Overview

- Last lecture
  - Sequential Logic Examples
- Today
  - State encoding
  - One-hot encoding
  - Output encoding

State encoding

- Assume n state bits and m states
  - \(2^n / (2^n - m)\) possible encodings \([m \geq n \geq \log_2(m)]\)
  - From binomial expansion
  - Example: 3 state bits, 4 states, 1680 possible state assignments
- Hard problem, with no known algorithmic solution
  - Can try heuristic approaches
  - Can try to optimize some metric
    - FSM size (amount of logic and number of FFs)
    - FSM speed (depth of logic and fanout)
    - FSM dependencies (decomposition)
  - Need to consider startup
    - Self-starting FSM or explicit reset input

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

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
  - Use no-hot (000...0) for the initial (reset state)
  - Assumes you never revisit the reset state

One-hot encoding (con't)

- Often the best approach for FPGAs
  - FPGAs have many flip-flops
  - One-hot machines use the least next-state logic
- Draw FSM directly from the state diagram
  - One product term per incoming arc
  - But complex state diagram \(\Rightarrow\) complex design
- One-hot designs have many possible failure modes
  - All states that aren't one-hot
  - Can create logic to reset the FSM if it enters illegal state
- Large machines require many flip-flops
  - Decompose design into smaller one-hot encoded sub-designs
    - \(n\cdot m\) states for two machines versus \(n\cdot m\) states for one

Vending machine again...

- Release item after receiving 15 cents
  - Single coin slot for dimes and nickels
  - Sensor specifies coin type
  - Machine does not give change
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>Q3Q2Q1Q0 D N</td>
<td>D3 D2 D1 D0 open</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1 0 0 0 0 1 0</td>
<td>0 0 0 1 0 0 0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1 0 0 0 0 1 0</td>
<td>0 0 0 1 0 0 0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>1 0 0 1 0 0 0 0 1 0</td>
<td>1 0 0 1 0 0 0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

Designing from the state diagram

| D0 = Q0D'N' | D1 = Q0N + Q1D'N' | D2 = Q0D + Q1N + Q2D'N' | D3 = Q1D + Q2D + Q2N + Q3 | OPEN = Q3 |

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
- ad hoc - no tools
- Yields small circuits for most FSMs
- Fits nicely with synchronous Mealy machines

Digital combination lock again...

- An output-encoded FSM
  - Punch in 3 values in sequence and the door opens
  - If there is an error the lock must be reset
  - After the door opens the lock must be reset
  - Inputs: sequence of number values, reset
  - Outputs: door open/close

Separate data path and control

- Design datapath first
- Control has 2 outputs
- After the state diagram
- Mux control to datapath
- Before the state encoding
- Lock open/closed
Draw the state diagram

Output encode the FSM

FSM has 4 state bits and 2 inputs...

Preset and Reset?

Answer: Yes!
FSM design: A 5-step process

1. Understand the problem
   - State diagram and state-transition table

2. Determine the machine’s states
   - Consider missing transitions: Will the machine start?
   - Minimize the state diagram: Reuse states where possible

3. Encode the states
   - Encode states, outputs with a reasonable encoding choice
   - Consider the implementation target

4. Design the next-state logic
   - Minimize the combinational logic
   - Choices made in steps 2 & 3 affect the logic complexity

5. Implement the FSM