Module 4 – Design of logic systems

Overview

The preceding modules provided you with the building blocks of logic design. Now, you can utilise those building blocks to design logic systems, as shown in the following concept map.

Combinational logic (concept map)

Objectives

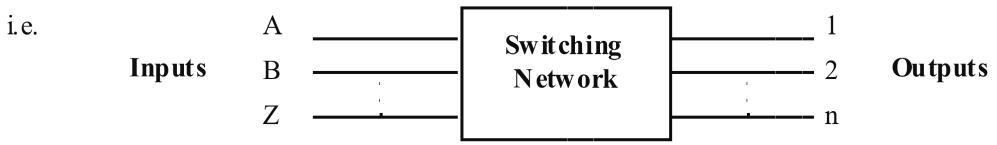
At the completion of this module you will be able to:

- develop general techniques for the design of typical logic systems
- explain the application of logic design techniques for developing computer arithmetic systems, such as the full adder.

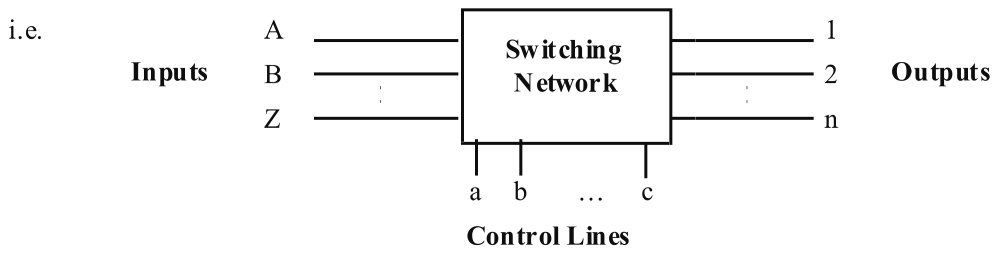
4.1 Switching networks

Logic systems may be small or large depending upon the application. The study of switching is fundamental and necessary to understanding how specific functions are accomplished in a computer or other digital devices.

A designer of a switching network is usually faced with the problem of providing switching between a set of inputs say A, B --- Z and a set of outputs say 1, 2 --- n;



The order and exact nature of the switches vary with each problem. Another set of inputs is usually required which dictate the **'sequence'** of operation of the switches. These are called the **control lines**.



For simple systems, say 3 inputs and 2 outputs, the design of the system is relatively simple and usually can be done intuitively.

However in most applications the number of combinations required is usually large and a systematic approach must be used to enable the designer to develop a workable solution.

4.2 Design techniques

The design of a logic system is easily achieved through an orderly series of steps which typically includes the following:

1. **Statement**

   A precise statement of the problem defining the conditions when the system must provide outputs for certain given inputs.

2. **Truth table**

   Formation of a detailed truth table listing all possible input conditions and the resulting output states is required.

3. **Boolean expression**

   Development of the necessary Boolean functions from the truth table.

4. **Simplification**

   Simplification of the Boolean functions to their simplest apparent form.

5. **Implementation**

   Implementation of the final Boolean functions with relays, switches or logic gates.

   The presentation of the detailed truth table can vary slightly depending upon the individual designer. An orderly listing of all the possible inputs and outputs with a true or false indication is the normal basic requirement. These basic steps are best illustrated by a design example.

Example

1. **Statement**

   The statement should clearly describe the logic of the system. For example:

   "This logic system is required to indicate when the 3-bit binary input is equal to decimal 4, 5 and 7. The system consists of three switches, A, B and C that provide the 3-bit binary input and one output (F). Output (F) will be true (logic 1) when the above conditions are met."

   Additional information may include:

   - the function of the circuit
   - its use in a logic system
   - unique features of the circuit.

2. **Truth table**

Count (Decimal)	Input (Binary) A	B	C	Output F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

   The accuracy of the truth table is critical as errors will be carried forward into the final circuit design.

