Finally *DeMorgan's Theorem* establishes two very important simplifications[3]:

(A · B)' = A' + B'
(A + B)' = A' · B'

Multiplexers

A digital *multiplexer* is a switching element, like a mechanical selector switch. A two-input switch which selects either (**A**) or (**B**) to an output line **X** on the basis of the setting (**S**) has the Boolean expression,

X = (A . S') + (B . S)

This would take two AND gates and an OR gate to implement, so often digital multiplexers are truly switches with several conduction channels selected by one or more selector addresses. Typical multiplexers are 2 × 1 (**S** is a single bit), 4 × 1 (**S** is a 2-bit number, up to about 16 × 1 (**S** is a 4-bit number). Some CMOS digital multiplexers have a linear transfer characteristic and are used as analogue multiplexers inside analogue audio equipment.

Tri-state

Although we have now got used to the idea of circuits which exist in two-states: HI and LO; or TRUE or FALSE etc., in fact most logic systems also incorporate another state in which the output of the gate is forced into a high-impedance state, or *tri-state*. This third state is incorporated so that multiple gates may contribute to the same output line. In other words, tri-state allows multiplexing of digital signals onto a common wire or track without the need for a dedicated multiplexer circuit. Tri-state gates find their greatest applications in data processing where data and address busses may have many devices contributing values to a data *bus* (multiple lines carrying whole data words) at specific times.

A NAND gate with a tri-state output is illustrated in Figure 17. Ignoring the role of D1 and D2 for a moment, this is a conventional NAND gate implemented with transistors and known as a TTL circuit (see below). If either of the input lines (A or B) is low, point ***p*** will be brought low and the output of the gate will be HI (because Tr1 is an inverting stage). *Only* when both A and B are both HI will the output of the gate be LO. It is thereby NOT an AND gate or a NAND gate. If however the line "enable/tri-state'" is brought LO, current is poached away from the collector of Tr1 and the output falls to near ground. Tr2 is thereby switched off. Furthermore, Tr3 is switched off by having the data at point *p* forced low by the action of D2. In this way, both the output transistors are off and the output is tri-state. Clearly the speed with which a gate goes into and comes back out of tri-state is of great importance. If wrongly allowed for in the accompanying circuitry this can lead to signal (or bus) *contention* when one or more outputs are trying to control the same signal line.

Logic families

You may hear the term *logic family* from time to time. These are largely historical as modern digital products have mostly absorbed discrete logic-gates into Application Specific Integrated Circuits (ASICS) or into Field Programmable Gate Arrays (FPGAs). Nevertheless, there are many examples of equipment from consoles, to outboard, to instruments which use these components. Each family actually refers to a different technology of logic gate. The logical functions never differ, but the voltages which represent high (1) and low (0) may be different. Also some families have input and output resistances which mean that gate outputs can effectively drive a virtually unlimited number of gate inputs (of the same family). These families are said to have high *fan-out*

[3] A philosopher would say, the first rule states that, “The negation of a conjunction is the disjunction of the negations.” And that the second says, “The negation of a disjunction is the conjunction of the negations.”

and high *fan-in*. All families based on MOS (CMOS) technology possess this advantage. Other families limit the number of gate inputs, like the TTL family and its derivatives.

TTL Logic

TTL stands for *Transistor-Transistor Logic* and was invented in the early nineteen-sixties when some early, rather exotic devices were produced. However the 54xx and 74xx series of TTL integrated circuits introduced by Texas Instruments in the mid-nineteen-sixties really popularised this technology and introduced the first digital logic *family*. TTL devices are made from bipolar transistors, rather than using FET technology. The distinguishing features of the basic TTL gates is that they demand a power rail which is very close to +5V, and they use a relatively high amount of current to drive their logic levels. Stated logical voltage levels are <1V for a logical LO, and above about 3.5V for a logical HI). However these voltage levels are a bit misleading as TTL is really a current operated logic family (see Figure 17). Inputs of TTL logic gates "float high" (rise to a logical HI) if left unconnected. This means that the main requirement for driving a TTL input is to pull enough current from the input to "pull down" the input level to near 0V. This takes a few milliamps per input (depending on the sub-family of logic gates). The outputs of TTL therefore have a requirement to sink current, rather than to supply it (source current). The TTL family has at least 6 subfamilies which offer different speed/power tradeoffs. The most common are LS and ALS. The *delay* figures relates to the propagation-delay from input to output in a gate: in other words, the *switching speed* of the gate.

