Module
2

www.learnabout-electronics.org

Digital Electronics

2.0 Digital Logic

What you'll learn in Module 2

Section 2.0 Introduction.

Section 2.1 Logic Gates.

- 74 Series standard logic gates.
- Standard logic functions.

  AND, OR, NAND, NOR, XOR, XNOR, NOT.
- Truth tables for standard logic functions.

Section 2.2 Combinational Logic

- Combining logic gates.
- Truth tables.
- Boolean equations.

Section 2.3 Boolean Algebra.

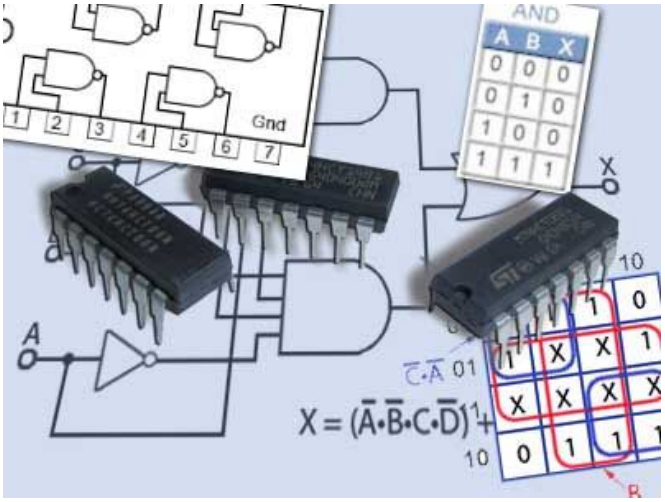
- Simplifying Boolean equations
- Boolean laws and rules
- De Morgan's theorem

Section 2.4 Karnaugh Maps.

- Constructing Karnaugh maps
- Minimising Karnaugh maps
- Software for Boolean simplification

Section 2.5 Digital Logic Quiz.

•Test your knowledge of Digital Logic.



Introduction.

Digital logic is the foundation, not only of computing but also many other electronic devices and control systems found in almost every part of modern life.

This module introduces the basics of digital logic and shows how the whole of digital electronics depends on just seven types of logic gates, connected together with a minimum of additional components. Combinations of logic gates then form circuits that can perform specific tasks within larger circuits or systems. The process of producing complex circuits using combinations of basic devices is called Combinational Logic.

There are many ways that a number of logic gates can be combined to perform a specific task. They may all work, but some combinations will perform the task that better than others. For example, a circuit designer may want to design a combinational logic circuit that uses the minimum number of gates, or performs the required task in the least time, or at the minimum cost.

This module also introduces the way digital logic gates work and teaches you key methods by which a basic digital logic circuit design may be minimised, made more efficient and/or cheaper.

2.1 Logic Gates

What you'll learn in Module 2.1

After studying this section, you should be able to:

Describe the action of logic gates.

- AND, OR, NAND, NOR, NOT, XOR and XNOR
- Using Boolean expressions.
- Using truth tables.

Understand the use of universal gates.

- NAND
- NOR

Recognise common 74 series ICs containing standard logic gates.

Seven Basic Logic Gates

Digital electronics relies on the actions of just seven types of logic gates, called AND, OR, NAND (Not AND), NOR (Not OR), XOR (Exclusive OR) XNOR (Exclusive NOR) and NOT.

Because, in binary logic there are only two states, 1 and 0 or 'on and off,' NOT in the world of binary logic therefore means 'the opposite of'. If something is not 1 it must be 0, if it is not on, it must be off. So NAND (not AND) simply means that a NAND gate performs the opposite function to an AND gate.

A logic gate is a small transistor circuit, basically a type of amplifier, which is implemented in different forms within an integrated circuit. Each type of gate has one or more (most often two) inputs and one output.

The principle of operation is that the circuit operates on just two voltage levels, called logic 0 and logic 1. When either of these voltage levels is applied to the inputs, the output of the gate responds by assuming a 1 or a 0 level, depending on the particular logic of the gate. The logic rules for each type of gate can be described in different ways, by a written description of the action, by a truth table, which is a table showing all the possible logic states at the inputs and output of the gate, or by a Boolean algebra statement.

Boolean statements use letters from the beginning of the alphabet, such as A, B, C etc. to indicate inputs, and letters from the second half of the alphabet, very commonly X or Y and sometimes Q or P to label an output. The letters have no meaning in themselves, other than just to label the various points in the circuit. The letters are then linked by a symbol indicating the logical action of the gate.

The • symbol indicates AND although in many cases the • may be omitted. (A•B may also be written as AB or A.B)

+ indicates OR

$\oplus$ indicates XOR (Exclusive OR)

Although the symbols • and + are the same as those used in normal algebra to indicate product (multiplication) and sum (addition) respectively, in binary logic the + symbol does not exactly correspond to sum. In digital logic 1 + (OR) 1 = 1, but the binary sum of 1 + (plus) 1 = 10_2, therefore in digital logic + must always be considered as OR.

Three further types of logic gate give an output that is an inverted version of the three basic gate functions listed above, and these are indicated by a bar drawn above a statement using the AND, OR, or XOR symbols to indicate NAND, NOR and XNOR.

$$A \bullet B \text{ means A AND B but } \overline{A \bullet B} \text{ means A NAND B}$$

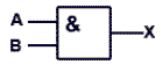
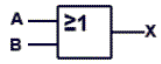
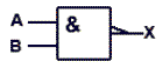
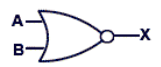
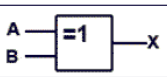
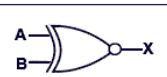
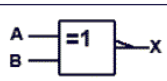
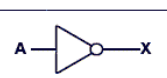
For example:

An AND gate gives an output of logic 1 when input A AND input B are at logic 1, but a NAND gate would give a logic 0 output for the same input conditions. Also where the AND gate gives a logic zero for a particular input combination, the NAND gate would give a logic 1. The 'N' in the gate's name, or the bar above the Boolean expression therefore indicates that the output logic is 'inverted'. In digital logic NAND is 'NOT' AND or the opposite of AND. Similarly NOR is 'NOT' OR and XNOR is 'NOT' XOR.

The final gate type, the NOT gate or inverter is a single input gate that has an output having the opposite logic state, or the inverse of the input.

Table 2.1.1 shows each of the seven basic logic gates, which may be illustrated by either the traditional "Distinctive Shape" ANSI symbol or the newer rectangular IEC symbol, and a written description of its logic function compared with its Boolean equation.

Table 2.1.1

ANSI Symbol	IEC Symbol	Description	Boolean
		The AND gate output is at logic 1 when, and only when all its inputs are at logic 1, otherwise the output is at logic 0.	$X = A \cdot B$
		The OR gate output is at logic 1 when one or more of its inputs are at logic 1. If all the inputs are at logic 0, the output is at logic 0.	$X = A+B$
		The NAND Gate output is at logic 0 when, and only when all its inputs are at logic 1, otherwise the output is at logic 1.	$X = \overline{A \cdot B}$
		The NOR gate output is at logic 0 when one or more of its inputs are at logic 1. If all the inputs are at logic 0, the output is at logic 1.	$X = \overline{A+B}$
		The XOR gate output is at logic 1 when one and ONLY ONE of its inputs is at logic 1. Otherwise the output is logic 0.	$X = A \oplus B$
		The XNOR gate output is at logic 0 when one and ONLY ONE of its inputs is at logic 1. Otherwise the output is logic 1. (It is similar to the XOR gate, but its output is inverted).	$X = \overline{A \oplus B}$
		The NOT gate output is at logic 0 when its only input is at logic 1, and at logic 1 when its only input is at logic 0. For this reason it is often called an INVERTER.	$X = \overline{A}$

Logic ICs

Fig. 2.1.1 illustrates a selection of the basic gates logic gates that are available from a number of manufacturers in standard families of integrated circuits. Each logic family is designed so that gates and other logic ICs within that family (and other related families) can be easily combined, and built into larger logic circuits to carry out complex functions with the minimum of additional components.

Typically, standard logic gates are available in 14 pin or 16 pin DIL (dual in line) chips. The number of gates per IC varies depending on the number of inputs per gate. Two-input gates are common, but if only a single input is required, such as in the 7404 NOT (or inverter) gates, a 14 pin IC can accommodate 6 (or Hex) gates. The greatest number of inputs on a single gate is on the 74133 13 input NAND gate, which is accommodated in a 16 pin package.

