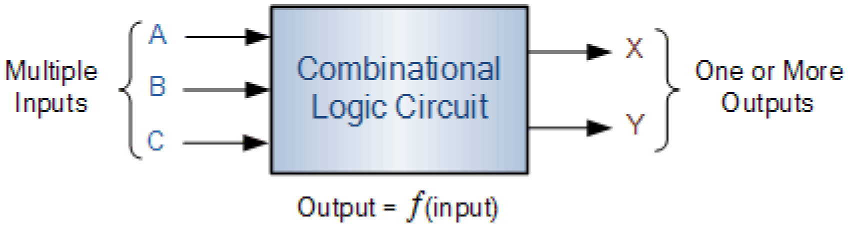
Combinational Logic Circuits

Unlike Sequential Logic Circuits whose outputs are dependant on both their present inputs and their previous output state giving them some form of Memory, the outputs of Combinational Logic Circuits are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a Combinational Logic Circuit, the output is dependant at all times on the combination of its inputs. So if one of its inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have "no memory", "timing" or "feedback loops" within their design.

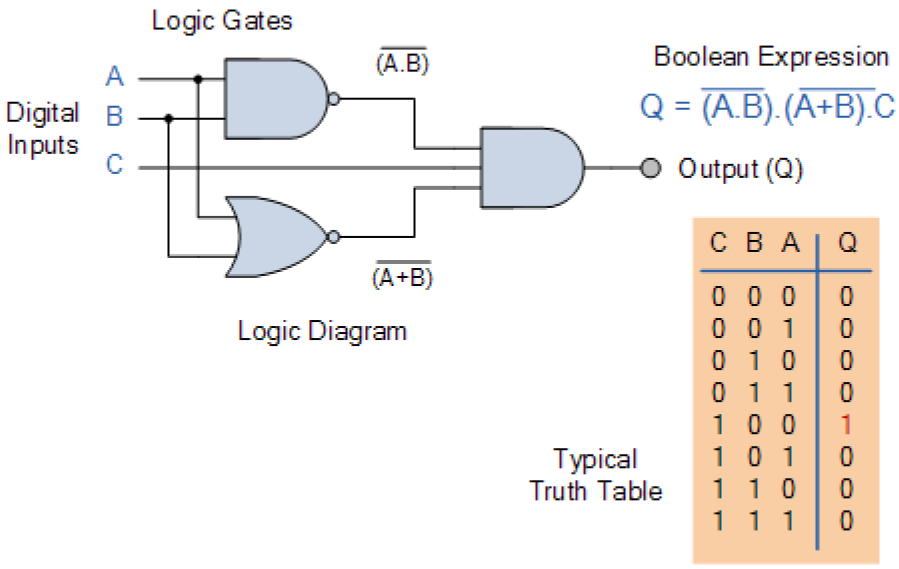


Combinational Logic Circuits are made up from basic logic NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of Combinational Logic Circuits circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.The three main ways of specifying the function of a combinational logic circuit are:

1. **Boolean Algebra** – This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic "1" output.

2. **Truth Table** – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.

3. Logic Diagram – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit and all three of these logic circuit representations are shown below.



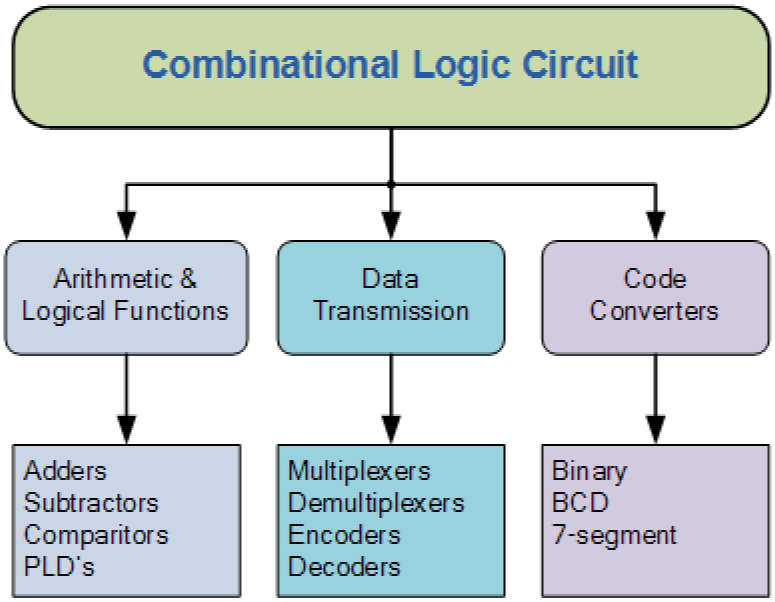
$Q = \overline{(A.B)}.\overline{(A+B)}.C$

C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

As combinational logic circuits are made up from individual logic gates only, they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application include Multiplexers, De-multiplexers, Encoders, Decoders, Full and Half Adders etc.

Classification of Combinational Logic

One of the most common uses of combinational logic is in Multiplexer and De-multiplexer type circuits. Here, multiple inputs or outputs are connected to a common signal line and logic gates are used to decode an address to select a single data input or output switch. A multiplexer consist of two separate components, a logic decoder and some solid state switches, but before we can discuss multiplexers, decoders and de-multiplexers in more detail we first need to understand how these devices use these "solid state switches" in their design.



Solid State Switches

Standard TTL logic devices made up from Transistors can only pass signal currents in one direction only making them “uni-directional” devices and poor imitations of conventional electro-mechanical switches or relays. However, some CMOS switching devices made up from FET’s act as near perfect “bi-directional” switches making them ideal for use as solid state switches. Solid state switches come in a variety of different types and ratings, and there are many different applications for using solid state switches. They can basically be sub-divided into 3 different main groups for switching applications and in this combinational logic section we will only look at the Analogue type of switch but the principal is the same for all types including digital.

