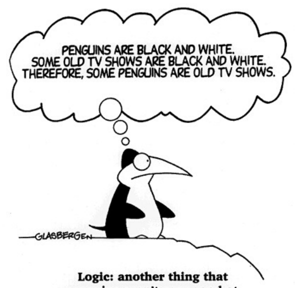
2. Boolean Logic & Basic Gates

2.1 Boolean Logic



Logic: another thing that penguins aren't very good at.

- Logic is the realm of human reasoning that allows you to determine whether or not a particular statement is TRUE or FALSE depending on the truth of other pertinent conditions.

- Logic based on true/false states lends itself very well to digital states which, as we now know, are also based on two states.

- By way of example, consider the logic statement *'the light is on' if 'the bulb is not burned out' and if 'the light switch is on'*.

- More explicitly, we can say that '*the light is on*' *is TRUE if 'the bulb is not burned out' is TRUE and if 'the light switch is on' is TRUE.*

- We can express this in tabular form to allow us to see the 'big picture' more clearly, as follows:

the bulb is not burned out	***the light switch is on***	***the light is on***
TRUE	FALSE	⇒ **FALSE**
FALSE	TRUE	⇒ **FALSE**
FALSE	FALSE	⇒ **FALSE**
TRUE	TRUE	⇒ **TRUE**

- Using a binary representation of '1' and '0', this table looks like:

the bulb is not burned out	***the light switch is on***	***the light is on***
1	0	**1**
0	1	**0**
0	0	**0**
1	1	**1**

- The above tabular form is known as a **truth table**, in which all possible combinations of the inputs are listed.

- The English logician and mathematician, George Boole, developed a mathematical system for such logic, allowing problems in logic to be solved in an algebraic way.

- This branch of mathematics is known as ***Boolean Algebra*** and is applied in the design and analysis of digital systems.

2.2 Basic Logic Gates

- Digital logic circuits contain several basic logic components, referred to as **gates**. These gates carry out important logical operations and form the fundamental building blocks of digital systems.

- We are now going to study a selection of these important gates.

- Each gate will be examined in terms of its symbol, its Boolean algebraic expression and operator and its truth table.

- Note, in the following section of the notes, **A**, **B**, **C**, etc… are used to **denote logical inputs** while **f denotes a logical output**.

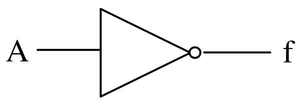
The NOT gate …

- The **NOT** gate (or the **Inverter**) simply inverts the input. Hence, if the input is a logic ‘1’ then the output will be a logic ‘0’ and vice versa.

- The truth table for the inverter is:

A	**f**

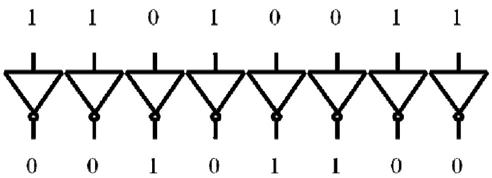
- The symbol for the gate is:



- The Boolean expression for the NOT gate is: f = $\overline{A}$ (or A′)

- An example application using a bank of inverters is to produce the 1’s complement of an 8-bit binary number, as illustrated below:

Input – an 8-bit binary number



Output – the 1’s complement

The AND gate …

- The **AND** gate performs logical ***multiplication***.

- For a 2-input AND gate, the output is only HIGH if both inputs are HIGH, otherwise the output is LOW.

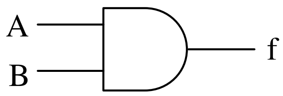
- This principle extends to an *n*-input gate. In other words, for an *n*-input AND gate, the output is HIGH if **ALL** *n* inputs are HIGH, otherwise it is LOW.

- The truth table for the 2-input AND gate is:

A	B	f

- *Note – as there are 2 inputs, there are 4 possible combinations to be covered in the truth table. Recall that for n-inputs there are 2^n possible combinations.*

- The symbol for the gate is:

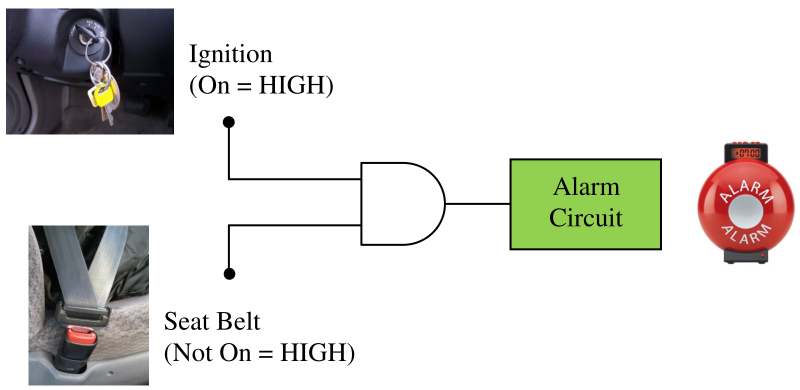


- The Boolean expression for a 2-input AND gate is: f = AB (or A.B)

- The Boolean expression for a 3-input AND gate is: f = ABC

- And so on for a higher number of inputs …

- An example seat belt alarm application using an AND gate is shown below:



__The OR gate …__

- The **OR** gate performs logical ***addition***.

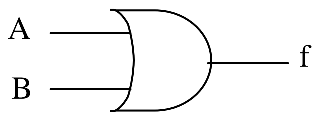
- For a 2-input OR gate, the output is HIGH if any of the inputs are HIGH. The output is only LOW if both inputs are LOW.

- For an *n*-input OR gate, the output is HIGH if **ANY** of the *n* inputs are HIGH, otherwise it is LOW.

- The truth table for the 2-input OR gate is:

A	B	f

- The symbol for the gate is:

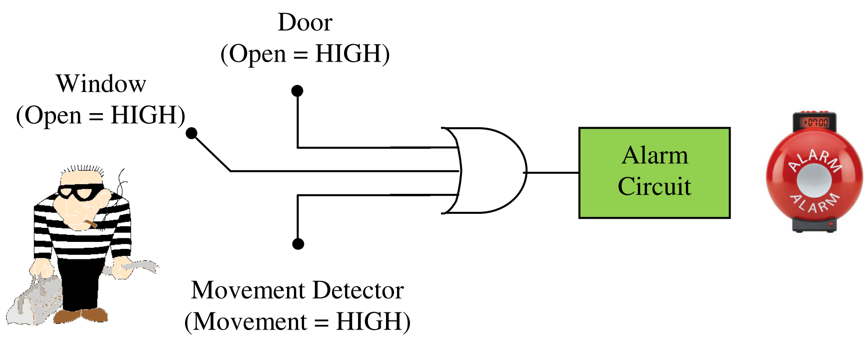


- The Boolean expression for a 2-input OR gate is: $f = A + B$

- The Boolean expression for a 3-input OR gate is: $f = A + B + C$

- And so on for a higher number of inputs …

- An example intruder alarm application using an OR gate is shown below:



The NAND gate …

- The **NAND** gate stems from NOT–AND. It implies an AND function followed by an inverter.

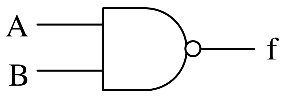
- For a 2-input NAND gate, the output is only LOW if both inputs are HIGH, otherwise the output is HIGH.

- In general, we can say that for an *n*-input NAND gate, the output is LOW if **ALL** *n* inputs are HIGH, otherwise it is HIGH.

- Thus, the truth table for the NAND gate is:

A B	**f**

- The symbol for the gate is:

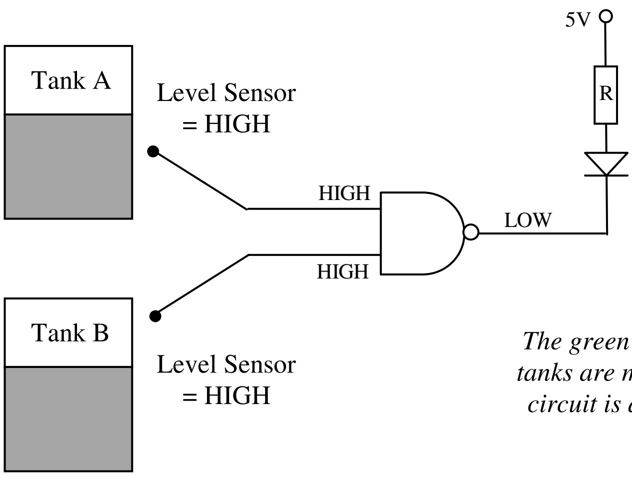


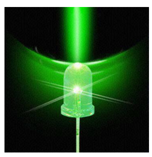
- The Boolean expression for a 2-input NAND gate is: $f = \overline{AB}$ (or $\overline{A.B}$)

- The Boolean expression for a 3-input NAND gate is: $f = \overline{ABC}$

- And so on for a higher number of inputs …

- An example fluid level monitoring application using a NAND gate is shown below:





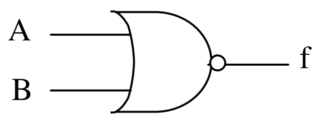
The green LED is ON, indicating that both tanks are more than half full – note how the circuit is arranged so that a LOW voltage turns the light on.

