Module 3 – Boolean algebra and mapping techniques

Module 3 overview

Boolean algebra and mapping techniques are essential tools in combinational logic design. This can be seen in the following concept map.

Combinational logic (concept map)

Objectives

At the completion of this module you will be able to:

- demonstrate a working knowledge of Boolean algebra by minimising logic equations
- understand mapping techniques and their use in minimising logic equations
- identify ‘**Don’t Care**’ conditions and use them to minimise logic equations.

3.1 Boolean algebra

In the mid 1800’s, a mathematical proposition was presented which simplified and allowed manipulation of algebraic expressions. The presenter was an English mathematician named George **Boole** and the technique he developed was known as **Boolean Algebra**.

Its application to two-state (binary) switching circuits was proposed in 1938 by Claude **Shannon** at Massachusetts Institute of Technology. The terms ‘AND gate’ and ‘OR gate’ originated from Shannon’s paper.

Boolean algebra provides a straight forward approach to the design and simplification of electronic switching systems. This of course leads to a reduction in the total number of gates required to implement a logical operation which in turn makes the final design more reliable, less complicated and therefore less expensive to produce.

A Boolean **variable** (A) can have two possible values which can be indicated by a 0 or a 1.

i.e. A can equal 0 in some cases and A can equal 1 in others.

A logic designer would always specify which of the above 2 conditions is being used for a particular design.

The Boolean inverse of A is $\overline{A}$. The inverse of C is $\overline{C}$ etc.

A Boolean **function** is an algebraic expression containing the variables in terms appropriate to a particular design. The variables may appear in the terms in either **true** (A) or **complement** ($\overline{A}$) form.

The terms may appear in the function (or algebraic expression) in either the 'product' form (ABC) or the 'sum' form (A+B+C). Here of course the 'product' form means logical AND and the 'sum' form means logical OR. No other mathematical symbols are used in Boolean algebra.

Two basic forms of Boolean functions are of special interest because of their common occurrence and symmetrical form.

These are called **canonical** forms. They are:

- the product of sums (called **maxterm** form)
- the sum of products (called **minterm** form).

In order for either form to be canonical, **all** variables involved must appear in **all** terms of the expression in either true or complement form.

A **minterm** for three variables A, B and C is a **product** form such as: ABC or $A\overline{B}C$ etc.

A **maxterm** for the same three variables is a **sum** form such as: $A + B + C$ or $A + \overline{B} + C$ etc.

For three variables there are eight possible combinations or terms. In minterm form we have:
$\overline{A}\,\overline{B}\,\overline{C}$, $\overline{A}\,\overline{B}C$, $\overline{A}B\overline{C}$, $\overline{A}BC$, $A\overline{B}\,\overline{C}$, $A\overline{B}C$, $AB\overline{C}$, ABC .

For four variables there are sixteen possible combinations etc.

3.1.1 Boolean operations

Several relationships can be identified between Boolean variables.

Using the 'AND' function and letting A = 1 we may write:

$A \cdot A = A$ i.e. 1 and 1 equals 1 etc.

and $A \cdot \overline{\ } = A$

Similarly letting A = 0, we may write:

$A \cdot 0 = 0$

$A \cdot 1 = A$

The same analysis applied to the 'OR' function yields:

$A + A = A$

$A + \overline{A} = 1$

$A + 0 = A$
$A + 1 = 1$

Two theorems are often used to assist in simplifying expressions. These are known as

“De Morgans” theorems.

i. $\overline{A} + \overline{B} = \overline{A\ \ B}$
ii. $\overline{A}\quad \overline{B} = \overline{A + B}$

Notice how these expressions transform an OR function to an AND function and vice versa.

From i. above we have:

$$\underbrace{\overline{A}+\overline{B}}_{\text{OR}} \quad = \quad \underbrace{\overline{A\ \ B}}_{\text{AND}}$$

function function

This is a very interesting and important property which has a good practical base. i.e. an OR gate can be used to perform an AND gate function as illustrated in module 2.4.4.