3. **Boolean expression**

   Using minterms an expression can be written for the three cases where an output (F = 1) is required.

   From above,

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

   i.e. output present for sequence 4 or sequence 5 or sequence 7.

   Note in passing that this expression is canonical in form since **all** variables appear in **all** terms in **either** true or complement form.

   Note also that, in this case, there is one output (F), hence one boolean expression. If there were 4 outputs, there would be 4 boolean expressions. Take care not to join the bars on adjacent variables as the logic meaning will be changed. For example:

   $$\overline{B}\ \overline{C} \quad \overline{BC}$$

   $$\overline{B} + \overline{C} = \overline{BC}$$

4. **Simplification**

   The canonical form can usually be reduced to a simpler form which requires fewer logic gates for implementation.

   From step 3 $F = \overline{A}BC + AB\overline{C} + A\overline{B}\overline{C}$

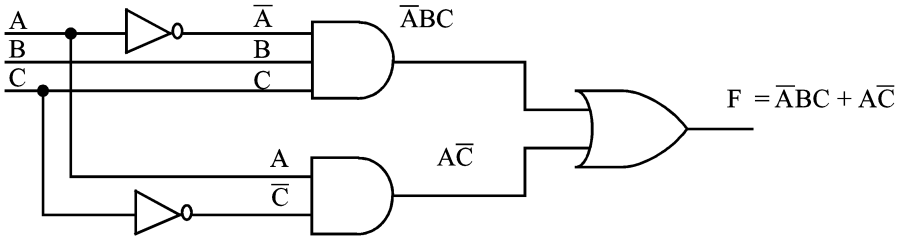
   $F = \overline{A}BC + A\overline{C}\ (B + \overline{B})$

   $\therefore F = \overline{A}BC + A\overline{C}$ as $(B + \overline{B}) = 1$

   Karnaugh maps may also be used to simplify the Boolean expressions.

5. **Implementation**

   This expression may be then implemented using AND and OR gates in this case.



6. **NAND equivalent**

   In section 2.2.4 it was shown that NAND gates could be used to replace the basic operations of OR, AND and INVERT. To convert a Boolean expression to its NAND equivalent we can use DeMorgans Theorems, as follows:

   The final Boolean expression from this example was

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

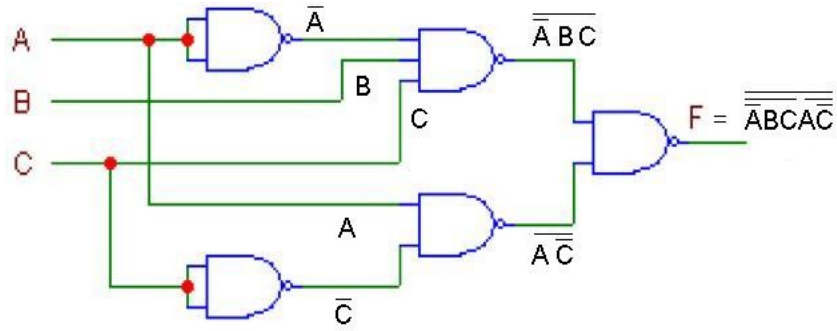
   To convert to NAND gates it is necessary to double-complement this expression. Double-complementing does not change the expressions as the two bars would normally cancel. That is:

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

   Now apply DeMorgan's Theorem by breaking the bar second from the top. This removes the OR gate and produces a 2 input NAND gate.

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

   This new expression can now be implemented directly using NAND gates.



4.3 Adding binary numbers

4.3.1 Binary addition rules

The basic rules for binary addition are:

0	+	0	=	0
1	+	0	=	1
1	+	1	=	
				0 plus a carry of 1 to the next column.

Let us use these rules to illustrate binary addition. Consider the arithmetic sum three plus two executed in decimal and binary.

