Encoders and Decoders
Decoders

Input: $n$-bit binary number
Output: 1-of-$2^n$ one-hot code

<table>
<thead>
<tr>
<th>2-to-4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>out</strong></td>
</tr>
<tr>
<td>00</td>
<td>0001</td>
</tr>
<tr>
<td>01</td>
<td>0010</td>
</tr>
<tr>
<td>10</td>
<td>0100</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
</tr>
</tbody>
</table>
## Decoders

Input: $n$-bit binary number  
Output: 1-of-$2^n$ one-hot code

<table>
<thead>
<tr>
<th>2-to-4 decoder</th>
<th>3-to-8 decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>out</strong></td>
</tr>
<tr>
<td>00</td>
<td>0001</td>
</tr>
<tr>
<td>01</td>
<td>0010</td>
</tr>
<tr>
<td>10</td>
<td>0100</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decoders

Input: $n$-bit binary number
Output: 1-of-$2^n$ one-hot code

<table>
<thead>
<tr>
<th>2-to-4</th>
<th>3-to-8 decoder</th>
<th>4-to-16 decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>out</td>
<td>in</td>
</tr>
<tr>
<td>00</td>
<td>0001</td>
<td>000</td>
</tr>
<tr>
<td>01</td>
<td>0010</td>
<td>001</td>
</tr>
<tr>
<td>10</td>
<td>0100</td>
<td>010</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1111</td>
</tr>
</tbody>
</table>
The 74138 3-to-8 Decoder
A '138 Spotted in the Wild

Pac-Man (Midway, 1980)
General $n$-bit Decoders

Every minterm

$I_1 \cdots I_2 I_1$

$I_1 \cdots I_2 I_1$

$\vdots$

$I_1 \cdots I_2 I_1$

$I_1 \cdots I_2 I_1$

$I_1 \cdots I_2 I_1$

$I_1 \cdots I_2 I_1$

$I_1 \cdots I_2 I_1$

Implementing a function with a decoder:

E.g., $F = A C + B C$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$F$</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>
General $n$-bit Decoders

Implementing a function with a decoder:

E.g., $F = A\overline{C} + BC$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Every minterm
The 74148 Priority Encoder

Input: 1-of-$2^n$

Output: $n$-bit binary number for highest priority input

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E 01234567</td>
<td>210 G E</td>
</tr>
<tr>
<td>1 XXXXXXXX</td>
<td>111 1 1</td>
</tr>
<tr>
<td>0 11111111</td>
<td>111 1 0</td>
</tr>
<tr>
<td>0 XXXXXXXX0</td>
<td>000 0 1</td>
</tr>
<tr>
<td>0 XXXXXXXX01</td>
<td>001 0 1</td>
</tr>
<tr>
<td>0 XXXXXXXX011</td>
<td>010 0 1</td>
</tr>
<tr>
<td>0 XXXXXXXX0111</td>
<td>011 0 1</td>
</tr>
<tr>
<td>0 XXXX01111</td>
<td>100 0 1</td>
</tr>
<tr>
<td>0 XXX011111</td>
<td>101 0 1</td>
</tr>
<tr>
<td>0 XX0111111</td>
<td>110 0 1</td>
</tr>
<tr>
<td>0 X01111111</td>
<td>111 0 1</td>
</tr>
<tr>
<td>0 011111111</td>
<td>111 0 1</td>
</tr>
</tbody>
</table>
A ’148 Spotted in the Wild

Users would connect wires to interrupt sources; pull-ups quiet unconnected interrupts

OB68K1A Single-board Computer (Omnibyte 1983)
Multiplexers
The Two-Input Multiplexer

<table>
<thead>
<tr>
<th>$S$</th>
<th>$B$</th>
<th>$A$</th>
<th>$Y$</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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Two-Input Multiplexer

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>A</th>
<th>Y</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>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>
The Four-Input Mux

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$A$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$B$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$C$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$D$</td>
</tr>
</tbody>
</table>
Two-input Muxes in the Wild

Quad 2-to-1 mux 3B selects color from a sprite or the background

Pac-Man (Midway, 1980)
General $2^n$-input muxes

\[ Y = I_0 \overline{S_n} \cdots \overline{S_2} \overline{S_1} + I_1 S_n \cdots S_2 S_1 + I_2 S_n \cdots S_2 S_1 + \cdots + I_{2^n-2} S_n \cdots S_2 \overline{S_1} + I_{2^n-1} S_n \cdots S_2 S_1 \]
Using a Mux to Implement an Arbitrary Function

\[ F = A\overline{C} + BC \]

