http://www.electronics-tutorials.ws

Binary Numbers **Tutorial: 1 of 5**

**Binary Numbers**

Unlike a linear, or analogue circuit which contains signals that are constantly changing from one value to another, such as amplitude or frequency, digital circuits process signals that contain just two voltage levels or states, labelled **logic "0"** and **logic "1"**. Generally, a logic "1" represents a higher voltage, which is referred to as a **HIGH** and a logic "0" is referred to as a **LOW**. These discrete voltage levels are commonly known as **BI**nary digi**TS** and are normally referred to as **BITS**. Because there are only two valid Boolean values for representing either a logic "1" or a logic "0", makes the system of using **Binary Numbers** ideal for use in digital or electronic circuits and systems. The Binary Numbers system is a Base-2 system which follows the same rules in mathematics as the common decimal system meaning instead of powers of ten, for example 1, 10, 100, 1000 etc, binary numbers use powers of two, doubling the value of each successive bit, 1, 2, 4, 8, 16, 32 etc.

The voltages used to represent a digital circuit are called "logic levels" and ideally one voltage level represents a HIGH and another represents a LOW. Digital waveforms or signals consist of discrete voltage levels that are changing back and forth between these HIGH and LOW levels or states. But what makes a signal or voltage "Digital" and how can we represent these voltage levels. Electronic circuits can be divided into two main categories.

* Analogue Circuits - Analogue or Linear circuits amplify or respond to continuously varying voltage levels over a period of time.
* Digital Circuits - Digital circuits produce or respond too two distinct voltage levels representing either a Logic level "1" or a Logic level "0".

**Analogue Voltage Output**

An example of an analogue (or analog) circuit and a digital circuit are shown below:

**Analogue Voltage Output Representation**

|  |
| --- |
| analogue circuit |

The output from the potentiometer varies as the wiper terminal is rotated producing an infinite number of voltage points between 0 volts and V max. As the voltage output varies either slowly or rapidly there is no sudden change between two voltage levels giving a continuous output voltage. Examples of analogue signals include temperature, pressure, liquid levels and light intensity.

**Digital Voltage Output**

In this example the potentiometer wiper has been replaced by a rotary switch which is connected to each junction of the resistor chain, forming a potential divider network. As the switch is rotated from one node to the next the output changes quickly in discrete voltage levels of multiples of 1 volt each, as shown in the graph. For example, 2 volts, 3 volts, 5 volts etc but NOT 2.5V, 3.1V or 4.6V. Finer output voltage levels could easily be produced by increasing the number of resistors within the potential divider chain.

**Digital Voltage Output Representation**

|  |
| --- |
| digital circuit |

Then the major difference between an analogue signal or quantity and a digital quantity is that an "Analogue" quantity is continuously changing over time while a "Digital" quantity has discrete (step by step) values. LOW to HIGH or HIGH to LOW. Another example of this could be a light dimmer that varies the light intensity up or down between fully-ON and fully-OFF, but with a light switch the light is either "ON", (HIGH) or it is "OFF", (LOW).

Some circuits combine both analogue and digital signals such as an analogue to digital converter (ADC) or a digital to analogue converter (DAC). Either way, the digital input or output signal represents a binary number value of the analogue signal.

**Logic Levels**

In all electronic circuits, only two logic levels are allowed and these levels are referred to as "logic 1 or logic 0", "high or low", "true or false". Most logic systems use positive logic, in which a logic "0" is represented by zero volts and a logic "1" is represented by a higher voltage, such as +5 volts and the switching from one voltage level, "0" to "1" or "1" to "0" is made as quickly as possible to prevent faulty operation of the logic circuit. In standard TTL (transistor-transistor-logic) IC's there is a defined range of input and output voltage limits for defining what is a logic "1" value and what is a logic "0" value and this is shown below.

**TTL Input & Output Voltage Levels**

|  |
| --- |
| digital logic levels |

Then, when using a +5 volt supply any voltage input between 2.0v and 5v is recognised as a logic "1" value and any voltage input of below 0.8v is recognised as a logic "0" value. While the output of a logic gate between 2.7v and 5v represents a logic "1" value and a voltage output below 0.4v represents a logic "0" value. This is called "positive logic" and is used in these tutorials.

**Noise**

However, between these defined HIGH and LOW values lies what is generally called a "no-man's land" (the blue area's above) and if we apply a signal voltage of a value within this no-man's land area we do not know whether the logic gate will respond to it as a level "0" or as a level "1", and the output will become unpredictable. **Noise** is the name given to a random and unwanted voltage that is induced into electronic circuits by external interference, such as from nearby switches, power supply fluctuations or from wires and other conductors that pick-up stray electromagnetic radiation. Then in order for a logic gate not to be influence by noise in must have a certain amount of noise margin or noise immunity.