When simplifying an expression using De Morgans theorems, the line to be broken is the one immediately above the sign. Lines crossing a bracket, or an implied bracket, must first be broken outside the bracket. Variables ANDed together have implied brackets around them. Implied brackets should be inserted to avoid mistakes.

e.g. $\overline{A\ B+\overline{A}\ \overline{B}} \longrightarrow \overline{(A\ B)+(\overline{A}\ \overline{B})}$

AB and $\overline{A}\overline{B}$ have implied brackets around them so first break the line above the + symbol. The AND symbol is not required, but is implied.

i . e $\overline{\quad .\quad}\ \overline{(\ \overline{\ }\ \overline{A}\ }$

The lines above A B and $\overline{A}\,\overline{B}$ can now be broken – retain brackets. Remember, break the line change the sign.

i.e. $(\overline{A} + \overline{B})(\overline{\overline{A}} + \overline{\overline{B}})$

Note: Double bars cancel, i.e. $\overline{\overline{A}} = A$

i.e.$(\overline{A}+\overline{B})$ $(A+B)$

3.1.2 Other properties and relationships

Commutative	$A + B$	=	$B + A$
	$A \cdot B$	=	$B \cdot A$
Associative	$A + (B + C)$	=	$(A + B) + C$
	$A \cdot (B \cdot C)$	=	$(A \cdot B) \cdot C$
Distributive	$A + (B \cdot C)$	=	$(A + B) \cdot (A + C)$
	$A \cdot (B + C)$	=	$(A \cdot B) + (A \cdot C)$

Other relationships which often occur in logic design are:

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

Proof

$$\begin{aligned}
\text{LHS} &= A + \overline{A}B \\
&= A \quad 1 + \overline{A}B && \text{– using the relationships shown in section 3.1.1} \\
&= A(1 + B) + \overline{A}B \\
&= A \quad 1 + AB + \overline{A}B \\
&= A + B(A + \overline{A}) \\
&= A + B \\
&= \text{RHS}
\end{aligned}$$

2. $A + AB = A$

Proof

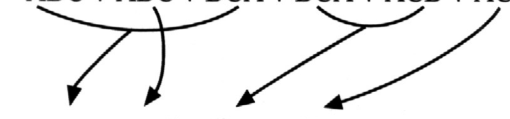
$$\begin{aligned}
\text{LHS} &= A + AB \\
&\ A(1 + B) \\
&\ A \\
&\ \text{RHS}
\end{aligned}$$

Now let us consider an example using Boolean algebra to reduce an expression.

Prove $AB + BC + \overline{A}C = AB + \overline{A}C$

$\text{LHS} = AB(C + \overline{C}) + BC(A + \overline{A}) + \overline{A}C(B + \overline{B})$ adding the missing variable since $(C + \overline{C}) = 1$

$= ABC + AB\overline{C} + BCA + BC\overline{A} + \overline{A}CB + \overline{A}C\overline{B}$



$= ABC + AB\overline{C} + \overline{A}BC + \overline{A}\overline{B}C$

$= AB(C + \overline{C}) + \overline{A}C(B + \overline{B})$

$= AB + \overline{A}C$

$= \text{RHS}$

3.2 Mapping techniques

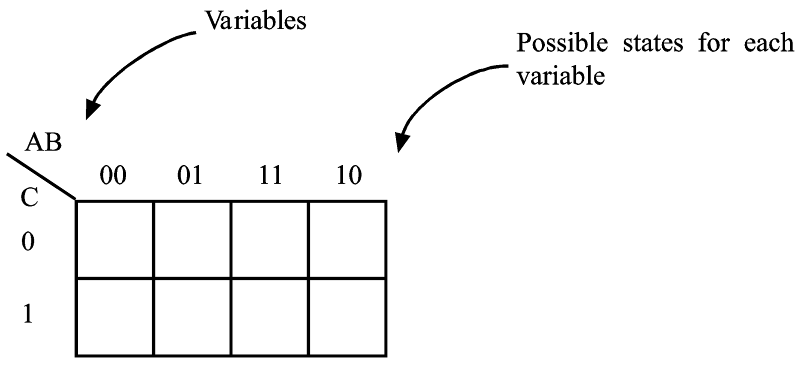
Boolean expressions may be simplified using mapping techniques instead of algebra.

