2.4 BINARY ADDER & SUBTRACTOR

Addition and Subtraction are two basic Arithmetic Operations that must be performed by any Digital Computer. If both these operations can be properly implemented, then Multiplication and Division tasks become easy (as multiplication is repeated addition and division is repeated subtraction).

Consider the operation of adding two binary numbers, which is one of the fundamental tasks performed by a digital computer. The four basic addition operations two single bit binary numbers are:

- 0 + 0 = 0
- 1 + 0 = 1
- 0 + 1 = 1
- 1 + 1 = (Carry)1 0

	1	1	0	0
	+1	+0	+1	+0
(carry)1	0	1	1	0

In the first three operations, each binary addition gives sum as one bit , i.e., either 0 or 1. But for the fourth addition operation (where the inputs are 1 and 1), the result consists of two binary digits. Here, the lower significant bit is called as the 'Sum Bit', while the higher significant bit is called as the 'Carry Bit'.

For single bit additions, there may not be an issue. The problem may arise when we try to add binary numbers with more than one bit.

The logic circuits which are designed to perform the addition of two binary numbers are called as Binary Adder Circuits. Depending on how they handle the output of the '1+1' addition, they are divided into:

- Half Adder
- Full Adder

Let us take a look at the binary addition performed by various adder circuits.

2.4.1 Half Adder

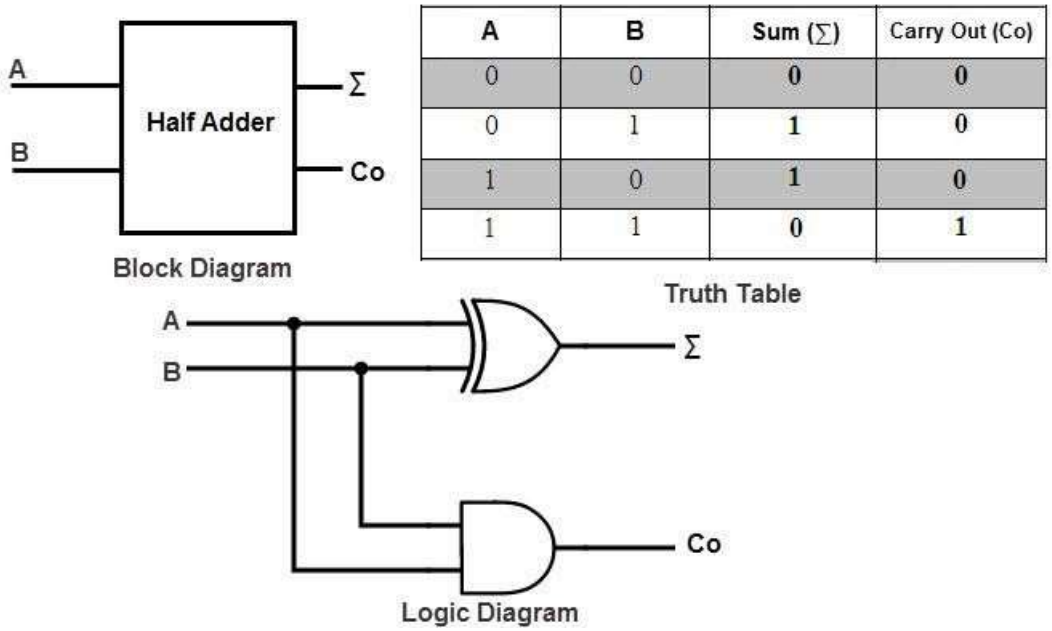
A logic circuit used for adding two 1-bit numbers or simply two bits is called as a Half Adder circuit. This circuit has two inputs and two outputs. The inputs are the two 1-bit binary numbers (known as Augend and Addend) and the outputs are Sum and Carry.

The following image shows the block diagram of Half Adder.

The truth table of the Half Adder is shown in the following table.

INPUT		OUTPUT
A	B	Sum
0	0	0
0	1	1
1	0	1
1	1	0

If we observe the 'Sum' values in the above truth table, it resembles an Ex-OR Gate. Similarly, the values for 'Carry' in the above truth table resembles an AND Gate.



A	B	Sum (∑)	Carry Out (Co)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table

Figure 2.4.1 Half Adder

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

So, to properly implement a Half Adder, you need two Logic Gates: an XOR gate for 'Sum' Output and an AND gate for 'Carry' output.

In the above half adder circuit, inputs are labeled as A and B. The 'Sum' output is labeled as summation symbol (∑) and the Carry output is labeled with C_O.

Half adder is mainly used for addition of augend and addend of first order binary numbers i.e., 1-bit binary numbers. We cannot add binary numbers with more than one bit as the Half Adder cannot include the 'Carry' information from the previous sum.

Due to this limitation, Half Adder is practically not used in many applications, especially in multi-digit addition. In such applications, carry of the previous digit addition must be added along with two bits; hence it is a three bit addition.

2.4.2 Full Adder

A Full Adder is a combinational logic circuit which performs addition on three bits and produces two outputs: a Sum and a Carry. As we have seen that the Half Adder cannot respond to three inputs and hence the full adder is used to add three digits at a time.

It consists of three inputs, of which two are input variables representing the two significant bits to be added, whereas the third input terminal is the carry from the previous addition. The two outputs are a Sum and Carry outputs.

The following image shows a block diagram of a Full Adder where the inputs are labelled as A, B and C_{IN}, while the outputs are labelled as ∑ and C_{OUT}.

C_{in}	B	A	∑	C_{out}
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**

Truth Table

Coming to the truth table, the following table shows the truth table of a Full Adder.

INPUT	OUTPUT

A	B	C_{IN}	Sum	C_{OUT}
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 the above truth table, we can obtain the Boolean Expressions for both the Sum and Carry Outputs. Using those expressions, we can build the logic circuits for Full Adder. But by simplifying the equations further, we can derive at a point that a Full Adder can be easily implemented using two Half Adders and an OR Gate.

The following image shows a Full Adder Circuit implemented using two Half Adders and an OR Gate. Here, A and B are the main input bits, C_{IN} is the carry input, $\sum$ and C_{OUT} are the Sum and Carry Outputs respectively.

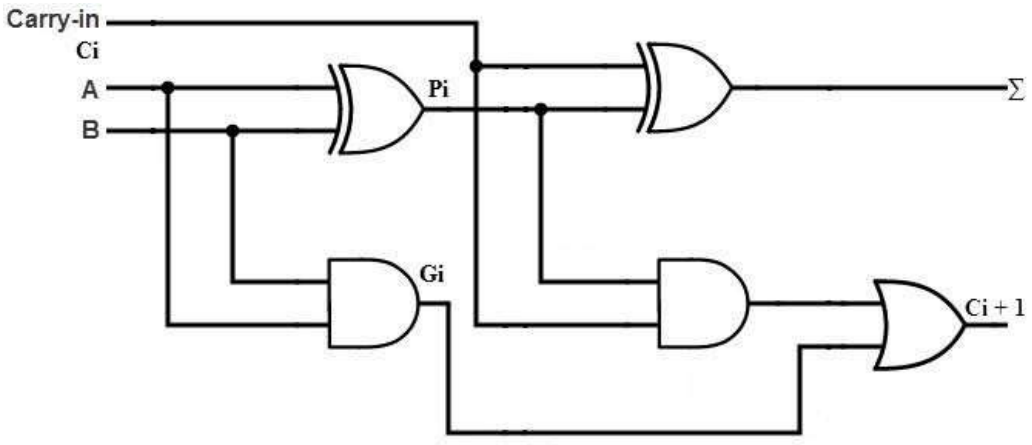


Figure 2.4.2 Full Adder

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

2.4.3 Parallel Binary Adders

As we discussed, a single Full Adder performs the addition of two one bit numbers and also the carry input. For performing the addition of binary numbers with more than one bit, more than one full adder is required and the number of Full Adders depends on the number bits. Thus, a Parallel Adder, is a combination of Multiple Full Adders and is used for adding all bits of the two numbers simultaneously.

By connecting 'n' number of full adders in parallel, an n-bit Parallel Adder can be constructed. From the below figure, it is to be noted that there is no carry at the least significant position, hence we can use either a half adder or make the carry input of full adder as zero at this position.

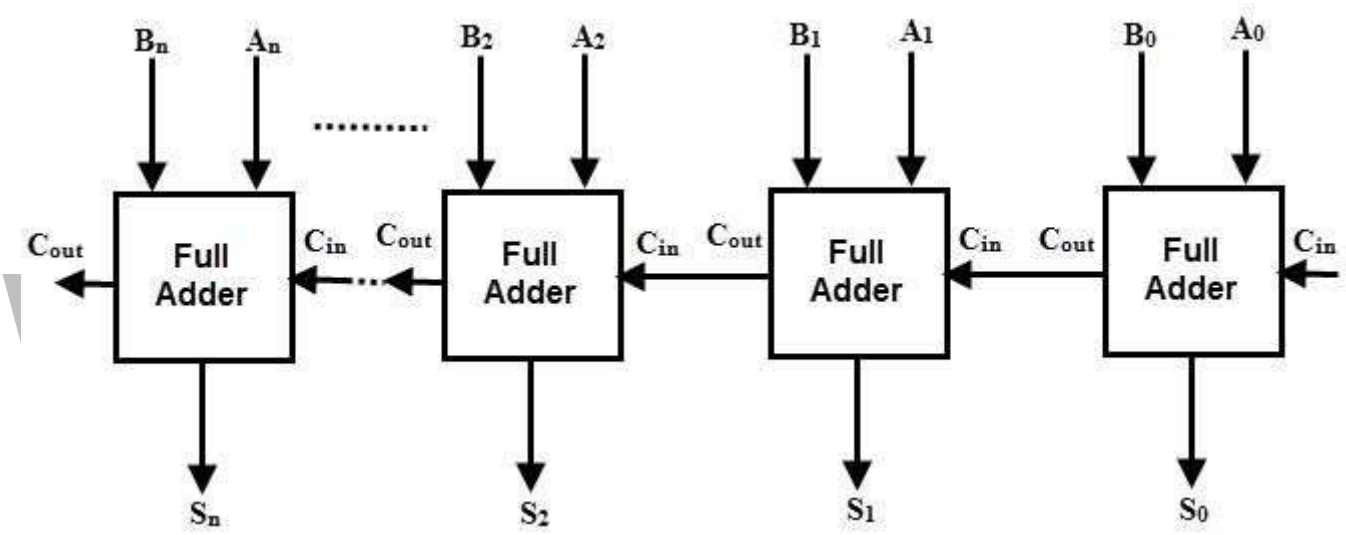


Figure 2.4.3 Parallel Binary Adders

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

The following figure shows a Parallel 4-bit Binary Adder, which has three full adders and one half adder. The two binary numbers to be added are '$A_3\ A_2\ A_1\ A_0$' and '$B_3\ B_2\ B_1\ B_0$', which are applied to the corresponding inputs of the Full Adders. This parallel adder produces their result as '$C_4\ S_3\ S_2\ S_1\ S_0$', where C_4 is the final carry.

In the 4 bit adder, first block is a half-adder that has two inputs as $A_0\ B_0$ and produces a sum S_0 and a carry bit C_1. The first block can also be a full adder and if so, then the input Carry C_0 must be 0.

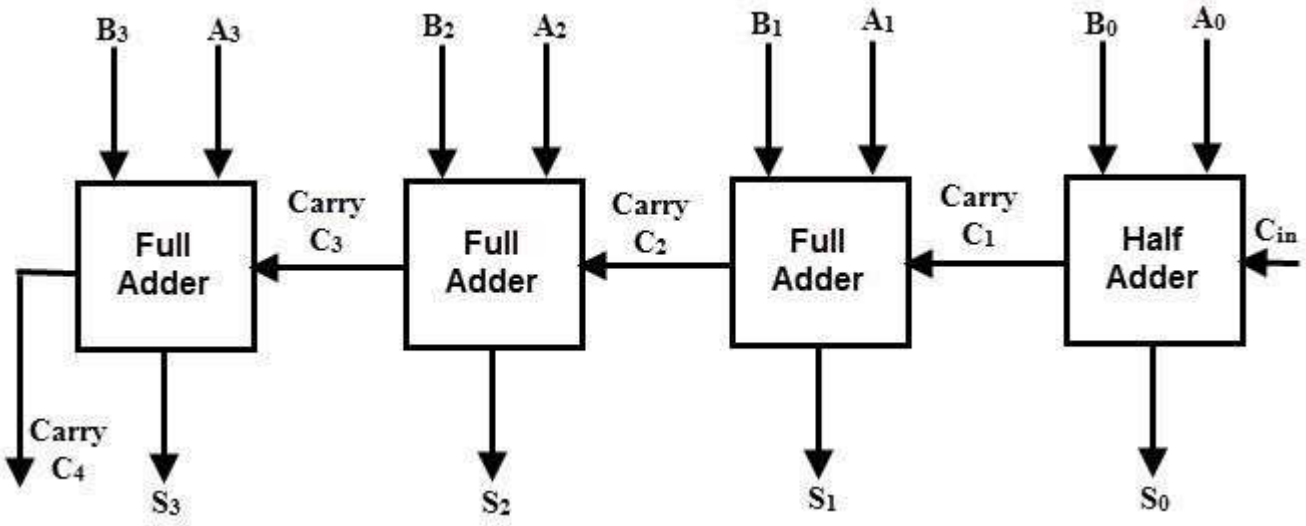


Figure 2.4.4 Parallel 4- Bit Binary Adders

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

Next three blocks should be full adders, as there are three inputs applied to them (two main binary bits and a Carry bit from the previous stage).

Hence, the second block full adder produces a sum S_1 and a carry C_2. This will be followed by other two full adders and thus the final result is C_4 S_3 S_2 S_1 S_0.

Commonly, the Full Adders are designed in dual in-line package integrated circuits. 74LS283 is a popular 4-bit full adder IC. Arithmetic and Logic Unit or ALU of an unit computer consist of these parallel adders to perform the addition of binary numbers.

2.4.4 Binary Subtraction Circuits

Another basic arithmetic operation to be performed by Digital Computers is the Subtraction. Subtraction is a mathematical operation in which one integer number is deducted from another to obtain the equivalent quantity. The number from which other number is to be deducted is called as 'Minuend' and the number subtracted from the minuend is called 'Subtrahend'.

Similar to the binary addition, binary subtraction is also has four possible basic operations. They are:

- 0 – 0 = 0
- 0 – 1 = (Borrow)1 1
- 1 – 0 = 1

- 1 – 1 = 0

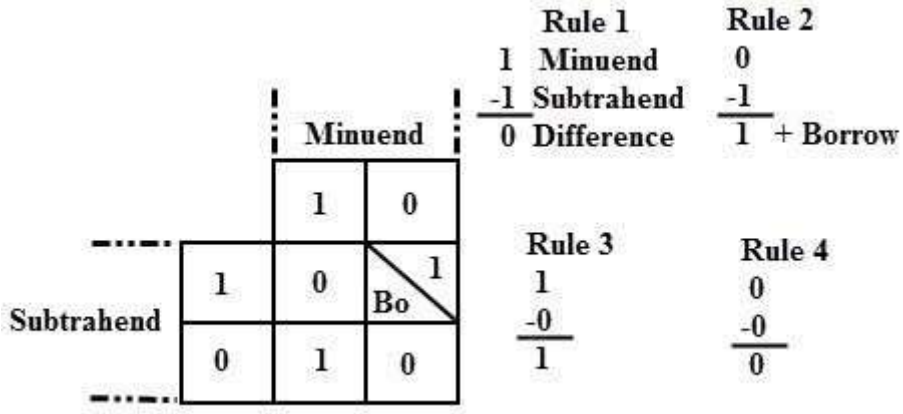


Figure 2.4.5 Subtraction

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

The above figure shows the four possible rules or elementary operations of the binary subtractions. In all the operations, each subtrahend bit is deducted from the minuend bit. But in the second rule, minuend bit is smaller than the subtrahend bit, hence 1 is borrowed to perform the subtraction. Similar to the adder circuits, basic subtraction circuits are also of two types:

- Half Subtractor
- Full Subtractor

Half Subtractors

A Half Subtractor is a multiple output Combinational Logic Circuit that does the subtraction of two 1-bit binary numbers. It has two inputs and two outputs. The two inputs correspond to the two 1-bit binary numbers and the two outputs corresponds to the Difference bit and Borrow bit (in contrast to Sum and Carry in Half Adder).