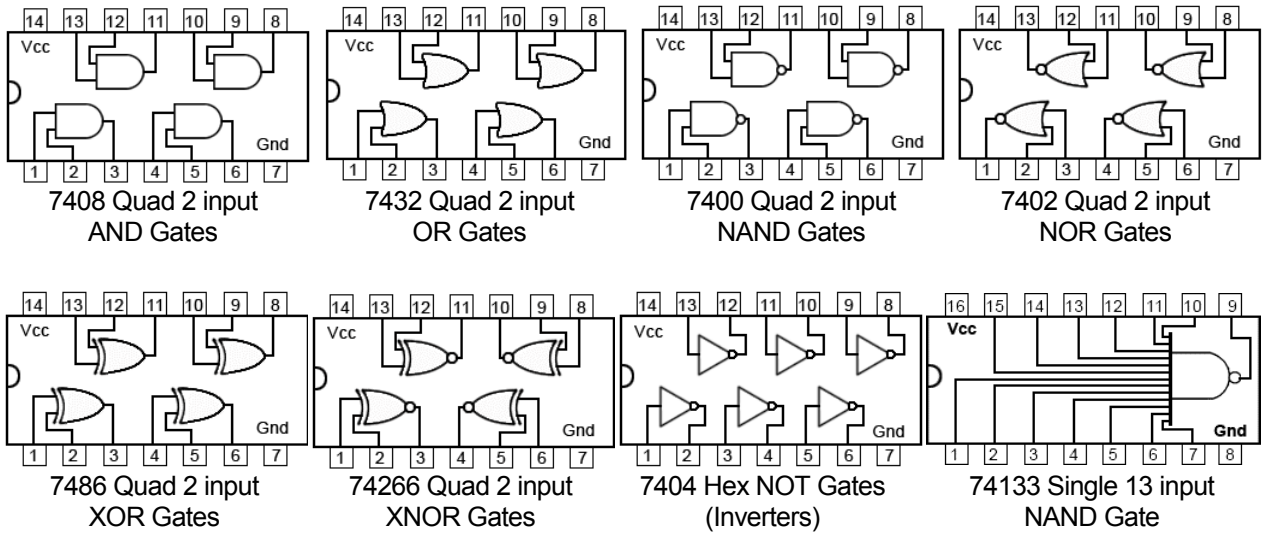
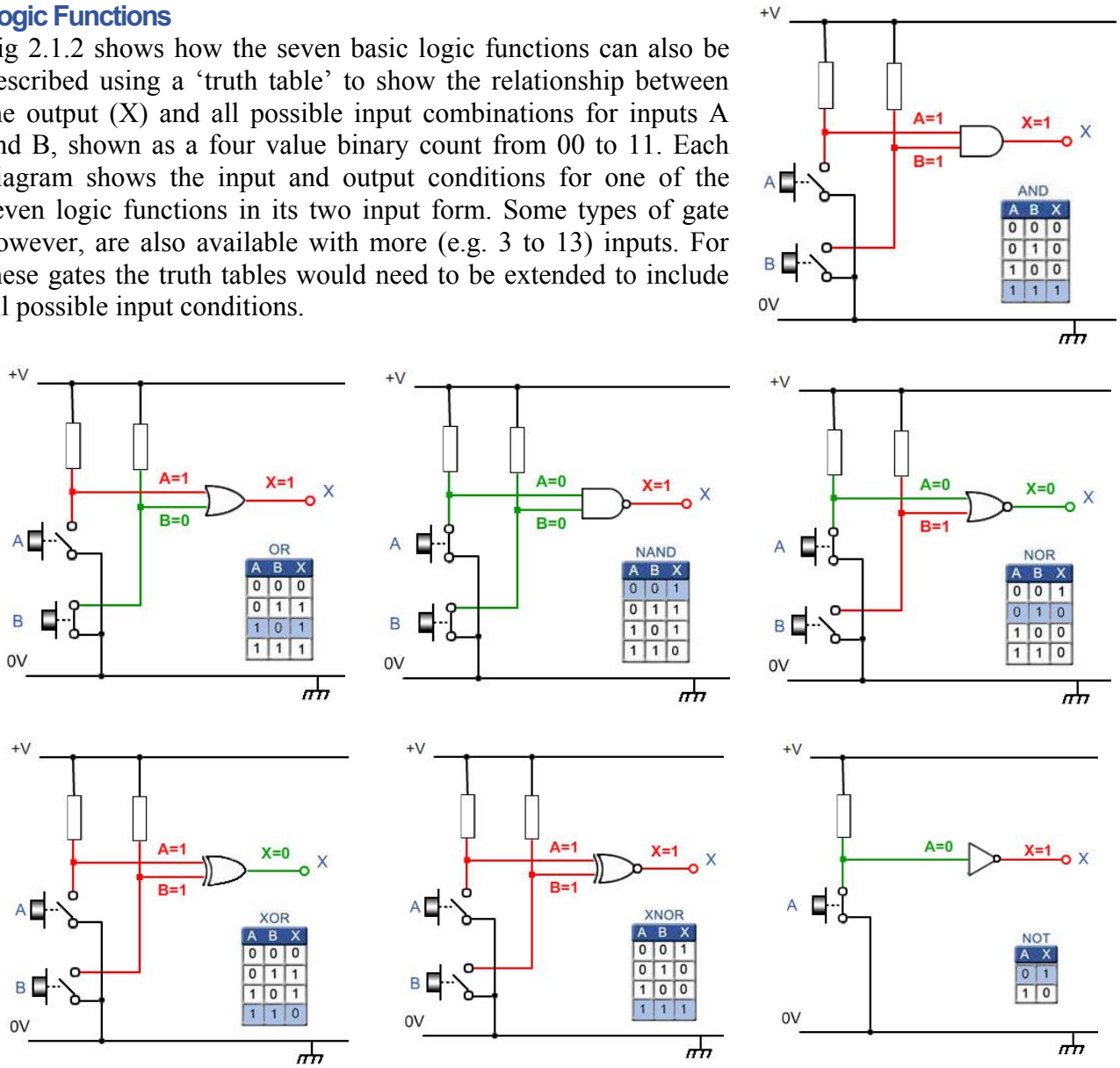


Fig. 2.1.1 Logic Gates From the 74 series TTL IC Family

Logic Functions

Fig 2.1.2 shows how the seven basic logic functions can also be described using a 'truth table' to show the relationship between the output (X) and all possible input combinations for inputs A and B, shown as a four value binary count from 00 to 11. Each diagram shows the input and output conditions for one of the seven logic functions in its two input form. Some types of gate however, are also available with more (e.g. 3 to 13) inputs. For these gates the truth tables would need to be extended to include all possible input conditions.



Logic Functions

Universal Gates

Because gates are manufactured in IC form, typically containing two to six gates of the same type, it is often uneconomical to use a complete IC of six gates to perform a particular logic function. A better solution may be to use just a single type of gate to perform any of the logic operations required. Two types of gate, NAND and NOR are often used to perform the functions of any of the other standard gates, by connecting a number of either of these 'universal' gates in a combinational circuit. Although it may not seem efficient to use several universal gates to perform the function of a single gate, if there are a number of unused gates in one or more NAND and NOR ICs, these can be used to perform other functions such as AND or OR rather than using extra ICs to perform that function. This technique is especially useful in the design of complex ICs where whole circuits within the IC can be fabricated using a single type of gate.

Fig. 2.1.3 shows how NAND gates can be used to obtain any of the standard functions, using only this single gate type.

NOT Function

a. Connecting the inputs of the NAND gate together creates a NOT function.

b. Alternatively the NOT function can be achieved by using only 1 input and connecting the other input permanently to logic 1.

AND Function

c. Adding the NOT function (an inverter) to the output of a NAND gate creates an AND function.

OR Function

d. Inverting the inputs to a NAND gate creates an OR function.

NOR Function

e. Using a NOT function to invert the output of an OR function creates a NOR function.

XOR function

f. Four NAND gates (a single IC) connected as shown creates an XOR function (and a Quad NAND IC is about 15% cheaper than a Quad XOR IC).

XNOR Function

g. Inverting the output of the XOR function creates an XNOR function.

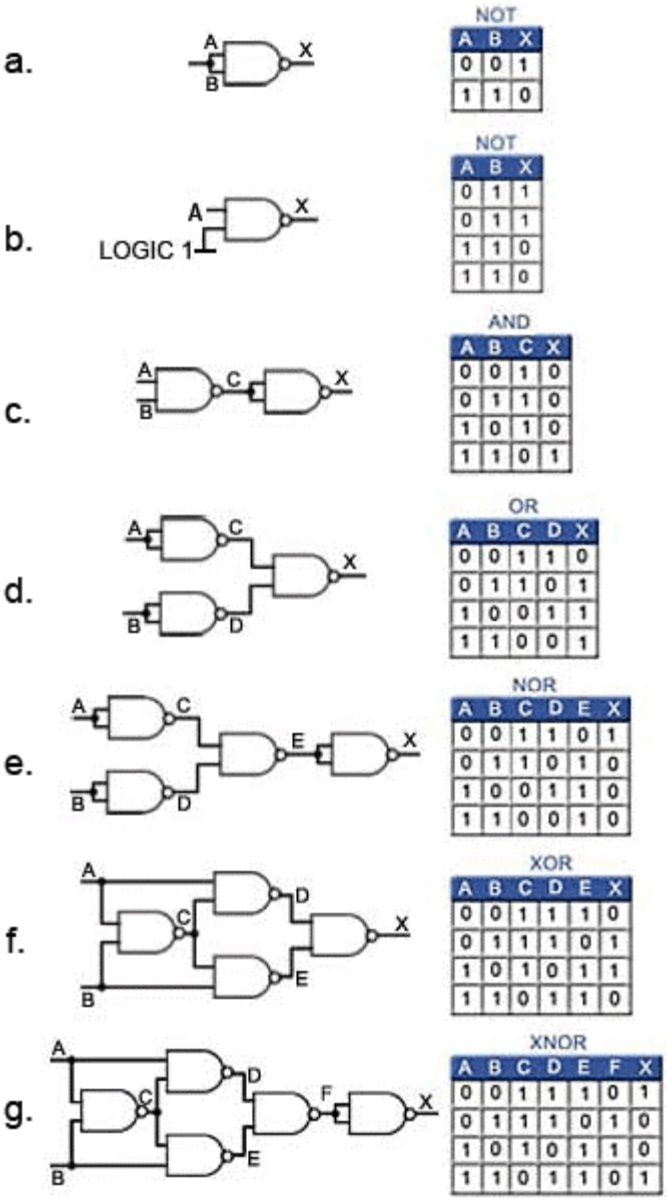


Fig 2.1.3 Creating any Logic Function Using NAND Gates

Similar conversions can be achieved using NOR gates, but as NAND gates are generally the least expensive ICs, the conversions shown in Fig. 2.1.3 are more frequently used. The reason for such conversions is usually cost. This may not seem very useful since none of the basic 74 series ICs are expensive, but when several thousand units of a particular circuit are to be manufactured, the small savings in cost and space on printed circuit boards by maximising the use of otherwise unused gates in multi gate ICs can become very important.

