Lecture 18

◆ Logistics
  ■ HW5 due today (with extra 10%)
  ■ HW5 due Friday (20% off on Mon 10:29am, Sol’n posted 10:30am)
  ■ HW6 out, due Wednesday
  ■ Office hours canceled on Friday (am out of town)
  ■ Brian will cover lecture on Friday
  ■ Midterm 2 covers materials up to Monday lecture & HW6

◆ Last lecture
  ■ Registers/counters
  ■ Design counters

◆ Today
  ■ More counter designs
  ■ Finite state machine design

The “WHY” slide

◆ Finite State Machine (FSM)
  ■ This is what we have been waiting for in this class. Using combinational and sequential logics, now you can design a lot of clever digital logic circuits for functional products. We will learn different steps you take to go from word problems to logic circuits. We first talk about a simplified version of FSM which is a counter.
Another 3-bit up counter: with T flip flops

1. Draw a state diagram
2. Draw a state-transition table
3. Encode the next-state functions
   - Minimize the logic using k-maps
4. Implement the design

1. Draw a state diagram

111 → 110 → 101
100

001
010 → 011
000

3-bit up-counter
2. Draw a state-transition table

- Like a truth-table
  - State encoding is easy for counters → Use count value

<table>
<thead>
<tr>
<th>current state</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 000</td>
<td>001 1</td>
</tr>
<tr>
<td>1 001</td>
<td>010 2</td>
</tr>
<tr>
<td>2 010</td>
<td>011 3</td>
</tr>
<tr>
<td>3 011</td>
<td>100 4</td>
</tr>
<tr>
<td>4 100</td>
<td>101 5</td>
</tr>
<tr>
<td>5 101</td>
<td>110 6</td>
</tr>
<tr>
<td>6 110</td>
<td>111 7</td>
</tr>
<tr>
<td>7 111</td>
<td>000 0</td>
</tr>
</tbody>
</table>

3-bit up-counter

3. Encode the next state functions

T flip-flops

\[
T_1 := \\
T_2 := \\
T_3 := \\
\]

<table>
<thead>
<tr>
<th>C3</th>
<th>C2</th>
<th>C1</th>
<th>N3</th>
<th>N2</th>
<th>N1</th>
<th>T3</th>
<th>T2</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>1 0</td>
<td>0 1</td>
<td>0 1</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>1 1</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CSE370, Lecture 18
4. Implement the design

One more counter example:
A 5-state counter with D flip flops

- Counter repeats 5 states in sequence
  - Sequence is 000, 010, 011, 101, 110, 000

**Step 1: State diagram**

**Step 2: State transition table**
Assume D flip-flops

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>C  B  A</td>
<td>C+  B+  A+</td>
</tr>
<tr>
<td>0  0  0</td>
<td></td>
</tr>
<tr>
<td>0  0  1</td>
<td></td>
</tr>
<tr>
<td>0  1  0</td>
<td></td>
</tr>
<tr>
<td>0  1  1</td>
<td></td>
</tr>
<tr>
<td>1  0  0</td>
<td></td>
</tr>
<tr>
<td>1  0  1</td>
<td></td>
</tr>
<tr>
<td>1  1  0</td>
<td></td>
</tr>
<tr>
<td>1  1  1</td>
<td></td>
</tr>
</tbody>
</table>
5-state counter (con’t)

Step 3: Encode the next state functions

\[ C' = \]

\[ B' = \]

\[ A' = \]

5-state counter (con’t)

Step 4: Implement the design
5-state counter (con’t)

- Is our design robust?
  - What if the counter starts in a 111 state?

Does our counter get stuck in invalid states???

![State diagram of a 5-state counter]

5-state counter (con’t)

- Back-annotate our design to check it

Fill in state transition table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

A+ = BC’
B+ = B’ + A’C’
C+ = A

Draw state diagram
Self-starting counters

- Invalid states should always transition to valid states
  - Assures startup
  - Assures bit-error tolerance

- Design your counters to be self-starting
  - Draw all states in the state diagram
  - Fill in the entire state-transition table
  - May limit your ability to exploit don't cares
    - Choose startup transitions that minimize the logic

Finite state machines: more than counters

- FSM: A system that visits a finite number of logically distinct states

- Counters are simple FSMs
  - Outputs and states are identical
  - Visit states in a fixed sequence without inputs

- FSMs are typically more complex than counters
  - Outputs can depend on current state and on inputs
  - State sequencing depends on current state and on inputs
FSM design

- **Counter-design procedure**
  1. State diagram
  2. State-transition table
  3. Next-state logic minimization
  4. Implement the 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

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 diagram

A vending machine: State transition table
A vending machine: State minimization

A vending machine: State encoding
A vending machine: Logic minimization

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