State & Finite State Machines

Hakim Weatherspoon
CS 3410, Spring 2012
Computer Science
Cornell University

See P&H Appendix C.7, C.8, C.10, C.11
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

Finite State Machines
Bistable Devices

• Stable and unstable equilibria?

• In stable state, \( \bar{A} = B \)

• How do we change the state?
SR Latch

- **Set-Reset (S-R) Latch**
- Stores a value $Q$ and its complement

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>$\bar{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
SR Latch

- Set-Reset (S-R) Latch
- Stores a value $Q$ and its complement

<table>
<thead>
<tr>
<th>$S$</th>
<th>$R$</th>
<th>$Q$</th>
<th>$\overline{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$S=1$ and $R=1$ ?
SR Latch

- Set-Reset (S-R) Latch
- Stores a value Q and its complement
- S=1 and R=1?
(Unclocked) D Latch

- Data (D) 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
**Data (D) 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

- **Clock** helps coordinate state changes
  - Usually generated by an oscillating crystal
  - Fixed period; frequency = 1/period
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
Clock Disciplines

• **Level sensitive**
  – State changes when clock is high (or low)

• **Edge triggered**
  – State changes at clock edge
Clock Methodology

- Negative edge, synchronous
  - Signals must be stable near falling clock edge

- Positive edge synchronous

- Asynchronous, multiple clocks, ...
D Latch with Clock

<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>0</td>
<td>0</td>
<td>Q</td>
<td>Q̅</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>forbidden</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clk</th>
<th>D</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>Q</td>
<td>Q̅</td>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>
D Latch with Clock

Level Sensitive D Latch

Clock high:
set/reset (according to D)

Clock low:
keep state (ignore D)

<table>
<thead>
<tr>
<th>clk</th>
<th>D</th>
<th>Q</th>
<th>$\overline{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>$\overline{Q}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>$\overline{Q}$</td>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>
Edge-Triggered D Flip-Flop

- D Flip-Flop
  - Edge-Triggered
  - Data is captured when clock is high
  - Outputs change only on falling edges

D D L L Q Q L L Q Q

clk clk

D X Q Q

Q Q Q Q Q Q
Registers

Register

- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, ...

 clk

4-bit reg

D0

D1

D2

D3
An Example: What will this circuit do?

- Reset
- Run

32-bit reg

WE R

Decoder

Clk

+1

Decoder

. . . . . . .
Recap

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

We can also store data values
  • In state-holding elements
  • Coupled with clocks
Administrivia

Make sure partner in same Lab Section *this week*

Lab2 is out

Due in one week, next Monday, start early

Work *alone*

But, use your resources

- Lab Section, Piazza.com, Office Hours, Homework Help Session,
- Class notes, book, Sections, CSUGLab

No Homework this week
Administrivia

Check online syllabus/schedule
• http://www.cs.cornell.edu/Courses/CS3410/2012sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):
• Tuesday, February 28th
• Thursday, March 29th
• Thursday, April 26th

Schedule is subject to change
Collaboration, Late, Re-grading Policies

“Black Board” Collaboration Policy
• Can discuss approach together on a “black board”
• Leave and write up solution independently
• Do not copy solutions

Late Policy
• Each person has a total of four “slip days”
• Max of two slip days for any individual assignment
• Slip days deducted first for any late assignment, cannot selectively apply slip days
• For projects, slip days are deducted from all partners
• 20% deducted per day late after slip days are exhausted

Regrade policy
• Submit written request to lead TA, and lead TA will pick a different grader
• Submit another written request, lead TA will regrade directly
• Submit yet another written request for professor to regrade.
Finite State Machines
Revisit Voting Machine

How do we create a vote counter machine?
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
Revisit Voting Machine

![Diagram of a voting machine circuit with labeled components: 32-bit registers (WE), decoder (3-to-8), mux, LED decoder, and input ports labeled with 'detect' and 'enc'.]
Automata Model

Finite State Machine

- inputs from external world
- outputs to external world
- internal state
- combinational logic
Input: up or down
Output: on or off
States: A, B, C, or D
Input: =up or =down
Output: =on or =off
States: =A, =B, =C, or =D
Input: 0=up or 1=down
Output: 1=on or 1=off
States: 00=A, 01=B, 10=C, or 11=D
General Case: **Mealy Machine**

Outputs and next state depend on both current state and input
Special Case: **Moore Machine**

