

Chapter 1: Logic systems

1: Logic gates

Learning Objectives:

At the end of this topic you should be able to:

- identify the symbols and truth tables for the following logic gates:
 - NOT
 - AND
 - NAND
 - OR
 - NOR
 - XOR
 - XNOR
- construct truth tables for simple combinations of these logic gates.

Introduction

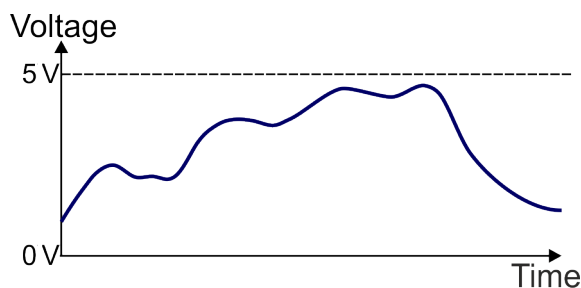
This course is designed to provide a broad introduction to electronics, giving a foundation on which to build further studies. The emphasis throughout is on practical work, though computer simulation may be used to illustrate aspects of circuit behaviour.

In this first section, we concentrate on digital and analogue circuits.

An analogue signal:

- gives an 'analogy' - copies the behaviour of a physical quantity. A temperature sensor, for example, outputs a voltage that copies the behaviour of the temperature of an object. A change in temperature produces a corresponding change in voltage.
- can have any value between the minimum and maximum of the power supply.

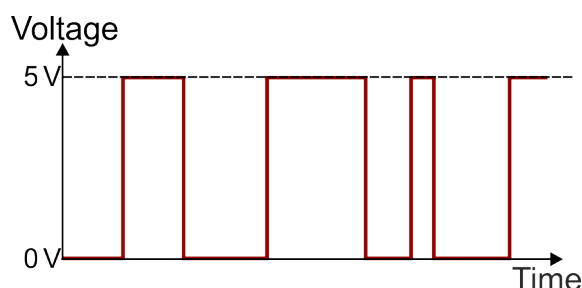
The graph shows a typical analogue signal in a circuit powered by a 5 V supply.



A digital signal:

- carries information in the form of a number. Electronic systems usually employ the binary number system, which uses only the numbers **0** and **1**, coded as voltages.
- has only two possible values - we could decide that **0** = 0 V and **1** = 5 V, for example. Changes between these occur instantaneously.

The graph shows a typical digital signal in a circuit powered by a 5 V supply.

**Terminology:**

When a digital signal is at minimum voltage (usually 0 V,) it is referred to as a **LOW** signal or **LOGIC 0** signal.

When the signal is at maximum voltage, it is referred to as a **HIGH** signal or **LOGIC 1** signal.

Logic Gates

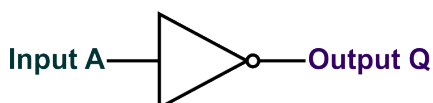
Logic gates are the basic building block of all digital electronic systems.

They are the decision making units in electronic systems and there are different types for different situations.

The NOT gate (or inverter)

The simplest form of logic gates has only one input and one output. Its function is to invert the input signal - it turns a logic **0** input into a logic **1** output and vice-versa.

The symbol for a NOT gate is as follows:



The behaviour of a logic gate is summarised in a table, called a 'Truth Table'.

The truth table for a NOT gate is the simplest of all and is shown below:

Input A	Output Q
0	1
1	0

There is also a shorthand way of writing down the function of this logic gate, using a special type of algebra called **Boolean algebra**.

The Boolean expression for a NOT gate is: $Q = \bar{A}$

The 'bar' over the A indicates that output **Q** is the opposite of input signal **A**.

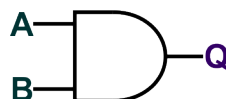
After the NOT gate, the four most common logic gates are:

- AND gate
- OR gate
- NAND gate
- NOR gate

Note: These logic gates have a minimum of two inputs but can have up to eight. However, this course will use gates with a maximum of three inputs.

The AND gate

The symbol for a two input AND gate is as follows:



Its truth table is:

Inputs		Output
B	A	Q
0	0	0
0	1	0
1	0	0
1	1	1

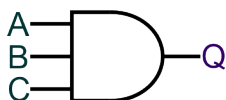
The output is logic 1 only when input **A AND** input B are both at logic 1.

The Boolean expression for a two input AND gate is: $Q = A \cdot B$

The '.' between the **A** and **B** means **AND** in Boolean algebra.

Now, the three input AND gate:

The symbol is:



The truth table is:

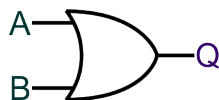
Inputs			Output
C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

The output is logic 1 only when input **A AND** input **B AND** input **C** are at logic 1.

The Boolean expression for a three input AND gate is: $Q = A \cdot B \cdot C$

The OR gate

The symbol for a two input OR gate is as follows:



The truth table for it is:

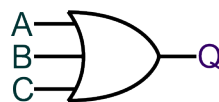
Inputs		Output
B	A	Q
0	0	0
0	1	1
1	0	1
1	1	1

The output is logic **1** when input **A** OR input **B** OR both are at logic **1**.

The Boolean expression for a two input OR gate is: $Q = A + B$

(The '+' between the A and B means OR in Boolean algebra).

Unsurprisingly, the symbol for a three input OR gate is:



and the truth table:

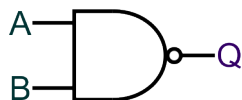
Inputs			Output
C	B	A	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

The output is logic **1** when either input **A** OR input **B** OR input **C** OR any combination is/are at logic **1**.

The Boolean expression for a three input OR gate is: $Q = A + B + C$

The NAND gate

The symbol for a two input NAND gate is:



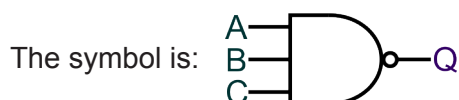
Its truth table is:

Inputs		Output
B	A	Q
0	0	1
0	1	1
1	0	1
1	1	0

Compare this truth table with that for the AND gate. For the NAND gate, output Q is the exact opposite of that for the AND gate.

The Boolean expression for the two input NAND gate is: $Q = \overline{A \cdot B}$

As before the '.' between **A** and **B** means **AND**, and the 'bar' means 'invert' in Boolean algebra. Now, the three input NAND gate:



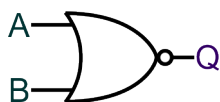
The truth table is:

Inputs			Output
C	B	A	Q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

The Boolean expression for a 3-input NAND gate is: $Q = \overline{A \cdot B \cdot C}$

The NOR gate

The symbol for a 2-input NOR gate is:



The truth table for this gate is:

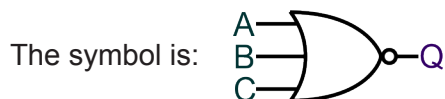
Inputs		Output
B	A	Q
0	0	1
0	1	0
1	0	0
1	1	0

Compared to the OR gate, the NOR gate outputs are the exact opposite.

The Boolean expression for a 2-input NOR gate is: $Q = \overline{A + B}$

As before, the '+' means **OR** and the 'bar' means 'invert' in Boolean algebra.

Next, the 3-input NOR gate:



The truth table is:


Inputs			Output
C	B	A	Q
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

The Boolean expression for a 3-input NOR gate is: $Q = \overline{A + B + C}$

The XOR gate

The XOR (EXclusive OR) gate is related to the OR gate.

The symbol for a two input XOR gate is:



The truth table for this gate is:

Inputs		Output
B	A	Q
0	0	0
0	1	1
1	0	1
1	1	0

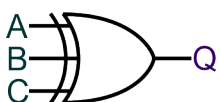
The output is logic 1 when **either A or B** is logic 1, but **not** when both **A and B** are logic 1.

The Boolean expression for a two input XOR gate is: $Q = A \oplus B$

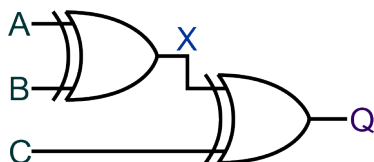
or, alternatively: $Q = \bar{A}.B + A.\bar{B}$

The ' \oplus ' between the **A** and **B** indicates **Exclusive** OR. The alternative form is more useful when simplifying Boolean expressions.

Next, the three input XOR gate. The symbol is:



The behaviour is that of **two** 2 input XOR gates connected one after the other, as the following diagram shows:



The output of the first XOR gate, labelled **X**, is obtained using inputs **A** and **B**. Output **Q**, the output of the second XOR gate, is obtained using **C** and **X** as inputs.

Putting this information into a truth table gives:

Inputs			Outputs	
C	B	A	X	Q
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

The output **Q** is logic 1 when **either A OR B OR C** is logic 1, **or A AND B AND C** are all logic 1 but **NOT** when any **two** inputs are logic 1.

Written as a Boolean expression, this is: $Q = A.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.\bar{B}.C + A.B.C$

The final truth table for the 3-input XOR is:

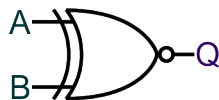
Inputs			Output
C	B	A	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The Boolean expression for a three input XOR gate is usually written as: $Q = A \oplus B \oplus C$

The XNOR gate

The XNOR (eXclusive-NOR) gate is the inverted form of the XOR gate.

The symbol for a two input XNOR gate is:



Its truth table is:

Inputs		Output
B	A	Q
0	0	1
0	1	0
1	0	0
1	1	1

The output is the exact opposite to that of the XOR.

The Boolean expression for the two input XNOR gate is: $Q = \overline{A \oplus B}$

or, alternatively: $Q = \overline{A} \cdot \overline{B} + A \cdot B$

Once again however the alternative form will prove to be more useful later on in the course when simplifying Boolean expressions.

The 3-input XNOR is the exact opposite of the 3-input XOR gate. Its truth table is:

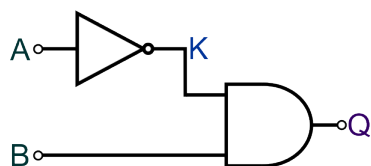
Inputs			Output
C	B	A	Q
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The Boolean expression for the 3-input XNOR is: $Q = \overline{A \oplus B \oplus C}$

Analysing simple logic circuits

Questions may focus on individual logic gates but are more likely to consider combinations of gates. For example, you could be asked to complete the truth table for a combinational logic system.

Example 1: Study the following logic system carefully and then complete the truth table that follows:



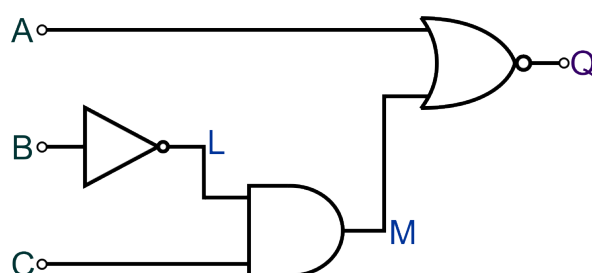
Inputs		Outputs	
B	A	K	Q
0	0		
0	1		
1	0		
1	1		

The output of the NOT gate is labelled **K**. The first step is to complete the column for output **K**, the inverse of **A**. This is shown below:

Inputs		Outputs	
B	A	K	Q
0	0	1	
0	1	0	
1	0	1	
1	1	0	

The next step is to complete the final column. Signal **Q** is the output of the AND gate which has B and **K** as inputs.

Example 2: Complete the truth table for the system shown below:



Inputs			Outputs		
C	B	A	L	M	Q
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

First step - complete the column for the NOT gate output (L). [Remember its input is B].

Second step - complete the column for the AND gate output (M). [Its inputs are M and C].
Finally - complete the NOR gate output (Q). [Its inputs are A and M].

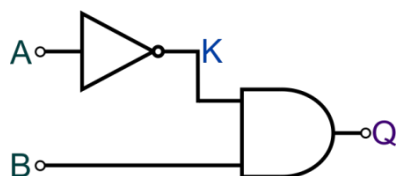
Solution:

Inputs			Outputs		
C	B	A	L	M	Q
0	0	0	1	0	1
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	0	0	0

Investigation 1.1

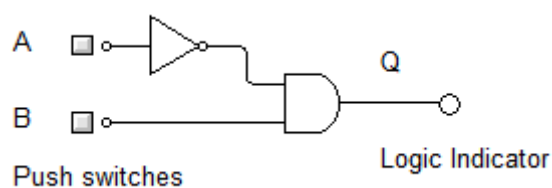
Set up each of the logic circuits given using a simulation program such as 'Circuit Wizard' or 'Yenka Technology'. Complete the truth table for each circuit.

1.

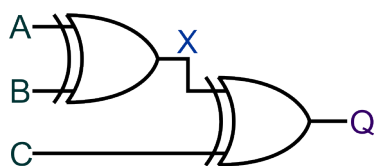


Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	

Note: Using the “built in” inputs and outputs, the resulting circuit would resemble the following:



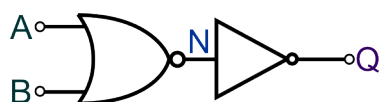
2.



Inputs			Output
C	B	A	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

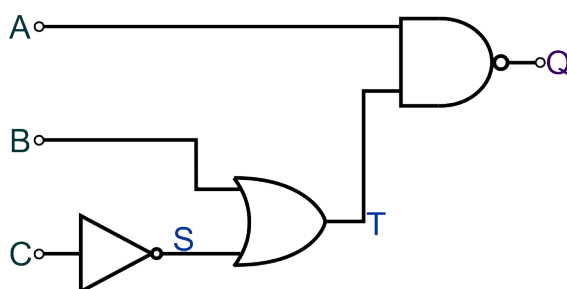
Exercise 1.1

1. Complete the truth table for the system shown below:



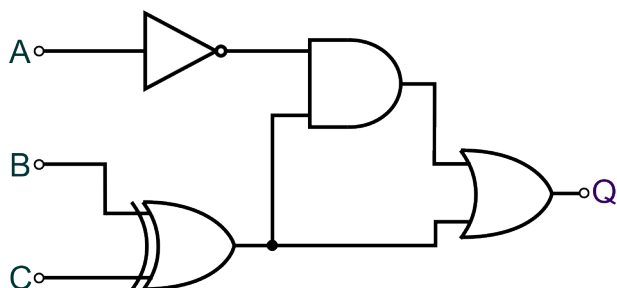
Inputs		Outputs	
B	A	N	Q
0	0		
0	1		
1	0		
1	1		

2. Complete the truth table for the system shown below:



Inputs			Outputs		
C	B	A	S	T	Q
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

3. In this question, the intermediate outputs are not labelled.
Create and complete the truth table for this system:



The same Q output could be produced by a single two input gate.

What type of gate is this and what are the two inputs?

Practical Logic Gates

Logic gates are available within an **integrated circuit (IC)** - a set of electronic circuits built on the same wafer of semiconductor material.

