Digital Signal Theory and Components

7

Digital Fundamentals

Digital signal theory is an important aspect of Amateur Radio. With a knowledge of digital theory, there are many new worlds for the radio amateur to explore. Applications of digital signal theory include digital communications, code conversion, signal processing, station control, frequency synthesis, amateur satellite telemetry, message handling, word processing and other information handling operations.

This chapter, written by Christine Montgomery, KG0GN, 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. Integrated circuits continue with memory chips and microprocessors, culminating in a synthesis of these components in the modern digital computer. Where possible, this chapter mentions Amateur Radio applications associated with the technologies being discussed, as well as pointers to other chapters that discuss such applications in greater depth.

DIGITAL VS ANALOG

An essential first step in understanding digital theory is to understand the difference between a *digital* and an *analog* signal. An analog value, a real number, has no end; for example, the number 1/3 is 0.333... where the 3 can be repeated forever, or 3/4 equals 0.7500... with infinite repeated 0s. A digital approximation of an analog number breaks the real number line into discrete steps, for example the integers. This process of approximating a value with discrete steps either truncates or rounds an analog value to some number of decimal places. For example, rounding 1/3 to an integer gives 0 and rounding 3/4 gives 1.

For a simple physical example, look at your wristwatch. A watch with a face — with the hands of the watch rotating in a continuous, smooth motion — is an analog display. Here, the displayed time has a *continuous* range of values, such as from 12:00 exactly to 12:00 and 1/3 second or any values in between. In contrast, a watch with a digital display is limited to *discrete* states. Here the displayed time jumps from 12:00 and 0 seconds to 12:00 and 1 second, without showing the time in between. (A watch with a second hand that jerks from one second to another could also fit the digital analogy.)

In the digital watch example, time is represented by ten distinct states (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9). Digital electronic signals, however, will usually be much more limited in the number of states allowed.

Binary, the most common system, has only two states: 0 and 1. Ternary (3 states), quaternary (4 states) and other digital systems also exist. **Fig 7.1** illustrates the contrast of an analog signal (in this case a sine wave) and its digital approximation.

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.

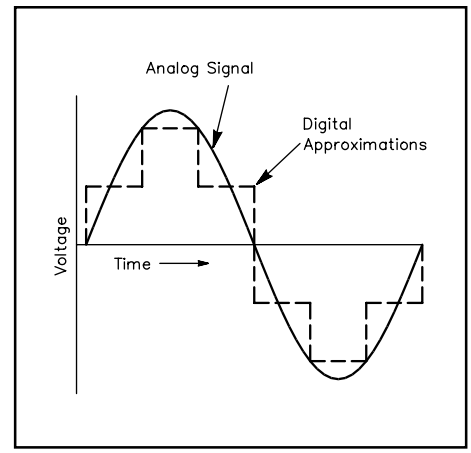


Fig 7.1 — An analog signal and its digital approximation. Note that the analog waveform has continuously varying voltage while the digital waveform is composed of discrete steps.

Number Systems

In order to understand digital electronics, you must first understand the digital 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 left-most 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. This is the "traditional" number system we are all familiar with.

DECIMAL

The decimal system is a base-10 system, with ten symbols: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. In the decimal number, 548.21, the digits are 5, 4, 8, 2 and 1, where 5 is the most significant digit since it is positioned to the far left and 1 is the least significant digit since it is positioned to the far right. The value of this number is a weighted sum of its digits, as shown in **Table 7.1**.

The weight of a position is the system's base raised to a power, 10^P with the power determined by the position relative to the decimal. For example, digit 8, immediately to the left of the decimal, is at position 0; therefore, its weight factor is $10^0 = 1$. Similarly, digit 5 is 2 positions to the left of the decimal and has a weight factor $10^2 = 100$. The value of the number is the sum of each digit times its weight.

BINARY

Binary is a *base-2* number system that is 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 7.2**.

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* and two nibbles, or eight bits, make a *byte*. A *word* can consist of two or four bytes, and two words (a most significant word, *MSW*, and a least significant word, *LSW*) is sometimes called a *longword*. These groupings are useful when converting to hexadecimal notation, which is explained later.

Table 7.1
Decimal Numbers

Example: $5(10^2)$

Digit = 5; Weight = 10; Position = 2

548.21	=	$5(10^2)$	+	$4(10^1)$	+	$8(10^0)$	+	$2(10^{-1})$	+	$1(10^{-2})$
	=	5(100)	+	4(10)	+	8(1)	+	2(0.1)	+	1(0.01)
	=	500	+	40	+	8	+	0.2	+	0.01
	=	5		4		8		. 2		1
		MSD						decimal		LSD

Table 7.2
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																		

OCTAL

Octal is a *base-8* number system, using the symbols {0,1,2,3,4,5,6,7}. The weight factors are now powers of 8, such as 8^0, 8^1 and 8^2. For example, the decimal number 163 is equivalent to octal 243.

Since $2^3 = 8$, it is easy to switch between binary and octal just by viewing the binary number in groups of 3. (Add a leading 0 on the left most group, if the number of digits doesn't divide evenly into groups of three.)

HEXADECIMAL

The hexadecimal, or hex, *base-16* number system 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 decimal number 163 is equivalent to hex A3.

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 7.3** for hexadecimal, octal 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.

Using the repeated division for binary is rather cumbersome, since the tower quickly grows large. Combining this technique with the grouping technique discussed earlier should make conversions fairly easy. Simply perform the tower division on one of the larger numbers, such as hexadecimal 16, then use grouping to put the result into binary form.

Another technique that should be briefly mentioned can be even easier: get a calculator with a binary and/or hex mode option. 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, always use a leading 0 when you enter a number in binary or hex into your calculator.

Table 7.3
Number System Conversions

Hex	*Remainder*	*Octal*	*Remainder*	*Binary*	*Remainder*	
16 |163		8 |163		2 |163		
|10	3	|20	3	|81	1	LSB
|0	A	|2	4	|40	1	
		|0	2	|20	0	
				|10	0	
				|5	0	
				|2	1	
				|1	0	
				|0	1	MSB
A3 hex		243 octal		1010 0011 binary		

BINARY CODED DECIMAL (BCD)

Scientists have experimented with many devices out of a desire for fast computations. The first generation computers were born when J. Vincent Atanasoff decided to use binary numbers instead of decimal to do his computa-

tions. A binary number system representation is the most appropriate form for internal computations since there is a direct mathematical relationship for every bit in the number. To interface with a user — who usually wants to see I/O in terms of decimal numbers — other codes are more useful. The *Binary Coded Decimal (BCD)* system is the simplest and most widely used form for inputs and outputs of user-oriented digital systems.

In the Binary Coded Decimal (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 7.4**.

A generic code could use any n-bit string to represent a piece of information. BCD uses 4 bits because that is the minimum needed to represent a 9. All four bits are always written; even a decimal 0 is written as 0000 in BCD.

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. Instead of using the 4-bit code strings 1010 to 1111 for decimal 10 to 15, BCD uses 0001 0000 to 0001 0101. There are other n-bit decimal codes in use and, even for specifically 4 bits, there are millions of combinations to represent the decimal digits 0-9. BCD is the simplest way to convert between decimal and a binary code; thus it is the ideal form for I/O interfacing. The binary number system, since it maintains the mathematical relationship between bits, is the ideal form for the computer's internal computations.

Table 7.4
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**

Physical Representation Of Binary States

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,” a punched hole or the absence of a hole in paper tape or a card, 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

Fig 7.2 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*.

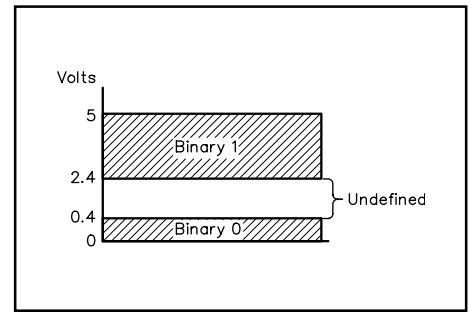


Fig 7.2 — Representation of binary states 1 and 0 by a selected range of voltage levels.

Transition Time

The gap in **Fig 7.2**, 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*. Similarly, the transition from a 1 to a 0 state is called the *fall time*. Note that these times need not be the same. **Fig 7.3A** shows an ideal signal, or *pulse*, with zero-time switching. Fig 7.3B 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 PC-board traces may cause rise and fall times to increase as the pulse moves away from the source.

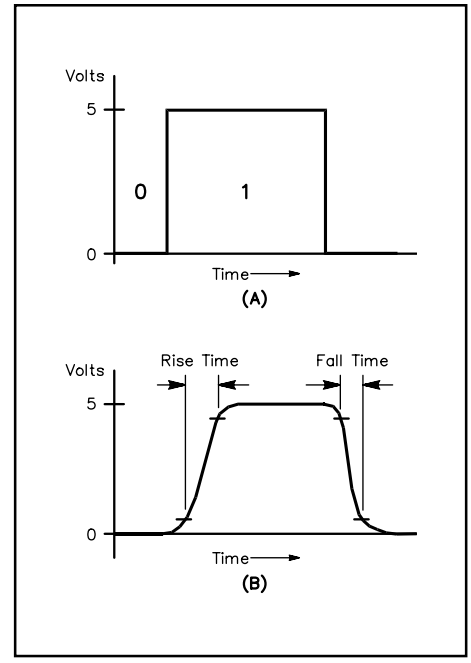


Fig 7.3 — (A) An ideal digital pulse and (B) a typical actual pulse, showing the gradual transition between states.

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 to seeing a

response at the output is the *propagation delay*, and is illustrated by **Fig 7.4**.

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. Solutions to this timing problem include adding a buffer amplifier to the circuit and synchronization of circuits. These are discussed in later sections.

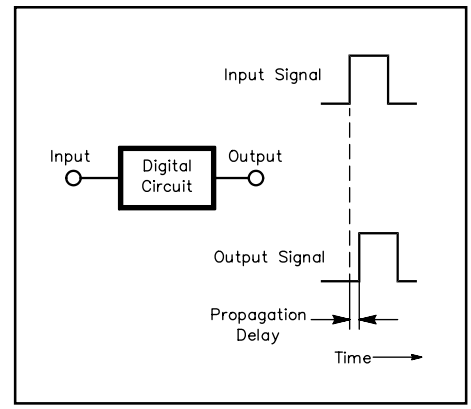


Fig 7.4 — Propagation delay in a digital circuit.

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. John F. Wakerly, on page 147 of *Digital Design Principles and Practices*, described this difference as follows: "The rotary channel selector knob on an inexpensive TV is like a combinational circuit — its 'output' selects a channel based only on the current position of the knob ('input'). In contrast, the channel selector controlled by the up [+] and down [-] pushbuttons on a fancy TV or VCR is a sequential circuit — the channel selection depends on the past sequence of up/down pushes... as far back as when you first powered-up the device."

The next section discusses combinational logic circuits. Later, we will build sequential logic circuits from the basics established here.

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 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) a Boolean equation or (B) 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 **Fig 7.5**. 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 flag represents "NOT." The new notation is an attempt to replace the detailed logic drawing of a complex function with a simpler block symbol.

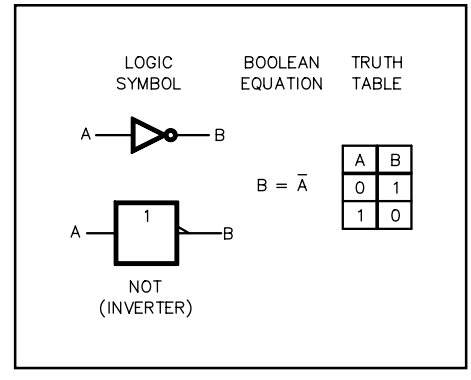


Fig 7.5 — Inverter.

Figs 7.5, 7.6 and 7.7 show the truth tables, Boolean algebra

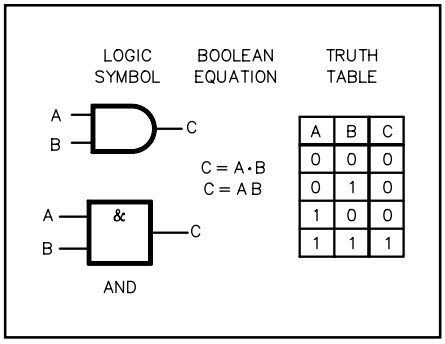


Fig 7.6 — Two-input AND gate.

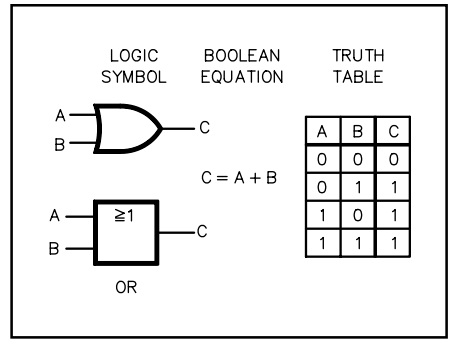


Fig 7.7 — Two-input OR gate.

equations and circuit symbols for the three basic Boolean operations: NOT, AND and OR. All combinational logic functions, no matter how complex, can be described in terms of these three operators.

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."

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 many 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.

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."

Other Common Gates

More complex logical functions are derived from combinations of the basic logical operators. These operations — NAND, NOR, XOR, XNOR and the noninverter — are illustrated in **Figs 7.8** through **7.12** 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.

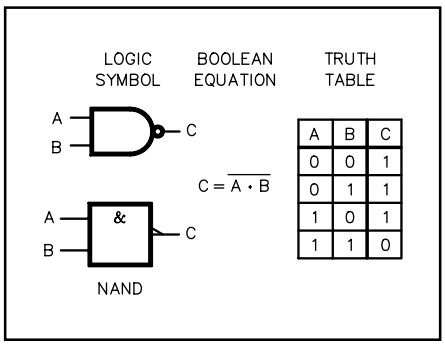


Fig 7.8 — Two-input NAND gate.

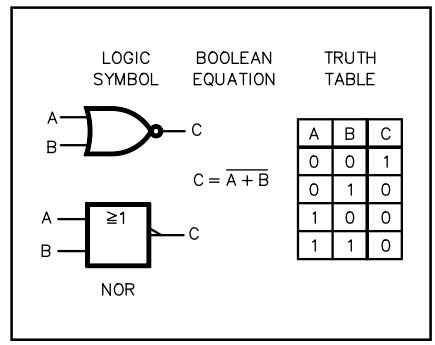


Fig 7.9 — Two-input NOR gate.

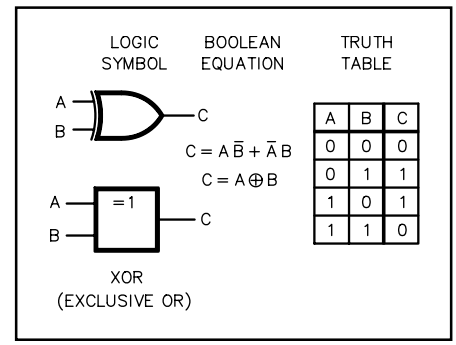


Fig 7.10 — Two-input XOR gate.

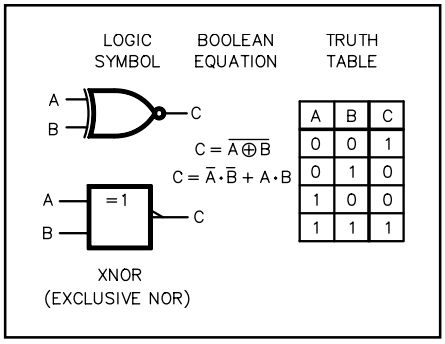


Fig 7.11 — Two-input XNOR gate.

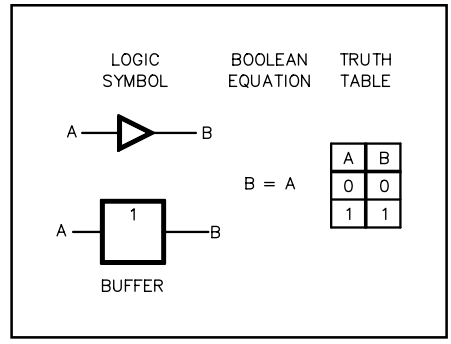


Fig 7.12 — Noninverting buffer.

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) 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 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.

0 0 1 1
<u>0</u> <u>1</u> <u>0</u> <u>1</u>
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.

Noninverter

A non*inverter*, 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. The practical uses of a noninverter include (A) providing sufficient current to drive a number of gates, (B) interfacing between two logic families, (C) obtaining a desired pulse rise time and (D) providing a slight delay to make pulses arrive at the proper time.

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 *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 commutativity and associativity. Proofs of the Boolean algebra theorems can be found in an introductory digital design textbook.

BASIC THEOREMS

Table 7.5 lists the theorems for a single variable and **Table 7.6** 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 "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: $\overline{A \bullet B} = \overline{A} + \overline{B}$ and its dual $\overline{A + B} = \overline{A} \bullet \overline{B}$. The truth table in **Table 7.7** 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 **Fig 7.13**.

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 in-

Table 7.5
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 7.6
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:	$A \cdot B + \overline{A} \cdot C + B \cdot C = A \cdot B + \overline{A} \cdot C$ $(A + B) \cdot (\overline{A} + C) \cdot (B + C) = (A + B) \cdot (\overline{A} + C)$

Table 7.7
DeMorgan's Theorem

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

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

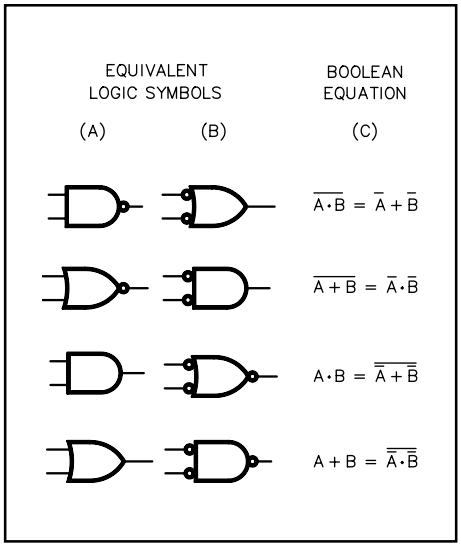


Fig 7.13 — Equivalent gates from DeMorgan's Theorom: Each gate in column A is equivalent to the opposite gate in column B. The Boolean equations in column C formally state the equivalences.

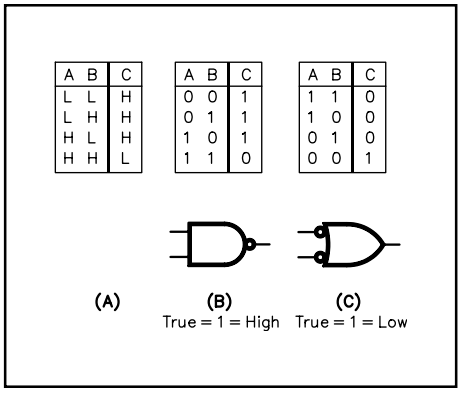


Fig 7.14 — (A) A general truth table, (B) a truth table and NAND symbol for positive logic and (C) a truth table and NOR symbol for negative logic.

verter 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.) The function of the table can differ depending on whether it is interpreted for positive logic or negative logic. **Fig 7.14** 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 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 "/," like /READY, while active high signals have no prefix. 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.

