Dealing with digital data: **Booleans**

Reference: Baran, Integrated Circuits ..., Ch. 3.

Boolean variable: has only two possible values, called 0, 1 or False, True. We represent these variables by letters, as in ordinary algebra: A, B, Z, ...

---

Operations with Boolean variables:

1. **Negation** (complementation): \( \overline{A} \)

   \( \overline{A} \) is 1 when \( A \) is 0 and 0 when \( A \) is 1.

   We can represent this with a "truth table":

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

   Truth table specifies the value of the logical function \( \overline{A} \) for all values of its variables (\( A \)).

2. **AND**: \( A \cdot B \)

   (requires both \( A \) and \( B \) to be 1)

   (dot does not denote multiplication)

   \[
   \begin{array}{c|c|c}
   A & B & A \cdot B \\
   \hline
   0 & 0 & 0 \\
   0 & 1 & 0 \\
   1 & 0 & 0 \\
   1 & 1 & 1 \\
   \end{array}
   \]

3. **OR**: \( A + B \)

   (1 if either \( A \) or \( B \) or both 1)

   (plus does not denote addition)

   \[
   \begin{array}{c|c|c}
   A & B & A + B \\
   \hline
   0 & 0 & 0 \\
   0 & 1 & 1 \\
   1 & 0 & 1 \\
   1 & 1 & 1 \\
   \end{array}
   \]

   (electronic symbols shown in boxes)
4. A Boolean expression, then, will look like \((A \cdot B) + C\), \((A \cdot B) + (D \cdot \overline{B})\) or some such. \(AB\) is the same as \(A \cdot B\). Often it is assumed that \(\cdot\) takes precedence over + so that \(A \cdot B + C\) means \((A \cdot B) + C\) (instead of \(A \cdot (B + C)\)).

5. Equality of two expressions means they have identical values for identical values of the variables making them up; in short, the expressions have identical truth tables if and only if they are equal.

Example: Prove that \(\overline{A + B} = (A \cdot B)\).

Demonstrate the truth tables:

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

Identical, hence, \(\overline{A + B} = (A \cdot B)\).

6. The example above demonstrates the complementary nature of \(\cdot\) relative to +. Expressions involving + are easily transformed into ones involving \(\cdot\) and vice-versa through suitable complementation:

\[\text{if } C = A + B \text{, then } \overline{C} = \overline{A \cdot B};\]
\[\text{if } D = A \cdot B \text{, then } \overline{D} = \overline{A + B}.\]

This result is known as De Morgan's theorem and is useful in simplifying Boolean expressions.
Summary of properties of Boolean algebra (following Barna)

Basic postulates are given in pairs of related (dual) equations.

P1 a) \( 0 \cdot 0 = 0 \)  
    b) \( 1 + 1 = 1 \)  

P2 a) \( 0 \cdot 1 = 0 \)  
    b) \( 1 + 0 = 1 \)  

P3 a) \( 1 \cdot 1 = 1 \)  
    b) \( 0 + 0 = 0 \)

P4 a) \( \overline{0} = 1 \)  
    b) \( \overline{1} = 0 \)

P5 a) \( x = 1 \) if \( x \neq 0 \)  
    b) \( x = 0 \) if \( x \neq 1 \)

Duel! For a valid relation, exchange \( 0 \leftrightarrow 1 \) to get another valid relation.

