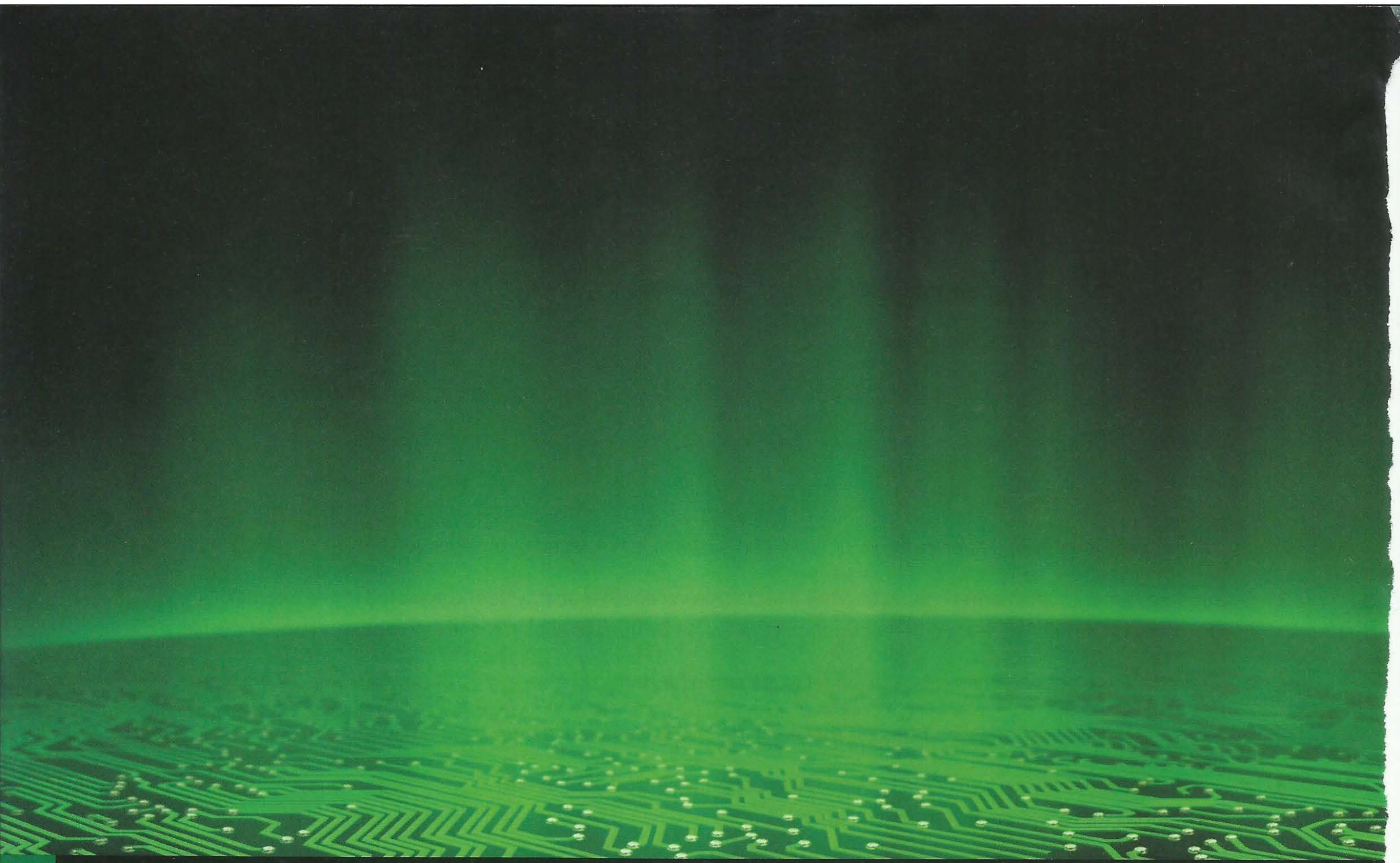
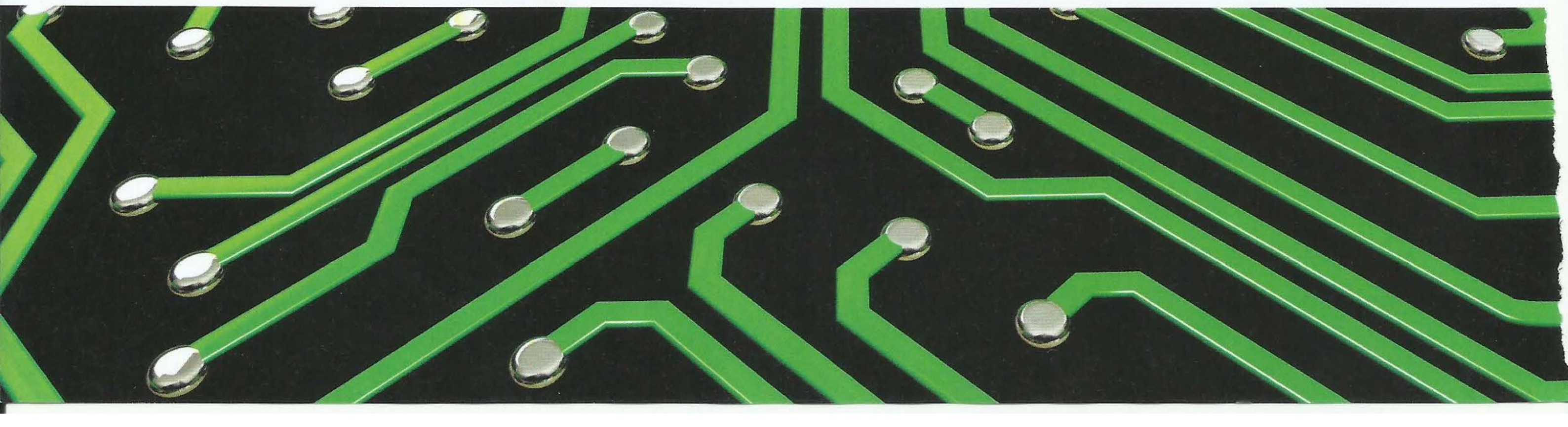