**Noise Immunity**

|  |
| --- |
| Digital Logic Noise Immunity |

In the example above, the noise signal is superimposed onto the Vcc supply voltage and as long as it stays above the min level (Von-min) the input an corresponding output of the logic gate are unaffected. But when the noise level becomes large enough and a noise spike causes the HIGH voltage level to drop below this minimum level, the logic gate may interpret this spike as a LOW level input and switch the output accordingly producing a false output switching. Then in order for the logic gate not to be affected by noise it must be able to tolerate a certain amount of unwanted noise on its input without changing the state of its output.

Then binary numbers are represented by either a logic "0" or a logic "1" and in the next tutorial about Binary Logic we will look at converting decimal numbers into binary numbers and vice versa and introduce the concept of the **Byte** and the **Word** to represent parts of a binary number.

Convert Decimal to Binary **Tutorial: 2 of 5**

**Convert Decimal to Binary**

The **Decimal** or "denary" counting system uses the **Base of 10** numbering system where each digit in a number takes on one of ten possible values from 0 to 9, eg 21310 (Two Hundred and Thirteen). In a decimal system each digit has a value ten times greater than its previous number and this decimal numbering system uses a set of symbols, b, together with a base, q, to determine the weight of each digit within a number. For example, the six in sixty has a lower weighting than the six in six hundred and in a binary numbering system we need some way of converting **Decimal to Binary**.

Any numbering system can be summarised by the following relationship:

|  |  |
| --- | --- |
|  | N = bi qi |
|  | where: | N is a real positive numberb is the symbolq is the base valueand integer (i) can be positive, negative or zero |
| N = b2 q2 + b1 q1 + b0 q0 + b-1 q-1 ... etc. |

In the decimal or denary system, the columns have values of units, tens, hundreds etc as we move from right to left and mathematically these values are written as 100, 101, 102, 103 etc. The decimal numbering system has a base of 10 or modulo-10 (sometimes called MOD-10) with the position of each digit in the decimal system indicating the magnitude or weight of the number. For example, 20 (twenty) is the same as saying 2 x 101 and 400 (four hundred) is the same as saying 4 x 102. Likewise, for fractional numbers the weight of the number is negative, 10-1, 10-2, 10-3 etc.

The value of a decimal number is equal to the sum of the digits multiplied by their respective weights. For example: N = 616310 (Six Thousand One Hundred and Sixty Three) in a decimal format is equal to:

(6×103) + (1×102) + (6×101) + (3×100) = 6163

Unlike the decimal numbering system which uses the base of 10, digital logic uses just two values or states, a logic level "1" or a logic level "0", so each "0" and "1" is considered to be a single digit in a **Base of 2** or **Binary** numbering system. In the binary numbering system, each digit has a value twice that of the previous digit but can only have a value of either "1" or "0" therefore, q = "2" and the position of either a "0" or a "1" indicates its weight.

For example:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Decimal Digit Value** | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|  |
| **Binary Digit Value** | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |

By adding together all the decimal number values from right to left at the positions that are represented by a "1" gives us: (256) + (64) + (32) + (4) + (1) = 35710 or three hundred and fifty seven in decimal.

Then, the binary array of digits 1011001012 is equivalent to 35710 in decimal or denary. As the decimal number is a weighted number, converting from decimal to binary will also produce a weighted binary number with the right-hand most bit being the **Least Significant Bit** or **LSB**, and the left-hand most bit being the **Most Significant Bit** or **MSB**. and we can represent these as.

|  |  |  |
| --- | --- | --- |
| MSB | Binary Digit | LSB |
| 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**Repeated Division-by-2 Method**

Another method of converting *Decimal to Binary* number equivalents is to write down the decimal number and to continually divide by 2 (two) to give a result and a remainder of either a "1" or a "0" until the final result equals zero.

Example. Convert the decimal number 29410into its binary number equivalent.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number | 294 |  |  |  | Dividing each number by "2" gives a result plus a remainder. The binary result is obtained by placing the remainders in order with the least significant bit (LSB) being at the top and the most significant bit (MSB) being at the bottom.  |
| divide by 2 |
| result | 147 | remainder | **0** (LSB) |
| divide by 2 |
| result | 73 | remainder | **1** |
| divide by 2 |
| result | 36 | remainder | **1** |
| divide by 2 |
| result | 18 | remainder | **0** |
| divide by 2 |
| result | 9 | remainder | **0** |
| divide by 2 |
| result | 4 | remainder | **1** |
| divide by 2 |
| result | 2 | remainder | **0** |
| divide by 2 |
| result | 1 | remainder | **0** |
| divide by 2 |
| result | 0 | remainder | **1** (MSB) |

Then, the decimal to binary conversion gives the decimal number 29410 equivalent of 1001001102 in binary, reading from right to left.