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*.

This section first introduces a number of terms necessary to understand sequential logic: types of synchronicity, types of control signals and ways to illustrate circuit function. Numerous sequential logic circuits are then introduced. These circuits provide an overview of the basic sequential circuits that are commercially available. Depending on your approach to learning, you may choose to either (1) read the material in the order presented, definitions then examples, or (2) start with the example circuits, which begin with the flip-flop, referring back to the definitions as needed.

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.

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.

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: (noun) a circuit that stores a bit of information or (verb) to hold at the same output state. *Gate* has many meanings, some unrelated to synchronous control: (A) a signal used to trigger the passage of other signals through a circuit (for example, "A gate circuit passes a signal only when a gating pulse is present."), (B) any logic circuit with two or more inputs and one output (used earlier in this chapter) or (C) one of the electrodes of an FET (as described in the **Analog Signals and Components** 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.

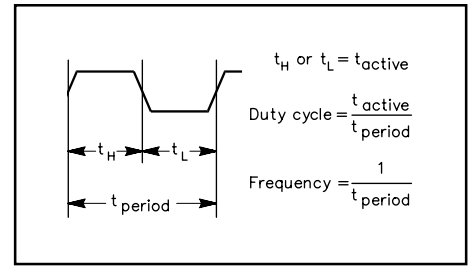


Fig 7.15 — Clock signal terms. The duty cycle would be t_H / t_{PERIOD} for an active high signal and t_L / t_{PERIOD} for an active low signal.

Clock control is the most common method, so it has some additional terms, illustrated by **Fig 7.15**. 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.

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 opposite 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 **Fig 7.16**. Input D 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.

ILLUSTRATING CIRCUIT FUNCTION

Since the action of sequential circuits is more complex, many ways have been developed to examine circuit function. **Fig 7.17** shows an example for each type of table or diagram. Each type has advantages and disadvantages.

State Transition Tables

Describing a sequential circuit with the conventional truth table would require an infinite number of

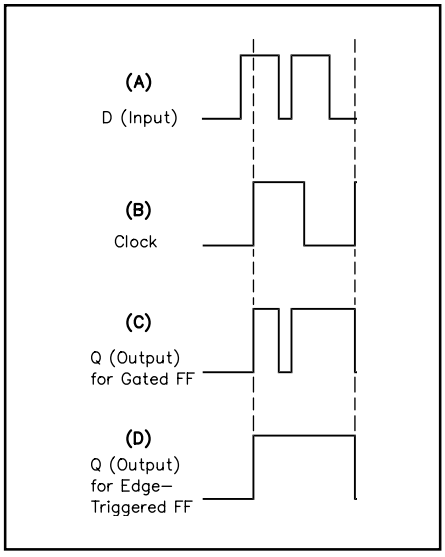


Fig 7.16 — Level-triggered vs edge-triggered for a D flip-flop: (A) input D, (B) clock input, (C) output Q for level-triggered: circuit responds whenever clock is 1. (D) output Q for edge-triggered: circuit responds only at rising edge of clock. Notice that the short negative pulse on the input D is not reproduced by the edge-triggered flip-flop.

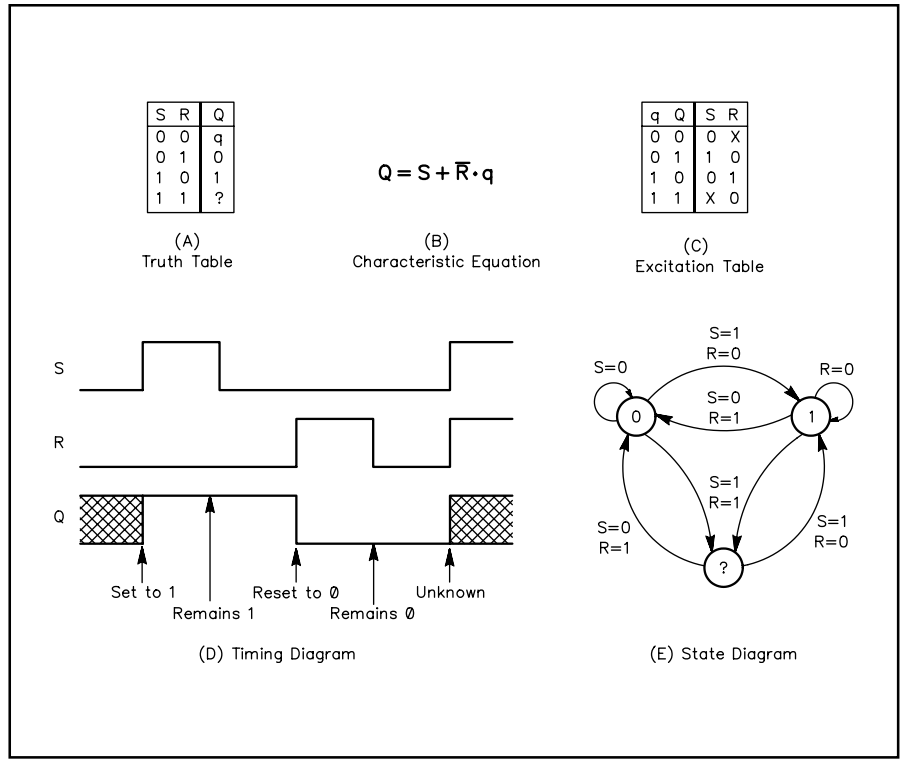


Fig 7.17 — Sequential circuit function for a clocked S-R flip-flop: (A) state transition table or truth table, (B) characteristic equation, including its derivation from the state table, (C) excitation table, (D) partial timing diagram, (E) state diagram.

previous events as the possible inputs. Instead, the sequential logic form of the truth table is called a state transition table. A *state transition* (or excitation) *table* lists all combinations of present inputs and feedback outputs — the *current state*, represented by small q, and the resulting outputs, both feedback and nonfeedback and the *next state*, represented by capital Q. The state transition table is the most common way to describe sequential circuit function, since it lists all possibilities.

Characteristic Equation

An equivalent equation for a state transition table or excitation table can be written in terms of the *state variables*, a symbol for each input and output. Although the number of state variables can be very large for a complex system, there is always a limited number of inputs and outputs; thus, the combinations of inputs and outputs is also limited. A circuit with n binary state variables has 2^n possible states. Since the possible states, 2^n, are always finite, sequential circuits are also known as *finite-state machines*. Each of the sequential logic circuits in this section includes a state table to illustrate the circuit functions. As before, these tables are for positive logic, so 1s and 0s are already substituted for Highs and Lows.

Excitation Table

An *excitation table* is derived from the truth table. Its usefulness is to show, for each possible output Q_n what inputs are needed to obtain a desired output Q_{n+1}. For some outputs, one or more of the input variables may not have an effect. In this case, the input variable corresponds to a *don't care* state, represented by an "X" or dash.

Timing Diagram

When a clock is the controlling signal, a timing diagram can describe the circuit operation. A *timing diagram* draws a circuit's signals as a function of time. This form emphasizes the cause-and-effect delays between critical signals. It is especially useful in detecting errors, as will be shown in the flip-flop implementations to be presented. There are software packages available to *simulate* a circuit. The simulation produces the timing diagram for a given set of inputs, allowing the designer to examine the timing diagram for expected results.

State Diagram

A *state diagram* depicts output changes in terms of a flow chart. Each possible state is listed inside a circle with arrows between the circles to indicate possible next states. State diagrams are especially useful for studying a sequence of inputs and the corresponding result.

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) or the *reset* state (0). (The reset state is also called the *cleared* state.) 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 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 first section examines the S-R flip-flop for each of the various control methods. The next section introduces each of the other basic flip-flops and their uses.

S-R Flip-Flop

The S-R flip-flop is one of the simplest circuits for storing a bit of information. It has two inputs,

represented by S (set) and R (reset). These inputs, naturally, cause the two possible output states: if S = 1 and R = 0, then output Q is set to 1; if S = 0 and R = 1, then output Q is reset to 0; if both inputs are 0, then the output remains unchanged; and if both inputs are 1, then the output cannot be determined. The S-R flip-flop can illustrate each of the types of control signals: unclocked (asynchronous, no control signal), clocked or gated, master-slave and edge-triggered.

Unclocked/Sequential

The unclocked S-R flip-flop, shown in **Fig 7.18**, is an asynchronous device; its outputs change immediately to reflect changes on its inputs. The circuit consists of two NOR gates. The sequential nature of the circuit is a result of the output of each NOR gate being fed back as an input to the opposite gate. The state transition table shows the expected set/reset pattern of inputs to outputs. The table shows an unpredictable result for inputs S = 1 and R = 1. In actual circuits, the results vary and are usually either Q = $\overline{Q}$ = 1 or Q = $\overline{Q}$ = 0. While Q = $\overline{Q}$ is a logical impossibility, real flip-flops may present this output. The designer should avoid the R = S = 1 input and make no assumptions about the resulting output. The flip-flop is not predictable if both inputs go to 0 at exactly the same time.

Fig 7.18C shows an alternate implementation of the S-R flip-flop, with two NAND gates and two inverters. Since a NAND gate can become an inverter by having its two inputs receive the same signal, the S-R flip-flop can be implemented with four NAND gates. This alternate version is important because a 4-NAND gate chip is one of the most readily available commercial integrated circuits; thus, the 4-NAND gate S-R flip-flop can be implemented on a single IC.

Gated or Level-Triggered

The gated S-R flip-flop, or gated latch, has a controlling input in addition to its S and R inputs. The inputs S and R produce the same results as those on an unclocked S-R flip-flop, but a change in output will only occur when the control input is high. A gated S-R flip-flop is illustrated in **Fig 7.19** along with a timing diagram for a clock input. This flip-flop is also called the R-S-T flip-flop, where "T," for toggle, is the clock input. Although not often used, the R-S-T flip-flop is important because it illustrates a step between the R-S flip-flop and the J-K flip-flop.

A problem with the level-triggered flip-flop is that the Q output can change more than once while the clock is asserted. We would prefer the output to change only once per clock period for easier timing design. A second problem can occur when flip-flops are connected in series and triggered by the same clock pulse or, similarly, when a flip-flop is in series with itself, using its own output as an input. Since the series-connected flip-flop feeds back to itself, its output will be changing at about the same time as it receives new input. This can result in an erroneous output.

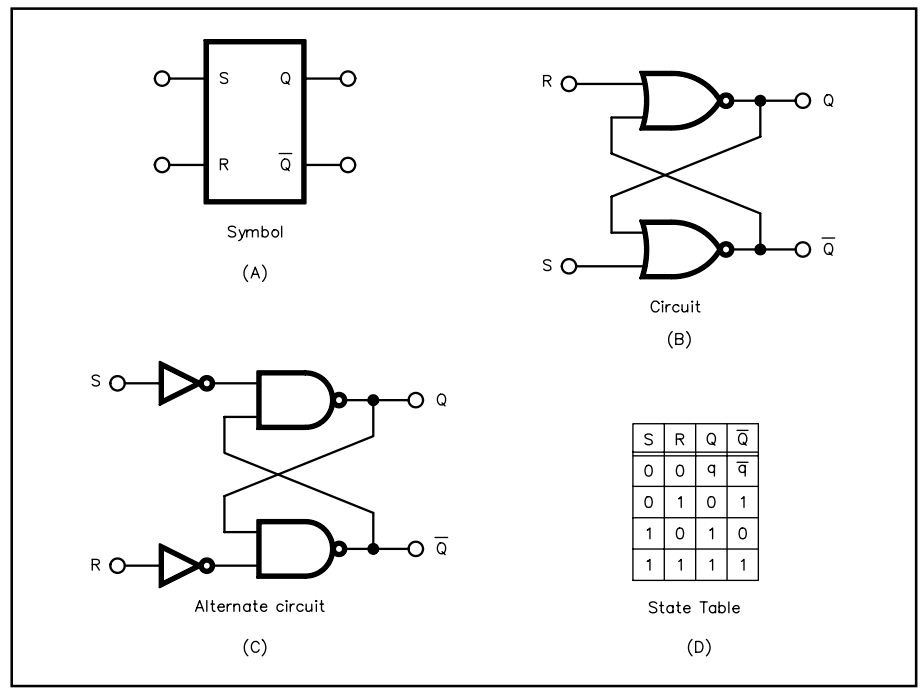


Fig 7.18 — Unclocked S-R Flip-Flop. (A) schematic symbol. (B) circuit diagram. (C) alternate circuit. (D) state table.

Master/Slave Flip-Flop

A solution to the problems of the level-triggered method 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 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 is active when the clock is high, sampling and storing the inputs. The second, or *slave*, flip-flop gets its input from the master and acts 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. A master/slave flip-flop is built with either two S-R flip-flops, as shown by **Fig 7.20**, or with two J-K flip-flops. Note that a bubble appears 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.

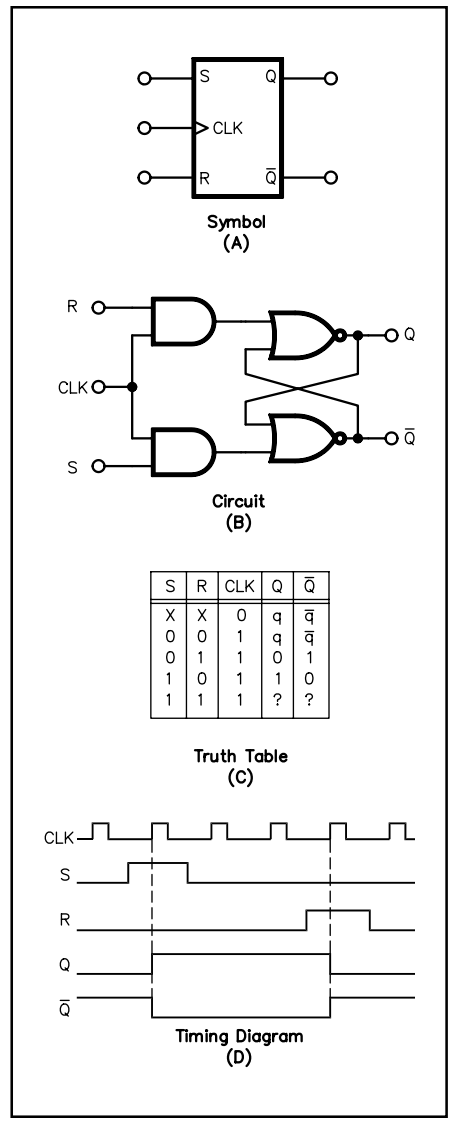


Fig 7.19 — Clocked S-R Flip-Flop. (A) schematic symbol. (B) circuit. (C) truth table. (D) timing diagram.

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.

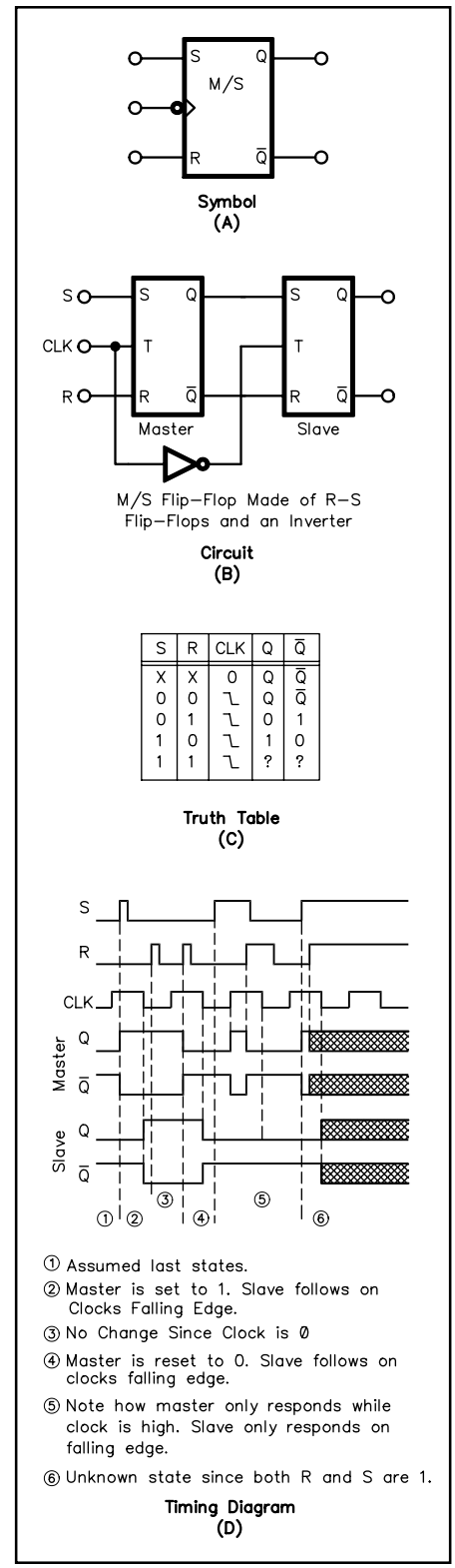


Fig 7.20 — Master/Slave S-R Flip-Flop. (A) schematic symbol. (B) circuit. (C) truth table. (D) timing diagram.

Edge-Triggered Flip-Flop

The edge-triggered flip-flop solves the problem of noise. An example of noise is the glitch shown earlier in Fig 7.16. The different outputs for the level and edge-triggered methods in this figure show

how a glitch can cause an output error. Edge-triggering avoids the problem of noise by minimizing the time during which a circuit responds to its 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 such as ⌜ or ⌞; sometimes an arrow is included such as ⇡ or ⇣. This symbol appears in the circuit's truth table and can also appear, instead of the clock triangle, inside the schematic symbol.

Other Flip-flops

Table 7.8 provides a summary of the four basic flip-flops: the S-R (Set-Reset), D (Data or Delay), T (Toggle) and J-K. Each is briefly explained below, including its particular applications. The internal circuitry of each of these flip-flops is similar to the components and complexity of the S-R flip-flop. Readers may be interested in trying to design their own circuit implementation for a flip-flop type and control method; however, in practical use, a commercially available integrated circuit chip would probably be used. Company databooks include the individual circuit implementation for each IC. Digital design textbooks will also show sample circuit implementations for each of the flip-flops.

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 input D is transferred to Q when the clock is positive; the Q output retains this logic level until the next positive clock pulse (see **Fig 7.21**). The flip-flop is also called a *delay* flip-flop because, once enabled, it passes D after a propagation delay. A D flip-flop is useful to store one bit of information. A collection of D flip-flops forms a register.