This is a very powerful reduction technique resulting in minimal errors sometimes caused by mistakes in Boolean algebra application. The map generally used is called a **Karnaugh** map.

Consider the equation: $F = \overline{A}BC + AB\overline{C} + A\overline{B}\overline{C}$

This is a 3 variable (A,B,C) problem so we would use a 3 variable Karnaugh map.

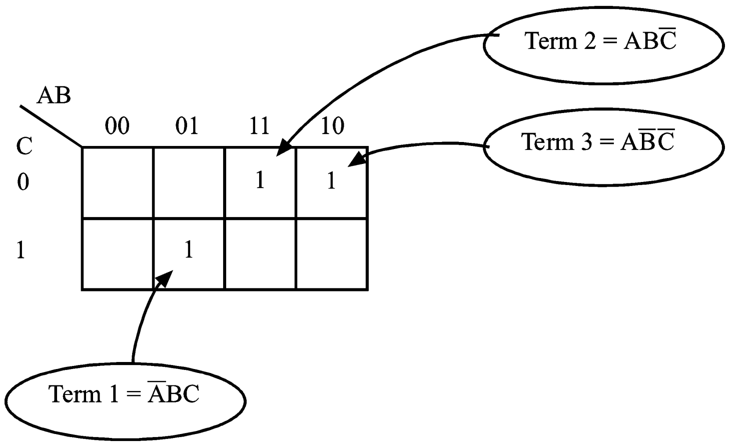
The map is



Three Variable Karnaugh Map

Note the order of possible states of AB, i.e. **00 01 11 00**

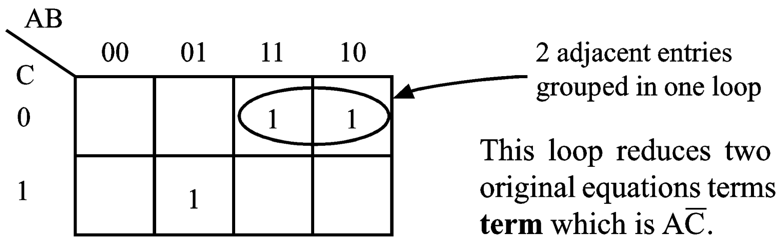
Placing the equation on the map involves the use of ‘1’s as entries on the map.


Term 2 = $AB\overline{C}$
Term 3 = $A\overline{B}\overline{C}$
Term 1 = $\overline{A}BC$
AB
C
00
01
11
10
0
1
1
1
1
1

After all terms have been entered on the map, entries are grouped together with loops, according to these basic rules:

- Grouping can only include adjacent entries.
- Grouping can only be horizontal or vertical.
- The map is considered continuous i.e. spherical.
- The number of entries grouped together **must** be a power of 2, and should be as large as possible.
- Entries may be included in more than one loop if necessary.

So our example becomes:



This loop reduces two of the original equations terms to **one term** which is $A\overline{C}$.

The variable B disappears from the term as the loop can't have B and $\overline{B}$ present at once.

The other map entry cannot be looped with anything else so it remains untouched.