These logic IC's are usually supplied in plastic DIL (dual in line) packages containing several logic gates of the same type.

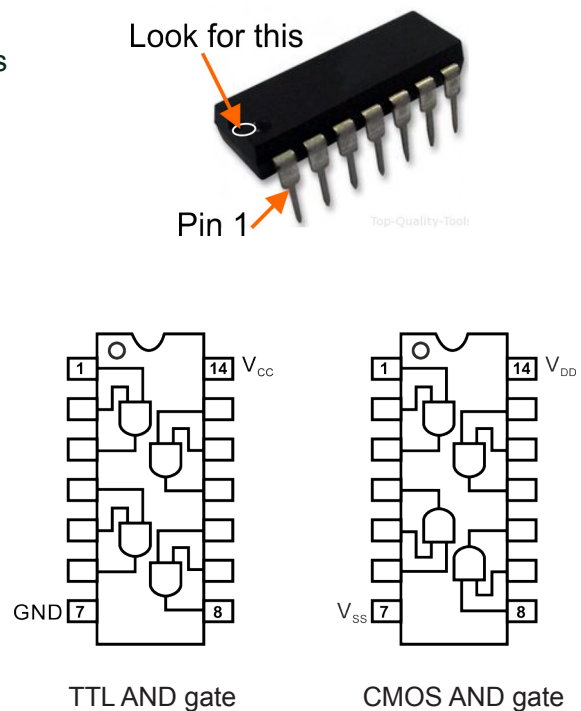
The diagram shows one of these logic IC packages.

At one end, there is a 'D'-shaped notch and nearby a circular indentation.

This identifies the location of pin 1.

There are two common types of package available, known as TTL or 74 series and CMOS or 4000 series.

Examination questions will not test the differences between TTL and CMOS devices. However, these are important for any practical tests that you carry out and will be particularly important for your project work.



The data sheet for a logic gate package includes the pin out - how the pins connect to the logic gates inside it. The pin outs opposite relate to ICs known as quad 2-input AND gates. Using the right pin out is important, as incorrect connections can damage the whole package.

Pull-up and pull-down resistors

When switches are used to input digital signals into logic gates, they are always combined with a resistor into a switch unit.

The circuits shown below use a 12 V power supply, for illustrative purposes. The results apply to any power supply voltage.



There are two ways to set up a switch unit.

These are shown above and produce opposite effects:

- Circuit A outputs a 0 V signal until the switch is pressed. It then outputs ~12 V.
- Circuit B outputs a 12 V signal until the switch is pressed. It then outputs ~0 V.

In Circuit A:

- before the switch is pressed, there is no connection to the 12 V power rail. The input to the logic gate is 'pulled-down' to 0 V by resistor R, inputting logic **0** to the logic system.
- when the switch is operated, it has virtually zero resistance and so the voltage drop across it is ~ 0 V. All the supply voltage appears across the resistor, so voltage drop across it is ~ 12 V, changing the input logic level to logic **1**.

In Circuit B:

- before the switch is pressed, there is no connection to the 0 V power rail. The input to the logic gate is 'pulled-up' to 12 V by resistor R, inputting logic **1** to the logic system.
- when the switch is operated, the voltage drop across the switch is ~ 0 V. All of the supply voltage, 12 V, is dropped across the resistor, changing the input to logic **0**.

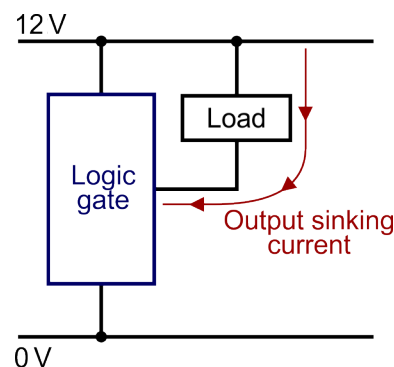
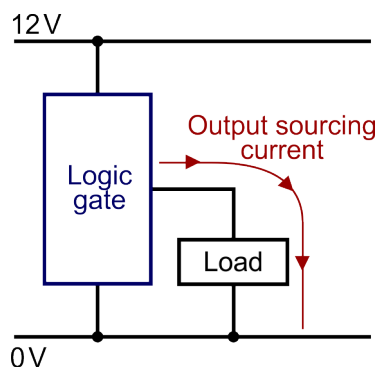
In summary

- Circuit A outputs a logic **0** signal until the switch is pressed. It then outputs logic **1**.
- Circuit B outputs a logic **1** signal until the switch is pressed. It then outputs logic **0**.

Sourcing and Sinking

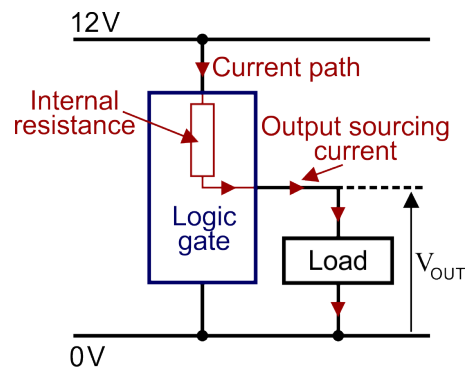
When a load is connected to a logic gate in such a way that the logic gate supplies current to the load, the logic gate is said to be **sourcing current**.

When a load is connected so that current flows from the power supply through the load to the logic gate, then the logic gate is said to be **sinking current**.

**Sourcing current out of a logic gate:**

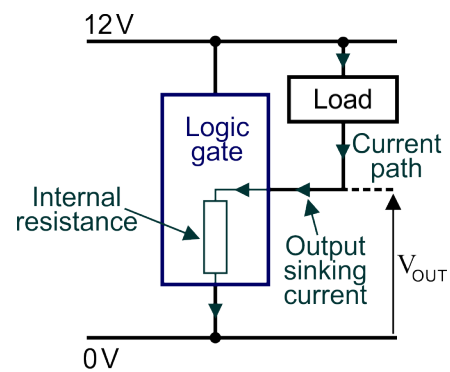
The schematic diagram on the right show the current path when a logic gate is sourcing current.

With an output signal of logic **1**, current flows from the 12 V power supply through an internal resistance within the logic gate **out of** the output terminal of the logic gate and through the load to 0 V.



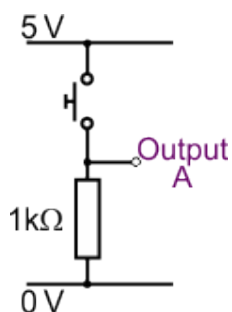
Sinking current into a logic gate:

The schematic diagram on the right show the current path when a logic gate is sinking current. With an output signal of logic **0**, current flows from the 12 V power supply through the load **into** the output terminal of the logic gate, through an internal resistance within the logic gate to 0 V.

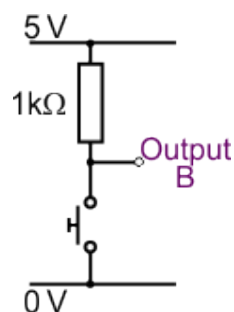


Investigation 1.2

1. Set up and investigate the following circuits, using a logic probe.
Complete the tables below by inserting 'high' or 'low' for the outputs of the switches



Switch SW2	Output A
Open	
Closed	



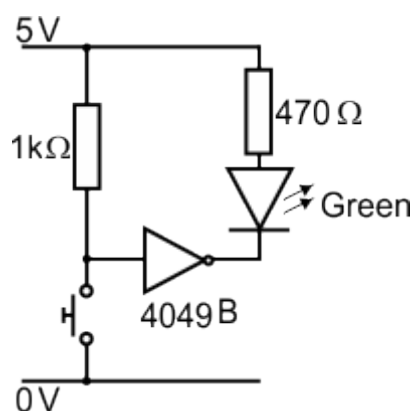
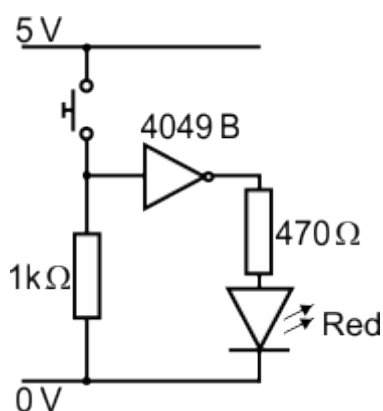
Switch SW1	Output B
Open	
Closed	

2. Set up and investigate the following circuits.

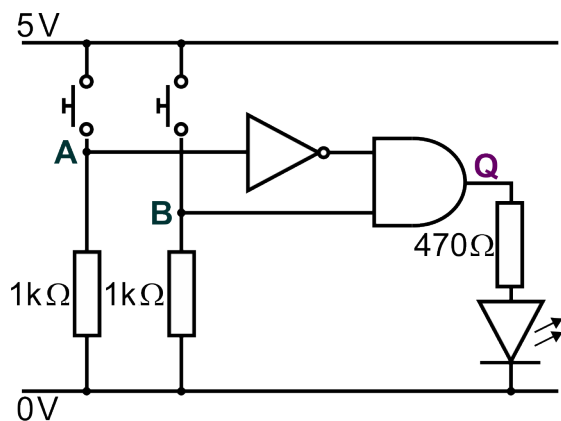
Complete the following statements:

When SW1 is open the red LED is

When SW2 is closed the green LED is



3. Set up this circuit and complete the truth table.
When the LED is on, the output is high. When it is off the output is low.



Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	

2: Boolean algebra

Learning Objectives:

At the end of this topic you should be able to:

- generate the Boolean expression for a system with up to three inputs from a truth table.
- generate the Boolean expression for a system with up to four inputs from a logic circuit diagram.
- recall and use the following identities:
 - $A \cdot 1 = A$
 - $A \cdot 0 = 0$
 - $A \cdot A = A$
 - $A \cdot \bar{A} = 0$
 - $A + 1 = 1$
 - $A + 0 = A$
 - $A + A = A$
 - $A + \bar{A} = 1$
- select and use the following identities:
 - $A + \bar{A} \cdot B = A + B$
 - $A \cdot B + A = A \cdot (B + 1) = A$
- apply de Morgan's theorem to simplify a logic system with up to three inputs.

Logic systems can become quite complex. We need a range of simplification techniques. The first of these is Boolean algebra manipulation.

Here is a brief summary of the results that have emerged so far:

- A bar, on top of a variable, represents the NOT function e.g. $\bar{C} = \text{NOT } C$.
- A dot '.', represents the AND function e.g. $A \cdot B = A \text{ AND } B$.
- A plus '+', represents the OR function e.g. $E + F = E \text{ OR } F$.
- A plus with a circle ' \oplus ', represents the Exclusive OR (EXOR) function e.g. $C \oplus D = C \text{ EXOR } D$.

There are two ways in which Boolean expressions for a logic system can be generated, either from a truth table or from a logic circuit diagram.

First skill - complete a Boolean expression from a truth table.

Consider the following truth table for a logic system (don't worry about what it is meant to do).

C	B	A	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

In this table, four combinations of inputs produce an output of logic 1. The first step is to write down the Boolean equation for each of these. All input variables must be included in each individual Boolean equation, as shown below:

C	B	A	Q	Boolean Equation
0	0	0	0	
0	0	1	1	$\overline{C}.\overline{B}.A$
0	1	0	0	
0	1	1	1	$\overline{C}.B.A$
1	0	0	1	$C.\overline{B}.\overline{A}$
1	0	1	1	$C.\overline{B}.A$
1	1	0	0	
1	1	1	0	

To obtain the complete Boolean expression, we take each term and 'OR' them together:

$$Q = \overline{C}.\overline{B}.A + \overline{C}.B.A + C.\overline{B}.\overline{A} + C.\overline{B}.A$$

The resulting expression may not be the simplest. We look at simplification techniques later.

Exercise 1.2

1.

C	B	A	Q	Boolean Equation
0	0	0	1	
0	0	1	0	
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	0	
1	1	0	1	
1	1	1	0	

2.

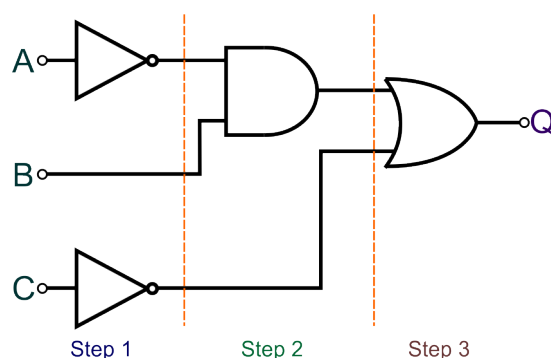
C	B	A	Q	Boolean Equation
0	0	0	0	
0	0	1	0	
0	1	0	1	
0	1	1	0	
1	0	0	0	
1	0	1	1	
1	1	0	0	
1	1	1	1	

3.

D	C	B	A	Q	Boolean Equation
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	1	
0	1	0	1	0	
0	1	1	0	1	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	1	
1	1	1	1	0	

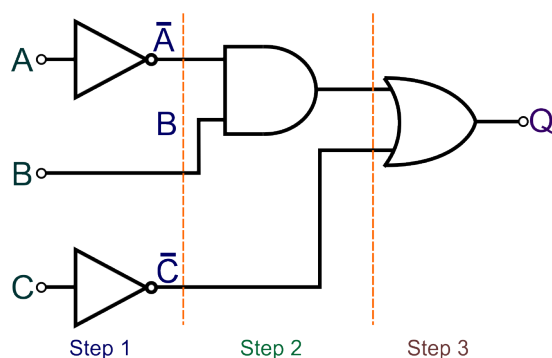
Second skill - generate the Boolean expressions from a circuit diagram.

Example 1: Derive the Boolean expression for the following logic circuit.

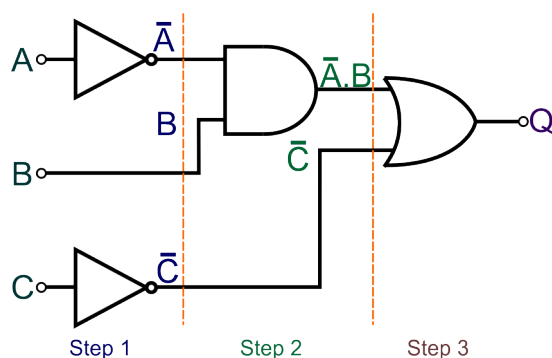


First of all, divide the circuit into stages as shown above. This breaks down a large circuit into smaller more manageable chunks.

Then, write down the Boolean expressions for all the outputs of **Step 1**.



