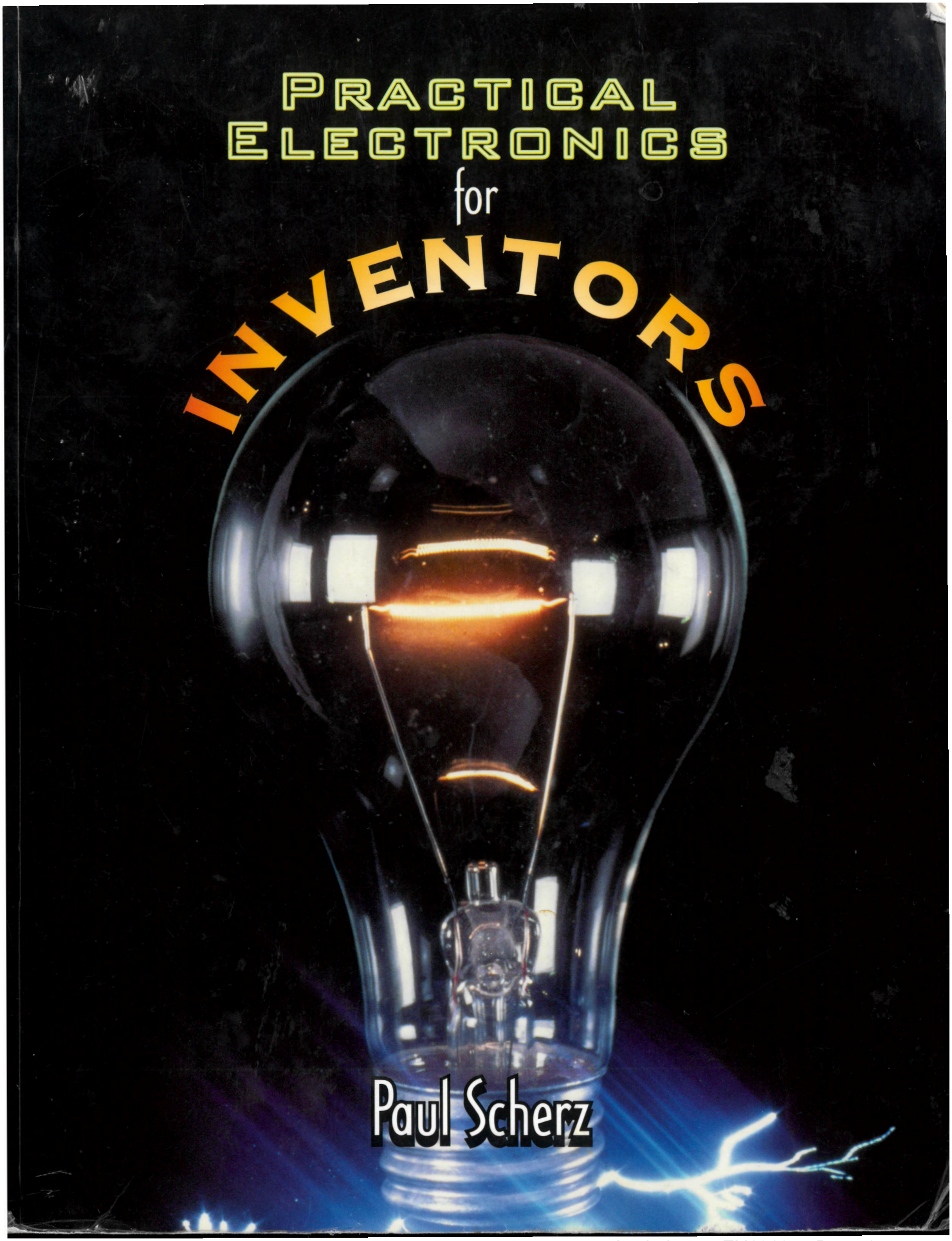