2.2 Combinational logic.

What you'll learn in Module 2.2

After studying this section, you should be able to:

Describe complex logic functions.

- Using truth tables.
- Using Boolean expressions.

Understand the relationship between truth tables and logic circuits.

- Analyse simple digital circuits using truth tables.
- Formulate Boolean equations from truth tables.
- Use truth tables to simplify logic circuits.

Combinational logic.

Combining a number of basic logic gates in a larger circuit to produce more complex logical operations is called combinational logic. Using such circuits, logical operations can be performed on any number of inputs whose logic state is either 1 or 0 and this technique is the basis of all digital electronics.

Combinational logic circuits can vary in complexity from simple combinations of two or three standard gates, to circuits containing hundreds of thousands, or even millions of gates. It is this ability to combine just a few simple gate circuits, which can be manufactured to microscopic dimensions, but in almost limitless combinations that makes digital electronics so powerful.

To understand the operation of a combinational logic circuit, and what logic state should be present at any particular point in the circuit, it is necessary to accurately analyse the operation of the circuit. For this purpose, several methods can be used, depending on the complexity of the circuit. These include truth tables, Boolean algebra, Karnaugh maps and computer software methods.

Truth Tables.

A truth table can be used for analysing the operation of logic circuits. A simple example of a combinational logic circuit is shown in Fig. 2.2.1. To analyse its operation a truth table can be compiled as shown in the following tree steps. Firstly a number of columns are written down which will describe, using ones and zeros, all possible conditions that can occur at the inputs and outputs of the circuit. For the circuit in Fig 2.2.1, three inputs A, B and C are used.

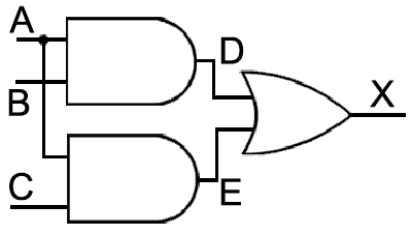


Fig 2.2.1 Combinational Logic

Step 1

Three columns marked A, B and C are needed, filled with a binary count from 000 to 111, i.e. a decimal count from 0 to 7. These columns now contain ALL possible input conditions because three inputs can have only 2^3 (eight) combinations of 1 and 0. More inputs would of course have more possible combinations, but as long as a binary count is used with one column per input, all possible input conditions are covered.

Step 2

Two more columns are added next, for the intermediate points D and E in the circuit, showing in column D, the result of 'ANDing' columns A and B, and in column E the results of 'ANDing' columns A and C. Each column is labelled with a Boolean expression for that particular gate output.

Step 1		
Inputs		
A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Step 2				
Inputs			A•B	A•C
A	B	C	D	E
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

Step 3					
Inputs			A•B	A•C	D+E
A	B	C	D	E	X
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	0	1	1

Table 2.2.1 Making a Truth Table

Each cell in columns D and E is filled with the appropriate 1 or 0 by working out the logic state that would occur at that gate output for the given inputs. In this case each column follows the rule for an AND gate, illustrated in Digital Electronics Module 2, Table 2.1.1.

Step 3

Then the final column X is completed by 'ORing' the intermediate columns D and E. This final column now shows all the logic states at the output X for any combination of logic states at the inputs A, B and C. A truth table produced in this way is also very valuable in fault finding in combinational logic circuits, as it shows the logic states at any point in the circuit for a given combination of inputs. These may be checked against the actual operation of the circuit to reveal faults.

Circuit Simplification Using Truth Tables

Creating a circuit from a truth table reverses to the process described above, and looking at Table 2.2.1 it can be seen that a logic 1 is produced at output X whenever the circuit inputs A, B and C are at logic 1. This can be described by compiling an appropriate Boolean equation from the truth table, which shows that X is 1 (is true) when A and B are 1, or when A and C are 1, or when A and B and C are 1. This can be written as:

$$X = (A \bullet B) + (A \bullet C) + (A \bullet B \bullet C)$$

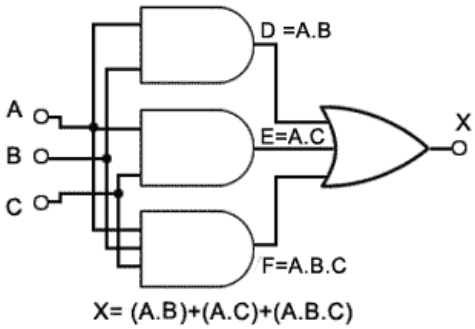


Fig. 2.2.2 Three Input Combinational Logic Circuit

The circuit therefore provides a logic 1 output at X for any input combination where the binary value of the inputs is greater than 100_2 (4_{10}). Building a circuit to implement the Boolean equation would give the result shown in Fig. 2.2.2. Notice however, that this circuit gives the same output as the original circuit in Fig 2.2.1 so could the simpler circuit of Fig. 2.2.1 do the job just as well?

The Boolean equation derived from Table 2.2.1 suggests that a more complex circuit, as shown in Fig 2.2.2 would be needed, which requires two 2 input AND gates for columns D and E and a three input AND gate for column F. These are then 'ORed' together by a 3 input OR gate to provide the single output X.

Compiling a truth table for Fig. 2.2.2 to check its operation produces Table 2.2.2. The output column X shows that the circuit in Fig. 2.2.2 does give the same outputs as Fig. 2.2.1. However, although a logic 1 at X is produced on the bottom row, where all three inputs (A•B•C) are logic 1, the third row up from the bottom of the table where A•C (shaded cells) also provides a logic 1 in column E and at output X.

Table 2.2.2

Inputs			A•B	A•C	A•B•C	D+E+F
A	B	C	D	E	F	X
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	✗	✗	✗	1

Therefore it doesn't matter whether columns D, E or F in the bottom row are at logic 1 or not. With the inputs at 111 the logic 1s on inputs A and C will still produce a logic 1 at E and therefore logic 1 at the output X. The bottom row for Columns D, E an F can therefore be marked with ✗ to indicate "Don't Care", it doesn't matter whether these cells are 1 or 0, column X will still be logic 1.

This means that column F (and the three input AND gate) are not needed, also the three input OR gate can be replaced by a two input OR gate.

Although the circuit shown in Fig. 2.2.2, designed from a Boolean equation derived directly from a truth table, does give the required output, the simpler (and cheaper) circuit shown in Fig. 2.2.1 does the job just as well. Using a truth table in this way will certainly give workable results and produce

a working circuit, however it may not be the best circuit. In this case, the Boolean equation could be reduced and simplified by getting rid of the redundant A•B•C. The simplified circuit produced is then adequately described by the shorter Boolean equation:

$$X = (A \bullet B) + (A \bullet C)$$

This shows that although truth tables are an excellent method for analysing the operation of a digital circuit, they may not be the best design tool, when used on their own, for arriving at the simplest design. Simplifying circuit design using truth tables does require some practice in reading the truth table, although possible simplifications are still much easier to see in the truth table than by trying to visually analyse the circuit schematic diagram. However, with more complex circuits and more than two or three inputs, simplification using truth tables becomes a very laborious process, and therefore more prone to errors. For circuits using up to three or four inputs, better results can be obtained by the direct manipulation of the Boolean algebra equations obtained from a truth table.

2.3 Boolean Algebra.

What you'll learn in Module 2.3

After studying this section, you should be able to:

Describe logic circuits using Boolean equations.

- Create Boolean expressions for intermediate gate outputs.
- Use complex Boolean equations to describe complete logic circuits.

Simplify Boolean equations using Boolean laws.

- Commutative.
- Associative.
- Distributive.
- Identity.
- Complement.
- Reduction.
- Duality.
- De Morgan's Theorem.

Use De Morgan's theorem to convert multiple gate circuits to universal gates.

Digital Electronics Module 2.1 showed that the operation of a single gate could be described by using a Boolean expression. For example the operation of a single AND gate with inputs A and B and an output X can be expressed as:

$$X = A \bullet B$$

Note:

The symbol • represents a logical AND, but because the use of special symbols can be inconvenient in printed material, the AND symbol is often omitted, as in AB or separated by a full stop as in A.B as used to indicate multiplication in standard algebra. The multiplication symbols x and * can also be seen in some texts, because the logical AND is similar to binary multiplication, (but not the same when numbers having more than one-bit are used).

Module 2.2 showed the relationship between a truth table that describes the operation of a circuit, and a Boolean equation that describes the logic of the circuit.

A combinational logic circuit such as that shown in Fig 2.3.1 (a repeat of Fig 2.2.2) is described by a Boolean equation as:

$$X = (A \bullet B) + (A \bullet C) + (A \bullet B \bullet C).$$

This could also be written (less clearly) as "The output X will be 1 when A and B or A and C or A and B and C are 1, otherwise X will be 0".

However Module 2.2 also showed that although a Boolean equation may give an accurate description of a logic process described by a truth table, it might require simplification before being interpreted as an actual circuit. The circuit shown in Fig 2.3.1 was simplified in Module 2.2 by closely examining a truth table to find redundant gates. However, with anything but the simplest circuits this can be tedious and it is easy to make mistakes.