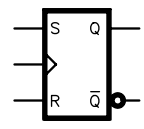
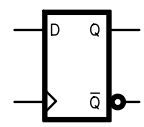
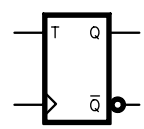
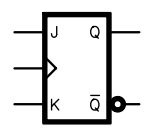
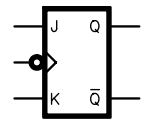
Toggle Flip-Flop

In a T flip-flop, the output *toggles* (changes state) with each positive clock pulse. The T flip-flop is also called a *complementing* flip-flop. **Fig 7.22** shows how a T flip-flop can be created from either an S-R or D flip-flop. The timing diagram in Fig 7.22 shows an important result of the T-flip-flop: the output frequency is one half of the input frequency. Thus, a T flip-flop is a

Table 7.8

Summary of Standard Flip–Flops

q = current state Q = next state X = don't care

Flip–Flop Type	Symbol	Truth Table	Characteristic Equation	Excitation Table
SR		S R CLK Q X X 0 q 0 0 1 q 0 1 1 0 1 0 1 1 1 1 1 ?	$Q = \left(S + \bar{R} \cdot q\right) \cdot CLK$ with $S \cdot R = 0$	q Q S R CLK 0 0 0 X 1 0 1 1 0 1 1 0 0 1 1 1 1 X 0 1
D		D CLK Q X 0 q 0 1 0 1 1 1	$Q = D \cdot CLK$	q Q D CLK 0 0 X 0 0 1 1 1 1 0 0 1 1 1 X 0
T		T CLK Q X 0 q 0 1 q 1 1 $\bar{q}$	$Q = (T \oplus q) \cdot CLK$	q Q T CLK 0 0 0 X 0 1 1 1 1 0 1 1 1 1 0 X
JK		J K CLK Q X X 0 q 0 0 1 q 0 1 1 0 1 0 1 1 1 1 1 $\bar{q}$	$Q = \left(J \cdot \bar{q} + \bar{K} \cdot q\right) \cdot CLK$	q Q J K CLK 0 0 0 X 1 0 1 1 X 1 1 0 X 1 1 1 1 X 0 1
JK		J K CLK Q 0 0 ↓ 0 0 1 ↓ 0 1 0 ↓ 1 1 1 ↓ $\bar{q}$	$Q = \left(J \cdot \bar{q} + \bar{K} \cdot q\right) \cdot CLK$	q Q J K CLK 0 0 0 X ↓ 0 1 1 X ↓ 1 0 X 1 ↓ 1 1 X 0 ↓

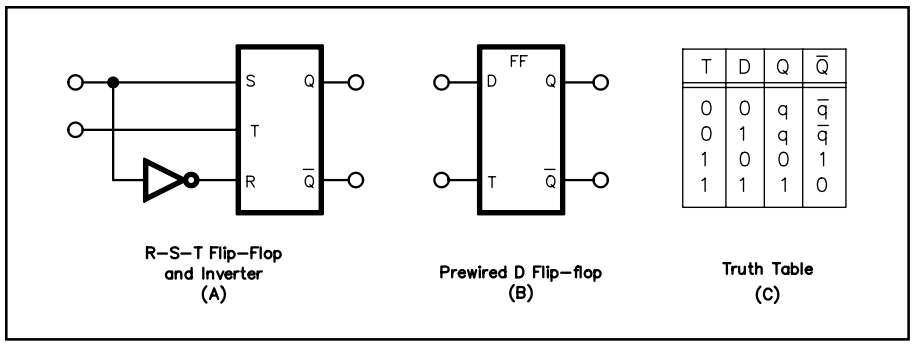


Fig 7.21 — (A) The D flip-flop. When T = 0, Q and $\overline{Q}$ states don't change. When T = 1, the output states change to reflect the D input. (C) A truth table for the D flip-flop.

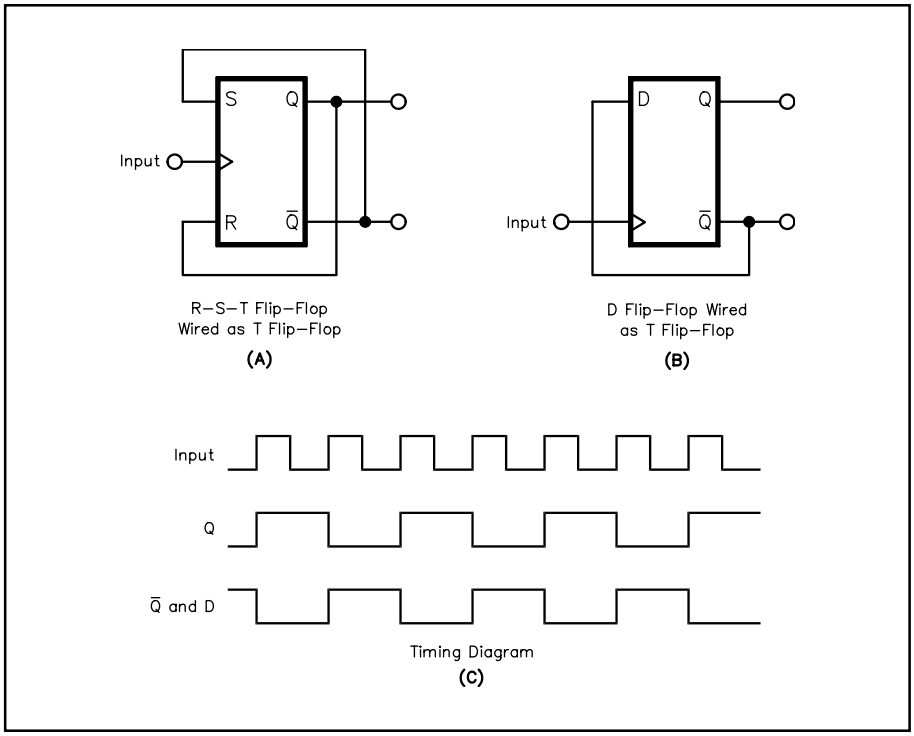


Fig 7.22 — (A) A clocked S-R-T flip-flop wired as a T flip-flop. (B) A D flip-flop wired as a T flip-flop. (C) Timing diagram. Notice that the output frequency is half the input frequency.

2:1 (also called *modulo-2* or *radix-2*) frequency divider. Two T flip-flops connected in series form a 4:1 divider and so on.

J-K Flip-Flop

It's somewhat ironic that the most readily available flip-flop, the J-K flip-flop, is discussed last and so briefly. The discussion is short because the J-K flip-flop acts the same as the S-R flip-flop (where J = S and K = R) with only one difference: The S-R flip-flop had the disadvantage of invalid results for the inputs 1,1. For the J-K flip-flop, simultaneous 1,1 inputs cause Q to change state after the clock transition.

Summary

Only the D and J-K flip-flops are generally available as commercial integrated circuit chips. Since memory and temporary storage are so often desirable, the D flip-flop is manufactured as the simplest way to provide memory. When more functionality is needed, the J-K flip-flop is available. The J-K flip-flop can substitute for an S-R flip-flop and a T flip-flop can be created from either the D or J-K flip-flop.

COUNTERS

Groups of flip-flops can be combined to make counters. Toggle flip-flops are the most common for implementing a counter. 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 **Fig 7.23**. 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, Fig 7.23A, the clock is also passed from stage to stage and the circuit is called *ripple* or *ripple-carry*. In the synchronous counter, Fig 7.23B, each stage is controlled by a common clock signal.

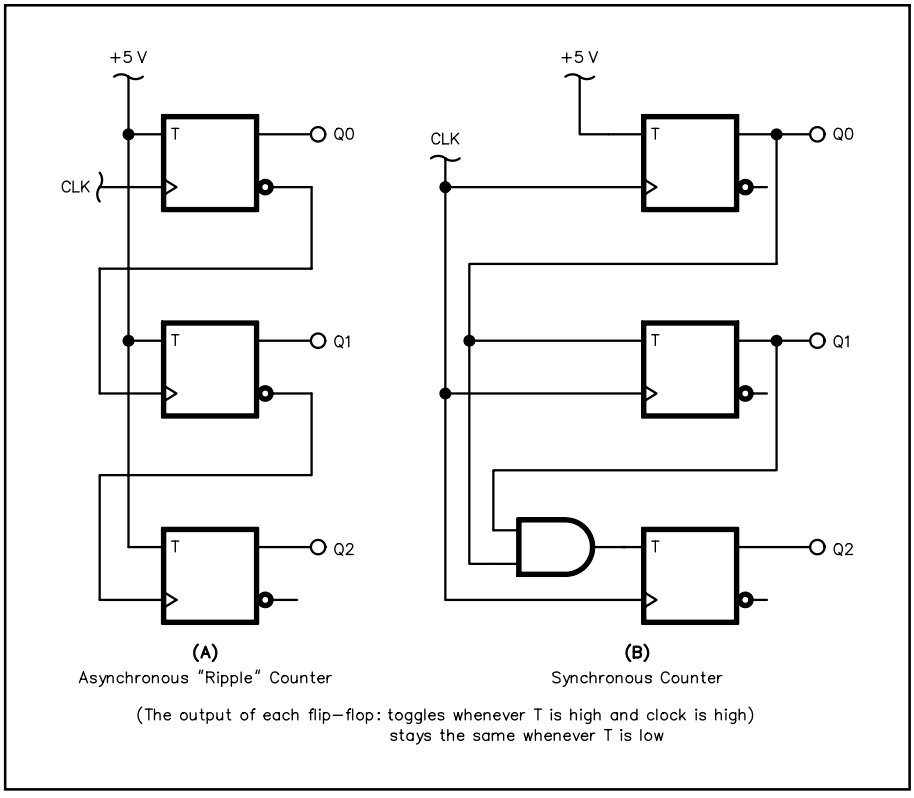


Fig 7.23 — Three-bit binary counter: (A) asynchronous or ripple counter, (B) synchronous counter.

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*. **Fig 7.24** shows a typical circuit and schematic symbols for an 8-bit storage register. In (C), although the bits are passed on 8 separate lines (from 8 flip-flops), a slash and number, "/8," is used to simplify the symbol. Storage registers are important to computer architecture; this topic is discussed in depth later in the chapter.

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

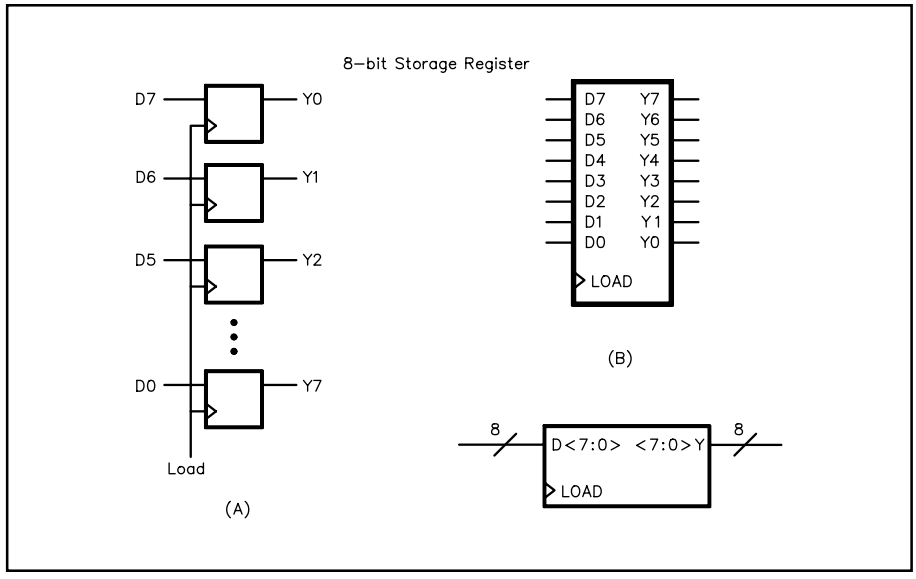


Fig 7.24 — An eight-bit storage register: (A) circuit, (B) and (C) schematic symbols.

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. This is useful in interfacing between devices, and is discussed in detail in the Digital Interfacing section.

Additional uses for a shift register are to (1) delay or synchronize data, (2) multiply or divide a number by a factor 2^n or (3) provide random data. 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 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. Random data is provided via a ring counter. A *ring counter* is a shift register with its output fed back to its input. At each clock pulse, the register is shifted up or down and some of the flip-flops feedback to other flip-flops, generating a random binary number. Shift registers with several feedback paths can be used as a *pseudorandom number generator*, where the sequence of bits output by the generator meets one or more mathematical criteria for randomness.

MULTIVIBRATORS

A multivibrator is widely used as a switch and comes in three basic forms: bistable, monostable and astable. It is broadly defined as a closed-loop, regenerative circuit that alternates between two stable or quasi-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. To create *quasi-stable* or *unstable* states, energy-storing devices (capacitors) are added in the feedback loops of the multivibrator; the instability is a result of the exponential decay of the stored energy. A *monostable multivibrator* is the result of adding one energy-storing element into a feedback loop. An *astable multivibrator* is the result of adding two energy-storing elements, one in each feedback loop.

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, the one-shot multivibrator puts out a pulse of some duration, T. (Note that T is not the period, but the duration of the quasi-stable state.) Triggering during the stable state results in the pulse, as expected. Triggering during the unstable state has two possibilities: A *nonretriggerable* multivibrator is not affected. A *retriggerable* multivibrator will start counting its pulse duration from the most recent trigger pulse. Both types of one-shots are common.

Fig 7.25 shows a 555 timer IC 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 is the resistance in ohms and C is the capacitance 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.

Fig 7.26 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:

$$f = 1.46 / (R1 + 2\ R2)\ C$$

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

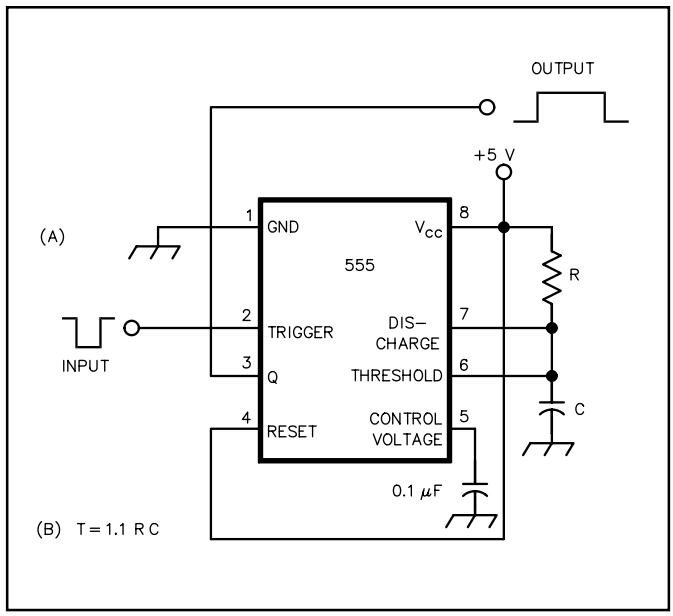


Fig 7.25 — (A) A 555 timer connected as a monostable multivibrator. (B) The equation to calculate values for R (in ohms) and C (in farads), where T is the pulse duration (in seconds).

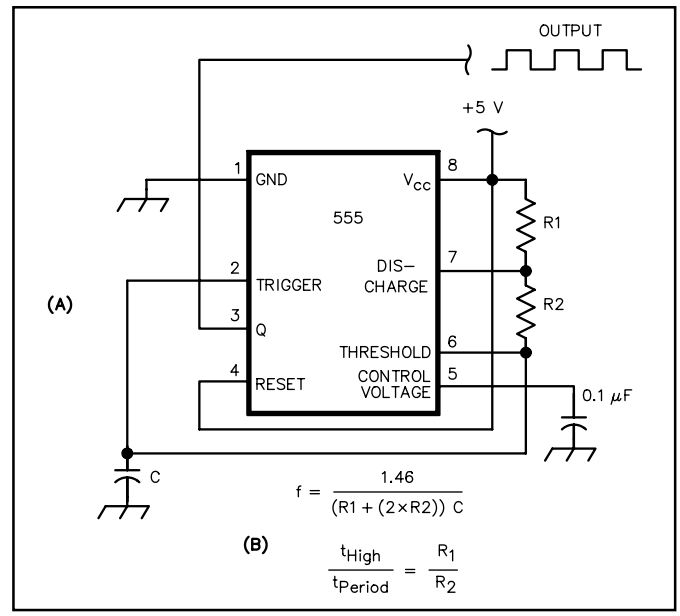


Fig 7.26 — (A) A 555 timer connected as an astable multivibrator. (B) The equations to calculate values for R1, R2 (in ohms) and C (in farads), where f is the clock frequency (in Hertz).

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.

Applications

An astable multivibrator is useful in generating clock pulses. When triggered by a clock pulse, the one-shot multivibrator acts to lengthen or "stretch" the pulse, which is useful to delay digital events. Either of these pulse signals, when input to the bistable multivibrator (flip-flop), can be the control of a sequential circuit. The three types of multivibrators can ensure synchronicity, so that a sequential circuit will execute correctly.

SUMMARY

Digital logic plays an increasingly important role in Amateur Radio. Most of this logic is binary and can be described and designed using Boolean algebra. Using the NOT, AND and OR gates of combinational logic, designers can build sequential logic circuits that have memory and feedback. The simplest sequential logic circuit is called a flip-flop. By using control inputs, a flip-flop can latch a data value, retaining one bit of information and acting as memory. Combinations of flip-flops can form useful circuits such as counters, storage registers and shift registers. The primary method of controlling sequential circuits is via a clock pulse, which can be created with a multivibrator.

Digital Integrated Circuits

Integrated circuits (ICs) are the cornerstone of digital logic devices. Modern technology has enabled electronics to become miniature in size and less expensive. Today's complex digital equipment would be impossible with vacuum tubes or even with discrete transistors.

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.

This chapter will primarily deal with SSI ICs, the basic digital building blocks. Microprocessors, memory chips and programmable logic devices are discussed later in the Computer Hardware section.

The previous section discussed the design of a digital circuit. To build that circuit, the designer must choose between IC chips available in various logic families. Each family and subfamily has its own desirable characteristics. This section reviews the primary IC logic families of interest to radio amateurs. The designer may also be challenged to interface between different logic families or between a logic device and peripheral device. The former is discussed at the end of this section; the latter with Computer Hardware, later in the chapter.

COMPARING LOGIC FAMILIES

When selecting devices for a circuit, a designer is faced with choosing between many families and subfamilies of logic ICs. Which subfamily is right for the application at hand is among several desirable characteristics: logic speed, power consumption, fan-out, noise immunity and cost.

Speed

Logic device families operate at widely varying clock speeds. Standard transistor-transistor logic (TTL) devices can only operate up to a few MHz while some emitter-coupled logic (ECL) ICs can operate at several GHz. Gate propagation delay determines the maximum clock speed at which an IC can operate; the clock period must be long enough for all signals within the IC to propagate to their destinations. ICs with capacitively coupled inputs have minimum, as well as maximum, clock rates. While the initial reaction may be to use the fastest available ICs, the designer must usually choose between a trade-off of high speed and low power consumption.

Power Consumption

In some applications, power consumption by logic gates is a critical design consideration. This power consumption can be divided into two parts: Dynamic power is the power consumed when a gate changes state. Static power is the power consumed when a gate is holding a state, either high or low. Each of these has different power requirements. Calculating the total required power can be a complex task when several gates and diverse functions are involved. Nominal power requirements, however, can be used to compare logic subfamilies.

Fan-out

Gate impedance is another parameter to consider. To deliver high current without dropping consid-

erable voltage, an ideal gate would have low output impedance. To draw minimal current, an ideal gate would have infinite input impedance. Such a gate does not exist. The designer must compromise on input and output impedances.

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 **Fig 7.27**.