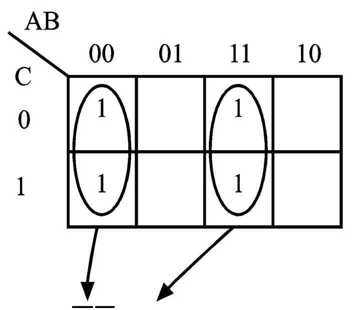
The reduced expression is then:

$$\boxed{F = \overline{A}BC + A\overline{C}}$$

Activity 3.1

Example 1

Simplify: $F = AB\overline{C} + ABC + \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C}$



$$\underline{F = \overline{A}\overline{B} + AB}$$

Check this result using Boolean algebra:

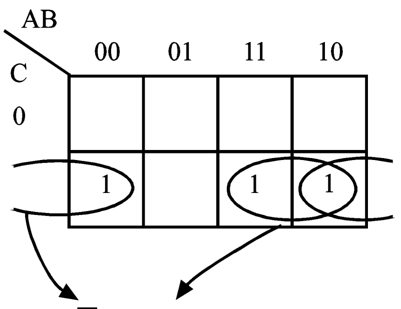
$$F = AB(\overline{C} + C) + \overline{A}\overline{B}(C + \overline{C})$$

$$\underline{F = AB + \overline{A}\overline{B}}$$

Note: This is the same result as obtained by the mapping technique.

Example 2

Simplify: $F = ABC + A\overline{B}C + \overline{A}\overline{B}C$



$$\underline{F = \overline{B}C + AC}$$

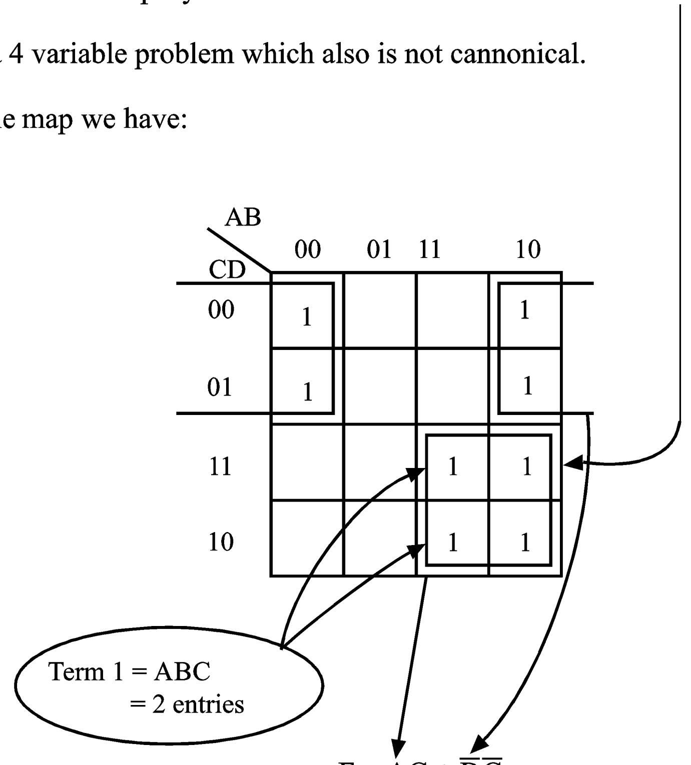
Note: The LHS and RHS of the map are identical hence the second loop shown broken in the diagram.

Example 3

Simplify: $F = ABC + A\overline{B}\,\overline{D} + \overline{B}\,\overline{C} + A\overline{B}CD$

Note: This is a 4 variable problem which also is not cannonical.

Using a 4 variable map we have:



$F = AC + \overline{B}\,\overline{C}$

Notes: You will see that to place the term ABC on a 4 variable map we will need 2 entries. The term BC will require 4 entries to specify on the map. The term ABCD which as all 4 variables present will only require 1 entry on the map. Also note if an entry is already present from a previous term, it is not necessary to duplicate it.

Example 4

Simplify using Boolean algebra the following expression and check your answer with a map.