Chapter 18
Boolean Algebra and Logic Circuits

Learning objectives

By the end of this chapter you should be able to:

- show understanding of Boolean algebra
- show understanding of De Morgan's Laws
- perform Boolean algebra using De Morgan's Laws
- simplify a logic circuit/expression using Boolean algebra
- produce truth tables for common logic circuits
- show understanding of how to construct a flip-flop
- describe the role of flip-flops as data storage elements
- show understanding of Karnaugh Maps and the benefits of using them
- solve logic problems using Karnaugh Maps.



18.01 Boolean algebra basics

Chapter 4 (Section 4.01) introduced logic expressions consisting of logic propositions combined using Boolean operators. Boolean algebra provides a simplified way of writing a logic expression and a set of rules for manipulating an expression.

Whenever a form of algebra is used it is vital that there is an understanding of its meaning. As a simple example we can consider the following four interpretations of the meaning of 1 + 1:

$$1 + 1 = 2$$
$$1 + 1 = 10$$
$$1 + 1 = 0$$
$$1 + 1 = 1$$

The first shows denary arithmetic, the second binary arithmetic and the third bit arithmetic. The last one applies if Boolean algebra is being used. This is because in Boolean algebra 1 represents TRUE, 0 represents FALSE, and + represents OR. Therefore the fourth statement represents the logic statement:

TRUE OR TRUE is TRUE

There are options for the representation of Boolean algebra. For example, the symbols for AND and OR are sometimes represented as $\wedge$ and $\vee$. There is the option of writing A.B or AB for AND. The dot notation is used in this book. Finally, there are options for how NOT A (the inverse of A) can be represented. $\bar{A}$ is used here.