Solid State Switch Applications

- Analogue Switches – Used in Data Switching and Communications, Video and Audio Signal Switching, Instrumentation and Process Control Circuits …etc.

- Digital Switches – High Speed Data Transmission, Switching and Signal Routing, Ethernet, LAN’s, USB and Serial Transmissions …etc.

- Power Switches – Power Supplies and General “Standby Power” Switching Applications, Switching of Larger Voltages and Currents …etc.

Combinational Logic Summary

To summarize, Combinational Logic Circuits consist of inputs, two or more basic logic gates and outputs. The logic gates are combined in such a way that the output state depends entirely on the input states. Combinational logic circuits have “no memory”, “timing” or “feedback loops”, there operation is instantaneous. A combinational logic circuit performs an operation assigned logically by a Boolean expression or truth table.Examples of common Combinational Logic Circuits include: half adders, full adders, multiplexers, demultiplexers, encoders and decoders.

The Binary Adder

Another common and very useful combinational logic circuit which can be constructed using just a few basic logic gates and adds together binary numbers is the Binary Adder circuit. The Binary Adder is made up from standard AND and Ex-OR gates and allow us to “add” together single bit binary numbers, a and b to produce two outputs called the addition and a CARRY called the Carry-out, (C_{out}) bit. One of the main uses for the Binary Adder is in arithmetic and counting circuits.

Consider the addition of two denary (base 10) numbers below.

123	A	(Augend)
+ 789	B	(Addend)
912	SUM	

Each column is added together starting from the right hand side and each digit has a weighted value depending upon its position in the columns. As each column is added together a carry is generated if the result is greater or equal to ten, the base number. This carry is then added to the result of the addition of the next column to the left and so on, simple school math’s addition. The adding of binary numbers is basically the same as that of adding decimal numbers but this time a carry is only generated when the result in any column is greater or equal to “2”, the base number of binary.

Binary Addition

Binary Addition follows the same basic rules as for the denary addition above except in binary there are only two digits and the largest digit is “1”, so any “SUM” greater than 1 will result in a “CARRY”. This carry 1 is passed over to the next column for addition and so on. Consider the single bit addition below.

0 0 1 1

+ 0 + 1 + 0 + 1

0 1 1 10

The single bits are added together and “0 + 0”, “0 + 1”, or “1 + 0” results in a sum of “0” or “1” until you get to “1 + 1” then the sum is equal to “2”, (a zero plus a carry). For a simple 1-bit addition problem like this, the resulting carry bit could be ignored which would result in an output truth table resembling that of an Ex-OR Gate as seen in the Logic Gates section and whose result is the sum of the two bits but without the carry. An Ex-OR gate only produces an output “1” when either input is at logic “1”, but not both. However, all microprocessors and electronic calculators require the carry bit to correctly calculate the equations so we need to rewrite them to include 2 bits of output data as shown below.

00 00 01 01

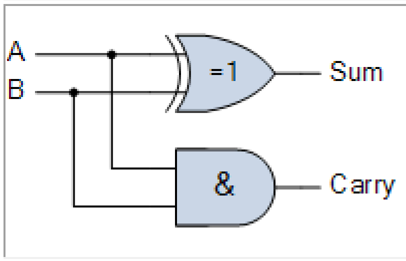
+ 00 + 01 + 00 + 01

00 01 01 10

From the above equations we know that an Ex-OR gate will only produce an output “1” when “EITHER” input is at logic “1”, so we need an additional output to produce a carry output, “1” when “BOTH” inputs “A” and “B” are at logic “1” and a standard AND Gate fits the bill nicely. By combining the Ex-OR gate with the AND gate results in a simple digital binary adder circuit known commonly as the “Half Adder” circuit.

The Half Adder Circuit

1-bit Adder with Carry-Out

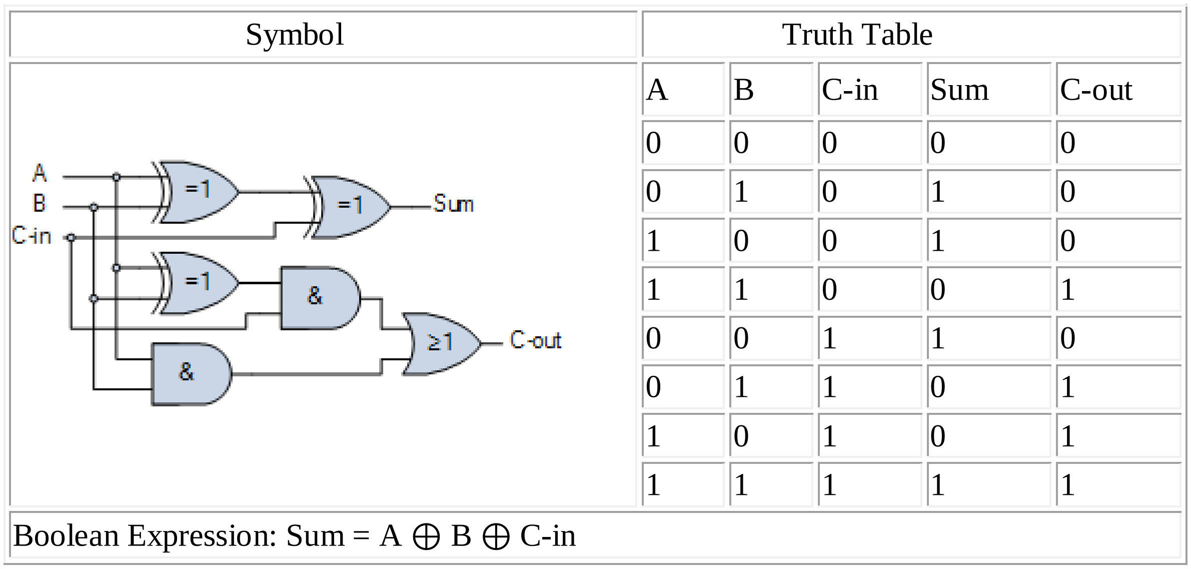
<table>
<tr><th>Symbol</th><th colspan="4">Truth Table</th></tr>
<tr><td rowspan="5"> </td><td>A</td><td>B</td><td>SUM</td><td>CARRY</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td colspan="5">Boolean Expression: Sum = $A \oplus B$ Carry = $A . B$</td></tr>
</table>

