UNIT-IV Combinational Logic

Introduction:

The signals are usually represented by discrete bands of analog levels in digital electronic circuits or digital electronics instead of continuous ranges represented in analogue electronics. The simple electronic representations of Boolean logic functions, large assemblies of logic gates are typically used to make digital electronic circuits. In digital circuit theory, the circuits, thus formed from logic gates are used to generate outputs based on the input logic. Hence, these circuits are called as logic circuits and are classified into two types such as sequential logic and combinational logic circuits.

The logic gates can be defined as simple physical devices used to implement the Boolean function. Logic gates are used to perform a logical operation with one or more inputs and generates a logical output. These logic circuits are formed by connecting one or more logic gates together. These logic circuits are classified into two types: sequential logic circuits and combinational logic circuits.

❖ Combinational Logic Circuit Definition

The combinational logic circuits or time-independent logic circuits in digital circuit theory can be defined as a type of digital logic circuit implemented using Boolean circuits, where the output of logic circuit is a pure function of the present inputs only. The combinational logic circuit operation is instantaneous and these circuits do not have the memory or feedback loops.

This combinational logic is in contrast compared to the sequential logic circuit in which the output depends on both present inputs and also on the previous inputs. Thus, we can say that combinational logic does not have memory, whereas sequential logic stores previous input in its memory. Hence, if the input of combinational logic circuit changes, then the output also changes.

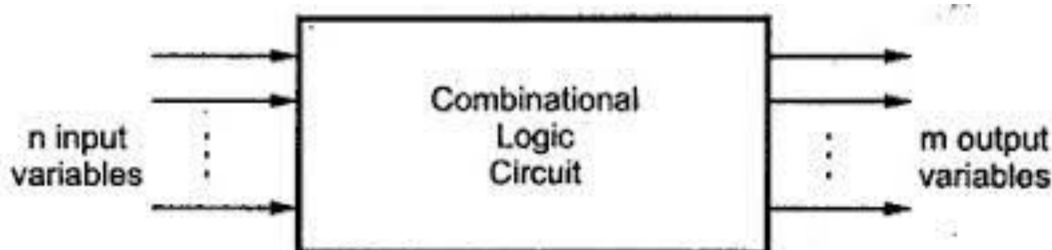
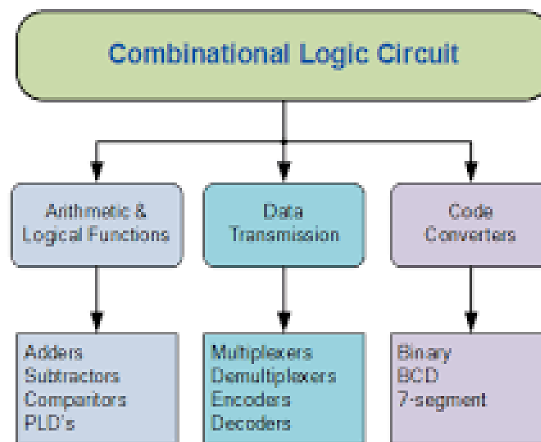


Fig. 3.7 Block diagram of a combinational circuit

❖ *Classification of Combinational Logic*



The combinational logic circuits can be classified into various types based on the purpose of usage, such as arithmetic & logical functions, data transmission, and code converters. To solve the arithmetic and logical functions we generally use adders, subtractors, and comparators which are generally realized by combining various logic gates called as combinational logic circuits. Similarly, for data transmission, we use multiplexers, demultiplexers, encoders, and decoders which are also realized using combinational logic. The code converters such as binary, BCD, and 7-segment are designed using various logic circuits.

❖ ANALYSIS OF COMBINATIONAL LOGIC FUNCTIONS

There are 3 ways to represent combinational logic functions

1. **Logic gates** - Logic gates are used as the building blocks in the design of combinational logic circuits. These gates are the AND, OR, NOT, NAND, NOR gates.
2. **Boolean Algebra** - Boolean Algebra specifies the relationship between Boolean variables which is used to design digital circuits using Logic Gates. Every logic circuit can be completely described using the Boolean operations, because the OR, AND gate, and NOT gates are the basic building blocks of digital systems.
3. **Truth table** - A truth table is used in logic to compute the functional values of logical expressions on each combination of values taken by their logical variables. If a combination logic block have more than one bit output, each single-bit output gets its own truth-table. Often they are combined into a single table with multiple output columns, one for each single-bit output.

❖ Design procedure

The design procedure for combinational logic circuits starts with the problem specification and comprises the following steps:

1. Determine required number of inputs and outputs from the specifications.
2. Derive the truth table for each of the outputs based on their relationships to the input.
3. Simplify the boolean expression for each output. Use Karnaugh Maps or Boolean algebra.
4. Draw a logic diagram that represents the simplified Boolean expression. Verify the design by analysing or simulating the circuit.

EXAMPLE: *Is input greater than or equal to 5?*

Specification

Design a circuit that has a 3-bit binary input and a single output (Z) specified as follows:

- Z = 0, when the input is less than 5_{10}
- Z = 1, otherwise

1. Determine the inputs and Outputs

1. Label the inputs (3 bits) as A, B, C
    - A is the most significant bit
    - C is the least significant bit
2. The output (1 bit) is Z
    - Z = 1 -> 101_2, 110_2, 111_2
    - Z = 0 -> other inputs

2. Derive the Truth Table

Truth Table

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

3. **Simplify the Boolean Expression**

From the truth table, we use one of the following 2 methods to obtain the simplified boolean expression

- Use Karnaugh Map to minimise the logic or
- From the truth table, get the Canonical Sum of Products boolean expression. Z = A * ~B * C + A * B * ~C + A * B * C
  Use Boolean Algebra to simplify the boolean expression to:
  Z = (B + C) * A

4. **Draw the logic diagram**

Draw a logic diagram that represents the simplified Boolean expression. Verify the design by analysing or simulating the circuit.

Bool Expression

Z = (B + C) * A

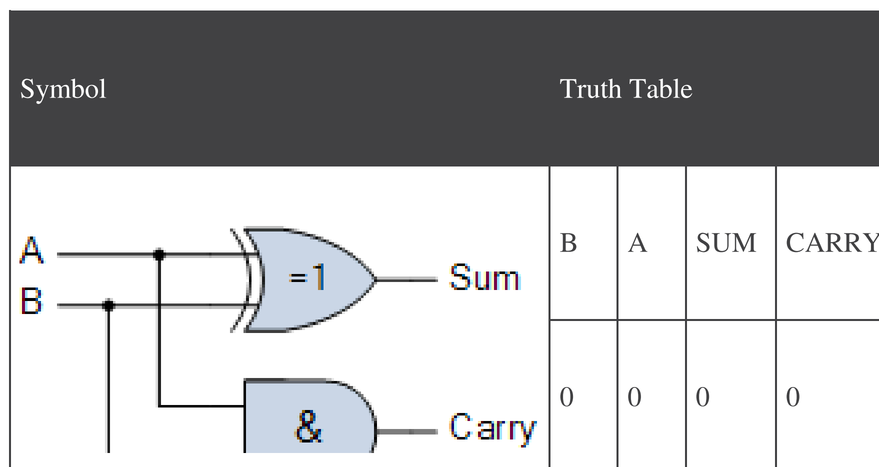
Truth Table

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

❖ A Half Adder Circuit

A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits.

Half Adder Truth Table with Carry-Out



Symbol	Truth Table			
	B	A	SUM	CARRY
	0	0	0	0

	0	1	1	0
	1	0	1	0
	1	1	0	1

From the truth table of the half adder we can see that the SUM (S) output is the result of the Exclusive-OR gate and the Carry-out (Cout) is the result of the AND gate. Then the Boolean expression for a half adder is as follows.

For the **SUM** bit:

SUM = A XOR B = $A \oplus B$

For the **CARRY** bit:

CARRY = A AND B = A.B

One major disadvantage of the *Half Adder* circuit when used as a binary adder, is that there is no provision for a "Carry-in" from the previous circuit when adding together multiple data bits.

❖ Full Adder Truth Table with Carry

Symbol	Truth Table				
	C-in	B	A	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

Then the Boolean expression for a full adder is as follows.

