Introduction to Logic Gates

- Using transistor technology, we can create basic logic gates that perform boolean operations on high (5V) and low (0V) signals.
- Example: NAND gate
Intro to Logic Gates

- Digital logic gates form the basis for all implemented computing machines.

- Basic gates:
  - AND: *, ^
  - OR: +, ∨
  - NOT: ¬, ~
  - NAND
  - NOR
  - XOR: ⊕
  - XNOR

- Relationships between inputs and outputs are outlined in truth tables.
Truth Tables

• **AND gate**

  \[ C = A \cdot B \]

  ![AND gate diagram with truth table]

• **OR gate**

  \[ C = A + B \]

  ![OR gate diagram with truth table]
Truth Tables (cont’d)

- **Buffer**

  ![Buffer Diagram]

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

- **NOT Gate**

  ![NOT Gate Diagram]

  $$B = \overline{A}$$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Tables (cont’d)

- NAND Gate
  
  \[
  \begin{array}{ccc}
  A & B & C \\
  0 & 0 & 1 \\
  1 & 0 & 1 \\
  0 & 1 & 1 \\
  1 & 1 & 0 \\
  \end{array}
  \]

- NOR Gate
  
  \[
  \begin{array}{ccc}
  A & B & C \\
  0 & 0 & 1 \\
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 1 & 0 \\
  \end{array}
  \]
Truth Tables (cont’d)

• XOR Gate

\[ C = A \oplus B \]

• XNOR Gate
Logic Operations

- Logic gates all follow the same rules as logic operators in programming languages.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>$1 \cdot x = x$</td>
</tr>
<tr>
<td></td>
<td>$0 \cdot x = 0$</td>
</tr>
<tr>
<td>OR</td>
<td>$1 + x = 1$</td>
</tr>
<tr>
<td></td>
<td>$0 + x = x$</td>
</tr>
<tr>
<td>NOT</td>
<td>$x + \overline{x} = 1$</td>
</tr>
<tr>
<td></td>
<td>$x \cdot \overline{x} = 0$</td>
</tr>
<tr>
<td>XOR</td>
<td>$x \oplus y = y \oplus x$</td>
</tr>
</tbody>
</table>

- Order of operations also applies to these operations, when putting together the digital circuit.
Designing a Circuit

• **Task #1:** Given a logic expression, determine the equivalent gate representation.
  
  – Example: \( f = x_1 \cdot x_2 + x_1 \cdot \overline{x_2} \)

• Group terms according to **order of operations**:
  – NOT terms first
  – AND terms next
  – OR terms last

• The example gets rewritten as:
  
  \[ f = ((\overline{x_1}) \cdot x_2) + (x_1 \cdot (\overline{x_2})) \]
Circuit Example

- By evaluating the terms of the expression from the inside-out, we construct the diagram above.
- Of course, this can also be represented as a single gate, but the underlying complexity is the same.
More Circuit Design

- **Task #2:** Given a truth table that specifies a logic circuit’s behaviour, design the equivalent circuit.
  - Example: three-input circuit

- **Sum-of-product technique:**
  - Group all rows with an output of $f=1$ into a single AND term (product)
  - Combine these AND terms with a single OR gate (sum)

- **Note:** All truth tables can be converted into gate form by using this technique.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</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>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>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>0</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>
Sum-of-Products

- A **minterm** is a product term where each of the input variables occurs exactly once.
  - The sum-of-products technique is also referred to as a disjunction of minterms
- Grouping together the minterms for all rows where \( f=1 \) yields the following expression:

\[
f = \overline{x_1}x_2x_3 + \overline{x_1}x_2x_3 + \overline{x_1}x_2x_3 + x_1x_2\overline{x_3}
\]
Reducing Minterms

- This is an ugly expression. Can we find some way to **minimize** the expression, to make it more compact?
- We can, by employing a set of binary logic rules:

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Algebraic Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commutative</td>
<td>$x + y = y + x$</td>
</tr>
<tr>
<td>Associative</td>
<td>$(x + y) + z = x + (y + z)$</td>
</tr>
<tr>
<td>Distributive</td>
<td>$x + yz = (x + y)(x + z)$</td>
</tr>
<tr>
<td>Idempotent</td>
<td>$x + x = x$</td>
</tr>
<tr>
<td>Involution</td>
<td>$\overline{x} = x$</td>
</tr>
<tr>
<td>Complement</td>
<td>$x + \overline{x} = 1$</td>
</tr>
<tr>
<td>de Morgan</td>
<td>$\overline{x + y} = \overline{x} \cdot \overline{y}$</td>
</tr>
</tbody>
</table>

$$f = \overline{x_1}x_2x_3 + \overline{x_1}x_2\overline{x_3} + \overline{x_1}x_2x_3 + x_1x_2\overline{x_3}$$
Simplification Example

• Reduce the following sum of products:
  \[ f = \overline{x_1}x_2x_3 + \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 \]

• Steps:
  - Distributive: \[ f = \overline{x_1}x_3(\overline{x_2} + x_2) + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 \]
  - Complement: \[ f = \overline{x_1}x_3 \cdot 1 + x_1x_2x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 \]
  - Identity: \[ f = \overline{x_1}x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 \]
  - Idempotent: \[ f = \overline{x_1}x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + (x_1x_2x_3 + x_1x_2x_3) \]
  - Associative: \[ f = \overline{x_1}x_3 + (x_1\overline{x_2}x_3 + x_1x_2\overline{x_3}) + (x_1x_2\overline{x_3} + x_1x_2x_3) \]
  - Distributive: \[ f = \overline{x_1}x_3 + x_1x_3(\overline{x_2} + x_2) + x_1x_2(\overline{x_3} + x_3) \]
  - Complement: \[ f = \overline{x_1}x_3 + x_1x_3 \cdot 1 + x_1x_2 \cdot 1 \]
  - Identity: \[ f = \overline{x_1}x_3 + x_1x_3 + x_1x_2 \]
Simplification Example