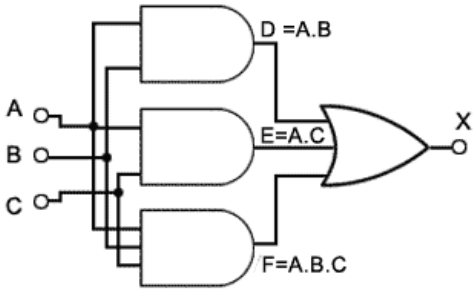


Fig. 2.3.1 Three Input Combinational Circuit with Redundant Gates

This module therefore describes methods for simplifying Boolean equations directly, using Boolean algebra, rather than by the use of truth tables.

Circuit Simplification Using Boolean Algebra

The algebraic method used to simplify digital circuits applies a number of Boolean laws to successively simplify complex equations. Selected laws and rules are applied, step by step, to the original equation, so as to eventually arrive at a simplified version that can be implemented with a smaller number of gates and therefore lead to a simpler circuit.

Boolean Laws

The laws of Boolean algebra are similar in some ways to those of standard algebra, but in some cases Boolean laws are unique. This is because when logic is applied to digital circuits, any variable such as A can only have two values 1 or 0, whereas in standard algebra A can have many values.

Commutative Laws

In a group of variables connected by operators AND or OR, the order of the variables does not matter.

1a. Boolean addition (OR): A+B = B+A

1b. Boolean multiplication (AND): A•B = B•A

Associative Laws

The order of calculation can be changed without affecting the result (Change which terms are in brackets, or remove brackets). Note: This is only OK so long as all signs (+ or •) are the same.

2a. Boolean addition (OR): (A+B)+C = A+(B+C) = A+B+C

2b. Boolean Multiplication (AND): (A•B)•C = A•(B•C) = A•B•C = ABC

Distributive Laws

The same answer is arrived at when multiplying (ANDing) a variable by a group of bracketed variables added (ORed) together, as when each multiplication (AND) is performed separately.

Law 3a is similar to factoring in normal algebra, but law 3b is unique to Boolean algebra because unlike normal algebra, where A x A=A^2, in Boolean algebra A•A = A

3a. A•(B+C) = A•B+A•C

3b. A+(B•C) = (A+B) • (A+C)

Identity Elements

In rule 4a, when the variable A is ANDed with logic 1 (called the Identity Element for the AND operator). The variable ANDed with 1 retains its identity.

Rule 4b, shows that the Identity Element for the OR operator is 0, and any variable (e.g. A) ORed with 0 it retains its identity.

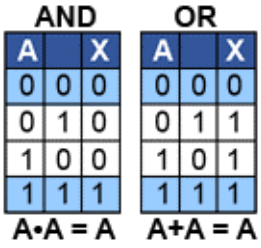
4a. A•1 = A

4b. A+0 = A

5a and 5b show how by 'forcing the Identity Element', (in B column of the truth tables) to the opposite states to those used in 4a and 4b, produces an output that is the same as the Identity Element.

5a. A•0 = 0

5b. A+1 = 1



6a and 6b show that ANDing or ORing two identical variables, produces an output equal to a single variable, showing that one of the variables is redundant, a useful rule when simplifying Boolean equations.

6a. A•A = A

6b. A+A = A

Complement Law

7a. $A + \overline{A} = 1$ Any variable ORed with its inverse is 1

7b. $A \bullet \overline{A} = 0$ Any variable ANDed with its inverse is 0

Note:

$\overline{\overline{A}} = A$ Double inversion (NOT NOT) returns the variable to its previous state.

Reduction

8a. When a single variable (A) is ANDed with itself OR a second variable (A+B), the result is equal to the single variable.

8a $A \bullet (A+B) = A$

8b. When a single variable (A) is ORed with itself AND a second variable (A•B), the result is equal to the single variable.

8b $A + (A \bullet B) = A$

8c. When a single variable (A) is ORed with itself OR a second variable (A+B), the single variable disappears.

8c $A + (A+B) = (A+B)$

8d. When a single variable (A) is ANDed with itself AND a second variable (A•B), the single variable disappears.

8d $A \bullet (A \bullet B) = (A \bullet B)$

Table 2.3.1 The Reduction Rules

8a				8b			
A	B	A+B	A•(A+B)	A	B	A•B	A+(A•B)
0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0
1	0	1	1	1	0	0	1
1	1	1	1	1	1	1	1

8c				8d			
A	B	A+B	A+(A+B)	A	B	A•B	A•(A•B)
0	0	0	0	0	0	0	0
0	1	1	1	0	1	0	0
1	0	1	1	1	0	0	0
1	1	1	1	1	1	1	1

Duality Rules

It is possible to derive additional identities by obtaining the **Dual** of an identity. This involves changing the AND operators to OR and the OR operators to AND. Additionally any 0s are changed to 1s and 1s to 0s as shown in Table 2.3.2.

Table 2.3.2

Identity	Dual
$\overline{0} = 1$	$\overline{1} = 0$
$A + 1 = 1$	$A \bullet 0 = 0$
$A + A = A$	$A \bullet A = A$
$A + \overline{A} = 1$	$A \bullet \overline{A} = 0$

The duality rule can be used to change a logic expression containing both AND and OR elements to its equivalent dual expression.

Table 2.3.3 shows that A•(B+C) is the same as A+(B•C).

Table 2.3.3

A	B	C	A•(B+C)	A+(B•C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Simplifying Boolean Equations

Minimising complex Boolean expressions to their simplest form using Boolean laws and rules is a matter of choosing the most appropriate law or rule to reduce the expression step by step. If the resulting minimisation is correct, the minimised equation and the original equation should give identical output columns when truth tables for the original and minimised circuits are compared.

These Boolean algebraic methods would normally be used on logic circuits with just a few gates and only two or three inputs, as this method of simplification becomes more difficult and cumbersome to use when more gates or inputs are involved.

Which laws are applied to change an equation, and in what order is a matter of practice and intuition. This method involves looking at the original complex equation and selecting a law that will simplify a particular part, so obtaining a simpler equation, and then choosing another law that will simplify the equation further, and so on until the equation can no longer be made simpler.

There is no simple algorithm to specify the order of steps to be taken and several routes may be taken to reach the goal of a simplified and ideally minimised circuit.

Whether the result is also the minimum possible circuit can only be judged by looking for any possible further reduction using the Boolean laws.

In practice, small circuits with just two or three inputs can very often be simplified just by looking at the truth table and picking out any redundant logic combinations, as shown in Table 2.2.2 in Module 2.2 (Combinational Logic), but Boolean simplification is useful for more complex circuits.

Boolean Simplification Examples

Example 1

Suppose the cash room at a store has access restricted to certain employees, each of who has a key, which produces a logic 1 at particular inputs to an unlocking circuit.

Only the store manager (M) can enter alone. The assistant manager (A) and the cashier (C) also have access, but only when accompanied by each other, or by the store manager. Design a combinational logic circuit that will allow access by producing a logic 1 when the above conditions are met.

Table 2.3.4

A	M	C	X	Boolean
0	0	0	0	
0	0	1	0	
0	1	0	1	M
0	1	1	1	M • C
1	0	0	0	
1	0	1	1	A • C
1	1	0	1	A • M
1	1	1	1	A • C • M

Truth table

The conditions requiring a logic 1 output can be arranged as a truth table (Table 2.3.4) and Boolean expressions can be derived from the truth table for each input combination that produces a logic 1 output.

The five Boolean AND expressions can be separated by OR operators to form a complete Boolean equation.

$$X = M + M•C + A•C + A•M + A•C•M$$

This suggests a circuit like that shown in Fig. 2.3.2, which would require four I.Cs:

1x 74HCT08 2 input AND (containing 4 gates).

1x 74HCT10 3 input AND (containing 3 gates).

1x 74HCT32 2 input OR (containing 4 gates).

1x 74HCT4075 3 input OR (containing 3 gates).

However, by choosing appropriate laws and rules from those listed above, the circuit can be considerably simplified.

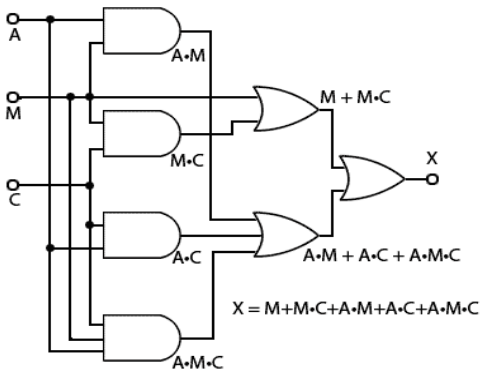


Fig. 2.3.2 Cash Room Access Circuit

Starting with the equation derived from Table 2.3.4:

X = M + M•C + A•C + A•M + A•C•M

Since *M + M•C = M (Reduction rule 8b)*

X = M + A•C + A•M + A•C•M

And as *M + A•C + A•M = M + A•M + A•C (Commutative Law 1a)*

X = M + A•M + A•C + A•C•M

And as *M + A•M = M (Reduction rule 8b)*

X = M + A•C + A•C•M

And as *M + A•C + A•C•M = M + A•C•M + A•C (Commutative Law 1a)*

X = M + A•C•M + A•C

And as *M + A•C•M = M (Reduction rule 8b)*

X = M + A•C

No further reduction possible.