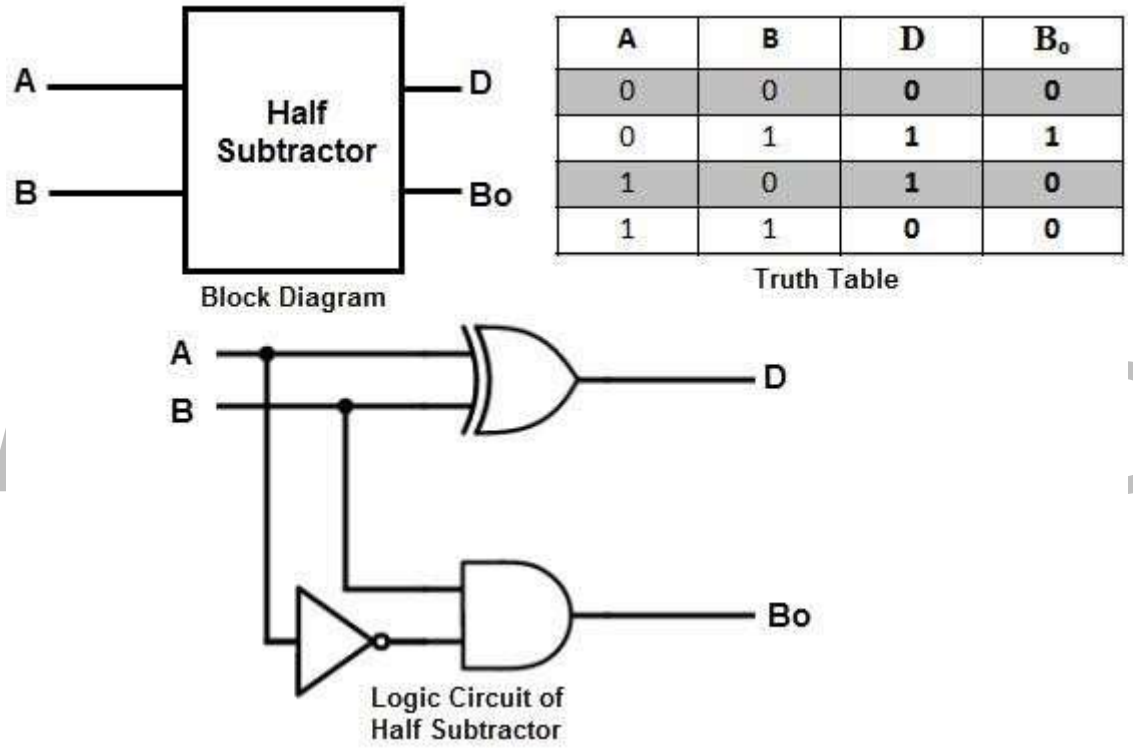
From the above truth table, we can say that the 'Difference' output of the Half Subtractor is similar to an XOR output (which is also same as the Sum output of the Half Adder).

Thus, the Half Subtraction is also performed by the Ex-OR gate with an AND gate with one inverted input and one normal input, requiring to perform the Borrow operation.

Following table shows the truth table of a Half Subtractor.

INPUT		OUTPUT	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The following image shows the logic circuit of a Half Adder.



A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth Table

Figure 2.4.5 Half Subtractors

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

This circuit is similar to that of the Half Adder with only difference being the minuend input i.e., A is complemented before applied at the AND gate to implement the borrow output.

In case of multi-digit subtraction, subtraction between the two digits must be performed along with borrow of the previous digit subtraction, and hence a subtractor needs to have three inputs, which is not possible with a Half Subtractor. Therefore, a half subtractor has limited set of applications and strictly speaking, it is not used in practice.

Full Subtractor

A Full Subtractor is a combinational logic circuit which performs a subtraction between the two 1-bit binary numbers and it also considers the borrow of the previous bit i.e., whether 1 has been borrowed by the previous minuend bit.

So, a Full Subtractor has three inputs, in which two inputs corresponding to the two bits to be subtracted (minuend A and subtrahend B), and a borrow bit, usually represented as B_{IN}, corresponding to the borrow operation. There are two outputs, one corresponds to the difference D output and the other Borrow output B_O.

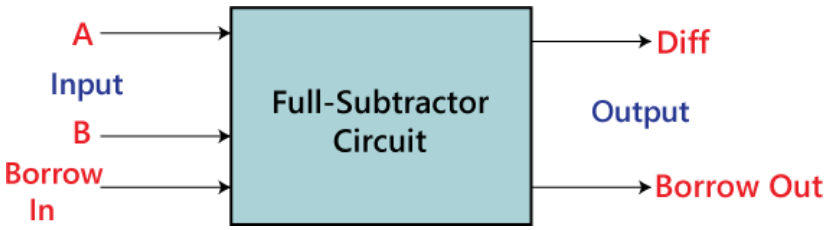
The following table shows the truth table of a Full Subtractor.

INPUT			OUTPUT	
A	B	B_{IN}	D	B_{OUT}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

By deriving the Boolean expression for the full subtractor from above truth table, we get the expression that tells that a full subtractor can be implemented with half subtractors with OR gate as shown in figure below.

By comparing the adder and subtractor circuits and truth tables, we can observe that the output D in the full subtractor is exactly same as the output S of the full adder. And the only difference is that input variable A is complemented in the full subtractor.

The following image shows the block diagram of a full subtractor.



A	B	B_{in}	D	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth Table

Figure 2.4.5 Full Subtractors

[Source: https://www.javatpoint.com/full-subtractor-in-digital-electronics]

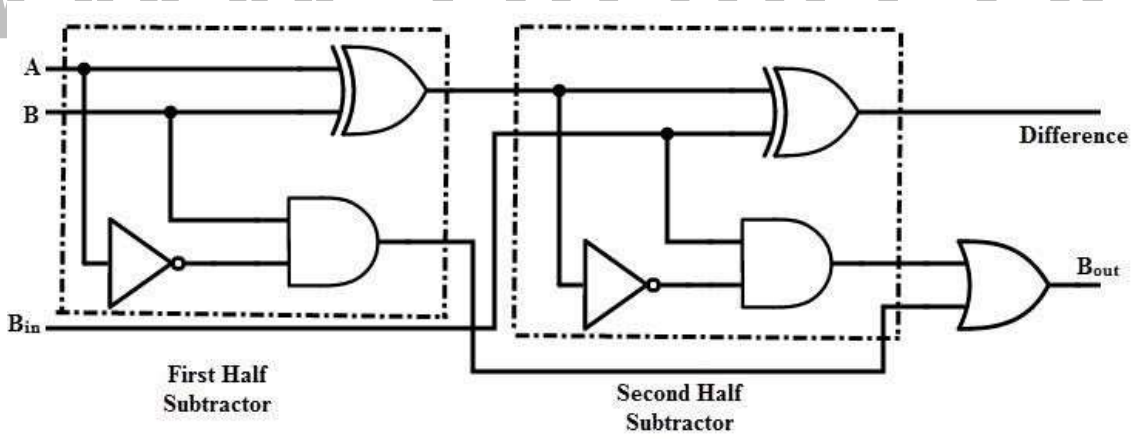


Figure 2.4.5 Full Subtractors by Two Subtractors

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

Therefore, it is possible to convert the full adder circuit into full subtractor by simply complementing the input A before it is applied to the gates to produce the final borrow bit output Bo.

Parallel Binary Subtractors

To perform the subtraction of binary numbers with more than one bit, we have to use the Parallel Subtractors. This parallel subtractor can be designed in several ways, including combination of half and full subtractors, all full subtractors, all full adders with subtrahend complement input, etc.

The below figure shows a 4 bit Parallel Binary Subtractor formed by connecting one half subtractor and three full subtractors.

In this subtractor, 4 bit minuend 'A_3 A_2 A_1 A_0' is subtracted by 4 bit subtrahend 'B_3 B_2 B_1 B_0' and the result is the difference output 'D_3 D_2 D_1 D_0' . The borrow output of each subtractor is connected as the borrow input to the next subtractor.

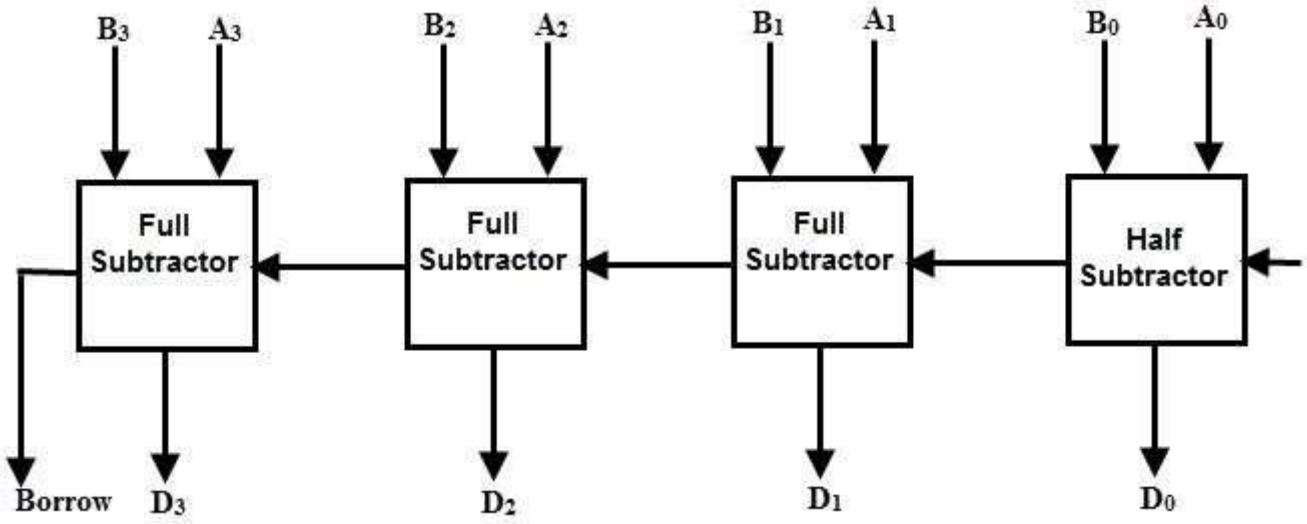


Figure 2.4.6 Parallel Binary Subtractors

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

It is also possible to design a 4 bit parallel subtractor using 4 full adders as shown in the below figure. This circuit performs the subtraction operation by considering the principle that the addition of minuend and the complement of the subtrahend is equivalent to the subtraction process.

We know that the subtraction of A by B is obtained by taking 2's complement of B and adding it to A. The 2's complement of B is obtained by taking 1's complement and adding 1 to the least significant pair of bits.

Hence, in this circuit 1's complement of B is obtained with the inverters (NOT gate) and a 1 can be added to the sum through the input carry.

Parallel Adder / Subtractor

The operations of both addition and subtraction can be performed by a one common binary adder. Such binary circuit can be designed by adding an Ex-OR gate with each full adder as shown in below figure. The figure below shows the 4 bit parallel binary adder/subtractor which has two 4 bit inputs as 'A_3 A_2 A_1 A_0' and 'B_3 B_2 B_1 B_0' .

The mode input control line M is connected with carry input of the least significant bit of the full adder. This control line decides the type of operation, whether addition or subtraction.

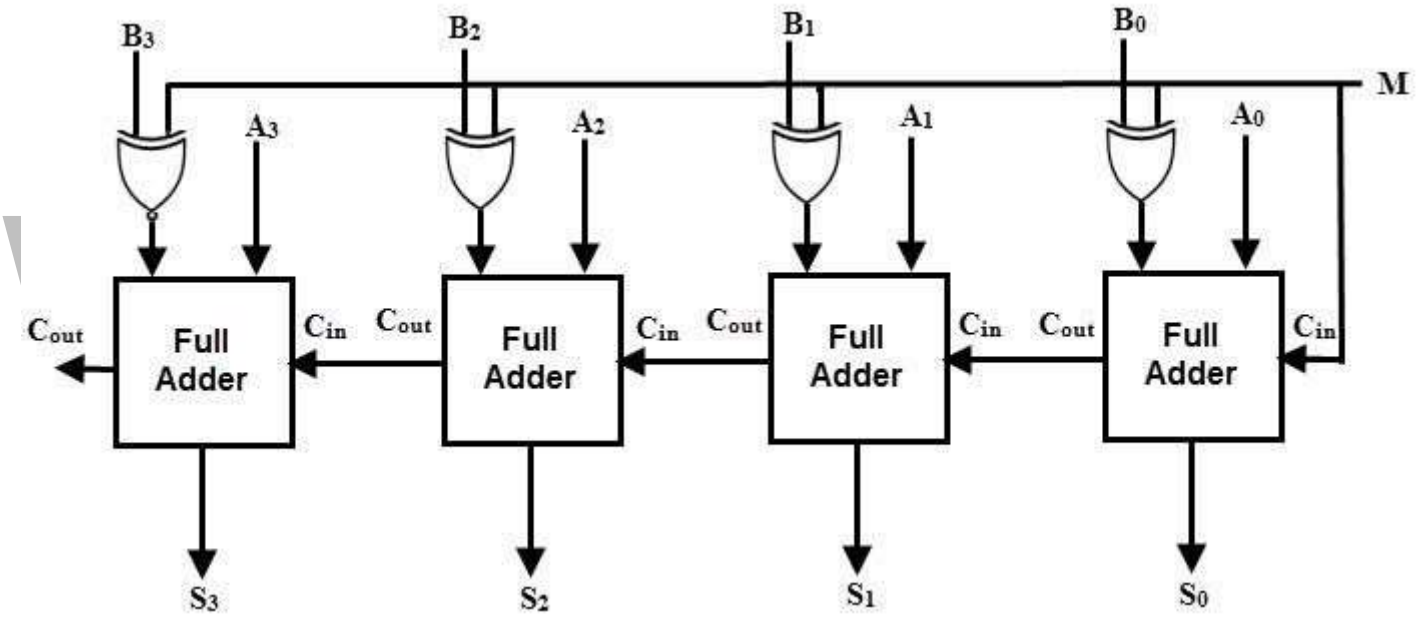


Figure 2.4.7 Parallel Adder / Subtractor

[Source: https://www.electronicshub.org/binary-adder-and-subtractor/]

When M= 1, the circuit is a subtractor and when M=0, the circuit becomes adder. The Ex-OR gate consists of two inputs to which one is connected to the B and other to input M. When M = 0, B Ex-OR of 0 produce B. Then, full adders add the B with A with carry input zero and hence an addition operation is performed.

When M = 1, B Ex-OR of 0 produce B complement and also carry input is 1. Hence the complemented B inputs are added to A and 1 is added through the input carry, nothing but a 2's complement operation. Therefore, the subtraction operation is performed.

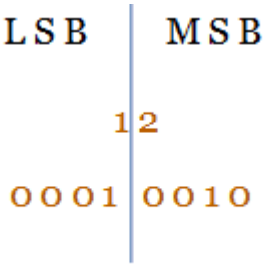
2.6 CODE CONVERTER

The Code converter is used to convert one type of binary code to another. There are different types of binary codes like BCD code, gray code, excess-3 code, etc. Different codes are used for different types of digital applications.

2.6.1 Binary to BCD code converter

For BCD code, 0 to 9 numbers represent the equivalent binary numbers. For the numbers above 10, LSB of a decimal number is represented by its equivalent binary number and MSB of a decimal number is also represented by their equivalent binary numbers.