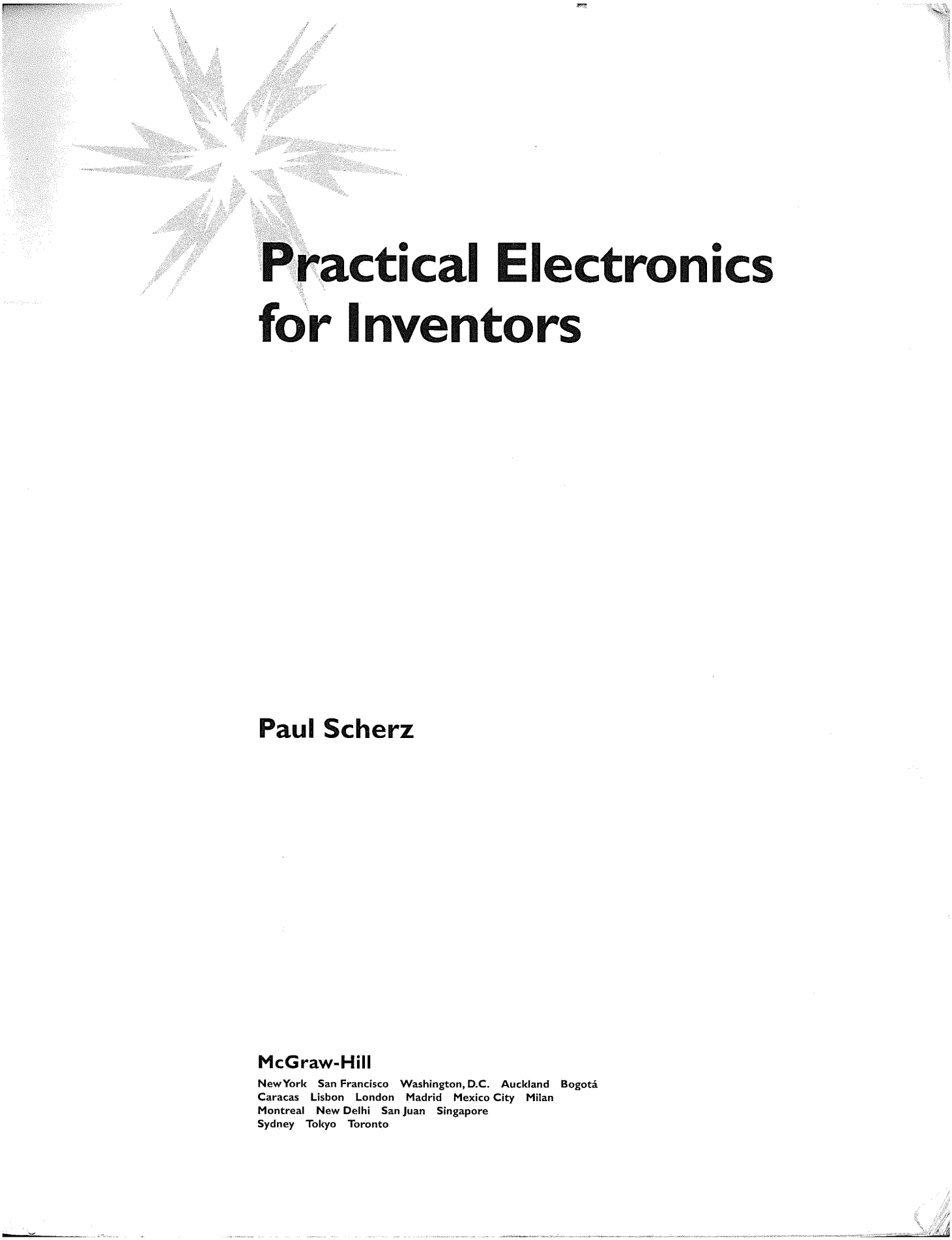
Practical Electronics for Inventors