For the **SUM** (S) bit:

SUM = (A XOR B) XOR Cin = (A ⊕ B) ⊕ Cin

For the **CARRY-OUT** (Cout) bit:

CARRY-OUT = A AND B OR Cin(A XOR B) = A.B + Cin(A ⊕ B)

❖ An n-bit Binary Adder

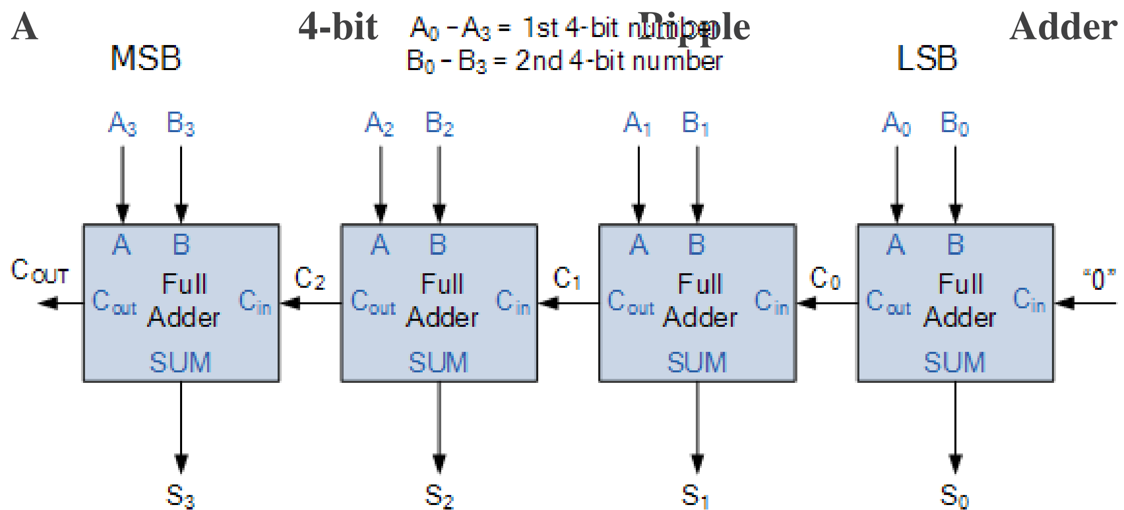
We have seen above that single 1-bit binary adders can be constructed from basic logic gates. But what if we wanted to add together two n-bit numbers, then n number of 1-bit full adders need to be connected or "cascaded" together to produce what is known as a **Ripple Carry Adder**.

A "ripple carry adder" is simply "n", 1-bit full adders cascaded together with each full adder representing a single weighted column in a long binary addition. It is called a ripple carry adder because the carry signals produce a "ripple" effect through the binary adder from right to left, (LSB to MSB).

For example, suppose we want to "add" together two 4-bit numbers, the two outputs of the first full adder will provide the first place digit sum (S) of the addition plus a carry-out bit that acts as the carry-in digit of the next binary adder.

The second binary adder in the chain also produces a summed output (the 2nd bit) plus another carry-out bit and we can keep adding more full adders to the combination to add larger numbers, linking the carry bit output from the first full binary adder to the next full adder, and so forth. An example of a 4-bit adder is given below.

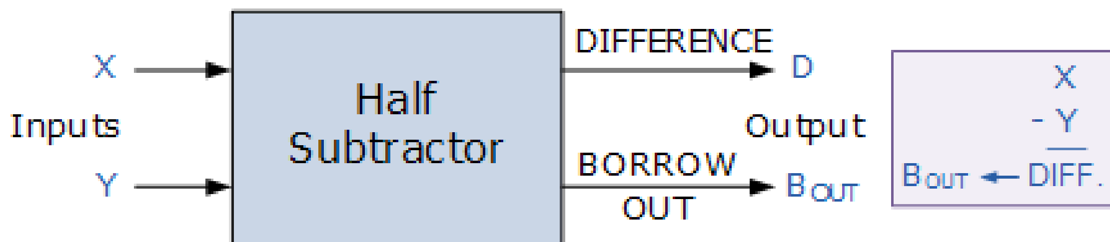
A 4-bit Ripple Adder



❖ A Half Subtractor Circuit

A half subtractor is a logical circuit that performs a subtraction operation on two binary digits. The half subtractor produces a sum and a borrow bit for the next stage.

Half Subtractor with Borrow-out



Symbol	Truth Table			
	Y	X	DIFFERENCE	BORROW
	0	0	0	0
	0	1	1	0
	1	0	1	1
	1	1	0	0

From the truth table of the half subtractor we can see that the DIFFERENCE (D) output is the result of the Exclusive-OR gate and the Borrow-out (Bout) is the result of the NOT-AND combination. Then the Boolean expression for a half subtractor is as follows.

For the **DIFFERENCE** bit:

D = X XOR Y = X ⊕ Y

For the **BORROW** bit

B = not-X AND Y = X.Y

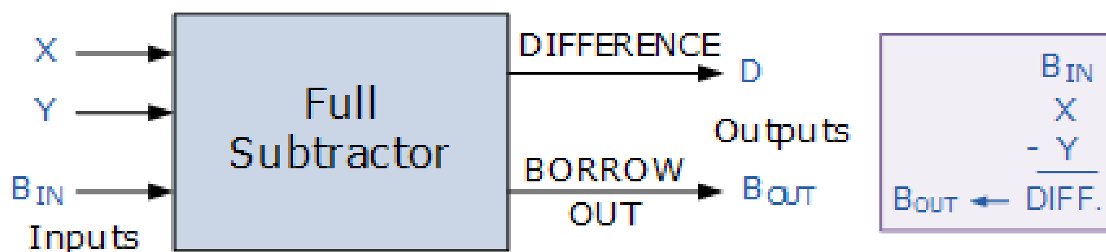
If we compare the Boolean expressions of the half subtractor with a half adder, we can see that the two expressions for the SUM (adder) and DIFFERENCE (subtractor) are exactly the same and so they should be because of the Exclusive-OR gate function. The two Boolean expressions for the binary subtractor BORROW is also very similar to that for the adders CARRY. Then all that is needed to convert a half adder to a half subtractor is the inversion of the minuend input X.

One major disadvantage of the *Half Subtractor* circuit when used as a binary subtractor, is that there is no provision for a "Borrow-in" from the previous circuit when subtracting multiple data bits from each other. Then we need to produce what is called a "full binary subtractor" circuit to take into account this borrow-in input from a previous circuit.

❖ A Full Binary Subtractor Circuit

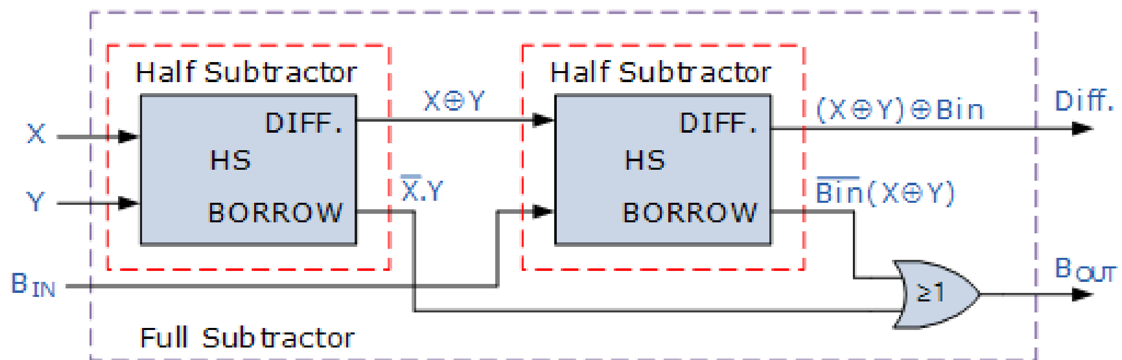
The main difference between the **Full Subtractor** and the previous **Half Subtractor** circuit is that a full subtractor has three inputs. The two single bit data inputs X (minuend) and Y (subtrahend) the same as before plus an additional *Borrow-in* (B-in) input to receive the borrow generated by the subtraction process from a previous stage as shown below.

Full Subtractor Block Diagram



Then the combinational circuit of a "full subtractor" performs the operation of subtraction on three binary bits producing outputs for the difference Dand borrow B-out. Just like the binary adder circuit, the full subtractor can also be thought of as two half subtractors connected together, with the first half subtractor passing its borrow to the second half subtractor as follows.

Full Subtractor Logic Diagram



As the full subtractor circuit above represents two half subtractors cascaded together, the truth table for the full subtractor will have eight different input combinations as there are three input variables, the data bits and the *Borrow-in*, B_{IN} input. Also includes the difference output, D and the Borrow-out, B_{OUT} bit.

Full Subtractor Truth Table

Symbol	Truth Table				
	B-in	Y	X	Diff.	B-out
	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	1
	0	1	1	0	0
	1	0	0	1	1
	1	0	1	0	0
	1	1	0	0	1
	1	1	1	1	1

Then the Boolean expression for a full subtractor is as follows.