For example, the BCD code of 12 is represented as



The BCD code for 12 is 10010

The following truth table shows the conversion between the binary code input and the BCD code output. As you see from the table, the 4-bit binary number is converted into 5-bit BCD code. Decimal code is added in the table to understand the equivalence of Binary and BCD code.

The converter has 5 outputs D0, D1, D2, D3 and D4. From the truth table, the minterms can be obtained for each output.

$D_4 = \sum m(10, 11, 12, 13, 14, 15)$

$D_3 = \sum m(8, 9)$

$D_2 = \sum m(4, 5, 6, 7, 14, 15)$

$D_1 = \sum m(2, 3, 6, 7, 12, 13)$

$D_0 = \sum m(1, 3, 5, 7, 9, 11, 13, 15)$

The minterms are plotted in the karnaugh map and the simplified boolean expressions are obtained. Learn, How to minimize a boolean function using K-map

Decimal Number	Binary code (Input)				BCD code (Output)				
	B3	B2	B1	Bo	D4	D3	D2	D1	Do
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

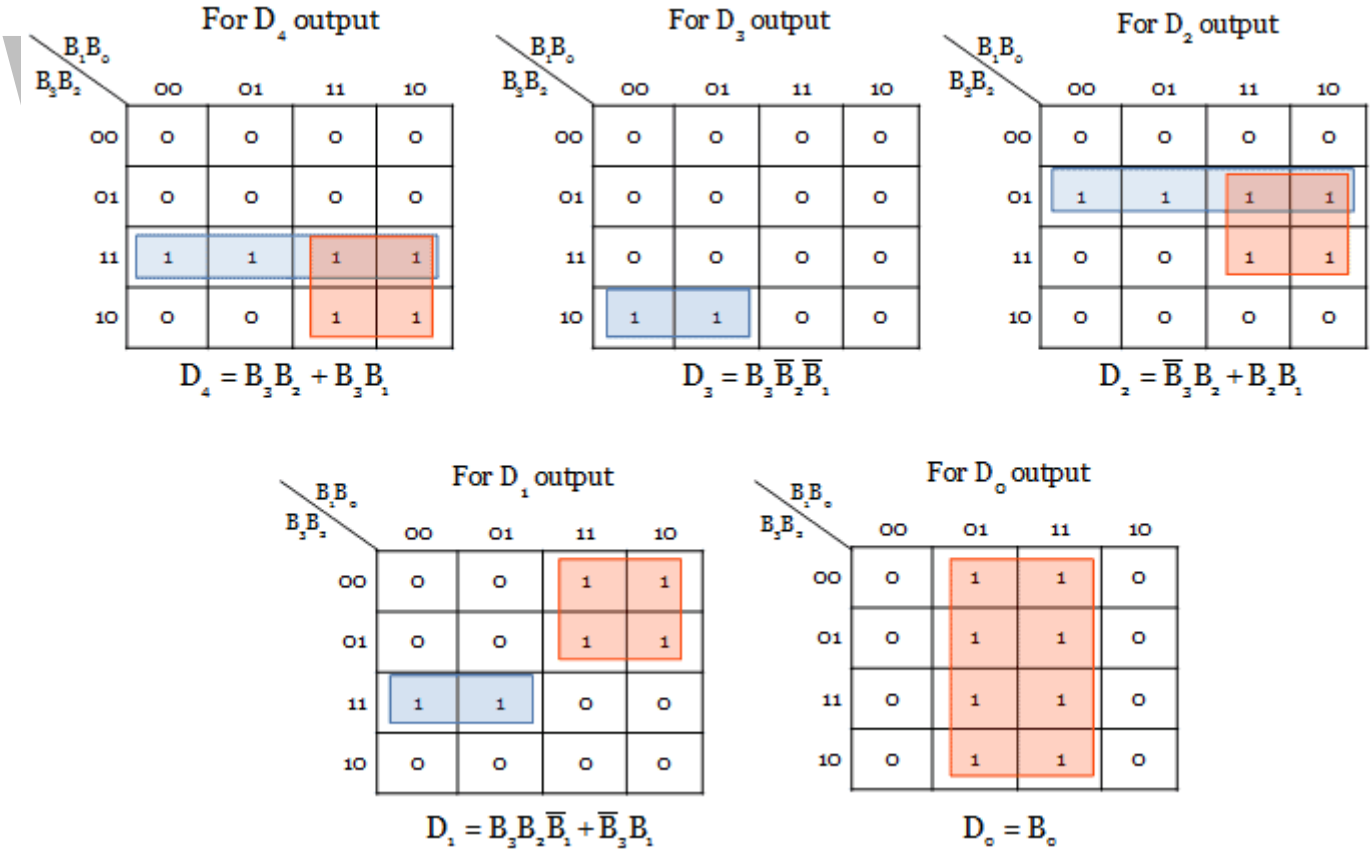


Figure 2.6.1 Truth Table – Kmap –Binary to BCD Code Converter

[Source: https://www.electrically4u.com/code-converter-types-truth-table-and-logic-circuits/]

The digital logic circuit for Binary to BCD code converter is designed from the simplified output expressions obtained from karnaugh map.

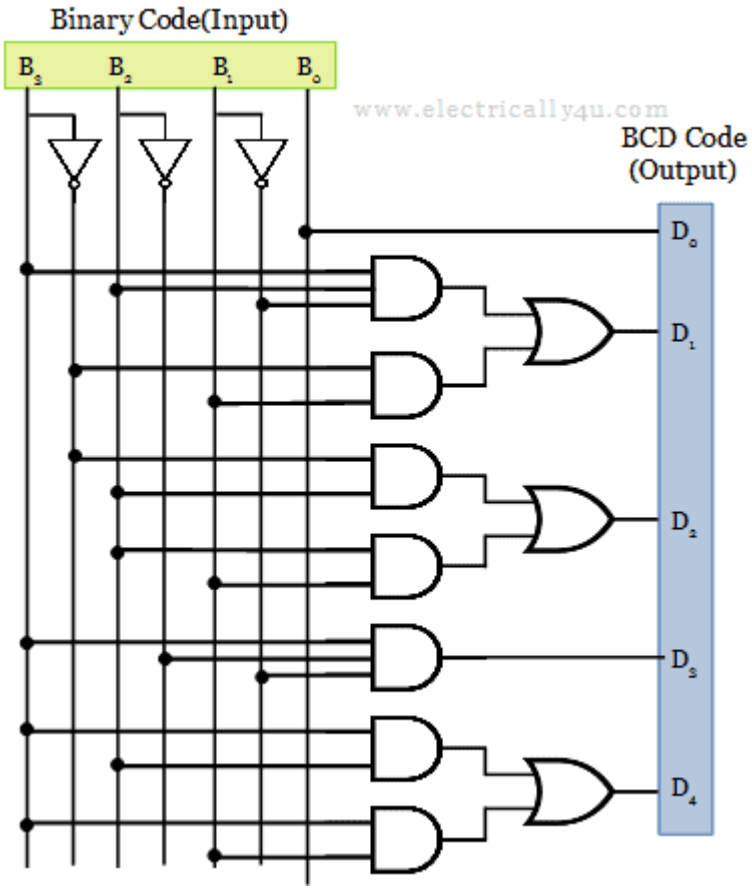


Figure 2.6.2 Binary to BCD Code Converter

[Source: https://www.electrically4u.com/code-converter-types-truth-table-and-logic-circuits/]

2.6.2 BCD to Excess-3 code converter

For this conversion process, 4-bit BCD code is considered as input, which is converted into 4-bit Excess-3 code. Since 4-bit is considered for BCD code, the output is produced only for the inputs from 0 to 9.

The truth table shown below has only the valid 4-bit BCD codes. For the remaining input combinations, the output cannot be predicted. So they are don't care outputs.

From the truth table, the minterms are obtained for each outputs(E3, E2, E1, E0).

$E_3 = \sum m(5, 6, 7, 8, 9)$, $E_2 = \sum m(1, 2, 3, 4, 9)$, $E_1 = \sum m(0, 3, 4, 7, 8)$, $E_0 = \sum m(0, 2, 4, 6, 8,)$ the minterms of each output in plotted in k-map and simplified expression is obtained.

Decimal Number	BCD code (Input)				Excess-3 code (Output)			
	D3	D2	D1	Do	E3	E2	E1	Eo
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

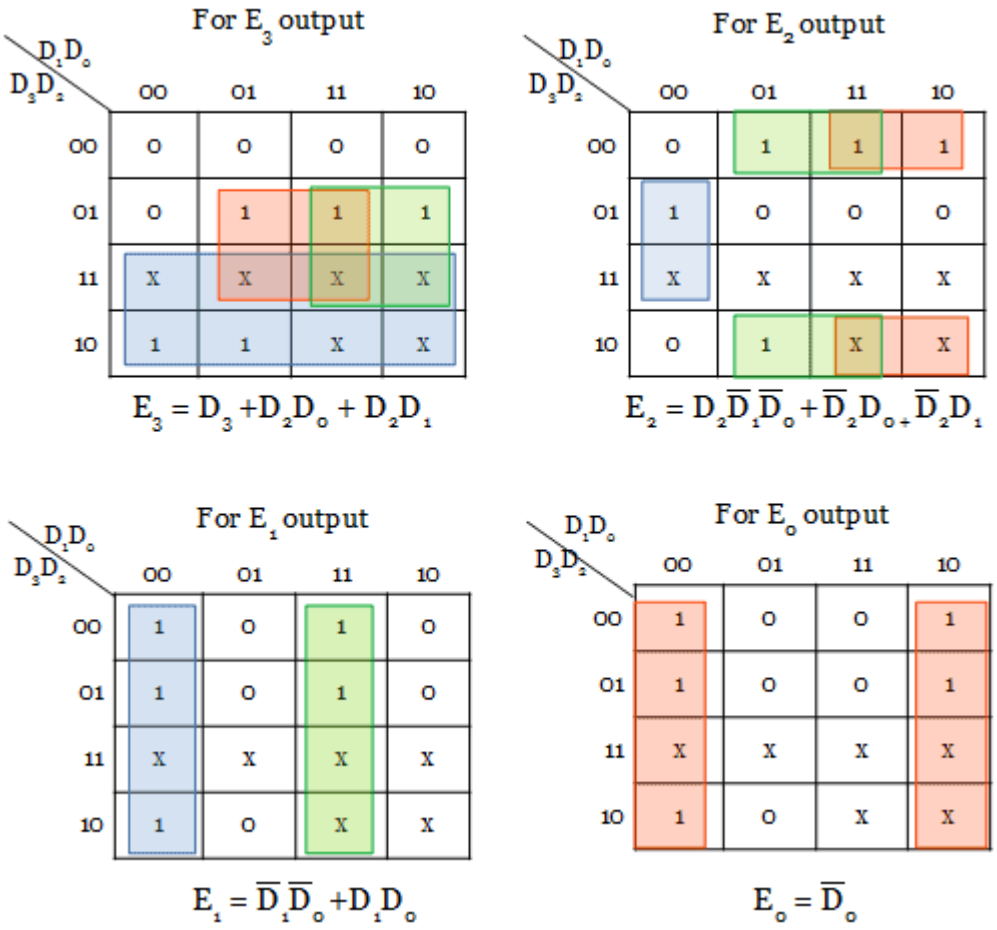


Figure 2.6.3 Truth Table BCD to Excess-3 code converter

[Source: https://www.electrically4u.com/code-converter-types-truth-table-and-logic-circuits/]

The combinational logic circuit for BCD code to Excess-3 code conversion is drawn from the obtained boolean expressions.

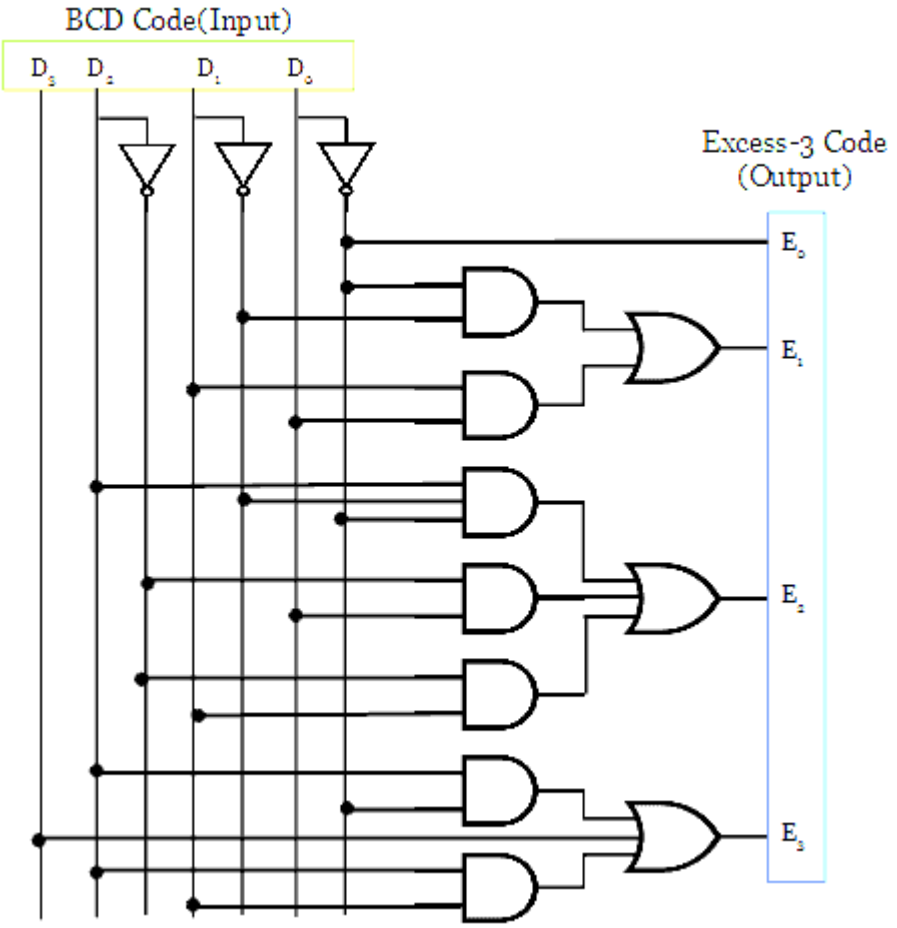


Figure 2.6.4 BCD to Excess-3 code converter

[Source: https://www.electrically4u.com/code-converter-types-truth-table-and-logic-circuits/]

2.6.3 BCD TO GRAY CODE CONVERTER

The truth table having the conversion from BCD code to gray code is shown below. Since the BCD code has only 4 bits, a total of 9 BCD digits have been considered. The output is unpredictable for other input combinations.

From the minterms of each output G3, G2, G1 and G0, the karnaugh map is implemented to simplify the function.

The code converter circuit for BCD to gray code is drawn as below from the obtained expression.

