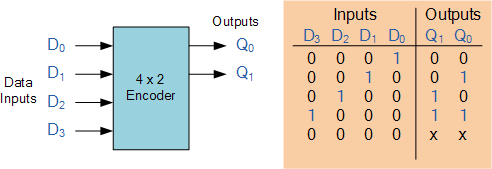
**The Digital Encoder**

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. output code.

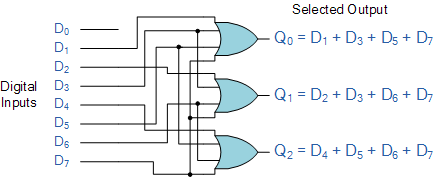
**4-to-2 Bit Binary Encoder**



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** for short.

**Digital Encoder using Logic Gates**



**Encoder Applications**

**Keyboard Encoder**

Priority encoders can be used to reduce the number of wires needed in a particular circuits or application that have multiple inputs. For example, assume that a microcomputer needs to read the 104 keys of a standard QWERTY keyboard where only one key would be pressed either "HIGH" or "LOW" at any one time. One way would be to connect all 104 wires from the keys directly to the computer but this would be impractical for a small home PC, but another better way would be to use a priority encoder.

The 104 individual buttons or keys could be encoded into a standard ASCII code of only 7-bits (0 to 127 decimal) to represent each key or character of the keyboard and then inputted as a much smaller 7-bit B.C.D code directly to the computer. Keypad encoders such as the 74C923 20-key encoder are available to do just that.

**Positional Encoders**

Another more common application is in magnetic positional control as used on ships navigation or for robotic arm positioning etc. Here for example, the angular or rotary position of a compass is converted into a digital code by a 74LS148 8-to-3 line priority encoder and inputted to the systems computer to provide navigational data and an example of a simple 8 position to 3-bit output compass encoder is shown below. Magnets and reed switches could be used at each compass point to indicate the needles angular position.

**Digital Encoder Summary**

Then to summarise, the **Digital Encoder** is a combinational circuit that generates a specific code at its outputs such as binary or BCD in response to one or more active inputs. There are two main types of digital encoder. The **Binary Encoder** and the **Priority Encoder**.

The **Binary Encoder** converts one of 2n inputs into an n-bit output. Then a binary encoder has fewer output bits than the input code. Binary encoders are useful for compressing data and can be constructed from simple AND or OR gates. One of the main disadvantages of a standard binary encoder is that it would produce an error at its outputs if more than one input were active at the same time. To overcome this problem priority encoders were developed.

The **Priority Encoder** is another type of combinational circuit similar to a binary encoder, except that it generates an output code based on the highest prioritised input. Priority encoders are used extensively in digital and computer systems as microprocessor interrupt controllers where they detect the highest priority input.

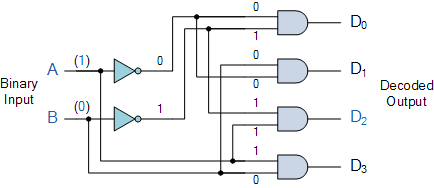
**Binary Decoder**

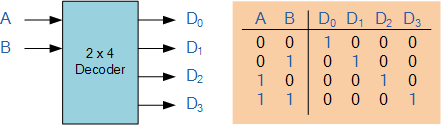
A **Decoder** is the exact opposite to that of an "Encoder" we looked at in the last tutorial. It is basically, a combinational type logic circuit that converts the binary code data at its input into one of a number of different output lines, one at a time producing an equivalent decimal code at its output. **Binary Decoders** have inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, and a n-bit decoder has 2n output lines.

Therefore, if a binary decoder receives n inputs (usually grouped as a binary or Boolean number) it activates one and only one of its 2n outputs based on that input with all other outputs deactivated. 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.

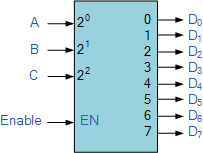
A *Binary Decoder* converts coded inputs into coded outputs, where the input and output codes are different and decoders are available to "decode" either a Binary or BCD (8421 code) input pattern to typically a Decimal output code. Commonly available BCD-to-Decimal decoders include the TTL 7442 or the CMOS 4028. An example of a 2-to-4 line decoder along with its truth table is given below. It consists of an array of four NAND gates, one of which is selected for each combination of the input signals A and B.

