Lecture 22

- 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

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

After state minimization

![State diagram](image)

How many state encodings?

- Assume n state bits and m states
  - $2^n / (2^n - m)!$ possible encodings
    - Example: 3 state bits, 4 states, 1680 possible state assignments
- Which encoding is best?
  - 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"

- One-hot variants
  - "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

One-hot encoding

- 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

<table>
<thead>
<tr>
<th>present state inputs</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q0.Q1.Q2.D N D N</td>
<td>Q0.Q1.Q2.D N D N</td>
<td>Open</td>
</tr>
<tr>
<td>0 0 0 1 0 0</td>
<td>0 0 0 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0 0 1 0</td>
<td>0 1 0 0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1 0 0 0</td>
<td>1 0 1 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1 0 0 0</td>
<td>1 1 1 0 0 0</td>
<td>-</td>
</tr>
</tbody>
</table>

Designing from state diagram

\[
D_0 = Q_0D'N' \\
D_1 = Q_0N + Q_0D'N' \\
D_2 = Q_1D + Q_1D + Q_1N + Q_1 \\
D_3 = D + N \\
OPEN = Q_3
\]
### 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
- Yields small circuits for most FSMs

### FSM partitioning

- Break a large FSM into two or more smaller FSMs
  - Less states in each partition
    - Simpler minimization and state assignment
    - Smaller combinational logic
    - Shorter critical path
  - But more logic overall

### Introduce idle states

- SA and SB handoff control between machines

### Example

- Partition into two halves

### 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
Partitioning rules

Rule #3: Multiple transitions with same source or destination
Source \(\Rightarrow\) Replace by transitions to idle state (SA)
Destination \(\Rightarrow\) Replace with exit transitions from idle state

Example

Example 1
- Count sequence S0, S1, S2, S3, S4, S5
  - S2 goes to S4 and holds, leaves after S5
  - S5 goes to S6 and holds, leaves after S2
  - Down sequence is similar

Example 2
- 4-state machines need 2 state bits each – total 4 state bits
  - Enough to represent 16 states, though the combination of the two FSMs has only 6 different configurations
- Why do this?
  - Each FSM may be much simpler to think about (and design logic for) than the original FSM (not here, though)
  - Essential to do this partitioning for large FSMs

Example 3

Minimize communication

- 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