Then the main characteristics of a **Binary Numbering System** is that each "digit" or "bit" has a value of either "1" or "0" with each digit having a weight or value double that of its previous bit starting from the lowest or least significant bit (LSB) and this is called the "sum-of-weights" method. So we can convert a decimal number to binary either by using the sum-of-weights method or by using the repeated division-by-2 method.

**Binary Number Names & Prefixes**

Binary numbers can be added together and subtracted just like decimal numbers with the result being combined into one of several size ranges depending upon the number of bits being used. The classification of individual bits into larger groups are generally referred to by the following more common names of:

|  |  |
| --- | --- |
| Number of Binary Digits **(bits)** | Common Name |
| 1 | Bit |
| 4 | Nibble |
| 8 | Byte |
| 16 | Word |
| 32 | Double Word |
| 64 | Quad Word |

Also, when converting from **Decimal to Binary** or even from **Binary to Decimal**, we need to be careful that we do not mix up the two sets of numbers. For example, if we write the digits 10 on the page it could mean the number "ten" if we assume it to be a decimal number, or it could equally be a "1" and a "0" together in binary, which is equal to the number two in decimal from above. One way to overcome this problem and to identify whether the digits or numbers being used are decimal or binary is to write a small number called a **subscript** after the last digit to show the base of the number system being used. If we were using a binary number we would add the subscript 2 so the number would be written as 102, and if it were a decimal number we would add the subscript 10 so the number would be written as 1010.

Today, as micro-controller or microprocessor systems become increasingly larger, the individual binary digits (bits) are now grouped together into 8´s to form a single BYTE with most computer hardware such as hard drives and memory modules commonly indicate their size in Megabytes or even Gigabytes.

|  |  |
| --- | --- |
| Number of Bytes | Common Name |
| 1,024 (210) | kilobyte (kb) |
| 1,048,576 (220) | Megabyte (Mb) |
| 1,073,741,824 (230) | Gigabyte (Gb) |
| a very long number! (240) | Terabyte (Tb) |

**Decimal to Binary Summary**

* A "BIT" is the abbreviated term derived from BInary digiT
* A Binary system has only two states, Logic "0" and Logic "1"
* A Decimal system uses 10 different digits, 0 to 9
* A Binary number is a weighted number who's weighted value increases from right to left
* The weight of a binary digit doubles from right to left
* A decimal number can be converted to a binary number by using the sum-of-weights method or the repeated division-by-2 method
* When we convert numbers from decimal to binary, subscripts are used to avoid errors

Converting **Decimal to Binary** numbers can be done in a number of different ways as shown above. When converting decimal numbers to binary numbers it is important to remember which is the least significant bit (LSB), and which is the most significant bit (MSB). In the next tutorial about Binary Logic we will look at converting binary numbers into **Hexadecimal Numbers** and vice versa and show that binary numbers can be represented by letters as well as numbers.

Hexadecimal Numbers **Tutorial: 3 of 5**

**Hexadecimal Numbers**

The one main disadvantage of **Binary Numbers** is that the binary equivalent of a large decimal number can be quite long, which makes it difficult to both read or write without producing errors especially when working with 16 or 32-bit numbers. One common way of overcoming this problem is to arrange the binary numbers into groups of four as **Hexadecimal Numbers**, starting with the least significant digit at the right hand side. This Hexadecimal or simply "**Hex**" numbering system uses the **Base of 16** system. Hence, it uses 16 (sixteen) different digits with a combination of numbers from 0 to 9 and the capital letters A to F to represent its Binary or Decimal equivalent.

We can make life easier by splitting these large binary numbers up into even groups to make them more easier to write down and understandable. For example, the following group of binary digits 1101 0101 1100 11112 are much easier to read and understand than 11010101110011112 when they are all bunched up together. In the everyday use of the decimal numbering system we use groups of three digits or 000's from the right hand side to make a very large number such as a million or trillion easier to understand and the same is true in digital systems.