**A 2-to-4 Binary Decoders.**





In this simple example of a 2-to-4 line binary decoder, the binary inputs A and B determine which output line from D0 to D3 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 and these types of binary decoders are commonly used as **Address Decoders** in microprocessor memory applications.



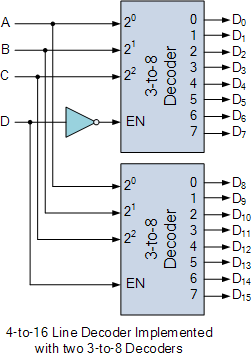
74LS138 Binary Decoder

Some binary decoders have an additional input labelled "Enable" that controls the outputs from the device. This allows the decoders outputs to be turned "ON" or "OFF" and we can see that the logic diagram of the basic decoder is identical to that of the basic demultiplexer.

Then, 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 address which can access a location having that address.

Sometimes it is required to have a **Binary Decoder** with a number of outputs greater than is available, or if we only have small devices available, we can combine multiple decoders together to form larger decoder networks as shown. Here a much larger 4-to-16 line binary decoder has been implemented using two smaller 3-to-8 decoders.

**A 4-to-16 Binary Decoder Configuration.**



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".

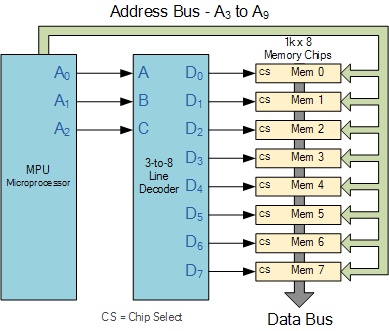
**Memory Address Decoder.**

**Binary Decoders** are most often used in more complex digital systems to access a particular memory location based on an "address" produced by a computing device. In modern microprocessor systems the amount of memory required can be quite high and is generally more than one single memory chip alone. One method of overcoming this problem is to connect lots of individual memory chips together and to read the data on a common "Data Bus". In order to prevent the data being "read" from each memory chip at the same time, each memory chip is selected individually one at time and this process is known as **Address Decoding**.

In this application, the address represents the coded data input, and the outputs are the particular memory element select signals. Each memory chip has an input called **Chip Select** or **CS** which is used by the MPU to select the appropriate memory chip and a logic "1" on this input selects the device and a logic "0" on the input de-selects it. By selecting or de-selecting each chip, allows us to select the correct memory device for a particular address and when we specify a particular memory address, the corresponding memory location exists ONLY in one of the chips.

For example, Lets assume we have a very simple microprocessor system with only 1Kb of RAM memory and 10 address lines. The memory consists of 128x8-bit (128x8 = 1024 bytes) devices and for 1Kb we will need 8 individual memory devices but in order to select the correct memory chip we will also require a 3-to-8 line binary decoder as shown below.

**Memory Address Decoding.**



The binary decoder requires 3 address lines, (A0 to A2) to select each one of the 8 chips (the lower part of the address), while the remaining 7 address lines (A3 to A9) select the correct memory location on that chip (the upper part of the address). Having selected a memory location using the address bus, the information at the particular internal memory location is sent to the "Data Bus" for use by the microprocessor. This is of course a simple example but the principals remain the same for all types of memory chips or modules.

**Binary Decoders** are very useful devices for converting one digital format to another, such as binary or BCD type data into decimal or octal etc and commonly available decoder IC's are the TTL 74LS138 3-to-8 line binary decoder or the 74ALS154 4-to-16 line decoder. They are also very useful for interfacing to 7-segment displays such as the TTL 74LS47 which we will look at in the next tutorial.

