Digital Basics

This supplement to *The ARRL Handbook* presents digital theory fundamentals and some applications of that theory in Amateur Radio. The fundamentals introduce digital mathematics, including number systems, logic devices and simple digital circuits. Next, the implementation of these simple circuits is explored in integrated circuits, their families and interfacing. Finally, some Amateur Radio applications are discussed involving digital logic, embedded microcontrollers and interfacing to personal computers. This material was written by Dale Botkin, NØXAS, building on material in previous editions by Christine Montgomery, KGØGN and Paul Danzer, N1II.

1 Digital vs Analog

An analog signal can represent an infinitely variable indication of voltage, current, frequency, the position of a dial, or some other condition or value. As an example, using a potentiometer as a volume control will give you infinitely variable control over the volume of a signal. In theory, there is no limit to the difference in volume that can be produced. Though the control may be marked from 1 to 10, the actual value would have to be represented by a real number somewhere between 0 and 10. There are an infinite number of settings in between.

In its simplest form, a digital signal simply indicates the *on* or *off* state of some value or input signal. For example, the straight key you may use to key your CW transmitter (or the PTT switch of your voice transmitter) produces an on or off binary signal. In one state the transmitter produces an output signal of some sort; in the other state it does not. Another example is a simple light switch. The light is either on, or it's off. We represent these two states using 0 for off and 1 for on.

Digital electronics gets more interesting when we combine several or many simple on/off digital states to perform more complex tasks. For example, a relatively simple digital circuit can connect the antenna to either the transmitter or the receiver depending on a PTT or other keying signal. It can turn a preamp on or off depending on the state of the transmitter, mute the speaker while transmitting, and even select an antenna based on the selected frequency band. No special digital integrated circuits (chips) are needed to do any of these tasks; we can simply use bipolar transistors or MOSFETs, driven to saturation, as on/off switches. Simple circuits like this can often even be implemented with relays or diodes. The important fact is that the system is digital. There is no "almost transmitting" or "PTT switch partially pressed" state — it's either on, or it's off.

A very useful aspect of digital electronics is our ability to construct simple circuits that can maintain their on/off state indefinitely, until some event causes them to change. These *flip-flop* circuits can be used in various combinations to form *registers* that store information for later use or *counters* that count events and can be read or reset when needed. All of these circuits can be combined in ever larger groups until we finally arrive at the modern microprocessor. A microprocessor can accept input signals from many sources, follow a stored program to perform complex data storage and mathematical calculations, and produce output that we can use to do things that would be far more difficult with analog circuits.

So let's revisit our volume control example from the earlier paragraph. Let's assume we have a volume control, but it is used as an input to a digital system that will produce output at the desired level. This is quite
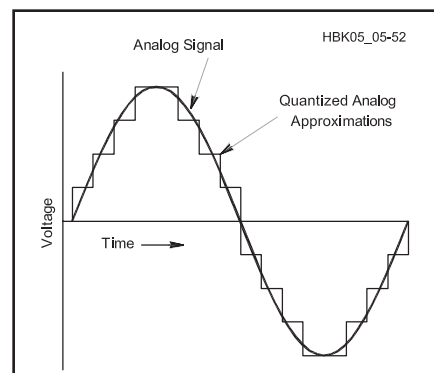

Figure 1 — An analog signal and its analog approximation. Note that the analog waveform has continuously varying voltage while the approximated waveform is composed of discrete steps.

common in modern equipment, whether it is amateur or consumer gear. Since the control is now digital, we know we can't have an infinite number of values. However, a simple on/off volume control would not be very useful. Using digital electronics, we can break the range between "off" and "fully on" into as many discrete steps as we need. With enough steps, we can give the user of the equipment an approximation of the original analog control while keeping the actual control digital.

By using coding, as discussed in the following pages, the two binary values (off and on, or 0 and 1) can represent any number of real values. **Figure 1** illustrates the contrast of an analog signal (in this case a sine wave) and its digital approximation. Four positive and four negative values are shown as an approximation to the sine wave, but any number of coded value steps can be used as an approximation. The more values are used to approximate the wave, the closer you can come to the actual wave form.

While the focus in this chapter will be on digital theory, many circuits and systems involve *both* digital and analog components. Often, a designer may choose between using digital technology, analog technology or a combination.

2 Number Systems

If you have been around computer hobbyists, some of whom are also hams, you may have seen a T-shirt or bumper sticker that reads, "There are 10 kinds of people in the world: those who understand binary, and those who don't." If this has puzzled you in the past, after reading this chapter you will be able to laugh with the rest of us.

In order to understand digital electronics, you must first understand the binary numbering system. Any number system has two distinct characteristics: a set of *symbols* (digits or numerals) and a *base* or radix. A *number* is a collection of these digits, where the leftmost digit is the *most significant digit (MSD)* and the right-most digit is the *least significant digit (LSD)*. The value of this number is a weighted sum of its digits. The *weights* are determined by the system's base and the digit's position relative to the decimal point.

While these definitions may seem strange with all the technical terms, they will be more familiar when seen in a decimal system example. See **Table 1**. This is the "traditional" number system with which we are all familiar. In the *base-10* or decimal numbering system we use every day, the digits used are 0 through 9. The weights are powers of ten: 10^0 or 1 for the right-most column, 10^1 or 10 for the next column, 10^2 or 100 for the next and so on. Thus the number 548 represents five hundreds, four tens and eight ones. In this case, 5 is the MSD, and 8 is the LSD. Once you understand this concept, it can be applied to numbering systems using bases other than 10 such as base-2, base-8, or even base-16.

2.1 Binary

Binary is a *base-2* number system and therefore limited to two symbols: {0, 1}. The weight factors are now powers of 2, like 2^0, 2^1 and 2^2. For example, the decimal number, 163 and its equivalent binary number, 10100011, are shown in **Table 2**.

The digits of a binary number are now *bits* (short for binary digit). The MSD is the *most significant bit (MSB)* and the LSD is the *least significant bit (LSB)*. Four bits make a *nibble* (which you will occasionally see spelled *nybble*) and two nibbles, or eight bits, make a *byte*. The length of a *word* is dependent upon the hardware; it generally can consist of two or four or more bytes, but occasionally will be some other number of bits. These groupings are useful when converting to hexadecimal notation, which is explained later. It is important to remember that while everyone agrees on the meaning of a bit, a nibble (regardless of spelling) and a byte, the meaning of *word* can vary.

Counting in binary follows the same pattern we would use for decimal or any other number system. Consider the three digit binary number XXX. First fill up the right-hand column.

Binary Number	Decimal Equivalent
0000	0
0001	1

The column has been filled, and much quicker than with decimal, since there are only two values instead of 10. But just as we would with a decimal number, we now reset the right-hand column to 0, increase the next column by 1, and continue.

Table 1
Decimal Numbers

Example: 548

Digit = 5; Weight = 10; Position = 2

548	=	$5(10^2)$	+	$4(10^1)$	+	$8(10^0)$
	=	5(100)	+	4(10)	+	8(1)
	=	500	+	40	+	8
	=	5		4		8
		MSD				LSD

Table 2
Decimal and Binary Number Equivalents

163	=	128	+	0	+	32	+	0	+	0	+0	+	2+	1	decimal
	=	1(128)	+	0(64)	+	1(32)	+	0(16)	+	0(8)	+0(4)	+	1(2)	+1(1)	
	=	$1(2^7)$	+	$0(2^6)$	+	$1(2^5)$	+	$0(2^4)$	+	$0(2^3)$	$+0(2^2)$	+	$1(2^1)$	$+1(2^0)$	
10100011	=	1		0		1		0		0	0		1	1	binary
		MSB												LSB	

Nibble   Nibble

Byte = 8 digits

0010	2
0011	3

Now the first two columns are full, so reset both back to 0 and increase the next column by 1 and continue:

0100	4
0101	5
0110	6
0111	7
1000	8
…	
1111	15

And so on.

2.2 Hexadecimal

The *hexadecimal*, or hex, *base-16* number system is widely used in computer systems for its ease in conversion to and from binary numbers and the fact that it is somewhat more human-friendly than long strings of 1s and 0s. A base-16 number requires 16 symbols. Since our normal mathematical number, as set up in the decimal system, has only 10 digits (0 through 9), a set of additional new symbols is required. Hex uses both numbers and characters in its set of sixteen symbols: {0, 1, 2, 3, 4, 5,6, 7, 8, 9, A, B, C, D, E, F}. Here, the letters A to F have the decimal equivalents of 10 to 15 respectively: A=10, B=11, C=12, D=13, E=14 and F=15. Again, the weights are powers of the base, such as 16^0, 16^1 and 16^2.

The four-bit binary listing in the previous paragraph shows that the individual 16 hex digits can be represented by a four-bit binary number. Since a byte is equal to eight binary digits, two hex digits provide a byte — the equivalent of 8 binary digits. Conversion from binary to hex is therefore simplified. Take a binary number, divide it into groups of four binary digits starting from the right, and convert each of the four binary digits to an individual value.

Conversion from hex to binary is equally convenient; simply replace each hex digit with its four-bit binary equivalent. As an example, the decimal number 163 is shown in Table 2 as binary 10100011. Divide the binary number in groups of four, so 1010 is equivalent to decimal 10 or "A" hex, and 0011 is equivalent to decimal 3, thus decimal number 163 is equivalent to hex A3.

2.3 Binary Coded Decimal (BCD)

The binary number system representation is the most appropriate form for fast internal computations since there is a direct mathematical relationship for every bit in the number. To interface with a human user — who usually wants to see inputs and outputs in terms of decimal numbers — other codes are more useful. The *Binary Coded Decimal* *(BCD)* system is a simple method for converting binary values to and from decimal for inputs and outputs for user-oriented digital systems. Back in the days when the most common method of presenting output to a user was via seven-segment LED displays, BCD was widely used. Since we now mostly use powerful microprocessors that can easily present information in decimal form, BCD is not nearly as common as it once was. You may, however, run into BCD when using or repairing older digital gear. It is also used in some chips intended for use in digital voltmeters.

In the BCD system, each decimal digit is expressed as a corresponding 4-bit binary number. In other words, the decimal digits 0 to 9 are encoded as the bit strings 0000 to 1001. To make the number easier to read, a space is left between each 4-bit group. For example, the decimal number 163 is equivalent to the BCD number 0001 0110 0011, as shown in **Table 3**.

The important difference between BCD and the previous number systems is that, starting with decimal 10, BCD loses the standard mathematical relationship of a weighted sum. BCD is simply a cut-off hexadecimal. Instead of using the 4-bit code strings 1010 to 1111 for decimal 10 to 15, BCD uses 0001 0000 to 0001 0101. This is one of the reasons that we have moved away from BCD.

2.4 Conversion Techniques

An easy way to convert a number from decimal to another number system is to do repeated division, recording the remainders in a tower just to the right. The converted number, then, is the remainders, reading up the tower. This technique is illustrated in **Table 4** for hexadecimal and binary conversions of the decimal number 163.

For example, to convert decimal 163 to hex, repeated divisions by 16 are performed. The first division gives 163/16 = 10 remainder 3. The remainder 3 is written in a column to the right. The second division gives 10/16 = 0 remainder 10. Since 10 decimal = A hex, A is written in the remainder column to the right. This division gave a divisor of 0 so the process is complete. Reading up the remainders column, the result is A3. The most common mistake in this technique is to forget that the Most Significant Digit ends up at the bottom.

Another technique that should be briefly mentioned can be even easier: use a calculator with a binary and/or hex mode option. Many inexpensive and readily available calculators intended for scientific and programming use will convert between number systems quite easily. In addition, calculator programs are available for all types of personal computers regardless of the operating system used.

One warning for this technique: this chapter doesn't discuss negative binary numbers. If your calculator does not give you the answer you expected, it may have interpreted the number as negative. This would happen when the number's binary form has a 1 in its MSB, such as the highest (leftmost) bit for the binary mode's default size. To avoid learning about negative binary numbers the hard way, always use a leading 0 when you enter a number in binary or hex into your calculator.

Table 3
Binary Coded Decimal Number Conversion

		0 0 0 1	0 1 1 0	0 0 1 1	**BCD**
	=	$1(2^0)$	$1(2^2) + 1(2^1)$	$1(2^1) + 1(2^0)$	
	=	(1)	(4 + 2)	(2 + 1)	
163	=	1	6	3	**decimal**

Table 4
Number System Conversions

Hex		Remainder		Binary		Remainder	
16	|163			2	|163		
	|10	3	LSB		|81	1	LSB
	|0	A	MSB		|40	1	
					|20	0	
					|10	0	
					|5	0	
					|2	1	
					|1	0	
					|0	1	MSB
A3 hex				1010 0011 binary			

3 Physical Representation of Binary States

3.1 State Levels

Most digital systems use the binary number system because many simple physical systems are most easily described by two state levels (0 and 1). For example, the two states may represent "on" and "off" or a "mark" and "space" in a communications transmission. In electronic systems, state levels are physically represented by voltages. A typical choice is
state 0 = 0 V
state 1 = 5 V

Since it is unrealistic to obtain these exact voltage values, a more practical choice is a range of values, such as
state 0 = 0.0 to 0.4 V
state 1 = 2.4 to 5.0 V

Figure 2 illustrates this representation of states by voltage levels. The undefined region between the two binary states is also known as the *transition region* or *noise margin*.

3.2 Transition Time

The gap in Figure 2, between binary 0 and binary 1, shows that a change in state does not occur instantly. There is a *transition time* between states. This transition time is a result of the time it takes to charge or discharge the stray capacitance in wires and other components because voltage cannot change instantaneously across a capacitor. (Stray inductance in the wires also has an effect because the current through an inductor can't change instantaneously.) The transition from a 0 to a 1 state is called the *rise time*, and is usually specified as the time for the pulse to rise from 10% of its final value to 90% of its final value. Similarly, the transition from a 1 to a 0 state is called the *fall time,* with a similar 10% to 90% definition. Note that these times need not be the same. **Figure 3A** shows an ideal signal, or *pulse*, with zero-time switching. Figure 3B shows a typical pulse, as it changes between states in a smooth curve.

Rise and fall times vary with the logic family used and the location in a circuit. Typical values of transition time are in the microsecond to nanosecond range. In a circuit, distributed inductances and capacitances in wires or

Figure 2 — Representation of binary states 1 and 0 by a selected range of voltage levels.

PC-board traces may cause rise and fall times to increase as the pulse moves away from the source. One reason rise and fall times may be of interest to the radio designer is because of the possibility of generating RF noise in a digital circuit.
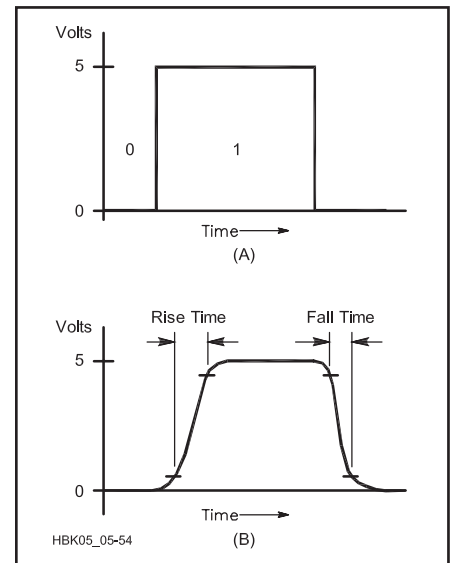
3.3 Propagation Delay

Rise and fall times only describe a relationship within a pulse. For a circuit, a pulse input into the circuit must propagate through the circuit; in other words it must pass through each component in the circuit until eventually it arrives at the circuit output. The time delay between providing an input to a circuit and seeing a response at the output is the *propagation delay* and is illustrated by **Figure 4.**

For modern switching logic, typical propagation delay values are in the 1 to 15 nanosecond range. (It is useful to remember that the propagation delay along a wire or printed-circuit-board trace is about 1.0 to 1.5 ns per inch.) Propagation delay is the result of cumulative transition times as well as transistor switching delays, reactive element charging times and the time for signals to travel through wires. In complex circuits, different propagation delays through different paths can cause problems when pulses must arrive somewhere at exactly the same time.

The effect of these delays on digital devices can be seen by looking at the speed of the digital pulses. Most digital devices use *clock*
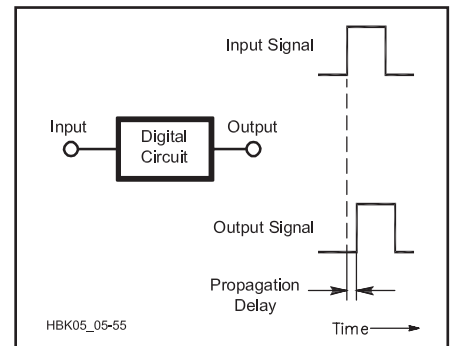

Figure 3 — (A) An ideal digital pulse and (B) a typical actual pulse, showing the gradual transition between states.

Figure 4 — Propagation delay in a digital circuit.

pulses. If two pulses are supposed to arrive at a logic circuit at the same time, or very close to the same time, the path length for the two signals cannot be any different than two to three inches. This can be a very significant design problem for high-speed logic designs.

4 Combinational Logic

Having defined a way to use voltage levels to physically represent digital numbers, we can apply digital signal theory to design useful circuits. Digital circuits combine binary inputs to produce a desired binary output or combination of outputs. This simple combination of 0s and 1s can become very powerful, implementing everything from simple switches to powerful computers.

A digital circuit falls into one of two types: combinational logic or sequential logic. In a *combinational logic* circuit, the output depends only on the *present inputs* (if we ignore propagation delay). In contrast, in a *sequential logic* circuit, the output depends on the present inputs, the *previous sequence of inputs* and often a clock signal. Later sections of this chapter will examine some circuits

built using the basics established here.

4.1 Boolean Algebra and the Basic Logical Operators

Combinational circuits are composed of logic gates, which perform binary operations. Logic gates manipulate binary numbers, so you need an understanding of the algebra of

binary numbers to understand how logic gates operate. *Boolean algebra* is the mathematical system used to describe and design binary digital circuits. It is named after George Boole, the mathematician who developed the system. Standard algebra has a set of basic operations: addition, subtraction, multiplication and division. Similarly, Boolean algebra has a set of basic operations, called *logical operations*: NOT, AND and OR.

The function of these operators can be described by either a Boolean equation or a truth table. A Boolean *equation* describes an operator's function by representing the inputs and the operations performed on them. An equation is of the form "B = A," while an *expression* is of the form "A." In an assignment equation, the inputs and operations appear on the right and the result, or output, is assigned to the variable on the left.

A *truth table* describes an operator's function by listing all possible inputs and the corresponding outputs. Truth tables are sometimes written with Ts and Fs (for true and false) or with their respective equivalents, 1s and 0s. In company databooks (catalogs of logic devices a company manufactures), truth tables are usually written with Hs and Ls (for high and low). In the figures, 1 will mean high and 0 will mean low. This representation is called positive logic. The meaning of different logic types and why they are useful is discussed in a later section.

Each Boolean operator also has two circuit symbols associated with it. The traditional symbol — used by ARRL and other US publications — appears on top in each of the figures; for example, the triangle and bubble for the NOT function in Figure 7. In the traditional symbols, a small circle, or *bubble*, always represents "NOT." (This *bubble* is called a state indicator.)

