CSE140: Components and Design Techniques for Digital Systems

Tajana Simunic Rosing
Announcements

• Midterm on Thursday, 5/15 at class time
  – Everything up to and including chap 8, app. A,B,C
  – 8 ½ x 11 sheet of handwritten notes, pencil and eraser

• Today:
  – FSM optimization
    • State assignment
    • FSM partitioning
  – FSM design example

Sources: TSR, Katz, Boriello, Vahid, Perkowski
State assignment strategies

- Choose bit vectors to assign to each “symbolic” state
  - huge number even for small values of state bits and states
    - intractable for state machines of any size
    - heuristics are necessary for practical solutions – no guarantee of optimality
  - optimize some metric for the combinational logic
    - size (amount of logic and number of FFs)
    - speed (depth of logic and fanout)
    - dependencies (decomposition)

- Possible strategies
  - sequential – just number states as they appear in the state table
  - random – pick random codes
  - one-hot – use as many state bits as there are states
  - output – use outputs to help encode states
  - heuristic – rules of thumb that seem to work in most cases

Sources: TSR, Katz, Boriello, Vahid, Perkowski
One-hot state assignment

• Simple
  – easy to encode
  – easy to debug

• Small logic functions
  – each state function requires only predecessor state bits as input

• Good for programmable devices
  – lots of flip-flops readily available
  – simple functions with small support (signals its dependent upon)

• Impractical for large machines
  – too many states require too many flip-flops
  – decompose FSMs into smaller pieces that can be one-hot encoded

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Heuristics for state assignment

- Encode adjacent states to minimize # of state bit changes
  - Use state maps

<table>
<thead>
<tr>
<th>Transition</th>
<th>1st case</th>
<th>2nd case</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0 - S1</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>S0 - S2</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>S1 - S3</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>S2 - S3</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>S3 - S4</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>S4 - S1</td>
<td>-</td>
<td>2</td>
</tr>
</tbody>
</table>

Total: 13 - 7

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Heuristics for state assignment

- **Goal:** maximize groupings of 1s in the next state & output functions
  - Helps minimize next state logic
- **Guidelines:**
  1. Adjacent codes to states that share a common next state
     
     \[
     \begin{array}{c|c|c|c}
     I & Q & Q^+ & O \\
     \hline
     i & a & c & j \\
     i & b & c & k
     \end{array}
     \]
     
     \[c = i \ast a + i \ast b\]

     ![Diagram 1]

  2. Adjacent codes to states that share a common ancestor state
     
     \[
     \begin{array}{c|c|c|c}
     I & Q & Q^+ & O \\
     \hline
     i & a & b & j \\
     k & a & c & l
     \end{array}
     \]
     
     \[b = i \ast a \quad c = k \ast a\]

     ![Diagram 2]

  3. Adjacent codes to states that have a common output behavior
     
     \[
     \begin{array}{c|c|c|c}
     I & Q & Q^+ & O \\
     \hline
     i & a & b & j \\
     i & c & d & j
     \end{array}
     \]
     
     \[j = i \ast a + i \ast c \quad b = i \ast a \quad d = i \ast c\]

     ![Diagram 3]

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Heuristics example

- Guidelines:
  1. Adjacent codes to states that share a common next state
  2. Adjacent codes to states that share a common ancestor state
  3. Adjacent codes to states that have a common output behavior

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Output-based encoding

- Reuse outputs as state bits
  - why create new functions for state bits when output can serve as well
  - fits in nicely with synchronous Mealy implementations

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Present State</th>
<th>Next State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C TL TS</td>
<td></td>
<td></td>
<td>ST H F</td>
</tr>
<tr>
<td>0 0</td>
<td>HG</td>
<td>HG</td>
<td>0 0 10</td>
</tr>
<tr>
<td>- 0</td>
<td>HG</td>
<td>HG</td>
<td>0 0 10</td>
</tr>
<tr>
<td>1 1</td>
<td>HG</td>
<td>HY</td>
<td>1 0 10</td>
</tr>
<tr>
<td>- - 0</td>
<td>HY</td>
<td>HY</td>
<td>0 0 10</td>
</tr>
<tr>
<td>- - 1</td>
<td>HY</td>
<td>FG</td>
<td>1 0 10</td>
</tr>
<tr>
<td>1 0</td>
<td>FG</td>
<td>FG</td>
<td>0 0 10</td>
</tr>
<tr>
<td>0 -</td>
<td>FG</td>
<td>FY</td>
<td>1 1 00</td>
</tr>
<tr>
<td>- 1</td>
<td>FG</td>
<td>FY</td>
<td>1 1 00</td>
</tr>
<tr>
<td>- 0</td>
<td>FY</td>
<td>FY</td>
<td>0 0 01</td>
</tr>
<tr>
<td>- 1</td>
<td>FY</td>
<td>HG</td>
<td>1 1 01</td>
</tr>
</tbody>
</table>

HG = ST'H1' H0' F1 F0' + ST H1 H0' F1' F0
HY = ST H1' H0' F1 F0' + ST' H1' H0 F1 F0'
FG = ST H1' H0 F1 F0' + ST' H1 H0' F1' F0'
FY = ST H1 H0' F1' F0' + ST' H1 H0' F1' F0

Output patterns are unique to states, we do not need ANY state bits - implement 5 functions (one for each output) instead of 7 (outputs plus 2 state bits)

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Current state assignment approaches

• For tight encodings using close to the minimum number of state bits
  – used in custom chip design

• One-hot encoding
  – easy for small state machines
  – generates small equations with easy to estimate complexity
  – common in FPGAs and other programmable logic

• Output-based encoding
  – ad hoc - no tools
  – most common approach taken by human designers
  – yields small circuits for most FSMs

Sources: TSR, Katz, Boriello, Vahid, Perkowski
State Partitioning

• Helps for large state machines
  – E.g. when next state logic is too large to implement in a programmable logic component

• Introduce *idle* states to synchronize partitioned FSMs

![Diagram of state transitions between S2, S3, S4, S5, SA, SB with connections labeled C3, C4, C5]

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Partition rules

- Source/destination transformation

- Hold condition for the idle state

- Multiple transitions to same source/destination
State Partitioning Example

Sources: TSR, Katz, Boriello, Vahid, Perkowski
FSM design example: Modulo-3 counter

- Design up/down modulo-3 counter with two inputs: count enable (C, where C=1 starts counting) and count direction (D). When D=1 count up, else count down.
Summary of FSM Optimization

• State minimization
  – straightforward in fully-specified machines
  – computationally intractable, in general (with don’t cares)

• State assignment
  – many heuristics
  – best-of-10-random just as good or better for most machines
  – output encoding can be attractive (especially for PAL implementations)

• State partitioning
  – Used for larger state machines for ease of implementation
  – Introduce “idle” states at the interface
  – Change transition conditions according to the rules of partitioning

Sources: TSR, Katz, Boriello, Vahid, Perkowski