Theorems (provable from postulates)

T1 a) \( x \cdot y = y \cdot x \)  
    b) \( x + y = y + x \)

T2 a) \( x \cdot (y + z) = (x \cdot y) + (x \cdot z) \)  
    b) \( x + (y + z) = (x + y) + z \)

T3 a) \( x \cdot (y + z) = (x \cdot y) + (x \cdot z) \)  
    b) \( x + (y \cdot z) = (x + y) \cdot (x + z) \)

T4 a) \( x \cdot 1 = x \)  
    b) \( x + 0 = x \)

T5 a) \( x \cdot 0 = 0 \)  
    b) \( x + 1 = 1 \)

T6 a) \( x \cdot \overline{x} = 0 \)  
    b) \( x + \overline{x} = 1 \)

T7 a) \( x \cdot x = x \)  
    b) \( x + x = x \)

T8 a) \( x + (x \cdot y) = x \)  
    b) \( x \cdot (x + y) = x \)

T9 a) \( \overline{x} = x \)

T10 a) \( \overline{x \cdot y \cdot \ldots \cdot z} = \overline{x} + \overline{y} + \ldots + \overline{z} \)  
    b) \( \overline{x + y + \ldots + z} = \overline{x} \cdot \overline{y} \cdot \ldots \cdot \overline{z} \)

Theorems (provable from postulates)

\[ \begin{align*} 
T1 & \quad x \cdot y = y \cdot x \\
T2 & \quad x + (y + z) = (x + y) + z \\
T3 & \quad x \cdot (y + z) = (x \cdot y) + (x \cdot z) \\
T4 & \quad x \cdot 1 = x
\end{align*} \]
Typical proof using theorems:

\[
\overline{A \cdot (B \cdot C)} \equiv \overline{A} \cdot (\overline{B} + \overline{C})
\]  \hspace{1cm} (1)

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

Truth table approach:

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

A consequence of De Morgan's Theorem is the following:

To obtain the complement, \( \overline{f(A, B, C, \ldots)} \),

of some logical function, \( f(A, B, C, \ldots) \), change all "+" operations to "·", all "·" to "+" and complement each variable.

Thus we can deduce the correctness of expression (1) above directly:

\[
\overline{A + (B \cdot C)} = \overline{A} \cdot (\overline{B} + \overline{C})
\]
Sets and Venn diagrams

Set: well-defined collection of objects
e.g., the integers \{1, 2, 3, \ldots, 100\} 
or the possible states of 3 logical variables
\{\overline{ABC}, \overline{AB}, \overline{AB}, \overline{ABC}, \overline{ABC}, ABC, ABC, ABC, ABC\}

A set consisting of some members of another set is a subset of that set. The original set is called the universal set, \(U\). A set containing no objects is called the empty set, \(\varnothing\).

We can define useful operations on subsets and represent them by diagrams (Venn diagrams). Here, the universal set is shown as a rectangle and the subsets as circles. The fact that some elements may belong to both subsets means that the circles may overlap in general.

**Complement**, \(P\): is the set of elements not included in \(P\), \((U\) is the entire rectangle including \(P\))

**Intersection**, \(P \cap Q\): set of elements in common for subsets \(P, Q\)

**Union**, \(P \cup Q\): set of elements belonging to \(P, Q\) or both.

So \(P \cup \varnothing = U\), \(P \cap \varnothing = \varnothing\),

\[P \cup Q = (P \cap \overline{Q}) \cup (\overline{P} \cap Q) \cup (P \cap Q),\] etc.
Relation to logical variables

Let $\mathcal{U}$ be the set of all logical possibilities for logical variables $A, B, \ldots$. Let the symbol $A$, also represent the subset of $\mathcal{U}$ for which $A$ is true, $B$, the subset for which $B$ is true, etc. We can now translate a problem concerning logical variables into one about sets and visualise it with the Venn diagram.

Example: the subset for which $A$ or $B$ is true, corresponding to $A \lor B$, is $A \cup B$.

De Morgan's Theorem: $\overline{A \land B} = \overline{A} \lor \overline{B}$.
Question: How many distinct logical functions are there for 2 logical variables?