Appearing just below the traditional symbol is the newer ANSI/IEEE Standard symbol. This symbol is always a square box with notations inside it. In these newer symbols, a small triangular flag represents "NOT." The new notation is an attempt to replace the detailed logic drawing of a complex function with a simpler block symbol. Adoption of the newer symbols has been spotty, and you are therefore still more likely to see the traditional symbols for basic logic functions than the ANSI/IEEE symbols.

4.2 Common Gates

Figures **5**, **6** and **7** show the truth tables, Boolean algebra equations and circuit symbols for the three basic Boolean operations: AND, OR and NOT, respectively. All combinational logic functions, no matter how complex, can be described in terms of these three operators. Each truth table can be converted into words. The truth table for the two-input AND gate can be expressed as "the output C
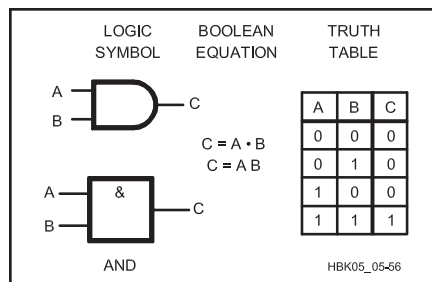

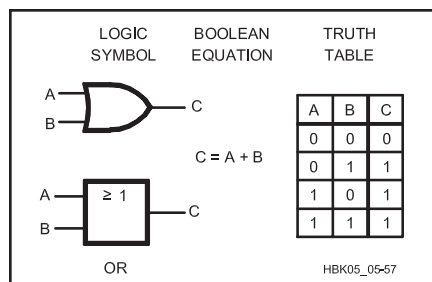Figure 5 — Two-input AND gate.

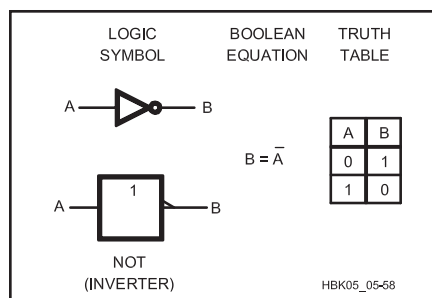Figure 6 — Two-input OR gate.

Figure 7 — Inverter.

is a 1 only when the inputs are both 1s." This can be seen by examining the output column C — it remains at a 0 and becomes a 1 only when the input column A and the input column B are both 1s — the last line of the table.

The NOT operation is also called *inversion*, *negation* or *complement*. The circuit that implements this function is called an *inverter* or *inverting buffer*. The most common notation for NOT is a bar over a variable or expression. For example, NOT A is denoted $\overline{A}$. This is read as either "Not A" or as "A bar." A less common notation is to denote Not A by A', which is read as "A prime." You will also see various other notations in schematic diagrams and component data sheets, such as a leading exclamation point or has symbol — !A or #A indicating "Not A."

While the inverting buffer and the noninverting buffer covered later have only one input and output, many combinational logic elements can have multiple inputs. When a combinational logic element has two or more inputs and one output, it is called a *gate*. (The term "gate" has a number of different but

specific technical uses. For a clarification of the many definitions of gate, see the section on **Synchronicity and Control Signals**, later in this chapter.) For simplicity, the figures and truth tables for multiple-input elements will show the operations for only two inputs, the minimum number. Remember, though, that it is quite common to have gates with more than two inputs. A three-, four-, or eight-input gate works in the exact same manner as a two-input gate.

The output of an AND function is 1 only if *all* of the inputs are 1. Therefore, if *any* of the inputs are 0, then the output is 0. The notation for an AND is either a dot (•) between the inputs, as in C = A•B, or nothing between the inputs, as in C = AB. Read these equations as "C equals A AND B."

The OR gate detects if one or more inputs are 1. In other words, if *any* of the inputs are 1, then the output of the OR gate is 1. Since this includes the case where more than one input may be 1, the OR operation is also known as an INCLUSIVE OR. The OR operation detects if *at least one* input is 1. Only if all the inputs are 0, then the output is 0. The notation for an OR is a plus sign (+) between the inputs, as in C = A + B. Read this equation as "C equals A OR B."

4.3 Additional Gates

More complex logical functions are derived from combinations of the basic logical operators. These operations — NAND, NOR, XOR and the noninverter or buffer — are illustrated in **Figures 8** through **11**, respectively. As before, each is described by a truth table, Boolean algebra equation and circuit symbols. Also as before, except for the noninverter, each could have more inputs than the two illustrated.

The NAND gate (short for NOT AND) is equivalent to an AND gate followed by a NOT gate. Thus, its output is the complement of the AND output: The output is a 0 only if all the inputs are 1. If any of the inputs is 0, then the output is a 1.

The NOR gate (short for NOT OR) is equivalent to an OR gate followed by a NOT gate. Thus, its output is the complement of the OR output: If any of the inputs are 1, then the output is a 0. Only if all the inputs are 0, then the output is a 1.

The operations so far enable a designer to determine two general cases: (1) if *all* inputs have a desired state or (2) if *at least one* input has a desired state. The XOR and XNOR gates enable a designer to determine if *one and only one* input of a desired state is present.

The XOR gate (read as EXCLUSIVE OR) is a combination of an OR and a NAND gate. It has an output of 1 if one and only one of the inputs is a 1 state. The output is 0 otherwise. The symbol for XOR is ⊕. This is easy to remember if you think of the "+" OR symbol
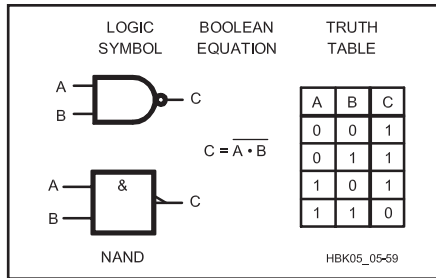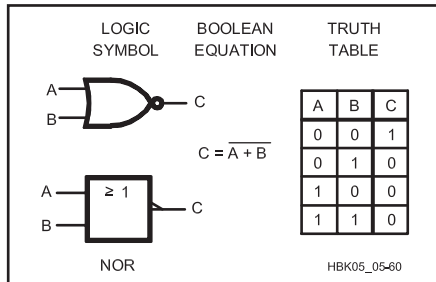
Figure 8 — Two-input NAND gate.
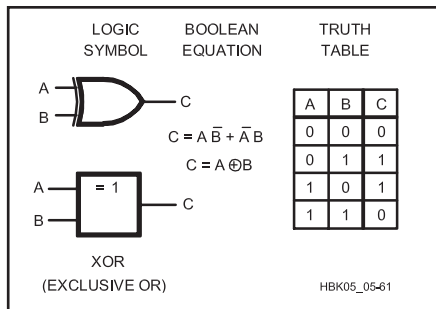

Figure 9 — Two-input NOR gate.

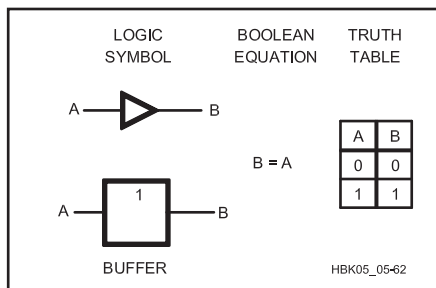**Figure 10 — Two-input XOR gate.**

Figure 11 — Noninverting buffer.

enclosed in an "O" for *only one*.

The XOR gate is also known as a "half adder," because in binary arithmetic it does everything but the "carry" operation. The following examples show the possible binary additions for a two-input XOR.

A	0	0	1	1
B	0	1	0	1
Sum	0	1	1	0

The XNOR gate (read as EXCLUSIVE

NOR) is the complement of the XOR gate. The output is 0 if one and only one of the inputs is a 1. The output is 1 either if all inputs are 0 or more than one input is 1.

NONINVERTERS (BUFFERS)

A *noninverter*, also known as a *buffer*, *amplifier* or *driver*, at first glance does not seem to do anything. It simply receives an input and produces the same output. In reality, it is changing other properties of the signal in a useful fashion, such as amplifying the current level. While not useful for logical operations, applications of a noninverter include providing sufficient current to drive a number of gates or some other circuit such as a relay; interfacing between two logic families; obtaining a desired pulse rise time; and providing a slight delay to make pulses arrive at the proper time.

TRI-STATE GATES

Under normal circumstances, a logic element can drive or feed several other logic elements. A typical AND gate might be able to drive or feed 10 other gates. This is known as *fan-out*. However, with certain exceptions only one gate output can be connected to a single wire. If you have two possible driving sources to feed one particular wire, some logic network that probably includes a number OR gates must be used.

In many applications, including computers, data is routed internally on a set of wires called *buses*. The data on the bus can come from many circuits or drivers, and many other devices may be *listening* on the bus. To eliminate the need for the network of OR gates to drive each bus wire, a set of gates known as *tri-state* gates are used.

The symbol and truth table for a tri-state gate are shown in **Figure 12**. A tri-state gate can be any of the common gates previously described, but with one additional control lead. When this lead is enabled (it can be designed to allow either a 0 or a 1 to enable it) the gate operates normally, according to the truth table for that type of gate. However, when the gate is not enabled, the output goes to a high impedance (Hi-Z), and so far as the output wire is concerned, the gate does not exist.

Each device that has to send data down a bus wire is connected to the bus wire through a tri-state gate. However, as long as only one device, through its tri-state gate, is enabled, it is as though all the other connected tri-state gates do not exist.

4.4 Boolean Theorems

The analysis of a circuit starts with a logic diagram and then derives a circuit description. In digital circuits, this description is in the form of a truth table or logical equation. The
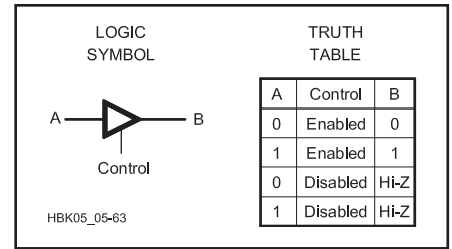

Figure 12 — Tri-State gate.

synthesis, or design, of a circuit goes in the reverse: starting with an informal description, determining an equation or truth table and then expanding the truth table to components that will implement the desired response. In both of these processes, we need to either simplify or expand a complex logical equation.

To manipulate an equation, we use mathematical *theorems*. Theorems are statements that have been proven to be true. The theorems of Boolean algebra are very similar to those of standard algebra, such as commutivity and associativity. Proofs of the Boolean algebra theorems can be found in an introductory digital design textbook.

BASIC THEOREMS

Table 5 lists the theorems for a single variable and **Table 6** lists the theorems for two or more variables. These tables illustrate the *principle of duality* exhibited by the Boolean theorems: Each theorem has a dual in which, after swapping all ANDs with ORs and all 1s with 0s, the statement is still true.

The tables also illustrate the *precedence* of the Boolean operations: the order in which operations are performed when not specified by parenthesis. From highest to lowest, the precedence is NOT, AND then OR. For example, the distributive law includes the expression "A + B•C." This is equivalent to
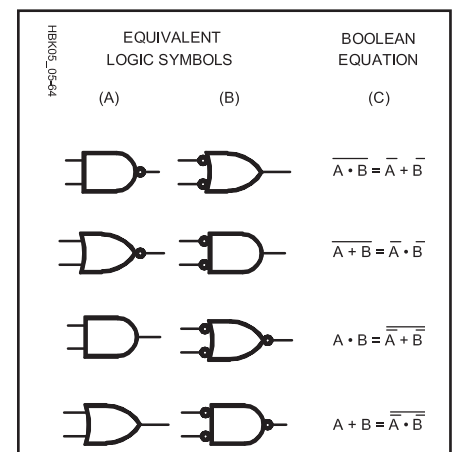

Figure 13 — Equivalent gates from DeMorgan's Theorem: Each gate in column A is equivalent to the opposite gate in column B. The Boolean equations in column C formally state the equivalences.

"A + (B•C)." The parenthesis around (B•C) can be left out since an AND operation has higher priority than an OR operation. Precedence for Boolean algebra is similar to the convention of standard algebra: raising to a power, then multiplication, then addition.

DeMORGAN'S THEOREM

One of the most useful theorems in Boolean algebra is DeMorgan's Theorem:

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

and its dual

$$\overline{A + B} = \overline{A} \cdot \overline{B}.$$

Table 5
Boolean Algebra Single Variable Theorems

Identities:	$A \cdot 1 = A$	$A + 0 = A$
Null elements:	$A \cdot 0 = 0$	$A + 1 = 1$
Idempotence:	$A \cdot A = A$	$A + A = A$
Complements:	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
Involution:	$\overline{(\overline{A})} = A$	

Table 6
Boolean Algebra Multivariable Theorems

Commutativity:	$A \cdot B = B \cdot A$ $A + B = B + A$
Associativity:	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$ $(A + B) + C = A + (B + C)$
Distributivity:	$(A + B) \cdot (A + C) = A + B \cdot C$ $A \cdot B + A \cdot C = A \cdot (B + C)$
Covering:	$A \cdot (A + B) = A$ $A + A \cdot B = A$
Combining:	$(A + B) \cdot (A + \overline{B}) = A$ $A \cdot B + A \cdot \overline{B} = A$
Consensus:	$A \cdot B + \overline{A} \cdot C + B \cdot C = A \cdot B + \overline{A} \cdot C$ $(A + B) \cdot (\overline{A} + C) \cdot (B + C) = (A + B) \cdot (\overline{A} + C)$ $A + \overline{A}B = A + B$

Table 7
DeMorgan's Theorem

(A) $\overline{A \cdot B} = \overline{A} + \overline{B}$
(B) $\overline{A + B} = \overline{A} \cdot \overline{B}$
(C)

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
A	B	$\overline{A}$	$\overline{B}$	$A \cdot B$	$\overline{A \cdot B}$	$A + B$	$\overline{A + B}$	$\overline{A} \cdot \overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	0	1	0	1	1	1
0	1	1	0	0	1	1	0	0	1
1	0	0	1	0	1	1	0	0	1
1	1	0	0	1	0	1	0	0	0

(A) and (B) are statements of DeMorgan's Theorem. The truth table at (C) is proof of these statements: (A) is proven by the equivalence of columns 6 and 10 and (B) by columns 8 and 9.

The truth table in **Table 7** proves these statements. DeMorgan's Theorem provides a way to simplify the complement of a large expression. It also enables a designer to interchange a number of equivalent gates, as shown by **Figure 13**.

The equivalent gates show that the duality principle works with symbols the same as it does for Boolean equations: just swap ANDs with ORs and switch the bubbles. For example, the NAND gate — an AND gate followed by an inverter bubble — becomes an OR gate preceded by two inverter bubbles. DeMorgan's Theorem is important because it means any logical function can be implemented using either inverters and AND gates or inverters and OR gates. Also, the ability to change placement of the bubbles using DeMorgan's Theorem is useful in dealing with mixed logic, to be discussed next.

POSITIVE AND NEGATIVE LOGIC

The truth tables shown in the figures in this chapter are drawn for positive logic. In *positive logic*, or *high true*, a higher voltage means true (logic 1) while a lower voltage means false (logic 0). This is also referred to as *active high*: a signal performs a named action or denotes a condition when it is "high" or 1. In *negative logic*, or *low true*, a lower voltage means true (1) and a higher voltage means false (0). An *active low* signal performs an action or denotes a condition when it is "low" or 0.

In both logic types, true = 1 and false = 0; but whether true means high or low differs. Company databooks are drawn for general truth tables: an H for high and an L for low. (Some tables also have an X for a "don't care" state. "Don't care" means that the output does not depend on the state of that variable.) The function of the table can differ depending on whether it is interpreted for positive logic or negative logic.

Device data sheets often show positive logic convention, or positive logic is assumed. However, a signal into an IC is represented with a bar above it, indicating that the "enable" on that wire is active low — it does *not* mean negative logic (0 V = a logical 1) is used! Similarly a bubble on the input of a logic element also usually means active low. These can be sources of confusion.

Figure 14 shows how a general truth table differs when interpreted for different logic types. The same truth table gives two equivalent gates: positive logic gives the function of

Figure 14 — (A) A general truth table, (B) a truth table and NAND symbol for positive logic and (C) a truth table and NOR symbol for negative logic.

a NAND gate while negative logic gives the function of a NOR gate.

Note that these gates correspond to the equivalent gates from DeMorgan's Theorem. A bubble on an input or output terminal indicates an active low device. The absence of bubbles indicates an active high device.

Like the bubbles, signal names can be used to indicate logic states. These names can aid the understanding of a circuit by indicating control of an action (GO, /ENABLE) or detection of a condition (READY, /ERROR). The action or condition occurs when the signal is in its active state. When a signal is in its active state, it is called *asserted*; a signal not in its active state is called *negated* or *deasserted*.

A prefix can easily indicate a signal's active state. Active low signals are preceded by a symbol such as /, |, ! or # (for example /READY or !READY). Active low signals are also denoted by an overscore, such as $\overline{\text{CL}}$. Active high signals have no prefix or overscore. As an example, see the truth table for a flip-flop later in this chapter. Standard practice is that the signal name and input pin match (have the same active level). For example, an input with a bubble (active low) may be called /READY, while an input with no bubble (active high) is called READY. Output signal names should always match the device output pin.

In this chapter, positive logic is used unless indicated otherwise. Although using mixed logic can be confusing, it does have some advantages. Mixed logic combined with DeMorgan's Theorem can promote more effective use of available gates. Also, well-chosen signal names and placement of bubbles can promote more understandable logic diagrams.

5 Sequential Logic

The previous section discussed combinational logic, whose outputs depend only on the present inputs. In contrast, in *sequential logic* circuits, the new output depends not only on the present inputs but also on the present outputs. The present outputs depended on the previous inputs and outputs and those earlier outputs depended on even earlier inputs and outputs and so on. Thus, the present outputs depend on the previous *sequence of inputs* and the system has *memory*. Having the outputs become part of the new inputs is known as *feedback*.

5.1 Synchronicity and Control Signals

When a combinational circuit is given a set of inputs, the outputs take on the expected values after a propagation delay during which the inputs travel through the circuit to the output. In a sequential circuit, however, the travel through the circuit is more complicated. After application of the first inputs and one propagation delay, the outputs take on the resulting state; but then the outputs start trickling back through and, after a second propagation delay, new outputs appear. The same happens after a third propagation delay. With propagation delays in the nanosecond range, this cycle around the circuit is rapidly and continually generating new outputs. A user needs to know when the outputs are valid.

There are two types of sequential circuits: synchronous circuits and asynchronous circuits, which are analyzed differently for valid outputs. In *asynchronous* operation, the outputs respond to the inputs immediately after the propagation delay. To work properly, this type of circuit must eventually reach a *stable* state: the inputs and the fed back outputs result in the new outputs staying the same. When the nonfeedback inputs are changed, the feedback cycle needs to eventually reach a new stable state. Generally, the output of this type of logic is not valid until the last input has changed, and enough time has elapsed for all propagation delays to have occurred.

In *synchronous* operation, the outputs change state only at specific times. These times are determined by the presence of a particular input signal: a clock, toggle, latch or enable. Synchronicity is important because it ensures proper timing: all the inputs are present where needed when the control signal causes a change of state.