Family	delay (ns)	power (mW)
basic	**10**	**10**
low-power (L)	**35**	**1**
Schottky (S)	**3**	**18**
low-power Schottky (LS)	**9**	**2**
advanced Schottky (AS)	**1.5**	**10**
advanced low-power Schottky (ALS)	**4**	**1**

CMOS

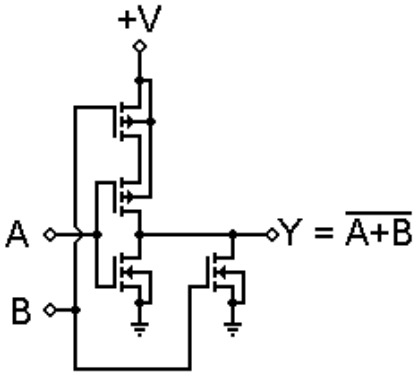

Figure 19 - CMOS NOR gate

CMOS logic post-dated TTL and is based on the use of complementary MOSFET transistors to perform logic functions. Whereas TTL is essentially current operated logic, CMOS is a voltage-operated logic family. Because the gate input circuit is the high impedance of a MOSFET gate (see Figure 19), no current is required to drive an input. This has the advantage that a single logical output can drive many downstream inputs. The very low power consumption of CMOS and the fact that the original CMOS family operated from a wide range of supply voltage (3 to 15V) made CMOS very popular in battery-powered applications in audio. However CMOS isn't perfect and logical operation speeds are limited by the ability to charge and discharge the gate (and Miller) capacitances of following gates. In fact, current consumption varies linearly with operation speed for most families of CMOS.

ECL and LVDS

ECL or *Emitter Coupled Logic* was developed at IBM for very high-speed logical operation it is the one family here which still has widespread application; although not especially in audio equipment. Both TTL and CMOS employ transistors (or MOSFETs) which are operated either in cut-off or in saturation; they are either "fully-on", or "fully-off". The problem with this form of operation is that for various physical reasons a transistor or MOSFET is at its most sluggish as it exits one or other of these states.

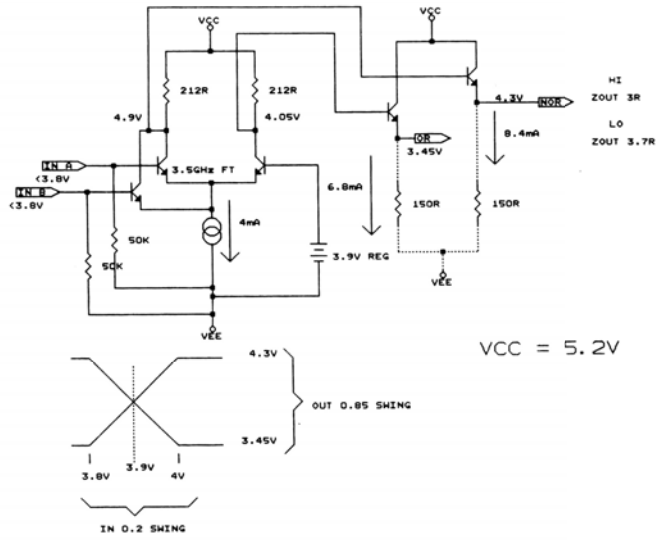

Figure 20 - ECL logic gate

The ECL family avoids this by operating transistor switches within the transistors' active range and adopting a balanced configuration for all the gates, as shown in Figure 20. In an ECL gate it is not the presence or absence of current through a transistor which indicates the logical state but the *path of the current*. This architecture (which in many ways we might describe as analogue digital-electronics), ensures fast switching and low noise - the latter because current is only ever being steered rather than switched on and off. It pays for these virtues by consuming significantly greater power than other logic families. For historical reasons, ECL operated with the collectors of the transistors (***Vcc***) at 0V (GND) and with a negative supply of -5V. *PECL* refers to ECL operated with a positive supply. The voltages annotated in Figure illustrate the gate operated as PECL. A low voltage variant of PECL exists, designed to operate on 3.3V, instead of 5V, this is called LV(Low Voltage) PECL. In many modern designs where PECL would previously have been used, a new technique is employed using balanced CMOS technology to drive signals in a controlled-impedance current-loop. This technique is known as *Low Voltage Differential Signalling* or *LVDS*. The general principle is illustrated in Figure 21.

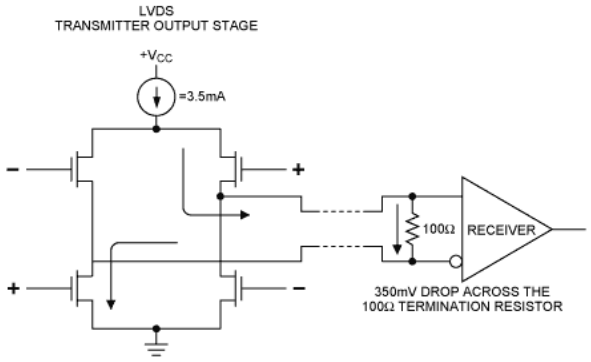

Figure 21 - Low Voltage Differential Signaling

Low Voltage Technologies

For nearly 20 years, the standard supply for digital circuits was 5V and the logical states were ≈ 0V for LO and ≈ 5V for HI. This voltage was perpetuated once CMOS was introduced, although CMOS had the potential to run on much lower voltages than TTL.

