

Chapter 1 - Logic

Digital design inevitably starts with a discussion of *logic*, that is, the stuff that digital electronics is made of. It is pretty unlikely that anyone interested in programmable logic doesn't already know something about logic. Hopefully this will fill in a few cracks. The mathematics of logic is called Boolean algebra. That will be considered in the next chapter. Here we are concerned with the basic logic functions and how they are implemented electronically.

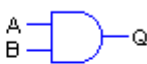
Basic Logic Functions

The basic logic functions are the AND function, the OR function and the invert function. To create the electronic equivalent of ANY Boolean function you need inversion and either AND or OR. Having both makes it easier but it's not essential. Inversion is essential.

For our definitions we'll consider functions of two Boolean input variables, A and B. The AND function can be stated in words as follows. The AND is true if A is true AND if B is true. That is, for the function to be true both inputs (all inputs if there are more) must be true. What does 'true' mean? For conventional positive logic we say a variable is true if it is 'one' and false if it is 'zero'. But what does 'one' or 'zero' mean? Again, for positive (electronic) logic, a 'one' is a voltage close to the positive supply and a 'zero' is a voltage close to ground. The range of values that a one or zero can take is a function of the particular logic family. Keep in mind that the functionality of a logic device is defined for positive logic. You are free to consider it to be negative logic and the functionality will be different. More about that later.

Figure 1-1 shows the traditional symbol for the AND gate with two inputs A and B and an output Q, and a *truth table*. A truth table is a tabular listing showing an output (or outputs if more than one) as a function of the inputs. The input rows usually represent all of the possible combinations of N inputs which is 2^N . In this case, N is two and there are 2^2 or four different combinations.

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




$Q = A * B$

Figure 1-1 The AND Function

The AND function is represented as a product of variables. In a 'product' of several variables in the Euclidian algebraic sense, if any of the terms is zero, the result is zero. The same is true in an AND expression. The traditional product operator is a dot spaced halfway up vertically. Since we have no such symbol on our keyboards, a popular character for representing multiplication, just as some high-level languages, is the asterisk. Thus, we write $Q = A * B$ which reads Q is equal to A AND B. In Euclidian algebraic notation, the product operator is usually dropped so AB reads as A times B and 8A reads as eight times A. The same is true in logic. So it is common to write $Q = AB$ to represent an AND function. (Some authors abhor this. In compiled logic languages, variables can have names of more than one character, so AB could, in fact, be a single name. I do not share this view. I will use variables of a single literal.) The ABEL language uses the '&' symbol for AND. To me, A&B&C looks terrible. A*B*C looks much better. ABEL permits, however, the alternative '*' for product if enabled.

The OR function is shown in Figure 1-2. The output Q is true if A is true OR if B is true. That is, one or the other must be true. (This also includes them both being true). This function is also called inclusive OR since A and B can both be true. For more inputs, the OR is true if one (or more) inputs are true. The OR operator is traditionally a plus sign, and the resulting expression is called a sum. Clearly, this is not a sum in the algebraic sense, but it still has some properties in common. In ABEL, the sum operator is '#'.

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



$Q = A + B$

Figure 1-2 The OR Function

Thus, $Q = A * B + C * D$ would be written
 $Q = A \& B \# C \& D$ Maybe you'll get used to it.

The invert function is shown in Figure 1-3. The symbol is a triangle with a circle at the output. The triangle itself is used to represent a buffer, a circuit element that does not change the signal voltage, but allows it more drive capability without degrading the symbol. So an inverter looks like a buffer with a circle attached. The circle can also be read as “low-true”. Thus, if the input is true, that is high, the output is true, but low since the output is low true. However you choose to look at it, a one or high at the input produces a low or zero at the output and vice versa. One traditional way to represent inverted variables is to put a bar over the name. The variable, for example A, would be called A-bar. The problem with bar notation, however, is that our keyboards have no over bar. Even if the word processor would allow over striking, there is no suitable over bar symbol. Another alternative favored by most logic texts is to indicate an inverted variable with a trailing apostrophe. Thus, an inverted A becomes A', called A-prime. The PAL assembler, called PALASM, used a leading slash to indicate a primed variable, for example, /A. ABEL uses an exclamation point, thus !A. Whatever notation is used, it is probably best to call the variable ‘prime’.

