Digital Logic

Gateway to computer science

transistors, gates, circuits, Boolean algebra

https://cs.wellesley.edu/~cs240/s20/
Digital data/computation = Boolean

Boolean value (**bit**): 0 or 1

Boolean functions (AND, OR, NOT, ...)

Electronically:

bit = high voltage vs. low voltage

Boolean functions = logic gates, built from transistors

Abstraction!
Transistors (more in lab)

If *Base* voltage is high:
Current may flow freely from *Collector* to *Emitter*.

If *Base* voltage is low:
Current may not flow from *Collector* to *Emitter*.

<table>
<thead>
<tr>
<th>( V_{in} )</th>
<th>( V_{out} )</th>
<th>( in )</th>
<th>( out )</th>
<th>( in )</th>
<th>( out )</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>high</td>
<td>0</td>
<td>1</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>high</td>
<td>low</td>
<td>1</td>
<td>0</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

**NOT gate**

Abstraction!
Digital Logic Gates

Tiny electronic devices that compute basic Boolean functions.

**NOT**

\[
\begin{array}{c|c}
V_{\text{in}} & V_{\text{out}} \\
0 & 1 \\
1 & 0 \\
\end{array}
\]

**Abstraction!**

\[
\begin{array}{c|c}
V_1 & V_2 \\
0 & 1 \\
\end{array}
\]
Integrated Circuits (1950s - )

Early (first?) transistor

Small integrated circuit

Chip

Wafer
Five basic gates: define with truth tables

**NOT**

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

**NAND**

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

**NOR**

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

**AND**

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

**OR**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Boolean Algebra**

for combinational logic

\[ A \quad B \quad (A \cdot B) \]

AND = Boolean product

\[
\begin{array}{c|cc}
0 & 0 & 1 \\
\hline
0 & 0 & 0 \\
1 & 0 & 1 \\
\end{array}
\]

\[ A \quad \overline{A} \]

NOT = inverse or complement

\[
\begin{array}{c|c}
0 & 1 \\
1 & 0 \\
\end{array}
\]

\[ A \quad A \quad A + B \]

OR = Boolean sum

\[
\begin{array}{c|cc}
0 & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[ A \quad A \quad \text{wire = identity} \]

\[
\begin{array}{c|c}
0 & 0 \\
1 & 1 \\
\end{array}
\]

inputs = variables

wires = expressions

gates = operators/functions

circuits = functions
Circuits

Connect inputs and outputs of gates with wires. Crossed wires touch *only if* there is a dot.

What is the output if A=1, B=0, C=1?
What is the truth table of this circuit?
What is an equivalent Boolean expression?
Translation

Connect gates to implement these functions. Check with truth tables. Use a direct translation -- it is straightforward and bidirectional.

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

\[ Z = \overline{W} + (X + \overline{WY}) \]
Identity law, inverse law

Note on notation: bubble = inverse/complement
Commutativity, Associativity

\[ A + B = B + A \]

\[ (AB)C = A(BC) \]
Idempotent law, Null/Zero law

\[ A + A = A \]

\[ A \cdot A = A \]
DeMorgan's Law

(double bubble, toil and trouble, in Randy's words...)

Note on notation: bubble = inverse/complement

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \overline{A+B} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \overline{A} \cdot \overline{B} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( A + B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
One law, Absorption law

Write truth tables. Do they correspond to simpler circuits?

\[ A + 1 = 1 \]

\[ A + AB = AB \]
NAND is universal.

All Boolean functions can be implemented using only NANDs. Build NOT, AND, OR, NOR, using only NAND gates.
XOR: Exclusive OR

Output = 1 if exactly one input = 1.

Truth table:

Build from earlier gates:

Often used as a one-bit comparator.
Larger gates

Build a 4-input AND gate using any number of 2-input gates.
Circuit simplification

Is there a simpler circuit that performs the same function?

Start with an equivalent Boolean expression, then simplify with algebra.

\[ F(A, B, C) = \]

Check the answer with a truth table.
Circuit derivation: **code detectors**

AND gate + NOT gates = code detector, recognizes exactly one input code.

Design a 4-input code detector to output 1 if ABCD = 1001, and 0 otherwise.

A ________
B ________
C ________
D ________

Design a 4-input code detector to accept two codes (ABCD=1001, ABCD=1111) and reject all others. (accept = 1, reject = 0)
Circuit derivation: *sum-of-products* form

logical sum (OR)
of products (AND)
of inputs or their complements (NOT)

Draw the truth table and design a sum-of-products circuit for a 4-input code detector to accept two codes (ABCD=1001, ABCD=1111) and reject all others.

How are the truth table and the sum-of-products circuit related?
Voting machines

A majority circuit outputs 1 if and only if a majority of its inputs equal 1. Design a majority circuit for three inputs. Use a sum of products.

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

Triply redundant computers in spacecraft
- Space program also hastened Integrated Circuits.
Computers

- Manual calculations
- powered all early US space missions.
- Facilitated transition to digital computers.

**Katherine Johnson**
- Supported Mercury, Apollo, Space Shuttle, ...

**Dorothy Vaughn**
- First black supervisor within NACA
- Early self-taught FORTRAN programmer for NASA move to digital computers.
Early pioneers in reliable computing

Katherine Johnson
- Calculated first US human space flight trajectories
- Mercury, Apollo 11, Space Shuttle, ...
- Reputation for accuracy in manual calculations, verified early code
- Called to verify results of code for launch calculations for first US human in orbit
- Backup calculations helped save Apollo 13
- Presidential Medal of Freedom 2015

Margaret Hamilton
- Led software team for Apollo 11 Guidance Computer, averted mission abort on first moon landing.
- Coined “software engineering”, developed techniques for correctness and reliability.
- Presidential Medal of Freedom 2016