For the **DIFFERENCE** (D) bit:

$D = (X.Y.B_{IN}) + (X.Y.B_{IN}) + (X.Y.B_{IN}) + (X.Y.B_{IN})$

which can be simplified too:

$D = (X \text{ XOR } Y) \text{ XOR } B_{IN} = (X \oplus Y) \oplus B_{IN}$
For the **BORROW OUT** (B_{OUT}) bit:

$B_{OUT} = (X.Y.B_{IN}) + (X.Y.B_{IN}) + (X.Y.B_{IN}) + (X.Y.B_{IN})$

which will also simplify too:

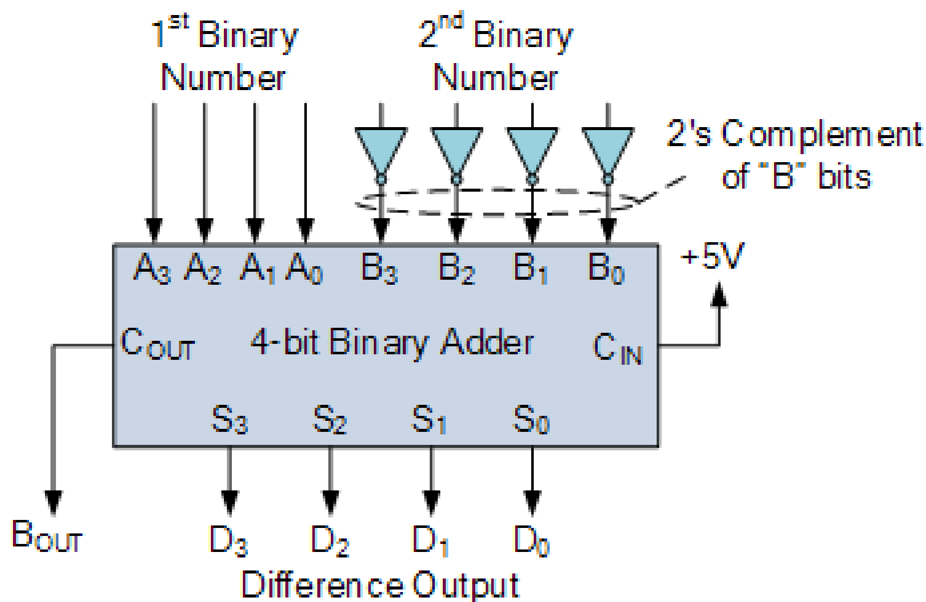
$B_{OUT} = X \text{ AND } Y \text{ OR } (X \text{ XOR } Y)B_{IN} = X.Y + (X \oplus Y)B_{IN}$

❖ An n-bit Binary Subtractor

As with the binary adder, we can also have n number of 1-bit full binary subtractor connected or "cascaded" together to subtract two parallel n-bitnumbers from each other. For example two 4-bit binary numbers. We said before that the only difference between a full adder and a full subtractor was the inversion of one of the inputs.

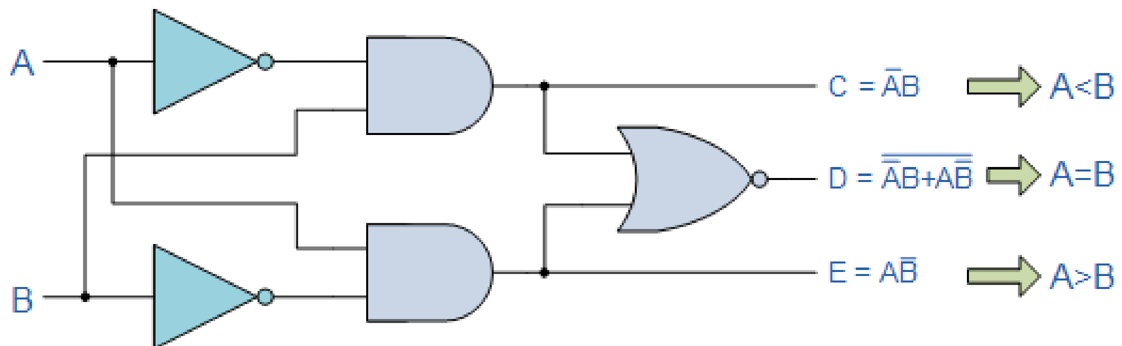
So by using an n-bit adder and n number of inverters (NOT Gates), the process of subtraction becomes an addition as we can use two's complement notation on all the bits in the subtrahend and setting the carry input of the least significant bit to a logic "1" (HIGH).

Binary Subtractor using 2's Complement



❖ COMPARATOR

1-bit Digital Comparator Circuit



Then the operation of a 1-bit digital comparator is given in the following Truth Table.

Digital Comparator Truth Table

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

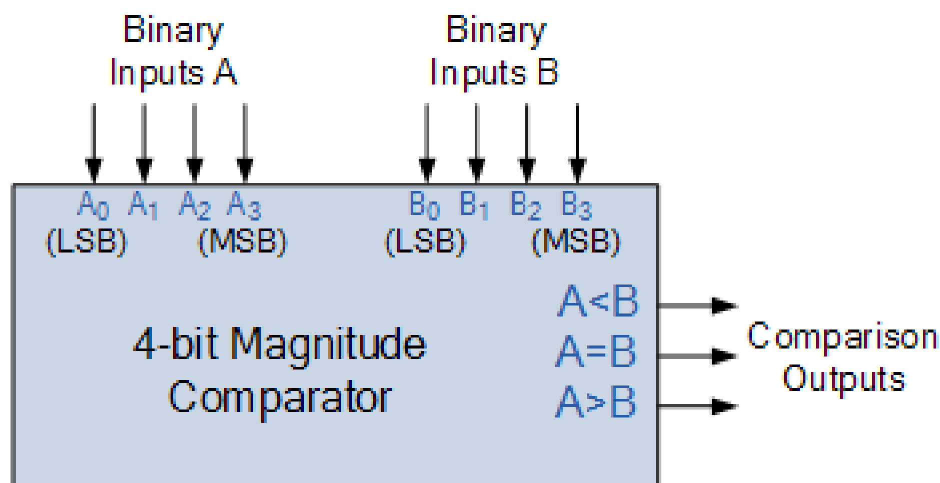
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two “0” or two “1”‘s as an output A = B is produced when they are both equal, either A = B = “0” or A = B = “1”. Secondly, the output condition for A = B resembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR function (equivalence) on each of the n-bits giving: $Q = A \oplus B$

Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the “magnitude” of these values, a logic “0” against a logic “1” which is where the term **Magnitude Comparator** comes from.

As well as comparing individual bits, we can design larger bit comparators by cascading together n of these and produce a n-bit comparator just as we did for the n-bit adder in the previous tutorial. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other.

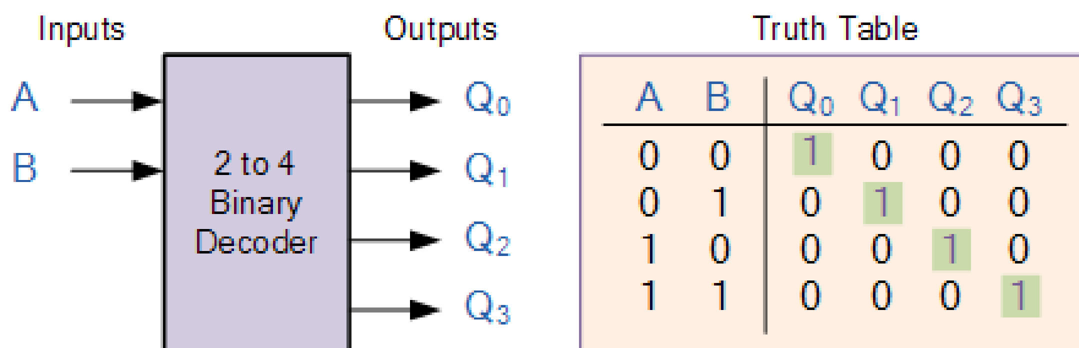
A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words (“nibbles”) are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

❖ 4-bit Magnitude Comparator

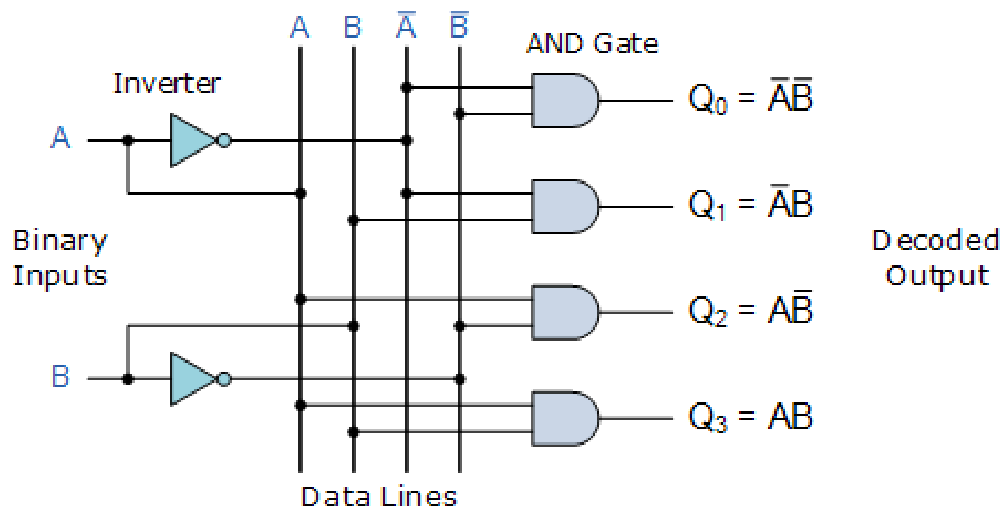


❖ Binary Decoder

Binary Decoder is another combinational logic circuit constructed from individual logic gates and is the exact opposite to that of an Encoder



A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



This simple example above of a 2-to-4 line binary decoder consists of an array of four AND gates. The 2 binary inputs labelled A and B are decoded into one of 4 outputs, hence the description of 2-to-4 binary decoder. Each output represents one of the miniterms of the 2 input variables, (each output = a miniterm).