**BCD to 7-Segment Display Decoder**

As we saw in the previous tutorial, a **Digital Decoder** IC, is a device which converts one digital format into another and one of the most commonly used device for doing this is called the Binary Coded Decimal (BCD) to 7-Segment Display Decoder. 7-segment **LED** (Light Emitting Diode) or **LCD** (Liquid Crystal Display) type displays, provide a very convenient way of displaying information or digital data in the form of numbers, letters or even alpha-numerical characters.

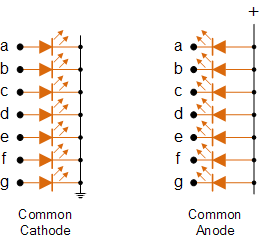
Typically 7-segment displays consist of seven individual coloured LED's (called the segments), within one single display package. In order to produce the required numbers or HEX characters from 0 to 9 and A to F respectively, on the display the correct combination of LED segments need to be illuminated and **BCD to 7-segment Display Decoders** such as the 74LS47 do just that.

A standard 7-segment LED display generally has 8 input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal display segments. Some single displays have also have an additional input pin to display a decimal point in their lower right or left hand corner.

In electronics there are two important types of 7-segment LED digital display.

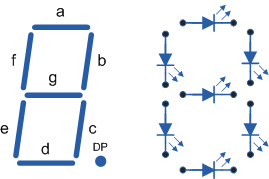
* 1. The Common Cathode Display (CCD) – In the common cathode display, all the cathode connections of the LED's are joined together to logic "0" or ground. The individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual Anode terminals.
* 2. The Common Anode Display (CAD) – In the common anode display, all the anode connections of the LED's are joined together to logic "1" and the individual segments are illuminated by connecting the individual Cathode terminals to a "LOW", logic "0" signal.

**Common Cathode and Common Anode Format**



Electrical connection of the individual diodes for a common cathode display and a common anode display and by illuminating each light emitting diode individually, they can be made to display a variety of numbers or characters.

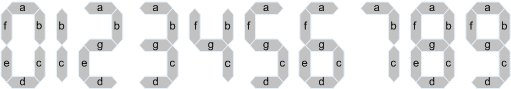
**7-Segment Display Format**



So in order to display the number 3 for example, segments a, b, c, d and g would need to be illuminated. If we wanted to display a different number or letter then a different set of segments would need to be illuminated. Then for a 7-segment display, we can produce a truth table giving the segments that need to be illuminated in order to produce the required character as shown below.

**Truth Table for a 7-segment display**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Individual Segments | | | | | | | Display | | a | b | c | d | e | f | g | | × | × | × | × | × | × |  | 0 | |  | × | × |  |  |  |  | 1 | | × | × |  | × | × |  | × | 2 | | × | × | × | × |  |  | × | 3 | |  | × | × |  |  | × | × | 4 | | × |  | × | × |  | × | × | 5 | | × |  | × | × | × | × | × | 6 | | × | × | × |  |  |  |  | 7 | | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Individual Segments | | | | | | | Display | | a | b | c | d | e | f | g | | × | × | × | × | × | × | × | 8 | | × | × | × | × |  | × | × | 9 | | × | × | × |  | × | × | × | A | |  |  | × | × | × | × | × | b | | × |  |  | × | × | × |  | C | |  | × | × | × | × |  | × | d | | × |  |  | × | × | × | × | E | | × |  |  |  | × | × | × | F | |



7-Segment Display Elements for all Numbers.

It can be seen that to display any single digit number from 0 to 9 or letter from A to F, we would need 7 separate segment connections plus one additional connection for the LED's "common" connection. Also as the segments are basically a standard light emitting diode, the driving circuit would need to produce up to 20mA of current to illuminate each individual segment and to display the number 8, all 7 segments would need to be lit resulting a total current of nearly 140mA, (8 x 20mA).

Obviously, the use of so many connections and power consumption is impractical for some electronic or microprocessor based circuits and so in order to reduce the number of signal lines required to drive just one single display, display decoders such as the BCD to 7-Segment Display Decoder and Driver IC's are used instead.