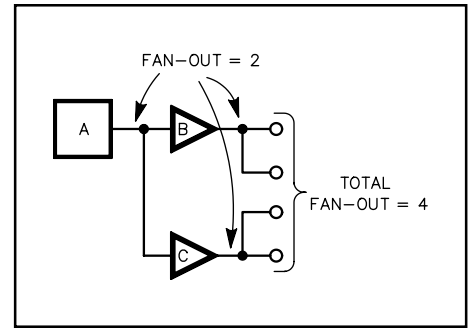


Fig 7.27 — Nonverting buffers used to increase fan-out: Gate A (fan-out = 2) is connected to two buffers, B and C, each with a fan-out of 2. Result is a total fan-out of 4.

Noise Immunity

The noise margin was illustrated in Fig 7.2. 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.

Other Considerations

The parameters above are the basic considerations to influence the selection of a logic family for a specific application. These considerations have complex interactions that come into play in demanding low-current, high-speed or high-complexity circuits. Numerous other parameters can also be examined. These are provided in the electrical specification of a device. These are given on a data sheet and usually include four sections: (1) absolute maximum ratings specify worst-case conditions, including safe storage temperatures, (2) recommended operating conditions specify power-supply, input voltage, dc output loading and temperatures for normal operation, (3) electrical characteristics specify other dc voltages and currents observed at the inputs and outputs and (4) switching characteristics specify propagation delays for “typical” operation.

The list of parameters can seem overwhelming to the novice; but with experience, the important information will be more easily spotted. If you are designing a circuit, always consult the data sheet for specific information on the device you are considering, because these parameters vary not only between the logic families but also vary between the manufacturers and with changing technologies. Each manufacturer has data books available listing their devices and the corresponding data sheets.

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.

Bipolar semiconductor ICs usually employ NPN junction transistors. (Bipolar ICs are possible using PNP transistors, but NPN transistors make faster circuits.) While early bipolar logic was faster and had higher power consumption than MOS logic, these distinctions have blurred 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 digital 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 has seen widespread acceptance because it is fast and has good noise immunity. It is by far the most commonly used logic family. TTL levels were shown earlier in Fig 7.2: An input voltage between 0.0-0.4 V will represent LOW and an input voltage between 2.4-5.0 V will represent HIGH.

TTL Subfamilies

The original standard TTL is infrequently used today. In the standard TTL circuit, the transistors saturate, reducing the operating speed. TTL variations cure this by clamping the transistors with Schottky diodes to prevent saturation, or by using a dopant in the chip fabrication to reduce transistor recovery time. Schottky-clamped TTL is the faster of these two manufacturing processes.

TTL IC identification numbers begin with either 54 or 74. The 54 prefix denotes a 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. Recall, from Fig 7.18, the alternative implementation of the S-R flip-flop.) The following TTL subfamilies are available:

	74xx	*standard TTL*
H	74Hxx	High-speed
L	74Lxx	Low-power
S	*74Sxx*	*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. Because the speed-power product is approximately constant, less power consumption results in less speed and vice versa. For the amateur, an additional consideration to the speed-versus-power trade-off is the cost trade-off. The advanced Schottky devices offer both increased speed and reduced power consumption but at a higher cost.

In addition to the above power/speed/cost trade-offs, each TTL subfamily has particular characteristics that can make it suitable or unsuitable for a specific design. **Table 7.9** shows some of these parameters. The actual parameter values may vary slightly from manufacturer to manufacturer so always consult the manufacturers' data books for complete information.

TTL Circuits

Fig 7.28A shows the schematic representation of a TTL hex inverter. A 7404 chip contains four of these inverters. When the input is low, Q1 is ON, conducting current from base to emitter through the input lead and into ground. Thus a low TTL input device must be prepared to sink current from the input. Since Q1 is saturated, Q2 is OFF because there is not enough voltage at its base. Similarly, Q4 is also OFF. With Q2 and Q4 OFF, Q3 will be ON and pull the output high, about one volt below V_{CC}. When the input is high, an unusual situation occurs: Q1 is operating in the inverse mode, with current flowing from base to collector. This current causes Q2 to be ON, which causes Q4 to be ON. With Q2 and Q4 ON, there is not enough current left for Q3, so Q3 is OFF. Q4 is pulling the output low.

By replacing Q1 with a multiple-emitter transistor, as is done with the two-input Q5 in Fig 7.28B, the

Fig 7.28 — 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.)

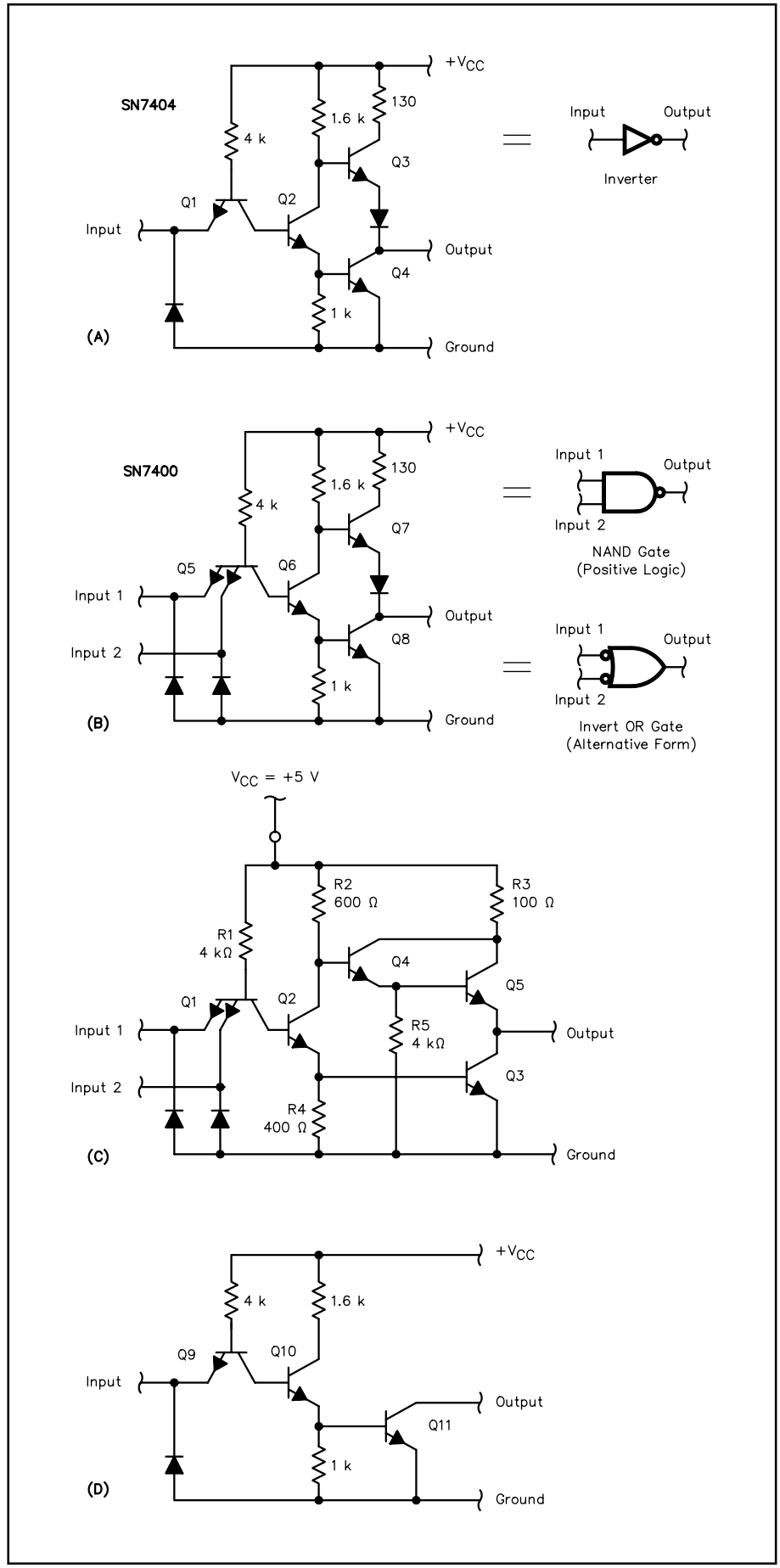


Table 7.9
TTL and CMOS Subfamily Performance Characteristics

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
LS	9	2	18
AS	1.6	20	32
ALS	5	1.3	6.5

CMOS Family Operating with 4.5 <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	4.75	0.080	0.755	7.505	0.4	3.6	36

inverter circuit becomes a NAND gate. Commercially available TTL NAND gates have as many as 13 inputs, the limiting factor being the number of input pins on the standard 16-pin chip. The operation of this multiple-input NAND circuit is the same as described for the inverter, the difference being that any one of the emitter inputs being low will conduct current through the emitter, leading to the conditions described above to produce a high at the output. Similarly, all inputs must be high to produce the low output.

In the TTL circuit of Fig 7.28A, transistors Q3 and Q4 are arranged in a *totem-pole* configuration. This configuration gives the output circuit a low source impedance, allowing the gate to source (supply) or sink substantial output current. The 130-Ω resistor between the collector of Q3 and $+V_{CC}$ limits the current through Q3.

When a TTL gate changes state, the amount of current that it draws changes rapidly. These changes in current, called switching transients, 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, 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 (disk-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 rather than rolled-foil mylar or aluminum-electrolytic capacitors.

Darlington and Open-Collector Outputs

Fig 7.28C 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 fanout 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 light-emitting diodes (LEDs), relays and solenoids. Inductive devices like relay coils and solenoids need a "flyback" 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 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. **Fig 7.29** 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 nonTTL 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 enable is low, the device provides high and low outputs just as it would normally; when enable is high 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 re-

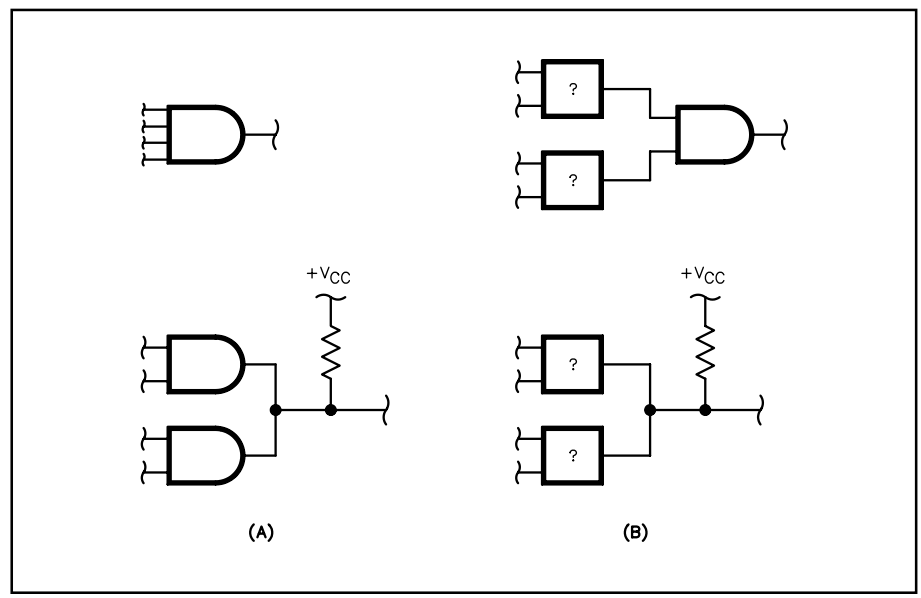


Fig 7.29 — The outputs of two open-collector-output AND gates are shorted together (wire ANDed) to produce an output the same as would be obtained from a 4-input AND gate.

maining 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.

Emitter-Coupled Logic (ECL)

ECL, also called current-mode logic (CML), is the fastest commercially available logic family, with some devices operating at frequencies higher than 1.2 GHz. The fast speed is a result of reducing the propagation delay by keeping the transistors from saturating. ECL devices operate with their transistors in the active region. The voltage swing is small, less than a volt; and the circuit internally switches between two possible paths depending on the output state. The two-path arrangement provides a significant feature of ECL: complementary output states are always available.

Naturally, high speed comes at some cost, in this case high power consumption. Heat sinking is sometimes necessary because of the great deal of power being dissipated. Because of its poor speed-power product and also because it is not directly compatible with TTL and CMOS, ECL is less popular than TTL. ECL devices are most likely to be found where performance is more important than cost, including UHF frequency counters, UHF frequency synthesizers and high-speed mainframe computers.

ECL Subfamilies

There are several ECL subfamilies, to balance the trade-off of high speed versus low power dissipation. The subfamilies differ mostly in resistance values and the presence or absence of input and output pull-down resistors.

The most popular subfamily is the 10K series, with five-digit part numbers of the form "10xxx." This family's design started one of ECL's most familiar characteristics: operation with V_{CC} = 0 V (ground) and V_{EE} at a negative voltage. This feature provides immunity to power supply noise, since noise on V_{EE} is rejected by the circuit's differential amplifier. The design voltage, V_{EE} = –5.2 V for the 10K subfamily, provides the best noise immunity; but other voltages can be used. Typically, a high logic state corresponds to -0.9 V and a low is -1.75 V.

ECL Circuits

ECL gets its name from the emitter-coupled pair of transistors in the circuit, connected as a differential amplifier. For example, in **Fig 7.30**, either Q1 or Q2 together with Q3 form a differential amplifier. This arrangement produces the complementary outputs available from each ECL circuit. The circuit in Fig 7.30 provides both an OR output and a NOR output. When an input is high, its transistor (Q1 or Q2) is

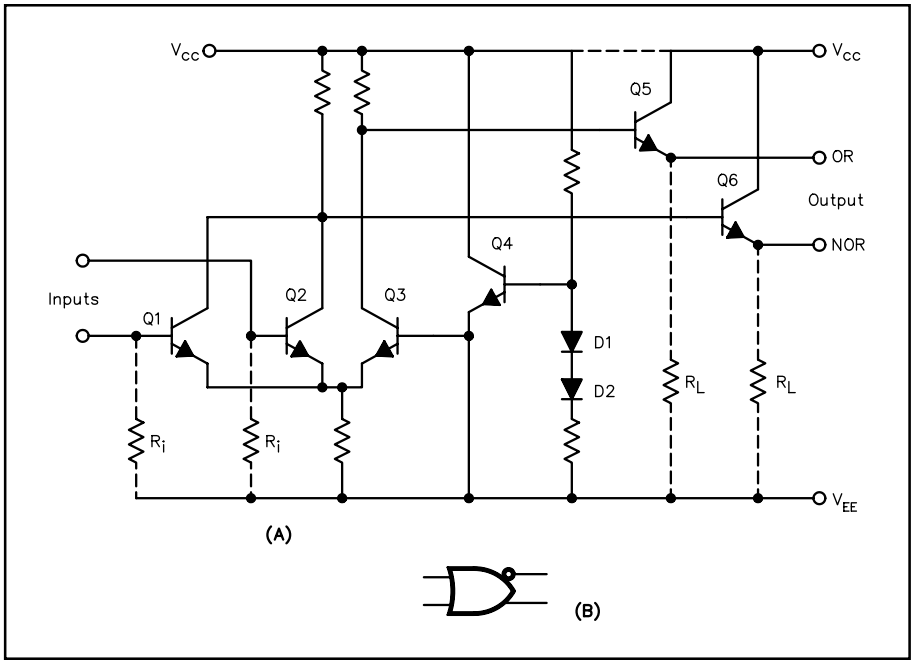


Fig 7.30 — (A) Circuit topology of the ECL family. (B) The modified logic symbol to indicates the availability of the complementary output.

ON but not in saturation; and Q3 is OFF. Q6 is then OFF so its emitter output is low, while Q5 is ON and its output high. Similarly, when both inputs are low, Q1 and Q2 are OFF so the NOR output from Q6 is high; and Q3 is ON, so the OR output from Q5 is low. Q4, D1, D2 and associated circuitry form a bias generator. The reference voltage at the base of Q3 determines the input switching threshold.

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 are used extensively in digital watches, calculators and VLSI circuits such as microprocessors and memories.

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 lower. The only extensive use of PMOS is in calculators and watches, where low speed is acceptable and low power consumption and low cost are desirable.

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.

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 $^1/_4$-wavelength long between input and output. The output is a low-impedance point so the trace is effectively grounded at this point. You can get high RF potentials $^1/_4$-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 number of CMOS subfamilies available. Like TTL, the original CMOS has largely been replaced by later subfamilies using improved technologies. This 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 provided improved performance and 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 3 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.

The following CMOS device families are available:

4000	4071B	standard CMOS
C	74Cxx	CMOS versions of TTL

Devices in this 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 this 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.

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.

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

CMOS Circuits

A simplified diagram of a CMOS logic inverter is shown in **Fig 7.31**. 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, the high current flows to the output. When the input is high, the opposite occurs: Q2's resistance is low, Q1's is high and the output is low. The diodes are to protect the circuit against static charges.

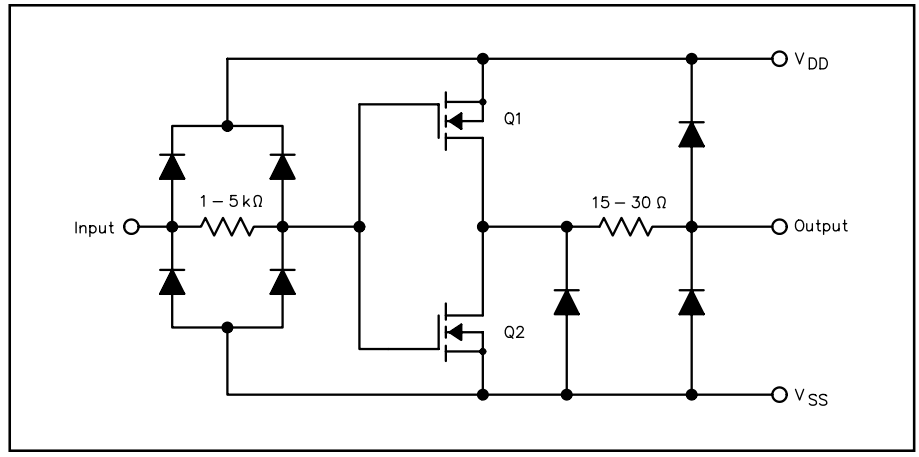


Fig 7.31 — Internal structure of a CMOS inverter.

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 material 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 any current that might flow through your body.

All CMOS inputs should be tied to an input signal; a positive supply voltage or ground if a constant input is desired. Undetermined CMOS inputs, even on unused gates, may cause gate outputs to oscillate. Oscillating gates draw high current, 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. If you want low power consumption, you should probably use CMOS. If you want ultra-high-speed logic, you will have to use ECL. Whatever the application, consult up-to-date literature when designing logic circuits. IC databooks and applications notes are usually available from IC manufacturers and distributors.

INTERFACING LOGIC FAMILIES

Each semiconductor logic family has its own advantages in particular applications. For example, the highest frequency stages in a UHF counter or a frequency synthesizer would use ECL. After the frequency has been divided down to less than 25 MHz, the speed of ECL is unnecessary; and its expense and power dissipation are unjustified. TTL or CMOS are better choices at lower frequencies.

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 between the point at which one logic family ends and another begins. A knowledge of the specific input/output (I/O) characteristics of each device is necessary, and a knowledge of the general internal structure is desirable, to ensure reliable digital interfaces. Typical internal structures have been illustrated for each common logic family. **Fig 7.32** illustrates the logic level changes for different TTL and CMOS families; databooks 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.