The binary inputs A and B determine which output line from Q0 to Q3 is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0" so only one output can be active (HIGH) at any one time. Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "de-codes" the binary input.

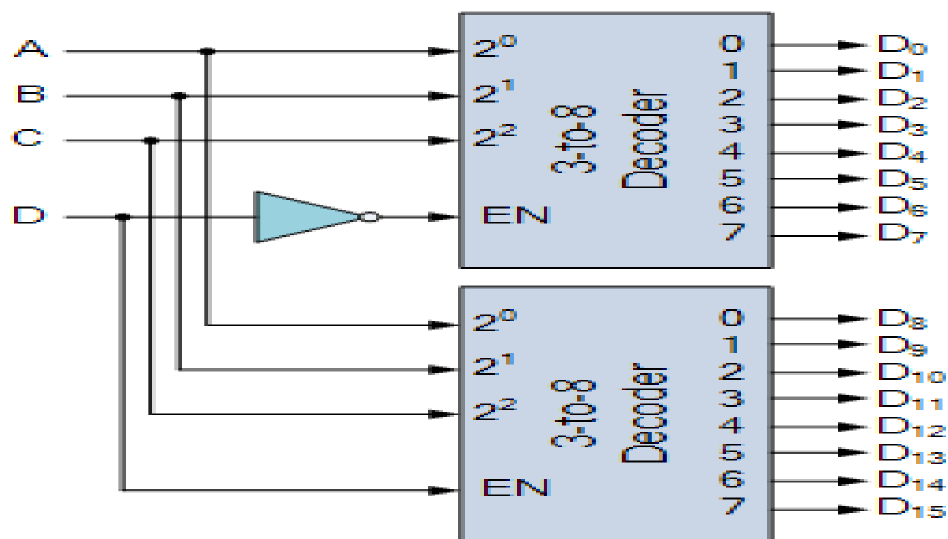
Some binary decoders have an additional input pin labelled "Enable" that controls the outputs from the device. This extra input allows the decoders outputs to be turned "ON" or "OFF" as required. These types of binary decoders are commonly used as "memory address decoders" in microprocessor memory applications.

We have seen that a 2-to-4 line binary decoder (TTL 74155) can be used for decoding any 2-bit binary code to provide four outputs, one for each possible input combination. However, sometimes it is required to have a **Binary Decoder** with a number of outputs greater than is available, so by adding more inputs, the decoder can potentially provide 2^n more outputs.

So for example, a decoder with 3 binary inputs ($n = 3$), would produce a 3-to-8 line decoder (TTL 74138) and 4 inputs ($n = 4$) would produce a 4-to-16 line decoder (TTL 74154) and so on. But a decoder can also have less than 2^noutputs such as the BCD to seven-segment decoder (TTL 7447) which has 4 inputs and only 7 active outputs to drive a display rather than the full 16 (2^4) outputs as you would expect.

Here a much larger 4 (3 data plus 1 enable) to 16 line binary decoder has been implemented using two smaller 3-to-8 decoders.

A 4-to-16 Binary Decoder Configuration

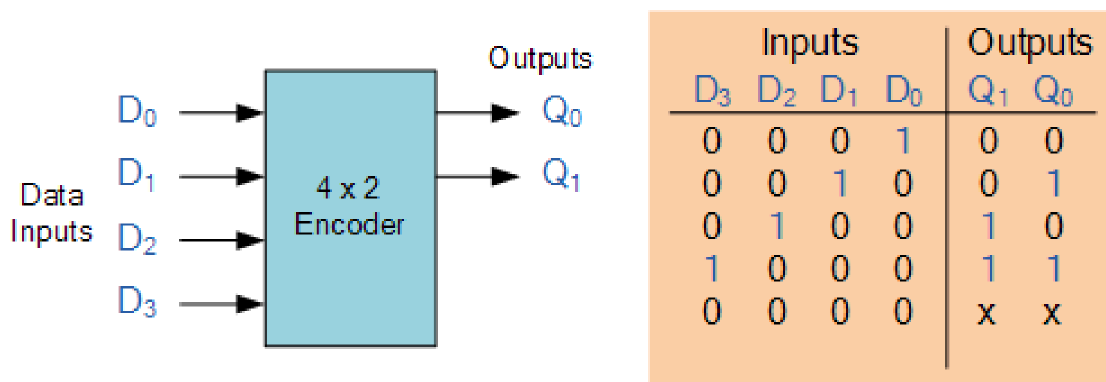


4-to-16 Line Decoder Implemented with two 3-to-8 Decoders

Inputs A, B, C are used to select which output on either decoder will be at logic "1" (HIGH) and input D is used with the enable input to select which encoder either the first or second will output the "1".

❖ Encoder

Encoders take all of their data inputs one at a time and converts them into an equivalent binary code at its output



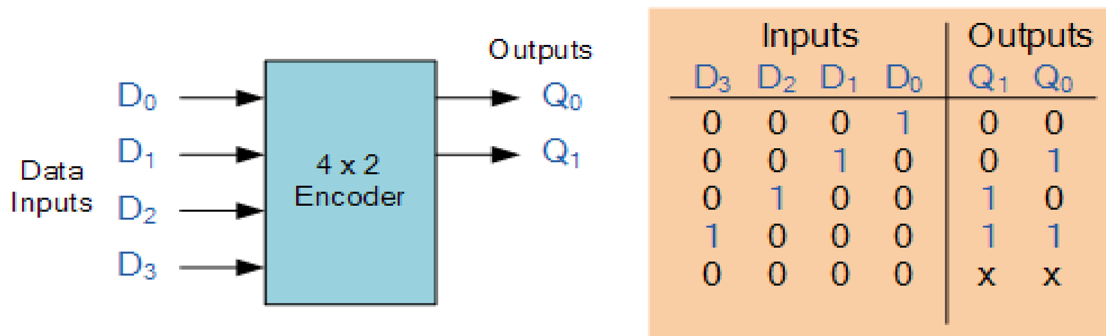
Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, **Digital Encoder** more commonly called a **Binary Encoder** takes ALL its data inputs one at a time and then converts them into a single encoded output. So we can say that a binary encoder, is a multi-input combinational logic circuit that converts the logic level "1" data at its inputs into an equivalent binary code at its output.

Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An "n-bit" binary encoder has 2^n input lines and n-bit output lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations.

The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to "1" and are available to encode either a decimal or hexadecimal input pattern to typically a binary or "B.C.D" (binary coded decimal) output code.

4-to-2 Bit Binary Encoder



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_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
0	0	0	0	x	x

One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs D_1 and D_2 HIGH at logic "1" both at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output binary number that is different to the actual input present. Also, an output code of all logic "0"s can be generated when all of its inputs are at "0" OR when input D_0 is equal to one.

One simple way to overcome this problem is to "Prioritise" the level of each input pin. So if there is more than one input at logic level "1" at the same time, the actual output code would only correspond to the input with the highest designated priority. Then this type of digital encoder is known commonly as a **Priority Encoder** or **P-encoder** for short.

❖ Priority Encoder

The **Priority Encoder** solves the problems mentioned above by allocating a priority level to each input. The *priority encoders* output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

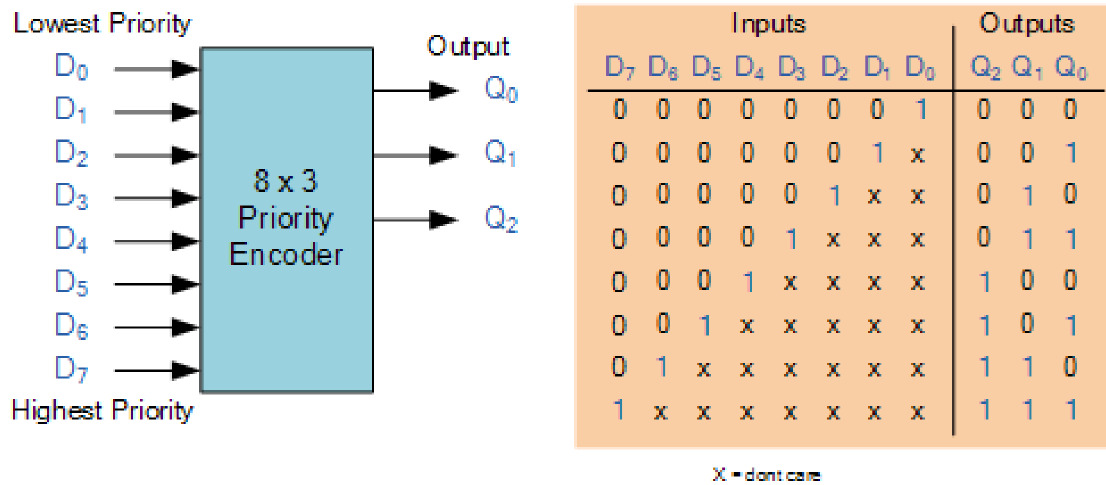
It's applications includes

- used to control interrupt requests by acting on the highest priority request

- to encode the output of a flash analog to digital converter

The priority encoder comes in many different forms with an example of an 8-input priority encoder along with its truth table shown below.