From the truth table we can see that the SUM (S) output is the result of the Ex-OR gate and the Carry-out (Cout) is the result of the AND gate. 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. For example, suppose we want to add together two 8-bit bytes of data, any resulting carry bit would need to be able to “ripple” or move across the bit patterns starting from the least significant bit (LSB). The most complicated operation the half adder can do is “1 + 1” but as the half adder has no carry input the resultant added value would be incorrect. One simple way to overcome this problem is to use a Full Adder type binary adder circuit.

The Full Adder Circuit

The main difference between the Full Adder and the previous seen Half Adder is that a full adder has three inputs, the same two single bit binary inputs A and B as before plus an additional Carry-In (C-in) input as shown below.

Full Adder with Carry-In

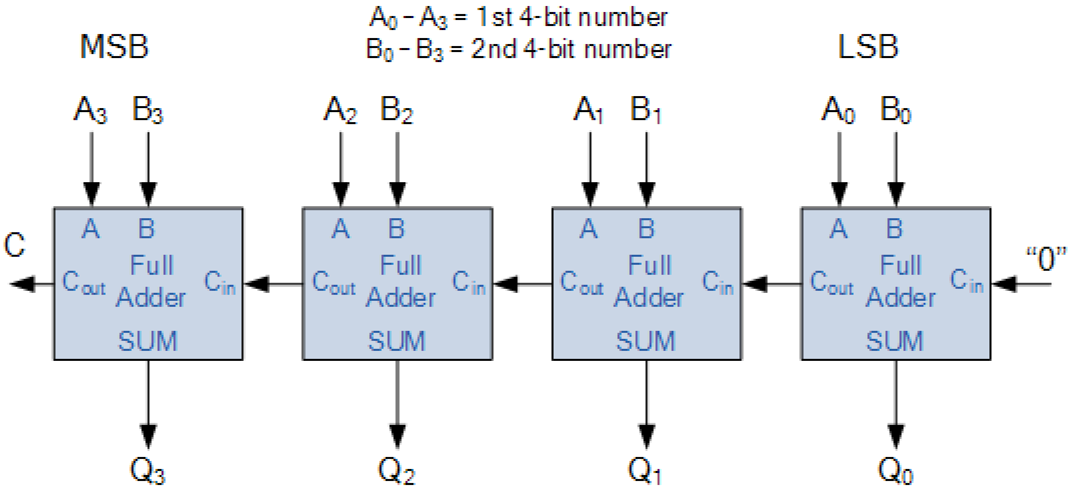

A
B
C-in
=1
=1
=1
Sum
&
&
≥1
C-out

Symbol	Truth Table				
	A	B	C-in	Sum	C-out
	0	0	0	0	0
	0	1	0	1	0
	1	0	0	1	0
	1	1	0	0	1
	0	0	1	1	0
	0	1	1	0	1
	1	0	1	0	1
	1	1	1	1	1
Boolean Expression: Sum = A ⊕ B ⊕ C-in					

The 1-bit Full Adder circuit above is basically two half adders connected together and consists of three Ex-OR gates, two AND gates and an OR gate, six logic gates in total. The truth table for the full adder includes an additional column to take into account the Carry-in input as well as the summed output and carry-output. 4-bit full adder circuits are available as standard IC packages in the form of the TTL 74LS83 or the 74LS283 which can add together two 4-bit binary numbers and generate a SUM and a CARRY output. But what if we wanted to add together two n-bit numbers, then n 1-bit full adders need to be connected together to produce what is known as the Ripple Carry Adder.

Ripple Carry Adder

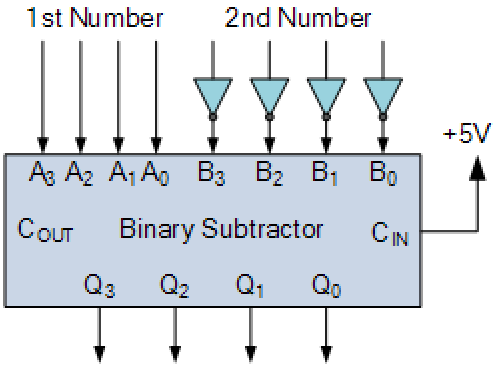
The Ripple Carry Binary Adder is simply n, full adders cascaded together with each full adder represents a single weighted column in the long addition with the carry signals producing a “ripple” effect through the binary adder from right to left. For example, suppose we want to “add” together two 4-bit numbers, the two outputs of the first full adder will provide the first place digit sum 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.



Major drawback of “cascading” together 1-bit binary adders to add large binary numbers is that if inputs A and B change, the sum at its output will not be valid until any carry-input has “rippled” through every full adder in the chain. Consequently, there will be a finite delay before the output of a adder responds to a change in its inputs resulting in the accumulated delay, especially in large multi-bit binary adders, becoming prohibitively large. This unwanted delay time is called Propagation delay. Also another problem called “overflow” occurs when an n-bit adder adds two numbers together whose sum is greater than or equal to 2n. One solution is to generate the carry-input signals directly from the A and B inputs rather than using the ripple arrangement above. This then produces another type of binary adder circuit called a Carry Look Ahead Binary Adder were the speed of the parallel adder can be greatly improved using carry-look ahead logic.