(1) By Boolean Algebra: $F = A\overline{B}C + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C}$

It can be seen that BC is common to the last two terms, and the expression can be rewritten:

$F = A\overline{B}C + \overline{B}\overline{C}\,(A + \overline{A})$

$= A\overline{B}C + \overline{B}\overline{C}$ as $A + \overline{A} = 1$ from section 3.1.1

$= \overline{B}\,(AC + \overline{C})$

$= \overline{B}\,(AC + \overline{C} \quad 1)$

$= \overline{B}\,(AC + \overline{C}\,[1 + A])$

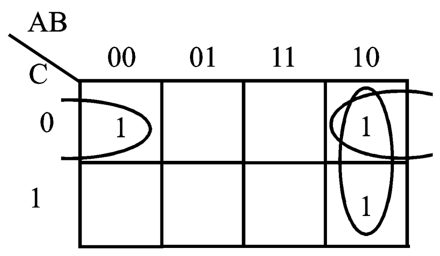
$= \overline{B}\,(AC + \overline{C} + \overline{C}A)$

$= \overline{B}\,(A\,(C + \overline{C}) + \overline{C})$

$F = \overline{B}\,(A + \overline{C})$

(2) By map:

F =



$F = A\overline{B} + \overline{B}\,\overline{C}$

$\underline{F = \overline{B}\,(A + \overline{C})}$

Mapping techniques can be superior!

Example 5

A submarine is to have an automatic control system which will cause it to submerge if the wind speed is greater than 30 knots, the water speed is greater than 10 knots and if an unidentified surface ship appears within 20 miles. It will also submerge if an air target appears closer than 150 miles when the wind speed is greater than 30 knots and will also submerge if the water speed is less than 10 knots and an unidentified surface ship appears outside 20 miles range. It should also submerge if an air target appears outside 150 miles with an unidentified surface ship within 20 miles, the wind speed being greater than 30knots and it should also submerge if manually instructed to do so.

Design a logic system which will accept the defined inputs and produce the required single “submerge” output instruction.

Hint: Define your own symbol table for all the required conditions.

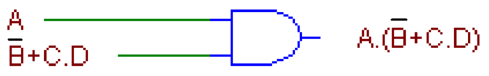
i.e. Let W = water speed greater than 10 knots etc.

Note: The solution to this problem is given later. However you should not look at it until you have at least made a serious attempt at solving this problem yourself.

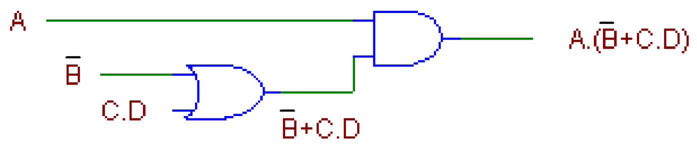
3.3 Implementation of a boolean expression using logic gates

Implementation of the final boolean expression with logic gates is done backwards. That is, commencing with the final gate rather than the first. Consider the expression $A \quad (\overline{B}+C \quad D)$

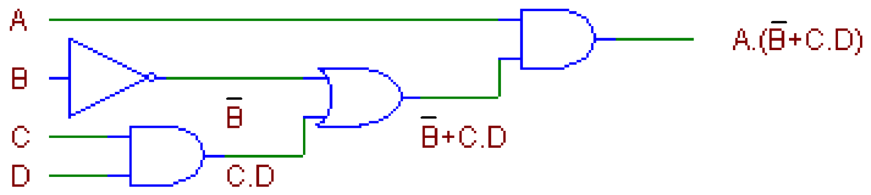
1. The output of the final gate will be the expression $A \quad (\overline{B}+C \quad D)$.
2. To determine the inputs to the final gate we do the operation outside the brackets first. That is, an AND operation where one input to the AND gate is A and the other input is $(\overline{B}+C \quad D)$. Note – A is an initial input, so this input is complete.



3. To determine the intermediate gates you must undo the OR operation (inside the brackets) before the AND operation. Since, the output of the OR operation is $\overline{B}+C \quad D$, one input to the OR gate is $\overline{B}$ and the other is $C \quad D$, as shown below.



