DIGITAL ELECTRONICS FOR BROADCASTERS

BY JOHN E. CUNNINGHAM

No. 1260
$14.95

Digital Electronics For Broadcasters

by John E. Cunningham

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

FIRST EDITION

FIRST PRINTING

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Cunningham, John Edward, 1923-
Digital electronics for broadcasters.

Includes index.
1. Digital electronics. 2. Radio—Apparatus and supplies. 3. Television—Apparatus supplies.
I. Title.
TK7868.D5C86 621.3815 80-20587
ISBN 0-8306-9705-5

Contents

Preface

In recent years there has been an increasing amount of comment and concern in management and educational circles about technical personnel becoming obsolete. The expression "engineering obsolescence" is generally taken to mean that certain engineers and technicians are not keeping up with the rapid changes that are taking place in all branches of technology.

Alarmingly, these comments are not being directed exclusively to older people who are nearing retirement. It is not uncommon to hear managers complain that engineers who have been out of school not more than 10 or 15 years and who should be in the prime of their careers, are no longer able to cope with the latest technological developments. Although technical people themselves rarely talk about it, they are fully as concerned as management that they may be considered no longer technically useful.

One solution to this problem is to promote the older engineers to management positions and turn the technically exacting tasks over to recent graduates. This approach has two rather serious limitations. In the first place, many engineers have neither the interest nor the aptitude to handle management functions. Secondly, and more important, the recent graduate rarely has the ingredient known as engineering judgment which seems to be best acquired through experience.

I have long claimed that no engineer worth his salt has ever had any trouble *keeping up* with changes in the state-of-the-art that affect his job. Where the trouble comes is when he suddenly has to *catch up* with a field that he has never used or has forgotten all about.

The sudden appearance of digital electronics in radio and television broadcasting is a good example of this phenomenon. Although most broadcasters have been aware for many years that rapid changes have been occuring in the fields of digital electronics and computers, they have paid little attention, because until quite recently neither of these fields has much to do with broadcasting. Admittedly computers were being used in the office for billing and payroll, but many ancient transmitters differed little from their more modern replacements. In fact, many engineers felt

that even if digital systems did become available for broadcasting, it would be decades before station management would shell out the cash to buy one. After all, didn't it take five years to get the owner to shell out the cash for the new field strength meter?

The result of this was quite naturally that most broadcast engineers didn't bother to learn much about digital electronics, and if they have ever studied the subject, they didn't bother to keep up with the state of the art. Why be concerned with a branch of technology that would probably never affect them?

When digital systems finally charged into the broadcast field, they entered with great speed. First, there were a few digital components in some of the control systems. Just a few integrated circuits, not enough to get the engineer excited. Then came digitally-controlled automation and remote control systems. Finally, the TV people found that the video signal itself is often converted to digital form for processing. Now, digital audio is becoming a reality.

It is no longer enough for the engineers to try to *keep up* with digital electronics. He must *catch up* and he must do it fast. Catching up with any area of technology isn't easy. Most of the available books are either too elementary, or too involved. The texts used in colleges and technical schools are usually not much help. They usually contain far too much time-consuming material that the practicing broadcast engineer doesn't need, and not enough of the practical information that he needs desperately.

This book is an attempt to correct the situation. You don't need any familiarity with digital electronics to understand the material, but it is assumed that you are familiar with broadcasting. All of the functional elements are covered and most of the examples are taken from broadcasting so that you will feel that you are on familiar ground.

Much of the material is presented in an original way and much has been suggested by my home study students. Many suggestions were made by engineers who attended the various seminars that I conducted on the subject of digital electronics in broadcasting.

I would like to acknowledge the help of my associates at Cleveland Institute of Electronics, particularly Jim Arcaro, Jerry Casebeer, Steve Simcic, and Ron Zeldman. Their help and suggestions were invaluable. Much of the work was done in connection with digital seminars sponsored by the National Association of Broadcasters under the direction of George Bartlett who has probably done more to help the technical broadcaster than any other one man. Some of the work was done under the direction of Carl E. Smith who is always a gentleman and unselfishly shares his knowledge with anyone he can help.

I would also like to thank Susan M. Heygi without whose help the book would never have been finished. Last but far from least I would like to thank Grace L. Slavik who has always been an inspiration to all who know her.

John E. Cunningham

Chapter 1
Introduction to the Digital World

To the uninitiated, probably the most confusing aspect of digital electronics is the fact that the term "digital system" seems to embrace many different types of systems that, at least on the surface, seem to have nothing in common. Although a pushbutton control for a video recorder, a remote control system for a transmitter, and a television time-base corrector don't seem to have very much in common, they may all be called digital systems. It is hard to see how these apparently quite different devices can be classified together. So, we will start by defining what we mean by digital system

ANALOG-DIGITAL SIGNAL COMPARISON

The biggest difference between a digital system and an ordinary system is in the nature of the signals that we find inside. Figure 1-1 shows the waveform of a signal that might be found in an ordinary audio amplifier. This signal, at any instant of time, is directly proportional to the sound being amplified. For our purposes here, there are two important characteristics of this type of signal:

☐ It is continuous; that is, it varies smoothly from one value to another. It may have any value between zero and its maximum value.

☐ The amplitude, frequency, and phase of the signal will be the same as the amplitude, frequency, and phase of the sound that is being amplified. The signal might be thought of as an "analog" of the sound. Thus, ordinary signals and systems with which we are

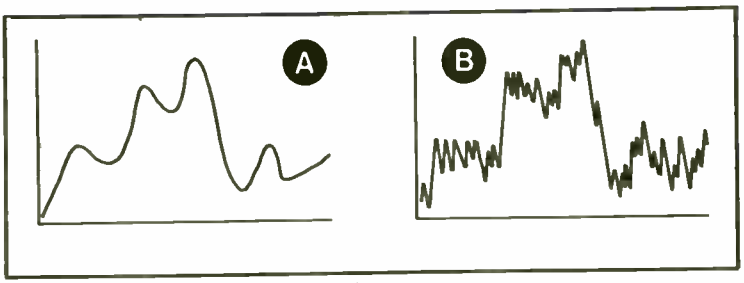


Fig. 1-1. Analog signal corrupted by noise.

all familiar are often called analog signals and systems to distinguish them from their digital counterparts.

Before we look into digital systems, let's take a look at what happens to an ordinary AM analog signal as it passes through a system. We all know only too well that the signal will be subject to both noise and distortion. Figure 1-1B shows what might happen to the signal of Fig. 1-1A under extreme conditions. Here noise and distortion have corrupted the signals so badly that we may never be able to recover it.,

Now let's look at a digital signal. Figure 1-2A shows a typical digital signal. It has two properties that distinguish it from an analog signal:

☐ The signal has only two possible values. At any instant of time, a voltage is either present or not present. When the voltage

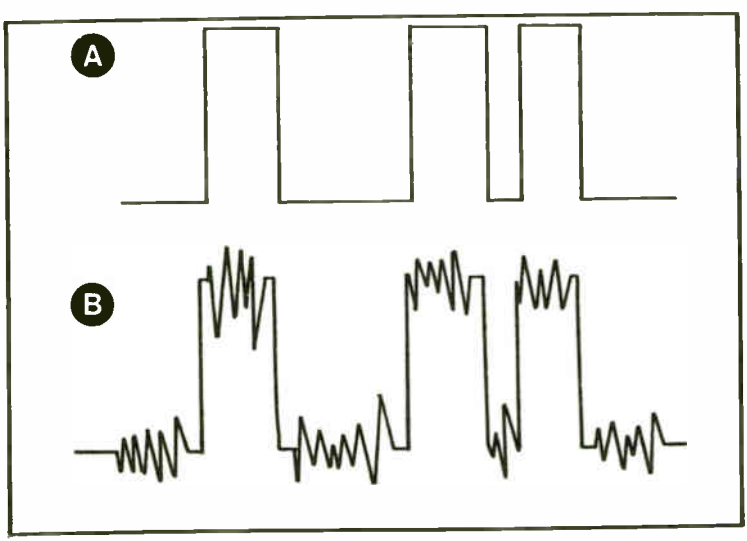


Fig. 1-2. Digital signal corrupted by noise.

is present, we normally refer to the signal as being high; when there is no voltage present, we refer to the signal as being low.

☐ The digital signal bears no resemblance to the waveform of the analog signal it is representing.

Now let's look at what noise and distortion might do to a digital signal. In Figure 1-2B we have subjected our digital signal to the same amount of noise and distortion that we applied to our analog signal in Fig. 1-1B. Here again, the corrupted signal bears little resemblance to the original. In this case, however, by studying the signal we can rather easily tell when the original signal is present. In fact, it would be rather easy to draw a plot of the original signal if all we had to work with was the corrupted signal of Fig. 1-2B. This is one of the important advantages of using digital signals. Up to a limit, all of the effects of noise and distortion can be removed from the signal by rather simple signal processing.

Another advantage of digital signals is that they can be stored for any desired period of time by rather simple circuits. All that our storage element has to do is to record the fact that a signal is present at some instants and absent at others. This is a far cry from what would be involved if we tried to store an analog signal that was varying with time.

Now that we have found what digital systems have in common, let's look at how they might differ from each other. The first distinction that we might make is between what we call simple logic systems and the more complicated systems.

The simplest type of digital system, and probably the easiest one to understand, is a simple logic system. Such a system causes something to happen when and only when certain conditions are either true or false. A good example is the system used to control the plate voltage relay in a transmitter. We want the plate voltage to be applied when and only when:

—the heaters of the tubes have warmed up,

—the plate voltage switch is closed,

—the cabinet is safely closed.

There is nothing magic about such a system. Broadcasters have been using arrangements like this since long before anyone ever thought of the term "digital electronics." One simple way to accomplish the desired function is to wire the plate voltage switch, a time-delay relay, and a safety interlock switch in series with the plate relay as shown in Fig. 1-3A.

In spite of its simplicity, the arrangement of Fig. 1-3A is a logic system and has much in common with logic systems using

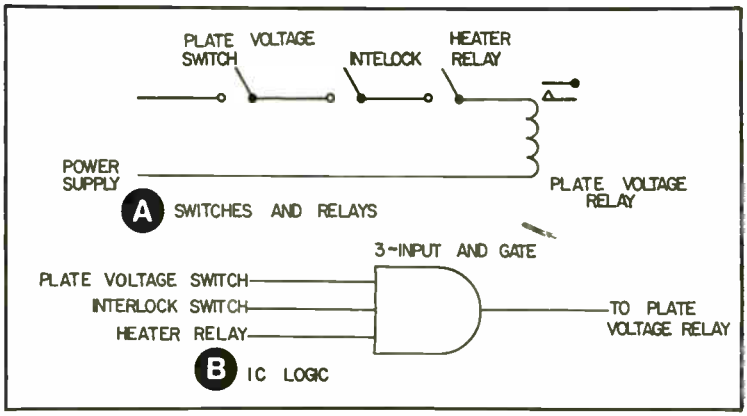


Fig. 1-3. Logic circuit (A) with switches and relays, (B) with an integrated circuit.

digital integrated circuits. One way to implement the system using digital circuit elements is shown in Fig. 1-3B. Here the circuit element is called a 3-input AND gate. We will have more to say about all sorts of logic gates in Chapter 2.

One point of interest in our simple example is that the signals we're working with, in this case switch closures, are really digital in nature. A switch has only two possible positions, either it is closed or it isn't. That is why our system is so simple. Of course, if there were many more switch closures to consider, the system would become more complicated, but its principle of operation would still be simple and easy to understand.

ANALOG-TO-DIGITAL CONVERSION

The next type of system we will consider is one where the things we are interested in are not digital in nature. An example might be the digital voltmeter shown in Fig. 1-4. The voltage we want to measure isn't digital. It is just a plain old DC voltage that might have any value between zero and, say, 10 volts.

The first stage of our digital meter is the same as in any DC meter—a voltage divider that scales the voltage down to a value we can handle conveniently. The next stage is new—brand new. We call it an analog-to-digital converter, or simply an A/D converter. What this stage does is to take our DC voltage and produce some sort of digital signal that uniquely defines the voltage.

After the A/D converter we have a digital signal. We can store it, or process it in any way that we wish. When we get through with it, it's no good unless we can read some sort of instrument to tell what the voltage is. This is handled by the output device. In this case the output device is a series of 7-segment readouts.

Still greater complication finds its way into a system when the signals themselves are digitized. A typical system of this type might be a TV time-base corrector. Again, we need an A/D converter to get the signal into digital form and several digital circuits to process the signal in the way that we wish. When we get through with this signal we can't apply it to a readout device, but rather we must convert it back to analog form so that it can be applied to a TV transmitter. This last conversion is appropriately called a digital-to-analog or simply D/A.

SAMPLING

The first thing we have to do to convert an analog signal into a digital signal is to sample the value of the signal periodically. Of course, the question naturally rises as to how often and how accurately we must take samples of the signal.

Figure 1-5 shows an analog signal that varies between zero and 5 volts. We will assume that the highest frequency component of the signal is 10,000 Hz. That is, if we were to pass the signal through a low-pass filter that cut off sharply at 10,000 Hz, that waveform of the signal wouldn't be changed.

The first question about sampling was as to how often we must sample the signal to capture all of the information in it. Fortunately, this problem has been solved for us so we won't have to get into an involved mathematical investigation of the subject. Workers at Bell Telephone Laboratories have found that we can capture all the information in a signal if we sample it at a rate equal to at least twice the highest frequency component of the signal. Of course, there is

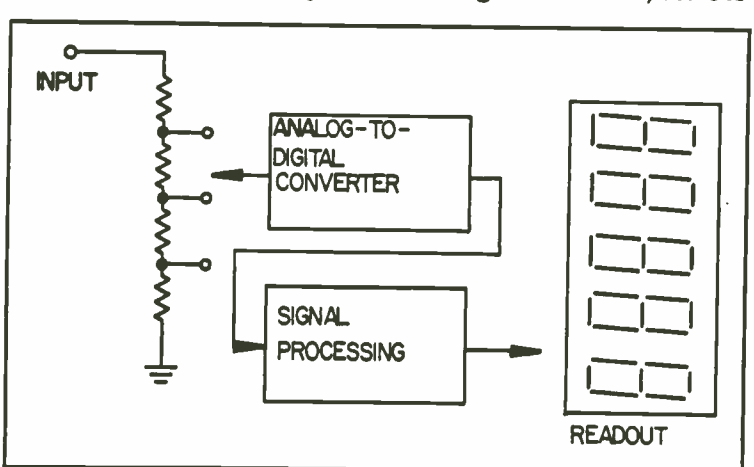


Fig. 1-4. Digital voltmeter block diagram.

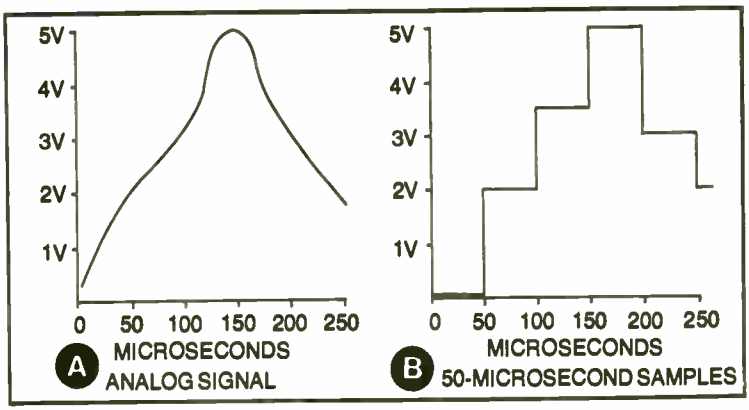


Fig. 1-5. Analog signal with step samples.

no harm in sampling at a higher rate. The limitation is that we can't sample at a lower rate without losing some of the information in the signal. In Fig. 1-5B we show that we're taking a sample of our signal every 50 microseconds. That is, we are sampling the signal at a rate of 20,000 samples per second, which is twice the highest frequency component of the signal.

The next question is how accurate our samples must be if we are to capture enough of the information in the signal so that we can reproduce it faithfully later on. This is a rather involved question, because it involves a subjective judgment of the quality of a signal. We'll have a lot more to say about this later in the book, but right now our main interest is how we can convert our samples into a digital signal, so we will allow ourselves the very generous specification that we'll be happy if the accuracy of the sample is within one-third of a volt. Later we can be concerned with the higher resolution that will be required in real life.

On the basis of what we've said so far, we will be happy with the sample shown in Fig. 1-5B. What we've done is to take the samples of the smooth waveform of Fig. 1-5A and replace it with a series of samples, which for our purposes contain all of the information of interest in the original waveform. The next part of the problem is to find a way to express the values of these samples digitally. This will bring us to the subject of the binary number system, which is the basis of all digital systems.

A LOOK AT NUMBERS IN GENERAL

Before we go rushing off trying to find some binary numbers to represent our samples, let's look at numbers in general. We're so familiar with the decimal system which we use everyday that we

rarely take time to think about what it really means. As a simple example of the use of numbers, let's suppose we counted the number of resistors in a supply cabinet and found there were 174 of them. There are many ways we can record this fact. For example, we could make 174 marks on a piece of paper. This would indeed record the number of resistors, but it would be a very cumbersome system. If anybody wanted to interpret our count, he would have to count off all the little marks that we made. It is much better to simply write the digits 174 on a piece of paper. This is easy to do and the numbers are easy to interpret. The reason that it is easy is that the decimal numbering system is second nature to us.

Let's take a look at just what the number 174 really means. With a little thought, we can see that it means we have:

$$\begin{aligned} 1 \times 100 &= 100 \\ 7 \times 10 &= 70 \\ 4 \times 1 &= 4 \\ \hline & 174 \end{aligned}$$

This is rather obvious, but we can carry the analysis a step further and see just what we mean when we say that the *base* of our decimal number system is ten. We can see that each of the digits in our numbering system corresponds to some power of 10. In fact, the power depends upon the column the digit happens to be in. Thus, we can write the number 174 in the following way, remembering that any number raised to the "0" power is equal to 1, and any number raised to the first power is equal to itself. Now we have:

$$\begin{aligned} 1 \times (10^2 = 100) &= 100 \\ 7 \times (10^1 = 10) &= 70 \\ 4 \times (10^0 = 1) &= 4 \\ \hline & 174 \end{aligned}$$

The fact that each column of a decimal number corresponds to some power of 10 is the essence of the decimal number system. Realizing this, we feel at home when we develop a number system with some base other than 10.

THE BINARY NUMBER SYSTEM

The binary number system is a system that has the base two. Of course, the first question to come up is why anyone would want to fool around with such a crazy numbering system in the first place. Well, there is a good reason that will become more apparent as we go along. To put it simply, in a digital system the signals can

have only one of two possible values at any time. This means that all our equipment has to do is recognize whether or not a voltage is present at any given time. The voltage can vary all over the lot, but as long as we can distinguish between the presence or absence of a voltage, we can reconstruct all of our signals to any desired degree of accuracy. This is such an advantage over the old analog signals, that it is worth the slight inconvenience of learning about a new numbering system.

The binary number system has the base two. This means that the value of each of the columns in a number is equal to some power of two. Probably the best way to become familiar with the system is to start counting in it. This is done in Fig. 1-6. Here we find that the values of the columns of the table, starting with the right-hand column are:

$$2^0 = 1$$
$$2^1 = 2$$
$$2^2 = 4$$
$$2^3 = 8$$

Naturally the system doesn't stop here, but this is enough for us to get the general idea of how the binary number system really works. Looking at Fig. 1-6, we see that the first line is all zeros which means exactly that—the number is zero. As we start counting, we will put a 1 in the first column at the right. As shown, this corresponds to the decimal number 1. So far the system is the same as the decimal system. Now let's add one more to get a decimal 2. We don't have the symbol 2 in the binary system, so we will put a zero in the first column and a 1 in the next column to the left. As shown at the heading, this column is the number of two's that we have. As the figure shows we can count as high as we wish using only 1's and 0's.

Once you get familiar with the system, it is very easy to use, although it is cumbersome. For example, the binary number 1011 means that we have:

$$1 \times (2^3 = 8) = 8$$
$$0 \times (2^2 = 4) = 0$$
$$1 \times (2^1 = 2) = 2$$
$$1 \times (2^0 = 1) = \underline{1}$$
$$11$$

An extensive table of binary numbers is given in Appendix 1. Now that we know the rudiments of the binary number system, we

can go back to our problem of finding a way of expressing the value of an analog signal digitally.

DIGITIZING THE SAMPLES

Figure 1-7 shows the same samples of a waveform that we showed in Fig. 1-5B. The first thing that we want to do is to express the value of each of the samples as a binary number. This isn't very hard to do. In the following tabulation we show the value of the sample at each sample time in both decimal and binary numbers:

Time	Decimal Value	Binary Value
0	0	0000
50	2	0010
100	3	0011
150	5	0101
200	3	0011
250	2	0010

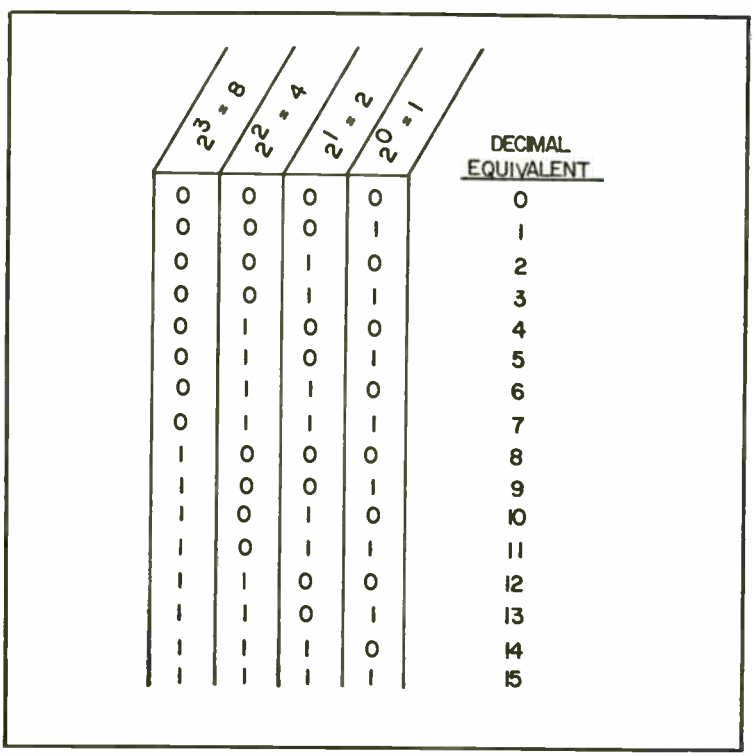


Fig. 1-6. Counting in the binary number system.

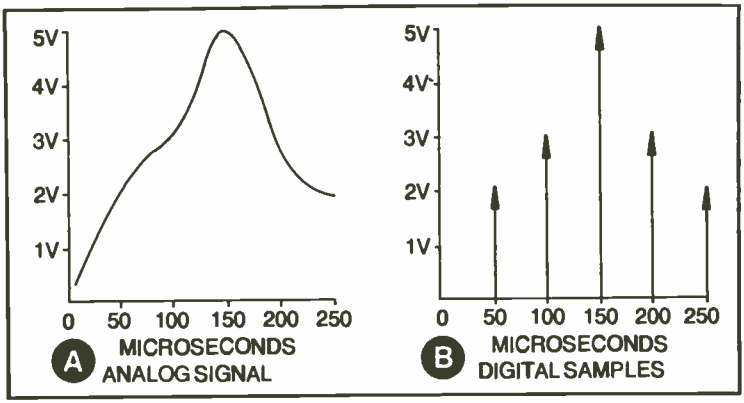


Fig. 1-7. Analog signal and digital sample.

We have binary numbers for each of our samples, but at the moment, we don't quite know what to do with them. One approach is shown in Fig. 1-8. Here we have a little box that will convert our analog signal into a digital signal. We won't have any idea of what is actually in the box until we get to a much later chapter. For now, we will just assume that it works. The output of the box consists of four wires, each of which may have some voltage with respect to ground. That is, at any instant any of the wires coming out of the box may be either high or low.

Note that we have labeled the top wire MSB and the bottom wire LSB. These terms stand for *M*ost *S*ignificant *B*it and *L*east *S*ignificant *B*it. A bit here stands for a binary digit. All that this means is that if we want to read out the binary number represented by the voltages on our four wires, we will start at the top, the same as we would start at the left to read a number. The top wire stands for the left-hand digit in our binary number.

In Fig. 1-8A, we show how the voltages on the four wires correspond to the signal samples that were shown in Fig. 1-7. The time marked in microseconds corresponds to the time marked on the axis of Fig. 1-8. Thus, 150 microseconds after the start of the sampling process, the output of our little box is, starting from the top wire LO HI LO HI. If we merely cross out the H's and L's, we will find the signal will be 0101, which we know is the binary number for number 5. This is exactly what we want because the value of the sample taken at this time happens to be 5 volts.

PARALLEL AND SERIAL TRANSMISSION

The four wires of Fig. 1-8 are one way that we might transmit a digital signal from one point to another. This arrangement is

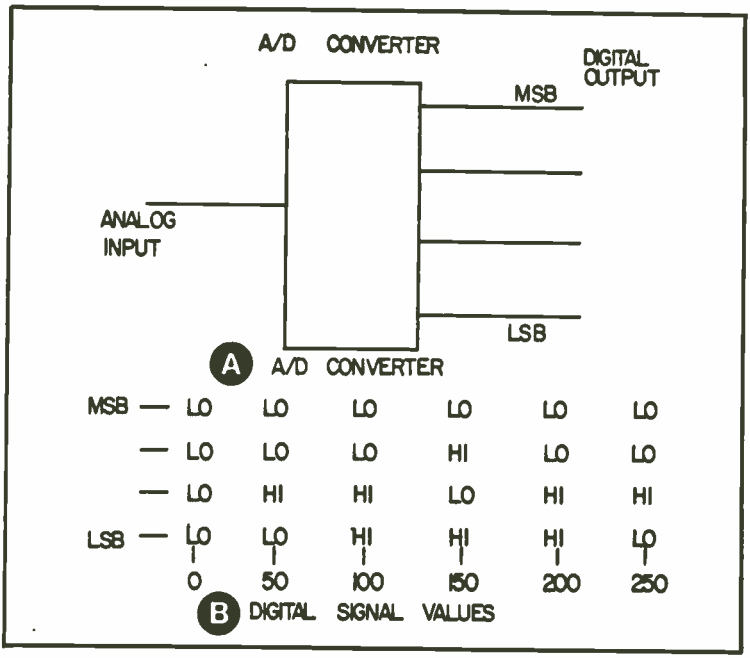


Fig. 1-8. Digital values of the signal in Fig. 1-7.

called parallel transmission because all of the bits corresponding to the value of a signal at a given instant are transmitted in parallel at the same time. This is fine system for use inside equipment where the distances between components is short. Because of the fact that the signal is distributed among several wires, in this case four, the bandwidth of the signal on any one wire is not very great. The disadvantage of parallel transmission is obvious when we want to transmit the data over a long distance. We certainly don't want to use four or more wires to carry one signal.

The other approach to transmission uses only two wires, or one wire and ground. Here the bits corresponding to each value of the signal are transmitted one after the other. Because so many pulses are transmitted for each value of the signal, serial transmission will require a much greater bandwidth.

BITS, BYTES, AND WORDS

We noted earlier that the term *bit* is short for binary digit. It means one of the digits in a binary number. When several bits are used to represent something like the value of a sample, the collection of bits is usually called a *word*. The number of bits in a

word depends on the resolution that we require in a system. The binary number 1111 corresponds to the decimal number 15. This means that if we are to express a quantity in a 4-bit word, we will be able to distinguish 16 discrete values, counting zero as one of the values.

When writing a long string of 1's and 0's to represent a large binary number, life is much easier if we break up the string with a few spaces. A common practice is to write binary digits in groups of four. Thus the binary number, 11111111, which corresponds to the decimal number 255, is usually written as 1111 1111. This is an 8-bit word. Thus, with an 8-bit word, we can distinguish 256 possible values, again including zero as a value.

The 8-bit word has become widely used in digital systems, and is usually called a "byte." Thus, if we are told that a certain word is a byte of data, we will know that is has 8 bits. Recently, someone deduced that if 8 bits constitutes a byte, then four bits should be called a *nibble*. The term nibble is gaining some acceptance to represent a 4-bit word.

Chapter 2
The Easy Way To Understand Logic Gates

The basic building block of digital systems is the logic gate. Any type of digital system that you can think of can be built of logic gates if you use enough of them. A complete understanding of logic gates is essential to an understanding of complex digital systems and is useful in digital troubleshooting. There are several different types of logic gates and they are all very easy to understand if you take the simple approach described in this chapter.

In general a logic gate has two or more inputs and one output as shown in Fig. 2-1. The output will be either high or low, depending on the states of the various inputs. To keep things simple, for the present we will assume that a low state corresponds to zero voltage and a high state corresponds to a voltage of +5V. We will have more to say about the actual voltages in a later chapter. In the tables that we use to describe the various types of logic gates we will use a "0" to represent a low state and a "1" to represent a high state.

Now for the easy way to understand any logic gate that you will ever encounter. All that you have to do is to remember the four symbols shown in Fig. 2-2 and their meaning. These symbols are used in various ways to represent all possible logic gates. Before we get into the meaning of the various symbols let's use the concept of an input being either *active* or *inactive*, rather than being high or low. The reason will soon become obvious.

The first symbol in Fig. 2-2A is called an *active low indicator*. The reason for this symbol is that normally we will consider the inputs or the outputs of a gate to be active if they are high. It is only when we see the active low indicator that we know that the rules

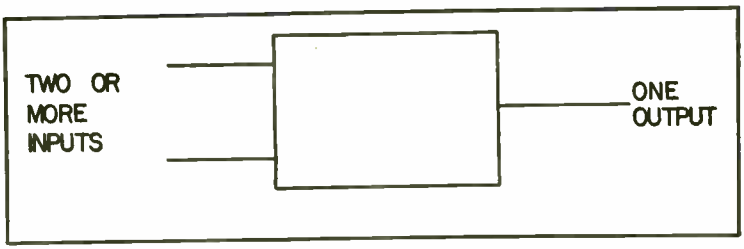


Fig. 2-1. Basic logic gate.

have changed temporarily, and as far as that particular input or output is concerned we will consider a low signal to be active and a high signal to be inactive. Whenever the symbol doesn't appear we will return to the normal rule that a high state is an active state.

The next symbol shown in Fig. 2-2B is called an *ALL* symbol. It represents a logic gate where the output is active only when *all* of the inputs are active. The symbol shown in Fig. 2-2C is called an *ANY* symbol and it represents a gate where the output is active when *any* of the inputs are active. The last symbol, shown in Fig. 2-2D is called an EXCLUSIVE symbol and we will defer its description until after we have described the use of the other symbols.

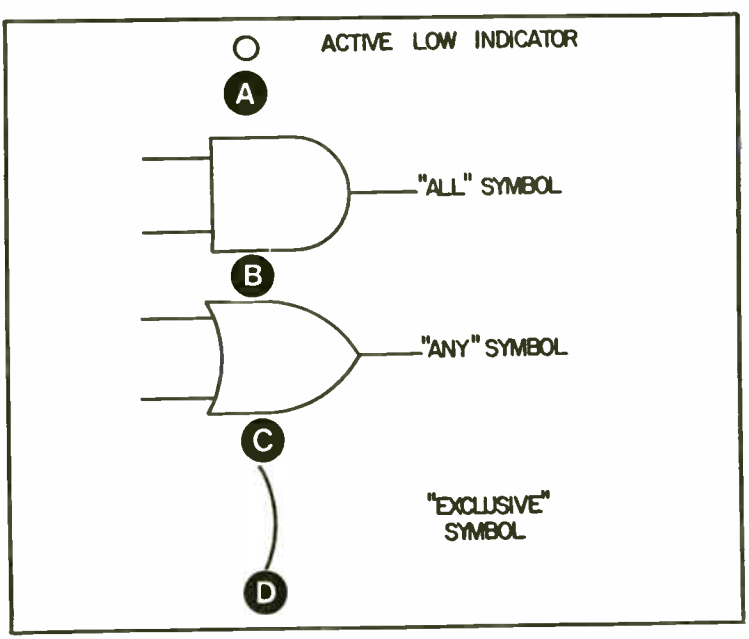


Fig. 2-2. Symbols used to represent logic gates.

THE "AND" GATE

In Fig. 2-3A we show the symbol for a two-input AND gate. As we can see, it consists of an ALL symbol with two inputs and one output. Nothing being indicated to the contrary, we know that the two inputs and the output can be considered to be active whenever they are in a high state, which we agreed for the time being to be +5V.

From the basic definition of the ALL symbol we know that the output will be active only when both of the inputs are active. In this particular case, this means that the output will be high only when the two inputs are both high. The easy way to show this is to construct a ***truth table***, like that shown in Fig. 2-3B. In this table, the first two columns show the states of the two inputs and the third column shows the corresponding state of the output. In the first row in the table, the inputs are shown by "0's" to both be low. The output is also low. In the next two rows of the table, one of the inputs is high as shown by the "1's." Of course, the output is also "0." In the last row of the table where the two inputs are both high, or active, the output is also active, or high.

As we noted at the outset, this type of logic gate is called an AND gate. The reason is simply that the output is high whenever input "A" *AND* input "B" are high. Let's now look at similar gates where we use the active low indicator.

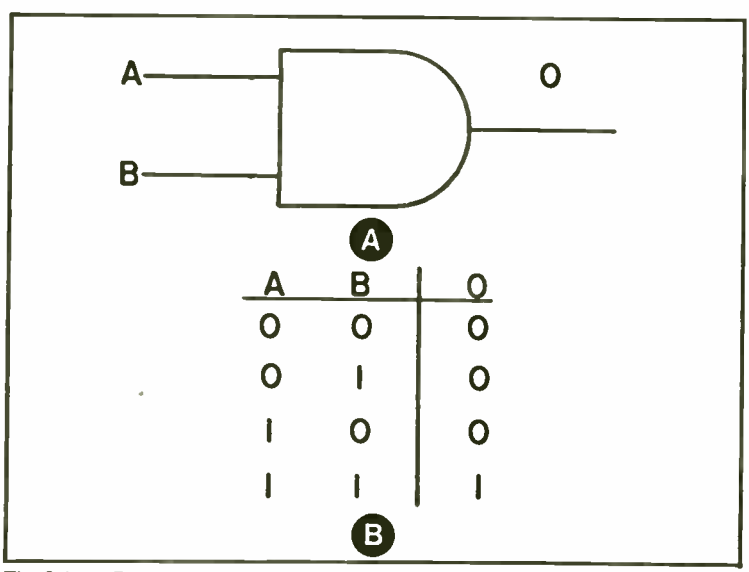


Fig. 2-3. AND gate and its truth table.

THE "NAND" GATE

Figure 2-4A shows the symbol for a NAND gate, and its truth table is shown in Fig. 2-4B. Once again we have the ALL symbol that indicates that the output will be active only when all of the inputs are active. As far as the inputs are concerned, nothing has changed from the preceding example. They are considered to be active when they are high. When we come to the output, however, we see an active low indicator. This means that we will now consider the *output* to be *active* when it is *low*.

In constructing the truth table, we will again show all possible combinations of the inputs. The output will be active, or *low*, when all of the inputs are active, or *high*. The result is the truth table shown in Fig. 2-4B. The only condition when the output will be low is when both of the inputs are high.

This gate is called a NAND gate. The name comes from NOT AND. It means that the output, for various combinations of inputs, will be just the opposite of that from an AND gate. A comparison of the truth tables in Fig. 2-4B and 2-3B will show that this is true.

An actual practice, the NAND gate is much more common than the AND gate. The reason is that when digital integrated circuits were developed, a transistor was added to the basic gate structure to provide isolation. This transistor inverts the output of the gate. This concept will become clearer when we look into the various types of logic families.

THE "OR" GATE

Figure 2-5A shows the symbol for an OR gate and Fig. 2-5B shows its truth table. The basic symbol in this example is an ANY symbol which means that the output will be active whenever *any* of the inputs is active. There being no active low indicators in the symbol, we know that we consider inputs and outputs to be active when they are high. From the definition of the ANY symbol, we know immediately that the output will be high whenever any of the inputs is high. A look at the truth table will verify this. In fact the only condition under which the output will be low is when both of the inputs are low.

The name, OR gate, for this symbol comes from the fact that the output of the gate will be high when input "A" OR input "B" is high. Although it isn't explicit in the OR symbol, the OR condition also embraces the condition when both inputs are high.

THE NOR GATE

In Fig. 2-6 we have the symbol for an NOR gate. Here again we use the ANY symbol which means that the output is active

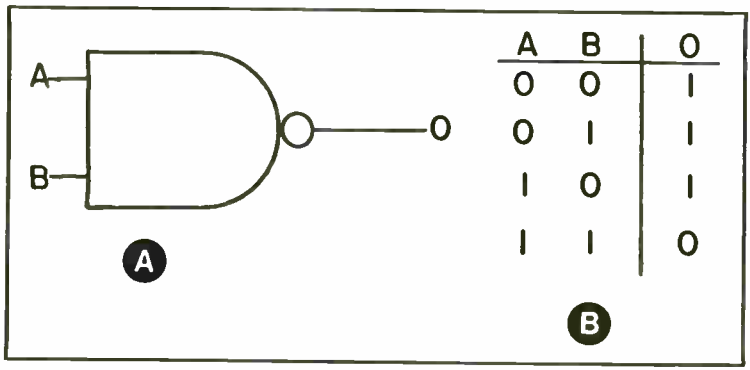


Fig. 2-4. NAND gate and its truth table.

whenever any of the inputs are active. As far as the inputs are concerned, there is no change from the preceding example; they are active when they are high. But on the output of the gate we have an active low indicator that flags us to the fact that as far as this one terminal is concerned, the rules have changed again. The *output* will be considered to be *active* when it is *low*.

The behavior of the gate is shown in the truth table in Fig. 2-6B. Here we see that the output is active, or *low*, whenever ANY of the inputs are active, or *high*. In practice, NOR gates are more common than OR gates for the same reason that we gave under the discussion of NAND gates.

WORKING WITH ZEROES, OR NEGATIVE LOGIC

In some of the older literature you will find references to what was called negative logic. Fortunately, the term is seldom if ever used anymore. Negative logic referred to the case where a high signal was considered to be a 0 and a low signal was considered to be a 1. You can imagine the confusion that the whole idea caused to

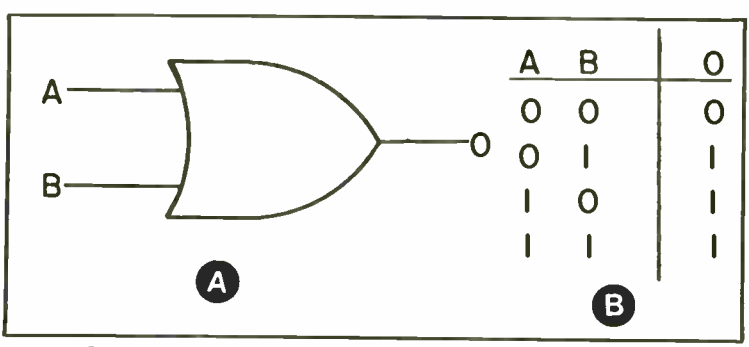


Fig. 2-5. OR gate and its truth table.

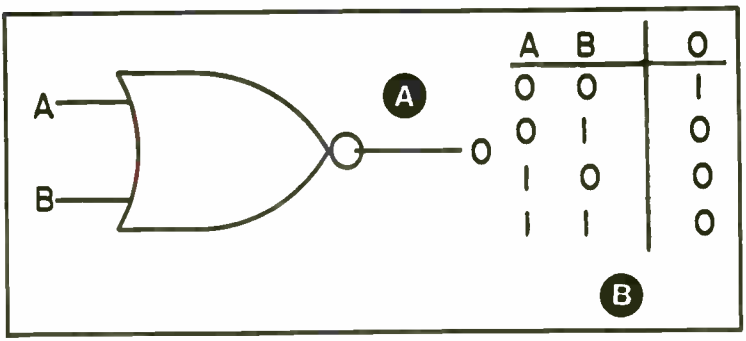


Fig. 2-6. NOR gate and its truth table.

anyone approaching the subject for the first time. Lest you suspect that the originators of the idea were completely insane, we should explain that there was a valid reason for the concept. It arose in connection with using Boolean algebra in the design of logic circuits. Incidentally, Boolean algebra usually isn't necessary for anyone working with digital systems.

Even though we no longer feel any need for negative logic, there will indeed be cases where we will be more interested in 0's or lows than we are in 1's or highs. For example, we may have a system in which we are interested in the condition where we want to work with switch openings, rather than closings. The signals on which we might wish to operate will be 0's instead of 1's.

Figure 2-7A shows a case of this type. We want to get a high signal when two other signals are both low. We can symbolize this operation by the ALL symbol with the two inputs designated as active low. This means that the output will be active, or *high*, whenever ALL of the inputs are active or *low*. By now the truth table shown in Fig. 2-7C that describes this type of gate is clear. It shows that the only condition under which the output is high is when both inputs are low. We could call this operation "ANDing" zeroes.

There is only one problem with the type of gate that is shown in Fig. 2-7A, and that is that it isn't available commercially. You can't run down to the store and buy one. When you see this symbol on a diagram you can be sure that the actual gate in the circuit isn't this type of gate at all. The symbol is used to show the logical behavior of the gate. Fortunately, the fact that this gate isn't available isn't any problem at all. All we are really interested in is how the gate behaves. Any gate, no matter what its symbol happened to be, that behaved according to the truth table in Fig.

2-7C will do the job. If you will simply look back at the truth table for the NOR gate in Fig. 2-6, you will see that it has exactly the same truth table!

What this means in practical terms is that ANDing zeroes is exactly the same thing as NORing ones. Thus, if you were to find the symbol of Fig. 2-7A on a diagram, you would not expect to find this gate in the actual system. What you would find is a NOR gate like that shown in Fig. 2-7B. Logically they are exactly the same thing.

This concept of performing logical operations on low signals instead of high signals may seem confusing at first, but as soon as you begin to work with a few real-world digital systems you will find that it can be very useful.

Another example of working with 0's is shown in Fig. 2-8A. Here, what we want to do is to provide a high signal whenever any of the inputs is low. To symbolize this we draw the ANY symbol with two active low inputs. The truth table for this symbol is shown in Fig. 2-8C. Here again, the gate has the disadvantage that it is not available. It doesn't need to be because it has exactly the same truth table as the NAND gate that we studied in connection with Fig. 2-4. Again, the two logic symbols result in the same physical gate. All gates are normally described in terms of how they will operate on high signals. But if we are familiar with the ALL symbol, the ANY symbol, and the active low indicator, we can quickly convert from any logic symbol to a real, physically available logic gate.

In Fig. 2-9A and B we have shown two other symbols that are sometimes used. By constructing a truth table for these gates, you

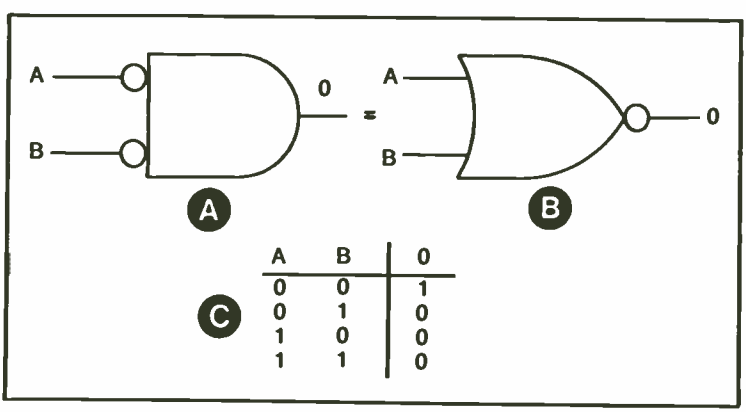


Fig. 2-7. ANDing zeros is equivalent to NORing 1's.

can prove to yourself that they are exactly equivalent to the gates shown in Figs. 2-9C and 9D.

THE EXCLUSIVE GATES

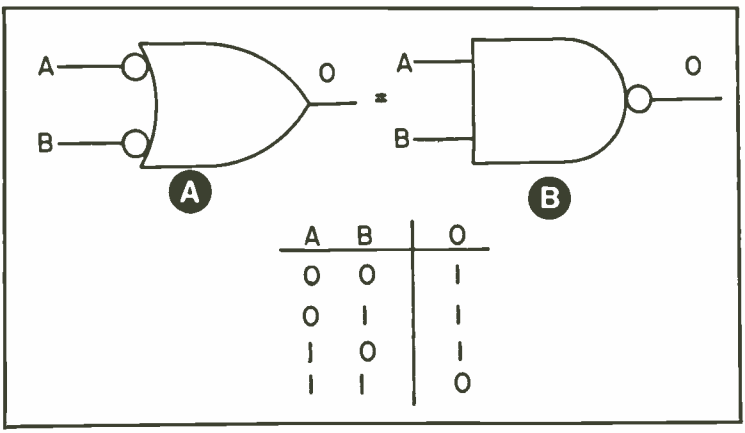
There is one more basic symbol in Fig. 2-2 that we haven't used yet. It is the one that we called the *EXCLUSIVE* symbol in Fig. 2-2D. The symbol is used only in connection with OR and NOR gates and it has a rather simple meaning. It could, in fact, be called a "BUT NOT ALL" symbol. Figure 2-10A shows the symbol for what is called an *exclusive OR gate*. Its truth table is shown in Fig. 2-10B. From the truth table we see that the output is high whenever any of the inputs is high, *but not both*. The only place where the truth table of the exclusive OR gate differs from that for a regular OR gate is in the bottom line. The OR gate has a high output whenever any of the inputs, or all of them, are high. The exclusive OR gate has a high output whenever any of the inputs, but not all are high.

Figures 2-10C and D show the symbol and truth table for the exclusive NOR gate. Its truth table is just the opposite of that of the exclusive OR gate. The output is low whenever any, but not all, of the inputs are high.

The exclusive OR and NOR gates can be very useful in many logic circuits, particularly when performing arithmetic operations with the binary number system.

THE INVERTER, OR "NOT" GATE

The simplest possible type of gate is the inverter shown in Fig. 2-11. It is really an amplifier with a gain of only one and where



A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 2-8. ORing 0's is equivalent to NANDing 1's.

the output is out of phase with the input. As shown in the truth table, if the input is high, the output is low; and if the input is low the output is high.

This gate is easy enough to understand, but to one unfamiliar with digital circuits it may seem so simple as to be of little practical use. To the contrary, the inverter, or as it is sometimes called, the NOT gate, is one of the handiest devices the digital system designer can have available. In many systems, there is need for an operation that can be performed easily if some of the signals can be inverted. The problem can be solved by simply inserting an inverter whenever it is desired to invert a signal. Inverters come six to a single integrated circuit, so they are easy to include in a design. As we continue to investigate the application of logic gates, we will find the inverter will appear frequently.

GATES WITH MORE THAN TWO INPUTS

At the beginning of this chapter when we defined logic gates, we said that a logic gate has one output and two or more inputs. Then for the sake of keeping things simple, we ignored the "or more" part of the statement. All of the practical gates that we have investigated, except the inverter, have only two inputs. In actual practice, logic gates with three, four, or even eight inputs are rather common. The number of inputs should cause us no trouble as long as we remember the meaning of the basic symbols that were shown in Fig. 2-2.

When confronted with a gate with any number of inputs, all we have to do is to construct a truth table showing the state of the

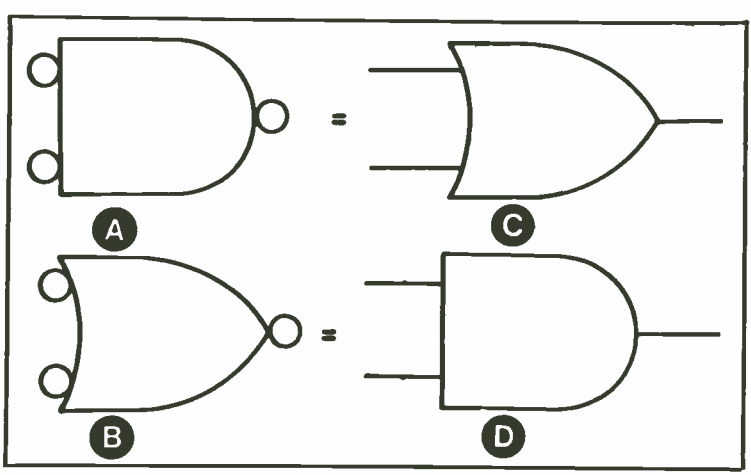


Fig. 2-9. Other logic gate symbols sometimes used (see text).

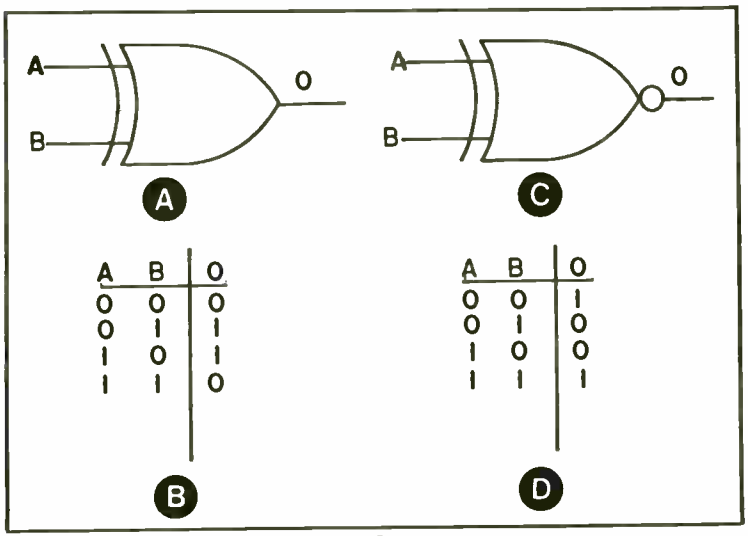


Fig. 2-10. Exclusive OR and exclusive NOR gates.

output for all possible combinations of input signals. For example, Fig. 2-12 shows an AND gate with three inputs. Below the symbol is the truth table. Knowing that the basic symbol is an ALL symbol we would recognize immediately that the only time that the output would be active, in this case high, is when ALL of the inputs are high. Constructing the truth table verifies this.

All that we have said so far about two-input gates applies to gates with more than two inputs. They are just as easy to understand and use as two-input gates. About the only difference is that the truth table will be longer because there are more possible combinations of input signals. As a matter of interest, there are 2^n

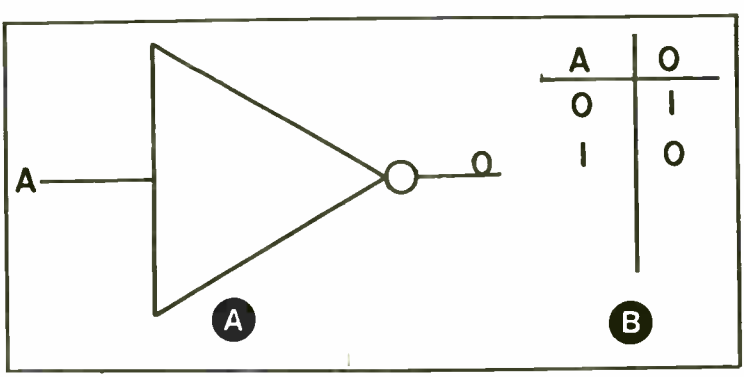


Fig. 2-11. The inverter or NOT gate and its truth table.

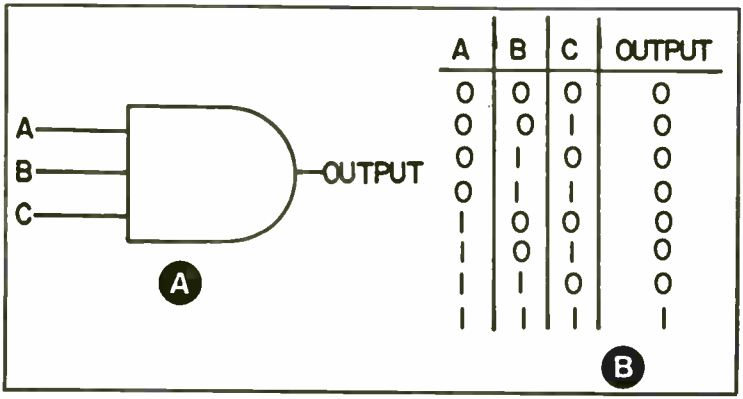


Fig. 2-12. Three-input AND gate (A), and its truth table (B).

possible combinations of input states of a logic gate where "n" is the number of inputs. Thus, with a two-input gate, we have $2^2 = 4$ possible combinations of input signals, so there are four lines in the corresponding truth table. Likewise, with the three-input gate of Fig. 2-12, there are $2^3 = 8$ possible combinations of input states, so there are 8 lines in the truth table. An eight-input gate is just as easy to work with, but there will be 2^8 possible input states and, therefore, 256 lines in the truth table. Usually, when we use a gate with a large number of inputs, we can see the intended operation of the circuit well enough so that we don't have to construct the entire truth table.

THE LOGIC GATE AS A SWITCH

When studying the operation of the various types of logic gates it is easy to visualize some possible applications. In general these are rather simple functions. The logic gate actually has many different applications which might not be immediately obvious. One such application is as a switch for digital signals.

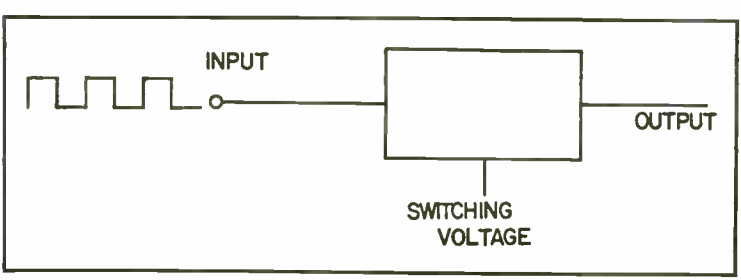


Fig. 2-13 Switching circuit block diagram.

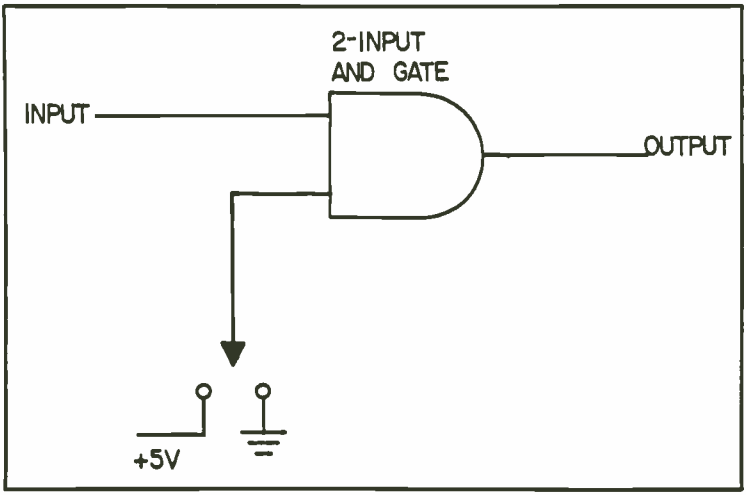


Fig. 2-14. Circuit using an AND gate as a switch.

Figure 2-13 shows a black box with an output and two inputs. One of the inputs is connected to something like a pulse generator which furnishes a string of pulses which we can think of as a series of logical 1's and 0's. The other input is connected to what we call a switching voltage. What we want the box to do is to allow the pulses to appear at the output when the switching voltage is high and to hold the output at a low level when the switching voltage is low.

This function is extremely easy to provide. All that we need is a simple AND gate as shown in Fig. 2-14. Here input "A" of the gate is connected to the source of pulses. If input "B" is low, nothing will happen at the output because we need all of the inputs active before the output will be anything other than a logical 0. When input "B" is made high, the output of the gate will follow input "A." The pulses will appear at the output.

In this chapter, we have described how logic gates work. We certainly haven't exhausted the subject. We'll have more to say about gates throughout the book.

Before going any farther, let's take a look at what's actually inside an integrated circuit.

Chapter 3
Logic Families, or What's Inside An Integrated Circuit

One question that often arises in connection with digital electronics is that of just how much an engineer or technician needs to know about what is actually inside a digital integrated circuit. There are two general schools of thought on the subject. One group maintains that if a system designer or technician knows what the IC does functionally, as well as such things as the required voltages, time delays and similar characteristics, he doesn't need to have any knowledge at all about what is actually inside the IC. He could care less how the thing is built. His only concern is how it operates functionally.

Using this approach, the diagram of a typical IC would be like that shown in Fig. 3-1. Here we can tell what all of the pins of the IC are used for, and if we also have a data sheet, we will know the proper voltage levels. At first glance it looks like a reasonably good way to approach the subject. It does, however, have some limitations. Perhaps the biggest limitation is that the technical mind of the engineer or technician is rarely satisfied with such a simplified approach. He might be warned in the data sheet that he can't connect pins 3 and 6 of the IC of Fig. 3-1 in parallel. Usually, he will waste so much time wondering why that he would have been better off learning a little more about what is actually inside the IC.

At the other extreme, we might use the complete schematic diagram of the same IC. This is shown in Fig. 3-2. There is no question that this figure contains much more information than Fig. 3-1. The trouble is that the information isn't in a form that is very useful to anyone. The first thought may simply be a doubt as to the sanity of anyone that would design a circuit like this.

In this book, we will take the approach that the block diagram of Fig. 3-1 is adequate, *provided that we have some idea of what is actually inside the IC.* We are particularly interested in the type of circuitry that is connected to the various pins of the IC. With this knowledge, we will have a good idea of what might happen if we were to connect pins 3 and 5 of the IC of Fig. 3-1 in parallel.

Before considering the actual circuits inside an IC, we must realize that the designer of an IC faces many problems that are quite different than those encountered when designing a similar circuit using discrete transistors, resistors, diodes, etc. Usually the transistor is the easiest component to build in an IC. For this reason integrated circuits usually use more transistors than a similar discrete circuit would use. Resistors and diodes are probably the next easiest components to build, but it is hard to hold the value of a resistor to a predetermined value. It is much easier to hold the ratio of the values of two resistors to a given tolerance than their absolute values. This, of course, will affect the design of the circuit. Finally, capacitors are really hard to make in an integrated circuit, so the IC designer will usually avoid them like the plague.

Many different technologies have been used to build digital integrated circuits. Most of these have had a rather limited use.

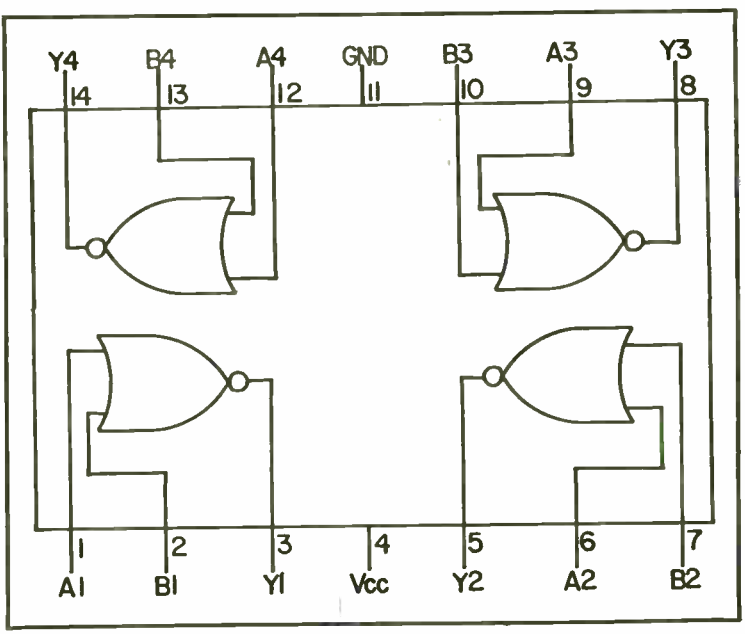


Fig. 3-1. IC basing diagram, Type 7402 (W).

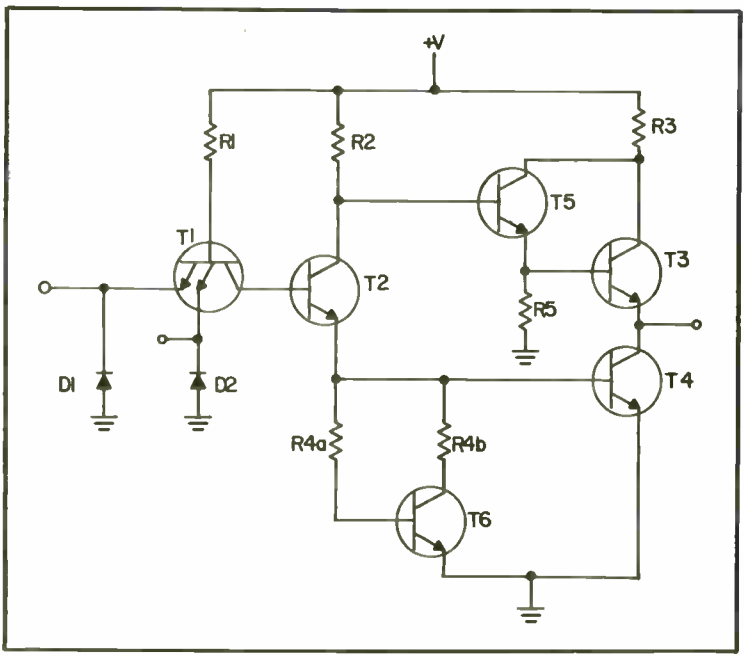


Fig. 3-2. Circuit diagram of a TTL integrated circuit.

The logic families that you are most likely to encounter at present are TTL (*T*ransistor *T*ransistor *L*ogic) and CMOS (*C*omplimentary *M*etal-*O*xide-*S*emiconductor logic). In this book we will stick to these two families because they are the ones that you will be most apt to encounter and because if you once became familiar with these two logic families and their application, you can learn about the others rather easily if you should ever find it necessary.

TTL CIRCUITRY

Figure 3-3 shows a somewhat simplified version of the diagram of Fig. 3-2. The most striking part of this diagram is the fact that the input stage uses a rather strange-looking transistor that seems to be drawn sideways. Without a further hint, it isn't easy to figure out just how this circuit is supposed to work.

We have a little help in analyzing the circuit because we know that it is a two-input NOR gate. This tells us that the output is high when either or both of the inputs is grounded and the only condition under which the output will be low is when all of the inputs are high.

Let's start with the input stage of the device, but let's redraw it so that we have a little better chance of finding out just what goes on in the circuit. In Fig.3-4 we have shown part of the circuit, but

we have drawn the transistor right side up and for the moment we only show one emitter. This makes Q_1 just like any other NPN transistor. For the moment we have connected the emitter of the transistor to ground.

This circuit is fairly easy to analyze. Inasmuch as the emitter is grounded and the base is connected to the positive supply through a 4K resistor, the transistor will be turned on hard. There will be very little voltage drop between the emitter and the collector. This will assure that transistor Q2 is turned off because its base will, for all practical purposes, be grounded. This, of course, means that point A will be at the +5V supply level and point B will be at ground potential.

So far we have considered only one of the emitters of transistor Q1, but we might suspect that the actual transistor would act like Q1 of Fig. 3-4 if either of the emitters were grounded and this suspicion would be correct. So far, so good. The input portion of the circuit of Fig. 3-3 isn't very hard to understand if either of the two inputs happens to be grounded. The next question is, "What happens if both of the inputs are connected to the +5V

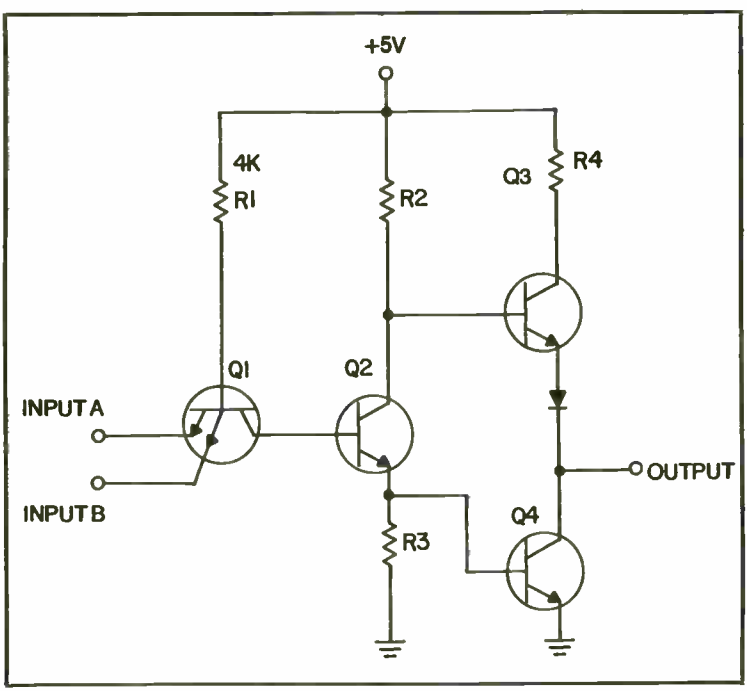


Fig. 3-3. Circuit of a TTL NAND gate.

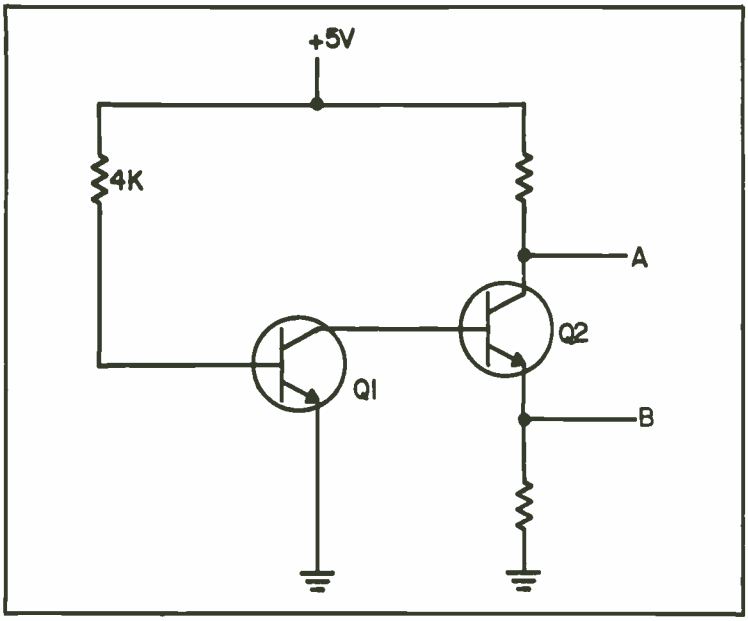


Fig. 3-4. Equivalent of the circuit in Fig. 3-3, with input grounded.

supply?" This question isn't as easy to answer without giving a little thought about how a transistor works.