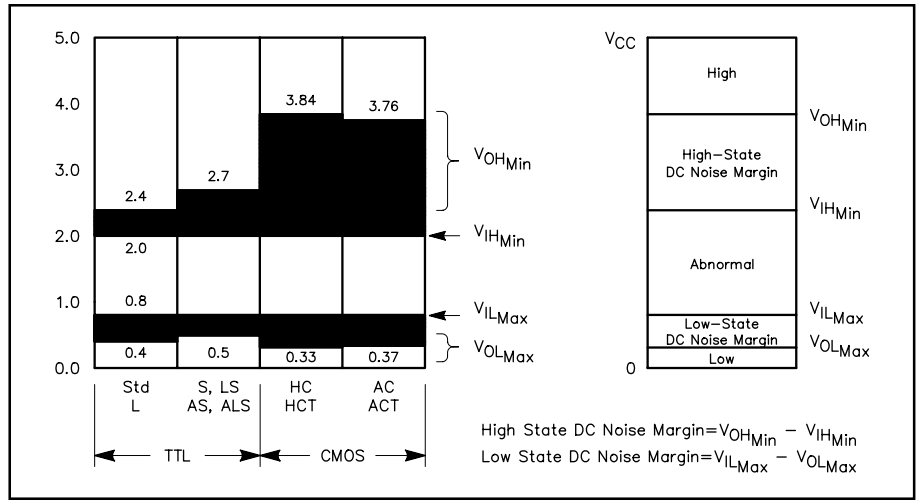


Fig 7.32 — Differences in logic levels for some TTL and CMOS families.

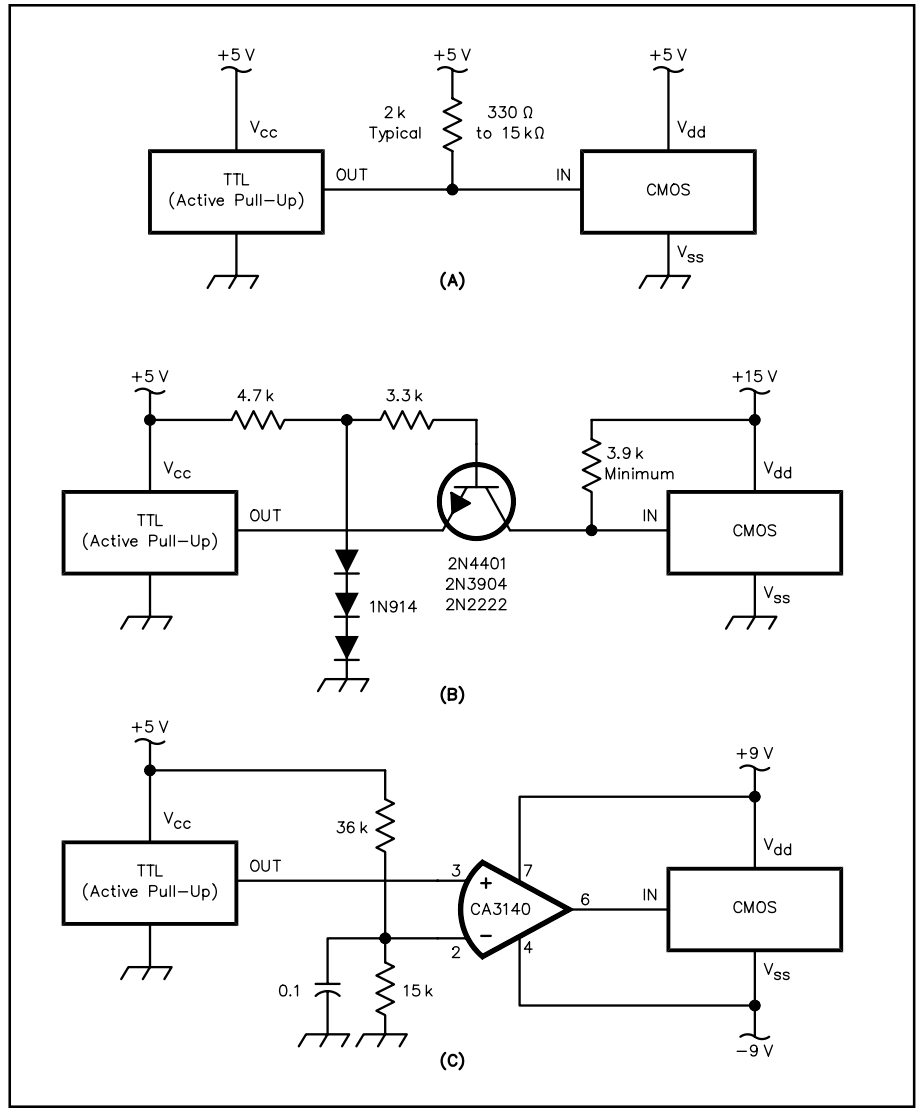


Fig 7.33 — TTL to CMOS interface circuits: (A) pull-up resistor, (B) common-base level shifter and (C) op amp configured as a comparator.

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 **Fig 7.33A**, 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. Resistors values in the range 1.5 kΩ to 4.7 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 nonCMOS devices to a CMOS system. An HCT device acts as a simple buffer between the nonCMOS (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 Fig 7.33B, 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 Fig 7.33C. An FET op amp is shown because its output voltage can usually swing closer to the rails (+ and – supply voltages) than a bipolar unit.

CMOS Driving TTL

Certain CMOS devices 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 **Fig 7.34A**. The diode blocks the high voltage from the CMOS gate when it is in the high output state. A germanium diode is used because its lower forward-voltage drop provides higher noise immunity for the TTL device in the low state. The 68-kΩ resistor pulls the input high when the diode is back biased.

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 Fig 7.34B. The CD4049 is an inverting buffer that has the same capabilities as the CD4050.

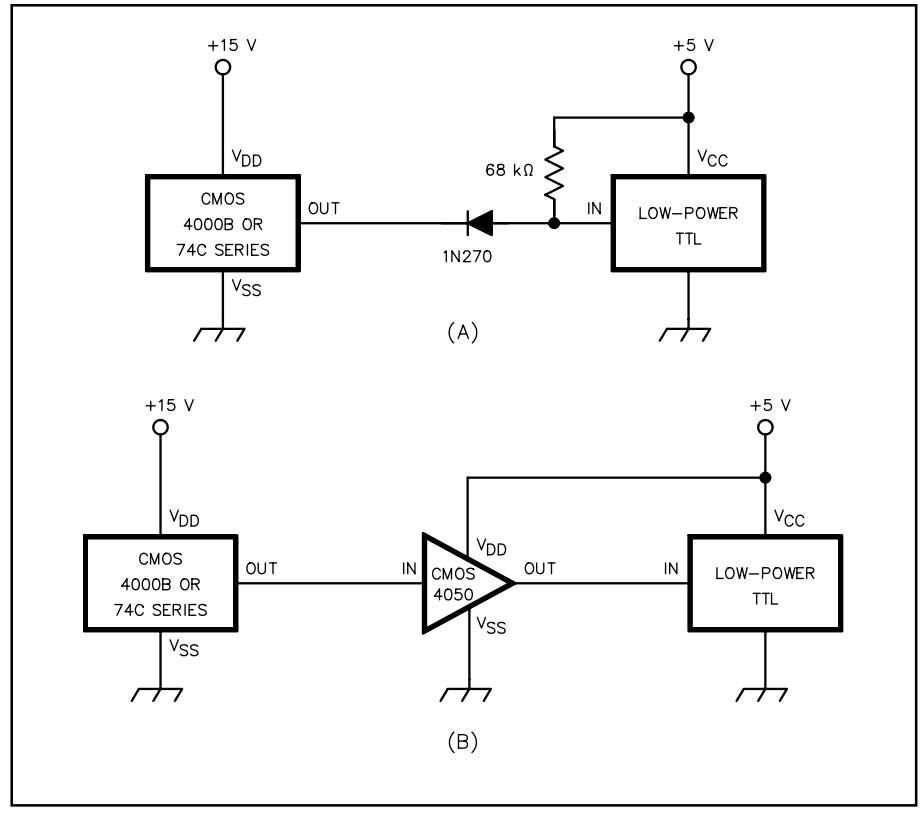


Fig 7.34 — CMOS to TTL interface circuits: (A) blocking diode used when different supply voltages are used. The diode is not necessary if both devices operate with a +5 V supply. (B) CMOS noninverting buffer IC.

Computer Hardware

So far, this chapter has discussed digital logic, the implementation of that logic with integrated circuits, interfacing IC logic families and the use of memory to store information used by the ICs. The synthesis of all this technology is the microcomputer — combining a microprocessor IC, memory chips and user interface into the modern digital computer. A computer has both physical components; hardware and a collection of programs; software, to tell it what to do. This section will focus on the physical components of the computer: its internal physical components, the chips and how they work and interact and its external I/O devices for communication with a user.

COMPUTER ORGANIZATION

The *architecture* of a computer is the arrangement of its internal subsystems: the microprocessor(s), memory, I/O and interfacing. Each subsystem may be concentrated on a single IC or spread between many chips. The microprocessor, also known as the *central processing unit* (*CPU*) and usually a single chip, consists of three parts: a control unit, an arithmetic logic unit (ALU) and temporary storage registers. A *bus* — a set of wires to carry address, data and control information — interconnects all of the subsystems. Most modern computers are some variation on the basic architecture shown in **Fig 7.35**.

The microprocessor, memory chips and other circuitry are all part of the system's *hardware*, the physical components of a system. The computer case, the nuts and bolts and physical parts are other parts of the hardware. A computer also includes *software*, a collection of programs or sequence of instructions to perform a specified task. Some microprocessors internally are complete circuitry. The design of general purpose computers is so complex, however, that it is nearly impossible to design an original architecture without any bugs. Thus many designers use microprocessors that include *microcode* or microinstructions: instructions in the control unit of a microprocessor. This hybrid between hardware and software is called *firmware*. Firmware also includes software stored in ROM or EPROM rather than being stored on magnetic disk or tape.

Computer designers make decisions on hardware, software and firmware based on cost versus performance. Thus, today's computer market includes a wide range of systems, from high-performance supercomputers, which cost millions of dollars, to the personal microcomputer, with costs in the thousands new and in the hundreds for older used models.

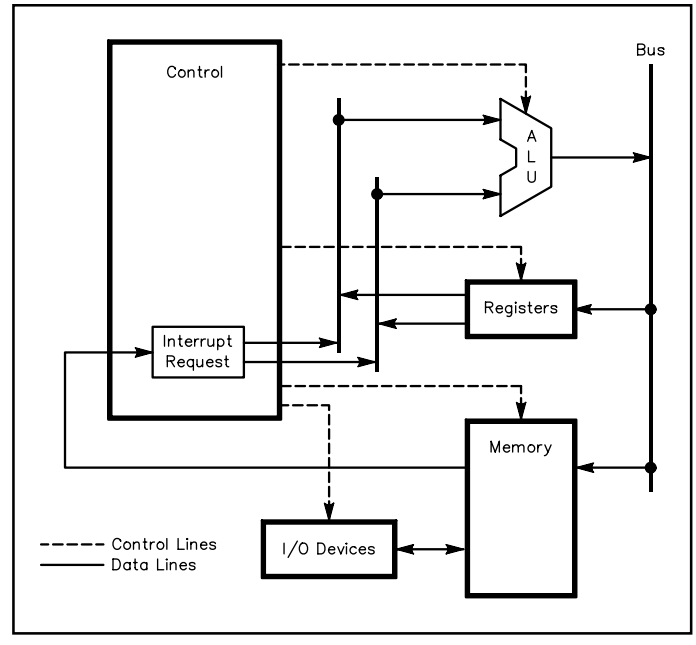


Fig 7.35 — Example of a basic computer architecture.

THE CENTRAL PROCESSING UNIT

The central processing unit is usually a single microprocessor chip, although its subsystems can be on more than one chip. The CPU at least includes a control unit, timing circuitry, an arithmetic logic unit (ALU) and also usually contains registers for temporary storage.

Control Unit

The control unit directs the operation of the computer, managing the interaction between subunits. It takes instructions from the memory and executes them, performing tasks such as accessing data in memory, calling on the ALU or performing I/O. Control is one of the most diffi-

cult parts to design; thus it is the most likely source of bugs in designing an original architecture.

Microprocessors consist of both hardwired control and microprogrammed control. In both cases, the designer determines a sequence of states through which the computer cycles, each with inputs to examine and outputs to activate other CPU subsystems (including activating itself, indicating which state to do next). For example, the sequence usually starts with "Fetch the next instruction from memory," with control outputs to activate memory for a read, a program counter to send the address to be fetched and an instruction register to receive the memory contents. Hardwired control is completely via circuitry, usually with a programmed logic array. Microprogrammed control uses a microprocessor with a modifiable control memory, containing microcode or microinstructions. An advantage of microprogrammed control is flexibility: the code can be changed without changing the hardware, making it easier to correct design errors. **Fig 7.36** shows examples of both types of control.

Timing

Usually, an oscillator controlled by a quartz crystal generates the microcomputer's clock signal. The output of this clock goes to the microprocessor and to other ICs. The clock synchronizes the microcomputer subunits. For example, each of the microinstructions is designed to take only one clock cycle to execute, so any components triggered by a microinstruction's control outputs should finish their actions by the end of the clock cycle. The exception to this is memory, which may take multiple clock cycles to finish, so the control unit repeats in its same state until memory says it's done. Since the clock rate effectively controls the rate at which instructions are executed, the clock frequency is one way to measure the speed of a computer. Clock frequency, however, cannot be the only criteria considered because the actions performed during a clock cycle vary for different designs.

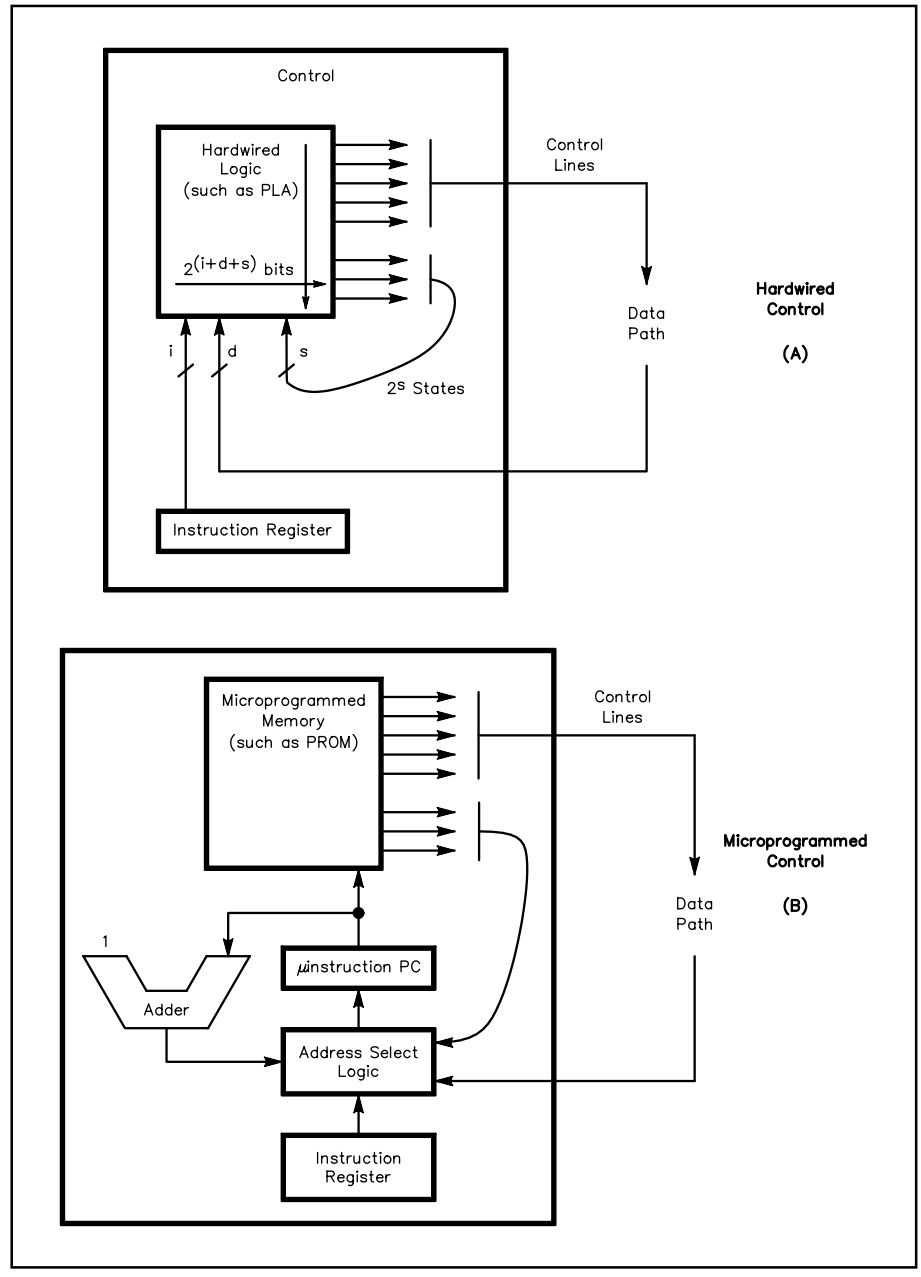


Fig 7.36 — Example arrangements of a control unit and related components: (A) hardwired control and (B) microprogrammed control.

Arithmetic Logic Unit

The *arithmetic logic unit (ALU)* performs logical opera-

tions such as AND, OR and SHIFT and arithmetic operations such as addition, subtraction, multiplication and division. The ALU depends on the control unit to tell it which operation to perform and also to trigger other devices (memory, registers and I/O) to supply its input data and to send out its results to the appropriate place.

The ALU often only performs simple operations. Complex operations, such as multiplication, division and operations involving decimal numbers, are performed by dedicated hardware, called floating-point processors, or "coprocessors." These may be included on the original motherboard or may be optional upgrades.

Registers

Microprocessor chips have some internal memory locations that are used by the control unit and ALU. Because they are inside the microprocessor IC, these registers can be accessed more quickly than other memory locations. Special purpose registers or *dedicated registers* are purely internal, have predefined uses and cannot be directly accessed by programs. *General purpose registers* hold data and addresses in use by programs and can be directly accessed, although usually only by assembly level programs.

The dedicated registers include the instruction register, program counter, effective address register and status register. The first step to execute an instruction is to fetch it from memory and put it in the *instruction register (IR)*. The *program counter (PC)* is then incremented to contain the address of the next instruction to be fetched. An instruction may change the program counter as a result of a conditional branch (if-then), loop, subroutine call or other nonlinear execution. If data from memory is needed by an instruction, the address of the data is calculated and fetched with the *effective address register (EAR)*. The *status register (SR)* keeps track of various conditions in the computer. For example, it tells the control unit when the keyboard has been typed on so the control unit knows to get input. It also notices if something goes wrong during an instruction execution, for example an attempted divide by 0, and tells the control unit to halt the program or fix the error. Certain bits in the status register are known as the *condition codes*, flags set by each instruction. These flags tell information about the result of the latest instruction — such as if the result was negative or positive or zero and if an arithmetic overflow or a carry error occurred. The flags can then be used by a conditional branch to decide if that branch should be taken or not.