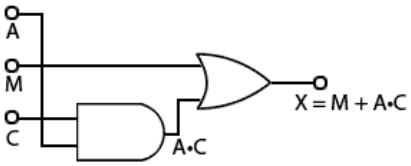


Fig. 2.3.3 Simplified Cash Room Access Circuit

Table 2.3.5				
A	M	C	X	M + A•C
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

The simplified circuit is shown in Fig 2.3.3, which provides exactly the same function as Fig. 2.3.2. This can be confirmed by comparing the simplified equation X = M + A•C with the original column X in Table 2.3.5.

The simplified circuit in Fig 2.3.3 still requires two I.Cs, (AND and OR) and it now only uses one gate per chip. Unless the spare gates are to be used elsewhere in another part of the circuit, this is still wasteful.

A better option could be to replace the AND and OR functions with universal gates such as NOR or NAND. A 'NAND only' version of the simplified circuit is shown in Fig. 2.3.4. This version uses three gates instead of two, but all the gates are the same and can be accommodated within a single 7400 IC. The original circuit has therefore now been reduced from four ICs to one.

NAND Circuit Operation

NAND 1 has both its inputs connected together, which converts it to an inverter or NOT gate and therefore produces $\overline{M}$ at its output. NAND 2 works as an AND gate with its output inverted and so produces an output of $\overline{A \bullet C}$. The Boolean expression for the circuit using NAND gates now becomes:

X = M + A•C

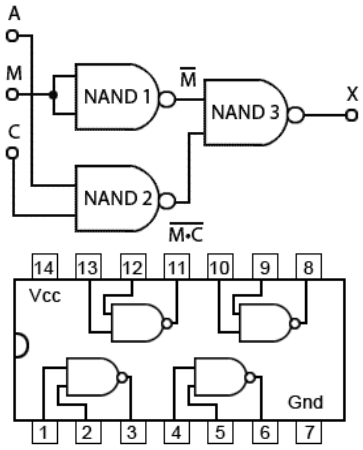


Fig. 2.3.4 NAND only Cash Room Access Circuit

The inversion bars have now disappeared because of the application of another very useful Boolean algebra law, De Morgan's Theorem. Before looking at how the theorem works, note the difference in the use of the inversion bars in Boolean expressions. This is an important feature in the application of De Morgan's Theorem:

A broken bar $\overline{A} \bullet \overline{B}$ indicates that the inputs (to an AND gate for example) are inverted, whilst an unbroken bar $\overline{A \bullet B}$ indicates an inverted output.

De Morgan's Theorem

Augustus De Morgan formulated an extension to George Boole's Algebraic logic that has become very important in digital logic. Not only is it used in the simplification of Boolean expressions but can also be used to change the function of logic gates, so that NAND gates (or NOR gates) can carry out any of the other standard logic functions of gates. The theorem comprises two laws that describe how inverting the inputs to a gate, changes the gate's function.

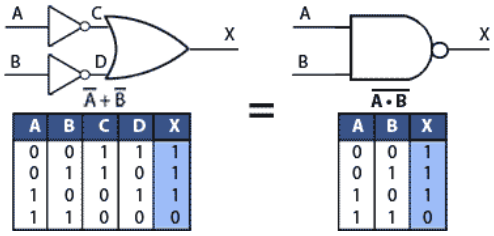
Law 1. $\overline{A} + \overline{B} = \overline{A \bullet B}$ Inverting the inputs to an OR gate changes its function to NAND.

Law 2. $\overline{A} \bullet \overline{B} = \overline{A + B}$ Inverting the inputs to an AND gate changes its function to NOR

Considering these two equalities the other way round, $\overline{A + B} = \overline{A} \bullet \overline{B}$ De Morgan's Theorem is often described as "Break the bar and change the sign." Meaning that by placing, or removing the bar above the AND or OR operator (• or +) the operator is inverted. Therefore the complement of the AND function is OR.

Applying De Morgan's Theorem

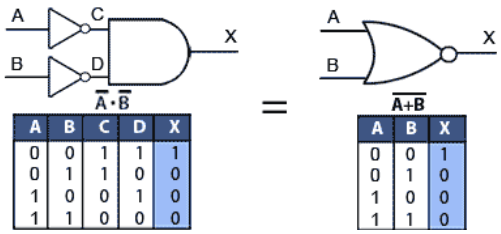
These equalities, generally called De Morgan's Laws 1 and 2 are illustrated in Fig. 2.3.5 and Fig. 2.3.6 as they apply to AND, NOR, NAND and OR gates. Note however, that when De Morgan's theorem is applied to the XOR and XNOR gates there is no change in the gate's function.



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

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

Fig. 2.3.5 De Morgan's Law 1



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

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

Fig. 2.3.6 De Morgan's Law 2

The usefulness of De Morgan's theorem when applied to circuits can be seen by comparing Fig. 2.3.3 and Fig. 2.3.4 where it was instrumental in changing the functions of the AND and OR gates in Fig.2.3.3 to all NAND gates in Fig. 2.3.4, so the circuit can be made using one IC instead of two.

Inverting the Inputs

In Fig. 2.3.4 an additional gate NAND 1 appears in the circuit, and has its two inputs connected together to act as a NOT gate (check this by looking at the truth table for a NAND gate in Fig. 2.3.5), when both inputs are the same (row 1 and row 4) the output (X) is an inverted version of the inputs (A•B).

This additional gate in Fig. 2.3.4 provides $\overline{M}$ at the top input to NAND 3 instead of the M applied to the top input of the OR gate in Fig 2.3.3.

NAND 2 in Fig. 2.3.4 replaces the AND gate in Fig 2.3.3 so that the bottom input to NAND 3 is now $\overline{A \bullet C}$ instead of A•C.

Therefore inputs to NAND 3 are now $\overline{M}$ and $\overline{M \bullet C}$. Therefore both inputs to NAND 3 have been inverted (without actually using any NOT gates) to make NAND 3 act, according to De Morgan's theorem, as an OR function, so giving the correct output of $X = M + A \bullet C$.

Summary

Boolean algebra gives a more compact way to describe a combinational logic circuit than truth tables alone. It can also be used for simplification of circuits, however this can also be cumbersome and error prone. When circuits with more than two or three inputs are involved a better method of circuit reduction that works well with circuits having up to four or six inputs is the Karnaugh Map.

2.4 Karnaugh Maps

What you'll learn in Module 2.4

After studying this section, you should be able to:

Understand the use of Karnaugh maps.

- Draw maps for Multi input circuits.
- Use Gray code notation.
- Derive Karnaugh maps from truth tables.

Uses Karnaugh Maps.

- Group Karnaugh map cells.
- Simplify logic circuits.
- Produce minimised Boolean equations.

Make choices in cell selection to achieve a desired circuit result.

- Cost reduction.
- Propagation delay.

Understand manual and software based Boolean minimisation.

- Minimise a complex Boolean equation using appropriate software.

Why Karnaugh Maps?

Karnaugh Maps offer a graphical method of reducing a digital circuit to its minimum number of gates. The map is a simple table containing 1s and 0s that can express a truth table or complex Boolean expression describing the operation of a digital circuit. The map is then used to work out the minimum number of gates needed, by graphical means rather than by algebra. Karnaugh maps can be used on small circuits having two or three inputs as an alternative to Boolean algebra, and on more complex circuits having up to 6 inputs, it can provide quicker and simpler minimisation than Boolean algebra.

Constructing Karnaugh Maps

The shape and size of the map is dependent on the number of binary inputs in the circuit to be analysed. The map needs one cell for each possible binary word applied to the inputs.

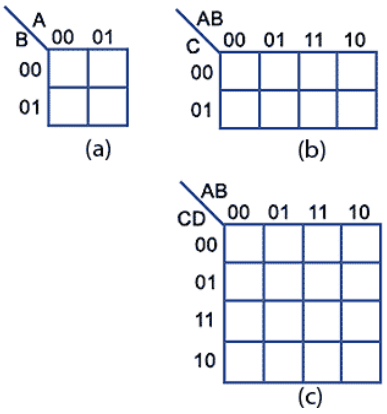


Fig. 2.4.1 Karnaugh Maps

Therefore:

2 input circuits with inputs A and B require maps with $2^2 = 4$ cells (Fig 2.4.1a).

3 input circuits with inputs A B and C require maps with $2^3 = 8$ cells (Fig 2.4.1b).

4 input circuits with inputs A B C and D require maps with $2^4 = 16$ cells (Fig 2.4.1c).

The input labels are written at the top left hand corner, divided by a diagonal line. The top and left edges of the map then represent all the possible input combinations for the inputs allocated to that edge.

For example, in the 3 input map (b) in Fig. 2.4.1, the top edge of the map represents the 4 possible combinations for inputs A and B, so the cells are labelled 00,01, 11, and 10 (See *Important note).

Because example (b) in Fig. 2.4.1 is a 3 input map, input C on the left hand edge only has two possible combinations, 0 and 1. This map is therefore rectangular rather than square to cover the 8 possible combinations available from 3 inputs.

*Important

Notice that this edge numbering does not follow the normal binary counting sequence, but uses a Gray Code sequence where only one bit changes from one cell to the next. This is an important feature of Karnaugh maps; get the sequence wrong and the map will not work!

Using the Karnaugh Map

The Karnaugh map can be populated with data from either a truth table or a Boolean equation.