Paul Scherz

Practical Electronics for Inventors



Practical Electronics for Inventors

Paul Scherz

McGraw-Hill

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

Cataloging-in-Publication Data is on file with the Library of Congress

McGraw-Hill
A Division of The McGraw-Hill Companies

5 6 7 8 9 10 11 QWK 0 9 8 7 6 5 4 3 2 1

ISBN 0-07-058078-2

The sponsoring editor for this book was Scott Grillo, the editing supervisor was Steven Melvin, and the production supervisor was Sherri Souffrance.
It was set in Palatino by North Market Street Graphics.
Phoenix Color was printer and binder.

This book is printed on recycled, acid-free paper containing a minimum of 50% recycled, de-inked fiber.

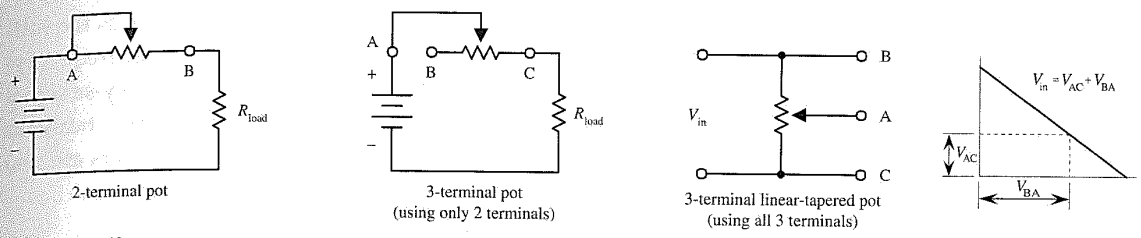


FIGURE 3.50

When purchasing variable resistors, make sure that you understand the distinction between linear-tapered and nonlinear-tapered variable resistors. The "taper" describes the way in which the resistance changes as the control knob is twisted. Figure 3.51 shows how the resistance changes as the control knob is turned for both a linear and nonlinear-tapered variable resistor.

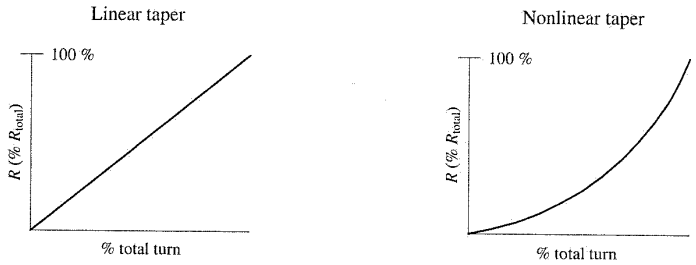


FIGURE 3.51

Why do variable resistors come with nonlinear tapers? Well, as it turns out, human physiology has a weird way of perceiving changes in signal intensity, such as sound and light intensities. For example, you may think that if you doubled the intensity of sound or light, you would perceive a doubling in sound and light. Unfortunately—at least in terms of intuition (not in terms of safety control for our brains)—humans do not work this way. In fact, our perceptions of sight and sound work as follows: Perceived loudness/brightness is proportional to $\log_{10}$ (actual intensity measured with a nonhuman instrument). Thus, if you are building an amplifier for a set of speakers or building a home light-dimming circuit, it would be wise to use a variable resistor with a nonlinear taper.

3.6 Capacitors

As you discovered in Chap. 2, capacitors act as temporary charge-storage units, whose behavior can be described by $I = CdV/dt$. A simple explanation of what this equation tells us is this: If you supply a 1-μF capacitor with a 1-mA, 1-s-long pulse of current, the voltage across its leads will increase 1000 V ($dV = Idt/C$). More generally, this equation states that a capacitor "likes" to pass current when the voltage across its leads is changing with time (e.g., high-frequency ac signals) but "hates" to pass current when the applied voltage is constant (e.g., dc signals). A capacitor's "dislike" for passing a current is given by its capacitive reactance $X_c = 1/\omega C$ (or $Z_c = -j/\omega C$ in complex form). As the applied voltage's frequency approaches infinity, the capacitor's reactance goes to zero, and it acts like a perfect conductor. However, as the frequency approaches zero, the capacitor's reactance goes to infinity, and it acts like an infinitely large resistor. Changing the value of C also affects the reactance. As C gets large, the reactance decreases, and the displacement current increases.