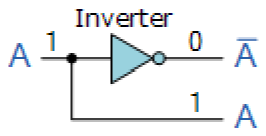
4-bit Binary Subtractor

Now that we know how to "ADD" together two 4-bit binary numbers how would we subtract two 4-bit binary numbers, for example, A – B using the circuit above. The answer is to use 2's-complement notation on all the bits in B must be complemented (inverted) and an extra one added using the carry-input. This can be achieved by inverting each B input bit to the binary subtractor by using an inverter or NOT-gate on each input. Also, in the above circuit for the 4-bit binary adder, to perform addition the first carry-in input is held LOW at logic "0". But for the circuit to perform the mathematical condition of subtraction this input pin needs to be held HIGH at logic "1". With this in mind a ripple carry adder can with a small modification be used to perform half subtraction, full subtraction and/or comparison. There are a number of 4-bit full-adder ICs available such as the 74LS283 and CD4008. which will add two 4-bit binary number and provide an additional input carry bit, as well as an output carry bit, so you can cascade them together to produce 8-bit, 12-bit, 16-bit, etc. adders.



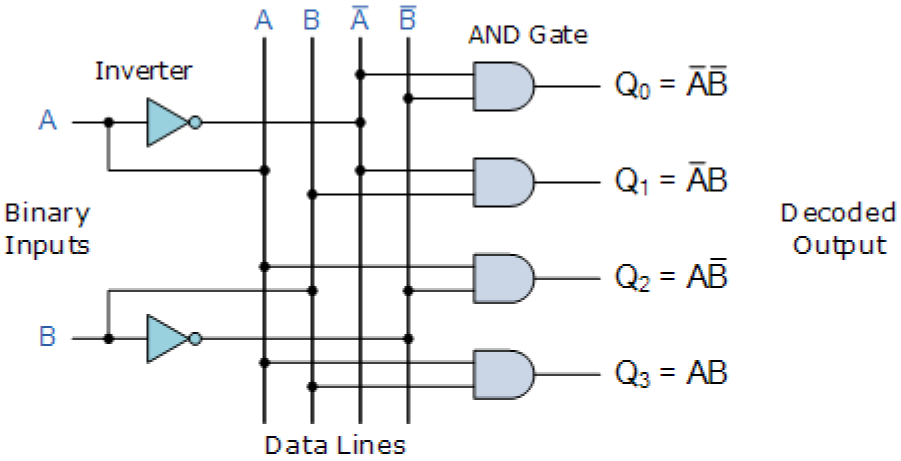
Binary Decoder

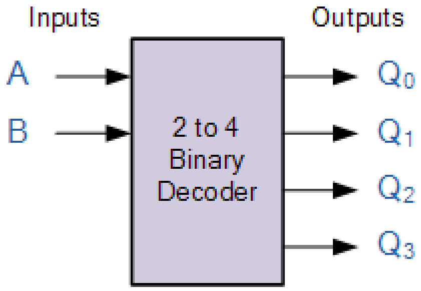
The Binary Decoder is another combinational logic circuit constructed from individual logic gates and is the exact opposite to that of an "Encoder" we looked at in the last tutorial. The name "Decoder" means to translate or decode coded information from one format into another, so a digital decoder transforms a set of digital input signals into an equivalent decimal code at its output. Binary Decoders are another type of Digital Logic device that has inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, so a decoder that has a set of two or more bits will be defined as having an n-bit code, and therefore it will be possible to represent 2n possible values. Thus, a decoder generally decodes a binary value into a non-binary one by setting exactly one of its n outputs to logic "1". If a binary decoder receives n inputs (usually grouped as a single Binary or Boolean number) it activates one and only one of its 2n outputs based on that input with all other outputs deactivated.



For example, an inverter (NOT-gate) can be classed as a 1-to-2 binary decoder as 1-input and 2-outputs (21) is possible because with an input A it can produce two outputs A and A (not-A) as shown. Then we can say that a standard combinational logic decoder is an n-to-m decoder, where m ≤ 2n, and whose output, Q is dependent only on its present input states. In other words, a binary decoder looks at its current inputs, determines which binary code or binary number is present at its inputs and selects the appropriate output that corresponds to that binary input.

A Binary Decoder converts coded inputs into coded outputs, where the input and output codes are different and decoders are available to “decode” either a Binary or BCD (8421 code) input pattern to typically a Decimal output code. Commonly available BCD-to-Decimal decoders include the TTL 7442 or the CMOS 4028. Generally a decoders output code normally has more bits than its input code and practical “binary decoder” circuits include, 2-to-4, 3-to-8 and 4-to-16 line configurations. An example of a 2-to-4 line decoder along with its truth table is given below.



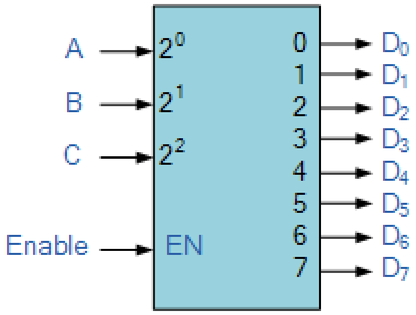


Truth Table

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

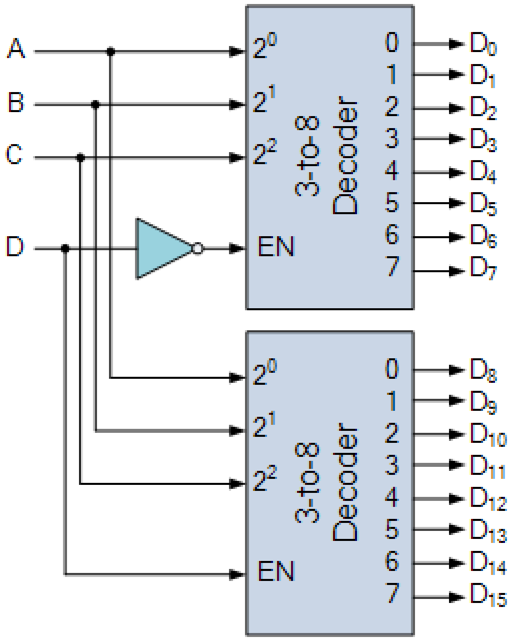
This is a simple example of 2-to-4 line binary decoder consisting of an array of four AND gates. The 2 binary inputs labelled A and B are decoded into one of 4 outputs, hence the description of 2-to-4 binary decoder. Each output represents one of the miniterms of the 2 input variables, (each output = a minterm). 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 can say that a binary decoder is a demultiplexer with an additional data line that is used to enable the decoder. An alternative way of looking at the decoder circuit is to regard inputs A, B and C as address signals. Each combination of A, B or C defines a unique memory address.