Decimal	**Binary**	
3	011	
+ 2	+ 010	
	1	← 'carry'
5	101	

The binary addition is achieved by adding each column in turn. Start at the right hand column, and using the rules for binary addition we have: *zero plus one equals one*, in the second column we have *one plus one equals zero plus a carry of one to the next column.* In the left most column we have *one plus zero plus zero equals one.*

4.3.2 The half adder

A computer must be capable of adding binary numbers so a logic circuit which can add two binary numbers together is required. This is called a half adder for reasons we shall see later.

Consider the practical problem of adding two binary digits A and B together where A = 1 and B = 1.

We have

$$\begin{array}{rcl} A & = & 01 \\ B & = & 01 \\ & & 1 \leftarrow \text{carry} \\ & & \overline{10} \leftarrow \text{sum} \\ & & = \end{array}$$

The system must be capable of giving us a Sum output (So) and a Carry output (Co).

From the rules of binary addition we can draw a truth table for all possible conditions of A and B.

A	B	Sum (So)	Carry (Co)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

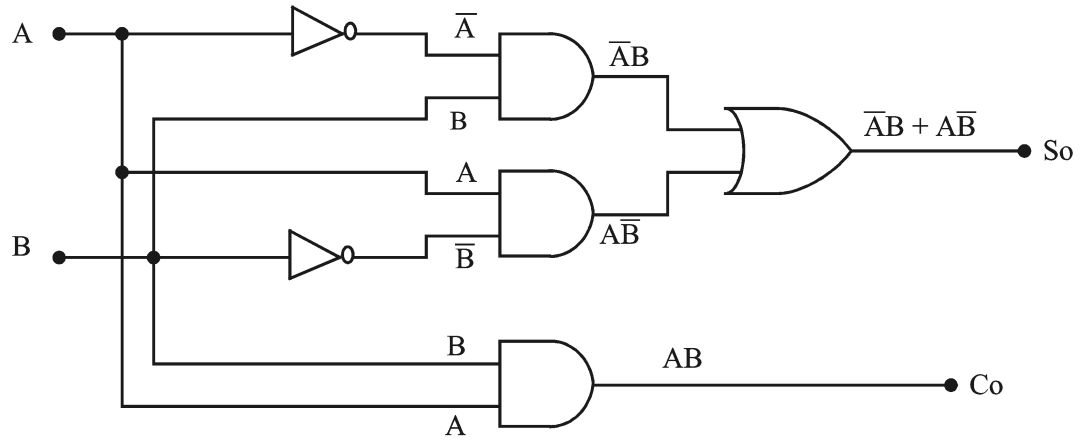
From this truth table we may write two Boolean expressions:

$$So = \overline{A}B + A\overline{B}$$

and

$$Co = AB$$

These expressions cannot be simplified and may be therefore implemented in logic thus:



This configuration is called a half adder because, although it accepts two digits adds them and produces a sum and carry, no provision is made for any extra input from a previous addition should there be one.

For example suppose we had:

$$\begin{array}{rcl} A & = & 01 \\ B & = & 11 \\ & & 1 \\ \hline & & 100 \\ & & \equiv \end{array}$$

In adding the second column we are really adding three digits together. Not only do we have the 'one' from B and the 'zero' from A but we also have a 'one' carry from the first column addition, called the carry input (Ci). A logic system which accommodates this situation is called a full adder.

4.3.3 The full adder

We can now generate a truth table for all possible conditions.

Inputs			**Outputs**	
Ci	A	B	S_0	C_0
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

From this table we may write the two Boolean expressions.

$$So = \overline{C}i\overline{A}B + \overline{C}iA\overline{B} + Ci\overline{A}\overline{B} + CiAB$$

$$Co = \overline{C}iAB + Ci\overline{A}B + CiA\overline{B} + CiAB$$

Simplifying firstly for So, we have:

$$So = \overline{C}i(\overline{A}B + A\overline{B}) + Ci(\overline{A}\overline{B} + AB) \qquad \text{①}$$

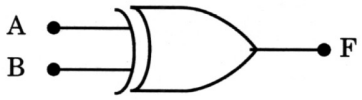
Using our knowledge to date this equation cannot be simplified further and could be implemented at this stage in logic hardware.