In terms of applications, the ability of a capacitor to vary its reactance as the voltage across its leads fluctuates makes it a particularly useful device in frequency-sensitive applications. For example, capacitors are used in frequency-sensitive voltage dividers, bypassing and blocking networks, filtering networks, transient noise suppressors, differentiator circuits, and integrator circuits. Capacitors are also used in voltage-multiplier circuits, oscillator circuits, and photoflash circuits.

3.6.1 How a Capacitor Works

A simple capacitor consists of two parallel plates. When the two plates are connected to a dc voltage source (e.g., a battery), electrons are "pushed" onto one plate by the negative terminal of the battery, while electrons are "pulled" from the other plate by the positive terminal of the battery. If the charge difference between the two plates become excessively large, a spark may jump across the gap between them and discharge the capacitor. To increase the amount of charge that can be stored on the plates, a nonconducting dielectric material is placed between them. The dielectric acts as a "spark blocker" and consequently increases the charge capacity of the capacitor. Other factors that affect capacitance levels include the capacitor's plate surface area and the distance between the parallel plates. Commercially, a capacitor's dielectric may be either paper, plastic film, mica, glass, ceramic, or air, while the plates may be either aluminum disks, aluminum foil, or a thin film of metal applied to opposite sides of a solid dielectric. The conductor-dielectric-conductor sandwich may be left flat or rolled into a cylinder. The figure below shows some examples.

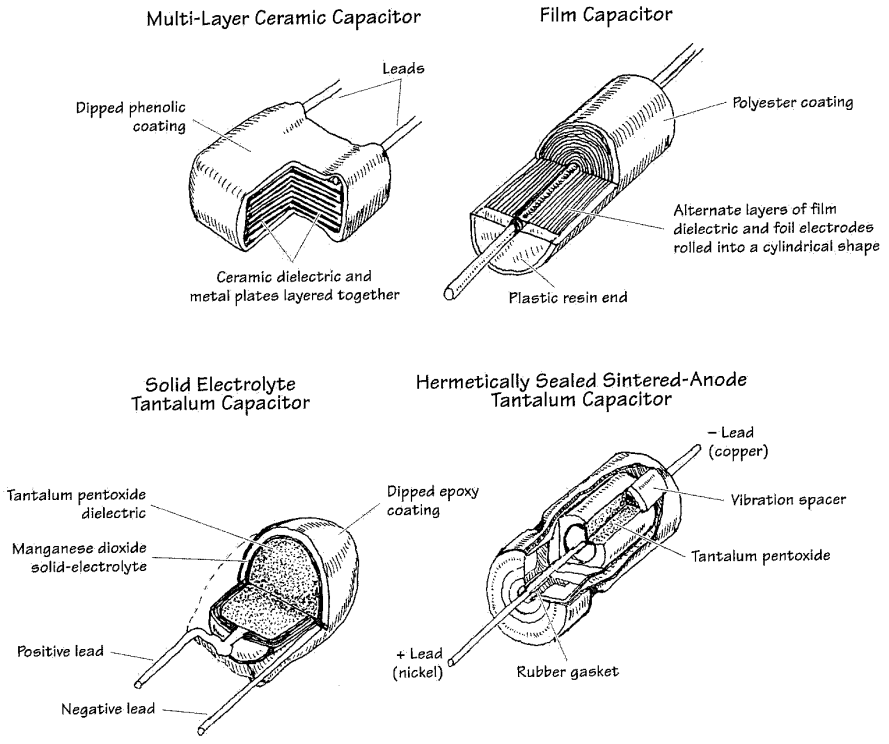


FIGURE 3.52

3.6.2 A Note about I = CdV/dt

Note that according to the laws of physics, no individual charges (electrons) ever make it across the gap separating the capacitor plates. However, according to $I = CdV/dt$, it appears as if there is a current flow across the gap. How can both these statements be true? Well, as it turns out, both are true. The misleading thing to do is to treat I as if it were a conventional current, such as a current through a resistor or wire. Instead, I represents a *displacement current*. A displacement current represents an apparent current through the capacitor that results from the act of the plates charging up with time and produces magnetic fields that induce electrons to move in the opposite plates. Current never goes across the gap—instead, charges on one plate are "shoved" by the changing magnetic fields produced by the opposing plate. The overall effect makes things appear as if current is flowing across the gap.