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.

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 2n outputs 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 (24) 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.



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”. However, there is a limit to the number of inputs that can be used for one particular decoder, because as n increases, the number of AND gates required to produce an output also becomes larger resulting in the fan-out of the gates used to drive them becoming large. This type of active-“HIGH” decoder can be implemented using just Inverters, (NOT Gates) and AND gates. It is convenient to use an AND gate as the basic decoding element for the output because it produces a “HIGH” or logic “1” output only when all of its inputs are logic “1”. But some binary decoders are constructed using NAND gates instead of AND gates for their decoded output, since NAND gates are cheaper to produce than AND’s as they require fewer transistors to implement within their design. The use of NAND gates as the decoding element, results in an active-“LOW” output while the rest will be “HIGH”. As a NAND gate produces the AND operation with an inverted output, the NAND decoder looks like this with its inverted truth table.

The Digital Comparator

Another common and very useful combinational logic circuit is that of the Digital Comparator circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs. For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean Algebra. There are two main types of Digital Comparator available and these are

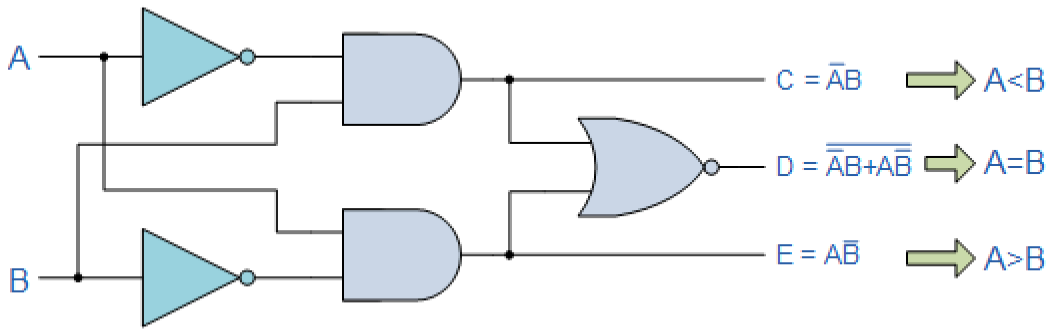
1. **Identity Comparator** – an Identity Comparator is a digital comparator that has only one output terminal for when A = B either "HIGH" A = B = 1 or "LOW" A = B = 0

2. **Magnitude Comparator** – a Magnitude Comparator is a type of digital comparator that has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

The purpose of a Digital Comparator is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, …. An, etc) against that of a constant or unknown value such as B (B1, B2, B3, …. Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

A>B, A=B , A<B

Which means: A is greater than B, A is equal to B, and A is less than B. This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

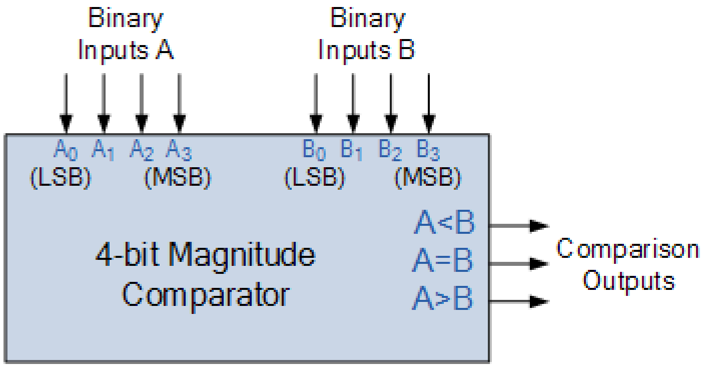


Truth Table

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.

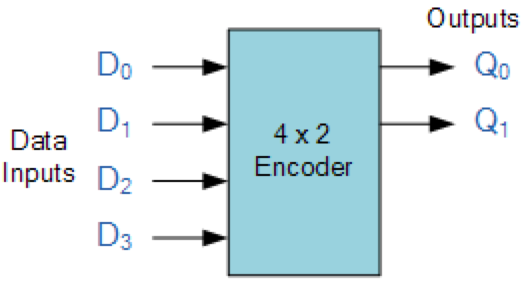


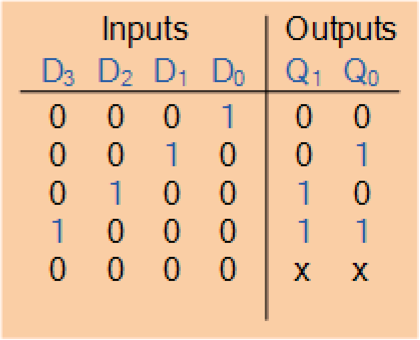
Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

The Digital Encoder

Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, a 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 2n 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