Decimal Number	BCD code (Input)				Gray code (Output)			
	D_3	D_2	D_1	D_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
0	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1

For G_3 output

D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	1	1	X	X

$G_3 = D_3$

For G_2 output

D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$G_2 = D_2 + D_3$

For G_1 output

D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$G_1 = D_2\overline{D}_1 + \overline{D}_2D_1$

For G_0 output

D_3D_2 \ D_1D_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	X	X	X	X
10	0	1	X	X

$G_0 = \overline{D}_1D_0 + D_1\overline{D}_0$

Figure 2.6.5 Truth Table BCD To Gray Code Converter

[Source: https://www.electrically4u.com/code-converter-types-truth-table-and-logic-circuits/]

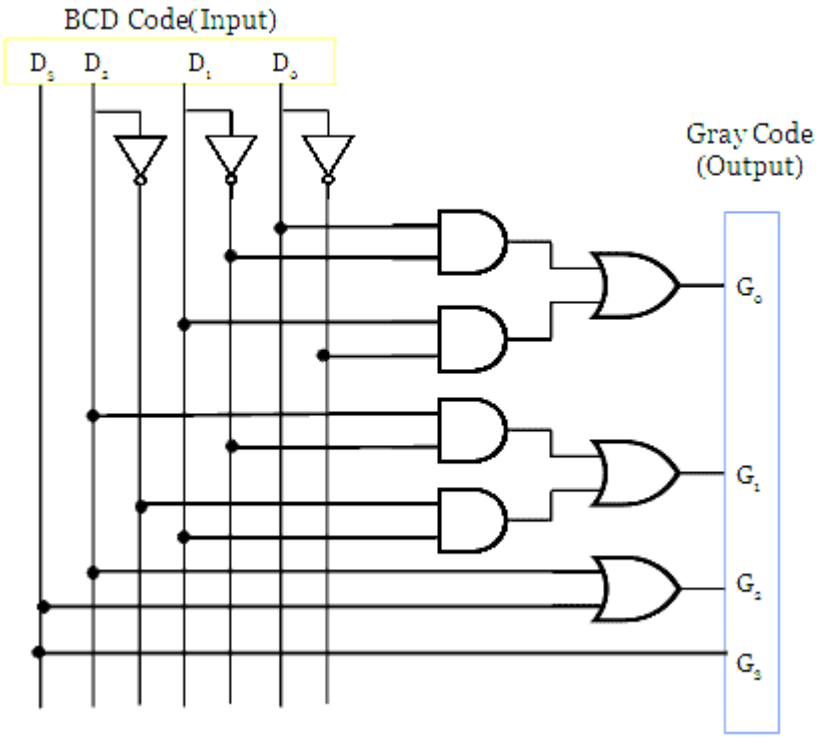


Figure 2.6.6 BCD To Gray Code Converter

[Source: https://www.electrically4u.com/code-converter-types-truth-table-and-logic-circuits/]

2.1 COMBINATIONAL LOGIC

Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following −

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs.

Figure 1.1 Block diagram

[Source: https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm]

2.7 DECODER

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the min terms of 'n' input variables lineslines, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The block diagram of 2 to 4 decoder is shown in the following figure.

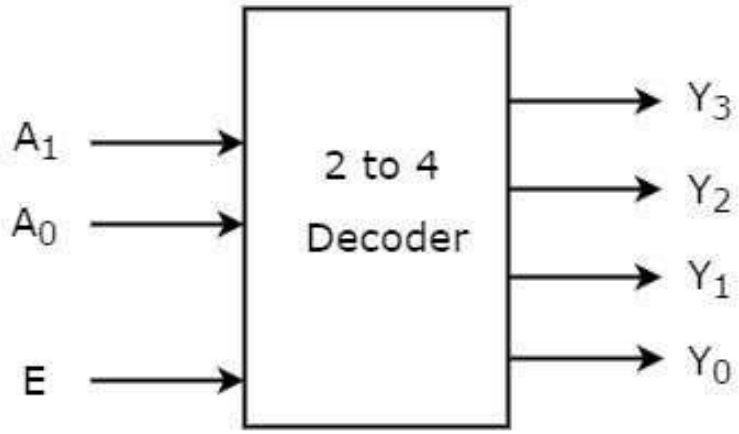


Figure 2.8.1 Block diagram

[Source: https://www.tutorialspoint.com/digital_circuits/digital_circuits_decoders.htm]

One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The Truth table of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the Boolean functions for each output as

Y3=E.A1.A0Y3=E.A1.A0

Y2=E.A1.A0′Y2=E.A1.A0′

Y1=E.A1′.A0Y1=E.A1′.A0

Y0=E.A1′.A0′Y0=E.A1′.A0′

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The circuit diagram of 2 to 4 decoder is shown in the following figure.

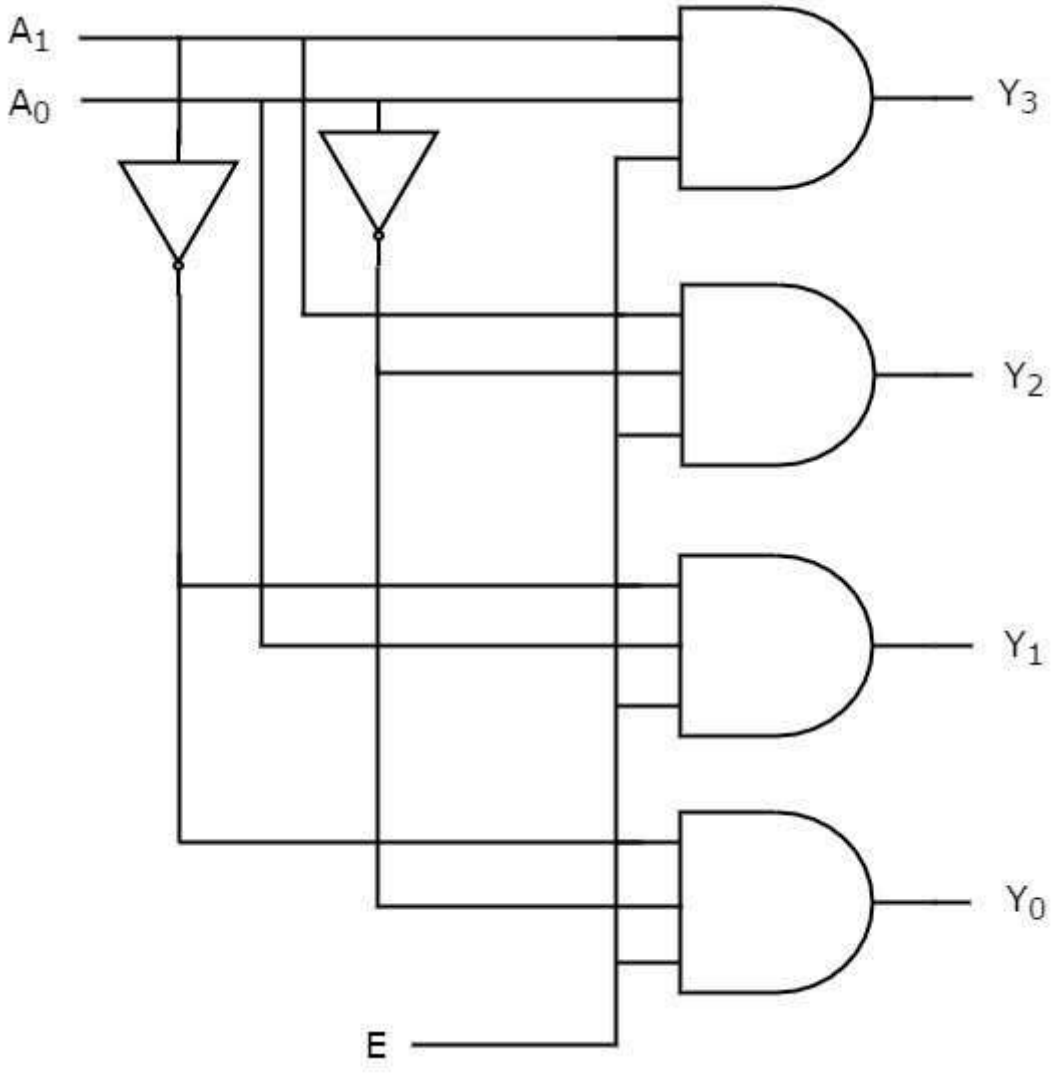


Figure 2.8.2 2 to 4 Decoder

[Source: https://www.tutorialspoint.com/digital_circuits/digital_circuits_decoders.htm]

Therefore, the outputs of 2 to 4 decoder are nothing but the min terms of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.

Implementation of Higher-order Decoders

Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder
- 4 to 16 decoder

3 to 8 Decoder

In this section, let us implement 3 to 8 decoder using 2 to 4 decoders. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.

We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders=m2m1

Where,

m1m1 is the number of outputs of lower order decoder.

m2m2 is the number of outputs of higher order decoder.

Here, m1m1 = 4 and m2m2 = 8. Substitute, these two values in the above formula.

Requirednumberof2to4decoders=84=2Requirednumberof2to4decoders=84=2

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder.

The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the lower four min terms. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the higher four min terms.

The block diagram of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.

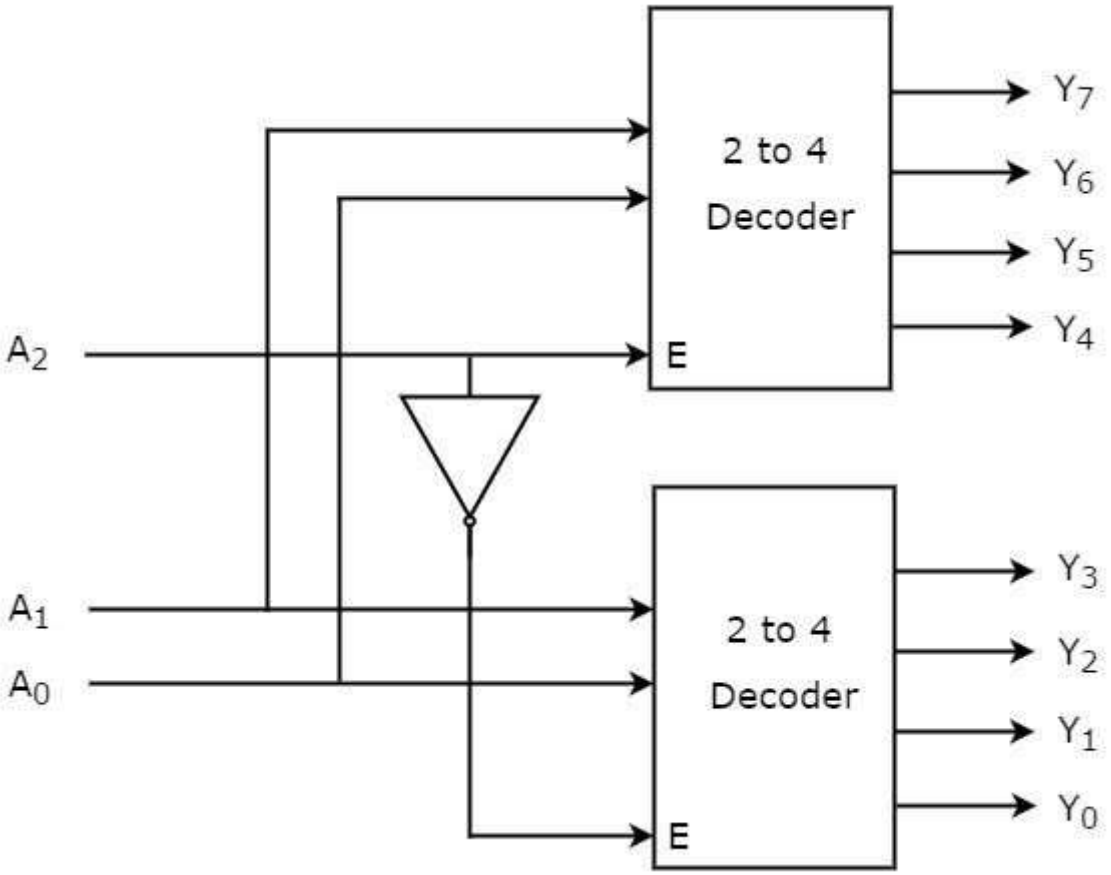


Figure 2.8.3 3 to 8 Decoder

[Source: https://www.tutorialspoint.com/digital_circuits/digital_circuits_decoders.htm]

4 to 16 Decoder

In this section, let us implement 4 to 16 decoder using 3 to 8 decoders. We know that 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0. Whereas, 4 to 16 Decoder has four inputs A_3, A_2, A_1 & A_0 and sixteen outputs, Y_{15} to Y_0

We know the following formula for finding the number of lower order decoders required.
Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders=m2m1

Substitute, m1m1 = 8 and m2m2 = 16 in the above formula.
Requirednumberof3to8decoders=168=2Requirednumberof3to8decoders=168=2
Therefore, we require two 3 to 8 decoders for implementing one 4 to 16 decoder.
The block diagram of 4 to 16 decoder using 3 to 8 decoders is shown in the following figure.

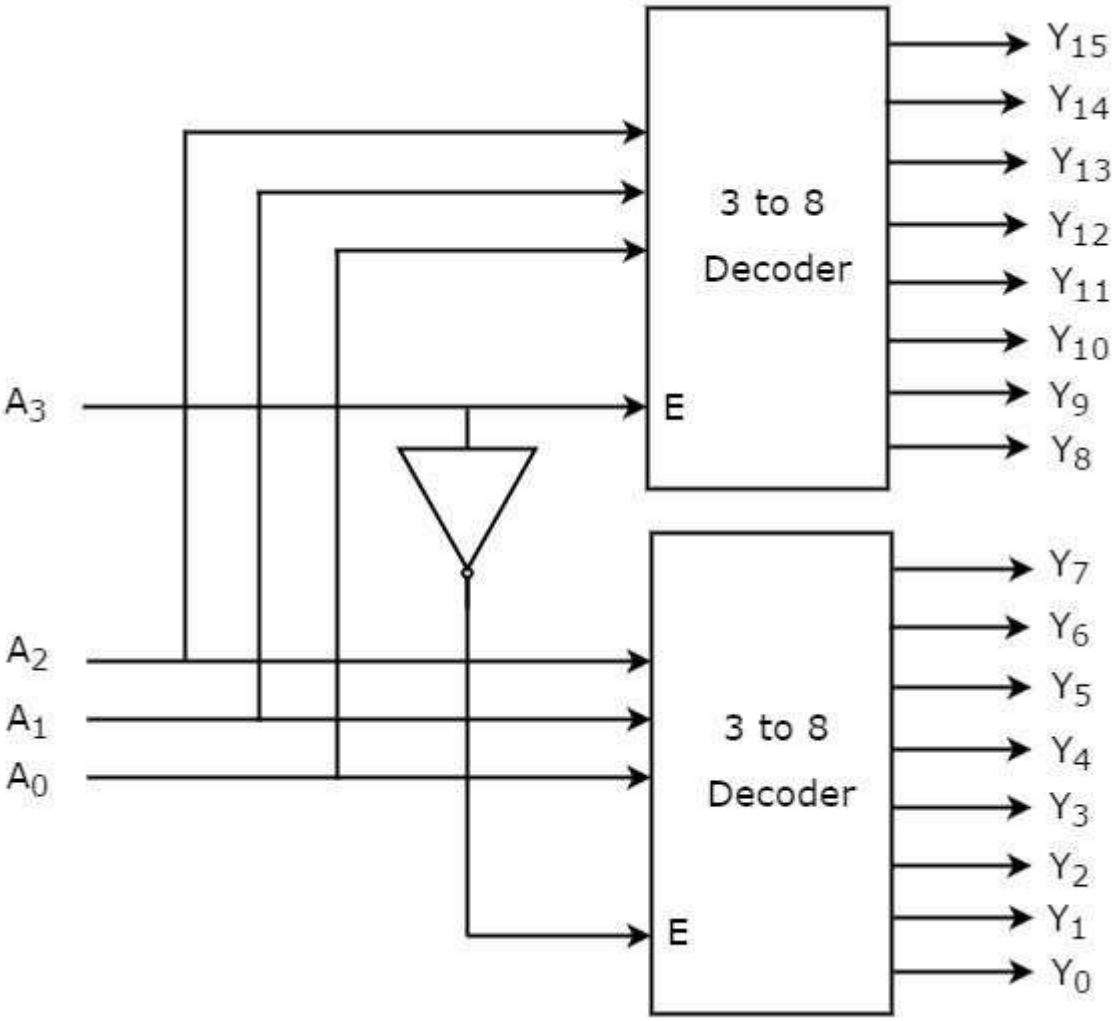


Figure 2.8.4 4 to 16 Decoder

[Source: https://www.tutorialspoint.com/digital_circuits/digital_circuits_decoders.htm]

The parallel inputs A_2, A_1 & A_0 are applied to each 3 to 8 decoder. The complement of input, A3 is connected to Enable, E of lower 3 to 8 decoder in order to get the outputs, Y_7 to Y_0. These are the lower eight min terms. The input, A_3 is directly connected to Enable, E of upper 3 to 8 decoder in order to get the outputs, Y_{15} to Y_8. These are the higher eight min terms.