Over the past ten years, supply-voltage reduction has been a major differentiator in the competitive field of digital integrated circuits; driven by demand for faster and smaller devices which dissipate less power. CMOS transistor geometries have shrunk enormously since the early 1980s. This decrease in size has allowed many more CMOS transistors to be fabricated in the same space. But as size of the active elements has diminished, so have the insulation regions between them. There is thereby the necessity to keep potentially damaging leakage-currents under control by reducing operating voltage. This is illustrated in current FPGAs, microprocessors, and DSPs, where the optimum core operating voltage may be as low as 1 V or even less.

Circuits with memory

So far, we have looked at, so called, *combinational logic*. But digital electronics' power rests particularly on its ability to store and to recall, not only data, but also program instructions. It is this capacity of digital circuits which differentiates it from - and accounts for its superiority to - its analogue antecedents.

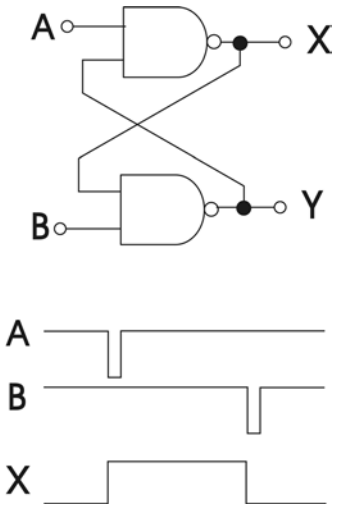

Figure 22 – A basic memory device

The basic memory device is formed from two simple gates as illustrated in Figure 22. Assume that inputs A and B are both TRUE (HI), look at the top gate and imagine that the other input (equivalent to output Y) is also HI. Because the gate is a NAND, the output X will be LO. The bottom gate will therefore be in the following condition, input B will be HI, its other input (effectively X) will be LO and its output (Y) will therefore be HI; which is what we imagined. In other words, this circuit will sit happily in this condition; it is said to be a *stable* state.

Because the circuit is symmetrical, it should be pretty obvious that the circuit will just as happily sit in the alternative condition where X is HI and Y is LO. (If it's not, work through the circuit with the NAND truth-table.) This arrangement of gates is stable in two distinct states and is known as a *bi-stable*. Another - more colourful name is *flip-flop*. This circuit element is the basis for all digital electronic memory.

To set the state of the memory it's necessary momentarily to pull either the A or B input LO. The action of these two inputs is illustrated in Figure 22 too. Note that pulling A LO causes the flip-flop to go into one of its possible states and input B causes it to go to the other. For this reason inputs A and B are usually termed the *RESET/SET* inputs and this

type of bi-stable is called as *RESET-SET (or R-S) flip-flop*. Note that in the basic NAND flip-flop circuit in Figure 22, if both inputs go to 0, both outputs go to 1. This condition should be avoided in normal operation.

Referring to Figure 22, because the input A "sets" the ***Q*** output to 1 (TRUE), it is labelled ***/S*** (the bar is there to indicate a low-state to set). Similarly the ***/R*** for *reset*. The output ***/Q*** is the logical complement of the ***Q*** output. The addition of further gates enhances the bi-stable so that it may be forced one or other sate by the action of a single data (***D***) input, the logical state (0 or 1) of this input being gated by a further clock pulse (as illustrated in Figure 23).

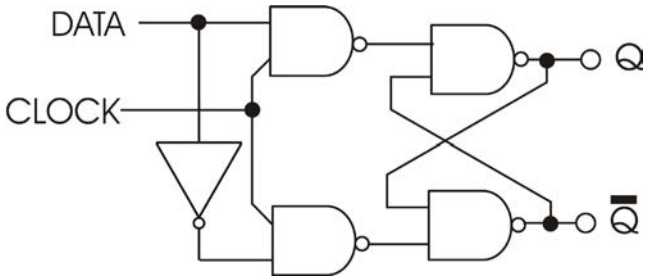

Figure 23 – D-type bi-stable

This type of flip-flop is called a *D-type bi-stable*, or more often, a *latch* because data is "latched into" the memory element. The addition of extra gates still can modify the action of the flip-flop so that it operates only on the rising (or falling) edge of the clock. This is termed an *edge-triggered, D-Type flip-flop*, a fantastically important and ubiquitous circuit element.