Now do the same for the outputs of **Step 2**, as shown below:

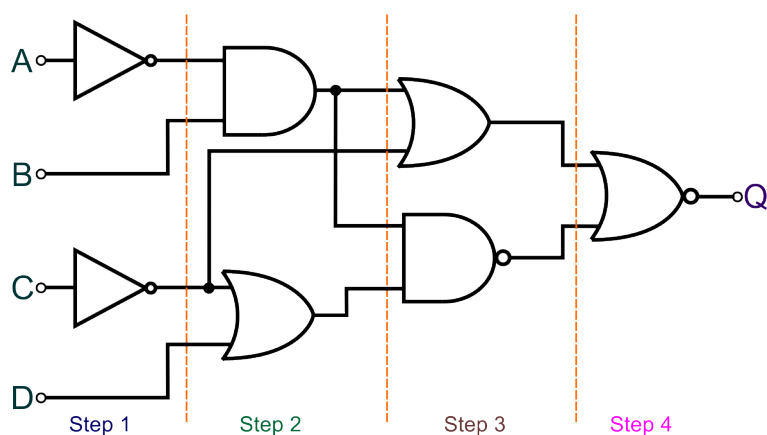


Finally do the same for **Step 3**, to arrive at the expression for the output of the whole system as:

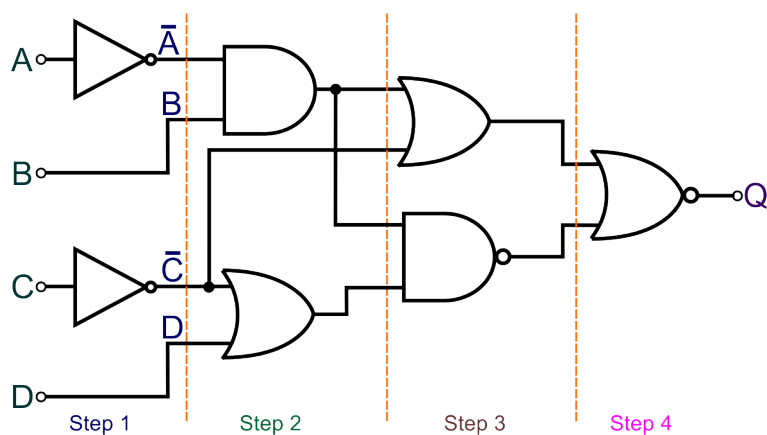
$$Q = \bar{A}.B + \bar{C}$$

Example 2: Now do the same for this much larger system.

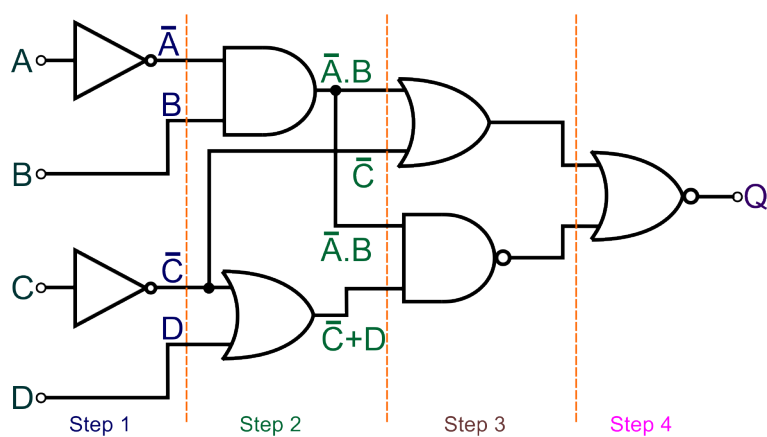
First of all, divide it into small steps:



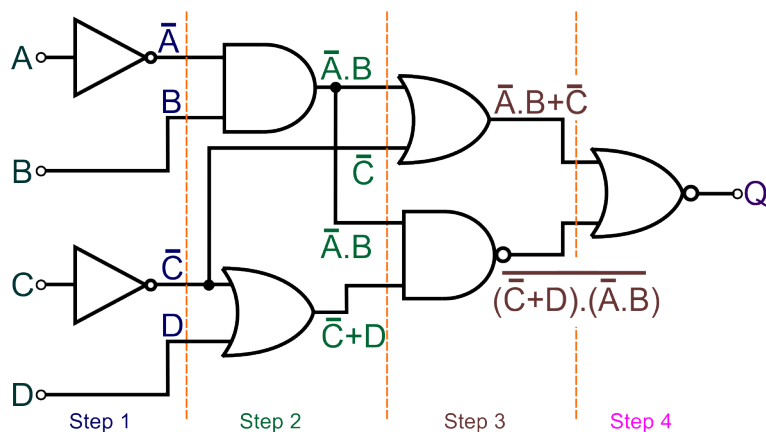
Next, work out all of the outputs in **Step 1**



Now, do the same for **Step 2**



Now, the same for Step 3



Notice the use of brackets to keep the inputs to the NAND gate together.

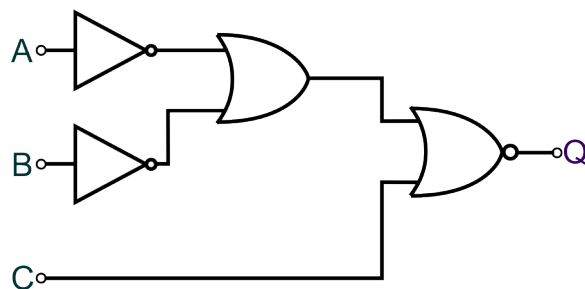
Finally, the two large expressions are inputs into the NOR gate. Again brackets are used to keep the inputs together. The final Boolean equation is:

$$Q = (\bar{A}.B + C) + (\bar{A}.B).(C + D)$$

Exercise 1.3

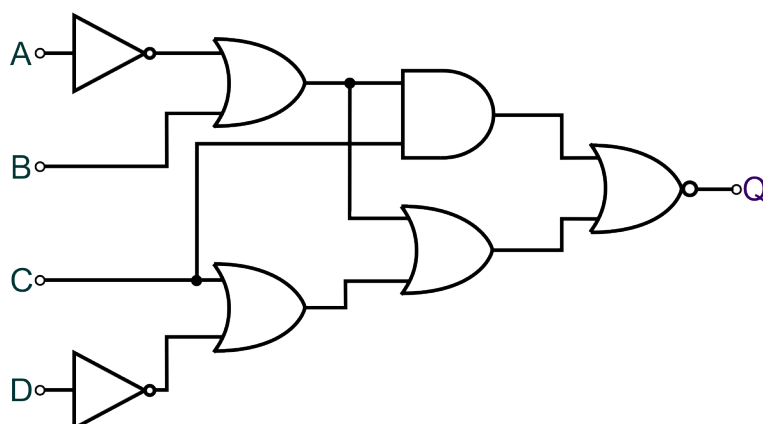
Derive the Boolean equations for the output of the following logic systems.

1.



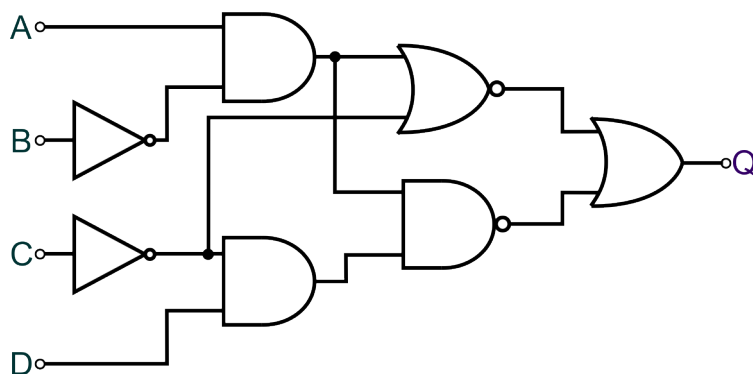
Q =

2.



Q =

3.



Q =

The next requirement is a method of simplifying these Boolean expressions. This relies on extracting specific identities from the complex expressions. These can be reduced to simpler expressions. We now examine these special identities.

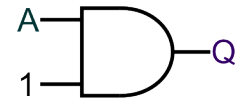
Identity 1 $A \cdot 1 = A$

The AND gate opposite has inputs **A** and logic **1**. Hence output **Q** = **A** · **1**.

If **A** is logic **0**, then **Q** will also be logic **0**, since $0 \cdot 1 = 0$.

If **A** is logic **1**, then **Q** will also be logic **1**, since $1 \cdot 1 = 1$.

Output **Q** is always the same logic level as **A**.



Wherever the term **A** · **1** appears, it can be replaced with **A**.

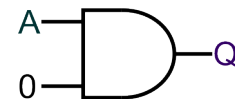
Identity 2 $A \cdot 0 = 0$

Now, the AND gate has inputs **A** and logic **0**. Hence output **Q** = **A** · **0**.

If **A** is logic **0**, then **Q** will also be logic **0**, since $0 \cdot 0 = 0$.

If **A** is logic **1**, then **Q** will still be logic **0**, since $1 \cdot 0 = 0$.

Output **Q** is always logic **0**.



Wherever the term **A** · **0** appears, it can be replaced with logic **0**.

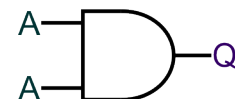
Identity 3 $A \cdot A = A$

The AND gate has inputs **A** and **A**. Hence output **Q** = **A** · **A**.

If **A** is logic **0**, **Q** will also be logic **0**, since $0 \cdot 0 = 0$.

If **A** is logic **1**, **Q** will also be logic **1**, since $1 \cdot 1 = 1$.

Output **Q** is therefore always the same logic level as **A**.



Wherever the term **A** · **A** appears, it can be replaced with **A**.

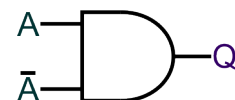
Identity 4 $A \cdot \bar{A} = 0$

The AND gate has inputs **A** and \bar{A} . Hence output **Q** = **A** · \bar{A} .

If **A** is logic **0**, \bar{A} will be logic **1**, so **Q** will be logic **0**, since $0 \cdot 1 = 0$.

If **A** is logic **1**, \bar{A} will be logic **0**, so **Q** will be logic **0**, since $1 \cdot 0 = 0$.

Output **Q** is always logic **0**.



Wherever the term **A** · \bar{A} appears, it can be replaced with logic **0**.

Identity 5 $A + 1 = 1$

The OR gate opposite has inputs **A** and logic **1**. Hence output **Q** = **A** + **1**.

If **A** is logic **0**, then **Q** will be logic **1**, since $0 + 1 = 1$.

If **A** is logic **1**, then **Q** will also be logic **1**, since $1 + 1 = 1$.

Output **Q** is always logic **1**.

Wherever the term **A** + **1** appears, it can be replaced with logic **1**.



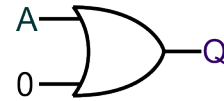
Identity 6 $A + 0 = A$

Now, the OR gate has inputs **A** and logic **0**. Hence output $Q = A + 0$.

If **A** is logic **0**, then **Q** will be logic **0**, since $0 + 0 = 0$.

If **A** is logic **1**, then **Q** will also be logic **1**, since $1 + 0 = 1$.

Output **Q** is therefore always the same logic level as **A**.



Wherever the term $A + 0$ appears, it can be replaced with **A**.

Identity 7 $A + A = A$

The OR gate has inputs **A** and **A**. Hence output $Q = A + A$.

If **A** is logic **0**, **Q** will also be logic **0**, since $0 + 0 = 0$.

If **A** is logic **1**, **Q** will also be logic **1**, since $1 + 1 = 1$.

Output **Q** is therefore always the same logic level as **A**.



Wherever the term $A + A$ appears, it can be replaced with **A**.

Identity 8 $A + \bar{A} = 1$

The OR gate has inputs **A** and \bar{A} . Hence output $Q = A + \bar{A}$.

If **A** is logic **0**, \bar{A} will be logic **1**, so **Q** will be logic **1**, since $0 + 1 = 1$.

If **A** is logic **1**, \bar{A} will be logic **0**, so **Q** will be logic **1**, since $1 + 0 = 1$.

Output **Q** is always logic **1**.



Wherever the term $A + \bar{A}$ appears, it can be replaced with logic **1**.

In all of these examples the variable **A** has been used, but the rules apply for any variable.

Use these identities to simplify some Boolean expressions.

Example 1: Simplify the following expression:

$$Q = A.B.\bar{C} + A.B.C + A.\bar{B}$$

We start by looking for common terms. In this case, A appears in all terms so can be extracted and placed in a bracket as shown:

$$Q = A.(B.\bar{C} + B.C + \bar{B})$$

Similarly, B is common to the first two terms inside the bracket and can be extracted as a common factor, to give:

$$Q = A.(B.(\bar{C} + C) + \bar{B})$$

Using identity 8, the term $\bar{C} + C = 1$, so the expression now becomes:

$$Q = A.(B.(1) + \bar{B})$$

Using identity 1, the term $B.1 = B$, so the expression becomes:

$$Q = A.(B + \bar{B})$$

Using identity 8 again, the expression becomes:

$$Q = A.(1)$$

Using the identity 1 again, the final expression is:

$$Q = A$$

Extracting more complicated common factors can reduce the number of simplification steps. To show this, the same example is simplified by this different technique.

Simplify the following expression:

$$Q = A.B.\bar{C} + A.B.C + A.\bar{B}$$

The first two terms are combined as follows:

$$Q = A.B.(\bar{C} + C) + A.\bar{B}$$

Using identity 8, the term $\bar{C} + C = 1$ so the expression becomes:

$$Q = A.B.1 + A.\bar{B}$$

Using the first identity, this reduces to:

$$Q = A.B + A.\bar{B}$$

Now, extracting the common factor A leaves:

$$Q = A.(B + \bar{B})$$

Using identity 8 again gives:

$$Q = A.1$$

With identity 1, the final expression is, as before:

$$Q = A$$

Example 2: Simplify the following expression:

$$Q = \bar{A}.B + A.B.\bar{C} + A.B + C$$

There is a common factor, **A.B**, in the second and third terms. Extracting it gives:

$$Q = \bar{A}.B + A.B.(\bar{C} + 1) + C$$

In doing so, we used identity 1 in reverse, replacing **A.B** with **A.B.1**.

Using identity 5, the term $(\bar{C} + 1) = 1$, so the expression becomes:

$$Q = \bar{A}.B + A.B.1 + C$$

Using the first identity then reduces the expression to:

$$Q = \bar{A}.B + A.B + C$$

There is a common factor, **B**, in the first two terms. Extracting it gives:

$$Q = B.(\bar{A} + A) + C$$

Using identity 8, the expression reduces to:

$$Q = B.1 + C$$

With identity 1, the final expression is:

$$Q = B + C$$

In the next two examples, the intermediate descriptions have been omitted to streamline the process.