Having established the notation for Boolean algebra we have to consider the rules that apply. These can formally be described as 'laws' or 'identities'. Table 18.01 contains a full listing.

Identity/Law	AND form	OR form
Identity	$1.A = A$	$0+A = A$
Null	$0.A = 0$	$1+A = 1$
Idempotent	$A.A = A$	$A+A = A$
Inverse	$A.\bar{A} = 0$	$A+\bar{A} = 1$
Commutative	$A.B = B.A$	$A+B = B+A$
Associative	$(A.B).C = A.(B.C)$	$(A+B)+C = A+(B+C)$
Distributive	$A+B.C = (A+B).(A+C)$	$A.(B+C) = A.B+A.C$
Absorption	$A.(A+B) = A$	$A+A.B = A$
De Morgan's	$\overline{(A.B)} = \bar{A} + \bar{B}$	$\overline{(A+B)} = \bar{A}.\bar{B}$
Double Complement	$\bar{\bar{A}} = A$	

Table 18.01 Boolean algebra identities (laws)

Some of the names used for the identities may be unfamiliar to you. This is not a concern. You should note that for all but one of the identities there is an AND form and an OR form. Furthermore, it is important to note that an identity written in one form can be transformed into the other by interchanging each 0 or 1 and each AND and OR. For example, 0.A = 0 which reads FALSE AND A is FALSE transforms into TRUE OR A is TRUE, written in the algebra as 1+A = 1.

It can also be seen that some of the identities look like those applying in normal algebra with AND functioning as multiplication and OR functioning as addition. Thus it is allowed for the terms 'product' and 'sum' to be used in the context of Boolean algebra.

TASK 18.01

It is vital that you can interpret a Boolean expression correctly. Go through Table 18.01 item by item and in each case read out the full meaning. For example:

1+A = 1 can be read as 'one plus A equals 1'

but must be understood as 'TRUE OR A is TRUE'

Although De Morgan's laws look complicated at first glance, they can be rationalised easily. The inverse of a Boolean product becomes the sum of the inverses of the individual values in the product. The inverse of a Boolean sum is the product of the individual inverses.

Unfortunately, using the algebra to simplify expressions is not something which can be learnt as a routine. It almost inevitably requires a little lateral thinking as Worked Example 18.01 will show.

WORKED EXAMPLE 18.01

Using Boolean algebra to simplify an expression

Let's consider a simple example:

A+Ā.B can be simplified to A+B

In order to simplify the expression we have to first make it more complicated! This is where the lateral thinking comes in. The OR form of the absorption identity is A+A.B = A. This can be used in reverse to replace A by A+A.B to produce the following:

A+A.B+Ā.B

Applying the AND form of the commutative law and the OR form of the distributive law in reverse we can see that:

A.B+Ā.B is the same as B.A+B.Ā which converts to B.(A+Ā)

This allows us to use the OR form of the inverse identity which converts A+Ā to 1. As a result the expression has become:

A+B.1

When the OR form of the commutative law and the AND form of the identity law are applied to the B.1 term, it then becomes A+B.

18.02 Logic circuits

Chapter 4 introduced the symbols for logic gates that are used in logic circuits and discussed the relationships between logic circuits, truth tables and logic expressions. This chapter introduces some specific circuits that are used to construct components that provide functionality in computer hardware.

The half adder

A fundamental operation in computing is binary addition. The result of adding two bits is either 1 or 0. However, when 1 is added to 1 the result is 0 but there is a carry bit equal to 1. This cannot be ignored if two numbers with several bits in each are being added.

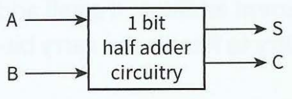


Figure 18.01 A half adder

The simplest circuit that can be used for binary addition is the half adder. This can be represented by the diagram in Figure 18.01. The circuit takes two input bits and outputs a sum bit (S) and a carry bit (C).

The circuit required can be considered in the context of the truth table which is shown as Table 18.02.

Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 18.02 The truth table for a half adder

One possible circuit can be defined directly by examination of the truth table. It can be seen that the only combination of inputs that produces a 1 for the carry bit is when two 1 bits are input. The truth table for the C output is in fact the AND truth table. The truth table for the S output can be seen to match that for the XOR operator which is shown in Figure 4.02 in Chapter 4 (Section 4.04). Therefore, one circuit that would produce the half adder functionality would contain an AND gate and an XOR gate with each gate receiving input from A and B.

This is only one of several circuits that would provide the functionality. As was explained in Chapter 4 (Section 4.05), circuit manufacturers prefer to use either NAND or NOR gates. The circuit shown in Figure 18.02 consisting only of NAND gates has the correct logic to produce the C and S outputs and is a likely choice for implementation.

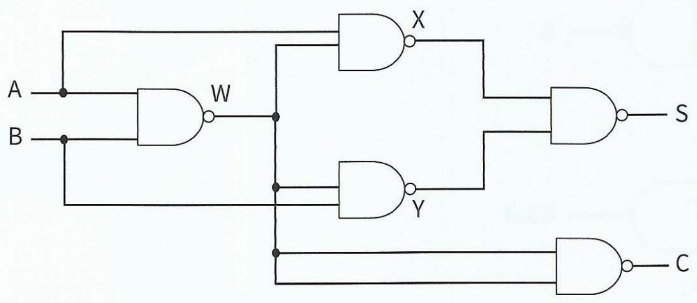


Figure 18.02 A half adder circuit constructed from NAND gates

Question 18.01

In Figure 18.02, can you identify the individual circuits that represent the AND operator and the XOR operator?

TASK 18.02

Use the intermediate points labelled W, X and Y to construct a truth table for the circuit shown in Figure 18.02. Check that this reproduces the truth table shown as Table 18.02.

The full adder

If two numbers expressed in binary with several bits are to be added, the addition must start with the two least significant bits and then proceed to the most significant bits. At each stage the carry from the previous addition has to be incorporated into the current addition. If a half adder is used each time, there has to be separate circuitry to handle the carry bit because the half adder only takes two inputs.

The full adder is a circuit that has three inputs including the previous carry bit. The truth table is shown as Table 18.03.

Input			Output	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 18.03 The truth table for a full adder

One possible circuit for implementation contains two half adder circuits and an OR gate as shown in Figure 18.03.

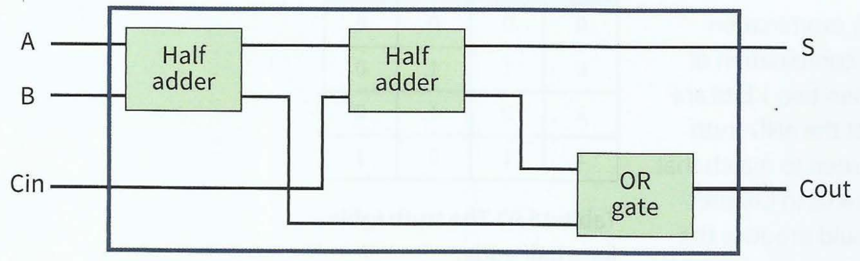


Figure 18.03 A possible implementation of a full adder

As before, it is possible to construct the circuit entirely from NAND gates as shown in Figure 18.04.

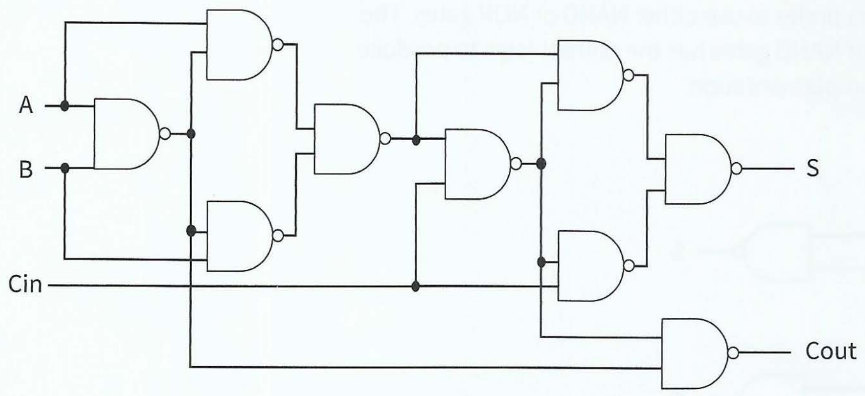


Figure 18.04 A full adder circuit constructed from NAND gates

Discussion Point:

Can you see how full adders could be combined to handle addition of, for example, four-bit binary numbers? What happens to the carry input for the first addition?

The SR flip-flop

All of the circuits so far encountered in this book have been **combinational circuits**. For such a circuit the output is dependent only on the input values. An alternative type of circuit is a **sequential circuit** where the output depends on the input and on the previous output.

KEY TERMS

Combinational circuit: a circuit in which the output is dependent only on the input values

Sequential circuit: a circuit in which the output depends on the input values and the previous output

The SR flip-flop or 'latch' is a simple example of a sequential circuit. It can be constructed with two NAND gates or two NOR gates. Figure 18.05 shows the version with two NOR gates. The flip-flop is a two-state device. Either it has Q set to 1 and Q' set to 0 or it has the reverse.

The truth table for the circuit can be presented as shown in Table 18.04. The two lines of the truth table where both S and R are input as 0 produce no change in the values set for Q or Q'. This is the condition when no signal is input to the flip-flop. Input of S = 1 and R = 0 always produces Q = 1 and Q' = 0. Input of S = 0 and R = 1 always produces the reverse.

This explains why the SR flip-flop can be used as a storage device for 1 bit and therefore could be used as a component in RAM because a value is stored but can be altered. The circuit must be protected from receiving input on R and S simultaneously because this leads to an invalid state with both Q and Q' set to 0.

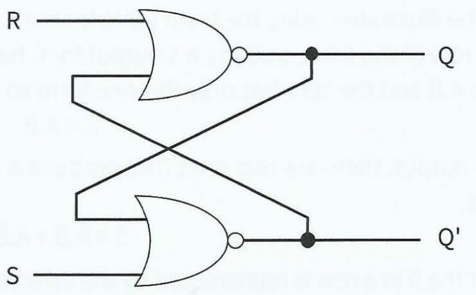


Figure 18.05 A circuit for an SR flip-flop using NOR gates

Input signals		Initial state		Final state	
S	**R**	**Q**	**Q'**	**Q**	**Q'**
0	0	1	0	1	0
1	0	1	0	1	0
0	1	1	0	0	1
0	0	0	1	0	1
1	0	0	1	1	0
0	1	0	1	0	1

Table 18.04 A representation of a truth table for an SR flip-flop

The JK flip-flop

In addition to the possibility of entering an invalid state there is also the potential for a circuit to arrive in an uncertain state if inputs do not arrive quite at the same time. In order to prevent this, a circuit may include a clock pulse input to give a better chance of synchronising inputs. The JK flip-flop is an example.

The JK flip-flop can be illustrated by the symbol shown in Figure 18.06(a). A possible circuit is shown in Figure 18.06(b).

