ELEC 1202 Digital Systems and Microprocessors

Minimisation and Implementation of Logic Circuits

Dr. JingHuey Khor
Given a binary system:

**AND**

\[
\begin{align*}
A \cdot 0 &= 0 \\
A \cdot 1 &= A \\
A \cdot \overline{A} &= 0 \\
A \cdot B &= B \cdot A \\
(A \cdot B) \cdot C &= A \cdot (B \cdot C) \\
A \cdot (B + C) &= A \cdot B + A \cdot C
\end{align*}
\]

**De Morgan’s law:**

\[
\overline{A \cdot B} = A + B
\]

**OR**

\[
\begin{align*}
A + 0 &= A \\
A + 1 &= 1 \\
A + \overline{A} &= 1 \\
A + B &= B + A \\
(A + B) + C &= A + (B + C) \\
A + B \cdot C &= (A + B) \cdot (A + C)
\end{align*}
\]
These two laws are frequently used in minimisation.
Laws of Logic (implementation)

**AND**

\[ A \cdot \overline{A} = 0 \]

Consequence of binary logic

**OR**

\[ A + \overline{A} = 1 \]

**Commutative Law**

\[ A \cdot B = B \cdot A \]

\[ A + B = B + A \]
Laws of Logic
(implementation)

AND

\[(A \cdot B) \cdot C = A \cdot (B \cdot C)\]

OR

\[(A + B) + C = A + (B + C)\]

Associative Law
Laws of Logic (implementation)

**AND**

\[ A \cdot (B + C) = A \cdot B + A \cdot C \]

**OR**

**Distributive Law**

\[ A + B \cdot C = (A + B) \cdot (A + C) \]

\[ (A + B) \cdot (A + C) = A \cdot A + A \cdot C + B \cdot A + B \cdot C \]

\[ = A \cdot A + A \cdot C + A \cdot B + B \cdot C = \]

\[ A + A \cdot C + A \cdot B + B \cdot C = \]

\[ = A + A + A \cdot C + A \cdot B + B \cdot C = \]

\[ A \cdot (1 + C) + A \cdot (1 + B) + B \cdot C = \]

\[ = A \cdot 1 + A \cdot 1 + B \cdot C = \]

\[ A + A + B \cdot C = A + B \cdot C \]
De Morgan’s Law

It allows to swap the gate implementation

**AND**

\[
\overline{A + B} = \overline{A} \cdot \overline{B}
\]

Break the line and change the sign.

**OR**

\[
\overline{A \cdot B} = \overline{A} + \overline{B}
\]

If there is no bar, add two and break one.
De Morgan’s Law

It allows to swap the gate implementation

\[ A + B = A \cdot \overline{B} \]

\[ A \cdot B = A + \overline{B} \]

It implies **universality** of NAND and NOR gates. Since both NAND and NOR gates can be used to implement OR and AND respectively, by inverting the inputs, **any function can be implemented using NAND or NOR gates only.**
De Morgan’s Law

\[ \overline{A \cdot \overline{B}} = A + B \]

From one gate to three gates!!!(is this the right approach???)

Inputs are usually provided in their straight or inverted forms.

Hence if the two inputs are provided in their inverted form, the first two NAND gates are not needed.

NAND gates are faster than OR gates.
Minimisation Laws

\[ A + A \cdot B = A \]

Absorption Law

\[ A + A \cdot B = A \cdot (1 + B) = A \cdot 1 = A \]

Minimisation = remove the redundant gates keeping the functionality (the functionality has not changed)
Minimisation Laws

\[ A + \overline{A} \cdot B = A + B \]

\[ A + \overline{A} \cdot B = A + A \cdot B + \overline{A} \cdot B \]

\[ A + A \cdot B + \overline{A} \cdot B = A + B \cdot (A + \overline{A}) = A + B \cdot 1 = A + B \]

Minimisation = remove the redundant gates keeping the functionality (the functionality has not changed)