However if we examine the first part of equation in brackets we have the expression $\left(\overline{A}B + A\overline{B}\right)$. This expression occurs quite often in logic design and consequently has an identifying name. It is called the **exclusive-or function** and has a symbol ⊕. This distinguishes it from our earlier symbol + which is known as the **inclusive-or function**.

Thus: $\boxed{\overline{A}B + A\overline{B} = A \quad B}$

This is an important relationship and a commercial logic gate is available to perform this function. It is sometimes called the **not-equivalent** or **modulo-2** function.

Its symbol is:



$\boxed{F = A \oplus B}$

Exclusive-Or Gate

Let us now consider the first part of equation further. In fact we will now determine its inverse, that is:

$$\overline{\overline{A}B + A\overline{B}}$$

Using De Morgans theorem we have:

$$\begin{aligned}\overline{\overline{A}B + A\overline{B}} &= \left(\overline{\overline{A}B}\right) . \left(\overline{A\overline{B}}\right) \\ &= \left(A + \overline{B}\right) . \left(\overline{A} + B\right) \\ &= \overset{0}{\cancel{A\overline{A}}} + AB + \overline{A}\overline{B} + \overset{0}{\cancel{B\overline{B}}}\end{aligned}$$

Thus $\overline{\overline{A}B + A\overline{B}} = AB + \overline{A}\overline{B}$

Note this result ($AB + \overline{A}\overline{B}$) equals the second part of equation ①.

We can now re-write equation ① in terms of exclusive-or functions.

Equation ①, So $= \overline{C}i(\overline{A}B + A\overline{B}) + Ci(\overline{A}\overline{B} + AB)$

From above $= \overline{C}i(A \quad B) + (Ci(\overline{A \quad B})$

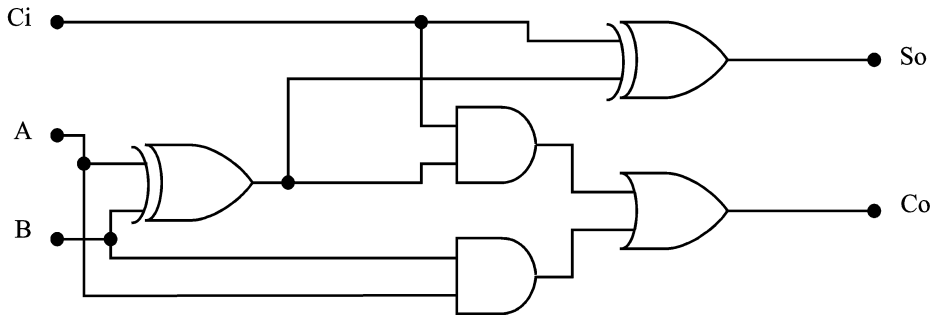
Look further at this expression and you will see it can also be again rewritten in exclusive-or terms,

$$\underline{So = Ci \oplus A \oplus B}$$

To Carry output (Co) may now be simplified. From before:

$$Co = \overline{Ci}AB + Ci\overline{A}B + CiA\overline{B} + CiAB$$
$$= Ci(\overline{A}B + A\overline{B}) + AB(Ci + \overline{Ci})$$
$$\therefore \quad Co = Ci(A \quad B) + AB$$

This two simplified expressions may now be implemented in logic hardware to form a full adder.



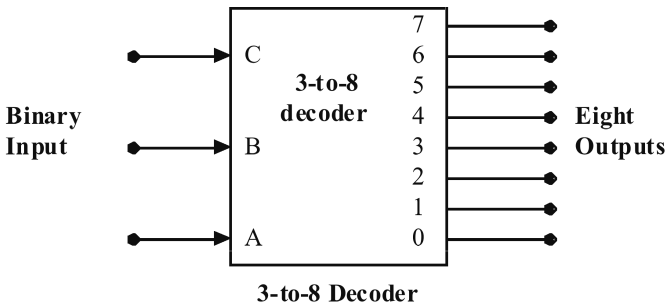
Full-Adder

4.4 Decoders

In digital computers binary numbers are used to present various types of information such as instructions, addresses, data and control signals. A number containing n bits of binary information can represent 2^n different combinations.