8-to-3 Bit Priority Encoder



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

Priority encoders are available in standard IC form and the TTL 74LS148 is an 8-to-3 bit priority encoder which has eight active LOW (logic “0”) inputs and provides a 3-bit code of the highest ranked input at its output.

Priority encoders output the highest order input first for example, if input lines “D2“, “D3” and “D5” are applied simultaneously the output code would be for input “D5” (“101”) as this has the highest order out of the 3 inputs. Once input “D5” had been removed the next highest output code would be for input “D3” (“011”), and so on.

The truth table for a 8-to-3 bit priority encoder is given as:

Digital Inputs								Binary Output		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	**1**	X	0	0	1
0	0	0	0	0	**1**	X	X	0	1	0

0	0	0	0	**1**	X	X	X	0	1	1
0	0	0	**1**	X	X	X	X	1	0	0
0	0	**1**	X	X	X	X	X	1	0	1
0	**1**	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

Where X equals "dont care", that is logic "0" or a logic "1".

From this truth table, the Boolean expression for the encoder above with data inputs D_0 to D_7 and outputs Q_0, Q_1, Q_2 is given as:

Output Q_0

$$Q_0 = \sum(1, 3, 5, 7)$$

$$Q_0 = \sum\left(\bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4\bar{D}_3\bar{D}_2 D_1 + \bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4 D_3 + \bar{D}_7\bar{D}_6 D_5 + D_7\right)$$

$$Q_0 = \sum\left(\bar{D}_6\bar{D}_4\bar{D}_2 D_1 + \bar{D}_6\bar{D}_4 D_3 + \bar{D}_6 D_5 + D_7\right)$$

$$Q_0 = \sum\left(\bar{D}_6\left(\bar{D}_4\bar{D}_2 D_1 + \bar{D}_4 D_3 + D_5\right) + D_7\right)$$

Output Q_1

$$Q_1 = \sum(2, 3, 6, 7)$$

$$Q_1 = \sum\left(\bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4\bar{D}_3 D_2 + \bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4 D_3 + \bar{D}_7 D_6 + D_7\right)$$

$$Q_1 = \sum\left(\bar{D}_5\bar{D}_4 D_2 + \bar{D}_5\bar{D}_4 D_3 + D_6 + D_7\right)$$

$$Q_1 = \sum\left(\bar{D}_5\bar{D}_4\left(D_2 + D_3\right) + D_6 + D_7\right)$$

Output Q_2

$$Q_2 = \sum(4, 5, 6, 7)$$

$$Q_2 = \sum\left(\bar{D}_7\bar{D}_6\bar{D}_5 D_4 + \bar{D}_7\bar{D}_6 D_5 + \bar{D}_7 D_6 + D_7\right)$$

$$Q_2 = \sum\left(D_4 + D_5 + D_6 + D_7\right)$$

Then the final Boolean expression for the priority encoder including the zero inputs is defined as:

Priority Encoder Output Expression

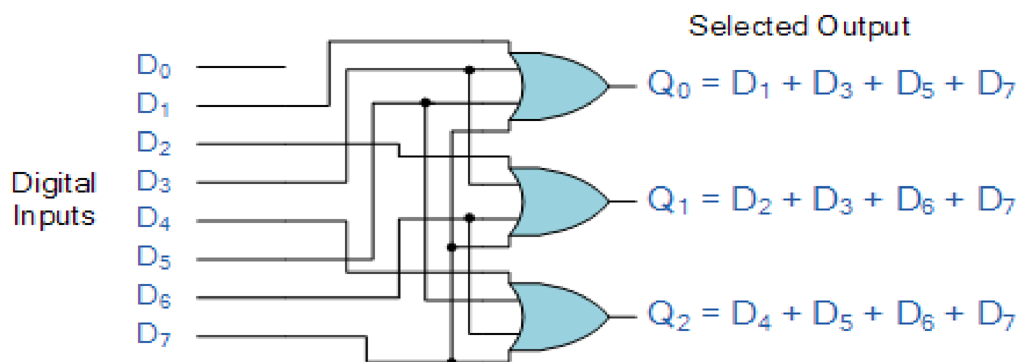
$$Q_0 = \sum\left(\overline{D}_6\left(\overline{D}_4\overline{D}_2D_1 + \overline{D}_4D_3 + D_5\right) + D_7\right)$$

$$Q_1 = \sum\left(\overline{D}_5\overline{D}_4\left(D_2 + D_3\right) + D_6 + D_7\right)$$

$$Q_2 = \sum\left(D_4 + D_5 + D_6 + D_7\right)$$

In practice these zero inputs would be ignored allowing the implementation of the final Boolean expression for the outputs of the 8-to-3 **priority encoder**. We can constructed a simple encoder from the expression above using individual OR gates as follows.

Digital Encoder using Logic Gates



4 to 2 priority encoder

A 4-to-2 priority encoder takes 4 input bits and produces 2 output bits. In this truth table, for all the non-explicitly defined input combinations (i.e. inputs containing 2, 3, or 4 high bits) the lower priority bits are shown as don't cares (X). Similarly when the inputs are 0000, the outputs are not valid and therefore they are XX.

Truth Table

I3	I2	I1	I0	O1	O0
0	0	0	0	X	X

0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

From the above truth table, we can obtain the full truth table required for our design.

Truth Table

I3	I2	I1	I0	O1	O0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1

1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

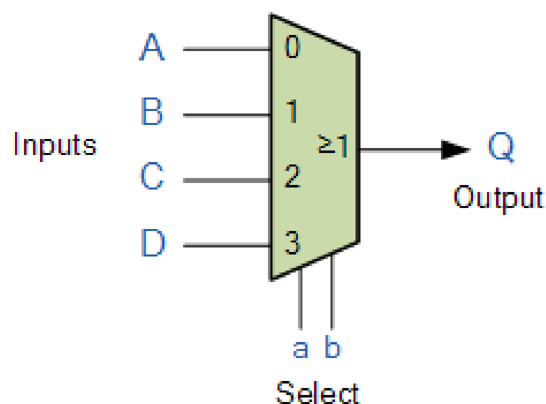
From this truth table, we use the Karnaugh Map to minimise the logic to the following boolean expressions:

- **O1 = I2 + I3**
- **O0 = ~I2 * I1 + I3**

Implementation of the 4 to 2 priority encoder using combinational logic circuits.

❖ The Multiplexer

The multiplexer is a combinational logic circuit designed to switch one of several input lines to a single common output line



Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a **Multiplexer**.

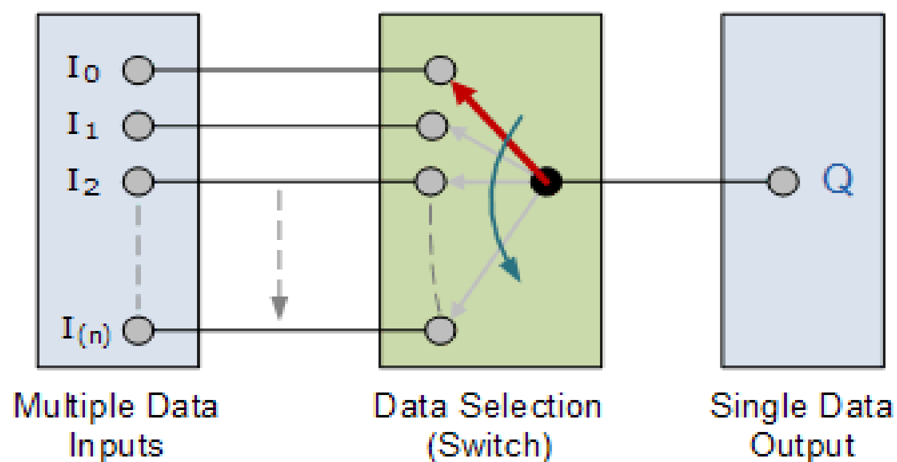
The *multiplexer*, shortened to “MUX” or “MPX”, is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very

fast acting multiple position rotary switches connecting or controlling multiple input lines called “channels” one at a time to the output.

Multiplexers, or MUX’s, can be either digital circuits made from high speed logic gates used to switch digital or binary data or they can be analogue types using transistors, MOSFET’s or relays to switch one of the voltage or current inputs through to a single output.

The most basic type of multiplexer device is that of a one-way rotary switch as shown.