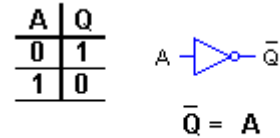


Figure 1-3 The Invert Function

Exclusive OR

AND and OR are the basic logic functions. A third function, not a basic one, is Exclusive OR, usually denoted as XOR. The exclusive OR is true if A is true (but not B) or if B is true, but not A. In terms of the basic functions, we write A XOR B equals $A*B + /A*B$, (read A and not B or B and not A). The traditional symbol for XOR is a circled plus sign. While this is manageable in typeset texts, there is no convenient way to make this symbol with a PC keyboard. We will elect, therefore, to use the ABEL symbol, \$. Figure 1-4 show the exclusive OR function. Is the exclusive OR defined for a function of three inputs or more? The answer is yes. Note, that the symbol for the XOR gate is the same shape as the OR gate, but has an extra curved line at the input end of the symbol. You might guess that for three inputs A, B and C, the Exclusive OR is true if A is true but not B or C, etc. and you would almost be correct. But the XOR of three variables includes $A*B*C*$. In fact, the XOR of any number of inputs is true if the number of inputs that are one is an odd number.

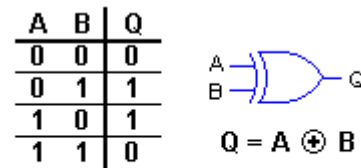


Figure 1-4 The Exclusive OR Function

Low-true Logic Functions

It is perfectly reasonable to make a logic gate whose output is low true. The gates of some logic families, maybe most logic families, are inherently low-true outputs. Shortly, we will discuss logic families and their origins and show that this is true. A gate with a low-true output is also more versatile. An AND gate with a low-true output is called a NAND (for negative AND). An AND gate can only make AND gates, but by using a second NAND to perform inversion, two NAND gates can make an AND gate. So aside from being easier to make, inverting gates are more versatile. Momentarily we will learn about the duality of the AND and OR functions. Because of this, an inverting function like NAND (or NOR) is capable of implementing any logic function!

An OR with a low-true output is called a NOR.

An XOR with a low-true output is called an Exclusive NOR or XNOR. Consider that the output of an XNOR is true for $A*B + /A*/B$. That is, XNOR is true if both inputs are the same. This is a ‘compare’ function. Thus to make a function that compares two binary words to see if they are equal, each bit in the word is compared with an XNOR gate and the XNOR outputs combined in a large AND (or NAND if a low-true output is desired). If one of the binary words being compared is low-true, then XOR gates should be used. Figure 1-5 shows the inverting NAND, NOR, and XNOR functions.

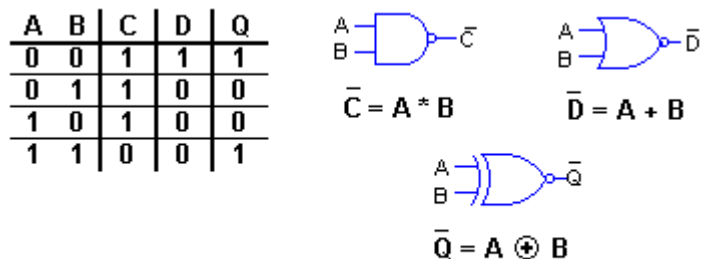


Figure 1-5 NAND, NOR and Exclusive NOR Function

Duality

Consider the circuit of Figure 1-6a. Three switches are run to three inputs of a NAND gate. With no switch depressed, the pull-up resistors cause all three inputs to be high, the AND condition realized, and the output to be true (low). When any input is switched, the AND condition is taken away and the output goes false (high). In other words, the output is low if no switches are pressed, but goes high when any switch is pressed (input goes low). This is an OR function! The output goes high if S1 OR S2 OR S3 is pressed. The figure can be redrawn as in Figure 1-6b as an OR gate with a high-true output and three low-true inputs. This is the very