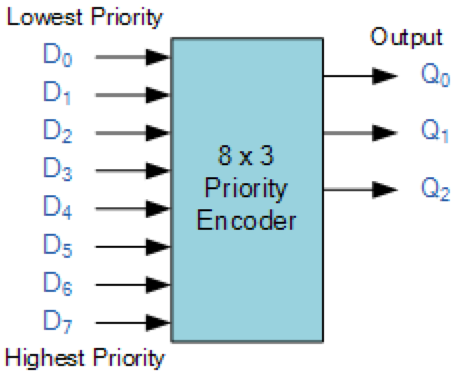


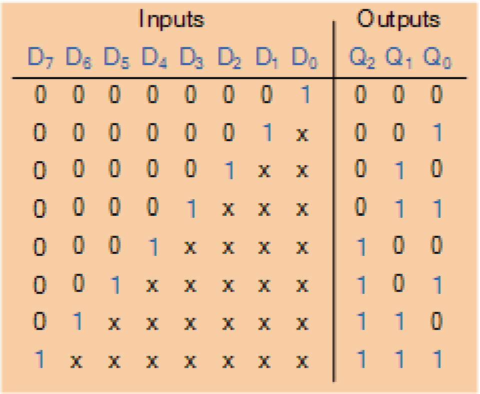
Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

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 D1 and D2 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 D0 is equal to one. One simple way to overcome this problem is to “Prioritise” the level of each input pin and if there was more than one input at logic level “1” 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.

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





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

X = dont care

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

From this truth table, the Boolean expression for the encoder above with inputs D0 to D7 and outputs Q0, Q1, Q2 is given as:

Output Q0

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

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

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

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

Output Q_1

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

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

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

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

Output Q_2

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

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

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

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

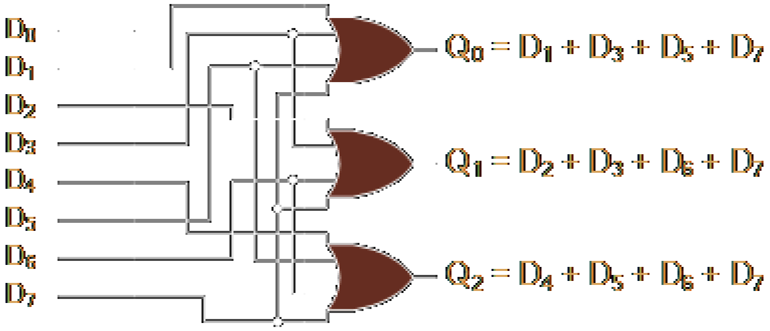
$$Q_0 = \sum\left(\overline{D}_6\left(\overline{D}_4\overline{D}_2 D_1 + \overline{D}_4 D_3 + D_5\right) + D_7\right)$$

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

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

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 above to be constructed using individual OR gates as follows.

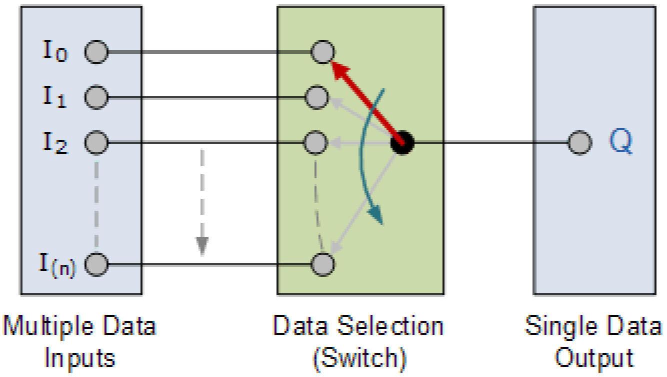
Digital Encoder using Logic Gates



The Multiplexer (MUX)

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.

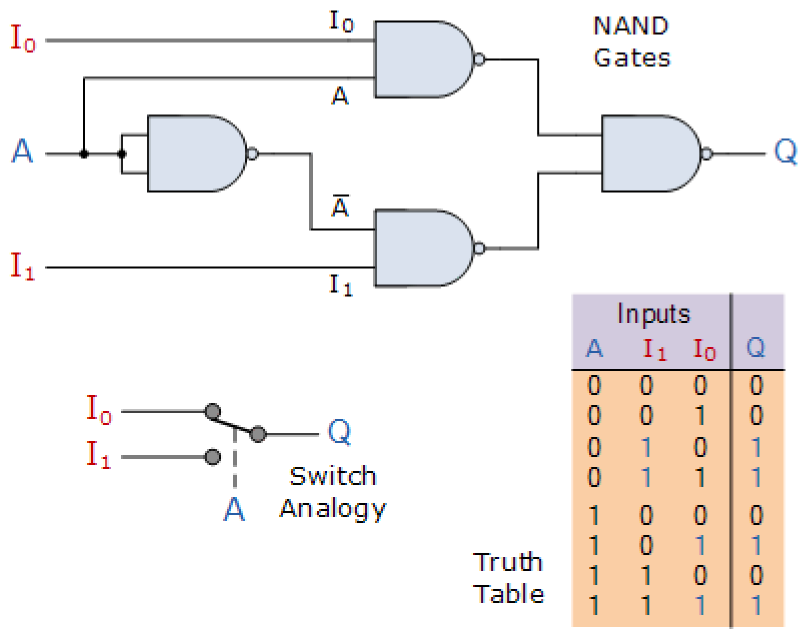
The most basic type of multiplexer device is a one-way rotary switch as shown below.



The rotary switch, also called a wafer switch as each layer of the switch is known as a wafer, is a mechanical device whose input is selected by rotating a shaft. In other words, the rotary switch is a manual switch that you can use to select individual data or signal lines simply by turning its inputs "ON" or "OFF". So how can we select each data input automatically using a digital device. In digital electronics, multiplexers are also known as data selectors because they can "select" each input line, are constructed from individual Analogue Switches encased in a single IC package as opposed to the "mechanical" type selectors such as normal conventional switches and relays. They are used as one method of reducing the number of logic gates required in a circuit design or when a single data line or data bus is required to carry two or more different digital signals. For example, a single 8-channel multiplexer.

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 2N 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.



Inputs			
A	I_1	I_0	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The input A of this simple 2-1 line multiplexer circuit constructed from standard NAND gates acts to control which input (I_0 or I_1) gets passed to the output at Q.