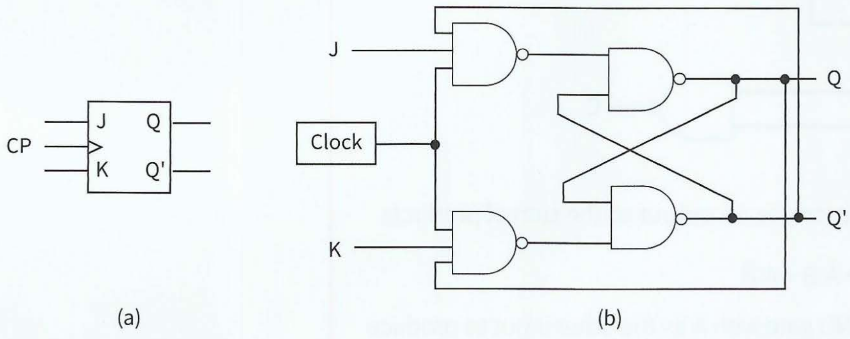


Figure 18.06 (a) A symbol for a JK flip-flop and (b) a possible circuit

The workings of the circuit are viewed in terms of the value of the Q output immediately after the circuit detects a clock pulse. The J input acts as a set input and the K as a clear so there is some similarity to the functioning of the SR flip-flop. However, if both J and K are input as a 1 then Q always switches value. The significant part of the truth table is shown as Table 18.05.

J	K	Clock	Q
0	0	↑	Q unchanged
1	0	↑	1
0	1	↑	0
1	1	↑	Q toggles

Table 18.05 Part of the truth table for a JK flip-flop

18.03 Boolean algebra applications

The Boolean algebra representation of a truth table

One approach to creating a Boolean algebra expression for a particular problem is to start with the truth table and apply the sum of products method. This establishes a minterm for each row of the table that results in a 1 for the output.

This can be illustrated using the truth table for the half adder circuit shown in Figure 18.02. The only row of the table creating a 1 output for C has a 1 input for A and for B. The product becomes A.B and the sum has only this one term so we have:

$$C = A.B$$

For the S output, there are two rows that produce a 1 output so there is a sum of two minterms:

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

Note that the 0 in a row is represented by the inverse of the input symbol.

The Boolean algebra representation of a logic circuit

This approach can also be used as part of the process of creating a Boolean algebra logic expression from a circuit diagram. The truth tables for the individual logic gates are used and then some algebraic simplification is applied.

WORKED EXAMPLE 18.02

Creating a Boolean algebra logic expression from a half adder circuit

For convenience Figure 18.02 is reproduced here as Figure 18.07. Examination of the figure shows inputs A and B to a NAND gate with output W.

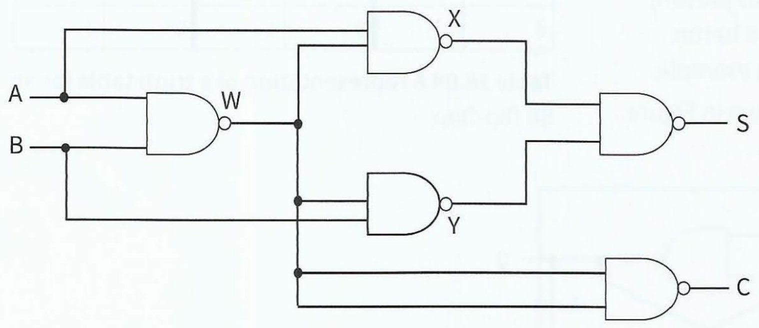


Figure 18.07 A half adder circuit

The first three rows of the NAND truth table produce a 1 output so the sum of products has three minterms:

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

We can now consider the input of W to a NAND gate with A as the other input to produce the X output. The NAND gate operates as an AND gate followed by a NOT gate. The result of the AND operation is the product of the inputs so:

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

Applying the distributive and inverse laws now gives:

$$X = 0 + 0 + A.A.\bar{B} \text{ which is simply } A.\bar{B}$$

We have to take the inverse of this to complete the NAND operation. This is where we need the AND version of De Morgan's law, which transforms the $A.\bar{B}$ into $\bar{A}+B$.

The same laws applied to the output Y from the other intermediate NAND gate to give

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

Finally, we need to consider $A+\bar{B}$ and $\bar{A}+B$ being input to the final NAND gate. Again we can consider the AND operation first as the product of the inputs:

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

If we pause to think we will not multiply this out but instead we will apply De Morgan's law directly to this to perform the inverse operation to complete the NAND operation. This gives:

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

This is the value obtained directly from the truth table so the algebra has been used correctly.