The fact is that if both of the inputs are high, Q1 doesn't act like a transistor at all. In fact, when both of the inputs of the circuit of Fig. 3-3 are high, we can ignore them completely. No current will flow in the emitters of transistor Q1. We can omit the emitters from our equivalent circuit.

Figure 3-5 shows the equivalent circuit. The only part of transistor Q1 that we have shown is the base-collector junction and it is easy to see that under this condition, the base-collector junction is a forward-biased diode. Base current will flow in transistor Q2, turning it on hard. There will be very little voltage drop across the transistor, so resistors R2 and R3 will act as a voltage divider. If nothing else were connected to the circuit, the voltage at point A would be slightly higher than that at point B.

Now that we have the input stage under control, let's try to figure out how the entire circuit of Fig. 3-3 works. To do this we will consider two conditions, namely when the output is high and when the output is low. We will first look at when the output is high; that is, when both inputs are low. We will only show enough of the circuit to enable us to figure out how it works.

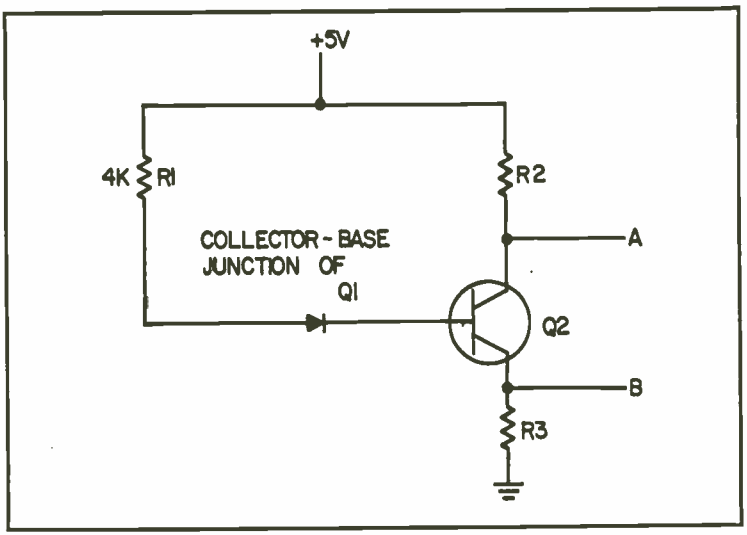


Fig. 3-5. Equivalent of the circuits in Figs. 3-3 and 3-4, with both inputs high.

Figure 3-6 shows the equivalent circuit of the entire gate when one or both of the inputs are low. The input portion of the circuit is the same as that shown in Fig. 3-4. Here, as we noted earlier, transistor Q1 is turned on hard and transistor Q2 is turned off. Point A is at a high potential and allows base current to flow in transistor Q3, turning it on. Point B is still at ground potential so transistor Q4 cannot turn on. The output is thus at a high level. It will not go all the way to +5V, because there will be voltage drops across resistor R4, transistor Q3, and diode D1. Usually the high output level is around 3 volts.

Now let's look at the opposite situation, when the output is low. As we noted earlier, this is when both of the inputs are high. This gives us the equivalent circuit of Fig. 3-7, where the input is the same as that shown in Fig. 3-5. Here transistor Q2 is turned on. The obvious result of this is that base current will flow in transistor Q4, turning it on. It might appear superficially that transistor Q3 might also want to turn on, but it won't. The circuit is designed so that either Q3 or Q4, but not both, will be turned on at any particular time. The diode, D1, used in the output of the circuit is included to be sure that Q3 will not be turned on at the same time that Q4 is on.

With transistor Q4 turned on and Q3 turned off, the output will be at nearly ground potential. Usually the low output level of a TTL circuit is in the order of +0.2V.

Although the circuit that we have used to understand TTL is typical, it is important to remember that the internal circuitry may vary from one manufacturer to another. The circuit we have described is adequate for our purpose which is to learn how to work with TTL ICs intelligently.

THE TOTEM-POLE OUTPUT

The reader will realize that you can't do very much with a single logic gate. In digital systems, there are very many logic gates connected together. For this reason, we should look at what happens when the output of one gate is connected to the input of another. Figure 3-8 shows a typical arrangement. Let's first look at what happens when the output of the driver is low. In the figure, Q4 will be turned on and Q3 will be turned off. As we noted earlier, the input of a gate will draw a little current when it is low because transistor Q1 is on. If we adopt the conventional current direction as from positive to negative, we will see that current will flow out of Q1 and into Q4. Thus the output of the driving stage doesn't furnish current to the next stage, rather, it provides a connection to ground. It is for this reason that TTL is often referred to as a *current sinking* type of logic. There are other logic families where the output of one stage furnishes current to the input of the following stage. Such logic families are said to be *current sourcing* logic.

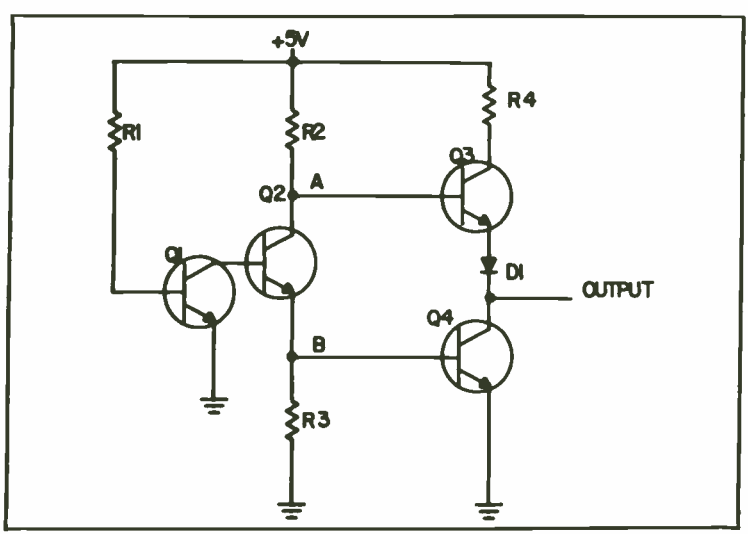


Fig. 3-6. Equivalent of circuits in Figs. 3-3, 3-4, and 3-5 with the output high.

Under the opposite condition, Q3 is turned on and Q4 is turned off. Now the input is Q1 is high and as we noted, no current flows. This brings up the interesting question that if transistor Q3 doesn't supply current to the following stages when it is turned on, why do we bother to have it in the first place. The fact is that this transistor isn't needed to raise the inputs of a following stage to a high level. In fact there are gates that we will consider later where the transistor is omitted. We can see how this transistor serves a useful purpose by considering what would happen in a practical circuit if it were not included.

Figure 3-9 shows the output of a TTL gate where the top transistor of the totem pole has been replaced with a resistor. Also shown in the circuit is some capacitance connected between the output pin and ground. This is not a capacitor, but rather the inevitable capacitance of leads, wiring on a printed circuit board, and the capacitances of other gates connected to this line. Assume that the output is initially low; that is, that transistor Q4 is turned on. There will be no charge in the capacitor. Now let's assume that the circuit changes states; that is that transistor Q4 turns off and the output tries to go high. We know from elementary electronics that the voltage across a capacitor cannot change instantaneously. This would require an infinite current. What would happen in the circuit of Fig. 3-9 is that the capacitor would charge through the

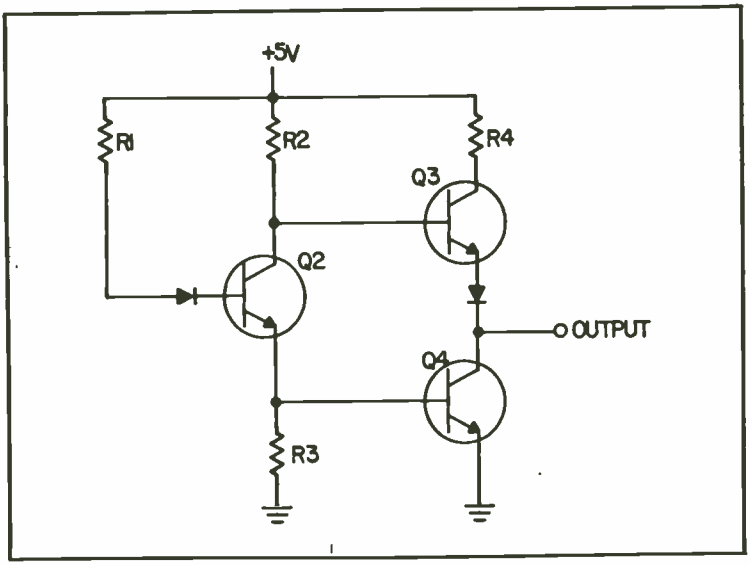


Fig. 3-7. Equivalent of the circuits in Figs. 3-3, 3-4 and 3-5 with the output low.

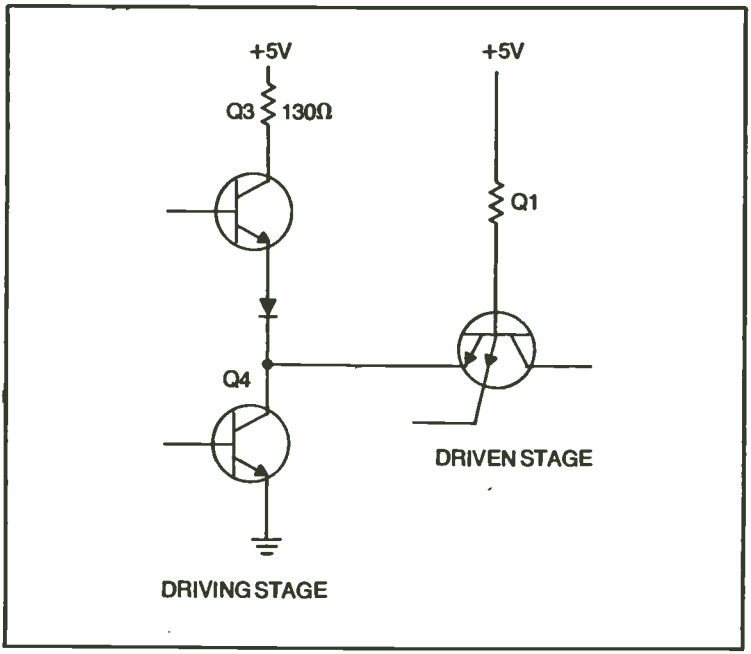


Fig. 3-8. Interconnection between TTL gates.

resistor R1. If we made R1 very large, it would take a long time for the capacitance to charge and the circuit would be very slow. If on the other hand, we made resistor R1 very small so that the circuit would be fast, there would be a large current through it when transistor Q4 was turned on, and the stage would consume a lot of power.

From this it can be seen that the top transistor in the totem pole output is used to supply the necessary current to charge any capacitance in the circuits connected to the output so that the output will be pulled to a high level very quickly. Once this has been done, Q3 just sits there. It doesn't have to supply current to hold the inputs of following gates at a high level. It is easy to see why the arrangement that uses transistor Q3 is referred to as an *active pull up.*

OPEN COLLECTOR OUTPUTS

It is easy to imagine a situation where it is desirable to connect the outputs from two different integrated circuits to the input of another circuit. Figure 3-10A shows the arrangement. As long as both of the stages that are connected together are at the same

output state—both high or both low—there is no problem, but on the other hand, if we know that both of the outputs will always have the same state, there is no point in connecting them together. We could simply look at one, knowing that the other would have the same state.

When the two stages that are connected in parallel have opposite states—one high and the other low—things begin to happen. Suppose that in Fig. 3-10A, stage A has a high output state and stage B has a low state. The output of stage A will try its best to pull the common connection to a high level, while the output of stage B is trying equally hard to pull the common connection to a low level. Usually the output that is trying to go low will win the struggle. The result is as shown in Fig. 3-10B. In this figure, both of the transistors are on and there is a high current path between the +5V supply and ground. The 130-ohm resistor will provide some current limiting, but usually one or the other of the circuits will not survive.

Although the totem-pole outputs of a regular TTL gate cannot be connected in parallel, the arrangement shown in Fig. 3-11 is very useful in many applications., All that we have to do to use it is to find some gates whose outputs can safely be connected in parallel. The arrangement of Fig. 3-11 is called by many different names. The arrangement in the dashed box where the outputs are connected together is called *wired-OR*, *dot-OR* and *dotted collector*.

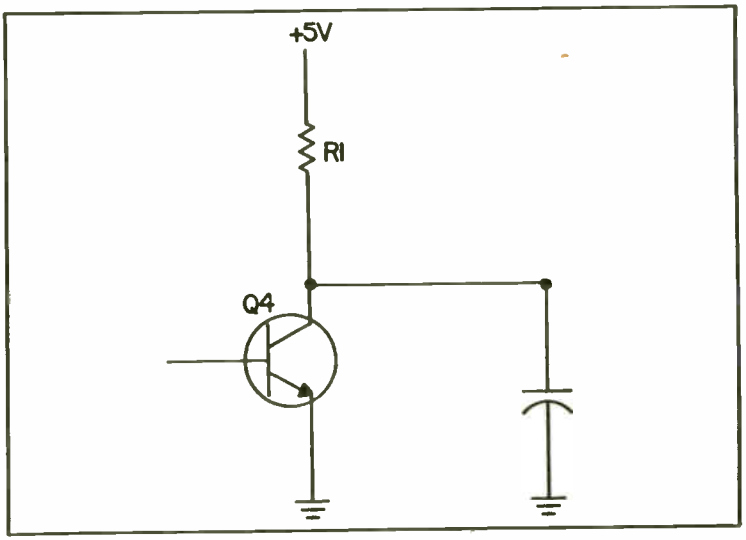


Fig. 3-9. Charging currents in a TTL output stage.

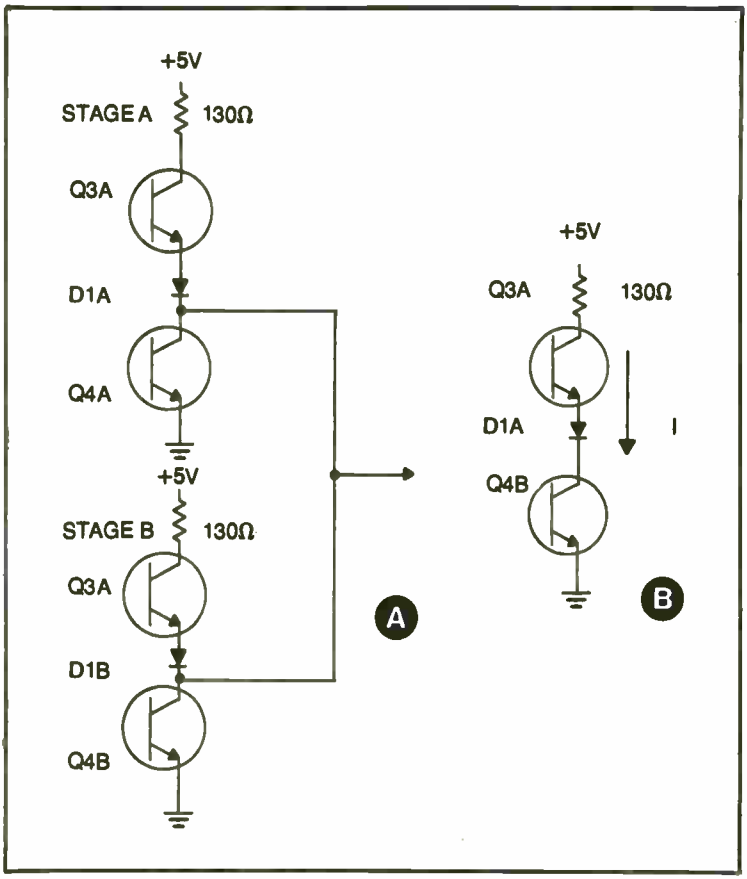


Fig. 3-10. Two totem-pole outputs connected in parallel.

All of these names are misleading. The function represented by the box is really an AND or an ALL function. The output goes high only when *all* of the lines, A, B, and C are high. As soon as any one of the lines goes low, the output will go low. For this reason, probably the most descriptive name for the arrangement is a *wired AND*.

Now to find the gates that will make this arrangement practical. Figure 3-12 shows the schematic diagram of a special TTL NAND gate which is called an open-collector gate. Comparing this circuit with the circuit we showed earlier of a TTL gate, we see that the top transistor of the totem-pole output, together with its series resistor and diode, have been omitted. Otherwise the gate is the same as any other TTL gate. The output is the collector of transistor Q4 which isn't connected to anything else. We can see

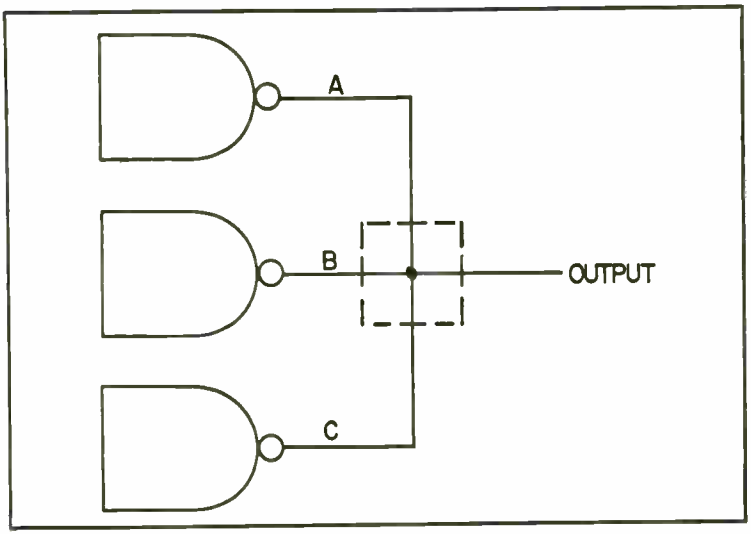


Fig. 3-11. Wired-AND arrangement.

that if Q4 is turned on, the output will be pulled to a low level, but unless we add something to the gate, the output can't go to a high level. This something that we add is a pull-up resistor shown by the dashed lines. With this resistor, the output will go to a high level whenever transistor Q4 is turned off. Inasmuch as we no longer have the active pull-up transistor, we can now connect the outputs of several gates of this type in parallel with a single pull-up resistor

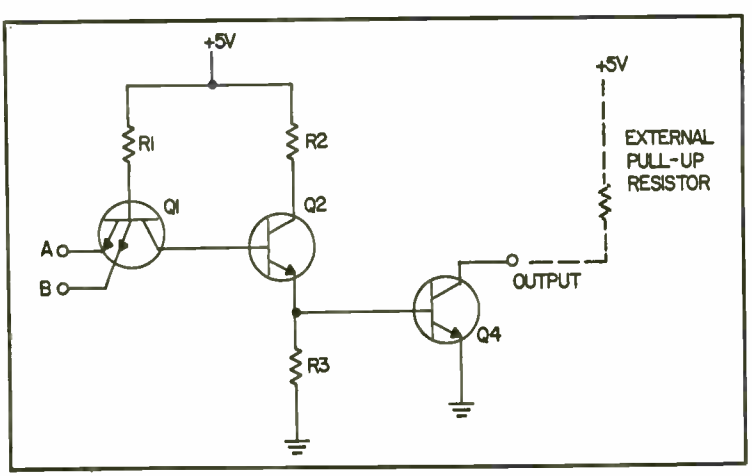


Fig. 3-12. Circuit of an open-collector gate with external pull-up resistor.

and the gates will not be damaged if one should happen to go high and another low.

Figure 3-13 shows two such gates connected in parallel with a single pull-up resistor. From this figure we can clearly see that the output will go high only when the outputs of all gates that are connected together are high. If the output of any of the gates should go low, its output transistor will pull all of the outputs low, but no damage will be done to any of the stages.

The open-collector gate is widely used when it is necessary to connect the outputs of several gates to a common line. There is also another arrangement that can be used for this purpose called *tri-state logic*. We will get to this a little later.

TTL CHARACTERISTICS

So far we have treated digital ICs in a sort of qualitative way. We have noted that when the inputs of a NAND gate all go high, the output will go low. We haven't bothered with such questions as to just what voltage levels constituted a high or a low level. Neither have we questioned such things as to how much time is required for the gate to respond after the input levels change. If we are to be able to work with or troubleshoot digital systems intelligently, we must have quantitative answers to these questions.

Input Levels

Whenever the voltage at the input of a TTL gate is below 0.8V, the gate will treat it as a low level. Usually manufacturers

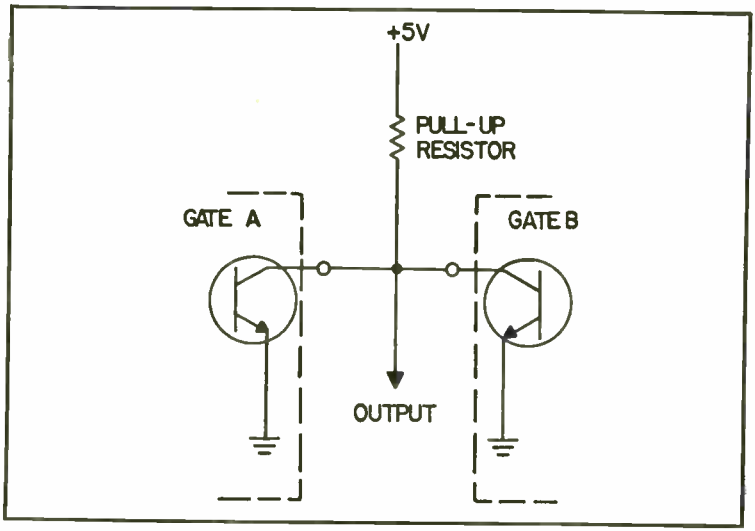


Fig. 3-13. Two open-collector gates connected in parallel.

guarantee this level. When the voltage rises above about 2V, the circuit will treat it as a high. Note that there is a gap between what constitutes a low and what constitutes a high level. This is not accidental. Levels in this no-man's land are avoided to allow room for noise spikes that will inevitably get onto the signal lines in a system. We will have more to say about this later.

Output Levels

On the output pin, the level will usually be about 0.2V for the low state, and about +3V for a high level. The manufacturer usually guarantees that the low will not be higher than 0.8V and the high will not be lower than +2.4V.

Timing

Obviously, it will take some time for a gate to respond to an input signal. This time period, although very short, can be significant in a complex digital system. The time required for a digital circuit element to respond to a change in the level of an input signal is called the *propagation delay* or *propagation time* of the unit. Actually, the time required is not the same when the output of a gate is going from a high state to a low state as it is when it is going from low to high. This is shown in Fig. 3-14.

The waveform at the top is the signal applied to the input of a gate. It is going from a low level to a high level. The output responding to this signal switches from a high level to a low level. The propagation time required for the output of the gate to go from a high to a low level is labeled T_{PHL} in the figure. This time is usually in the order of 7 nS or less. The time required for the output to go from a low to a high state is labeled T_{PLH} and is usually 11 nS or less.

The term propagation delay, which is often specified, is the numerical average of the two times described above. Both of the propagation times depend on the amount of loading on the output and on the supply voltage. The time required for the output to go from high to low decreases as the temperature of the device increases and the time required to go in the opposite direction is independent of temperature.

As we will see later, the propagation time of a gate can be used to advantage to prevent circuits from oscillating.

LOADING OR FAN-OUT

We have noted several times that various characteristics of TTL ICs depend on loading. The load of an IC is really the amount of current that the output pin must handle. This, of course, is not

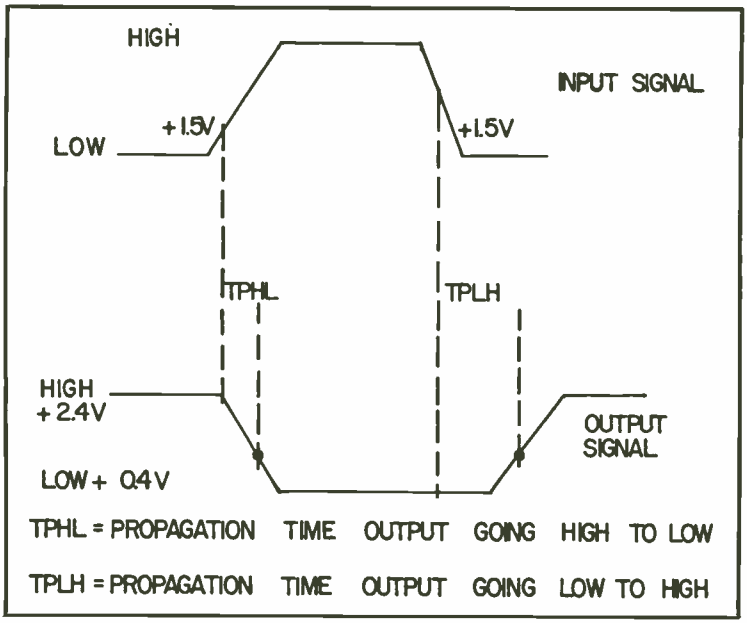


Fig. 3-14. Waveforms showing propagation times in digital devices.

the same when the output is at a high level as when it is at a low level. Inasmuch as most TTL outputs are connected to TTL inputs, the loading capability of TTL outputs is specified in terms of the number of TTL inputs that can be connected to it. The parameter is usually called the *fan-out*. An output that has a fan-out of 10 can drive ten input terminals of standard TTL gates.

Specifying the output capability of a gate in terms of its fan out is fine as long as the only thing connected to the output is the input of another gate. Sometimes, we must connect something else to the output and we need to know how much current it will supply. Similarily, the output capability of some digital ICs is given in terms of the amount of current the output will supply in the high and low states. Under these conditions, we need to know a little more about the relationship of fan-out to actual current capability.

The output capability of a TTL gate can also be specified in terms of what is called a *unit load*. This is the current required by a single TTL input. Figure 3-15 shows the output of one TTL gate connected to one of the inputs of another. In Fig. 3-15A the output of the first gate is high. As we said above, this means that the output voltage will be somewhat higher than half the supply voltage, usually in the order of +3V. All that this signal has to do is

to hold the emitter in the input of the following stage at a high level. You will remember that the input transistor doesn't act like a transistor at all under this condition. Therefore, in theory, no current is required. In practice there will be some leakage current and it has been agreed that the maximum current required by a gate input under this condition is 40 μA.

Figure 3-15B shows the opposite situation where the output of the driving gate is in the low state. Its output voltage is very nearly OV. Under this condition, current will flow from the input pin of the driven gate to ground through transistor Q4 of the driving gate. The current that must be carried by this transistor is 1.6 mA. Thus, a unit load can be defined as:

High	40 uA
Low	1.6 mA

Another, somewhat confusing term that you will sometimes encounter is *fan-in*. This doesn't mean the number of outputs that can be connected to the input of a gate. It means the amount of current required at the input of a gate in the high and low states. All modern TTL logic has a fan-in of one unit load. Some types of devices may require a greater or less current; it would be rated as having a fan-in of two.

Standard TTL usually has a fan-out rating of 10. This means that a standard TTL gate may be connected to as many as 10 different inputs without any of its operating characteristics going out of limits. Some larger devices, usually called buffers, have a fan-out rating as high as 30.

NOISE IMMUNITY

An important rating of any digital device is its ability to reject the noise that will inevitably be present in digital systems. This parameter is usually specified in terms of what is called *noise margin*. This is illustrated in Fig. 3-16. In Fig. 3-16A, the line between the two gates is high. The way in which noise could disrupt this situation is for a negative-going noise pulse to drop the voltage on this line low enough so that gate B would think it was a low signal. The immunity to this type of noise is called the high-state noise margin, V_{NH}. The formula for calculating this voltage is given in the figure as:

$$V_{NH} = V_{OH} - V_{IN}$$

Thus, if the lowest high-level voltage out of gate A is +3V, and the minimum voltage that gate B will think is a high level is 1.8V, the

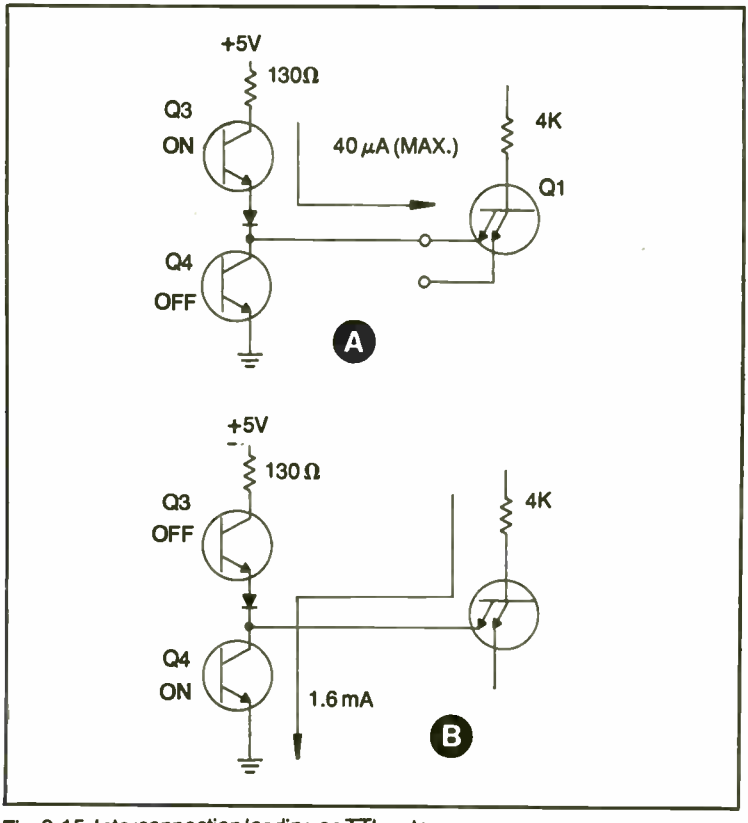


Fig. 3-15. Interconnection loading on TTL gates.

high-state noise immunity is:

$$+3V - +1.8V = 1.2V$$

This means that any negative-going noise pulse on the line between the gates of Fig. 3-16A will not cause any trouble as long as the pulse is smaller than 1.2V.

In Fig. 3-16B we show the situation where the line between the gates is at a low level. The effect of noise on this arrangement would for a positive-going noise pulse to raise the voltage on this line high enough so that gate B would think it was a high-level signal. The formula for the low state noise margin is:

$$V_{NL} = V_{IL} - V_{OL}$$

Thus if the highest value of voltage that gate B will think is a low level is 0.7V and the highest level that the output of gate A will have in the low state is 0.5V, the low-state noise margin will be:

$$0.7V - 0.5V = 0.2V$$

Strictly speaking, the noise margins that we have been talking about are DC noise margins. But, at the speed at which TTL logic operates, a pulse having a duration of only one microsecond may be thought of as DC. Very short pulses having durations of only a few nanoseconds may be so short that they will be gone before the circuit has a chance to respond. In a case like this, the noise margin would be higher than the value that we have calculated.

OTHER INPUT CONSIDERATIONS

From our discussion of the way in which a TTL gate works, we know that if nothing is connected to any input of a gate, the effect is the same as if that input were connected to a high level. In other words, all TTL inputs can be considered to be in a high state until something pulls them low. Because of this, many experimenters and, unfortunately some designers, will leave any unused input pins on AND and NAND gates floating because this is the same as connecting them to a high level. This is shown in Fig. 3-17A. While this arrangement will usually work, it isn't a good idea, because the unconnected pin will act like an antenna and may well pick up noise pulses that will cause problems.

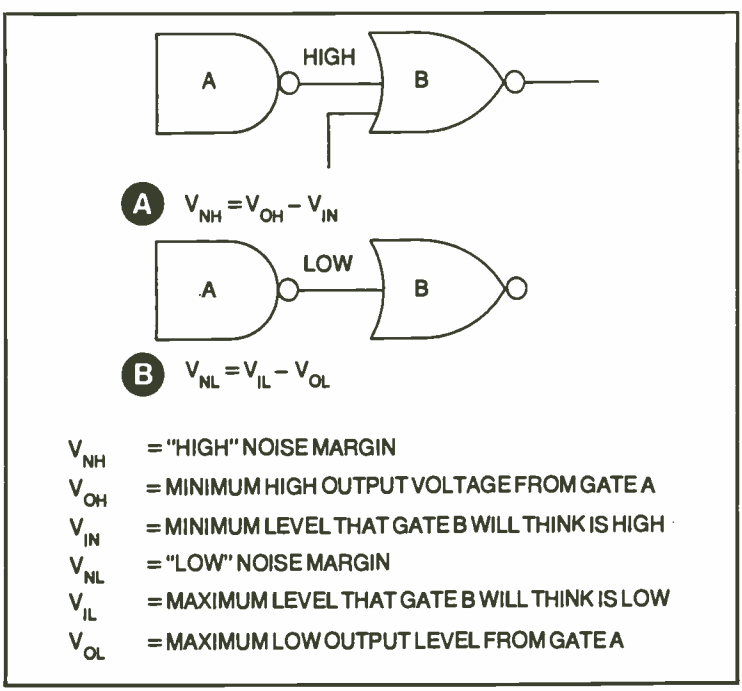


Fig. 3-16. Noise margins calculation.

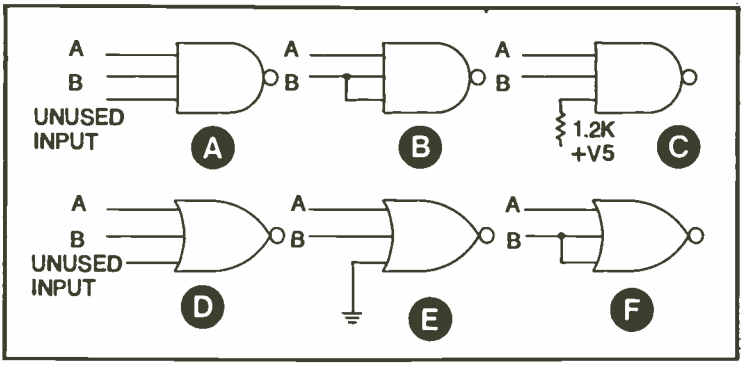


Fig. 3-17. Six ways of handling unused TTL inputs.

A better practice is to connect any unused pins of an AND or NAND gate to one of the used inputs as shown in Fig. 3-17B. The only time that this arrangement will not work is when the added input might exceed the fan-out rating of a driving stage. In such a case, the unused pin can be connected to the positive supply as shown in Fig. 3-17C. The resistor is included to protect the input against any transients that might be present on the power line.

With an OR or NOR gate, unused inputs cannot be left floating as in Fig. 3-17D. This arrangement would cause the output to be low at all times regardless of the states of inputs A and B. The way to handle this situation is to either ground the unused pin as in Fig. 3-17E or to connect it to one of the used input pins as in Fig. 3-17E.

ANOTHER LOOK AT THE OUTPUT CIRCUIT

We have noted that the high-level output of a TTL gate is at least 2.4V and usually around 3V. When the output is used to drive the input of another TTL gate, this is fine. The output is high enough to actuate the input with a good noise margin. There are, however, instances when we will want to use the output of a TTL gate to drive something else. In such a case, it might be useful if we were able to raise the level of the output voltage. This can be accomplished with the arrangement of Fig. 3-18. Here, an external pull-up resistor is added even though the gate has a regular totem-pole output. This external resistor will pull the high-state output voltage to nearly the full +5V supply voltage.

Earlier we stated that the totem-pole output of a TTL gate was arranged so that when either of the transistors was turned on, the other would be turned off. This is true, but there is a joker that can

potentially cause problems. It revolves around the fact that a transistor that is in the off state can be turned on faster than a saturated transistor can be turned off.

Figure 3-19 shows our familiar totem-pole output stage again. Assume that the output is low; that is, transistor Q3 is turned off and transistor Q4 is turned on into saturation. Now when the gate starts to switch so that the output will go to a high state, transistor Q3 will turn on rather rapidly. At the same time, transistor Q4 starts to turn off, but inasmuch as it is saturated, some time will be required to remove all of the charge carriers. As a result, Q4 will not be fully off until a few nanoseconds after Q3 has turned on.

What all this means in practical terms is that when a TTL stage switches so that its output goes from low to high, there will be a very short period of time when there is a very low resistance path between the supply voltage and ground. Thus there will be a very short current pulse on the supply line. This pulse is very short in duration—never more than about 10 nS; but when a circuit has several gates that switch at the same time, these pulses can cause troublesome noise glitches on the supply lines. There are ways of coping with the situation and these will be treated in the chapter on power supplies and noise.

OTHER MEMBERS OF THE TTL FAMILY

So far we have been talking about what is usually called standard TTL. The integrated circuits in this branch of the family have numbers like 7400 or 5400 for the version that will meet military specifications. For example, a 7408 IC contains four 2-input AND gates. Standard TTL has proven to be one of the best logic families for many applications. It is easy to imagine, however, that there applications where it is worthwhile to sacrifice one of the operating characteristics in order to get an improvement in another. For example, it might be worthwhile to sacrifice some

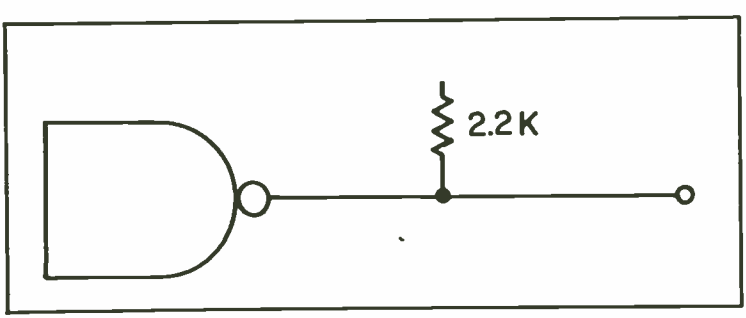


Fig. 3-18. A pull-up resistor used to raise the high-state voltage from a TTL gate.

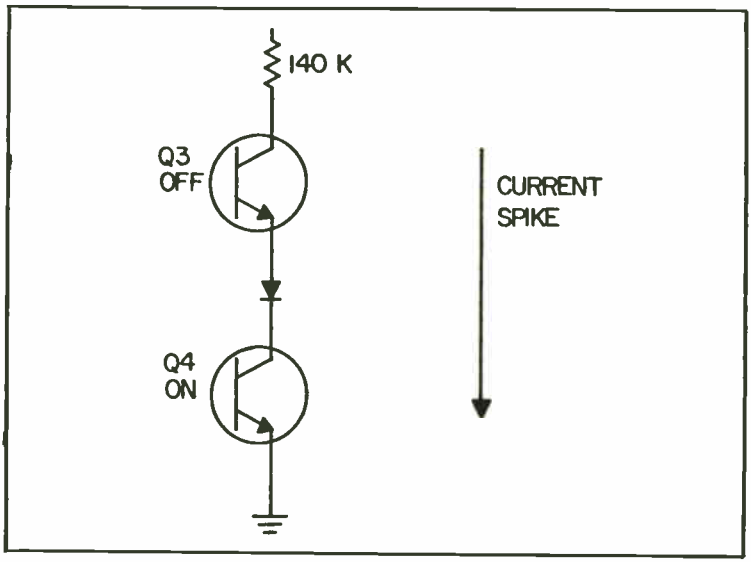


Fig. 3-19. Switching can induce a transient spike in a TTL output.

speed of operation in order to get a circuit that doesn't dissipate as much power. As a result of factors such as this, several branches of the TTL family have been developed. Most of these either operate faster, or consume less power.

As a standard of comparison of the various branches of the family we can assume that a standard TTL gate will have a propagation delay in the order of 10 nS and will have a power dissipation of about 10 mW per gate.

Low-Power TTL

In this branch of the family, the ICs are about the same as those in standard TTL except that the values of all of the resistances have been increased. The result is that the power dissipation is reduced to about 1 mW per gate. The price that we pay for this reduction in power dissipation is that the propagation time is increased to about 23 nS. ICs in this branch of the family have the letter "L" in the number. Thus a type 74L00 would be the low-power version of the Type 7400 quad two-input NAND gate.

High-Speed TTL

At the opposite extreme, we have high-speed TTL where the values of all of the resistors have been decreased to speed it up. It is indeed faster, having a propagation delay of only about 6 nS, but

the power dissipation is increased to about 23 mW per gate. ICs in the series have the letter "H" in their numbers.

Schottky TTL

As we have noted previously, one of the factors that limits the speed of TTL is that transistors are switched into saturation and it takes time to get them out of saturation. In the Schottky TTL branch of the family, the transistors are kept out of saturation by connecting Schottky diodes across them. A Schottky diode is a very fast diode that has a forward voltage drop of only about 0.3V as compared with the typical 0.7V of a regular silicon diode. These diodes keep the transistors out of saturation so that they can be turned on much more quickly, thus increasing the speed of operation, and at the same time improving the speed-power tradeoff.

A Schottky TTL gate will have a propagation time of about 3 nS and a power dissipation of about 23 mW. These ICs have the letter "S" in their type numbers.

Low-Power Schottky TTL

The Schottky TTL has been modified to reduce its power consumption. The transistors are shunted by diodes to improve their speed, but the values of the resistances in the circuit have been increased to reduce power dissipation. The result is that the speed is about the same as standard TTL, but the power-dissipation is reduced to about 2 mW per gate.

CMOS INTEGRATED CIRCUITS

There is another family of digital integrated circuits which are of interest to broadcasters. Although not as popular as TTL, CMOS ICs are widely used. CMOS stands for complimentary MOS. The name comes from the fact that these ICs use complimentary MOS field-effect transistors. The MOS comes from the way that the device is built. As shown in Fig. 3-20, it consists of a layer of metal on top of a layer of oxide which is an insulator, which in turn is on a layer of semiconductor material; hence the name *M*etal *O*xide *S*emiconductor.

There are many different types of MOS ICs. The one that we are concerned here is the CMOS variety. The reason for the word complimentary will soon become apparent.

Most texts describe CMOS ICs in terms of the details of their construction. We won't bother with details, but will work out a description in terms of the schematic symbol for the MOS FET.

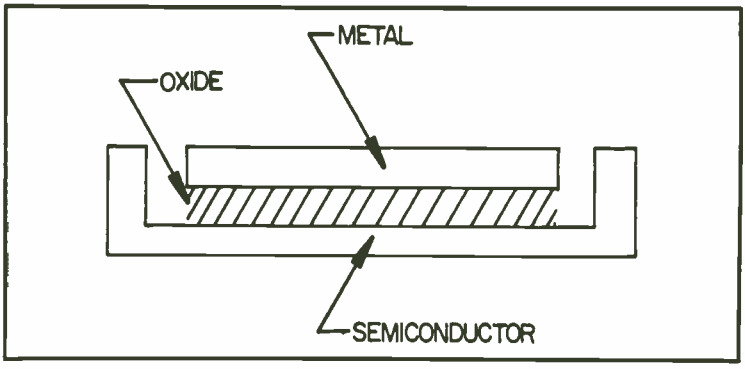


Fig. 3-20. MOS structure.

Figure 3-21 shows the symbol for a N-type, enhancement-mode FET. It has four electrodes—a source, a drain, a substrate, and a gate. We won't be particularly concerned with the substrate, except to note the fact that the arrow points inward, which tells us that it is an N-channel device, and that in N-channel devices the substrate is connected either to the source or to ground.

The device of Fig. 3-21 is intended to serve as a switch. We want to be able to either complete the circuit between the source and drain or to leave it open. In this particular device, if we connect the gate to ground, the circuit between the source and drain will be open. If we apply a positive voltage to the gate, the circuit between the source and drain will be closed. There will be some resistance, but in general it will be small.

Now we have a switch that is controlled by the voltage that we apply to the gate. We should note carefully that the gate doesn't make an electrical connection to the other electrodes. It is just like one plate of a capacitor. Thus this switch will have a very high input impedance and the gate will draw no current most of the time and only a very small current when it is charging.

Here is where we get to the word, complimentary. Figure 3-22 shows the symbol for a P-channel, enhancement-mode FET. We know that it is a P-channel device because the arrow points away from the substrate. This device is complimentary to the N-channel device of Fig. 3-21. If we connect the gate to ground, it will conduct between the source and drain. If we make the gate positive, we will open the circuit between the source and drain.

Now we have the makings of all sorts of digital logic circuits. We have two devices where with the same voltage we can open one switch and close another. Figure 3-23 shows a CMOS inverter that

consists of the two types of FETs that we have just described. The one at the top is a P-channel device that will turn off when its gate becomes positive, and the one at the bottom is an N-channel device that will turn on when its gate becomes positive. It is easy to see that if we keep the two gates at ground potential, the device at the top of the figure will be turned on and the one at the bottom will be turned off. This means that the output will be high—at the power supply voltage. Now if we were to make the gates positive, the situation would be reversed. The device at the top would be turned off and the one at the bottom would be turned on, bringing the output to ground potential. Thus we have the truth table of an inverter shown in the figure.

Before getting into any more detail about the operation of a CMOS stage, let's look at a logic gate. Figure 3-24 shows a CMOS two-input NAND gate. It uses two P-channel and two N-channel enhancement-mode FETs. We can again think of each of these FETs as a switch. Thus if either P1 or P2 is turned on, the output will be connected to the positive supply. Similarly, if both N1 and N2 are turned on, the output will be connected to ground. Naturally, we want to avoid the situation where the output is connected to the positive supply and to ground at the same time. The devices are connected so that we will indeed avoid this situation.

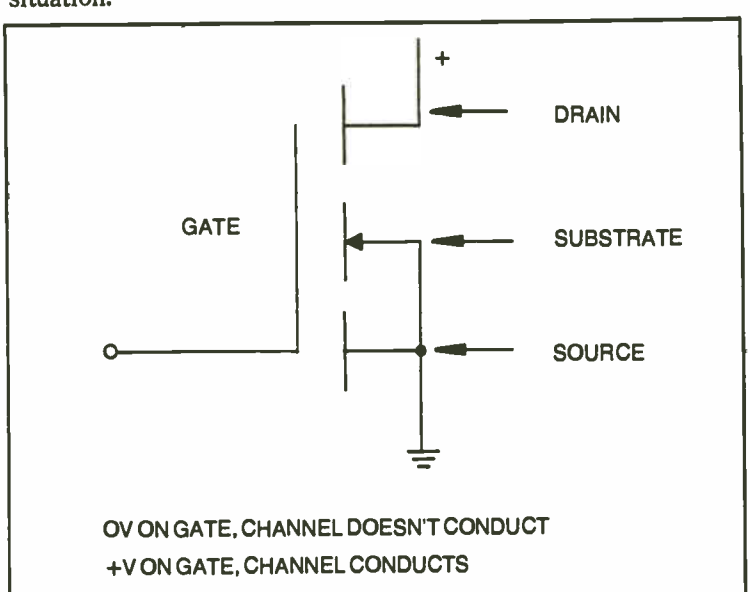


Fig. 3-21. N-channel enhancement-mode FET.

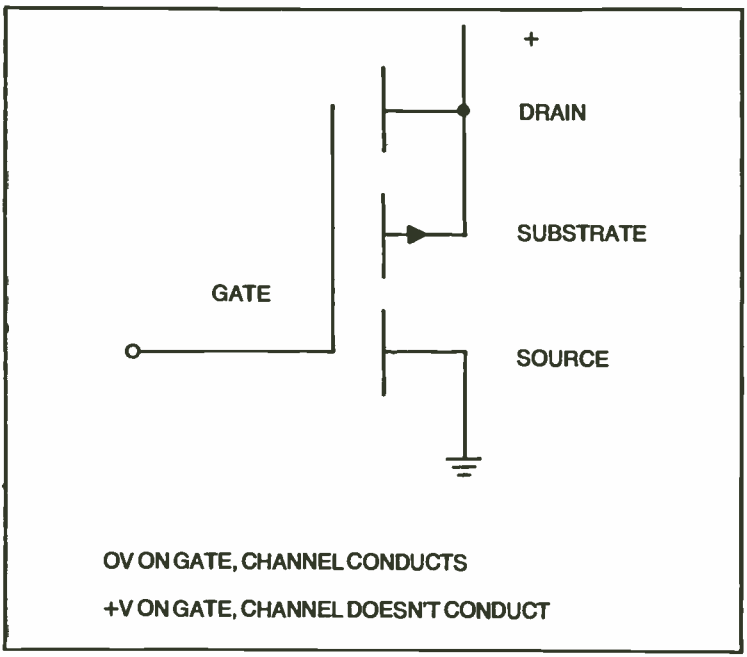


Fig. 3-22. P-channel enhancement-mode FET.

One of the easiest ways to make a truth table for a gate like that shown in Fig. 3-24A is to contrast the small table shown in Fig. 3-24B, which shows which transistors are turned on and off when inputs A and B are made high and low. By just glancing at the table we can see that the only condition under which both N1 and N2 are turned on is when both inputs A and B are high. Inasmuch as both of these FETs must be on for the output to be connected to ground, A and B must both be high for the output to go low. A further inspection of this table will show that under all other input conditions either P1 or P2 will be turned on. This means that the output will be high. This leads to the truth table given in Fig. 3-24C, which will be recognized as the truth table for a NAND gate.

CMOS CHARACTERISTICS

Now that we have seen how logic gates can be made with CMOS, let's look at some of its characteristics. Probably the most important feature is the fact that there is never a direct low-resistance path between the supply voltage and ground. This means that a CMOS gate consumes very little power. In fact, when the gate is not changing state, but remaining in one state, there is

almost no power at all. This is one of the most attractive features of CMOS.

The input of the CMOS gate is interesting. The only connections to the input pins are the gates of the MOS FETs. These, you will remember, are like the plate of a capacitor. There is no direct conducting connection between the gate of such an FET and its channel. Thus when the circuit is not changing state, there is essentially no input current. For this reason, we define a unit CMOS load as a capacitance of about 5 pF, which includes the capacitance of the case. A typical CMOS gate will have a fan-out of 50.

One bad feature of this type of input is that time will be required to charge the input capacitance. This characteristic limits the speed of CMOS. It is still pretty fast, however, being able to handle pulse rate of about 5 MHz.

CMOS VOLTAGE LEVELS

The CMOS gate will operate over a wide range of supply voltages, usually anything between 3 and 12 or 15V. The

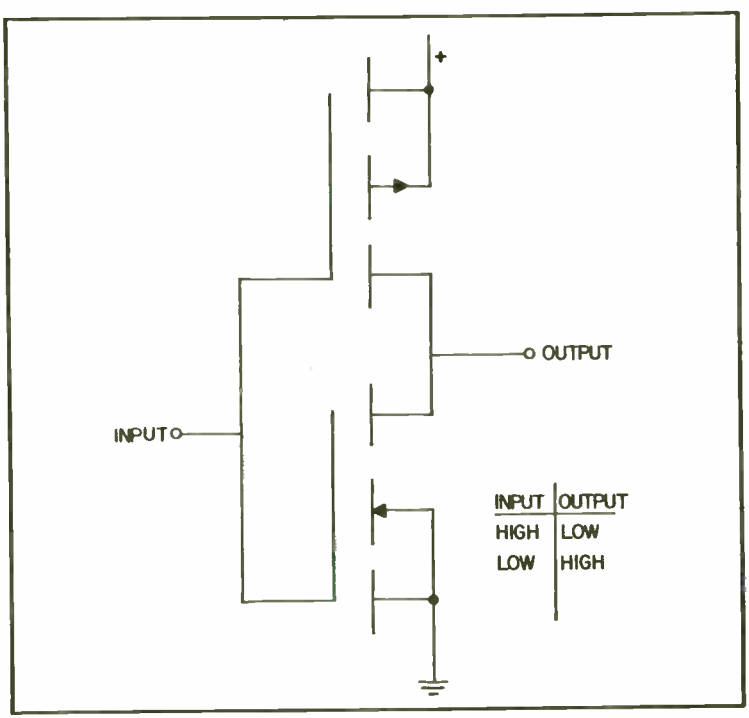


Fig. 3-23. CMOS inverter.

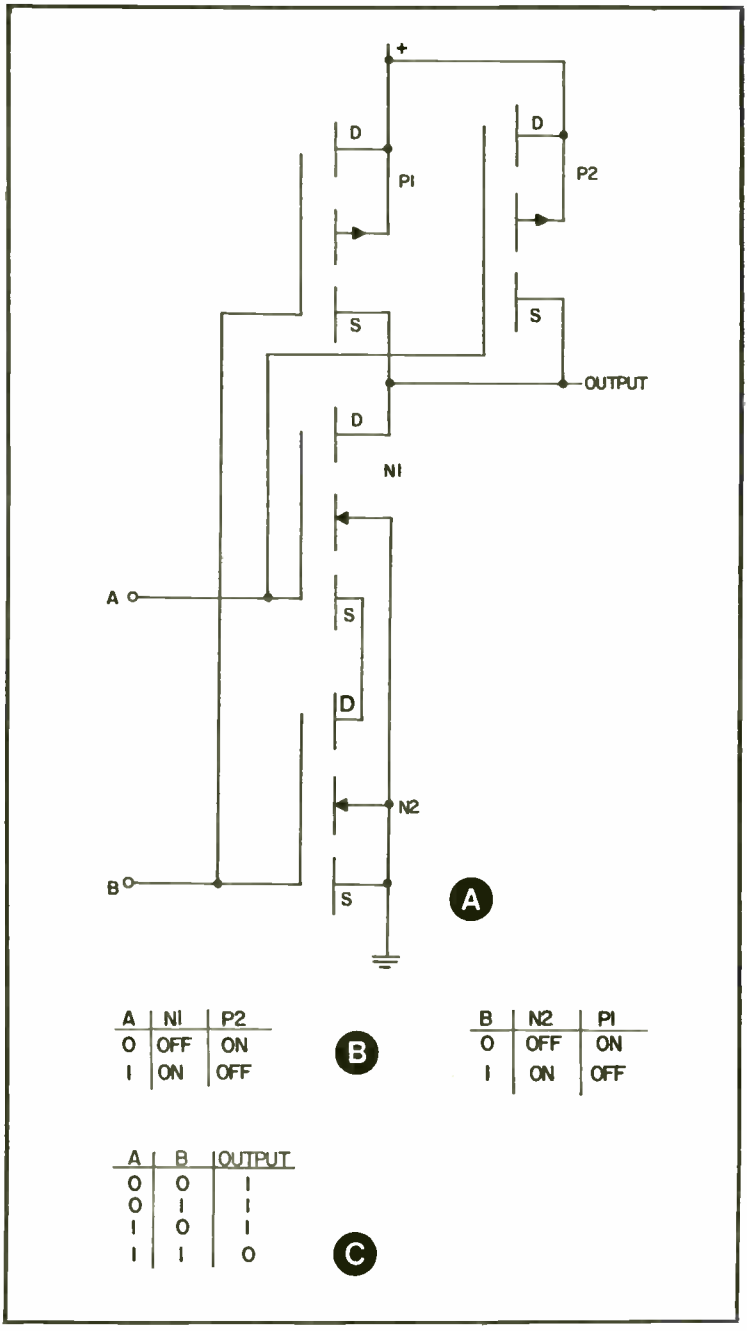


Fig. 3-24. Diagram of a CMOS 2-input NAND gate.

output-voltage levels depend on the load connected. When one of the FETs is turned on it will have a resistance of somewhere between 200 and 400 ohms. This is no problem when driving the input of another gate because it will not draw any significant current. It can be a problem when driving some external device of some other logic family such as TTL. Special CMOS devices having a greater output capability are available for this purpose.

CMOS NOISE IMMUNITY

Another very attractive aspect of CMOS is that it can have a very high noise immunity. This can be best seen by looking at the transfer characteristic of a typical CMOS gate. Such a curve is shown in Fig. 3-25. The horizontal axis of the graph is the voltage applied between the input and ground. The vertical axis is the output voltage. Three sets of curves are shown, for supply voltages of 5V, 10V, and 15V. The interesting feature of these curves is that the output voltage doesn't change at all until the input voltage reaches nearly 50% of the supply voltage. Then a small additional

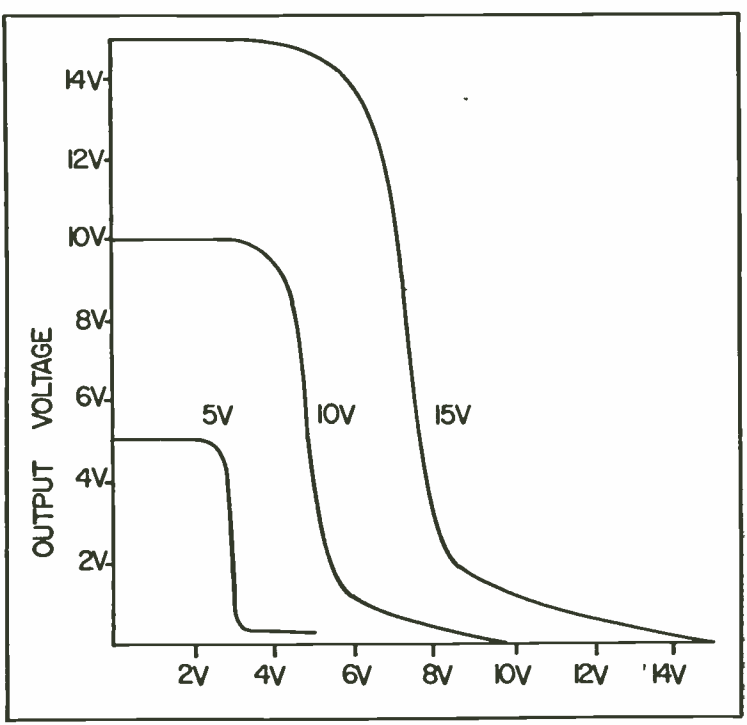


Fig. 3-25. Characteristic curve of a CMOS inverter.

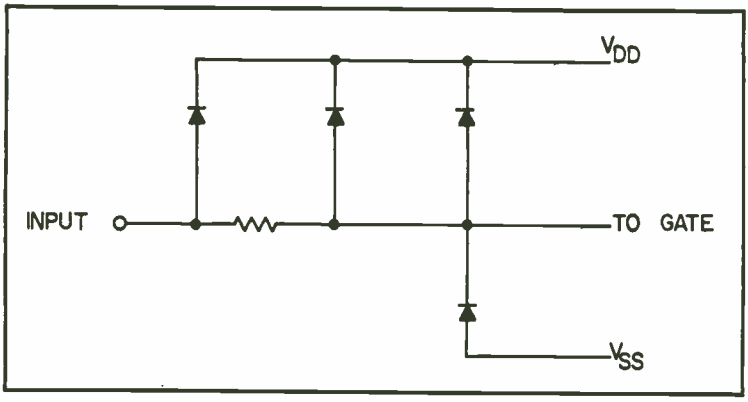


Fig. 3-26. Typical input circuit arrangement used to protect a CMOS gate.

change in input voltage will drive the output to the final state. This means that in a system with a supply voltage of 10V, an input signal of nearly 5V is required to switch the gate from one state to another. Thus the noise margin is nearly half of the supply voltage.

CMOS INPUT PROTECTION

In describing the construction of a MOS FET we noted that the metal gate was insulated from the channel by a thin layer of oxide which served as an insulator. Inasmuch as this layer is really very thin, it can easily be damaged by a voltage higher than it was designed to withstand. Although circuits can be designed to use safe voltage levels, there is still the possibility that CMOS ICs can be damaged by high voltages from transients and static electricity. In fact, CMOS devices are usually packed in conductive packages so that stray static charges cannot damage the input circuits.

Most CMOS ICs have some internal protection against high transient voltages. A typical arrangement is shown in Fig. 3-26, where diodes are used to provide protection. This circuit will indeed provide a great deal of protection for the high-resistance gates. However, because of this circuitry, it is important that power be applied to CMOS gates before the inputs are made high. Otherwise the gate can be permanently damaged.

Chapter 4
Flip-Flops, or Circuits That Remember

The logic gate that we considered in Chapter 2 is the fundamental building block of all digital systems. In fact, you could build any digital system, no matter how complex, with nothing but NAND or NOR gates.

In Chapter 2, we thought of a logic gate as a device that had one of two possible output levels, depending on a combination of input levels. When gates are used in this way, the system is said to use *combinational logic*. There is another way to use digital circuit elements where the state of the output of a device depends not only on the present inputs, but also on what the input happened to be some time ago. This type of arrangement is called *sequential logic*. The simplest building block where the present output level depends on some past input level is called a flip-flop. Essentially a flip-flop is a circuit element that has two steady states. Various inputs can be used to switch the element from one state to the other.

Figure 4-1 shows a block diagram of a flip-flop. Note that it has two output terminals, labeled Q and $\overline{Q}$. The little bar over the second Q can be though of as meaning "not." It can also be thought of as meaning the same as an active low indicator. Thus the two outputs of our flip-flop are Q and not Q. It is apparent that, in general, when Q is high, $\overline{Q}$ will be low and vice versa.

In our block diagram we have shown several inputs to the flip-flop. We haven't stated what any of them might be called or how any entered into the operation of the device. In fact, there are many different types of flip-flops and they differ mainly in the types of inputs that are provided and how they affect the operation.

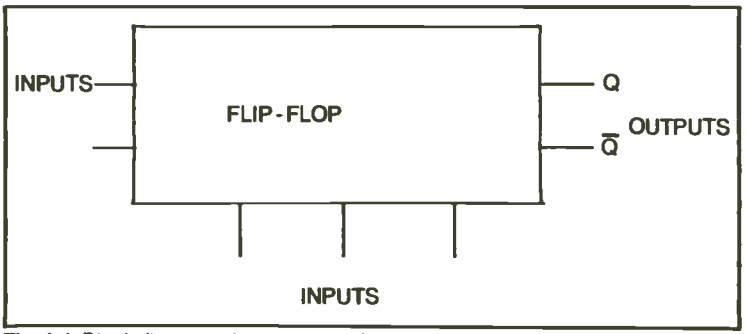


Fig. 4-1. Block diagram of a generalized flip-flop circuit.

THE BISTABLE LATCH

Figure 4-2 shows how a very simple flip-flop can be made with two NAND gates. As with gates, we can find out how the circuit works by constructing a truth table. In this instance the truth table is apt to be a little confusing to construct, because one of the inputs of each of the gates is the output of the other. Unless you are careful, you will find yourself going around in circles.

The easy way to construct a truth table for the circuit of Fig. 4-2 is to use the rule that with a NAND gate, as long as either of the inputs is low, or logical 0, the output will be high. With this bit of information, we can start with the top gate in the figure and assume that the input labeled "R" is low or zero. This tells us immediately that the output of this gate, which is Q, must be high or 1. Thus we can fill in the first line of the truth table of Fig. 4-2B. Inasmuch as Q, which is one of the inputs of the lower gate, is high, and we will assume that S is high, we know that $\overline{Q}$ will be low, or 0. We can use this information to construct the first line of the truth table to the right in Fig. 4-2B.

Now to the opposite situation. Let's make S low. This tells us right away that $\overline{Q}$ will be high. Now, inasmuch as $\overline{Q}$ is one of the inputs of the top gate, we know that Q will be low. We now have two lines of a truth table for the circuit, and so far there isn't anything exciting. In fact, there is no indication that this circuit can remember anything.

Now let's try a little further manipulation of the data as shown in Fig. 4-2C. We will start out with R low, S high, Q high, and $\overline{Q}$ low. Let's change R from low to high. Before we made this change, both of the inputs were low. Now one of them will be high and the other low. From what we know about a NAND gate, we know that if one or both of the inputs is low, the output will be high. In other

words, when we make R high, nothing happens. The circuit will remember its previous state and will stay there.

By a very similar line of reasoning, we can take the situation shown in Fig. 4-2D and change S from low to high. Again, nothing will happen. Again the circuit remembers.

Let's summarize all of this in a complete truth table as shown in Fig. 4-2E. On the top line of the table we see that when both R and S are high, the outputs will have the same state that they had before both R and S were made high. This is the memory feature of the circuit. On the next two lines we show the output states when R and S have opposite states. Finally, on the last line we indicate that if both R and S are low, both the outputs will be a high, a condition that we said wouldn't happen when we started talking about flip-flops. In fact this condition can actually exist, but it will not be remembered. In practice, either R or S will change before the other and this will be the condition that the circuit will remember.

We have been using the letters R and S without any explanation of where we got them. Originally, they came from the terms *set* and *reset*. Be careful when using them, however, because some flip-flops might be set—Q made high—by a high signal and others by a low signal. We'll have more to say about this later.

Before going further, it is interesting to note that we could also build a bistable latch using NOR gates instead of NAND gates. The circuit is shown in Fig. 4-3. To construct the truth table for this arrangement use the fact that in a NOR gate, if either or both of the inputs are high or 1, the output will be low, or 0. The resulting truth table is shown in the figure. It is interesting to note that with this arrangement the memory feature occurs when both R and S go to 0.

The two preceding examples of latches are useful not only because they give a little insight into how a flip-flop can remember a past input, but also because the circuits are practical. It is handy to be able to build a little latch out of a couple of NAND or NOR gates.

Figure 4-4 shows the block representation of an R-S flip-flop, which is similar to the latches that we have just discussed. You will find many different types of truth tables for flip-flops. They all tell the same thing, but often they are set up differently. In the truth table of Fig. 4-4, the third column is labeled Q_{n+1}. This column shows the state of the Q output after the inputs have been set as shown in the preceding columns. The expression Q_n indicates the state of the Q output with earlier inputs. The symbols have

meaning only in the first line of the table, which tells us that when R and S are both low, the state of the Q pin will not change. It will be whatever it was when R and S had different values.

Another interesting thing about the truth table is the comment "indeterminant" in the fourth row. We said earlier that we couldn't be exactly sure of what the output would be under this condition. Sometimes the truth table will label this situation "disallowed," meaning that the R and S inputs should not be allowed to go high at the same time. The reason is that, not knowing what output this would produce, we could not be certain of how a system would operate.

Note that in the truth table of Fig. 4-4, nothing is shown about the state of the $\overline{Q}$ pin. There is no need to do this because it will always be just the opposite of the state of the Q pin.

