Starting from the bottom: 0 vs 1 vs MUM

1. Recognizing that voltage levels can represent binary values.


CMOS - 3 terminals: source, drain, gate

- Two types of transistors
- Voltage is applied at the source

N-type: When high voltage is applied at the gate, the drain current is high (voltage). When voltage is low at the gate, drain current is low.

P-type: Opposite behavior of N-type.

So, can build an inverter: Vin → Vout
2. An inverter is just one example of a gate. Others are:

- **AND**
- **OR**
- **Not** (inverter)
- **NOR**
- **NAND**

These correspond to boolean functions, which can be represented by truth tables: (1 = true, 0 = false)

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

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

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

Truth tables can have arbitrarily many inputs:
- multiple functions can be represented by a single table with multiple output columns.

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

Note: number of rows = \(2^n\) number of inputs.
Boolean functions can also be represented by formulas, where + is OR, \( \cdot \) is AND, and \( \overline{A} \) is not \( A \).

In previous chart:
\[
D = (A + B) \cdot \overline{C} \\
E = B + C
\]

These formulas form Boolean algebra, with the following laws:

Identity:
\[
A + 0 = A \\
A \cdot 1 = A
\]

Inverse:
\[
A \cdot \overline{A} = 0 \\
A + \overline{A} = 1
\]

Associativity:
\[
(A + B) + C = A + (B + C) \\
(A \cdot B) \cdot C = A \cdot (B \cdot C)
\]

Commutativity:
\[
A \cdot B = B \cdot A \\
A + B = B + A
\]

Distributivity:
\[
A \cdot (B + C) = (A \cdot B) + (A \cdot C) \\
A + (B \cdot C) = A + B \cdot (A + C)
\]

DeMorgan's Laws:
\[
\overline{(A + B)} = \overline{A} \cdot \overline{B} \\
\overline{(A \cdot B)} = \overline{A} + \overline{B}
\]
Now that we know about boolean functions, we can build circuits to represent them!
\[ A + (B \cdot C) \]

Any boolean function can be implemented using AND, OR, and NOT gates.

Also can be implemented using a single kind of gate, NAND.

- **NOT:** \( \overline{A} = \overline{A} \cdot \overline{A} \)
- **AND:** \( A \cdot B = \overline{(A \cdot B)} \)
- **OR:** \( A + B = \overline{(A \cdot B)} \)

The circuits that implement boolean functions are called **combinational** logic.
- No state result depends only on inputs.

Circuits that retain state are called **sequential** logic.
- Memory elements.

Address logical unit

Logical unit

Registers
- Memory