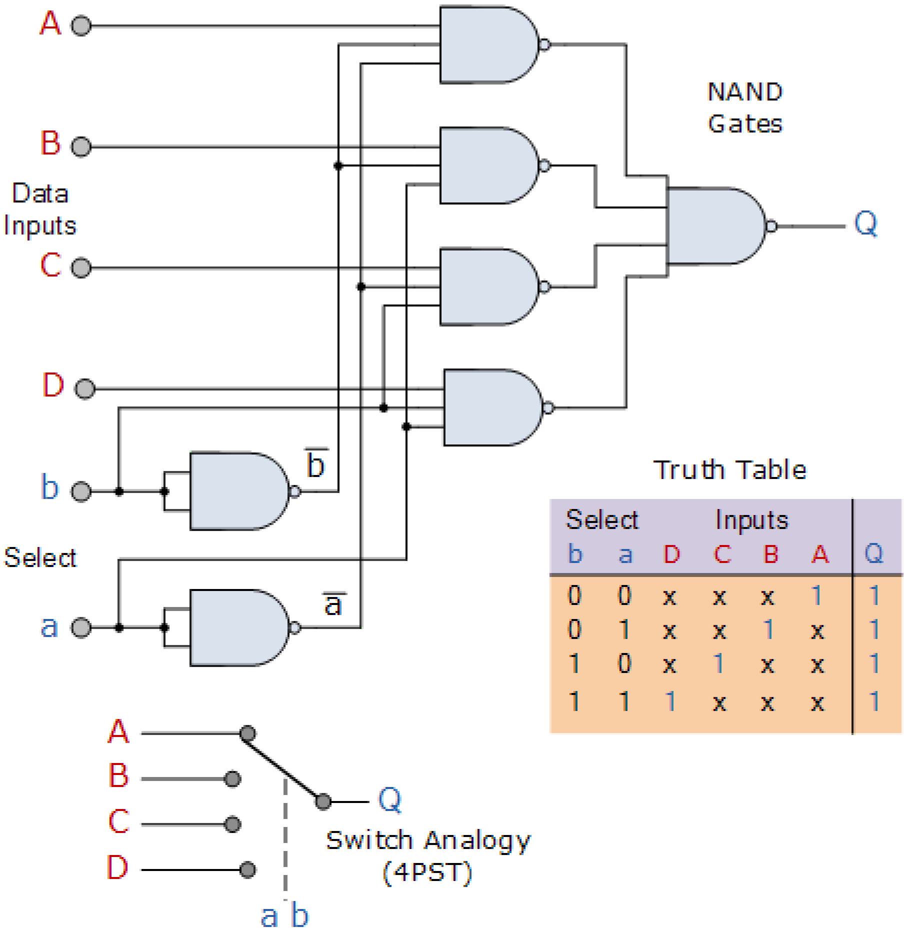
From the truth table we can see that when data select input, A is LOW (logic 0), input I_1 passes its data to the output while input I_0 is blocked. When data select A is HIGH (logic 1), input I_0 is passed to Q while input I_0 is blocked. So by the application of either a logic "0" or a logic "1" at A we can select the appropriate input with the circuit acting a bit like a single pole double throw (SPDT) switch. Then in this simple example, the 2-input multiplexer connects one of two 1-bit sources to a common output, producing a 2-to-1-line multiplexer and we can confirm this in the following Boolean expression.

$$Q = A.I_0.I_1 + A.I_0.I_1 + A.I_0.I_1 + A.I_0.I_1$$

and for our 2-input multiplexer circuit above, this can be simplified too:

$$Q = A.I_1 + A.I_0$$

We can build a simple 2-line to 1-line (2-to-1) multiplexer from basic logic NAND gates as shown.We can increase the number of data inputs to be selected further simply by following the same procedure and larger multiplexer circuits can be implemented using smaller 2-to-1 multiplexers as their basic building blocks. So for a 4-input multiplexer we would therefore require two data select lines as 4-inputs represents 2^2 data control lines give a circuit with four inputs, I_0, I_1, I_2, I_3 and two data select lines A and B as shown.



Truth Table

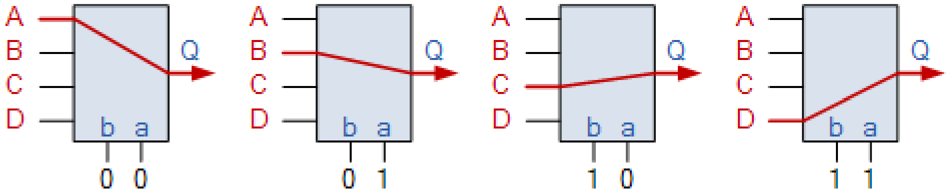
Select		Inputs				
b	a	D	C	B	A	Q
0	0	x	x	x	1	1
0	1	x	x	1	x	1
1	0	x	1	x	x	1
1	1	1	x	x	x	1