Some architectures also use a *stack pointer (SP)* and/or an *accumulator*. In a stack system, a memory location is designated as the "bottom" of the stack. Data to be stored is always added to the next memory location, the "top" of the stack; and data to be accessed must always be taken off the "top." (This technique is called "last in, first out," or LIFO.) The stack pointer keeps track of the current "top" address. Stack and accumulator architectures are distinguished from an ALU design by how they handle operations. For an ALU to perform an addition, two inputs are provided from the general-purpose registers. In the stack and accumulator systems, only one input is provided with the default second input being the top of the stack or the contents of the accumulator respectively. These different approaches significantly affect the CPU design. Stack and accumulator architectures are simpler to design but less flexible, causing them to be slower. Stacks are still in use in some machines but only for temporary storage rather than arithmetic operations. The ALU and general-purpose registers are the dominant architecture today.

MEMORY

Computers and other digital circuits rely on stored information, either data to be acted upon or instructions to direct circuit actions. This information is stored in memory devices, in binary form. This section first discusses how to access an individual item in memory and then compares different memory types, which can vary how quickly and easily an item is accessed.

Accessing a Memory Item

Memory devices consist of a large number of memory cells each capable of remembering one bit of binary information. The information in memory is stored in digital form with collections of bits, called words, representing numbers and symbols. The most common symbol set is the American National Standard Code for Information Interchange (ASCII). Words in memory, just like the letters in this sentence, are stored one after the other. They are accessed by their location or address. The number of bits in each word, equal to the number of memory cells per memory location, is constant within a memory device but can vary for different devices. Common memory devices have word sizes of 8, 16 and 32 bits.

Addresses and Chip Size

An *address* is the identifier, or name, given to a particular location in memory. Since this address is expressed as a binary number, the number of unique addresses available in a particular memory chip is determined by the number of bits to express the address. For example, a memory chip with 8 bit addresses has $2^8 = 256$ memory locations. These locations are accessed as the addresses 00000000 through 11111111, 0 through 255 decimal or 00 through FF hex. (For ease of notation, programmers and circuit designers use hexadecimal, base 16, notation to avoid long strings of 1s and 0s.) The memory chip size can be expressed as M × N, where M is the number of unique addresses, or memory locations and N is the word size, or number of bits per memory location. Memory chips come in a variety of sizes and can be arranged, together with control circuitry and decoders, to meet a designer's needs.

Basic Structure

Memory chips, no matter how large or small, have several things in common. Each chip has address, data and control lines, as shown by the example chip in **Fig 7.37**. A memory chip must have enough address lines to uniquely address each of its words and as many data lines as there are bits per word. For example, the 256 × 1 memory in Fig 7.37 has 8 address lines and 1 data line.

The control lines for a memory chip can vary. Fig 7.37 shows a simple example: two control lines, a $R/\overline{W}$ and CS. In this case, data lines transfer both inputs (when writing) and outputs (when reading) so the $R/\overline{W}$ control line is needed to put the memory chip in read mode or write mode. The chip select, CS, control line tells the chip whether it is in use. When the chip is selected, it is "on," acting upon the address, data and $R/\overline{W}$ information presented to it. When the chip is not selected, the data line enters a high-impedance state so that it does not affect, and is not affected by, devices or circuits attached to it.

Reading and Writing

To write (store data in) or read (retrieve data from) a memory device, it is necessary to gain access to specific memory cells. A small 256 × 1 memory chip is used as an example. Later, this example will be expanded to a larger computer memory system.

If we want to write a 1 to the 11th word of the 256 × 1 memory (such as memory location 10 decimal or 00001010 binary), we must execute the following steps:

(1) Place the correct address (00001010) on the address lines.

(2) Place the data to be written (1) on the data line.

(3) Set the $R/\overline{W}$ control line to write (low, 0).

(4) Set the CS control line to select (high, 1). (Many memory devices use an active low chip select, $\overline{CS}$.)

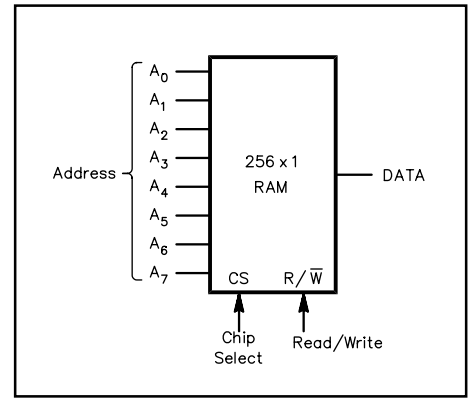


Fig 7.37 — Example of a 256 × 1 memory chip.

This writes the data on the data line (1) to the address on the address lines (00001010).

The steps to read the contents of the 11th word are similar except that the R/$\overline{W}$ control line is set to read (high, 1).

Timing

Subtle timing requirements must also be incorporated into the above steps. While writing, the address and data information must be present for a minimum setup time before and hold time after, the CS and signals have been activated. This is to avoid spurious signals spraying all over the memory array. While reading, address line changes are not harmful, but the output data is only valid a minimum access time after the last address input is stable. Manufacturers' data sheets and application notes provide the timing specifications for the particular IC you are using.

Larger Words

The one-bit-wide memory described above provides a good introduction, but usually we want a wider memory. One way to get wider memory is to use several 1-bit-wide memory chips, as shown in **Fig 7.38**. The address and control lines go to each chip, and data from each chip is used as a single bit in the large word. It is easy to see that when reading from address 0A (hex), the data lines D0 through D3 contain the data from address 0A of chips U0 through U3.

An address placed on the shared address lines (called an address bus) now specifies an entire word of data. Notice that one line of the address bus connects to the CS pin of each memory chip. This line is labeled ME, or memory enable, and sets all four ICs to read or write data at the same time.

If all four memory chips were put in a single package, they would make a 256 × 4 IC. This IC would look like the chip in Fig 7.37, except that it would have 4 data lines.

More Address Space

For even larger memory systems, the same principles as shown in Fig 7.38 can be applied. **Fig 7.39** shows a 1024 × 8 (1 kilobyte) memory built from four 256 × 8 memory chips. A kilobyte, or kbyte is usually abbreviated as K. Notice this is not quite the same as the metric prefix kilo, because it represents 1024, rather than 1000.

Ten address lines are needed to address 1024 locations ($2^{10} = 1024$). Eight of the 10 address lines, A0 to A7, are used as a normal address bus for chips 0 through 3. The remaining 2 address lines, A8 and A9, are run through a 2-to-4 line decoder to choose between the 4 memory chips. When employed in this manner, the 2-to-4 line decoder is called an address decoder.

To assert the CS input for one of the memory chips, ME must be 1 and the correct output of the 2-to-4 line decoder must also be 1. When an address is placed on A0 through A9, a single memory chip is selected by ME, A8 and A9. The other 8 address lines address a single word from that chip. The three chips that are

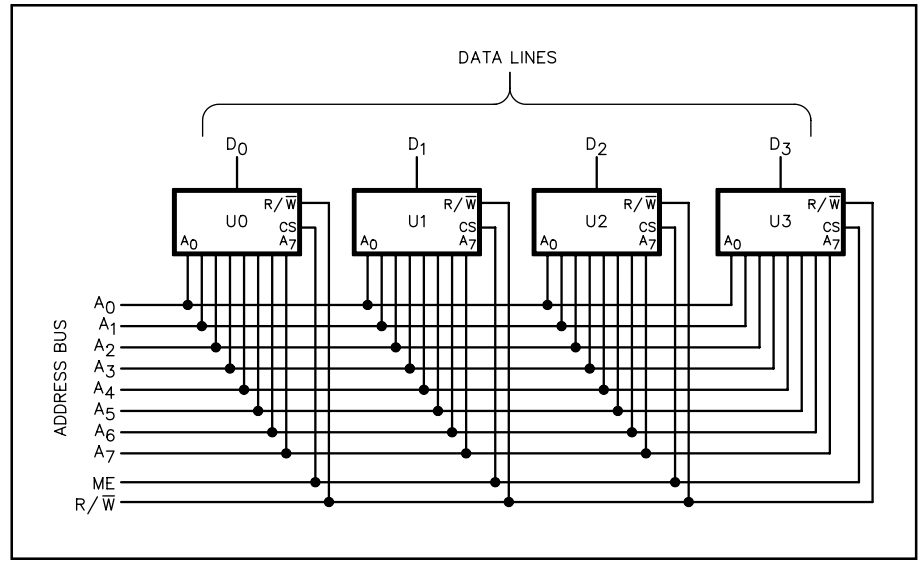


Fig 7.38 — A 256 × 4 memory built with four 256 × 1 memory chips.

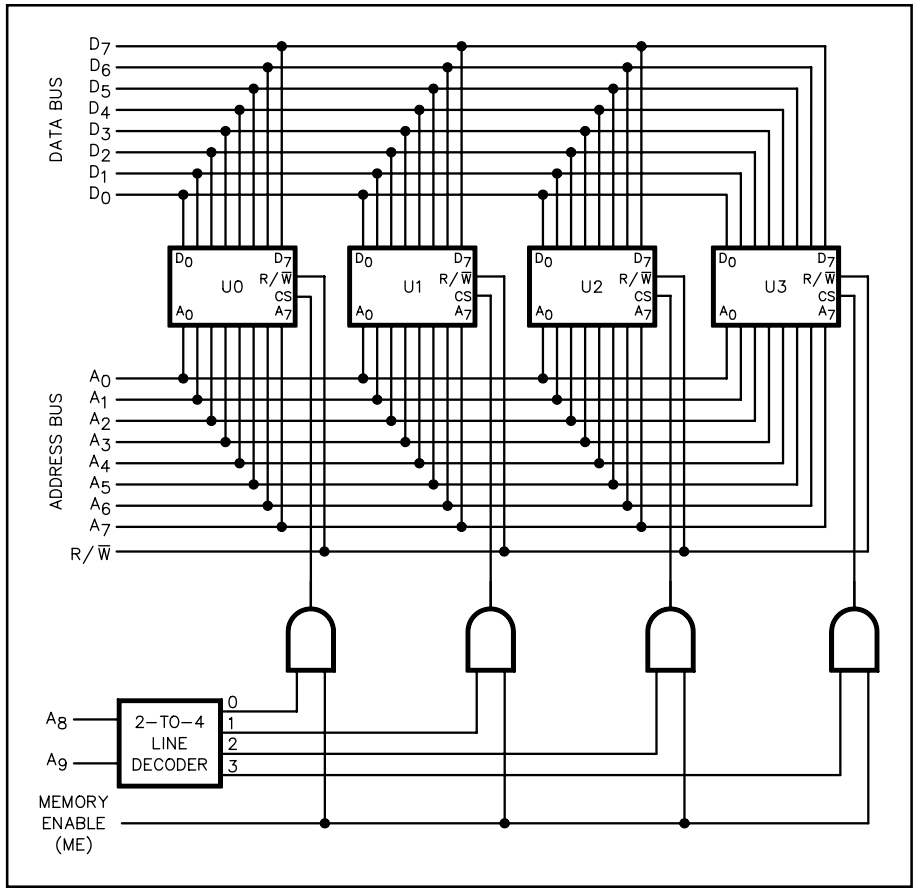


Fig 7.39 — A 1024 × 8 memory built with four 256 × 8 memory chips and appropriate control circuitry.

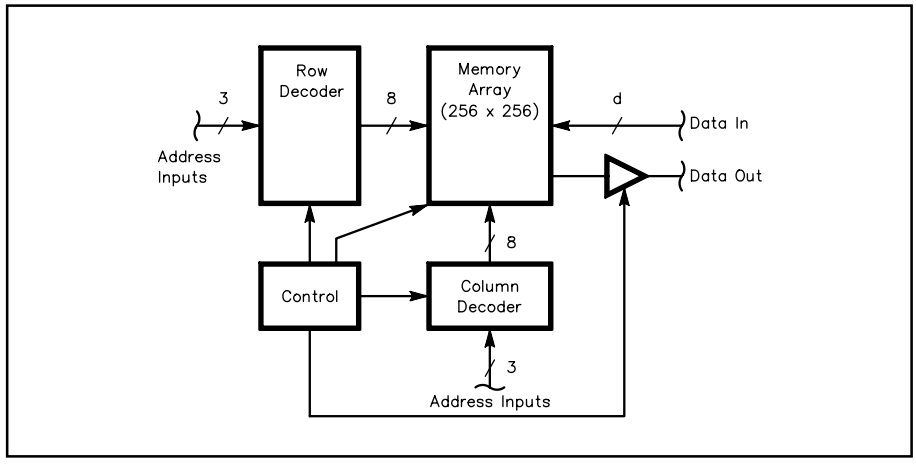


Fig 7.40 — Row and column decoders allow a memory array to be accessed in a variety of formats.

not selected enter a high-impedance state and do not affect the data lines. This example shows that, using the proper memory chips and address decoding, any size memory with any word length can be built.

Alternate Structures

Fig 7.40 shows how the same chip can be accessed in different ways by using two decoders, a row decoder and a column decoder. The same 256 × 256 memory array can be treated as a 64 K × 1 array, a 256 × 256 array or other possibilities. In fact, most larger memory chips are made as square arrays: 32 × 32 (1024 bytes or 1 K), 64 × 64 (4096 bytes or 4 K), 256 × 256 (65536 bytes or 64 K), 1024 × 1024 (1 M), 2048 × 2048 (4 M), 4096 × 4096 (16 M) and so on. (Here the M represents a megabyte, which is 1048576 bytes.) The square array makes the chips more cost effective to manufacture (easier quality control and less waste) and easier to incorporate into a printed-circuit-board circuit layout. Notice that each M × N is a power of 2. So while we refer to the chips by shorter names like 1 Mbyte, the actual number of memory cells is larger than 1000000. The product, M × N, only refers to the number of memory cells in the chip; and designers are free to choose the word size appropriate to their needs. In fact, they may access one location as an 8-bit word and another as a 16-bit word. For example, a computer with a Motorola MC68000 microprocessor automatically accesses a character, such as "A," as 8 bits (a byte), an integer as 16 bits (a word) and a real value as 32 bits (a longword). (Apple Macintosh computers use the MC68000 microprocessor.) To further complicate things, a different manufacturer may call 16 bits a halfword and 32 bits a word. In using memory, the controller chips and circuitry to access the memory can be just as important as the memory itself.

Memory Types

The concepts described above are applied to several types of random-access, semiconductor memory. Semiconductor memories are categorized by the ease and speed with which they can be accessed and their ability to "remember" in the absence of power.

SAM versus RAM

One way to categorize memory is by what memory cells can be accessed at a given instant. *Sequential-access memory (SAM)* must be accessed by stepping past each memory location until the desired location is reached. Magnetic tapes implement SAM; to reach information in the middle of the tape, the tape head must pass over all of the information on the beginning of the tape. Two special types of SAM are the queue and the push-down stack. In a *queue*, also called a first-in, first-out (FIFO) memory, locations must be read in the order that they were written. The queue is a "first-come, first-served" device, like a line at a ticket window. The push-down stack is also called last-in, first-out (LIFO) memory. In LIFO memory, the location written most recently is the next location read. LIFO can be visualized as a stack, always adding to and removing from the "top" of the stack. *Random-access memory (RAM)* allows any memory cell to be accessed at any instant, with no time wasted stepping past the "beginning" parts of the data. Random-access memory is like a bookcase; any book can be pulled out at any time.

It is usually faster to access a desired word in RAM than in SAM. Also, all words in RAM have the same access time, while each word in a SAM has a different access time based on its position. Generally, the semiconductor memory devices internal to computers are random-access memories. Magnetic devices, such as tapes and disks, have at least some sequential access characteristics. We will leave tapes and disks for a later section and concentrate here on random-access, solid-state memories.

Random Access Memory

Most RAM chips are *volatile*, meaning that stored information is lost if power is removed. RAM is either static or dynamic. *Dynamic RAM (DRAM)* stores a bit of information as the presence or absence of charge. This charge, since it is stored in a capacitor, slowly leaks away. It must be refreshed periodically. Memory refresh typically occurs every few milliseconds and is usually performed by a dynamic RAM controller chip. *Static RAM (SRAM)* stores a bit of information in a flip-flop. Since the bit will retain its value until either power is removed or another bit replaces it, refresh is not necessary.

Both types of RAM have their advantages and disadvantages. The advantage of DRAM is increased density and ease of manufacture, making them significantly less expensive. SRAMs, however, have much faster access times. Most general purpose computers use DRAMs, since large memory size and low cost are the major objectives. Where the amount of memory required doesn't justify the use of DRAM, and the faster access time is important, SRAMs are common, for example, in embedded systems (telephones, toasters) and for cache memories. Both types of RAM are available in MOS families; SRAMs are also available in bipolar. Generally, MOS RAMs have lower power consumption than bipolar RAMs, while access speeds vary widely. Cost, power consumption and access time, provided in manufacturers' data sheets, are factors to consider in selecting the best RAM for a given application.

Read-Only Memory

Read-only memory (ROM) is nonvolatile; its contents are not lost when power is removed from the memory. Despite its name, all ROMs can be written or programmed at least once. The earliest ROM designs were "written" by clipping a diode between the memory bit and power supply wherever a 0 was desired. Modern MOS ROMs use a transistor instead of a diode. Mask ROMs are programmed by having ones and zeros etched into their semiconductors at manufacturing time, according to a pattern of connections and nonconnections provided in a mask. Since the "programming" of a mask ROM must be done

by the manufacturer, adding expense and time delays, this type of ROM is primarily used only in high volume applications.

For low-volume applications, the programmable ROM (PROM) is the most effective choice since the data can be written after manufacture. A PROM is manufactured with all its diodes or transistors connected. A PROM programmer device then "burns away" undesired connections. This type of PROM can be written only once.

Two types of PROMs that can be "erased" and reprogrammed are EPROMs and EEPROMs. The transistors in UV erasable PROMs (EPROMs) have a floating gate surrounded by an insulating material. When programming with a bit value, a high voltage creates a negative charge on the floating gate. Exposure to ultraviolet light erases the negative charge. Similarly, electrically erasable PROMs (EEPROMs) erase their floating-gate values by applying a voltage of the opposite polarity. **Table 7.10** summarizes these ROM characteristics.

Besides being nonvolatile, PROMs are also distinguished from RAMs by their read and write times. Naturally, since PROMs are only written to infrequently, they can have slow write times (in the millisecond range). Their read times, however, are near those of RAM (in the nanoseconds). Read and write times for RAMs are nearly equal, both in the nanosecond range. Two other factors make it hard to write to PROMs: (1) PROMs must be erased before they can be reprogrammed and (2) PROMs often require a programming voltage higher than their operating voltage.

ROMs are practical only for storing data or programs that do not change frequently and must survive when power is removed from the memory. The programs that start up a computer when it is first switched on or the memory that holds the call sign in a repeater IDer are prime candidates for ROM.