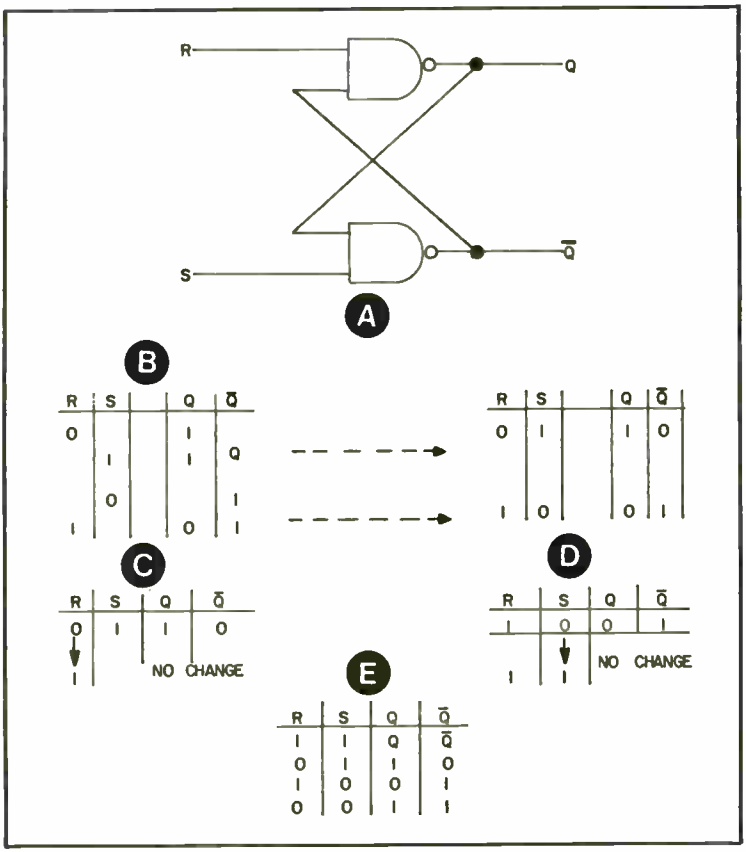


Fig. 4-2. Bistable latch circuit, using NAND gates, and truth tables.

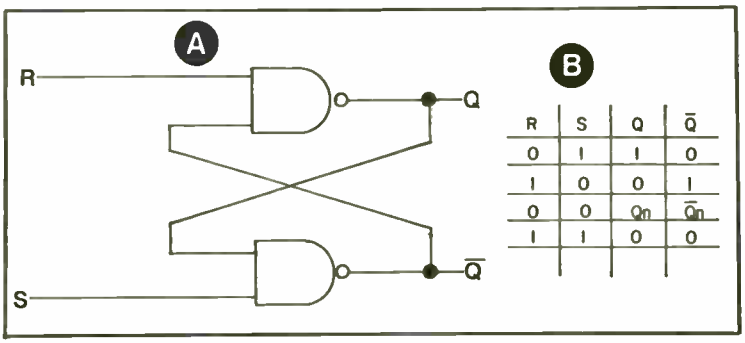


Fig. 4-3. Bistable latch made with NOR gates.

The flip-flop arrangements that we have described so far can be called *asynchronous*, because the output will change as soon as the input is changed. In general, inputs labeled set, reset, or clear are asynchronous. Changing their state will produce an immediate change in the output.

SYNCHRONOUS OR CLOCKED FLIP-FLOPS

There are many places in digital systems where it is desirable for all the flip-flops in a part of a system to change state at the same instant. This is called *synchronous* or *clocked* operation and synchronous or clocked flip-flops are used for the purpose.

Figure 4-5 shows a general block diagram for a clocked flip-flop. Note that at the bottom of the block, there is a new type of input called a clock input. In this type of flip-flop, the states of the various synchronous data inputs can be changed, but nothing will happen until something is done at the clock input. Usually a pulse is applied to this input to make the outputs change state if the proper

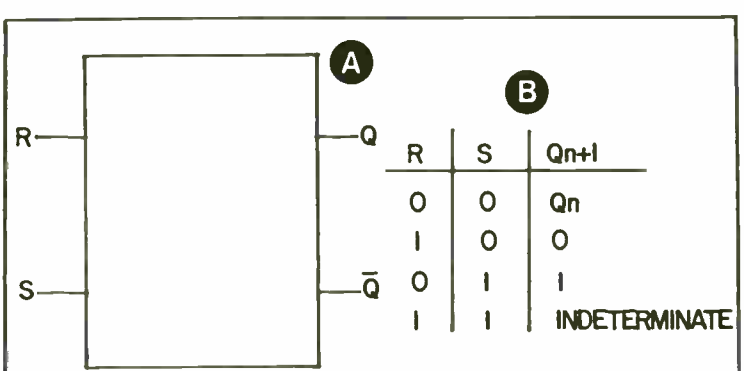


Fig. 4-4. R-S flip-flop with its truth table.

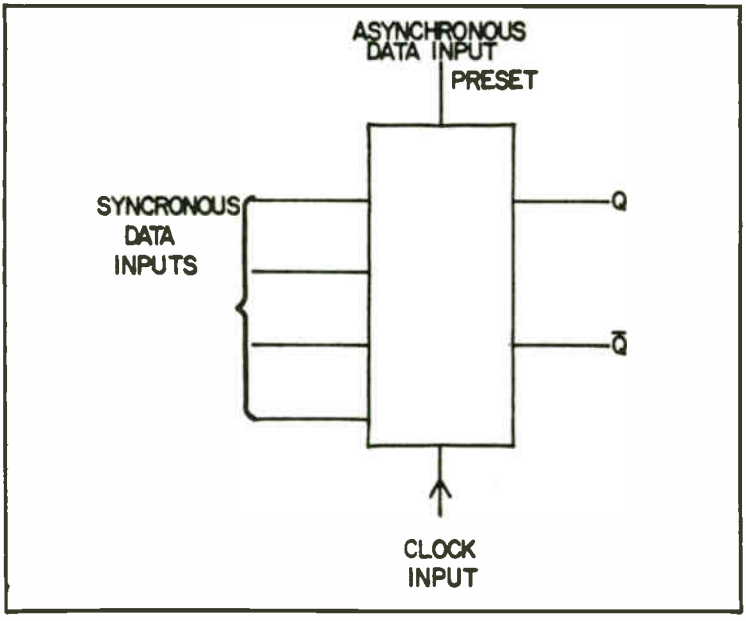


Fig. 4-5. Block diagram of a clocked or synchronous flip-flop.

combination of levels is present at the inputs. This type of flip can also have one or more asynchronous inputs such as the preset input at the top of the block.

There are several different types of data inputs that can be used with a clocked flip-flop, and usually the device derives its name from the types of inputs provided. There are three different types of clock action that may be used on any type of flip-flops. In general, any type of clocking may be used on any type of flip-flop. There is usually only one clock input on a flip-flop, but there may be many different data inputs. Designers of integrated circuits are continually coming up with new versions of the flip-flop that will simplify system design.

One type of clocking is called *edge clocking*. As shown in Fig. 4-6A, the flip-flop may change state when the clock pulse reaches a certain level on its positive-going edge. Or it may be arranged so that it will change state at a certain level on it negative-going edge. Both types of edge triggering are widely used, but not in the same unit. With edge triggering, the operation doesn't depend on the rise time or duration of the clock pulse, but when the rise and fall times are greater than about 150 nS, the noise immunity of the flip-flop will suffer.

Another type of clock action that is widely used is found in the *master-slave* flip-flop. It is called *level* or *master-slave* clocking. Figure 4-7A shows a rough block diagram of the master-slave flip-flop. It consists of two separate latches and at no time are any of the data inputs connected directly to the output. This results in very good isolation between input and output. When the clock pulse is applied to clock input, the flip-flop reacts in four distinct steps as shown in Fig. 4-7B. First the slave latch is disconnected from the master, then the data inputs are connected to the master, changing its state. Following this, the data inputs are disabled, and the data is transferred to the slave where it will appear at the output.

Although not widely used, still another form of clocking is available. It is called *AC* or *capacitive-coupled clocking*. Here, the clock pulse is capacitively coupled into the flip-flop. It responds to either the positive or negative rate of change of the pulse. With such an arrangement, the rise and fall times and the duration of the clock pulse are critical.

THE TYPE "D" FLIP-FLOP

Figure 4-8 shows the Type D flip-flop with its truth table. This flip-flop has one data input which appropriately is labeled D. The truth table is a little different than the ones that we have been using

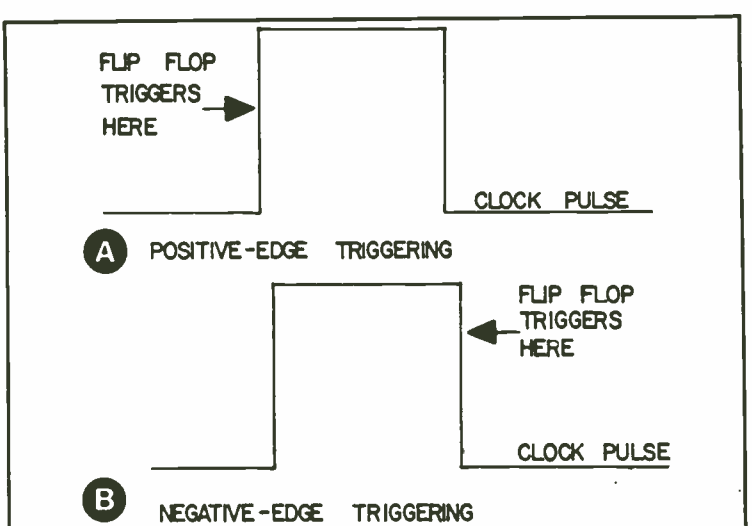


Fig. 4-6. Flip-flop may be edge-triggered by either the positive-going or negative-going edge of the clock pulse.

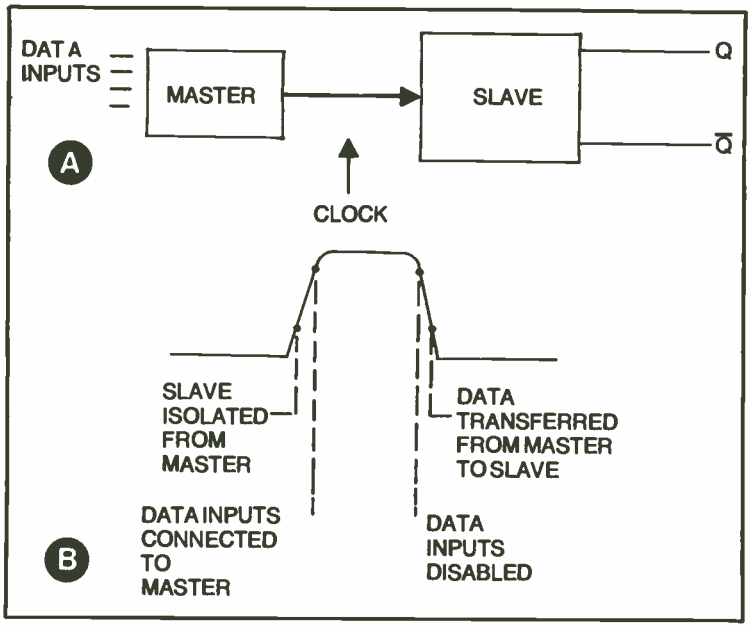


Fig. 4-7. Block diagram and waveform showing clocking of a master-slave flip-flop.

in that we have shown two times, T_n and T_{n+1}. T_n means that we are interested in what state the D input is at, at the time just before the clock pulse arrives. T_{n+1} is the time immediately after the clock pulse, when we are interested in the state of the output.

The truth table tells simply that after the clock pulse, the Q pin will have the same state that the D pin had immediately before the clock pulse arrived. Type D flip-flops are handy for latching data at a given time. The data input is connected to the D pin. When we are ready to latch on to it, we can actuate the clock pin. Either both the Q and $\overline{Q}$ pins or just the Q pin may be available on the package and several flip-flops may be contained in a single IC. Sometimes one or more asynchronous inputs, such as a preset or a clear, may also be provided. Being asynchronous, inputs to these pins can force the output to a given state even before the flip-flop is clocked.

TYPE "T" FLIP-FLOP

Figure 4-9 shows an interesting flip-flop that may not be available in an integrated circuit package in a particular logic family. However, it can be made from other flip-flops and is used in

some large-scale integrated circuits. The name, T flip-flop, comes from the word toggle. The truth table tells us that if the T input is low, the Q output will not change when the flip-flop is clocked. On the other hand, if the T input is made high, the Q output will change state whenever the flip-flop is clocked. This arrangement is handy in counting circuits.

CLOCKED RS FLIP-FLOP

Th RS flip-flop shown in Fig. 4-10 is very similar to the RS flip-flop that we discussed earlier, except that it is clocked rather than asynchronous. The R and S inputs in this case are synchronous data inputs. In some instances, they have designations other than R and S.

A clocked RS flip-flop may also have asynchronous inputs such as present and clear.

THE TYPE J-K FLIP-FLOP

Figure 4-11 shows what is probably the most versatile flip-flop of all. It is called a J-K type because it has two data inputs labeled J and K. Whereas there seems to be some reason why most of the other input designations such as R, S, T, and D were chosen, no one seems to know where the letters J and K originated. Very often J-K flip-flops also have asynchronous inputs as shown. The truth table shown in Fig. 4-11B shows how the states of the J and K inputs before the clock pulse will affect the state of the Q pin right after the flip-flop has clocked.

Figure 4-11C shows another rather interesting table, which is sometimes called an excitation table. What this table shows is how we must connect the J and K pins if we have a certain state of the Q output and want another state right after the clock pulse. The "X" in the table represents a "don't care" state. For example, looking at

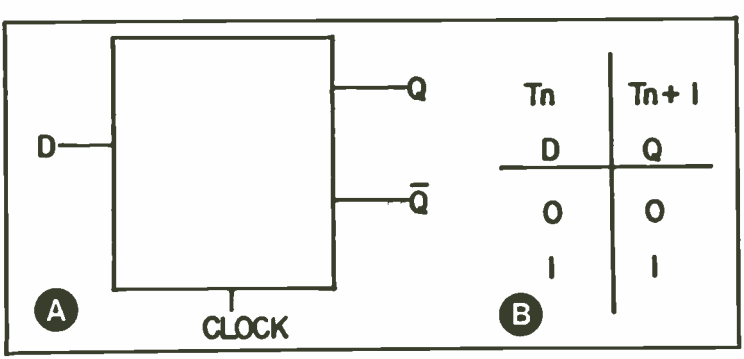


Fig. 4-8. Block diagram of a type D flip-flop and its truth table.

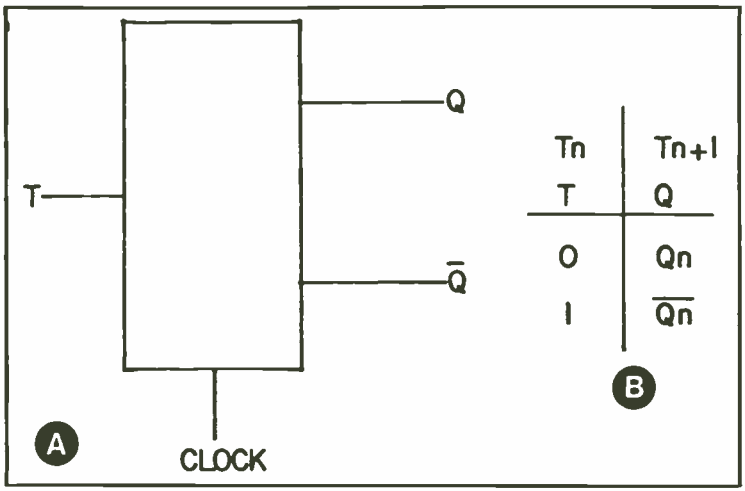


Fig. 4-9. Block diagram of a type T flip-flop and its truth table.

the second line of the table, we see that if Q is 0 before clocking and we want it to be 1 right after the clock pulse, we must connect the J pin to a high level. The X in the K column means that the flip-flop doesn't care in the slightest what state the K pin has.

TIMING CONSIDERATIONS

In a combinational logic system consisting only of various logic gates, timing may be relatively unimportant. For example if one of the high signals applied to the inputs of a 2-input NAND gate is a little late in arriving, the output will be a little late in going to the low state, but this might have no affect at all on the operation of the system. On the other hand, in a clocked system using synchronous flip-flops, a slight error in timing can make the difference between the system working and not working at all. If the system includes a counter, a slight extra delay in the arrival of a pulse might mean that the pulse wouldn't be counted at all.

When using clocked flip-flops, questions arise such as, "How long before the clock pulse arrives must the data be present at the data inputs?" and "How soon after a clock pulse can the data inputs be changed without causing problems?" To aid in answering these and similar questions, the designer of an integrated circuit specifies several timing parameters. Chief among these are the *propagation delay*, the *set-up-time*, and the *hold time*. We will consider each of these parameters first as applied to the various types of edge-triggered flip-flops and then applied to the master-

slave arrangement. Later we will discuss a problem called clock skew that applies equally to either type of flip-flop.

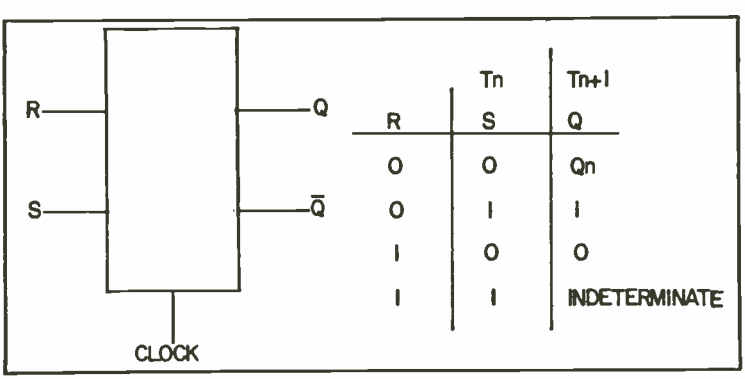
All of the parameters of an integrated circuit depend to some extent on the temperature, the supply voltage and various manufacturing tolerances. For this reason, most of the timing parameters will spread over a small range. The important thing is to look at each parameter and be sure that the worst possible case is taken into consideration.

Timing In The Edge-Triggered Flip-Flop

In an edge-triggered flip-flop, the output changes when the clock pulse passes through some transition. It may be either the positive-going or the negative-going edge of the clock pulse. For the moment let's assume that the flip-flop will trigger when the clock pulse goes from a low to a high level. This is a very common arrangement, although there are flip-flops that trigger on the negative-going edge of the clock pulse.

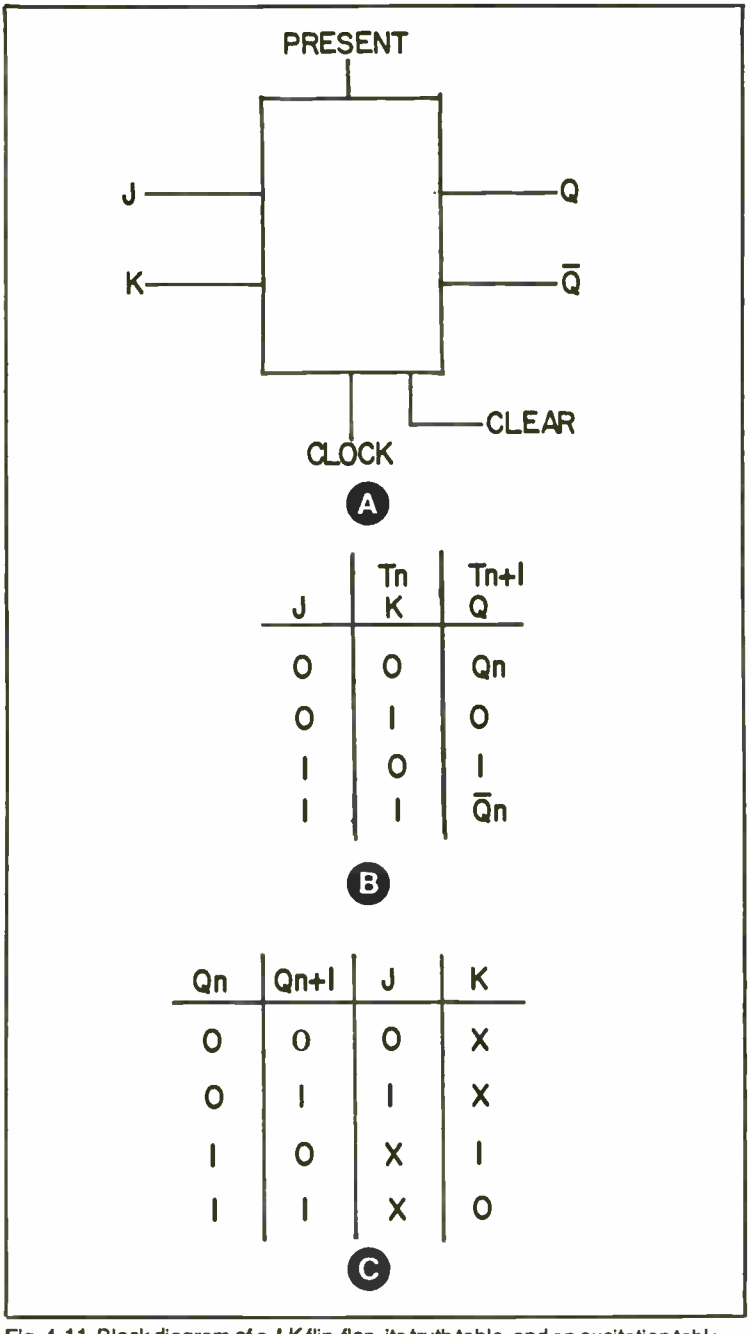
The propagation delay of an edge-triggered flip-flop is the time that elapses between the positive-going edge of the clock and when the output changes state. Usually the propagation time isn't the same when the output is going from high to low as when it is going from low to high. The two symbols usually used for propagation delay are T_{PHL} and T_{PLH}.

One of the most confusing timing parameters of a flip-flop is what is now usually known as the *set-up time*. The reason that it tends to be confusing isn't that it is a complicated subject, but rather that different manufacturers have used different methods of expressing it. In an edge-triggered flip-flop, the clock pulse is what starts the action. The action that is taken depends on what signals



	Tn	Tn+1
R	S	Q
0	0	Qn
0	1	1
1	0	0
1	1	INDETERMINATE

Fig.4-10. Block diagram of a clocked type R-S flip-flop and its truth table.



J	Tn K	Tn+1 Q
0	0	Qn
0	1	0
1	0	1
1	1	$\bar{Q}$n

Qn	Qn+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Fig. 4-11. Block diagram of a J-K flip-flop, its truth table, and an excitation table.

are present at the data inputs when the clock pulse arrives. Obviously, the signals must be present at the data inputs before the clock pulse arrives, but how long before? This is what the term set-up time is supposed to answer.

The set-up time is, then, the time that the signals have to be present at the data inputs before the positive-going edge of the clock pulse arrives. Some manufacturers have called this the maximum set-up time, and others have called it the minimum set-up time. By examining the range of variation in this specification, you can resolve any possible confusion.

The next question is how long after the positive-going edge of the clock pulse arrives must the signals at the data inputs be held still? This parameter is called the *holding time*. In many modern flip-flops the data inputs can be changed anytime after the active edge of the clock pulse. In such a case the holding time would be zero.

Timing In The Master-Slave Flip-Flop

Earlier, in connection with Fig. 4-7, we described how a master-slave flip-flop responds to the clock pulse. This might be summarized by saying that the flip-flop "looks at" the data inputs when the clock pulse is high, and transfers the data to the output terminals when the clock goes low.

Because of this action, the propagation times of a master flip-flop specified in a slightly different way. Usually the time between a data input and the output and the time between the clock and the output are specified.

Inasmuch as the data inputs cannot change while the clock is high, the set-up time is usually specified in terms of the width of the clock pulse. During the period when the clock is high, the master might be thought of as "looking" for an input to latch on to. This leads to a phenomenon known as "ones catching." If the data inputs should change during the period when the clock is high, the master will usually lock on to a 1, regardless of which way the data changes.

Chapter 5
Counters, Registers and Counting Systems

A counter is a collection of flip-flops and logic gates. It is really a type of a memory, because the outputs of the flip-flops remain unchanged when the input signal is removed. In a counter, flip-flops are connected in a sequence, so that they will keep track of the number of pulses applied to the input. One obvious application of a counter is to count the number of pulses. In fact, the ordinary digital clock or watch, operates on this principle. Another application of a counter is to act as a frequency divider.

GENERAL COUNTER OPERATION

Figure 5-1, shows a general block diagram of a counter. It has one input at the left, where a pulse train is applied. The output terminals at the right have logic levels that indicate the number of pulses that have been applied to the input. The counter also has a reset input. Applying a signal to this input will reset the counter to zero, so that it can begin counting all over again.

Figure 5-2A, shows a practical binary counter. We have arranged it somewhat differently than you'll see in most circuit diagrams, in that we have the input at the right and the outputs at the top. The reason for doing this is that it will make the binary number that we get as a result appear with the least significant digit on the right, the way we usually write numbers.

The way the counter operates is shown in Fig. 5-2B. Before the counting starts, all of the output wires Q_0 through Q_3 have no voltage on them. That is, they are all at a logic zero level. When the first pulse is received, the output labeled Q_0 goes high. If this is a TTL system, it will probably have about +3V on it. At the second

pulse, the output labeled Q_0 goes back to a low state and the output labeled Q_1 goes high. This process continues until the 15th pulse is received, when all of the outputs are high.

Looking at the tabulation of Fig. 5-2B, we see two things. First of all, we see that the high states of the outputs correspond to the number of the pulse that has been received. That is, when five pulses have been received, the states of the outputs are 0101, the binary number corresponding to 5. Thus, our counter counts the pulses applied to its input and displays the count in the form of high and low signals in binary form on its outputs.

The other thing that is obvious, is that the maximum number of pulses that our counter can handle at any one time, is 15. When it counts up to 15 input pulses, it runs out of output leads.

Obviously, a binary counter must have enough flip-flops, and enough outputs to handle the largest number that it will be called on to count. The relationship between the highest number that can be handled and the number of outputs is given by the expression:

$$\text{Maximum count} = 2^n - 1$$

Where n is the number of binary stages or flip-flops. The -1 is included in the expression, because although the highest binary number that can be displayed by n digits, is 2^n, one of these numbers is zero, which is where the counter is set before any pulses are received. This information is summarized in Fig. 5-3.

BINARY CODED DECIMAL OR BCD SYSTEM

Figure 5-4 shows a somewhat larger counter. It consists of two of the counters that we had in Fig. 5-2. Let's imagine that the blocks labeled "decoder" will take a binary number and from it derive a signal that will actuate a 7-segment readout, as those shown at the top of the diagram.

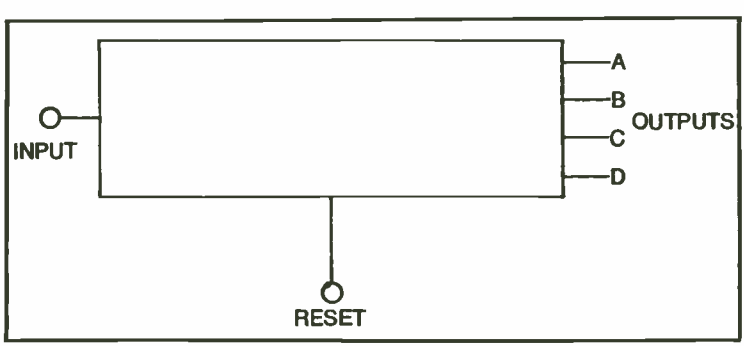
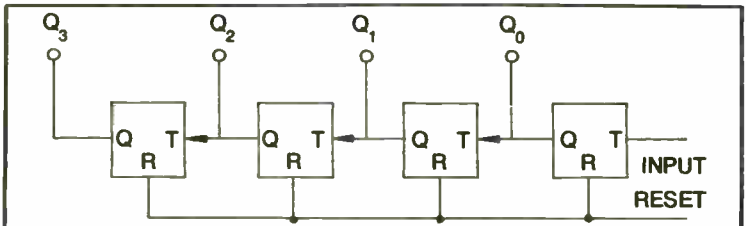


Fig. 5-1. Block diagram of a counter.



A COUNTER MADE OF FLIP-FLOPS

PULSE NO.	OUTPUTS Q_3	Q_2	Q_1	Q_0
-	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

B OUTPUT SEQUENCE

Fig. 5-2. Block diagram and operating table of a practical binary counter.

This system has a serious problem. It will do fine counting up to 9. When it gets to ten, we don't want the number going into the decoder and readout at the right. We want this one to read 0 and the one to its left, to read 1, so that between the two of them, they will display 10 on the tenth pulse. You might think that we could use a smaller counter, but the next smaller counter would have only three outputs and would count only to a binary 111, which is only 7.

The way around the dilemma is to decode the output of the counter so that when it gets to 9, it will reset to zero and provide a carry pulse to the counter at its left. The way this can be done is shown in Fig. 5-5. Here, the output of the first counter is connected to a 4-input NAND gate. The output of this gate is connected to the reset pin of the counter. Note that there is an active low indicator on this pin, meaning that when the pin goes to a low level, the counter will reset. Now the output of our NAND gate will only go low when all of the inputs go high. We want this to happen when the

NO. OF FLIP-FLOPS AND OUTPUT TERMINALS	NO. OF POSSIBLE STATES	COUNTING CAPACITY *
2	4	3
3	8	7
4	16	15
5	32	31
6	64	63
7	128	127
8	256	255

*ASSUMES COUNT BEGINS FROM ALL 0's STATE

Fig. 5-3. Counter capacity depends on the number of flip-flops and output terminals.

count reaches 10 or 1010 in binary. The way we get it to do this is to put inverters in the second and fourth lines from the counter. When the two 0s that appear in the number at the count of 10 are inverted, bringing the input of the NAND gate to 1111, which will drive the output low and will reset the counter. We can also invert the output of the counter and send it as a carry pulse to the next counter to the left, so that it will display 1.

This system of decoding with gates and inverters can be used to get a pulse on any count that we wish. Usually, this sort of thing is all built into a counter.

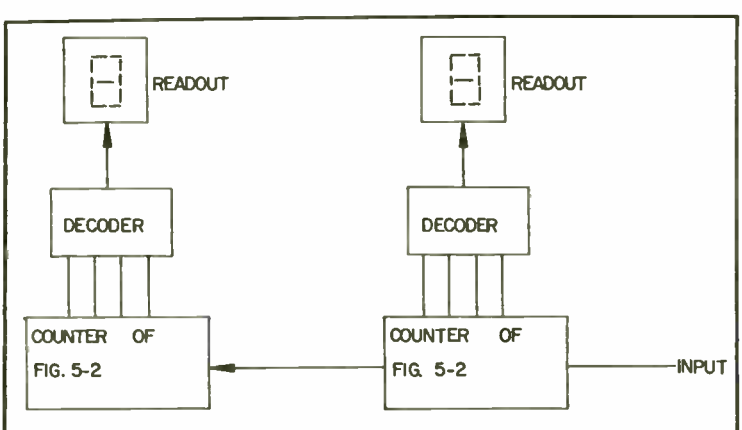


Fig. 5-4. Block diagram of a two-decade counter.

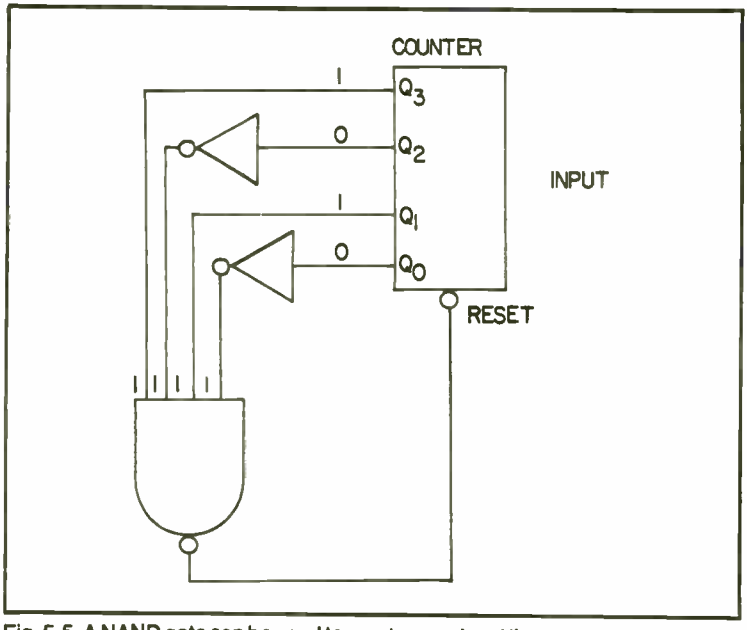


Fig. 5-5. A NAND gate can be used to reset a counter at the count of 10.

THE BCD NUMBERING SYSTEM

The decoding arrangement of Fig. 5-5 lets us build counters where each stage will count up to ten in the binary number system. This arrangement is called *Binary Coded Decimal* or simply BCD. The signals are carried on groups of four wires. Each group carries a binary signal to represent signals from one to nine.

The advantage of the BCD system is that it can easily be decoded into decimal form to operate decimal readouts. This system is frequently used in instruments such as digital voltmeters, frequency counters and calculators. The system is very simple and easy to use. The general idea is shown in Fig. 5-6.

A counter is a comparatively simple circuit. Many different counters are available in integrated circuits. In spite of its simplicity, the counter can perform an amazing number of different functions in digital circuits.

One example of the use of a counter is in frequency division. In Fig. 5-6, we show the same basic counter arrangement that we have been talking about. It counts the pulses that have been applied to the input and displays the count in binary form on its output lines.

Suppose we have an input signal with a frequency of 60 Hz and we want an output frequency of 1 Hz. That is, we want an output

signal consisting of one pulse per second, such as we might use in a clock. To get this 1 Hz signal, we must divide the input signal by 60. This isn't as difficult as it might seem.

The easy way to look at the problem is to say that we want a pulse or a change of state after our counter has reached a count of 60 (111100 in binary). Thus, we want an output pulse when lines Q_5, Q_4, Q_3 and Q_2 are high and lines Q_1 and Q_0 are low. When this is true, the count will be:

$$32 + 16 + 8 + 4 = 60$$

Again, we can accomplish this with a gating arrangement. A commercially available NAND gate has eight inputs. We need only

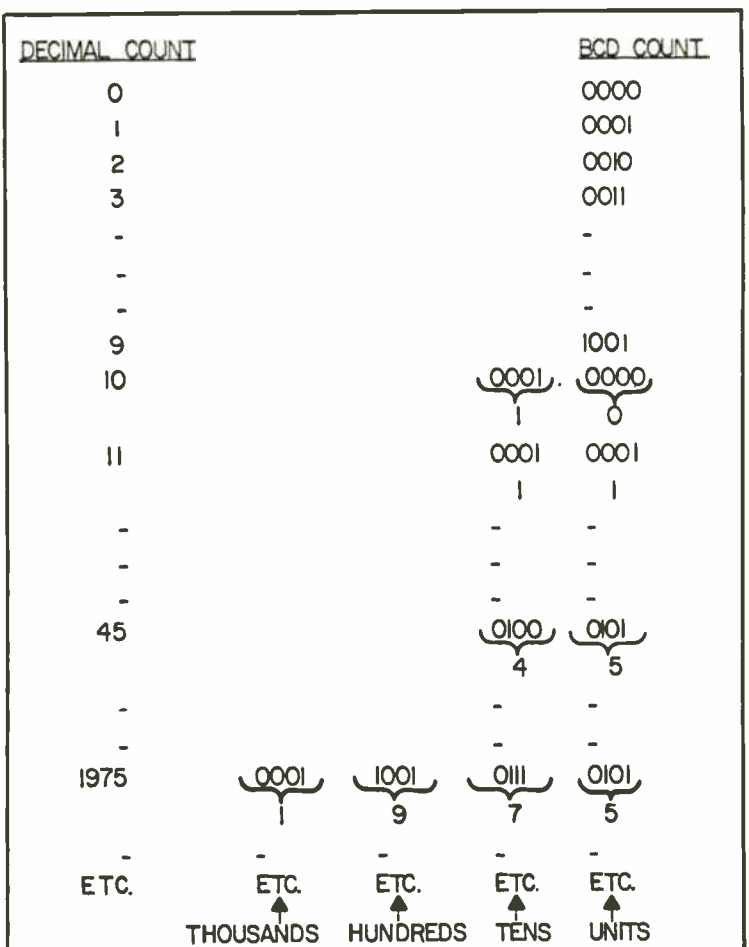


Fig. 5-6. Table illustrating the BCD counting system.

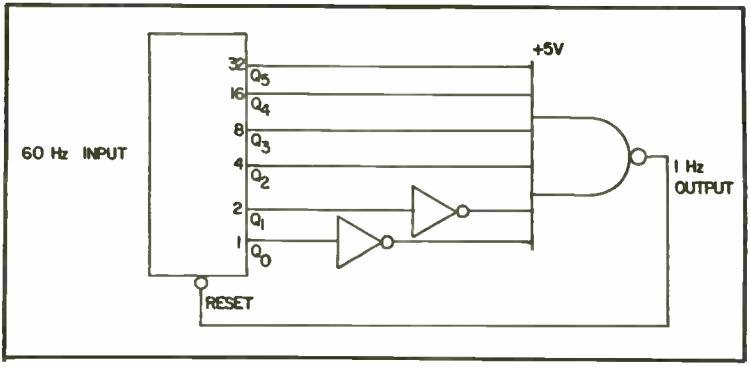


Fig. 5-7. Block diagram of a system using a 5-stage counter to divide by 60.

six inputs for our circuits, so we will tie two of the inputs to a logic high. The others are connected as shown in Fig. 5-7. Inspection of this circuit will show that the output of the gate will change state everytime that the input is 60 (111100 in binary). We can use the output of this gate not only to derive our output pulse, but also to reset our counter to zero.

It is easy to see that our arrangement can be used to divide any frequency within the response range of our circuit by any integer. For example, counters can be used to derive any of the pulse ratio we might need in a TV system. We could, for example, derive the color subcarrier, the horizontal sync pulses, and the vertical sync pulses, all from the same stable oscillator.

REGISTER

A register is a storage device that is much more sophisticated than a simple latch. It usually uses edge-triggered flip-flops as storage devices. Thus, a register is operatied by a clock pulse. This means that the output of a register can be connected back to the input without problems such as oscillation or racing.

Figure 5-8 shows a general block diagram of a register. Most registers don't have all of the functions of our sort of generalized model, but it will show us what we might expect to find in a register.

Note in Fig. 5-8 that we can put bits into a register and take them out in many different ways. These include:

- ☐ Serial-in, serial-out
- ☐ Serial-in, parallel-out
- ☐ Parallel-in, serial-out
- ☐ Parallel-in, serial-out

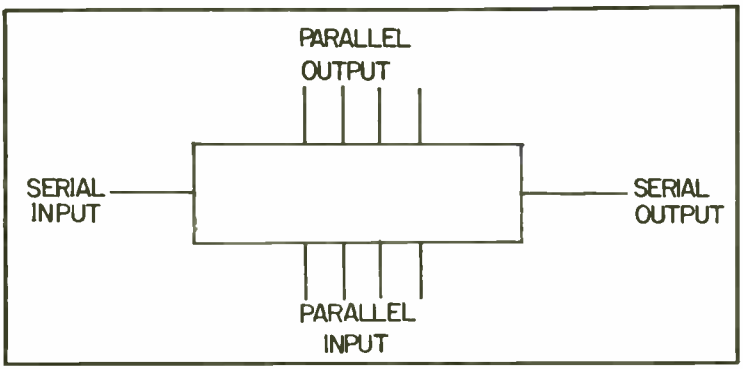


Fig. 5-8. Block diagram of a generalized register.

We can use registers to perform many useful functions in a digital system. For example, we could use a register to connect data from serial to parallel form, and vice versa. We can also use registers to temporarily store digital signals.

In Fig. 5-9, we show a parallel-out, serial-in shift register. The input data, consisting of a series of high and low signals, arrives in serial form at the input at the left of the figure. The input high and low signals are synchronized to a clock pulse train that is also applied to the register.

Each time that a new data pulse is applied to the input terminal, a clock pulse is applied. The first stage of the register then takes on the state of the input. At the same time, whatever state the first stage formerly had is transferred to the next stage to the right.

This process is continued, with the data being shifted to the right, until four bits of data have been received. Note that when this occurs, the data is stored in parallel form in the register.

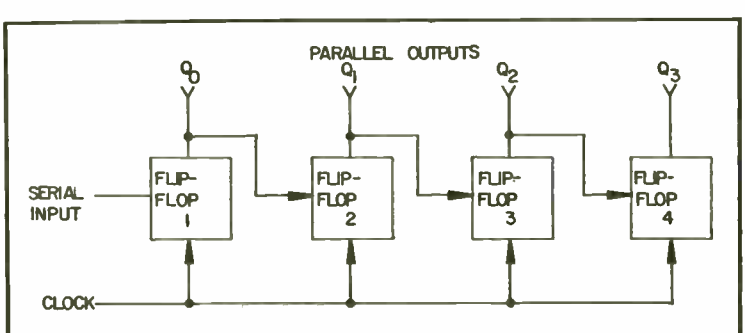


Fig. 5-9. Block diagram of a 4-bit serial in, parallel out, shift register.

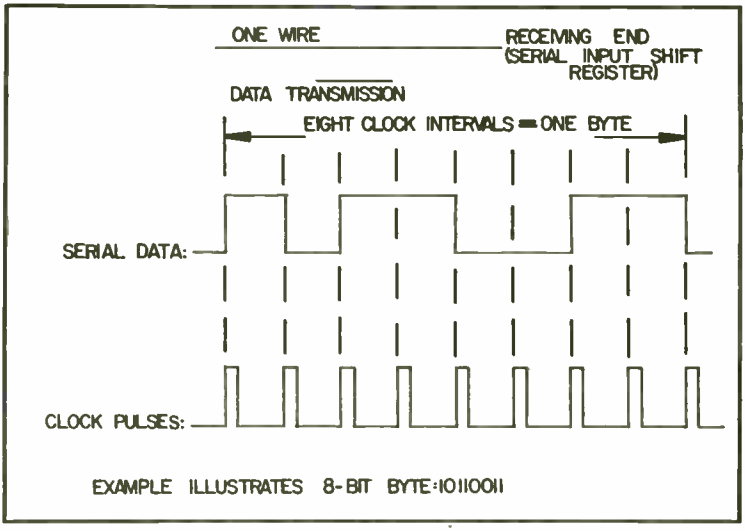


Fig. 5-10. Pulse waveforms of the type applied during the operation of the serial-input register in Fig. 5-9.

Figure 5-10 shows the pulses in an 8-bit register of the type shown in Fig. 5-9. Note that the clock pulses are absolutely essential to the operation of the register.

Another variation of a shift register is shown in Fig. 5-11. The operation here is just the opposite of the register in Fig. 5-9. The data is applied in parallel to the inputs labeled D_0 through D_3. The data is then stored in the flip-flops of the register.

Inasmuch as the input signal is now stored, it can be removed. When clock pulses are applied, the contents of the register are clocked out in serial form.

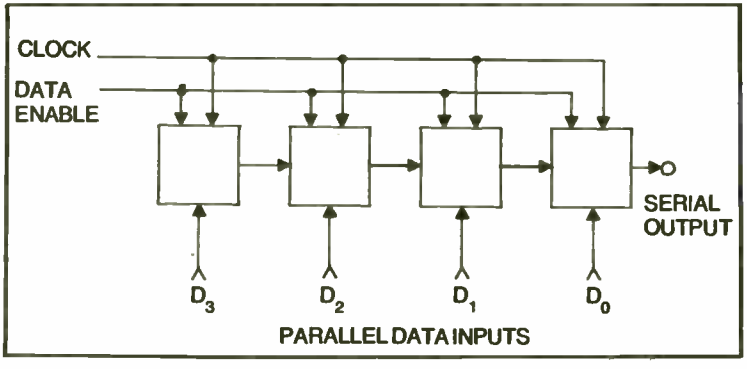


Fig. 5-11. Block diagram of a 4-bit parallel in, serial out, shift register.

One obvious use of shift registers is to convert data between serial and parallel form. It can also be used for short-time storage of data. For example, the parallel inputs of a register can be used to accept data from a keyboard. While the data is being entered, it is stored in the register. Once it has been entered, it can be clocked out serially into a digital system.

Shift registers can also be used to perform counting functions. For example, a "1" can be entered into the serial input, then the number of clock pulses that have occurred since the "1" was entered can be determined by seeing how many places in the register the "1" has advanced.

Shift registers are available in integrated circuits, and large-scale devices may contain many of them. They may be of any length. Some calculators have registers of more than 70 stages.

Chapter 6
Some Digital Integrated Circuits

So far we have discussed all of the basic digital functions that are performed in a digital system. We have mentioned that these functional units are available in integrated circuits but we haven't spent any time discussing the integrated circuits themselves. Inasmuch as there are thousands of different digital integrated circuits available commercially, we can't possibly cover them all in one chapter. In fact several books would be required. What we will do here is to give some idea of the types of integrated circuits that are commonly used. We will also describe a few functional units that are not mentioned elsewhere in the book.

Integrated circuits are usually grouped into three categories on the basis of their internal complexity. The simplest category is called *Small Scale Integration,* or simply SSI. Devices in this category have the equivalent of up to 10 logic gates in a single package. They may be packaged in a round transistor type can with the leads protruding from the bottom, or more commonly they are in a *dual-in-line package,* or DIP. A package of this type is shown in Fig. 6-1. SSI packages usually have either 14 or 16 pins.

At the other extreme we have what is called *Large Scale Integration* or LSI. These devices have the equivalent of over 100 logic gates in a single package. The packages often have either 28 or 40 pins, although other arrangements are sometimes used. Microprocessors and semiconductor memories fall into this category. Between these two extremes we have *Medium Scale Integration* or MSI. Counters and registers are in this category.

One of the most useful "tools" available to the engineer or technician working in the digital field is a familiarity with the

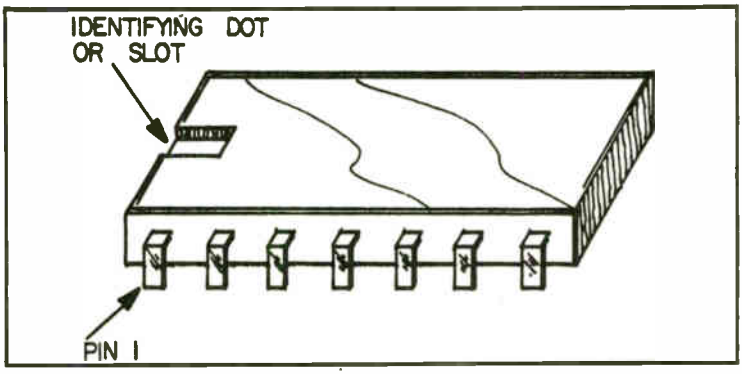


Fig. 6-1. Drawing of a typical dual-in-line package, or DIP, IC.

different types of digital integrated circuits that are commonly used. Of course, as we pointed out earlier, you don't have to know what is inside, a functional knowledge of what happens at the pins is usually adequate.

New digital integrated circuits are still being introduced at a rapid rate. The only way to keep up with them is to read the technical journals regularly. Many fine data books are available from manufacturers that will serve as a source of information on what is available at any given time.

What is actually made available in digital integrated circuits depends more on demand that on almost anything else. If there is a great demand for a rather complicated function in many systems, manufacturers will usually develop an LSI circuit that will perform

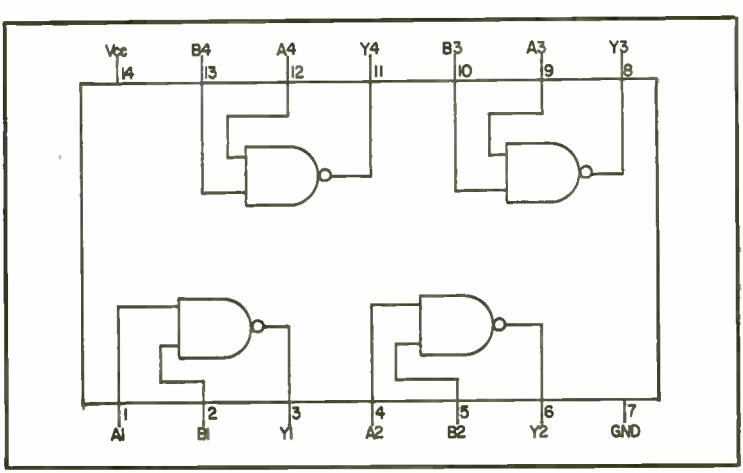


Fig. 6-2. Block diagram of a type 7400 quad 2-input NAND gate.

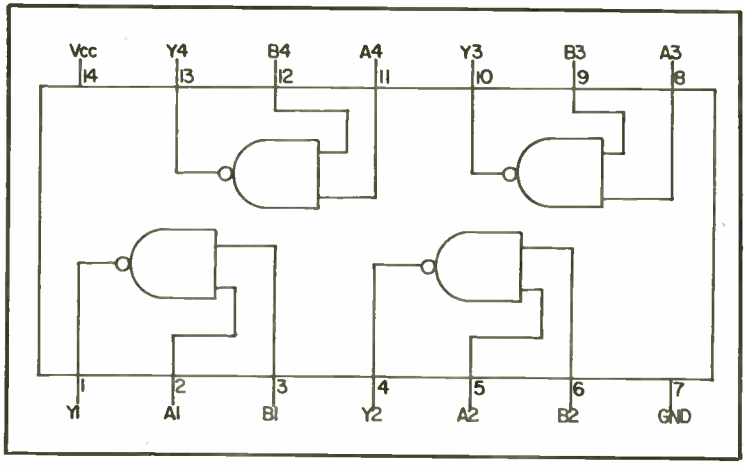


Fig. 6-3. Block diagram of the type 7401 quad 2-input NAND gate with open collectors.

many of the system functions in a single integrated circuit. If, on the other hand, there isn't much demand for a particular combination of functions, they must be realized by using many SSI circuits. The reason for this is that it is feasible to put almost any number of functions into a single integrated circuit. Much of the cost is in the original development. Another problem is that the yield is often low in a new product. If the demand is great enough the manufacturing process can be highly refined, giving a good yield, and the development cost may be amortized over a large number of devices.

LOGIC GATES

One of the simplest digital integrated circuits is the logic gate. Many different gating arrangements are available. Usually the number of gates in a single package is limited by the number of pins available on the package. The number of pins on SSI packages is usually limited to 14 or 16 in the interest of standardization.

Figure 6-2 shows the Type 7400 quad 2-input NAND gate which is typical of a digital integrated circuit. The word, quad, means that there are four separate gates in the package. Note that the positive supply voltage is applied to pin 14 and ground is connected to pin 7. This arrangement is rather common, but it is far from universal. Don't depend on it unless you have checked the diagram.

Figure 6-3 shows what appears to be an identical gate, but in this arrangement, the gates have open collectors as described in

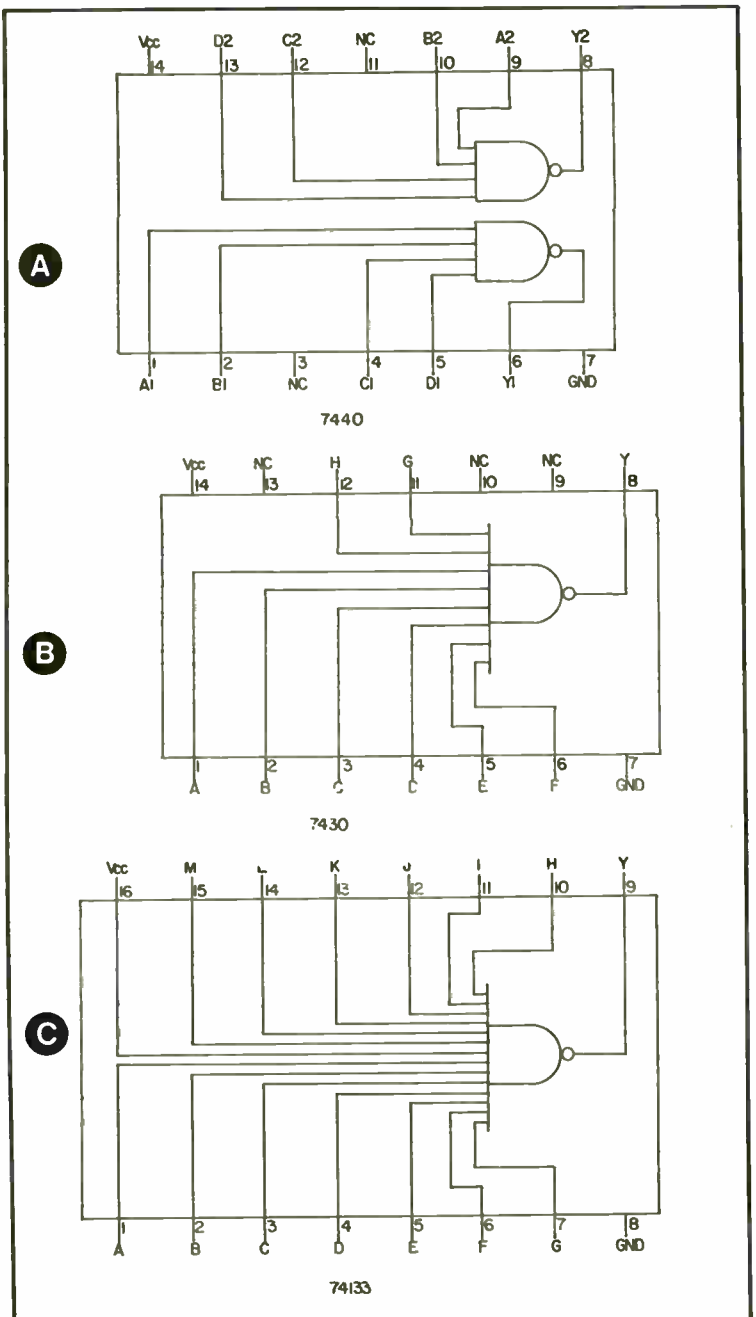


Fig. 6-4. Three logic gates with a variety of input configurations.

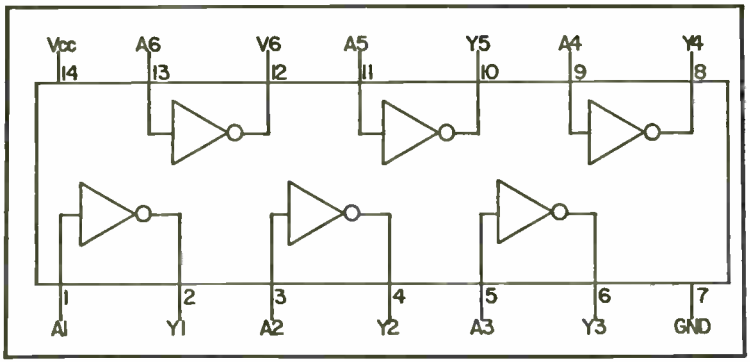


Fig. 6-5. Block diagram of the type 7404 hex inverter.

Chapter 2. This figure illustrates that just having the basing diagram doesn't necessarily tell you all you need to know about an IC. Just looking at Fig. 6-3 might lead to the misconception that the package contained four regular 2-input NAND gates. If an open collector gate were connected in a regular circuit without a pull-up resistor, the output would never go to a high level.

We have spent a lot of time talking about gates with two inputs. This has been primarily in the interest of keeping things simple. Gates are available with many inputs as shown in Fig. 6-4.

INVERTERS

The inverter is probably the most simple digital circuit that one can imagine. However, this doesn't mean that it isn't very useful. In designing a digital system there are often cases where a signal has the wrong polarity to do what we want done. In such a case, it is handy to be able to insert an inverter into the system. Many different inverter packages are available. The Type 7404 hex inverter shown in Fig. 6-5 is typical. The word, hex, means simply that there are six circuits in the package.

BUFFERS

Another name that you will find in the catalog of digital circuits is buffer. The name is applied to both inverters and gates. It merely means that the output stage has a greater capability to source and sink current than a regular gate or inverter. Buffers are used when it is necessary to drive something that consumes more current than a regular logic gate. For example, a buffer would be used to drive a line connecting two parts of a digital system.

THE SCHMIDT TRIGGER

The inverters and gates shown in Fig. 6-6 look just like regular gates and inverters except for the little symbol inside that looks like the hysteresis loop of a magnet. This symbol means that the gate exhibits a property known as hysteresis which we will discuss in a minute. The circuit that accomplishes this is called a Schmidt trigger circuit after a similar circuit made with discrete components.

The term hysteresis used in connection with a logic gate means that the input circuit has two, rather than one, trigger levels. If the input voltage slowly rises from zero, when it reaches a level of about +1.6V, the circuit will change state, that is, the output of the NAND gate will go low. O. K. so far. Now suppose the voltage starts to drop back toward zero. The circuit will not change

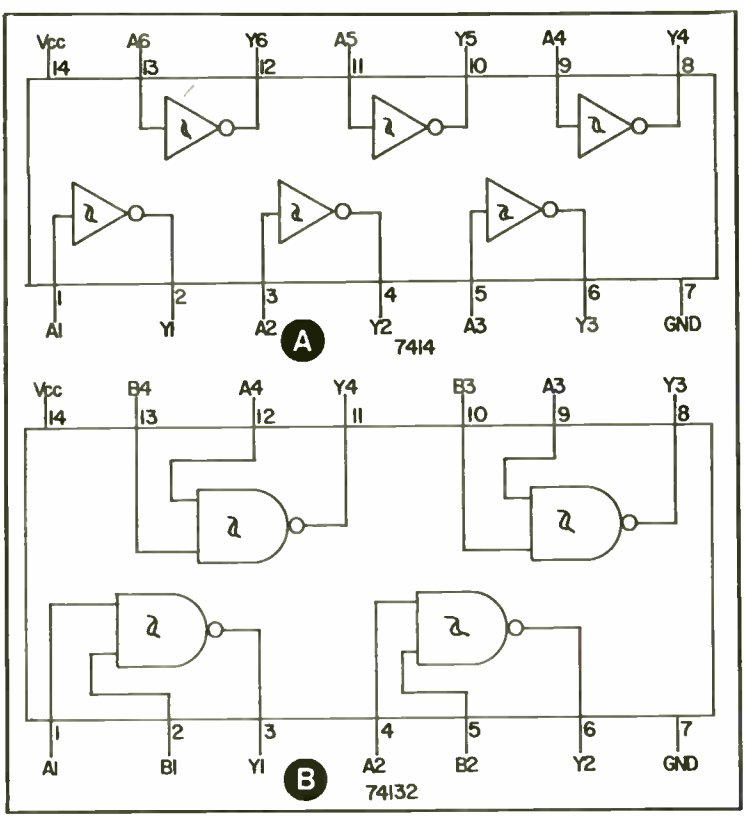


Fig. 6-6. The marking in the inverter and gate symbols indicate Schmidt trigger inputs.

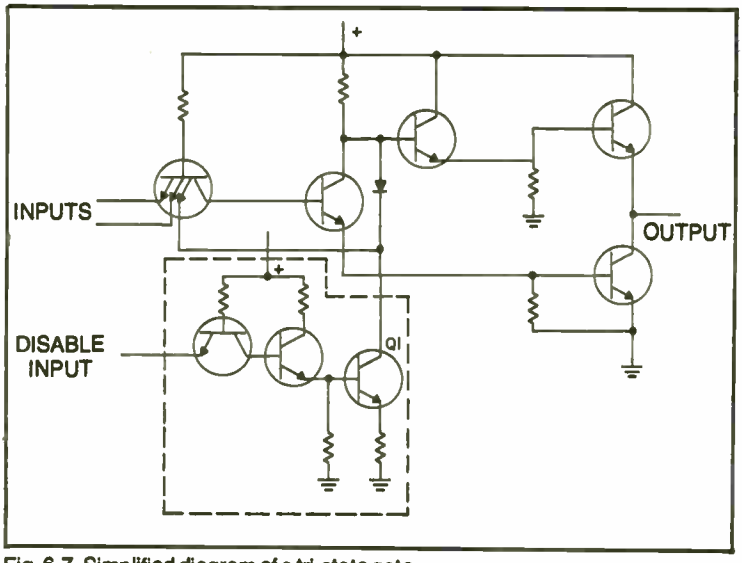


Fig. 6-7. Simplified diagram of a tri-state gate.

state when the voltage reaches +1.6V, but will remain in the low state until the voltage drops to about +0.8V, then it will change state. Thus the trigger voltage is higher when the input voltage is rising than when it is falling. This effect, which is called hysteresis, is useful when the input comes from the outside world, rather than from another digital IC. We will discuss the operation of the Schmidt trigger in instances when the input voltage has a rather long rise or fall time later in the book.

TRI-STATE LOGIC

The open collector arrangement described in a preceding chapter for connecting the outputs of logic gates in parallel has many limitations. To improve this situation a new type of TTL was developed. This is called *tri-state* logic. The name is apt to be deceiving because the logic signals still have only two significant states—high and low. The name tri-state means that the output of a gate has three possible states—high, low, and a third state where there is no output, just a high impedance.

Figure 6-7 shows a simplified diagram of a tri-sate gate. In many ways it is similar to a regular TTL gate. The interesting part of the circuit is shown within the dashed lines. When the disable input is high, transistor Q1 removes the drive current from the output transistors. Thus both of the output transistors are turned off, and the output pin is floating.

The circuit of Fig. 6-7 varies from a standard TTL gate in another respect. The top transistor in the totem-pole output state is actually a part of a Darlington circuit. The purpose for this is so that the output stage can supply leakage currents for the many other output stages that might be connected in parallel.

When the disable input of Fig. 6-7 is brought to a low logic level, the gate will act just like any other TTL gate. The state of the output will depend on what is connected to the inputs. Several of these gates can be connected with their outputs in parallel without

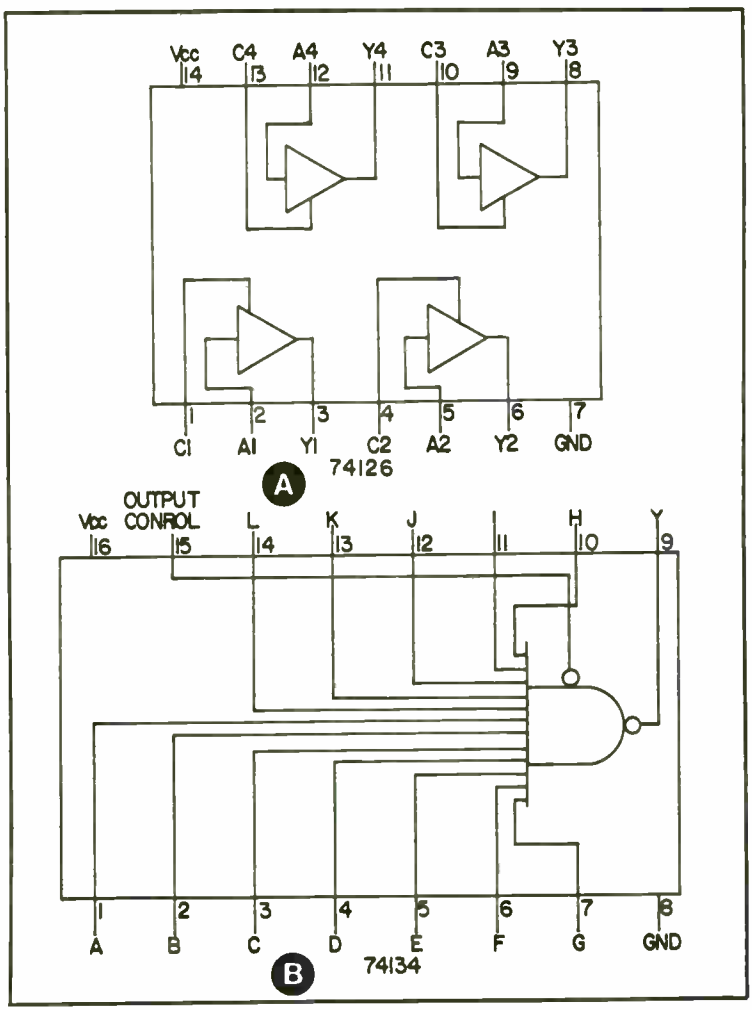


Fig. 6-8. Block diagrams of tri-state inverters and gates.

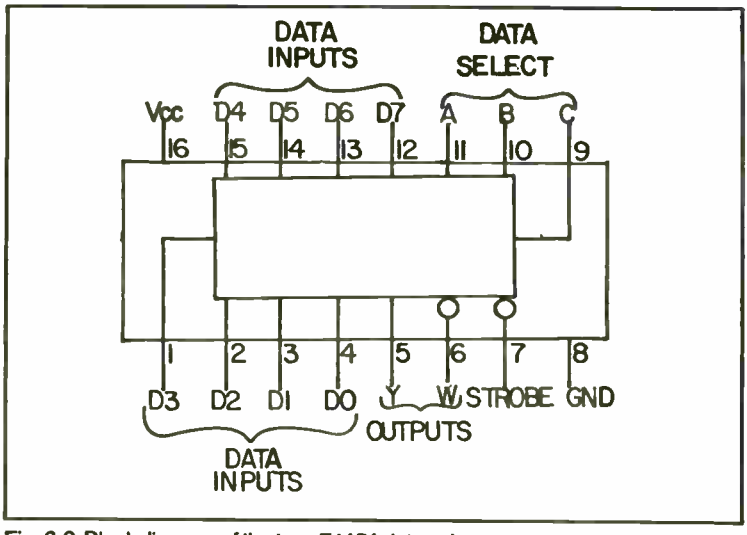


Fig. 6-9. Block diagram of the type 74151 data selector.

any ill effects. Normally all of the disable inputs are held high so that there will be no output from the gate. The normal output of any particular gate can be applied to the common connection by merely bringing its disable input to a low logic level.

INPUTS				OUTPUTS	
SELECT			STROBE		
C	B	A	S	Y	W
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

H = High Level, L = Low Level, X = Don't Care
E0, E1 ... E15 = the complement of the level of the respective E input
D0, D1 ... D7 = the level of the respective D input

Fig. 6-10. Truth table for the data selector in Fig. 6-9.

Figure 6-8 shows a typical tri-state inverter and gate package. All sorts of gates are available using tri-state technology.