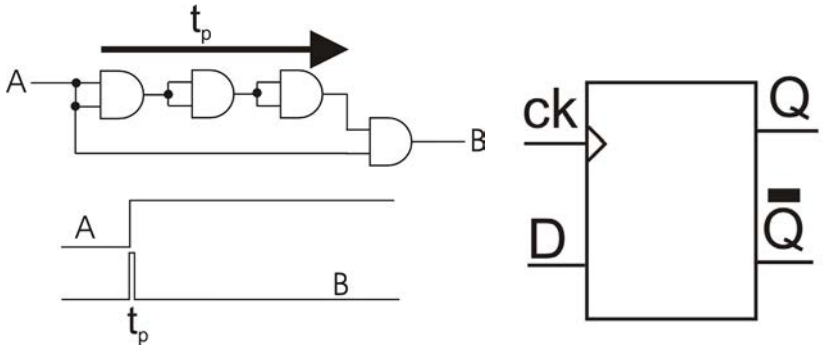

Figure 24 – Illustrates how an "edge" signal may be generated using a series of gates as shown (left). The addition of this edge triggering to the circuit of Figure 23 produces the edge-triggered, D-type flip-flop (right)

Figure 24 illustrates how an edge may be generated. Note that the length of the pulse is the combined propagation delay of the gates (***tp***). The advantage of using the clock edge to determine the sampling instant of the flip-flop is that it gives much greater accuracy as to the exact instant that the data value is sampled on the data line. Flip-flops are probably the most common circuit in modern digital electronics as program and data memory is formed from bi-stable elements. *SRAM* (static random-access-memory) chips are an array of bi-stable elements like this and their associated selection (address) lines[4]. Arrays of edge-triggered, D-type flip-flops are commonly packaged together in monolithic integrated circuits so that (for example) 8-bit words may be sampled and stored at a precise input from a dynamically changing data-bus, in which they are termed *n-bit*

[4] *DRAM* or *dynamic-RAM* is different; in this case a voltage is stored on the plates of a tiny capacitor (formed from a MOS semiconductor). DRAM needs arrangements so that these capacitors are refreshed as the charge decays on the capacitor plates due to the inevitable leakage resistances; the values being re-written back into their locations; hence the term *dynamic*.

latches or *registers*. Serial strings of edge-triggered, D-type flip-flops are arranged in IC packages as *shift-registers* and *counters*; circuit elements which are explained below.

Digital-machine based representation of numbers

So far, we have seen how we can solve simple true/false logic problems using logic gates. And we've seen how the results of the various decisions could be "remembered" or stored in bi-stable elements. But suppose we want to calculate problems with numerical magnitudes: we need a way of representing numbers within our digital machines.

Binary arithmetic

Probably because we have ten fingers, our arithmetic system is based on base ten. Remember that the position of each of our ten symbols, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 indicates to which power of ten the symbol was raised such that,

$$\mathbf{5467 = (5 \times 10^3) + (4 \times 10^2) + (6 \times 10^1) + (7 \times 10^0)}$$

We say the 5467 is a number expressed to the *base ten*. The same principle may be applied to other number bases. For example we might decide we would like to adopt a number system based on the number eight (sometimes called *octal*). In this system we would count like this,

1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22.....

This looks really strange! But remember in base-8 arithmetic, the symbol 10 means,

$$\mathbf{(1 \times 8^1) + (0 \times 8^0) = 8 + 0 = 8 \text{ in base ten.}}$$

In which case, the system makes sense. In everyday use, we don't bother to define the number base in which we are working because it is virtually always base 10. However, where we need to indicate the base in which we are working, it is usual to append a suffix to the numbers in the following manner:

$$\mathbf{10_8 = 8_{10}}$$

This changing of number bases is of rather abstract interest, except in electronics, where the number system to the *base 2* or the *binary system* is of fundamental importance. In our everyday base-ten system, we need ten symbols to express every number: in base eight, we only need eight symbols: in binary, we only need two symbols. If we count in binary it looks like this,

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011

This looks pretty wasteful to our human minds but this system is excellent for machines that need to remember and calculate numbers because (as we have seen) it is very easy to fabricate electronic circuits which only exist in two states. The binary system demands a lot of numbers to express limited quantities, for example $1000_{10} = 1111101000_2$, but it only needs very simple equipment to store and process these numbers.

Hexadecimal base

If we chose an arithmetic in a base greater than ten, for example to the base sixteen, then we have the slight complication that we run out of symbols before we get to $10_{16} = 16$. The usual convention is to employ the alphabet in these circumstances, so that counting in base sixteen, or the *hexadecimal* base looks like this,

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, ... etc.

The hexadecimal base is very widely used in digital electronics because of the phenomenon that an eight bit byte of binary numbers map to hexadecimal numbers in a very neat way, so that the lower and upper 4-byte parts of the eight-bit number