Lecture 23

- Logistics:
  - HW8 due Wednesday, March 11
  - Extra credit due Friday, March 13
  - Review session Monday, March 16, 4:30 pm, Place TBA

- Last lecture:
  - General FSM Minimization

- Today:
  - State encoding
    - One-hot encoding
    - Output encoding
  - State partitioning

FSM design

- FSM-design procedure
  1. State diagram
  2. State-transition table
  3. State minimization
  4. State encoding
  5. Next-state logic minimization
  6. Implement the design

Usual example: A vending machine

- 15 cents for a cup of coffee
- Doesn’t take pennies or quarters
- Doesn’t provide any change

A vending machine: After state minimization

A vending machine: State encoding

A vending machine: Logic minimization
State encoding

- Assume \( n \) state bits and \( m \) states
  - \( 2^n / (2^n - m)! \) possible encodings
    - Example: 3 state bits, 4 states, 1680 possible state assignments
- Want to pick state encoding strategy that results in optimizing your criteria
  - FSM size (amount of logic and number of FFs)
  - FSM speed (depth of logic and fan-in/fan-out)
  - FSM ease of design or debugging

State-encoding strategies

- No guarantee of optimality
  - An intractable problem
- Most common strategies
  - Binary (sequential) – number states as in the state table
  - Random – computer tries random encodings
  - Heuristic – rules of thumb that seem to work well
    - e.g. Gray code – try to give adjacent states (states with an arc between them) codes that differ in only one bit position
  - One-hot – use as many state bits as there are states
  - Output – use outputs to help encode states
  - Hybrid – mix of a few different ones (e.g. One-hot + heuristic)

One-hot encoding

- One-hot: Encode \( n \) states using \( n \) flip-flops
  - Assign a single “1” for each state
    - Example: 0001, 0010, 0100, 1000
  - Propagate a single “1” from one flip-flop to the next
    - All other flip-flop outputs are “0”
- The inverse: One-cold encoding
  - Assign a single “0” for each state
    - Example: 1110, 1101, 1011, 0111
  - Propagate a single “0” from one flip-flop to the next
    - All other flip-flop outputs are “1”
- “almost one-hot” encoding (modified one-hot encoding)
  - Use no-hot (000…0) for the initial (reset state)
  - Assumes you never revisit the reset state till reset again.

Vending Machine: One-hot encoded transition table

<table>
<thead>
<tr>
<th>Present State Inputs</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_3Q_2Q_1Q_0 D N )</td>
<td>( D_3D_2D_1D_0 )</td>
<td>( D' N' )</td>
</tr>
<tr>
<td>0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0 1 0 0 0 0 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 1 0 1 0 0 0 0 0 0 0 0 0</td>
<td>0 1 0 0 0 0 0 0 0 0 0 0</td>
<td>0 1 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 1 1 0 0 0 0 0 0 0 0 0</td>
<td>1 0 1 0 0 0 0 0 0 0 0 0</td>
<td>1 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1 1 0 0 0 0 0 0 0 0</td>
<td>0 1 0 1 0 0 0 0 0 0 0 0</td>
<td>0 1 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>1 0 0 1 0 0 0 0 0 0 0 0</td>
<td>1 0 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>0 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 1 0 0 0 0 0 0 0 0 0 0</td>
<td>1 0 1 0 0 0 0 0 0 0 0 0</td>
<td>1 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

One-hot encoding (con't)

- Often the best/convenient approach for FPGAs
  - FPGAs have many flip-flops
- Draw FSM directly from the state diagram
  - + One product term per incoming arc
  - - Complex state diagram ⇒ complex design
  - Many states ⇒ many flip-flops

Vending Machine: Implementation

A vending machine: Implementation
Advantage of one-hot encoding: Designing from the state diagram

Output encoding

- Reuse outputs as state bits
  - Why create new functions when you can use outputs?
  - Bits from state assignments are the outputs for that state
    - Take outputs directly from the flip-flops

Vending machine

--- already in output encoding form

Example: Partition the machine

- Partition into two halves

Introduce idle states for each partition

- SA and SB handoff control between machines
Partitioning rules

Rule #1: Source-state transformation
Replace by transition to idle state (SA)

Rule #2: Destination state transformation
Replace with exit transition from idle state

Example: Six-state up/down counter

Example: Six state up/down counter

Example: 4-state machines need 2 state bits each – total 4 state bits
Each FSM may be much simpler to think about (and design logic for)

Why do this?
- Each FSM may be much simpler to think about (and design logic for)
- Essential to do this partitioning for large FSMs
Minimize communication between partitions

- Ideal world: Two machines handoff control
  - Separate I/O, states, etc.
- Real world: Minimize handoffs and common I/O
  - Minimize number of state bits that cross boundary
  - Merge common outputs

Mealy versus Moore partitions

- Mealy machine partitioning is undesirable
  - Inputs can affect outputs immediately
  - "output" can be a handoff to another machine!!!
- Moore machine partitioning is desirable
  - Input-to-output path always broken by a flip-flop
  - But...may take several clock cycles for input to propagate to output