As an example, Table 2.4.1 shows the truth table for the 3 input 'cash room' example, together with the Boolean expressions derived from each input combination that results in a logic 1 output. This results in a Boolean equation for the un-simplified circuit:

$$X = M + M{\bullet}C + A{\bullet}C + A{\bullet}M + A{\bullet}M{\bullet}C$$

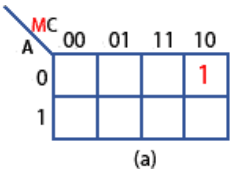
This table will serve to show the process of transferring the data from Table 2.4.1 into the cells of the Karnaugh map. The Process is shown step by step in Fig. 2.4.2

Table 2.4.1

A	M	C	X	Boolean
0	0	0	0	
0	0	1	0	
0	1	0	1	M
0	1	1	1	M • C
1	0	0	0	
1	0	1	1	A • C
1	1	0	1	A • M
1	1	1	1	A • M • C

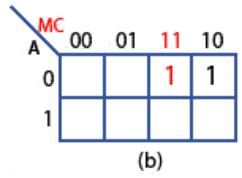
Step (a)

From Table 2.4.1 row 3, inputs AMC have values of 010, producing a logic 1 at the output (X) and giving the Boolean expression M in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=0 and MC=10 as shown at (a) in Fig. 2.4.2.



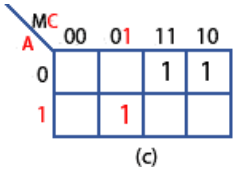
Step (b)

In Table 2.4.1 row 4, inputs AMC have values of 011, producing a logic 1 at the output (X) and giving the Boolean expression MC in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=0 and MC=11 as shown at (b) in Fig. 2.4.2.



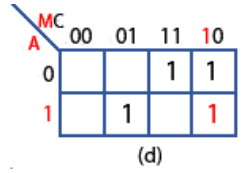
Step (c)

In Table 2.4.1 row 5, output (X), is 0 so this row is ignored. However, in row 6, inputs AMC have values 101, producing a logic 1 at the output (X) and giving the Boolean expression AC in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=1 and MC=01 as shown at (c) in Fig. 2.4.2.



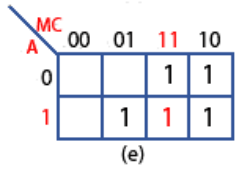
Step (d)

In Table 2.4.1 row 7, the inputs AMC have values of 110, producing a logic 1 at the output (X) and giving the Boolean expression AM in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=1 and MC=10 as shown at (d) in Fig. 2.4.2.



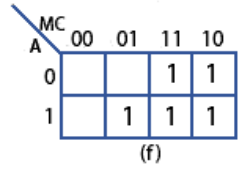
Step (e)

Finally, in Table 2.4.1 row 8 the inputs AMC have values of 111 producing a logic 1 at the output (X) and giving the Boolean expression of AMC in the Boolean column. Therefore 1 is placed in the map cell corresponding to A=1 and MC=11 as shown at (e) in Fig. 2.4.2.



The completed map (f)

All the truth table rows that produced a logic 1 have now been entered into the map and those lines that produced a logic 0 can be ignored, so the remaining three cells are left blank. Later it will be shown that these blank cells can be useful when mapping larger circuits, but for now the map is ready for simplification.



Simplifying Karnaugh Maps

Circuit simplification in any Karnaugh map is achieved by combining the cells containing 1 to make groups of cells. In grouping the cells it is necessary to follow six rules.

How these rules are applied is illustrated using a four input 16-cell map shown in Fig. 2.4.3.

Karnaugh Map Rules

1. Groups should contain as many '1' cells (i.e. cells containing a logic 1) as possible and no blank cells.
2. Groups can only contain 1, 2, 4, 8, 16 or 32… etc. cells (powers of 2).
3. A '1' cell can only be grouped with adjacent '1' cells that are immediately above, below, left or right of that cell; no diagonal grouping.
4. Groups of '1' cells can overlap. This helps make smaller groups as large as possible, which is an advantage in finding the simplest solution.
5. The top/bottom and left/right edges of the map are considered to be continuous, as shown in Fig. 2.4.3, so larger groups can be made by grouping cells across the top and bottom or left and right edges of the map.
6. There should be as few groups as possible.

Map (a) follows rules 2, 3 and 4 and shows three groups containing 8, 4 and 2 cells. This will simplify the circuit being produced, but it is not optimum.

Map (b) shows an improvement, still with 3 groups but they now contain 8, 4 and 4 cells. This map takes advantage of rule 5 by joining the 2 cells ringed in green in Map (a) with the top two cells in the blue group, see Map (b) to form a group of 4 (ringed in cyan) instead of a group of 2. The map now conforms to all 6 rules.

Map (c) (for a different circuit) shows how a potentially single '1'cell (second cell from the bottom in the right hand column) can be grouped with two other cells in the blue group, and one cell in the green group, to make a (cyan) group of 4.

Sometimes however there may be a single cell that cannot be joined with other groups, as shown in map (d). Rule 3 prohibits diagonal grouping so there is no alternative other than to leave a group of 1. This is permissible, but in map (d), which represents a four input circuit, the simplified Boolean equation will contain an un-simplified expression relating to the single cell, which will have all four possible terms e.g. A•B•C•D.

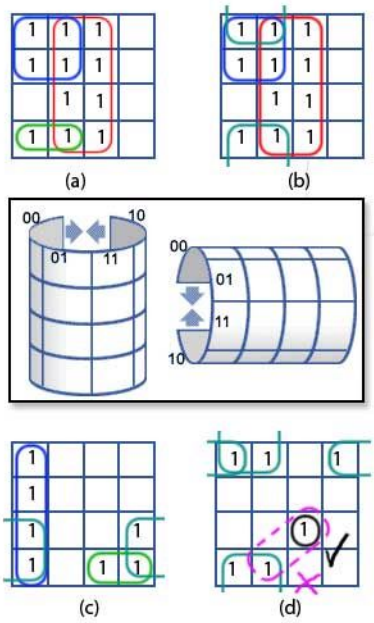


Fig. 2.4.3 Typical Karnaugh Groups

Example:

Using the Karnaugh map rules on the three input map created from Table 2.4.1, there are just 2 possible groups, as shown in Fig. 2.4.4. The next task is to simplify the original Boolean equation for this circuit:

$$X = M + M•C + A•C + A•M + A•M•C$$

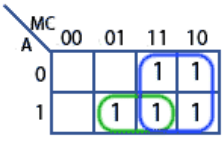


Fig 2.4.4 Groups for Example 1

Converting the two groups in the Karnaugh map to Boolean expressions is done by discovering which input or inputs (A, M or C) does NOT change within each group.

Step 1

Taking the (blue) group of 4 first, notice that it spans two rows

vertically, and so contains rows A=0 and A=1, therefore A changes within the group so cannot appear in the expression.

The blue group also spans two columns and so contains MC=11 and MC=10. Here, C = both 1 and 0, but M=1 in both columns.

Therefore the only input that does not change in the blue group is M, so the Boolean expression for the blue group is simply M.

Step 2

Looking at the (green) group of 2, A does not change but MC changes from 01 to 11. This indicates that although M changes, C does not. Therefore there are two non-changing inputs in this group A and C.

Putting the results of the simplification together by 'ANDing' any non-changing inputs within a single group, and 'ORing' the different groups, produces the simplified Boolean equation for the whole circuit:

$$X = M + A \bullet C$$

This result agrees with the simplification produced in Module 2.3 using Boolean algebra. The main advantage of using a Karnaugh map for circuit simplification is that the Karnaugh method uses fewer rules, and these rules can be applied systematically rather than intuitively as with Boolean algebra. Therefore with a little practice the Karnaugh system should produce more reliable minimisation. Although Karnaugh mapping may have only slight advantages over Boolean algebra in simple circuits, the advantages become more apparent when minimising more complex circuits.

Karnaugh Minimisation of a 4 Input Circuit

With four-input circuits, Karnaugh maps become more useful, compared with minimisation using Boolean algebra alone.

Table 2.4.2 shows an example of a truth table for a BCD to 7 segment decoder, the purpose of this circuit is to illuminate the LEDs (or activate the LCD segments) that make up typical numerical displays.

Fig. 2.4.5 LED 7 Segment Display

As shown in Fig. 2.4.5, a typical display consists of 7 LEDs arranged in a figure of 8 formation. The LEDs (labelled a to g) must be activated independently to make up the numbers 0 to 9. Because 9 is the highest number that can be displayed, the usual data driving each digit of the display is in the form of '8421 Binary Coded Decimal', which restricts the range of the binary data to between 0000_2 and 1001_2.

The truth table for a BCD to 7 segment decoder is shown in Table 2.4.2 and demonstrates the relationship between the four inputs ABC and D, and each of the display LEDs.

In columns a to g, an output of logic 1 lights one particular segment of the display. Logic 0 turns it off. An X output is called a 'Don't Care' as it does not matter what the possible binary value would be in the BCD input columns A to D as they will not occur, (BCD will not produce values higher than 9_{10} or 1001_2). The value of including these 'Don't Care' outputs however, will be seen when working on the Karnaugh maps.

Table 2.4.2

Decimal	BCD Inputs				7 Segment Outputs						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	X	X	X	X	0	0	0	0	0	0	0
11	X	X	X	X	0	0	0	0	0	0	0
12	X	X	X	X	0	0	0	0	0	0	0
13	X	X	X	X	0	0	0	0	0	0	0
14	X	X	X	X	0	0	0	0	0	0	0
15	X	X	X	X	0	0	0	0	0	0	0

Notice that it is the convention to list the BCD input columns A to D in reverse order, making A represent the least significant digit and D, the most significant digit.

Designing a Decoder Circuit

The processes, and some of the choices to be made when using Karnaugh maps to minimise the digital circuits derived from complex truth tables such as Table 2.4.2 can be illustrated by creating a circuit to decode the 4 bit input to drive a single segment (segment 'a') of a 7 segment display. A similar process could be used to design circuits for each of the other six outputs b to g.