Example 3: Simplify the following expression:

$$Q = \bar{A}.B + \bar{A}.B.C + \bar{A}.\bar{B}.\bar{C} + A.\bar{B}.\bar{C}$$

Solution:

$$Q = \bar{A}.B + \bar{A}.B.C + \bar{A}.\bar{B}.\bar{C} + A.\bar{B}.\bar{C}$$

$$Q = \bar{A}.B + \bar{A}.B.C + \bar{B}.\bar{C}.(\bar{A} + A)$$

$$Q = \bar{A}.B + \bar{A}.B.C + \bar{B}.\bar{C}.1$$

$$Q = \bar{A}.B.(1 + C) + \bar{B}.\bar{C}$$

$$Q = \bar{A}.B.1 + \bar{B}.\bar{C}$$

$$Q = \bar{A}.B + \bar{B}.\bar{C}$$

Example 4: Simplify the following expression:

$$Q = \bar{B}.C.(\bar{C} + D) + C.D + C + \bar{A}$$

Solution:

$$Q = \bar{B}.C.(\bar{C} + D) + C.D + C + \bar{A}$$

$$Q = \bar{B}.C.\bar{C} + \bar{B}.C.D + C.D + C + \bar{A}$$

$$Q = \bar{B}.0 + \bar{B}.C.D + C.(D + 1) + \bar{A}$$

$$Q = \bar{B}.C.D + C + \bar{A}$$

$$Q = C.(B.D + 1) + \bar{A}$$

$$Q = C.1 + \bar{A}$$

$$Q = C + \bar{A}$$

In this last example, notice that the original expression first needed to be expanded before the expression could be simplified.

Exercise 1.4

Simplify the following Boolean expressions:

1. $Q = A.B + B$

.....
.....
.....

2. $Q = C.(A + \overline{C})$

.....
.....
.....
.....
.....

3. $Q = A.B.\overline{C} + A.\overline{C} + \overline{A}.\overline{C}.D + \overline{A}.\overline{C}.\overline{D}$

.....
.....
.....
.....
.....
.....

4. $Q = A.B.(\overline{B} + C) + B.C + B$

.....
.....
.....
.....
.....
.....
.....
.....

Two Special Identities:

There are two special identities that we have not discussed in detail yet, even though we used one of them already in simplifying Boolean expressions. They are printed on the information sheet of the examination paper.

The first of these is: $A.B + A = A.(B + 1) = A$

The second is: $A + \bar{A}.B = A + B$

The second is more difficult to explain:

Consider the condition of variable **A**:

If **A** is logic **0**, then the expression becomes $0 + 1.B = B$

If **A** is logic **1**, then the expression becomes $1 + 0.B = 1$, i.e. the same as **A**.

The \bar{A} term is redundant. It has no effect on the output. It can be deleted.

This identity appears in different forms, for example:

$$\begin{aligned}\bar{A} + A.B &= \bar{A} + B \\ B.\bar{C} + C &= B + C \\ \bar{A}.\bar{C} + A &= \bar{C} + A\end{aligned}$$

In each case, one variable appears in both terms. In one term, it is inverted. The simplification requires the removal of this variable from the AND term.

Further proof is provided by completing the truth table for the same function:

For example, consider the function: $Q = A.\bar{B} + B = A + B$

A	B	\bar{B}	$A.\bar{B}$	$A.\bar{B} + B$	$A + B$
0	0	1	0	0	0
0	1	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

The results in the last two columns are identical. This means that the expression $A.\bar{B} + B$ has the same effect as the expression $A + B$.

This identity is very useful. Here is an example:

Simplify the expression : $Q = A.B.C + A.\bar{C} + C.(D + \bar{C}) + A$

Solution:

$$Q = A.B.C + A.\bar{C} + C.(D + \bar{C}) + A$$

$$Q = A.(B.C + \bar{C}) + C.D + C.\bar{C} + A$$

$$Q = A.(B + \bar{C}) + C.D + 0 + A$$

$$Q = A.B + A.\bar{C} + C.D + A$$

$$Q = A.(B + 1) + A.\bar{C} + C.D$$

$$Q = A + A.\bar{C} + C.D$$

$$Q = A.(1 + \bar{C}) + C.D$$

$$Q = A + C.D$$

The examples considered so far have dealt with only AND, OR and NOT gates.

For NAND and NOR gates, it is important to remember that the inversion takes place after the AND or OR operation.

When simplifying expressions that include these gates, it is not possible to remove one of the terms from under the 'bar' and leave the other behind.

For example:

$$\begin{aligned} &\overline{A.B} + \bar{A} \\ &\neq \bar{A}.(\bar{B} + 1) \\ &\neq A \end{aligned}$$

Completing the truth table below shows this to be the case.

A	B	\bar{A}	$\overline{A.B}$	$\overline{A.B} + \bar{A}$
0	0	1	1	1
0	1	1	1	1
1	0	0	1	1
1	1	0	0	0

Clearly the two highlighted columns are not the same. The terms are not equal.

Another rule, known as de Morgan's theorem is needed to simplify circuits involving these gates.

de Morgan's Theorem

Don't worry about how de Morgan arrived at his theorem. We just need to be able to use it to simplify Boolean expressions containing NAND and NOR terms.

To apply it, use the following rules:

- If you break a 'bar', change the sign underneath the break.
- If you complete a 'bar', change the sign underneath the join.

Example 1: Applying de Morgan's theorem to the expression $\overline{A \cdot B}$ gives:

$$\begin{array}{c} \text{Break the bar} \\ \downarrow \\ \overline{A \cdot B} \rightarrow \overline{A} + \overline{B} \rightarrow \overline{A} + \overline{B} \\ \uparrow \quad \uparrow \\ \text{Change the sign} \end{array}$$

The prediction is: $\overline{A \cdot B} = \overline{A} + \overline{B}$

We can check the result using the truth table method:

A	B	\overline{A}	\overline{B}	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

The last two columns are identical showing that the two logic expressions are the same and therefore showing that the prediction is correct.

Example 2: Applying de Morgan's theorem to the expression $\overline{A + B}$ gives:

$$\begin{array}{c} \text{Break the bar} \\ \downarrow \\ \overline{A + B} \rightarrow \overline{A} \cdot \overline{B} \rightarrow \overline{A} \cdot \overline{B} \\ \uparrow \quad \uparrow \\ \text{Change the sign} \end{array}$$

The prediction is: $\overline{A + B} = \overline{A} \cdot \overline{B}$

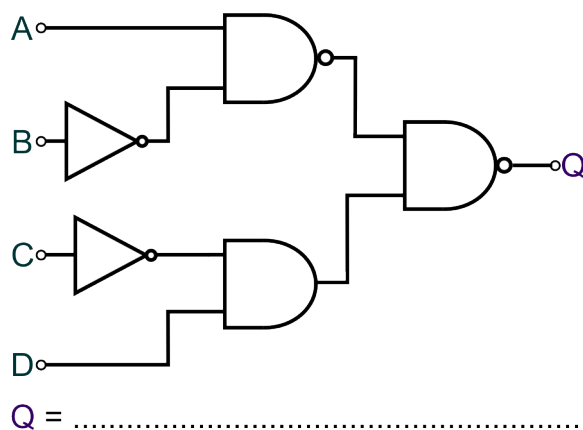
Once again, we can check this using the truth table method:

A	B	\overline{A}	\overline{B}	$\overline{A + B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

Once again, the last two columns are identical; the two logic expressions are the same and so showing that the prediction is correct.

Now we test these skills by looking at another example.

Example 3: Consider the following logic circuit.



Using the processes we looked at earlier, develop a Boolean expression for this logic circuit.

Solution: The correct Boolean expression is: $Q = \overline{\overline{A.B}} . \overline{C.D}$

Using de Morgan's theorem, we can try to simplify this expression.
As a general 'rule of thumb' start from the top and work downwards.

Step 1: Break the top 'bar' between the two brackets and change the sign, to give:

$$Q = \overline{\overline{A.B}} + \overline{C.D}$$

The first term has a double 'bar' over it, meaning that it is inverted and then inverted again, returning it to its original state. Therefore, this double inversion can be removed:

$$Q = (A.B) + \overline{C.D}$$

Step 2: Break the 'bar' over the second term and change the sign, to give:

$$Q = (A.B) + (\overline{\overline{C}} + \overline{\overline{D}})$$

Again, the double inversion over **C** can be removed, giving the final expression:

$$Q = (A.B) + (C + \overline{D})$$

$$Q = A.B + C + \overline{D}$$

With more complicated expressions, further simplification might be needed using the rules of Boolean algebra discussed earlier.

Exercise 1.5

Simplify the following expressions as much as possible:

1. $Q = \overline{\overline{A}.B}$

.....

.....

.....

2. $Q = \overline{A+B} + \overline{A}$

.....

.....

.....

3. $Q = \overline{A+B.C}$

.....

.....

.....

4. $Q = \overline{\overline{A+B}.(\overline{A.B})} + \overline{A}.B$

.....

.....

.....

.....

.....

5. $Q = \overline{\overline{\overline{A.C.B}.D}} + \overline{\overline{C.D}}$

.....

.....

.....

.....

.....

.....

.....

.....

Creating a Logic Circuit Diagram from a Boolean Equation

A written specification or a truth table can be used to develop a Boolean expression for the system. From that, a logic circuit diagram can be designed.

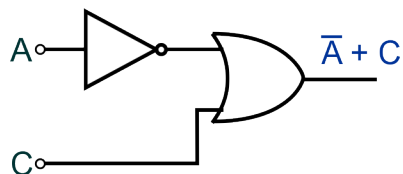
The rule of thumb is to start with any bracketed terms, then AND and finally OR functions.

The following example should make this clearer.

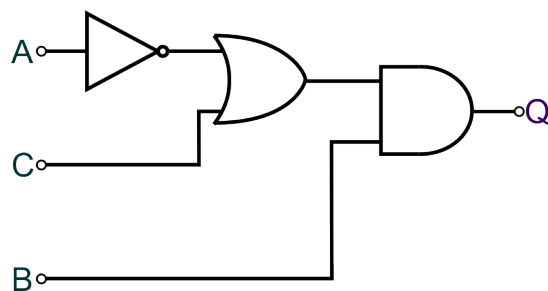
Example 1: Draw the logic circuit diagram for the following Boolean expression:

$$Q = B \cdot (\bar{A} + C)$$

First, create the logic for the bracketed term:



Then connect its output to an AND gate, with B as the other input:



Exercise 1.6

For each of the following, draw the corresponding logic circuit diagram:

1. $Q = A.\bar{B} + B.C$

2. $Q = \bar{C}.\bar{B}.A + \bar{B}.\bar{A}$

3. $Q = \overline{A+B} + \bar{A}.(C+B)$

3. Karnaugh maps

Learning Objectives:

At the end of this topic you should be able to:

- draw a Karnaugh map for a logic system with up to four inputs;
- use it to minimise the number of gates required.

Karnaugh maps offer another method of simplification for Boolean expressions. They represent the contents of a truth table as a grid or 'map'.

For example, the truth table for an OR gate:

Inputs		Output
B	A	Q
0	0	0
0	1	1
1	0	1
1	1	1

is represented as the following Karnaugh map:

		A	
		0	1
B	0	0	1
	1	1	1

Each square or cell in the Karnaugh map corresponds to a cell in truth table, as shown below:

Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	

		A	
		0	1
B	0		
	1		

Here are two examples of using this guide:

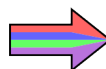
The AND gate:

Inputs		Output
B	A	Q
0	0	0
0	1	0
1	0	0
1	1	1

		A	
		0	1
B	0	0	0
	1	0	1

The NOR gate:

Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	



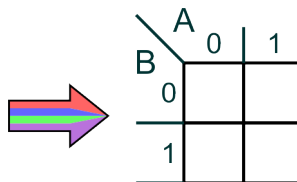
		A	
		0	1
B	0	1	0
	1	0	0

Exercise 1.7

Complete the truth table and Karnaugh map for the following gates:

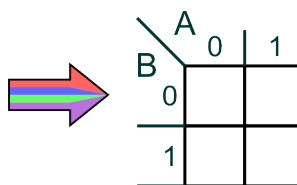
XOR gate:

Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	



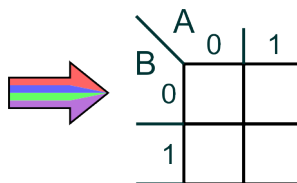
NAND gate

Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	



XNOR gate

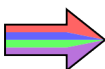
Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	



The previous section demonstrates how to fill in a Karnaugh map from a truth table. The next technique is to obtain a Boolean expression from a Karnaugh map.

For this, we take an alternative view of the Karnaugh map, using Boolean algebra:

Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	



		A	0	1
B	0	$\bar{A}.\bar{B}$	$A.\bar{B}$	
	1	$\bar{A}.B$	$A.B$	

This shows that each cell has a related Boolean expression.

For example:

The diagrams show the truth table and corresponding map for a logic system.

Inputs		Output
B	A	Q
0	0	0
0	1	1
1	0	0
1	1	0

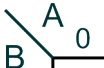
		A	0	1
B	0	0	1	
	1	0	0	

These do not match any of our standard logic gates, however, using our alternative view of the Karnaugh map, we can extract the Boolean equation straight from the cell that contains a logic 1, giving:

$$Q = A.\bar{B}$$

In this simple case, we could have obtained this result directly from the truth table, without having to draw the Karnaugh map. However, the map approach has advantages when the system becomes more complex. For example:

Inputs		Output
B	A	Q
0	0	1
0	1	1
1	0	0
1	1	1



		A	0	1
B	0	1	1	
	1	0	1	

From the truth table or from the map, we obtain the equation:

$$Q = \bar{A}.\bar{B} + A.\bar{B} + A.B$$

We can simplify it using Boolean algebra:

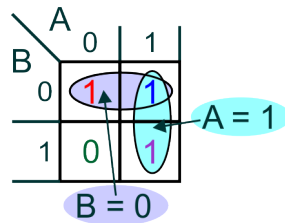
$$\begin{aligned}
 Q &= \bar{A}.\bar{B} + A.\bar{B} + A.B \\
 &= \bar{A}.\bar{B} + A.(\bar{B} + B) \\
 &= \bar{A}.\bar{B} + A \\
 &= \bar{B} + A
 \end{aligned}$$

However, Karnaugh maps allow us to do this simplification graphically.

To do this:

- identify groups of 2, 4, 8 (for bigger maps), neighbouring cells containing logic 1;
- find the term(s) common to each group;
- combine these terms using the OR operator.

Using the example above, we have two groups of two neighbouring cells that contain logic 1:



It does not matter that one cell is in both groups. The cells in the **vertical** group are in the '**A = 1**' column. This means that the term common to them is **A**.

The cells in the **horizontal** group are in the '**B = 0**' row. This means that their common term is **\bar{B}** .

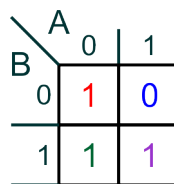
Combining them gives the result: $Q = \bar{B} + A$

This is the same result obtained using Boolean algebra.