THE DATA SELECTOR

Figure 6-9 shows an integrated circuit which is called a *data selector*, or decoder. It has two output lines which always have complimentary states, much like the Q and Q outputs of a flip-flop. When pin Y is high, W will be low and vice versa. There are eight input pins. In effect only one of these input pins is connected to the output at any one time. The particular input pin that is connected to the output depends on the signals applied to the three data select pins. There is one more pin, called the strobe, which must be brought to a low logic level for the circuit to operate.

The operation of the data selector is shown in the truth table of Fig. 6-10. On the first line of the table we see that if the strobe pin is at a high level, pin Y will be low and pin will be high. The X's in the columns pertaining to the C, B, and A inputs show that the circuit doesn't care what their states might be. Thus when the strobe pin is high, the output is fixed regardless of what is connected to the other pins.

The circuit gets more interesting when the strobe is brought to a low level. In the second line of the truth table, we see that inputs C, B, and A are all low, or 0's the output at pin Y will take on whatever state is applied to pin D0. On the next line we see that if the inputs, starting with pin C are 001, the output at pin Y will take on the state of pin D1.

As we look down the table, we see that the states of pins C, B, and A, in that order, represent the binary number corresponding to the input pin that is connected to the output, pin Y. As we noted earlier, pin W will always be complimentary to pin Y.

It is rather obvious that the data selector can be used to take the data that happens to be on any of the input pins and connect it to

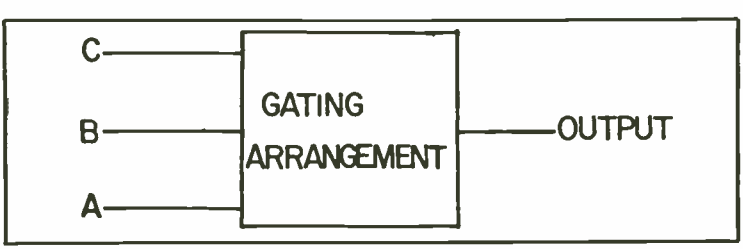


Fig. 6-11. Block diagram illustrating how a data selector can simplify a complex gating arrangement.

LINES A	B	C	DESIRED OUTPUT	CONNECT PIN	TO
0	0	0	0	D0	GND
0	0	1	1	D1	+
0	1	0	1	D2	+
0	1	1	1	D3	+
1	0	0	0	D4	GND
1	0	1	0	D5	GND
1	1	0	1	D6	+
1	1	1	1	D7	+

Fig. 6-12. Table of data selector terminal connections for the output shown.

the output by merely applying the address of the pin to the in data select pins C, B, and A. Thus if we wanted the output to have the same state as pin 5, we would apply the signal 101 to pins C, B, and A. 101 is the binary number corresponding to 5.

An application of the data selector that isn't immediately obvious is that it can be made to replace many gates to perform very complex gating functions. Suppose that we have three lines, C, B, and A as shown in Fig. 6-11. We can implement a gating arrangement that will perform any desired operation on the signals on these lines with a data selector. All we have to do is to decide what we want the output of our circuit to be for various combinations of 0's and 1's on lines C, B, and A. Then we connect the corresponding data pin to this level and take our desired output from pin Y.

Suppose, for example that we wanted the desired outputs shown in Fig. 6-12 for various combinations of signals on lines C, B, and A. The table in the figure shows which pin we have to connect to a high level to get this condition. By refering back to the truth table in Fig. 6-10 you can see how this works. The connections are shown in Fig. 6-13. Thus a data selector can be used in a new simple circuit to accomplish very complex gating function.

THE MONOSTABLE MULTIVIBRATOR

The monostable multivibrator, or one-shot as it is often called, is sort of a combination of analog and digital circuitry. It is a circuit that will develop an output pulse when it is triggered by an input pulse. The duration of the output pulse is determined by a

resistor and capacitor connected externally. The one-shot is used in digital system to introduce a time delay, or to generate a pulse having a definite duration.

Figure 6-14 shows the Type 74121 one-shot together with its truth table. An external resistor is connected between pins 9 and 11 and a capacitor between pins 10 and 11 to set the time period.

Although the one-shot looks like a very convenient way to adjust timing in a digital system, it has the disadvantatge that its timing depends on the value of the external resistor and capacitor. In many systems slight changes in timing have little or no effect. In other systems timing is very critical. In critical applications, it is better to generate timing periods, delays etc., by counting down from a stable source than to depend on the time constant of an external circuit.

BECOMING FAMILIAR WITH DIGITAL CIRCUITS

Only a very few digital IC's have been mentioned in this chapter. However, we have tried to take some of the mystery out of the subject. One of the best ways of increasing one's knowledge and familiarity with digital circuits is to leisurely scan through the manufacturers data books. Such books are available from most IC manufacturers either free or at a very nominal cost.

By merely looking over the various IC's that are available, one will gradually obtain a familiarity that will simplify both design and

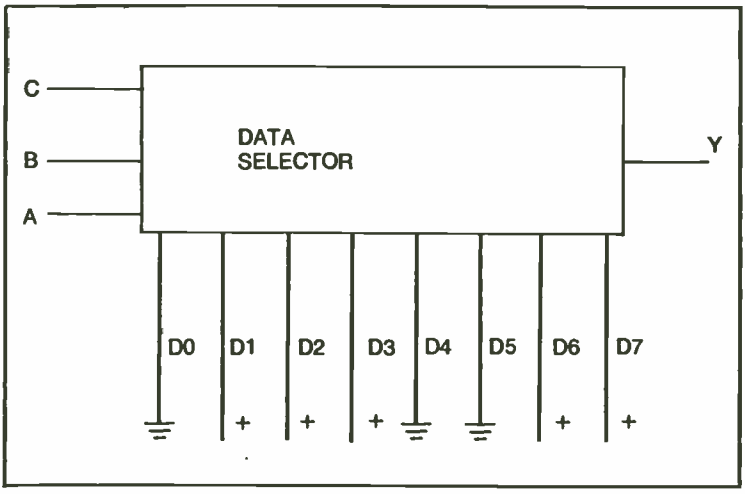
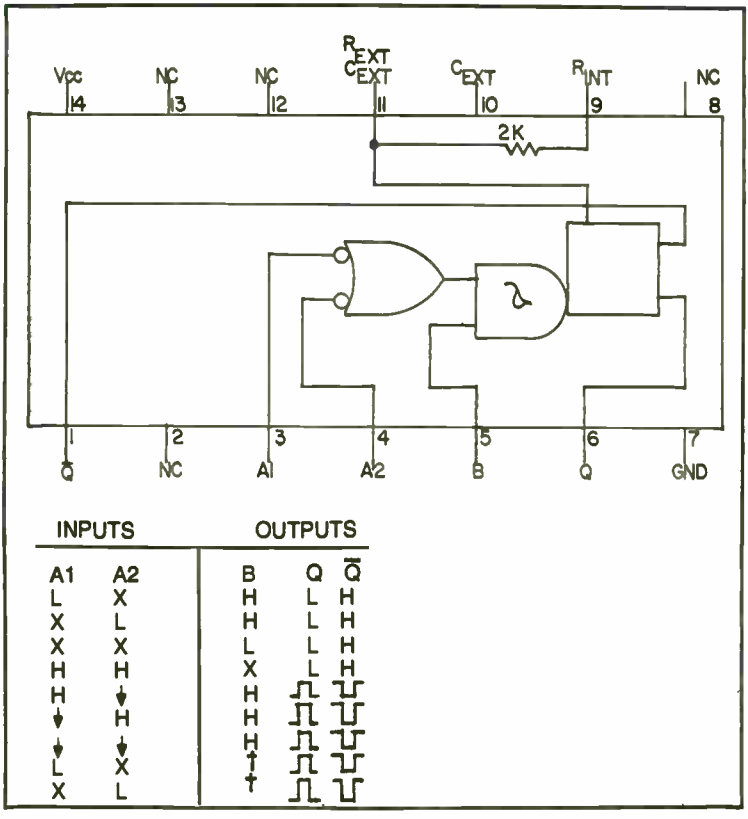


Fig. 6-13. Circuit connections needed for realizing logic states in the truth table of Fig. 6-12.



INPUTS		OUTPUTS		
A1	A2	B	Q	$\overline{Q}$
L	X	H	L	H
X	L	H	L	H
X	X	L	L	H
H	H	X	L	H
H	↓	H	⎍	⎍
↓	H	H	⎍	⎍
↓	↓	H	⎍	⎍
L	X	↑	⎍	⎍
X	L	↑	⎍	⎍

Fig. 6-14. Block diagram and truth table for the type 74121 monostable multivibrator.

troubleshooting. Some of the functions that are available in a single package are rather complex and may appear rather confusing at first. Fortunately, the manufacturer also gives the truth table for each circuit, and this makes it possible to understand even the most complex circuit.

Chapter 7
Power Supplies And Noise

Although power supplies and noise might seem to be unrelated subjects, we have a reason for covering them in the same chapter. The reason is simply that power supplies and their associated leads are the most common means for propagating objectionable noise through digital systems.

The power supplies used with digital systems are somewhat unique. The voltages are low, often only +5V. Currents are high. It is not uncommon to find power supplies capable of delivering many amperes. Voltage regulation is universally used because digital ICs are easily damaged by over-voltage.

Another unique aspect of the power supplies used in digital systems is that they must be capable of delivering current in short spikes. Digital systems are, by their nature, switching systems; as a result, power supply current is furnished in pulses. Because of these factors, any consideration of the power supply must include the entire power distribution system.

Figure 7-1 shows a simplified schematic diagram of the power supply of a digital system. It consists of a power transformer, a rectifier, a couple of capacitors, and a regulator. Inasmuch as digital circuit elements operate at comparatively low voltages—between 5 and 15 volts—the power supply produces a low voltage, and if the system has any size at all, a large current. Currents of several amperes are common in larger systems.

Because of the high currents involved, filter chokes are usually not considered practical. This increases the problem of regulating the output voltage and removing AC ripple from the output. Fortunately, the regulator itself is a pretty good filter.

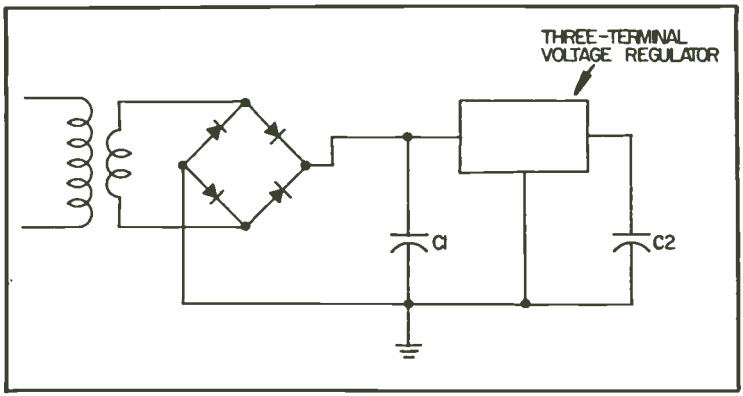


Fig. 7-1. Schematic of a typical power supply for a digital system.

Voltage regulation is necessary because digital ICs are usually not very tolerant of overvoltages. Whereas a vacuum tube might overheat when the supply voltage exceeded the specified limits, it will usually forgive the offense if the over-voltage doesn't last too long. A solid-state device, on the other hand, has no forgiving spirit at all. This caused one engineer to remark that, "Vacuum tubes are kind and forgiving, whereas solid-state devices seem to have been designed to protect fuses."

THE REGULATOR

In smaller power supplies, the regulator is a solid-state device with three terminals—the input, the output, and ground. In larger power supplies, there are often a few external components, such as a transistor to carry currents that are too large for an IC regulator. In all cases, the regulator is a feedback circuit that compares the actual output voltage with the voltage from some standard, such as a Zener diode. The error voltage is amplified and is applied to some device, such as a pass transistor that will, in turn, limit the output voltage.

Any closed-loop feedback system can oscillate under the right conditions. Unfortunately, we can't always predict what these right conditions might happen to be, so precautions are always taken to keep voltage regulators from oscillating. This is usually accomplished by the input and output filter capacitors shown in Fig. 7-1. Thus, in addition to the regular filtering function, these capacitors also help to make sure that the regulator will not oscillate.

The input filter capacitor, C1 in Fig. 7-1, must be large enough so that the input voltage to the regulator will never fall below the

specified value of the output voltage. Thus, its value will depend on how much current the power supply must deliver. In a high-current supply, the voltage across C1 will drop more between cycles of the voltage from the transformer and the rectifier. In a one-ampere supply, the value of C1 will be about 4000 μF.

The output capacitor, C2, is much smaller. Its main function is to remove any high-frequency noise and to smooth out transients that may be generated in the digital system itself.

Most IC regulators have built-in protective features. They will shut down if the load current becomes excessive or if they overheat. This should be remembered when troubleshooting a power supply.

POWER DISTRIBUTION SYSTEM

The best way to look at the power supply of a digital system is not as a single stage called the power supply, but as a power distribution system that not only furnishes the proper voltages and currents, but distributes them to the various parts of the system. Thus, the power distribution system consists not only of what we normally call the power supply, but also the wiring, connectors, shielding, and miscellaneous filtering capacitors that might be in the system.

We have noted earlier that in a digital system, things happen very quickly. This results in transients that can travel all through the system. To get a better idea of the result of rapid switching, consider the voltage that can be induced in the inductance of a lead by a switching transient. The induced voltage is given by:

$$E = L \frac{di}{dt}$$

where E is the induced voltage, and di/dt is the rate of change of current.

In most digital systems, the inductance of leads, the values of the currents and the switching time are all small, but the rate of change of current can be very large indeed. Suppose, for example, that a current switched from zero to 10 mA in 10 nS, a reasonable thing to expect in a digital system. The rate of change of current would be ten million amperes per second. Of course, the current will never get very high, but this extremely high rate of change will cause induced voltages to be much higher than one might expect. A lead having an inductance of one microhenry would have an induced

voltage of 10 volts from such a transient. In a much shorter lead, having an inductance of only one tenth of a microhenry, the induced voltage could be one volt.

DESPIKING CAPACITORS

The discussion in the preceding paragraphs shows how serious the induced voltages can be in digital power distribution systems. The best approach is to keep these rapid changes in current off the distribution lines by the judicious use of capacitors. Figure 7-2 shows in an elementary way what happens when there is a switching surge in a system. In Fig. 7-2A, we have a rather rough equivalent circuit of a power distribution line with a switching device connected to it. When the switch is closed, note that voltage E on the supply line will suddenly drop. This is shown in Fig. 7-2B.

We can avoid the voltage drop on the supply line by installing a capacitor close to the switching device as shown in Fig. 7-2C. Now, when the switch is closed, the voltage will not drop instantly. The capacitor will hold the voltage up, while it is discharging through the load. If the switch is only closed for a very short time, the voltage will not have a chance to drop very much before the switch is opened again.

The situation shown in Fig. 7-2 is analogous to what happens when a TTL IC switches from one state to another. We noted earlier that during this very brief switching period, there is a low-resistance path between the supply voltage and ground. By the judicious use of what are often called "despiking capacitors" the transients can be kept off the supply lines. It is usual practice to install a 0.01 to 0.1 ceramic capacitor on each PC board for every three or four TTL packages. Also on each board is at least one larger capacitor in the order of 10 μF, preferably a tantalum type.

Although use of despiking capacitors is properly a function of system design, sometimes failure of a capacitor can cause very elusive problems in a system.

LEADS SHIELDING AND GROUNDING

The leads that carry the operating voltages throughout the system are an important part of the power distribution system. Even though capacitors are used throughout the system, the leads should be heavy enough to carry the required current. The hot line and ground line should be run close together, and, in many systems, shielding of the power lines is advisable.

Ground loops should be carefully avoided, the shielding and cabinet should be connected to the power supply ground at only one point. Figure 7-3, shows a sketch of a sandwich type of construction that is sometimes used for power lines. The hot line is sandwiched between two conductors that are grounded. This arrangement not only provides shielding for the lines, but the capacitance of the structure helps to suppress surges.

Most manufactured equipment has good wiring and shielding, but many times a piece of manufactured equipment cannot be installed in a broadcast station without modification or the addition of other components. This is particularly true when computers are used to control various functions. The addition or modification often violates the wiring and shielding rules, with the result that the system will not be reliable.

NOISE AND INTERFERENCE

We have already discussed the way that impulses that are generated inside a digital system may interfere with its operation.

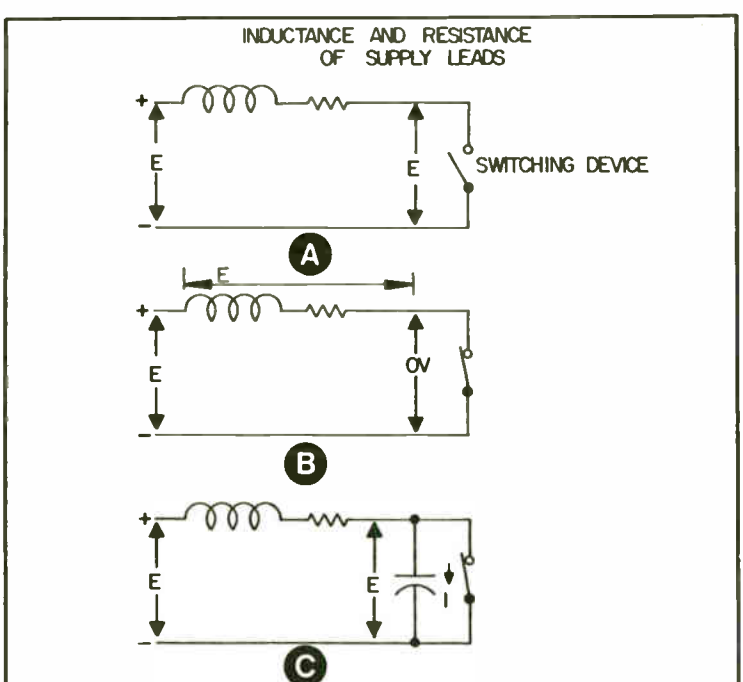


Fig. 7-2. Schematics showing the effect of a switching device on supply voltages and the purpose of a despiking capacitor.

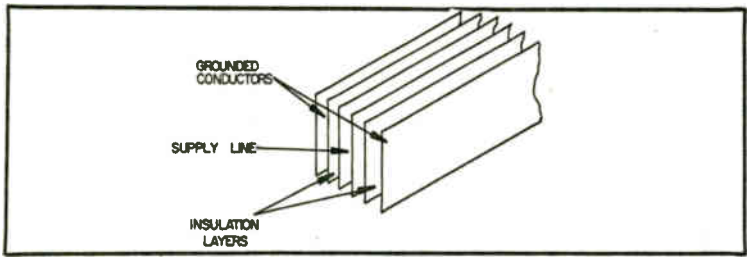


Fig. 7-3. Drawing of a sandwich conductor used to provide good shielding of power supply lines.

There are two other considerations regarding noise and interference from outside sources. One is that external things, such as a transmitter, may interfere with the system. The other is that the digital system may generate noise that will interfere with the operation of other equipment, which may or may not be digital. If you have any doubts about the ability of digital devices to cause interference, just hold your pocket calculator close to a TV set and watch the interference pattern on the screen.

Noise in analog systems is pretty easy to understand and to identify. We are all familiar with this hiss, roar, clicking and popping that we call noise. We can't always minimize the noise as much as we would like, but usually we have no trouble identifying it. In a digital system, noise isn't as easy to identify.

In an analog system, if noise gets into one of the stages it will travel along with the signal through all the other stages and will show up, usually amplified considerably, in the output. In a digital system, however, noise in one stage will have no effect on any of the stages until it is strong enough to overcome the noise margin of the stage. Then it will introduce a false high or low into the system.

Figure 7-4, shows the effect of noise graphically. The maximum level that the system will see as a low signal and the

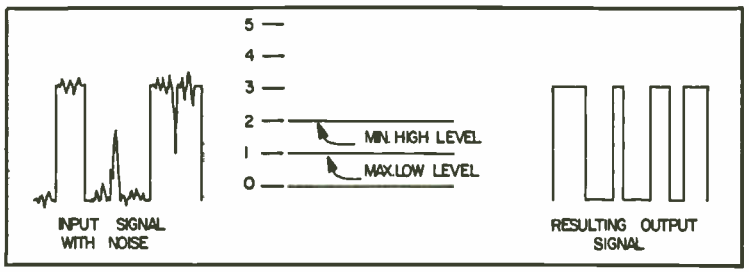


Fig. 7-4. These drawings show how noise on an input signal can cause spurious pulses an output signal.

minimum level that the system will think is a high are marked on the graph. We have also shown a digital signal, which is corrupted by noise. Looking at the plot of the output, we see that most of the time the noise isn't strong enough to cause false triggering. However, at two points in the figure the noise does cause the output signal to be incorrect.

There are two points about Fig. 7-4 that are important. The first is that even though the noise at the input has an effect on the output at two points, the output doesn't look noisy. If we were to look at the output on an oscilloscope, we wouldn't necessarily know that the errors were due to noise. The other important point is that, often, there is enough noise on a digital signal so that there is practically no noise margin left and we have no way of knowing this. The system will work fine, but the noise is so high that even a very slight additional amount of noise getting into the system will cause errors. This sometimes makes troubleshooting very difficult. A system with some noise, but not enough to cause trouble, will often be erratic.

An old broadcast engineer once remarked that the vicinity of a radio transmitter was the worst possible place to try to operate electronic equipment, including a radio station. Of course, the remark refers to the fact that there is often a very high RF field strength in the vicinity of a radio or TV station. Everything that is located in this region of high field strength will try to act as an antenna and pick up some of the energy. Digital signals are quite low in voltage, and although they tend to discriminate against noise, it doesn't take a very great shielding or grounding fault to pick up interfering signals that are much stronger than the regular digital signals.

Some items of digital equipment were apparently not designed to operate in a region of high RF field strength. Items of this type need work on the shielding and grounding. Figure 7-5 shows

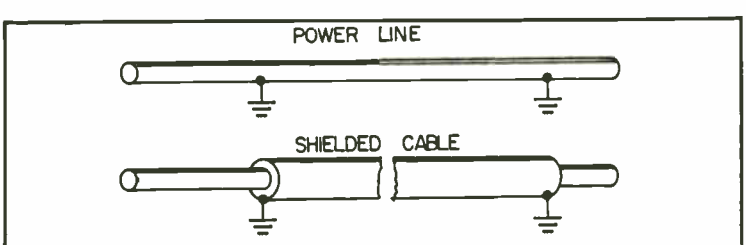


Fig. 7-5. Ground loops exist on a shielded cable when it is grounded at both ends.

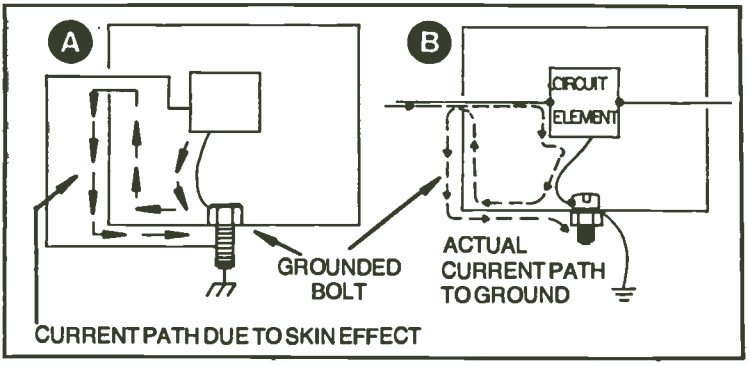


Fig. 7-6. Skin effect causes current to flow on the outer surface of conductors.

what may happen when the shield of a cable carrying a digital signal is grounded at both ends. Note that there is also a power line running between the same two points and it also is grounded at both ends. It is easy to see from the figure that the shield is electrically in parallel with the grounded side of the power line. This means that the shield will actually carry some of the current that would normally flow in the grounded side of the power line. Now ideally the power line current should flow only on the outside of the shield. But, as is often the way in the real world, some of it will also get on the inside of the shield and will show up mixed in well with the signal. We will have more to say about running long lines carrying digital signals in another chapter.

One thing that is often neglected in connection with shielding is the skin effect. We all know that at high frequencies, current flows only on a thin layer on the surface of a conductor. Of course, radio frequencies fall into this category. So do the high-frequency components of digital signals. Although a digital signal may have a fairly low pulse repetition rate, say 1000 pulses per second, the very steep leading and trailing edges of the pulses will have very high frequency components. Thus, both RF and digital shielding must take the skin effect into consideration.

Figure 7-6A shows a shielded enclosure. Note that the equipment inside the enclosure is connected to a screw at the bottom of the shield. The other end of the screw is connected to ground. This certainly looks like as good a ground as you can get—until we consider the skin effect. When we realize that the high-frequency components of the signal will not penetrate the metal of the shield but will travel along its surface, we see that this isn't a good arrangement at all. The signal will follow the path

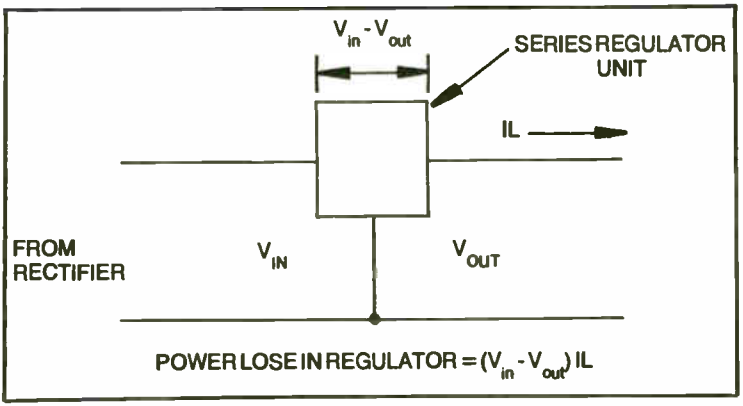


Fig. 7-7. Calculation of power loss in a series regulator.

shown by the dashed lines in Fig. 7-6A, out over the outer surface of the shield. This means that rather than being a good shield, our arrangement is much more like an antenna. The correct way to connect the shield of the cable to the shielded cabinets is shown in Fig. 7-6B.

SWITCHING REGULATORS

The IC voltage regulators that we discussed earlier in this chapter consume quite a bit of power in the process of voltage regulation. As shown in Fig. 7-7, the series-pass element of the regulator acts as a variable resistor that changes value as required to keep the output voltage constant.

As shown in the figure, the power dissipated in the regulator at any instant will be the product of the current flowing through it and the voltage drop across it. Although this power dissipation is small, it is significant in battery-operated equipment and in the interest of energy conservation.

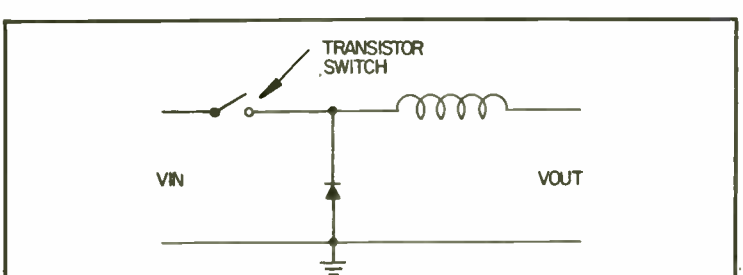


Fig. 7-8. A switching regulator relates voltage by varying the time that a switch is closed.

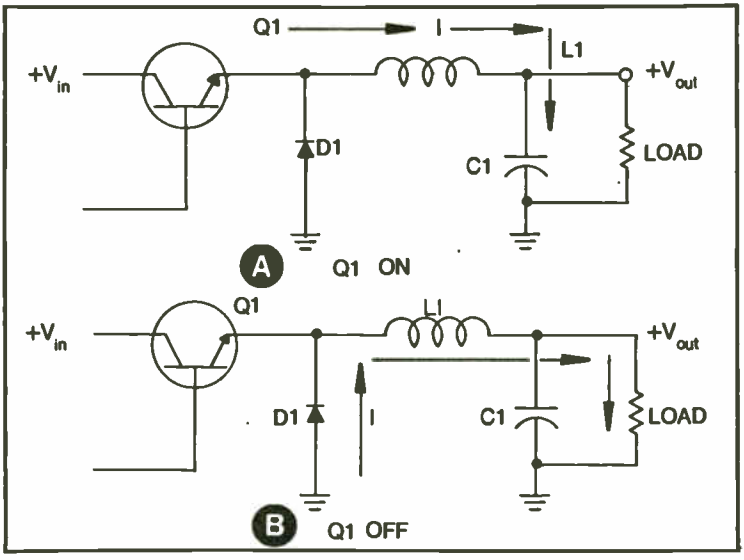


Fig. 7-9. Diagrams showing the action of a switching regulator.

There is another approach to voltage regulation, which is much more energy efficient. As shown in Fig. 7-8, the series element of the regulator acts more like a switch than a resistor. The output voltage is controlled by the percentage of the time the switch is closed.

To appreciate how a switching regulator can be more efficient, consider an ideal switch. When it is open, there is a voltage across it, but no current. When it is closed, current flows, but there is no voltage across the switch. If the switch changes instantaneously, the product of voltage and current will always be zero. Hence, no power will be dissipated in the switch.

Of course, a transistor isn't an ideal switch. There is always some voltage drop, however small, when it is turned on. As a result, there is always some power dissipation in the switching transistor, but it is much less than the dissipation in a conventional series regulator.

In addition to its high efficiency, the switching regulator also has the advantage of making the power supply smaller and lighter, because there is less heat to be dissipated.

Figure 7-9A, shows the circuit of a typical switching regulator. Note that, unlike earlier power supply regulators, it has two different components—diode D1 and inductor L1. The presence of these components in the power supply is a good hint that a switching regulator circuit is used.

First, let's look at how the switching transistor, together with the other components, accomplishes voltage regulation. When transistor Q1 is turned on, as in Fig. 7-9A, current flows through inductor L1. This current is applied to the load and also charges capacitor C1. When transistor Q1 is turned off, as in Fig. 7-9B, no current can flow through it. However, the magnetic field of inductor L1 collapses and the stored energy of the field causes current to continue to flow through L1 and through the load. This current can't flow through the transistor, since it is open, so it flows through D1, as shown.

The voltage across the load will tend to rise when transistor Q1 is turned on and to fall when it is turned off. To hold the average voltage across the load constant, we need to furnish the base of Q1 with an oscillating voltage that will vary the on and off times of Q1.

Figure 7-10 shows a circuit that can accomplish this. Here, we have an amplifier connected so that when voltage e_1 is higher than e_2, transistor Q1 will be turned on. When e_2 is higher, Q1 will be turned off. Voltage e_1 is obtained from a reference voltage and a circuit consisting of resistors R1 and R2 that provides positive feedback to the amplifiers.

Voltage e_2 is obtained from resistors R3 and R4 and is proportional to the voltage across the load. Now, assume for a minute that the load voltage is lower than it should be.

Under this condition, e_1 would be greater than e_2 for more than half of the cycle of oscillation. This would allow Q1 to be turned on for more than half of the cycle, which, in turn, would allow more current to pass, thus, boosting the output voltage to the desired value. Just the opposite situation would occur if the output voltage were to rise above the desired value.

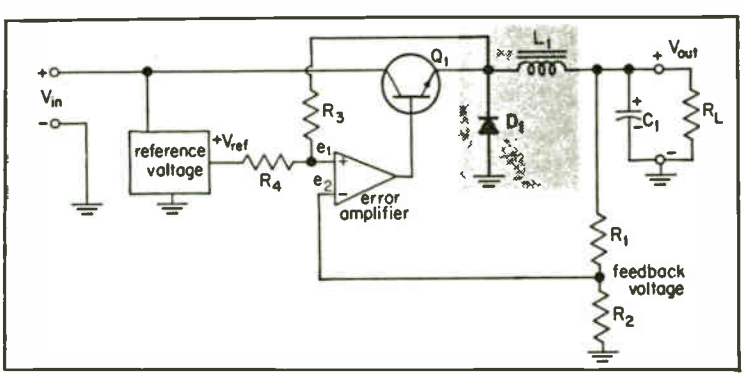


Fig. 7-10. Practical switching regulator circuit.

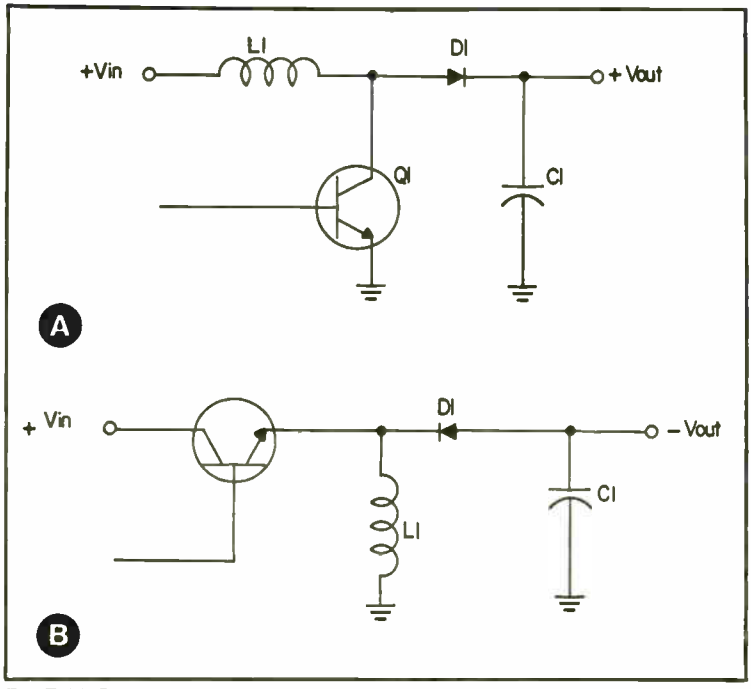


Fig. 7-11. Step-up (A) and polarity-reversing (B) switching regulator circuits.

In order to properly regulate the voltage, the frequency of oscillation of the switching regulator must vary with changes in supply voltage and load current. In practical regulators, the switching frequency varies between 5 and 100 kHz.

This switching phenomenon can cause considerable radiation, unless proper precautions are taken. This, of course, is a design rather than a maintenance problem. Sometimes, however, interference may be encountered when a system using a switching regulator is connected to another system with inadequate shielding.

The switching regulator has another unique feature. By rearranging the components, it can step up voltage so that the output voltage is actually higher than the input voltage. It can also reverse the polarity of the input voltage so that the output voltage is negative with respect to ground.

Figure 7-11A shows how the components are arranged to step up the voltage and Fig. 7-11B shows a circuit where the polarity of the output voltage will be inverted with respect to the input voltage.

Chapter 8
The Operational Amplifier

Although it isn't a digital component, the operational amplifier, or op-amp as it is usually called, is a very useful integrated circuit and is used extensively in circuits that convert signals from analog to digital form. For this reason, we will take a brief look at how it works.

OP-AMP CHARACTERISTICS

Figure 8-1 shows the symbol for an op-amp. If this amplifier were ideal, it would have infinite gain, an infinite input impedance and practically zero output impedance. In real life, op-amps don't realize these specifications, but they come close enough for many practical purposes. Note that the op-amp has two power supply leads, one positive and one negative, with respect to ground. With this arrangement, the output can swing linearly above and below zero. In a typical IC op-amp, the output swing may be ±10V. The gain of an op-amp is typically between 100,000 and 1,000,000.

The op-amp shown in Fig. 8-1 has two inputs, one marked positive (+) and one marked negative (−). This doesn't mean that positive voltages are applied to one input and negative voltages to the other. Rather, it indicates the phase of the output voltage with respect to the input voltage. If a signal is applied to the plus (+) or noninverting input, the output voltage will be in phase with the input voltage. If the signal is applied to the minus (−) or inverting input, the output will be inverted or 180 degrees out of phase with the input. This inverting input is the one most commonly used, because an op-amp is almost always used with negative feedback to stabilize it and hold its parameters constant.

As we noted, the gain of an op-amp is extremely high as compared with other amplifiers with which we may be familiar. It takes only a very small input signal to saturate the amplifier. In Fig. 8-2, we have plotted the output of an op-amp as a function of the input voltage. Note that as the input voltage changes from zero the output voltage will change linearly as we might expect, but that as soon as the input voltage is either positive or negative a few microvolts with respect to ground, the amplifier will saturate. That is, the output will have reached its highest possible value—in this case, either plus or minus 10V with respect to ground.

THE OP-AMP WITH FEEDBACK

It doesn't take much imagination to tell that if we were to attempt to operate an op-amp without feedback, we could easily run into problems with oscillation. With the gain over 100,000 and the pins separated only by a small fraction of an inch, conditions leading to oscillation could readily be encountered.

By using negative feedback around the op-amp, we will reduce its gain, but we will also produce a circuit where such things as the gain depend only on the external components, such as resistors, and not on the properties of the op-amp itself.

Figure 8-3 shows an op-amp with a feedback network consisting of only two resistors. Here, the input current is I_1 and the feedback current is I_2. The circuit is easy to analyze because the gain of the op-amp is so high that for all practical purposes, we can assume that no current flows into the input of the amplifier at all. Of course, there must be an extremely small current flowing into the

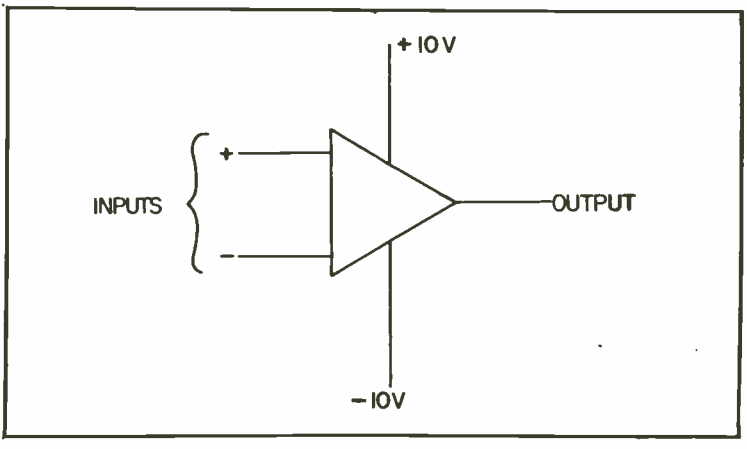


Fig. 8-1. Symbolic representation of an op-amp.

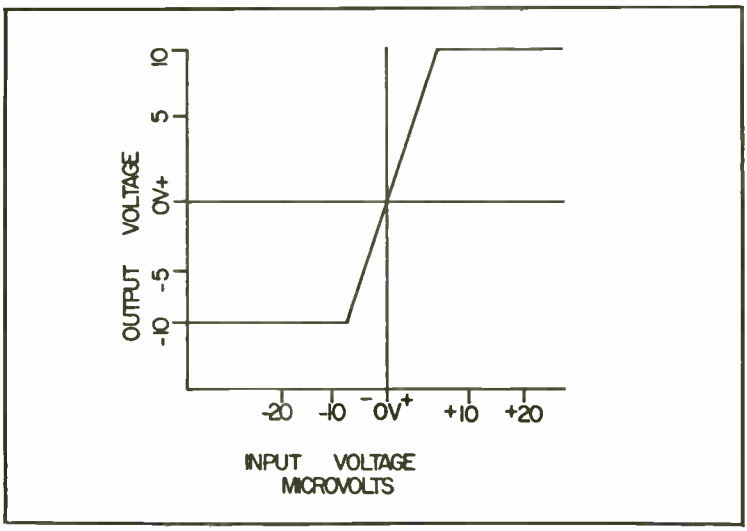


Fig. 8-2. Output voltage of an op-amp as a function of input voltage.

input of the amplifier, if we are to get any output, but this current is really infinitesmal compared with the other currents in the circuit. There is practically no error at all in neglecting it.

We can, therefore, consider that the input current to the circuit I_1 is numerically equal to the feedback current I_2. Another way of looking at the same point is to assume that the input voltage of the op-amp is so close to zero, for any real output, that we can consider the voltage at point A of Fig. 8-3 to be zero; that is, we can consider point A to be a "virtual" ground. With these simplifying assumptions, the analysis of the circuit of Fig. 8-3 comes very easy.

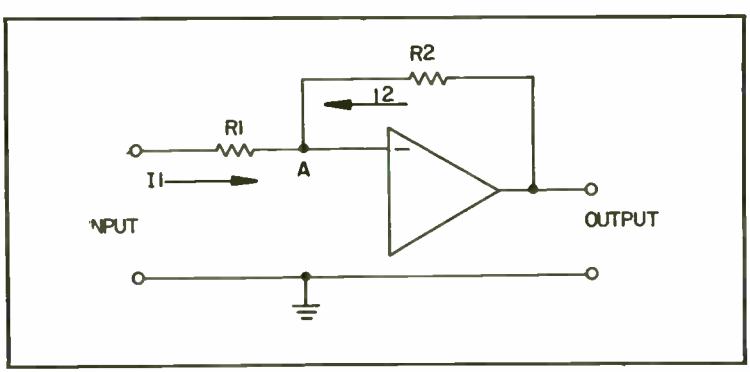


Fig. 8-3. Op-amp with negative feedback.

The input current I_i is simply equal to $V_{in}/R1$, and the feedback current is equal to $-V_o/R2$. Inasmuch as these two currents are equal, we can write the following equation:

$$\frac{-V_o}{R2} = \frac{V_{in}}{R1}$$

With a simple rearrangement of terms, this becomes:

$$V_o V_i = \frac{-R2}{R1}$$

which means that the gain of the circuit is equal only to the ratio of the feedback resistance to the input resistance.

Note carefully that the gain depends only on the ratio of the resistances, and not on anything else, such as supply voltages or the parameters of the op-amp itself. This relationship will hold as long as the actual gain of the op-amp, by itself, is much higher than the gain of the final circuit, which is almost always the case. The minus sign in the equations merely indicates that the output voltage is 180 degrees out of phase with the input voltage.

So far, we have only used the minus (–) or inverting input of the op-amp. We can, if we wish, build a noninverting stage by applying the input voltage to the plus (+) or noninverting input and use the minus (–) or inverting input for our negative feedback. The arrangement is shown in Fig. 8-4. The expression for the gain of the complete stage is derived in a similar way.

Figure 8-5 shows a practical example of the use of feedback around an op-amp to set the gain. Here, the input resistance is 10k,

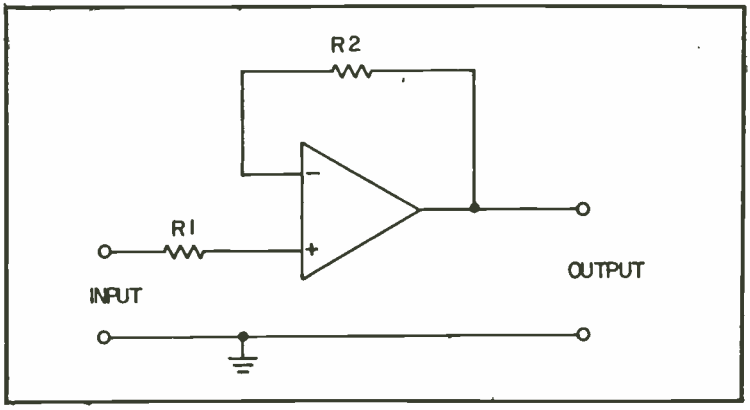


Fig. 8-4. Op-amp using the non-inverting input for signal input and the inverting input for negative feedback.

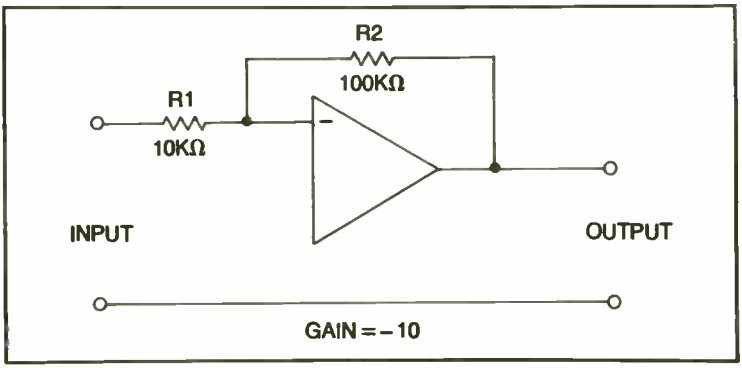


Fig. 8-5. Practical amplifier circuit.

and the feedback resistance is 100k. From the equations that we derived earlier, we can see that the gain of the stage is:

$$\frac{V_o}{V_{in}} = \frac{-R2}{R1} = \frac{-100K}{10K} = -10$$

Although this simple circuit arrangement has many useful applications in digital systems, more complex circuits are often found. We will look at the most common of these in the following paragraphs.

SUMMING WITH THE OP-AMP

Figure 8-6 shows how an op-amp can be used to add two signals. This doesn't look very impressive, but it is very useful in converting signals from digital to analog form, as we will see in a later chapter. As shown in Fig. 8-6, if we connect two separate input signals to an op-amp through separate input resistors, the output will be equal to the sum of the two input voltages. Before going any further with this, let's see how it works.

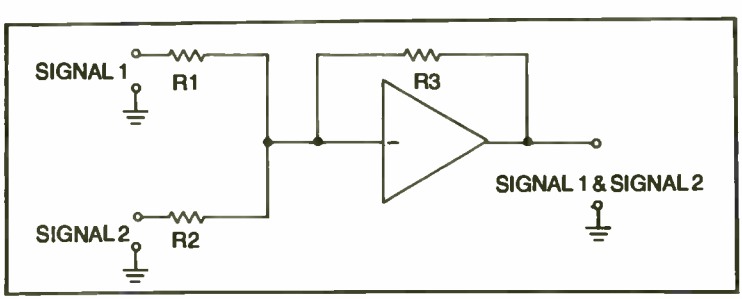


Fig. 8-6. Circuit for using an op-amp to sum two signals.

Figure 8-7 shows a simplified form of the circuit of Fig. 8-6. As we noted earlier, the input terminals of the op-amp, for all practical purposes, don't draw any current at all. So, we have left it out of the diagram. Now we have two currents flowing into point A, and the only place these currents can go is through the feedback resistor. We also noted earlier that, with the feedback arrangement, the output voltage adjusted itself, so that the voltage at point A would be so close to ground potential that we could consider it to be at ground. Now we can write an equation that says that the sum of the currents flowing into point A from the two inputs will be equal to the current in the feedback resistor, R3:

$$\frac{V_1}{R1} + \frac{V_2}{R2} = \frac{V_3}{R3}$$

We can arrange the equation to read:

$$V_1 \frac{R1}{R3} + V_2 \frac{R3}{R2} = -V_0$$

and if we assume that the input resistors and the feedback resistor are all equal, the equation simplifies to:

$$V_1 + V_2 = -V_0$$

Here the minus sign only means that the output voltage is out of phase with the input voltage, as we might expect.

WEIGHTING THE INPUTS

In the two circuits that we have just discussed, all of the input voltages are added directly. We don't have to do this. By changing

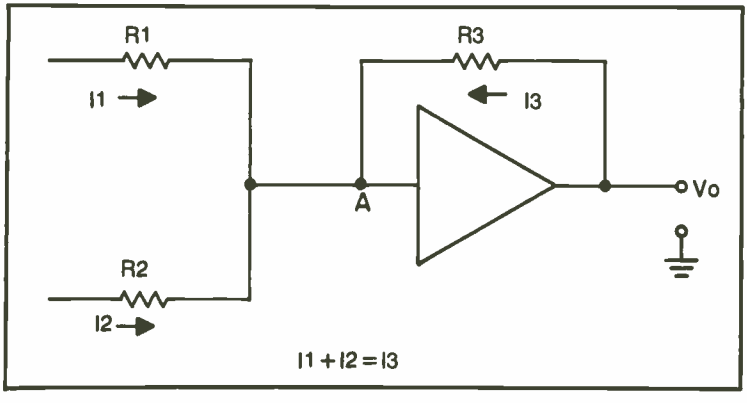


Fig. 8-7. Schematic showing currents and voltages in a summing amplifier.

the values of the input resistors, we can scale, or weight, the inputs so that we will only add a fraction of any given input voltage to the others. This feature is again very useful in digital circuits.

Figure 8-8 shows a summing circuit where the input resistances are not equal. To keep things comparatively simple, we have made all of the input voltages equal.

From the equations we developed earlier, it is easy to see that we get twice as much output voltage when we apply a signal to input 2 as when we apply the same signal to input 1.

This concept of summing signals, particularly with weighted inputs, should be well understood. It is important in many digital systems.

The op-amp is the basic circuit element of some other so-called linear ICs that are often found in digital systems.

THE COMPARATOR

A comparator circuit, such as that shown in Fig. 8-9, is basically a high-gain op-amp, operated without feedback. It's purpose is simply to compare one voltage with another.

As we pointed out earlier in this chapter, an op-amp without feedback, has an extremely high gain—usually at least 100,000. In practical terms, this means that if the input voltage in Fig.8-9 is even very slightly more positive than the reference voltage, the amplifier will go into positive saturation. The ouput will have its maximum positive voltage. Similarly, if the input voltage is even slightly more negative than the reference voltage, the output will go into negative saturation. The output will have its maximum negative value.

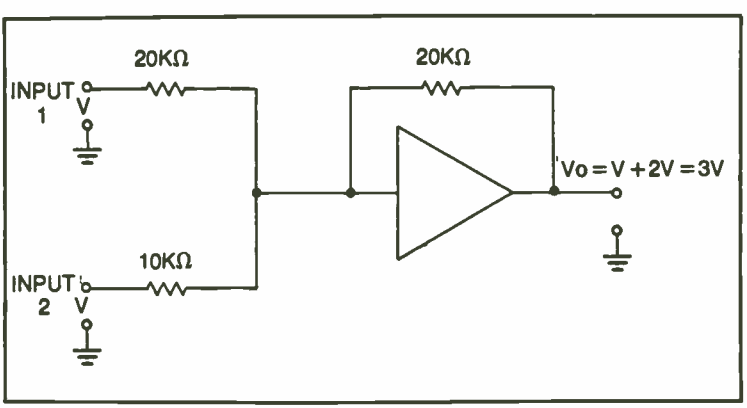


Fig. 8-8. Circuit showing unequal weighting in a summing amplifier.

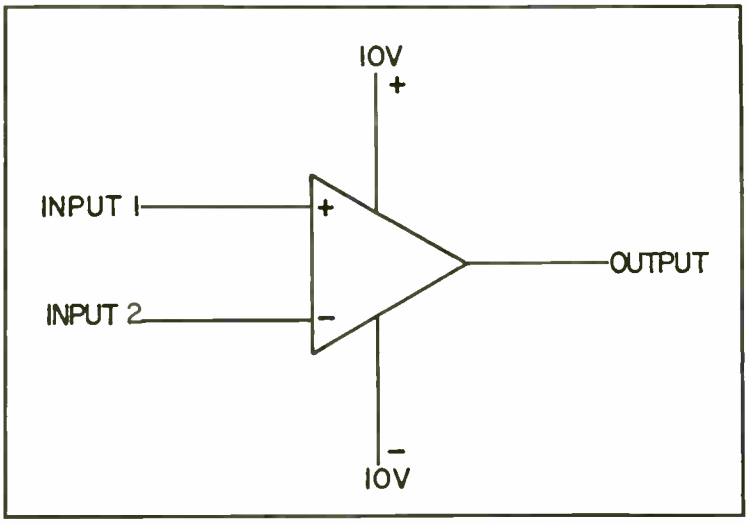


Fig. 8-9. Op-amp without feedback as a comparator.

Although comparator circuits can be made from operational amplifiers, there is no need to do this. Comparator circuits are available in ICs. The advantage of using IC comparators rather than building comparators using regular op-amps is that all of the design subtleties have been taken into consideration. The IC comparator will usually work when it is connected in a circuit. The homemade comparator may involve a considerable amount of debugging before it will operate properly.

Another advantage of the IC comparator is that it usually operates with a single power supply and the output can be made to swing between ground and +5V, so that it will be fully compatible with TTL logic ICs.

Chapter 9
Getting In and Out of the Digital World

From the preceding chapters, we can see that various digital components are quite compatible with each other. Gates and flip-flops can be connected together in almost any arrangement with few problems. When we attempt to interface a digital system with the real world, the situation becomes more complicated.

If a digital system is to perform a useful function, it must receive input from the real world and then deliver outputs back to the real world. The input signal will come either from regular analog equipment or from a human being, neither of which is very compatible with a digital system.

In the first chapter of this book, we discussed the process of sampling and "quantizing," so that we could convert an analog signal into digital form. In this chapter, we will describe some of the circuits that are actually used for this purpose.

There are two different classes of devices that we use to get in and out of a digital system. The first class of such devices are called analog-to-digital (A/D) and digital-to-analog (D/A) converters. These devices actually operate on signals. In an A/D converter, an analog signal is actually converted into a digital signal. Similarly, in a D/A converter, a digital signal is actually converted into an analog signal.

There is another, somewhat simpler, class of input/output (I/O) devices used to get in and out of digital systems. These devices use things such as switches and keyboards to generate inputs to digital systems. The outputs of digital systems are used to activate something like a light or a lamp, to give an indication of the output.

The familiar pocket calculator, shown in Fig. 9-1, is a good example of such input and output devices. The input device is a keyboard labeled with regular decimal numbers. An operator can press the keys with no knowledge of what happens inside the calculator. The entries are in the familiar decimal number system. Of course, the calculator, being a digital system, can't use decimal numbers. The interfacing is accomplished by circuits that produce binary numbers, corresponding to the decimal numbers that are entered. Although the output of the calculator circuits is a binary number, the interface circuits decode these numbers so that they

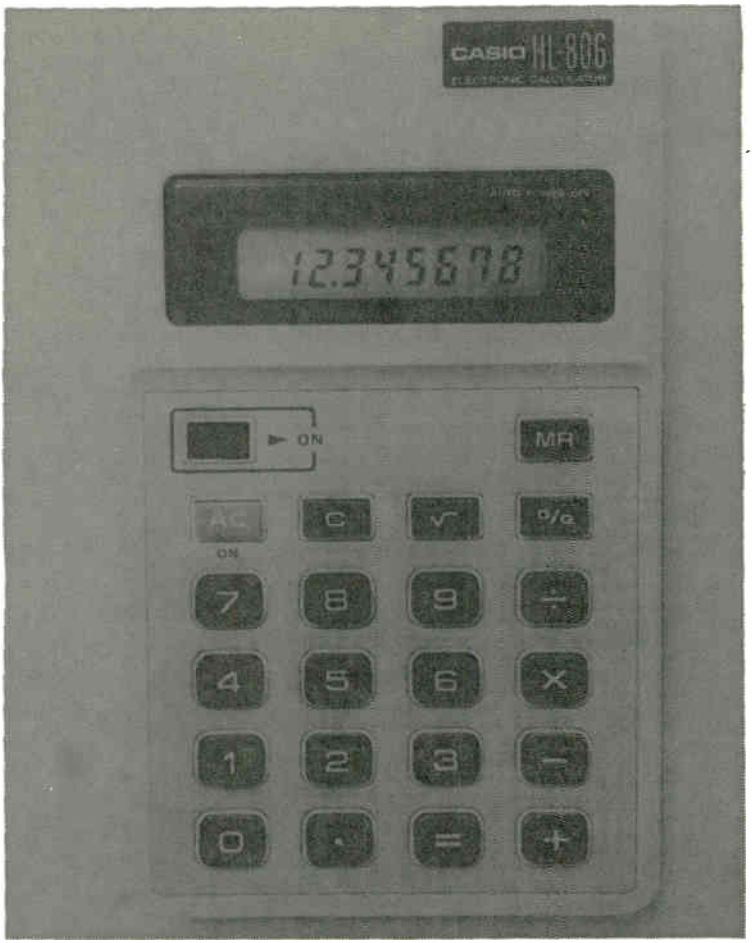


Fig. 9-1. The familiar calculator uses a keyboard as an input device and a liquid-crystal readout as an output device (courtesy of CASIO).

can energize the familiar 7-segment readout. The input and output devices of a digital system are often called encoders and decoders, respectively, or simply I/O (for Input/Output) devices.

SWITCHES AND KEYBOARDS

Figure 9-2 shows a very simple switching arrangement that can be used to enter data into a digital system. Here, switches are arranged to pull each of the lines of a bus to either a high or low logic level. Of course, this arrangement has many limitations. It is slow, and the switches must be reset for each bit of data that is entered. Nevertheless, the arrangement is practical in application, where the data is seldom changed once it is entered.

A much more useful input device is the keyboard shown in Fig. 9-3. Each of the keys is merely a switch, but the keyboard is connected to an IC that produces a unique binary number corresponding to each key of the keyboard. Also shown is a cathode-ray tube display of data.

In a similar way, switches and relays can be used to generate inputs for digital systems. For example, interlock switches on cabinet doors can be used to generate inputs for control systems.

SHAFT ENCODERS

A shaft encoder is an electromechanical device that looks something like a precision potentiometer. Its input is a mechanical shaft and its output is a parallel digital number that accurately represents the postion of the shaft. Figure 9-4 shows several shaft encoders.

The shaft encoder is of interest for two reasons. In the first place, with the proliferation of digital systems there is an increasing need for devices that will convert mechanical displacements. Secondly, the fact that the shaft encoder is a mechanical device enables us to get a better insight into the conversion process.

There are several different devices that are used to produce digital signals from shaft rotation. The simplest is a *tachometer encoder*. This device simply generates pulses as the shaft rotates. The output is simply a train of pulses, such as that shown in Fig. 9-5A. The pulses can be counted digitally to determine the total number of revolutions of the shaft. Similarly, the pulses can be counted for a known time interval to determine velocity.

An improvement over the tachometer encoder is the incremental encoder. This device usually has three outputs. Two

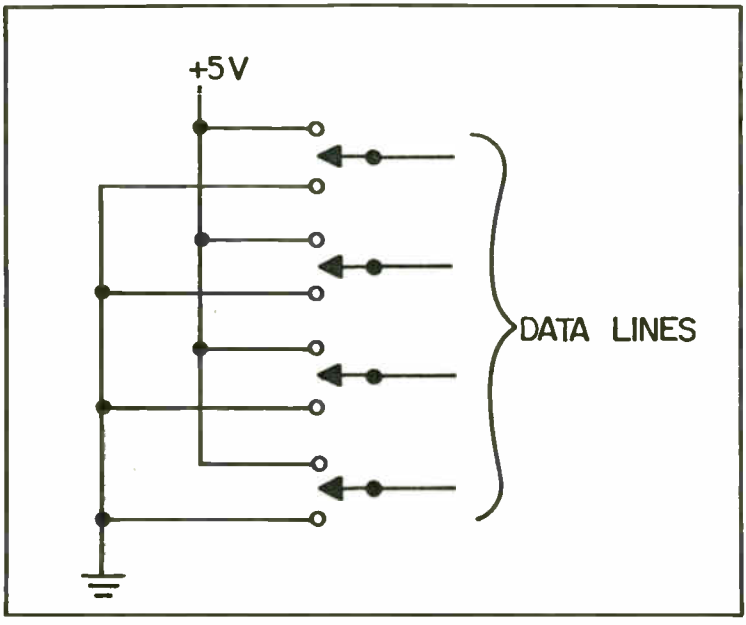


Fig. 9-2. Switches can be used to enter digital data into a system.

outputs produce pulses that result from rotation of the shaft. The third output, called the incremental output, identifies a unique position of the shaft. The pulses, shown in Fig. 9-5B, can be used in

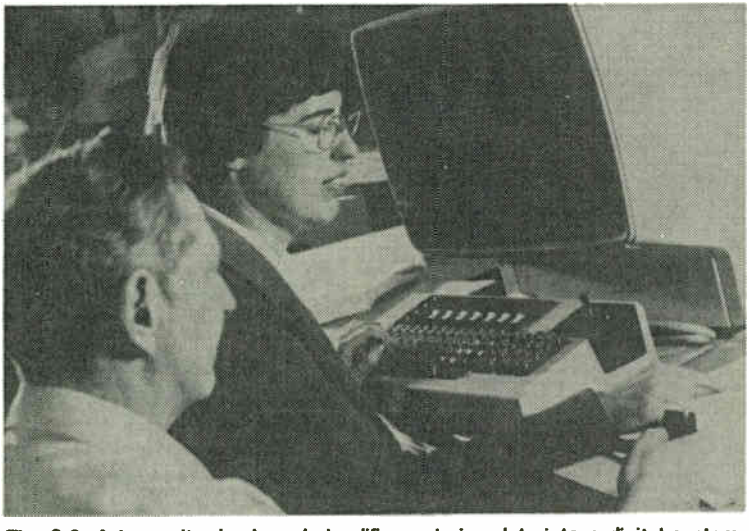


Fig. 9-3. A typewriter keyboard simplifies entering data into a digital system (courtesy of IBM).

Fig. 9-4. Shaft position encoders convert a shaft position into a digital signal (courtesy of BEI Electronics Inc.)

the same way as those from a tachometer encoder, but can also be processed digitally to show the *direction* of rotation of the shaft.

The most elaborate shaft encoder is called an absolute position encoder. It has between 6 and 20 output connections. The output leads carry a digital number that identifies the shaft position. Figure 9-6 shows how the outputs of a 4-bit encoder change with shaft rotation.

The resolution of an encoder increases with the number of outputs. Inasmuch as a 4-bit binary number can represent 16 different values, including zero, its resolution will be one part in 16. That is, it can resolve the shaft position into increments of 360/16 = 22.5 degrees. For precision applications, much higher resolution is required.

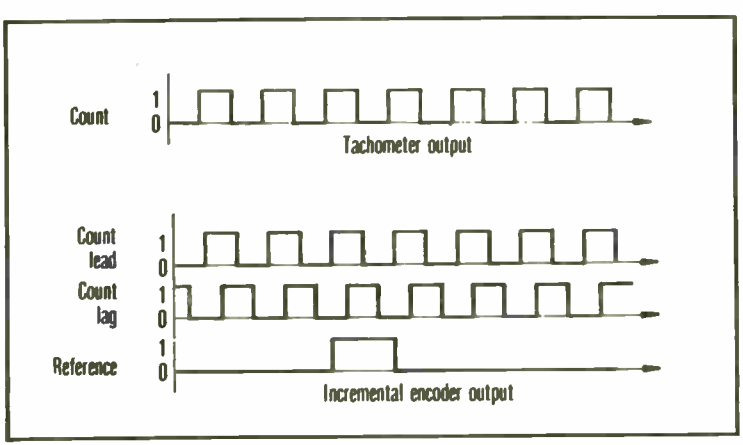


Fig. 9-5. Pulses from shaft encoders (courtesy of BEI Electronics Inc.)

READOUTS

By far, the most common readout used with digital systems is the familiar 7-segment readout seen on calculators, cash registers, and all sorts of electronic equipment.

The most common 7-segment readout uses light-emitting diodes (LEDs) to make up its segments. However, fluorescent tubes with similar elements are also available. For low-power applications, 7-segment readouts using liquid crystals are also widely used.

The segments of the 7-segment readout, together with their commonly used alphabetical designations, are shown in Fig. 9-7A. Figure 9-7B shows which segments are lighted to form each of the numerals 0 through 9 and some commonly used letters.

Figure 9-8 shows a typical decoder that accepts a BCD input and displays the numerals 0 through 9 and the letters A through F.

DIGITAL TO ANALOG (D/A) CONVERTERS

Since in a real system we start out with an analog signal and convert it to digital form, it would seem that the logical way to approach the subject would be to treat the A/D converter first. Unfortunately, some A/D converters use a D/A converter as an integral part. For this reason, things will be easier to explain if we start out with the D/A converter.

Figure 9-9A, shows a 4-bit binary signal. Inasmuch as all of the lines are high, the decimal equivalent of the binary number is:

$$1111 = 8 + 4 + 2 + 1 = 15$$

From this, we can see that if we could find a way to assign the proper weight to the signals on the lines of Fig. 9-9A, we could simply add these signals together to get the decimal equivalent.

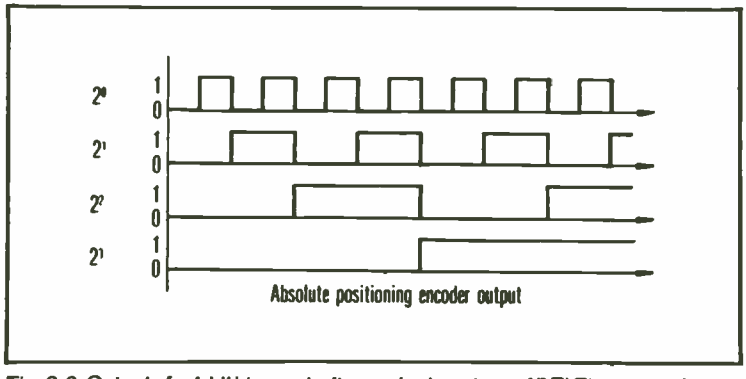


Fig. 9-6. Output of a 4-bit binary shaft encoder (courtesy of BEI Electronics Inc.)

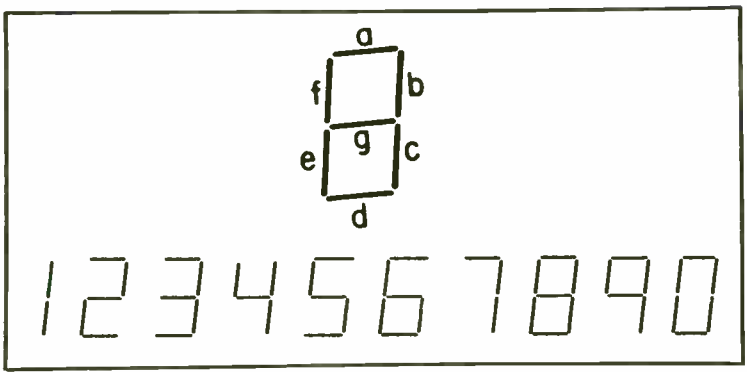
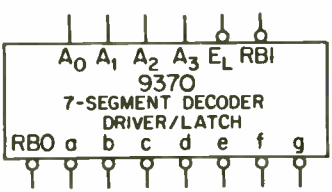


Fig. 9-7. Seven-segment readout used to display decimal numbers.

