CPS104: Logic Design and Finite State Machines

Alexandru Duțu
Boolean function reduction

- \( A \cdot A = A \)
- \( A \cdot 0 = 0 \)
- \( A \cdot 1 = A \)
- \( A \cdot \neg A = 0 \)
- \( A + A = A \)
- \( A + 0 = A \)
- \( A + 1 = 1 \)
- \( A + \neg A = 1 \)
- \( A \cdot B = B \cdot A \)
- \( A \cdot (B + C) = (B + C) \cdot A = A \cdot B + A \cdot C \)

from Prof. Alvin Lebeck’s slides
DeMorgan’s theorem

- \( \sim(A+B) = \sim A \ast \sim B \)
- \( \sim(A \ast B) = \sim A + \sim B \)

from Prof. Alvin Lebeck’s slides
Logisim demo

• 4 bit adder
• AND with 4 inputs
• D flip-flops
• Why do we care about flip-flops?
Finite-state machines (FSM)

- Mealy state machines
- represents states and transitions
- symbolic state transition table
- encoded state transition table
- get the next state as a function of current state and input
- get the output as a function of next state
FSM example

• let us design the FSM of an odd parity checker

• output 1 for odd number of appearances of 1
Finite state machine

- $S_1$ - even
- $S_2$ - odd
Finite state machine

- S1 - even
- S2 - odd
Symbolic table

s - current state
i - input
n - next state
o - output

<table>
<thead>
<tr>
<th>s</th>
<th>i</th>
<th>n</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>0</td>
<td>s1</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>1</td>
<td>s2</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>s2</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>s1</td>
<td>1</td>
</tr>
</tbody>
</table>
Symbolic table

s - current state
i - input
n - next state
o - output

<table>
<thead>
<tr>
<th>s</th>
<th>i</th>
<th>n</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td>0</td>
<td>s1</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>1</td>
<td>s2</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>0</td>
<td>s2</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>s1</td>
<td>1</td>
</tr>
</tbody>
</table>
Encoded table

\[ n = s \wedge i \]

\[ o = s \]

<table>
<thead>
<tr>
<th>s</th>
<th>i</th>
<th>n</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Implementation

s - current state
n - next state
i - input
o - output

s - current state
n - next state
i - input
o - output