Nonvolatile RAM

For some situations, the ideal memory would be as nonvolatile as ROM but as easy to write to as RAM. The primary example is data that must not be allowed to perish despite a power failure. Low-power RAMs can be used in such applications if they are supplied with NiCd or lithium cells for backup power. A more elegant and durable solution is nonvolatile RAM (NVRAM), which includes both RAM and ROM. The standard volatile RAM, called shadow RAM, is backed up by nonvolatile EEPROM. When the RECALL control is asserted, such as when power is first applied, the contents of the ROM are copied into the RAM. During normal operation, the system reads and writes to the RAM. When the STORE control is triggered, such as by a power failure or before turning off the system, the entire contents of the RAM are copied into the ROM for nonvolatile storage. In the event of primary power failure, to successfully save the RAM data, some power must be maintained until the memory store is complete (+5 V for 20 ms).

Table 7.10
ROM Characteristics

Type	*Technology*	*Read Time*	*Write Time*	*Comments*
Mask ROM	NMOS, CMOS	25 - 500 ns	≈ 4 weeks	Write once; low power
Mask ROM	Bipolar	< 100 ns	≈ 4 weeks	Write once; high power; low density
PROM	Bipolar	< 100 ns	≈ 5 minutes	Write once; high power; no mask required
EPROM	NMOS, CMOS	25 - 500 ns	≈ 5 minutes	Reusable; low power; no mask required
EEPROM	NMOS	50 - 500 ns	10 ms / byte	10000 write cycles per location limit

Cache versus Main Memory

Memory is in high demand for many applications. To balance the trade-off of speed versus cost, most computers use a larger, slower, but cheaper main memory in conjunction with a smaller, faster, but more expensive cache memory. As you run a computer program, it accesses memory frequently. When it needs an item, a piece of data or the next part of the program to execute, it first looks in the cache. If the item is not found in the cache, it is copied to the cache from the main memory. As you run a computer program, it often repeats certain parts of the program and repeatedly uses pieces of data. Since this information has been copied to the high-speed cache, your computer game or other application can run faster. Information used less often or not being used at all (programs not currently being run) can stay in the slower main memory.

A "cache" is a place to store treasure; the treasure, the information you are using frequently, can be accessed quickly because it is in the high-speed cache. The use of cache versus main memory is managed by a computer's CPU so it is transparent to the user. The improvement in program execution time is similar to accessing a floppy disk versus the computer's internal memory.

I/O TRANSFERS

Input and output allow the computer to react to and affect the outside world. The ability to interact with their environment is a primary reason why computers are so useful and cost-effective. Usually, I/O is provided by a user, and a great deal of effort goes towards making computers user-friendly. Alongside the drive for user-friendly computers is the drive for automation. Data is acquired and operations are performed automatically, such as the packet bulletin board automatically forwarding a message. This section discusses the relationship of I/O to the internal operation of the computer: how the computer knows when and what I/O has been provided. The next section, on peripherals, discusses the range of devices that provide this information.

Program-Controlled I/O

Program-controlled I/O might be understood by thinking of a cook who returns to the oven every few minutes to see if a meal is ready. Under program-controlled I/O, or polling, input and output events are initiated by the program currently running on the microcomputer. The program polls the I/O device, constantly checking if it is ready to accept or deliver data. When the I/O device indicates that it is ready, then the instruction that actually sends or receives the data is executed.

An advantage of program-controlled I/O is its simplicity. Program-controlled I/O is easily written and debugged. A disadvantage is wasted time. The program must spend its time checking the status of the I/O device rather than doing other useful things. If the program must have the input data before continuing, then no time is wasted; but if it could have been performing other tasks, then polling can be expensive and wasteful. Packet radio provides a familiar example of polling: the TNC repeatedly sends a packet until a confirmation message has been received from the BBS.

Interrupt-Driven I/O

Interrupt-driven I/O avoids wasting time in a polling loop. The cook, rather than constantly checking the oven, goes off to other work until the timer rings. This efficiency is especially important on multiuser systems, where one program may be waiting for I/O while another program is executing.

The ring of the timer is called an *interrupt*, a temporary break in the normal execution of a program. The act of taking the food out of the oven is coded in an *interrupt service routine*. An interrupt service routine (ISR) is any code that performs the appropriate actions in response to a certain interrupt. Each interrupt has a number, and the location of each ISR is listed in a table next to its number. From the machine's perspective, the process is as follows: One of the bits in the microprocessor's status register

is called the interrupt request indicator. When this bit becomes a 1, an interrupt has occurred. Circuitry indicates the number of the device requesting the interrupt. The machine temporarily suspends whatever it was working on and looks at its table of service routines. From the table, the machine finds the location of the appropriate ISR and automatically jumps to that code and begins executing it. When finished with the ISR, the machine automatically returns to whatever it was doing before the interrupt.

The "getchar" subroutine below shows how an interrupt service routine for keyboard input might look in assembly language.

```
getchar:
MOVE RCVDATA,R7; Move the data from the receiver to the temporary storage register
RTE ; Return to normal execution
```

There are two key differences from the previous example: (1) No polling loop is involved. The READY bit, instead of being polled, triggers the interrupt request bit. (2) Leaving the main program and returning to it are done automatically by the machine instead of with a subroutine call inside the code.

The advantage of interrupt-driven I/O is that no time is wasted in a polling loop. This is especially advantageous in a multiuser environment where processing time must be juggled between the user demands. The disadvantage of interrupts is that program flow can become very confusing; for example, what happens if an interrupt service routine gets interrupted? This is usually handled by assigning priorities to each possible interrupt and, when inside an ISR, ignoring other interrupts of lesser or equal priority.

A familiar example of an interrupt is when you want to crash out of a program; for example, the CTRL - ALT - DEL key combination on an IBM, the reset button on a Macintosh, or the CTRL C or CTRL Y combination on a UNIX machine. Examples of a timer interrupt are the sending every ten minutes of a repeater's ID; the automatic save of some word processors; and, when doing packet, the BBS automatically kicks you out for too many minutes of inactivity.

Memory-Mapped I/O

In *memory-mapped I/O*, addresses that are treated like RAM by the microprocessor are actually I/O devices. Thus, a command that would usually be used to read or write to a memory location might actually result in an I/O operation. Since memory mapping is an addressing technique, it can be used with either interrupt-driven or polled I/O.

Direct Memory Access

Direct Memory Access (DMA) enables data to be transferred directly between memory and an I/O device without involving the CPU. The advantages of DMA are to provide high-speed transfer of data, such as from a peripheral disk drive or communications device, while the CPU is performing internal tasks. The data transfer operation is managed by a DMA controller, either a separate chip or internal to the microprocessors. The following illustrates some of the steps involved in the I/O transfer:

An I/O device requests DMA operation.

The DMA controller requests the bus from the CPU.

The CPU acknowledges the request and releases the bus.

The DMA controller tells the I/O device to send its information.

The DMA technique is used by I/O processors. In large computer systems, these auxiliary processors perform most of the I/O functions, thus freeing the CPU for other tasks.

PERIPHERALS

Peripherals are any devices outside the CPU. They provide additional capabilities. One of the most

common examples being communication with a user via input devices and output devices. Input devices provide the computer both data to work on and programs to tell it what to do. Output devices present the results of computer operations to the user or another system and may even control an external system. Both input and output combine to provide user friendly interaction. This section discusses the most common user interfaces.

Most of these devices have adapted to certain standards and use readily available connection cables. Thus, they can be easily incorporated into a system, and a knowledge of the internal actions is not necessary. A knowledge of how external memory devices work is more useful and will be discussed in more detail.

Input Devices

The keyboard is probably the most familiar input device. A keyboard simply makes and breaks electrical contacts. The open or closed contacts are usually sensed by a microprocessor built into the circuit board under the keys. This microprocessor decodes the key closures and sends the appropriate ASCII code to the main computer unit. Keyboards will generate the entire 128 character ASCII set and often, with CONTROL and ALT (Alternate) keys, the 256 character extended ASCII set.

The mouse is becoming increasingly popular for use with graphical user interfaces. The mouse casing holds a ball and circuitry to act as a multidirectional detection device. By moving the mouse, the ball rolls, controlling the relative position of a cursor on the screen. Buttons on the mouse make and break connections (clicking) to select and activate items (icons) on the screen. The trackball is a variation of the mouse.

Other input devices include modems and magnetic disks and tapes. Magnetic disks and tapes, discussed at length later, provide additional external memory. Newer input devices include voice activated devices, touch screens and scanners.

Output Devices

The most familiar output device is the computer screen, or monitor. For smaller character displays, LED arrays can be used. The next most common output device is the printer, to produce paper hardcopy. Modems and magnetic disks and tapes are output devices as well as input devices. Newer output devices include speech synthesizers.

The output devices (except the sound device) share a common display technique: images, such as characters and graphics, are formed by tiny dots, called *pixels* (picture elements). On screens, these are dots of light turned on and off. In printers, they are dots of ink imposed onto the paper. For color displays, pixels in red, green and blue (RGB) are spaced closely together and appear as numerous colors to the human eye. **Fig 7.41A** and B show two examples of how characters can be formed using an array of dots. Part C shows how a series of 14 bars can be arranged to form a character display.

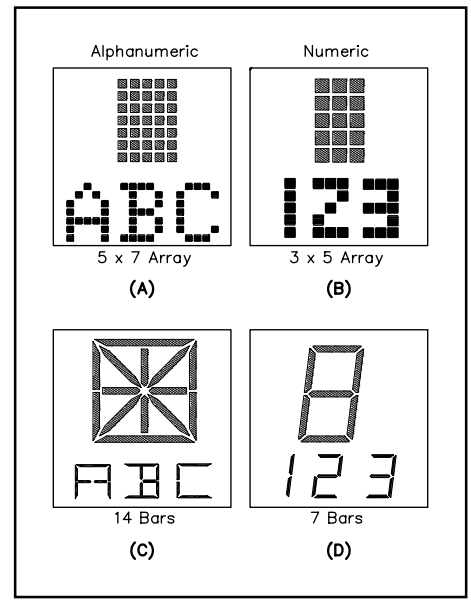


Fig 7.41 — Various character display formats.

Video Displays

Video monitors are usually specialized cathode-ray tube (CRT) displays. In newer notebook computers, the monitors are being fabricated with monochrome or color liquid-crystal displays (LCDs), but color LCD displays are still quite expensive. A standard TV set can even be used as a computer monitor. Two techniques are used to turn on the screen pixels. *Raster scanning* cov-

ers the screen by writing one row of pixels at a time, from left to right and top to bottom. Then, a vertical retrace brings the beam back to the top of the screen to begin again. Raster scanning signals every pixel on or off for each screen pass. An alternative, *vector mode*, only signals the pixels where something on the screen has changed.

Light-emitting diodes (LEDs) are handy when a full screen display is not necessary. A single diode easily indicates on/off states such as the power light on many devices. A popular single character display is the seven segment version, shown in Fig 7.41D, which is good for displaying numbers.

Various seven-segment decoders are available to drive common-cathode and common-anode seven-segment displays. These drivers receive a number, usually in BCD format, and decode the number into signal levels to activate the proper a-g segments of the display. **Fig 7.42** shows one example of a seven segment decoder and display. Part B shows a TTL 7447 IC and a common-anode LED display. The TTL 7448 is designed to drive common-cathode displays. The dc illumination method shown is the easiest to implement; but higher light output with lower energy consumption can be obtained by pulsing and multiplexing the display voltage. A pulse rate of 100 Hz is imperceptible to the human eye.

As more digits are added, using a separate decoder/driver for each display becomes unfeasible from economic and PC-board points-of-view. **Fig 7.43** shows how a single decoder can be set up to drive several displays using a multiplexing setup. The multiplexer logic sends input data for a digit into the decoder and enables the common element of the correct display, ensuring that a display will be energized only when both the segment and its common lead are selected. With this system, only one digit is energized at any instant, a factor that greatly reduces power-supply requirements. To maintain the brightness of each digit, the current to each display segment must be increased. When implementing a mulitplexed display, be sure not to exceed the peak and average current specifications for the display.

Disk Drives

Magnetic media are essential input/output devices since they provide additional memory. The earliest ways to store programs and data were on punched cards and tape. Some early home computers used audio cassettes.

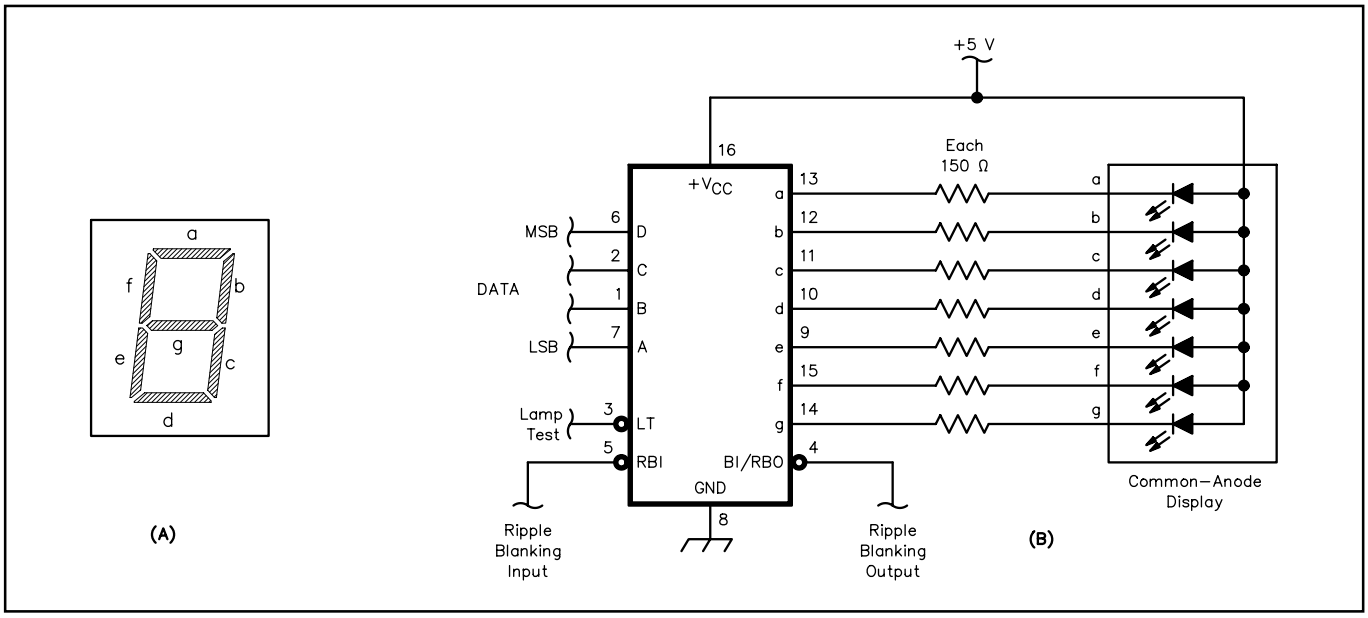


Fig 7.42 — (A)The segments and their arrangement in a seven-segment display. (B) Shows how a 7447 decoder/driver IC converts BCD data into the appropriate driving signals for a common-anode seven-segment LED display. A 7448 IC will drive a common-cathode LED display.

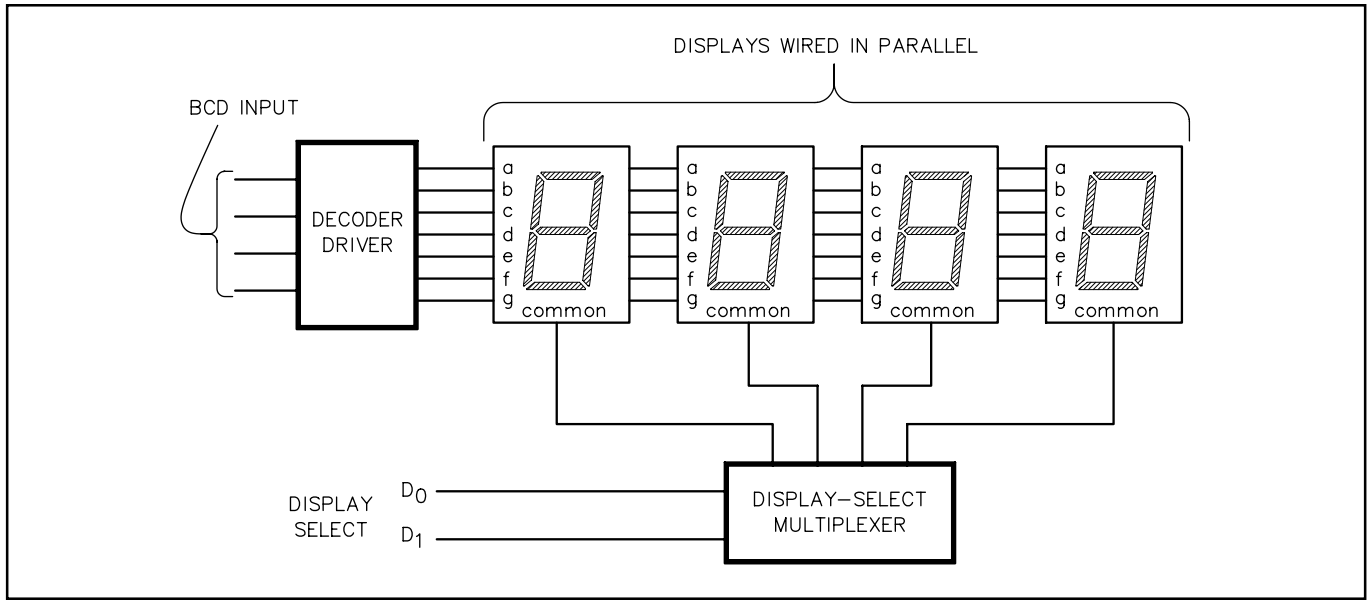


Fig 7.43 — Multiplexed character displays. The digits are wired in parallel, or with all of the "a" segments connected together and so on. BCD data for a particular display is placed at the decoder input and the desired display is selected by its address.

Disk storage is prevalent when random access is needed. In some ways, disk storage is similar to that of a record player. The data is stored in circles (tracks) on a round platter (disk or diskette) and accessed by a device (a head) moving over the platter. Unlike the record player, the tracks are concentric rather than spiral and the head can write as well as read.