The advantage of a Karnaugh map is that we can quickly spot common terms in a complicated expression. This will be more obvious when we deal with four input expressions.

Every cell containing logic 1 must be included in at least one group.

Quick test: Use this procedure on the following Karnaugh map:



Check your answer by simplifying the expression obtained from the truth table using the rules of Boolean algebra.

Inputs		Output
B	A	Q
0	0	
0	1	
1	0	
1	1	

The true power of Karnaugh maps becomes clearer when we look at three and four input logic systems. A three input logic system has eight possible input combinations:

Inputs			Output
C	B	A	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

This requires a bigger Karnaugh map:

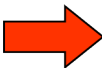
		BA			
		00	01	11	10
C	0				
	1				

- Inputs **A** and **B** are combined on the top row of the map.
- The sequence for the **BA** combinations is important. They do **not** increase numerically from one to the next. Moving from one cell to the next, only one variable must change.

Populating the Karnaugh map is carried out in a similar way as for the two input map.

In the diagrams, letters have been used to identify how the cells in the truth table correspond to those on a Karnaugh map:

Inputs			Output
C	B	A	Q
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h

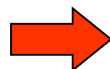


		BA			
		00	01	11	10
C	0	a	b	d	c
	1	e	f	h	g

Notice the break in alphabetical sequence that happens on the third column of the map.

Example 1: $Q = \bar{A}.\bar{B}.\bar{C} + A.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.B.C$

Inputs			Output
C	B	A	Q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



		BA			
		00	01	11	10
C	0	1	1	0	1
	1	0	0	0	1

Simplification uses the same rules as before.

		BA			
		00	01	11	10
C	0	1	1	0	1
	1	0	0	0	1

B = 0 AND C = 0
B = 1 AND A = 0

The cells in the vertical **group** are in the '**B = 1, A = 0**' column. This means that the term common to them is $\bar{A}.B$.

The cells in the horizontal **group** are in the '**C = 0, B = 0**' row. This means that their common term is $\bar{B}.\bar{C}$.

Combining them gives the result: $Q = \bar{B}.\bar{C} + \bar{A}.B$

Now compare this to simplification using Boolean algebra:

$$\begin{aligned}
 Q &= \bar{A}.\bar{B}.\bar{C} + A.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + \bar{A}.B.C \\
 &= \bar{B}.\bar{C}(\bar{A} + A) + \bar{A}.B(\bar{C} + C) \\
 &= \bar{B}.\bar{C} + \bar{A}.B
 \end{aligned}$$

Example 2: $Q = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C + \bar{A}B\bar{C} + A\bar{B}C$

The truth table for this equation and the corresponding Karnaugh map follow:

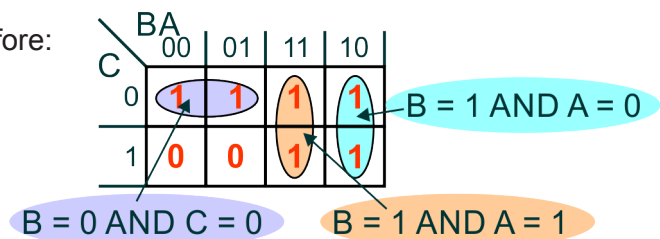
Inputs			Output
C	B	A	Q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

C \ BA	00	01	11	10
0	1	1	1	1
1	0	0	1	1

There are several ways to create groups for this Karnaugh map.

We could create three groups of two cells, as before:
This gives us the Boolean expression:

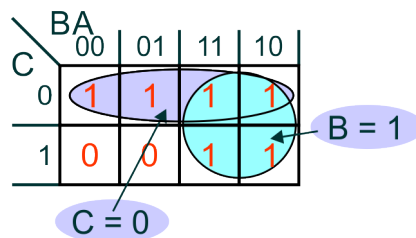
$$Q = \bar{B}\bar{C} + A\bar{B} + \bar{A}B$$



This is not the simplest solution.
Boolean algebra will simplify it further:

$$\begin{aligned} Q &= \bar{B}\bar{C} + A\bar{B} + \bar{A}B \\ &= \bar{B}\bar{C} + B\cdot(A + \bar{A}) \\ &= \bar{B}\bar{C} + B \\ &= \bar{C} + B \end{aligned}$$

To obtain this directly from the Karnaugh map, use groups of four cells rather than groups of two:



The cells in the **group** on the right are in the four cells where '**B = 1**'.
This means that the term common to them is **B**.

The cells in the horizontal **group** are in the '**C = 0**' row.
This means that their common term is **\bar{C}** .

Combining them gives the result: $Q = \bar{C} + B$

This demonstrates another rule for working with Karnaugh maps:

Use the largest groups possible!

Exercise 1.8

A design for a logic system generates the following Boolean equation:

$$Q = A.\overline{B}.\overline{C} + A.B.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.C + A.B.C$$

(a) Complete the truth table for this equation:

Inputs			Output
C	B	A	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(b) Complete the corresponding Karnaugh map:

		BA			
		00	01	11	10
C	0	0	1	1	0
	1				

(c) Use the Karnaugh map to simplify the Boolean equation:

.....

Another feature of Karnaugh maps

Suppose that a logic circuit has the following Karnaugh map:

C \ BA	00	01	11	10
	0	1	1	0
0	1	0	0	1
1	1	0	0	1

As it stands, there are two groups of two cells, resulting in the Boolean equation: $Q = \bar{A}.\bar{B} + \bar{A}.B$

C \ BA	00	01	11	10
	0	1	1	0
0	1	0	0	1
1	1	0	0	1

With Boolean algebra, this can be simplified to just $Q = \bar{A}$

To obtain this result directly from the map, we imagine it rolled around so that the two ends meet.

This allows a group of four cells,

where the only common factor is that $A = 0$.

In other words, this is the \bar{A} group.

C \ BA	00	01	11	10
	0	1	1	0
0	1	0	0	1
1	1	0	0	1

$A = 0$

The result, as before: $Q = \bar{A}$

The rule:

Use the largest groups possible, including those that wrap around at the edges!

Four input logic systems, giving sixteen possible combinations of inputs, generating even bigger truth tables and Karnaugh maps:

As before, letters show how the cells in the truth table relate to those on the Karnaugh map:

Inputs				Output
D	C	B	A	Q
0	0	0	0	a
0	0	0	1	b
0	0	1	0	c
0	0	1	1	d
0	1	0	0	e
0	1	0	1	f
0	1	1	0	g
0	1	1	1	h
1	0	0	0	i
1	0	0	1	j
1	0	1	0	k
1	0	1	1	l
1	1	0	0	m
1	1	0	1	n
1	1	1	0	o
1	1	1	1	p

DC \ BA	00	01	11	10
	00	01	11	10
00	a	b	d	c
01	e	f	h	g
11	m	n	p	o
10	i	j	l	k

Notice that the sequence breaks in both the third column and the third row of the map. Once the map is populated, the rules described so far enable us to produce a simplified Boolean expression.

Example:

Inputs				Output
D	C	B	A	Q
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Confirm that the Boolean equation for this truth table is:

$$Q = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}C\bar{D}$$

The corresponding Karnaugh map is:

		BA			
		00	01	11	10
DC	00	1	1	0	1
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	0	0

The next diagram shows one way to form groups of logic 1 cells:

		BA			
		00	01	11	10
DC	00	1	1	0	1
	01	0	1	1	0
	11	0	1	1	0
	10	0	1	0	0

D = 0 AND C = 0 AND A = 0
 B = 0 AND A = 1
 C = 1 AND A = 1

The result: $Q = A.C + A.\bar{B} + \bar{A}.\bar{C}.\bar{D}$

Note:

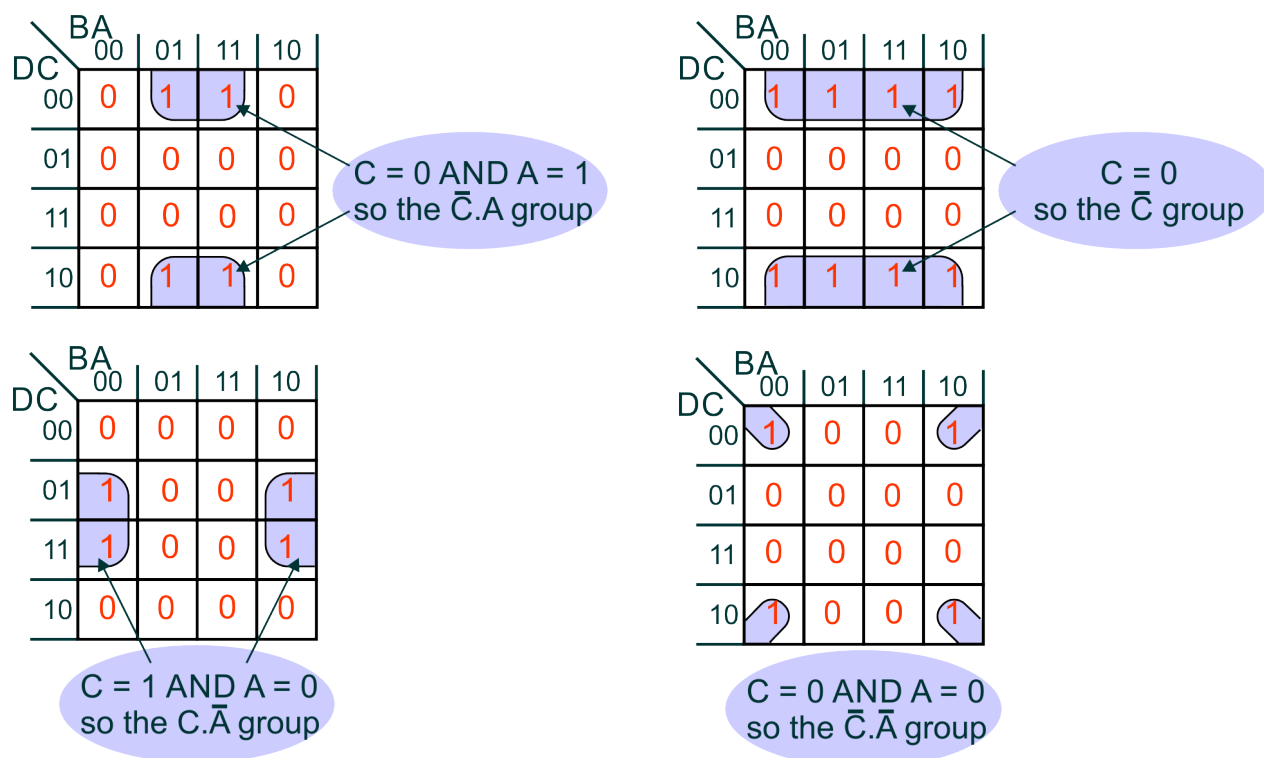
- all cells containing logic 1 are placed in groups;
- one group wraps around the map to give a small group of two cells;
- two cells appear in two groups (the groups of four cells). This is better than having a group of four and a group of two - work it out!

This expression was obtained from the Karnaugh map in one step.

Doing the same using Boolean algebra would require around eight stages of simplification.

As before, it is possible to create groups by 'folding the map'.

The following diagrams show examples:



Exercise 1.9

The following truth table specifies the behaviour of a logic system.

Inputs				Output
D	C	B	A	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

(a) Complete the Karnaugh map for this system:

		BA			
		00	01	11	10
DC	00				
	01				
	11				
	10				

(b) Use it to produce the simplest possible Boolean equation for the output **Q** of the logic system in terms of inputs **A**, **B**, **C** and **D**.

(Hint: you should be able to find three groups in the resulting Karnaugh map.)

.....

A word of caution - with Karnaugh maps, the way the groups are formed determines the resulting expression. This may lead to a number of **different correct** answers.

To illustrate this consider the following example:

A particular design has produced the following Karnaugh map:

		BA			
		00	01	11	10
DC	00	0	1	0	0
	01	0	1	1	1
	11	0	0	0	1
	10	1	1	0	1

Here are two different ways to group the cells and the answers they produce:

		BA			
		00	01	11	10
DC	00	0	1	0	0
	01	0	1	1	1
	11	0	0	0	1
	10	1	1	0	1

$$Q = A.\bar{B}.\bar{D} + B.C.\bar{D} + \bar{A}.B.D + \bar{B}.\bar{C}.D$$

		BA			
		00	01	11	10
DC	00	0	1	0	0
	01	0	1	1	1
	11	0	0	0	1
	10	1	1	0	1

$$Q = A.\bar{B}.\bar{C} + A.C.\bar{D} + \bar{A}.B.C + \bar{A}.\bar{C}.D$$

They produce different but equally correct answers.

Bear this in mind when tackling the following problems.

You may obtain different but **equally correct** solutions.

Exercise 1.10

For each of the following, produce the truth table and from it draw a Karnaugh map. Use the map to simplify the equations.

1. $Q = \bar{A}\bar{B}\bar{C} + A.B.\bar{C} + A.\bar{B}.C + A.B.C$

Inputs			Output
C	B	A	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

		BA			
		00	01	11	10
C	0				
	1				

$Q = \dots\dots\dots$

2. $Q = \bar{A}.B.\bar{C}.\bar{D} + A.\bar{B}.C.\bar{D} + \bar{A}.B.C.\bar{D} + A.B.C.\bar{D} + \bar{A}.B.\bar{C}.D + A.\bar{B}.C.D + \bar{A}.B.C.D + A.B.C.D$

Inputs				Output
D	C	B	A	Q
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

		BA			
		00	01	11	10
DC	00				
	01				
	11				
	10				

$Q = \dots\dots\dots$

3. $Q = A.\bar{B}.\bar{C} + \bar{A}.B.\bar{C} + A.B.\bar{C} + A.\bar{B}.C + A.B.C$

Inputs			Output
C	B	A	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

C \ BA	00	01	11	10
0				
1				

Q =

4. $Q = \bar{A}.\bar{B}.\bar{C}.\bar{D} + A.\bar{B}.\bar{C}.\bar{D} + \bar{A}.B.\bar{C}.\bar{D} + A.B.\bar{C}.\bar{D} + \bar{A}.\bar{B}.\bar{C}.D + \bar{A}.B.\bar{C}.D + A.\bar{B}.C.D$

Inputs				Output
D	C	B	A	Q
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

DC \ BA	00	01	11	10
00				
01				
11				
10				

Q =

We have looked at the processes involved in completing Karnaugh maps for two, three and four input logic systems from their truth tables.

However, what if we are not given the truth table but just a Boolean expression to simplify?

We now look at how to create the Karnaugh map directly from the Boolean equation.

This is not as difficult as it might sound as long as we remember some simple rules:

In a Karnaugh map for a two input function:

- any term that contains two variables is represented by one cell;
- any term that contains one variable is represented by two cells.