2.7 ENCODER

An Encoder is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The block diagram of 4 to 2 Encoder is shown in the following figure.

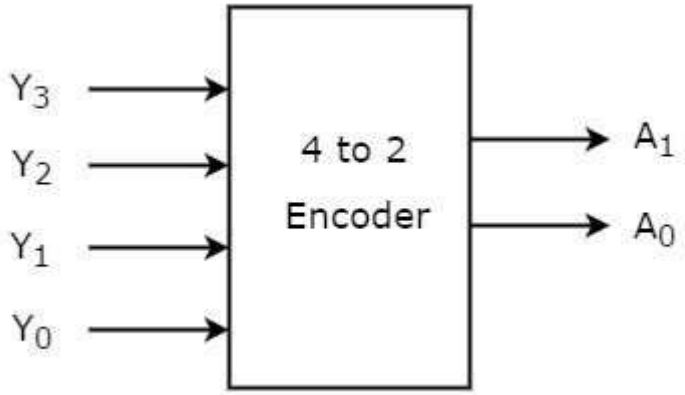


Figure 2.7.1 4 to 2 Encoder

[Source: https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm]

At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The Truth table of 4 to 2 encoder is shown below.

Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the Boolean functions for each output as re.

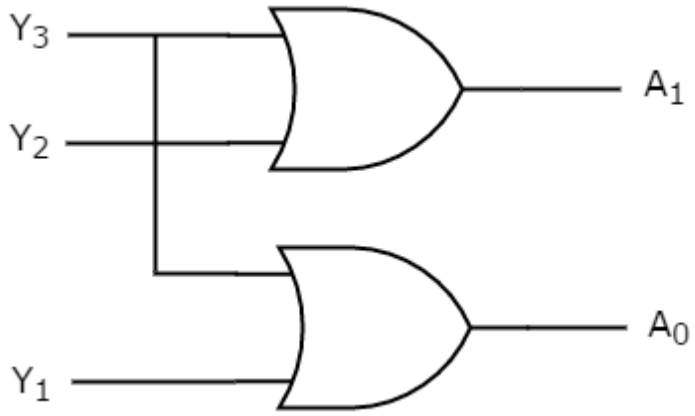


Figure 2.7.2 4 to 2 Encoder- Logic Diagram

[Source: https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm]

The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y7 to Y0 and three outputs A2, A1 & A0. Octal to binary encoder is nothing but 8 to 3 encoder. The block diagram of octal to binary Encoder is shown in the following figure.

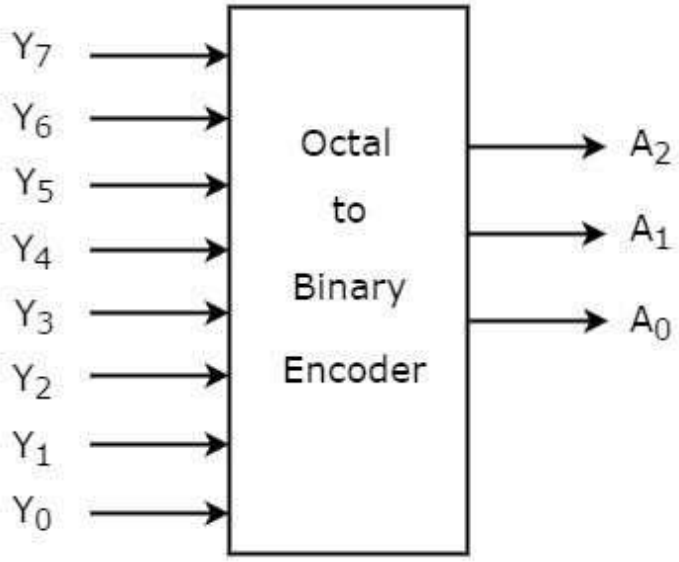


Figure 2.7.3 Block Diagram Octal to Binary Encoder

[Source: https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm]

At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The Truth table of octal to binary encoder is shown below.

Inputs								Outputs		
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

From Truth table, we can write the Boolean functions for each output as

A2=Y7+Y6+Y5+Y4A2=Y7+Y6+Y5+Y4

A1=Y7+Y6+Y3+Y2A1=Y7+Y6+Y3+Y2

A0=Y7+Y5+Y3+Y1A0=Y7+Y5+Y3+Y1

We can implement the above Boolean functions by using four input OR gates. The circuit diagram of octal to binary encoder is shown in the following figure.

The below circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Drawbacks of Encoder

Following are the drawbacks of normal encoder.

- There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.

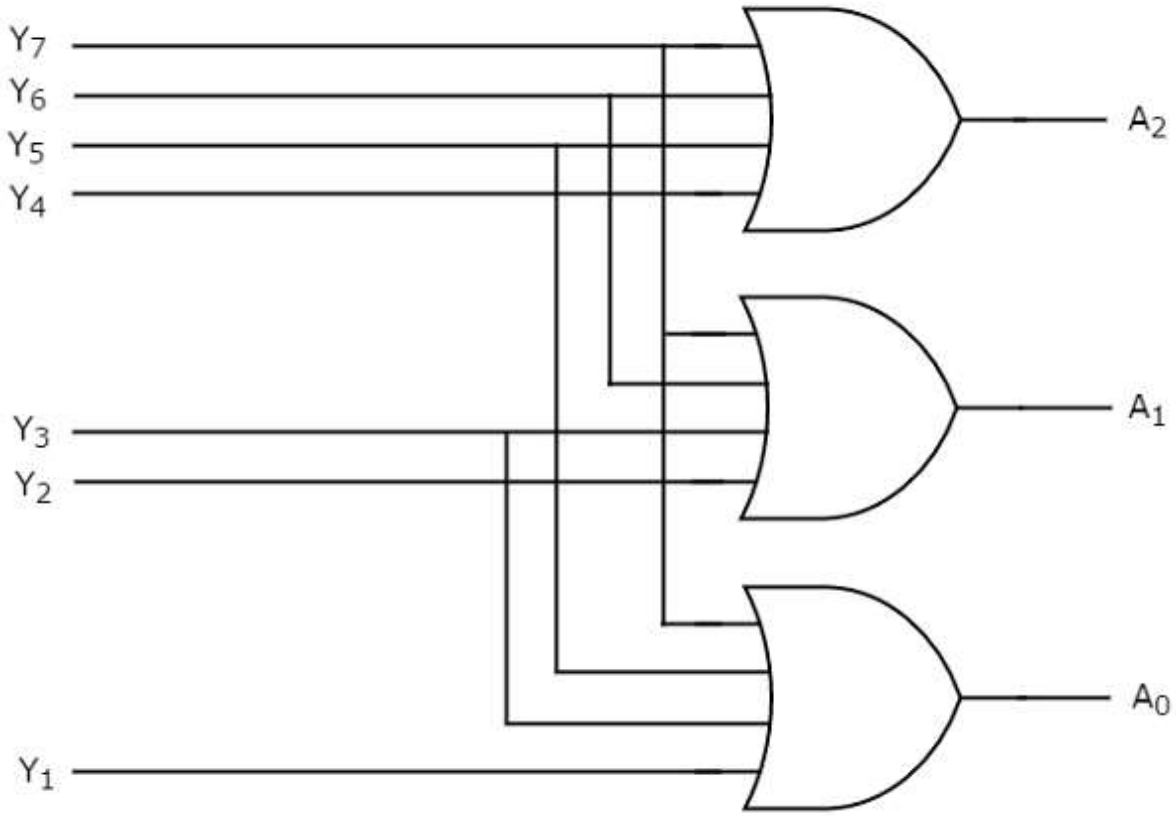


Figure 2.7.4 Octal to Binary Encoder

[Source: https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm]

Drawbacks of Encoder

Following are the drawbacks of normal encoder.

- There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code. For example, if both Y3 and Y6 are ‘1’, then the encoder produces 111 at the output. This is neither equivalent code corresponding to Y3, when it is ‘1’ nor the equivalent code corresponding to Y6, when it is ‘1’.

So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the binarybinary code corresponding to the active High inputss, which has higher priority. This encoder is called as priority encoder.

Priority Encoder

A 4 to 2 priority encoder has four inputs Y3, Y2, Y1 & Y0 and two outputs A1 & A0. Here, the input, Y3 has the highest priority, whereas the input, Y0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the binarybinary code corresponding to the input, which is having higher priority.

The Truth table of 4 to 2 priority encoder is shown below.

Inputs				Outputs		
Y3	Y2	Y1	Y0	A1	A0	V
0	0	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

Use 4 variable K-maps for getting simplified expressions for each output.

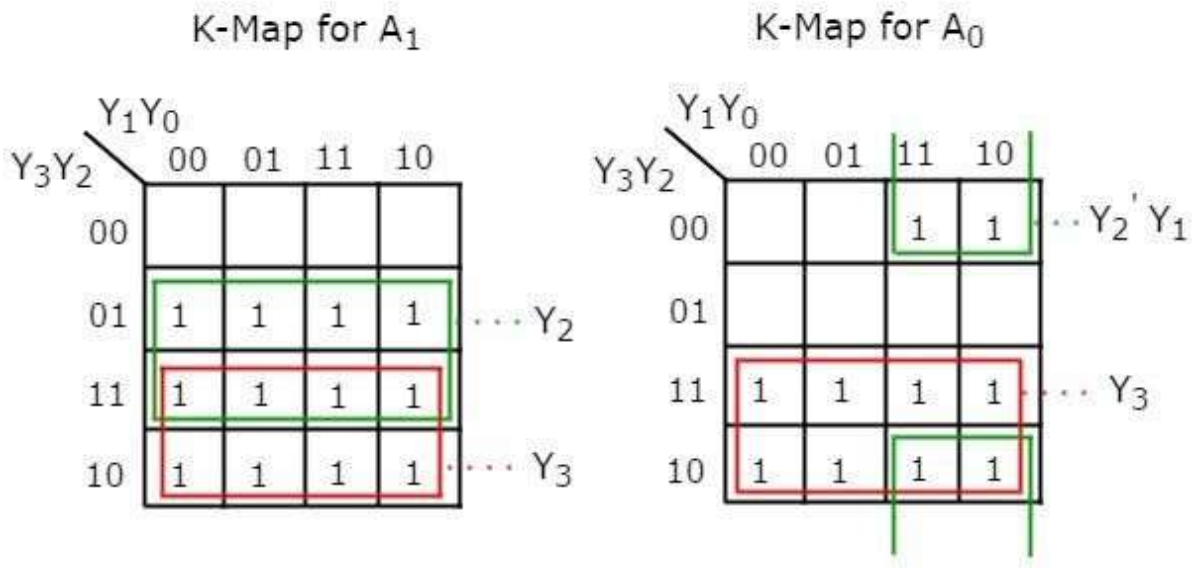


Figure 2.7.5 K-Map

[Source: https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm]

We considered one more output, V in order to know, whether the code available at outputs is valid or not.

- If at least one input of the encoder is '1', then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
- If all the inputs of encoder are '0', then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.

The simplified Boolean functions are

$A1=Y3+Y2A1=Y3+Y2$

$A0=Y3+Y2'Y1A0=Y3+Y2'Y1$

Similarly, we will get the Boolean function of output, V as

$V=Y3+Y2+Y1+Y0V=Y3+Y2+Y1+Y0$

We can implement the above Boolean functions using logic gates. The circuit diagram of 4 to 2 priority encoder is shown in the following figure.

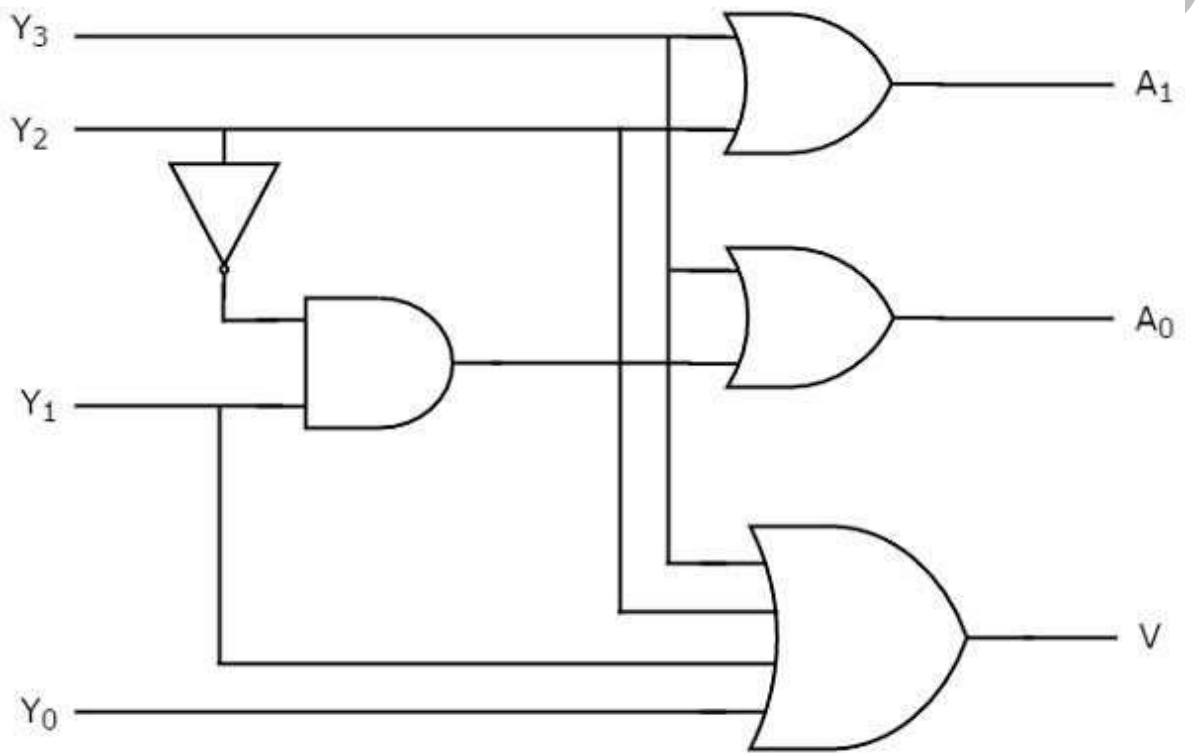


Figure 2.7.6 4 to 2 priority encoder

[Source: https://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm]

The above circuit diagram contains two 2-input OR gates, one 4-input OR gate, one 2input AND gate & an inverter. Here AND gate & inverter combination are used for producing a valid code at the outputs, even when multiple inputs are equal to ‘1’ at the same time. Hence, this circuit encodes the four inputs with two bits based on the priority assigned to each input.

2.3 K-MAP (KARNAUGH MAP)

Karnaugh map method or K-map method is the pictorial representation of the Boolean equations and Boolean manipulations are used to reduce the complexity in solving them. These can be considered as a special or extended version of the 'Truth table'.

By using Karnaugh map technique, we can reduce the Boolean expression containing any number of variables, such as 2-variable Boolean expression, 3-variable Boolean expression, 4-variable Boolean expression and even 7-variable Boolean expressions, which are complex to solve by using regular Boolean theorems and laws.

Minimization with Karnaugh Maps and advantages of K-map

- K-maps are used to convert the truth table of a Boolean equation into minimized SOP form.
- Easy and simple basic rules for the simplification.
- The K-map method is faster and more efficient than other simplification techniques of Boolean algebra.
- All rows in the K-map are represented by using a square shaped cells, in which each square in that will represent a minterm.
- It is easy to convert a truth table to k-map and k-map to Sum of Products form equation.