A logic circuit which can take the n-bit binary number as an input and then generate an appropriate output signal to indicate which of the 2^n combinations is present is called a decoder.

A decoder may have fewer than 2^n outputs if desired. A decoder with n inputs and m outputs is called an n-to-m line decoder. A typical decoder is shown in the diagram below. This is a 3-to-8 decoder which means its three inputs are decoded to 8 outputs.

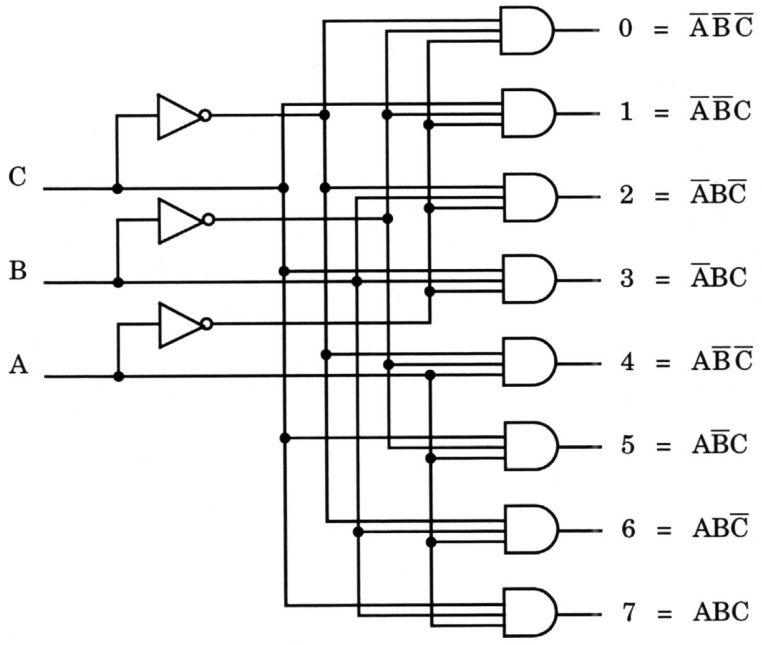


3-to-8 Decoder

The truth table for this decoder is shown below.

Inputs			Outputs							
A	B	C	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

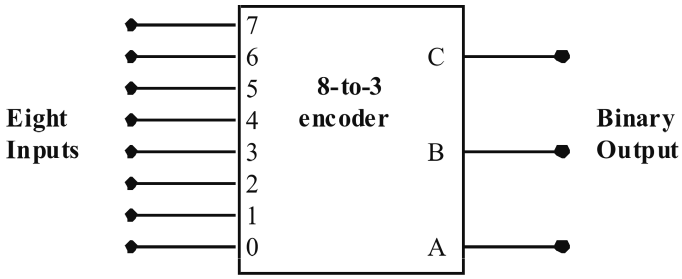
The logic diagram for this decoder is constructed from AND gates and is shown below. It is sometimes called a Binary-to-Octal decoder as it takes a binary code and produces eight outputs.



3-to-8 Decoder Logic Diagram

4.5 Encoders

An encoder performs the reverse characteristic to a decoder. An encoder produces a binary output signal corresponding to which of its inputs has been activated. It has 2^n inputs maximum and n outputs. Let us consider an 8-to-3 encoder as shown in the diagram.



8-to-3 Encoder

The truth table is identical to that for the decoder except that the input and output roles are reversed.

This encoder has some limitations. Consider the case if all the inputs are zero. This means the output will also be zero. However notice from the truth table that the output is meant to be zero also if the input 0 is a logic 1.

