
<tr>
<th>State</th>
<th>Assigned Code ai bi</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0 0</td>
</tr>
<tr>
<td>S1</td>
<td>0 1</td>
</tr>
<tr>
<td>S2</td>
<td>1 1</td>
</tr>
<tr>
<td>S3</td>
<td>1 0</td>
</tr>
</tbody>
</table>

---

# Iterative Circuit Synthesis: Mealy-based

### Encoded State Table:

<table>
<thead>
<tr>
<th>PS ai bi</th>
<th>input xi</th>
<th>NS ai+1 bi+1 output zi</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 0</td>
<td>01 0</td>
<td></td>
</tr>
<tr>
<td>00 1</td>
<td>00 1</td>
<td></td>
</tr>
<tr>
<td>01 0</td>
<td>11 0</td>
<td></td>
</tr>
<tr>
<td>01 1</td>
<td>11 1</td>
<td></td>
</tr>
<tr>
<td>11 0</td>
<td>11 1</td>
<td></td>
</tr>
<tr>
<td>11 1</td>
<td>10 0</td>
<td></td>
</tr>
<tr>
<td>10 0</td>
<td>00 1</td>
<td></td>
</tr>
<tr>
<td>10 1</td>
<td>00 0</td>
<td></td>
</tr>
</tbody>
</table>
Iterative Circuit Synthesis: Mealy-based

Karnaugh Maps:

Final 2-Level (Sum-of-Products) Minimized Equations -- with some logic optimization added

\[ \begin{align*}
ai+1 &= bi \\
bi+1 &= ai' \cdot xi' + xi' \cdot bi + ai' \cdot bi \\
zi &= xi ai' + xi\ ai = xi @ ai
\end{align*} \]

Iterative Circuit Synthesis: Mealy-based

Iterative Cell: Gate-Level Implementation

*regular instance (i.e., typical cell):* use equations on previous page

\[ \begin{align*}
x_i &\quad ai \\
&\quad bi \\
ai &\quad ai +1 \\
bi &\quad bi +1 \\
zi &
\end{align*} \]
Iterative Circuit Synthesis: Mealy-based

Iterative Cell: Gate-Level Implementation

Optimized leftmost cell (= Cell #1): simplify earlier equations

The equations for the optimized leftmost cell are:

- \( a_2 = 0 \)
- \( b_2 = a_1' \cdot x_1' + x_1' \cdot b_1 + a_1' \cdot b_1 = x_1' \)
- \( z_1 = x_1 \cdot a_1' + x_1' \cdot a_1 = x_1 \)

Special condition for cell #1: \( a_1 \cdot b_1 = 00! \)

Note: can eliminate \( a_2 \) output, and simply have a '0' \( a_2 \) input to next cell (Cell #2)

---

Iterative Circuit Synthesis: Mealy-based

Iterative Cell: Gate-Level Implementation

Optimized second-to-leftmost cell (= Cell #2): propagate simplification

The equations for the optimized second-to-leftmost cell are:

- \( a_2 = 0 \)
- \( b_2 = a_1' \cdot x_1' + x_1' \cdot b_1 + a_1' \cdot b_1 = x_1' \)
- \( z_2 = x_2 \cdot a_2' + x_2' \cdot a_2 = x_2 \oplus a_2 = x_2 \)

Special condition for cell #2: \( a_2 = 0! \)

\[ a_3 = b_2 \]
\[ b_3 = a_2' \cdot x_2' + x_2' \cdot b_2 + a_2' \cdot b_2 = x_2' + x_2' \cdot b_2 + b_2 = x_2' + b_2 \]
\[ z_2 = x_2 \cdot a_2' + x_2' \cdot a_2 = x_2 \oplus a_2 = x_2 \]
Iterative Circuit Synthesis: Mealy-based

Iterative Cell: Gate-Level Implementation

Optimized rightmost cell (= Cell #N): no need for next-state logic -- DELETE!

\[ zn = xn' + xn \cdot an \]

Special condition for cell #N (Mealy only):
no next-state logic