There are 2 forms in converting a Boolean equation into K-map:

1. Un-optimized form
2. Optimized form

- Un-optimized form: It involves in converting the number of 1's into equal number of product terms (min terms) in an SOP equation.
- Optimized form: It involves in reducing the number of min terms in the SOP equation.

Grouping of K-map variables

- There are some rules to follow while we are grouping the variables in K-maps. They are

- The square that contains ‘1’ should be taken in simplifying, at least once.
- The square that contains ‘1’ can be considered as many times as the grouping is possible with it.
- Group shouldn’t include any zeros (0).
- A group should be the as large as possible.
- Groups can be horizontal or vertical. Grouping of variables in diagonal manner is not allowed.

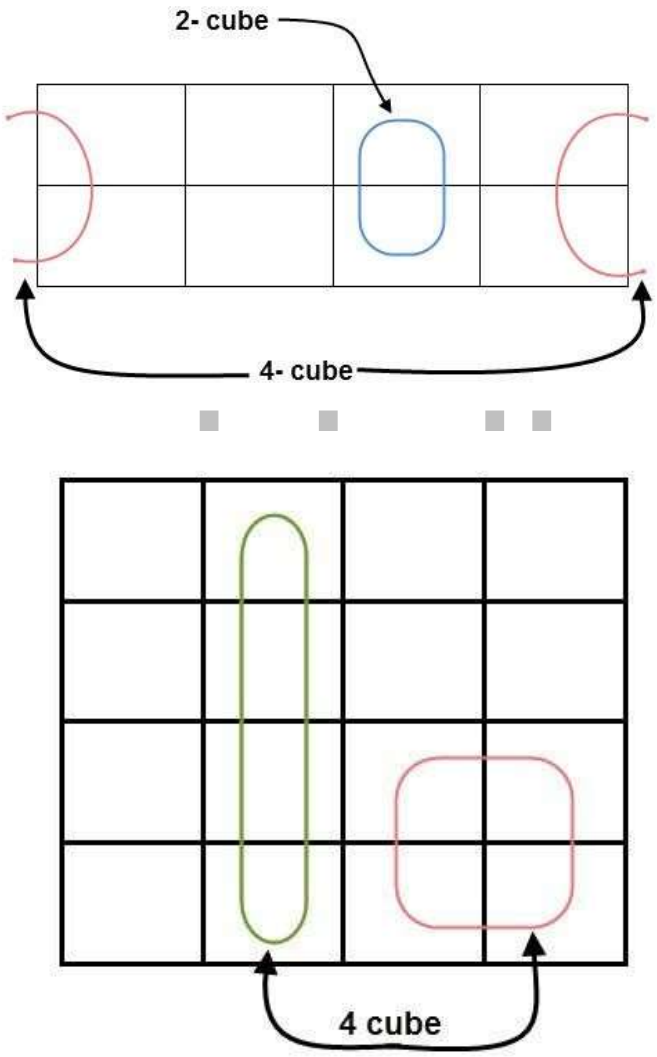


Figure 2.3.1 K-map

[Source: https://www.electronicshub.org/k-map-karnaugh-map/]

- If the square containing ‘1’ has no possibility to be placed in a group, then it should be added to the final expression.
- Groups can overlap.

The number of squares in a group must be equal to powers of 2, such as 1, 2, 4, 8 etc.

- Groups can wrap around. As the K-map is considered as spherical or folded, the squares at the corners (which are at the end of the column or row) should be considered as they adjacent squares.
- The grouping of K-map variables can be done in many ways, so the obtained simplified equation need not to be unique always.
- The Boolean equation must be in must be in canonical form, in order to draw a K-map.

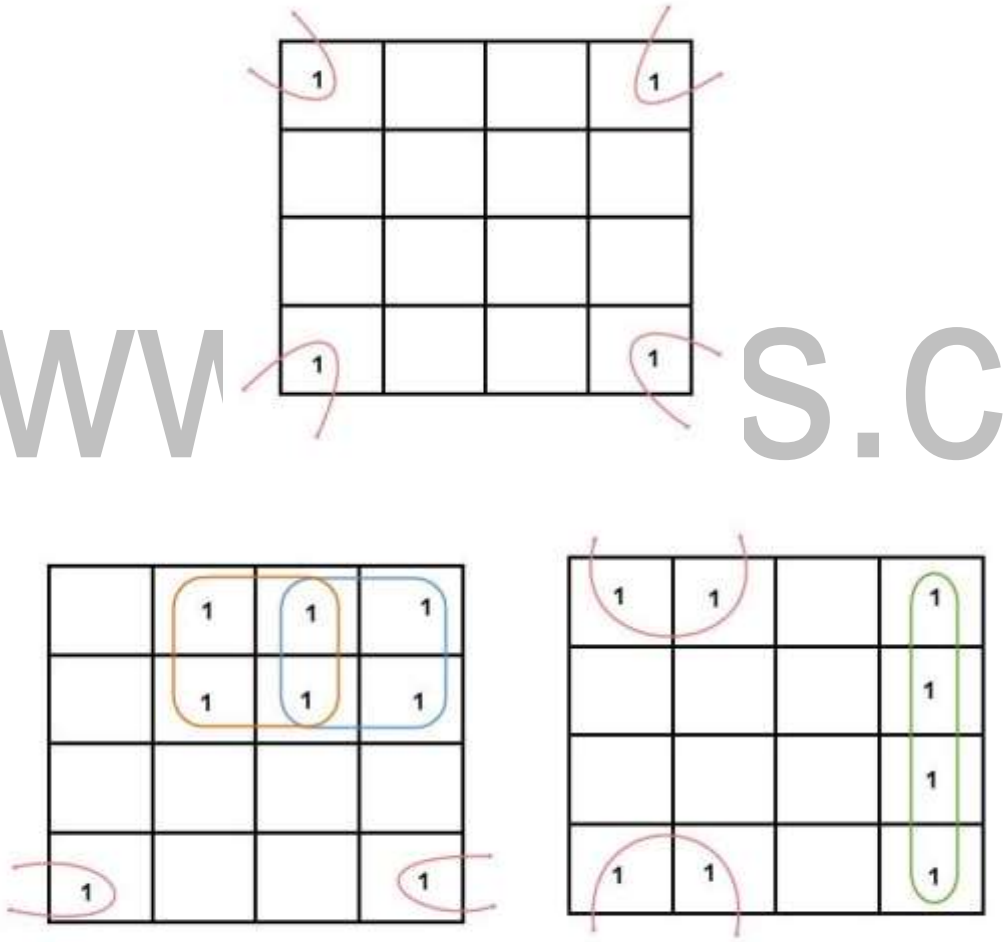
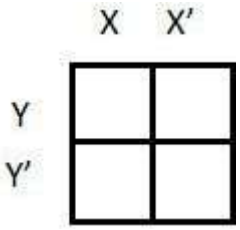


Figure 2.3.2 K-map Combination

[Source: https://www.electronicshub.org/k-map-karnaugh-map/]

2 variable K-maps

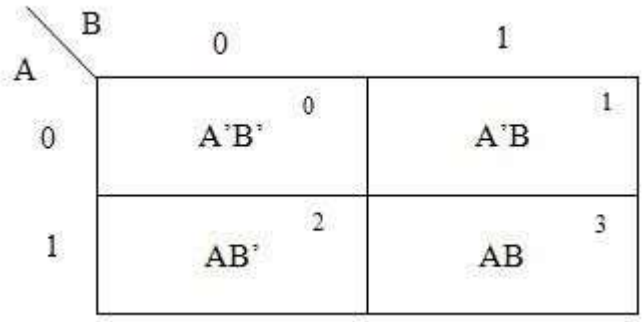
There are 4 cells (22) in the 2-variable k-map. It will look like (see below image)



The possible min terms with 2 variables (A and B) are A.B, A.B', A'.B and A'.B'. The conjunctions of the variables (A, B) and (A', B) are represented in the cells of the top row and (A, B') and (A', B') in cells of the bottom row. The following table shows the positions of all the possible outputs of 2-variable Boolean function on a K-map.

A	B	Possible Outputs	Location on K-map
0	0	A'B'	0
0	1	A'B	1
1	0	AB'	2
1	1	AB	3

A general representation of a 2 variable K-map plot is shown below.



When we are simplifying a Boolean equation using Karnaugh map, we represent the each cell of K-map containing the conjunction term with 1. After that, we group the adjacent cells with possible sizes as 2 or 4. In case of larger k-maps, we can group the variables in larger sizes like 8 or 16.

The groups of variables should be in rectangular shape, that means the groups must be formed by combining adjacent cells either vertically or horizontally. Diagonal shaped or

L-shaped groups are not allowed. The following example demonstrates a K-map simplification of a 2-variable Boolean equation.

Example

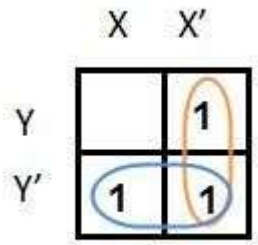
Simplify the given 2-variable Boolean equation by using K-map.

F = X Y' + X' Y + X'Y'

First, let's construct the truth table for the given equation,

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

We put 1 at the output terms given in equation.



In this K-map, we can create 2 groups by following the rules for grouping, one is by combining (X', Y) and (X', Y') terms and the other is by combining (X, Y') and (X', Y') terms. Here the lower right cell is used in both groups. After grouping the variables, the next step is determining the minimized expression.

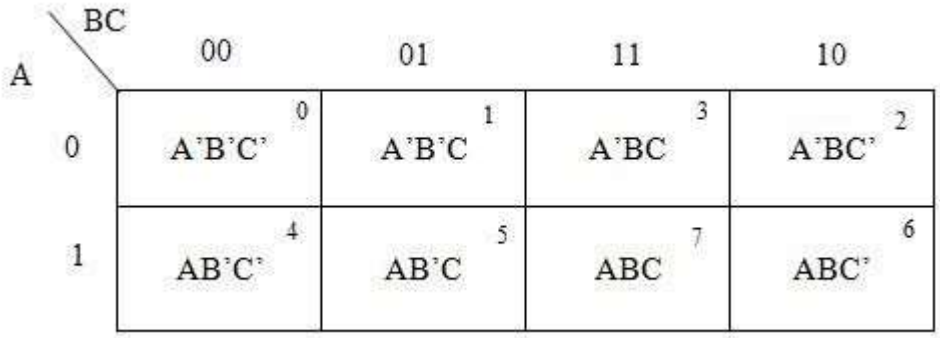
By reducing each group, we obtain a conjunction of the minimized expression such as by taking out the common terms from two groups, i.e. X' and Y'. So the reduced equation will be X' +Y'.

3 variable K-maps

For a 3-variable Boolean function, there is a possibility of 8 output min terms. The general representation of all the min terms using 3-variables is shown below.

A	B	C	Output Function	Location on K-map
0	0	0	A'B'C'	0
0	0	1	A'B'C	1
0	1	0	A'BC'	2
0	1	1	A'BC	3
1	0	0	AB'C'	4
1	0	1	AB'C	5
1	1	0	ABC'	6
1	1	1	ABC	7

A typical plot of a 3-variable K-map is shown below. It can be observed that the positions of columns 10 and 11 are interchanged so that there is only change in one variable across adjacent cells. This modification will allow in minimizing the logic.



Up to 8 cells can be grouped in case of a 3-variable K-map with other possibilities being 1,2 and 4.

Example

Simplify the given 3-variable Boolean equation by using k-map.

F = X' Y Z + X' Y' Z + X Y Z' + X' Y' Z' + X Y Z + X Y' Z'

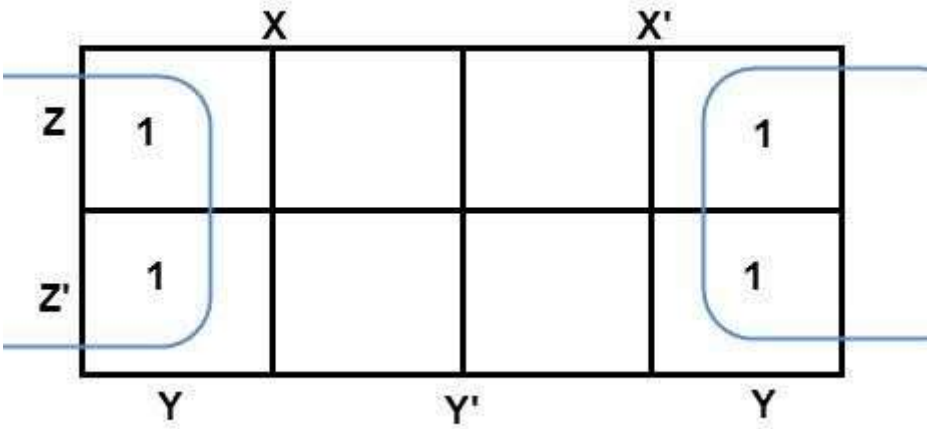
First, let's construct the truth table for the given equation,

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

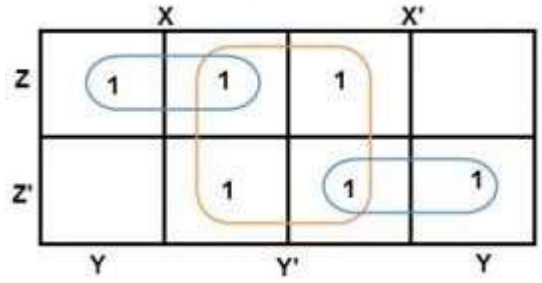
We put 1 at the output terms given in equation.

There are 8 cells (23) in the 3-variable k-map. It will look like (see below image).

The largest group size will be 8 but we can also form the groups of size 4 and size 2, by possibility. In the 3 variable Karnaugh map, we consider the left most column of the k-map as the adjacent column of rightmost column. So the size 4 group is formed as shown below.



And in both the terms, we have ‘Y’ in common. So the group of size 4 is reduced as the conjunction Y. To consume every cell which has 1 in it, we group the rest of cells to form size 2 group, as shown below.



The 2 size group has no common variables, so they are written with their variables and its conjugates. So the reduced equation will be X Z' + Y' + X' Z. In this equation, no further minimization is possible.

4 variable K-maps

There are 16 possible min terms in case of a 4-variable Boolean function. The general representation of minterms using 4 variables is shown below.

A	B	C	D	Output function	K-map location
0	0	0	0	A' B' C' D'	0
0	0	0	1	A' B' C' D	1
0	0	1	0	A' B' C D'	2
0	0	1	1	A' B' C D	3
0	1	0	0	A' B C' D'	4
0	1	0	1	A' B C' D	5
0	1	1	0	A' B C D'	6
0	1	1	1	A' B C D	7
1	0	0	0	A B' C' D'	8
1	0	0	1	A B' C' D	9
1	0	1	0	A B' C D'	10
1	0	1	1	A B' C D	11
1	1	0	0	A B C' D'	12
1	1	0	1	A B C' D	13
1	1	1	0	A B C D'	14
1	1	1	1	A B C D	15

A typical 4-variable K-map plot is shown below. It can be observed that both the columns and rows of 10 and 11 are interchanged.

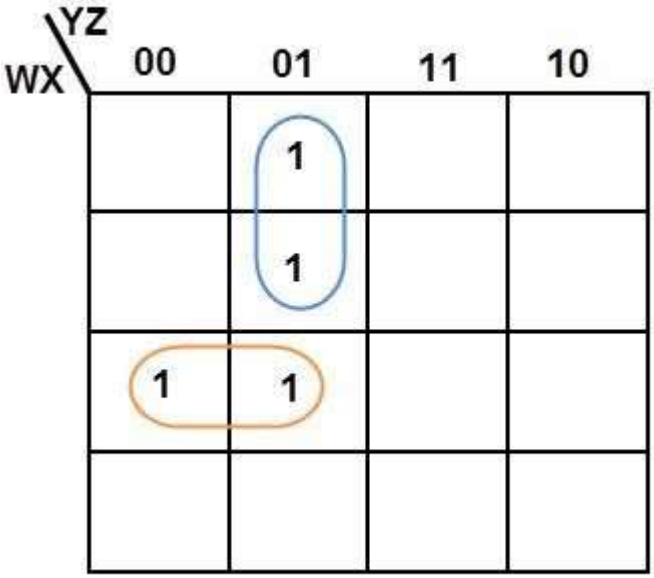
AB \ CD	00	01	11	10
00	0 A' B' C' D'	1 A' B' C' D	3 A' B' C D	2 A' B' C D'
01	4 A' B C' D'	5 A' B C' D	7 A' B C D	6 A' B C D'
11	12 A B C' D'	13 A B C' D	15 A B C D	14 A B C D'
10	8 A B' C' D'	9 A B' C' D	11 A B' C D	10 A B' C D'

The possible number of cells that can be grouped together are 1, 2, 4, 8 and 16.