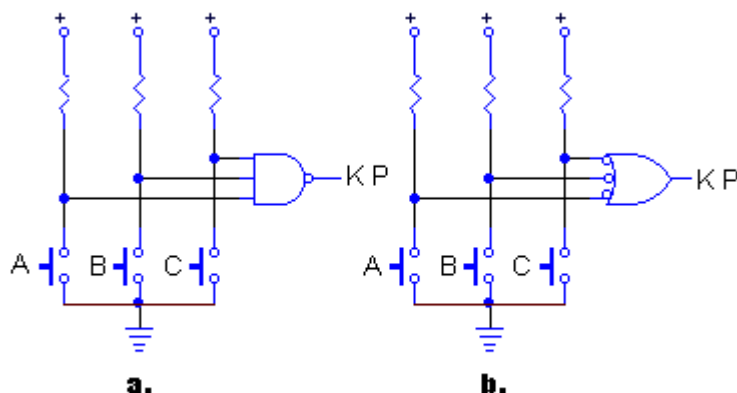


Figure 1-6 Duality of AND and OR

same gate, but just redrawn to better convey its usage in the circuit. This illustrates a powerful feature, duality, of the AND and OR functions. An AND function can be interpreted as an OR or vice versa. This shows how a single inverting function such as NAND can provide ANY logic function since it provides an AND function, inversion, and because of the duality principle, an OR function as well.

If you use a standard logic gate such as a 74X00 which is a quad 2-input NAND gate, it should be drawn as a NAND function on the schematic only if it actually provides an AND function. Any gates that function as an OR should be drawn as such. Figure 1-7 shows the duals of several gates. Here is the rule: to form the dual of a gate, change the symbol from an AND to an OR (or an OR to an AND) and complement the sense of all inputs and outputs. Thus, the NAND in Figure 1-7a changes to an OR function, the low-true output is made high-true, and the high-true inputs are made low-true.

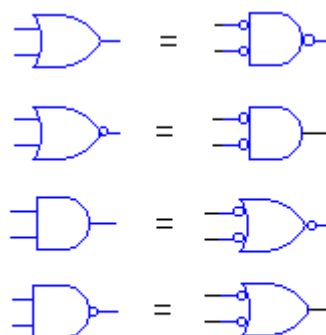


Figure 1-7 Common Logic Gates and their Duals

Combining Outputs

The output of a logic gate can go to many inputs, but it is not permissible ordinarily to connect outputs of different gates together. For example, suppose the outputs of two ANDs were connected together to make an AND function with a larger number of inputs. If the inputs on one of the ANDs indicated a false or low condition and the inputs on the other indicated a true or high condition, the two outputs would be fighting or opposing each other. Large currents would flow. Even if the gates weren't destroyed, the voltage level of the output would be invalid and probably not convey the intended output.

The availability of TTL was probably a major factor in the beginning of the computer revolution. The first affordable computers were the so-called minicomputers appearing in the early sixties. The data bus in a computer system is a situation where any number of devices need to be able to drive the bus (and be physically connected together), but somehow without violating the prohibition of directly connecting outputs together. How can this be done?

Early computers used logic gates for driving buses whose outputs could pull only in one direction, specifically down. A pull-up resistor made the output go high when the active device 'let go' and no longer pulled low. Consider connecting two such devices together. (Look ahead to Figure 1-9, the RTL NOR gate). If neither pulls low the resistor pulls the output high. If either pulls low, the output goes low. Since the other gate is not pulling high, but only letting go, it is not affected by the gate that is pulling low. Traditionally an NPN bipolar transistor is the active low pulling device. Nothing internally in the gate can pull high. The output is said to be 'open-collector'.

In the early days of digital logic chips, open-collector gates were common and popular. They still have some uses. This connection is commonly called a wired-OR, although a more appropriate name would be wired-NOR.

While this solution worked, it had a major shortcoming, namely speed. To get the highest speed, the pull-up resistor should be as small as possible. But a small resistor dissipates significant power. This required a solution that had an active pull-up like a regular TTL output, but one that could be disabled. So a new kind of output stage was developed that was able to pull either high or low when enabled, but when disabled pulls neither high or low but is very high impedance. Such a device is said to be a three-state device, the three states being high, low and disabled or high impedance. Typically, the outputs of a large number of three-state devices can be tied together provided that never more than one device is enabled at any one time. Figure 1-8 shows a symbol for a three-state buffer. Note that there is an input, an output and a control input. The control input is indicated as low-true. Thus, when the control input is low, the gate functions as a normal gate with the output actively pulling high or low according to the input. When the control input is high, however, the output is high-impedance, or effectively internally disconnected from the output pin.