Table 2.4.3 illustrates the Boolean expressions derived from the BCD input columns that cause a logic 1 output at 'a'.

Segment 'a' must be illuminated when any of the numbers 0,2,3,5,6,7,8 or 9 are present at the decoder inputs as a BCD value. Therefore 8 Boolean expressions are derived from Table 2.4.3, which will cause the decoder circuit to output logic 1 for these inputs.

Table 2.4.3

Decimal	BCD Inputs					
	D	C	B	A	a	Boolean
0	0	0	0	0	1	$\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot \overline{A}$
1	0	0	0	1	0	
2	0	0	1	0	1	$\overline{D} \cdot \overline{C} \cdot B \cdot \overline{A}$
3	0	0	1	1	1	$\overline{D} \cdot \overline{C} \cdot B \cdot A$
4	0	1	0	0	0	
5	0	1	0	1	1	$\overline{D} \cdot C \cdot \overline{B} \cdot A$
6	0	1	1	0	1	$\overline{D} \cdot C \cdot B \cdot \overline{A}$
7	0	1	1	1	1	$\overline{D} \cdot C \cdot B \cdot A$
8	1	0	0	0	1	$D \cdot \overline{C} \cdot \overline{B} \cdot \overline{A}$
9	1	0	0	1	1	$D \cdot \overline{C} \cdot \overline{B} \cdot A$
10	X	X	X	X	0	
11	X	X	X	X	0	
12	X	X	X	X	0	
13	X	X	X	X	0	
14	X	X	X	X	0	
15	X	X	X	X	0	

The Boolean equation needed for the design of an appropriate circuit will therefore contain these 8 Boolean expressions, and so will be long and complex. It is therefore essential that such an equation is minimised, in order to make a practical circuit.

Decoder Minimisation Using Karnaugh Maps

The full Boolean formula for segment 'a' of the display, derived from Table 2.4.3 is:

$$a = (\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot \overline{A}) + (\overline{D} \cdot \overline{C} \cdot B \cdot \overline{A}) + (\overline{D} \cdot \overline{C} \cdot B \cdot A) + (\overline{D} \cdot C \cdot \overline{B} \cdot A) + (\overline{D} \cdot C \cdot B \cdot \overline{A}) + (\overline{D} \cdot C \cdot B \cdot A) + (D \cdot \overline{C} \cdot \overline{B} \cdot \overline{A}) + (D \cdot \overline{C} \cdot \overline{B} \cdot A)$$

Each of the individual AND expressions in the formula are now used to populate a 16 cell (four input) Karnaugh map with logic 1s, corresponding with the cell values for A B C and D around the edges of the map, as shown in Fig. 2.4.6.

For example, the cell in the second row down, and the first column from the left contains a logic 1 that is labelled $D \cdot \overline{C} \cdot \overline{B} \cdot \overline{A}$ corresponding to CD = 01 and AB = 00. Each logic 1 cell in the map is therefore equivalent to one of the Boolean expressions derived from Table 2.4.3. The complete Boolean formula for output ‘a’ is therefore contained in the Karnaugh map.

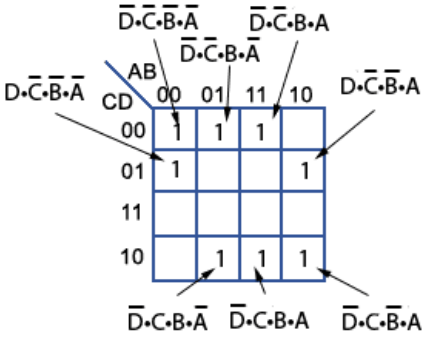


Fig. 2.4.6 Karnaugh map populated with Logic 1s

The logic 1 cells in the Karnaugh map can then be grouped as described in 'Simplifying Karnaugh Maps' to produce minimal Boolean expressions as shown in Fig. 2.4.7.

In this example one group of four (in blue) has been found, containing:

$(\overline{D} \cdot C \cdot B \cdot \overline{A}) + (\overline{D} \cdot C \cdot B \cdot A) + (\overline{D} \cdot \overline{C} \cdot B \cdot \overline{A}) + (\overline{D} \cdot \overline{C} \cdot B \cdot A)$

This group follows the Karnaugh Map Rule 5 and the cells that DON'T change are $B \cdot \overline{D}$, so this group simplifies to $\mathbf{B \cdot \overline{D}}$.

There is also a Karnaugh Map Rule 5 group of two (in red) containing:

$(D \cdot \overline{C} \cdot \overline{B} \cdot \overline{A}) + (D \cdot \overline{C} \cdot \overline{B} \cdot A)$

This group also follows the Karnaugh Map Rule 5 and the cells that DON'T change are $D \cdot \overline{C} \cdot \overline{B}$, so this group simplifies to $\mathbf{\overline{B} \cdot \overline{C} \cdot D}$.

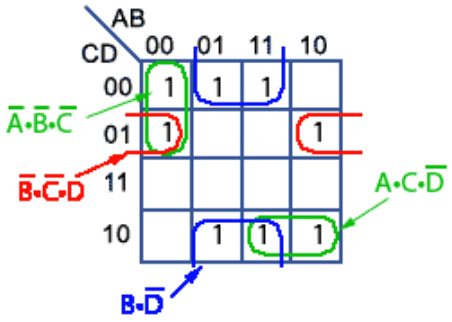


Fig. 2.4.7 Minimising the Karnaugh Map

Two further groups of two (green) simplify:

$(D \cdot \overline{C} \cdot \overline{B} \cdot \overline{A}) + (\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot \overline{A})$ **to** $\mathbf{\overline{A} \cdot \overline{B} \cdot \overline{C}}$

And:

$(\overline{D} \cdot C \cdot \overline{B} \cdot A) + (\overline{D} \cdot C \cdot B \cdot A)$ to $\mathbf{A \cdot C \cdot \overline{D}}$

This produces a simplified Boolean equation for output 'a' of:

$$\mathbf{a = (B \cdot \overline{D}) + (\overline{B} \cdot \overline{C} \cdot D) + (\overline{A} \cdot \overline{B} \cdot \overline{C}) + (A \cdot C \cdot \overline{D})}$$

This equation could be implemented as a circuit in a number of different ways, using AND OR and NOT gates, but Fig. 2.4.8 shows a circuit for the 'a' output, produced from the Karnaugh simplified equation, using NOT gates and the universal gates, NAND and NOR. Because there are not more than four of either NAND or NOR gates used, and less than 6 NOT gates, such a circuit would require one integrated circuit of each type, 3 in total.

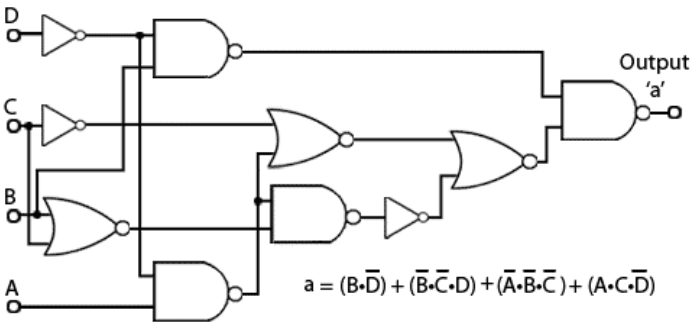


Fig. 2.4.8 Simplified Circuit for BCD to 7 segment decoder 'a' Output.

However, although this circuit has been produced from a simplified Boolean equation, this does not automatically mean that the circuit is fully minimised, so may not yet be in its most economical form.

Using 'Don't Care' Cells

The Karnaugh map in Fig. 2.4.7 used only logic 1s to make the simplified groups; if use is also made of the 'don't care' cells, larger groups can be made, resulting in shorter Boolean expressions.

Groups of one make 4 term expressions.

Groups of two make 3 term expressions.

Groups of four make 2 term expressions.

Groups of eight make 1 term expressions.

Including the 'don't care' cells will not change the correct 'a' output of the resulting circuit, as although the input combinations creating these cells could (in pure binary) occur, in BCD, numbers greater than 1001_2 (9_{10}) can't happen.

Fig. 2.4.9 shows the result of including the 'don't care' cells in the groups. This allows two (red) 8 cell groups and two (blue) 4 cell groups to be made. The cells containing 0 are still ignored, as they do not produce the required logic 1 outputs. The map now results in a minimised, and therefore simpler Boolean equation than in Fig. 2.4.7. All relevant cells are now grouped using the maximum possible sizes for the groups, which results in a Boolean equation of:

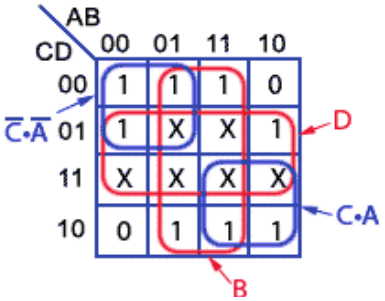


Fig. 2.4.9 Karnaugh Map with 'Don't Care' Cells Included

$$a = B + D + (C \bullet A) + (\overline{C} \bullet \overline{A})$$

This produces the circuit shown in Fig. 2.4.10, using only six gates instead of the ten required for Fig 2.4.8. The circuit now requires only two ICs. Additionally, as each gate has a finite propagation delay (the time from when the gate input becomes valid to the time when the output becomes valid), so the fewer gates there are between the circuit's input and output, the faster the data can be processed. Notice that in Fig. 2.4.10 one of the NAND gates has one input connected permanently to logic 1 to convert it to a NOT gate, as described in Module 2.1. As there are four two input gates per chip, using a spare NAND gate in this way saves the use of a NOT IC.