**Hexadecimal Numbers** are a more complex system than using just binary or decimal and is mainly used when dealing with computers and memory address locations. By dividing a binary number up into groups of 4 bits, each group or set of 4 digits can now have a possible value of between "0000" (0) and "1111" (8+4+2+1 = 15) giving a total of **16** different number combinations from 0 to 15. Don't forget that "0" is also a valid digit. We remember from the first tutorial about **Binary Numbers** that a four-bit group of digits is called a "nibble" and as a four-bits are also required to produce a hexadecimal number, a hex digit can also be thought of as a nibble, or half-a-byte. Then two hexadecimal numbers are required to produce one full byte from 00 to FF. Also, since 16 in the decimal system is the fourth power of 2 (or 24), one hex digit has a value equal to four binary digits so now **q** = "16".

The numbers 0 to 9 are still used as in the original decimal system, but the numbers from 10 to 15 are now represented by capital letters of the alphabet from A to F inclusive and the relationship between binary and hexadecimal is shown below.

**Hexadecimal Numbers**

|  |  |  |
| --- | --- | --- |
| Decimal | 4-bit Binary | Hexadecimal |
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |
| 16 | 0001 0000 | 10 (1+0) |
| 17 | 0001 0001 | 11 (1+1) |
| Continuing upwards in groups of four |

Using the original binary number from above 1101 0101 1100 11112 this can now be converted into an equivalent hexadecimal number of D5CF16 which is much easier to read and understand than a long row of 1´s and 0´s that we had before. Similarly, converting Hex based numbers back into binary is simply the reverse operation.

Then the main characteristics of a **Hexadecimal Numbering System** is that there are 16 distinct counting digits from 0 to F with each digit having a weight or value of 16 starting from the least significant bit (LSB). In order to distinguish Hexadecimal numbers from Denary numbers, a prefix of either a "#", (Hash) or a "$" (Dollar sign) is used before the actual **Hexadecimal** value, #D5CF or $D5CF.

As the base of a hexadecimal system is 16, which also represents the number of individual symbols used in the system, the subscript 16 is used to identify a number expressed in hexadecimal. For example, D5CF16

**Counting using Hexadecimal Numbers**

So we now know how to convert 4 binary digits into a hexadecimal number. But what if we had more than 4 binary digits how would we count in hexadecimal beyond the final letter F. The simple answer is to start over again with another set of 4 bits as follows.

0...to...9, A,B,C,D,E,F, 10...to...19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21....etc

Do not get confused, 10 or 20 is **NOT** ten or twenty it is 1 + 0 and 2 + 0 in hexadecimal. In fact twenty does not even exist in hex. With two hexadecimal numbers we can count up to FF which is equal to decimal 255. Likewise, to count higher thanFF we would add a third hexadecimal digit to the left so the first 3-bit hexadecimal number would be 10016, (25610) and the last would be FFF16, (409510). The maximum 4-digit hexadecimal number is FFFF16 which is equal to 65,535 in decimal and so on.

This adding of additional hexadecimal digits to convert both decimal and binary numbers into an **Hexadecimal Number** is very easy if there are 4, 8, 12 or 16 binary digits to convert. But we can also add zero's to the left of the most significant bit, the MSB if the number of binary bits is not a multiple of four. For example, 110010110110012 is a fourteen bit binary number that is to large for just three hexadecimal digits only, yet too small for a four hexadecimal number. The answer is to ADD additional zero's to the left most bit until we have a complete four bit binary number or multiples thereof.

**Adding of Additional 0's to a Binary Number**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Binary Number | 0011 | 0010 | 1101 | 1001 |
|  |
| Hexadecimal Number | 3 | 2 | D | 9 |

The main advantage of a **Hexadecimal Number** is that it is very compact and by using a base of 16 means that the number of digits used to represent a given number is usually less than in binary or decimal. Also, it is quick and easy to convert between hexadecimal numbers and binary.

**Example No1**

Convert the following Binary number 1110 10102 into its Hexadecimal number equivalent.

|  |  |
| --- | --- |
| 111010102 |  |
| Group the bits into four's starting from the right hand side |
| = | 1110 | 1010 |  |
| = | 14 | 10 | (in decimal) |
| = | E | A | (in Hex) |
| Then, the hexadecimal equivalent of the binary number 1110 10102 is #EA16 |

**Example No2**

Convert the following Hexadecimal number #3FA716into its Binary equivalent, and also into its Decimal or Denary equivalent using subscripts to identify each numbering system.

|  |  |
| --- | --- |
| #3FA716 |  |
|  | = 0011 1111 1010 01112 |
|  | = (8192 + 4096 + 2048 + 1024 + 512 + 256 + 128 + 32 + 4 + 2 + 1) |
|  | = 16,29510 |

Then, the Decimal number of **16,295** can be represented as:-

#3FA716 in Hexadecimal

or

0011 1111 1010 01112 in Binary.