In a Karnaugh map for a three input function:

- any term that contains three variables is represented by one cell;
- any term that contains two variables is represented by two cells;
- any term that contains one variable is represented by four cells.

In a Karnaugh map for a four input function:

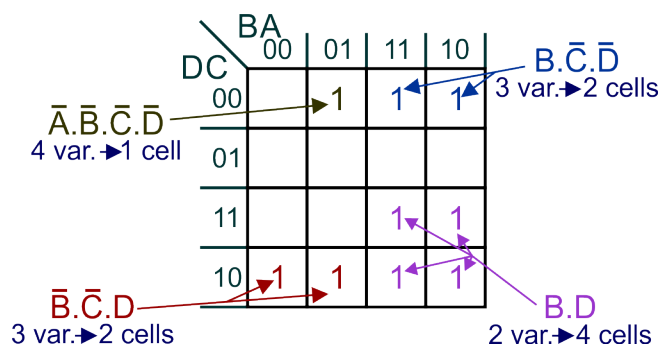
- any term that contains four variables is represented by one cell;
- any term that contains three variables is represented by two cells;
- any term that contains two variables is represented by four cells;
- any term that contains one variable is represented by eight cells.

Here are two examples:

Example 1: Simplify the following Boolean Expression

$$Q = \bar{A}.\bar{B}.\bar{C}.\bar{D} + \bar{B}.\bar{C}.D + B.\bar{C}.\bar{D} + B.D$$

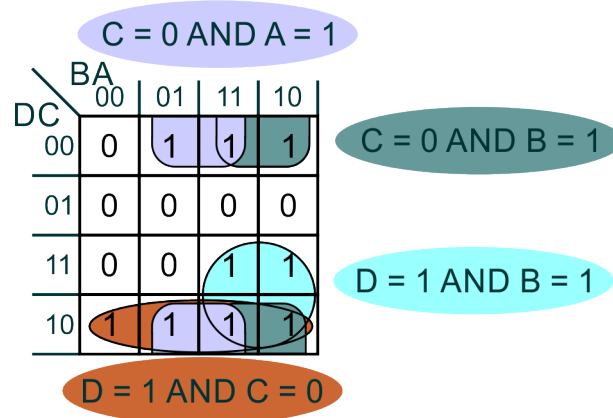
Take each term individually and put a '1' in the corresponding cell(s) of a blank Karnaugh map:



Finally, fill in the remaining cells with '0's to complete the map:

		BA			
		00	01	11	10
DC	00	0	1	1	1
	01	0	0	0	0
	11	0	0	1	1
	10	1	1	1	1

Create groups, as large as possible, to include every cell containing '1' at least once.



Some cells, for example the right-hand pair on the bottom row, appear in a number of groups.

The final equation: $Q = A.\bar{C} + B.\bar{C} + B.D + \bar{C}.D$

Example 2: Simplify the following Boolean equation:

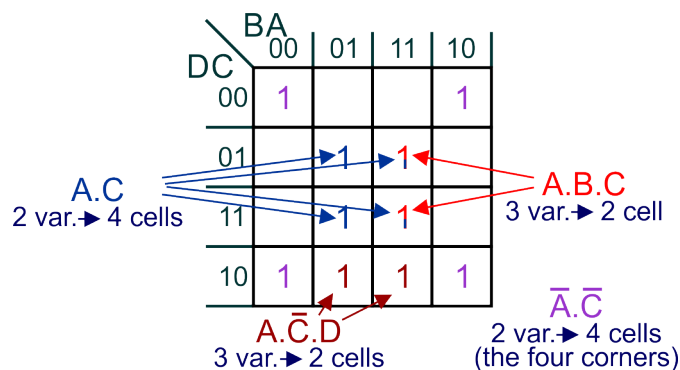
$$Q = \overline{(A + C)} + A.B.C + A.\bar{C}.D + A.C$$

This example includes an inverted function, a NOR function. This is difficult to put into a Karnaugh map directly. It is easier to apply de Morgan's theorem first, to get rid of the overall inversion.

Applying de Morgan's theorem: $Q = \overline{A + C}$
 $= \bar{A}.\bar{C}$

The equation becomes: $Q = \bar{A}.\bar{C} + A.B.C + A.\bar{C}.D + A.C$

As before, the map is built by taking each term and putting a '1' in the corresponding cell(s):



Then, the remaining cells have '0's added to complete the map:

DC \ BA	BA			
	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

Groups are created to include every cell containing '1' at least once:

	BA 00	01	11	10
DC 00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

$C = 0 \text{ AND } A = 0$
 $C = 1 \text{ AND } A = 1$
 $D = 1 \text{ AND } C = 0$

The resulting Boolean equation: $Q = \bar{A}.\bar{C} + \bar{C}.D + A.C$

Exercise 1.11

Simplify the following Boolean equations using a Karnaugh map:

1. $Q = \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB + BC$

C \ BA	00	01	11	10
0				
1				

2. $Q = \bar{A}\bar{B}\bar{C} + A.B.\bar{C} + A\bar{B} + \bar{A}\bar{C}$

C \ BA	00	01	11	10
0				
1				

3. $Q = A\bar{B}\bar{D} + A.B.\bar{C}\bar{D} + A.B.C + B.\bar{C}.D + \bar{A}.B.C.D$

DC \ BA	00	01	11	10
00				
01				
11				
10				

4. $Q = \bar{A}\bar{B}\bar{C}\bar{D} + A.B.\bar{D} + \overline{\bar{A}\bar{B}} + \bar{A}\bar{B}.C.\bar{D}$
[Remember to split the NAND function]

DC \ BA	00	01	11	10
00				
01				
11				
10				

4. NAND Gate Logic

Learning Objectives:

At the end of this topic you should be able to:

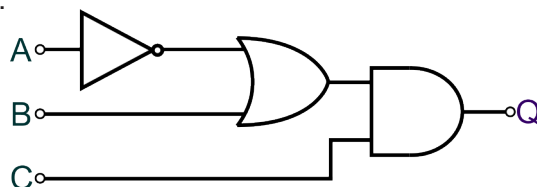
- use combinations of NAND gates to perform other logic functions;
- implement a logic system using only NAND gates;
- recognise redundant gates in a logic system constructed only from NAND gates.

The inverted gates, NAND and NOR, are special. The function of all other gates can be made from combinations of these gates. In this course, only NAND gate combinations are discussed.

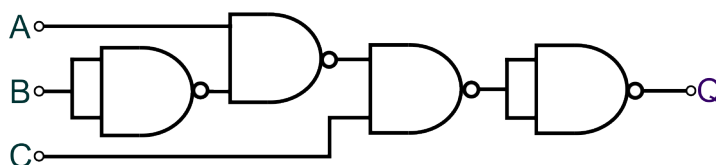
Why do this?

Look at the following two ways to make the same logic system:

1. Using a mixture of gates:



2. Using only NAND gates:



The second system has more logic gates than the first. However, the first system has three different types of gates, NOT, OR and AND.

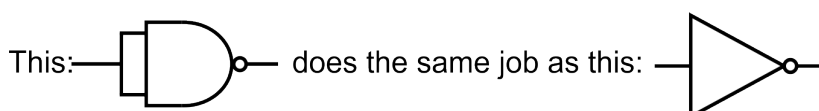
Logic gate packages contain a number of individual logic gates. For example, the 4049 hex inverting buffer IC contains six NOT gates. The 4081 IC contains four AND gates.

To construct the first system requires three different logic ICs and most of the logic gates on them would not be used.

Even though the second system requires four logic gates, they are all the same type, NAND gates. A 4011 IC contains four NAND gates and so the system can be built from just one IC, a significant cost saving over the first system - around one-third of the price.

Now that we know why NAND gate logic is used, let's look at how to carry out this procedure and create combinations of NAND gates that perform like the other logic gates.

The NOT gate:



The NAND gate has its two inputs connected together so both must sit at the same logic level.

The truth table for a two input NAND gate has four possible input combinations. However, with the inputs connected together, only two are possible - either both inputs at logic 0 or both at logic 1.

The result - the Q output is the inverse of the input.

The arrangement behaves like a NOT gate and is often referred to as a “NAND inverter”.

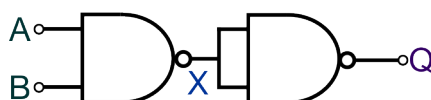


B	A	Q
0	0	1
0	1	1
1	0	1
1	1	0

These cannot happen

The AND gate:

This is the inverse of a NAND gate, and so is a NAND gate followed by a NAND inverter.

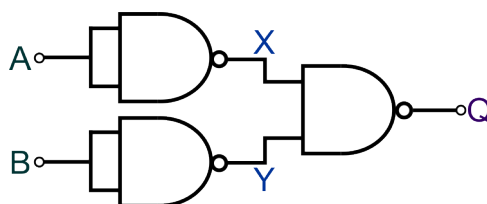


Confirm that this arrangement behaves like an AND gate by completing the truth table.

Inputs			Output
B	A	X	Q
0	0		
0	1		
1	0		
1	1		

The OR gate:

The OR gate requires three NAND gates, two connected as NOT gates.

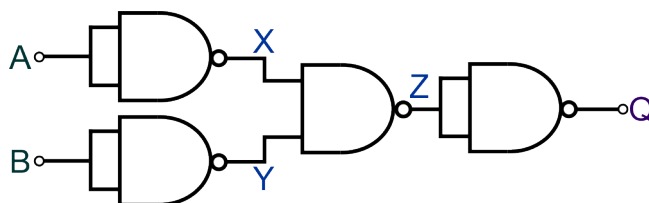


Confirm that this arrangement behaves like an OR gate by completing the truth table.

Inputs				Output
B	A	X	Y	Q
0	0			
0	1			
1	0			
1	1			

The NOR gate:

The NOR gate is the inverse of the OR gate, so needs a NAND inverter added to it:



Confirm that this arrangement behaves like a NOR gate by completing the truth table:

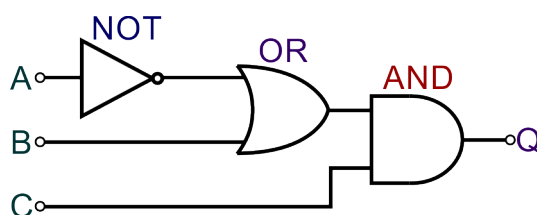
Inputs					Output
B	A	X	Y	Z	Q
0	0				
0	1				
1	0				
1	1				

Converting logic systems to NAND gates

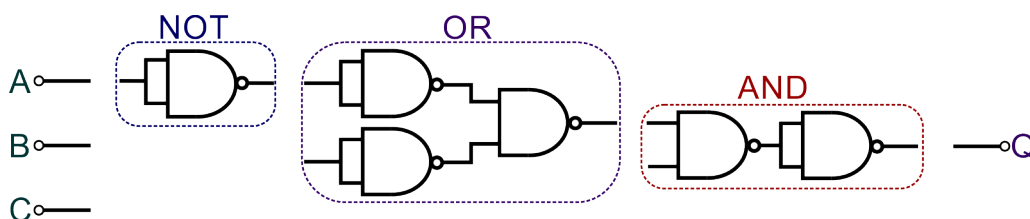
Each gate is replaced by its NAND equivalent, connected up in the same way.

Example 1:

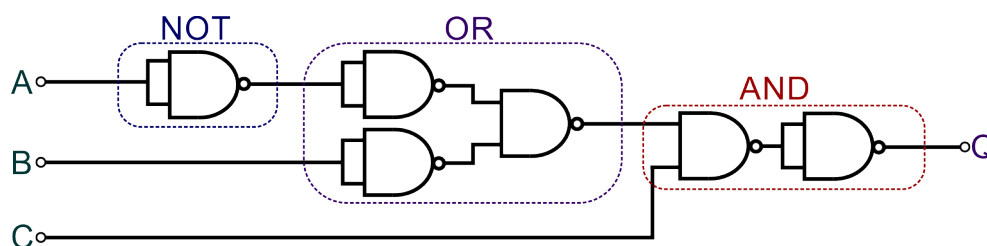
Convert the following logic system into NAND gates only.



Stage 1: Replace each gate with its NAND equivalent.
(drawing a box around each one allows you to check the conversion more easily).



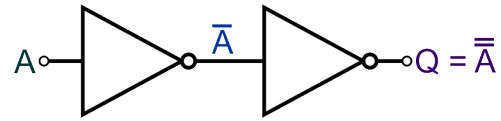
Stage 2: Connect the equivalent circuits together.



There is a further simplification to make:

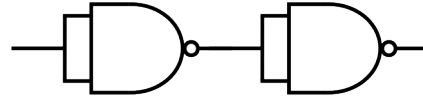
Stage 3: Look for redundant gates!

Where one NOT gate is followed by another, they cancel each other out.



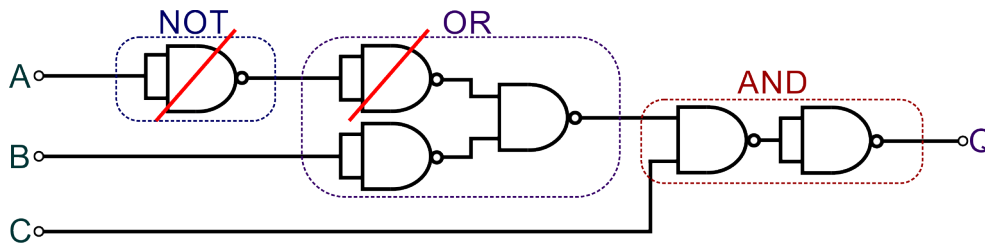
As the diagram shows, the final output, $\bar{\bar{A}}$, is identical to the original input, A.

The same arrangement, in NAND gates, is:



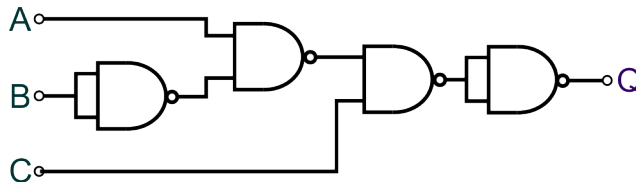
This is known as a double inversion and both NAND gates can be removed.

In the system developed above, two such NAND gates follow input A.



In an examination, make it absolutely clear what you are doing by identically marking the redundant pair of gates. Where there are more than one pair of redundant gates, choose a different marking method for each pair.

The final circuit in the example studied here is:



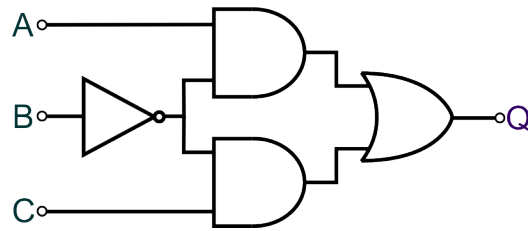
In this example, all the stages have been set out in fine detail. In practice, some of these stages can be carried out simultaneously, but take care to show the examiner what you are doing.

