State

Prof. Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

[Weatherspoon, Bala, Bracy, and Sirer]
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

\[ S \rightarrow Q \rightarrow \bar{Q} \rightarrow R \]
Third Attempt: Set-Reset Latch

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

### 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>

### OR and NOR

<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?
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

- Negative edge, synchronous

Edge-Triggered signals must be stable near falling edge

“near” = before and after

\[ t_{\text{setup}} \quad t_{\text{hold}} \]
Round 2: D Latch (1)

- Inverter prevents SR Latch from entering 1,1 state

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

\(\overline{Q}\) No Change
C = 0:
\(\overline{Q}\) Reset
\(Q\) Set
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>( \bar{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>( \bar{Q} ) hold</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1 reset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0 set</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>( \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>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>1</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>
Round 3: D Flip-Flop

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

Clock = 1: L1 \textit{transparent} \quad L2 \textit{opaque}

When \textit{CLK} rises (0\rightarrow1),
\textit{now X} can change,
\textit{Q does not change}

Clock = 0: L1 \textit{opaque} \quad L2 \textit{transparent}

When \textit{CLK} falls (1\rightarrow0),
\textit{Q gets X, X cannot change}
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

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

Register

![Diagram of a 4-bit register with D flip-flops and clock inputs.](image-url)
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.

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?
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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>0</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>0</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>1</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>0</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>1</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>0</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>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Basic Building Blocks We have Seen

- Binary Encoder
- Binary Decoder
- Multiplexor
Encoders

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.
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

detected

enc

8 3

7LED decode
Basic Building Blocks We have Seen

- Binary Encoder
- Binary Decoder
- Multiplexor

Symbols:
- $2^N$
- $N$
- $M$
- $2^{M-1}$
Recap
We can now build interesting devices with sensors
• Using combinational 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