That is:

$$1111 = (8\times1) + (4\times1) + (2\times1) + (1\times1) = 15$$

A functional arrangement is shown in Fig. 9-9B. Here, the output will be a voltage that is numerically equal to the decimal



Segments Active Low
Open Collector Output Circuit

	Inputs					Outputs								
	$\overline{RBI}$	A_3	A_2	A_1	A_0	$\overline{a}$	$\overline{b}$	$\overline{c}$	$\overline{d}$	$\overline{e}$	$\overline{f}$	$\overline{g}$	$\overline{RBO}$	Display
*	X	X	X	X	X	H	H	H	H	H	H	H	L*	blank
0	L	L	L	L	L	H	H	H	H	H	H	H	L	blank
0	H	L	L	L	L	L	L	L	L	L	L	H	H	0
1	X	L	L	L	H	H	L	L	H	H	H	H		1
2		L	L	H	L	L	L	H	L	L	H	L		2
3		L	L	H	H	L	L	L	L	H	H	L		3
4		L	H	L	L	H	L	L	H	H	L	L		4
5		L	H	L	H	L	H	L	L	H	L	L		5
6		L	H	H	L	L	H	L	L	L	L	L		6
7		L	H	H	H	L	L	L	H	H	H	H		7
8		H	L	L	L	L	L	L	L	L	L	L		8
9		H	L	L	H	L	L	L	H	H	L	L		9
10		H	L	H	L	L	L	L	H	L	L	L		A
11		H	L	H	H	H	H	H	L	L	L	L		b
12		H	H	L	L	L	H	H	L	L	L	L		C
13		H	H	L	H	H	L	H	L	L	H	L		d
14		H	H	H	L	L	H	H	L	L	L	L		E
15	X	H	H	H	H	L	H	H	H	L	L	L	H	F

*RBO used as an input

Fig. 9-8. Decoder used to drive a 7-segment readout from a 4-bit BCD signal.

value of the binary signal applied to the input. For example, if the input were 1001, the output would be:

$$(8\times1)+(4\times0)+(2\times0)+(1\times1)=9$$

which, of course, is the decimal equivalent of 1001.

One problem, then, is to find a circuit that will perform the function of the arrangement shown in Fig. 9-9B. It turns out that this function can be performed by a single operational amplifier, connected in a summing circuit with weighted inputs. This arrangement was discussed in Chapter 8.

Figure 9-10, shows the circuit of a simple 4-bit D/A converter, using an op-amp. The resistor values are chosen to properly weight the inputs. The output voltage of this circuit will be:

$$V=-\left[8\frac{R}{R}A_8+4\frac{R}{R}A_4+2\frac{R}{R}A_2+\frac{R}{R}A_1\right]V$$

or:

$$V=-\left[8A_8+4A_4+2A_2+A_1\right]V$$

where V is the voltage level at each of the inputs and the minus sign is due to the fact that the output of the op-amp is out of phase with its input.

To keep things simple, let's assume that we have OV at an input to represent a logical O and 1V to represent a logical 1. In other words, if the input is 0110 (6 in decimal), the input voltages will be:

$$A_1=0V$$
$$A_2=1V$$
$$A_4=1V$$
$$A_8=0V$$

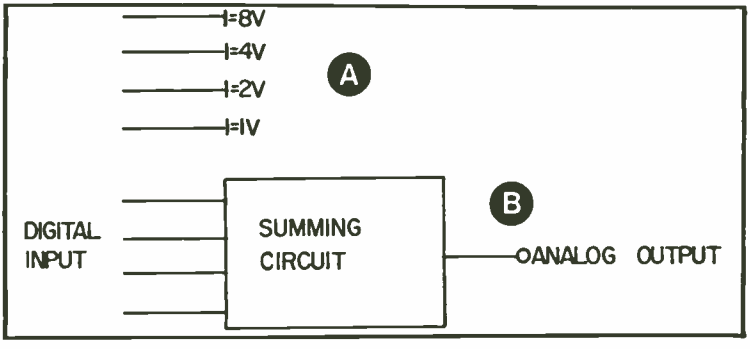


Fig. 9-9. Summing and weighting digital signals to get an equivalent analog signal.

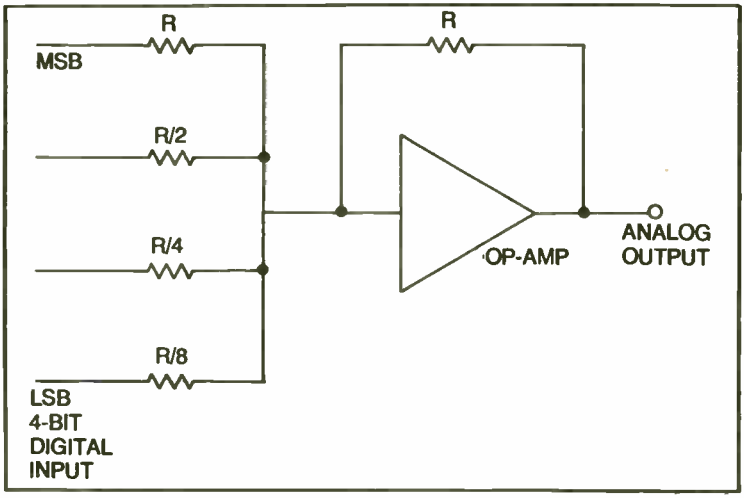


Fig. 9-10. Diagram of a 4-bit A/O converter.

Then the output of the circuit will be:

$$V = -(0\times8) + (1\times4) + (1\times2) + (0\times1) = -6V$$

showing that we have, indeed, obtained a voltage whose value corresponds to the decimal value of the digital input.

In Fig. 9-11A, we show a table of the output of the circuit for all possible input values. A plot of the output is shown in Fig. 9-11B. Note that the output is a staircase, varying in discrete steps. This is because the digital input signal changes in discrete steps. Usually, however, the output curve of a D/A converter is plotted as a straight line, as shown by the dashed line of Fig. 9-11B.

There are several features of our simple D/A circuit that make it impractical. It has poor accuracy and stability. Furthermore, most op-amps of the type we might use will not swing to a full 15V output. A 10V output is much more common.

SCALING DATA

We can overcome the limitation of the voltage range of an op-amp by scaling the data so that the circuit will never call for an output of which our op-amp is not capable.

Suppose, for example, that the output of our op-amp can only swing to $-10V$. We can use the same setup with resistor values, shown in Fig. 9-12. The output voltage of this circuit is given by:

$$V = -\left[\frac{8R}{R}A_8 + \frac{4R_f}{R}A_4 + \frac{2R_f}{R}A_2 + \frac{R_f}{R}A_1\right]V$$

We can simplify this a little to:

$$V = \frac{-R_f}{R} (8A_8 + 4A_4 + 2A_2 + A_1) V$$

The greatest output will occur when all of the input bits are high and will be:

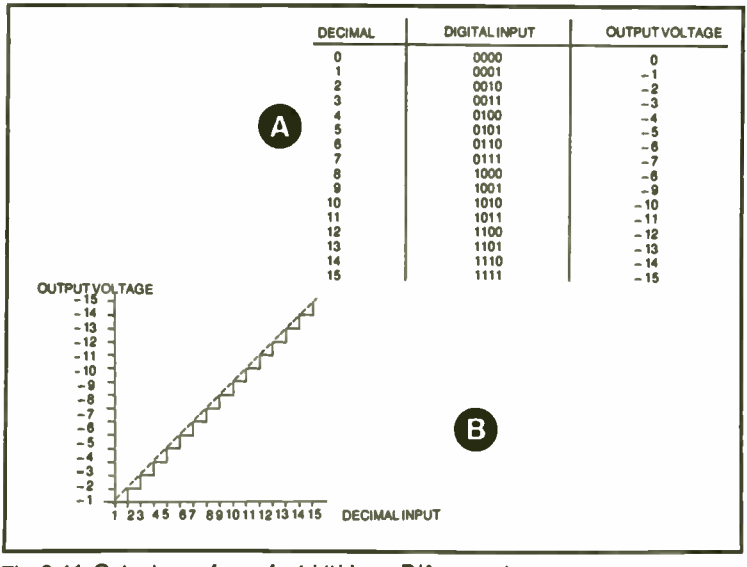
$$V = -\frac{R_f}{R} (8 + 4 + 2 + 1) V$$

$$V = -15 \frac{R_f}{R} V$$

Thus, we can set the ratio of R_f R to take into consideration both the desired maximum output voltage as well as the input voltage.

One limitation of our circuit is that we have tacitly assumed that when an input was a logical zero, the input voltage would be zero, and that when the input was a logical 1, the input would be exactly 1V. Neither of these values is apt to prevail in the real world. A logical zero might produce a voltage anywhere between zero and +0.4V, and the voltage corresponding to a logical high will vary even more.

Our op-amp will sum the actual voltages applied to its inputs, so any change in an input voltage from the ideal value will introduce errors.



DECIMAL	DIGITAL INPUT	OUTPUT VOLTAGE
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	-10
11	1011	-11
12	1100	-12
13	1101	-13
14	1110	-14
15	1111	-15

Fig. 9-11. Output waveform of a 4-bit binary D/A converter.

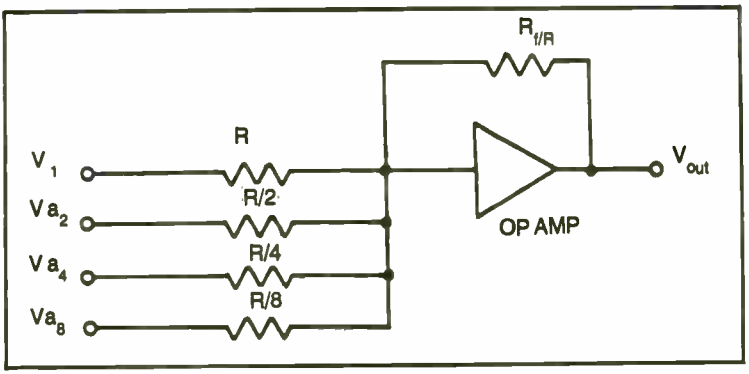


Fig. 9-12. Output voltage is scaled by choosing $R_{f/R}$.

This problem may be solved by deriving the actual input voltages from a regulated source and using the digital input merely to turn semiconductor switches on and off.

One of the limitations of this type of D/A converter is that it is not suitable for use with digital words much longer than eight bits. For the circuit to work properly the highest value of resistor must be much smaller than the actual input resistance of the operational amplifier. We have been assuming that the input resistance of the op-amp is infinite. But, of course, in practice this is not true. At the other extreme, if we use semiconductor switches, the smallest resistance must be much larger than the internal resistance of the semiconductor switch when it is turned on. The R2R converter, described in the following paragraphs, overcomes these limitations.

R2R D/A CONVERTER

Figure 9-13 shows what is called an R2R digital-to-analog converter. It derives its name from the fact that the heart of the

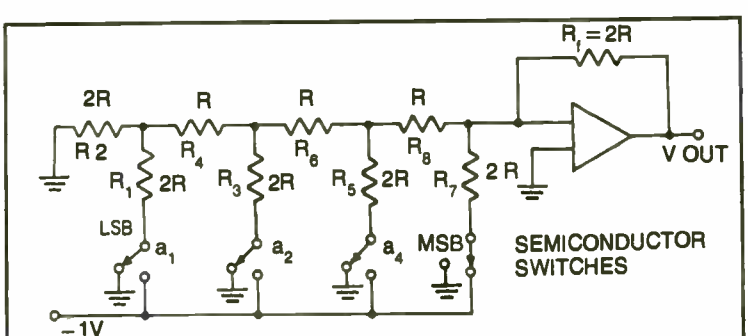


Fig. 9-13. Schematic of an R2R D/A converter.

converter is a ladder network, consisting of resistors, where the series resistors have a value of R and the shunt resistors have a value of 2R.

The operation of this converter can be rather difficult to understand at first. The heart of the circuit is the ladder network itself which has some rather interesting properties. If we were to cut the network along the dashed line A in Fig. 9-14, it is easy to see that the resistance to ground of the portion at the left is equal to R, because it consists of two resistances equal to 2R connected in parallel.

Similarly, if we were to cut the network along the line B, we would also find that the resistance to ground of the portion to the left of the line would again be R ohms. The same thing would also be true at lines C and D.

Now, the best way to see how the R2R network, working together with the summing op-amp, weights the various inputs is to consider the inputs one at a time. Suppose that only the most significant bit is turned on. The circuit would be as in Fig. 9-15A. This is simply a summing op-amp with one input connected to ground and the other input connected to the +1V source. Inasmuch as the resistor in the –1V branch is equal to the feedback resistor, the output will be +1V. The input voltage is negative because the op-amp inverts the signal, thus the output will be a positive voltage with respect to ground.

If we try to see what happens when only the second most significant bit, A4, is turned on, we can get really confused. As shown in Fig. 9-15B, at point C the network looks like a voltage divider with R5 making up the top half and the rest of the network

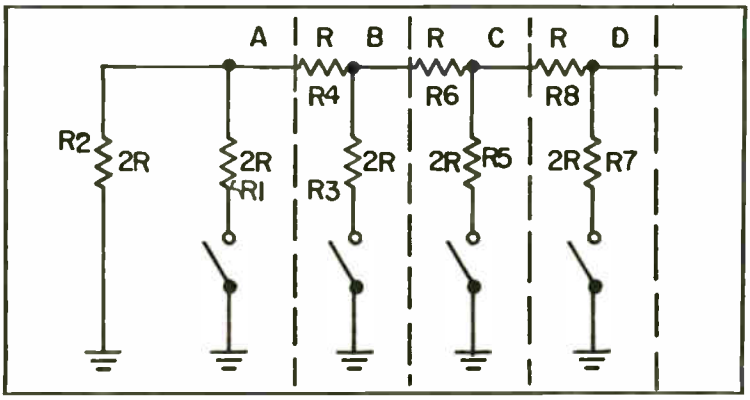


Fig. 9-14. The resistance to ground to the left of points A, B, C, or D is equal to R.

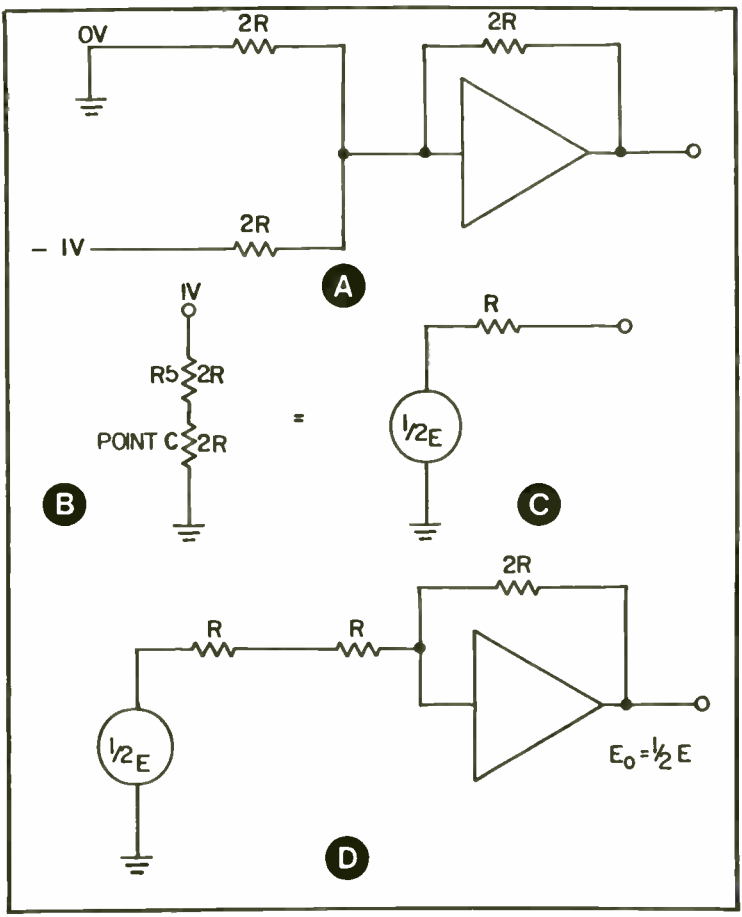


Fig. 9-15. Analysis of the R2R D/A converter circuit.

making up the bottom. The voltage at point C would thus be $-1/2$V. So far so good, but if we start to apply this to the summing circuit, the gain of the amplifier will turn out to be wrong. The way around the problem is to get a little theoretical and apply Thevenin's theorem to the circuit of Fig. 9-15B. All this means is that a circuit like that of Fig. 9-15B looks electrically exactly like the circuit shown in Fig. 9-15C. That is, it looks like an ideal voltage source in series with a resistor of R ohms. Now that we know this, we can construct the rest of the equivalent circuit as in Fig. 9-15D. Here we see that we are in effect applying a $-\frac{1}{2}$V source to the amplifier through a summing resistor equal to 2R so that the gain of the amplifier will be 1 and the output will be $+\frac{1}{2}$V.

If we continue this method of analysis to the other inputs we will see that each input will be weighted so that when it is energized, the output will be one half of what the next most significant input would provide. Therefore, we have the binary weighting we want in a D/A converter. Figure 9-16 shows a plot of the output of the circuit as a function of which inputs are energized.

Note that as we add more steps to the R2R converter, the output voltage doesn't increase. All that happens is that the increments between the various steps become smaller. The result

DECIMAL	DIGITAL INPUT Q_8	Q_4	Q_2	Q_1	OUTPUT, VOLTS
0	0	0	0	0	0.000
1	0	0	0	1	0.125
2	0	0	1	0	0.250
3	0	0	1	1	0.375
4	0	1	0	0	0.500
5	0	1	0	1	0.625
6	0	1	1	0	0.750
7	0	1	1	1	0.875
8	1	0	0	0	1.000
9	1	0	0	1	1.125
10	1	0	1	0	1.250
11	1	0	1	1	1.375
12	1	1	0	0	1.500
13	1	1	0	1	1.625
14	1	1	1	0	1.750
15	1	1	1	1	1.875

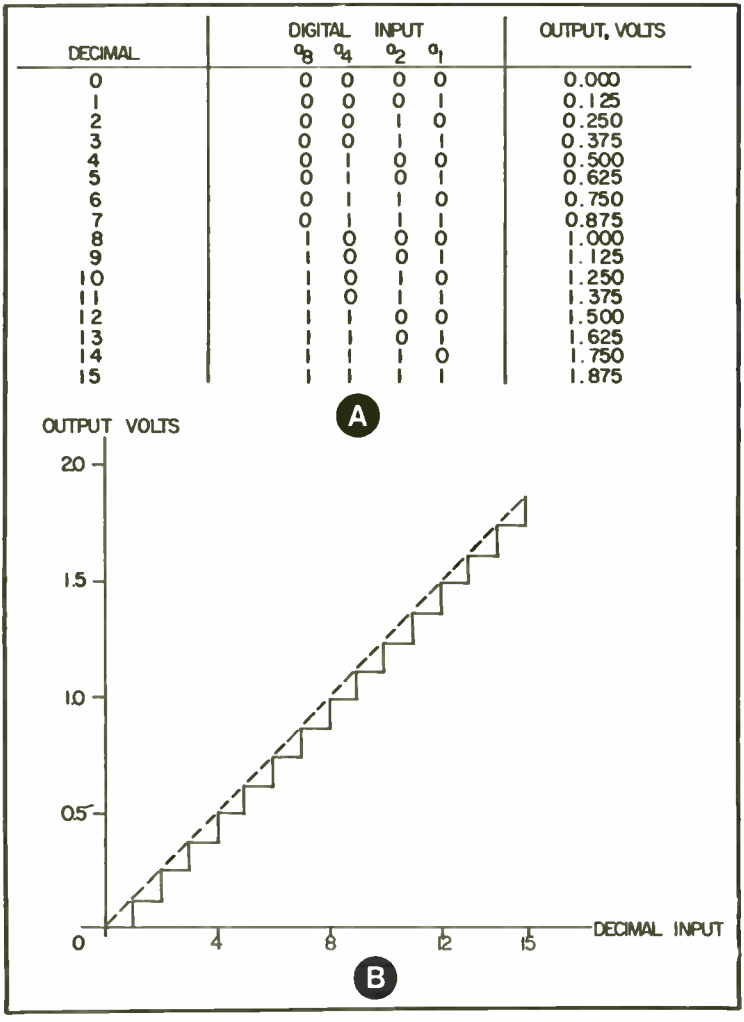


Fig. 9-16. Output of an R2R D/A converter.

is that we can build D/A converters with as much resolution as we wish using this principle. The resistance values are still simply R and 2R ohms.

SPECIFICATION OF D/A CONVERTERS

One of the problems in maintaining a digital system is ascertaining that all of its subsystems are working properly. Inasmuch as the D/A converter involves both digital and analog circuits, it is sometimes difficult to tell whether it is performing the way it should. For this reason, some specifications are given and some of these will be described in the following paragraphs.

Accuracy

The accuracy of the D-to-A converter is a very general term, relating the actual output to what we expect to get for an output. Accuracy is usually stated as a percentage of a full-scale output. For example, if a D/A converter has an accuracy of 0.1% and the full-scale output voltage is 10V, then the ouput for any digital input should not vary more than 10V from its expected value for that particular digital input.

Resolution

Resolution really means the number of binary bits of input. It is common, for D/A converters to have 8-,10-, or 12-bit inputs. This resolving ability can also be expressed as a percentage of the full-scale output. For example, an 8-bit converter has a resolution of one part in 256 (2^8), which can be expressed as 1/256 of the output or 0.39% of the full-scale output. Inasmuch as the resolution is determined by the number of bits, the accuracy specification usually tells us much more than resolution.

Linearity

We have seen that the actual output of the D/A converter is a staircase, rather than a straight line. However, it is common to express the output characteristic as a straight line drawn through points of the staircase, as shown in Fig. 9-16.

Linearity is a measure of how much a line drawn through the same points would deviate from a straight line. This is shown in Fig. 9-17. If the linearity is stated as 0.1%, then the actual curve drawn through these points should not deviate from a straight line more than 0.1% of the full-scale output voltage. For example, in the figure with 0.1% linearity, the curve must not deviate from the expected straight line more than 20mV.

Scale Factor

The scale factor is also known as the gain error and it is the factor most responsible for the actual output of a D/A converter deviating from the expected output. It is affected by changes in the values of resistances and by any changes in the reference voltage.

Offset

An offset is a fixed voltage which is added to or subtracted from the output voltage. Offset is usually caused by faults in the op-amp. It can be measured rather simply by setting all of the digital inputs to zero and measuring the output voltage. Under ideal conditions the output voltage should be zero. Any deviation from zero is the offset voltage.

ANALOG-TO-DIGITAL (A/D) CONVERTERS

In order to get real world signals into a digital system we must convert the analog voltages to digital signals. This is done by means of an A/D converter. The simplest type of A/D converter is shown in Fig. 9-18. It consists mostly of components with which we are already familiar. It contains a voltage comparator which we

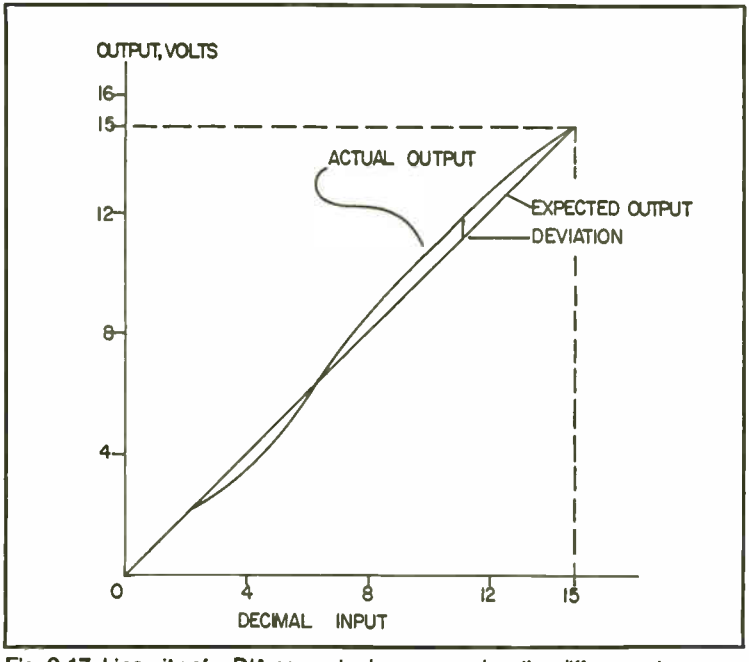


Fig. 9-17. Linearity of a D/A converter is measured as the difference between the expected linear output (straight line) and the actual output (curved line).

studied in Chapter 8, as well as an AND function and a binary counter. There is one component which we have not looked at previously, and that is the ramp generator.

The purpose of the ramp generator is to develop a voltage that increases linearly with time. This is accomplished by a circuit that is designed to charge a capacitor with a constant current. When a constant current flows into a capacitor, the voltage across the capacitor will increase linearly with time.

Of course, no constant-current generator is perfect, because a perfect constant-current generator would eventually produce an infinite voltage across the capacitor. However, over limited ranges current generators can be built good enough so that the ramp voltage will be as linear as we want it to be. In Fig. 9-19 we have shown an ideal ramp where the voltage increases at a rate of 1V per millisecond.

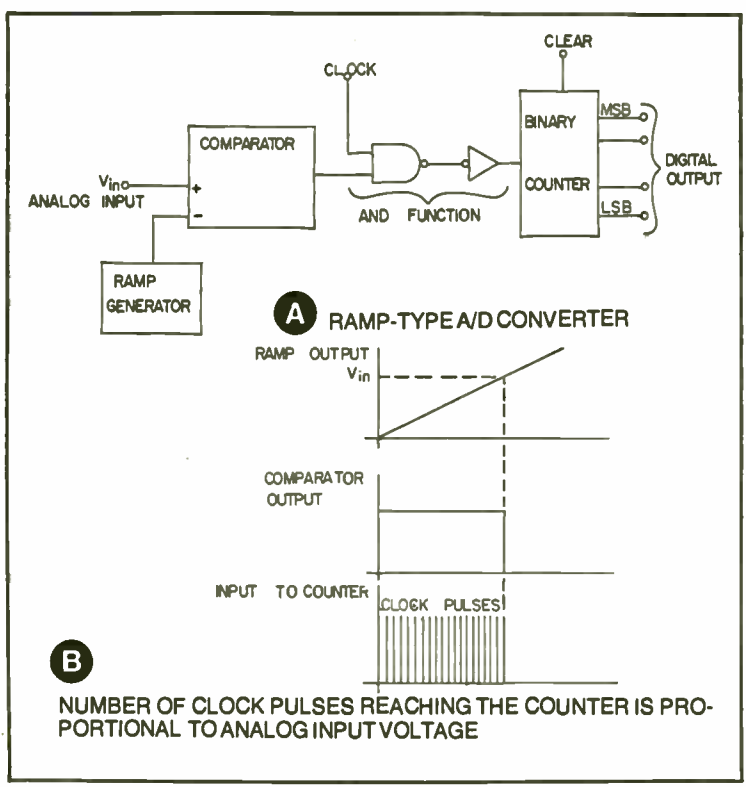


Fig. 9-18. Block diagram and input/output characteristic curves for a simple A/D converter.

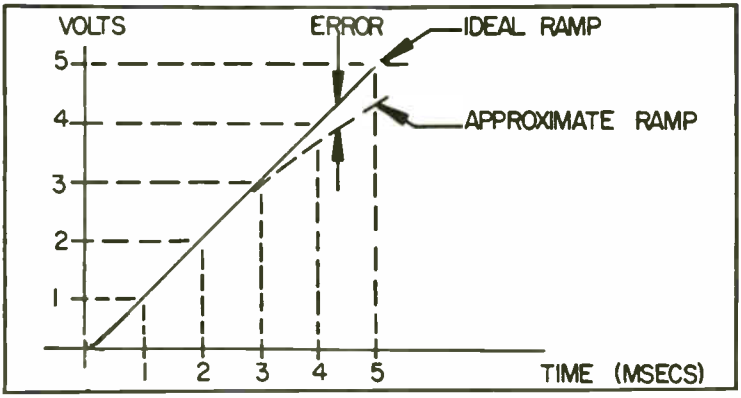


Fig. 9-19. In an ideal linear ramp, the ramp voltage is directly proportional to time.

The important point about the ideal ramp is that if we're able to measure accurately the time from when the ramp started to a particular time, then we would know the voltage. For example, in Fig. 9-19, if we measure the time from the start of the ramp until 4 milliseconds later, we would know that the ramp voltage was exactly 4 volts. This is important, because it is very easy to measure time with a high degree of accuracy with digital curcuits.

Now, we can get back to our simple A-to-D converter. The operation of the circuit is rather simple. The first thing that happens is that a clear command resets the binary counter to zero. And at the same time, the output of the ramp generator is applied to the comparator. The analog signal that we wish to convert is applied to the other input of the comparator. Since the ramp voltage is initially much less than the voltage we want to measure, the output of the comparator is high. Now, the output of the AND gate is also high. As long as the output of the comparator is high, one input of the AND gate will be high and the output will go high whenever a clock pulse occurs.

While the output of the comparator is high, clock pulses will go to the counter and will be counted. The instant that the output of the ramp generator even slightly exceeds the analog input voltage, the comparator will switch states and its output will go low. This will inhibit the transmission of the clock pulses, so that counter will stop counting at the exact instant when the ramp voltage is equal to the analog input voltage. Thus, the number of clock pulses that where counted will be proportional to the input analog voltage.

For example, suppose we had a clock frequency of 200 kHz; that is, the clock produces 200 pulses per millisecond. If we apply

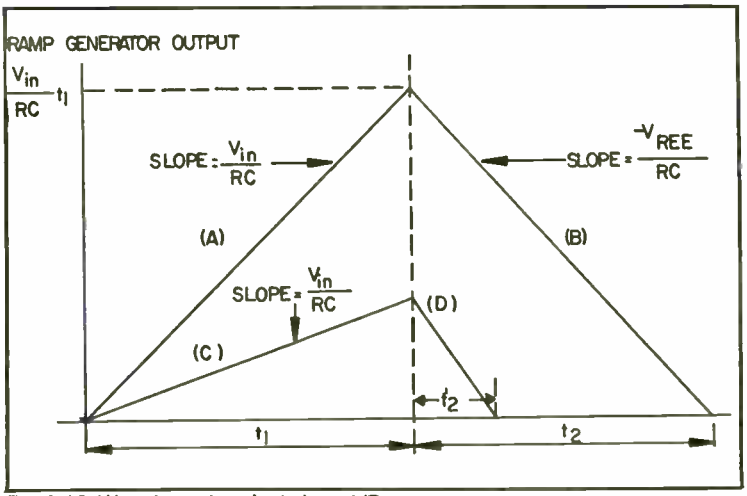


Fig. 9-20. Waveforms in a dual-slope A/D converter.

an input voltage of 2 volts, the counter would read to the binary equivalent of 400 during the conversion.

The accuracy of this system is limited to just how linear we can make the ramp waveform and how stable the output of our clock generator will be.

THE DUAL-SLOPE: AN ANALOG-TO-DIGITAL CONVERTER

A more sophisticated A/D converter, which has become very popular in digital voltmeters and is available in integrated circuits, is called the dual-slope A/D converter. Unfortunately, the operation of this particular device isn't very easy to follow. The best way to understand it is first take a look at what it actually does and then later, see how it does it.

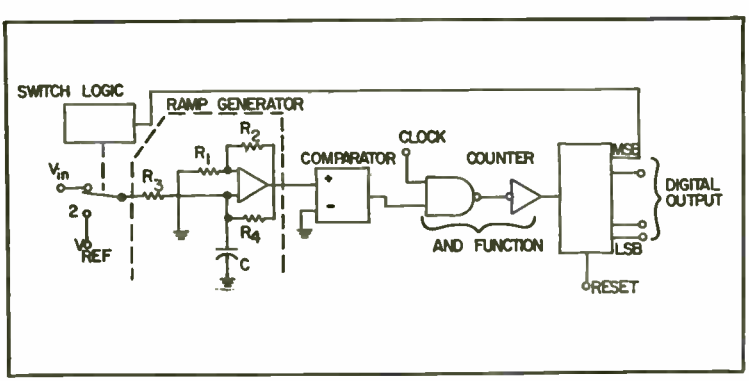


Fig. 9-21. Schematic of a dual-slope A/D converter.

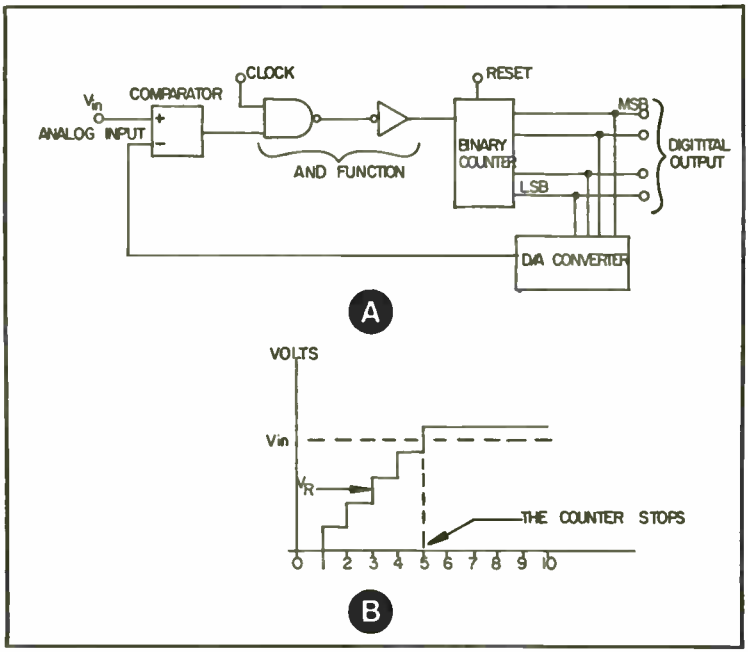


Fig. 9-22. Block diagram of an A/D converter using D/A feedback.

Figure 9-20 shows a plot of the two slopes that are generated inside the converter. At the start, ramp voltage is developed, the slope of which is proportional to the analog input voltage. Then this voltage is brought back to zero by a ramp, which is proportional to a fixed reference voltage inside the converter. The analog voltage that we wish to convert to a digital value is applied to a ramp generator, which produces a linearly increasing voltage, shown at the left of Fig. 9-21. The ramp generator is turned on for a fixed period of time, which is accurately known. Then, the analog voltage is disconnected from the ramp generator and a fixed reference voltage is applied to generate a negative-going ramp, as shown. This negative-going ramp is continued until the ramp voltage reaches zero. The important thing to note in the figure is that the time required to bring the ramp voltage from its maximum value to zero is proportional to the analog input voltage. The fact that the converter uses two separate slopes, each of which count the same clock, shows that long-term variations in the clock frequency will not effect the accuracy of the conversion. For this reason, this particular type of converter has become very popular in small digital voltmeters.

A/D CONVERTERS USING D/A FEEDBACK

One of limitations of the ramp type A/D converters that we have been discussing is that the accuracy is limited by how linear and how stable we can make the ramp generator. There is a way around this problem, and that is to eliminate the ramp generator completely. One way of doing this is shown in Fig. 9-22A.

Here, the analog input signal that we wish to convert is compared to a digital signal. To do this, we first have to convert the digital signal to analog form. As shown in the figure, the output of a binary counter is converted to analog form and applied to one input of a comparator. The signal to be converted is applied to the other input. The output of the comparator is then used to inhibit or enable clock pulses to the binary counter.

The action of the converter is as shown in Fig. 9-22B. Here, when the input voltage is applied, the counter starts. Its output is continually converted to analog form and applied to one input of the comparator. When the output of the counter (converted to analog form) reaches the value of the input voltage, the counter is

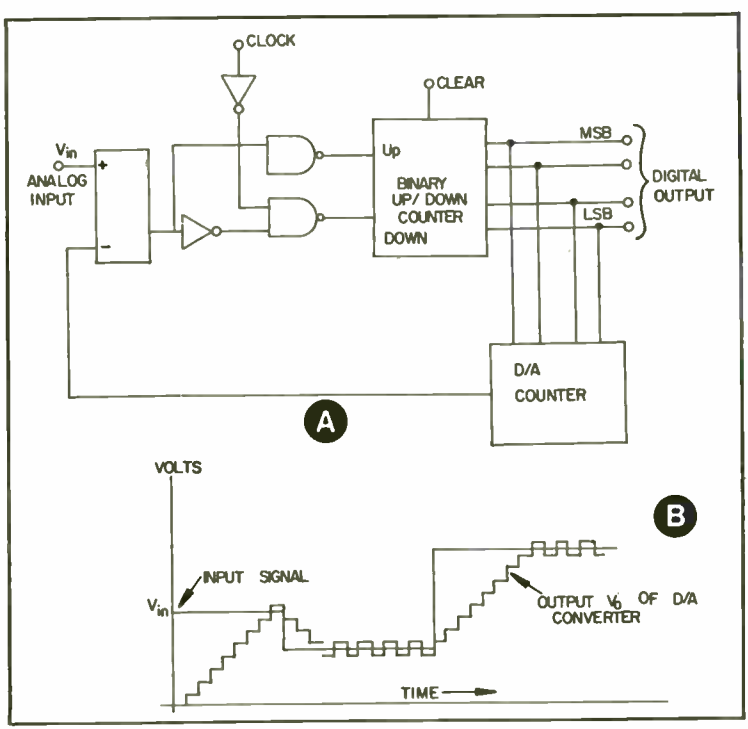


Fig. 9-23. Block diagram of an A/D converter with an up-down counter.

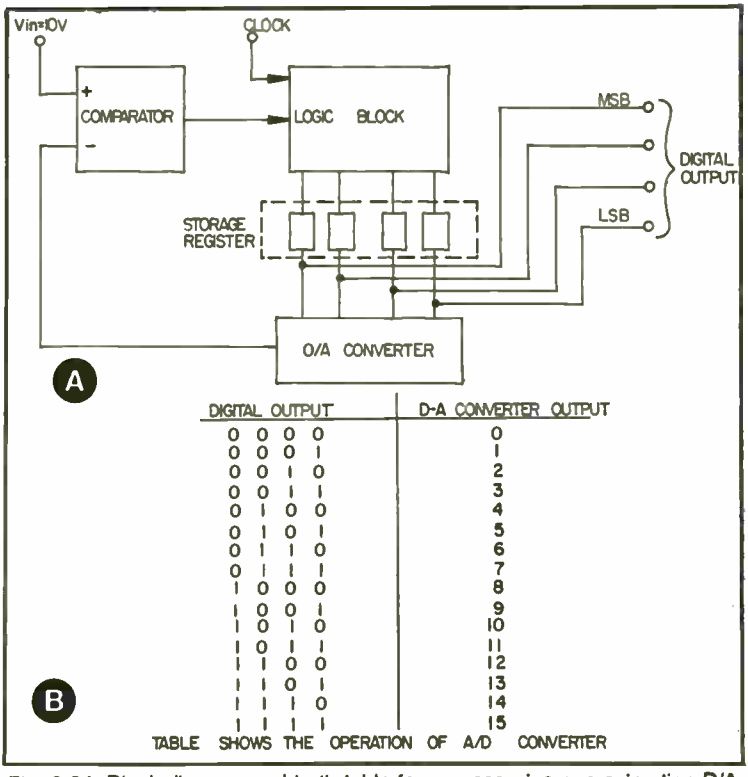


Fig. 9-24. Block diagram and truth table for a successive approximation D/A converter.

stopped. The output of the counter will then be a digital word representing the analog input.

One of the limitations of this conversion scheme is that each time a conversion is made, the counter has to start from zero. It is possible to modify the system, as shown in Fig. 9-23A, to make it much faster in operation. In this arrangement, which is called a continuous counter converter, the basic digital counter is an up/down converter, which can count in either direction.

Assume, for the moment that everything is set to zero, and analog voltage is applied to the input. The counter starts counting up until its output, converted to analog form, equals the value of the input signal. The counter then stops. Now, let's suppose that the input voltage decreases in value, by some sort of step. This will be sensed by the circuitry ahead of the counter, and the counter will start counting down until its output again equals the value of the input voltage. In this way, it can be seen that the output of our

counter will oscillate about the value of the input signal and will count in the proper direction whenever the value of the input signal changes.

Obviously, this arrangement is faster than a ramp-type converter, where the counter has to be reset for each conversion. It does, however, have some limitations in that with large changes in the value of the input signal the conversion process is slow because the counter must increment one bit at a time.

In converting fast signals, such as, in television, this type converter is still much too slow. A faster type is the successive approximation A/D converter, described below.

SUCCESSIVE APPROXIMATION A/D CONVERTER

Figure 9-24 shows a successive approximation D/A converter. For convenience we have shown a 4-bit counter. But, of course, a much larger counter can be used to get better resolution. Assume that the input voltage to this converter can vary anywhere between 0 and 15 volts. The output of the D/A converter in the feedback loop can also vary from 0 to 15 volts on one-volt steps.

To see how the thing works, assume that we apply 11 volts to the input. The operation starts with the most significant bit of the counter being set to one. This means that the output of the D/A feedback converter will be eight volts. The comparator senses that this eight volts isn't enough, so the next step is to increment the next most significant bit giving the counter an output of binary 1100 or 12 volts. The converter senses that this is too much, so the second most significant bit is reset to zero, and the third most significant bit is incremented to one, giving us binary 1010 or 10 volts. This isn't enough, so the next most significant bit is set to one, giving us an output of binary 1011 or 11 volts, which is exactly what we want.

The fascinating thing about this converter is that it only takes four steps to match any input voltage. This the successive approximation converter can be made very fast indeed. It is found adequate for use in digitizing television signals.

PARALLEL A-TO-D CONVERTERS

The fastest possible way of converting from analog to digital form is to perform the operation in parallel as shown in Fig. 9-25. Here, we have a separate converter for each bit of the output number. The reference voltage is divided into descrete steps by a resistive voltage divider. Each of these steps is applied to one input of a comparator. The analog signal to be converted is applied

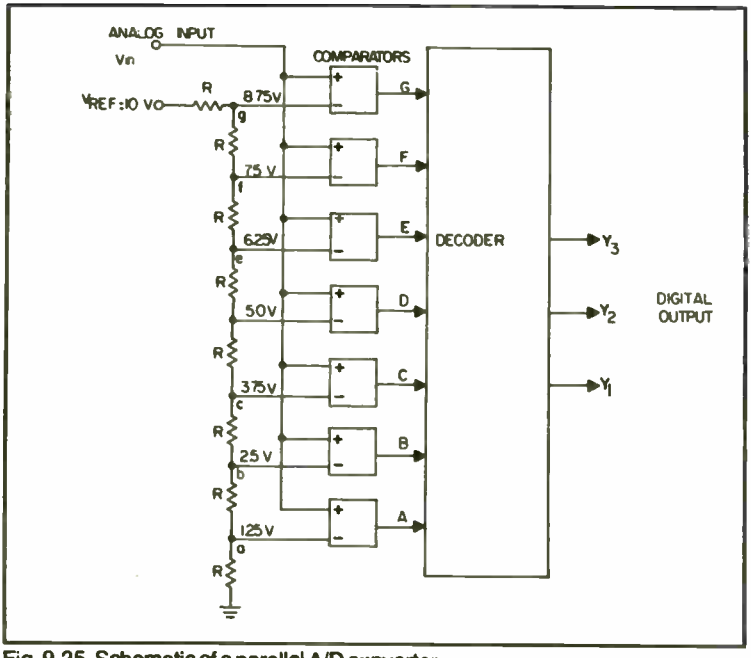


Fig. 9-25. Schematic of a parallel A/D converter.

to the other inputs of the comparator. Thus, an output of a comparator is provided for each level of the analog input voltage. Inasmuch as all the operations are being performed at the same time, the conversion is very fast. The output of the comparators is fed to a decoder network which produces the proper digital word.

Although this converter is extremely fast, it does require a large amount of circuitry. For only three bits at the output, seven comparators are required. In general, for M bits, 2^n-1 levels of comparison are required. This means for a 10-bit output, you would need 1023 comparators. Obviously, this system is used only where extremely high speed is an absolute necessity.

RESOLUTION OF A/D CONVERTERS

One question that comes up in connection with A/D converters is just how well the digital output approximates the true value of the analog input. This, as you probably have suspected, depends upon the number of bits in the output. For a converter with N bits in the output, the resolution is ± one part in 2n. For example, in a 7-bit converter, the resolution would be one part in 128, which is the same as ± 0.8%. If the analog input voltage varied between 0 and 10 volts, the resolution would be $0.008 \times 10 = 80$mv.

Chapter 10
What Is Digital Data And How Is It Handled?

So far, we have discussed the various ways that digital signals can be processed. We have also shown repeatedly how a digital signal can be used to represent a number. The most straightforward way to use a digital signal to represent a number is to use the binary numbering system. If the number is to be converted to and from the decimal system, binary coded decimal is usually a better method.

So much for numbers. What else can we represent by digital signals? Just about anything. An 8-bit number can take on 256 different values. This really means that we can use such a signal to represent 256 different things. All that is required is that we agree on what the signal will mean in a given system. For example, the signal 1011 can mean that the heater voltage of a transmitter is applied, the interlock switches are safely closed, ventilating fans or blowers are operating and the plate voltage has been applied. By means of gating or decoding circuits we can recognize this signal and know that the above stated conditions are true.

In other words, a digital signal can mean just about anything that we want it to mean. Included in the things that we can represent by digital signals are the letters of the alphabet. There is nothing new about this because teletype systems have used on-and-off signals to represent characters for many years, before the word digital found its way into our vocabulary. In fact, teletypewriter keyboards are often used to enter data into digital systems because they are readily available and are competitively priced.

ASCII DATA CODE

In addition to the code used in teletype systems, there are several other different codes that have been developed for various purposes. Actually, the only requirement of any code that enables digital signals to represent other things is that the code be consistent within the system. In order to permit units such as keyboards and printers to be used with many different systems. The American Standards Association has issued a standard code that can be used for all systems.

This code, the American Standard Code for Information Interchange, or ASCII (Ass-key) as it is commonly called, uses a 7-bit word to represent each of the characters of the alphabet, numbers from 0 to 9, and some other things that are called machine commands. The code is shown in Fig. 10-1.

The three most significant bits, bit 5 through bit 7, are used in much the same way that a shift key is used on a typewriter. These bits tell what the next four bits will represent. If the three most significant bits are 000 or 001, the character transmitted will be a machine code. For example, the symbol or digital word 000 1101 is listed in the figure as CR. This is a carriage return command. When it is received by a printer, the carriage of the printer will be returned to the left side of the page. Similarly, if the three most significant bits are 100 or 101, the digital word will represent a capital letter or a punctuation mark.

Thus, a 7-bit word can be used to represent just about anything that might be required on a printer. Inasmuch as the code is a standard, it is possible to buy a digital system from one manufacturer and a printer from another and find that the two will be compatible.

PARITY

You have probably noticed that whereas one byte of data is usually considered to consist of eight bits, the ASCII words are only seven bits long. There is a reason for this. An eighth bit is often added to aid in detecting errors that might occur in a transmission system. This extra bit is called a *parity bit*.

The way the parity bit works is to make all of the digital words contain either an even or an odd number of 1's. In the first case it is called *even parity* and in the other, *odd parity*.

Suppose that we were using an even-parity system and the word "act" was to be transmitted in capital letters. The ASCII codes for these three letters are:

$$100\,0001 = A$$
$$100\,0011 = C$$
$$101\,0100 = T$$

Note that the code for A contains two 1's—an even number. This is what we want, so the 8th parity bit would be 0. The code for C has three 1's—an odd number. To make this word an even number of 1s, we would make the parity bit 1. Similarly, the code for T has an odd number of 1's, so its parity bit would also be 1.

By means of a relatively simple circuit arrangement, the device that receives the data can be made to check the number of 1's in each word. Inasmuch as we have made sure that when the words are transmitted they all have an even number of 1's, we can check to see if any of the bits were "dropped" while the word was being transmitted. This is called a *parity check*. If at the receiving end of the link we find a word with an odd number of 1's, we know that something went wrong. Many systems are arranged so that a word like this will automatically be rejected and a repeat will be requested. This mode of transmission is sometimes called ARQ, for *Automatic Repeat Request*.

In addition to the simple parity described above, many sophisticated codes have been developed that will not only detect errors, but will identify many types of errors and will automatically correct them.

LONG-DISTANCE TRANSMISSION

So far in this book we have considered two ways to get digital signals from one point to another. One way is to use parallel transmission. Here a separate wire is provided for each bit of the byte and all bits of the byte are transmitted at the same time. This arrangement is fast, but it suffers from the fact that a lot of wires are required. It is fine inside a cabinet or between two adjoining cabinets. For longer distances, it is preferable to use serial transmission where the bits are transmitted one after the other and where only two wires are required. As we mentioned in an earlier chapter, shift registers can be used to convert data between serial and parallel form.

When we want to transmit digital signals over distances of more than a few hundred feet, we run into another problem. That is, ordinary wires such as telephone wires are not very well suited to carrying digital signals which have very short rise and fall times. When these signals are transmitted over long lines, the pulse

BIT NUMBERS								0_0	0_0	0_1	0_1	1_0	1_0	1_1	1_1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	COLUMN → / ROW ↓	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	11	VT	ESC	+	;	K	[	k	{
			1	1	0	0	12	FF	FS	,	<	L	\	l	|
			1	1	0	1	13	CR	GS	-	=	M	]	m	}
			1	1	1	0	14	SO	RS	.	>	N	^	n	~
			1	1	1	1	15	SI	US	/	?	O	_	o	DEL

NUL Null, or all zeros
SOH Start of heading
STX Start of text
ETX End of text
EOT End of transmission
ENQ Enquiry
ACK Acknowledge
BEL Bell, or alarm
BS Backspace
HT Horizontal tabulation
LF Line feed
VT Vertical tabulation
FF Form feed
CR Carriage return
SO Shift out
SI Shift in
DLE Data link escape

DC1 Device control 1
DC2 Device control 2
DC3 Device control 3
DC4 Device control 4
NAK Negative acknowledge
SYN Synchronous idle
ETB End of transmission block
CAN Cancel
EM End of medium
SUB Substitute
ESC Escape
FS File separator
GS Group separator
RS Record separator
US Unit separator
SP Space
DEL Delete

Fig. 10-1. American Standard code for Information Interchange (ASCII).

shape will be changed so that the various bits will tend to overlap. The only way that these signals can be transmitted directly on wire lines is to use an inconveniently slow speed of transmission. The solution to the problem is to use a carrier signal just as we do in radio and television.

Figure 10-2 shows the arrangement. The digital signal that is to be transmitted is applied to a modulator where it modulates a carrier in the audio frequency range. At the receiving end of the link, a demodulator is used to recover the original digital signal. Inasmuch as both a modulator and demodulator are usually used at each end of a data communication link, the device is usually called a *MODEM*, from *MOdulator-DEModulator*. MODEMS are commercially available from many different manufacturers. They are specified in terms of how fast they can transmit data and how many channels are provided.

For transmission purposes, digital signals are usually specified in terms of bits per second. Lines and MODEMS are specified in terms of their capacity in a unit called the *baud*. One baud is equal to one unit time interval per second.

DIGITAL MODULATION CONSIDERATIONS

Of course, the audio frequency tone that is used for a carrier for transmitting digital signals can be modulated in any of the conventional methods. That is, either the amplitude, the frequency, or the phase of the tone can be modulated in accordance with the digital signal that is to be transmitted. Because of the noise immunity features of angle modulation, either frequency or phase modulation is usually used in preference to amplitude modulation.

First let's look at frequency modulation. Here, the frequency of the audio tone is varied in accordance with the modulating signal. Inasmuch as the digital signal only has two possible values, 1 and 0, there will only be two frequencies used in transmission. Thus, the system is more like what is commonly called frequency-shift keying or FSK than like regular voice frequency modulation systems.

Figure 10-3 shows one possible FSK arrangement. Here we have two oscillators operating at different frequencies. One frequency can represent a 1 and the other a 0. The incoming digital signal is applied to a switching arrangement so that when the signal is a 1, oscillator A will be connected to the line, and when the signal is a zero, oscillator B will be connected to the line.

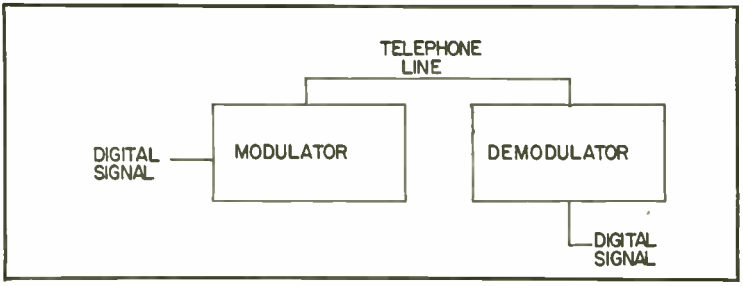


Fig. 10-2. Block diagram of the modulation arrangement used to transmit digital signals over long distances.

The principal limitation of the simple arrangement of Fig. 10-3 is that the transmitted signal switches from one frequency to the other very rapidly when the digital signal changes value. This rapid switching causes something like key clicks in CW systems. It will tend to spread the signal over a rather broad bandwidth. This will limit the number of channels that can be carried on a pair of wires and will also tend to limit the speed at which the data can be transmitted.

Figure 10-4 shows an arrangement that overcomes some of these limitations. Here the digital signal is applied to a waveshaping circuit that removes the sharp corners and slows down the rise and fall times. Doing this reduces the bandwidth required for transmission or permits a higher speed of transmission at the same bandwidth.

Although in voice communication it is sometimes hard to distinguish between frequency and phase modulation, in MODEMS the equipment used to accomplish phase modulation differs significantly from that used for FSK. Phase-shift modulation is becoming very popular in present day MODEMS. Several different arrangements are used, but in each case there is an abrupt change in the phase of the audio carrier when the digital signal changes between a 1 and a 0.

PROBLEMS IN DATA TRANSMISSION

In the average broadcast station we have two different types of data transmission. First there is the transmission over short paths of a few feet from one console to another. For example, sometimes a teletype printer is located a few feet from an automation console. The other type of transmission is over a relatively long path. For example, digital signals may be transmitted from an unmanned transmitter site to a control room miles away. Usually, if the path is

longer than that a few hundred feet, MODEMS are used at each end of the path.

Digital data transmission is complicated somewhat in the broadcast situation because there are often very strong RF fields in the vicinity. The first problem that is usually encountered is to get the thing working in the first place. After this, keeping it going is usually much simpler. Keeping RF out of the transmission path is just like solving any other RFI problem. It is frustrating and time consuming, but usually the problem can be solved. The approach is to carefully go over all grounding and shielding, then begin installing RFI filters as required.

One problem that is often overlooked in broadcast applications is that the digital equipment can be damaged, or errors can be introduced, by strong static discharges. Broadcast stations have tall towers that are plagued by lightning surges. If even a small amount of the discharge gets into the digital system, it will introduce spurious signals. If a large surge enters the system, components will be destroyed.

Again the approach is proper grounding and shielding. Multiple grounds often cause problems because when a high current due to a static charge flows through the ground there will be an appreciable voltage drop. This voltage drop can often get inside the system and raise havoc.

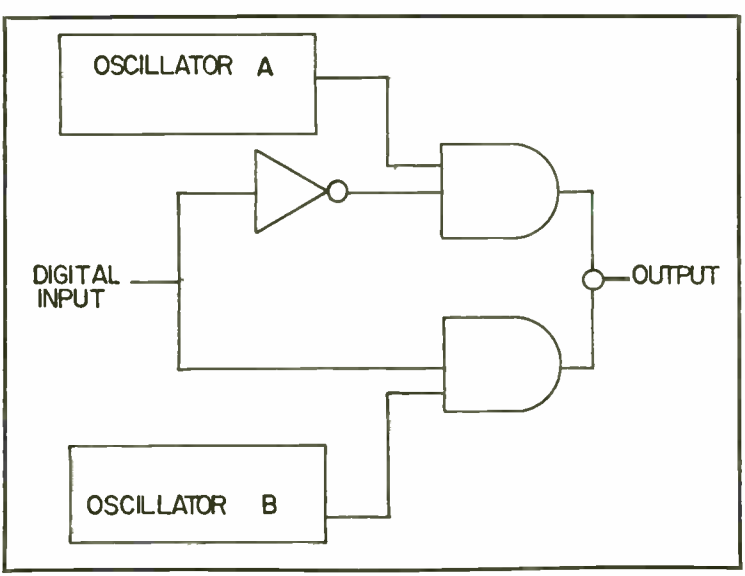


Fig. 10-3. Block diagram of an FSK modulator for digital signals.

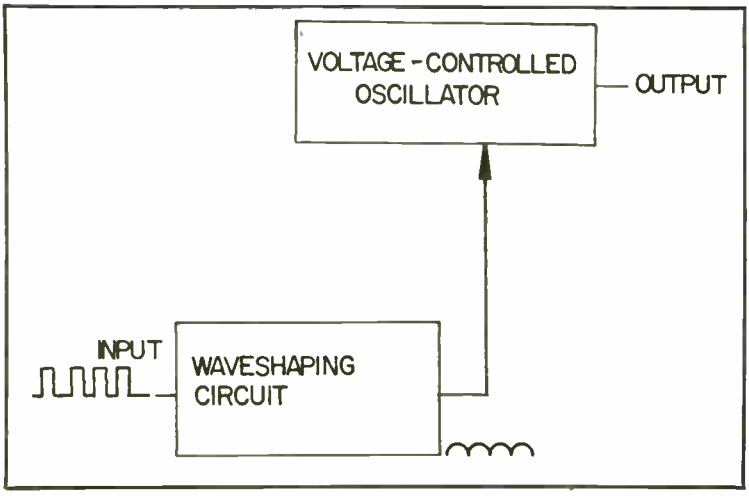


Fig. 10-4. Block diagram of a modulator with waveshaping circuit.

A COMPLETE DATA LINK

Figure 10-5 shows a functional block diagram of a complete data link. In this system certain transmitter parameters are measured, converted to digital form and transmitted back to a control room a few miles away.

The first elements in the system are the input devices. Some of the parameters are simple on-off indications. For example, there is a signal that indicates that the transmitter is on the air. There is also a burglar alarm that will be a logical high signal if anyone enters the premises. In addition, there are analog-to-digital converters. For example, the DC plate current is sampled and is converted into a binary signal that represents the actual value of the plate current.

The signals from the input devices are fed into shift registers where they are temporarily stored until the system is ready to transmit them. A multiplexer sequentially samples each of the storage registers and reads out its signal in serial form. This signal is then applied to an FSK modulator or MODEM.

It isn't enough to merely digitize the data and transmit it. We must have some way of knowing at the receiving end what data is being transmitted. We can accomplish this by assigning a number to each channel of information and transmit this number just ahead of the data. To keep things simple, we can assign the numbers 1 through 7 to the channels of data. We can express the numbers 1 through 7 in digital form with a 3-bit word as follows:

Channel	Number
1	001
2	010
3	011
4	100
5	101
6	110
7	111

In addition, we must transmit some sort of synchronizing signal so that the receiving equipment will know that we are about to send some data. In our system, this is accomplished by transmitting both tones of our FSK signal simultaneously.

At the receiver, a circuit recognizes when both tones are being transmitted and resets everything to zero and gets ready to receive data. Suppose that the first parameter that is transmitted is the value of the plate current, and suppose for the sake of example that it happens to be 250 mA. The word that will be transmitted along the link will be

001 1111 1010

The first three bits identify the fact that information is being transmitted in channel 1. The next eight bits identify the value of the plate current as 250 mA.

At the receiving end, the number of the channel is recognized and the data that follows is fed serially into register 1. The data is stored here and the system moves on to the next channel. Meanwhile, the data that was fed into register 1 serially can be read out in parallel to a decoder which, in turn, operates 7-segment readouts that will display the value of the plate current.

The simple channels that transmit on-off signals to indicate that the transmitter is on the air, or that a burglar may have entered the premises, can be readout on simple lights. It would probably be a good idea to have these signals trigger an audible alarm that would attract attention to the fact that the transmitter is off the air or that there had been an entry.

Although the foregoing paragraphs explain the general operation of a data link, our simple system has many limitations. In practice there could be many more refinements. One obvious limitation is that we used a straight binary number to transmit the value of the plate current. In practice, we don't need to be able to transmit every possible value of plate current from zero to the

maximum possible value. We could transmit just a few numbers in the normal range of operation. This could be handled by assigning a different meaning to the binary numbers that we transmitted and properly decoding them at the receiving end of the link.

In a practical system it might be advisable to transmit our numerical data in the form of ASCII characters. This would require several words to transmit the value of the plate current. Inasmuch as the system will respond much faster than it needs to, we can easily afford to transmit the extra words.

Figure 10-6 shows one channel of a system that would use several digital words to transmit the value of some parameter such as the plate current. Here the plate current is again so sampled and digitized in an A/D converter. The information can be in either binary or BCD form. Probably BCD is easier to handle. Then the signals from the register are applied to an encoder, which may be a single integrated circuit, that converts each BCD number to an ASCII character.

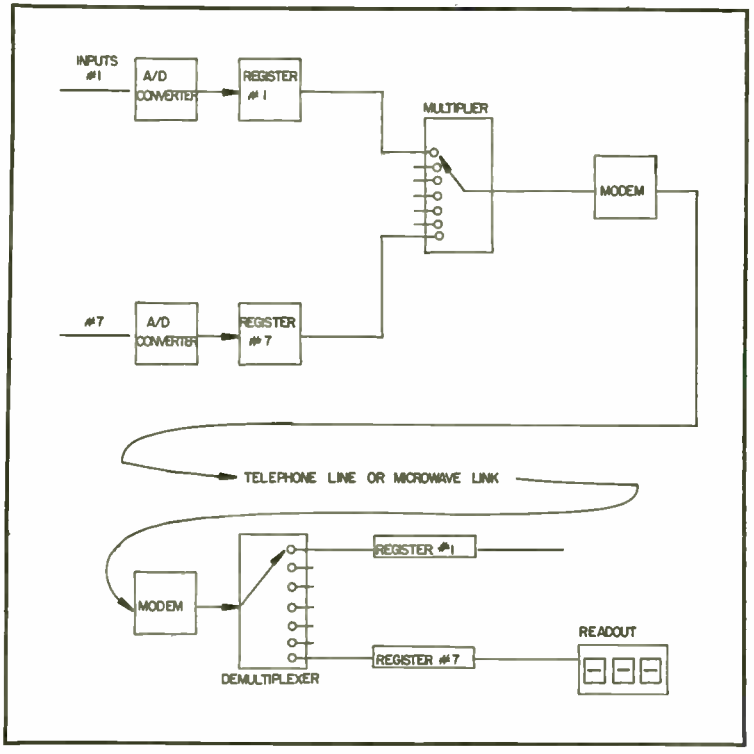


Fig. 10-5. Simplified functional block diagram of a remote metering system.

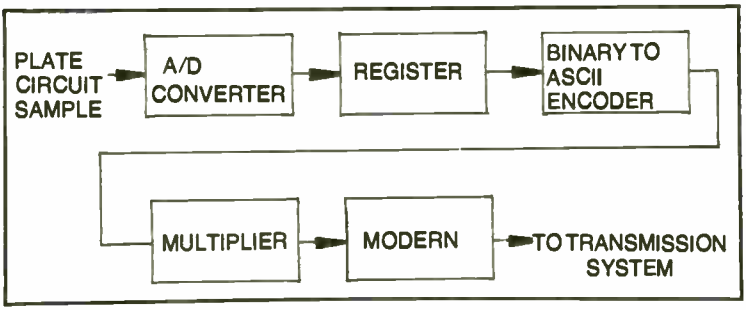


Fig. 10-6. Block of a single channel of a remote metering system using ASCII code.

When data is transmitted, a few synchronizing pulses are transmitted first. The receiving system then knows that some data is coming. It further knows how many words will be used to transmit this particular parameter. Suppose that we provide four characters to send the data and that the value of the plate current is again 250 mA. The decimal, BCD, and ASCII codes are as shown below:

DECIMAL	*BCD*	*ASCII*
0	0000	011-0000
2	0010	011-0010
5	0101	011-0101
0	0000	011-0000

To be sure that the system knows that we have sent the value of one parameter, we will again send a few synchronizing pulses.

One advantage of using ASCII characters for transmission is that the signal may be applied directly to a printer that will recognize each character and print it out. The receiving system can have instructions stored in it that will recognize the channel designation and print it out in English. For example, if the identification 001 is received, the printer will not print out 1, but rather "plate current."

Many more refinements may be added to our system. We can, for example, add a parity bit so that the system can detect certain classes of errors in transmission. We can use ARQ so that if a defective character is received, the signal can be sent from the receiving location back to the transmitter requesting that the data be transmitted again. We can arrange it so that the defective data will not be printed out and an operator watching the system might never even know that an error was transmitted and corrected.

Chapter 11
A Few Simple Digital Systems

In the first part of this book, we concentrated on digital ICs and their functions. We considered both combinational functions using logic gates and sequential functions using flip-flops or counters. In the following chapters we will look at complex digital systems where it would be completely impossible to consider all of the functional units such as gates and flip-flops. Many systems use hundreds or even thousands of these simple functions. To understand the larger systems, we must concern ourselves with more advanced functions such as data storage or manipulation.

This approach, which seems to be the only practical one, tends to leave us with a gap between simple functional units and large subassemblies of a system. For example, we may be confronted with a remote metering system of the type we discussed briefly in the preceding chapter. We might find that there is trouble somewhere between where the transmitter parameters are sampled and the MODEM. Upon opening the cabinet we find several printed circuit boards packed tightly with SSI packages. It is often hard to find a correlation between the overall functional operation of the cabinet and all of those little ICs. The situation isn't made any easier by the fact that there are many different ways that different ICs can be combined to perform the same overall function. The particular arrangement will depend to some extent on the designer's preference.

In this chapter, we are going to take a few very simple tasks and see how we might use digital circuits to accomplish them. Of course, in many applications the systems are much more complex than those that we will consider. Nevertheless the principles are

the same and by going through these examples, you can see the way that you should look at larger systems.

THE SYSTEMS APPROACH

When studying a digital system, it is helpful to think a little bit about how the designer might have approached the task of designing the system, Of course, this task can never be complete, because you have no way of knowing all of the factors that the designer had to take into consideration. Often, when looking at a part of a system, you will find an arrangement that certainly looks like the hard way to solve a rather simple problem. This is often true. Usually, though, there is a good reason for the design being the way that it is.

Any designer must take design costs into consideration. If a system is produced in small quantities, like many items used in broadcasting, the design cost is often a large part of the total expense of producing a system. In such a case, the designer must take every step that he can to minimize the design costs. Often this means that if he has already designed a circuit that works and can be used in a particular application, it is advisable to use it, rather than to take the time to design another circuit that might use fewer components or be more straightforward. Sometimes you can recognize this when you see that a certain manufacturer's products often use the same circuitry in different pieces of equipment.

