Combinational circuits are stateless.
Their output is a function *only* of the current input.
Basic Combinational Circuits

Encoders and Decoders
Multiplexers
Shifters

Circuit Timing

Critical and Shortest Paths
Glitches

Arithmetic Circuits

Ripple Carry Adder
Adder/Subtractor
Carry Lookahead Adder
Overview: Decoder

A decoder takes a $k$-bit input and produces $2^k$ single-bit outputs.

The input determines which output will be 1, all others 0. This representation is called \textit{one-hot encoding}.
1:2 Decoder

The smallest decoder: one bit input, two bit outputs
2:4 Decoder
Decoder outputs are simply minterms. Those values can be constructed as a flat schematic (manageable at small sizes) or hierarchically, as below.
3:8 Decoder

Applying *hierarchical design* again, the 2:4 DEC helps construct a 3:8 DEC.
Priority Encoder

An encoder designed to accept any input bit pattern.

<table>
<thead>
<tr>
<th>$l_3$</th>
<th>$l_2$</th>
<th>$l_1$</th>
<th>$l_0$</th>
<th>$V$</th>
<th>$O_1$</th>
<th>$O_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$V = l_3 + l_2 + l_1 + l_0$

$O_1 = l_3 + \overline{l_3}l_2$

$O_0 = l_3 + \overline{l_3} \overline{l_2}l_1$
Overview: Multiplexer (or Mux)

A mux has a $k$–bit selector input and $2^k$ data inputs (multi or single bit).

It outputs a single data output, which has the value of one of the data inputs, according to the selector.
2:1 Mux Circuit

There are a handful of implementation strategies. E.g., a truth table and k-map are feasible for a design of this size.

<table>
<thead>
<tr>
<th>$S$</th>
<th>$l_1$</th>
<th>$l_0$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
</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>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
2:1 Mux Circuit

There are a handful of implementation strategies. E.g., a truth table and k-map are feasible for a design of this size.

<table>
<thead>
<tr>
<th>$S$</th>
<th>$l_1$</th>
<th>$l_0$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
</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>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
There are a handful of implementation strategies. E.g., a truth table and k-map are feasible for a design of this size.

<table>
<thead>
<tr>
<th>$S$</th>
<th>$I_1$</th>
<th>$I_0$</th>
<th>$O$</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
</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>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1:2 Mux Circuit

1:2 Decoder!
4:1 Mux Circuit
Muxing Wider Values (Overview)
Muxing Wider Values (Components)
Overview: Shifters

A shifter shifts the inputs bits to the left or to the right.

There are various types of shifters.

- **Barrel**: Selector bits indicate (in binary) how far to the left to shift the input.
- **L/R with enable**: Two control bits (upper enables, lower indicates direction).

In either case, bits may “roll out” or “wraparound”
Example: Barrel Shifter with Wraparound

\[ 11001010 \]

\[ 01010110 \]

\[ 011 \]

\[ 8 \]

\[ 3 \]

\[ 011 \]

\[ 8 \]

\[ 01010110 \]

\[ 15 \] / \[ 31 \]
Main idea: wire up all possible shift amounts and use muxes to select correct one.
Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.
Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.
Main idea: wire up all possible shift amounts and use muxes to select correct one.
Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.
Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.
There is a delay between inputs and outputs, due to:

- Limited currents charging capacitance
- The speed of light
The Simplest Timing Model

- Each gate has its own propagation delay \( t_p \).
- When an input changes, any changing outputs do so after \( t_p \).
- Wire delay is zero.
It is difficult to manufacture two gates with the same delay; better to treat delay as a range.

- Each gate has a minimum and maximum propagation delay $t_p(\text{min})$ and $t_p(\text{max})$.
- Outputs may start changing after $t_p(\text{min})$ and stabilize no later than $t_p(\text{max})$. 

A More Realistic Timing Model
Critical Paths and Short Paths

How slow can this be?
How slow can this be?

The **critical path** has the longest possible delay.

\[
t_p(\text{max}) = t_p(\text{max, AND}) + t_p(\text{max, OR}) + t_p(\text{max, AND})
\]
How fast can this be?

The **shortest path** has the least possible delay.

\[ t_p(\text{min}) = t_p(\text{min, AND}) \]
A glitch is when a single change in input values can cause multiple output changes.

Glitches *may* occur when there are multiple paths of different length from input $I$ to output $O$. 
Glitches

A glitch is when a single change in input values can cause multiple output changes.

Glitches *may* occur when there are multiple paths of different length from input \( I \) to output \( O \).
Glitches

A glitch is when a single change in input values can cause multiple output changes.