The NOR gate …

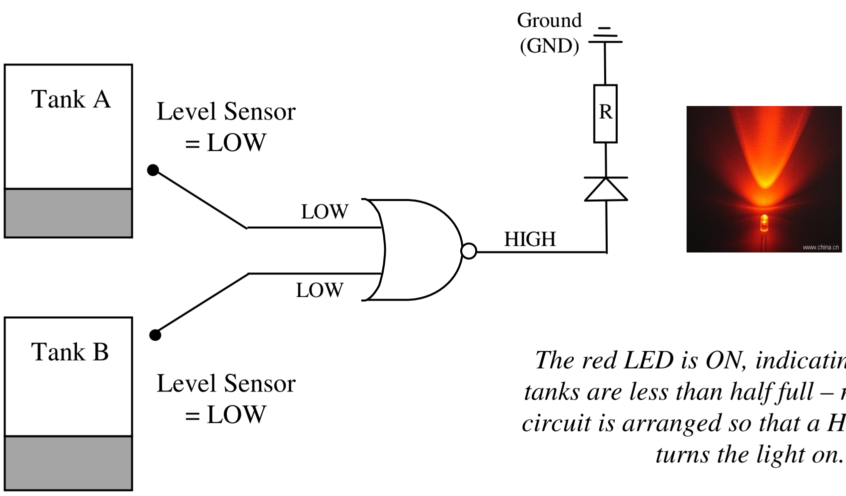
- The **NOR** gate stems from NOT–OR. It implies an OR function followed by an inverter.
- For a 2-input NOR gate, the output is LOW if any of the inputs are HIGH otherwise it is HIGH.
- In general, we can say that for an *n*-input NOR gate, the output is LOW if **ANY** of the *n* inputs are HIGH, otherwise it is HIGH.
- Thus, the truth table for the NOR gate is:

A	B	f

- The symbol for the gate is:



- The Boolean expression for a 2-input NOR gate is: $f = \overline{A + B}$
- The Boolean expression for a 3-input NOR gate is: $f = \overline{A + B + C}$
- And so on for a higher number of inputs …
- An example low fluid level warning system using a NOR gate is shown below:



The red LED is ON, indicating that both tanks are less than half full – note how the circuit is arranged so that a HIGH voltage turns the light on.

__The XOR gate …__

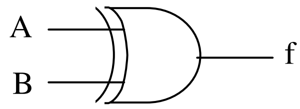
- The **XOR** (**Exclusive – OR**) gate is a combination of the previous gates. However, it is an important gate in many applications and, as such, has its own symbol.

- For a 2-input XOR gate, the output is HIGH if both inputs are different and the output is LOW if both inputs are the same.

- Thus, the truth table for the XOR gate is:

A	B	f

- The symbol for the gate is:



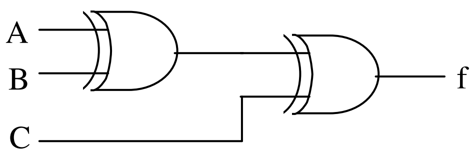
- The Boolean expression for the XOR gate is: $f = A \oplus B \quad (= \overline{A}B + A\overline{B})$

- The XOR operation is a binary operation and is therefore defined for two inputs only.

- However, it is nevertheless common in electronic design to use the XOR operation on 3 or more signals.

- For 3 (or more) inputs $A \oplus B \oplus C = 1$ only if the number of 1's in the input combination is odd.

- Since XOR gates are only designed for 2 inputs, the 3-input XOR function is implemented by using two 2-input XOR gates as follows:



- On closer inspection of the truth table, we can see that the XOR gate acts like a 2-bit adder:

  0 + 0 = 0
  0 + 1 = 1
  1 + 0 = 1
  1 + 1 = 0 *… the carry of '1' is lost*

The XNOR gate …

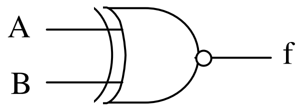
- The final gate that we are going to look at is the **XNOR** (**Exclusive – NOR**) gate.