Getting back to the systems approach, the designer usually approaches the problem first from an overall point of view, then gets to the details. Let's take as an example a very familiar product, the digital electric clock. Of course, this particular product has become so popular that all of the required functions are now available in a single large-scale integrated circuit, but it wasn't always this way. Early electric clocks were made from small-scale integrated circuits. By taking the SSI approach, we can get an idea of how a system might evolve.

Faced with the task of designing an electric clock, we might start with the simple approach of Fig. 11-1. We know that we must have some way of reading out the time, so we have shown 7-segment readouts for hours, minutes and seconds. We also know that we must have some accurate source of time to drive the clock. At this point, we haven't the slightest idea of how we might process the time information, so we have just shown a box.

So far, there is no way of telling at which end of the box the design might start. It might be handled alternately at each end of

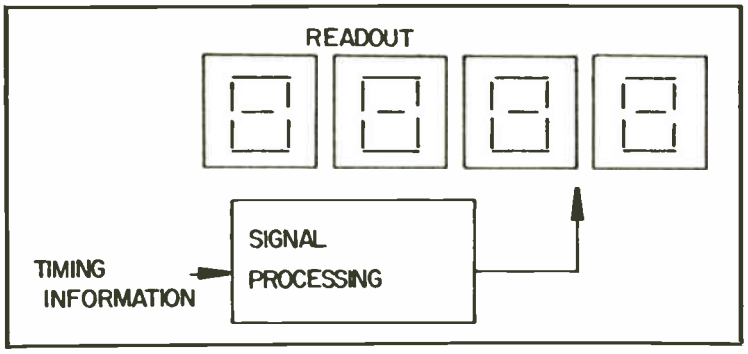


Fig. 11-1. Functional diagram of an electric clock.

the box. For the sake of example, lets assume that we will use the 60-Hz power line as a source of timing information. Incidentally, this is a very accurate source of timing information over a long period of time. The frequency of a power line does indeed vary during the day, but the utilities, knowing that their frequency is used to operate clocks, corrects for this. Every night, usually during the early hours of the morning, utility operators speed up, or slow down the frequency of the system so that any errors in electric clocks operated from the system will be corrected. The result is that the 60-Hz line frequency is one of the most accurate long-term time standards available.

Of course, other sources such as crystal oscillators have greater short-term accuracy, but they always have some error however small. A clock tends to accumulate errors. If a clock runs ever so slightly slow, eventually it will be off by some amount. Usually, at the end of a long period such as a year, the line-operated clock will be more accurate.

Before we even start to use the power line as a time standard, we know that the voltage is too high. Thus, we will start our design with a transformer that will step the voltage down to about 5V so that it will be compatible with TTL. We also know that the line voltage is sinusoidal and that digital circuits don't like slowly varying signals like sine waves. Thus, we will add something to square up the waveform. A latch made out of two gates would do the job. The next thing that we would like to do is to divide the line frequency by 60 to give us a pulse train having a pulse repetition frequency of 1 pulse per second.

With these considerations in mind, we arrive at the block diagram of Fig. 11-2. Now let's go back to the other end of the system. Here we have six 7-segment readouts for hours, minutes

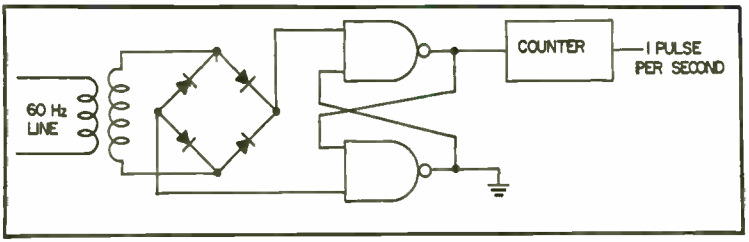


Fig. 11-2. Time-base circuit for an electric clock.

and seconds. Our design philosophy will be to count our one-pulse-per-second signal from our time base, and then decode the output of the timer counter to drive our readouts. The approach is shown functionally in Fig. 11-3.

The decoding operation is quite simple. The Type 7447 IC will accept a BCD signal and will drive a 7-segment readout to display the decimal equivalent of the BCD input. The only problem remaining is the counting.

The Type 7490 counter will divide by either 10 or 12, depending on how it is connected. We can apply the one-pulse-per-second pulse to one of these counters as shown in Fig. 11-4A. Thus, the counter will count up to 10 seconds, display the count on the readout, and then reset and supply a count to the next counter which counts ten of seconds.

The tens of seconds counter can't be the same type as the one that we just used, because we want it to count to six, corresponding to 60 seconds and then reset. A Type 7492 counter will do the job, as shown in Fig. 11-4B.

By following this general line of reasoning, we will eventually arrive at the digital clock functional diagram of Fig. 11-5. We have in a general way taken a requirement and found a functional approach to a digital system that will meet the requirement.

SPECIAL DECODING

There are applications where we have the output of a counter which we wish to decode for some special purpose. This type of

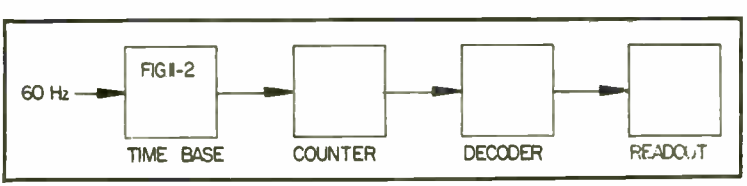


Fig. 11-3. Block diagram showing further development of electric clock.

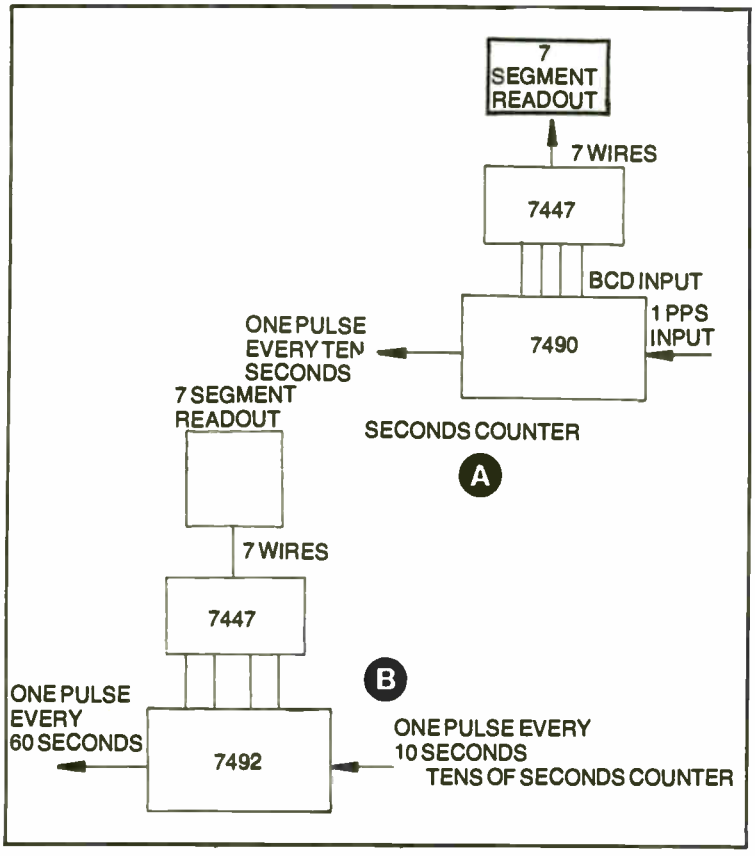


Fig. 11-4. Block diagram of the counters and decoders for an electric clock.

decoding can be accomplished by the use of gates and inverters. Suppose, for example, that we wish to get one output pulse whenever the output of a BCD counter reaches the decimal count of nine (1001 in binary or BCD). We can accomplish this by using the arrangement of Fig. 11-6. Here, when the input to the gates and inverters is 1001, the two inverters in the two middle leads will make the signal at the input to the gate 1111. This, in turn, drives the output of the NAND gate to a low level. An inverter following the gate will produce a high level. Thus, the output of the decoder will go high whenever the input is 1001.

Special decoding arrangements much more complex than this are often used for special purposes in digital systems. Once the general function is recognized, the circuit can be understood by constructing a truth table.

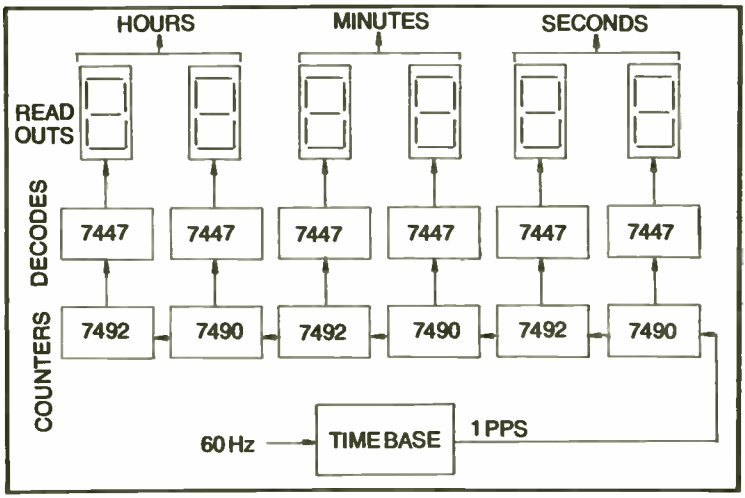


Fig. 11-5. Block diagram of the complete electric clock.

FREQUENCY COUNTER

The frequency counter shown in the block diagram of Fig. 11-7 might be a good example of a case where a designer used circuits that were developed for another purpose. Suppose the designer had developed and tested the digital clock that we discussed in preceding paragraphs and was now called on to design a simple frequency counter. The approach would surely be to use the same type of readout and decoder. The frequency counter differs from a digital clock in that we count a pulse train whose frequency is unknown. The accuracy of the system comes from accurately controlling the time interval during which the count is made.

In the diagram of Fig. 11-7, the display and decoder arrangement is similar to that used in the clock. We can't, however, drive

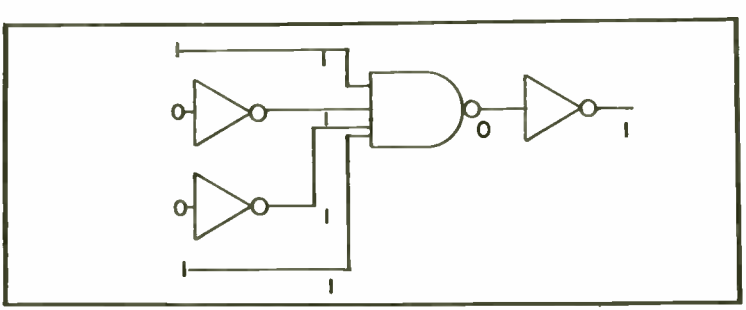


Fig. 11-6. Decoding arrangement to provide a pulse at the count of 9 (1001 in binary)

the decoder directly from the outputs of our counters. The reason is that the frequency that we may be counting might be quite high. If the outputs of the counters were applied directly to the decoders and displays, the displays would be changing very rapidly with the result that the readout would look like a blur instead of discrete numbers. We can correct this situation by inserting latches between the counter outputs and the decoders. These latches will act as memories and will store the count at the end of a counting interval long enough for a reading to be taken. They will then be reset and will store another count.

The counting chain will be slightly different in that we will use decade counters for all stages. The rest of the circuit is also straightforward. The timing interval is obtained by dividing down a signal from a time base.

Here, we are not as much interested in the long-term accuracy of the time base as in the short-term accuracy. We will measure the frequency in question over a comparatively short period such as one second at the most. Suppose for example that we applied a signal of unknown frequency to the counter, counted it for one second and found the count to be 99,950. We would then know that the unknown frequency was 99,950 Hz. If there were an error in the interval over which we did the counting, there would be the same percentage error in our frequency measurement.

For this reason, we will use a crystal oscillator as a time standard rather than the power line. Both this signal and the unknown signal are applied to a logic gate that allows the unknown signal to be counted during the counting interval. The signal processing in the chain is to "square up" the unknown signal so that it will be compatible with out TTL logic and to adjust its level.

UNDERSTANDING A DIGITAL SYSTEM

The preceding paragraphs show in a general way the process of solving a problem by combining various digital functions. When one is confronted with a digital system that has already been designed and wants to understand it, the process is reversed and in general is less straightforward. Faced with a complicated diagram containing hundreds of ICs, it isn't easy to see the function of each one. Often, the best approach is to start with the inputs and outputs and trace from each end until a functional picture of the entire system can be constructed.

Suppose, for example, that the frequency counter of Fig. 11-7 were simply a part of a larger system and we had no clue to its function. We could start at the readouts. We know their function.

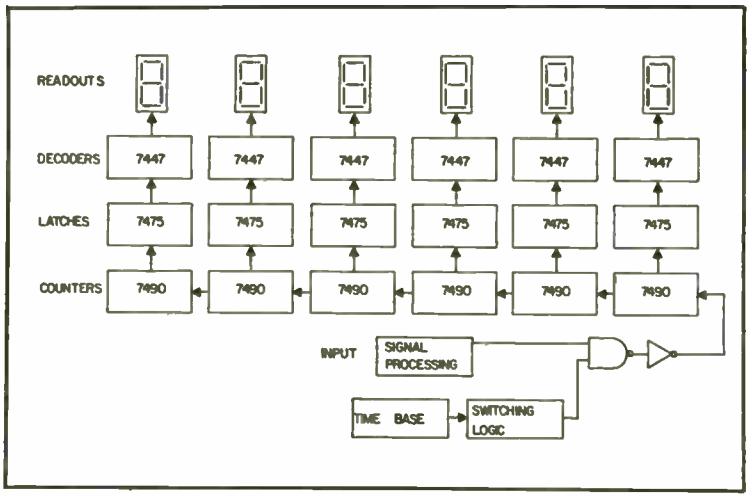


Fig. 11-7. Block diagram of a frequency counter.

They display decimal numbers in response to signals that cause the various segments to light up. The decoders are almost as easy to understand. Checking the data book, we find that the inputs to these decoders are BCD signals.

The next step isn't so easy. When we come to the 7475 latches, we know that their function is to latch some digital signals and hold them. We might not have the slightest idea of why they might need to be stored. There are two possible reasons for using the latches. One is that the signal which is to be displayed on the readout isn't available all the time. It might be a signal that occurs occasionally and is latched so that it can be read out on the display, or it might be a signal that is usually changing so fast that it can't be read on the display. With a little more investigation we can find that the latter is true in this case.

The counter chain will probably be the easiest to understand. Counters are straightforward. By checking the data sheet we can see what the counter is supposed to do, and with that information we can usually figure out what the designer intended. There is one potential problem with counters in that some counter ICs can be connected to count in different ways. For example, a certain counter may divide by 6, 10, or 12, depending on how it is connected. To be sure of just what the counter is doing, we must carefully check the connections.

In reading the block diagrams of digital systems, it is common to encounter an arrangement that doesn't make any sense at all at

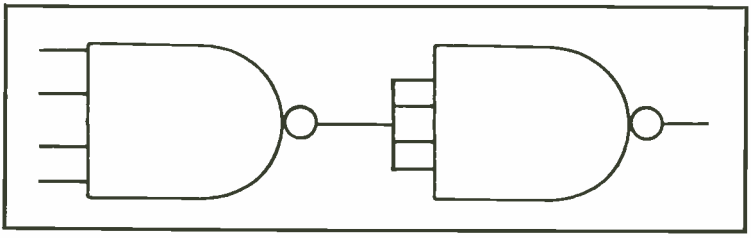


Fig. 11-8. With this arrangement, half of an integrated circuit is used as an inverter.

first glance. For example, one might find the arrangement shown in Fig. 11-8. Here, the first stage is obviously a 4-input NAND gate. The second stage isn't as clear, however, because although the function is also a 4-input NAND gate, all four of the inputs are tied together. A little thought reveals that when a 4-input NAND gate is connected in this way, it functions as a simple inverter. If we construct a truth table of the whole arrangement, we will find that it operates as a 4-input NAND gate.

The first question that might arise is why the designer didn't use an inverter in place of the second NAND gate. The reason, of course, is that there are six inverters in the usual package and he may need only one. Futhermore, there are two 4-input NAND gates in a package such as the TTL Type 7420. There is still the question as to why he didn't use half of the Type 7421, which contains two 4-input AND gates. Perhaps, in many applications, he could indeed have used the Type 7421. On the other hand, if the output of the circuit of Fig. 11-8 were used to reset a flip-flop or counter, the additional propagation time of the second gate might be needed to assure proper operation of the overall circuit. In this case, using the Type 7420 with one half operating as a 4-input NAND gate and the other half operating as an inverter might be the optimum choice.

Thus, by carefully considering the various possibilities, one can usually at least figure out how the system is supposed to work. Sometimes, indeed, the design isn't optimum. For example, if one type of IC is used in very large quantities, it might be advisable to use it in places where it would not normally be used just because of the cost savings. This, of course, is something that one not involved in the design would not know.

Chapter 12
Digital Video

One aspect of broadcasting where digital technology is having an ever-increasing effect is in television. It seems that many of the things that we have always wanted to do with TV signals have been impossible or at least impractical with analog techniques. Such things include time-base correction, generation of special effects, elaborate graphics, reduction of noise, and the conversion from one transmission standard to another. These things are not only practical, but in many cases rather easy with digital techniques.

The pioneering work on digital television was done by Bell Telephone Laboratories in connection with the development of a picture phone. This work was expanded considerably in connection with the space program, with the result that most of the required technical knowledge is readily available.

The application of digital technology to television is limited primarily by the cost and availability of components. The first commercially available digital television device was a time-base corrector introduced in 1973. Since then, many more digital television devices have become available and more can be expected on a regular basis.

Many of the digital television devices available today are organized in accordance with the block diagram of Fig. 12-1. At the input of the system an analog video signal is converted to digital form and at the output a digital signal is converted back to analog form. The reason is, simply, that TV cameras are analog devices, and present transmission standards require that the TV transmitter be modulated with an analog signal. In spite of the fact that a TV system must start and end with an analog signal, it is well

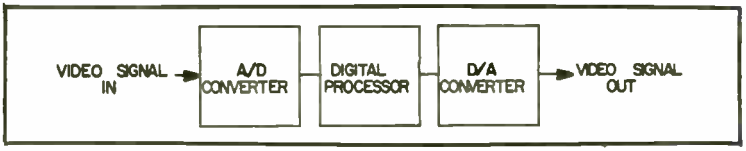


Fig. 12-1. Block diagram of a typical digital TV signal processor.

worthwhile to convert the signal to digital form because of the many useful things that we can do with digital techniques.

Most of the advantages of digital processing stem from the facts that digital signals can be stored easily, and that digital signals can be completely recovered after they have been corrupted with noise.

Inasmuch as the input and output stages of many digital video signal processors are A/D and D/A converters, they can be installed somewhere between the camera and the modulator, and the fact that they are digital will not be apparent to the operator. Devices of this type are sometimes said to be "transparent" to the video signal.

The A/D converter at the input of a digital processor is often called a *coder* and the D/A converter at the output is called a *decoder*. Frequently, the combination of the two is called a *codec*. Before we look at what a digital system can do with a video signal, lets look in more detail at the coder and decoder.

THE CODER

The coder, or device which is used to convert an analog video signal into digital form, performs three functional operations:

1. Sampling. Here a sample is taken of the signal and is held while the conversion process is taking place.

2. Quantizing, which is adjusting the signal to specific levels.

3. The coding, or actual analog-to-digital conversion. These three functions are shown in Fig. 12-2.

The first question that arises in connection with digitizing any analog signal is how frequently we must take samples in order not to lose any of the detail in the original signal. In an earlier chapter, we mentioned the so-called "Nyquist Criterion," which tells us

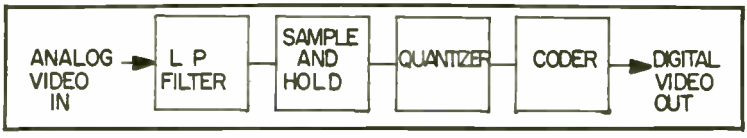


Fig. 12-2. Stages involved in typical video analog-to-digital conversion.

that in order not to lose any of the information in a signal we must take samples at twice the highest frequency present in the signal. Let's take another look at this and see if we can make a little more sense out of it.

One way to look at sampling is to consider it to be amplitude modulation. The sampling signal is the carrier and the signal being sampled is the modulating signal. From what we know about amplitude modulation, we will expect two sidebands to be produced, one at a higher frequency than that of the carrier and one at a lower frequency.

Figure 12-3A shows a regular video signal in the video frequency domain. This figure shows that all of the information in the signal lies between zero frequency and some high frequency, which we will call f_0. Now let's use this signal to amplitude-modulate some sampling signal which has a frequency f_s. Figure 12-3B shows the original signal, called the baseband signal, which occupies that part of the spectrum between O and f_0, the sampling signal, f_s, and the upper and lower sidebands. Each of the sidebands has a bandwidth equal to f_0. Looking at the figure we can see that if we don't want the lower sideband to fall on top of the original signal, the lowest frequency that we can use for the sampling signal must be equal to twice f_0. This gives us another way of looking at the Nyquist criterion.

Taking another look at Figs. 12-3A and B, we note that we have shown the signal as occupying a rectangular piece of the frequency spectrum. This, of course, is an ideal situation which we will not find in practice. In the real world, the frequency domain representation of a signal will start to fall off at some frequency, but there will be still higher frequency components that will extend beyond where the signal starts to fall off. Thus, the spectrum of a real-world signal will look like that shown at the left of Fig. 12-3C and the sidebands of the modulated signal will occupy a spectrum like that shown farther to the right. From this, we can see that in the real world it would be a good idea to use a sampling frequency that is much higher than $2f_0$.

In practice, television sampling is usually done at either three or four times the frequency of the 3.58-mHz color subcarrier. In addition, the higher frequency components of the original signal are attenuated with a low-pass filter as shown in Fig. 12-2.

Now that the sampling frequency is out of the way, we must take a look at the number of levels into which we will quantize our signal. Here we encounter the first difference between analog and

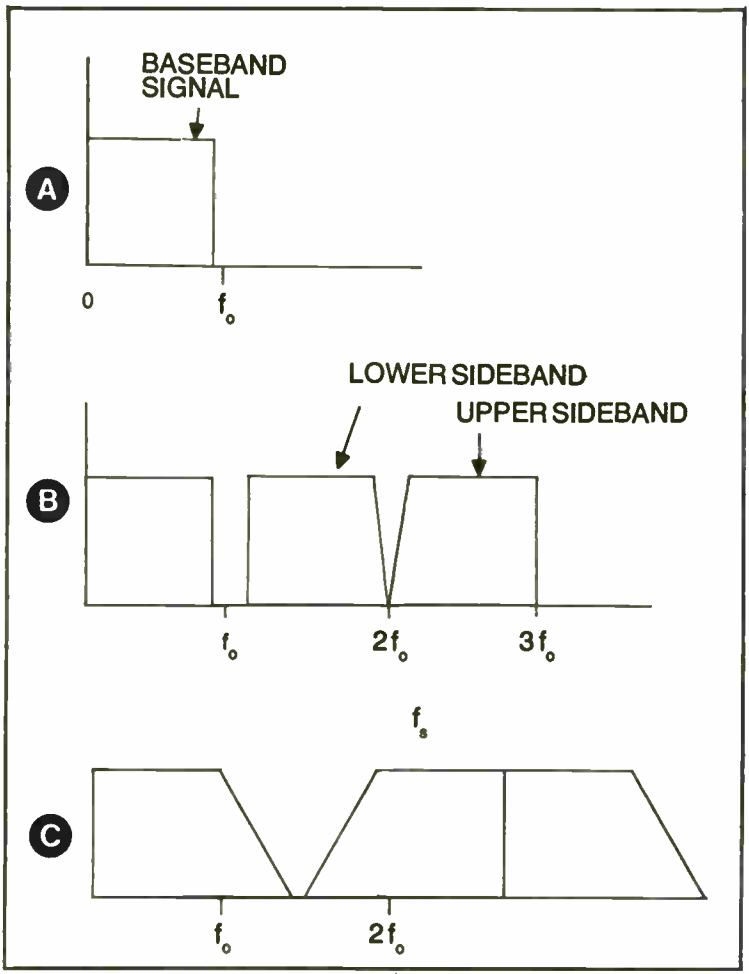


Fig. 12-3. Practical video signal sampling considerations regarding bandwidth.

digital signals. Whereas the analog signal can, at least in theory, assume an infinite number of values, the digital signal can have only a discrete number of values, depending on how many bits we use to represent each sample. A 4-bit digital signal can represent 16 different values if we count zero as one of the values. The effect of quantizing is shown in Fig. 12-4. Here we have allowed our signal to take on four different values regardless of the number of samples that we might take. In the figure we have shown a typical analog signal that varies smoothly from one value to another. If at the moment that we take our sample the value of the signal lies

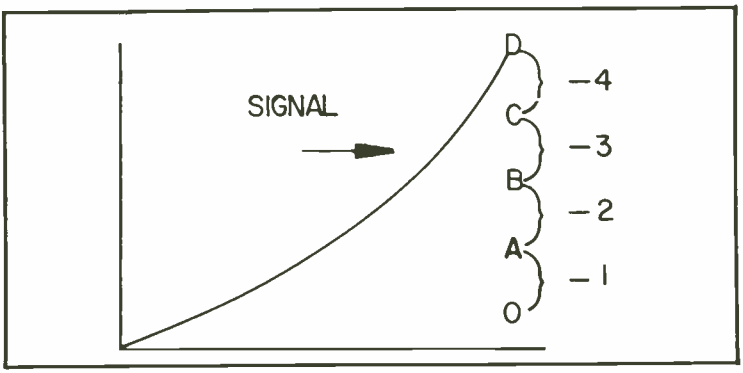


Fig. 12-4. Typical method of quantizing video signal levels.

between O and A, we will let it take on the value 1. If it lies between A and B, it will take the value 2. If it is within B and C, it will take the value 3, and if it is between C and D it will take the value 4.

The same information is also given in slightly different form in Fig. 12-5. Here we see that if during the sampling interval the signal being sampled varies from one value to another, the quantized signal will have a value somewhere between the two extremes.

Naturally, if the brightness of a signal on a TV screen could have only four possible values, the picture would be far from satisfactory. We know that we must have more than four quantizing levels, but how many more?

An insufficient number of quantizing levels can distort the signal in two different ways. The first type of distortion that we will consider is called contour distortion. This is caused when a slight variation in the brightness of a part of a scene results in the signal jumping from one quantizing level to another. This is very noticeable when only a few quantizing levels are used.

Another form of distortion associated with quantizing is called *quantizing noise*. This is due to the fact that the elements of a quantized picture do not have their exact original value. As the number of quantizing levels is increased, this effect is less noticeable, but tends to occur on a random basis in different parts of the picture, thus the effect resembles noise in the picture.

There is another consideration in selecting the number of quantizing levels. That is, with the advent of many different types of digital processing units, each of which has its own codec, a signal may pass through several conversions between the time that it

leaves the cameras and is finally broadcast. If there aren't enough quantizing levels, the effect of several codecs in tandem can add up, causing a considerable amount of distortion.

Inasmuch as the quantized signal will be converted into a digital signal, the number of quantizing levels selected will usually be some integral power of two. Experience has shown that if the signal is quantized into 256 different levels, it will be practically indistinguishable from a regular analog signal. Furthermore, up to seven codecs can be connected in tandem without any noticeable effect of the quality of the signal.

The number 256 is equal to the eighth power of two, so the signal can be represented by an 8-bit digital word. Thus, each sample of the signal will result in one byte of data.

An important consideration in the transmission of any signal, digital or analog, is the frequency range or bandwidth of the signal. If in a codec we sampled a signal at the rate of three times the color subcarrier frequency, we would sample at a rate of 10.7 mHz. Each sample of the signal results in eight bits, so we would have a data rate of $8 \times 10.7 = 85.6$ million bits per second. In addition to the data bits representing the actual signal, we must add some other bits for synchronizing and possibly parity checking. So the actual bit rate would be higher than this. There is a generally accepted rule of thumb that to transmit a 525-line, 60 field-per-second picture a data rate of 100,000,000 bits per second is required. If, in the worst case, the data consisted of alternate 1s and 0s, the corresponding frequency would be 50 mHz.

Obviously, one of the prices we pay for having a video signal in digital form is that the bandwidth required for tansmission is much greater than with the same signal in analog form. One thing that can

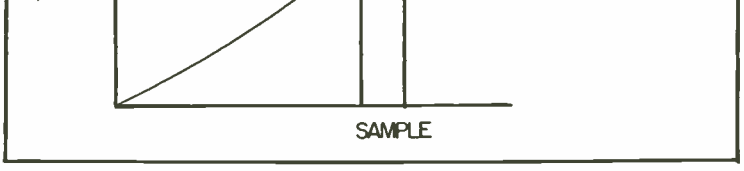


Fig. 12-5. Quantizing errors.

be done to keep the bandwidth within reasonable limits is to keep the data in parallel form inside the processing unit. This is practical, because the paths within the unit are relatively short.

Figure 12-6 shows a parallel system with eight bits. Here, although all of the data may be processed at the rate of 100 megabits per second, the bandwidth on any one of the eight lines is only about half the sampling frequency. Thus, with sampling at three times the color subcarrier frequency, the bandwidth on each line is about 5 mHz. If we sample at four times the color subcarrier frequency, the bandwidth will be about 7 mHz. In practical terms, this means that video digital processing units can use Schottky TTL logic.

The coder is actually an analog-to-digital converter. Because of the speed required, a successive approximation converter of the type described in an earlier chapter is used. This type of converter is fast enough to keep up with the digital signal.

BANDWIDTH REDUCTION

One of the areas of digital signal processing where a great deal of work is being done is that of bandwidth reduction. If the rules are ever changed to the point where we can actually transmit digitally modulated TV signals, it is imperative that the bandwidth be reduced.

Another area where bandwidth is critical is in magnetic recording of TV signals. Although the state of the art is continually being improved, the limitations of the magnetic media tend to limit the bandwidth of the recording. Still another area where bandwidth is critical is in the distribution of TV signals by landlines and satellite systems.

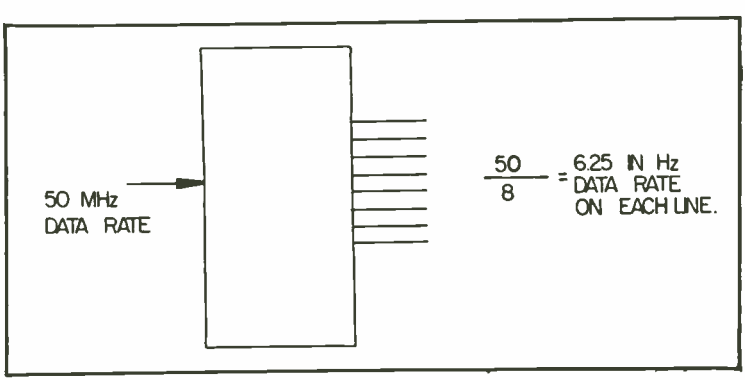


Fig. 12-6. A narrower bandwidth is needed with parallel transmission.

The most obvious way of reducing the bandwidth of a digital TV signal is to not digitize the blanking intervals. The horizontal and vertical blanking intervals in a regular picture contain no information and they can easily be reconstructed later. By leaving them out, we can reduce the bit rate by as much as 20%.

Another technique that is often used in digital systems to reduce bandwidth is called "differential modulation," or differential pulse code modulation DPCM . This system works on the basis that a great deal of the information in the picture doesn't change very much from one picture element to the next, from one line to another. The system is based on the principle of merely transmitting the differences in signal level or color, rather than transmitting all of the information in each picture element. Problems connected with this scheme are that it is not easy to predict when one picture element will change drastically from another.

Another method of bandwidth reduction is sampling at below the Nyquist rate. The way this is done is exactly the same way that the NTSC color subcarrier is interleaved with illuminate signal. The reason we have to sample at at least $2F_0$ is so that the lower subcarrier that results from that sampling will not overlap the baseband signal. If a way is found to interleave the information is the lower subcarrier with the baseband carrier, the two can in fact overlap without causing distortion.

There are still other methods of processing signals, so that when digitized the spectral components are interleaved or compressed. The result is that a smaller bit rate is required for adequate sampling.

The area of bandwidth reduction by digital techniques is one that we can expect to see continual advancement. An information-theory analysis of the TV signal shows that we are transmitting much more information than is really required to actually represent the signal. The amount of bandwidth to be used in the future to transmit TV signals will depend greatly on developments in the state of the art of bandwidth compression.

The decoder of our codec is really a digital-to-analog converter. It's function is just the inverse of the A-to-D converter at the input of the processor.

TIME-BASE CORRECTION

Modern TV programming is based on the use of portable, inexpensive tape recorders. Electronic news gathering or electonic journalism would be impossible without recorders that could

be carried to the scene of the action and operated under all sorts of environmental conditions.

The main obstacle in the use of such recorders is the presence of what are commonly called time-base errors. There are three general classes of these errors.

- ☐ The sync pulses, vertical or horizontal, or color subcarrier may become displaced in phase.
- ☐ There may be actual discontinuities in these sync pulses.
- ☐ The frequencies of the sync pulses can be changed.

Some of these errors can be attributed to limitations of the magnetic tape itself. The tolerances and manufacturing processes are such that the tape may stretch or distort when operated under adverse environmental conditions. Other errors may be traced to the recorders. As the size and weight of the recorders are reduced, there is a greater probability of error. The situation is complicated by the fact that portable recorders are forced to operate under all sorts of adverse conditions.

The use of such recorders has been made possible by the introduction of the time-base corrector. The early time-base correctors were analog in nature. They effectively inserted varying amounts of delay into the system so that the recorded signal being played back could be locked to the studio sync. Many of these analog time-base correctors were designed specifically for use with particular video tape recorders.

The digital time-base corrector which resulted from lower cost digital components is a stand-alone device. It can be used with almost any video tape recorder and is often billed as a cureall for all sorts of problems encountered with recorded video signals.

It might be useful to the broadcast engineer to point out a few things that a time-base corrector cannot do. These things fall under the general heading that the time-base corrector can't improve poor picture quality if it is not directly related to time-base errors. If the recorded program had poor lighting, wrong color, poor signal-to-noise ratio, the time-base corrector can't do much about it. It basically corrects time-base errors in recorded signals. Some of the time-base errors can be recognized as shift in color, blurring, flag waving, color streaking, or rolling upon switching.

Figure 12-7 shows a block diagram of a digital time-base corrector. The main element of the processing portion of the system is a semiconductor memory. This memory is controlled by a block called "memory control logic."

The signal coming in will have the proper phase with respect to its own sync pulses. This signal is detected and applied to the

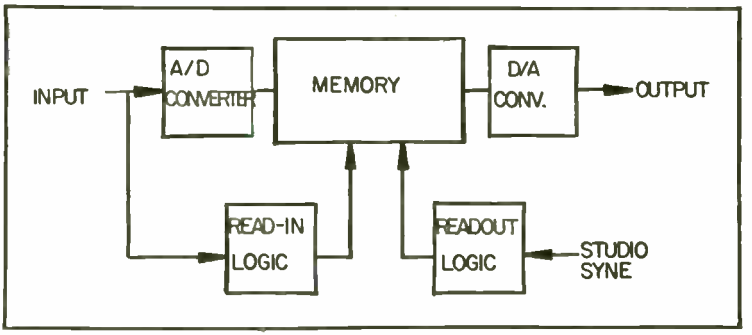


Fig. 12-7. Block diagram of a digital time-base corrector.

control logic that will make it write directly into the memory. Thus, in our memory we will have a pretty good representation of the incoming signal. The rate at which it is written in may vary with the time-base errors. But, once in the memory the signal will be in pretty good form. Now, the question is to get it out.

The reading of the memory is controlled by a digital signal derived from the reference sync signal and color subcarrier. These, in turn, are derived from the studio sync. Thus, the readout of the picture that is stored in the memory will be controlled entirely by the studio sync. Just about any time-base error can be eliminated in the corrector. Finally, the digital signal is converted back into analog form and applied to the video chain.

The range of time-base errors that can be handled by a time-base corrector is often called the "window." For example, if a time-base corrector can correct any delay error up to one horizontal scan period, it is said to have a range or window of 63 microseconds.

As digital technology continues its rapid pace of development, we can expect time-base correctors to become smaller, less expensive, and to consume less power.

ELECTRONIC GRAPHICS

One place where digital technology has found widespread application is in the generation of numerals and alphabetic characters on TV screens, as well as the generation of all sorts of special graphic effects. The TV picture is derived by scanning. This means that any spot on the picture can be very accurately specified in terms of time intervals with respect to the sync pulses in the signal.

Suppose that we wish to generate a segment on the screen so that we can develop an alphabetic character. As shown in Fig. 12-8, the segment can be completely specified in terms of time intervals. The vertical position of the segment will be determined by the time that elapses from the vertical sync pulse. In Fig. 12-8, this is interval "C." The horizontal position of the segment is specified in terms of interval "A" from the horizontal sync pulse. The horizontal length of the segment is determined by the duration of pulse "B." Thus, the first segment in our picture is the top of the letter A. To form the complete character, segments of the proper duration are generated on each horizontal line for the required number of lines, "D."

Using this procedure, any number of graphic elements can be developed to provide almost any desired graphic effect. By changing the pulse repetition frequency of the graphics generator as compared to the sweep frequencies of the signal, the characters can be made to crawl in either direction horizontally or to roll vertically.

Early character generators used a very large number of small-scale integrated circuits. Modern systems take advantage of available large-scale intergrated circuits and often use microcomputers for control of the overall system.

THE VERTICAL BLANKING INTERVAL

The vertical blanking interval of a TV signal has always been a challenge to broadcast engineers. During this interval, the signal is blanked from the screen, but horizontal sync pulses are being transmitted. There is nothing in that part of the signal that normally carries picture information. In effect, we are transmitting a carrier without any information on it. Naturally, many different proposals have been advanced to make use of this part of the signal.

One of the first uses to be made of the blanking interval was the transmission of test signals. Test signals are added to the signal before broadcasting. Knowing what the test signals should look like makes it possible to make corrections for distortion suffered in transmission. Digital computers can be used to analyze these test signals and make automatic corrections to compensate for distortion.

Several other uses have been proposed for the vertical blanking interval. Many of these center around transmitting alphanumeric characters using ASCII characters. The word *teletext* has been used as a generic term to embrace all of these systems. In

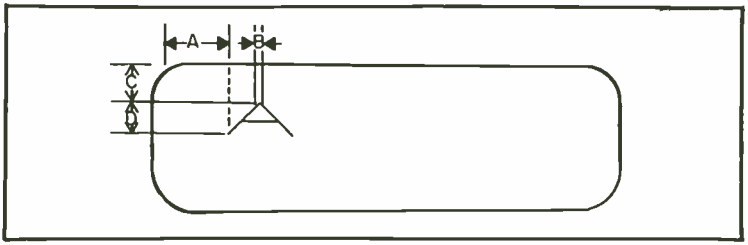


Fig. 12-8. The letters in the dimension line indicate time intervals used in generating graphics.

Europe, many such systems are actually on the air, transmitting everything from weather and time information to stock market quotations. At the time of this writing, these systems have only been used experimentally in the United States.

An interesting application involved adding captions to pictures for the benefit of deaf viewers. Captioning has been used on some networks for years, but while it is very helpful for those who are hard-of-hearing, it tends to be annoying to those who can hear the audio perfectly well.

The proposed captioning system would transmit the ASCII characters that make up the captions during the vertical blanking interval so that they would not normally appear on the screen. Thus, a viewer who was not hard-of-hearing would never know that the captions were being transmitted. A deaf person could, however, add a converter to his receiver. The converter would pick up and store the characters that were transmitted during the vertical blanking interval. These characters could then be displayed on the screen in response to commands that were also transmitted during the blanking interval.

There are a few unique problems associated with transmitting digital signals during the vertical blanking interval. In the first place, the data is transmitted at a slow rate, because the intervals only occur at a rate of 60 times per second. The individual bits of data must be transmitted at a much higher rate, however, because a complete character must be transmitted during the time of one horizontal line—about 63 microseconds.

These characters which are caught on the fly and stored are critical as to timing, and thus are much more susceptible to problems from such things as multipath transmission than are ordinary video signals. This problem is overcome by transmitting special synchronizing words that permit detection and correction of timing errors.

Chapter 13
Digital Audio

Although digital video processing got somewhat of a head start on digital audio in the broadcast field, digital audio is rapidly catching up. The advantages are fully as great in audio as in video, and in some respects the problems are not as formidable. It will probably come as a surprise that the number of bits per sample required for faithful reproduction of an audio signal is greater than that required with a video signal.

As we noted in our discussion of digital video, there are two separate considerations in digitizing an analog signal. The first is the sampling rate, which is determined entirely by the highest frequency component of the signal that we wish to digitize. Of course, the highest frequency in an audio signal is much lower than that of a video signal, so we can live with lower sampling frequency.

The other consideration is the number of bits required to represent each sample. This cannot be determined by a formula such as the Nyquist criterion, but depends on subjective considerations. The number of bits per sample in a video picture depends on how the resulting picture looks to an observer. Similarly, the number of bits per sample in a digitized audio signal depends on how the signal sounds to a listener. We saw in the preceding chapter that an 8-bit picture looked, for all practical purposes, as good as an analog signal. In audio, however, we find that we need about 12 bits per sample for the resulting signal to sound natural.

One of the things that makes digital systems easy to understand is the fact that all digital systems seem to have a great deal in common. Thus, it is not surprising to find that the digital

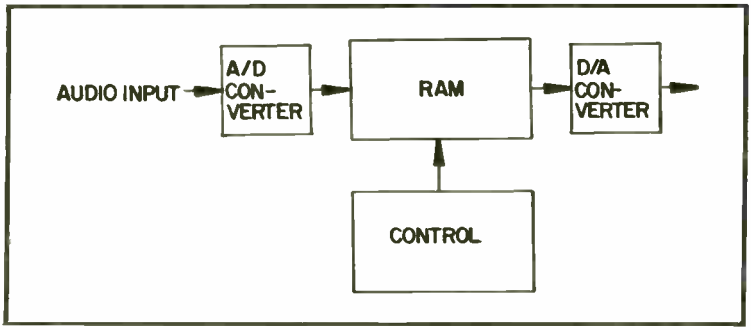


Fig. 13-1. Block diagram of a digital audio processing system.

audio processing system shown in Fig. 13-1 is practically identical to the video systems that we studied earlier. Functionally, the systems are nearly identical, but they differ in sampling rates and the actual use that is made of the samples.

One of the problems that digital processing solves in video is related to time-base errors. In audio, we are not concerned with accurate timing. About the only timing consideration is the development of echo effects by time-delay units. Of course, this can be accomplished digitally as shown in Fig. 13-2. Here we introduce a variable amount of delay into the signal by connecting the output to various stages of a shift register.

NOISE REDUCTION

One of the most promising uses of digital technology in the audio field is in reduction of noise in recordings. There are two different aspects of noise reduction involving digital technology. The first, which we have mentioned in Chapter 1, involves the fact that once we have a signal in digital form, it can acquire quite a bit of

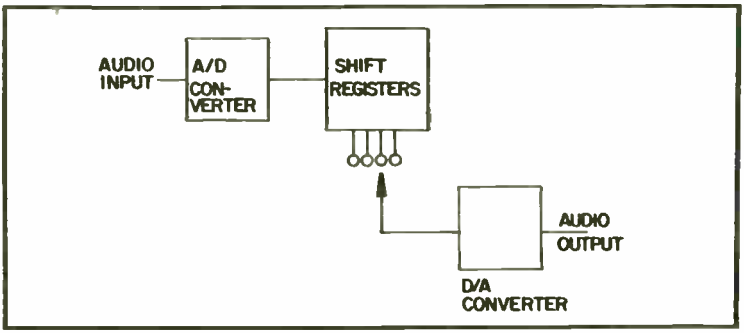


Fig. 13-2. Block diagram of an elementary digital audio delay line.

noise which can be removed by digital techniques. The second, which is more involved, is that digital techniques can be used to remove noise from an analog signal.

First let's review the way that the effects of noise can be removed from a digital signal. Figure 13-3A shows a digital signal. It consists of a series of pulses that are either high or low. Figure 13-3B shows this same signal after it has been corrupted by noise. By simply looking at the signal in Fig. 13-3B, we can quite easily tell where the highs and lows of the original signal are. Of course, looking at an audio signal doesn't do very much for us. We must have an electronic system that will do the job for us.

Figure 13-4A shows an arrangement that can be used to recover a digital signal from noise. The input stage is a Schmidt trigger of the type that we have mentioned several times. The output of this stage will go low whenever the input reaches a certain level. It will not go high if the signal falls below this level. It will go high only when the input falls to an even lower level.

Figure 13-4B shows how the hysteresis of the Schmidt trigger will recover a clean digital signal from noise. The hysteresis will prevent the output signal from swinging high and low with small noise perturbations of the input signal.

Looking at Fig. 13-4 we can easily imagine situations where the noise corruption can be so bad that even our Schmidt trigger won't recover the original signal. One such situation is shown in Fig. 13-5. Here, the noise perturbation is so bad that there are

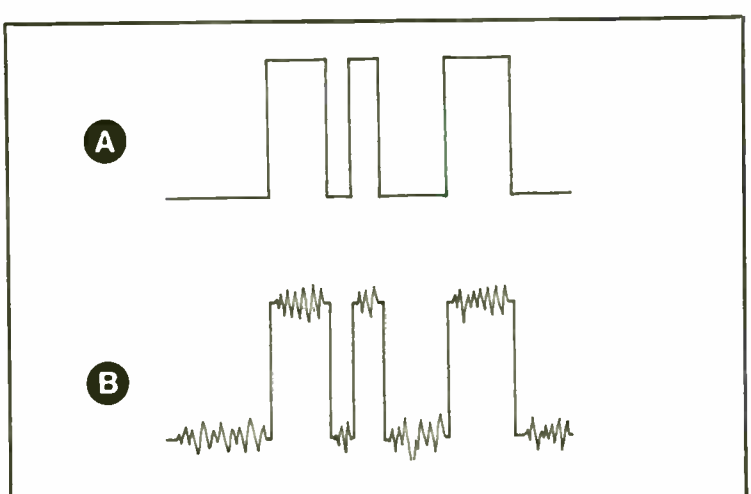


Fig. 13-3. Distorted digital signal.

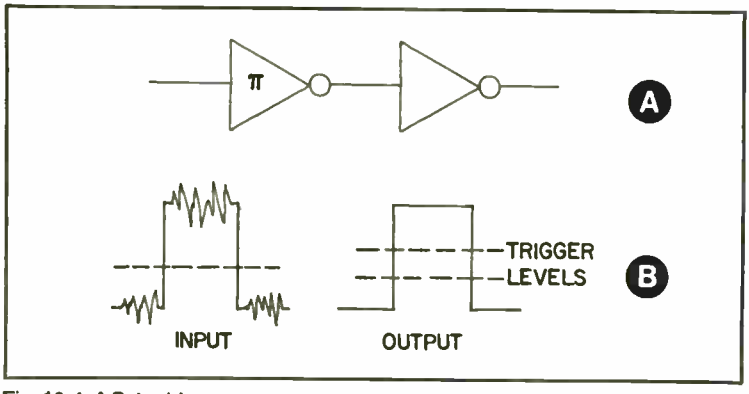


Fig. 13-4. A Schmidt trigger can be used to clean up a digital signal.

places within the high level of a pulse where the signal actually goes to zero (points A and B in the figure). The Schmidt trigger has no way of distinguishing the noise pulse from a low level.

We still aren't completely helpless. We have another tool at our disposal. If transmission standards have been adopted, we know a lot about the type of pulse train that is buried in the noise. For example, we may well know the duration of each pulse that makes up the system. In this case, we can install some sort of delay unit that will not let our signal change level until the end of a normal pulse period. Figure 13-6 shows a simplified approach to this. Here, our Schmidt trigger is followed by a one-shot multivibrator that will generate a pulse equal in duration to one of the pulses in

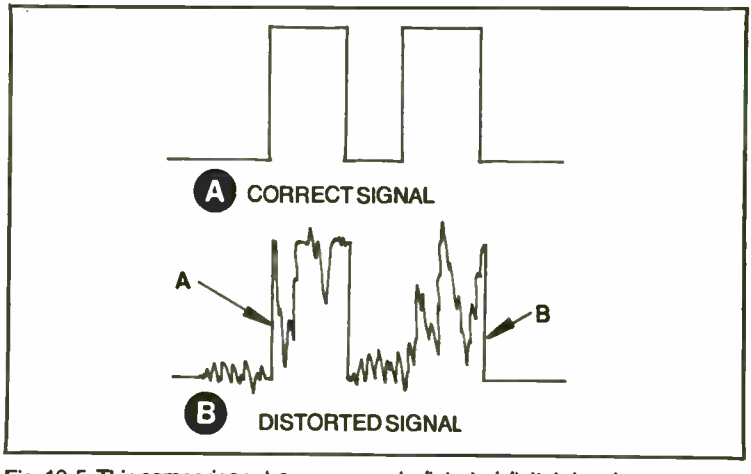


Fig. 13-5. This comparison shows a severely distorted digital signal.

our system whenever a high level is detected. This pulse is applied to a gate that will not pass any pulses between the limits of this time period. Thus, if a spurious noise pulse should occur during one of the signal pulses, it will not appear in the output.

There is still another way in which a digital signal can be corrupted by noise. Although we can recover the original high and low signals, we can't necessarily determine the exact instant at which the original signal changed state. There are two ways that we can attack this problem. One is to store our recovered signals and then read them out of the memory smoothly by a clock that operates at the data rate of the original signal.

There is another technique that can be used to recover signals from noise. It involves correlation techniques. We will discuss this in the following paragraphs.

REMOVING NOISE FROM ANALOG SIGNALS

Before we consider how to use correlation techniques in digital signals, let's look at just what we mean by correlation. Figure 13-7 shows a simple sinusoidal signal. Figure 13-7B shows a very noisy signal that may or may not contain the sinusoidal signal of Fig. 13-7A. Our problem is to find out whether or not the signal of Fig. 13-7A is indeed present in the noisy signal of Fig. 13-7B, and, if so, to recover it.

Suppose that we have a signal source that produces the signal of Fig. 13-7A. Also suppose for the moment that if this signal is present in the noisy signal of Fig. 13-7B, it will be in phase. We can get rid of this restriction later.

In Fig. 13-8 we have a system where we apply both the noisy signal and the signal from our generator to a multiplier. First let's suppose that the sinusoidal signal that we are looking for isn't present in the noisy signal at all. The noisy signal will be alternately swinging positive and negative, as shown in Fig. 13-9.

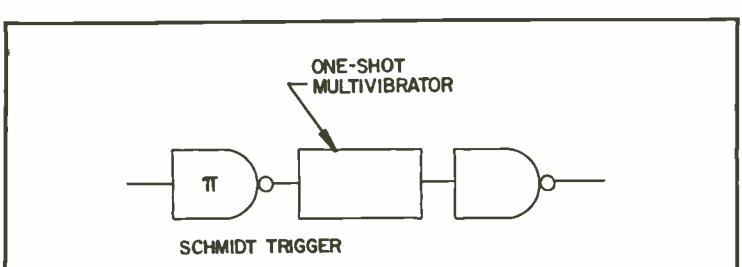


Fig. 13-6. A one-shot multivibrator can be used to establish pulse duration.

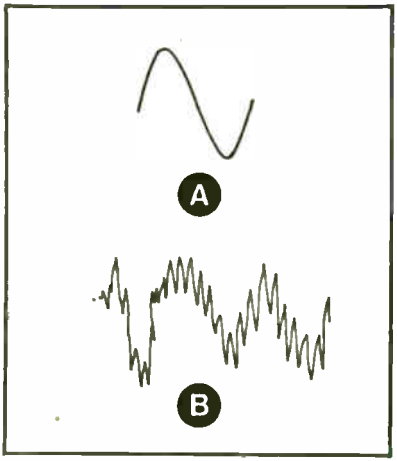


Fig. 13-7. The sinusoidal signal (A) is buried in noise (B).

Thus, the product of this signal and our sinusoidal signal will alternately be positive and negative. In the next stage of our system we integrate the output of the multiplier. If the signal is alternately positive and negative, the voltage across the capacitor will alternately go positive and negative so that the net charge in the capacitor will not increase.

Now let's look at the other situation. Suppose that the signal that we are looking for really is present in the noisy signal as shown in Fig. 13-10A. Now our system will multiply our sinusoidal signal in Fig. 13-10B by the noisy signal. Inasmuch as the sinusoidal signal buried in the noise is assumed to be in phase with our locally generated sinusoid, both signals will be positive at the same time.

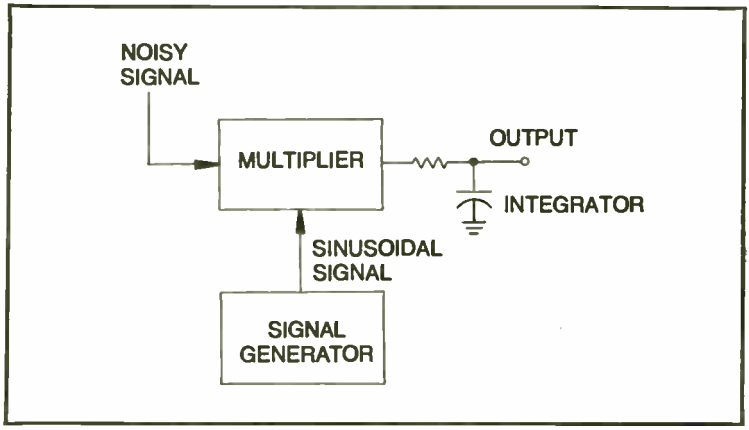


Fig. 13-8. Block diagram of a correlator used to detect a signal in noise.

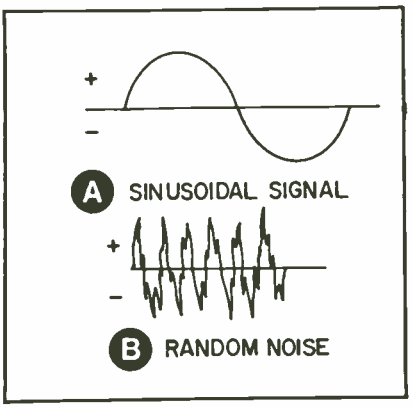


Fig. 13-9. The average value of the product of a sinusoid and noise is zero.

Inasmuch as the product of two positive numbers is positive, and the product of two negative numbers is also positive, the output of the multiplier will be positive at all times. If this signal is applied to our integrating capacitor, the net charge will tend to increase. This is shown in Fig. 13-10C.

To summarize, we now have a system that will tell us whether or not a locally generated signal is actually present in a signal that looks for all practical purposes like random noise. We call this technique the cross correlation of our locally generated signal and the noisy signal. We can look at the output of the integrator. And when it exceeds a certain value, we can use the locally generated signal as the received signal, because we know that it is actually buried in the noise.

So far we have tacitly assumed that if the signal we are looking for is actually present in the noise it will actually be in phase with our locally generated signal. In the real world, we would never be so lucky. What we must do is to add a phase shifter to our system as shown in Fig. 13-11A. Here we will vary the phase of our locally generated signal until the output of the integrator is maximum. As we vary the phase, the output of the integrator will be as shown in Fig. 13-11B. When the two signals are in phase, the output of the integrator will be maximum. Thus, we cannot only recover the signal that we are looking for from the noise, we can also determine its phase.

One thing that is apparent from our discussion of correlation is that it takes time to perform. We must have time to shift the phase of our signal and it takes time for the capacitor to integrate the signal to tell us whether or not the signal we are looking for is

actually present. In the meantime the noisy signal will continue to go on. It won't wait for us to perform our magic operations.

There is another limitation to our scheme. If we applied an analog signal, such as speech or music, we would need a separate locally generated signal for each frequency that might be present buried in the noise.

Both of these limitations can be overcome by using digital techniques. First, digital signals are easy to store. We can store the digitized version of the noisy input signals and correlate it at a more leisurely pace and again store the result. The other advantage of using digital techniques is that we are only looking for highs and lows, not for every frequency that might be present.

DIGITAL CORRELATION

A digital correlation system would have a block diagram similar to that of the analog system that was described above. However, the low cost of digital computer equipment makes it possible to perform very elaborate correlation functions at low cost. Some phonograph recordings of famous artists who are now dead have been resurrected and had all or almost all of the noise that was characteristic of recordings of their day removed.

The audio techniques that can be implemented with a computer far exceed those that we have discussed above. For example, we know a great deal about the nature of what might be on a recording. If the subject matter of a recording is a known musical piece, we know a great deal about just what we should be looking

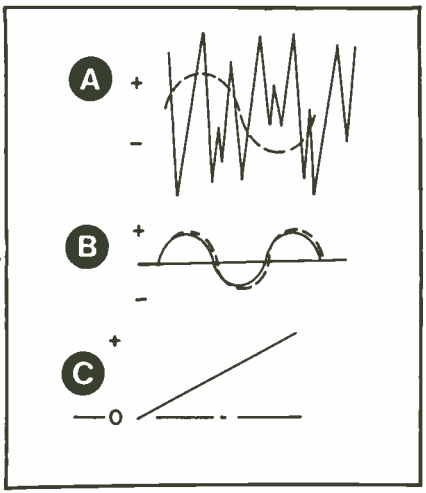


Fig. 13-10. Waveforms illustrating the operation of a correlator circuit.

for when we process the signal. Using this knowledge, we can operate on a signal several times to recover as much of the original as we wish.

DIGITAL AUDIO RECORDINGS

At the time of this writing there have been no universally accepted standards for digital recording of audio signals. Video recorders have been developed to the stage where ample recording bandwidth is available to accommodate elaborate digital encoding of audio signals for recording.

The advantages of digital recordings are obvious. The principal advantage is the freedom from noise. If the recording track on the recording is digital, all that the pickup system has to do is to decide whether the signal at any instant is a zero or a one. This means that the recording will not be seriously affected by dust and greasy fingerprints.

We can expect many advances in this field in the very near future. Probably the biggest obstacle at present is the lack of universally accepted standards that will lead to mass production of both recordings and reproduction equipment.

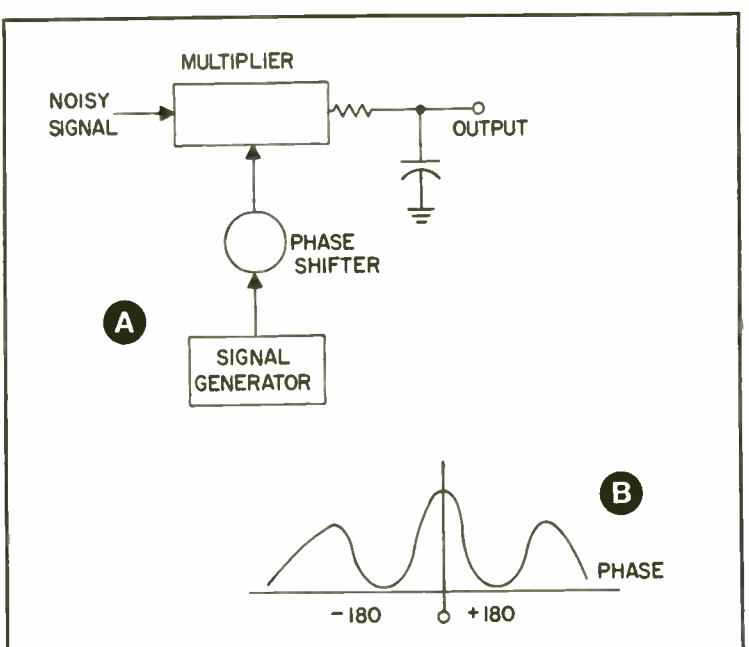


Fig. 13-11. Correlator system with phase shifter.

QUASI-DIGITAL TECHNIQUES

Although they do not properly fall under the heading of digital circuits, there are many techniques used in audio amplification that are nearly digital in nature.

The efficiency of vacuum tube amplifiers was seriously limited by the fact that there is an inevitable voltage drop across a tube even when it is conducting at its maximum value of plate current. The transistor does not suffer from this limitation. A transistor can be operated nearly like a switch. When it is turned on fully there is a large current, but a very low voltage drop. Inasmuch as the dissipation in the transistor is equal to the product of voltage and current, the dissipation will be low. Similarly, when the transistor is turned off, there will be a voltage across it but there will be little or no current. Thus, again, the dissipation is low. In a transistor connected as a switch, about the only time that the internal dissipation is significant is when the transistor is switching from on to off or vice versa. That is, when we have a significant value of both voltage and current.

Because of this possibility of very efficient operation of transistors, there is a trend to operate transistors as close to switches as possible. This permits the design of very efficient amplifiers.

The ideal way to operate a transistor amplifier is to operate on truly digital signals. In this way, the transistor is either on or off, except during the very short periods when the signal is changing from a high to a low or vice versa.

Because of the difficulty of changing from analog signals to digital and back again, the use of true digital signals, as we have been discussing them, in home and studio audio equipment is rarely practical. There are, however, many other techniques that can be used that are very similar in nature.

One technique is to represent the amplitude of a signal by the duration of a pulse. This technique, which is called pulse-duration modulation, keeps the signal at a high or low level almost all of the time. Pulse-duration modulation is also used in an amplitude-modulation scheme that eliminates the high-power modulator tubes and transformer that formerly characterized an AM transmitter.

We can expect increasing use of switching, or quasi-digital techniques, in the amplification and processing of audio signals.

Chapter 14
Digital Remote And Automatic Control

The subject of digital control systems is very complex. Many digital control systems include their own digital computer. Naturally, we can barely scratch the surface in a single chapter. What we will try to do is to include enough information to enable the reader to read and understand the instruction manuals covering control systems so that he can learn to handle them.

THE OPEN-LOOP SYSTEM

One type of control system is called an open-loop control system. As shown in Fig. 14-1, control information is fed into one end of the system and something happens at the other end. The switching arrangement used to change the antenna networks of an AM station from its daytime to its nighttime pattern is an open-loop system. Looking at the block diagram of Fig. 14-1, we can see that the open-loop system might be adequate for some situations, but rather inadequate for others. For example, an open-loop system is fine for controlling tower lighting. The switch can be thrown to switch on the tower lights, and the operator can look out the window to be sure that the lights turned on.

On the other hand, if an open-loop system is used to do something such as increase or decrease the power of a remote transmitter, the system in itself isn't capable of telling us whether or not the transmitter power actually changed, Thus, in addition to most open-loop control systems, we have some sort of instrumentation system to monitor the parameters that we are changing.

In Fig. 14-2, we show an open-loop system used in connection with a remote measuring system. The measurement system

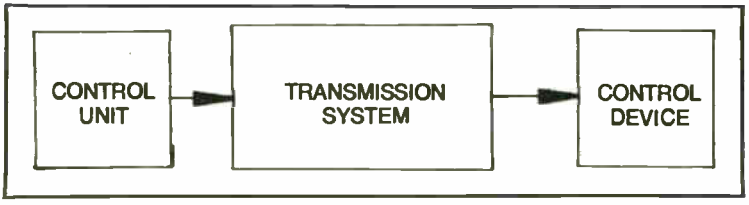


Fig. 14-1. Block diagram of an open-loop control system.

displays the values of many of the parameters of a remote transmitter, and the control system provides a method of changing these parameters. Looking at the figure we can see that both the measurement system and the control system are data transmission systems. The measurement system sends measurement data from the remote location to the control point. The control system sends control data from the control point to the remote location.

In Fig. 14-3, we have carried the development of our system a step further. Here we have used a single two-way data transmission system to carry data in both directions. This is still an open-loop system, because at the control point the measurement data is not connected directly to the control system.

THE CLOSED-LOOP SYSTEM

Let us for a moment consider the operator at the control point of the arrangement of Fig. 14-3 to be a part of our system. The system is now called a closed-loop system because the data follows a closed path. The measurement data is first transmitted to the control point where it is displayed on a readout. The data then enters the brain of the operator. In the brain the measured value of the parameters is compared with the correct value as known to the

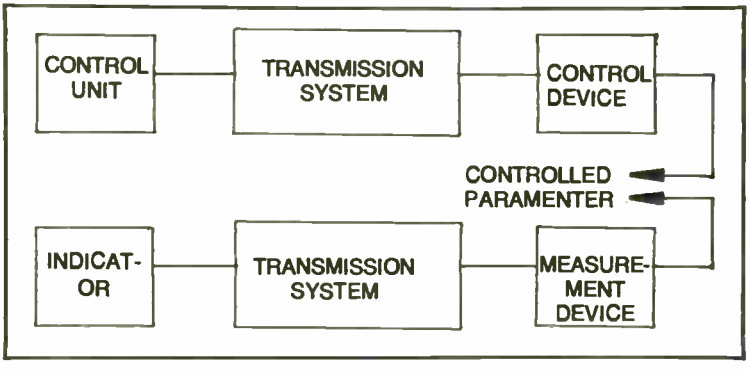


Fig. 14-2. An open-loop control system used with a measurement system.

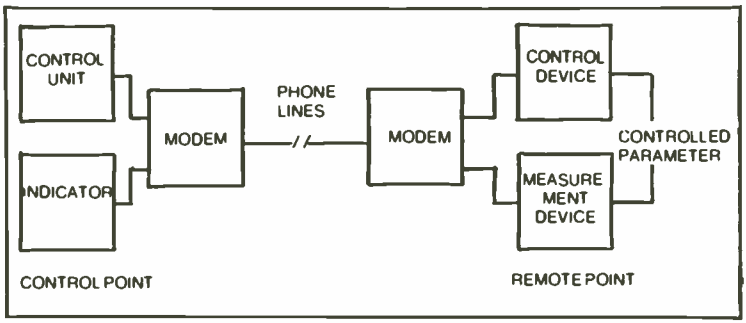


Fig. 14-3. Open-loop control and measurement systems using a single transmission system.

operator. If any of the parameters do not have the correct value, the operator's hand enters correction data into the control system. This data changes the value of the parameter and the new value is sent back to the operator.

Of course, in a true closed-loop system, we want something other than a human operator in the control loop. But for now we can learn a little about the behavior of a closed-loop system by leaving our operator in the loop a little longer.