Glitches *may* occur when there are multiple paths of different length from input $I$ to output $O$. 
A glitch is when a single change in input values can cause multiple output changes.

Glitches may occur when there are multiple paths of different length from input $I$ to output $O$. 
Preventing Single Input Glitches

Additional terms can prevent single input glitches (at a cost of a few extra gates).

\[ \begin{array}{c|cccc}
\text{A} & 1 & 0 & 0 & 0 \\
\text{B} & 1 & 1 & 1 & 0 \\
\text{C} & 1 & 1 & 1 & 0 \\
\end{array} \]
Preventing Single Input Glitches

Additional terms can prevent single input glitches (at a cost of a few extra gates).
Arithmetic: Addition

Adding two one-bit numbers: $A$ and $B$

Produces a two-bit result: $C$ and $S$ (carry and sum)

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$S$</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>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>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Half Adder
Full Adder

In general, due to a possible carry in, you need to add \textit{three bits}:

\[
\begin{array}{c|c|c|c}
C_i & A & B & C_o \\hline
000 & 0 & 0 & 0 \\hline
001 & 0 & 1 & 0 \\hline
010 & 0 & 1 & 1 \\hline
011 & 1 & 0 & 0 \\hline
100 & 0 & 1 & 1 \\hline
101 & 1 & 0 & 1 \\hline
110 & 1 & 0 & 0 \\hline
111 & 1 & 1 & 1
\end{array}
\]
A Four-Bit Ripple-Carry Adder
A Two’s Complement Adder/Subtractor

Overflow in Two’s-Complement Representation

When is the result too positive or too negative?

<table>
<thead>
<tr>
<th>+</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>+10</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>+11</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>+00</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>+00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>+00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>+00</td>
<td>00</td>
<td></td>
</tr>
</tbody>
</table>

The result does not fit when the top two carry bits differ.

A

B

A

−1

B

−1

S

−1

S

Overflow

· · ·

27 / 31
Overflow in Two’s-Complement Representation

When is the result too positive or too negative?

<table>
<thead>
<tr>
<th></th>
<th>−2</th>
<th>−1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>10</td>
<td>10</td>
<td>00</td>
<td>×</td>
</tr>
<tr>
<td>−2</td>
<td>+10</td>
<td>00</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>−1</td>
<td>10</td>
<td>11</td>
<td>01</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>+11</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00</td>
<td>11</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+00</td>
<td>10</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>+01</td>
<td>00</td>
<td>01</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>+01</td>
<td>01</td>
<td>10</td>
<td>×</td>
</tr>
</tbody>
</table>

The result does not fit when the top two carry bits differ.
Ripple-Carry Adders are Slow

The *depth* of a circuit is the number of gates on a critical path.

This four-bit adder has a depth of 8.

*n*-bit ripple-carry adders have a depth of $2n$. 
Carry Generate and Propagate

The carry chain is the slow part of an adder; carry-lookahead adders reduce its depth using the following trick:

\[
\begin{array}{cccc}
0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 \\
\end{array}
\]

For bit \( i \),

\[
C_{i+1} = A_i B_i + A_i C_i + B_i C_i \\
= A_i B_i + C_i (A_i + B_i) \\
= G_i + C_i P_i
\]

K-map for the carry-out function of a full adder

Generate \( G_i = A_i B_i \) sets carry-out regardless of carry-in.

Propagate \( P_i = A_i + B_i \) copies carry-in to carry-out.
Carry Lookahead Adder

Expand the carry functions into sum-of-products form:

\[ C_{i+1} = G_i + C_i P_i \]

\[ C_1 = G_0 + C_0 P_0 \]
\[ C_2 = G_1 + C_1 P_1 \]
\[ = G_1 + (G_0 + C_0 P_0) P_1 \]
\[ = G_1 + G_0 P_1 + C_0 P_0 P_1 \]

\[ C_3 = G_2 + C_2 P_2 \]
\[ = G_2 + (G_1 + G_0 P_1 + C_0 P_0 P_1) P_2 \]
\[ = G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2 \]

\[ C_4 = G_3 + C_3 P_3 \]
\[ = G_3 + (G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2) P_3 \]
\[ = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3 \]
The 74283 Binary Carry-Lookahead Adder (From National Semiconductor)

Carry out $i$ has $i + 1$ product terms, largest of which has $i + 1$ literals.

If wide gates don’t slow down, delay is independent of number of bits.

More realistic: if limited to two-input gates, depth is $O(\log_2 n)$. 