CONTROL SIGNALS

Some authors vary the meanings slightly for the different control signals. The following is a brief illustration of common uses, as well as showing uses for noun, verb and adjective. *Enabling* a circuit generally means the control signal goes to its asserted level, allowing the circuit to change state. *Latch* implies memory: a latch circuit can store a bit of information. A latch signal can cause a circuit to keep its present state indefinitely. *Gate* can have several meanings, some unrelated to synchronous control. For example, a gate can be a signal used to trigger the passage of other signals through a circuit. A gate can also be a logic circuit with two or more inputs and one output, as used earlier in this chapter. Of course, "gate" can also be one of the electrodes of an FET as described in another chapter. To *toggle* means a signal changes state, from 1 to 0 or vice versa. A *clock* signal is one that toggles at a regular rate.

Clock control is the most common method of synchronizing logic circuits, so it has some additional terms as illustrated by **Figure 15**. The *clock period* is the time between successive transitions in the same direction; the *clock frequency* is the reciprocal of the period. A *pulse* or *clock tick* is the first edge in a clock period, or sometimes the period itself or the first half of the period. The *duty cycle* is the percentage of time that the clock signal is at its asserted level. A common application of the use of clock pulses is to limit the input to a logic circuit such that the circuit is only enabled on one clock phase; that is the inputs occur before the clock changes to a logic 1.

Figure 15 — Clock signal terms. The duty cycle would be t_H / t_{PERIOD} for an active high signal and t_L / t_{PERIOD} for an active low signal.

Figure 16 — Level-triggered vs edge-triggered for a D flip-flop: (A) D input, (B) clock input, (C) output Q for level-triggered: circuit responds whenever clock is 1. (D) output Q for edge-triggered: circuit responds only at rising edge of clock. Notice that the short negative pulse on the D input is not reproduced by the edge-triggered flip-flop.

The outputs are sampled only after this point; perhaps when the clock next changes back to a logic 0.

The reaction of a synchronous circuit to its control signal is *static* or *dynamic*. Static, *gated* or *level-triggered* control allows the circuit to change state whenever the control signal is at its active or asserted level. Dynamic, or *edge-triggered*, control allows the circuit to change state only when the control signal *changes* from unasserted to asserted. By convention, a control signal is active high if state changes occur when the signal is high or at the rising edge and active low in the oppo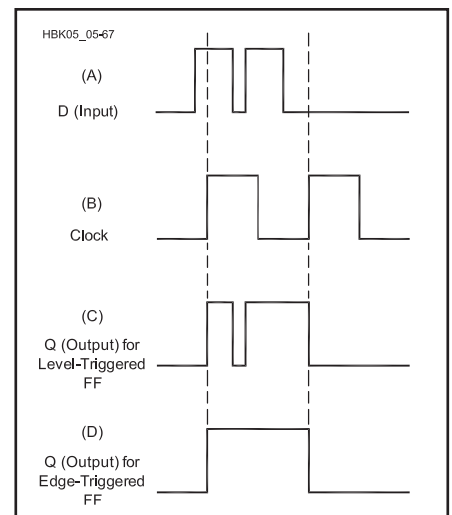site case. Thus, for positive logic, the convention is enable = 1 or enable goes from 0 to 1. This transition from 0 to 1 is called *positive edge-triggered* and is indicated by a small triangle inside the circuit box. A circuit responding to the opposite transition, from 1 to 0, is called *negative edge-triggered*, indicated by a bubble with the triangle.

Whether a circuit is level-triggered or edge-triggered can affect its output, as shown by **Figure 16**. The D input includes a very brief pulse, called a *glitch*, which may be caused by noise. The differing results at the output illustrate how noise can cause errors. We have both edge and level triggered circuits available so that we can meet the requirements of our particular design.

5.2 Flip-Flops

Flip-flops are the basic building blocks of sequential circuits. A *flip-flop* is a device with two stable states: the *set* state (1) and the *reset* or *cleared* state (0). The flip-flop can be placed in one or the other of the two states by applying the appropriate input. Since a common use of flip-flops is to store one bit of information, some use the term *latch* interchangeably with flip-flop. A set of latches, or flip-flops holding an n-bit number is called a *register*. While gates have special symbols, the schematic symbol for most sequential logic components is a rectangular box with the circuit name or abbreviation, the signal names and assertion bubbles. For flip-flops, the circuit name is usually omitted since the signal names are enough to indicate a flip-flop and its type. The four basic types of flip-flops are the S-R, D, T and J-K. The most common flip-flops available to Amateurs today are the J-K and D- flip-flops; the others can be synthesized if needed by utilizing these two varieties. Table 8 provides a summary of symbols and truth tables for these four types of flip-flops.

TRIGGERING A FLIP-FLOP

Although the S-R (Set-reset) flip flop is no longer generally available or used, it does provide insight in basic flip-flop operations and triggering. It is also not uncommon to

S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	?	?

Symbol
(A)

State Table
(C)

HBK0095
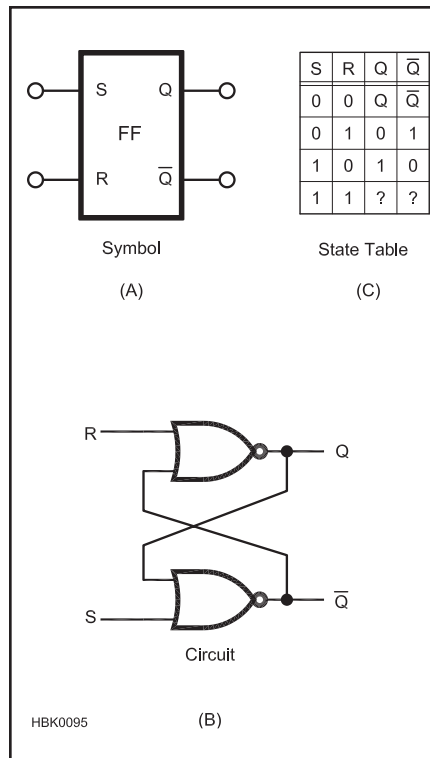
Circuit
(B)

Figure 17 — Unclocked S-R Flip-Flop. (A) schematic symbol. (B) circuit diagram. (C) state table or truth table.

build S-R flip-flops out of gates for jobs such as switch contact debouncing. In **Figure 17** the symbol for an S-R flip flop and its truth table are accompanied by a logic implementation, using NOR gates. As the truth table shows, this basic implementation requires a positive or logic 1 input on the set input to put the flip-flop in the Q or set state. Remove the input, and the flip flop stays in the Q state, which is what is expected of a flip-flop. Not until the S input receives a logic 1 input does the flip- flop change state and go to the reset or Q=0 state.

Note that the input can be a short pulse or a level; as long as it is there for some minimum duration (established by the propagation delay of the gates used), the flip-flop will respond. By contrast the clocked S-R flip-flop in **Figure 18** requires both a positive level to be present at either the S or R inputs and a positive clock pulse. The clock pulse is ANDed with the S or R input to trigger the flip–flop. In this case the flip-flop shown is implemented with a set of NOR gates.

A final triggering method is edge triggering. Here, instead of using the clock pulse as shown in the timing diagram of Figure 18, just the edge of the clock pulse is used. The edge-triggered flip-flop helps solves a problem with noise. Edge-triggering minimizes the time during which a circuit responds to its

S	R	CLK	Q	$\overline{Q}$
X	X	0	q	$\overline{q}$
0	0	1	q	$\overline{q}$
0	1	1	0	1
1	0	1	1	0
1	1	1	?	?

Truth Table
(C)

HBK05_05-69

Timing Diagram
(D)

Figure 18 — Master-Slave Flip-Flop. (A) logic symbol. (B) NAND gate implementation. (C) truth table. (D) timing diagram.

Figure 19 — (A & B) The D flip-flop. (C) A truth table for the positive edge-triggered D flip-flop.
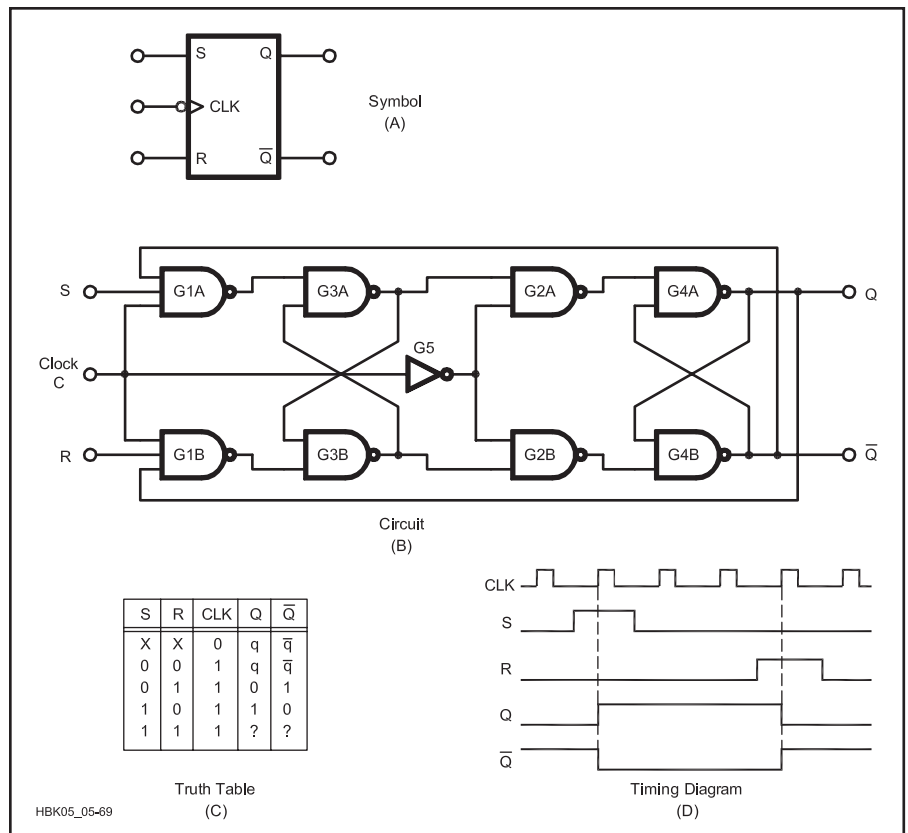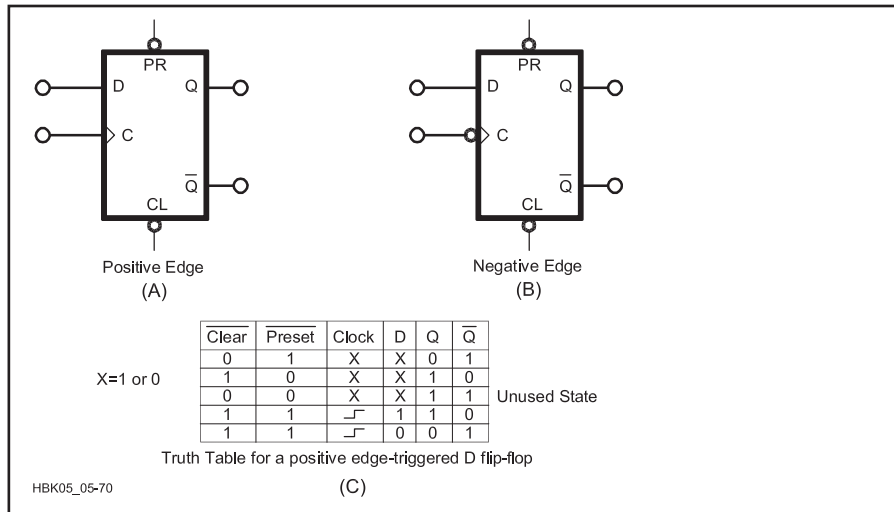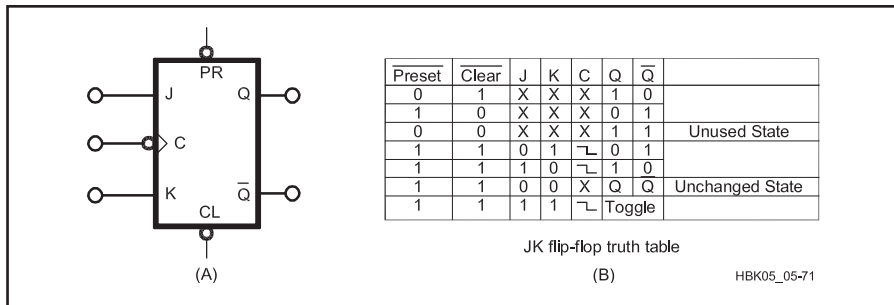
Truth Table for a positive edge-triggered D flip-flop

Clear	Preset	Clock	D	Q	Q̄	
0	1	X	X	0	1	
1	0	X	X	1	0	
0	0	X	X	1	1	Unused State
1	1	⌐	1	1	0	
1	1	⌐	0	0	1	

X=1 or 0

HBK05_05-70 (C)

Figure 20 — (A) JK flip-flop. (B) JK flip-flop truth table.

JK flip-flop truth table

Preset	Clear	J	K	C	Q	Q̄	
0	1	X	X	X	1	0	
1	0	X	X	X	0	1	
0	0	X	X	X	1	1	Unused State
1	1	0	1	⌐	0	1	
1	1	1	0	⌐	1	0	
1	1	0	0	X	Q	Q	Unchanged State
1	1	1	1	⌐	Toggle		

HBK05_05-71

inputs: the chance of a glitch occurring during the nanosecond transition of a clock pulse is remote. A side benefit of edge-triggering is that only one new output is produced per clock period. Edge-triggering is denoted by a small rising-edge or falling-edge symbol in the clock column of the flip-flop's truth table. It can also appear, instead of the clock triangle, inside the schematic symbol.

MASTER/SLAVE FLIP-FLOP

One major problem with the simple flip-flop shown up to now is the question of when is there a valid output. Suppose a flip-flop receives input that causes it to change state; at the same time the output of this flip-flop is being sampled to control some other logic element. There is a real risk here that the output will be sampled just as it is changing and thus the validity of the output is questionable.

A solution to this problem is a circuit that samples and stores its inputs before changing its outputs. Such a circuit is built by placing two flip-flops in series; both flip-flops are triggered by a common clock but an inverter on the second flip-flop's clock input causes it to be asserted only when the first flip-flop is not asserted. The action for a given clock pulse is as follows: The first, or master, flip-flop can change only when the clock is high, sampling and storing the inputs. The second, or slave, flip-flop gets its input from the master and changes when the clock is low. Hence, when the clock is 1, the input is sampled; then when the clock becomes 0, the output is generated. Note that a bubble may appear on the schematic symbol's clock input, reminding us that the output appears when the clock is asserted low. This is conventional for TTL-style J-K flip-flops, but it can be different for CMOS devices.

The master/slave method isolates output changes from input changes, eliminating the problem of series-fed circuits. It also ensures only one new output per clock period, since the slave flip-flop responds to only the single sampled input. A problem can still occur, however, because the master flip-flop can change more than once while it is asserted; thus, there is the potential for the master to sample at the wrong time. There is also the potential that either flip-flop can be affected by noise.

A master-slave, S-R clocked input flip-flop synthesized from NAND gates, Figure 18B, is accompanied by its logic symbol, Figure 18A. From the logic symbols you can tell that the output changes on a negative-going clock edge.

G3A and G3B form the master set-rest flip-flop, and G4A and G4B the slave flip flop. The input signals S and R are controlled by the positive going edge of the clock through gates G1A and G1B. G2A and G2B control the inputs into the slave flip-flop; these inputs are the outputs of the master flip-flop. Note G5 inverts the clock; thus while the positive-going edge places new data into the master flip-flop, the other edge of the clock transfers the output of the master into the slave on the following negative clock edge.

D FLIP-FLOP

In a D (data) flip-flop, the *data* input is transferred to the outputs when the flip-flop is enabled. The logic level at the D input is transferred to Q when the clock is positive; the Q output retains this logic level until the next positive clock pulse (see **Figure 19**). The truth table summarizes this operation. If D = 1 the next clock pulse makes Q = 1. If D = 0, the next clock pulse makes Q = 0. A D flip-flop is useful to store one bit of information. A collection of D flip-flops forms a register.

J-K FLIP-FLOP

The J-K flip-flop, shown schematically in **Figure 20A**, has five inputs. The unit shown uses both positive active inputs (the J and K inputs) and negative active inputs (note the bubbles on the C or clock, PR or preset and CL or clear inputs). With these inputs almost any other type of flip-flop may be synthesized.

The truth table of Figure 20B provides an explanation. Lines (rows) 1 and 2 show the preset and clear inputs and their use. These are active low, meaning that when one (and only one) of them goes to a logic 0, the flip-flop responds, just as if it was a S-R or set-reset flip-flop. Make PR a logic 0, and leave CL a logic 1, and the flip-flop goes into the Q = 1 state (line 1). Do the reverse (line 2) – PR = 1, CL = 0 and the flip-flop goes into a Q' = 1 state. When these two inputs are used, J, K and C are marked as X or don't care, because the PR and CL inputs override them. Line 3 corresponds to the unused state of the R-S flip-flop.

Line 5 shows that if J = 1 and K = 0, the next clock transition from high to low sets Q = 1 and Q' = 0. Alternately, line 4 shows J = 0 and K = 1 sets Q = 0 and Q' = 1. Therefore if a signal is applied to J, and the inverted signal sent to K, the J-K flip-flop will mimic a D flip-flop, echoing its input.

The most unique feature of the J-K flip-flop is line 7. If both J and K are connected to a 1, then each clock 1 to 0 transition will flip or toggle the flop-flop. Thus the J-K flip-flop can be used as a T flip-flop, as in a ripple counter (see the following **Counters** section.)

Table 8

D Flip-Flop		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

SR Flip-Flop			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

T Flip-Flop		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	1
1	1	0

JK Flip-Flop			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

HBK0920

5.3 Groups of Flip-Flops

COUNTERS

Groups of flip-flops can be combined to make counters. Intuitively, a counter is a circuit that starts at state 0 and sequences up through states 1, 2, 3, to m, where m is the maximum number of states available. From state m, the next state will return the counter to 0. This describes the most common counter: the *n-bit binary counter*, with n outputs corresponding to $2^n = m$ states. Such a counter can be made from n flip-flops, as shown in **Figure 21**. This figure shows implementations for each of the types of synchronicity. Both circuits pass the data count from stage to stage. In the asynchronous counter, Figure 21A, the clock is also passed from stage to stage and the circuit is called *ripple* or *ripple-carry*.

The J-K flip-flop truth table shows that with PR (Preset) and CL (Clear) both positive, and therefore not effecting the operation, the flip-flop will toggle if J and K are tied to a logic 1. In Figure 21A the first stage has its J and K inputs permanently tied to a logic 1, and each succeeding stage has its J and K inputs tied to Q of the proceeding stage. This provides a direct ripple counter implementation.

Design of a synchronous counter is bit more involved. It consists of determining, for a particular count, the conditions that will make the next stage change at the same clock edge when all the stages are changing.

To illustrate this, notice the binary counting table of Figure 21. The right-hand column represents the lowest stage of the counter. It alternates between 1 and 0 on every line. Thus, for the first stage the J and K inputs are tied to logic 1. This provides the alternation required by the counting table.

