The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.
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 Friday 11:40am or 1:25pm
  • Bring laptop to Labs
• Project partners are required for projects starting w/ project 2
  • Have project partner in same Lab Section, if possible
  • WICC hosting a partner finding event Feb 12 @ 6pm in 3rd floor lounge of Gates
Announcements

Make sure to go to your Lab Section this week

Completed **Proj1** due **before** winter break, Friday, Feb 16th

Note, a Design Document is due when you submit Proj1 final circuit

Work **alone**

Work alone, **BUT** use your resources

- Lab Section, Piazza.com, Office Hours
- Class notes, book, Sections, CSUGLab
Announcements

Check online syllabus/schedule
  • http://www.cs.cornell.edu/Courses/CS3410/2018sp/schedule

  Slides and Reading for lectures

  Office Hours

  Pictures of all TAs

  Project and Reading Assignments

  Dates to keep in Mind
    • Prelims: Thur Mar 15th and Thur May 3rd
    • Proj 1: Due next Friday, Feb 16th before Winter break
    • Proj3: Due before Spring break
    • Final Project: May 15th

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

“White Board” Collaboration Policy
• Can discuss approach together on a “white board”
• Leave, watch a movie (e.g. Strange Things), 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 regrade within a week of receiving score
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
Goal

How do we 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>̅Q</th>
<th>OR</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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?
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?
Aside: 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

Clock Methodology

• Negative edge, synchronous

Edge-Triggered $\rightarrow$ signals must be stable near falling edge

“near” = before and after

t_{setup} $\rightarrow$ thold
• Inverter prevents SR Latch from entering 1,1 state
## Round 2: D Latch (1)

- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

### C = 1, D Latch *transparent*: set/reset (according to D)

### C = 0, D Latch *opaque*: keep state (ignore D)

<table>
<thead>
<tr>
<th>C</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>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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Round 2: D Latch (1)

- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

C = 1, D Latch **transparent**: set/reset (according to D)
C = 0, D Latch **opaque**: keep state (ignore D)

<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>0</td>
<td>1</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>C</th>
<th>D</th>
<th>Q</th>
<th>̅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>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

No Change
Reset
Set
Round 2: D Latch (1)
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>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( \overline{1} )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( \overline{0} )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Round 3: D Flip-Flop

- Edge-Triggered
- Data captured when clock high
- Output changes only on falling edges
### Round 3: D Flip-Flop

When \( CLK \) falls \((1 \rightarrow 0)\), \( Q \) gets \( X \), \( X \) cannot change.

When \( CLK \) rises \((0 \rightarrow 1)\), now \( X \) can change, \( Q \) does not change.

**Clock = 1:** L1 transparent  
L2 opaque

**Clock = 0:** L1 opaque  
L2 transparent
Edge-Triggered D Flip-Flop

- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges
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?
Register

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

4-bit reg

clk
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.

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

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

- 7 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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
N Input wires

Encoder

Log₂(N) outputs wires

e.g. Voting: Can only vote for one out of N candidates, so N inputs. But can encode vote efficiently with binary encoding.
A 3-bit encoder with 4 inputs for simplicity

Example Encoder Truth Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>o0</th>
<th>o1</th>
<th>o2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Basic Building Blocks Example: Voting

The 3410 optical scan vote reader machine
Recap

We can now build interesting devices with sensors

• Using combinatorial logic

We can also store data values (aka Sequential Logic)

• In state-holding elements
• Coupled with clocks
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