Extension question 18.01

Worked Example 18.02 did not show that the circuit produced the correct output for C. Also a shortcut was used to reach the final form of S. Can you use Boolean algebra to find the form of C from the circuit and can you convert the expression for S if you start by using the distributive law before applying De Morgan's law?

18.04 Karnaugh maps (K-maps)

A Karnaugh map is a method of creating a Boolean algebra expression from a truth table. It can make the process much easier than if you use sum-of-products to create minterms. The truth table for an OR gate, shown as Table 18.06, can be used to illustrate the method.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Table 18.06 The truth table for the OR operand

Using the sum-of-products approach gives the following expression for X:

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

This is not instantly recognisable as A+B but, with a little effort, using Boolean algebra laws it could be shown to be the same.

The Karnaugh map approach is simpler. The corresponding K-map is shown in Figure 18.08. Each cell in a Karnaugh map shows the value of the output X for a combination of input values for A and B.

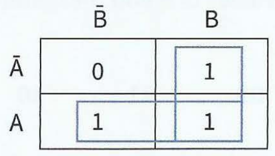


Figure 18.08 A K-map of the truth table in Table 18.06

The interpretation of a Karnaugh map follows these rules:

- Only cells containing a 1 are considered.
- Groups of cells containing 1s are identified where possible, with a group being a row, a column or a rectangle.
- Groups must contain 2, 4, 8 and so on cells.
- Each group should be as large as possible.
- Groups can overlap.
- If an individual cell cannot be contained in any group it is treated as being a group.
- Within each group, the only input values retained are those which retain a constant value throughout the group.

These rules define a column and a row group as indicated by the blue outlines. In the column group, B remains unchanged but A changes so B is retained. In the row group, it is A that remains unchanged. The Boolean algebra expression is then just the sum of these retained values:

$$X = A+B$$

Thus the Karnaugh map has found the OR expression without using any algebra.

WORKED EXAMPLE 18.03

Using a K-map to interpret a three-input problem

Let's consider the truth table shown in Table 18.07.

A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Table 18.07 A sample truth table with three inputs

Before starting any application of a method it is always worth looking to see if there are any trends. In this case you can see that whenever B = 1 the output for X is 1. This means that the final algebra should have B + something.

Applying sum of products gives the following five-minterm expression:

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

There are options for how the K-map is presented. We will choose to combine input values in the columns. Figure 18.09 shows the result. This follows the convention of having the rows corresponding to values of A and the columns to combinations of values for B and C.

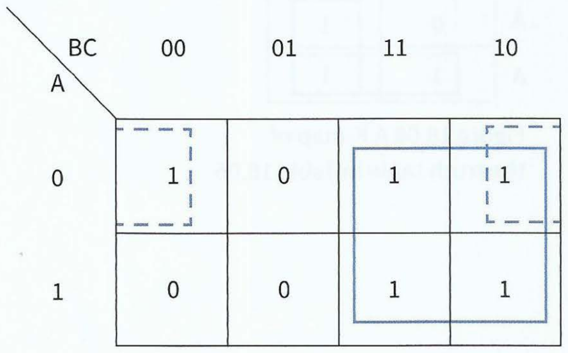


Figure 18.09 A K-map representation of the truth table shown in Table 18.07

It is important to note that the labelling of the columns does not follow a binary value pattern. Instead it follows the Gray coding sequence, where only one bit changes value each time.

Following the rules stated above, the first group to identify is the square of four cells with a value 1 as identified by the blue rectangle in the diagram. For these it can be seen that A has different values, B has a constant value but C changes values. So, only B is retained. Note this was anticipated from the initial inspection of the truth table.

This apparently only leaves the top left cell. It looks like an isolated cell but it is not because K-maps wrap round. The cell is defined by BC = 00. This has two adjacent cells under Gray coding rules. One is immediately obvious – BC = 01 but this contains 0 so can be ignored. The other adjacent cell is the BC = 10 combination. Thus, there is a row group containing BC = 00 and BC = 10, indicated by the dotted line partial group outlines. Note that we cannot include the 11 cell in the same row because a group cannot contain three members. For this row, the value $\bar{A}$ remains unchanged, B changes but $\bar{C}$ remains unchanged so the product $\bar{A}.\bar{C}$ results. So by adding this to the B for the other group the final expression becomes:

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

This is much simpler than the expression with five minterms derived directly from the truth table.