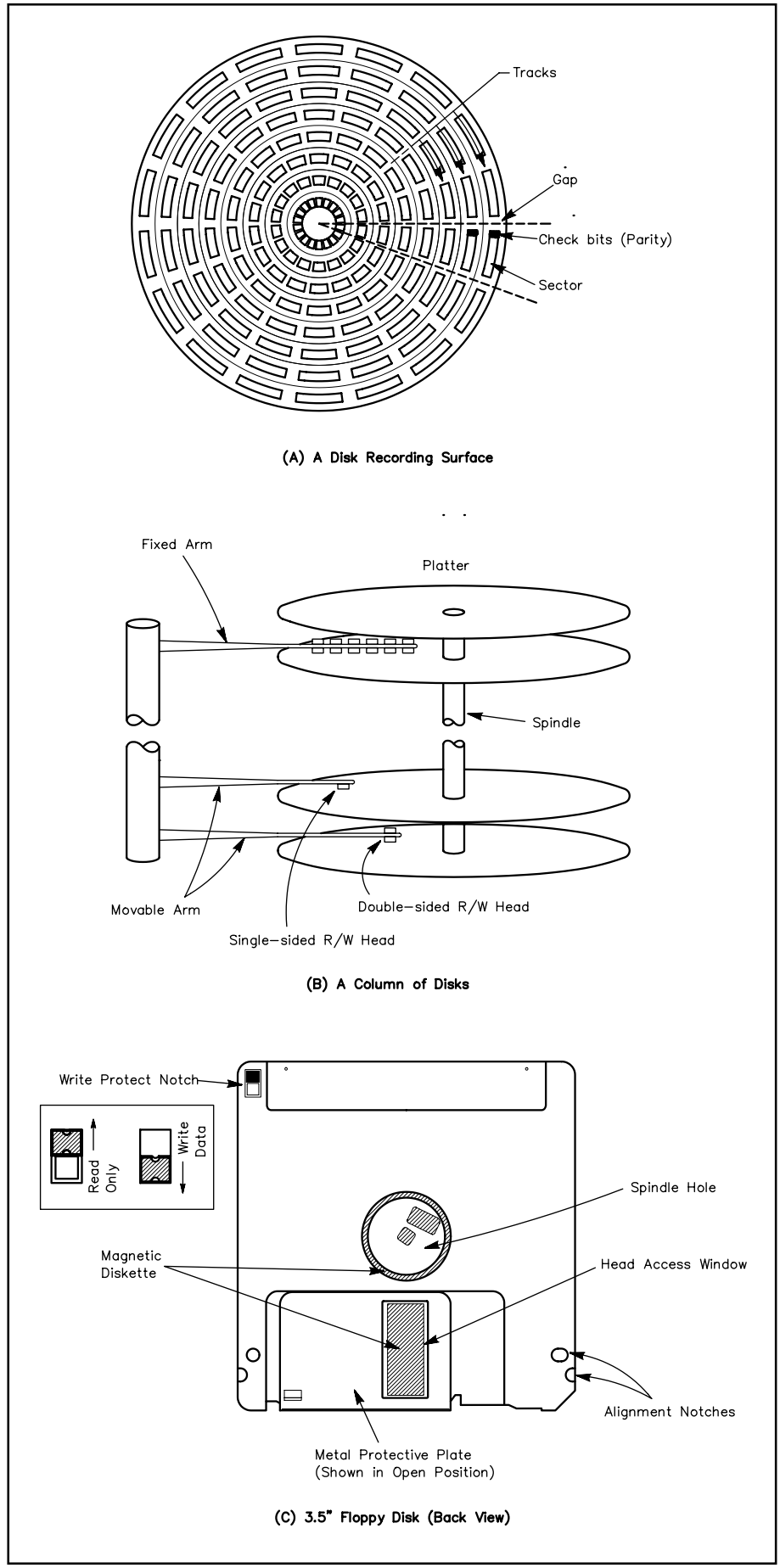
Fig 7.44A shows an example of the disk recording surface. Usually the tracks are divided into equal-sized storage units, called sectors. Also, since the disk has two sides, most disks can store information on both sides. Therefore, locating a piece of information in disk memory means identifying three coordinates: the side, track and sector.

Accessing a piece of information on the disk system involves a number of wait times until the data access is complete. First, disks may be either movable-head or fixed-head, as shown in Fig 7.44B. In the movable system, a single read/write head is attached to a movable arm, so there is a seek time for the movable arm to position the read/write head on the appropriate track. In a fixed system, each track has its own read/write head, so seek time is zero since the head is immediately in position. Second, the data must rotate into position under the read/write head. This time is called latency. Finally, there is the normal time for the read/write to occur.

A number of types of disk technology are available. In hard disk systems, the disk is rigid and the read/write head does not contact the disk directly. The absence of friction between the head and disk allows finer head positioning and higher disk speeds. Thus, hard disks hold more data and are accessed more quickly than floppy disks. Floppy disks enclose the magnetic-media platter in a casing, as shown in Fig 7.44C, so the disk can be carried around. The floppy disk can be inserted into a disk drive and the read/write head automatically extended; when done, the read/write head is automatically retracted before the disk is ejected from the drive. Variations in floppy disks include single-sided (SS) or double-sided (DS); single, double or high density; and $3^1/_2$ or $5^1/_4$ inches. The density refers to the disk format used by the disk controller. High data density allows more data to be written to the disk but requires a higher quality diskette. Not all disks can be written as high density and not all disk drives can read high density disks.

Dust and dirt on the disk and the imperfections in the disk surface gradually damage both the disk and

Fig 7.44 — Disk storage: (A) a disk recording surface, (B) a column of disks illustrating fixed versus movable head and single versus double sided, (C) a floppy disk.



the head. This means that disks eventually wear out, and the data on the disk will probably be lost. Therefore, it is prudent to make backup copies of your disks, stored in a clean, dry, cool place.

Tape

Tape is one of the more inexpensive options for auxiliary memory. Tape access time is slow, since the data must be accessed sequentially, so tape is primarily used for backup copies of a system's memory. Tape is available in cassette form (common sizes are comparable to the cassettes for a portable tape player and VCR tapes) and on reels (diameter is approximately one foot).

Digital audio tape (DAT) is replacing other forms of tape backup system in newer computer systems. A single 4-mm-wide DAT cartridge, which fits in the palm of your hand, can hold over 2 gigabytes (GB) of data (1 GB = 1000 MB).

INTERNAL AND EXTERNAL INTERFACING

Designing an interface, or simply using an existing interface, to connect two devices involves a number of issues. For example, digital interfacing can be categorized as parallel or serial, internal or external and asynchronous or synchronous. Additional issues are the data rate, error detection methods and the signaling format or standards. The format can be especially important since many standards and conventions have developed that should be taken into consideration. This chapter focuses on some basic concepts of digital communications for interfacing between devices.

Parallel Versus Serial Signaling

To communicate a word to you 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 flash-cards only 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. **Fig 7.45** gives a graphic illustration of 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 on internal communications. For longer distance communications, such as to an external device, serial signaling is more appropriate. Each bit is sent in turn, so communication is slower; but it is also 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 parallel or serial signaling also depends on the operational mode: one channel 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 for full-duplex (simultaneous communications in both directions).

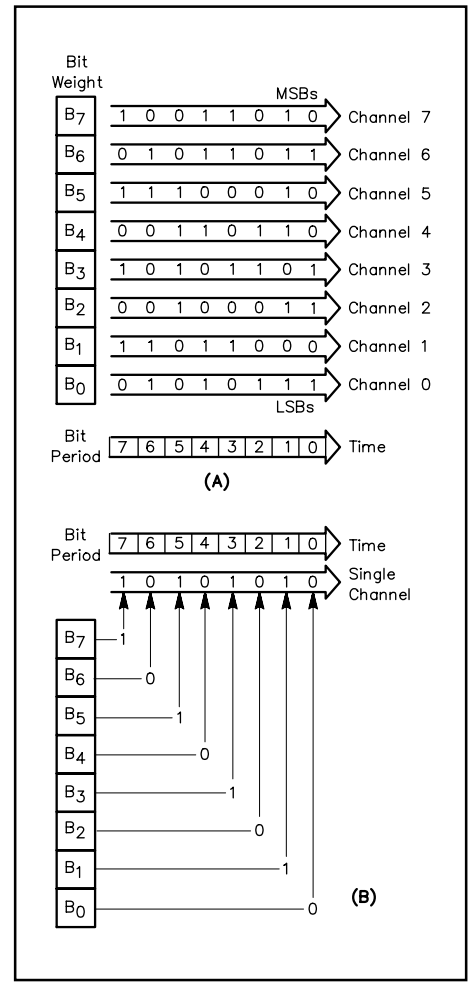


Fig 7.45 —Parallel (A) and serial (B) signaling. Parallel signaling in this example uses 8 channels and is capable of transferring 8 bits per bit period. Serial transfer only uses 1 channel and can send only 1 bit per bit period.

Parallel I/O Interfacing

Fig 7.46 shows an example of a parallel input/output chip. 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 is available on all 8 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 watch to ensure the transmitter is ready for the next character.

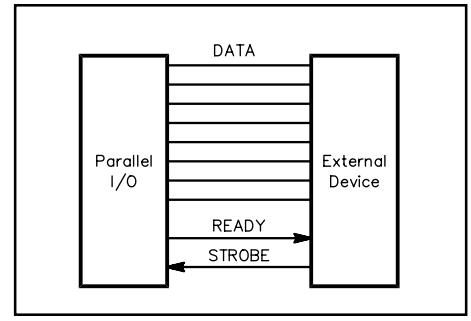


Fig 7.46 — Parallel interface with READY and STROBE handshaking lines.

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

Conversions

Within computers and other digital circuits, data is usually operated on, stored and transmitted in parallel. For communicating with an external device, data must usually be converted from parallel to serial format and vice versa. This conversion is usually handled by shift registers.

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.

A register with parallel input and shift left serial output will be described, as was shown in Fig 7.24. (A serial input/parallel output register would work in the opposite fashion.) Since the register receives information in parallel, the n-bit register has n inputs, one to each flip-flop. A parallel load control input is asserted to pass the initial value. The register sends out information in serial fashion so there is only one output line. Since this example shifts left, the output comes from the left-most register, the most significant bit. On each clock pulse, one bit is output and the other flip-flops cycle their value up to the next flip-flop. A 0 is usually input to the least-significant bit so 0s will cycle up to fill the register. After n clock pulses, all data bits have been shifted out and the register has a value of 0.

Asynchronous versus Synchronous Communication

To correctly receive data, the receiving interface must know when data bits will occur; it must be synchronized with the sender. In *asynchronous* communication, the receiver synchronizes on each incoming character. Each character includes start and stop bits to indicate the beginning and end of that character. In *synchronous* communication, data is sent in long blocks, without start and stop bits or gaps between characters.

Asynchronous Communication

In asynchronous communication, each transmitted character begins with a start bit and ends with a stop bit, as shown in **Fig 7.47A**. The start bit (usually a zero) tells the receiver to begin receiving a character. The stop bit (usually a one) signals the end of a character. Between characters, the transmitting circuit sends the stop bit state (steady one or zero).

Since the receiver is always told when a character begins and ends, characters can be sent at irregular intervals. This is especially advantageous for typed input, since the person typing is usually slower than

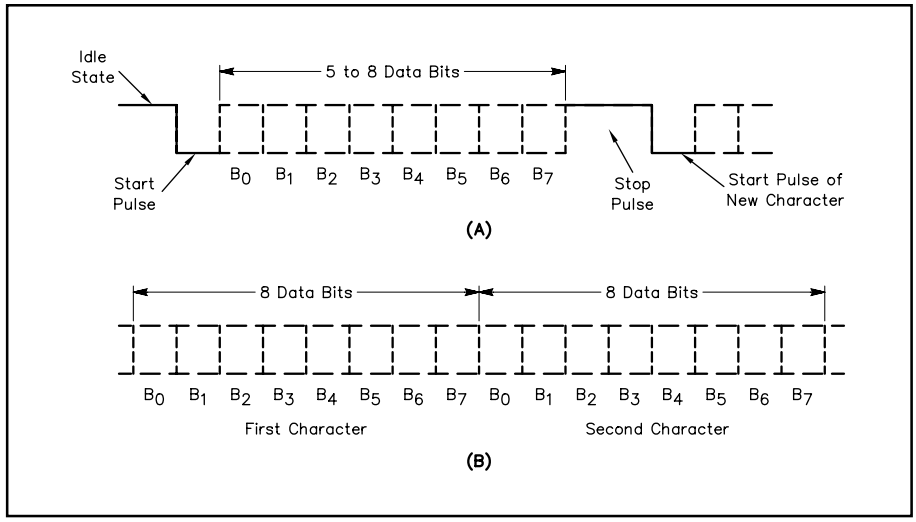


Fig 7.47 — Serial data transmission format. In asynchronous signaling at A, a start pulse of one bit period is followed by the data bits and a stop pulse of at least one bit period. In synchronous signaling at B, the data bits are sent continuously without start or stop pulses.

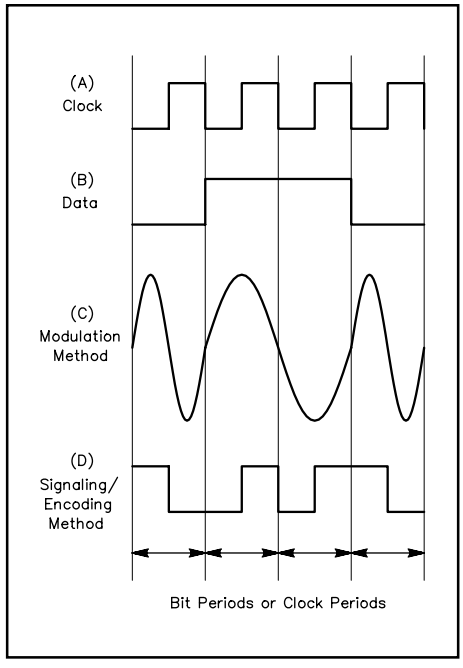


Fig 7.48 — Recovering the clock (A) when the data (B) is transmitted allows a receiver to maintain synchronization during synchronous communication. The modulation method shown at C results in a transition of the received carrier at the beginning and end of each clock period. The encoding method shown at D results in a data transition in the middle of each clock period. Either of these methods provides enough information for clock recovery.

the data communications equipment and will usually work at an uneven pace. Another advantage of asynchronous data is that it does not need complex circuits to keep it synchronized. Since the receiver is newly synchronized at the beginning of each character, the characters need not be sent in a steady stream and no stringent demands are made on the person or process generating the characters.

A disadvantage of asynchronous communications is the inclusion of the start and stop bits, which are not "useful" data. If you are transmitting 8 data bits, 1 start bit and 1 stop bit, then 20% (2 of 10 bits) is overhead.

Synchronous Communication

In synchronous communications, data is sent in blocks, usually longer than a single character, as shown in Fig 7.47B. At the beginning of each block, the sender transmits a special sequence of bits that the receiver uses for initial synchronization. After becoming synchronized at the beginning of a block, the receiver must stay synchronized throughout the block. The sender and receiver may be using slightly different clock frequencies, so it is usually not adequate for them to merely be synchronized initially. There are several ways for the receiver to stay synchronized. The transmitter may send the clock signal on a separate channel, but this is wasteful. The modulation technique used on the communications channel may convey clock information or the clock may be implicit within the data. See **Fig 7.48**.

One disadvantage of synchronous transmission is that the data must be sent as a continuous stream; characters must be placed in a buffer until there are enough to make a block. Also, while errors in asynchronous signaling usually only affect one character (the receiver can resynchronize at the beginning of the next character), error recovery on synchronous channels may be a longer process involving several lost characters or an entire lost block.

The major advantage of synchronous signaling is that it does not impose overhead (the start and stop bits) on each character. This is an important consideration during large data transfers.

Data Rate

There are a number of limitations on how fast data can be transferred: (1) The sending equipment has an upper limit on how fast it can produce a continuous stream of data. (2) The receiving equipment has an upper limit on how fast it can accept and process data. (3) The signaling channel itself has a speed limit, often based on how fast data can be sent without errors. (4) Finally, standards and the need for compatibility with other equipment may have a strong influence on the data rate.

Two ways to express data transmission rates are *baud* and *bits per second (bps)*. These two terms are not interchangeable: Baud describes the signaling, or symbol, rate — a measure of how fast individual signal elements **could** be transmitted through a communications system. Specifically, the baud is defined as the reciprocal of the shortest element (in seconds) in the data encoding scheme. For example, in a system where the shortest element is 1 ms long, the maximum signaling rate would be 1000 elements per second. (Note that, since baud is measured in elements per second, the term "baud rate" is incorrect since baud is already a measure of speed, or rate.) Continuous transmission is not required, because signaling speed is based only on the shortest signaling element.

Signaling rate in baud says nothing about actual information transfer rate. The maximum information transfer rate is defined as the number of equivalent binary digits transferred per second; this is measured in bits per second.

When binary data encoding is employed, each signaling element represents one bit. Complications arise when more sophisticated data encoding schemes are used. In a quadriphase shift keying (QPSK) system, a phase transition of 90° represents a level shift. There are four possible states in a QPSK system; thus, two binary digits are required to represent the four possible states. If 1000 elements per second are transmitted in a quadriphase system where each element is represented by two bits, then the actual information rate is 2000 bps.

This scheme can be extended. It is possible to transmit three bits at a time using eight different phase angles (bps = 3 × baud). In addition, each angle can have more than one amplitude. A standard 9600 bps modem uses 12 phase angles, 4 of which have two amplitude values. This yields 16 distinct states, each represented by four binary digits. Using this technique, the information transfer rate is four times the signaling speed. This is what makes it possible to transfer data over a phone line at a rate that produces an unacceptable bandwidth using simpler binary encoding. This also makes it possible to transfer data at 2400 bps on 10 m, where FCC regulations allow only 1200 baud signals.

When are transmission speed in bauds and information rate in bps equal? Three conditions must be met: (1) binary encoding must be used, (2) all elements used to encode characters must be equal in width and (3) synchronous transmission at a constant rate must be employed. In all other cases, the two terms are not equivalent.

Within a given piece of equipment, it is desirable to use the highest possible data rate. When external devices are interfaced, it is normal practice to select the highest standard signaling rate at which both the sending and receiving equipment can operate.

ERROR DETECTION

Since data transfers are subject to errors, data transmission should include some method of detecting and correcting errors. Numerous techniques are available, each used depending on the specific circumstances, such as what types of errors are likely to be encountered. Some error detection techniques are discussed in the **Modulation Sources** chapter. One of the simplest and most common techniques, parity check, is discussed here.

Parity Check

Parity check provides adequate error detection for some data transfers. This method transmits a parity bit along with the data bits. In systems using odd parity, the parity bit is selected such that the number of 1 bits in the transmitted character (data bits plus parity bit) is odd. In even parity systems, the parity bit is chosen to give the character an even number of ones. For example, if the data 1101001 is to be transmitted, there are 4 (an even number) ones in the data. Thus, the parity bit should be set to 1 for odd parity (to give a total of 5 ones) or should be 0 for even parity (to maintain the even number, 4). When a character is received, the receiver checks parity by counting the ones in the character. If the parity is correct, the data is assumed to be correct. If the parity is wrong, an error has been detected.

Parity checking only detects a small fraction of possible errors. This can be intuitively understood by noting that a randomly chosen word has a 50% chance of having even parity and a 50% chance of having odd parity. Fortunately, on relatively error-free channels, single-bit errors are the most common and parity checking will always detect a single bit in error. Parity checking is a simple error detection strategy. Because it is easy to implement, it is frequently used.

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 transistion 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 may also 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, polar keying (technically bipolar keying) is used on longer lines. Polar 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.

Since neutral keying is usually used inside equipment and polar keying for lines leaving the equipment, signals must be converted between polar and neutral. Op amp circuits, line drivers and line receivers are ways to handle this conversion. There are a number of different types available, but the most popular ones are the 1488 quad line driver and the 1489 quad line receiver. The 1488 is capable of converting four data streams at standard TTL levels to output levels that meet EIA RS-232-C or CCITT V.24 standards. The 1489 has four receivers that can convert RS-232-C or V.24 levels to TTL/DTL levels.

Connectors

These digital interfacing issues are simply digital communications issues for the specific case of direct wire connections between devices. Since direct wire connections are involved, the designer should consider the type of connector between the devices and its pin assignments. Numerous connectors are available or can be created.