The middle column or second stage of the counter changes state right after the lower stage is a 1 (lines C, E and G). Thus if the Q output of the lowest stage is tied to the J and K inputs of the second stage, each time the output of the lowest stage is a 1 the second stage toggles on the next clock pulse.

Finally, the third column (third stage) toggles when both the first stage and the second stage are both 1s (line D). Thus by ANDing the Q outputs of the first two stages, and then connecting them to the J and K inputs of the third stage, the third stage will toggle whenever the first two stages are 1s.

There are formal methods for determining the wiring of synchronous counters. The illustration above is one manual method that may be used to design a counter of this type. The advantage of the synchronous counter is that at any instant, except during clock pulse transition, all counter stage outputs *are correct* and delay due to propagation through the flip flops is not a problem.

In the synchronous counter, Figure 21B, each stage is controlled by a common clock signal.

There are numerous variations on this first example of a counter. Most counters have the ability to *clear* the count to 0. Some counters can also *preset* to a desired count. The clear and preset control inputs are often asynchronous — they change the output state without being clocked. Counters may either count up (increment) or down (decrement). *Up/down* counters can be controlled to count in either direction. Counters can have sequences other than the standard numbers, for example a BCD counter.

Counters are also not restricted to changing state on every clock cycle. An n-bit counter that changes state only after m clock pulses is called a *divider* or *divide-by-m* counter. There are still $2^n = m$ states; however, the output after p clock pulses is now p / m. Combining different divide-by-m counters can result in almost any desired count. For example, a base 12 counter can be made from a divide-by-2 and a divide-by-6 counter; a base 10 (decade) counter consists of a divide-by-2 and a BCD divide-by-5 counter.

The outputs of these counters are binary. To produce output in decimal form, the output of a counter would be provided to a binary-to-decimal decoder chip and/or an LED display.

REGISTERS

Groups of flip-flops can be combined to make registers, usually implemented with D flip-flops. A *register* stores n bits of information, delivering that information in response to a clock pulse. Registers usually have asynchronous *set* to 1 and *clear* to 0 capabilities.

Storage Register

A storage register simply stores temporary information, for example, incoming information or intermediate results. The size is related to the basic size of information handled by a computer: 8 flip-flops for an 8-bit or *byte register* or 16 bits for a *word register*. **Figure 22** shows a typical circuit and schematic symbols for an 8-bit storage register.

Shift Register

Shift registers also store information and provide it in response to a clock signal, but they handle their information differently: When a clock pulse occurs, instead of each flip-flop passing its result to the output, the flip-flops pass their data to each other, up and down the row. For example, in up mode, each flip-flop receives the output of the preceding flip-flop. A data bit starting in flip-flop D0 in a left shifter would move to D1, then D2 and so on until it is shifted out of the register. If a 0 was input to the least significant bit, D0, on each clock pulse then, when the last data bit has been shifted out, the register contains all 0s.

Shift registers can be left shifters, right shifters or controlled to shift in either direction. The most general form, a *universal shift register*, has two control inputs for four states: Hold, Shift right, Shift left and Load. Most also have asynchronous inputs for preset, clear and parallel load. The primary use of shift registers is to convert parallel information to serial or vice versa. Additional uses for a shift register are to delay or synchronize data, and to multiply or divide a number by a factor 2^n. Data can be delayed simply by taking advantage of the Hold feature of the register control inputs. Multiplication and division with shift registers is best explained by example: Suppose a 4-bit shift register currently has the value 1000 = 8. A right shift

Figure 21 — Three-bit binary counter using J-K flip-flops: (A) asynchronous or ripple, (B) synchronous.

Binary Counting Table

	Q_2	Q_1	Q_0
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0
F	1	0	1
G	1	1	0
H	1	1	1

results in the new parallel output 0100 = 4 = 8 / 2. A second right shift results in 0010 = 2 = (8 / 2) / 2. Together the 2 right shifts performed a division by 2^2. In general, shifting right n times is equivalent to dividing by 2^n. Similarly, shifting left multiplies by 2^n. This can be useful to compiler writers to make a computer program run faster.

5.4 Multivibrators

Multivibrators are a general type of circuit with three varieties: bistable, monostable and astable. The only truly digital multivibrator is bistable, having two stable states. The flip-flop is a *bistable multivibrator*: both of its two states are stable; it can be triggered from one stable state to the other by an external signal. The other two varieties of multivibrators are partly analog circuits and partly digital. While their output is one or more pulses, the internal operation is strictly analog.

MONOSTABLE MULTIVIBRATOR

A *monostable* or *one-shot* multivibrator has one energy-storing element in its feedback paths, resulting in one stable and one quasi-stable state. It can be switched, or *triggered*, to its quasi-stable state; then returns to the stable state after a time delay. Thus, when triggered, the one-shot multivibrator puts out a pulse of some duration, T.

A very common integrated circuit used for non-precision generation of a signal pulse is the 555 timer IC. **Figure 23** shows a 555 connected as a one-shot multivibrator. The one-shot is activated by a negative-going pulse between the trigger input and ground. The trigger pulse causes the output (Q) to go positive and capacitor C to charge through resistor R. When the voltage across C reaches two-thirds of V_{CC}, the capacitor is quickly discharged to ground and the output returns to 0. The output remains at logic 1 for a time determined by

$$T = 1.1\ RC$$

where:
R = resistance in ohms, and
C = capacitance in farads.

A very common, but again, non-precision application of this circuit is the generation of a delayed pulse. If there is a requirement to generate a 50 µs pulse, but delayed from a trigger by 20 ms, two 555s might be used. The first 555, configured as an astable multivibrator, generates the 20-ms pulse, and the trailing edge of the 20-ms pulse is used to trigger a second 555 that in turn generates the 10 µs pulse. See the **Circuits and Components** chapter for more information on the 555 timer and related circuits.
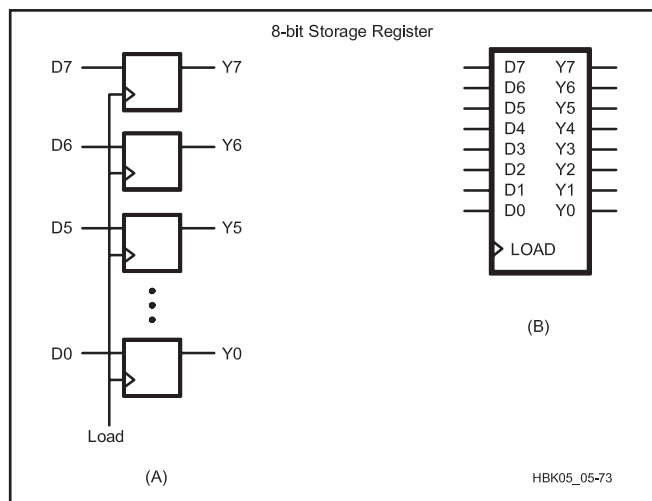

Figure 22 — An eight-bit storage register: (A) circuit and (B) schematic symbol.
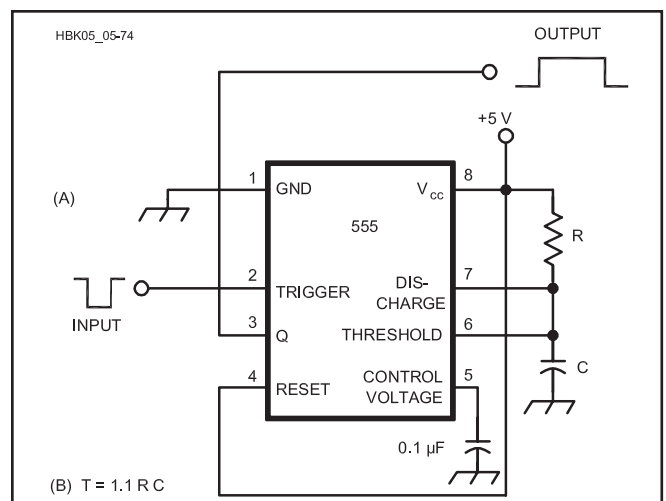

Figure 23 — (A) A 555 timer connected as a monostable multivibrator. (B) The equation to calculate values where T is the pulse duration in seconds, R is in ohms and C is in farads.

ASTABLE MULTIVIBRATOR

An *astable* or *free-running* multivibrator has two energy-storing elements in its feedback paths, resulting in two quasi-stable states. It continuously switches between these two states without external excitation. Thus, the astable multivibrator puts out a sequence of pulses. By properly selecting circuit components, these pulses can be of a desired frequency and width.

Figure 24 shows a 555 timer IC connected as an astable multivibrator. The capacitor C charges to two-thirds V_{CC} through R1 and R2 and discharges to one-third V_{CC} through R2. The ratio R1 : R2 sets the asserted high duty cycle of the pulse: t_{HIGH} / t_{PERIOD}. The output frequency is determined by:

$$f = \frac{1.46}{(R1 + 2\,R2)\,C}$$

where:
R1 and R2 are in ohms,
C is in farads and
f is in hertz.

It may be difficult to produce a 50% duty cycle due to manufacturing tolerance for the resistors R1 and R2. One way to ensure a 50% duty cycle is to run the astable multivibrator at 2f and then divide by 2 with a toggle flip-flop.

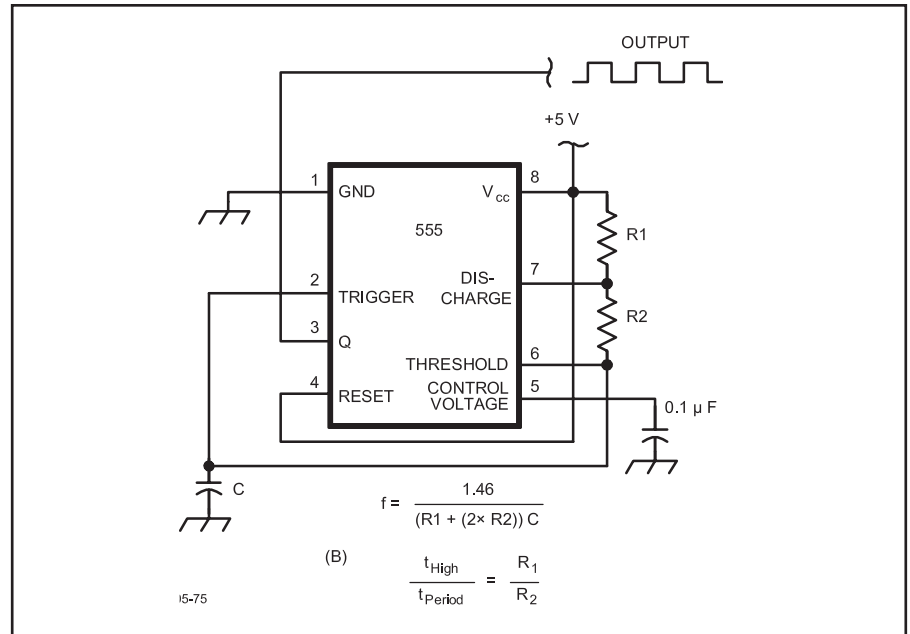Astable multivibrators, and the 555 integrated circuit in particular, are very often used to generate clock pulses. Although this is a very inexpensive and minimum hardware approach, the penalty is stability with temperature. Since the frequency and the pulse dimensions are set by resistors and capacitors, drift with temperature and to some extent aging of components will result in changes with time. This is no different than the problem faced by designers of L-C controlled VFOs.

Figure 24 — (A) A 555 timer connected as an astable multivibrator. (B) The equations to calculate values for R1, R2 in ohms and C in farads, where f is the clock frequency in Hertz.

6 Digital Integrated Circuits

Integrated circuits (ICs) are the cornerstone of digital logic devices. Modern technology has enabled electronics to become smaller and smaller in size and less and less expensive. Much of today's complex digital equipment would be impossible to build with discrete transistors and discrete components.

An IC is a miniature electronic module of components and conductors manufactured as a single unit. All you see is a ceramic or black plastic package and the silver-colored pins sticking out. Inside the package is a piece of material, usually silicon, created (fabricated) in such a way that it conducts an electric current to perform logic functions, such as a gate, flip-flop or decoder.

As each generation of ICs surpassed the previous one, they became classified according to the number of gates on a single chip. These classifications are roughly defined as:

Small-scale integration (SSI):
10 or fewer gates on a chip.

Medium-scale integration (MSI):
10-100 gates.

Large-scale integration (LSI):
100-1000 gates.

Very-large-scale integration (VLSI):
1000 or more gates.

Though SSI and MSI logic chips are still useful for building circuits to handle very simple tasks, it is more common to see them either used along with or completely replaced by programmable logic arrays and microcontrollers. In many cases you will see the smaller logic circuits referred to as "glue logic."

6.1 Comparing Logic Families

When selecting devices for a circuit, a designer is faced with choosing between many families and subfamilies of logic ICs. The determination of which logic subfamily is right for a specific application is based upon several desirable characteristics: logic speed, power consumption, fan-out, noise immunity and cost. From a practical viewpoint, the primary IC families available and in common use today are CMOS, with TTL a distant second place. Within these families, there are tradeoffs that can be made with respect to individual circuit capabilities, especially in the areas of speed and power consumption. Except under the most demanding circumstances, normal commercial grade temperature rating will do for amateur
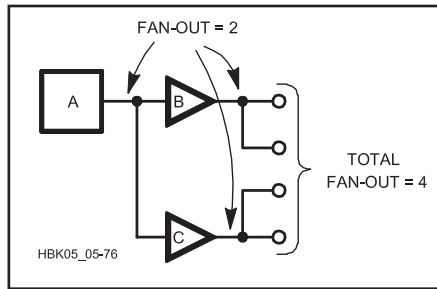
Figure 25 — Nonverting buffers used to increase fan-out: Gate A (fan-out = 2) is connected to two buffers, B and C, each with a fan-out of 2. Result is a total fan-out of

service.

FAN-OUT

Fan-out is a term with which you will need to become familiar when working with TTL logic families such as 7400, 74LS or 74S. A gate output can supply only a limited amount of current. Therefore, a single output can only drive a limited number of inputs. The measure of driving ability is called fan-out, expressed as the number of inputs (of the same subfamily) that can be driven by a single output. If a logic family that is otherwise desirable does not have sufficient fan-out, consider using noninverting buffers to increase fan-out, as shown by **Figure 25**.

Another approach is to use a CMOS logic family. These families typically have output drivers capable of sourcing or sinking 20 to 25 mA, and input current leakage in the microampere range. Thus, fan-out is seldom a problem when using these devices.

NOISE IMMUNITY

The noise margin was illustrated in Figure 2. The choice of voltage levels for the binary states determines the noise margin. If the gap is too small, a spurious signal can too easily produce the wrong state. Too large a gap, however, produces longer, slower transitions and thus decreased switching speeds.

Circuit impedance also plays a part in noise immunity, particularly if the noise is from external sources such as radio transmitters. At low impedances, more energy is needed to change a given voltage level than at higher impedances.

6.2 Bipolar Logic Families

Two broad categories of digital logic ICs are *bipolar* and *metal-oxide semiconductor* (MOS). Numerous manufacturing techniques have been developed to fabricate each type. Each surviving, commercially available family has its particular advantages and disadvantages and has found its own special niche in the market. The designer is cautioned, however, that sometimes this niche is simply the

ongoing maintenance of old products. There are still very old logic families available for reasonable prices that would be considered quite obsolete and generally not suitable for new designs.

Bipolar semiconductor ICs usually employ NPN junction transistors. (Bipolar ICs can be manufactured using PNP transistors, but NPN transistors make faster circuits.) While early bipolar logic was faster and had higher power consumption than MOS logic, the speed difference has largely disappeared as manufacturing technology has developed.

There are several families of bipolar logic devices, and within some of these families there are subfamilies. The most-used bipolar logic family is transistor-transistor logic (TTL). Another bipolar logic family, Emitter Coupled Logic (ECL), has exceptionally high speed but high power consumption.

TRANSISTOR-TRANSISTOR LOGIC (TTL)

The TTL family saw widespread acceptance through the 1960s, 1970s and 1980s because it was fast compared to early MOS and CMOS logic, and has good noise immunity. It was by far the most commonly used logic family for a couple of decades. Though TTL logic is not in widespread use today for new designs, the device numbering system devised for TTL chips survives to this day for newer technologies. You will also often see TTL, especially the later low power, higher speed TTL subfamilies, in various equipment you may use and repair.

TTL Subfamilies

The original standard TTL used bipolar transistors and "totem-pole" outputs (see **Figure 26A** and B), which were a great improvement over the earlier diode-transistor logic (DTL) and resistor-transistor logic (RTL). Still, TTL logic consumed quite a bit of power even at idle, and there were limits on how

many inputs could be driven by a single output. Later versions used Schottky diodes to greatly improve switching speed, and reduced power requirements were introduced.

TTL IC identification numbers begin with either 54 or 7 The 54 prefix denotes an extended military temperature range of –55 to 125 °C, while 74 indicates a commercial temperature range of 0 to 70 °C. The next letters, in the middle of the TTL device number, indicate the TTL subfamily. Following the subfamily designation is a 2, 3 or 4-digit device-identification number. For example, a 7400 is a standard TTL NAND gate and a 74LS00 is a low-power Schottky NAND gate (The NAND gate is the workhorse TTL chip). A partial list of TTL subfamilies includes:

|		
	74xx	standard TTL
H	74Hxx	High-speed
L	74Lxx	Low-power
S	74Sxx	Schottky
F	74Fxx	Fairchild Advanced Schottky
LS	74LSxx	Low-power Schottky
AS	74ASxx	Advanced Schottky
ALS	74ALSxx	Advanced Low-power Schottky

Each subfamily is a compromise between speed and power consumption. **Table 5.9** shows some of these characteristics. Because the speed-power product is approximately constant, less power consumption generally results in lower speed and vice versa. The advanced low power Schottky devices (ALS, F) offer both increased speed and reduced power consumption. Historically, an additional consideration to the speed-versus-power trade-off has been the cost trade-off. For the amateur, this is not nearly the factor it once was as component costs are relatively low for the newer, faster, lower powered parts.

