Overview

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

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
  - 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

State encoding

- Assume \( n \) state bits and \( m \) states
  - \( 2^n / (2^m - m!) \) 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

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
    - \( n \times m \) states for two machines versus \( n \times m \) states for one

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

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>Q0 Q1 Q2 Q3 Q4</td>
<td>D2 D3 D4 D5</td>
<td>open</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 1 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 0 0 1 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ D_0 = Q_0D_4' \]
\[ D_1 = Q_1Q_3 + Q_1D_4' \]
\[ D_2 = Q_2Q_3 + Q_2D_4 + Q_2Q_1 + Q_1 \]
\[ OPEN = Q_1 \]

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

One-hot encoded vending machine

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

Designing from the state diagram

Separate data path and control

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

FSM has 4 state bits and 2 inputs...
- Output encoded!
  - Outputs and state bits are the same
- How do we minimize the logic?
  - FSM has 4 state bits and 2 inputs (equal, new)
  - 6-variable kmap?
- Notice the state assignment is close to one-hot
  - ERR state (0000) is only deviation
  - Is there a clever design we can use?

Design the datapath
- Choose simple control
  - 3-wire mux for datapath
  - Control is 001, 010, 100
  - Open/closed bit for lock state
  - Control is 0/1

Preset and Reset?
- Assume flip-flops have preset & reset inputs
- Can we encode the ERR state as reset?

Output encode the FSM
- FSM outputs
  - Mux control is 100, 010, 001
  - Lock control is 0/1
- State are: S0, S1, S2, S3, or ERR
  - Can use 3, 4, or 5 bits to encode
  - Have 4 outputs, so choose 4 bits
    - Encode mux control and lock control in state bits
    - Lock control is first bit, mux control is last 3 bits
      - S0 = 0001 (lock closed, mux first code)
      - S1 = 0010 (lock closed, mux second code)
      - S2 = 0100 (lock closed, mux third code)
      - S3 = 1000 (lock open)
    - ERR = 0000 (error, lock closed)

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