Outputs depend only on current state
Moore Machine Example

Input: **up** or **down**
Output: **on** or **off**
States: A, B, C, or D
Example: Digital Door Lock

Digital Door Lock

Inputs:
• keycodes from keypad
• clock

Outputs:
• “unlock” signal
• display how many keys pressed so far
Door Lock: Inputs

Assumptions:

• signals are synchronized to clock
• Password is B-A-B

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Ø (no key)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>‘A’ pressed</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>‘B’ pressed</td>
</tr>
</tbody>
</table>
Door Lock: Outputs

Assumptions:

- High pulse on U unlocks door
Door Lock: Simplified State Diagram

- Idle "0"
  - "B" to G1 "1"
  - else to G2 "2"
  - any to G3 "3", U

- G1 "1"
  - "A" to G2 "2"
  - "B" to B3 "3"
  - else to Idle "0"
  - any to B2 "2"

- G2 "2"
  - "B" to G3 "3", U
  - else to B3 "3"
  - any to B2 "2"

- B1 "1"
  - else to B2 "2"
  - any to B3 "3"

- B2 "2"
  - else to B3 "3"
  - any to B3 "3"
Door Lock: Simplified State Diagram

Idle

Ø

G1

“1”

Ø

G2

“2”

“A”

G3

“3”, U

“B”

any

“B”

else

else

else

else

B1

“1”

B2

“2”

Ø

else

Ø

else

else

Ø

else
Door Lock: Simplified State Diagram

<table>
<thead>
<tr>
<th>Cur. State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Idle "0"

G1 "1"

G2 "2"

G3 "3", U

B1 "1"

B2 "2"

Ø

“B”

“A”

“B”

else

else

else

else

else

else
Door Lock: Simplified State Diagram

- **Idle** "0"
- **G1** "1"
- **G2** "2"
- **G3** "3", U
- **B1** "1"
- **B2** "2"

### Table: Cur. State vs. Output

<table>
<thead>
<tr>
<th>Cur. State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>G1</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>G2</td>
<td>&quot;2&quot;</td>
</tr>
<tr>
<td>G3</td>
<td>&quot;3&quot;, U</td>
</tr>
<tr>
<td>B1</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>B2</td>
<td>&quot;2&quot;</td>
</tr>
</tbody>
</table>
Door Lock: Simplified State Diagram

<table>
<thead>
<tr>
<th>Cur. State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1 &quot;1&quot;</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td></td>
<td>&quot;A&quot;</td>
<td>&quot;B&quot;</td>
</tr>
<tr>
<td>Idle &quot;0&quot;</td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>&quot;B&quot;</td>
<td>&quot;B&quot;</td>
</tr>
<tr>
<td></td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td>B1 &quot;1&quot;</td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>B2 &quot;2&quot;</td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;1&quot;</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;2&quot;</td>
<td>&quot;2&quot;</td>
</tr>
</tbody>
</table>
Door Lock: Simplified State Diagram

**Cur. State** | **Input** | **Next State**
---|---|---
Idle | Ø | Idle
Idle | “B” | G1
Idle | “A” | B1
G1 | Ø | G1
G1 | “A” | G2
G1 | “B” | B2
G2 | Ø | B2
G2 | “B” | G3
G2 | “A” | Idle
G3 | any | Idle
B1 | Ø | B1
B1 | K | B2
B2 | Ø | B2
B2 | K | Idle
## State Table Encoding

<table>
<thead>
<tr>
<th>S₂</th>
<th>S₁</th>
<th>S₀</th>
<th>D₃</th>
<th>D₂</th>
<th>D₁</th>
<th>D₀</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### State to Input Encoding

<table>
<thead>
<tr>
<th>S₂</th>
<th>S₁</th>
<th>S₀</th>
<th>K</th>
<th>A</th>
<th>B</th>
<th>S'₂</th>
<th>S'₁</th>
<th>S'₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### State Table

<table>
<thead>
<tr>
<th>State</th>
<th>S₂</th>
<th>S₁</th>
<th>S₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Door Lock: Implementation

Strategy:
(1) Draw a state diagram (e.g. Moore Machine)
(2) Write output and next-state tables
(3) Encode states, inputs, and outputs as bits
(4) Determine logic equations for next state and outputs
We can now build interesting devices with sensors

- Using combinational logic

We can also store data values

- Stateful circuit elements (D Flip Flops, Registers, ...)
- Clock to synchronize state changes
- But be wary of asynchronous (un-clocked) inputs
- State Machines or Ad-Hoc Circuits