When a TTL gate changes state, the amount of current that it draws changes rapidly. These changes in current, called switching tran-

Table 9
TTL and CMOS Subfamily Performance Characteristics

TTL Family	Propagation Delay (ns)	Per Gate Power Consumption (mW)	Speed Power Product (pico-joules)
Standard	9	10	90
L	33	1	33
H	6	22	132
S	3	20	60
F	3	8.5	25.5
LS	9	2	18
AS	1.6	20	32
ALS	5	1.3	6.5

CMOS Family Operating with 5 < V_{CC}	<5.5 V	f=100 kHz	f=1 MHz	f=10 MHz	f=100 kHz	f=1 MHz	f=10 MHz
HC	18	0.0625	0.6025	6.0025	1.1	10.8	108
HCT	18	0.0625	0.6025	6.0025	1.1	10.8	108
AC	5.25	0.080	0.755	7.505	0.4	3.9	39
ACT	75	0.080	0.755	7.505	0.4	3.6	36
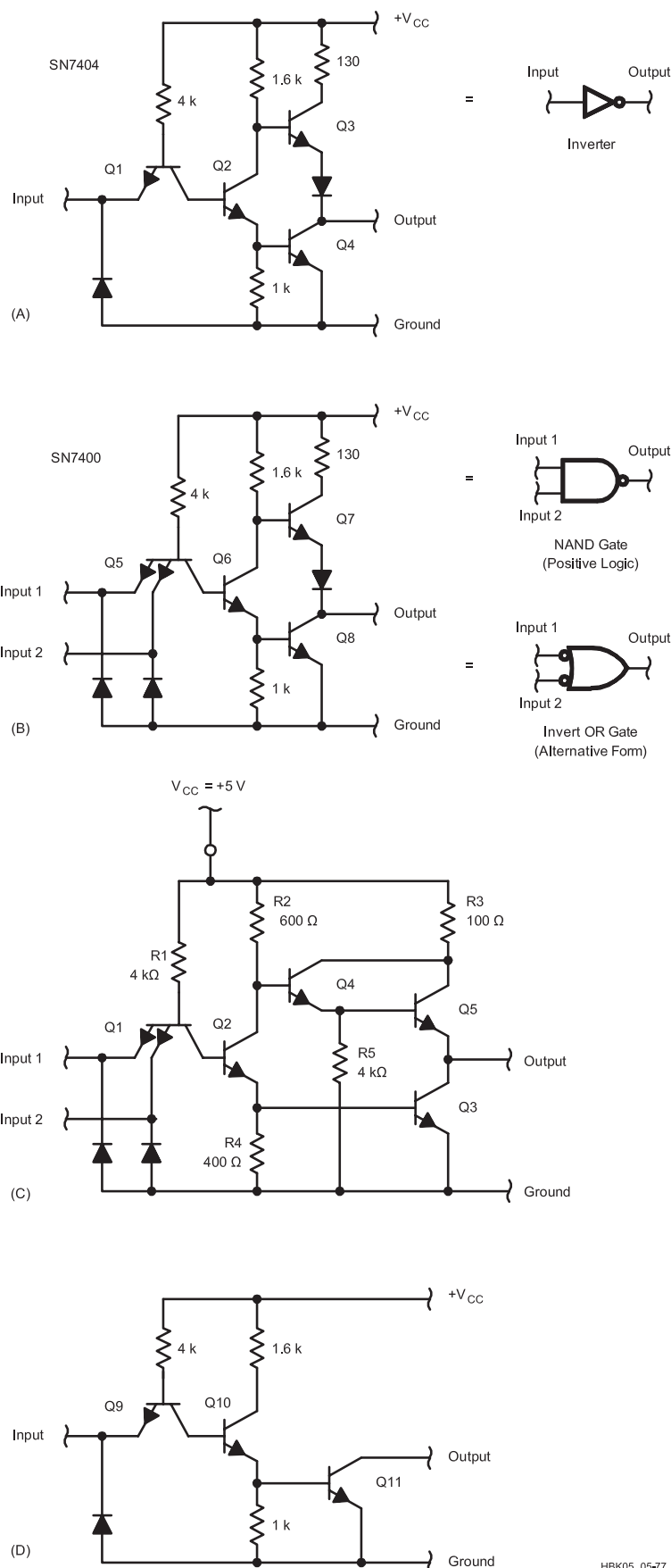
sients, appear on the power supply line and can cause false triggering of other devices. For this reason, the power bus should be adequately decoupled. For proper decoupling of TTL circuits, connect a 0.01 to 0.1 µF capacitor from V_{CC} to ground near each device to minimize the transient currents caused by device switching and magnetic coupling. These capacitors must be low-inductance, high-frequency RF capacitors (ceramic capacitors are preferred). In addition, a large-value (50 to 100 µF) capacitor should be connected from V_{CC} to ground somewhere on the board to accommodate the continually changing I_{CC} requirements of the total V_{CC} bus line. These are generally low-inductance tantalum capacitors.

Darlington and Open-Collector Outputs

Figure 26C and D show variations from the totem-pole configuration. They are the Darlington transistor pair and the open-collector configuration respectively.
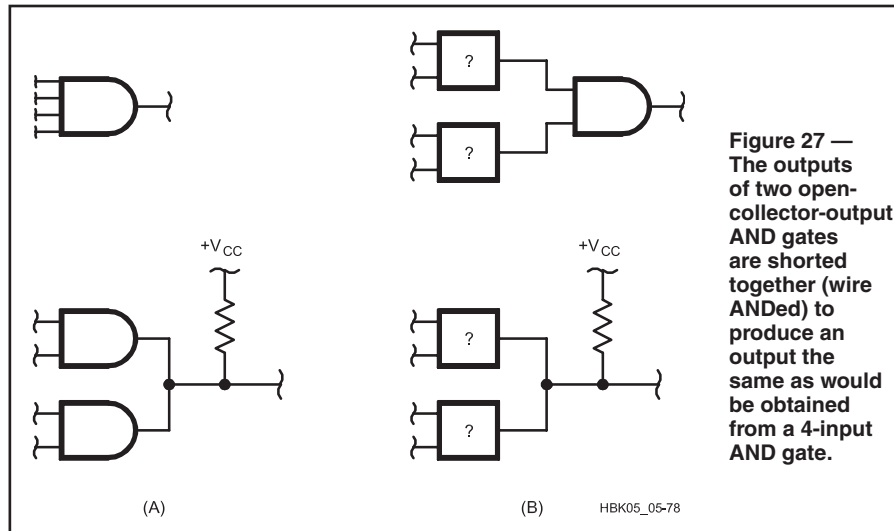
The Darlington pair configuration replaces the single transistor Q4 with two transistors, Q4 and Q5. The effect is to provide more current-sourcing capability in the high state. This has two benefits: (1) the rise time is decreased and (2) the fan-out is increased.

Transistor(s) on the output in both the totem-pole and Darlington configurations provide active pull-up. Omitting the transistor(s) and providing an external resistor for passive pull-up gives the open-collector configuration. This configuration, unfortunately, results in slower rise time, since a relatively large external resistor must be used. The technique has some very useful applications, however: driving other devices, performing wired logic, busing and interfacing between logic devices.

Devices that need other than a 5-V supply can be driven with the open-collector output by substituting the device for the external resistor. Example devices include LEDs, relays and solenoids. Inductive devices like relay coils and solenoids need a protection diode across the coil. You must pay attention to the current ratings of open-collector outputs in such applications. You may need a switching transistor to drive some relays or other high-current loads.

Open-collector outputs can perform

Figure 26 — Example TTL circuits and their equivalent logic symbols: (A) an inverter and (B) a NAND gate, both with totem-pole outputs. (C) A NAND gate with a Darlington output. (D) A NAND gate with an open-collector output. (Indicated resistor values are typical. Identification of transistors is for text reference only. These are not discrete components but parts of the silicon die.)

HBK05_05-77

Figure 27 — The outputs of two open-collector-output AND gates are shorted together (wire ANDed) to produce an output the same as would be obtained from a 4-input AND gate.

(A) (B) HBK05_05-78

wired logic, rather than gated IC logic, by wire-ANDing the outputs. This can save the designer an AND gate, potentially simplifying the design. Wire-ANDed outputs are several open-collector outputs connected to a single external pull-up resistor. The overall output, then, will only be high when all pull-down transistors are OFF (all connected outputs are high), effectively performing an AND of the connected outputs. If any of the connected outputs are low, the output after the external resistor will be low. **Figure 27** illustrates the wire-ANDing of open-collector outputs.

The wire-ANDed concept can be applied to several devices sharing a common bus. At any time, all but one device has a high-impedance (off) output. The remaining device, enabled with control circuitry, drives the bus output.

Open-collector outputs are also useful for interfacing TTL gates to gates from other logic families. TTL outputs have a minimum high level of 2.4 V and a maximum low level of 0.4 V. When driving non-TTL circuits, a pull-up resistor (typically 2.2 kΩ) connected to the positive supply can raise the high level to 5 V. If a higher output voltage is needed, a pull-up resistor on an open-collector output can be connected to a positive supply greater than 5 V, so long as the chip output voltage and current maximums are not exceeded.

Three-State Outputs

While open-collector outputs can perform bus sharing, a more popular method is three-state output, or tristate, devices. The three states are low, high and high impedance, also called Hi-Z or *floating*. An output in the high-impedance state behaves as if it is disconnected from the circuit, except for possibly a small leakage current. Three-state devices have an additional disable input. When the enable input is active, the device provides high and low outputs just as it would normally; when enable is inactive, the device goes into its high-impedance state.

A bus is a common set of wires, usually used for data transfer. A three-state bus has several three-state outputs wired together. With control circuitry, all devices on the bus but one have outputs in the high-impedance state. The remaining device is enabled, driving the bus with high and low outputs. Care should be taken to ensure only one of the output devices can be enabled at any time, since simultaneously connected high and low outputs may result in an incorrect logic voltage. (The condition when more than one driver is enabled at the same time is called *bus contention*.) Also, the large current drain from V_{CC} to ground through the high driver to the low driver can potentially damage the circuit or produce noise pulses that can affect overall system behavior.

Unused TTL Inputs

A design may result in the need for an n-input gate when only an n + m input gate is available. In this case, the recommended solution for extraneous inputs is to give the extra inputs a constant value that won't affect the output. A low input is easily provided by connecting the input to ground. A high input can be provided with either an inverter whose input is ground or with a pull-up resistor. The pull-up resistor is preferred rather than a direct connection to power because the resistor limits the current, thus protecting the circuit from transient voltages. Usually, a 1-kΩ to 5-kΩ resistor is used; a single 1-kΩ resistor can handle up to 10 inputs.

It's important to properly handle all inputs. Design analysis would show that an unconnected, or floating TTL input is usually high but can easily be changed low by only a small amount of capacitively-coupled noise.

6.3 Metal Oxide Semiconductor (MOS) Logic Families

While bipolar devices use junction transistors, MOS devices use field effect transistors (FETs). MOS is characterized by simple device structure, small size (high density) and ease of fabrication. MOS circuits use the NOR gate as the workhorse chip rather than the NAND. MOS families, specifically CMOS, are used extensively in most digital devices today because of their low power consumption and high speed.

P-CHANNEL MOS (PMOS)

The first MOS devices to be fabricated were PMOS, conducting electrical current by the flow of positive charges (holes). PMOS power consumption is much lower than that of bipolar logic, but its operating speed is also lower. The only extensive use of PMOS was in calculators and watches, where low speed is acceptable and low power consumption and low cost are desirable. PMOS was replaced by NMOS, which offered substantially higher switching speeds.

N-CHANNEL MOS (NMOS)

With improved fabrication technology, NMOS became feasible and provided improved performance and TTL compatibility. The speed of NMOS is at least twice that of PMOS, since electrons rather than holes carry the current. NMOS also has greater gain than PMOS and supports greater packaging density through the use of smaller transistors. NMOS has been almost completely obsoleted by CMOS.

COMPLEMENTARY MOS (CMOS)

CMOS combines both P-channel and N-channel devices on the same substrate to achieve high noise immunity and low power consumption: less than 1 mW per gate and negligible power during standby. This accounts for the widespread use of CMOS in battery-operated equipment. The high impedance of CMOS gates makes them susceptible to electromagnetic interference, however, particularly if long traces are involved. Consider a trace ¼-wavelength long between input and output. The output is a low-impedance point, hence the trace is effectively grounded at this point. You can get high RF potentials ¼-wavelength away, which disturbs circuit operation.

A notable feature of CMOS devices is that the logic levels swing to within a few millivolts of the supply voltages. The input-switching threshold is approximately one half the supply voltage ($V_{DD} - V_{SS}$). This characteristic contributes to high noise immunity on the input signal or power supply lines. CMOS input-current drive requirements are

minuscule, so the fan-out is great, at least in low-speed systems. For high-speed systems, the input capacitance increases the dynamic power dissipation and limits the fan-out.

CMOS Subfamilies

There are a large number of CMOS subfamilies available. Like TTL, the original CMOS has largely been replaced by later subfamilies using improved technologies; in turn, these will be replaced with even newer families as technology evolves. The original family, called the 4000 series, has numbers beginning with 40 or 45 followed by two or three numbers to indicate the specific device. 4000B is second generation CMOS. When introduced, this family offered low power consumption but was fairly slow and not easy to interface with TTL.

Later CMOS subfamilies offer improved performance and, in some cases, TTL compatibility. For simplicity, the later subfamilies were given numbers similar to the TTL numbering system, with the same leading numbers, 54 or 74, followed by 1 to 4 letters indicating the subfamily and as many as 5 numbers indicating the specific device. The subfamily letters usually include a "C" to distinguish them as CMOS.

Following is a description of some of the CMOS device families available. As there are a substantial number of families offered by specific manufacturer, and there are new families being introduced frequently, this information is by no means exhaustive. A check of IC suppliers' and manufacturers' Web sites will provide the designer with a complete selection of choices to meet his or her requirements.

4000	*4071B*	*Standard CMOS*
C	*74Cxx*	*CMOS versions of TTL*

Largely obsolete today, devices in the 74C subfamily are pin and functional equivalents of many of the most popular parts in the 7400 TTL family. It may be possible to replace all TTL ICs in a particular circuit with 74C-series CMOS, but this family should not be mixed with TTL in a circuit without careful design considerations. Devices in the C series are typically 50% faster than the 4000 series.

HC   74HCxx   High-speed CMOS

Devices in the 74HC subfamily have speed and drive capabilities similar to Low-power Schottky (LS) TTL but with better noise immunity and greatly reduced power consumption. High-speed refers to faster than the previous CMOS family, the 4000-series.

HCT   74HCTxx   High-Speed CMOS, TTL compatible

Devices in this subfamily were designed to interface TTL to CMOS systems. The HCT inputs recognize TTL levels, while the outputs are CMOS compatible. HCT chips are commonly used as lower powered, drop-in replacements for their pin compatible LS TTL counterparts.

AC   74ACxxxxx   Advanced CMOS

Devices in this family have reduced propagation delays, increased drive capabilities and can operate at higher speeds than standard CMOS. They are comparable to Advanced Low-power Schottky (ALS) TTL devices.

ACT   74ACTxxxxx   Advanced CMOS, TTL compatible

This subfamily combines the improved performance of the AC series with TTL-compatible inputs.

AHCT   74AHCTxxxx   Advanced High Speed CMOS, TTL Compatible

This subfamily is substantially faster than HC and HCT, but only slightly faster than ACT in switching speed. It has lower output drive capacity than AC/ACT.

New CMOS subfamilies are being introduced regularly. The current move is to lower voltage operation; where 5 V was the most common supply voltage until a few years ago, 3.3 V and lower supplies are becoming more common. Many newer microprocessors, microcontrollers and intelligent peripheral chips require 3.3 V logic.

LVC   74LVCxxxx   Low Voltage CMOS

This subfamily uses a 3.3 V supply rather than 5 V. It offers low propagation delay (under 5 ns typical), extremely low supply current, robust output drive capabilities, and 5-V TTL compatible inputs.

ALVC   74ALVCxxxx   Advanced Low Voltage CMOS

Even faster than LVC, this family offers propagation delays if under 3ns. Current demands are slightly higher, but still lower than 5V CMOS.

VCX   74VCXxxxx   Low voltage CMOS

This family operates at supply voltages of 1.8 or 3.6 V. Offering high switching speeds and low propagation delays, it does not have TTL-compatible buffered inputs.

As with TTL, each CMOS subfamily has characteristics that may make it suitable or unsuitable for a particular design. You should consult the manufacturer's data books for complete information on each subfamily being considered.

CMOS Circuits

A simplified diagram of a CMOS logic inverter is shown in **Figure 28**. When the input is low, the resistance of Q2 is low so a high current flows from V_{CC}. Since Q1's resistance is high, Q2 sinks current from the output, pulling it low. When the input is high, the opposite occurs: Q1's resistance is low, Q2's is high and Q1 sources current to the output, driving it high. The diodes are to protect the circuit against static charges.

Special Considerations

Some of the diodes in the input- and output-protection circuits are an inherent part of the manufacturing process. Even with the protection circuits, however, CMOS ICs are susceptible to damage from static charges. To protect against damage from static, the pins should not be inserted in Styrofoam as is sometimes done with other components. Instead, a spongy conductive foam, usually black or pink in color, is available for this purpose. Before removing a CMOS IC from its protective material, make certain that your body is grounded. Touching nearly any large metal object before handling the ICs is probably adequate to drain any static charge off your body. Some people prefer to touch a grounded metal object or to use a conductive bracelet connected to the ground terminal of a three-wire ac outlet through a 10-MΩ resistor. Since wall outlets aren't always wired properly, you should measure the voltage between the ground terminal and any metal objects you might touch. Connecting yourself to ground through a 1-MΩ to 10-MΩ resistor will limit
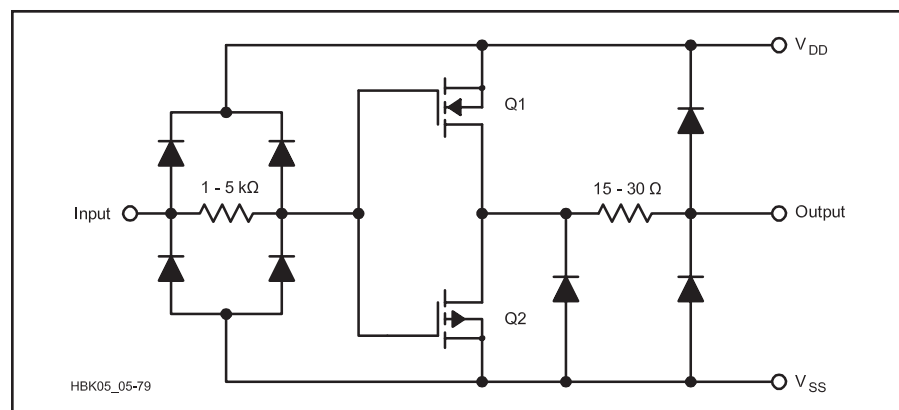

Figure 28 — Internal structure of a CMOS inverter.

any current that might flow through your body.

All CMOS inputs should be tied to an input signal. A positive supply voltage or ground is suitable if a constant input is desired. Undetermined CMOS inputs, even on unused gates, may cause gate outputs to oscillate. Oscillating gates draw high current, and may overheat and self destruct.

The low power consumption of CMOS ICs made them attractive for satellite applications, but standard CMOS devices proved to be sensitive to low levels of radiation — cosmic rays, gamma rays and X rays. Later, radiation-hardened CMOS ICs, able to tolerate 10^6 rads, made them suitable for space applications. (A rad is a unit of measurement for absorbed doses of ionizing radiation, equivalent to 10^{-2} joules per kilogram.)

SUMMARY

There are many types of logic ICs, each with its own advantages and disadvantages. Regardless of the application, consult up-to-date product specification sheets and manufacturer literature when designing logic circuits. IC data sheets, application notes, databooks and more are available from IC manufacturers via their Web sites. By using a search engine and entering a few key word specifications, you will locate application notes, tutorials and a host of other information.

6.4 Interfacing Logic Families

Each semiconductor logic family has its own advantages in particular applications. When a design mixes ICs from different logic families, the designer must account for the differing voltage and current requirements each logic family recognizes. The designer must ensure the appropriate interface exists between the point at which one logic family ends and another begins. Knowledge of the specific input/output (I/O) characteristics of each device is necessary, and knowledge of the general internal structure is desirable to ensure reliable digital interfaces. Typical internal structures have been illustrated for each common logic family. **Figure 29** illustrates the logic level changes for different TTL and CMOS families. Data sheets should be consulted for manufacturer's specifications.