Let's suppose that our operator is a little slow compared with what we want him to be. He notices that a parameter has fallen outside of limits, but he takes his time about sending the proper control signal. Even when he does initiate the control signal, he sends a small signal at first, then slowly increases it as necessary. If, for example, the parameter being measured happened to be the plate current of a final amplifier, the plate current might follow the curve of Fig. 14-4. Here, something happened that caused the plate current to become excessive. Because of the fact that the operator wasn't particularly speedy, the plate current remained at the excessive value for some time, and then corrected slowly. We would probably feel that this lag in the control is undesirable.

Let's look at the other extreme. Suppose our operator is particularly fast and makes corrections instantly. Now, when he notices that a parameter is out of limits, he reaches for the control and applies full correction. Before the measurement system can respond, the parameter has gone too far and is again out of limits in the opposite direction. The operator again applies a correction, and again the parameter overshoots in the other direction. The parameter, such as plate current, might behave as shown in Fig. 14-5. Here we see that instead of controlling a parameter, we have simply managed to make it oscillate about the desired value.

These two examples show us that a closed-loop control system might either have excessive lag or it might oscillate. This is exactly what we find when we get to a true closed-loop system. In fact there are three ways that our system can respond. As shown in Fig. 14-6, our system can have a slow lagging response, it can oscillate, or it can make the correction in a minimum period of time. Of course, this is the response that we really want.

THE CONTROL UNIT

Figure 14-7 shows a functional block diagram of the control unit of a closed-loop digital control system. The value of the controlled parameter is constantly monitored and sent to the control point by a link such as a phone line. At the control point, a digital signal is recovered from a modem. This signal is applied to a digital comparator. Also applied to the comparator is a digital signal that specifies the desired value of the controlled parameter.

The set, or desired, value of the controlled parameter is a digital signal that is derived from a switching arrangement. It may be taken directly from switches, or it may be stored in some sort of memory. In either case, the operator can change it at will.

In the comparator, the actual value of the controlled parameter is compared with the desired value. If the two agree, there is no output from the comparator. If the actual value of the parameter does not agree with the desired value, an error signal is produced. The error signal will tell whether the parameter is greater or less than the desired value. A signal is then sent to the remote point to change the parameter in such a direction as to restore it to its proper value.

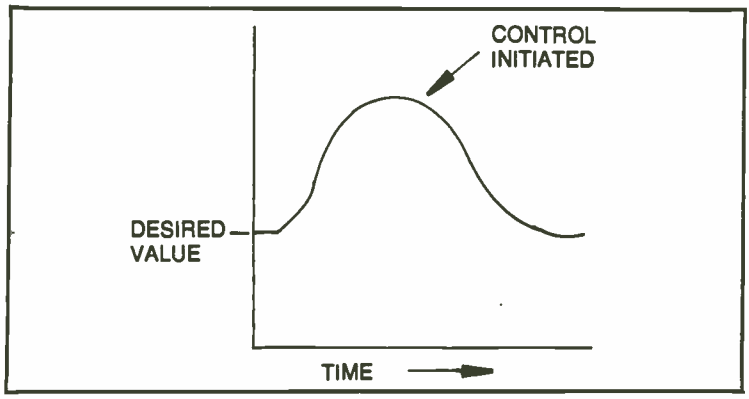


Fig. 14-4. Curve illustrating correction time lag in a control system.

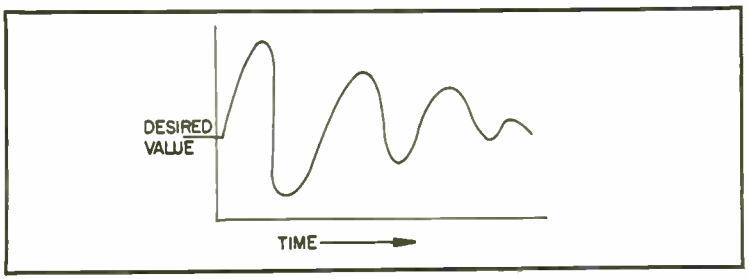


Fig. 14-5. Curve showing over-correction in a control system.

The system of Fig. 14-7 has been simplified considerably. In an actual system, the error signal will usually not indicate the direction of the error, but will be proportional to its magnitude. The error signal can be processed digitally so that the system will respond in an optimum way without either excessive lag or oscillation.

Although many digital control systems have been built using small-scale integration, the low cost of microprocessors has changed this trend. Now almost every control system of any complexity at all incorporates a microprocessor. This drastically reduces the number of components in a system. However, the functions that are performed are just about the same as those we have described.

COMPLEX CONTROL SYSTEMS

The closed-loop control system that we have been discussing in the preceding paragraphs is very simple compared with many readily available control arrangements. In fact, it isn't easy to understand how very complex functions can be performed by simple digital functional units. There doesn't seem to be much

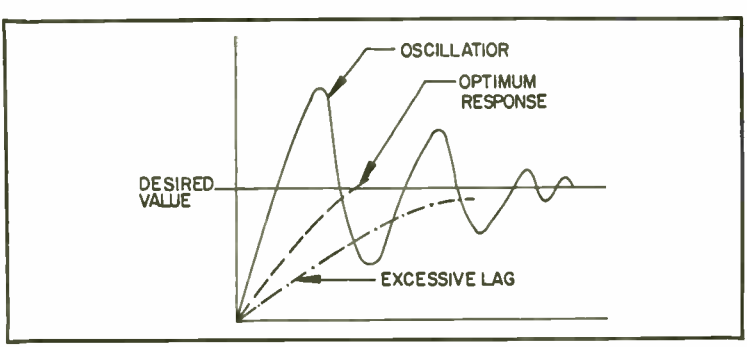


Fig. 14-6. These curves illustrate the modes of response of a closed-loop control system.

resemblance between a complex system and the simple functions that we have been discussing. The problem seems to be that although the individual digital functions are very simple, it takes a tremendous number of them to perform a complex operation.

One approach to understanding complex controllers is to realize that any imaginable control function can be performed by using both combinational and sequential logic. Combinational logic will enable us to derive an output signal whenever any combination of inputs is present. This function can be performed with a collection of logic gates or circuits that perform similar functions. By adding sequential logic, we can insist that certain operations must be performed before other functions can be performed. We can even specify the time that must elapse between functions. For example, we can design a circuit for turning a transmitter on and arrange the gates so that the heater voltage for the tubes must be applied before the plate voltage can be applied. By adding a counter in the circuit, we can specify how long the heater voltage must be applied before the plate voltage.

ARITHMETIC OPERATIONS

In this book we have frequently mentioned the binary number system, but about the only operation we used with the system was counting. We have had no occasion to mention that various arithmetic operations can also be performed with digital circuits. Before we look at the subject of binary addition, let's take another look at exactly what we mean by addition. Normally if we were to add 8 and 4 in the decimal system, we would say that the sum is twelve. This is not basic enough for us to apply it directly to binary arithmetic. Let's look at what we actually do when we add 8 and 4:

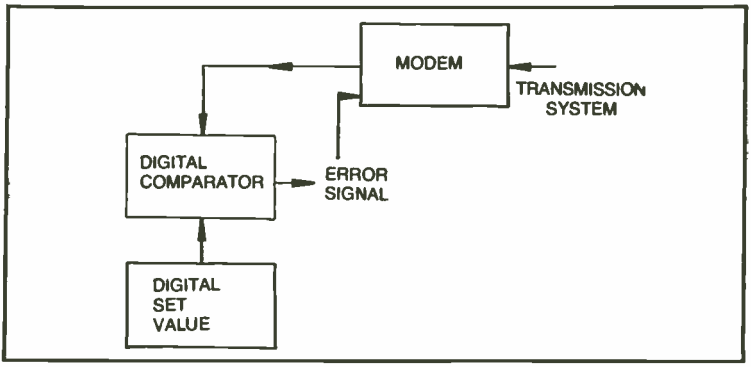


Fig. 14-7. Block diagram of a control unit.

$$\begin{array}{r} 8 \\ +4 \\ \hline 12 \end{array}$$

What we actually do when we perform this addition is to put a 2 in the right hand column, and then carry a 1 to the next column. Thus, we can say that:

$$8 + 4 = 2 \text{ (with a carry)}$$

Now let's look at binary addition. We stated the rules in Chapter 1. They are:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 1 = 10$$

Let's reword the last statement to read

$$1 + 1 = 0 \text{ (with a carry)}$$

With this little bit of information, we can see that the exclusive OR gate will perform binary addition, if we provide some means to generate the carry signal when we need it.

It is worth noting that the binary addition function is not the same as the logical AND function that we have performed so many times. For example, we know that if we AND the binary numbers for 4 and 2, one bit at a time we will get

$$\begin{array}{rll} & 0100 & (4) \\ \text{AND} & 0010 & (2) \\ \hline & 0000 & \end{array}$$

This is because the only time that we get a 1 from the logical AND process is when both of the inputs are 1. If we AND the bits of the two words above we see that none of the columns has two 1's in it.

On the other hand, if we were to add the two binary numbers above we would get:

$$\begin{array}{rll} & 0100 & (4) \\ + & 0010 & (2) \\ \hline & 0110 & (6) \end{array}$$

It is a good idea to keep these two functions separate when dealing with a complex system where we might encounter both operations.

Without getting into details, digital ICs are available that will perform the various arithmetical operations readily. Thus, we can multiply, add, subtract, and divide with little trouble. In fact a single microprocessor IC can perform both arithmetic and logic operations.

Chapter 15
Troubleshooting Digital Equipment

The well-accepted method of troubleshooting analog equipment is based on the troubleshooter having a rather thorough knowledge of just how each stage of the equipment works and how each stage contributes to the overall performance of the system. For example, in troubleshooting the audio console in a broadcast station, the technician carefully checks each stage until the faulty stage is located. He then proceeds to check individual components until he locates the defective component. The component is then replaced.

There is nothing wrong with this approach, if it is reasonable under a given set of circumstances. Unfortunately, there are several reasons why it cannot be adopted in many digital items. In the first place, the state-of-the-art in digital equipment is advancing so rapidly that equipment is being installed so fast that the busy engineer or technician can't keep up with all of the details. When trouble occurs, the pressure to get it fixed and to get the station back on the air is so great that there isn't time to learn the details of system operation.

Furthermore, in a digital system, all of the systems components are buried in integrated circuits so that a functional approach must be taken.

Naturally, the best approach to troubleshooting any piece of equipment is to be prepared in advance. If time is available, the person responsible for troubleshooting must become sufficiently familiar with each item of equipment so that he will be prepared to troubleshoot it when trouble occurs. Although there is probably universal agreement on the fact that this is a good approach to troubleshooting, there is probably also near universal agreement

that it will rarely happen. In all probability, when a digital system fails in a broadcast station, someone who has never had time to become familiar with the details of its operation will be called on not only to fix it, but to fix it quickly.

Fortunately, the situation isn't hopeless. There are many aspects of digital systems that tend to make them all alike. They all use digital signals. They all use the same functional operations. Similar pulse shape and timing considerations are involved in all of them. The result of all of this is that it is very frequently possible to isolate and correct faults in a digital system with only a very superficial knowledge of just how all of the components work together to perform the overall system functions.

THE SAME OLD STUFF

One thing that must be born in mind is that digital systems are subject to many of the same types of troubles that affect any electronic system. Many of these common faults are so simple that they are often overlooked as the technician becomes almost overwhelmed by the complexity of the complete system. Sometimes simple things are checked only after many hours of useless sophisticated troubleshooting.

As a first step in troubleshooting any type of equipment, look for the common faults such as poor connections, broken wires, short circuits, and even the power plug not being connected. Often, checks of this type will save hours of troubleshooting.

The printed-circuit board has some unique faults. Often there can be a faulty soldered joint that looks OK and works OK for a long time before it begins to cause problems. Solder bridges across the foil on the board may not make a complete connection for a long time. Therefore, an inspection of a printed-circuit board should include a close visual inspection—often with the aid of a light and magnifying glass. Carefully probing connections can help in locating poorly soldered joints.

THE INTEGRATED CIRCUIT AS A CAUSE OF TROUBLE

When solid-state devices first became popular we all learned that they were extremely reliable. We were told that once a device had lived through the initial burn-in period, its life was, for all practical purposes, infinite. The only possible cause of failure was exceeding its ratings. Everyone who had read the early articles in journals learned that transistors and integrated circuits rarely failed.

Those who gained experience with these devices soon learned that much of what they had been told about reliability was

little more than wishful thinking. Solid-state devices can and do fail. Perhaps not as frequently as capacitors and tubes, but that they can and do fail.

Looking at the schematic diagram of what is inside an integrated circuit shows that there are plenty of places for trouble to occur. In fact, the circuits are so complex that one wouldn't know where to begin. Fortunately, we need not concern ourselves with most of what is inside an integrated circuit.

The best approach to troubleshooting a system with integrated circuits is to know what is supposed to happen at the pins. We needn't concern ourselves with where the actual trouble might be. We can consider all faults as occuring just inside the package at the pins. With this approach, the number of different failures that can occur in any integrated circuit are quite limited. We can learn each of these faults and how to recognize them.

The types of faults that can occur in a digital integrated circuit fall into two classes. A pin can be opened, or it can be shorted to another pin. Most of the catastrophic faults can be thought of as being of these types.

There is another type of trouble that is familiar to anyone working in electronics. That is the intermittent fault. This may be one of the above faults that occurs intermittently, or it might be what we can call a dynamic fault that only occurs under certain conditions. First let's look at the IC as a cause of problems.

FAULTS IN INTEGRATED CIRCUITS

One of the most useful tools in investigating faults in an integrated circuit is a truth table showing how the circuit is supposed to behave. When the device is faulty, it will not behave in accordance with its truth table.

Open Pins

We can consider a fault to be caused by an open pin when the IC behaves as though there were no connection between the pin and what is inside. The IC will not be affected by whatever signal we apply to the open pin. If the open pin happens to be an output pin, it will have no effect on what is connected to it.

Pins Shorted to the Power Supply

Sometimes a pin of an IC will be permanently in a high state. It will behave as though it were connected internally to the positive supply pin. Usually, this "high" will have a higher voltage than a normal logic high. For example, in TTL logic a normal logic high is usually in the order of +3V. If the pin is shorted to the supply the

voltage will usually be closer to +5V. This is often a useful clue in troubleshooting.

Pin Shorted To Ground

A somewhat similar fault exists when a pin is permanently at ground level. This low is usually about the same as a regular low, so this type of fault isn't as easy to recognize.

Pins Shorted Together

In this instance, one pin will always have the same potential as another pin. They may not be at any state permanently.

In the following paragraphs we look at these types of faults in a little more detail and look at ways that we might recognize the faults.

The Open Pin

First let's look at what happens when the input pin of an IC is open. Figure 15-1 shows the input circuitry of a typical TTL IC. To make anything happen, we must connect the input pin, which is actually the emitter of a transistor, to a low level. If we leave the pin disconnected, nothing will happen. Now, if the connection between the emitter inside the IC and the input pin is open, the gate will behave as though the pin were stuck at a high level.

Note that we can't observe this condition at the input pin itself. Inasmuch as it isn't connected to anything, the pin will follow any voltage that we might connect to it. We can't definitely isolate an open pin by making a measurement on it. The best approach to locating an open input pin is to change the voltage on all of the input pins while observing the state of the output.

Figure 15-2 shows a TTL NAND gate and its truth table. If the gate is functioning properly, it will follow the truth table.

Now let's suppose that the lead inside pin A of the IC is open and not connected to the rest of the circuit at all. Of course, we can't tell this by looking at the voltage on the pin. If we apply a high

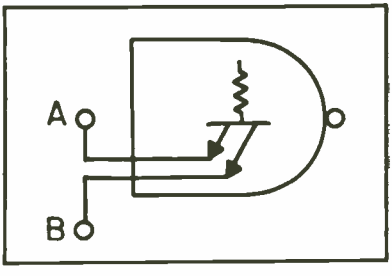
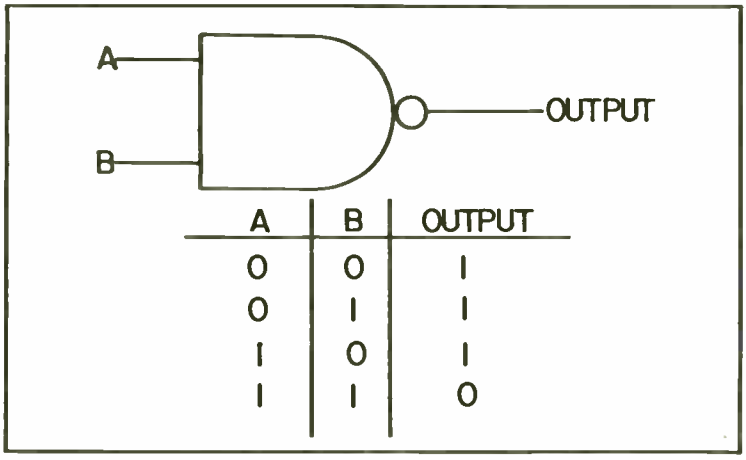


Fig. 15-1. Input circuit of a TTL IC.

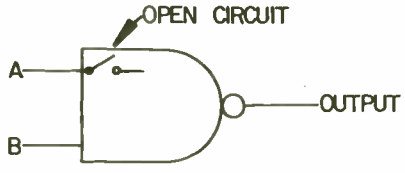


A	B	OUTPUT
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 15-2. TTL NAND gate and its truth table.

to the pin it will go high, and if we apply a low it will go low. What we can do is to apply different inputs to the pins while observing the output. Figure 15-3 shows what we will find.

The two columns at the left of the table show the input voltages that we actually apply to the inputs. The next column shows what we would see at the output if the circuit were functioning properly. So far we have a regular truth table. The last



INPUT		CORRECT OUTPUT	OUTPUT WITH PIN A OPEN
A	B		
0	0	1	1
0	1	1	0
1	0	1	1
1	1	0	0

Fig. 15-3. The truth table shows the output of a NAND gate with pin A open.

column at the right shows what we actually observe at the output pin. As shown in the figure, the only place where the observed result differs from the truth table is in the second line. This is because pin A, being open, drifts to a high level and the gate interprets it as being high even though we have tried to apply a low signal.

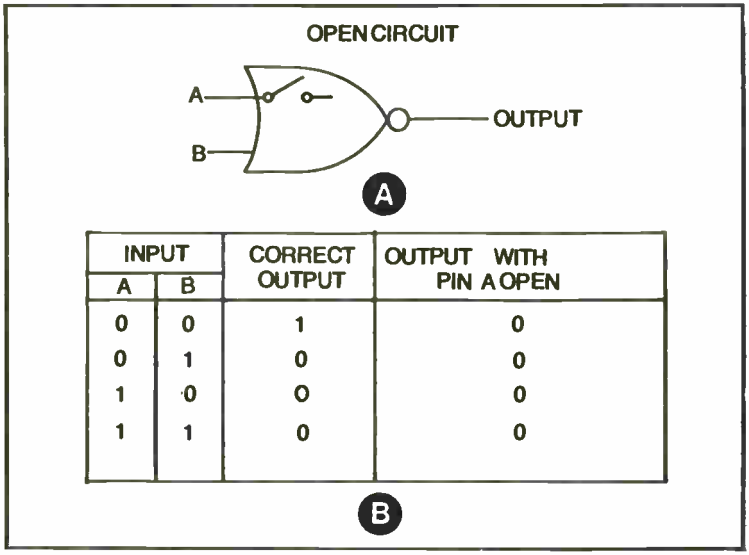
This figure also shows an interesting aspect of many faults that we find in digital systems; that is, the faulty IC may behave correctly for all but one possible combination of input signals. This could easily mean that a system using an IC with such a fault would behave properly most of the time. Suppose, for example, that the particular combination of input signals shown on the second line of Fig. 15-3 occured only rarely in the operation of the system. The fault would be evident only at these times. As can be seen, such a fault could be very elusive and hard to pin down.

Before we leave the subject of open input pins, let's look at a NOR gate with the same type of fault; that is, an open circuit inside the IC at pin A. Figure 15-4B shows the truth table for a NOR gate and the output we would get from the faulty gate. Note that the output will be low for all combinations of input signals. This is because the only time that the output of a NOR gate can go high is when both of its input pins are low. Inasmuch as pin A is open, the corresponding emitter will drift to a high level and stay there. Thus, there is no way that we can make both of the inputs low.

Looking over the table in Fig. 15-4B we see how the faulty gate behaves, but we also see something else. We see that the gate behaves exactly the way it would if the ***output pin were shorted to ground***. In either case, we would have isolated the fault to the same IC, which must, of course, be replaced.

While on the subject of open pins, let's look at the output pin of an IC. Of course, with the connection to the pin being open the output circuit of the IC will not change the voltage at the actual pin. It will be floating. Thus, the voltage at an open pin will depend on what it is connected to. In the usual case, the output of an IC is connected to an input pin of another IC.

Figure 15-5 shows such an arrangement. The input pin of IC2 will behave as though nothing were connected to it. Thus, it will drift to a level that *it* will consider to be a high. This high will, however, be lower than the usual TTL high. It will be in the order of about +1.5V. Such a level is considered to be a "bad level." This is because, although the gate will treat it as a high, it will actually fall into the "no man's land" between high and low logic



INPUT		CORRECT OUTPUT	OUTPUT WITH PIN A OPEN
A	B		
0	0	1	0
0	1	0	0
1	0	0	0
1	1	0	0

B

Fig. 15-4. Illustrated in the truth table is a NOR gate with an open input pin.

levels. This fact can be used to verify the fact that the output pin of IC1 is open.

Thus, whenever we find a bad level at the output pin of an IC, we can be reasonably sure that the output pin is open and that the bad level is coming from the input pin of the following IC. If the suspected IC happens to be in a socket, we can verify this very quickly. If the voltage at the output connection is the same whether or not the IC is in the socket, we can be sure that the fault is an open output pin.

Shorted Pins

There are three classes of faults that we can consider as shorts at the pins of an IC regardless of where the actual fault occurs. An IC can behave as though a pin were shorted to the positive supply,

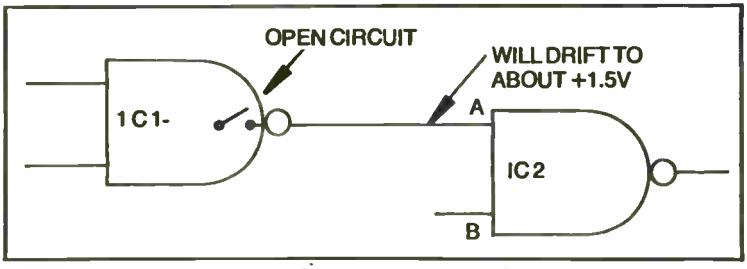


Fig. 15-5. An open output pin in IC1 will allow pin A on IC2 to drift.

to ground, or to some other pin. Frequently when an input pin is shorted to either the positive supply or to ground, the result will be catastrophic failure of the preceding IC.

Figure 15-6A shows an input pin shorted to the positive supply. As long as the output of the preceding stage is high, there will be no problem. When the output of the preceding stage goes low, there will be a short between the positive supply and the top of transistor Q4, which will usually result in excessive current and failure of IC1.

Figure 15-6B shows the situation where an input pin is shorted to ground. Here there will be no problem when the output of IC1 is low. When the output goes high, there will be a short from the bottom of transistor Q3 to ground. This will cause excessive current which can destroy IC1, but failure isn't quite as certain as in the case of Fig. 15-6A. There is a diode and often some resistance in the circuit, and if the short doesn't persist too long, IC1 can sometimes be saved.

A short in an output pin is more confusing. The effect will depend on the nature of the fault inside the IC. Frequently, the effect is the same as though the input pin of the following stage were shorted to either the positive supply or to ground. Usually,

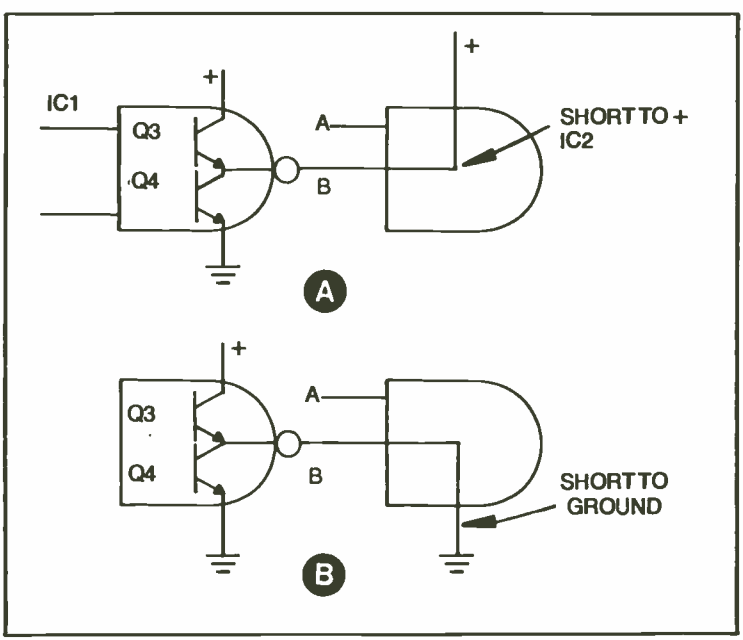


Fig. 15-6. These drawings show the effect of shorted input pins.

the result will be catastrophic failure of the IC. After this failure occurs, the output stage can act as though the output pin were shorted to either the supply or to ground. The result will be that the output pin is usually permanently either high or low.

The remaining case to consider is when a pin is shorted to some other pin, not either the positive supply or ground. The effect will, of course, depend on just where the short occurs. If two input pins are shorted together, the fault can often be deduced by drawing a truth table of the bahavior of the gate and comparing it with the actual truth table for this type of gate. When an input pin and an output pin are shorted together, the nature of the fault may indeed be strange. Sometimes the stage will oscillate. More often than not it will not obey its truth table and can be diagnosed as being defective.

CIRCUIT FAULTS

Often in digital systems, faults occur on the foil of the printed-circuit boards. Most of these faults will produce the same symptoms as open or shorted pins. They can be isolated more easily, however, because measurements can be made at points along the foil to isolate the fault.

DYNAMIC FAULTS

While discussing the various types of faults that can occur in digital ICs, we have tacitly considered that the faults would be what we might call "hard faults." That is, we assumed that once the fault occured it would persist and that we could troubleshoot until we located it. This is often the way that digital circuits behave. However, there is another class of faults that we might call dynamic faults. They occur only while the system is in operation. If we were to test each of the ICs in the system, we might not be able to find anything wrong with them. Under the general heading of dynamic faults we should consider three types of problems:

☐ The system may be subject to interference. This interference might come from the outside world, or it may be generated within the system itself.

☐ The system may be causing interference to some peripheral device that works with it, or

☐ There may be some dynamic problem in the system. Often such faults are thermally related, but sometimes they result from marginal design.

INTERFERENCE

A digital system may fail dynamically whenever interfering signals from the outside world find their way into the circuits. This is probably more common in broadcasting because so much of the equipment must operate in strong RF fields. Usually, such a fault can be traced to improper grounding or shielding.

First, let's consider the problem of shielding. There is a tendency to think of a shield as something that protects what is inside it in much the same way as a raincoat protects a person from the rain. While this idea is adequate in many cases, a better understanding of how a shield works can be had by taking a more fundamental approach.

Whenever an electron anywhere in the world moves at an RF rate, it produces a field that makes every other electron in the universe want to move at the same rate. Shielding is produced by arranging conductors so that when a field is set up one conductor, an equal and opposite field is set up in another conductor, with the result that the two fields cancel locally and have no effect on other conductors.

A good example of shielding is the familiar coaxial cable. Here the current in the inner conductor causes an electromagnetic field. An equal current in the opposite direction flows in the outer conductor and also causes a field. The two fields cancel at the outer conductor so there will effectively be no field outside the cable.

Another factor that must be taken into consideration in connection with shielding is the skin effect. RF currents will only flow on the surface of a conductor. The situation is shown in Fig. 15-7. Note that although a grounding bolt passes through the panel, the ground current will flow only on the surface of the panel.

Whenever RF is found inside a digital system, there can be trouble. Correcting the problem is often tedious, but careful attention to shielding and grounding will almost always clear it up.

Other Dynamic Faults

Most of the dynamic faults in a digital system that are not caused by interference are caused by one of three things:

1. Poor connections that operate properly most of the time, but occasionally open, causing spurious pulses by making and breaking the circuit on a random basis. This type of trouble is found by careful inspection of all connections. Suspect connections should be touched up with a soldering iron.

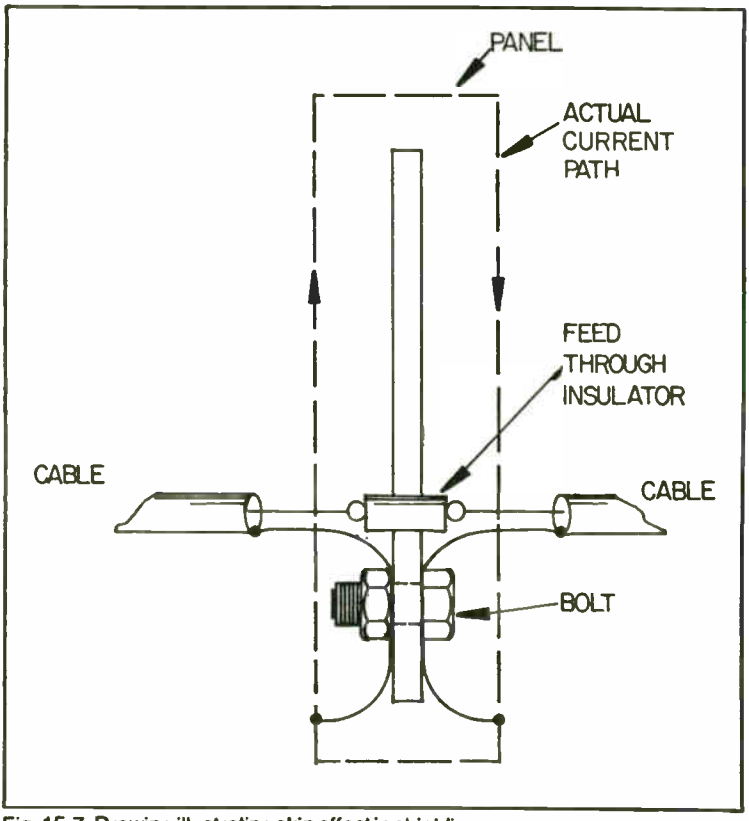


Fig. 15-7. Drawing illustrating skin effect in shielding.

2. Thermal problems that change the behavior of a digital IC. Thermal faults can cause the threshold level of a gate to change, or they can slow down the response of an IC. The best way to isolate thermally related faults is to heat suspected ICs with a blast of hot air from a hand-held hair drier and to cool them with a spray made for the purpose. Usually, one or the other of these will cause a thermally related fault to either appear or to clear.

3. Spurious pulses on power supply lines. We discussed this at some length in the chapter on power supplies and noise. Often a spurious pulse is of such short duration that it is hard to see on an oscilloscope. Frequently spurious pulses are caused by the failure of components that were included in the design to eliminate them. Frequently, a faulty despiking capacitor can be found by noticing that the trouble disappears when the suspected capacitor is bridged with a similar capacitor known to be good.

Sometimes spurious pulses are the result of marginal design. A frequently found case of this type is where the designer simply didn't include enough despiking capacitors to take care of what might happen as the system ages.

TEST EQUIPMENT

Now that we have an idea of the types of faults that we might find in digital systems, let's take a look at what we might use for test equipment to locate the cause of these troubles. Although much specialized test equipment is available that is specifically designed for use in digital systems, the old tried and true VOM and oscilloscope can also be used to advantage.

The VOM

In a digital system, the voltages are either logic highs or logic lows. In TTL, the high and low values of voltage are shown in Fig. 15-8. When the states of the various stages of the system are not changing too rapidly, these states can be checked using an ordinary VOM with a sensitivity of 20,000 ohms per volt or higher. Figure 15-9 shows the high, low, and bad levels marked on a 5-volt VOM scale.

If the system is one when logic states change only in response to inputs that can be disconnected, the levels at all of the pins of the ICs in the system can be checked rather quickly for a bad level that would indicate. trouble. If this fails, the various highs and lows can be checked against the truth tables for the various ICs.

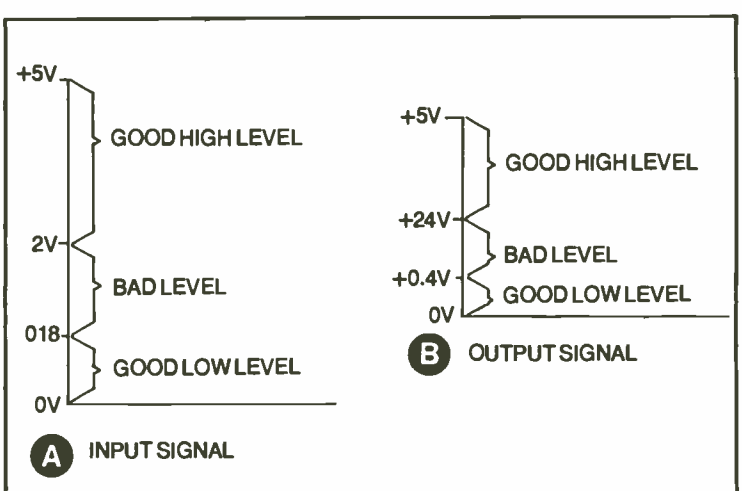


Fig. 15-8. High and low levels in TTL ICs.

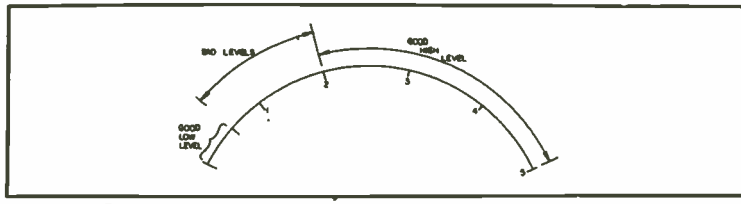


Fig. 15-9. Good high and low logic readings on a meter scale.

There are two limitations to this approach. First, there are some systems that operate with a clock operating at a high frequency. Usually, the system won't do much of anything if the clock is stopped. The result is that everything changes much too fast for a meter to follow. The other limitation of this approach is that some faults are dynamic in nature and won't show up on a meter. In such cases, the cathode-ray oscilloscope can be used to advantage.

The Oscilloscope

The best way to set the oscilloscope sensitivity for troubleshooting digital systems is to adjust it so that the top graticule of repetition rate and time of occurence of the pulses that we find in digital systems all at the same time. If the oscilloscope has a compensated probe, the compensation must be properly adjusted before intelligent measurements can be made. Figure 15-10 shows how a square wave will look on the oscilloscope when the probe is under-, over-, and properly compensated.

The best way to set the oscilloscope sensitivity for troubleshooting digital systems is to adjust it so that the top graticule of the display corresponds to the lowest voltage. In Fig. 15-11, the controls are set for checking TTL circuitry where the highest voltage is +5V and the lowest voltage is ground. The input is also set for DC coupling so that DC voltage measurements can be made. In Fig. 15-12A, the probe is connected to a point at +5V and at Fig. 15-12B the probe is applied to a ground level.

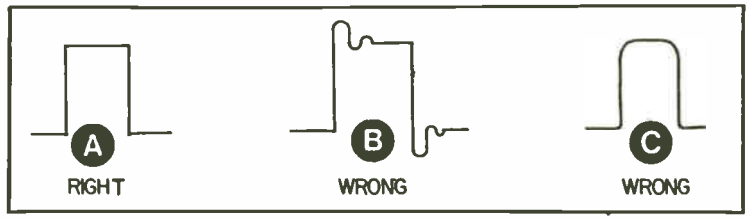


Fig. 15-10. Waveforms showing proper and improper compensation of an oscilloscope probe.

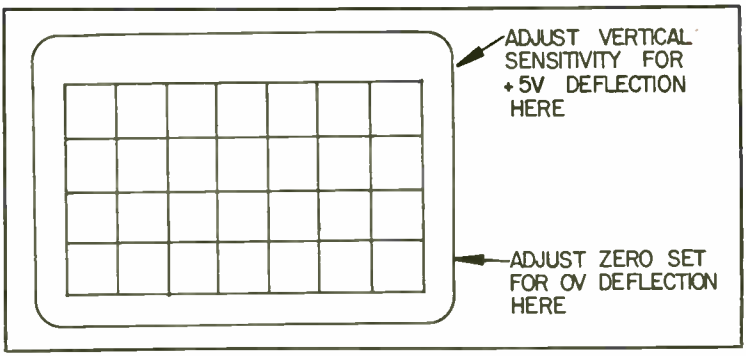


Fig. 15-11. Logic readings are simplified by adjusting oscilloscope vertical sensitivity to correspond high and low levels with the graticule markings.

So much for DC measurements. Most of the measurements we will make deal with pulse trains. If the sweep of the oscilloscope is not synchronized to the pulse train, the display will be in continuous motion, as shown in Fig. 15-13A. Although this display will not show any of the details of the pulses, it isn't completely useless. It will tell us that a pulse is present and that the pulses vary between ground and about +3V. Sometimes this is all we need to know about some point in a circuit.

If the oscilloscope is properly synchronized, we can see the details of the pulses as in Fig. 15-13B. Here, by properly setting the horizontal controls of the scope, we can measure both the duration and the repetition rate of the pulses.

In order to get a stationary display, we must properly synchronize the oscilloscope. There are two ways that we can do this. One is to use internal synchronizing. With this mode, the sweep of the oscilloscope is initiated by one of the pulses that is displayed. If we use positive triggering, the sweep will start when the pulse is going in a positive direction. If we use negative

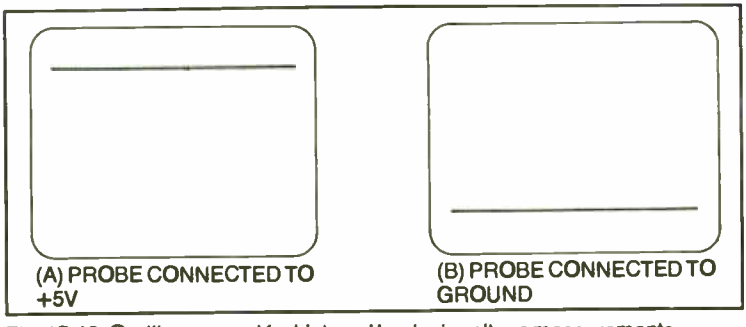


Fig. 15-12. Oscilloscope set for high and low logic voltage measurements.

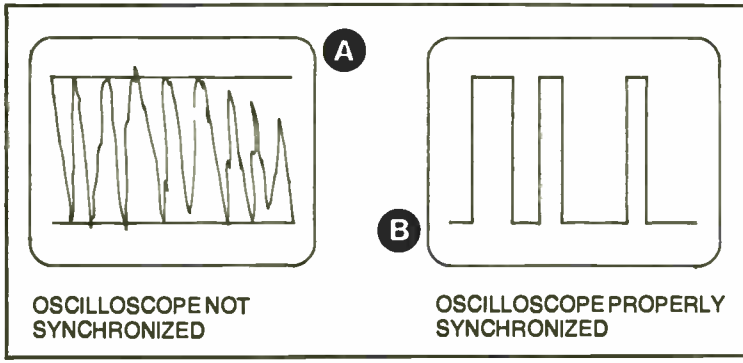


Fig. 15-13. The horizontal sweep of an oscilloscope must be synchronized with the frequency of a pulse train.

triggering, the sweep will start when the pulse is going in a positive direction. If we use negative triggering, the steep starts when the pulse is going in a negative direction. This is shown in Fig. 15-14 A and B. Usually, the edge of the pulse that actually starts the sweep will not show on the screen, but will be off to the left.

The other way we can use to trigger the oscilloscope is to use external triggering. In this mode, we get a pulse from some point in the system and apply in to the external triggering input of the oscilloscope. Some systems have synchronizing pulses available at some point that can be used to start the sweep at the beginning of a word or byte. Often, the biggest problem in using an oscilloscope is to find the proper triggering input so that we can see what we want to see.

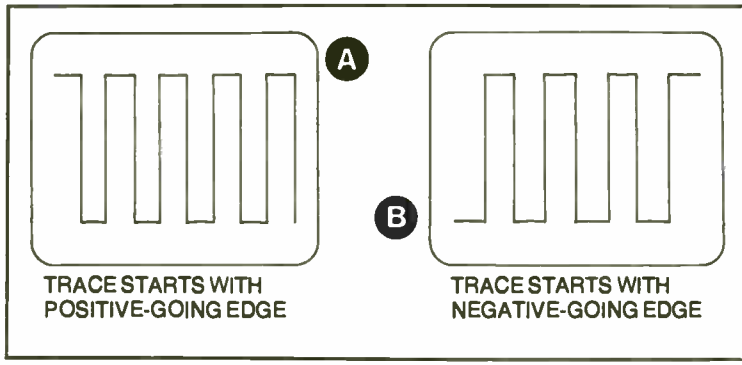


Fig. 15-14. Scope displays showing positive and negative horizontal sweep triggering.

One of the most obvious faults that can be detected with an oscilloscope is noise in the system. Figure 15-15 shows some oscillograms with permissible and excessive values of noise. Usually, the noise can be traced to the point in the system where it starts. More often than not, a defective component will be the cause.

Another thing that can be found with an oscilloscope is a spurious pulse. There are three different kinds of spurious pulses that get into digital signals. The first kind is the pulse that is caused by a defective component and is generally in synchronism with the other pulses in the system. Figure 15-16A shows a pulse of this type. It is probably caused by the switching of some stage in the system that is not directly in line with the pulses that we are observing. The cause of this pulse can be found by probing through the system to find out where it starts.

In Fig. 15-16A we are using external synchronizing so the trace will always start at the same time. The spurious pulse marked "T" doesn't seem to be directly related to the pulses that are being displayed. What we must look for is something else in the system that occurs at time "T".

Suppose that while probing around the system we find the waveform shown at Fig. 15-16B. There is nothing wrong with this pulse, but due to the fact that it occurs at time "T", we are suspicious. Further investigation shows that the points where the two traces of Fig. 15-16A and B were taken are both part of the same IC as, shown in Fig. 15-16C. Thus, we might well suspect that in some way the pulse of Fig. 15-16B is affecting the trace of Fig. 15-16A. It might be leakage in the IC. It could also be caused

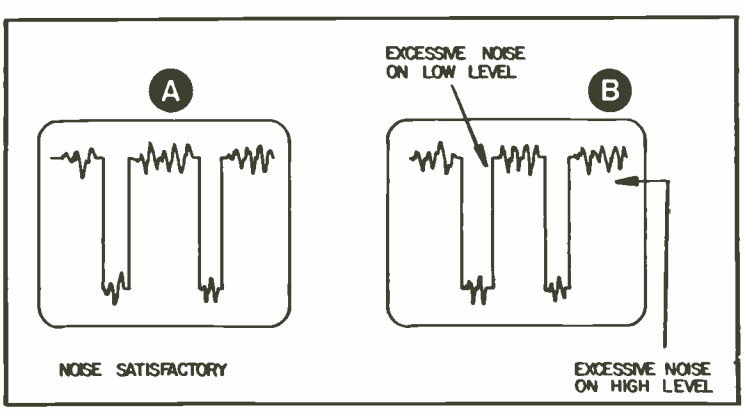


Fig. 15-15. Scope displays showing excessive noise

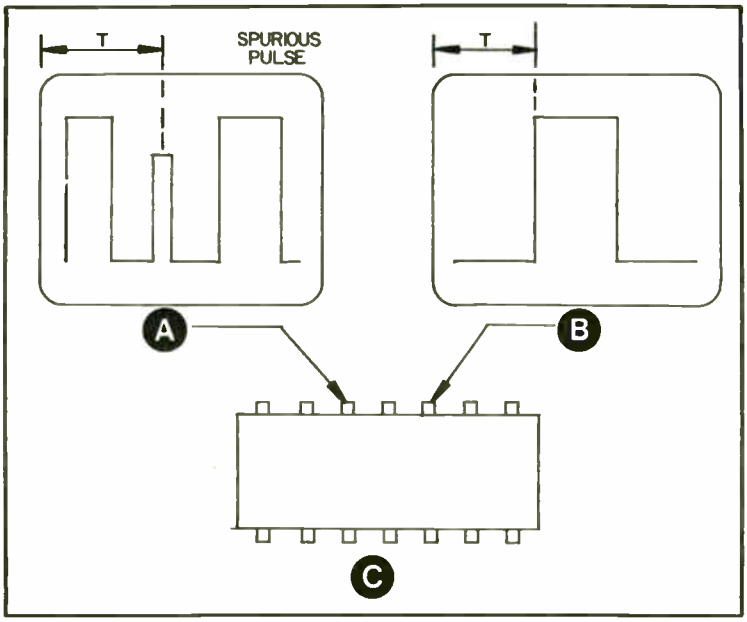


Fig. 15-16. Scope waveforms are useful in locating the cause of a spurious pulse.

by a faulty despiking capacitor, or possibly even a design weakness where the despiking capacitor was omitted. In any case, we have traced the spurious pulse to its source.

The next type of spurious pulse is related to the frequency of the power line, which is probably not harmonically related to the pulse frequency of the system. If we look at the pulse train of Fig. 15-17A, which has a pulse repetition frequency of 1 kHz, we

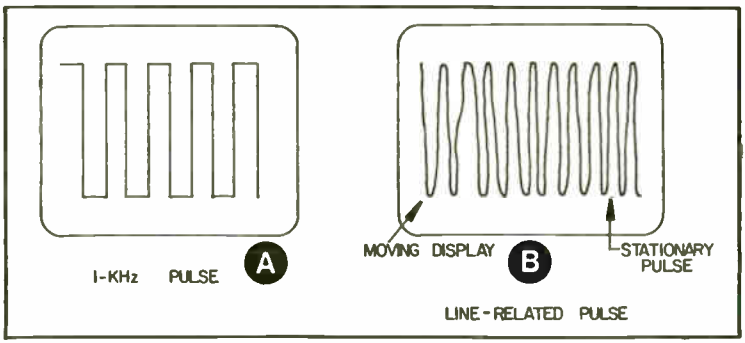


Fig. 15-17. Power line triggering can be used to isolate a "glitch" related to the power line frequency.

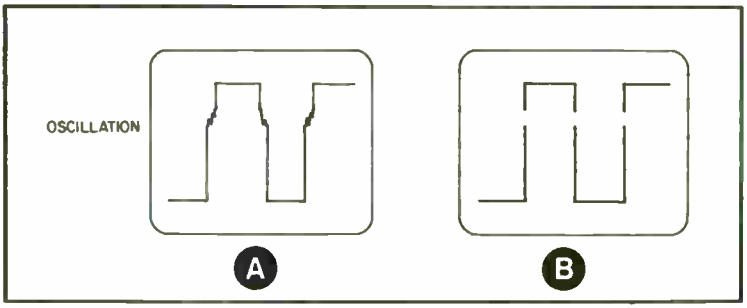


Fig. 15-18. Waveform showing oscillation on leading or trailing edges of a pulse.

probably won't see the pulse at all. After all, the pulses of the system occur every millisecond and the power-line-related pulse will occur only about once every 17 milliseconds. Furthermore, the power line pulse will be constantly moving with respect to the pulses of the system.

The way to spot pulses or "glitches" that are related to the power line frequency is to synchronize the oscilloscope to the power line. Now the main pulses of the system will not hold still but will be in continuous motion, making a blur on the screen. The spurious pulse will, however, hold still. This is shown in Fig. 15-17B.

The last class of spurious pulse or "glitch" will be one that isn't related in frequency to either the power line frequency or to the pulse rate of the system. It is probably caused by something outside the system. In a broadcast environment, it might be related to either the carrier frequency or to the modulation frequency. Sometimes sweeping the oscilloscope at some submultiple of either of these frequencies will make the pulse hold still on the scope. Once a spurious signal is made stationary on the oscilloscope screen, it can usually be traced to its source.

Often when integrated circuits start to fail, their timing will go off. The output of a stage that is about to fail might be much longer than normal. When this pulse reaches the input of the following stage, it might well cause oscillation as we discussed earlier in the book. Sometimes, if we are lucky, this oscillation will be evident on the front or back of a pulse as in Fig. 15-18A. More often than not, the frequency of oscillation will be much higher than the pulse frequency so that it will not be obvious, but will look more like the trace of Fig. 15-19B.

Chapter 16
The Microprocessor and the Broadcaster

No book on digital electronics as it applies to broadcasting would be complete without at least an introduction to the microprocessor. Certainly, no device has had such a great impact on the configuration of broadcast equipment since the introduction of the transistor. Just about every type of broadcast equipment from transmitters and video equipment to test equipment is being changed by the microprocessor. Sooner or later every broadcast engineer and technician who wishes to avoid becoming obsolescent must become familiar with it.

Before we even begin to discuss the technical details, let's look at some of the problems that will be encountered by one wishing to master the device. There are a surprising number of engineers and technicans who work with microprocessors who have rather superficial knowledge of the subject. They know enough to get by, but are often stumped when problems arise.

Probably the biggest reason that more people haven't really mastered the microprocessor is due to the fact that the task is underestimated. This may be because the device itself is so small. In the past, no one underestimated the task of mastering the digital computer. The very size and complexity of a computer was enough to convince one that its mastery would take a great deal of time and effort. The microprocessor is actually similar to a large digital computer. Functionally, it is very similar. In spite of its small size, it contains a very large number of functional elements. There seems to be no short cut to real mastery of the subject.

Another point that must be borne in mind when approaching the subject is that a microprocessor is a digital system. A complex

one to be sure, but nevertheless a digital system. Unless the principles of digital systems are well understood, it is nearly impossible to acquire any real mastery of the microprocessor.

Is there a good way to become familiar with the subject? Probably not as good as we wish, but there is an approach that works. It starts out with becoming familiar with all of the functional aspects of digital electronics. Frequently, questioning shows that one who has a great deal of difficulty with the microprocessor doesn't really understand one or more aspects of basic digital electronics.

Once one has a good knowledge of digital electronics, the next step is to start learning about the microprocessor itself. An important aspect of this is becoming familiar with the terminology of the field. There are many good books, and seminars on the subject, but until one has at least a passing knowledge of the terminology, they are apt to be most difficult.

Lastly, one should gain some hands-on experience. This should preferably be done on a system that is procured for training rather than on operational equipment. But, often the first exposure will be to a faulty piece of equipment that must be put back on the air in the shortest possible time.

In the following pages we will present a very elementary introduction to the subject. No claim is made for completeness. The order of presentation has been chosen so as to develop a gradual familiarity with some aspects of the subject.

WHAT IS A MICROPROCESSOR?

Perhaps the simplest description of a microprocessor is that it is a single integrated circuit that contains all of the control and processing sections of a digital computer and sometimes more. Of course, this definition doesn't explain why the microprocessor is having such an impact on broadcast equipment. Why should a digital computer improve such things as transmitters, control systems, and video pocessing equipment?

The question arises because of the way most of us have become assustomed to thinking of digital computers. To most of us a digital computer is something that can make out pay checks, do accounting, and perform mathematical operations. Of course, this has been the principal application of digital computers. But the computer is also capable of operating as a controller. The cost reduction that has resulted from the development of the microprocessor has made it practical to apply computer technology to all sorts of control problems.

As we have seen throughout this book, a digital system consists of many functional elements such as logic gates and flip-flops that perform simple logic functions. The overall operation of the system depends on the selection of these units and on how they are connected together.

Suppose that we have a single integrated circuit that contains many logic elements and can duplicate the functions of our ordinary logic elements. Suppose further that the way in which all of these elements are connected together doesn't depend on wiring, but on information that is stored in memories. We then would have a circuit that could be made to duplicate the function of any digital system that you can imagine. That is just about what we have in the microprocessor.

Thus, by using a microprocessor, together with peripheral devices, we can develop what amounts to a universal system. What the system actually does will be determined by the program we store in its memories. We can have two systems that have nearly identical wiring diagrams but completely different functions.

The microprocessor approach to digital systems has many advantages. Techniques of programming have been highly developed by the digital computer people, so putting a program into such a system is straightforward. Furthermore, design changes can be made by merely changing the program that is stored in memory, rather than by changing wiring and interconnections in the system. A system based on microprocessor might be said to have a wiring diagram that we can change at will without actually disturbing a single connection. All of the changes are actually made in the 1s and 0s that are stored in the memories.

LIMITATIONS

Although the number of places that microprocessors are used will continue to increase for a long time to come, the microprocessor is not the answer to all digital design problems. Two limitations, at least at the present time, involve the size of the system and the required speed of operation. Many digital systems consist of only a few integrated circuits. Obviously, there is nothing to be gained by replacing such a simple system with a microprocessor.

The other limitation involves the required speed of operation. Most microprocessors are simply not as fast as circuits that use discrete components such as transistors, although this situation could well change through the years. For this reason, in such applications as real-time processing of TV signals, the micro-

processor may well control the system, but the actual signal handling will be done by faster components.

PROBLEM AREAS IN LEARNING

There are several reasons why the microprocessor poses problems for the engineer or technician who is not familiar with it. Although the actual basic functions in such a system are very simple, there are very many steps in even the most elementary operation. This gives the system the illusion of complexity. Furthermore, a microprocessor system usually has several large-scale integrated circuits connected so that several things are happening at almost the same time. Here again, the system appears to be very complex.

We mentioned that one problem facing the newcomer to the field is the fact that there is a whole new vocabulary involved. Until the new terms are understood, the subject will indeed remain confusing.

Two terms that are bandied about in connection with microprocessor systems are *hardware* and *software*. The term, hardware, isn't bad, because it refers to physical devices such as integrated circuits, readouts and interconnecting wires. The term, software, is apt to be more confusing. It is used to describe numbers and programs that have no physical existence, but are stored in hardware such as semiconductor memories. The program or set of instructions that tells the microprocessor what to so is called software.

BASIC ELEMENTS

All programmable digital computers consist of three functional elements as shown in Fig. 16-1. The first element that we will consider is called the memory. As its name implies, the memory stores data. The data that is stored in the memory may be the actual numbers that the system is processing, or it may consist of binary numbers, called instructions, that tell the rest of the system what to do.

In a microprocessor system, the memory is usually a semiconductor device, although systems handling large amounts of data often use magnetic disks to store data. Memories are of two general types. The *R*ead *O*nly *M*emory, or *ROM*, has information stored in it that cannot be easily changed. Therefore, it is used to store instructions that will not change. The *R*andom *A*ccess *M*emory, or *RAM*, would probably better be called a read-write memory because data can be stored in it electrically at any time and

can just as easily be erased. A RAM is used for things that are to be stored temporarily. Thus, the actual numbers the system is working with, as well as instructions that are changed frequently, are stored in RAM.

The next basic element of a computer that we will discuss is the *A*rithmetic *L*ogic *U*nit or ALU. It can perform simple arithmetic operations on binary numbers as well as basic logic functions such as ANDing and ORing. Even the most complex mathematical or control operation can be broken down into very large number of very simple operation. Thus, the ALU can perform any functional operation that is performed in any digital system. The problem is that there are a very large number of operations. Inasmuch as microprocessors can perform these simple steps in less than a microsecond, this isn't much of a problem in most applications.

The next functional element in Fig. 16-1 is called the control unit. It is often called the brain of the computer because it coordinates the operation of the other elements. The instructions that are stored in the memory are deciphered in the control unit which then tells the ALU what to do.

These three functional elements are the essential parts of any digital computer. Of course, we can't do much with it unless we have some way of interfacing it with the real world. This interfacing is accomplished by the *I*nput/*O*utput or I/O devices.

COMPUTER OPERATION

One of the reasons that computer operation tends to be baffling to the newcomer is that all operations are performed by a series of many very simple steps. The steps are so elementary that

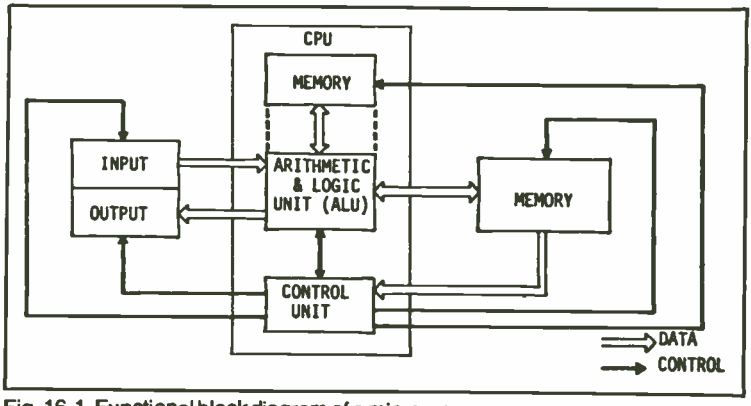


Fig. 16-1. Functional block diagram of a microprocessor.

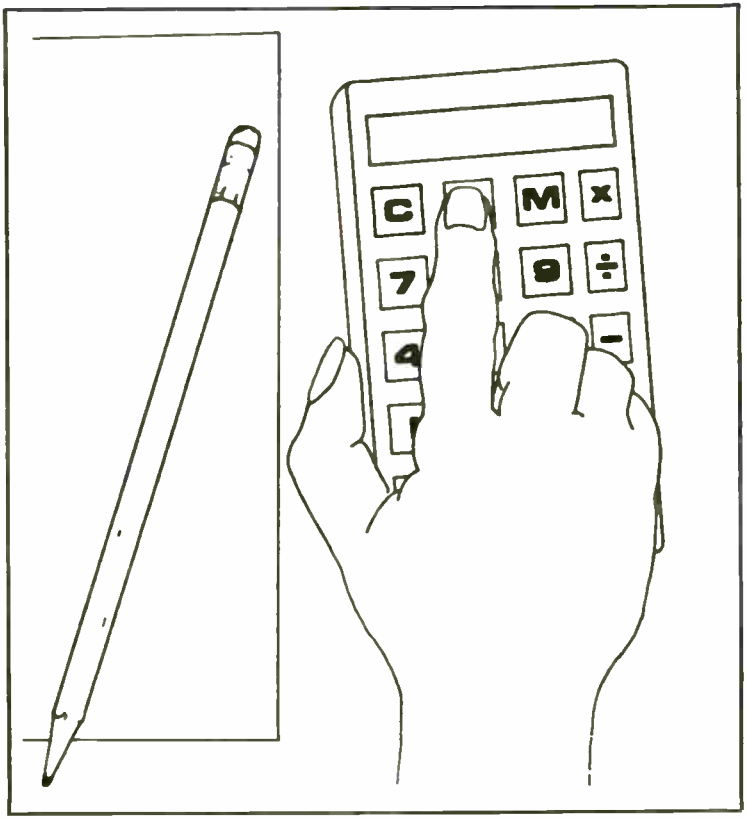


Fig. 16-2. Calculator, pencil and paper analogy of microprocessor operation.

they require a new way of thinking about problems. One way in which to gain a little insight into how a computer works is to use an analogy consisting of a man with a calculator, a pad of paper and a pencil as shown in Fig. 16-2. The man is analogous to the control unit of the computer. The calculator corresponds to the ALU, and the paper and pencil are analogous to the memory circuits.

Let's use this analogy to see how we would write program for adding the decimal number 2 to the decimal number 3. We can perform this operation so simply in our heads that we fail to recognize all of the steps involved in the process.

The first thing that we need is a list of instructions. These instructions would be stored in the memory of a computer, so we will write them on the pad of paper. A computer has no judgement other than what we program into it, so we must be very careful to include all of the steps.

The following list shows the steps necessary to simply add the decimal number 2 to the number 3, using our simple calculator:

1. Press the clear key.
2. Enter the first number into the calculator.
3. Press the "+" key.
4. Enter the second number into the calculator.
5. Press the "=" key.
6. Read the total and write it down

To find any of the above instructions, all we have to do is to go the proper numbered line above. In computer parlance, the number of the line is called the *address* of the instruction. For example, line 3 is the address of the instruction, "Press the + key."

Looking at our list of instructions, we see that something is missing. We haven't stated just which numbers we were going to add together. So we will take another piece of paper and enter:

A	2
B	3

These numbers, 2 and 3, are the numerical data that we are going to work with. To get the number 2, we have to go to line A, so we can say that A is the address of the number 2.

Although we may seem to be complicating the simple situation of adding two numbers together with a calculator, this example will give a considerable amount of insight into how a computer or microprocessor must be programmed to perform what seems like a simple task. From what we have said so far, we can see that a program consists of a series of two types of operations. First we must get, or *fetch*, the instruction or data; then we operate on, or *execute*, these instructions.

Let's continue to write a detailed program for adding the numbers 2 and 3. The steps will look something like this:

1. Fetch the instruction on line 1. This instruction is to press the clear key on the calculator.

2. Execute this instruction. This means to actually press the clear key.

3. Fetch the instruction on line 2. This instruction tells us to enter the first of the two numbers that we wish to add, but it doesn't tell us what the number is. We must have another line of the program to tell us the value of the number.

4. Fetch the number on line A. Now we find that the value of the number is 2.

5. Execute the instruction on line 2. This means to actually press the "2" key on the calculator.

6. Fetch the instruction on line 3. This tells that we must press the + key.

7. Execute the instruction on line 3. Actually press the "+" key.

8. Fetch the instruction on line 4. This tells us to enter the second of the two numbers that we wish to add, but, again, it does not tell us the actual value of the number, so we have the next step.

9. Fetch the number on line B. Now we know that the value of the second of our two numbers is 3 so we can proceed.

10. Execute the instruction on line 4. Actually press the "3" key on the calculator.

11. Fetch the instruction on line 5. This tells us to press the "=" key.

12. Execute the instruction on line 5—actually press the "=" key.

13. Fetch the instruction from line 6. This tells us to read the indication of the readout of the calculator.

14. Execute the instruction from line 6. Actually read the number 5 on the calculator and write it down.

In a computer program, this wouldn't be enough; we would actually have to specify some address at which we would write down the number 5. We might specify that the 5 be written on line C of the second piece of paper.

One thing that is obvious from this simple example is the fact that as human beings we perform many detailed steps of most things that we do almost automatically. Certainly, if we were to add the numbers 2 and 3, we wouldn't consciously go through all of the detailed steps that we have listed above. The computer, however, has no intelligence of its own. It can only do what we program it to do. This is one of the problems that faces the beginner at programming. He often neglects some of the simple steps that enter into an operation so that his program won't work.

In this analogy, the calculator, the human operator and the pencil and paper are hardware. The instructions and the numbers are software. Now we have a reasonably close approximation of a computer program that is used with a microprocessor. It points out the important fact that although a microprocessor operates very fast, time is required for the operation of each step and there are often many steps in an operation. Thus, when performing complex functions, the speed of operation of the microprocessor can be significant.

FLOW CHARTS

One of the questions that frequently arises in connection with the microprocessor is, "Just how can a device, which operates in much the same way as a computer, perform all of the operations that have traditionally been performed with more conventional systems?" The answer isn't simple, but it isn't very complicated either. To use the microprocessor intelligently, the designer must combine the skills of hardware design and programming.

One of the aids in either preparing programs or interpreting them is called a flow chart. The flow chart is to the programmer what the schematic diagram is to the circuit designer.

Several different symbols are used in flow charts, but all you need to know to get a good insight into the matter is the meaning of the three symbols shown in Fig. 16-3. The first symbol that looks like a rectangle with rounded ends marks the beginning or the end of the program or a part of a program. This seems like an unnecessarily simple symbol, but you must always remember that a computer-like device won't so anything unless you tell it to. For example, it will neither start nor stop unless it is properly instructed. These start and stop symbols in the flow chart remind the programmer to put in the proper instructions.

The second flow chart symbol in Fig. 16-3 is a rectangle. It is used to represent one or more instructions that the microprocessor must follow. In a detailed flow chart there will often be a separate rectangle for each instruction. In a more general flow chart a single rectangle may represent a whole set of instructions. Again this is analogous to a circuit diagram. In a detailed diagram, we will have a

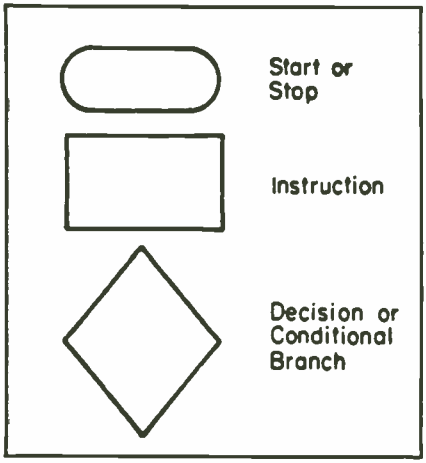


Fig. 16-3. Flow chart symbols.

symbol for each circuit component. In a more general diagram, several components are included in a single block.

The final symbol in Fig. 16-3, the diamond shape, is representative of what distinguishes a software-based system from a hardware-based system. It represents a point in the program where the system must make some sort of decision. The decision is based on something that the system can easily determine. For example, the system may compare two numbers to determine which is larger. If one number is larger, one set of instructions will be followed. If the other number is larger, another set of instructions will be followed. Another type of decision may be based on whether or not a number has been reduced to zero.

Figure 16-4 shows a simple application of a flow chart. Here we have a simplified program for a microprocessor-based system that is used to control the speed of a motor. The first block of the chart is the start instruction. Usually, it will tell the microprocessor to set all of the registers in the system to some known state.

Digital circuits, when they are first turned on, may be in either a high or a low state. In our motor controller the various circuits just might be in a state that will command the motor to go to full speed. We might not want this to happen, so we will provide instructions that will tell the system what state we want it to be in when it is turned on. This process is called *initialization*. The next block tells the system to get a signal that is some function of the speed of the motor we are controlling. It might be a digitized signal from a tachometer. Next, the actual speed of the motor as represented by a binary number is compared with another binary number that represents the desired speed of the motor.

Now, we come to the first point in the program where the system must make a decision. The diamond-shaped symbol labeled, "Is the motor speed correct?" is called a decision point or a conditional branch in the program. What happens next depends on whether the speed of the motor is what we want it to be. If the answer is yes, nothing will happen and the program will revert to the first step and again check the speed of the motor. If the answer is no, then another decision must be made. The system must determine whether the speed is to high or too low. In either case, the system is programmed to initiate corrective action.

Of course, our flow chart is simplified in that we haven't shown any of the detailed instructions that must be supplied to the microprocessor. Nevertheless, it will give a good overall view of what is going on in the system.