The process summarised:

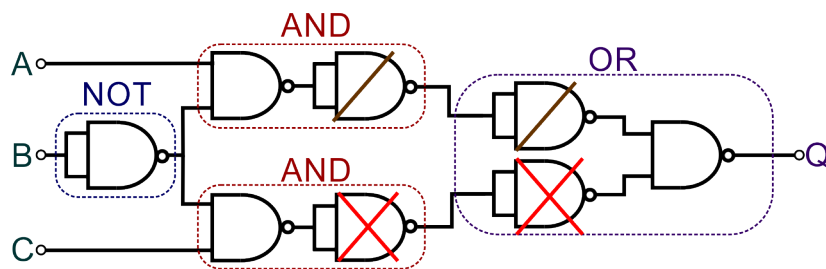
- replace each of the gates with its NAND equivalent;
- connect the equivalent NAND gate circuits as in the original;
- identify and cross out redundant pairs of gates caused by double inversions;
- **show clearly what you are doing.**

Example 2:

Convert the following logic system into NAND gates only:

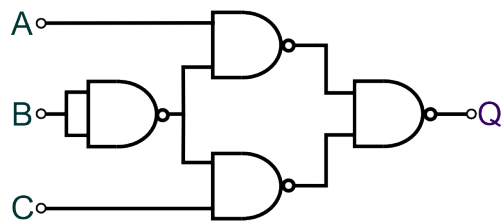


First of all, replace all gates with their NAND equivalent and connect them together. Then, check for any pairs of redundant gates and identify them.



Notice the way in which the two pairs of redundant gates are identified.

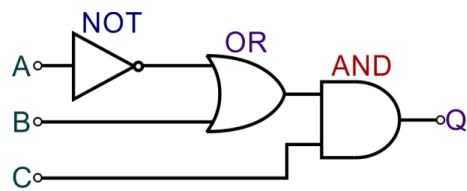
The final circuit is:



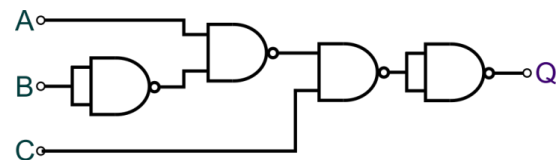
Investigation 1.3

Set up the following logic circuits

- (a) Complete the truth table for each circuit.
(If you use a simulation program you can use the built in inputs and outputs. If you build the circuits on breadboard you will need to use switches, pull-down resistors and a LED.)



C	B	A	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



C	B	A	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

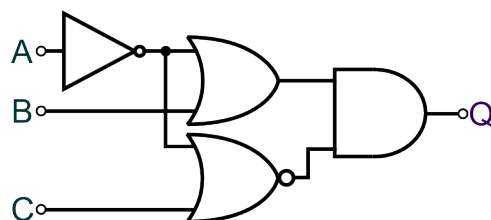
- (b) You should have found the Q outputs are identical. Can you think of a reason for this?
If you are not sure look back to the example on page 53

.....
.....
.....

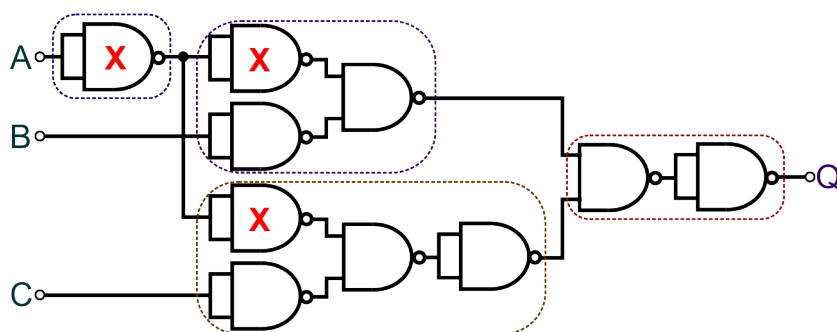
Multiple Redundancies

As we have seen, some systems can be simplified by removing redundant pairs of 'NAND inverters'. Occasionally, multiple redundancies may occur, where an input is inverted and then connected to two (or more) inverted inputs of other logic gates.

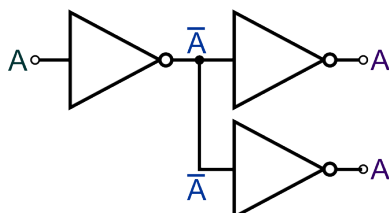
An example may make this clearer:



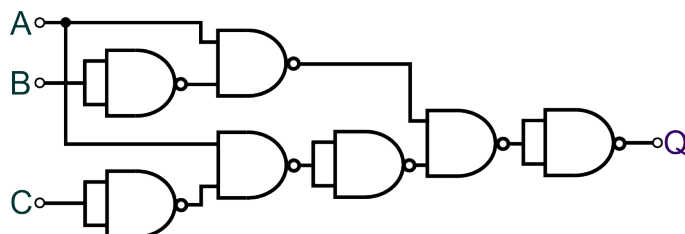
Converted to use only NAND gates, this becomes:



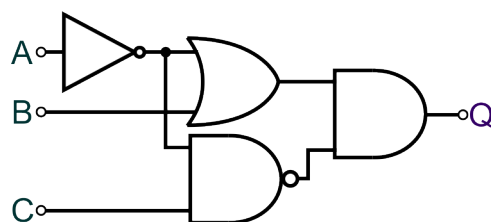
Multiple redundancy occurs in the three gates marked **X**. All three can be eliminated since the output of both right-hand gates is **A**, as the diagram below shows:



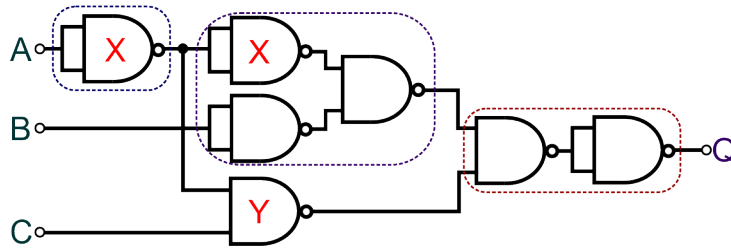
After eliminating redundant gates, the final system is:



Be careful! Look at the next example:



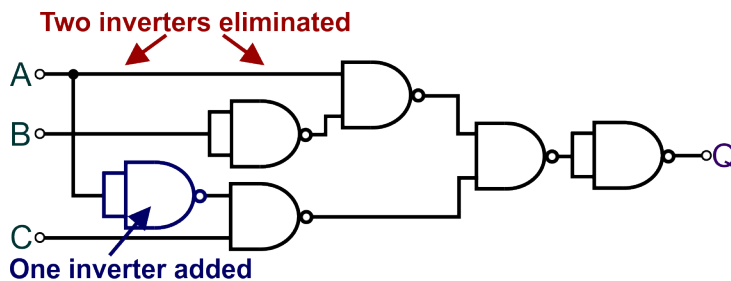
Using only NAND gates, this becomes:



At first glance, the circuit resembles the previous one. However, while the top two gates marked **X** are both NAND inverters, the bottom connection from the junction leads to one input of a NAND gate marked **Y**, not to another NAND inverter.

The top two gates cancel each other out and can be removed, but a NAND inverter is needed to provide a signal of \bar{A} to the input of the lower gate – nevertheless a net saving of one gate.

The circuit becomes:



Summary

The concept of multiple redundancies may appear complicated. It can be identified by looking at the original logic circuit containing NOT AND OR NAND and NOR gates:

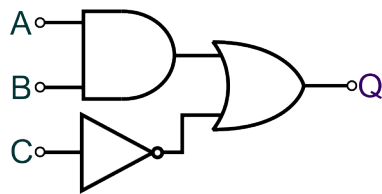
Where the output of a NOT gate is connected to two other logic gates, there is a possibility of multiple redundancy.

- Where the two other gates are OR/NOR, there will be three redundant gates
- Where one of other gates is a OR/NOR and the other one is AND/NAND, there will be two redundant gates but an additional NAND inverter will be needed
- Where the two other gates are AND/NAND there will be no redundant gates.

This idea can be extended to cover cases where the output of a NOT gate is connected to more than two other logic gates, but this will not be considered during the course.

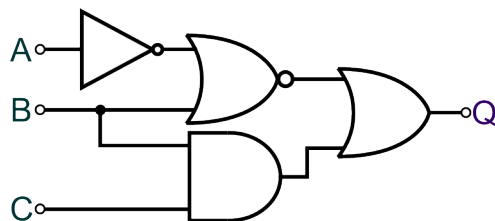
Exercise 1.12

1. (a) Redraw the following logic circuit using two input NAND gates only:



- (b) Identify any redundant gates.

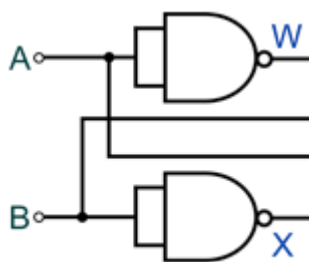
2. (a) Redraw the following logic circuit using two input NAND gates only:



- (b) Identify any redundant gates.

3. (a) Use the Boolean equation for an XOR gate $Q = \bar{A}.B + A.\bar{B}$ to draw its logic circuit based on NOT, AND and OR gates.

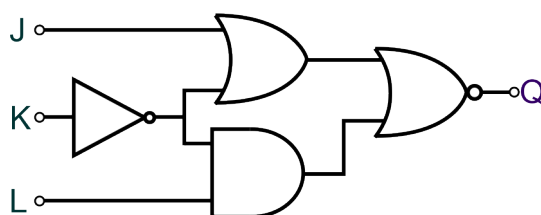
- (b) (i) Redraw the logic circuit using two input NAND gates only.
The first two gates have been provided for you.



- (ii) Cross out any redundant gates. Your final solution should contain five gates.
- (c) Label the outputs of the three right hand NAND gates with the letters **Y**, **Z** and **Q**.
Complete the truth table and confirm that this system behaves like an XOR gate.

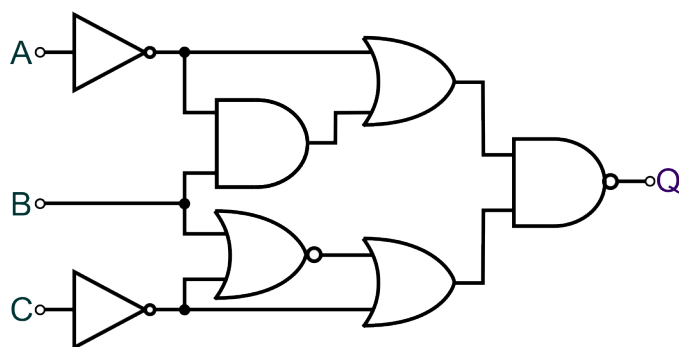
Inputs						Output Q
B	A	W	X	Y	Z	
0	0					
0	1					
1	0					
1	1					

4. (a) Redraw the following logic circuit using two input NAND gates only.



- (b) Cross out any redundant gates.

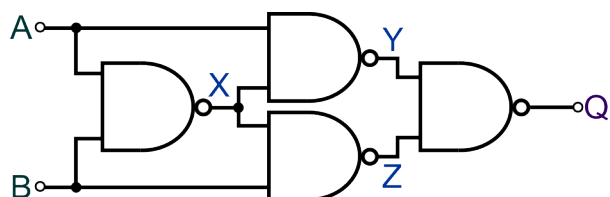
5. (a) Redraw the following logic circuit using 2 input NAND gates only.



- (b) Identify any redundant gates (there are some multiple redundancies).

5. (a) An engineer suggests that the NAND gate equivalent of an XOR gate can be designed using just four NAND gates.

The proposed equivalent is:



Complete the truth table to confirm that this behaves like an XOR gate.

Inputs					Output Q
B	A	X	Y	Z	
0	0				
0	1				
1	0				
1	1				

- (b) Use this information to draw an XNOR gate using five NAND gates.

5. Multiplexers

Learning Objectives:

At the end of this topic you should be able to:

- design and analyse a logic system with up to four inputs, using a multiplexer as a programmable logic system.

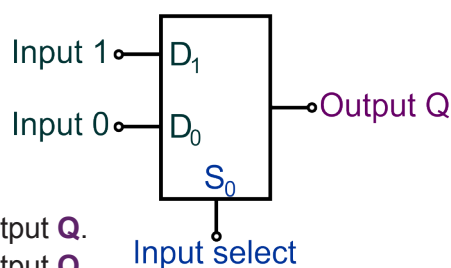
A multiplexer is a device able to transfer a signal from any one of a number of inputs to an output, using a combination of digital signals applied to its input select pins.

2:1 multiplexer

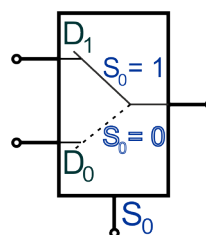
This has:

- two input terminals, D_0 and D_1 ;
- one input select terminal, S_0 ;
- one output terminal, Q .

When the input select pin is logic 0, D_0 is connected to the output Q .
When the input select pin is logic 1, D_1 is connected to the output Q .



It is rather like a rotary switch, whose position is set by the signal on the input select pin.



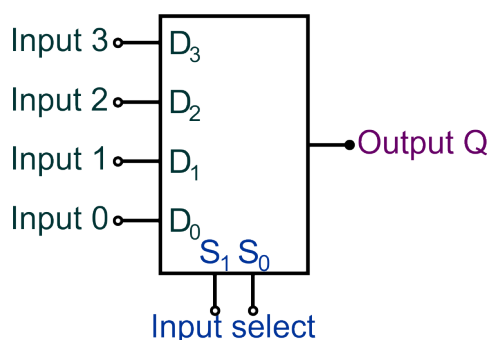
4:1 multiplexer

This has:

- four data input terminals, labelled D_0 to D_3 ;
- two input select terminals, S_0 and S_1 ;
- one output terminal Q .

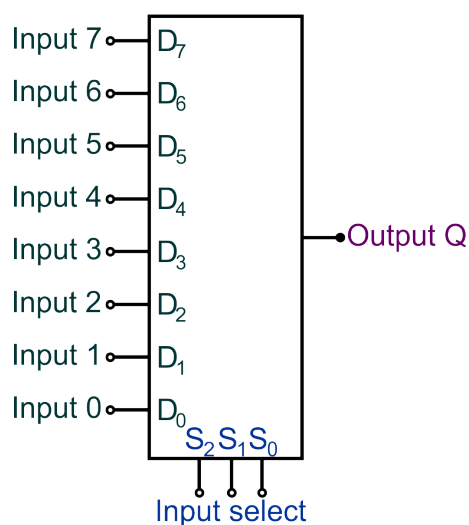
The truth table shows how the input select signals determine which data input is connected to the output.