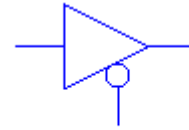


Figure 1-8 Three-state Gate

Later when we discuss the logic family known as CMOS we'll talk about another kind of device similar to a three-state gate, but whose output is physically (internally) connected to the input when enabled.

Logic Families

One of the most significant developments in electronics was the invention of the integrated circuit. It was not a planned event. No one set out to build one. The idea simply came to an engineer. The company where this happened was called Texas Instruments. This company made transistors. This was the beginning of the sixties. Transistors are made on a circular wafer of relatively pure silicon that is cut from a cylindrical rod. The rods were about an inch in diameter. The transistors were created photographically on the wafer by stepping a negative over a row/column matrix. The chips were built up with a diffusion process in layers. Eventually the wafer was cut or broken into rectangular chips that were tested and mounted on headers and encapsulated to become discrete transistors. An engineer by the name of Jack Kilby got the bright idea of adding a few more layers to the process by which neighboring transistors could be connected and resistors added to make a functional circuit. The first such circuit is shown in figure 1-9, and has two transistors and three resistors. This forms a logic gate. Let us analyze the circuit.

Consider only transistor Q1, input resistor Ra and pull-up resistor Rc. A positive voltage, Vcc is connected to the pull-up resistor. Let us assume this is +5 volts. Now let's apply a high voltage, a one, (+5Volts) to input A. This causes transistor Q1 to turn on hard and become a low-impedance switch to ground. The output at the collector is now close to ground or a logic zero. Now let us connect input A to an open switch and the output is connected to the +5V rail through the pull-up. The output is high or a one. Thus, a high input gives a low output and a low input gives a high output. This is precisely a logic inverter.

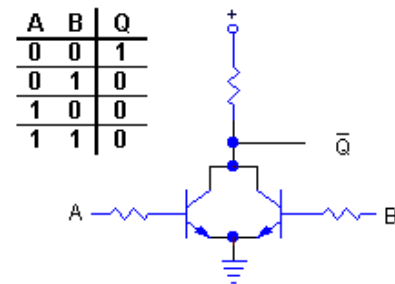


Figure 1-9 Kilby RTL NOR Gate

Now consider adding the transistor Q2. It works exactly the same way. But now the output can be high only if both Q1 and Q2 are off which can occur when both inputs A and B are low. If A or B is high, the output is low. This is a two-input NOR gate. So the very first integrated circuit was a digital one and specifically a logic NOR gate. The implications of this were enormous and also acknowledged by T.I. The idea was patented and T.I. collected millions in royalties, a penny or perhaps a fraction of a penny a chip, from every manufacturer of integrated circuits.

From this first integrated circuit arose the RTL family of logic circuits (Resistor Transistor Logic) based on the NOR gate. The next improvement was to add multiple diodes to the base of the transistor to permit multiple inputs with a single transistor. This gave rise to a new logic family called DTL for Diode Transistor Logic. The DTL family was short-lived. The diodes were incorporated into the base-emitter junction giving rise to a strange looking

NPN transistor with multiple emitters. This logic family was called TTL for Transistor Transistor Logic. The original family was called the 7400 series. The 7400 was a quad 2-input NAND gate. Unlike RTL and DTL, TTL has had a very long life. 7400 series parts are probably still made. At least they still can be bought. But the important thing about TTL was that it evolved to meet the needs of the users. For example, some users needed faster speed, so resistor values were lowered to increase currents and decrease internal delays. The result was the 74H00 family. The H was for high speed. Unfortunately, it also stood for high power and never became popular. For those who didn't need megahertz speed, another family, the 74L00, used higher value resistors. This gave rise to lower power and the L stood for low power. Neither family enjoyed much popularity or a long life.

One of the factors limiting speed is the fact that the transistors in TTL saturate. A saturated transistor stays on for a while after the drive is removed due to stored charge in the base-emitter diode. To prevent saturation a Schottky diode was connected reversed biased between the collector and base. As the transistor turns on, the collector approaches ground and becomes lower in voltage than the base. This causes the Schottky diode to conduct and shunt base current around the transistor, preventing the transistor from turning on. The transistor plus Schottky diode is called a Schottky transistor. The 74S00 family achieved a significant speed improvement due to lower value resistors, but mostly because of the use of Schottky transistors. The majority of TTL users were satisfied with TTL speed, but desired lower power. The solution was to use higher valued resistors giving rise to lower power, but getting back the speed lost by using Schottky transistors. The result was the 74LS00, low-powered Schottky family. The timing in the introduction of LS was fortuitous. Microprocessors, made with high impedance, low current MOS transistors could typically drive a single TTL load, but five LS loads. LS was immediately popular and still finds use in a lot of designs.