Often more than one conversion scheme is possible, depending on whether the designer wishes to optimize power consumption or speed. Usually one quality must be traded off for the other. The following section discusses some specific logic conversions. Where an electrical connection between two logic systems isn't possible, an optoisolator can sometimes be used.
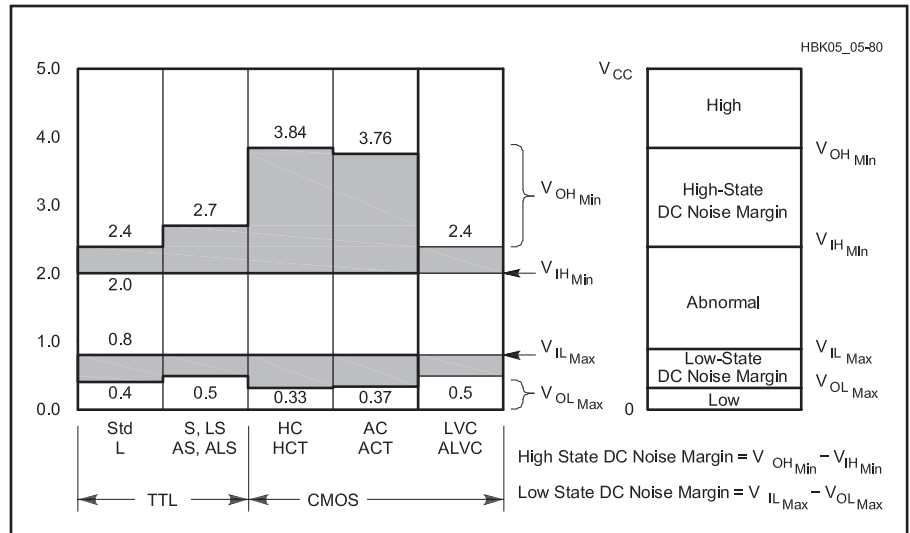

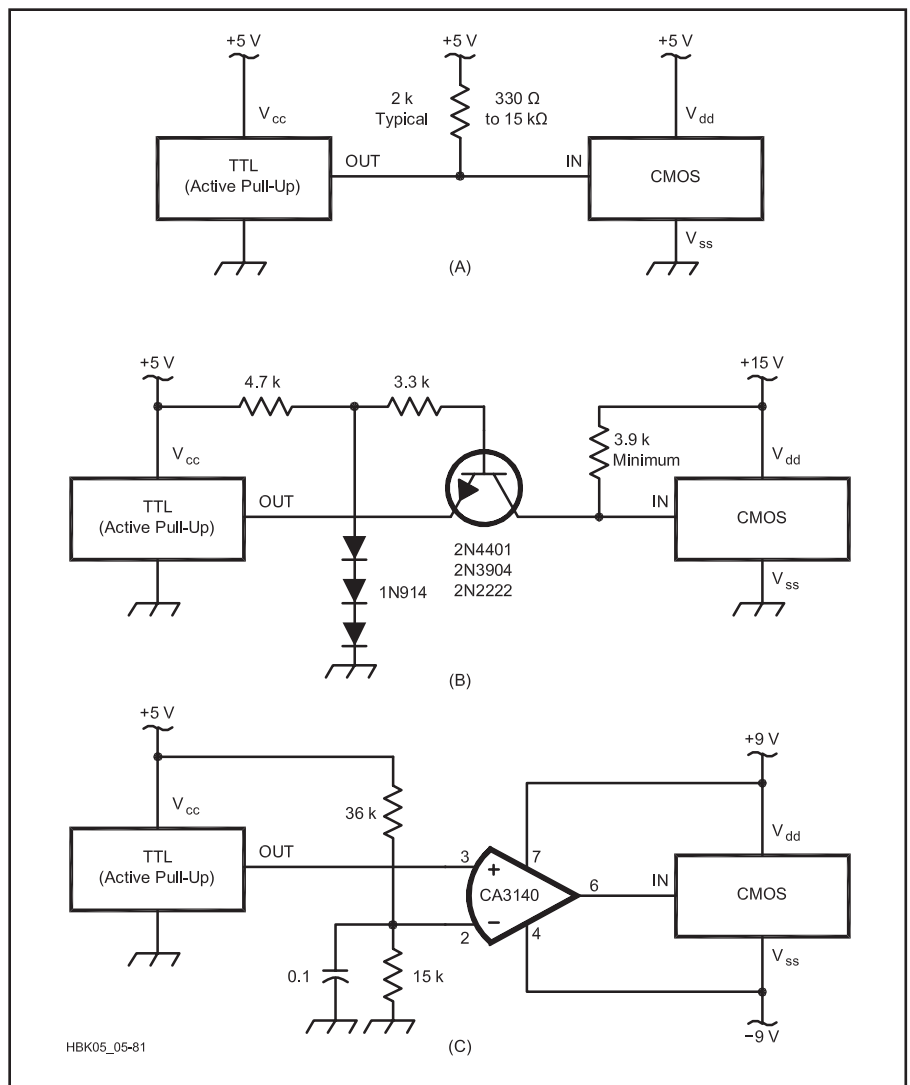Figure 29 — Differences in logic levels for some TTL and CMOS families.

Figure 30 — TTL to CMOS interface circuits: (A) pull-up resistor, (B) common-base level shifter and (C) op amp configured as a comparator.

TTL DRIVING CMOS

TTL and low-power TTL can drive 74C series CMOS directly over the commercial temperature range without an external pull-up resistor. However, they cannot drive 4000-series CMOS directly, and for HC-series devices, a pull-up resistor is recommended. The pull-up resistor, connected between the output of the TTL gate and V_{CC} as shown in **Figure 30A**, ensures proper operation and enough noise margin by making the high output equal to V_{DD}. Since the low output voltage will also be affected, the resistor value must be chosen with both desired high and low voltage ranges in mind. Resistor values in the range 1.5 kΩ to 7 kΩ should be suitable for all TTL families under worst conditions. A larger resistance reduces the maximum possible speed of the CMOS gate; a lower resistance generates a more favorable RC product but at the expense of increased power dissipation.

HCT-series and ACT-series CMOS devices were specifically designed to interface non-CMOS devices to a CMOS system. An HCT device acts as a simple buffer between the non-CMOS (usually TTL) and CMOS device and may be combined with a logic function if a suitable HCT device is available.

When the CMOS device is operating from a power supply other than +5 V, the TTL interface is more complex. One fairly simple technique uses a TTL open-collector output connected to the CMOS input, with a pull-up resistor from the CMOS input to the CMOS power supply. Another method, shown in Figure 30B, is a common-base level shifter. The level shifter translates a TTL output signal to a +15 V CMOS signal while preserving the full noise immunity of both gates. An excellent converter from TTL to CMOS using dual power supplies is to configure an operational amplifier as a comparator, as shown in Figure 30C. An FET op amp is shown because its output voltage can usually swing closer to the rails (+ and – supply voltages) than a bipolar device.

CMOS DRIVING TTL

Certain CMOS devices (including most modern 5 V powered CMOS logic families) can drive TTL loads directly. The output voltages of CMOS are compatible with the input requirements of TTL, but the input-current requirement of TTL limits the number of TTL loads that a CMOS device can drive from a single output (the fan-out).

Interfacing CMOS to TTL is a bit more complicated when the CMOS is operating at a voltage other than +5 V. One technique is shown in **Figure 31A**. The diode blocks the high voltage from the CMOS gate when it is in the high output state. A germanium diode is preferred because its lower forward-voltage drop provides higher noise immunity
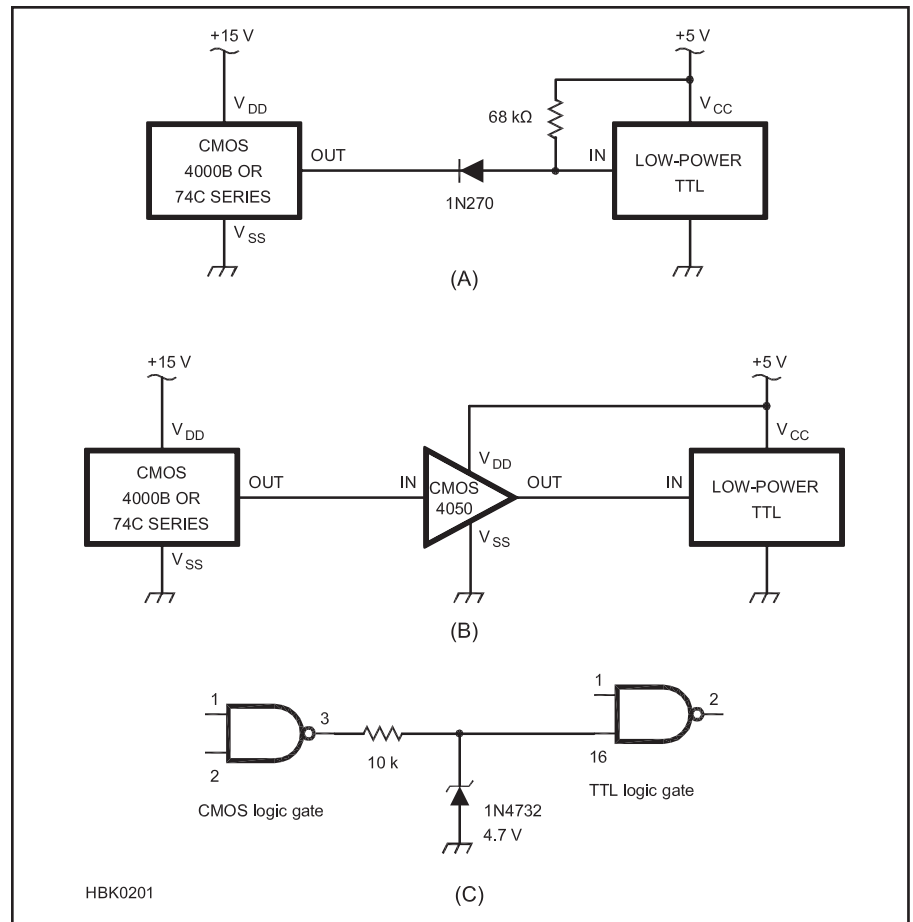

Figure 31 — CMOS to TTL interface circuits. (A) blocking diode chosen when different supply voltages are used. The diode is not necessary if both devices operate with a +5 V supply. (B) CMOS noninverting buffer IC. (C) Resistor/Zener circuit clamps the voltage to the TTL input at 7 V.

for the TTL device in the low state. The 68-kΩ resistor pulls the input high when the diode is reverse biased.

A simple resistor/Zener circuit, shown in Fig. 31C, can also be used. This clamps the voltage to the TTL input at 7 V.

There are two CMOS devices specifically designed to interface CMOS to TTL when TTL is using a lower supply voltage. The CD4050 is a noninverting buffer that allows its input high voltage to exceed the supply voltage. This capability allows the CD4050 to be connected directly between the CMOS and TTL devices, as shown in Figure 31B. The CD4049 is an inverting buffer that has the same capabilities as the CD4050.

5 V DRIVING 3.3 V LOGIC

Low voltage logic operating with 3.3 to 3.6 V supplies is becoming more and more common. Some of these logic families have 5 V tolerant inputs and can be driven directly by TTL levels; others require buffering to keep inputs at or below the supply voltage. Output levels may or may not be sufficient to drive TTL level inputs reliably under all conditions.

The use of TTL compatible buffers (74LVC, 74ALVC) on input signals is a safe way to drive these devices. Since the low logic voltages are compatible, a simple Zener clamp arrangement may also be sufficient.

3.3 V DRIVING 5 V LOGIC

Output voltages of most 3.3 V logic families are sufficient to drive the inputs of TTL-level devices. In some cases, the high level output voltage of a 3.3 V powered device may approach the lower end of the TTL level device's input range. In these cases, a pull-up resistor to 3.3 V will usually be sufficient. It is also possible to use an IC or transistor buffer.

6.5 Programmable Logic

As digital logic designs became more and more complex, the size and power consumption of their implementations grew. PLDs (*Programmable Logic Devices*) were introduced to allow the designer to produce application-specific logic without the expense and delay of fabricating custom chips. As explained earlier, the design of a logic circuit

begins with a description; this description is used to determine an equation; then the equation is expanded to components. With programmable logic we can shortcut this process by implementing the equations themselves, programming them into a generic array of gates.

Several families of programmable logic devices have been introduced over the years. The earliest were Programmable Array Logic (PAL) chips, consisting of an array of gates whose inputs, interconnections and outputs could be configured by blowing internal fuses according to a specific design. PALs were followed by Generic Array Logic (GALs), which have more gates and more flexible I/O pin logic. GALs are also erasable and reprogrammable.

Later developments led to the Field Programmable Gate Array (FPGA) and other programmable logic arrays. These devices can contain anywhere from several hundred to millions of logic gates, and up to more than a thousand I/O pins. The connections between gates, and thus the device's function, is determined by complex program code. PLDs can be made to replace large numbers of individual SSI, MSI and even LSI chips, up to and including entire microprocessor cores.

7 Analog-Digital Interfacing

Quite often, logic circuits must either drive or be driven from non-logic sources. A very common requirement is sensing the presence or absence of a high (as compared to +5 V) voltage or perhaps turning on or off a 120 V ac device or moving the motor in an antenna rotator. A similar problem occurs when two different units in the shack must be interfaced because induced ac voltages or ground loops can cause problems with the desired signals.

A slow speed but safe way to interface such circuits is to use a relay. This provides absolute isolation between the logic circuits and the load. **Figure 32A** shows the correct way to provide this connection. The relay coil is selected to draw less than the available current from the driving logic circuit. The diode, most often a 1N914 or equivalent switching diode, prevents the inductive load from back-biasing the logic circuit and possibly destroying it.

It is often not possible to find a relay that meets the load requirements *and* has a coil that can be driven directly from the logic output. Figure 32B shows two methods of using transistors to allow the use of higher power relays with logic gates.

Electro-optical couplers such as optoisolators and solid state relays can also be used for this circuit interfacing. Figure 32C uses an optoisolator to interface two sets of logic circuits that must be kept electrically isolated, and Figure 32D uses a solid state relay to control an ac line supply to a high current load. Note that this example uses a solid state relay with internal current limiting on the input side; the LED input has an impedance of approximately 300 Ω. Some devices may need a series resistor to set the LED current; always consult the device data sheet to avoid exceeding device limits of the relay or the processor's I/O pin. See the **Circuits and Components** chapter for more information on optoisolators.

For safely using signals with voltages higher than logic levels as inputs, the same simple resistor and Zener diode circuit similar to that
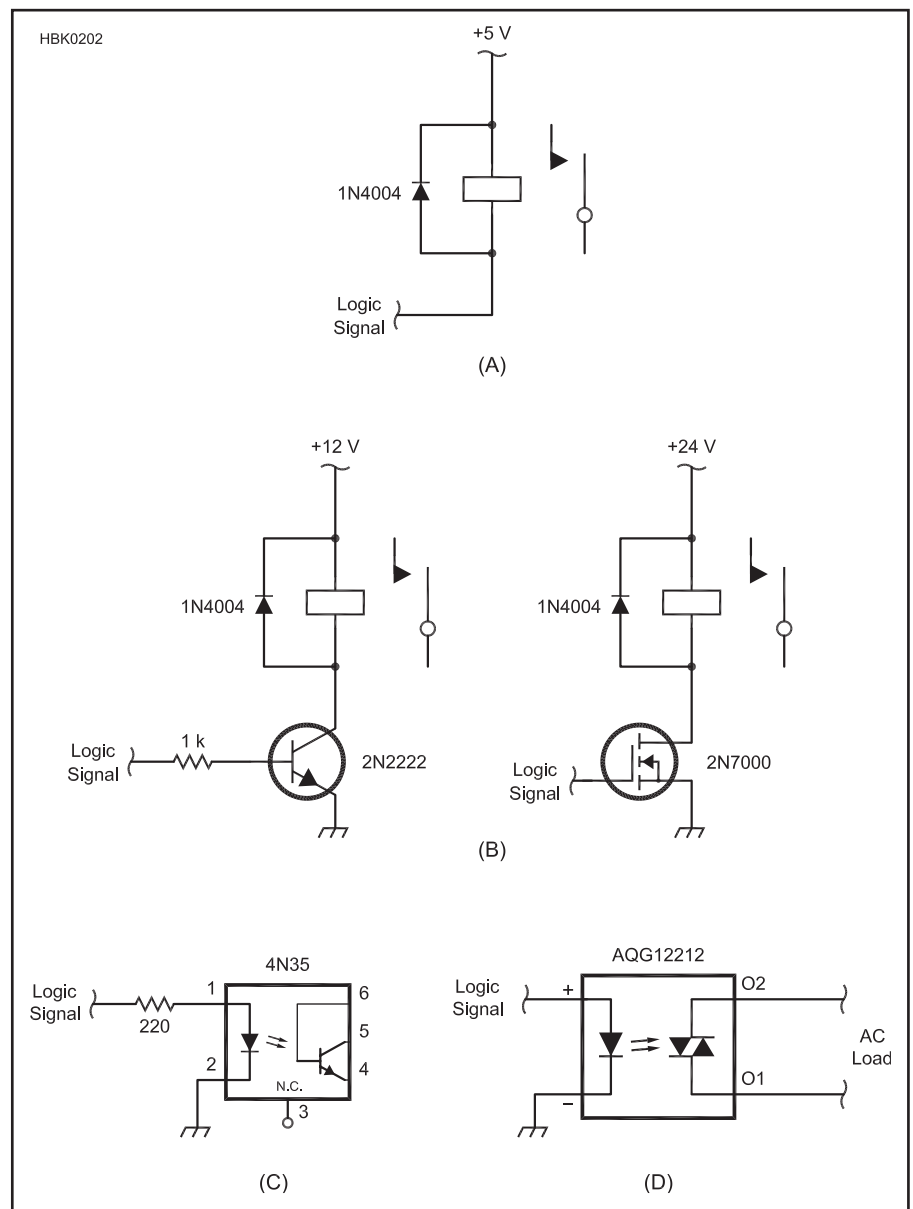

Figure 32— Interface circuits for logic driving real-world loads. (A) driving a relay from a logic output; (B) using a bipolar transistor or MOSFET to boost current capacity; (C) using an optoisolator for electrical isolation; (D) using a solid-state relay for switching ac loads.

shown in Figure 31C can be used to clamp the input voltage to an acceptable level. Care must be used to choose a resistor value that will not load the input signal unacceptably.

7.1 Analog-Digital Converters

Of course not all signals in the real world are digital. It is often desirable to know the exact voltage of an analog sensor, for example, or to measure the level of something — light, water, current or some other input. Similarly, an analog output can be quite useful for controlling or indicating things that are better left in the analog domain. For this reason we have two types of converters. The analog-to-digital converter (ADC) and digital-to-analog converter (DAC) handle these tasks for us, and they are often fairly easy to use. Additional information on ADCs and DACs may be found in the **Circuits and Components** chapter.