- For a 2-input XNOR gate, the output is LOW if both inputs are different and the output is HIGH if both inputs are the same.

- Thus, the truth table for the XNOR gate is:

A	**B**	**f**

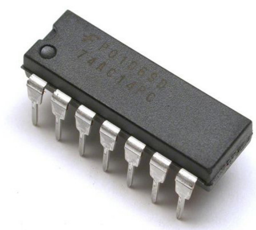
- The symbol for the gate is:



- The Boolean expression for a 2-input OR gate is: $f = \overline{A \oplus B}$ $(= AB + \overline{A}\,\overline{B})$

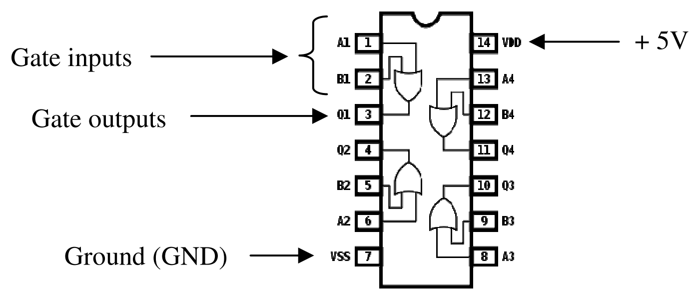
- On closer inspection of the truth table, we can see that the XNOR gate acts like a 2-bit comparator, i.e. when both bits are the same, the output is HIGH but when the bits are different the output is LOW.

Integrated Circuits (ICs) …



- From a laboratory viewpoint, you will notice that logic gates are provided on integrated circuits (ICs), where a single IC can contain multiple gates.

- For example, a ***quad 2-input OR*** integrate circuit contains 4 × 2-input OR gates, as illustrated below. Hence, this IC provides you with 4 OR gates on one chip.



- *In labs, make sure that you are using the appropriate ICs for your circuits and make sure to connect them correctly!*

- *Detailed information on integrated circuit technologies (such as CMOS and TTL) will be covered in other modules.*

2.3 Preliminary Definitions

- A **canonical truth table** is a table in which **all** possible combinations are listed.

- For example, the canonical truth table of the AND gate:

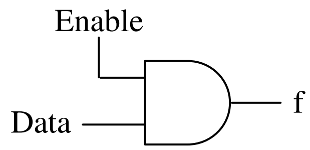
A	B	f
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

- Other possibilities exist – for example, the AND gate can be view as an ***enable*** gate and therefore, we can express its operation in tabular form as follows:

A	f
0	**0**
1	**B**

$\Longleftrightarrow$

Enable	f
0	**0**
1	**Data**



- A **variable** is a symbol used to represent a logical quantity. As we have seen, a single variable can have a value of 1 or 0.

- The **complement** is the inverse of a variable and is denoted by a bar over the variable. For example, the complement of variable A is $\overline{A}$.

- A **literal** is a variable or its complement.

- A **canonical product term** is a product expression containing all literals e.g. ABC for a 3 variable function.

- A **canonical sum term** is a sum expression containing all literals e.g. A + B + C for a 3 variable function.

- A **canonical expression** is a logical expression made up of canonical terms.

- There are two standard forms used, namely the **sum-of-products** (SOP) and the **product-of sums** (POS). For example:

$$f_{(x,y,z)} = xy\overline{z} + xyz + \overline{x}yz \qquad \textit{Sum of products (SOP)}$$

$$f_{(x,y,z)} = (x + \overline{y} + z)(\overline{x} + y + z) \qquad \textit{Product of sums (POS)}$$

Note, how each term is in canonical form!

- The canonical form can often be simplified or minimised to reduce the number of logic gates required to implement a particular Boolean function.

- For example a **simplified or minimised Boolean function** might look like:

$$f_{(x,y,z)} = x + x\overline{y}z + y\overline{z} \qquad \textit{Sum of products (SOP)}$$

$$f_{(x,y,z)} = (x + \overline{y})(y + \overline{z}) \qquad \textit{Product of sums (POS)}$$

- The main goal in digital system design is to obtain a minimised expression for a digital function since this generally leads to a more cost-effective design, i.e. fewer gates are required.

- There exist a few different techniques for carrying out Boolean minimisation. We are now going to look in detail at two such techniques, namely Boolean Algebra and Karnaugh Maps.