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

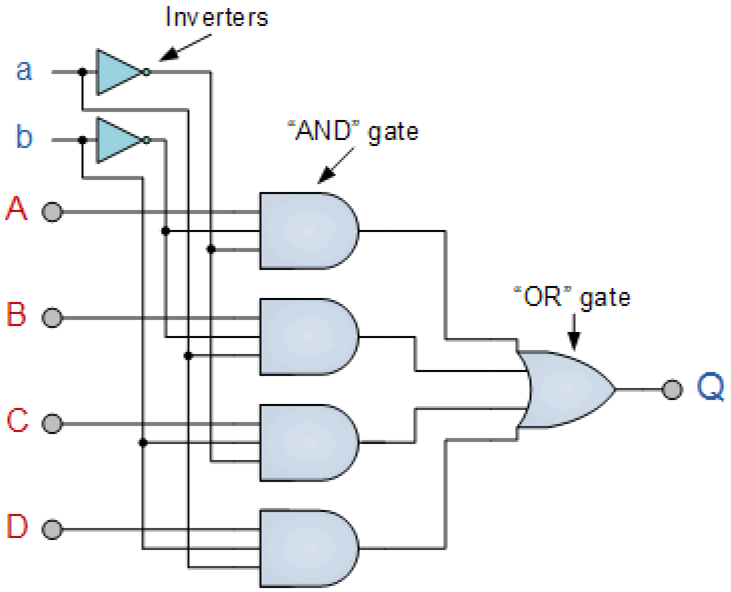
$Q = abA + abB + abC + abD$

In this example at any one instant in time only ONE of the four analogue switches is closed, connecting only one of the input lines A to D to the single output at Q. As to which switch is closed depends upon the addressing input code on lines “a” and “b“, so for this example to select input B to the output at Q, the binary input address would need to be “a” = logic “1” and “b” = logic “0”.

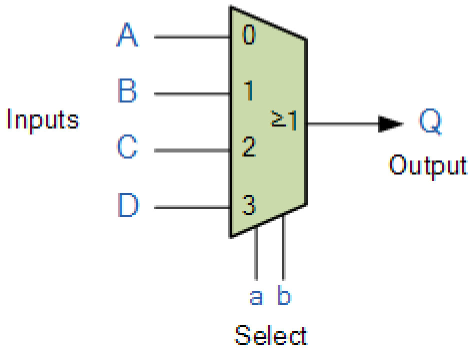
Then we can show the selection of the data through the multiplexer as a function of the data select bits as shown.



Adding more control address lines will allow the multiplexer to control more inputs but each control line configuration will connect only ONE input to the output. Then the implementation of the Boolean expression above using individual logic gates would require the use of seven individual gates consisting of AND, OR and NOT gates as shown



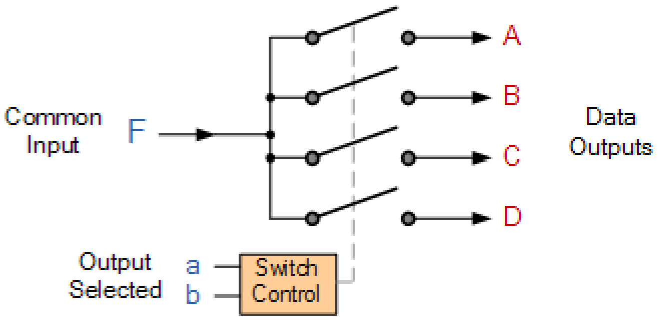
Multiplexer Symbol

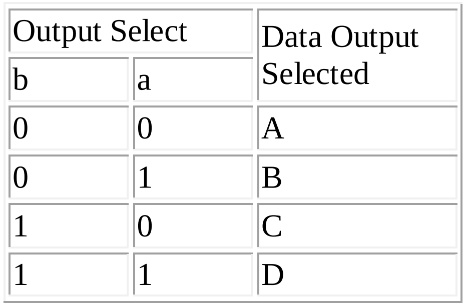


The Demultiplexer

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



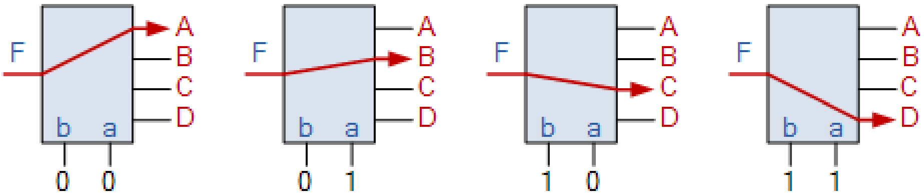


Output Select		Data Output Selected
b	a	
0	0	A
0	1	B
1	0	C
1	1	D

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 = a'b'A + a'bB + ab'C + abD

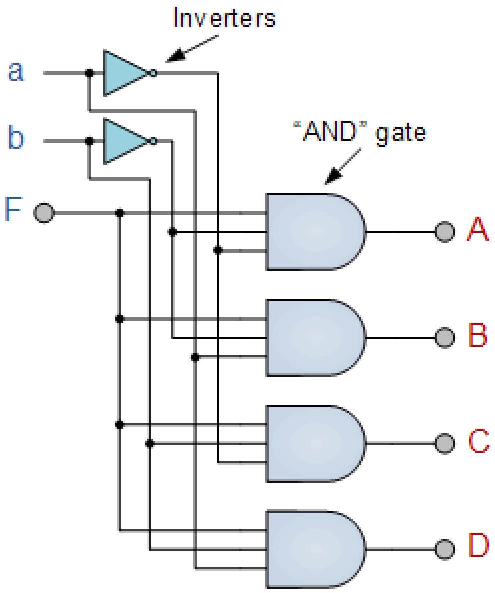
The function of the Demultiplexer is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" as shown.



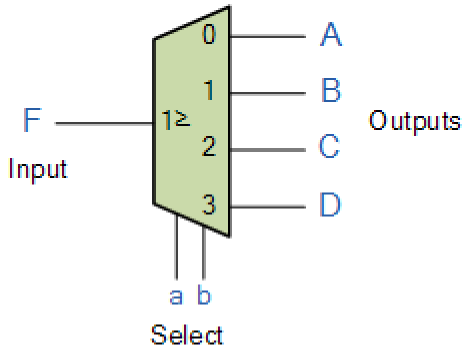
As with the previous multiplexer circuit, adding more address line inputs it is possible to switch more outputs giving a 1-to-2^n data line outputs.

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.



The Demultiplexer Symbol



Exercise

1. Explain the function of parallel adder, parallel subtractor with suitable logic diagram.
2. Design the logical circuit for decimal to binary encoder.
3. Design the logical circuit for binary to decimal decoder.
4. Design a full adder using MUX
5. Give the application of MUX & DEMUX.