ADC AND DAC DIGITAL INTERFACES

Interfaces for ADC and DAC chips are generally classed as serial or parallel. Serial interfaces can vary in speed, complexity and the number of wires required for operation. A serial interface has the advantage of requiring a small number of processor I/O pins to accommodate data of any length. Whether your ADC is providing 8, 10 or 12 bits, the same small number of I/O signals are used. Some of the most common serial interfaces used for ADC and DAC chips are as follows:

Serial Peripheral Interconnect (SPI). This four-wire, synchronous bus and protocol use a common CLK signal, plus data lines for master-to-slave (Master Out-Slave In, or MOSI) and slave-to-master (Master In-Slave Out, or MISO). Multiple devices can be used by providing each with its own Slave Select (SS) signal. Speeds can range up to 70 Mbit/s, depending on the capabilities of the master and slave devices.

Microwire. This earlier predecessor to SPI implements a half-duplex subset of SPI using the same signals.

Inter-IC Communication (I2C) bus. Originally developed by Philips, this synchronous two-wire bus and protocol use a pair of open-drain lines, Serial Clock (SCL) and Serial Data (SDA). Communication is controlled by a master node, though there may be more than one master attached to the bus. Many peripheral devices can be attached to an I2C bus; a device address is sent by the master to initiate communication with a slave. Speeds range from 10 kbit/s (low speed) to 100 kbit/s (standard) to 400 kbit/s (high speed) and higher.

Parallel interfaces generally have eight or more data lines, plus a chip select, read/write and interrupt controls. The read and write signals may be separate, or may be a single read/write signal — low for WR, and high for RD, for example, or vice versa. The interrupt signal can be used to indicate the end of a conversion cycle to the CPU. In this way the processor can tell the converter to start a conversion, then continue processing until the conversion is complete. This can allow more processing to be done without waiting idly for the ADC or DAC to complete its conversion.

PROGRAMMING AND COMMUNICATION

Many ADC chips have multiplexed inputs that allow you to use one chip to sample and digitize more than one analog input. In these devices, there is a single converter but several inputs that can be internally switched. Additionally, it is common to be able to program the inputs as single-ended or differential. In the case of the popular ADC0832, for example, you can use its two analog input pins as two separate inputs, or as a single differential input. The ADC0838 expands this to eight single-ended or four differential inputs. In the case of the ADC0832, modes can be mixed. This allows the use of various combinations of single-ended and differential inputs, as needed.

Programming and selecting the inputs is done by sending a series of bits from the processor to the ADC at the beginning of the conversion cycle. **Figure 33** shows a diagram of the signal timing used with the ADC0834 ADC. In the case of the ADC0834, a series of four bits are sent by the CPU to the ADC to select single ended or differential mode, the polarity of the input, and which of the four inputs (or two input pairs) is to be selected. Immediately following the fourth bit, the ADC starts its conversion and starts sending the resulting data to the CPU. Other chips in this family and many from different manufacturers use a similar scheme; the number of program bits sent by the CPU depends on the number of inputs present.

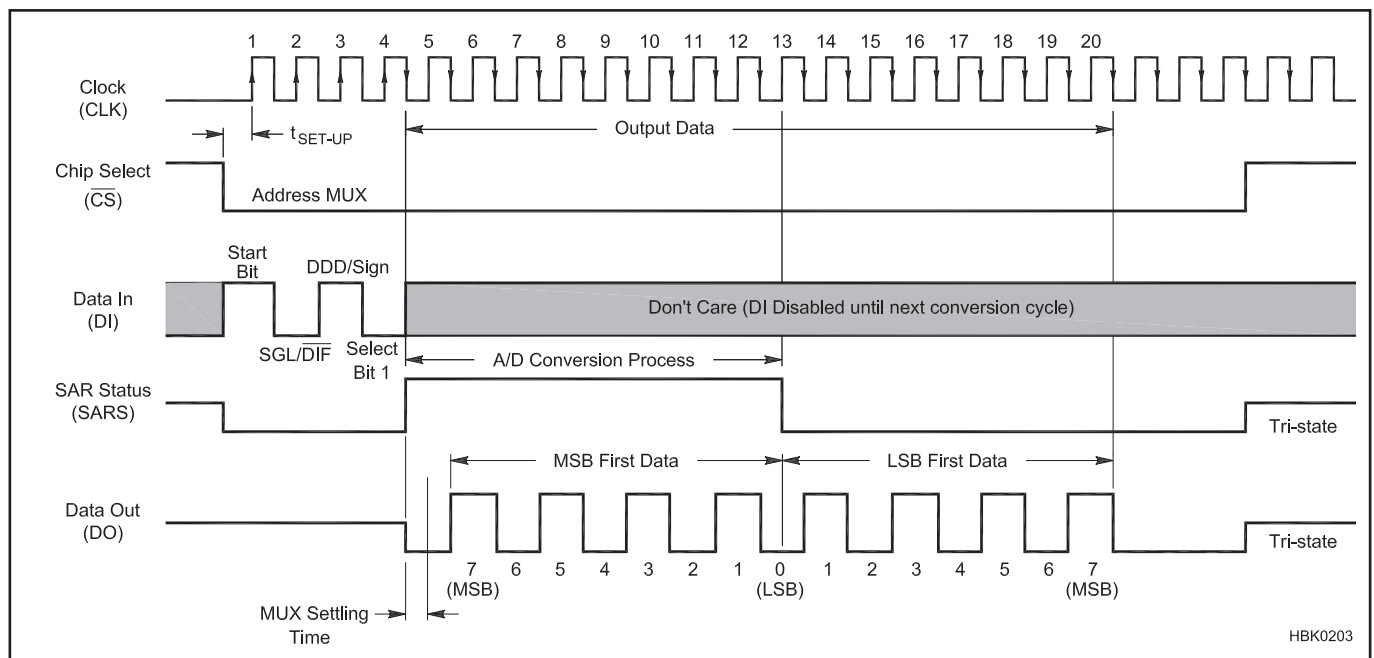Some chips are also configurable for data format as well. For example, a 12 bit ADC

Figure 33 — ADC0834 timing diagram. Much more information about this device is available from the National Semiconductor datasheet.
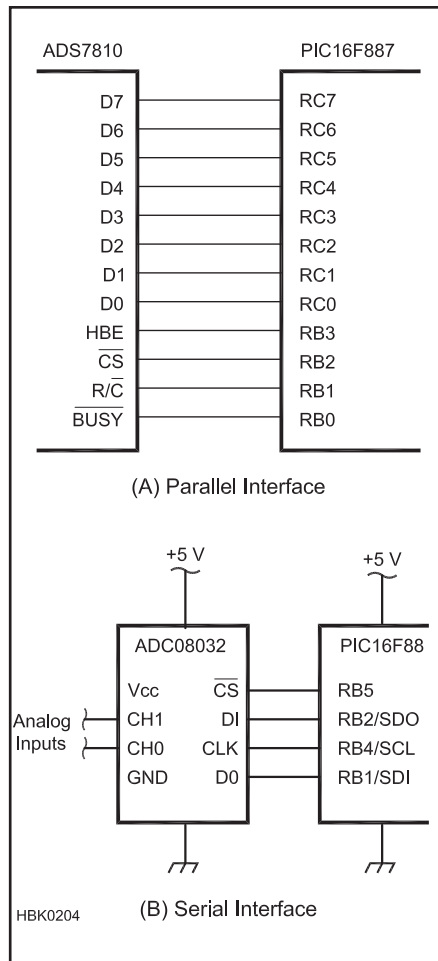
ADS7810 — PIC16F887 (A) Parallel Interface

ADS7810	PIC16F887
D7	RC7
D6	RC6
D5	RC5
D4	RC4
D3	RC3
D2	RC2
D1	RC1
D0	RC0
HBE	RB3
$\overline{CS}$	RB2
R/$\overline{C}$	RB1
$\overline{BUSY}$	RB0

(A) Parallel Interface

+5 V   +5 V

ADC08032	PIC16F88
Vcc $\overline{CS}$	RB5
CH1 DI	RB2/SDO
CH0 CLK	RB4/SCL
GND D0	RB1/SDI

Analog Inputs

HBK0204

(B) Serial Interface

Figure 34 — A/D converter to microcontroller interfacing: (A) parallel; (B) serial.

may be programmable to send a sign bit for a total of 13 data bits rather than 12. Data may also be sent LSB first or MSB first. Some chips are configurable for either parallel or serial data transfer; this selection is generally made by pulling a pin or pins high or low to select the mode. In all cases, a careful reading of the device data sheet is your best bet for successful integration of the converter into your project.

INTERFACING TO YOUR MICROCONTROLLER

Probably the simplest way to interface an external ADC to your microcontroller is by using a parallel interface. In this case, one simply manipulates the I/O pins from the CPU to start the conversion, then reads the resulting data from the data lines. This method is often less desirable than a serial connection, though, due to the limited number of I/O pins usually available. **Figure 34A** shows an example of a parallel interface between a 12 bit ADC (the ADS7810) and a PIC microcontroller (the PIC16F887). In this example, only eight data lines are used for twelve data bits. The HBE signal is used to gate the four high order bits onto the data lines after the low order bits have been read. This saves four I/O pins on the microcontroller.

Serial connections save I/O pins, allow the use of physically smaller IC packages, and can be quite easy to implement. Figure 34B shows an example of a serial interface. If your CPU has a hardware SPI/Microwire interface built in, communication with the ADC can be quite simple from a program standpoint. Simply send the program word, if required, and read the resulting data — either one or

two bytes, depending on the ADC resolution. If you do not have built-in hardware SPI, the program code to "bit-bang" the interface is fairly simple to implement. (See **www.dlpdesign.com/images/bit-bang-usb.pdf** for more information on bit-bang.)

Note that Figure 34B shows separate grounds for the processor and ADC. This is required to keep switching noise and transients from affecting the accuracy of readings. A separate analog supply is also recommended. The analog supply should be well filtered using a combination of smaller ceramic or monolithic, and bulk tantalum or other low-ESR capacitors. This should be done for the analog supply and reference voltage supplies. The more noise you can eliminate from the analog supply, the more reliable your readings will be.

MICROCONTROLLERS WITH BUILT-IN CONVERTERS

It is not at all unusual to find microcontrollers with built-in analog-to-digital converters, complete with multiplexers, analog supply pins and V_{REF} inputs. For example, many Microchip PIC processors are equipped with 10 bit ADCs, and have 5, 8 or more pins that can be configured as multiplexed analog inputs. A few also have digital-to-analog, though this is not nearly as common. While these devices can be extremely useful, there are some limitations. Reference voltages are generally limited to the device's supply voltage. Internal CPU-generated noise may also somewhat limit the usefulness of the converter. With careful attention to device specifications and limitations, though, a built-in ADC may meet the needs of a wide range of uses.

8 Microcontroller Overview

This section surveys microcontroller characteristics and applications as they pertain to typical Amateur Radio projects and equipment. The reader interested in using microcontrollers is directed to the many books and Internet resources that exist for learning how to apply microcontrollers. Manufacturer websites such as Microchip (**www.microchip.com**), Freescale (**www.freescale.com**) and Atmel (**www.atmel.com**) are a good place to start. There are also a number of message boards, forums and email lists devoted to specific processor families, such as the PICList (**www.piclist.com**) or the Arduino User Forum (**http://arduino.cc/forum**). Local users groups are often active in large cities. Online communities of microprocessor users are very fluid — use Internet search engines to find current resources.

8.1 Selecting a Microcontroller

There are a number of microcontroller families available from several manufactures, each with its own advantages and disadvantages. At the time of this writing, the more popular options for amateur use come from Microchip (PIC product line), Atmel (AVR, ATMega, 8051 and other product lines), Freescale (68HC11, HC08, HC16 and other product lines), and many manufacturers who are building chips based on the venerable Intel 8051 architecture. In addition to these you will see and hear of the Texas Instruments MSP430, ARM, and many others.

Designing projects using microcontrollers can range from relatively simple to quite complex. On the simpler end of the scale are several products intended to bring microcontroller based design ability to people who would otherwise not be able to use these devices. Various breadboard kits and development systems are available to simplify hardware development and program downloading. The following three systems are particular popular:

• Arduino (**www.arduino.cc**) — based on an AVR microcontroller, Arduino boards are available with a variety of capabilities and interfaces. Add-on modules called "shields" are added to interfaces provided on the Arduino board. Software is written in C or C++ and downloaded to the microcontroller. The manufacturer provides free development software.

• Parallax BASIC Stamp (**www.parallax.com**) — a single-board system containing a PIC microcontroller preloaded with PBASIC. Software is written in an editor program on the PC and then downloaded.

• PICAXE — a variety of PIC microcontroller with a special form of BASIC. Various sizes of microcontrollers are available with different numbers of I/O pins and clock speeds. The user is responsible for building the circuit board that supports the microcontroller.

In each of these cases, the user does not necessarily need to have a deep understanding of the details of the microcontroller or its operation. The circuit can be designed with attention paid to keeping signal levels within the limits of the device. Then, one simply writes a program to perform the functions required. To be sure, there are trade-offs. The built-in firmware required to allow the user to write a program may be in a simple language such as BASIC that takes up space and processor cycles, so execution speed is not as fast and programs are somewhat limited.

The BASIC Stamp and PICAXE use specific variants of the PIC microcontroller, so if you need features that are not present in those chips you will need to take a different approach. Still, these can be extremely valuable tools that the average amateur can use with very little time and money invested.

There is a very broad range of microcontrollers available to meet nearly any need. Available devices include 8 bit, 16 bit and 32 bit processors with internal program memory ranging from 512 words up to 256 kbytes and more. At the low end are very low cost, physically small devices (down to 8 pin ICs) that can perform relatively simple tasks with ease. Larger, faster processors may include more memory, more I/O pins, and specialized peripherals. Some of the more common features are ADCs, DACs, pulse width modulation (PWM) outputs, USB interface hardware or digital signal processing.

I/O REQUIREMENTS

Selecting a microcontroller for a new project involves evaluating your requirements and prior experience. First, evaluate the number of inputs and outputs you are likely to need. If, for example, you need to sample an analog voltage and two switch closures and generate a serial data stream, you will need one analog input and three or more digital I/O pins. Depending on memory requirements, even some of the smallest devices will fit the bill. For a more elaborate design, you may need to scan a 4×4 key switch matrix, drive an LCD module with a parallel interface, control a number of LEDs and interface to a PC through a USB port. In this case, the number of I/O pins needed will eliminate a large number of devices from consideration.

CPU SIZE, PERFORMANCE AND MEMORY

If you are a computer user, you are used to seeing 32 bit and 64 bit microprocessors, many with more than one CPU core on the same chip. In the world of embedded microcontrollers, things are a little different. Instead of handling an operating system and a large number of different programs, the microcontroller only has to execute one program and is completely dedicated to that task. Unless you require very high speed I/O, need very intensive processing or will be using DSP, a simple 8 bit processor may be more than sufficient. There are dozens of ham related products that use embedded controllers — iambic keyers, APRS transponders, repeater controllers, antenna rotator controls and many other accessories that all use simple 8 bit processors with surprisingly small amounts of program memory. On the other hand, if you are designing a new rig or antenna tuner that will require DSP, chances are you may need a more robust processor.

If you are unsure of what your project will require, you may want to pick a processor family that has scalable, pin-compatible members. For example, Microchip offers processors ranging from 8 bit and 5 million instructions per second (MIPS) with 7 kbytes of program memory, to 16 bit with 48 kbytes of program memory at 30 MIPS. All are pin-compatible, so if you find you need more speed, more memory or more built-in peripherals, you can drop in a different processor without changing the hardware.

Also take into account the physical packaging. Some products are available in through-hole, dual inline (DIP) packages; others are available only in surface mount versions. This can make a difference if you lack the equipment, expertise or desire to work with surface mount components.

SPECIALIZED PERIPHERALS

Will you need a USB interface? Analog inputs? DSP? Precision timekeeping? It may be easier to select a processor that has the features you need built in. On the other hand, it is relatively easy to add an external real-time clock, ADC or DAC, and it takes only a few I/O pins. There are several USB interface chips available that will handle communication with your PC, and present an asynchronous serial data interface to the microcontroller. This can save a lot of firmware development time and frustration.

HARDWARE COST AND DEVELOPMENT TOOLS

The cost of the actual microcontroller may be a factor, or it may not. If you are building a

one-off project for your own use, there is not much difference between using a $3 chip and a $30 chip with a built-in BASIC interpreter. The difference in time and effort to develop the firmware may make it well worth the extra money for the easier-to-use solution. If you plan to manufacture your device for a larger audience, the cost difference may lead you to make a different choice.

Take into account the cost of the software and hardware tools needed for development. Does the manufacturer offer a free set of firmware development tools, or will you have to spend extra money for an assembler or compiler? Is there a low cost development kit available with the PC interface you need? Does the device require a simple PC interface to program, or a more expensive dedicated programmer? Do you have any previous experience with the processor family, or will it be a completely new effort for you?

9 Personal Computer Interfaces

This section provides an overview of the personal computer (PC) digital interfaces used in the amateur station. Detailed information on using these interfaces is available online, especially in users groups and specialized forums for specific software and hardware applications. This book's chapters on **DSP and SDR Fundamentals**, **Digital Modes and Protocols**, and the online supplement on Digital Communications provide more information on specific applications and techniques. The **Station Accessories** chapter includes PC interface construction projects, as well.

9.1 Parallel vs Serial Signaling

To communicate a word to someone across the room, you could hold up flash cards displaying the letters of the word. If you hold up four flash cards, each with a letter on it, all at once, then you are transmitting in parallel. If, instead, you hold up each of the flashcards one at a time, then you are transmitting in serial. *Parallel* means all the bits in a group are handled exactly at the same time. *Serial* means each of the bits is sent in turn over a single channel or wire, according to an agreed sequence. **Figure 35** illustrates parallel and serial signaling.

Both parallel and serial signaling are appropriate for certain circumstances. Parallel signaling is faster, since all bits are transmitted simultaneously, but each bit needs its own conductor, which can be expensive. Parallel signaling is more likely to be used for internal communications. For spanning longer distances, such as to an external device, serial signaling is more appropriate. Each bit is sent in turn, so communication is slower, but it also is less expensive, since fewer channels are needed between the devices.

Most amateur digital communications use serial transmission to minimize cost and complexity. The number of channels needed for signaling also depends on the operational mode. One channel is required per bit for simplex (one-way, from sender to receiver only) and for half-duplex (two-way communication, but only one person can talk at a time), but two channels per bit are needed for full-duplex (simultaneous communications in both directions).

PARALLEL I/O INTERFACING

Figure 36 shows an example of a parallel input/output interface, similar to the parallel printer ports once common in PCs. Typically, they have eight data lines and one or more handshaking lines. *Handshaking* involves a number of functions to coordinate the data transfer. For example, the READY line indicates that data are available on all eight data lines. If only the READY line is used, however, the receiver may not be able to keep up with the data. Thus, the STROBE line is added so the receiver can determine when the transmitter is ready for the next character.

Interfacing to the parallel port is very simple and can be done in many languages and under many operating systems. It is an easy way to get a PC to act as a controller in the shack and around the home, with from 8 to 12 independent input or output wires. You can get an old computer either virtually free or for just a few dollars and use it exclusively as a controller. (New computers usually do not have a parallel port.)