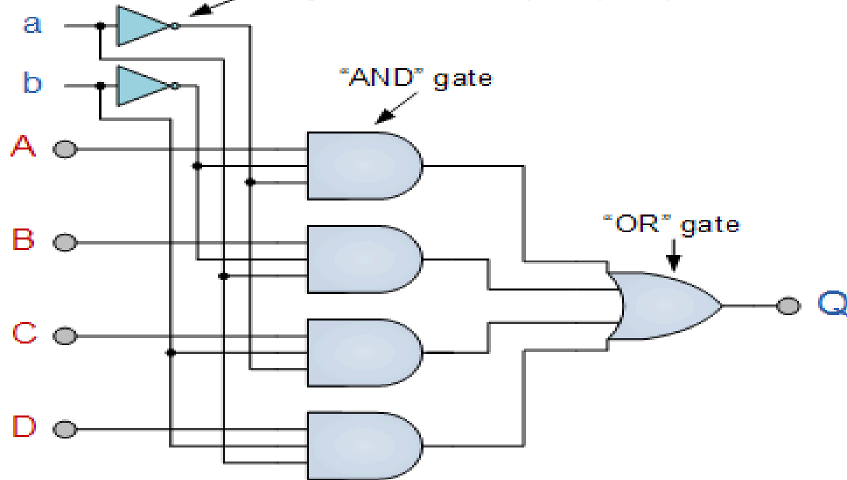
Basic Multiplexing Switch



Generally, the selection of each input line in a multiplexer is controlled by an additional set of inputs called *control lines* and according to the binary condition of these control inputs, either “HIGH” or “LOW” the appropriate data input is connected directly to the output. Normally, a multiplexer has an even number of 2^n data input lines and a number of “control” inputs that correspond with the number of data inputs.

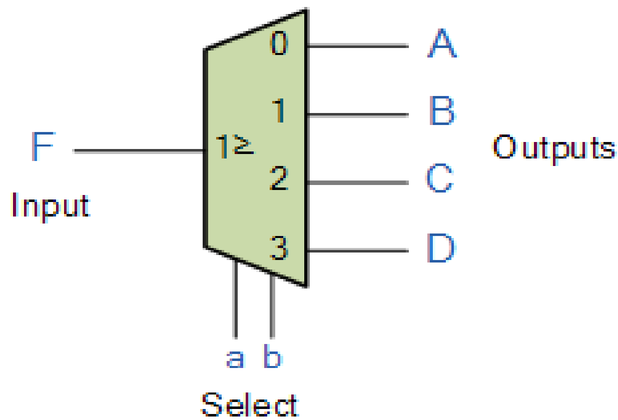
Note that multiplexers are different in operation to *Encoders*. Encoders are able to switch an n-bit input pattern to multiple output lines that represent the binary coded (BCD) output equivalent of the active input.

4 Channel Multiplexer using logic gates



❖ The Demultiplexer

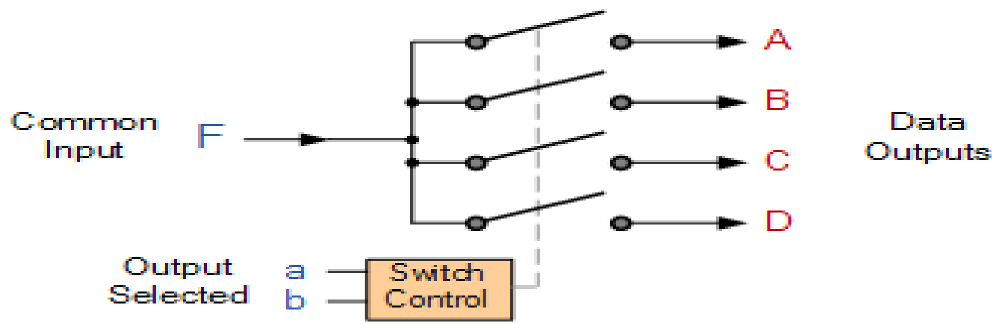
The demultiplexer is a combinational logic circuit designed to switch one common input line to one of several seperate output line



The data distributor, known more commonly as a **Demultiplexer** or "Demux" for short, is the exact opposite of the Multiplexer we saw in the previous tutorial.

The *demultiplexer* takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The **demultiplexer**converts a serial data signal at the input to a parallel data at its output lines as shown below.

1-to-4 Channel De-multiplexer



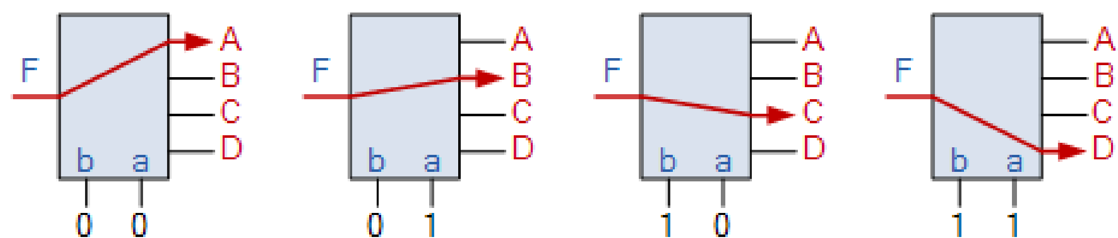
S1	S0	A	B	C	D
0	0	F	O	O	O
0	1	O	F	O	O
1	0	O	O	F	O
1	1	O	O	O	F

The Boolean expression for this 1-to-4 **Demultiplexer** above with outputs A to D and data select lines a, b is given as:

F =

The function of the **Demultiplexer** is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" as shown.

Demultiplexer Output Line Selection



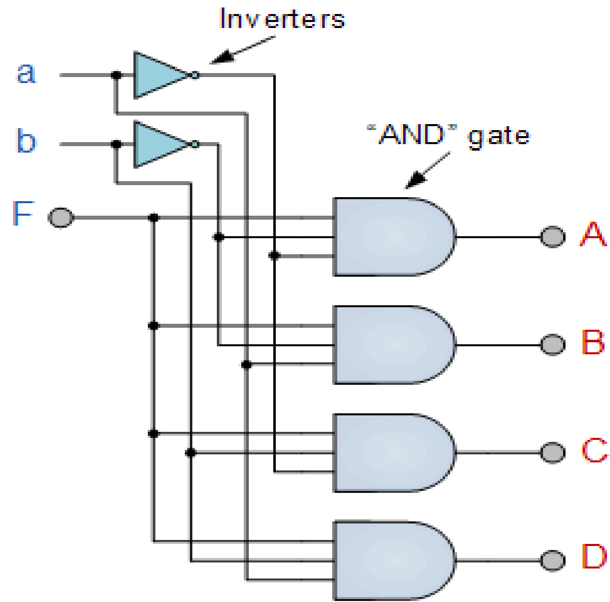
As with the previous multiplexer circuit, adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs.

Some standard demultiplexer IC´s also have an additional "enable output" pin which disables or prevents the input from being passed to the selected output. Also some have latches built into their outputs to maintain the output logic level after the address inputs have been changed.

However, in standard decoder type circuits the address input will determine which single data output will have the same value as the data input with all other data outputs having the value of logic "0".

The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown.

4 Channel Demultiplexer using Logic Gates



❖ CODE CONVERTERS

Converting Binary to Gray Code –

B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				

1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

b3,b2 \ b1,b0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

b3,b2 \ b1,b0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

b3,b2 \ b1,b0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

b3,b2 \ b1,b0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Converting Gray to Binary Code –

G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
				0	0	0	0
				0	0	0	1
				0	0	1	0
				0	0	1	1
				0	1	0	0
				0	1	0	1
				0	1	1	0
				0	1	1	1
				1	0	0	0
				1	0	0	1
				1	0	1	0