**Binary Coded Decimal**

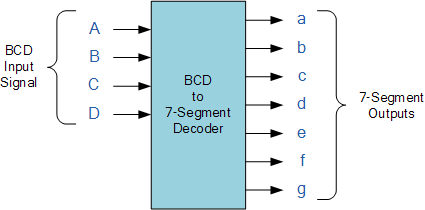
**Binary Coded Decimal** (BCD or "8421" BCD) numbers are made up using just 4 data bits (a nibble or half a byte) similar to the [**Hexadecimal**](http://www.electronics-tutorials.ws/binary/bin_3.html) numbers we saw in the binary tutorial, but unlike hexadecimal numbers that range in full from 0 through to F, BCD numbers only range from 0 to 9, with the binary number patterns of 1010 through to 1111 (A to F) being invalid inputs for this type of display and so are not used as shown below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Decimal | Binary Pattern | | | | BCD | | 8 | 4 | 2 | 1 | | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 1 | 1 | | 2 | 0 | 0 | 1 | 0 | 2 | | 3 | 0 | 0 | 1 | 1 | 3 | | 4 | 0 | 1 | 0 | 0 | 4 | | 5 | 0 | 1 | 0 | 1 | 5 | | 6 | 0 | 1 | 1 | 0 | 6 | | 7 | 0 | 1 | 1 | 1 | 7 | |  | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Decimal | Binary Pattern | | | | BCD | | 8 | 4 | 2 | 1 | | 8 | 1 | 0 | 0 | 0 | 8 | | 9 | 1 | 0 | 0 | 1 | 9 | | 10 | 1 | 0 | 1 | 0 | Invalid | | 11 | 1 | 0 | 1 | 1 | Invalid | | 12 | 1 | 1 | 0 | 0 | Invalid | | 13 | 1 | 1 | 0 | 1 | Invalid | | 14 | 1 | 1 | 1 | 0 | Invalid | | 15 | 1 | 1 | 1 | 1 | Invalid | |
|  |  |

**BCD to 7-Segment Display Decoders**

A binary coded decimal (BCD) to 7-segment display decoder such as the TTL 74LS47 or 74LS48, have 4 BCD inputs and 7 output lines, one for each LED segment. This allows a smaller 4-bit binary number (half a byte) to be used to display all the denary numbers from 0 to 9 and by adding two displays together, a full range of numbers from 00 to 99 can be displayed with just a single byte of 8 data bits.

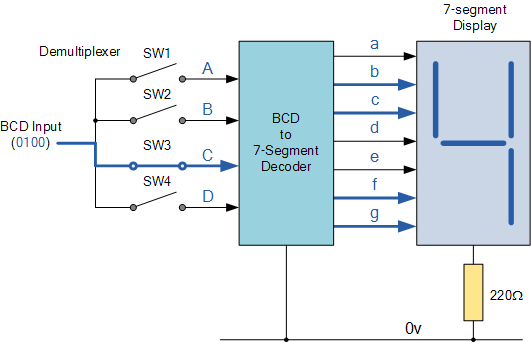
**BCD to 7-Segment Decoder**



The use of **packed** BCD allows two BCD digits to be stored within a single byte (8-bits) of data, allowing a single data byte to hold a BCD number in the range of 00 to 99.

An example of the 4-bit BCD input ( 0100 ) representing the number 4 is given below.

**Example No1**



In practice current limiting resistors of about 150Ω to 220Ω would be connected in series between the decoder/driver chip and each LED display segment to limit the maximum current flow. Different display decoders or drivers are available for the different types of display available, e.g. 74LS48 for common-cathode LED types, 74LS47 for common-anode LED types, or the CMOS CD4543 for liquid crystal display (LCD) types.

Liquid crystal displays (LCD´s) have one major advantage over similar LED types in that they consume much less power and nowadays, both LCD and LED displays are combined together to form larger Dot-Matrix Alphanumeric type displays which can show letters and characters as well as numbers in standard Red or Tri-colour outputs.