By searching the Internet for a combination of "parallel port interface" and the language of your choice (for example, Visual BASIC or C), you can find detailed interfacing and programming instructions. It is a good and relatively simple place to "roll your own," compared to the protocol requirements of interfacing with the serial port or USB.

SERIAL I/O INTERFACING

Serial input/output interfacing is more complex than parallel, since the data must be transmitted based on an agreed sequence. For example, transmitting the 8 bits (b7, b6, . . . b0) of a word includes specifying whether the least significant bit, b0, or the most significant
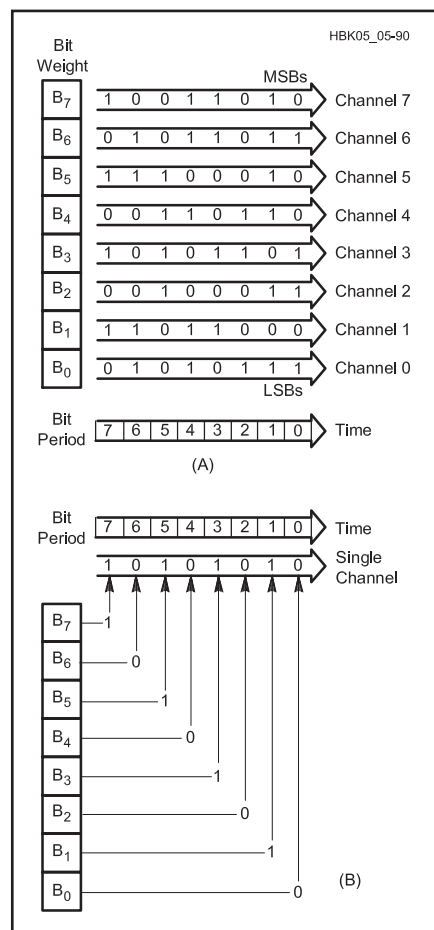
Figure 35 — Parallel (A) and serial (B) signaling. Parallel signaling in this example uses 8 channels and is capable of transferring 8 bits per bit period. Serial transfer only uses 1 channel and can send only 1 bit per bit period.
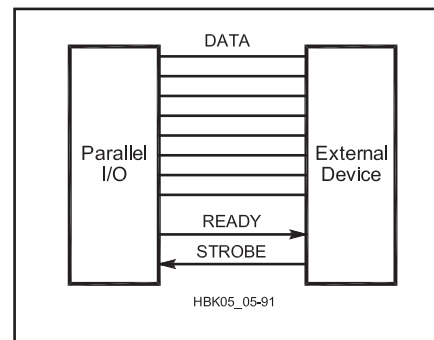
Figure 36 — Parallel interface with READY and STROBE handshaking lines.

bit, b7, is sent first. Fortunately, a number of standards have been developed to define the agreed sequence, or encoding scheme.

In order to use a serial interface between a computer and a serial device such as a modem, printer or a new transceiver, several things must be set the same on both ends of the connection. The first is the data rate, the choices for which are usually limited to specific values between 300 and 115200 bits per second for historical reasons. The next is the data format — 7 or 8 bits, and with even, odd or no parity. Finally, both devices must be using the same handshaking method. The usual choices are RTS/CTS, DSR/DTR, or XON/XOFF. The first two use extra signals (and therefore extra wires) to indicate when the transmitting device has data to send, and when the receiving device is able to accept data. In the third method, XON/XOFF, the device receiving data will send an XOFF control character to the transmitting device to indicate that it can no longer accept data. When the transmitting device sees the XOFF it stops sending until it receives an XON from the other end. Each method has its advantages and disadvantages. For example, XON/XOFF needs no extra wires, but response is not instantaneous. Both ends must have some data buffer space and logic to support the handshake.

9.2 Standard Data Interfaces

This section covers a set of common data interfaces used in the amateur station. See the **Component Data and References** chapter for details on computer connector pinouts.

SIGNALING LEVELS

Inside equipment and for short runs of wire between equipment, the normal practice is to use neutral keying; that is, simply to key a voltage such as +5 V on and off. In neutral keying, the off condition is considered to be 0 V. Over longer runs of wire, the line is viewed as a transmission line, with distributed inductance and capacitance. It takes longer to make the transition from 0 to 1 or vice versa because of the additional inductance and capacitance. This decreases the maximum speed at which data can be transferred on the wire and also may cause the 1s and 0s to be different lengths, called bias distortion. Also, longer lines are more likely to pick up noise, which can make it difficult for the receiver to decide exactly when the transition takes place. Because of these problems, bipolar keying is used on longer lines. Bipolar keying uses one polarity (for example +) for a logical 1 and the other (– in this example) for a 0. This means that the decision threshold at the receiver is 0 V. Any positive voltage is taken as a 1 and any negative voltage as a 0.

EIA-RS-232

The serial bus protocol EIA-RS-232 addresses this issue (however, a Mark "1" is a negative voltage and a Space "0" is positive). Generally called RS-232, this protocol defines connectors and voltages between data terminal equipment (DTE) such as a PC, and data communications equipment (DCE), such as a modem or TNC.

The connector is the DB-25 (25 pin), or DB-9 (9 pin) version — though RS-232 interfaces have also been implemented using nonstandard connectors such as RJ45, 3.5 mm audio plugs and many others. Signaling voltages are defined between +3 V and +25 V for logic "0" and between –3 V and –25 V for logic "1." Although the top data rate addressed in the specification is only 20 kbit/s, speeds of up to 115 kbit/s are commonly used. Communications distances of hundreds of meters are possible at reasonable data rates.

Since neutral keying is usually used inside equipment and bipolar keying for lines leaving equipment, signals must be converted between bipolar and neutral. Discrete level shifters or op amp circuits may perform this task, or low cost specialized IC line drivers and receivers are available.

UNIVERSAL SERIAL BUS (USB)

USB is a computer standard for an intelligent serial data transfer protocol, and it has become the standard for nearly all connections between a PC and its external peripherals. It has largely replaced serial, parallel, keyboard and mouse ports as well as SCSI and numerous other buses in consumer PCs.

In addition to higher speed than RS-232 and parallel ports, USB offers reasonable power availability to its loads, or functions. Under certain circumstances, up to 127 hubs and functions may connect to a single computer. USB requires that each function have on-board intelligence and that it negotiate with the host for power and bandwidth allocation. USB also has the major advantage of hot-pluggablity — the PC need not reboot when new functions are added.

The USB connectors use four-conductor cable, with two bidirectional, differential data lines, power, and ground. Approximately 5 V at 100 mA is allowed per function, with up to 500 mA available if the host system has the capability. This means that relatively sophisticated devices, such as modems, small video cameras, or hand-held scanners may operate from the bus without additional power supplies. It has also become common to use USB for connecting and charging a wide range of devices such as cellular phones, cameras and GPS units. When a USB connecting cable is used, proper connections and proper flow are ensured by using a rectangular (USB

"A") connector on the host and a different connector (USB "B", Mini-B or Micro-B) on the attached function.

There are currently two USB standards in general use. USB 1.1, somewhat obsolete but common in PCs just a few years old, is capable of 12 megabits per second (12 Mbit/s). USB 2.0 is the later standard and is rated up to 480 Mbit/s. Most USB 2.0 ports will allow the use of older USB 1.1 devices — that is, they are backward compatible. However maximum cable lengths and available power to devices may be affected. USB 3.0 uses additional wires and some new connectors to enable communication at up to 5 Gbit/s in its SuperSpeed configuration.

Older PCs that have just one or two USB ports can have additional ports by adding an inexpensive USB port expander, commonly called a USB hub. Some units obtain their power from the host computer's USB port, and distribute only the USB signals along with the remaining power available from the host. Others come with a small power supply that provides the normal power to each new USB port.

If the PC does not have any USB ports, an expansion card can be used to add USB ports. It is quite common to see new personal computers without any serial or parallel ports, with USB replacing these functions. If you need to use your older serial or parallel peripheral devices with these "legacy-free" PC systems, adapter cables are available to connect them to USB ports.

ETHERNET

For wired networks, 100BaseT or "fast Ethernet" has been the standard for a number of years, with low cost hubs and switches commonly available. The 100BaseT systems are rated to 100 Mbit/s and use unshielded, twisted-pair Category 5 network cable. Each computer on the network connects to a central hub or switch.

More recently, 1000BaseT (gigabit Ethernet, or GigE) and is quickly replacing 100BaseT even at the consumer level. Gigabit Ethernet also uses Cat 5 cable, though Cat 5e or Cat 6 is often recommended. Explanation of the MBaseN and Category N terminology can be found in many available networking books.

Mutual interference with ham station operation is not uncommon. A 10 Mbit/s signal, if all bit positions are filled, means that unshielded wire is carrying 10 MHz and various harmonics of 10 MHz around the shack. When digital words are going through the network, any number of frequencies may be present on the wires and the wires may be very susceptible to pick-up from HF, VHF and UHF signals. In addition, cable and DSL modems may not only transmit various

frequencies (especially on VHF) but lose data when a few hundred watts on a ham band is present. Some RF in the shack that can normally be ignored can easily bring down a network. Unfortunately, it is also common for this equipment to have "noisy" switching power supplies that can wreak havoc on a ham's sensitive receiver. See the **RF Interference** chapter for more information on dealing with RFI from networking equipment.

WIRELESS NETWORKS — WIFI

The other technology commonly found in home and office networks is IEEE 802.11 wireless. These networks use low-power, spread spectrum transceivers operating in the 2.4 and 5.2-5.8 GHz range to transfer data at nominal rates up to 54 Mbit/s and higher. Older equipment used the 801.11b protocol at 11 Mbit/s; newer gear commonly found at low prices uses 802.11g at 54 Mbit/s maximum data rate. The 802.11n protocol supports data rates up to several hundred megabits per second in the 2.4 and 5 GHz ranges.

These data rates are the maximum under ideal conditions, and it is not uncommon to see links running at significantly lower speeds as the *wireless access point* (WAP) and the wireless adapter dynamically adapt to the conditions. Still, performance is adequate for most uses, and the lack of network cabling makes wireless networking attractive in many situations.

Wireless networks actually appear to be less susceptible to mutual interference, since they are not connected to long runs of unshielded wire that easily act as both transmitting and receiving antennas. The frequency bands used for wireless networks are reasonably far removed from normal ham operations at frequencies below 2.4 GHz, but a high power VHF or UHF station may interfere with such a network. Also, some of the channels used for 802.11b and 802.11g network equipment fall within the amateur spectrum allocation in the 2.4 GHz band.

CABLE REPLACEMENT — BLUETOOTH AND ZIGBEE

A number of low power, short range data links have been created to replace cable based interfaces such as USB and RS-232. These are digital protocols that use unlicensed spectrum in the ISM bands. Two of the most popular as of early 2013 are Bluetooth (**www.bluetooth.com**) and Zigbee (**www.zigbee.org**). Most technologies of this type are either initially or permanently developed as implementations of an IEEE 802.x standard.

Bluetooth data links operate in the 2.4 GHz band and are primarily used for streaming audio and high speed digital links up to several megabits per second. Bluetooth links use frequency-hopping spread-spectrum technology. It can also support personal area networks (piconets or PANs) at various levels of security. Up to seven devices can share a Bluetooth interface with a host. A process called synchronization is used to associate the various devices.

Zigbee is designed for ad-hoc mesh networking applications and operates in the 902 MHz ISM band in North America. Zigbee's maximum data rate is 250 kbit/sec. Because of the lower data rate, most Zigbee applications are designed for control and monitoring applications.

REPLACING COM AND LPT PC INTERFACES

USB-to-serial and USB-to-parallel adapters are relatively inexpensive and easy to find. Note, however, that not all of them support all of the handshaking signals we may require, and some may have delays or inadequate drivers. Be prepared to try several different brands to find one that works with your equipment. Users groups for specific equipment or software are good resources to find out what brands and models of port replacement accessories are compatible and operate properly. While they are becoming less common, there is also still a market for add-in PCI bus cards to add serial, parallel and other ports to desktop computers.

10 Glossary of Digital Electronics Terms

AND gate — A logic circuit whose output is 1 only when all of its inputs are 1.

Astable (free-running) multivibrator — A circuit that alternates between two unstable states. This circuit could be considered as an oscillator that produces square waves.

Asynchronous flip-flop — A circuit, also called a *latch*, that changes output state depending on the data inputs, without requiring a clock signal.

Binary — A base-2 number system used in digital electronics that uses the symbols 0 and 1.

Binary coded decimal (BCD) — A simple method for converting binary values to and from decimal for inputs and outputs for user-oriented digital systems. BCD was widely used in the days of 7-segment LED displays but is not common today.

Bistable multivibrator — Another name for a flip-flop circuit that has two stable output states.

Boolean algebra — The mathematical system used to describe and design binary used digital circuits, named after George Boole.

Bus — A set of wires through which data is routed internally within computers and other digital devices.

Clock — A signal that toggles at a regular rate. Clock control is the most common method of synchronizing logic circuits.

Combinational logic — A type of circuit element in which the output depends on the present inputs. (Also see **Sequential logic**.)

Complementary metal-oxide semiconductor (CMOS) — A type of construction used to make digital integrated circuits. CMOS is composed of both N-channel and P-channel MOS devices on the same chip.

Counter (divider, divide-by-n counter) — A circuit that is able to change from one state to the next each time it receives an input signal. A counter produces an output signal every time a predetermined number of input signals have been received.

Decimal — The base-10 number system we use every day that uses the symbols 0 through 9.

DeMorgan's Theorem — In Boolean algebra, a way to simplify the complement of a large expression or to enable a designer to interchange a number of equivalent gates.

Digital IC — An integrated circuit whose output is either on (1) or off (0).

Dynamic (edge-triggered) input — A control signal that allows a circuit to change state only when the control signal changes from unasserted to asserted.

Exclusive OR gate — A logic circuit whose output is 1 when either of two inputs is 1 and whose output is 0 when neither input is 1 or when both inputs are 1.

Fan-out — The ability of a logic element

to drive or feed several other logic elements.

Field Programmable Gate Array (FPGA) — A type of programmable logic array that can contain several hundred to millions of logic gates and up to 1000 or more I/O pins.

FireWire (IEEE-1394) — A very high speed serial protocol capable of up to 400 Mbit/s of sustained transfer.

Flip-flop (bistable multivibrator) — A circuit that has two stable output states, and which can change from one state to the other when the proper input signals are detected.

Gate — A combinational logic element with two or more inputs and one output. The output state depends upon the state of the inputs.

Handshaking — Functions to coordinate data transfer.

Hexadecimal — A base-16 number system widely used in computer systems that uses the 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Integrated circuit — A device composed of many bipolar or field-effect transistors manufactured on the same chip, or wafer, of silicon.

Inverter — A logic circuit with one input and one output. The output is 1 when the input is 0, and the output is 0 when the input is 1.

Latch — Another name for a bistable multivibrator (flip-flop) circuit. The term latch reminds us that this circuit serves as a memory unit, storing a bit of information.

Linear IC — An integrated circuit whose output voltage is a linear (straight line) representation of its input voltage.

Logic probe — A simple piece of test equipment used to indicate high or low logic states (voltage levels) in digital-electronic circuits.

Microcontroller — A "computer on a chip" that usually consists of a relatively small microprocessor along with some amount of program memory, data memory, input/output ports and often some specialized peripheral devices.

Monostable multivibrator (one shot) — A circuit that has one stable state. It can be forced into an unstable state for a time determined by external components, but it will revert to the stable state after that time.

NAND (NOT AND) gate — A logic circuit whose output is 0 only when both inputs are 1.

Noninverter — A logic circuit with one input and one output, and whose output state is the same as the input state (0 or 1). Sometimes called a *noninverting buffer*.

NOR (NOT OR) gate — A logic circuit whose output is 0 if either input is 1.

OR gate — A logic circuit whose output is 1 when either input is 1.

Parallel — A digital signaling method in which all the bits in a group are handled exactly at the same time.

Programmable logic device (PLD) — A device that includes a generic array of gates that can be controlled by program code. PLDs can be made to replace large numbers of individual ICs.

Propagation delay — The time delay between providing an input to a digital circuit and seeing a response at the output.

Register — A set of latches or flip-flops storing an n-bit number.

RS-232 — The most common serial bus protocol.

RS-422 — A serial protocol similar to RS-232, but employing fully differential data lines.

Serial — A digital signaling method in which each bit is sent in turn over a single channel or wire, according to an agreed sequence.

Sequential logic — A type of circuit element in which the output depends on the present inputs, the previous sequence of inputs and often a clock signal. (Also see **Combinational logic**.)

Square wave — A periodic waveform that alternates between two values, and spends an equal time at each level. It is made up of sine waves at a fundamental frequency and all odd harmonics.

Static (level-triggered, or gated) input — A control signal that allows the circuit to change state whenever the control signal is at its active or asserted level.

Synchronous flip-flop — A circuit whose output state depends on the data inputs, but that will change output state only when it detects the proper clock signal.

Transition region — The undefined region between the two binary states. Also known as the *noise margin*.

Transition time — The time it takes a digital circuit to change state. The transition from a 0 to a 1 state is called the *rise time*, and the transition from a 1 to a 0 state is called the *fall time*.

Tri-state gate — A gate with one additional control lead. When enabled, the gate operates normally; when not enabled, the output goes to a high impedance.

Truth table — A chart showing the outputs for all possible input combinations to a digital circuit.

Universal serial bus (USB) — A computer standard for an intelligent serial data transfer protocol.

11 References and Bibliography

DIGITAL ELECTRONICS

Bignell and Donovan, *Digital Electronics* (Delmar Learning, 2006)

Holdsworth, B., *Digital Logic Design* (Newnes, 2002)

Lancaster and Berlin, *CMDS Cookbook* (Newnes, 1997)

Tocci, Widmer and Moss, *Digital Systems: Principles and Applications* (Prentice Hall, 2006)

Tokheim, R., *Digital Electronics: Principles and Applications* (McGraw-Hill, 2008)

Logic simulation software: "Getting Started with Digital Works" **www.spu.edu/cs/faculty/bbrown/circuits/howto.html**

MICROPROCESSORS AND MICROCONTROLLERS

Dumas, J., *Computer Architecture: Fundamentals and Principles of Computer Design* (CRC Press, 2005)

Gilmore, C., *Microprocessors: Principles and Applications* (McGraw-Hill, 1995)

Johnson, C., AAØZZ, ""Learning to PIC with a PIC-EL" (Parts 1 and 2), *QST*, May 2007 (pp. 37-42) and June 2007 (pp. 33-36)

Klotz, L., WA5ZNU (Ed.), *Ham Radio for Arduino and PICAXE* (ARRL, 2013)

Korneev and Kiselev, *Modern Microprocessors* (Charles River Press, 2004)

Popiel, G., KW5GP, *Arduino for Ham Radio* (ARRL, 2014)

Popiel, G., KW5GP, *More Arduino Projects for Ham Radio* (ARRL, 2017)

Spencer, M., WA8SME, ARRL's PIC Programming for Beginners (ARRL, 2010)

Tocci and Ambrosio, Microprocessors and Microcomputers: Hardware and Software (Prentice-Hall, 2002)