Figure 1-10 shows a two input NAND gate from the original 7400 and also a gate from the 74LS00. It is important to note and understand TTL characteristics. First, let us review the RTL gate. The input is applied to the base of NPN transistor. A positive voltage or one will turn the transistor on. The input is relatively high impedance and a small current is drawn. When ground, or a zero, is applied, the transistor is off and no current is drawn.

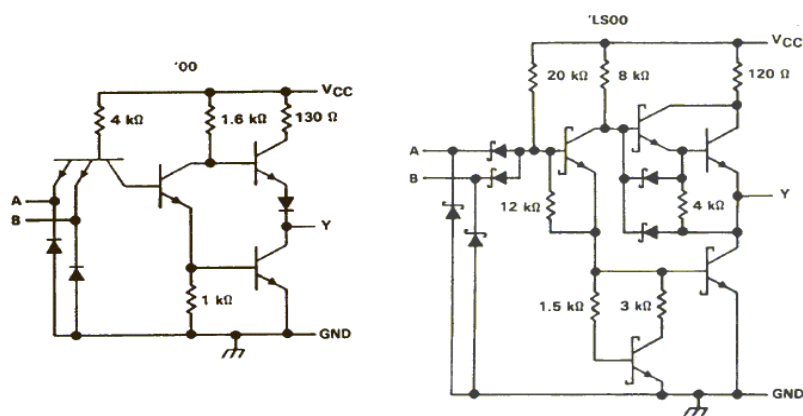


Figure 1-10 TTL NAND (7400) and LSTTL NAND (74LS00)

In contrast, a TTL input is applied to the emitter of the input transistor. This is rather strange. A significant current is necessary to turn this transistor on. For standard TTL this is 1.6 mA. An input to the base would have been 10 to 100 times smaller. Why did they do this? First, TTL was an improvement to DTL. Apparently, while the input diodes were better than having to use a separate transistor per input as in RTL, it still took some trouble and real estate. The diodes were effectively moved to the emitter by using the multiple emitters. This also provided a speed improvement which was highly desirable. But what about the high input current? As long as the input was driven by TTL outputs, that was no problem. The TTL output used an NPN switch to get a low voltage output. This switch could easily sink the current for the ten loads for which it was guaranteed to be within voltage spec.

While the TTL low output voltage is close to the ground rail, the high voltage is not close to the +5 rail. The TTL high minimum is 2.4V (with the maximum rated current of .4mA), but typically about 3.4V. (With a pull-up resistor you can make it +5 volts!) A TTL input needs a minimum of 2.0V to be recognized as a high or one. A low or zero can be as high as .8V. A low output is guaranteed to be a maximum of .4V at the full rated current of 16 mA.

TTL's rather strange I/O characteristics might have been seriously challenged with the appearance of MOSFET (Metal Oxide Semiconductor Field Effect Transistor, hereafter called just MOS) microprocessors which can sink about 2 mA or just a little more than a single TTL load, but 74LS was already available and became very popular precisely because a MOS output could drive typically five LS inputs.

Curiously, no logic families appeared using the technology of the early MOS microprocessors themselves. Microprocessors were possible because MOS technology was very high impedance and used little power, making it possible to put thousands of transistors in a small area. Still the MOS took some power and there were serious limits to speed because of power limitations. Then came another invention.

CMOS

MOS processors were built initially with PMOS, p-channel MOSFET transistors. Later, microprocessors used NMOS which could operate on a single 5 volt supply and was more or less TTL compatible. An NMOS transistor can be compared with an NPN bipolar transistor. The difference is that the bipolar transistor is turned on with a current and the NMOS transistor by a voltage with no current being drawn.