Further, consider the case if two of the inputs were to be activated simultaneously. This would cause the wrong output to occur. These problems are overcome by using a priority encoder.

4.5.1 Priority encoder

A priority encoder is a logic circuit such that if two inputs are activated simultaneously, the input having the highest priority only will generate the output. That is the highest numbered input takes precedence and no matter how many inputs are activated, the binary number for the highest one appears at the output.

The truth table for a 4 input priority encoder is given below:

Inputs				**Outputs**		
A	B	C	D	F_0	F_1	Valid
0	0	0	0	–	–	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

X = Don’t cares
– = undefined

Input A has the highest priority which means no matter what the other inputs are, the output binary number $F_0F_1 = 11$ will be valid.

B, C and D then have priority in that order.

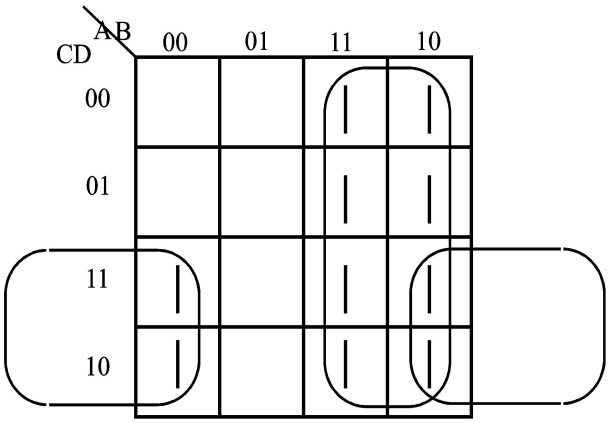
The output for D i.e. (0 0) is valid provided all the other higher priority inputs are zero, etc.

Logic design is accomplished using Karnaugh maps for F_1 and F_0.

For F_1 we have, using all the don't care variations:

$$F_1 = \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$$

The map for F_1 is:

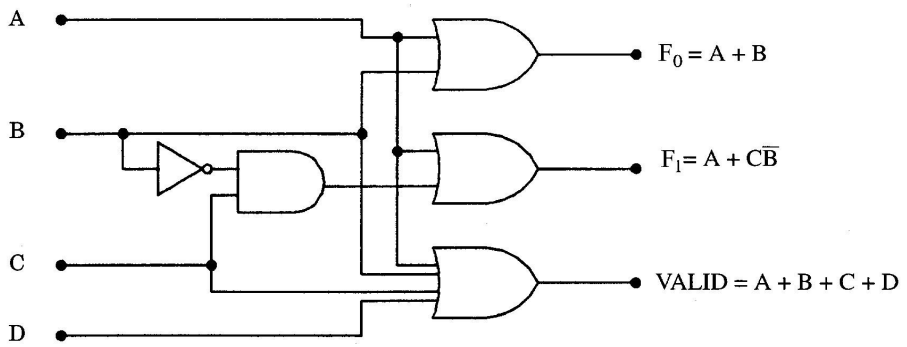


$$F_1 = A + C\bar{B}$$

You should generate the F_0 map and obtain the result:

$$F_0 = A + B$$

This may be implemented as:



Activity 4.1 (Typical exam questions)

1. Design a logic circuit which compares two 4 bit numbers A and B to determine if they are equal. The circuit output (F) is to be a logic 1 for equality.

   (Hint: Consider the characteristics of EXCLUSIVE OR gates.)

2. Design a 4 bit priority encoder with input D having the highest priority based on the truth table given in the section 4.5.1.

3. Refer to the ELE1301 course page on ‘Study Desk’ and complete the following experiments:

   a. Home experiment 4-1 – Full-adder

   b. Home experiment 4-2 – Priority encoder.