<table>
<thead>
<tr>
<th>( C )</th>
<th>( B )</th>
<th>( A )</th>
<th>( F )</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>
Using a Mux to Implement an Arbitrary Function

\[ F = \overline{AC} + BC \]

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
<th>F</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>

Apply each value in the truth table:
Using a Mux to Implement an Arbitrary Function

\[ F = \overline{AC} + BC \]

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
<th>F</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>

Apply each value in the truth table:
Using a Mux to Implement an Arbitrary Function

\[ F = \overline{A C} + B C \]

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
<th>F</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>

Apply each value in the truth table:
Using a Mux to Implement an Arbitrary Function

\[ F = \overline{A} \overline{C} + BC \]

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


Using a Mux to Implement an Arbitrary Function

\[ F = A\overline{C} + BC \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Using a Mux to Implement an Arbitrary Function

\[ F = A\overline{C} + BC \]

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

Can always remove a select and feed in 0, 1, S, or \( \overline{S} \).
Using a Mux to Implement an Arbitrary Function

\[ F = A\bar{C} + BC \]

Can always remove a select and feed in 0, 1, S, or \( \bar{S} \).

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

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Using a Mux to Implement an Arbitrary Function

\[ F = A \overline{C} + BC \]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Can always remove a select and feed in 0, 1, \( S \), or \( \overline{S} \).
Using a Mux to Implement an Arbitrary Function

\[ F = A\overline{C} + BC \]

Can always remove a select and feed in 0, 1, S, or \( \overline{S} \).

\[ \begin{array}{cccc}
C & B & A & F \\
0 & 0 & 0 & 0 \\
& 1 & 1 & \\
0 & 1 & 0 & 0 \\
& 1 & 1 & \\
1 & 0 & 0 & 0 \\
& 1 & 0 & \\
1 & 1 & 0 & 1 \\
& 1 & 1 & \\
\end{array} \]

\[ \begin{array}{cccc}
C & B & F \\
0 & 0 & A \\
0 & 1 & A \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array} \]

Diagram of a multiplexer (Mux) with inputs A, B, and C, and output Y.
Using a Mux to Implement an Arbitrary Function

\[ F = \overline{A}C + BC \]

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

Can always remove a select and feed in 0, 1, S, or \( \overline{S} \).

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

In this case, the function just happens to be a mux: (not always the case!)
Timing
Computation Always Takes Time

Always a delay between inputs and outputs. Causes:

- 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 $t_p(\text{min})$. 
How slow can this be?
How slow can this be?

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

\[ t_p(\text{max}) = t_p(\text{max, AND}) + t_p(\text{max, OR}) + t_p(\text{max, AND}) \]
Critical Paths and Short Paths

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

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

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

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

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

Adding such redundancy only works for single input changes; glitches may be unavoidable when multiple inputs change.
Arithmetic Circuits
Arithmetic: Addition

Adding two one-bit numbers:

\[ A \text{ and } B \]

Produces a two-bit result:

\[ C \quad S \]

(carry and sum)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Full Adder

In general, you need to add three bits:

\[
\begin{array}{c}
111000 \\
111010 \\
+ 11100 \\
\hline
1010110
\end{array}
\]

<table>
<thead>
<tr>
<th>CiAB</th>
<th>CoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0 0</td>
</tr>
<tr>
<td>001</td>
<td>0 1</td>
</tr>
<tr>
<td>010</td>
<td>0 1</td>
</tr>
<tr>
<td>011</td>
<td>1 0</td>
</tr>
<tr>
<td>100</td>
<td>0 1</td>
</tr>
<tr>
<td>101</td>
<td>1 0</td>
</tr>
<tr>
<td>110</td>
<td>1 0</td>
</tr>
<tr>
<td>111</td>
<td>1 1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
0 + 0 &= 00 \\
0 + 1 + 0 &= 01 \\
0 + 0 + 1 &= 01 \\
0 + 1 + 1 &= 10 \\
1 + 1 + 1 &= 11 \\
1 + 1 + 0 &= 10
\end{align*}
\]
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>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>10</td>
<td>10</td>
<td>-10</td>
<td>00</td>
</tr>
<tr>
<td>-1</td>
<td>10</td>
<td>11</td>
<td>-11</td>
<td>01</td>
</tr>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

The result does not fit when the top two carry bits differ.
### 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>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

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

![Overflow diagram](attachment:overflow_diagram.png)
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:

For bit $i$,

\[
C_{i+1} = A_i B_i + A_i C_i + B_i C_i
\]

\[
= A_i B_i + C_i (A_i + B_i)
\]

\[
= G_i + C_i P_i
\]

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

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 \( O(\log_2 n) \).