Answer: We can find the answer by enumerating all the possibilities: (4 rows in truth table, each can contain 0 or 1 so
\[ 2^4 = 16 \text{ possibilities} \]

| A B | f₀ f₁ f₂ f₃ f₄ f₅ f₆ f₇ f₈ f₉ f₁₀ f₁₁ f₁₂ f₁₃ f₁₄ f₁₅ |
|-----|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0 0 | 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 | 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 | 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 1 | 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1 1 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 1 0 | 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 | 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 | 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 | 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 | 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 |
| 1 1 | 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 | 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 | 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 | 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 | 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 |

<table>
<thead>
<tr>
<th>NAME</th>
<th>SYMBOL</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOR</td>
<td>N</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>AND</td>
<td>AND</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>IMPL</td>
<td>IMPL</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>EQ</td>
<td>EQ</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>IMP</td>
<td>IMP</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>EXOR</td>
<td>EXOR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>EOR</td>
<td>EOR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>XNOR</td>
<td>XNOR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>AND</td>
<td>AND</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>IMPL</td>
<td>IMPL</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>EQ</td>
<td>EQ</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>IMP</td>
<td>IMP</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>EXOR</td>
<td>EXOR</td>
<td>INHIBIT</td>
</tr>
<tr>
<td>EOR</td>
<td>EOR</td>
<td>INHIBIT</td>
</tr>
</tbody>
</table>

\[ f₆ = \text{exclusive or (⊕)} \]
\[ f₇ = \text{or (+) } \]
\[ f₈ = \text{nor} \]
\[ f₁ = \text{and (・)} \]
\[ f₉ = \text{equality (≡, 0)} \]
\[ f₁₀ = A \Rightarrow B \text{ (impliesation)} \]
\[ f₁₁ = B \Rightarrow A \text{ (implication)} \]

What about "implication"? (A \Rightarrow B)

Here, this means if A is true, B must be true, but if A is false, no statement is made about B.

There is only contradiction (implication function false) if B is not true when A is true.

Lawn sprinkler example: if the sprinklers are on then the lawn is getting wet. A \Rightarrow B. But the lawn could also be getting wet if the sprinklers are off but it is raining! (\text{A \& B})
If the sprinklers are on and the lawn is dry, there is a problem! (\text{A \& B})
small-and medium-scale integrated circuit logic elements and combinatorial logic implementation

In the lab this week, we will experiment with TIL IC’s. The first page of the data sheet of a 7400 quad 2-input NAND is provided on the next page. If we take “T” or “1” to be the “high” logic level (2.0-5V) and “F” or “0” to be “low” (0-0.8V), the truth table for each of the four logic gates on the 7400 is:

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

This choice of logic levels is called “positive logic” and we see that the function is \( \overline{AB} \), called “NAND” (for “not and”). The symbol is an AND with a circle at the output indicating “not” (complementation):

\[
\begin{array}{c}
A \\
\hline
B \quad \overline{\circ} \\
\hline
Y
\end{array}
\]

Note that the function table on the handout is independent of our choice of positive logic, giving the response in terms of H (high) and L (low). Also note the use of “X” meaning “don’t care.” If an input is marked “X,” the output will be the same if that input is H or L. For example, if A is low, the output will be high irrespective of the input on B.

*see note at bottom of p.10 about this terminology.
Package Options Include Plastic "Small Outline" Packages, Ceramic Chip Carriers and Flat Packages, and Plastic and Ceramic DIPs

Dependable Texas Instruments Quality and Reliability

description

These devices contain four independent 2-input NAND gates.

The SN5400, SN54LS00, and SN54S00 are characterized for operation over the full military temperature range of -55°C to 125°C. The SN7400, SN74LS00, and SN74S00 are characterized for operation from 0°C to 70°C.

FUNCTION TABLE (each gate)

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>L</td>
</tr>
</tbody>
</table>

logic symbol†

†This symbol is in accordance with ANSI/IEEE Std. 91-1984 and IEC Publication 517-12.
Pin numbers shown are for D, J, and N packages.

logic diagram (positive logic)

Copyright © 1986, Texas Instruments Incorporated
The ANSI logic symbols allow some freedom from the choice of positive logic. But we will stick with positive logic in this discussion for simplicity.