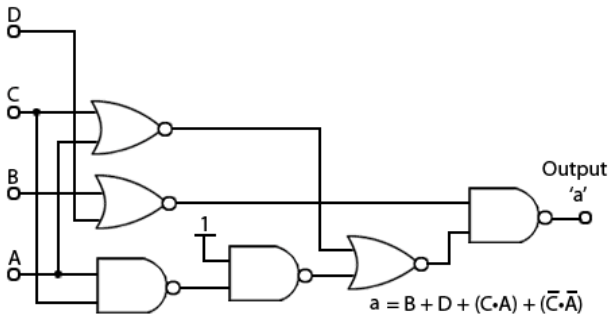


Fig 2.4.10 Minimised Circuit for BCD to 7 Segment decoder 'a' Output.

Minimisation Using Zeros

A further option in simplifying circuits using Karnaugh maps is to produce a map grouping zeros instead of ones. Using the Karnaugh map produced from Table 2.4.3 again, if Zeros and 'Don't Cares' are both included, this produces a map like that illustrated in Fig. 2.4.11.

The Boolean equation using zeros only, would produce:

$$a = (\overline{D} \bullet \overline{C} \bullet \overline{B} \bullet A) + (\overline{D} \bullet C \bullet \overline{B} \bullet \overline{A})$$

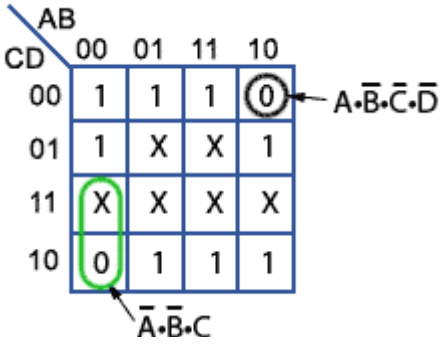


Fig. 2.4.11 Grouping Zeros and 'Don't Cares'

but including the one available X (Don't Care) produces a slightly simplified equation:

$$a = (C \cdot \overline{B} \cdot \overline{A}) + (\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot A)$$

However this produces logic 1 outputs where 'a' was at logic 0 (for inputs 0001 and 0100 in Table 2.4.3). If this output is inverted however, the correct 'a' output according to the truth table for the decoder is produced. A circuit implementing this method is illustrated in Fig. 2.4.12. This circuit uses the same number of ICs as Fig. 2.4.10 but has a 50% longer propagation delay due to the extra gates used.

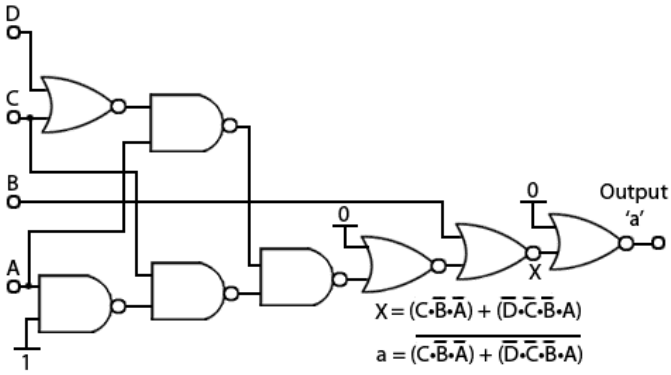


Fig. 2.4.12 Inverting the Output

A circuit with a shorter propagation delay can be made, by using just the zeros in the Karnaugh map as shown in Fig. 2.4.13. This map contains only two 4-term groups but provides the opportunity to use two 4 input AND gates in the circuit illustrated in Fig. 2.4.14

This circuit again uses three ICs, a Hex inverter (with 6 NOT gates), a Dual 4 input AND and a Quad 2 input OR.

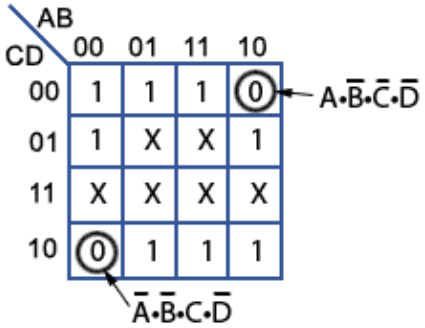


Fig 2.4.13 Two Expression Karnaugh Map Using Zeros

Because only two expressions are used the circuit is not minimised, but implemented as:

$$a = (\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot A) + (\overline{D} \cdot C \cdot \overline{B} \cdot \overline{A})$$

In terms of propagation delay this circuit should be the fastest version of those discussed in this section, but the cost (based on average prices for 74 series ICs) would be approximately 30% more expensive than the cheapest.

Clearly, in designing digital circuits there are choices the designer must make. Cost, speed, physical space and time to design are just some of the design considerations.

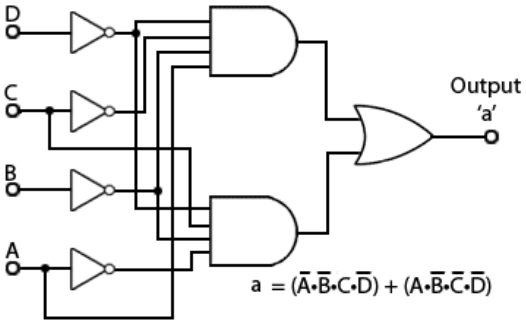


Fig 2.4.14 Karnaugh Derived Faster Circuit Using Zeros.

Relatively few new designs of medium to large systems will be implemented using 74 series ICs today. Now, large and complex Programmable Logic Devices will be used, where the actual logic functions within an integrated circuit, as well as the complex relationships between the functions are set by computer software, but for small scale and one-off designs, the low cost and reduced complexity of the 74 series chips is still valued.

Minimisation using Boolean algebra will largely be confined to simple circuits having few inputs, Karnaugh mapping being preferred as complexity increases. However both of these manual systems of circuit minimisation can be time consuming and error prone. Although Karnaugh mapping can theoretically handle circuits with up to six inputs, much work of this type can be better handled by computer based systems.

2.5 Digital Logic Quiz

Try our quiz, based on the information you can find in Digital Electronics Module 2 − Digital Logic. Check your answers at http://www.learnabout-electronics.org/Digital/dig25.php and see how many you get right. If you get any answers wrong. Just follow the hints to find the right answer and learn about digital logic as you go.

1.

What is the device illustrated in Fig.2.5.1?

a. A two input AND gate.

b. A two input NAND gate.

c. A two input OR gate.

d. A two input XNOR gate.

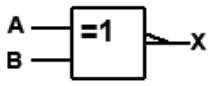


Fig. 2.5.1

2.

Which of the following Boolean equations describes the action of Fig. 2.5.2?

a. $X = (\overline{A \bullet B}) + (B \bullet C)$

b. $X = (A \bullet B) \bullet (B + C)$

c. $X = (\overline{A} \bullet \overline{B}) + (B \bullet C)$

d. $X = (\overline{A \bullet B}) + C$

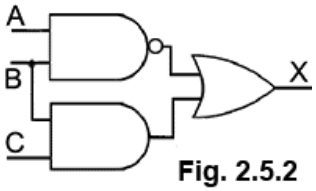


Fig. 2.5.2

3.

Which Boolean law is described by the equation $A \bullet (B+C) = A \bullet B + A \bullet C$?

a. Commutative law.

b. Associative law.

c. Distributive law.

d. Complement law.

4.

Which of the following logic functions is illustrated by Fig. 2.5.3?

a. XOR

b. NOR

c. AND

d. NAND

Fig.2.5.3

5.

Which of the following Boolean equations describes the truth table in Fig. 2.5.4?

a. $X = A + B$

b. $X = A \bullet (A + B)$

c. $X = (A \bullet \overline{B}) + B$

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

Fig.2.5.4

d. $X = A + \overline{B}$

6.

What type of I.C does Fig. 2.5.5 represent?

a. 7400

b. 7402

c. 7404

d. 7408

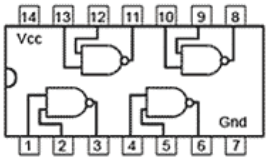


Fig. 2.5.5

7.

Which logic function does the circuit in Fig. 2.5.6 perform?

a. NAND

b. NOR

c. XOR

d. XNOR

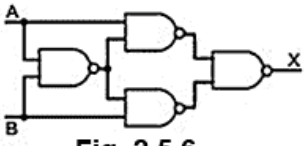


Fig. 2.5.6

8.

What is the Boolean expression that can be obtained for the Karnaugh map cell indicated in Fig. 2.5.7?

a. $\overline{A} \bullet \overline{B} \bullet C \bullet D$

b. $A \bullet \overline{B} \bullet \overline{C} \bullet D$

c. $A \bullet \overline{B} \bullet C \bullet \overline{D}$

d.. $\overline{A} \bullet \overline{B} \bullet \overline{C} \bullet D$

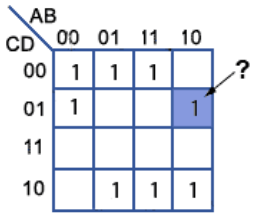


Fig. 2.5.7

9.

Refer to Fig. 2.5.7:

What is the minimum number of cell groups that can be obtained from the Karnaugh map using logic 1 cells only?

a. Two groups.

b. Three groups.

c. Four groups.

d. Five groups.

10.

Which of the segments on a 7 segment LED display need to be illuminated to display the decimal number 4?

a. Segments a, f, b, c

b. Segments c, d, e, f

c. Segments a, d, e, g

d. Segments b, c, f, g