One thing that isn't obvious from our flow chart is the fact that the microprocessor can perform all of the steps required to control the speed on a motor with plenty of time to spare. In practical terms this means that we might use the same system to control several other things. For example, we might start our program once every several hundred milliseconds. In the interim period we may use the same system to execute a completely different program that controls something else.

SIMULATING LOGIC ELEMENTS

There is another way in which we can gain a little insight into the way that a computer-type of system can duplicate the function

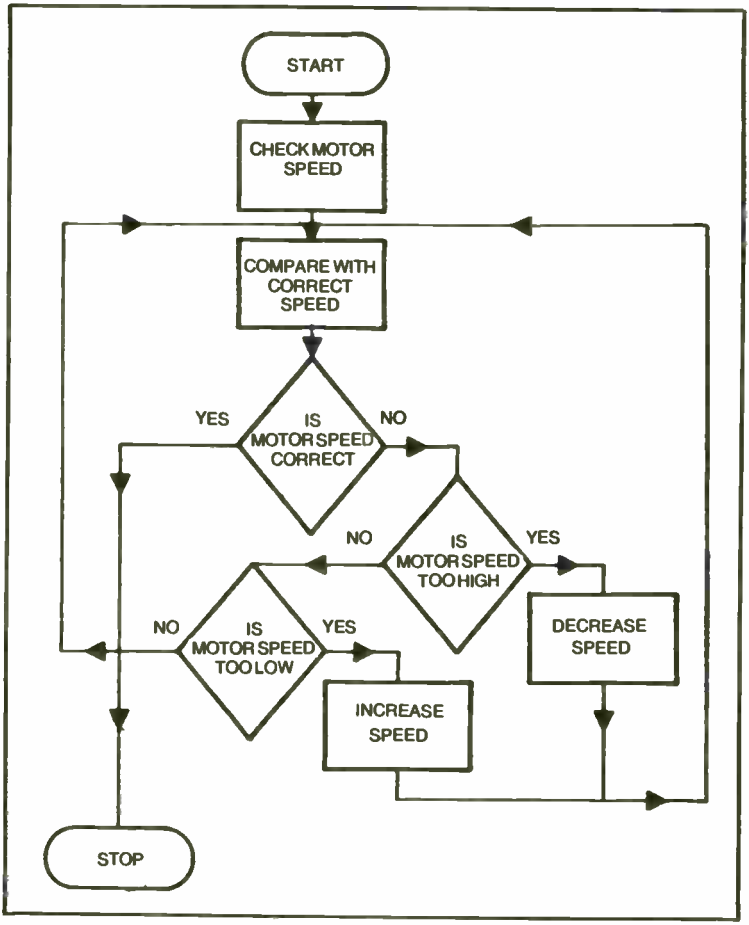


Fig. 16-4. Flow chart of a motor control program.

of another digital system. Earlier in this book we discussed the fact that NAND gates can be combined to perform any type of digital function. By cross coupling NAND gates we can make flip-flops, and with gates and flip-flops we can make any conceivable type of digital system if we have enough of them. Figure 16-5 shows the symbol for a NAND gate, together with its truth table. Figure 16-6 shows a microprocessor system with three external connections. There are two inputs labeled A and B, and one output connection.

Figure 16-7 shows a flow chart of a program that will enable our system to duplicate the function of a NAND gate. The first step in the program is to determine whether input A is high or low. If this input is low, the situation corresponds to one of the top two lines of the truth table of Fig. 16-5. In either case, the output will be high. If input A happens to be high, that is not low, the situation corresponds to the third or fourth lines of the truth table. The state of the output will be either high or low, depending on the state of input B. Thus, we next make a decision on the basis of this input. By comparing the flow chart with the truth table, we see that they both accomplish the same thing. Thus, we have shown two things: First, that we can use a flow chart to do the same thing that we can do with a logic truth table, and secondly, if we can build any imaginable type of digital system out of NAND gates, then we can also build any imaginable type of digital system using a microprocessor that will follow the program of Fig. 16-7.

SYSTEM FLEXIBILITY

One of the advantages of the microprocessor-based system is the fact that it is very flexible and can be adapted to many changing conditions. For example, suppose that we had a conventional digital system using gates and flip-flops that was designed to test a particular type of integrated circuit automatically. The system might check all of the parameters of the integrated circuit to be sure that they were within prescribed limits.

Now suppose that we wished to test a different type of integrated circuit. The pin connections might be different and the parameters might not be the same. To adapt a conventional system to the new IC would require a great deal of rewiring and possibly some redesign.

If, on the other hand, our system were based on a microprocessor, we could accommodate changes in both the pin connections and circuit parameters by merely changing the program that operates the system. It wouldn't be necessary to change a single soldered connection.

MACHINE LANGUAGE

So far, all of the instructions that we have talked about have been expressed in words. Obviously, we can't place words in English on the actual wires that connect to a microprocessor. The actual instruction that goes into a microprocessor is a binary number; that is, a set of high and low signals. The most popular microprocessors have a data bus and instructions that are one byte, or eight bits long. Thus, a particular instruction that would be placed on the 8-bit data bus might be 1001 0010. This set of binary signals might, for example, tell the microprocessor to add one binary number to another. As we get into discussions about computer languages, it is important to remember that all of the data in a system is stored in binary form, and binary numbers are the only language that a microprocessor understands. The signals at the various pins of the microprocessor are either high or low, and thus are binary digital signals. Any talk about languages other than binary numbers pertains to something that is done for the convenience of the human operator of the system. When the signals get into the system they are binary signals.

Looking at the binary signal that we mentioned above, 1001 0010, it is easy to see that regardless of how well the processor can handle it, a number like this isn't particularly well suited for use by human beings. Each microprocessor has what is called an instruc-

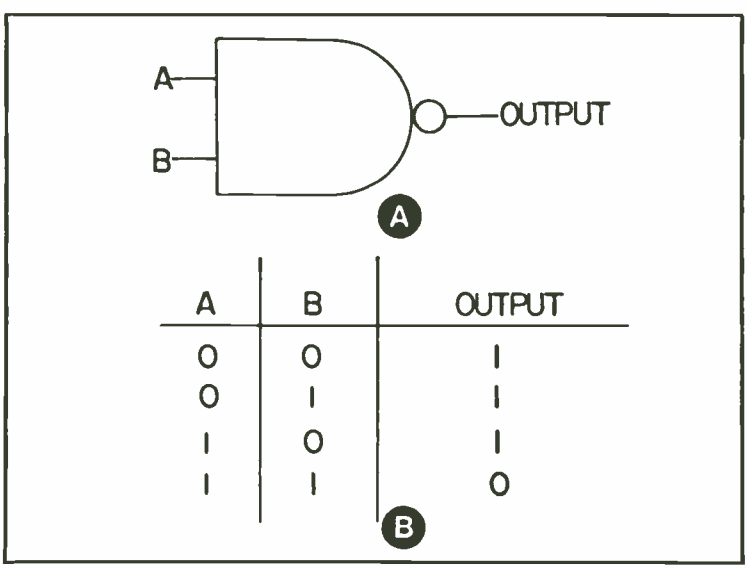


Fig. 16-5. NAND gate and its truth table (B).

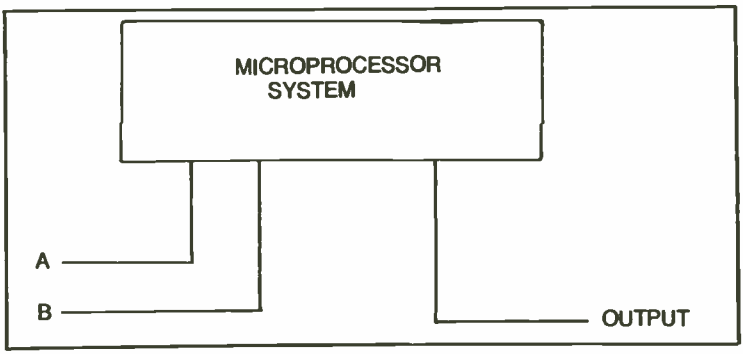


Fig. 16-6. Microprocessor system set up to simulate a NAND gate.

tion set. This is a set of binary numbers that will tell it to do different things. Most of these instructions consist of eight-bits. For example, the instruction 1001 0010 might tell the microprocessor to add two numbers together, and 1010 0101 might tell it to move the number from one location to another. Numbers of this type are not easy to remember and for that matter aren't particularly easy to talk about. For the convenience of human beings who might be associated with such systems, we need some sort of shortcut that will make the instructions easier to handle.

One solution to the problem is to express the various binary numbers in some other number system that has more symbols and is thus easier for a human being to handle. The first thing that comes to mind is that we might use something like the binary-coded decimal system that we learned about earlier in the book. With this system we could express all of the 8-bit binary words in the familiar decimal numbers. The idea has a serious limitation in that the BCD system isn't very efficient. We couldn't express every 8-bit binary number with two decimal numbers. For example, the binary number 1001 0010 could be expressed as 9 2 using binary-coded decimal, but the number 1001 1111 would be 9 15. Thus we would have difficulty trying to use the system.

The next best bet, and the one that is nearly universally used, is the hexadecimal number system. This is a numbering system having the base 16. Figure 16-8 shows a comparison of the hexadecimal, binary and decimal numbering systems. We can see that from 0 to 9, the hexadecimal system is the same as our familiar decimal system. Above 9 we have six new symbols, the first six letters of the alphabet. Before we go any further, it is obvious that it is much easier to work with a hexadecimal number such as 1A, than to work with its binary counterpart 0001 1010.

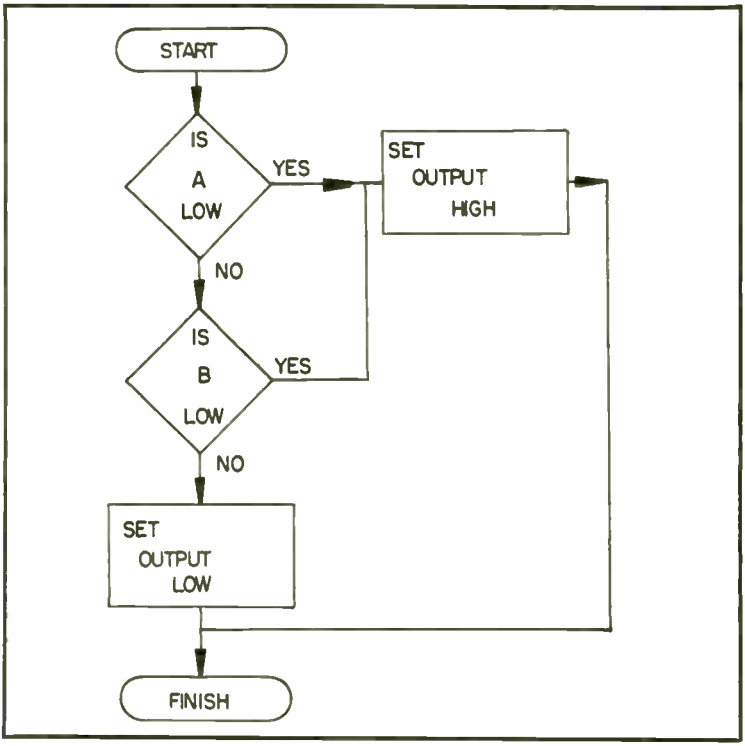


Fig. 16-7. Flow chart of the program to simulate the operation of a NAND gate.

There are other advantages to using the hexadecimal number system. One advantage is that it is very easy to convert between binary and hexadecimal numbers. First, suppose that we have a number in binary form. The first step is to separate the number into groups of four digits each. We usually do this anyway just to make the numbers easier to read and write. Then, we find the decimal value of each group of four digits. Finally, we assign a hexadecimal number for each group of four digits. The process is shown in Fig. 16-9.

Now let's go the other way. Suppose we have a hexadecimal number and we want to express it in binary form. All, we have to do is to assign a 4-bit binary number to each digit of the hexadecimal number. This is shown in Fig. 16-10.

Using the hexadecimal number system, it is common to see the instructions for a microprocessor written as F4, B2, or 11. These hexadecimal numbers are much easier to work with than their binary counterparts, but we must remember that the actual signals are in binary form.

DECIMAL NUMBER	BINARY NUMBER	HEXADECIMAL NUMBER
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

Fig. 16-8. Comparison of decimal, binary, and hexadecimal representations of a number.

WHAT IS AN INSTRUCTION SET?

In the preceding pages we have made reference to the instruction set of the microprocessor. The entire instruction set is rather lengthy and complex, but we can take at least a superficial look at it. An instruction is a digital signal, usually eight bits long, that when fed into the instruction register of a mpcroprocessor will

01001010	=74 DECIMAL	=4A HEXADECIMAL
1. BREAK BINARY NUMBER INTO GROUPS OF FOUR DIGITS	0100	1010
2. ASSIGN HEXADECIMAL NUMBER TO EACH GROUP	0100 = 4	1010 = A
3. WRITE HEXADECIMAL NUMBER	4A	

Fig. 16-9. These calculations show how to convert a binary number to a hexadecimal number.

instruct the ALU to do something useful. For example, the instruction 1000 0101 might make the ALU move the data that is stored in the accumulator to another register.

Inasmuch as the microprocessor can do just about anything that any other electronic system can do, we might expect that there are several different types of instructions in a typical instruction set. Indeed there are, but at first glance, the exact instructions tend to be a little confusing. We can understand what they do, but the immediate reaction is apt to be the question as to why anyone would want to do anything like that. The reason for this confusion is that the microprocessor does things in extremely small elementary steps. The fact that it operates at high speed is the only thing that makes it practical to accomplish things in this way. As an example, it usually requires more than ten instructions to simply add two numbers together.

To accomplish tasks with a microprocessor, we must break the task down into very elementary steps. At first this requires a different way of thinking. Later, we will see that there are things called "higher level languages" that will simplify the task considerably.

To make the instruction set a little easier to follow, we can somewhat arbitrarily break the instructions down into five categories:

☐ **Arithmetic Instructions.** These instructions cause binary numbers stored in various parts of the system to be operated on mathematically. For example, the contents of the accumulator might be added to the number stored in a given memory location. Note that addition is about the most complex operation that is performed by a single instruction. To do such things as subtract,

2B HEXADECIMAL	= 00101011 BINARY = 33 DECIMAL	
1. SEPARATE HEXADECIMAL DIGITS	2	B
2. ASSIGN A 4-BIT BINARY NUMBER TO EACH HEXADECIMAL NUMBER	2=0010	B = 1011
3. WRITE BINARY NUMBER	00101011	

Fig. 16-10. These calculations will convert a hexadecimal number to a binary number.

multiply, or divide, we need to have several sequential instructions. This emphasizes the fact that things are done in very simple steps.

☐ **Logic Instructions.** These instructions cause some logic operation to be performed on the numbers in the system. Typical logic instructions cause ANDing, ORing, and NOTing.

☐ **Moving Instructions.** These are sort of housekeeping instructions. They cause data to be moved in some way in the system. In most operations it is necessary to move data in and out of the accumulator. Multiplication is performed by combining addition with shifting the data to the right or left. Also included in this category are instructions that cause data to be stored in various memory locations. This operation may tend to be a little confusing, because the storing is done in a slightly unconventional way. For example, we might store the contents of the accumulator in a given memory location. After the operation is complete, the data will indeed be in the specified memory location, but it may also still be in the accumulator. This feature often causes many problems to beginners.

☐ **Jump or Branch Instructions.** These instructions define operations which make the microprocessor such a powerful tool. Instructions can cause something to be done only under certain conditions. This is the same as saying that the system makes a decision. For example, the system may be programmed to perform a series of instructions, but if certain conditions prevail to jump to another instruction. Typical conditional jumps are based on such things as whether the binary number resulting from an operation happens to be greater than, equal to, or less than zero. This conditional jumping is equivalent to being able to change the wiring diagram of a system under certain conditions.

Other jump instructions are unconditional. That is, the system will jump to a different instruction under any condition. This adds flexibility to programming. A typical use of an unconditional jump is to cause the system to jump to a "subroutine" that performs some functional operation that may be used many times. Thus, in any part of the program where this operation is needed, the program can jump to that location. In this way the instructions for the functional operation need to only be prepared once.

☐ **Input-Output Instructions.** These instructions cause the data to be taken from or delivered to registers in the I/O devices of the system.

Admittedly, this breakdown of instructions into categories is somewhat arbitrary, but it does tend to take some of the complexity out of an instruction set.

HIGHER LEVEL LANGUAGES

In order to make a microprocessor perform its various tasks, we must select the proper instructions from the instruction set and store them sequentially in a memory. When the system starts to operate, it will access the memory and fetch the instructions in the proper order.

The first thing that is obvious in connection with this is that it is a very difficult chore to actually store these instructions in the memory in binary form. The processor speaks binary, but the human being doesn't. It isn't easy to keep track of hundreds of instructions of the form 1100 0110. The fact that binary numbers are unfamiliar makes it very difficult to memorize them, even for a few seconds. We have to handle them almost on a bit by bit basis.

The hexadecimal number system which we mentioned earlier is a good compromise. It is easier to handle than binary, but it is also easy to convert to binary form, which must be done before it is actually entered into the system. This conversion can easily be done by the system itself.

A typical small microcomputer might include a keyboard, something like that of a calculator, for entering data. The keys are coded in hexadecimal, but when the key is pressed, the corresponding binary number will be generated in the system. For example, if key "A" is pressed, the binary number 1010 will be entered into the system. Note that the hexadecimal character A and the binary number 1010 are the same. They are both equivalent to 10 in our decimal system.

It is easy to see that the hexadecimal system is easier to handle than the binary system. It is much easier to enter a number such as 5F into a keyboard than the binary equivalent 0101 1111. The whole business is made still easier by the fact that if an instruction is listed in a book in hexadecimal form, we don't have to know either its binary or decimal equivalent. For example, if an instruction is listed as 4B in hexadecimal, that is all we have to know. We can simply enter 4B into our keyboard without knowing any more about it. It is only when we are actually looking at one of the buses of the system in troubleshooting that we have to get back to 1's and 0's.

An instruction set expressed in hexadecimal or even binary notation is said to be stated in machine language. This is because

the instructions are in a form that the system actually uses. Programming a system in this notation is said to be programming in machine language.

The principal limitation of an instruction set expressed in hexadecimal notation is the fact that the notation has no relation to the nature of the instruction. For example, suppose the hexadecimal code B6 is an instruction that tells the system to branch or jump to a subroutine. There is no way that we could recognize this by looking at the number B6. This brings to the first level of language above the simple machine language that we have been considering. This is called assembly language.

It would be very nice if we had an arrangement like that shown in Fig. 16-11. Here the input is a series of digital signals from a keyboard. They might, for example, be in the ASCII code. The output is the actual binary representation of the instruction. To use our same example, suppose we wanted to enter the instruction telling the system to branch to a subroutine into a memory location. We would simply type BSR into our keyboard and out of the box in Fig. 16-11 we would get the binary number 1011 0110, which is the hexadecimal number B6 which we saw represented this instruction.

The advantages of the box of Fig. 16-11 are so great that it would seem worthwhile to build one into every system. With the box, we could enter what is called assembly language where each instruction is represented by a mnemonic such as LDAA for load accumulator A, or INC for increment. This certainly makes programming much easier than it is using even hexadecimal notation. The entire instruction set can be memorized rather easily.

Fortunately, it isn't necessary to build the box shown in Fig. 16-11 at all. The microcomputer can do the same job that the black box can do. A program can be written that will accept ASCII characters from a keyboard and produce binary numbers corresponding to the mnemonic. This program can be prepared on another computer, and once it is prepared, it can be stored in the memory of the system that will use it.

Note that when we begin to use a higher level language, such as assembly language, we tend to be getting away from what is actually happening in our system in digital form. This is why it is a good idea to get a little experience working with machine language before graduating to a higher level language. Many engineers and technicians who have never actually used machine language tend to have a superficial knowledge of the systems with which they work.

Once a system is programmed to accept assembly language and translate it into machine language, we can use this feature to make programming even easier. For example, suppose we have to write a part of a program that consists of 15 steps and we find that we will want to use this same set of 15 instructions many times in the programs that we are writing. We can write this set of instructions into the system and assign a mnemonic to it. Then every time we enter the mnemonic, the 15 instructions will be written. This is often called *macroprogramming* and the resulting mnemonic is called a *macroinstruction*.

All of the languages that we have described so far have the disadvantage that the problem or task must be reduced to very elementary steps before the program can be written. This disadvantage can be overcome by using still higher level languages. Before discussing any of them we want to emphasize again that the microcomputer itself speaks only binary and must be actually programmed in very elementary steps. If we use a technique whereby we can program the system in an easier way, the translation to the elementary binary steps must be accomplished somewhere. This is usually done in the system itself and the program that tells the system how to do it is stored in memory. Thus, we see that the price that we pay for something that makes life simple for us is that we need memory locations to store the program that does the translation.

There is a way out of this that is practical if we are going to program a lot of systems. We can use a completely different system that translates things into the binary instructions. These are then entered directly into the system in question by some automatic means. While entering them we don't have to think, so the task is easier.

Getting back to the need for higher level languages, suppose we have a microprocessor system in a broadcast station and in order to perform its assigned task it must calculate the sine of an angle. You can just imagine that the amount of shifting and adding

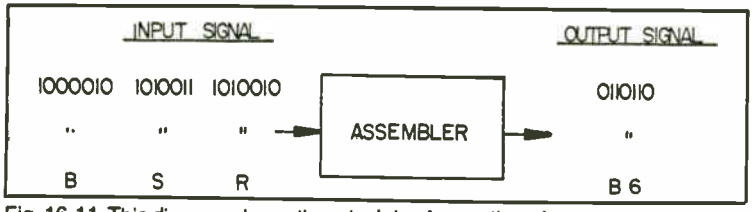


Fig. 16-11. This diagram shows the principle of operation of an assembler.

that would have to be done repeatedly to get sine of an angle with any accuracy at all. Here's where a higher level language like BASIC comes in handy. If the system includes a program that will translate BASIC, all we have to do is to type in SIN. This will automatically call up all of the necessary instructions to perform the required computation.

BASIC is the simplest of the higher languages. There are many more and each type has advantages for certain types of programs. FORTRAN is used in computers that are required to solve engineering type problems and COBOL is well suited to business type problems.

The actual program that translates a high-level language into machine language is called a *compiler*. This program looks at everything that is typed into the system and assigns memory locations for everything as well as generating the actual machine code in binary form. The result is that the final program that comes out of the compiler often is no where as efficient as a program written in machine code by a highly skilled programmer would be. This is usually only important in small systems, or in systems that are going to be produced in large quantities. Semiconductor memories are quite economical and it is often worth spending a little more for additional memory to avoid the difficulty of programming in machine language.

Another instance where the inefficiency of high-level languages is significant is where great speed is required. Home computer enthusiasts have noticed this, particularly when generating graphics. A graphic display generated in a higher level language will usually be much slower than the same thing done in machine language.

A CLOSER LOOK AT THE MICROPROCESSOR

With the background that we have acquired so far in this chapter, we can take a closer look at the microprocessor and get a little better understanding of how it works. We must do this in a general way for two reasons. First, the various functional elements of the microprocessor work so closely together that it is often hard to tell where one unit leaves off and another begins. Secondly, microprocessors are not alike. In the older units, the various functional elements were distributed between two or more integrated circuits. Some of the newer units, appropriately called "single chip" units, have all of the necessary functional units in one package.

All of the functional elements of the microprocessor are important, but we can consider the heart of the unit to be the Arithmetic Logic Unit, or ALU. This is the element that performs mathematical operations such as addition and subtraction, logic operations such as ANDing and ORing, and shifting of data to the right or to the left.

The ALU gets its inputs from and supplies its outputs to at least two registers. One of these is called the accumulator, usually abbreviated to ACC. This register is very similar to the readout of a pocket calculator. Much of the data that passes into and out of the ALU passes through the accumulator. The other data connection is usually to another data register such as a memory data register.

In both of these registers, the data may pass in either direction. For example, if two numbers are to be added together, one of them may be stored in the accumulator before the addition, and the sum may be stored in the accumulator after the addition. This use of the same connections to carry data in either direction is one of the things that make it possible to get all of the functional elements of a microprocessor into a single integrated circuit.

The control signals, that is the signals that actually tell the accumulator what to do, come from the control circuitry. The actual function of the control circuitry is to take two inputs—an instruction and pulses from a clock—and decode them to provide signals that will make the ALU perform the desired function. The arrangements of the actual circuits is rather complex and differs from one microprocessor to another. The ALU and its control circuits are so closely interrelated that the combination of the two functional elements is often referred to as the Central Processing Unit, or CPU.

The arrangement of the actual control circuits, and the ALU give rise to what is called the instruction set of the microprocessor. The instruction set is simply a set of binary digits that when applied to the microprocessor will make it perform certain functional operations. These operations are of the form of alternate fetch and execute operations as we saw earlier.

In Fig. 16-12 we show the ALU and control circuits connected to an instruction register, called the IR, and an instruction decoder. The instruction register gets its input from the data bus which is connected to external circuits through the data bus. What it actually gets in an 8-bit microprocessor is either high or low signals on each of eight data lines that make up the data bus. For example, in a typical microprocessor an instruction might be called *load*

accumulator. This instruction makes the inputs to the instruction register 1000 0110. When this particular combination of high and low signals reaches the instruction register, it will be decoded in the instruction decoder and eventually the signals that go to the ALU will cause it to perform the desired function of actually loading the accumulator. There are usually several different forms of an instruction such as this that will perform the operation in different ways. We'll have more to say about this later.

Getting back to our microprocessor, we can now add a couple more blocks to our diagram. These blocks contain memory. The read-only memory, or ROM, has things stored in it permanently. These are things that won't change, so they are actually built into the memory. The random-access memory, or RAM, on the other hand can be either loaded into or read out of. This is usually called "writing into" the memory, or reading out of it.

As shown in the figure, there are three sets of connections to a memory element. First there is the address bus. This is a set of wires, usually 16 in an 8-bit microprocessor, that are used to designate a particular cell in the memory. Then there is the data bus, eight wires in an 8-bit microprocessor, that will either write into or read out of the particular memory cell. Finally, there are control connections. For example a RAM has a connection that tells it whether it should accept data from the data bus or put data on the data bus.

Each combination of high and low signals on the input address of the memory specifies a certain location. In most 8-bit systems, this location is eight-bits wide. The eight bits in this location are the data that is stored in the memory. In a RAM, we may enter data into each of these locations and read it out. In a RAM, we can only read out.

WHAT IS A FLAG?

In looking at the specification sheets covering microprocessors and microcomputers, we often run across the term "flag." Usually what is called a flag is actually a flip-flop, or one stage of a register that is buried deep inside the microprocessor. Sometimes the flags are stages of a register that is called something like a "condition register." The purpose of the condition register is to keep track of what is a happening inside the microprocessor. At first it is confusing to learn that there is no direct access to these register stages from outside the device. None of the stages of such a register are connected to the pins of the device.

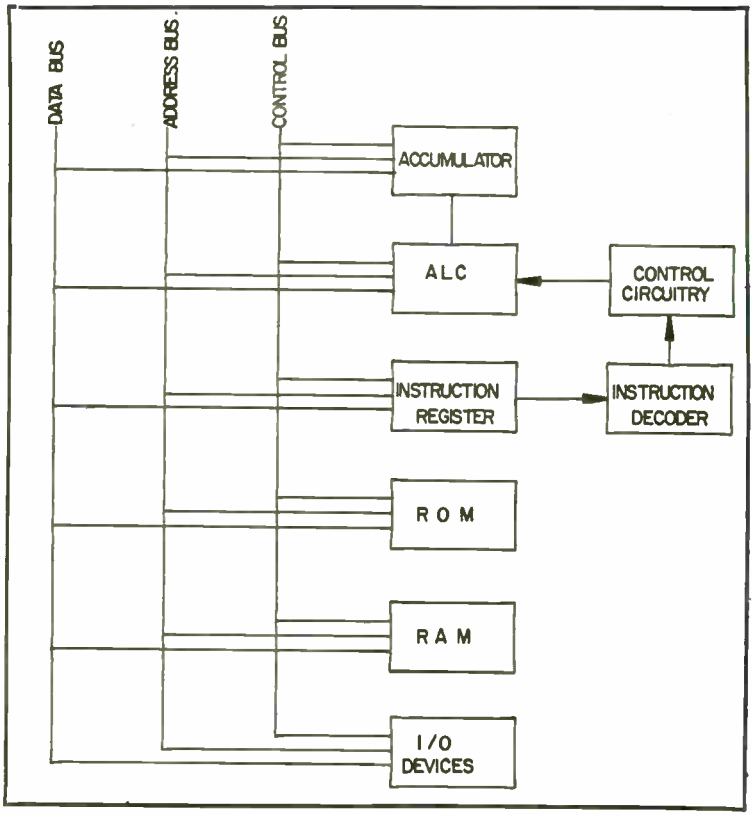


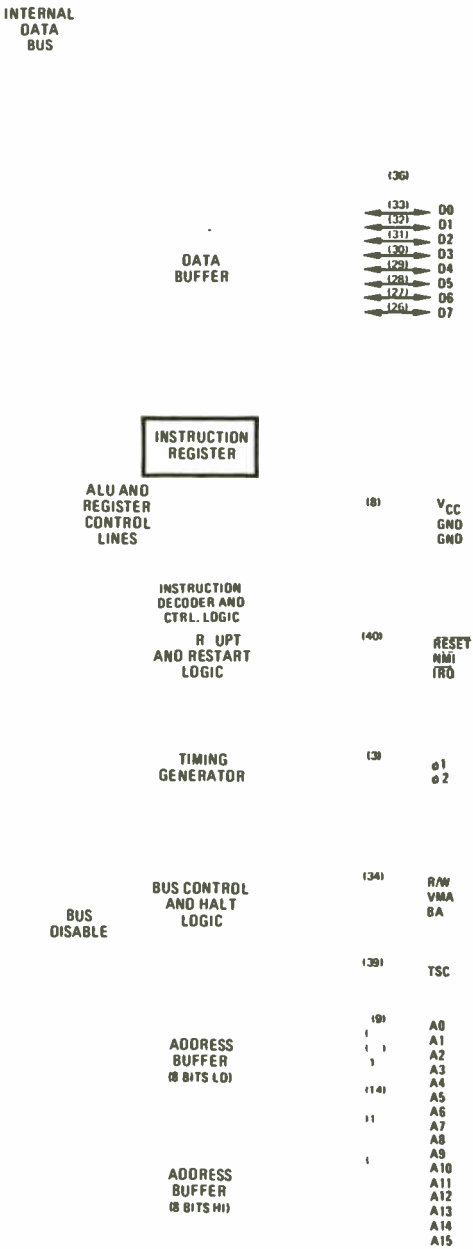
Fig. 16-12. Block diagram showing organization of components in a microprocessor.

A typical flag might be called an overflow flag. This flip-flop is set whenever the operation of the device causes the contents of the accumulator to overflow, that is, when an operation tries to cram nine bits into an 8-bit register. Similar flags are provided to indicate when the contents of a register go to zero. Inasmuch as we can't look at a pin on the device to determine the state of the various flags, they must be handled by programming. There are instructions that can be incorporated into a program to test the state of a flag, or even to set or reset a flag.

PUTTING IT ALL TOGETHER

Figure 16-13 shows a block diagram of a Type 6800 microprocessor. Each of the blocks in the diagram can be thought of as a register of some type. The numbered leads at the right of the figure correspond to the numbered pins of the device. Pins that are input

Fig. 16-13. Block diagram of the type 6800 micrprocessor (courtesy of American Microsystems, Inc.).



connections show arrows pointing to the device, whereas output pins show arrows pointing outward. Note that the arrows on the data bus point in both directions. This is because these pins serve both as input and output connections. Data is brought into the device and is also fed out on the same connections.

Essentially the device contains the ALU, two 8-bit accumulators, one condition code register, and three other 16-bit registers that are used for address storage (the index register, the stack pointer, and the program counter). All of these registers can be accessed in some way by programming.

There are also other registers that are not accessible by programming. These include the 16-bit address incrementer/decrementer, an 8-bit temporary register, and an 8-bit instruction register.

Other functional blocks in the diagram include the instruction-decoding ROM, cycle control logic, interrupt and restart logic, bus control and halt logic, and timing generator.

Inside the device the various instructions are carried out in incremental time periods, called processor cycles. Each cycle consists of one phase 1 clock period and one phase 2 clock period. A 2-phase clock has two outputs which do not go high at, the same time, as shown in Fig. 16-14. If the clock is operating at 1 mHz, each processor cycle is one microsecond long. It takes a minimum of two clock cycles to carry out a single instruction. Some instructions take more than two processor cycles.

In a typical cycle, during phase 1 of the clock signal, the processor puts an address on the address bus. This address effectively connects a memory location or an I/O register to the data bus. Then, during phase 2 of the clock, the data or instructions are fetched and loaded into an internal register. During phase 1 of the next processor cycle, internal operations are performed to execute the instruction. Thus, the operation generally consists of a series of fetch-and-execute operations.

Frequently, the microprocessor is doing more than one thing at the same time. For example, the device may be fetching data from the data bus while at the same time the ALU is performing some operation on some data that has been fetched earlier.

The actual addressing, or putting of the required addresses on the address bus, is a rather complex operation and is beyond the scope of this book. It is sufficient to note that there are several different ways that this can be done and, in fact, some of the major differences between various types of microprocessors lie in the way that addressing is handled.

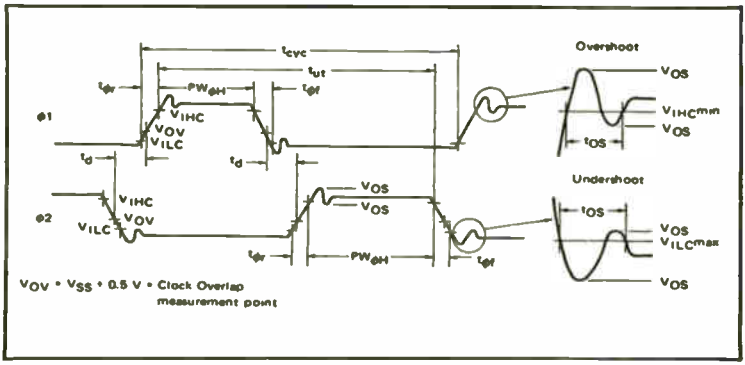


Fig. 16-14. Dual-phase microprocessor clock signal waveforms (courtesy of Motorola Semiconductor Products, Inc.).

Without getting into much detail, let's take a look at the address bus. Note that it handles 16 bits. This means that it can carry over 65,000 different high and low signal patterns and each of these corresponds to the address of something. A given system may not use all of these address locations, but they are available. For convenience in operation, the 16-bit bus is considered as two 8-bit buses, one corresponding to the eight least significant bits and the other to the most significant eight bits. In this way, when the entire 16 bits are not necessary for an operation, the least significant bits can be manipulated separately.

Normally the address on the address bus is generated by the program counter. Thus, for example, this counter can be set to zero and the program will start with what is stored at location 0000 0000 0000 0000. As the program proceeds, the program counter will increment, addressing the next instruction.

The data bus is the port through which all data and instructions pass in and out of the microprocessor. Note that this is a tri-state bus. If the data on the data bus happen to be an instruction, the information is fed to the instruction register. From there it goes to the instruction decode circuitry, which generates the necessary signals to tell the device to carry out the instruction in following processor cycles. Incoming data is fed directly to the data bus, and outgoing data is applied to the bus through the data buffer.

The registers of the device that are used in programming are often shown in block form as in Fig. 16-15. Note that this particular microprocessor has two accumulators. These are registers used in connection with the ALU to hold data while an operation is being performed. Each accumulator is much like the readout of a pocket calculator.

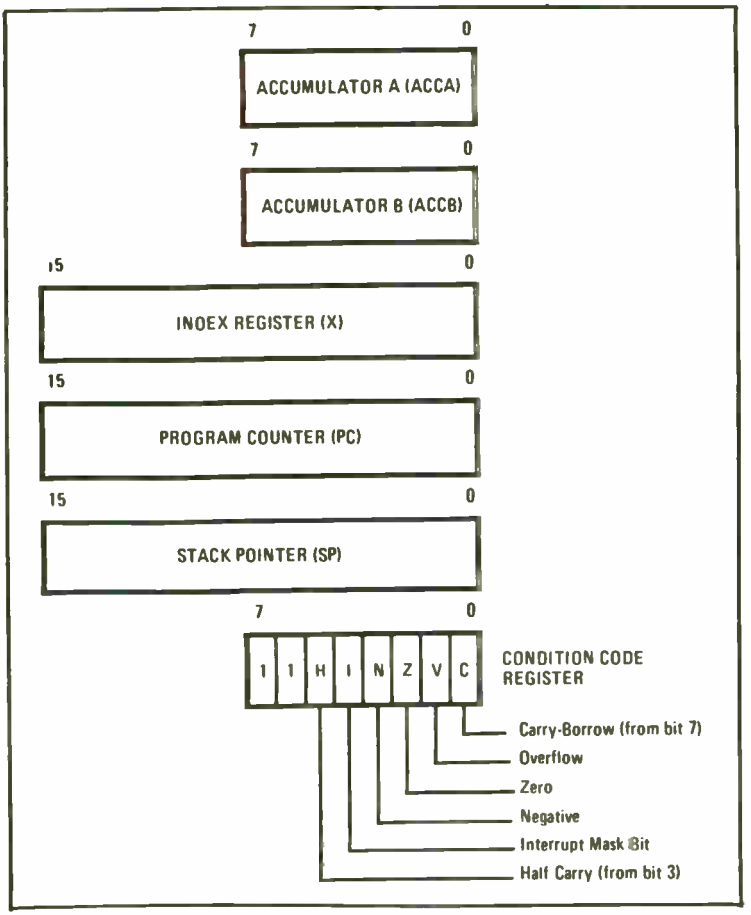


Fig. 16-15. Registers used in programming the type 6800 microprocessor (courtesy of American Microsystems, Inc.).

The index register is used to store addresses for other forms of addressing. This register adds a great deal of flexibility to the problem of addressing, and, when its use is mastered, many rather elaborate things can be accomplished in the programming.

We have already mentioned the program counter. It is used in connection with the incrementer/decrementer to develop the current address that goes on the address bus.

The stack pointer is a register that keeps track of a portion of the memory that is called a stack. This is much like the stack used in some pocket calculators. Data can be entered into a stack in order and recalled in the same order. The stack is often used in

connection with the interrrupt feature. Suppose the processor is performing some routine operation when an interrupt signal is received, telling the device that some more important operation must be performed. The processor will stop what it is doing so that it can attend to the more important chore. But before it does this, it will store enough of the data that it's working on in the stack so that when the interrupt operation is complete, it can go back to what it was doing without losing track of anything. The stack pointer carries the address at which things were stored. This operation is usually described by saying that the stack pointer "points" to the location that is to be remembered.

The condition code register was mentioned earlier. It contains the various flags that are required to keep track of internal operations. When an interrupt is received, the contents of the condition register are stored in the stack.

WHAT DOES IT LOOK LIKE FROM THE OUTSIDE?

In the past few pages we have been discussing the various functional elements that are found inside a microprocessor. Although a knowledge of these elements is a great help in understanding how a microprocessor works, we can't directly get at many of these elements, In a practical situation we are more interested in what happens at the pins of a microprocessor than in exactly what is inside.

Figure 16-16 shows the pin diagram of a type 6800 microprocessor. At first it appears formidable, because it has 40 pins

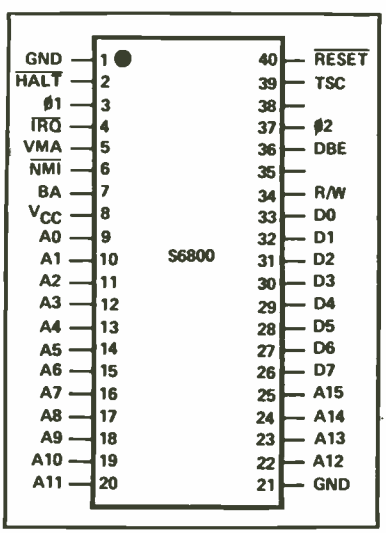


Fig. 16-16. Pin connections, type 6800 microprocessor (courtesy of American Microsystems, Inc.).

which aren't particularly familiar to a beginner in the field. Many of these pins work together, however, and can be considered as one input or output.

Pins 1 and 21 are ground connections, and pin 8, labeled V_{CC}, is the positive power supply connection.

Clock connections. Pins 3 and 37 are clock connections. The Type 6800 uses an external 2-phase clock signal which is applied to these two pins. When we say that we have a 2-phase clock signal, we mean that it consists of two pulse trains that do not go high at the same time as shown in Fig. 16-14.

The address bus. Pins 9 through 20 and 22 through 25 make up the address bus. It consists of 16 address lines that carry a digital signal that specifies the address to which data is to be sent or from which data is to be received. These are output pins because the address signal is generated inside the microprocessor. By using 16 lines, 65,536 different address locations can be specified. These 16 lines can be though of as a single output port of the microprocessor.

The data bus. Pins 26 through 33 are called the data bus. These pins tend to be somewhat unfamiliar because they can be used as either input or output pins. These eight lines carry the actual data and instructions used by the system.

Control functions. There are two general ways in which the action of the microprocessor can be controlled. One way is by applying signals to some of the pins that have various control functions. The other way is to feed digital signals which will be interpreted as instructions to the data bus.

Pin 39, labeled TSC, is the trip-state control for the address bus. This microprocessor uses tri-state gates on several of its connections. When pin 39 is brought to a high logic level, all of the address lines which are outputs will go to a high impedance state in which they are neither high nor low. As explained in the chapter on logic gates, this permits connecting other outputs in parallel with the address pins. Since the microprocessor is a dynamic device, it can only be held in this state for 5.0 microseconds, but this is enough for many purposes.

Pin 36, labeled DBE, is the tri-state control for the data bus. When it is high, the internal drivers are connected to the data pins. When this pin is low, the internal drivers will be disconnected and the unit can read data from the data bus. This pin is normally operated from phase 2 of the clock signal, but it can be externally controlled.

Pin 2, labeled $\overline{\text{HALT}}$, stops everything in the microprocessor. It is used to stop the microprocessor after a single instruction or whenever it is desired to stop the device.

Pin 4, labeled $\overline{\text{IRQ}}$, is the interrupt request pin. It is an input that will stop the microprocessor after it has executed the present instruction so that it can be used to perform some other function. One of the attractive features of a microprocessor is that it can be performing some routine job and be interrupted to do something else, then return to the original job. Usually, the device operates so fast that an observer would never realize that an interrupt had occured.

Pin 6, labeled $\overline{\text{NMI}}$, is also an interrupt request pin. The microprocessor can be programmed to have priorities for the various things that it does. Thus, it can mask out an interrupt signal applied to pin 4 under some conditions. The interrupt to pin 6 can't be masked out, hence it is called a *non-maskable interrupt*.

Housekeeping outputs. As we noted earlier, many of the internal elements of the microprocessor are not connected to any of the pins. We can only control them by means of instructions that we provide to the data bus. In order to do this, however, our system needs to know the state of many things inside the microprocessor. Outputs are provided to give this information.

Pin 34, labeled R/W, carries an output signal that tells whether the microprocessor in is in a state where it will accept or read data, or whether it is in a state where it will give out, or write data. A high signal on this pin indicates the read state, whereas a low level indicates a write state.

Pin 7, labeled BA, is normally in a low state. However if the HALT input is in a low state, or if the microprocessor is in a WAIT state as the result of instruction, this pin will go high, indicating that the address bus is available (BA).

THE MICROPROCESSOR IN A SYSTEM

Much of the terminology used in connection with microprocessors is used rather loosely. We have noted that the microprocessor contains at least the central processing unit (CPU) of a computer. Some include much more. Nevertheless, it is common to speak of the microprocessor integrated circuit as a CPU. When the microprocessor is mounted on a printed-circuit board along with other components, the entire arrangement is often called a microcomputer.

From what we have said about the microprocessor so far, it is obvious that if we are to do anything worthwhile with it, we must

connect some other components to it. The means of making these connections are through the buses as shown in Fig. 16-17. The address bus consists of the 16 leads that are connected to the address pins. Similarly, the data bus is made up of the eight leads connected to the data pins. The leads that are connected to all of the other pins such as the $\overline{\text{HALT}}$, R/W, and BA pins are grouped together and rather loosely called the control bus. The power supply connections are straightforward and are not shown.

One of the first things that comes to mind as a part of the system is something that will let us connect it to the real world. Devices of this type are called input, output, or simply input/output (I/O) devices as shown in Fig. 16-17. There are many different types of I/O devices for use in systems, but they all operate something like a register. When data is to be read into the system from the outside world, it is first stored in a register. Then, when the system is ready the data is read onto the data bus. Outputs from the system are handled in the same way. When the system is ready, the output data is taken from the data bus and stored in an output register.

This sounds like a strange way of doing things, but when one realizes that things happen very fast in a microprocessor, it all begins to make sense. The data that we want to take off the data bus may be present on the bus for only a fraction of a microsecond. Most devices that will use such data in the outside world can't operate at this speed, so the data is temporarily stored in a register in the I/O device. Similar factors govern the input signals.

It is usually due to speed and timing considerations that microprocessors and their peripheral components use a lot of registers. Almost any part of such a system includes registers.

So far we have treated input and output signals as though they were always digital in nature. Often they are. If the input is from a keyboard, it will be digital in nature. Similarly, a printer or 7-segment output device will be operated by digital signals.

In many applications, our inputs and outputs are analog signals. These signals are accommodated by incorporating A/D and D/A converters between the system and the outside world.

The other devices that we usually connect to the microprocessor are memories. Usually, we will need both ROM and RAM.

THE SINGLE-COMPONENT MICROCOMPUTER

The types of microprocessors and other integrated circuits produced depends much more on the size of the potential market

than it does on technology. The state-of-the-art in the development of integrated circuits has progressed to the point where it is possible to build many components that are not commercially available today. Usually, the initial development cost for a new unit is quite high. However, if the market is large enough, this cost can be amortized over many units. Once the development is complete, the manufacturing cost is usually surprisingly low.

Because of these considerations, the types of new devices that we see in the future are apt to depend on what managers think will sell in great volume. It is for this reason that we see highly specialized devices used in such things as TV games that sell in great volume, and standard general purpose devices used in such things as broadcast equipment that usually sell in much smaller volume.

One of the newer devices that is more specialized than the earlier microprocessors, yet still a general purpose device, is called the single-component or single-chip microcomputer. We noted earlier that the term, microprocessor, was generally used for the IC that contained the CPU of a computer system. The term, microcomputer, is reserved for systems that contain all of the items required in a computer. The circuit that we are talking about has all of the functional elements of a computer in a single IC, and hence is called a microcomputer.

A typical example of a device of this type is the Intel Type 8048 single-chip microcomputer, which is shown in the block diagram of Fig. 16-18. As shown in the figure, the device contains the CPU, clock circuitry, 1024 bytes of program memory (ROM), and 8-bit timer and event counter, as well as 27 I/O lines. No other components are needed to build an actual digital computer.

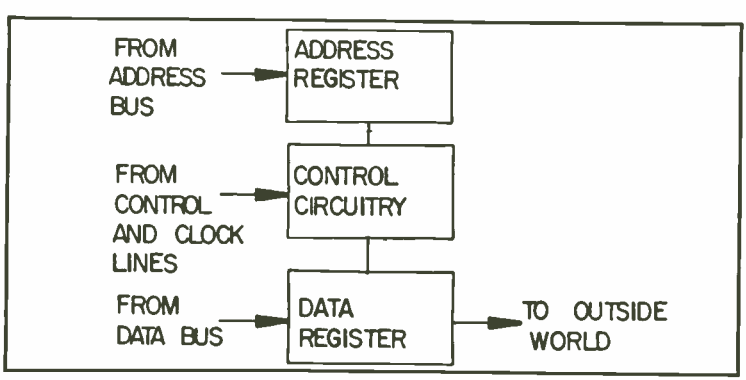


Fig. 16-17. Functional diagram of an I/O device.

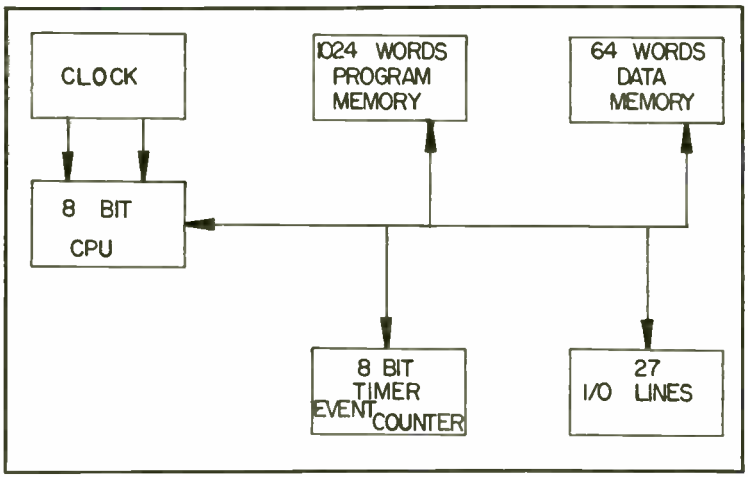


Fig. 16-18. Block diagram of the type 8048 single-component microcomputer. (Courtesy of Intel)

Several different versions of the device are available, differing primarily in the arrangement used for the program memory. In one version, the program is put into the ROM permanently at the factory. This is the best approach when the device is to be used in large quantities. In another version, the user can put a program into the memory which can be erased. This unit is ideal for small volume applications and in applications where the program might be changed for different versions of the final equipment.

We can gain a little more familiarity with the device by considering it as viewed from its pins. Figure 16-19 shows the pin diagram of the device.

Pin 1. This pin is used during programing and can also be used as a clock output. It also has other uses in system operation.

Pins 2 and 3. These are the clock input pins. Inasmuch as all of the clock circuitry is self contained, usually an oscillator crystal is connected between these two pins.

Pin 4, RESET. This pin is used to initialize the processor and is also used during verification of the program.

Pin 5 SS. This input pin is used to single-step the processor through each step of the program.

Pin 6, INT. This is an interrupt pin.

Pin 7, EA. This is an input pin, called "external access," that forces all program fetches to reference external memory. It is useful for debugging and essential for program verification.

Pin 8. This output pin is activated while the bus is being read. It can be used to put data onto the bus from an external source.

Pin 9, $\overline{PSEN}$. This output occurs only during a fetch to external program memory.

Pin 10. This output occurs during a bus write.

Pin 11. This output occurs once during each cycle and can be used as a clock output.

Pins 12 through 19. These eight pins are bidirectional and can carry data in either direction, using the $\overline{RD}$ and $\overline{WR}$ output signals.

Pin 20. This is the ground power supply connection.

Pins 21 through 24. These four pins form a bidirectional port.

Pin 25. A +25V signal is applied to this pin when a program is being burned into the internal ROA.

Pin 26. This pin also has a +25V input during programing, but has a +5V input during normal operation.

Pins 27 through 34. These pins form a bidirectional port.

Pins 35 through 38. These contain the four higher bits of the program counter.

Pin 39. This pin can be used as an input for the timer and counter functions, using the proper instruction.

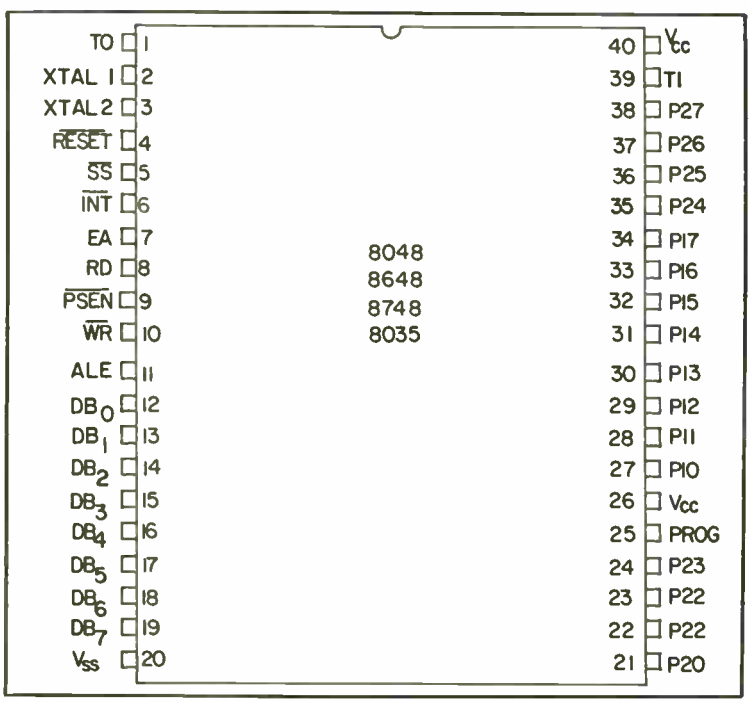


Fig. 16-19. Pin configuration of the type 8048 single-component microcomputer (courtesy of Intel).

Pin 40. This is the main positive power supply connection.

Without going into very much detail, we can see from the pin connections that the single-component microcomputer is quite a bit different from the microprocessor. In fact, its connections are much more like those of a complete computer system. Of course, this is what we might expect from the name of the device.

The fact that so many different functional elements are contained inside the device relieves the designer from many of the problems of circuit design. It is only when the system is large enough to require external memory that some of the problems that we usually associate with microprocessors come into the picture.

About the only technical skill required to apply the single-component microcomputer is the ability to program it. This, of course, requires knowledge of programing and is beyond the scope of this book.

A good example of the application of this device to broadcasting is in the Series 99 cartridge recording and playback machines manufactured by International Tapetronics Corporation. This equipment was developed to overcome many of the limitations usually associated with audio cartridge machines, particularly in stereo transmission.

The cartridge machines are completely controlled by a single-component microcomputer. This means that, except for manual cartridge insertion, most of the operation of the machine is either completely automatic or controlled by simple pushbuttons.

The functions controlled by the microcomputer are interesting. The crystal clock is accurate to within 0.05%, so it can accurately control the timing frequency in the system. It provides two cue tones in addition to those usually provided (3.35 and 3.65 kHz). These may be used as discrete cue tones or for other operations such as data logging. Provision is included for using these two tones in an FSK mode. The record head bias frequency is also derived from the clock.

In operation, the microcomputer looks at the status of all inputs to the system and controls the sequence of operations of the machine accordingly. The inputs that are looked at include all of the pushbuttons, the cartridge switch, the motor lock-in signal, cue tones and various status signals. Based on the status of these inputs, the system decides which operation, if any, is to be performed. Any necessary operations are then performed in a sequence that is stored in the memory. Internal buffers are

included in the system so that external devices such as lamps can be connected without any external buffering.

Some of the things that are accomplished by incorporating the microcomputer are shown in a description of the ELSA option, which stands for Erase, Locate Splice, and Azimuth. This option provides three functions—automatic azimuth adjustment, erasing, locating a splice, and stopping the machine. Any of these functions can be automatically skipped if desired by making a small change of a jumper wire.

The automatic azimuth adjustment is preceded by a bulk erasure of the cartridge to remove any recorded material. After the erasure is completed, a tone is recorded on all tracks—left, right, and cue. The signals from these tracks are then fed to a phase comparator. The output derived from the phase comparator is then applied to a drive motor which will move the head in such a direction as to correct the error. The procedure is then repeated with another tone to further reduce the phase error.

This is an example of a system which would require hundreds of LSI ICs to accomplish the same function. By using microcomputer technology, all of the control circuitry can be contained in a few ICs. This is the only thing that makes including so many functions economically viable.

DIGITAL FILTERING

Frequently, we have mentioned that a digital system can do anything that an analog system can do. In some instances this is easy to imagine. For example, it is easy to see how a digital system can be used to introduce a delay into a signal. On the other hand, there are some hard-to-imagine things that can be accomplished digitally. The filtering of a signal is an operation of this type. It is hard to see how we can filter a signal after it has been converted into a series of digital pulses. After all, the pulse repetition frequency depends on the sampling rate of the system and not on the frequency of the signal.

The fact is, however, that filtering can be accomplished digitally. In fact, it is easier to provide many rather elaborate filtering functions in the digital world than with more conventional analog filters. We will see digital filtering and signal processing increasingly in broadcast equipment, so it might be a good idea for us to look at the basic principles involved.

Figure 16-20A shows a block representing a filter. As we know, the most common filters fall into low-pass, high-pass,

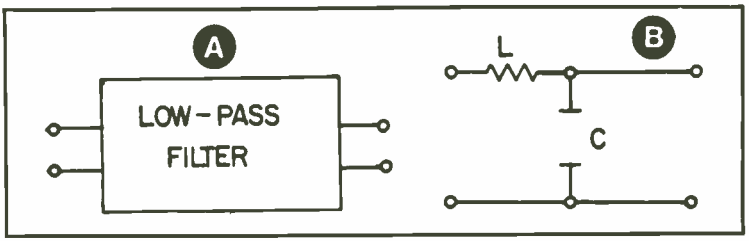


Fig. 16-20. Block diagram (A) and circuit diagram (B) if a low-pass filter.

band-pass or band-rejection categories. In conventional filters the box in Fig. 16-20 would contain either inductors and capacitors or resistors and capacitors. In more modern circuits, we find the so-called active filters that contain resistors, capacitors, and op-amps. Let's assume for the moment that the box of Fig. 16-20 contains a low-pass filter. We can imagine the circuit being something like that shown in Fig. 16-20B.

Without going into any of the mathematics used to describe filters, we can intuitively see how the circuit of Fig. 16-20B would indeed act as a low-pass filter. At very low frequencies, well below the cutoff frequency of the filter, the inductive reactance of the inductor will be very small and the capacitive reactance of the capacitor will be very high. Thus, a signal in this frequency range will pass through the circuit with very little attenuation. As the frequency increases, so does the reactance of the inductor. The reactance of the capacitor will get lower. Thus, the attenuation of the circuit will increase as the frequency of the signal increases. Eventually, at some frequency well above the filter's cutoff frequency, the inductor will look nearly like an open circuit and the capacitor nearly like a short circuit. Thus, the attenuation will be very high.

The circuit of Fig. 16-20B is the conventional approach to filtering, although the mathematics can become very involved. However, in electronics there are many situations where we may take two or more completely different approaches to a problem and get the same results. We can use a block diagram, like that of Fig. 16-20A to describe the overall behavior of a circuit. Although the circuit of Fig. 16-20B might well be what is inside the block, we often find that we can put something entirely different inside the box and get the same result.

A simple example is shown in Fig. 16-21A. Here we have a box with two terminals, and if we measure the resistance between the terminals we find it to be 25 ohms over a wide frequency range.

We find only resistance, no reactance. Our first guess as to what actually might be inside the box is a 25-ohm resistor. Of course, what is actually inside the box could be some series-parallel combination of resistors that provides a net resistance of 25 ohms. What probably wouldn't occur to us is the fact that a circuit like that of Fig. 16-21B could be inside the box. This circuit is probably not familiar, but if you take the time to calcualte the impedance across the terminals, you will find it to be 25 ohms of pure resistance at all frequencies.

The point of this example is that in addition to the circuits with which we are familiar, there are probably other unfamiliar circuit arrangements that will accomplish the same thing. This is the case in filtering. There is another approach to filtering which hasn't been popular because it hasn't been particularly easy to implement with analog circuits.

If we were to write down the mathematical expression for the performance of a filter to an arbitrary input signal, we would usually specify everything in terms of frequency. The equation would describe the frequency response of the filter and the frequency content of the signal. However, there is another way in which we could write the math. We could write everything in terms of time. Although the math is unpleasantly complex in either case, we would find that the time approach to the problem involved introducing time delays.

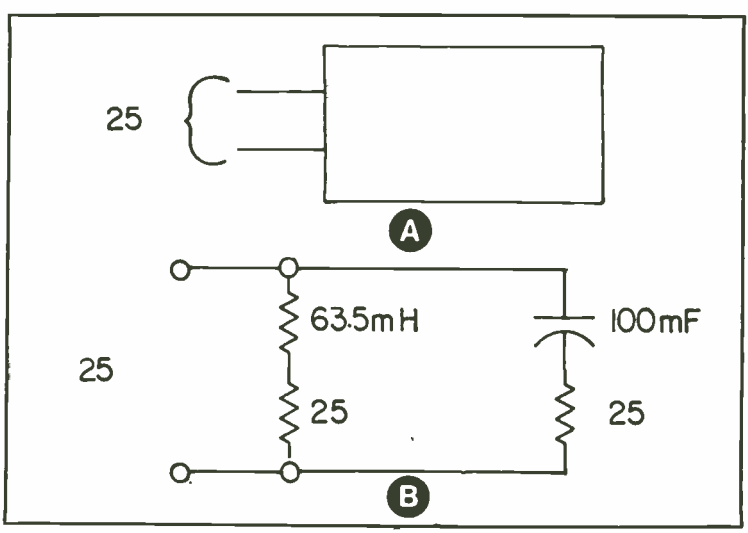


Fig. 16-21. Filter "boxes" often contain unexpected circuits.

The fact is that every filter takes some time to respond to the input signal. The time is indeed short, but there is always some time interval between when a signal is applied to the input of a filter and when this input has an effect on the output. This at least implies that we might be able to perform filtering by using things like time delays.

Figure 16-22A shows a box with an input signal consisting of just a pulse. Also shown is the output which the box would produce in response to this pulse. In Fig. 16-22B, we show how we can break any arbitrary signal into a series of pulses. From this it follows that if we can take the responses that would be produced by all of the pulses in our signal and add them together, we would get the same effect as though we had applied the continuous signal to the circuit. Actually, we would need a special circuit to this, but it could be done.

Although this mathematical technique has been known for many years, it hasn't been the basis of filter design because it resulted in circuit configurations that were very hard to build using conventional circuit elements. In the first place, stable time delays are hard to accomplish. Delay lines tend to be both bulky and difficult to build. Shifting analog signals in time isn't easy either.

This application is where some of the advantages of digital systems become apparent. We have seen that once we have a signal in digital form, it is easy to store in a memory without any signal degradation. Once a sample of a signal is stored in a memory, we can delay it as much as we wish, by simply reading it out after the required time delay. We can also vary this time delay at will by taking more or less time before we read the signal out of the memory. There is another advantage in that once we have a digital signal stored in memory, we can use the same sample as often as we wish without the need to take another sample.

This flexibility of a digital system makes it possible for us to take samples of our signal, delay them as long as we wish, multiply them by other samples or by other signals, and add the results. With this capability we can duplicate almost any conventional filter that one can imagine.

In addition to all that can be done with linear filters, the digital system can easily introduce nonlinearity into the system if desired. Thus, a filter can be made that treats loud signals in a completely different manner than the way it handles weaker signals. A further advantage is that any changes that might be necessary can be accomplished by merely changing the program in the digital system without even reaching for a screwdriver or a soldering iron.

Compare the requirement for changing a small program to the work required to change a conventional low-pass filter into a high-pass filter with a different cutoff frequency!

All devices that handle signals in a broadcast system, either audio or video, are made as linear as possible because nonlinearity introduces distortion that we can't remove with conventional techniques. Sometimes other parameters are sacrificed in order to get the required linearity. With digital processing equipment, linearity is no longer significant because we can remove the effect of nonlinearity with signal processing. Thus, as digital processing advances, we can expect changes in the design of analog components such as microphones and cameras.

THE SIGNAL PROCESSOR

From the preceding discussion, it is obvious that any digital signal processor requires analog-to-digital and digital-to-analog conversion. We must get the signal into digital form before we can process it, and we must get it back into analog form before we can use it in the real world. This might make it seem as though any digital processing system would require a large number of components. This isn't true.

Figure 16-23 shows a block diagram of an Intel 2920 Signal Processor. Note that the device includes both A/D and D/A converters inside the IC. Thus, the input and output signals are both analog signals, although all of the processing inside the device is done digitally. For this reason it is sometimes called an analog microcomputer.

Looking at the block diagram, we see that the input signal is sampled and the value of the sample is held constant long enough for it to be converted into digital form. It then enters the digital

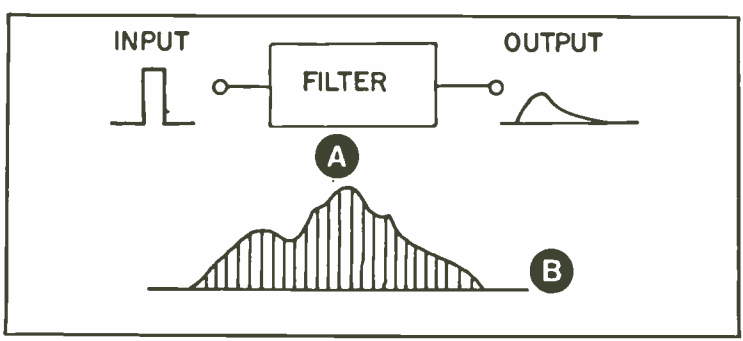


Fig. 16-22. Waveforms showing the response of a filter to a pulse (A) and an arbitrary signal broken into pulses.

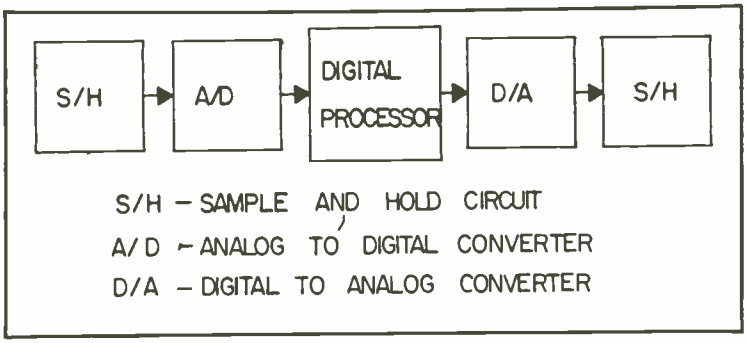


Fig. 16-23. Block diagram of the Intel 2920 signal processor.

circuitry which is controlled by a microprocessor. What happens to the signal depends on what is programmed into the microprocessor. Thus, the circuit can be made to process the signal in almost any imaginable way. Various filtering functions can be performed and even such things as recognizing a particular tone can be accomplished easily. There are four separate inputs and eight outputs. Each can be multiplexed so that many different things can be done at almost the same time.

The frequency response of the device is limited by how elaborate our processing is. Simple processing might only require one pass of the signal through the processor, whereas more elaborate processing may require many passes, hence more time.

As of this writing the signal processor IC is limited to the high audio frequencies, but it is dangerous to talk about limitations in a field that is advancing as fast as digital electronics!

Index