Note that De Morgan tells us $\overline{A \cdot B} = \overline{A} + \overline{B}$

so we could also draw the NAND symbol as

$$\begin{align*}
A & \quad - \quad Y = A \\
B & \quad - \quad \overline{A \cdot B} = \overline{A} + \overline{B}
\end{align*}$$

In fact, the NAND is an example of a "universal" gate, in the sense that any function could be implemented using NANDs alone. Examples:

$$\begin{align*}
A & \quad - \quad Y = \overline{A} \\
B & \quad - \quad Y = A + B
\end{align*}$$

or to redraw more obviously,

$$\begin{align*}
A & \quad - \quad \overline{A} \\
B & \quad - \quad Y = A + B
\end{align*}$$

Other common SSI gates:

$$\begin{align*}
A & \quad \rightarrow \quad Y \\
B & \quad \rightarrow \quad Y
\end{align*}$$

$$\begin{align*}
A + B & \quad \rightarrow \quad \overline{A \cdot B} \\
\text{NOR} & \quad \rightarrow \quad \overline{A}
\end{align*}$$

$$\begin{align*}
A & \quad \rightarrow \quad Y \\
B & \quad \rightarrow \quad Y
\end{align*}$$

Exclusive or (XOR)

$$\begin{align*}
A \oplus B & = \overline{A} \cdot B + A \cdot \overline{B}
\end{align*}$$

Why call them "gates"?

An "AND" can control a pulse train on input $A$ by a logic level on input $B$. When $B$ is true, the pulses get through but when $B$ is false, the output is low. This is like a gate in a road which is open when $B$ is true but closed when $B$ is false:

"gate open"  "gate closed"
Examples of implementing logical functions

1. Exclusive or: \( A \oplus B = \overline{A} \overline{B} + \overline{A}B \)

\[
\begin{array}{c}
A \\
\overline{A} \overline{B} \\
B \\
\overline{A}B
\end{array}
\]

\[ f = \overline{A} \overline{B} + \overline{A}B = A \oplus B \]

This is an example of two-stage logic. The first stage forms the needed "and" terms. These feed into an "or" stage to implement the function. Inverters are used at the input to the first stage as necessary. The property of the NAND is such that it can be used for both stages, as shown above.

2. \( f = (A+C)\overline{B} + \overline{A}B \)

We could implement this as

\[
\begin{array}{c}
A \\
\overline{A+C} \overline{(A+C)B} \\
C \\
B \\
\overline{A} \overline{B}
\end{array}
\]

Or we could express

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

and implement with two stage logic using NANDS and inverters.

\[
\begin{array}{c}
A \\
\overline{A} \overline{B} \\
\overline{B} \\
\overline{C}
\end{array}
\]

3-input NAND
Systematic simplification of logic functions

Minterms

We can represent a logical function in a standard lowest order "sum of products" form. These lower-order products are called "minterms" and are given numbers according to the pattern of uncomplemented variables (1) and complemented variables (0) making up the "product." For example, in the function, \( f = (A+C)\overline{B} + \overline{A}B \), can be written as

\[
\frac{f = \overline{A}BC + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + A\overline{B}C}{\text{standard lower-order "sum of products" form}}
\]

\( \overline{A}\overline{B}C = m_1 \) (001), \( A\overline{B}C = m_4 \) (100), etc.

Thus, \( f = m_1 + m_2 + m_3 + m_4 + m_8 \).

Note that the minterms correspond to the locations of the 1's in the truth table for the function (i.e., the binary number of the row in the truth table):

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( f )</th>
<th>row#</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>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

This representation lends itself directly to implementation using two-stage logic with ands and one big or.

It can also be used in conjunction with a computer minimization algorithm to simplify the resulting circuit. It is also a useful concept in constructing Karnaugh maps (next lecture). The function can also be written as a product of sums (maxterms — see text).