Synchronization Algorithm Example

• States can be represented as variables
  - Instead of binary state assignments
• Example: framer algorithm
  - When out-of-frame (OOF=1) it takes 3 consecutive frames with no framing errors (FE=0) to go in-frame (OOF=0)
  - When in-frame it takes 2 framing errors in 3 consecutive frames to go out-of-frame

G=good frame (no framing error, FE=0)
B=bad frame (framing error, FE=1)
Framer Example (continued)

Note alternate form of state table
State Reduction

• Minimizing the number of states usually helps to:
  ➢ Minimize # of flip-flops required for FSM
    • Minimum # FFs = $\lceil \log_2(\#\text{states}) \rceil$
  ➢ Minimize # gates (gate I/O) required for FSM
    • More Xs in K-maps

• Two states are equivalent if for all combinations of inputs:
  ➢ They give the same outputs
  ➢ They send the circuit either to the same state or to equivalent states

• When two states are equivalent one state can be removed without altering behavior of the circuit
Framer State Reduction Example

Equivalent states in red ⇒ remove GGB state & change GGB to 3G
State Assignment

• Assignment of binary values to states
  ➢ Determines locations of 1s in K-maps
  ➢ Determines locations of Xs in K-maps

• Helps to optimize in resultant circuit in terms of:
  ✓ Area
    » # gates
    » # gate I/O
  ✓ Performance – to maximize operating frequency
    » Gate delay
    » Propagation delay
**Framer State Assignment Example**

<table>
<thead>
<tr>
<th>Assign #1 (xyz)</th>
<th>Assign #2 (xyz)</th>
<th>Current State</th>
<th>Next State</th>
<th>OOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>0G</td>
<td>1G</td>
<td>0G</td>
</tr>
<tr>
<td>100</td>
<td>001</td>
<td>1G</td>
<td>2G</td>
<td>0G</td>
</tr>
<tr>
<td>110</td>
<td>010</td>
<td>2G</td>
<td>3G</td>
<td>0G</td>
</tr>
<tr>
<td>111</td>
<td>011</td>
<td>3G</td>
<td>3G</td>
<td>BGG</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
<td>BGG</td>
<td>GBG</td>
<td>0G</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>GBG</td>
<td>3G</td>
<td>0G</td>
</tr>
</tbody>
</table>

Green state assignment gives fewer gates & gate I/O for output OOF

\[
\text{OOF} = \bar{z}'
\]

\[
\text{OOF} = x'y' + yz'
\]
### Framer State Assignment Example

<table>
<thead>
<tr>
<th>Assign #1 (xyz)</th>
<th>Assign #2 (xyz)</th>
<th>Current State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>0G</td>
<td>1G</td>
</tr>
<tr>
<td>100</td>
<td>001</td>
<td>1G</td>
<td>2G</td>
</tr>
<tr>
<td>110</td>
<td>010</td>
<td>2G</td>
<td>3G</td>
</tr>
<tr>
<td>111</td>
<td>011</td>
<td>3G</td>
<td>3G</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
<td>BGG</td>
<td>GBG</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>GBG</td>
<td>3G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FE x</th>
<th>yz</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>100</td>
<td>XXX</td>
<td>101</td>
<td>XXX</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>110</td>
<td></td>
<td>111</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>11</td>
<td>000</td>
<td>000</td>
<td>011</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>000</td>
<td>XXX</td>
<td>000</td>
<td>XXX</td>
<td></td>
</tr>
</tbody>
</table>

Dx = \( FE' \)

Dy = \( FE'x + xyz \)

Dz = \( FE'z + FE'y + xyz \)

Dx = \( yz + FE'xz' \)

Dy = \( FE'y'z + FE'yz' \)

Dz = \( FE'z' + FE'y + FE'x \)

= \( FE' (x+y+z') \)
Framer State Assignment Example

\[ D_x = yz + FE'xz' \]
\[ D_y = FE'y'z + FE'yz' \]
\[ D_z = FE'z' + FE'y + FE'x \]
\[ = FE'(x+y+z') \]
\[ OOF = x'y' + yz' \]

12 gates
39 gate I/O
Framer State Assignment Example

\[ D_x = FE' \]
\[ D_y = FE'x + xyz \]
\[ D_z = FE'z + FE'y + xyz \]
\[ OOF = z' \]

Both circuits have identical behavior but choice of state assignment results in considerable difference in # gates and gate I/O for combinational logic.
State Assignment Rules

- General rules for assigning states such that the 1s are grouped in the K-maps:
  1. States that have the same next states for a given input value should be given logically adjacent assignments.
  2. States that are next states of a single present state under logically adjacent inputs should be given logically adjacent assignments.

If value1 = value2 then 
CS1 & CS2 should have d=1

If value1 & value2 have d=1, then 
NS1 & NS2 should have d=1
One-Hot FSMs

- One-hot is another state assignment method
  - Uses 1 state variable (and 1 FF) per state
  - Often uses more FFs but fewer combinational logic gates than traditional binary state assignments

- Two types of one-hots
  - One-hot-one
    - Each state has a single 1 in a field of 0s
  - One-hot-zero
    - One state is all 0s
    - Requires 1 less FFs than one-hot-one assignments

<table>
<thead>
<tr>
<th>Curr. State</th>
<th>Assignment</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-hot-1</td>
<td>1-hot-0</td>
<td>FE=0</td>
</tr>
<tr>
<td>0G</td>
<td>100000</td>
<td>00000</td>
</tr>
<tr>
<td>1G</td>
<td>010000</td>
<td>10000</td>
</tr>
<tr>
<td>2G</td>
<td>001000</td>
<td>01000</td>
</tr>
<tr>
<td>3G</td>
<td>000100</td>
<td>00100</td>
</tr>
<tr>
<td>BGG</td>
<td>000010</td>
<td>00010</td>
</tr>
<tr>
<td>GBG</td>
<td>000001</td>
<td>00001</td>
</tr>
</tbody>
</table>