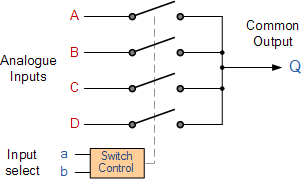
**The Multiplexer**

A data selector, more commonly called a **Multiplexer**, shortened to "Mux" or "MPX", are combinational logic switching devices that operate like a very fast acting multiple position rotary switch. They connect or control, multiple input lines called "channels" consisting of either 2, 4, 8 or 16 individual inputs, one at a time to an output.

Then the job of a "multiplexer" is to allow multiple signals to *share* a single common output. For example, a single 8-channel multiplexer would connect one of its eight inputs to the single data output. Multiplexers are used as one method of reducing the number of logic gates required in a circuit or when a single data line is required to carry two or more different digital signals.

Digital **Multiplexers** are constructed from individual [**analogue switches**](http://www.electronics-tutorials.ws/combination/comb_1.html) encased in a single IC package as opposed to the "mechanical" type selectors such as normal conventional switches and relays. Generally, multiplexers have an even number of data inputs, usually an even power of two, n2 , a number of "control" inputs that correspond with the number of data inputs and according to the binary condition of these control inputs, the appropriate data input is connected directly to the output. An example of a **Multiplexer** configuration is shown below.

**4-to-1 Channel Multiplexer**



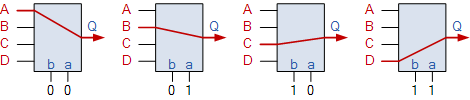
|  |  |  |
| --- | --- | --- |
| Addressing | | Input Selected |
| b | a |
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

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".

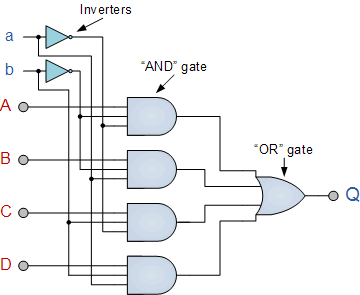
Then we can show the selection of the data through the multiplexer as a function of the data select bits as shown.



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.

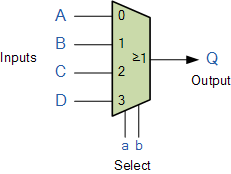
Then the implementation of this Boolean expression above using individual logic gates would require the use of seven individual gates consisting of AND, OR and NOT gates as shown.

**4 Channel Multiplexer using Logic Gates**



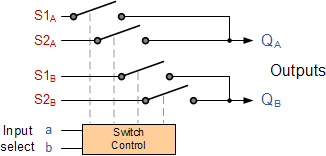
The symbol used in logic diagrams to identify a multiplexer is as follows.

**Multiplexer Symbol**



Multiplexers are not limited to just switching a number of different input lines or channels to one common single output. There are also types that can switch their inputs to multiple outputs and have arrangements or 4 to 2, 8 to 3 or even 16 to 4 etc configurations and an example of a simple Dual channel 4 input multiplexer (4 to 2) is given below:

**4-to-2 Channel Multiplexer**



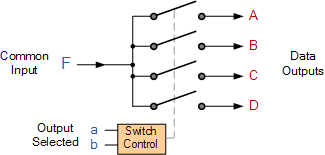
Here in this example the 4 input channels are switched to 2 individual output lines but larger arrangements are also possible. This simple 4 to 2 configuration could be used for example, to switch audio signals for stereo pre-amplifiers or mixers.

The **Multiplexer** is a very useful combinational device that has its uses in many different applications such as signal routing, data communications and data bus control. When used with a demultiplexer, parallel data can be transmitted in serial form via a single data link such as a fibre-optic cable or telephone line. They can also be used to switch either analogue, digital or video signals, with the switching current in analogue power circuits limited to below 10mA to 20mA per channel in order to reduce heat dissipation.

**The Demultiplexer**

The data distributor, known more commonly as a **Demultiplexer** or "Demux", is the exact opposite of the [**Multiplexer**](http://www.electronics-tutorials.ws/combination/comb_2.html) 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**

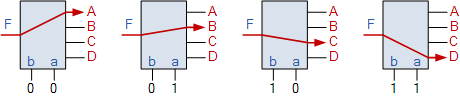


|  |  |  |
| --- | --- | --- |
| Addressing | | Input Selected |
| b | a |
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

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 = ab A + abB +  abC + 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**](http://www.electronics-tutorials.ws/combination/comb_2.html), adding more address line inputs it is possible to switch more outputs giving a 1-to-2n 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".