In the next tutorial about Binary Logic we will look at converting binary numbers into **Octal Numbers** and vice vesa.

Octal Numbers **Tutorial: 4 of 5**

**Octal Numbers**

**Octal Numbers** are very similar in principle to the previous hexadecimal numbering system except that in Octal a binary number is divided up into groups of only 3 bits, with each group or set of numbers having a distinct value of between "000" (0) and "111" (4+2+1=7) giving a range of just 8, (0, 1, 2, 3, 4, 5, 6, 7) therefore **q** = "8".

Then the main characteristics of an **Octal Numbering System** is that there are 8 distinct counting digits from 0 to 7 with each digit having a weight or value of just 8 starting from the least significant bit (LSB).

As the base of an **Octal Numbers** system is 8, which also represents the number of individual numbers used in the system, the subscript 8 is used to identify a number expressed in octal. For example, 2378

Like hexadecimal, the octal number system provides a convenient way of converting large binary numbers into smaller groups. However, octal numbers is used less frequently than the more common hexadecimal numbering system and has almost disappeared. As octal uses only eight digits there are no letters used but the conversion from binary or denary follows the same pattern as we have seen for hex.

To count above 7 in octal we add another column and start over again in a similar way to hexadecimal.

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21....etc

Again do not get confused, 10 or 20 is **NOT**ten or twenty it is 1 + 0 and2 + 0 in octal exactly the same as for hexadecimal. With two octal numbers, 778 we can count up to 63 in decimal, with three octal numbers,7778 up to 511 in decimal and with four octal numbers,77778 up to 4095 in decimal and so on.

**Example No1.**

Using our previous binary number of11010101110011112 converting it into the octal equivalent is shown as follows.

|  |  |
| --- | --- |
| Binary Digit Value | 001101010111001111 |
|  |
| Group the bits into three´s starting from the right hand side | 001 101 010 111 001 111 |
|  |
| Octal Number form | 1 5 2 7 1 78 |

Thus, 0011010101110011112 in its Binary form is equivalent to1527178 in Octal form or 54,735 in denary.

Binary Numbers **Tutorial: 5 of 5**

**Binary Numbers Summary**

There are different numbering systems used in digital electronic circuits and computers. However, the numbering system used in one type of circuit may be different to that of another type of circuit, for example, the memory of a computer would use hexadecimal numbers while the keyboard uses decimal numbers. Then the conversion from one number system to another is very important with the four main forms of arithmetic being.

* Decimal - The decimal numbering system has a base of 10 (MOD-10) and uses the digits from 0 to 9 to represent a number value.
* Binary - The binary numbering system has a base of 2 (MOD-2) and uses only two digits a "0" and a "1" to represent a number value.
* Octal - The octal numbering system has a base of 8 (MOD-8) and uses 8 digits between 0 and 7 to represent a number value.
* Hexadecimal - The Hexadecimal numbering system has a base of 16 (MOD-16) and uses a total of 16 numeric and alphabetic characters to represent a number value. Hexadecimal numbers consist of digits 0 to 9 and letters A to F.

Long binary numbers are difficult to both read or write and are generally converted into a system more easily understood or user friendly. The two most common derivatives based on binary numbers are the **Octal** and the **Hexadecimal** numbering systems, with both of these limited in length to a byte (8-bits) or a word (16-bits).

Octal numbers can be represented by groups of 3-bits and hexadecimal numbers by groups of 4-bits together, with this grouping of the bits being used in electronic or computer systems in displays or printouts. The grouping together of binary numbers can also be used to represent **Machine Code** used for programming instructions and control such as an **Assembly Language**.

Comparisons between the various **Decimal**, **Binary**, **Hexadecimal** and**Octal** numbers are given in the following table.

**Comparison Table**

|  |  |  |
| --- | --- | --- |
| Base, b | Byte (8-bits) | Word (16-bits) |
| Decimal | 0to25510 | 0to65,53510 |
| Binary | 0000 0000to1111 11112 | 0000 0000 0000 0000to1111 1111 1111 11112 |
| Hexadecimal | 00toFF16 | 0000toFFFF16 |
| Octal | 000to3778 | 000 000to177 7778 |

It can be seen from the above table that the Hexadecimal numbering system uses only four digits to express a single 16-bit word length, and as a result it is the most commonly used **Binary Numbering System** for electronic and micro-electronic systems.