Lecture 16

◆ Logistics
  ■ HW6 due Wednesday
  ■ Midterm 2 creeping up (next week Wednesday 5/21)
  ■ Midterm 2 covers materials up to Friday lecture & HW7
  ■ Review next Tuesday 6:30pm?

◆ Last lecture
  ■ Timing issues for asynchronous inputs
  ■ Registers/counters
  ■ Wrapped up on sequential logic building blocks

◆ Today
  ■ Introduction to finite state machines
  ■ Counters as finite state machines

“States” for finite state machines are kept in the storage elements

◆ Combinational logic and storage elements
  ■ Localized feedback loops
  ■ Choice of storage elements alters the logic
Finite-state machines (FSMs)

- States: Possible storage-element values
- Transitions: Changes in state
  - Clock synchronizes the state changes
- Sequential logic
  - Sequences through a series of states
  - Based on inputs and present state

Sequential logic

- Sequences through a series of states
- Based on inputs and present state

Drawing state diagrams

- Show input values on transition arcs
- Show output values in state nodes
Counters revisited

- Great simple examples of state machines
  - Output is the counter's state
- Next state is well defined
  - Does not depend on input (no inputs)

FSM design procedure (using counters)

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

We will use a ‘3-bit up counter’ as an example in two different ways today
1. Draw a state diagram

![State Diagram](image)

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. Encode the next state functions

- Assume D flip-flops as state elements

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

- \( N1 := C1' \)
- \( N2 := C1C2' + C1'C2 \)
- \( N3 := C1C2C3' + C1'C3 + C2'C3 \)

4. Implement the design

- 3 flip-flops hold state
  - Counter is synchronously clocked
- Minimized logic computes next state
Another example:
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

![State Diagram](image_url)
2. Draw a state-transition table

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

```
current state  next state
0   000    001    1
1   001    010    2
2   010    011    3
3   011    100    4
4   100    101    5
5   101    110    6
6   110    111    7
7   111    000    0
```

3-bit up-counter

3. Encode the next state functions

T flip-flops

\[
\begin{align*}
T1 & := 1 \\
T2 & := C1 \\
T3 & := C1 \land C2
\end{align*}
\]

<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</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

CSE370, Lecture 16
4. Implement the design