Extension question 18.02

Consider the Karnaugh map shown in Figure 18.10. This corresponds to a problem with four inputs. It wraps round horizontally and vertically. Use the map to create a Boolean algebra expression.

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	0	1	1
11	0	0	1	1
10	1	0	0	0

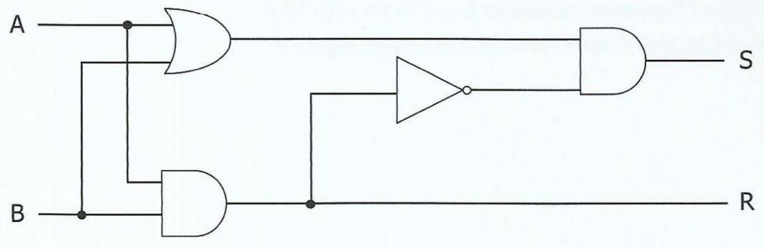
Figure 18.10 A K-map for a four input problem

Summary

- There are Boolean algebra laws that can be used to simplify logic expressions.
- Binary addition can be carried out using a half adder or a full adder circuit.
- SR or JK flip-flop circuits can be used to store a bit value.
- The sum-of-products method can be used to create an equivalent logic expression containing minterms from a truth table.
- A Karnaugh map is a representation of a truth table that allows a simplified logic expression to be derived from a truth table.

Exam-style Questions

1 a Consider the following circuit:



i Identify the three different logic gates used. [2]

ii Complete the following truth table for the circuit for the inputs shown for A and B: [5]

Inputs		Working space	Outputs	
A	B		S	R
0	0			
0	1			
1	0			
1	1			

b For the circuit shown in part (a), identify the type of circuit and what the outputs represent. [3]

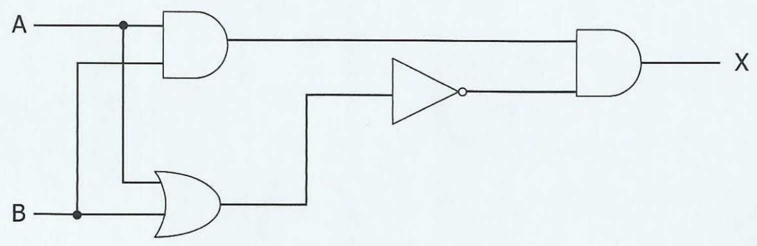
2 a Consider the following truth table:

A	B	X
0	0	1
0	1	0
1	0	1
1	1	1

i Using the sum-of-products approach, create a Boolean expression that matches the logic. [3]

ii For the rows that have A = 1, the output for X is 1. Explain how this would be reflected in a simplified form of Boolean expression matching the truth table. [2]

b Consider the following circuit:



i Using your knowledge of the truth table for an AND gate, create a Boolean algebra expression for the output from the first AND gate. [2]

ii Carry out the same exercise for the OR gate in the circuit. [3]

iii Using De Morgan's law, create the logic expression for the output from the NOT gate. [4]

3 a Consider the following truth table:

A	B	X
0	0	1
0	1	0
1	0	1
1	1	1

i Create a Karnaugh map to match this truth table. [4]

ii Use the Karnaugh map to create a Boolean algebra expression for this logic. [3]

b Consider the truth table shown in **3 a**.

i Use the sum-of-products method to create a Boolean algebra expression from the truth table. [3]

ii Use Boolean algebra to show that this expression can be simplified to give the same expression created from the Karnaugh map. (Hint: you might wish to use the fact that $A.\bar{B} = A.\bar{B} + A.\bar{B}$). [4]