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
Unstable Devices
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>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>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>
**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>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( S=1 \) and \( R=1 \)?
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>ĀQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>ĀQ</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>

\( S = 1 \) 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
(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
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

input

clock

√   X
Clock Disciplines

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

• Edge triggered
  – State changes at clock edge

positive edge-triggered

negative edge-triggered
Clock Methodology

- Negative edge, synchronous

- Positive edge synchronous

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

\[
\begin{array}{c}
\text{D} \\
\text{clk}
\end{array}
\quad
\text{S} \\
\text{R}
\]

\[
\begin{array}{c}
\text{Q} \\
\overline{\text{Q}}
\end{array}
\quad
\begin{array}{c}
\text{clk} \\
\text{D}
\end{array}
\]

<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>Q</td>
<td>\overline{Q}</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>forbidden</td>
<td></td>
</tr>
</tbody>
</table>

<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>
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>\bar{Q}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>\bar{Q}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>\bar{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

Diagram of Edge-Triggered D Flip-Flop

Clock (clk)

D Input

X Intermediate

Q Output
Registers

- D flip-flops in parallel
- Shared clock
- Extra clocked inputs: write_enable, reset, ...

 clk

 D0

 D1

 D2

 D3

 4-bit reg

 4 4
An Example: What will this circuit do?

- **Reset**
- **Run**

32-bit reg

Clk

+1

Decoder

Digital Display
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?
Revisit Voting Machine

[Diagram of voting machine components, including registers (reg), multiplexers (mux), decoder (3-to-8), and LED display.]
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 \textit{and} present state
Revisit Voting Machine

[Diagram of a voting machine system with components labeled reg, WE, mux, LED, decoder, and detect.]
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
**FSM Example**

Input: =up or =down  
Output: =on or =off  
States: =A, =B, =C, or =D
FSM Example

Legend

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

Legend

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>$\emptyset$ (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

Diagram: A state diagram showing the transitions between states.

- **Idle** ("0"): States without any input action.
- **G1** ("1"): States triggered by "A" input.
- **G2** ("2"): States triggered by "B" input.
- **G3** ("3", U): States triggered by "3", else...
- **B1** ("1"): States triggered by "3", else...
- **B2** ("2"): States triggered by "2", else...
- **B3** ("3"): States triggered by "3", else...

Transitions:
- From **Idle** to **G1** ("1") with "B" input.
- From **G1** ("1") to **B1** ("1") with "B" input.
- From **G1** ("1") to **G2** ("2") with "A" input.
- From **G2** ("2") to **G3** ("3", U) with "B" input.
- From **G2** ("2") to **B3** ("3") with "A" input.
- From **G3** ("3", U) to **Idle** ("0") with "any" input.
- From **B1** ("1") to **B2** ("2") with "any" input.
- From **B2** ("2") to **B3** ("3") with "any" input.
- From **B3** ("3") to **Idle** ("0") with "any" input.
Door Lock: Simplified State Diagram

- **Idle “0”**
  - \(\emptyset\)
  - else

- **G1 “1”**
  - \(\emptyset\)
  - “A”
  - else

- **G2 “2”**
  - \(\emptyset\)
  - “B”
  - else

- **G3 “3”, U**
  - any

- **B1 “1”**
  - \(\emptyset\)
  - else

- **B2 “2”**
  - \(\emptyset\)
  - else

- **B**
  - else
Door Lock: Simplified State Diagram

- **Idle** "0"
  - \( \emptyset \)
  - "B"
  - else

- **G1** "1"
  - \( \emptyset \)
  - "A"
  - else

- **G2** "2"
  - \( \emptyset \)
  - "B"
  - else

- **G3** "3", U
  - Cur. State
  - Output

- **B1** "1"
  - \( \emptyset \)
  - else

- **B2** "2"
  - \( \emptyset \)
  - else
Door Lock: Simplified State Diagram

The diagram illustrates the state transitions and outputs for the door lock system. The states include:

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

The transitions are labeled with events such as "A" and "B". The output table is as follows:

<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

- **Idle**
  - "0"
  - \(\emptyset\)
  - else

- **G1**
  - "1"
  - "A"
  - \(\emptyset\)

- **B1**
  - "1"
  - \(\emptyset\)

- **B2**
  - "2"

- **Cur. State**
- **Input**
- **Next State**
Door Lock: Simplified State Diagram

<table>
<thead>
<tr>
<th>Cur. State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Ø</td>
<td>Idle</td>
</tr>
<tr>
<td>Idle</td>
<td>“B”</td>
<td>G1</td>
</tr>
<tr>
<td>Idle</td>
<td>“A”</td>
<td>B1</td>
</tr>
<tr>
<td>G1</td>
<td>Ø</td>
<td>G1</td>
</tr>
<tr>
<td>G1</td>
<td>“A”</td>
<td>G2</td>
</tr>
<tr>
<td>G1</td>
<td>“B”</td>
<td>B2</td>
</tr>
<tr>
<td>G2</td>
<td>Ø</td>
<td>B2</td>
</tr>
<tr>
<td>G2</td>
<td>“B”</td>
<td>G3</td>
</tr>
<tr>
<td>G2</td>
<td>“A”</td>
<td>Idle</td>
</tr>
<tr>
<td>G3</td>
<td>any</td>
<td>Idle</td>
</tr>
<tr>
<td>B1</td>
<td>Ø</td>
<td>B1</td>
</tr>
<tr>
<td>B1</td>
<td>K</td>
<td>B2</td>
</tr>
<tr>
<td>B2</td>
<td>Ø</td>
<td>B2</td>
</tr>
<tr>
<td>B2</td>
<td>K</td>
<td>Idle</td>
</tr>
</tbody>
</table>
## State Table Encoding

<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>

### Truth Table

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

### Next State Table

<table>
<thead>
<tr>
<th>Next State</th>
<th>D₃</th>
<th>D₂</th>
<th>D₁</th>
<th>D₀</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### State Transition Table

<table>
<thead>
<tr>
<th>S₂</th>
<th>S₁</th>
<th>S₀</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>
</tr>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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
Summary

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