Input select		Output
S_1	S_0	Q
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



8:1 multiplexer behaves in similar manner, described in the truth table.

Input select			Output
S_2	S_1	S_0	Q
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7



Different families of IC use different labelling schemes for the input select pins. The diagram above used the labels ' S_0 ', ' S_1 ' and ' S_2 '. Other schemes use ' S_1 ', ' S_2 ' and ' S_4 ' respectively.

In examination questions, these pins will always be labelled with A, B, C, etc.

Sometimes, the **data pins** are labelled with the series ($D_0 \dots$) starting at the top of the diagram.

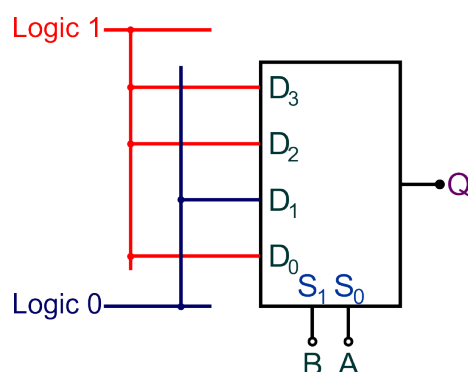
Multiplexers are commonly used in communication systems. However they can be used to simplify logic system design, as the following example shows:

Example 1: Show how a 4:1 multiplexer can be used to generate the following logic function:

Inputs		Output
B	A	Q
0	0	1
0	1	0
1	0	1
1	1	1

The solution requires just one IC, the multiplexer.

Its data inputs, D_0 to D_3 , are connected to either logic 0 or logic 1, reflecting the contents of the truth table. Inputs **A** and **B** are connected to the select input pins. S_0 and S_1 . They then determine which data input signal is transferred to the output **Q**:



When:

- **B = 0** and **A = 0**:
 - D_0 is connected to output **Q**, sending logic 1 to the output.
- **B = 0** and **A = 1**:
 - D_1 is connected to output **Q**, sending logic 0 to the output.
- **B = 1** and **A = 0**:
 - D_2 is connected to output **Q**, sending logic 1 to the output.
- **B = 1** and **A = 1**:
 - D_3 is connected to output **Q**, sending logic 1 to the output.

This demonstrates how a multiplexer can be used to generate a logic function. It offers an alternative approach to simplifying with Boolean algebra or using a Karnaugh map.

Example 2: Design a logic system that obeys the truth table:

Inputs			Output
C	B	A	Q
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The Karnaugh map is:

		BA			
		00	01	11	10
C	0	1	0	1	0
	1	0	1	0	1

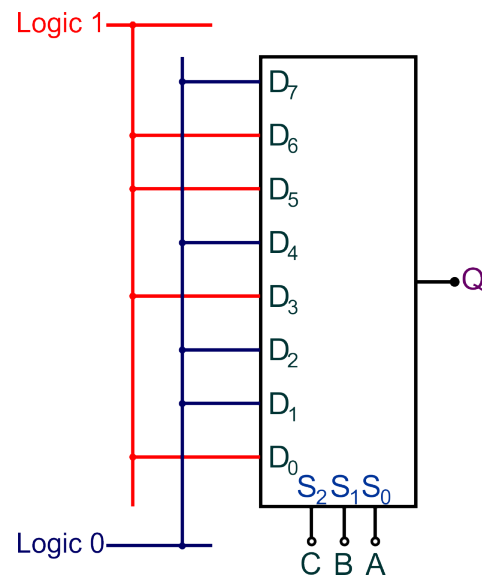
There are no groups of logic 1 cells to allow simplification.

Simplifying the equivalent Boolean expression gives:

$$\begin{aligned}
 Q &= \bar{A}.\bar{B}.\bar{C} + A.\bar{B}.C + A.B.\bar{C} + \bar{A}.B.C \\
 &= \bar{B}.(\bar{A}.\bar{C} + A.C) + B.(A.\bar{C} + \bar{A}.C) \\
 &= \bar{B}.(\overline{A \oplus C}) + B.(A \oplus C)
 \end{aligned}$$

This is not a straightforward expression. The resulting circuit requires two NOT gates, two AND gates, an OR gate and a XOR gate. Even with NAND gate reduction, it would still need nine gates and use three ICs.

However, the multiplexer solution is straightforward:



The multiplexer is useful with logic functions that are complex and require a large number of logic gates.

Why not use multiplexers for all logic functions?

The answer usually relates to cost. Multiplexers, particularly those with large number of inputs, are more expensive than logic gates.

Investigation 1.4

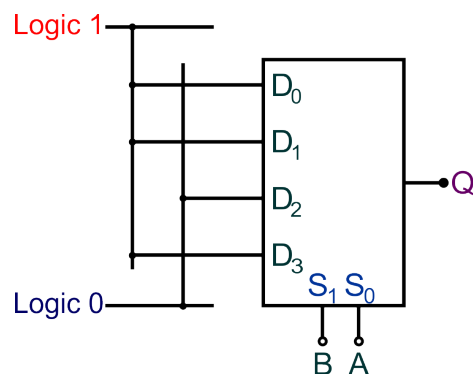
Using a circuit simulation package or breadboard, set up the circuit given as the solution to Example 2 to confirm that the truth table obtained is correct.

Note: The input pins on the 4051 IC are labelled S1, 2 and 4, not S₀, S₁ and S₂ given on the earlier diagram.

Exercise 1.13

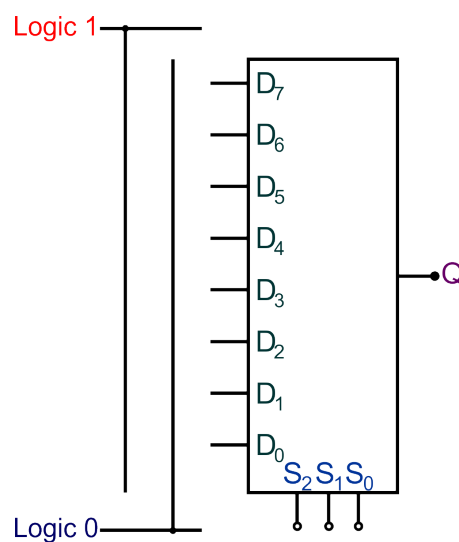
1. Show how the 4:1 multiplexer can produce the logic function shown in the truth table.

Inputs		Output
B	A	Q
0	0	1
0	1	1
1	0	0
1	1	1

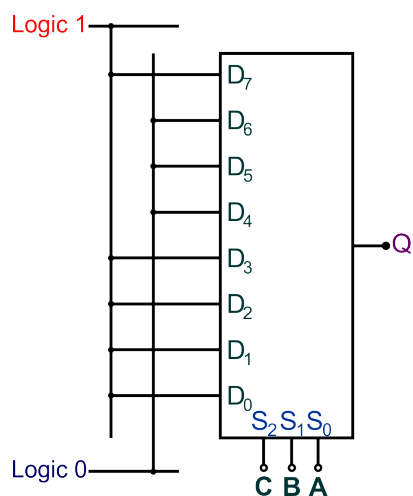


2. Complete the circuit diagram to show how an 8:1 multiplexer can generate the following logic function:

Inputs			Output
C	B	A	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



3. The diagram shows an 8:1 multiplexer used as a programmable logic system:



(a) Complete the truth table for this system:

Inputs			Output Q
C	B	A	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(b) Write down the Boolean expression for the output **Q**, in terms of **A**, **B** and **C**.

Q =

6. Interfacing analogue sensors to logic gates

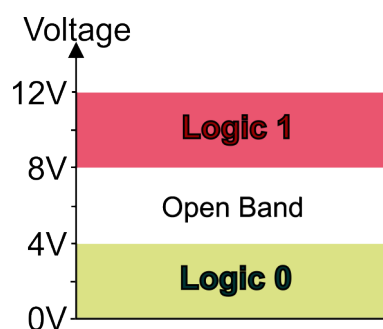
We have only dealt with digital signals being connected to the inputs of logic gates where the input was either equal to the positive supply voltage or 0 V. We will now consider how to connect an analogue signal to a logic gate.

Logic gates are designed to recognize a predetermined band of voltages as representing logic **0** and logic **1**. These bands are specific for each family of ICs as seen in the following table

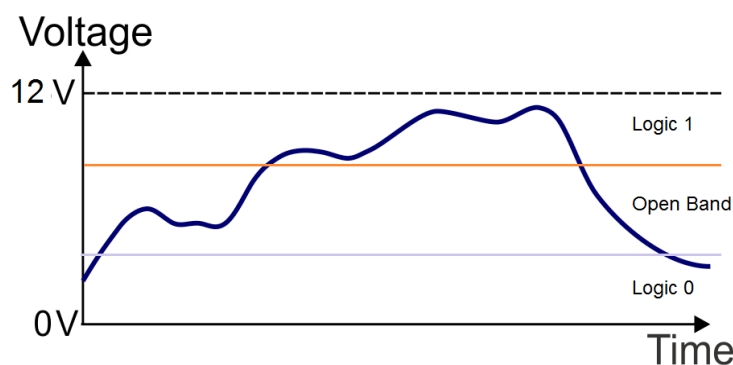
Parameter	TTL (74 series)	CMOS (4000 series)
Supply Voltage	5 V \pm 0.25 V only	3 V to 18 V
Logic 0 range	0 to 0.8 V	Below 30% of supply voltage
Logic 1 range	2.0 to 5.0 V	Above 70% of supply voltage

The intermediate levels or band of voltages below the logic 1 threshold and above the logic 0 threshold result in highly unpredictable circuit behaviour. This region between the logic 0 and logic 1 bands is sometimes referred to as the **open band**.

The diagram on the right illustrates the three bands for a CMOS logic gate connected to a 12 V power supply.



The graph below shows the graph of a typical analogue signal produced by light sensing sub-system.



Under certain light conditions the analogue signal could remain in the open band for long periods of time causing unpredictable behaviour of a logic system that the sensor is interfaced with.

The solution is to use the Schmitt inverter which was introduced in the core concepts chapter. It can be used to process or condition an analogue signal to remove the undesirable characteristics of:

- a slow rise time;
- fluctuations near the threshold

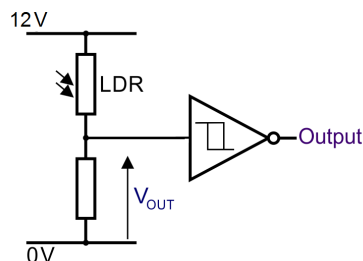
The Schmitt inverter produces a digital output which is compatible with logic systems.

For a CMOS 40106 Schmitt inverter:

- The switching threshold for a rising input signal is approximately two thirds of the supply voltage. At this point the output changes to logic 0
- The switching threshold for a falling input signal is approximately one third of the supply voltage. At this point the output changes to logic 1

Example:

The circuit on the right shows a light sensing sub-system connected to a Schmitt inverter.



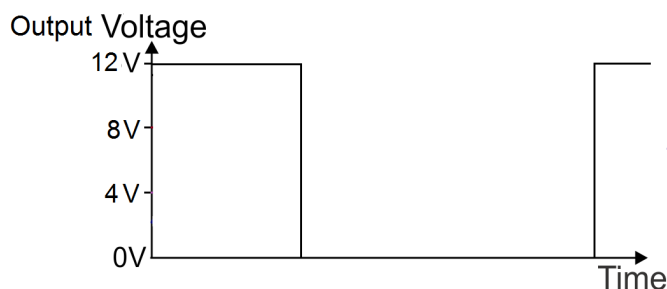
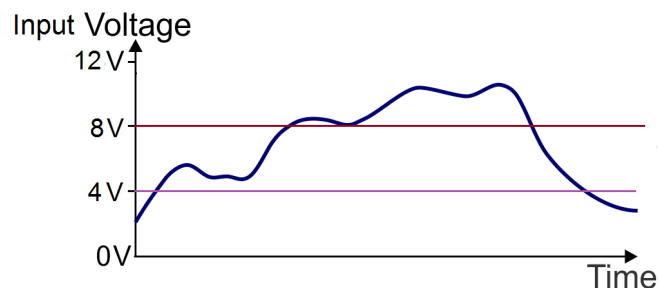
Here is the data sheet for the Schmitt inverter:

When connected to 12 V supply:

- Logic 0 = 0 V
- Logic 1 = 12 V
- The output changes from logic 1 to logic 0 when a **rising** input voltage reaches 8 V
- The output changes from logic 0 to logic 1 when a **falling** input voltage reaches 4 V

The signal produced by the light sensing sub-system is shown on the first set of axes.

Use the second set axes to draw the resulting output signal produced when the Schmitt inverter is used to convert the analogue input into a digital output.



Solution:

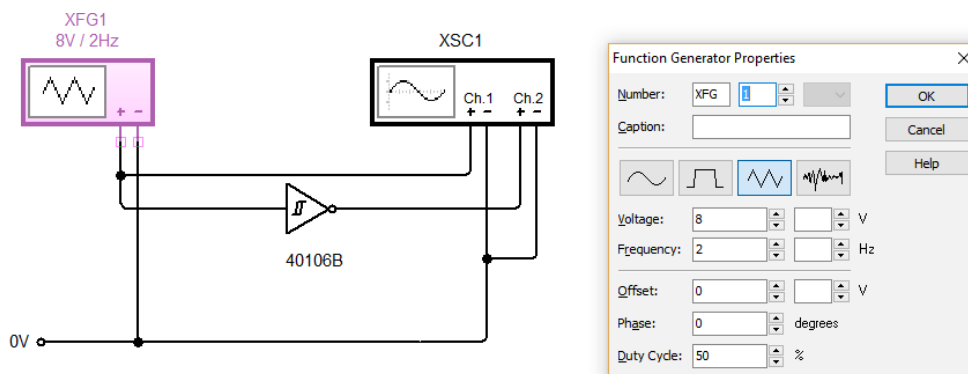
When the input voltage rises to 8 V the output instantly falls to zero.

Subsequent changes in input voltage are ignored until it drops to 4 V. At this point the output changes to 12 V.

Investigation 1.5

Set up the following circuit with the function generator outputs adjusted to the settings shown to produce a triangular waveform.

If you are setting this circuit up on Circuit Wizard ensure the voltage setting for the CMOS 40106 IC is set to 9 V. Go to *Project* → *Simulation* → *Power supply* and set the voltage to 9 V.



Set the oscilloscope time base to 100 ms.

- (a) Study the oscilloscope display and determine the switching thresholds for the Schmitt inverter

.....

- (b) Adjust the function generator outputs to the settings shown to produce a sinusoidal waveform.

Sketch the input and output waveforms obtained on the graph grid below. Label both axes with appropriate scales and use a different colour to represent the input and output waveforms.

