What We Will Learn?

- In Lab 1, you will design simple combinatorial circuits.

- We will cover a tutorial about:
  - Boolean Equations
    - Logic operations with binary numbers
  - Logic Gates
    - Basic blocks that are interconnected to form larger units that are needed to construct a computer
Boolean Equations and Logic Gates
**Simple Equations: NOT / AND / OR**

\[ \bar{A} \text{ (reads “not } A\text{”) is 1 iff } A \text{ is 0} \]

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

\[ A \cdot B \text{ (reads “A and B”) is 1 iff } A \text{ and } B \text{ are both 1} \]

\[
\begin{array}{ccc}
A & B & A \cdot B \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[ A + B \text{ (reads “A or B”) is 1 iff either } A \text{ or } B \text{ is 1} \]

\[
\begin{array}{ccc}
A & B & A + B \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]
Boolean Algebra: Big Picture

- An algebra on 1’s and 0’s
  - with AND, OR, NOT operations

- What you start with
  - **Axioms:** basic stuff about objects and operations you just assume to be true at the start

- What you derive first
  - **Laws and theorems:** allow you to manipulate Boolean expressions
  - ...also allow us to do some simplification on Boolean expressions

- What you derive later
  - More “sophisticated” properties useful for manipulating digital designs represented in the form of Boolean equations
### Common Logic Gates

<table>
<thead>
<tr>
<th></th>
<th>Buffer</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Buffer Diagram" /></td>
<td><img src="image" alt="AND Diagram" /></td>
<td><img src="image" alt="OR Diagram" /></td>
<td><img src="image" alt="XOR Diagram" /></td>
<td></td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td><strong>Output</strong></td>
<td><strong>Input</strong></td>
<td><strong>Output</strong></td>
<td><strong>Input</strong></td>
</tr>
<tr>
<td>A</td>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>Z</td>
</tr>
<tr>
<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>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Inverter

<table>
<thead>
<tr>
<th><img src="image" alt="Inverter Diagram" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
### Boolean Algebra: Axioms

<table>
<thead>
<tr>
<th><strong>Formal version</strong></th>
<th><strong>English version</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.</strong> $B$ contains at least two elements, 0 and 1, such that $0 \neq 1$</td>
<td>Math formality...</td>
</tr>
</tbody>
</table>
| **2.** Closure $a, b \in B$,  
(i) $a + b \in B$  
(ii) $a \cdot b \in B$ | Result of AND, OR stays in set you start with |
| **3.** Commutative Laws: $a, b \in B$,  
(i)  
(ii) | For primitive AND, OR of 2 inputs, order doesn’t matter |
| **4.** Identities: $0, 1 \in B$  
(i)  
(ii) | There are identity elements for AND, OR, give you back what you started with |
| **5.** Distributive Laws:  
(i)  
(ii) | • distributes over $+$, just like algebra  
...but $+$ distributes over $\cdot$, also (!!) |
| **6.** Complement:  
(i)  
(ii) | There is a complement element, ANDing, ORing give you an identity |
Boolean Algebra: Duality

- Interesting observation
  - All the axioms come in “dual” form
  - Anything true for an expression also true for its dual
  - So any derivation you could make that is true, can be flipped into dual form, and it stays true

- Duality -- More formally
  - A dual of a Boolean expression is derived by replacing
    - Every AND operation with... an OR operation
    - Every OR operation with... an AND
    - Every constant 1 with... a constant 0
    - Every constant 0 with... a constant 1
    - But don’t change any of the literals or play with the complements!

Example

\[ a \cdot (b + c) = (a \cdot b) + (a \cdot c) \]
\[ \rightarrow a + (b \cdot c) = (a + b) \cdot (a + c) \]
Boolean Algebra: Useful Laws

Operations with 0 and 1:

1. \( X + 0 = X \)
2. \( X + 1 = 1 \)
3. \( X + X = X \)
4. \( X + \overline{X} = 1 \)
5. \( X + Y = Y + X \)

And, OR with identities:
- gives you back the original variable or the identity

Idempotent Law:

1D. \( X \cdot 1 = X \)
2D. \( X \cdot 0 = 0 \)

Involution Law:

3D. \( X \cdot X = X \)

Laws of Complementarity:

5D. \( X \cdot \overline{X} = 0 \)

Commutative Law:

6D. \( X \cdot Y = Y \cdot X \)

Dual

AND, OR with identities:
- gives you back the original variable or the identity

AND, OR with self = self

Double complement = no complement

AND, OR with complement:
- gives you an identity

Just an axiom…
Useful Laws (cont)

**Associative Laws:**
7. \((X + Y) + Z = X + (Y + Z)\)
   \[= X + Y + Z\]
7D. \((X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)\)
   \[= X \cdot Y \cdot Z\]
   Parenthesis order doesn’t matter

**Distributive Laws:**
8. \(X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)\)
8D. \(X + (Y \cdot Z) = (X + Y) \cdot (X + Z)\)
   Axiom

**Simplification Theorems:**
9. \(X \cdot Y + X \cdot \overline{Y} = X\)
9D. \((X + Y) \cdot (X + \overline{Y}) = X\)
   Useful for simplifying expressions
10. \(X + X \cdot Y = X\)
10D. \(X \cdot (X + Y) = X\)
11. \((X + \overline{Y}) \cdot Y = X \cdot Y\)
11D. \((X \cdot \overline{Y}) + Y = X + Y\)
   Actually worth remembering — they show up a lot in real designs…
DeMorgan’s Law

DeMorgan's Law:

12. \((X + Y + Z + \cdots) = \overline{X} \cdot \overline{Y} \cdot \overline{Z} \cdots\)

12D. \((X \cdot Y \cdot Z \cdots) = \overline{X} + \overline{Y} + \overline{Z} + \cdots\)

Think of this as a transformation

- Let’s say we have:

\[ F = A + B + C \]

- Applying DeMorgan’s Law (12), gives us:

\[ F = \overline{(A + B + C)} = \overline{(A \cdot B \cdot C)} \]
DeMorgan’s Law (cont.)

Interesting — these are conversions between different types of logic
That’s useful given you don’t always have every type of gate

\[ A = (X + Y) = \overline{XY} \]

NOR is equivalent to AND with inputs complemented

\[ B = (XY) = \overline{X} + \overline{Y} \]

NAND is equivalent to OR with inputs complemented
Part 1: A Comparator Circuit

- Design a comparator that receives two 4-bit numbers A and B, and sets the output bit EQ to logic-1 if A and B are equal.

- **Hints:**
  - First compare A and B bit by bit.
  - Then combine the results of the previous steps to set EQ to logic-1 if all A and B are equal.
Part 2: A More General Comparator

- Design a circuit that receives two 1-bit inputs A and B, and:
  - sets its first output (O1) to 1 if $A > B$,
  - sets the second output (O2) to 1 if $A = B$,
  - sets the third output (O3) to 1 if $A < B$. 

![Comparator 2 Diagram]
Part 3: Circuits with Only NAND or NOR Gates

- Design the circuit of Part 2 using **only NAND or NOR gates**.

- **Logical Completeness:**
  - The set of gates \{AND, OR, NOT\} is **logically complete** because we can build a circuit to carry out the specification of any combinatorial logic we wish, without any other kind of gate.
  - NAND and NOR are also logically complete.
Last Words

- In this lab, you will draw the schematics of some simple operations.

- Part 1: A comparator circuit

- Part 2: A more general comparator circuit

- Part 3: Designing circuits using only NAND or NOR gates

- You will find more exercises in the lab report.