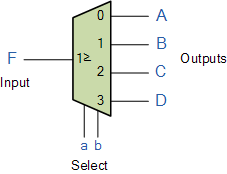
The implementation of the Boolean expression above using individual logic gates would require the use of six individual gates consisting of AND and NOT gates as shown.

**4 Channel Demultiplexer using Logic Gates**



The symbol used in logic diagrams to identify a demultiplexer is as follows.

**The Demultiplexer Symbol**



Standard **Demultiplexer** IC packages available are the TTL 74LS138 1 to 8-output demultiplexer, the TTL 74LS139 Dual 1-to-4 output demultiplexer or the CMOS CD4514 1-to-16 output demultiplexer. Another type of demultiplexer is the 24-pin, 74LS154 which is a 4-bit to 16-line demultiplexer/decoder. Here the individual output positions are selected using a 4-bit binary coded input. Like multiplexers, demultiplexers can also be cascaded together to form higher order demultiplexers.

Unlike multiplexers which convert data from a single data line to multiple lines and demultiplexers which convert multiple lines to a single data line, there are devices available which convert data to and from multiple lines and in the next tutorial about combinational logic devices, we will look at [**Encoders**](http://www.electronics-tutorials.ws/combination/comb_4.html) which convert multiple input lines into multiple output lines, converting the data from one form to another.

**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 SUM of 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". 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**](http://www.electronics-tutorials.ws/logic/logic_6.html) 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**](http://www.electronics-tutorials.ws/logic/logic_6.html) 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**](http://www.electronics-tutorials.ws/logic/logic_2.html) 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**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Truth Table | | | |
| Half Adder | A | B | SUM | CARRY |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| Boolean Expression: Sum = A ⊕ B     Carry = A **.** B | | | | |

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**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Symbol | Truth Table | | | | |
| Full Adder | 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 | | | | | |

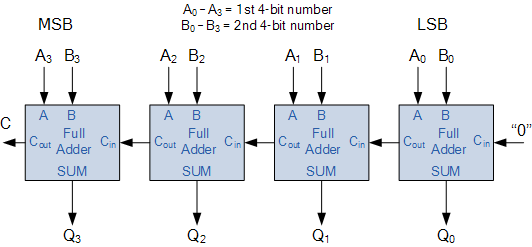
The 1-bit **Full Adder** circuit above is basically two half adders connected together and consists of three Ex-OR gates, two AND gates and an OR gate, six logic gates in total. The truth table for the full adder includes an additional column to take into account the Carry-in input as well as the summed 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**.

**The 4-bit Binary 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.

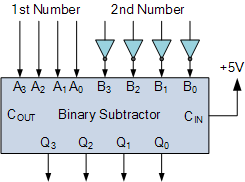
**A 4-bit Binary Adder**



One main disadvantage 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 delay is called **Propagation delay**. Also "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.

**The 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 using an inverter or NOT-gate.

Also, in the above circuit for the 4-bit binary adder, the first carry-in input is held LOW at logic "0", for the circuit to perform subtraction this input needs to be held HIGH at "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.

**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

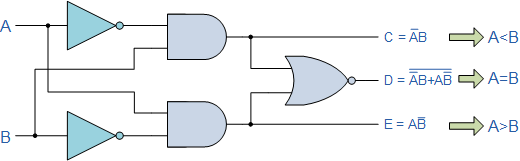
The purpose of a **Digital Comparator** is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, .... An, etc) against that of a constant or unknown value such as B (B1, B2, B3, .... Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For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

Digital Comparator

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.

**1-bit Digital Comparator**



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

**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 |

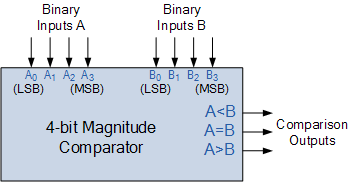
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = B resembles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



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.