Example

Simplify the given 4-variable Boolean equation by using k-map. F (W, X, Y, Z) = (1, 5, 12, 13)

Sol: F (W, X, Y, Z) = (1, 5, 12, 13)



By preparing k-map, we can minimize the given Boolean equation as

F = W Y' Z + W 'Y' Z

2.5 MUTLIPLEXER

Multiplexer means many into one. A multiplexer is a circuit used to select and route any one of the several input signals to a single output. A simple example of an non-electronic circuit of a multiplexer is a single pole multi-position switch.

Multi-position switches are widely used in many electronics circuits. However, circuits that operate at high speed require the multiplexer to be automatically selected. A mechanical switch cannot perform this task efficiently. Therefore, multiplexer is used to perform high speed switching are constructed of electronic components.

Multiplexers can handle two type of data i.e., analog and digital. For analog application, multiplexer are built using relays and transistor switches. For digital application, they are built from standard logic gates.

The multiplexer used for digital applications, also called digital multiplexer, is a circuit with many input but only one output. By applying control signals (also known as Select Signals), we can steer any input to the output. Some of the common types of multiplexer are 2-to-1, 4-to-1, 8-to-1, 16-to-1 multiplexer.

Following figure shows the general idea of a multiplexer with n input signal, m control signals and one output signal.

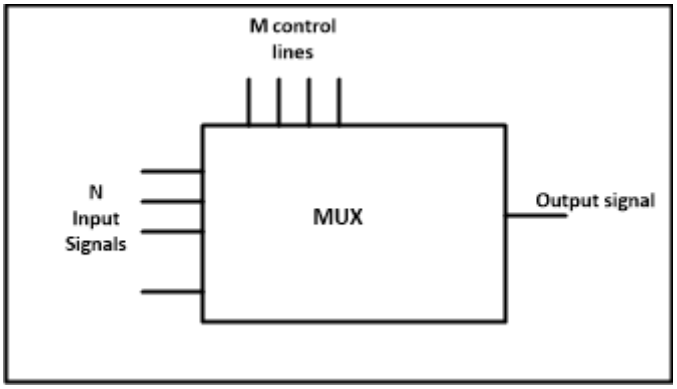


Figure 2.5.1 Multiplexer Block diagram

[Source: https://www.electronicshub.org/multiplexer-and-demultiplexer/]

The 4-to-1 multiplexer has 4 input bits, 2 control or select bits, and 1 output bit. The four input bits are D0,D1,D2 and D3. Only one of this is transmitted to the output Y. The output depends on the values of A and B, which are the control inputs. The control input determines which of the input data bit is transmitted to the output.

For instance, as shown in figure, when A B = 0 0 , the upper AND gate is enabled, while all other AND gates are disabled. Therefore, data bit D0 is transmitted to the output, giving Y = Do.

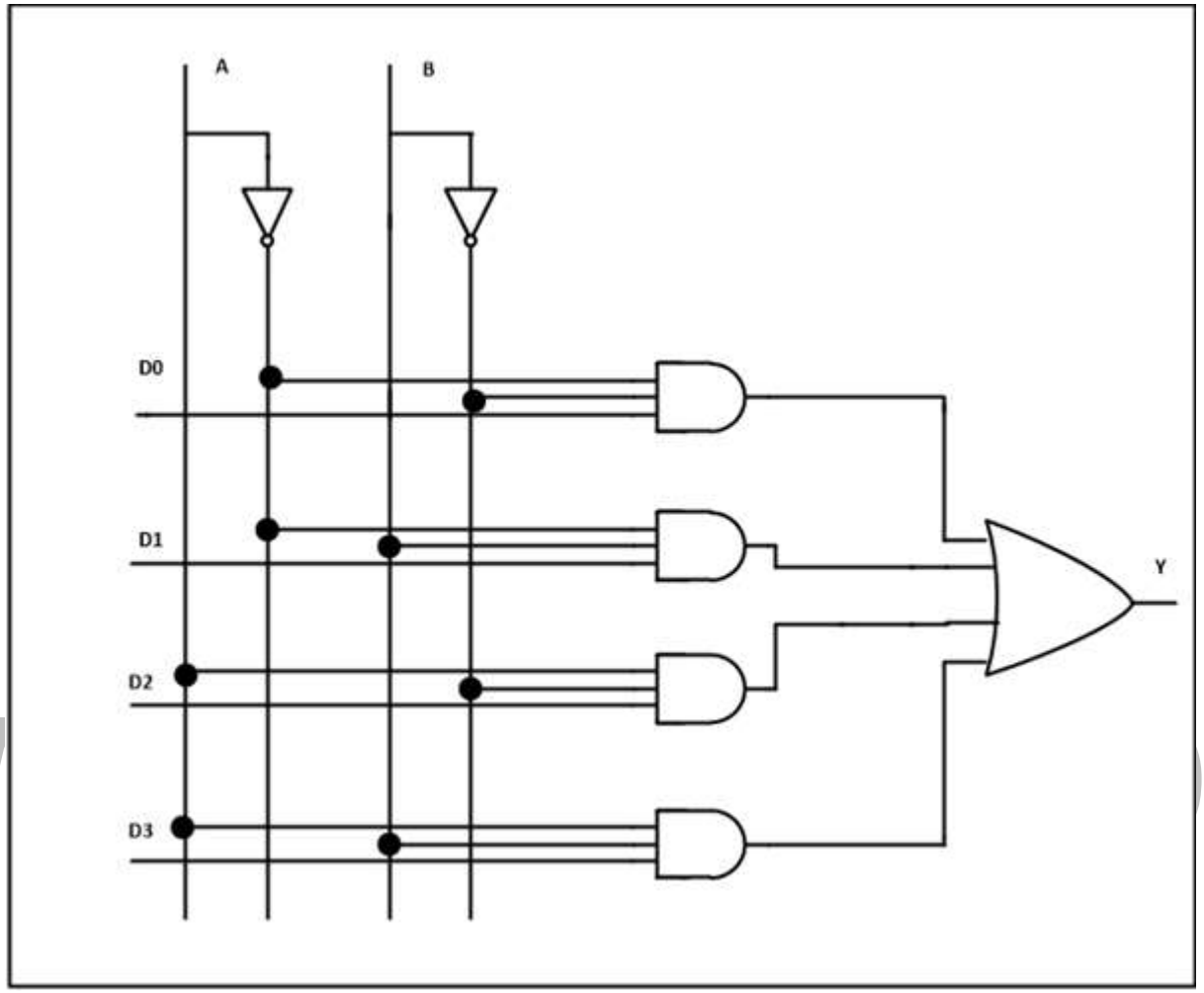


Figure 2.5.2 4-to-1 Multiplexer

[Source: https://www.electronicshub.org/multiplexer-and-demultiplexer/]

If the control input is changed to A B = 1 1 , all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output and Y = D3.

- An example of 4-to-1 multiplexer is IC 74153 in which the output is same as the input.
- Another example of 4-to-1 multiplexer is 45352 in which the output is the compliment of the input.
- Example of 16-to-1 line multiplexer is IC 74150.

Applications of Multiplexer

Multiplexer are used in various fields where multiple data need to be transmitted using a single line. Following are some of the applications of multiplexers –

1. Communication System – Communication system is a set of system that enable communication like transmission system, relay and tributary station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. Multiplexer allow the process of transmitting different type of data such as audio, video at the same time using a single transmission line.
2. Telephone Network – In telephone network, multiple audio signals are integrated on a single line for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.
3. Computer Memory – Multiplexers are used to implement huge amount of memory into the computer, at the same time reduces the number of copper lines required to connect the memory to other parts of the computer circuit.
4. Transmission from the Computer System of a Satellite – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites.

2.5.1 Demultiplexer

Demultiplexer means one to many. A demultiplexer is a circuit with one input and many outputs. By applying control signal, we can steer any input to the output. Few types of demultiplexer are 1-to 2, 1-to-4, 1-to-8 and 1-to 16 demultiplexer.

Following figure illustrate the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.

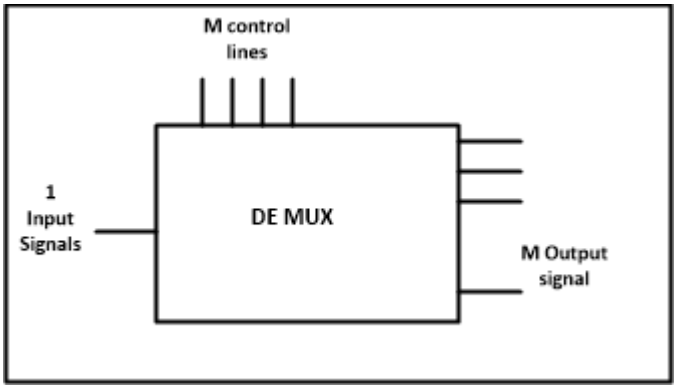


Figure 2.5.3 De-Multiplexer Block diagram

[Source: https://www.electronicshub.org/multiplexer-and-demultiplexer/]

1-to-4 Demultiplexer

The 1-to-4 demultiplexer has 1 input bit, 2 control or select bits, and 4 output bits. An example of 1-to-4 demultiplexer is IC 74155. The 1-to-4 demultiplexer is shown in figure below-

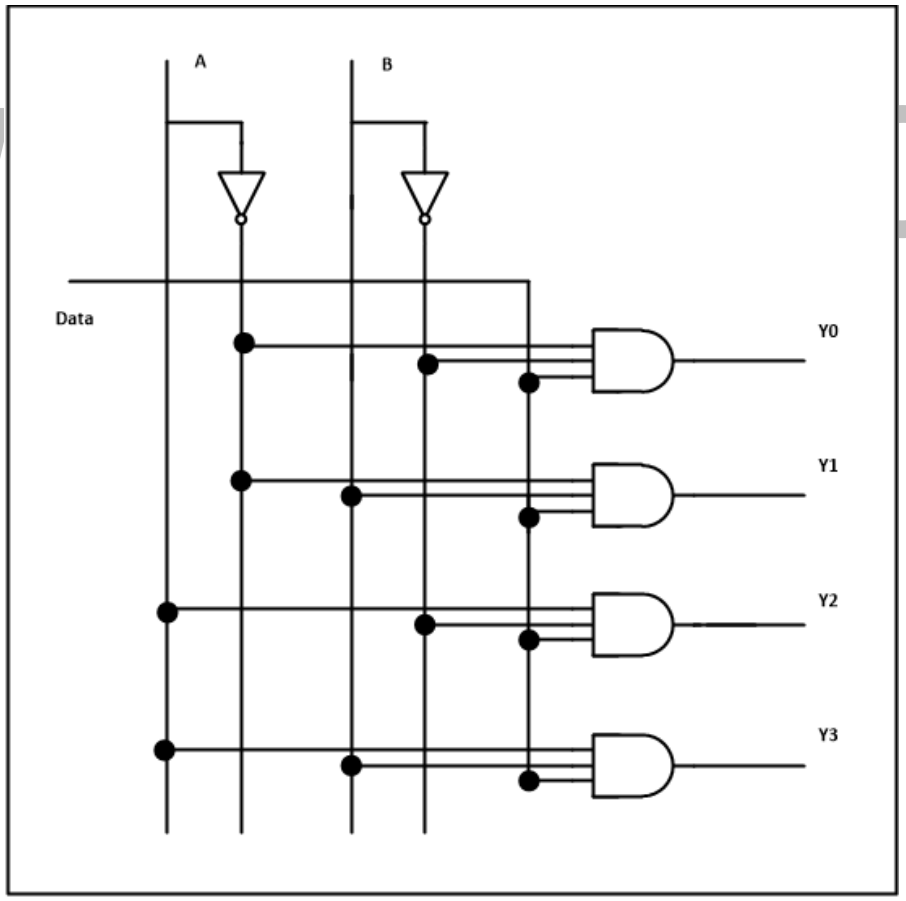


Figure 2.5.4 1-to-4 De-Multiplexer

[Source: https://www.electronicshub.org/multiplexer-and-demultiplexer/]

The input bit is labelled as Data D. This data bit is transmitted to the selected output lines, which depends on the values of A and B, the control or Select Inputs.

When A B = 0 1 , the second AND gate from the top is enabled while other AND gates are disabled. Therefore, data bit D is transmitted to the output Y1, giving Y1 = Data.

If D is LOW, Y1 is LOW. If D is HIGH, Y1 is HIGH. The value of Y1 depends upon the value of D. All other outputs are in low state.

If the control input is changed to A B = 1 0 , all the gates are disabled except the third AND gate from the top. Then, D is transmitted only to the Y2 output, and Y2 = Data.

Example of 1-to-16 demultiplexer is IC 74154. It has 1 input bit, 4 control / select bits and 16 output bit.

Applications of Demultiplexer

1. Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system, where multiplexers are used. Most of the communication system are bidirectional i.e., they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer are also used for reconstruction of parallel data and ALU circuits.
2. Communication System – Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process make the transmission easier. The demultiplexer receive the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.
3. ALU (Arithmetic Logic Unit) – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.
4. Serial to Parallel Converter – A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the

demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

Problem 1

Implement the boolean expression F(A, B, C) = ∑ m(0, 2, 5, 6) using 4 : 1 multiplexer.

Solution:

In the given boolean expression, there are 3 variables. We should use 2^3 : 1 = 8 : 1 multiplexer. But as per the question, it is to be implemented with 4 : 1 mux.

For 4 : 1 multiplexer, there should be 2 selection lines. So from the given 3 variables, the 2 least significant variables(B, C) are used as selection line inputs.

4 : 1 multiplexer implementation table.

		D_0	D_1	D_2	D_3
Row 1	$\overline{A}$	0	1	2	3
Row 2	A	4	5	6	7
		$\overline{A}$	A	1	0

The minterms given in the boolean expression is circled and analyzed. After analyzing, the input values of 4 : 1 mux is obtained as , A, 1, 0. Thus the circuit can be drawn as below.

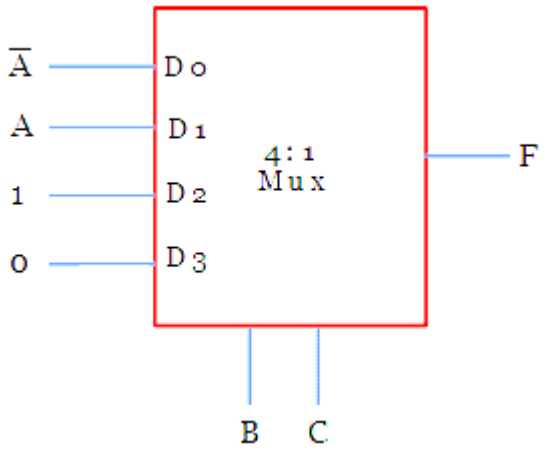


Figure 2.5.5 4 : 1 multiplexer implementation

[Source: https://www.electrically4u.com/solved-problems-on-multiplexer/]

Problem 2

Implement F(A, B, C, D) = ∑ m(0, 1, 5, 6, 8, 10, 12, 15) using 8 : 1 multiplexer.

Solution:

In the given boolean expression, there are 4 variables. We should use 2^4 : 1 = 16 : 1 multiplexer.