Solution to activity 4.1

1. Consider a circuit that compares two 2-bit numbers A and B to determine if they are equal. If you draw a truth table you will find that a logic 1 will occur when:

   $F=\overline{A}_1\,\overline{A}_0\,\overline{B}_1\,\overline{B}_0+\overline{A}_1\,A_0\,\overline{B}_1B_0+A_1\overline{A}_0B_1\overline{B}_0+A_1A_0B_1B_0$, then

   $F=\overline{A}_1\,\overline{B}_1(\overline{A}_0\,\overline{B}_0+A_0B_0)+A_1B_1(\overline{A}_0\,\overline{B}_0+A_0B_0)$

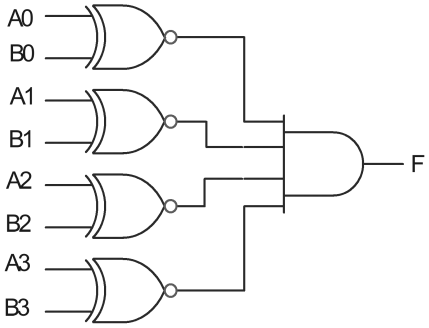
   $F=\overline{A}_1\,\overline{B}_1(\overline{A_0\quad B_0})+A_1B_1(\overline{A_0\quad B_0})$

   $F=(\overline{A}_1\,\overline{B}_1+A_1B_1)(\overline{A_0\quad B_0})$

   $F=(\overline{A_1\quad B_1})(\overline{A_0\quad B_0})$

   So for two 4-bit numbers the above can be extended to:

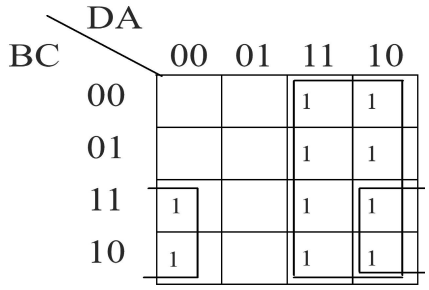
   $F=(\overline{A_3\quad B_3})(\overline{A_2\quad B_2})(\overline{A_1\quad B_1})(\overline{A_0\quad B_0})$



2.

Inputs				Outputs		
D	**A**	**B**	**C**	**F_0**	**F_1**	**Valid**
0	0	0	0	-	-	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

$F_1=\bar{D}\bar{A}BC+\bar{D}\bar{A}B\bar{C}+D\bar{A}\bar{B}\bar{C}+D\bar{A}\bar{B}C+D\bar{A}B\bar{C}+D\bar{A}BC+DA\bar{B}C+DA\bar{B}C+DAB\bar{C}+DABC$



$F_1=D+\bar{A}B$

etc

Self assessment

1. Show how the Exclusive-OR function can be implemented using NAND gates only. Assume two inputs (A) and (B). Neatly set out your work to provide a:
   i. Truth table
   ii. Boolean expression for the Exclusive OR output (F)
   iii. Boolean expression for the Exclusive OR output (F) – NAND equivalent using DeMorgan's Theorems.
   iv. Logic circuit using NAND gates.

2. Explain the purpose and operation of a half-adder. Neatly set out your work to provide:
   i. Truth table
   ii. Boolean expressions for So and Co

3. Design a 2 to 4 decoder and implement the design using logic gates. Neatly set out your work to provide:
   i. Truth table
   ii. Boolean expressions
   iii. Logic circuit using AND and inverter gates.

4. The truth table for a 4-input priority encoder is given below.

Inputs				Outputs		
A	B	C	D	F0	F1	Valid
0	0	0	0	-	-	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

   (X) = Don't Cares; (–) = undefined

   Design a logic system that will provide the output F1 only. Neatly set out your work to provide:
   i. Boolean expression for F1 (include all don't care conditions)
   ii. Karnaugh map simplification
   iii. Simplified boolean expression for F1
   iv. Logic circuit using AND, OR and inverter gates.

5. Design a 'Binary' to 'Gray code' converter using AND, OR and inverter