Take another look at the RTL NOR gate in Figure 1-9. Remove one of the transistors and we have an RTL inverter. All logic is based on the inverting action of a transistor. Note that when the transistor is on, the load resistor R_L is connected between +5 and ground. If R_L is 5K, for example, then 1 mA will be drawn, independent of the output being connected to any inputs. In fact, since RTL inputs draw no current for zero volts in, this is the worst case current. For a high output, current is drawn through R_L , but it will always be less than 1 mA. (Why?) So a characteristic of most logic families is that they dissipate power independent of any loads. On the average, half of all inverting stages in a logic circuit will be on at any time.

Bipolar logic has always been NPN. Building high gain PNP transistors together with NPN transistors appears to be difficult and it was never done. Similarly, it did not seem feasible to put PMOS and NMOS transistors on the same chip. But one company managed to do this quite successfully. The company was RCA and they called their invention COSMOS for Complementary Symmetry MOS. Later vendors shortened this to CMOS. A CMOS inverter is shown in Figure 1-11. Note, that a CMOS inverter consists of an NMOS transistor that can switch to the negative or ground rail, and a PMOS transistor that can switch to the positive rail.

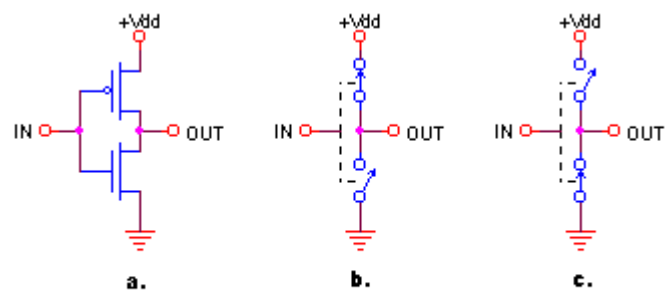


Figure 1-11 CMOS Inverter

Note, there are no resistors, only transistors. No input resistor is necessary since the transistors are voltage controlled and take no current. No output resistor is needed because each transistor serves as the load for the other. Consider a low voltage applied to the inverter. This voltage holds the NMOS transistor off and it looks like an open switch. The low voltage, however, turns the PMOS transistor on and it looks like a closed switch. Figure 1-11 b. shows the equivalent. Since the upper switch is on, the output is connected to the positive rail.

Now consider an input that is a high voltage (+V). The NMOS transistor is on and looks like a closed switch. The PMOS transistor is off and looks like a closed switch. This is shown in Figure 1-11c. This time, the output is connected to the ground rail. Clearly this is acting as an inverter since a low or zero in produces a high or +V out and a high or +V in produces a low or ground out. Note, however, that regardless of whether the output is high or low, there is no internal path between +V and ground and the gate dissipates no power. Of course the gate could deliver current through the closed switch to a load and that would dissipate power. Typically the on resistance of the NMOS or PMOS transistor is on the order of a couple of hundred ohms. On the other hand, if the loads are just other CMOS inputs then no current flows since the inputs are essentially infinite input impedance. Thus, it would appear that CMOS has the wonderful property that it dissipates no power! For a DC situation, this is essentially true. However, as a CMOS inverter changes state, the on transistor goes off and the off transistor goes on. This does not take place instantaneously. For a very short time both transistors are partially on and there is a direct path from +V to ground through a few hundred ohms.

There is a second component of power dissipation. Even though a CMOS input has infinite impedance, it does have a capacitance associated with it. Apply a voltage through a closed switch to an input amounts to charging a capacitor through a resistor. As soon as the capacitor is charged, the current goes to zero, but initially, a significant current

flows through the closed switch to the capacitor which looks like a short circuit. Thus, there is a pulse of current as an inverter pair changes state. The more frequently the state change, the more frequent the current pulse. Thus, both components of power dissipation in CMOS are proportional to frequency. The power dissipated by a CMOS circuit as a function of frequency is a straight line ramping up from zero. In summary, CMOS uses very little power when used in low frequency or DC applications, but uses an increasing amount of power as the frequency increases.

CMOS transistors can be made with modest voltage ratings of 15V but will still work fine down to voltages of two or three volts. Thus, it is practical to power a low frequency application with a pair of 1.5V batteries. The higher the voltage, the lower the on resistances, thus CMOS speed performance increases with an increasing supply voltage. Also for CMOS operating at a relatively high voltage such as 15 volts, the voltages needed to switch logic are correspondingly high and CMOS logic operating at higher voltage is much less sensitive to noise.

