Lecture 26

◆ Logistics
  ■ HW8 due Friday
  ■ Ant extra credit due Friday
  ■ Final exam a week from today, 12/8 8:30am-10:20am here
  ■ Review time/place TBA

◆ Last lecture
  ■ Simplification

◆ Today
  ■ State encoding
    □ One-hot encoding
    □ Output encoding

Example: A vending machine

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

■ 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
A vending machine: State minimization

### Symbolic State Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0¢</td>
<td>D 0  N 0</td>
<td>0¢ 0</td>
<td>open</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>5¢ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>10¢ 0</td>
<td></td>
</tr>
<tr>
<td>5¢</td>
<td>0 0</td>
<td>5¢ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>10¢ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>15¢ 0</td>
<td></td>
</tr>
<tr>
<td>10¢</td>
<td>0 0</td>
<td>10¢ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>15¢ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>15¢ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>15¢</td>
<td>-</td>
<td>15¢ 1</td>
<td></td>
</tr>
</tbody>
</table>

A vending machine: State encoding

### Present State Transition Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>1 0</td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0</td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>1 0</td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>1 0</td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1 0</td>
<td>-</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>1 1</td>
<td>-</td>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>
A vending machine: Logic minimization

\[ D_1 = Q_1 + D + Q_0 N \]
\[ D_0 = Q_0' N + Q_0 N' + Q_1 N + Q_1 D \]
\[ OPEN = Q_1 Q_0 \]

A vending machine: Implementation
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.

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
Example: A vending machine ... again

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

- 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

One-hot encoded transition table

<table>
<thead>
<tr>
<th>present state inputs</th>
<th>next state $D_3, D_2, D_1, D_0$</th>
<th>output open</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_3 Q_2 Q_1 Q_0$</td>
<td>$D$ $N$</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>- - - -</td>
<td>-</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>- - - -</td>
<td>-</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>- - - -</td>
<td>-</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>- -</td>
<td>1</td>
</tr>
</tbody>
</table>

CSE370, Lecture 26
Designing from the state diagram

\[ D_0 = Q_0D'N' \]
\[ D_1 = Q_0N + Q_1D'N' \]
\[ D_2 = Q_0D + Q_1N + Q_2D'N' \]
\[ D_3 = Q_1D + Q_2D + Q_2N + Q_3 \]
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

- ad hoc - no tools
  - Yields small circuits for most FSMs
Vending machine
--- already in output encoding form

\[ D_0 = Q_0D'N' \]
\[ D_1 = Q_0N + Q_1D'N' \]
\[ D_2 = Q_0D + Q_1N + Q_2D'N' \]
\[ D_3 = Q_1D + Q_2D + Q_2N + Q_3 \]

OPEN = Q_3

Example: Digital combination lock

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

Draw the state diagram
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

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)
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 for all five states?

- Notice the state assignment is close to one-hot
  - ERR state (0000) is only deviation
  - Is there a clever design we can use?

Encode 4 state bits

A clever way for ERR is to use Preset/reset in existing flipflops.

\[ S_0^+ = S_0' \]
\[ S_1^+ = S_0E' + S_1' \]
\[ S_2^+ = S_1E' + S_2' \]
\[ S_3^+ = S_2E' + S_3' \]

Preset_0 = start
Preset_{1,2,3} = 0
Reset_0 = start'(E'N + (Q_0'+Q_1'+Q_2'+Q_3'))
Reset_{1,2,3} = start + (E'N + (Q_0+Q_1+Q_2+Q_3'))

Not equal & new

Already in ERR
D_0 = Q_0N'
D_1 = Q_1EN + Q_1N'
D_2 = Q_1EN + Q_2N'
D_3 = Q_2EN + Q_3

Preset_0 = start
Preset_{1,2,3} = 0
Reset_0 = start'(E'N + (Q_0 + Q_1 + Q_2 + Q_3))'
Reset_{1,2,3} = start + (EN + (Q_0 + Q_1 + Q_2 + Q_3))'

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