The 8 inputs are derived using the implementation table shown below

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	$\overline{A}$	A	0	A	$\overline{A}$	$\overline{A}$	A

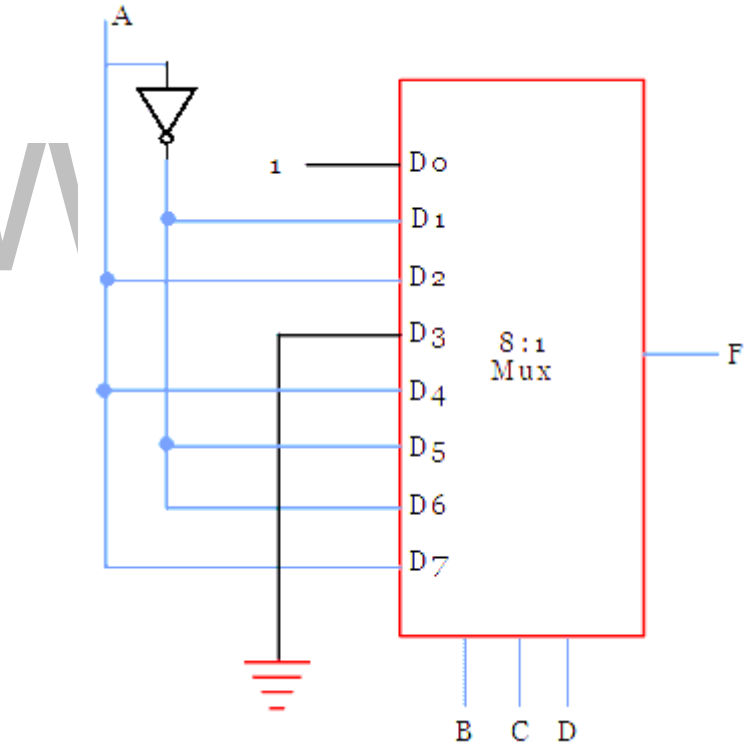


Figure 2.5.6 8 : 1 multiplexer implementation

[Source: https://www.electrically4u.com/solved-problems-on-multiplexer/]

2.2 REPRESENTATION OF LOGIC FUNCTION

The use of switching devices like transistors give rise to a special case of the Boolean algebra called as switching algebra. In switching algebra, all the variables assume one of the two values which are 0 and 1.

In Boolean algebra, 0 is used to represent the 'open' state or 'false' state of logic gate. Similarly, 1 is used to represent the 'closed' state or 'true' state of logic gate.

A Boolean expression is an expression which consists of variables, constants (0-false and 1-true) and logical operators which results in true or false.

A Boolean function is an algebraic form of Boolean expression. A Boolean function of n-variables is represented by f(x1, x2, x3….xn). By using Boolean laws and theorems, we can simplify the Boolean functions of digital circuits. A brief note of different ways of representing a Boolean function is shown below.

- Sum-of-Products (SOP) Form
- Product-of-sums (POS) form
- Canonical forms

There are two types of canonical forms:

- Sum-of-min terms or Canonical SOP
- Product-of- max terms or Canonical POS

Boolean functions can be represented by using NAND gates and also by using K-map (Karnaugh map) method. We can standardize the Boolean expressions by using by two standard forms.

SOP form – Sum Of Products form

POS form – Product Of Sums form

Standardization of Boolean equations will make the implementation, evolution and simplification easier and more systematic.

2.2.1 Sum of Product (SOP) Form

The sum-of-products (SOP) form is a method (or form) of simplifying the Boolean expressions of logic gates. In this SOP form of Boolean function representation, the

variables are operated by AND (product) to form a product term and all these product terms are ORed (summed or added) together to get the final function.

A sum-of-products form can be formed by adding (or summing) two or more product terms using a Boolean addition operation. Here the product terms are defined by using the AND operation and the sum term is defined by using OR operation.

The sum-of-products form is also called as Disjunctive Normal Form as the product terms are ORed together and Disjunction operation is logical OR. Sum-of-products form is also called as Standard SOP.

SOP form representation is most suitable to use them in FPGA (Field Programmable Gate Arrays).

Examples

AB + ABC + CDE

$(AB)^{\overline{}}$ + ABC + $CD\bar{E}$

SOP form can be obtained by

- Writing an AND term for each input combination, which produces HIGH output.
- Writing the input variables if the value is 1, and write the complement of the variable if its value is 0.
- OR the AND terms to obtain the output function.

Ex: Boolean expression for majority function F = A'BC + AB'C + ABC ' + ABC

Truth table:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Now write the input variables combination with high output. F = AB + BC + AC.

Checking

By Idempotence law, we know that

([ABC + ABC)] + ABC) = (ABC + ABC) = ABC

Now the function F = A'BC + AB'C + ABC ' + ABC

= A'BC + AB'C + ABC' + ([ABC + ABC)] + ABC)

= (ABC + ABC ') + (ABC + AB'C) + (ABC + A'BC)

= AB (C + C ') + A (B + B') C + (A + A') BC

= AB + BC + AC.

2.2.2 Product of Sums (POS) Form

The product of sums form is a method (or form) of simplifying the Boolean expressions of logic gates. In this POS form, all the variables are ORed, i.e. written as sums to form sum terms.

All these sum terms are ANDed (multiplied) together to get the product-of-sum form. This form is exactly opposite to the SOP form. So this can also be said as "Dual of SOP form".

Here the sum terms are defined by using the OR operation and the product term is defined by using AND operation. When two or more sum terms are multiplied by a Boolean OR operation, the resultant output expression will be in the form of product-of-sums form or POS form.

The product-of-sums form is also called as Conjunctive Normal Form as the sum terms are ANDed together and Conjunction operation is logical AND. Product-of-sums form is also called as Standard POS.

Examples

(A+B) * (A + B + C) * (C +D)

$(A+B)^{\overline{}}$* (C + D + $E^{\overline{}}$)

POS form can be obtained by

- Writing an OR term for each input combination, which produces LOW output.
- Writing the input variables if the value is 0, and write the complement of the variable if its value is 1.

- AND the OR terms to obtain the output function.

Ex: Boolean expression for majority function F = (A + B + C) (A + B + C ‘) (A + B’ + C) (A’ + B + C)

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Now write the input variables combination with high output. F = AB + BC + AC.

Checking

By Idempotence law, we know that

[(A + B + C) (A + B + C)] (A + B + C) = [(A + B + C)] (A + B + C) = (A + B + C)

Now the function

F = (A + B) (B + C) (A + C)

= (A + B + C) (A + B + C ‘) (A + B’ + C) (A’ + B + C)

= [(A + B + C) (A + B + C)] (A + B + C) (A + B + C ‘) (A + B’ + C) (A’ + B + C)

= [(A + B + C) (A + B + C ‘)] [(A + B + C) (A’ + B + C)] [(A + B + C) (A + B’ + C)]

= [(A + B) + (C * C ‘)] [(B + C) + (A * A’)] [(A + C) + (B * B’)]

= [(A + B) + 0] [(B + C) + 0] [(A + C) + 0] = (A + B) (B + C) (A + C)

2.2.3 Canonical Form (Standard SOP and POS Form)

Any Boolean function that is expressed as a sum of minterms or as a product of max terms is said to be in its “canonical form”.

It mainly involves in two Boolean terms, “minterms” and “maxterms”.

When the SOP form of a Boolean expression is in canonical form, then each of its product term is called ‘minterm’. So, the canonical form of sum of products function is also

known as “minterm canonical form” or Sum-of-minterms or standard canonical SOP form.

Similarly, when the POS form of a Boolean expression is in canonical form, then each of its sum term is called ‘maxterm’. So, the canonical form of product of sums function is also known as “maxterm canonical form or Product-of sum or standard canonical POS form”.

Min terms

A minterm is defined as the product term of n variables, in which each of the n variables will appear once either in its complemented or un-complemented form. The min term is denoted as mi where i is in the range of $0 \leq i < 2^n$.

A variable is in complemented form, if its value is assigned to 0, and the variable is un-complimented form, if its value is assigned to 1.

For a 2-variable (x and y) Boolean function, the possible minterms are:
x’y’, x’y, xy’ and xy.

For a 3-variable (x, y and z) Boolean function, the possible minterms are:
x’y’z’, x’y’z, x’yz’, x’yz, xy’z’, xy’z, xyz’ and xyz.

- 1 – Minterms = minterms for which the function F = 1.
- 0 – Minterms = minterms for which the function F = 0.

Any Boolean function can be expressed as the sum (OR) of its 1- min terms. The representation of the equation will be

- F(list of variables) = Σ(list of 1-min term indices)

Ex: F (x, y, z) = Σ (3, 5, 6, 7)

The inverse of the function can be expressed as a sum (OR) of its 0- min terms. The representation of the equation will be

- F(list of variables) = Σ(list of 0-min term indices)

Ex: F’ (x, y, z) = Σ (0,1, 2, 4)

Examples of canonical form of sum of products expressions (min term canonical form):
i) Z = XY + XZ′
ii) F = XYZ′ + X′YZ + X′YZ′ + XY′Z + XYZ

In standard SOP form, the maximum possible product terms for n number of variables are given by 2^n. So, for 2 variable equations, the product terms are 22 = 4. Similarly, for 3 variable equations, the product terms are 23 = 8.

Max terms

A max term is defined as the product of n variables, within the range of $0 \leq i < 2^n$. The max term is denoted as Mi. In max term, each variable is complimented, if its value is assigned to 1, and each variable is un-complimented if its value is assigned to 0.

For a 2-variable (x and y) Boolean function, the possible max terms are:

x + y, x + y', x' + y and x' + y'.

For a 3-variable (x, y and z) Boolean function, the possible maxterms are:

x + y + z, x + y + z', x + y' + z, x + y' + z', x' + y + z, x' + y + z', x' + y' + z and x' + y' + z'.

- 1 – Max terms = max terms for which the function F = 1.
- 0 – max terms = max terms for which the function F = 0.

Any Boolean function can be expressed the product (AND) of its 0 – max terms. The representation of the equation will be

- F(list of variables) = Π (list of 0-max term indices)

Ex: F (x, y, z) = Π (0, 1, 2, 4)

The inverse of the function can be expressed as a product (AND) of its 1 – max terms. The representation of the equation will be

- F(list of variables) = Π (list of 1-max term indices)

Ex: F' (x, y, z) = Π (3, 5, 6, 7)

Examples of canonical form of product of sums expressions (max term canonical form):

i. Z = (X + Y) (X + Y′)

ii. F = (X′ + Y + Z′) (X′ + Y + Z) (X′ + Y′ + Z′)

In standard POS form, the maximum possible sum terms for n number of variables are given by 2^n. So, for 2 variable equations, the sum terms are 22 = 4. Similarly, for 3 variable equations, the sum terms are 23 = 8.

Table for 2n min terms and 2n max terms

The below table will make you understand about the representation of the mean terms and max terms of 3 variables.

Variables			Min terms	Max terms
A	B	C	m_i	M_i
0	0	0	A' B' C' = m 0	A + B + C = M 0
0	0	1	A' B' C = m 1	A + B + C' = M 1
0	1	0	A' B C' = m 2	A + B' + C = M 2
0	1	1	A' B C = m 3	A + B' + C' = M 3
1	0	0	A B' C' = m 4	A' + B + C = M 4
1	0	1	A B' C = m 5	A' + B + C' = M 5
1	1	0	A B C' = m 6	A' + B' + C = M 6
1	1	1	A B C = m 7	A' + B' + C' = M 7

Conversions of Canonical Forms

We can represent the one canonical formed equation in other canonical form i.e. we can represent the SOP form of equation in POS form and POS form equation in SOP form. To convert the canonical equations, we interchange the Σ and Π symbols after listing out the index numbers of the equations, which are excluded from the original form of equation.

The important thing to remember about Boolean functions is that, the SOP and POS forms are Duals to each other. There are 2 steps to follow to convert the canonical form of the equations. They are

Step 1: Interchanging the operational symbols, Σ and Π in the equation.

Step 2: Use the De Morgan's principle of Duality to the index numbers of the Boolean function or writing the indexes of the terms that are not presented in the given form of equation.

2.2.4 Conversion of SOP form to POS form

To convert the SOP form into POS form, first we should change the Σ to Π and then write the numeric indexes of missing variables of the given Boolean function.

Example:

The SOP function

F = ∑ A, B, C (0, 2, 3, 5, 7) = A' B' C' + A B' C' + A B' C + ABC' + ABC is written in POS form by

Step 1: changing the operational sign to Π

Step 2: writing the missing indexes of the terms, 001, 100 and 110. Now write the sum form for these noted terms.

001 = (A + B + C) 100 = (A + B' + C') 110 = (A + B' + C')

Writing down the new equation in the form of POS form,

F = Π A, B, C (1, 4, 6) = (A + B + C) * (A + B' + C') * (A + B' + C')

Conversion of POS form to SOP form

To convert the POS form into SOP form, first we should change the Π to Σ and then write the numeric indexes of missing variables of the given Boolean function.

Ex: The POS function F = Π A, B, C (2, 3, 5) = A B' C' + A B' C + ABC' is written in SOP form by

Step 1: changing the operational sign to Σ

Step 2: writing the missing indexes of the terms, 000, 001, 100, 110, and 111. Now write the product form for these noted terms.

000 = A' * B' * C' 001 = A' * B' * C 100 = A * B' * C'

110 = A * B* C' 111 = A * B * C

Writing down the new equation in the form of SOP form,

F = Σ A, B, C (0, 1, 4, 6, 7) = (A' * B' * C') + (A' * B' * C) + (A * B' * C') + (A * B* C') + (A * B * C)

Conversion of SOP form to standard SOP form or Canonical SOP form

We can include all the variables in each product term of the SOP form equation, which doesn't have all the variables by converting into standard SOP form. The normal SOP form function can be converted to standard SOP form by using the Boolean algebraic law, (A + A' = 1) and by following the below steps.

Step 1:

By multiplying each non-standard product term with the sum of its missing variable and its complement, which results in 2 product terms

Step 2:

By repeating the step 1, until all resulting product terms contain all variables

By these two steps we can convert the SOP function into standard SOP function. In this process, for each missing variable in the function, the number of product terms will double.

Example:

Convert the non standard SOP function F = x y + x z + y z Sol:

F = x y + x z + y z

= x y (z + z') + x (y + y') z + (x + x') y z

= x y z + x y z' + x y z + x y' z + x y z + x' y z

= x y z + x y z' + x y' z + x' y z

The standard SOP form is F = x y z + x y z' + x y' z + x' y z

2.2.5 Conversion of POS form to standard POS form or Canonical POS form

We can include all the variables in each product term of the POS form equation, which doesn't have all the variables by converting into standard POS form. The normal POS form function can be converted to standard POS form by using the Boolean algebraic law, (A * A' = 0) and by following the below steps.

Step 1:

By adding each non-standard sum term to the product of its missing variable and its complement, which results in 2 sum terms

Step 2:

Applying Boolean algebraic law, A + BC = (A + B) * (A + C)

Step 3:

By repeating the step 1, until all resulting sum terms contain all variables

By these three steps we can convert the POS function into standard POS function.

Example:

F = (A’ + B + C) * (B’ + C + D’) * (A + B’ + C’ + D)

In the first term, the variable D or D’ is missing, so we add D*D’ = 1 to it. Then

(A’ + B + C + D*D’) = (A’ + B + C + D) * (A’ + B + C + D’)

Similarly, in the second term, the variable A or A’ is missing, so we add A*A’ = 1 to it. Then

(B’ + C + D’ + A*A’) = (A + B’ + C + D’) * (A’ + B’ + C + D’)

The third term is already in the standard form, as it has all the variables. Now the standard POS form equation of the function is

F = (A’ + B + C + D) * (A’ + B + C + D’) * (A + B’ + C + D’) * (A’ + B’ + C + D’) * (A + B’ + C’ + D)