CMOS logic functions are naturally inverting, based on an inverter. AND and OR functions are made with serial and parallel combinations of PMOS transistors for the top transistor and of NMOS transistors for the bottom transistor. Non-inverting gates are obtained by an additional inverting stage.

Consider a two-input NAND function. The NMOS switch is replaced by two NMOS switches in series. The PMOS switch is replaced by two PMOS switches in parallel. Only when both inputs are high is the NMOS path to ground closed. If either input is low, the PMOS path to +V is closed. The original CMOS family, the 4000 family (RCA CD4000), had various output characteristics based on the logic gate being implemented. The 2-input NAND is shown in Figure 1-12.

It should be observed for the NAND gate just described, the path from the output to plus is either a single PMOS transistor, or two PMOS transistors in parallel, whereas the output path to ground is through two NMOS transistors in series. The on resistances of the two NMOS transistors will add. A four-input NAND would have four transistors in series. This is not too important when driving CMOS inputs but affects the ability to drive resistive loads. In order to obtain more uniform drive results, the 4000B family employs a buffer output stage. The price for this is additional delay.

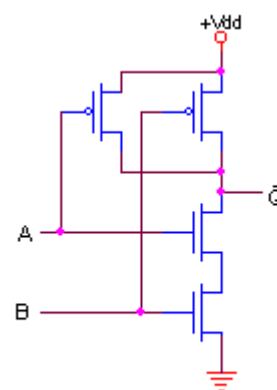


Figure 1-12 CMOS NAND

CMOS Families

Initially, TTL and other bipolar logic types were used where high speed was a requirement. CMOS was used where high speed wasn't required but low power and/or noise immunity were. An early CMOS family was the 74C00 family. The attraction of this family is that it had the same functionality and chip pinouts as the TTL 7400 families and also ran with a 5-volt supply. It was quite low power, but it also did not compare in speed with TTL.

The next development was 7400 CMOS that was just as fast as LS TTL. This family was called 74HC00, the HC standing for high-speed CMOS. This has proven immensely popular. One small problem is that when interfacing TTL to HC, the TTL output voltages are not high enough. This gave rise to the 74HCT00 family whose inputs can accept TTL outputs. Actually TTL can be applied to HC CMOS provided a pull-up resistor is used.

The trends in digital equipment are higher speed, lower power and lower voltage. This has given rise to a number of families among them 74AC, 74ACT, 74ALVHC, 74FACT, 74LCX, 74LVC, 74LVCT, 74LVT, 74LVX and 74VHC.

Now some remarkable advances in the processing technology has made it possible to build CMOS processors that can run with a 1 GHz (and faster) clock! By the same token CMOS has taken over programmable logic with higher speed performance and lower power than earlier bipolar types.

Transmission Gates

Before leaving CMOS we need to mention a structure that is unique to CMOS called the transmission gate. A transmission gate is an NMOS and PMOS transistor back to back. The controlling signal is connected to the gate of

one and the invert is connected to the gate of the other. Thus, either both are on or both are off. When both are off, the resistance between input and output or output and input is very high. When both are on, the resistance is a couple of hundred ohms. Figure 1-13 shows a transmission gate. Note that the path between the input and output is simply a resistance, so that there is really no distinction between input and output. Also note, that whatever voltage is applied to the input (between 0 and +V) the same voltage appears at the output. The transmission gate does not change the voltage in any way, and in fact, is suitable for passing (or blocking) analog voltages. Thus, it is an analog switch as well as a digital switch. It is common to use the transmission gate in the design of flip-flops.

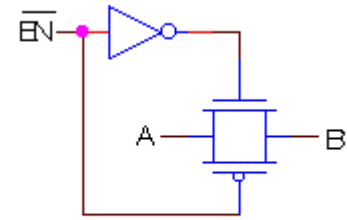


Figure 1-13 CMOS Transmission Gate

The original transmission gate chip is the 4016 which has four transmission gates each with its own control. One with improved characteristics is the 4066. The sequence 4051, 4052 and 4053 are analog switch multiplexor/demultiplexors. The 4051 is an 8:1 select gate. The 4052 is a dual 4:1 select gate, and the 4053 is a triple 2:1 select gate. Again, note that the inputs and output are interchangeable so a 4051 might be used to switch a signal to one of eight inputs.

Additional information is available on the Internet by looking at the data sheets of individual CMOS parts.