State and Finite State Machines

Prof. Hakim Weatherspoon
CS 3410, Spring 2015
Computer Science
Cornell University

See P&H Appendix B.7, B.8, B.10, B.11
Announcements

Make sure you are
• Registered for class, can access CMS
• Have a Section you can go to.
• *Lab Sections are required.*
  • “Make up” lab sections *only 8:40am Wed, Thur, or Fri*
  • Bring laptop to Labs
• Project partners are required for projects.
  • Have project partner in same Lab Section, if possible

HW1 and Lab1 available today, after lecture
• *Do HW1 problems with lecture, i.e. finish part 1 and 2 this week*
• Work alone on both
• But, use your resources
  - Lab Section, Piazza.com, Office Hours, Homework Help Session,
  - Class notes, book, Sections, CSUGLab
Announcements

Make sure to go to your Lab Section this week
Completed Lab1 due before winter break, Friday, Feb 13th
Note, a Design Document is due when you submit Lab1 final circuit
Work alone

Homework1 is out
Due a week before prelim1, Monday, February 23rd
Work on problems incrementally, as we cover them in lecture
Office Hours for help
Work alone

Work alone, BUT use your resources
• Lab Section, Piazza.com, Office Hours
• Class notes, book, Sections, CSUGLab
Announcements

Check online syllabus/schedule

• Slides and Reading for lectures
• Office Hours

• *Pictures of all TAs*

• Homework and Programming Assignments

• Dates to keep in Mind
  • Prelims: Tue Mar 3rd and Thur April 30th
  • *Lab 1: Due next Friday, Feb 13th before Winter break*
  • Proj2: Due Thur Mar 26th before Spring break
  • Final Project: Due when final would be (not known until Feb 14th)

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
- 25% 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.
Stateful Components

Until now is combinational 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
Goals for Today

State

• How do we store one bit?
• Attempts at storing (and changing) one bit
  – Set-Reset Latch
  – D Latch
  – D Flip-Flops
  – Master-Slave Flip-Flops
• Register: storing more than one bit, N-bits

Basic Building Blocks

• Decoders and Encoders

Finite State Machines (FSM)

• How do we design logic circuits with state?
• Types of FSMs: Mealy and Moore Machines
• Examples: Serial Adder and a Digital Door Lock
Goal

How do we store store *one* bit?
First Attempt: Unstable Devices
Second Attempt: Bistable Devices

• Stable and unstable equilibria?

A Simple Device
Third Attempt: Set-Reset Latch
Third Attempt: Set-Reset 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></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></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>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
<th>NOR</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Third Attempt: Set-Reset Latch

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

Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.
Next Goal
How do we avoid the forbidden state of S-R Latch?
Fourth Attempt: (Unclocked) D Latch

Fill in the truth table?

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
<th>NOR</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Takeaway

Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding the forbidden state.
Next Goal
How do we coordinate state changes to a D Latch?
Clocks

Clock helps coordinate state changes

- Usually generated by an oscillating crystal
- Fixed period; frequency = 1/period
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

- Edge-Triggered: Signals must be stable near falling clock edge

- Positive edge synchronous
Fifth Attempt: D Latch with Clock
Fifth Attempt: D Latch with Clock

Fill in the truth 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>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></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>
Fifth Attempt: D Latch with Clock

Fill in the truth table

<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></td>
<td></td>
<td>hold</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></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>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></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>
Fifth Attempt: D Latch with Clock

Level Sensitive D Latch

Clock high:
set/reset (according to D)

Clock low:
keep state (ignore D)

Fill output signal for Q

<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>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></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>
**Sixth Attempt: Edge-Triggered D Flip-Flop**

- **D Flip-Flop**
  - Edge-Triggered
  - Data captured when clock is high
  - Output changes only on falling edges

**Activity #1:** Fill in timing graph and values for X and Q

<table>
<thead>
<tr>
<th>clk</th>
<th>D</th>
<th>X</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clk</th>
<th>X</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Takeaway

Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flop (aka Master-Slave D Flip-Flop) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.
Next Goal

How do we store more than one bit, \( N \) bits?
Registers

Register

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

4-bit reg

clk
Takeaway

Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flip (aka Master-Slave D Flip-Flip) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.

An $N$-bit register stores $N$-bits. It is be created with $N$ D-Flip-Flops in parallel along with a shared clock.
An Example: What will this circuit do?
Decoder Example: 7-Segment LED

7-Segment LED

- photons emitted when electrons fall into holes
Decoder Example: 7-Segment LED Decoder

3 inputs
• encode 0 – 7 in binary

7 outputs
• one for each LED
## 7 Segment LED Decoder Implementation

<table>
<thead>
<tr>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></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>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</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>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

[Image of 7-segment LED display]
# 7 Segment LED Decoder Implementation

<table>
<thead>
<tr>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![LED Display Diagram](image)
Basic Building Blocks We have Seen

- Binary encoder
- Binary decoder
- Multiplexer

Diagram showing the interconnections and labels.
Encoder

\[ \text{Log}_2(N) \text{ outputs wires} \]

N Input wires

\[ \vdots \]

\[ 7 \]

\[ 6 \]

\[ 5 \]

\[ 4 \]

\[ 3 \]

\[ 2 \]

\[ 1 \]

\[ 0 \]

encoder

e.g. Voting: Can only vote for one out of N candidates, so N inputs.

But can encode vote efficiently with binary encoding.
Example Encoder Truth Table

A 3-bit encoder with 4 inputs for simplicity
Basic Building Blocks Example: Voting

Ballots

The 3410 optical scan vote reader machine
Recap

We can now build interesting devices with sensors

• Using combinationial logic

We can also store data values (aka Sequential Logic)

• In state-holding elements
• Coupled with clocks
Goals for Today

State

• How do we store one bit?
• Attempts at storing (and changing) one bit
  – Set-Reset Latch
  – D Latch
  – D Flip-Flops
  – Master-Slave Flip-Flops
• Register: storing more than one bit, N-bits

Basic Building Blocks

• Decoders and Encoders

Finite State Machines (FSM)

• How do we design logic circuits with state?
• Types of FSMs: Mealy and Moore Machines
• Examples: Serial Adder and a Digital Door Lock
Finite State Machines
Next Goal

How do we design logic circuits with state?
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
- **δ**: State transition function

Next state depends on present input *and* present state
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 0=off
States: 00=A, 01=B, 10=C, or 11=D
Mealy Machine

General Case: Mealy Machine

Outputs and next state depend on both current state and input
Moore Machine

Special Case: Moore Machine

Outputs depend only on current state
Moore Machine FSM Example

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

Input: **up** or **down**
Output: **on** or **off**
States: A, B, C, or D
Activity#2: Create a Logic Circuit for a Serial Adder

Add two infinite input bit streams

- streams are sent with least-significant-bit (lsb) first
- How many states are needed to represent FSM?
- Draw and Fill in FSM diagram

...10110

...01111

...00101

Strategy:
(1) Draw a state diagram (e.g. Mealy 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
FSM: State Diagram

---

___ states:
Inputs: ??? and ???
Output: ???
FSM: State Diagram

---

___ states:
Inputs: ??? and ???
Output: ???
FSM: State Diagram

(2) Write down all input and state combinations
FSM: State Diagram

(3) Encode states, inputs, and outputs as bits
### FSM: State Diagram

(4) Determine logic equations for next state and outputs

<table>
<thead>
<tr>
<th>??</th>
<th>??</th>
<th>Current state</th>
<th>?</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
  • State Machines or Ad-Hoc Circuits