4. Finally, the $\overline{B}$ input is B inverted and the $C \cdot D$ input is C AND D, as shown below.



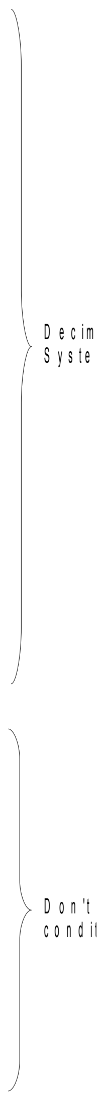
3.4 Don't care conditions

In some logic systems certain input conditions may occur which may not affect the output. In other cases they may affect the output but it will have no effect on the system. If these input conditions occur we 'don't care' and the inputs are called **don't cares**.

Consider a binary sequence for 4 variables:

Count	Inputs				Output
	A	B	C	D	F
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	X
11	1	0	1	1	X
12	1	1	0	0	X
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	X

As you can see for a decimal system (0-9) we only need some of the binary inputs as shown and their corresponding outputs, whether a logic one or zero. The remaining inputs are not required and may be called "don't care" conditions. It means we don't care if the inputs are a one or zero. For that reason their outputs are given the symbol (X) in order that they may be distinguished on a Karnaugh Map.



3.4.1 Minimisation using maps with don’t care terms

Don’t care conditions are chosen as either 1 or 0 depending on which choice gives the largest loop. However we do not try to loop as many or as few don’t cares as possible. i.e. Loops which contain **only** don’t cares should **not** be used.

Activity 3.2

Minimise the following using the don’t cares as shown.

1.

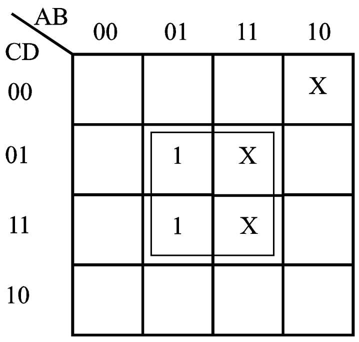
CD \ AB	00	01	11	10
00				X
01		1	X	
11		1	X	
10				

2.

CD \ AB	00	01	11	10
00		X	1	
01			1	
11			X	X
10			X	X

Solutions to activity 3.2

1.

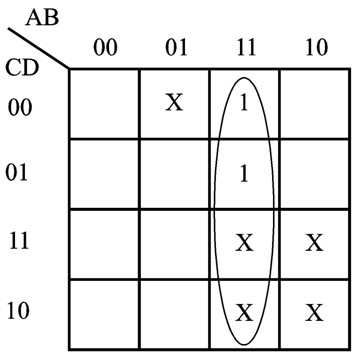


CD \ AB	00	01	11	10
00				X
01		1	X	
11		1	X	
10				

Assuming X = 1, we can create a 4-loop.

ANS = BD

2.



CD \ AB	00	01	11	10
00		X	1	
01			1	
11			X	X
10			X	X

ANS = AB

Solution to activity 3.1, Example 5

Symbol table

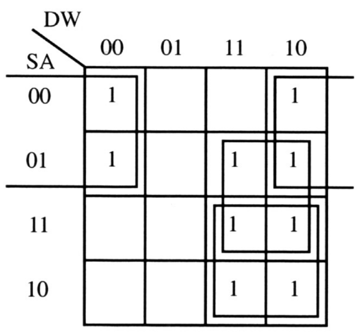
Let	D	=	wind speed $>$ 30 knots
	W	=	water speed $>$ 10 knots
	S	=	unidentified surface ship $<$ 20 miles
	A	=	air target $<$ 150 miles
	M	=	manual dive

Boolean expression

Analysing the problem from the statement, the Boolean equation may be written from each sentence:

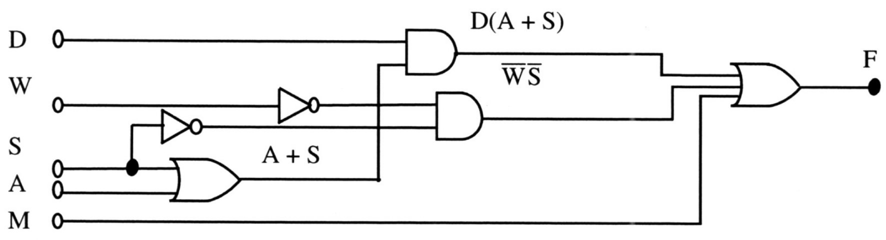
$$F = DWS + AD + \overline{W}\,\overline{S} + \overline{A}SD + M$$

Simplification



$$\begin{aligned} F &= AD + DS + \overline{W}\,\overline{S} \\ &= D(A + S) + \overline{W}\,\overline{S} \end{aligned}$$

Implementation



Activity 3.3

1. Find complements of the following expressions then using DeMorgan's laws simplify the expressions until DeMorgan's laws cannot be applied further.

   a. $A\overline{B} + C(\overline{D} + E)$ NOTE complement = $\overline{A\overline{B} + C(\overline{D} + E)}$

   b. $A\overline{B} + \overline{C}\overline{D}(E + \overline{F}G)$

2. Show that $A + A \cdot B = A$ by means of:

   i. Boolean algebra

   ii. a truth table

   and implement the function with logic.

3. Show that $(A + B) \cdot (A + C) = A + B \cdot C$ using boolean algebra and implement each side of the expression using logic.

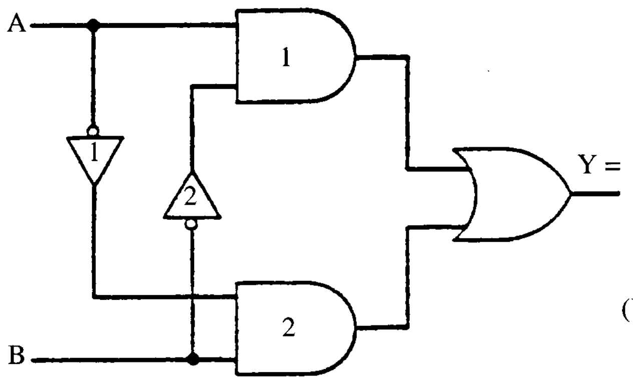
4. Given $D = A + \overline{A}\overline{B}C + ABC + CB + C\overline{B}$

   i. minimise the expression, $(D = A + C)$

   ii. draw the logic diagram for the given equation,

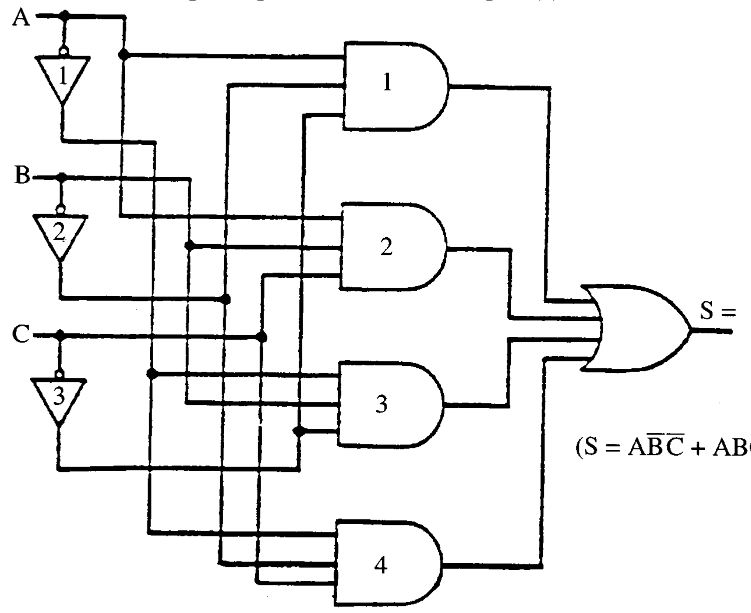
   iii. draw the reduced logic diagram.