• Steps (cont’d):
  – (from previous slide): \( f = \overline{x_1}x_3 + x_1x_3 + x_1x_2 \)
  – Distributive: \( f = x_3(\overline{x_1} + x_1) + x_1x_2 \)
  – Complement: \( f = x_3 \cdot 1 + x_1x_2 \)
  – Identity: \( f = x_3 + x_1x_2 \)

• Solution is:

\[ f = x_3 + x_1x_2 \]
More de Morgan

• Implications of de Morgan’s Law:

\[ \overline{x + y} = \overline{x} \cdot \overline{y} \]

\[ \overline{x \cdot y} = \overline{x} + \overline{y} \]
Karnaugh Maps

- **A Karnaugh map** is another way of representing a circuit’s truth table
  - Presented as a two-dimensional grid of $2^n$ squares.
  - Each axis represents the different input values to the circuit, and the contents of each square represents the output value for the intersecting input values.
    - Horizontally and vertically adjacent squares only differ by a single input variable
    - Number of rows and columns must be a power of 2
    - **Note:** row and column labels must also differ by a single digit.
  - Simplified circuit is found by circling groups of adjacent 1’s on the grid.
  - **Result:** The minimal expression for the circuit.
Karnaugh Maps

- Example:

\[ f = \overline{x_1}x_2x_3 + \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3 \]
Karnaugh Maps

- Next step: circle blocks of 1’s
  - Cannot contain any 0’s in the block
  - Height and width of block must be a power of 2
  - Blocks are allowed to overlap

- Circled blocks correspond to $x_3$ and $x_1x_2$
Karnaugh Example

- **Task:** Given the truth table on the right, determine the simplest equivalent gate arrangement.

\[
X = \overline{AC} + D + \overline{ABC}
\]
“Don’t Care” Conditions

- Sometimes certain outputs are not defined
  - Example: A three-input circuit where the output is 1 if any of the inputs is 1, and 0 if all of the inputs are 0.
  - The output can be anything in the case where two or more inputs are 1, since we don’t consider those cases.
  - In those cases, label the output as “don’t care” (or “X”).

- “Don’t care” conditions are useful because they can be treated as either 1 or 0, depending on which makes things easier.
  - In real life, you usually have to set it to some sensible value.
“Don’t Care” Example

- Adding milk to beverages:
  - If a patron orders coffee (C), add milk. If the patron orders tea (T), only add milk if lemon (L) hasn’t already been added.
  - The “don’t care” conditions are sometimes written as “d” (as in the textbook, for example). In industry though, “X” is more commonly used, as in the phrase “X-propagation”.
The Importance of NAND

- NAND gates are considered to be the “universal” gate, because any other gate can be synthesized using NAND.
- In fact, most gates are implemented in solid-state TTL chips (Transistor-Transistor Logic)
  - e.g. 74LS00 integrated circuit (IC)
Getting to NAND

• How do we create a NAND-based circuit, if our approach so far has been to find sum-of-products?
  – **Answer:** By using de Morgan’s rule!

• According to de Morgan (see slide 15),

\[
\begin{align*}
\text{equivalent to} \\
\end{align*}
\]

• Therefore, the following are equivalent as well:

\[
\begin{align*}
\text{equivalent to} \\
\end{align*}
\]
Neither...NOR

- NAND and NOR gates are the cheapest to make, and the most commonly found in IC chips.
- Typical process for creating a NAND-based circuit is:
  1. represent truth table in Karnaugh map
  2. isolate smallest terms that produce high output
  3. produce sum-of-product model with AND & OR gates
  4. convert model to NAND representation
- What about creating an equivalent circuit out of NOR gates? Is that possible?
  - Requires obtaining maxterms of the truth table, and producing a product-of-sums model.
Maxterms

- A **maxterm** is a sum of input values where every input value occurs exactly once.
- When expressing a truth table, each maxterm represents a row where the output is 0, such that the maxterm will give a true value in all cases except for that row.

  - Maxterms for example table:
    - \((x_1 + x_2 + x_3)\)
    - \((\bar{x}_1 + x_2 + x_3)\)
    - \((\bar{x}_1 + x_2 + \bar{x}_3)\)
    - \((\bar{x}_1 + \bar{x}_2 + \bar{x}_3)\)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>f</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>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>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>0</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>
POS (Product of Sums)

• Instead of creating a disjunction of cases where the output of the circuit is 1, the product-of-sums technique creates a conjunction of the cases where the output is 0.
  - From previous example:
    
    \[(x_1 + x_2 + x_3) \cdot (\bar{x}_1 + x_2 + x_3) \cdot (\bar{x}_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + \bar{x}_3)\]

• These equations can be reduced using the same techniques used on minterms.
  - boolean logic rules
  - Karnaugh maps
NOR Circuit

- A product-of-sum circuit will result in several OR gates, united by a single AND gate.
- Through de Morgan's rule, this converts to a circuit made up entirely of NOR gates.
- Design decision: of the maxterm and minterm representations, which has the fewest reduced terms?
Other TTL Logic

- In addition to NAND circuits, inverter circuits are commonly used as well (e.g. 74LS04).

  ![Inverter Circuit Diagram]

- When using these chips, make sure you have the pins and the orientation correct. For example, applying the voltage source to the ground pin can have some very unpleasant results.