Lecture A3: Boolean Circuits

George Boole
(1815 – 1864)

Claude Shannon
(1916 – present)

Digital Circuits

What is a digital system?

Why digital systems?

Digital circuits and you.
- Computer microprocessors.
- Antilock brakes.
- VCR.
- Cell phone.

Digital Circuits

Digital: signals only assume discrete values (0, 1).
Analog: signals vary continuously.

Why digital systems?
- Accuracy and reliability.

Digital circuits and you.
- Computer microprocessors.
- Antilock brakes.
- VCR.
- Cell phone.

Logical Gates

Logical gates.
- A smallest useful circuit.

Lecture A4: build sequential circuits from gates.
- Memory.
- Flip-flop, register, counter.

Lecture A5: build general-purpose machine from circuits.

Logical Gates

Logical gates.
- A smallest useful circuit.

NOT

AND

OR
### Multiway AND Gates

\[ \text{AND}(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \]
- 1 if all inputs are 1.
- 0 otherwise.

![AND gate diagram](image1)

### Multiway OR Gates

\[ \text{OR}(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \]
- 1 if at least one input is 1.
- 0 otherwise.

![OR gate diagram](image2)

### Boolean Algebra

**History.**
- Developed by Boole to solve mathematical logic problems (1847).
- Shannon first applied to digital circuits (1939).

**Basics.**
- Boolean variable: value is 0 or 1.
- Boolean function: function whose inputs and outputs are Boolean variable.

**Relationship to circuits.**
- Boolean variables: signals.
- Boolean functions: circuits.

### Truth Table

**Truth table.**
- Systematic method to describe Boolean function.
- One row for each possible input combination.
- \( N \) inputs \( \Rightarrow 2^N \) rows.

#### AND Truth Table

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

![AND truth table](image3)
Truth Table

1. 16 Boolean functions of two variables.
2. \(2^N\) Boolean functions of N variables!

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>AND</th>
<th>NAND</th>
<th>OR</th>
<th>NOR</th>
<th>EQ</th>
<th>XOR</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Truth Table

Truth Table for Some Functions of 2 Variables

Universality of AND, OR, NOT

Any Boolean function can be expressed using AND, OR, NOT.
- "Universal."
- XOR\(x,y) = xy' + x'y\)

Expressing XOR Using AND, OR, NOT

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x'</th>
<th>y'</th>
<th>x'y</th>
<th>x'y + xy'</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Exercise: \{AND, NOT\}, \{OR, NOT\}, \{NAND\} are universal.

Sum-of-Products

Any Boolean function can be expressed using AND, OR, NOT.
- Sum-of-products is systematic procedure.
  - form AND term for each 1 in Boolean function
  - OR terms together

Expressing MAJ Using Sum-of-Products

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>MAJ</th>
<th>x'yz</th>
<th>xy'z</th>
<th>xyz</th>
<th>x'yz + xy'z + xyz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<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>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>0</td>
<td>0</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>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</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>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Translate Boolean Formula to Boolean Circuit

Use sum-of-products form.
- XOR(x, y) = x'y + xy'.

![XOR Circuit Diagram]

Translate Boolean Formula to Boolean Circuit

Use sum-of-products form.
- MAJ(x, y, z) = x'yz + xy'z + xyz' + xyz.

![MAJ Circuit Diagram]

Simplification Using Boolean Algebra

Many possible circuits for each Boolean function.
- Sum-of-products not optimal.
- MAJ(x, y, z) = x'yz + xy'z + xyz' + xyz = xy + yz + xz.

![Simplification Diagram]

Recipe for Making Combinational Circuit

Step 1.
- Represent input and output signals with Boolean variables.

Step 2.
- Construct truth table to carry out computation.

Step 3.
- Derive (simplified) Boolean expression using sum-of-products.

Step 4.
- Transform Boolean expression into circuit.
Let’s Make an Adder Circuit

Goal: \( x + y = z \).

Step 1.
1. Represent input and output in binary.
2. We build 4-bit adder: 8 inputs, 4 outputs.

\[
\begin{array}{cccc}
1 & 1 & 1 & \\
2 & 4 & 8 & 7 \\
+ & 3 & 5 & 7 & 9 \\
6 & 1 & 6 & 6 \\
\end{array}
\]

Step 2.
1. Build truth table.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
<th>( c )</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>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>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>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>0</td>
</tr>
</tbody>
</table>

\( c_3 \)  \( c_2 \)  \( c_1 \)  \( c_0 \) = 0

\[
x_3 \ x_2 \ x_1 \ x_0 \\
y_3 \ y_2 \ y_1 \ y_0 \\
z_3 \ z_2 \ z_1 \ z_0 \\
\]

2\(^8\) = 256 rows!

Step 3.
1. Derive (simplified) Boolean expression.

Carry Bit

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Summand Bit

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Carry Bit

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Summand Bit

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

ODD

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Let’s Make an Adder Circuit

Goal: \( x + y = z \).

Step 4.
- Transform Boolean expression into circuit.

Lecture A1: ExtraSlides

ODD Parity Circuit

ODD(x, y, z).
- 1 if odd number of inputs are 1.
- 0 otherwise.

Expressing ODD Using Sum-of-Products

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>ODD</th>
<th>x’y’z</th>
<th>x’yz’</th>
<th>xy’z</th>
<th>xyz</th>
<th>x’y’z + x’yz’ + xy’z’ + xyz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**ODD Parity Circuit**

ODD(x, y, z).
- 1 if odd number of inputs are 1.
- 0 otherwise.

**2-Bit Multiplexer Circuit**

MUX(x_0, x_1, x_2, x_3, s_0, s_1).
- x_0 if s_1 = 0, s_0 = 0.
- x_1 if s_1 = 0, s_0 = 1.
- x_2 if s_1 = 1, s_0 = 0.
- x_3 if s_1 = 1, s_0 = 1.

**Decoder**

N-bit decoder.
- N inputs.
- 2^N outputs.
- Exactly one of outputs is 1; rest are 0.
- Which one?

**3-Bit Decoder**

DECODE(x, y, z).

<table>
<thead>
<tr>
<th>I_2</th>
<th>I_1</th>
<th>I_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

11_2 = 3_{10}

**3-Bit Decoder**

<table>
<thead>
<tr>
<th>Q_7</th>
<th>Q_6</th>
<th>Q_5</th>
<th>Q_4</th>
<th>Q_3</th>
<th>Q_2</th>
<th>Q_1</th>
<th>Q_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3-8 Decoder
Cheat Sheet

- **NOT**
  - $x \rightarrow x'$

- **OR**
  - $x \lor y$

- **AND**
  - $x \land y$

- **4-bit Adder**

- **4-1 Multiplexer**

- **3-8 Decoder**

- **Adder**