5. Write the expression for the output (y).



$(Y = A\overline{B} + \overline{A}B)$

6. Write the logic expression for the output (s).



($S = A\overline{B}\overline{C} + ABC + \overline{A}B\overline{C} + \overline{A}\overline{B}C$)

7. Refer to the ELE1301 course page on 'Study Desk' and complete the following experiments:

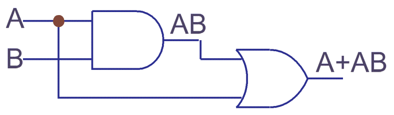
   a. Home experiment 3-1 – Boolean operations
   b. Home experiment 3-2 – DeMorgan's theorems.

Solutions to activity 3.3

1. a. $(\overline{A} + B)(\overline{C} + D\overline{E})$

   b. $(\overline{A} + \overline{B})[\overline{C}$ implied brackets removed

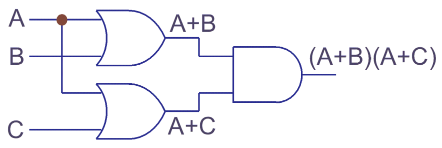
2. $$\begin{aligned} A + AB &= A \\ A(1 + B) &= A \\ A &= A \end{aligned}$$

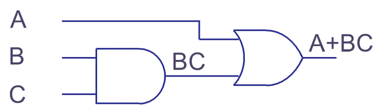


3. $(A + B)(A + C)$ $= A + BC$

   $AA + AC + BA + BC$

   $A + AC + BA + BC$
   $A(1 + C) + BA + BC$
   $A + BA + BC$
   $A(1+B) + BC$
   $A + BC$





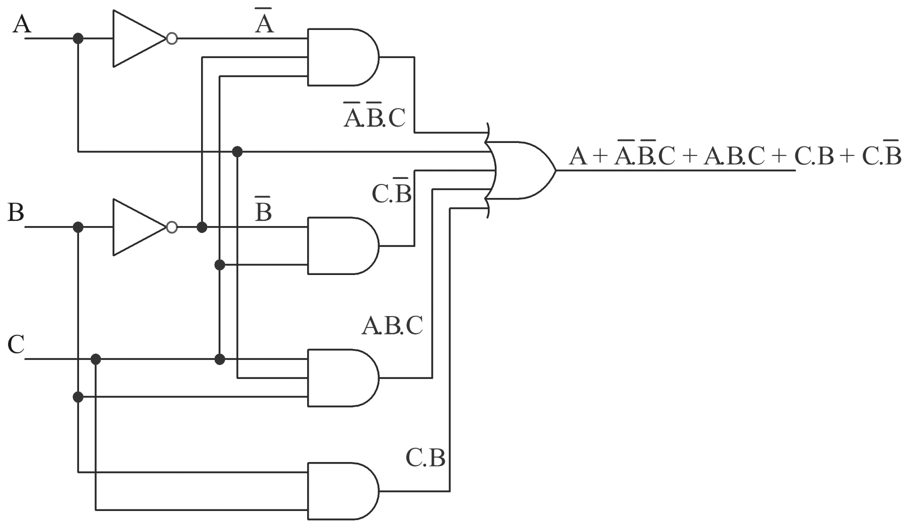
4. $A + \overline{A}\,\overline{B}C + ABC + CB + C\overline{B}$

   $A + C(\overline{A}\overline{B} + AB) + C(B + \overline{B})$

   $A + C(\overline{A}\overline{B} + AB) + C$

   $A + C[(\overline{A}\,\overline{B} + AB) + 1]$

   $A + C$


A
B
C
Ā
B̄
Ā.B̄.C
C.B̄
A.B.C
C.B
A + Ā.B̄.C + A.B.C + C.B + C.B̄