				1	0	1	1
				1	1	0	0
				1	1	0	1
				1	1	1	0
				1	1	1	1

g3,g2 \ g1,g0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

g3,g2 \ g1,g0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

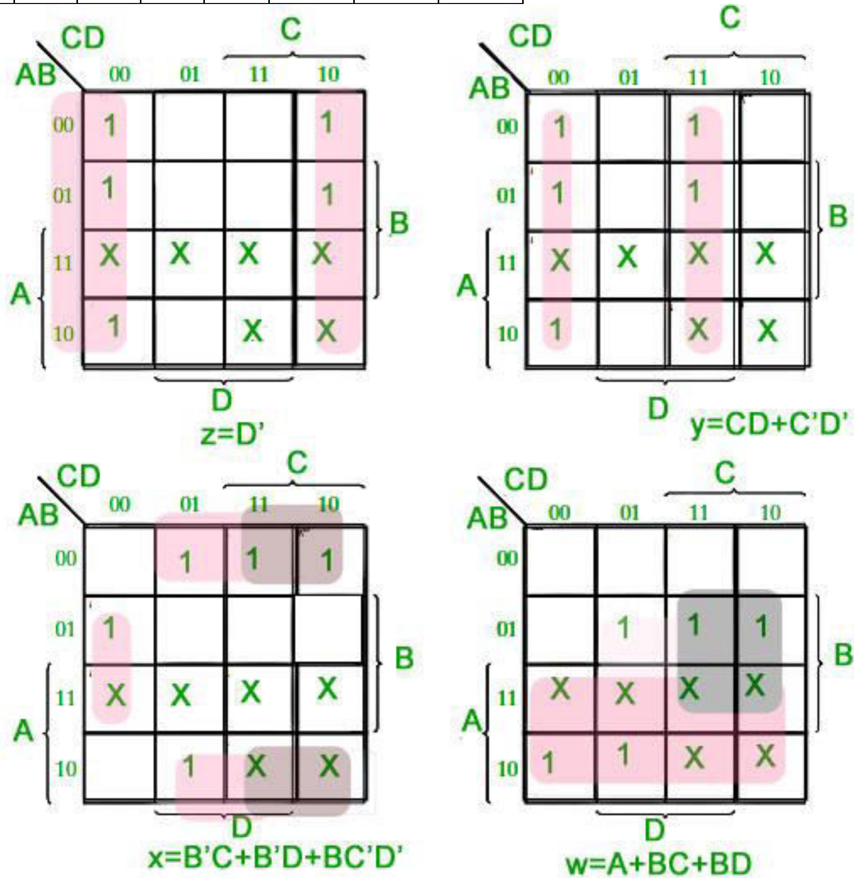
g3,g2 \ g1,g0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

g3,g2 \ g1,g0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Converting BCD(8421) to Excess-3 –

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



Converting Excess-3 to BCD(8421) –

W	X	Y	Z	A	B	C	D
0	0	0	0	X	X	X	X
0	0	0	1	X	X	X	X
0	0	1	0	X	X	X	X
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0

1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

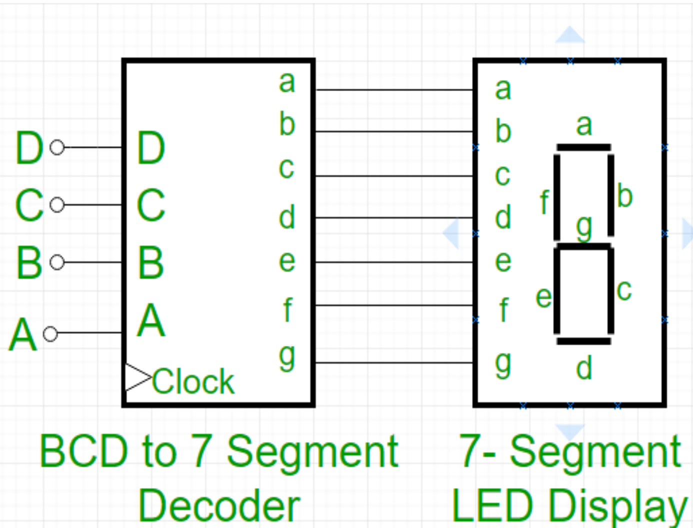
wx \ yz	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

wx \ yz	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

wx \ yz	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

wx \ yz	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

❖ BCD TO 7-SEGMENT DECODER



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

F(ABCD)= ¬B¬D + C + BD + A

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

F(ABCD)= ¬C + D + B

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1					1	0	0	1	1
0	1					1	1	0	1	1
0	1					1	1	1	1	1
0	1					1	0	0	0	0
1	0					1	1	1	1	1
1	0					1	1	0	1	1

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

F(ABCD)= ¬B + ¬C¬D + CD

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

F(ABCD)= ¬C¬D + B¬ + B¬D + A

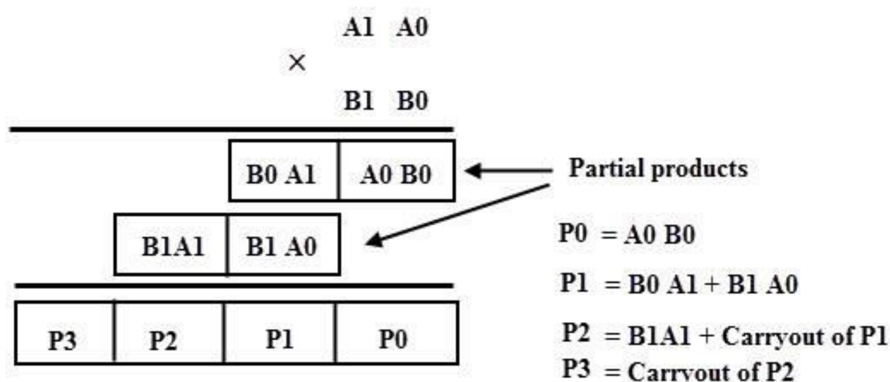
AB\CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

F(ABCD)= ¬BC + B¬C + A + B¬D

CIRCUIT DIAGRAM

❖ Binary Multiplier Circuit

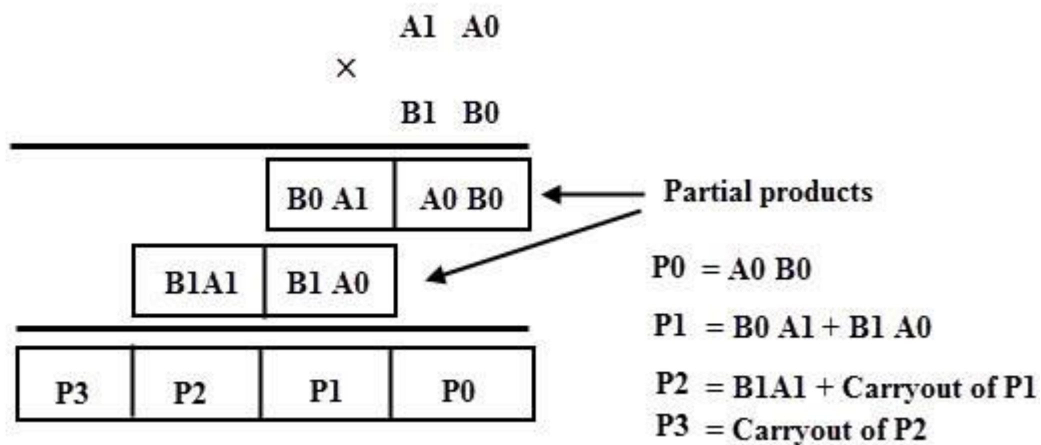
Let us consider two unsigned 2 bit binary numbers A and B to generalize the multiplication process. The multiplicand A is equal to A1A0 and the multiplier B is equal to B1B0. The figure below shows the multiplication process of two 2 bit binary numbers.



This process involves the multiplication of two digits and the addition of digits with or without carry. After the multiplication of the each bit to the multiplicand, partial products are generated, and then these products are added to produce the total sum which represents the binary multiplication value.

This multiplication is implemented by combinational circuit such that the multiplication is performed with AND gates whereas the addition is carried out

by using half adders as shown in figure.

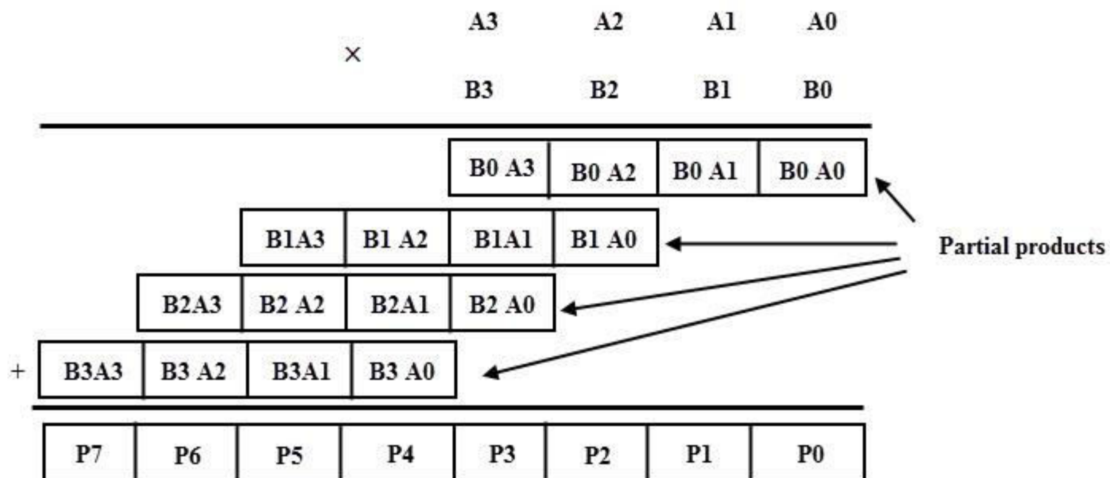


The first partial product is obtained by the AND gate which is nothing but a least significant bit of the multiplication result. Since the second partial product is shifted to the left position, the first partial second term and second partial product first term is added by half adder and produce the sum output along with the carry out.

This carry out is added at the next half adder as an input as shown in figure. Likewise, it produces the multiplication result of two binary numbers by using the simple circuit configuration. The multiplication of the two 2 bit number results a 4-bit binary number.

