Announcements

• Class newsgroup created
• Use it for partner finding
• First assignment is to find partners
  – Due this Friday

• Sections are on this week

• HW 1 out tomorrow
  – Work alone
Stateful Components

- Until now is combinatorial logic
  - Output is computed when inputs are present
  - System has no internal state
  - Nothing computed in the present can depend on what happened in the past!

- Need a way to record data
- Need a way to build stateful circuits
- Need a state-holding device

Bistable Devices

- In stable state, $\bar{A} = B$

- How do we change the state?
SR Latch

- Set-Reset (S-R) Latch
- Q: Stored value and its complement

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q̅</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q̅</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

D Latch

- Data Latch
  - Easier to use than an SR latch
  - No possibility of entering an undefined state
- When D changes, Q changes
  - ... immediately (after a delay of 2 Ors and 2 NOTs)
- Need to control when the output changes
Clocks

- Clocks help with modifying the contents of state-holding elements
- A free running signal
  - Generated by an oscillating crystal
- Clock signal has a fixed cycle time: cycle period
- Clock frequency = 1/cycle time

Edge-triggering

- Can design circuits to change on the rising or falling edge
- Trigger on rising edge = positive edge-triggered
- Trigger on falling edge = negative edge-triggered
- Inputs must be stable just before the triggering edge
First Attempt

• How does the output behave?

© Kavita Bala, Computer Science, Cornell University
First Attempt

- How does the output behave?
- Changes in D that occur when the clock is low are deferred until clock high
- Changes when clock is high are registered immediately

Master-Slave Flip-Flop

- Outputs change only on falling edges
- Data is captured on rising edges
- 1 cycle delay
  - but works out perfectly – data for the next stage is ready 1 cycle ahead of time
Registers

- A register is simply a set of master-slave flip-flops in parallel with a shared clock.

\[ \text{clk} \]

\[ \text{D0} \]
\[ \text{D1} \]
\[ \text{D2} \]
\[ \text{D3} \]

Example: Keyboard

- When a key is pressed
  - Compute a 7-bit key identifier

- Store this keycode
  - The computer may not be ready to read it right away.
Summary

• We can now build interesting devices with sensors
  – Using combinatorial logic

• We can also store data values
  – In state-holding elements
  – Coupled with clocks
Finite State Machines

- An electronic machine which has
  - external inputs
  - externally visible outputs
  - internal state

- Output and next state depend on
  - inputs
  - current state
Abstract Model of FSM

Machine is

\[ M = (S, I, O, \delta) \]

\( S \): Finite set of states
\( I \): Finite set of inputs
\( O \): Finite set of outputs
\( \delta \): State transition function

• Next state depends on present input and present state

Primitive State Diagram

Legend

input/output

state

Legend

etc.
Machine State Diagram

Legend

0 or 1/z

s1 → s0

Next State

0/1

0/1

10 → 11

e tc.

Automata Model

Input

Comb Ckt

Next State Variables

or

Present State Variables

Excitation Variables

Memory

Feedback

© Kavita Bala, Computer Science, Cornell University
Designing a FSM

- Draw a state diagram
- Write down state transition table
- Assign numbers to states
- Determine logic equations for all flip-flops and outputs

A Simple Example

- Goal: flash hello on LEDs
- Inputs: clock
- Outputs: Just one 7-segment LED

- Flash “h” then “e” then “l” then “l” then “o”
  - h = <0011101>
  - e = <0111110>
  - l = <0010110>
  - o = <1110111>

© Kavita Bala, Computer Science, Cornell University
HELLObox: State Diagram

• Determine the transitions
  – label all edges (transitions) with the inputs that cause them, unlabeled edges are unconditional transitions
  – show start state

HELLObox: State Table

• Build state table
  – rote encoding of the state diagram

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
<td>0011101</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>0111110</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>0010110</td>
</tr>
<tr>
<td>S3</td>
<td>S4</td>
<td>0010110</td>
</tr>
<tr>
<td>S4</td>
<td>S0</td>
<td>1110111</td>
</tr>
</tbody>
</table>
**HELLObox: State Assignment 1**

- Assign bit patterns to states
  - Try to make resulting device simple
  - One option is shown

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>0011101</td>
</tr>
<tr>
<td>001</td>
<td>010</td>
<td>0111110</td>
</tr>
<tr>
<td>010</td>
<td>011</td>
<td>0010110</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
<td>0010110</td>
</tr>
<tr>
<td>100</td>
<td>000</td>
<td>1110111</td>
</tr>
</tbody>
</table>

- Determine logic equations for
  - every bit of output
  - next state
  - for every flip-flop and output

**HELLObox #1**

- 10 bits of information (7 outputs + 3 bits of next state) are computed by the combinatorial circuit
- All 10 bits have non-trivial logic equations
HELLObox: State Assignment 2

- Assign bit patterns to states to make the resulting device simple
- Here, we use far more bits than necessary
  - to simplify the combinatorial circuit

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111010</td>
<td>01111100</td>
<td>0011101</td>
</tr>
<tr>
<td>01111100</td>
<td>00101100</td>
<td>0111110</td>
</tr>
<tr>
<td>00101100</td>
<td>00101101</td>
<td>0010110</td>
</tr>
<tr>
<td>00101101</td>
<td>11101110</td>
<td>0010110</td>
</tr>
<tr>
<td>11101110</td>
<td>00111010</td>
<td>1110111</td>
</tr>
</tbody>
</table>

HELLObox #2

- 15 bits of information (7 outputs + 8 bits of next state) are computed by the combinatorial circuit
- 8 bits have non-trivial logic equations
  - all 7 outputs are simple pass-throughs