Let us consider two unsigned 4 bit numbers multiplication in which the multiplicand, A is equal to A3A2 A1A0 and the multiplier B is equal to B3B2B1B0. The partial products are produced depending on each multiplier bit multiplied by the multiplicand.

Each partial product consists of four product terms and these are shifted to the left relative to the previous partial product as shown in figure. All these partial products are added to produce the 8 bit product.

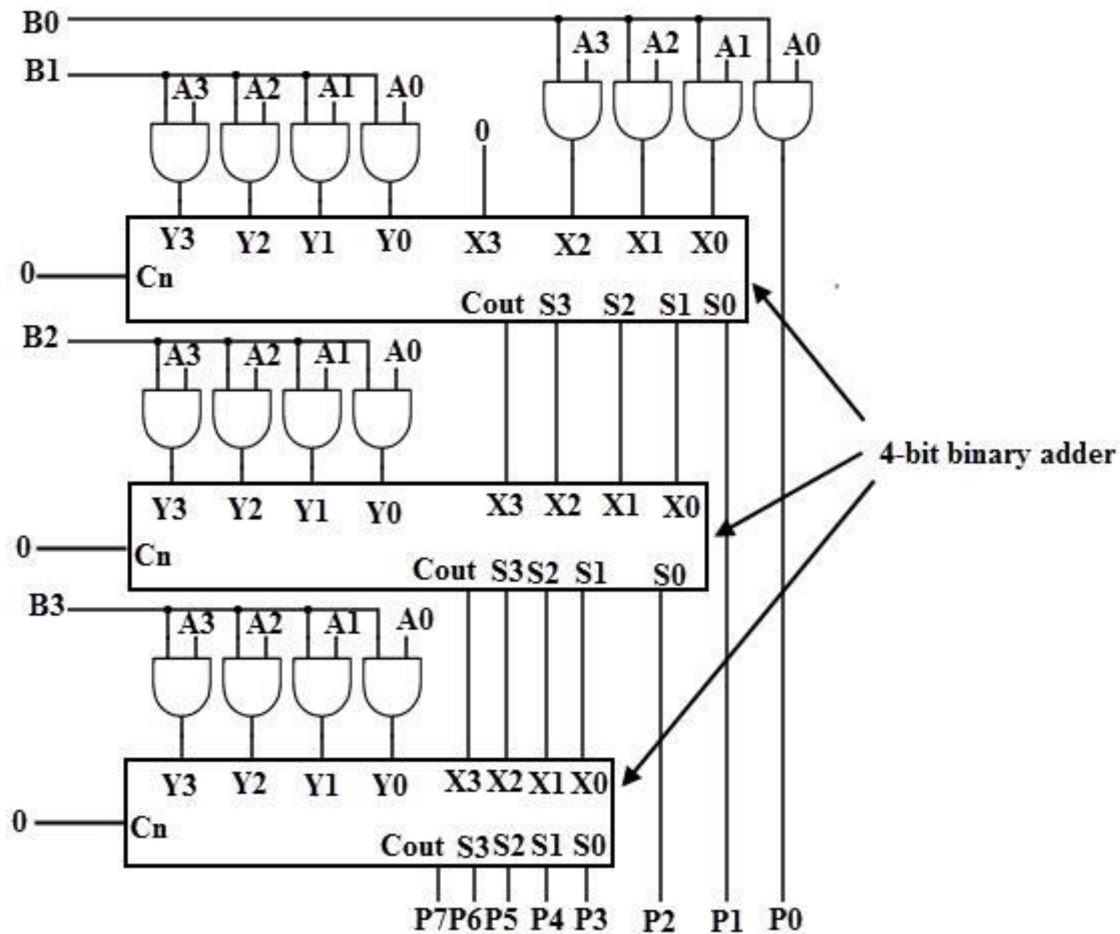


The logic circuit for the 4× 4 binary multiplication can be implemented by using three binary full adders along with AND gates.

In the above operation the first partial product is obtained by multiplying B0 with A3A2 A1A0, the second partial product is formed by multiplying B1 with A3A2 A1A0, likewise for 3rd and 4th partial products. So these partial products can be implemented with AND gates as shown in figure.

These partial products are then added by using 4 bit parallel adder. The three most significant bits of first partial product with carry (considered as zero) are added with second partial term in the first full adder.

Then the result is added to the next partial product with carry out and it goes on till the final partial product, finally it produces 8 bit sum which indicates the multiplication value of the two binary numbers.



❖ Decimal Adder / BCD Adder:

Decimal Adder – The digital systems handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 0 0 0 0 to **1 0 0 1,** i.e. each BCD digit is represented as a 4-bit binary number. When we write BCD number say 526, it can be represented as

5	2	6
↓	↓	↓
0 1 0 1	0 0 1 0	0 1 1 0

Here, we should note that BCD cannot be greater than 9.

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

Sum Equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.

6	0	1	1	0	←	BCD for 6
+ 3	0	0	1	1	←	BCD for 3
9	1	0	0	1	←	BCD for 9

The addition is carried out as in normal binary addition and the sum is 1 0 0 1,

which is BCD code for 9.

Sum greater than 9 with carry 0

The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown below

6		0	1	1	0	←	BCD for 6
+ 8		1	0	0	0	←	BCD for 8
14		1	1	1	0	←	Invalid BCD number
	+	0	1	1	0	←	Add 6 for correction
0 0 0 1		0	1	0	0	←	BCD for 14
1			4				

After addition of 6 carry is produced into the second decimal position. **Sum equals 9 or less with carry 1**

Let us consider addition of 8 and 9 in BCD

8					1	0	0	0	←	BCD for 8
+ 9					1	0	0	1	←	BCD for 9
17	0	0	0	1	0	0	0	1	←	Incorrect BCD result

In this, case, result (0001 0001) is valid BCD number, but it is incorrect. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown below

8						1	0	0	0	←	BCD for 8
+ 9						1	0	0	1	←	BCD for 9
17		0	0	0	1	0	0	0	1	←	Incorrect BCD result
	+	0	0	0	0	0	1	1	0	←	Add 6 for correction
		0	0	0	1	0	1	1	1	←	BCD for 17

Going through these three cases of BCD addition we can summarise the BCD addition procedure as follows :

1. Add two BCD numbers using ordina7 binary addition.
2. If four-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.
3. If the four-bit sum is greater than 9 or if a carry is generated from **the** four-bit sum, the sum is invalid.
4. To correct the invalid sum, add 0110_2 to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.
5. Thus to implement BCD adder we require :
6. 4-bit binary adder for initial addition
7. Logic circuit to detect sum greater than 9 and
8. One more 4-bit adder to add 01102 in the sum if sum is greater than 9 or carry is 1.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given truth table.

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	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
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 3.10

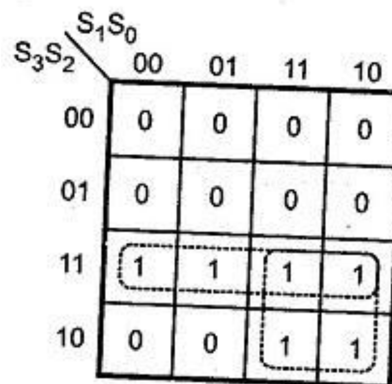


Fig. 3.31

$$Y = S_3S_2 + S_3S_1$$

With this design information we can draw the block diagram of BCD adder, as shown in the Fig. 3.32.

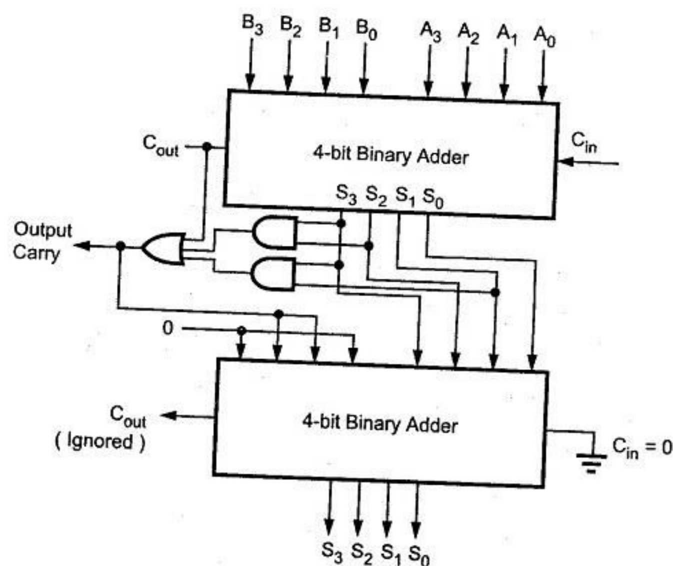


Fig. 3.32 Block diagram of BCD adder

As shown in the Fig. 3.32 , the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum. When the output carry is equal to zero (i.e. when sum ≤ 9 and C_{out} = 0) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum > 9 or C_{out} = 1), binary0110 is added to the binary sum through the bottom 4-bit binary adder. The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.