3.6.3 Water Analogy of a Capacitor

Here's the best thing I could come up with in terms of a water analogy for a capacitor. Pretend that electrons are water molecules and that voltage is water pressure.

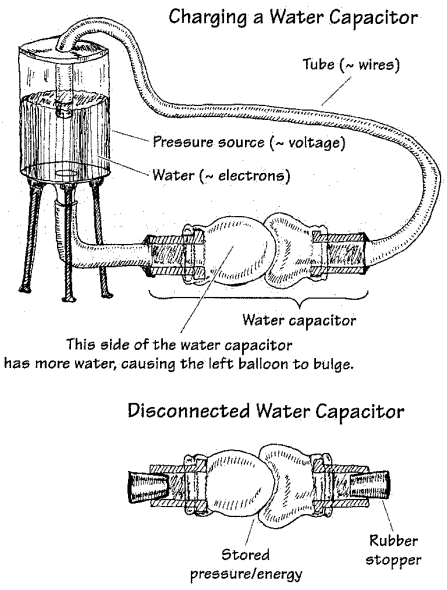


FIGURE 3.53

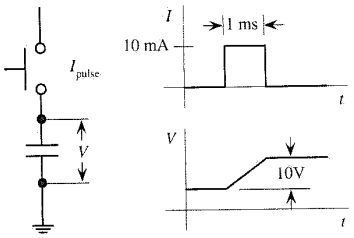
The water capacitor is built from two balloons. Normally, the two balloons are filled with the same amount of water; the pressure within each one is the same (analogous to an uncharged capacitor). In the figure, a real capacitor is charged by a battery, whereas the water capacitor is "charged" using a pump or pressurized water source. The real capacitor has a voltage across its plates, whereas in the water capacitor there is a difference in pressure between the two balloons. When the real capacitor is removed from the battery, it retains its charge; there is no conductive path through which charges can escape. If the water capacitor is removed from the pressurized water source—you have to pretend that corks are placed in its lead pipes—it too

retains its stored-up pressure. When an alternating voltage is applied across a real capacitor, it appears as if an alternating current (displacement current) flows through the capacitor due to the changing magnetic fields. In the water capacitor, if an alternating pressure is applied across its lead tubes, one balloon will fill with water and push against the other semifilled balloon, causing water to flow out it. As the frequency of the applied pressure increases, the water capacitor resembles a rubber membrane that fluctuates very rapidly back and forth, making it appear as if it is a short circuit (at least in ac terms). In reality, this analogy is overly simplistic and does not address the subtleties involved in a real capacitor's operation. Take this analogy lightheartedly.

3.6.4 Basic Capacitor Functions

FIGURE 3.54

Getting a Sense of What $I = CdV/dt$ Means

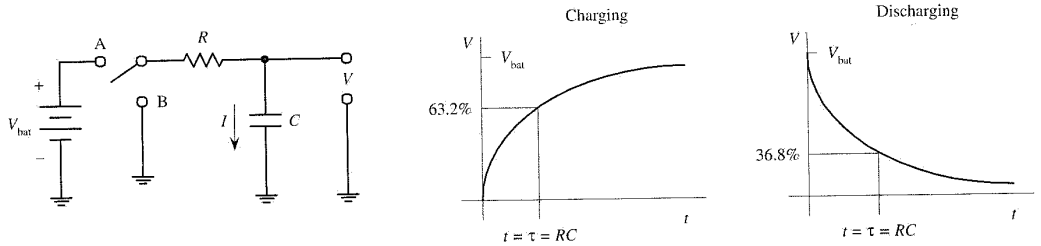


If you supply a 1-μF capacitor with a 10-mA, 1-ms pulse of current, the voltage across its leads will increase by 10 V. Here's how to figure it out:

$$I = C\frac{dV}{dt}$$

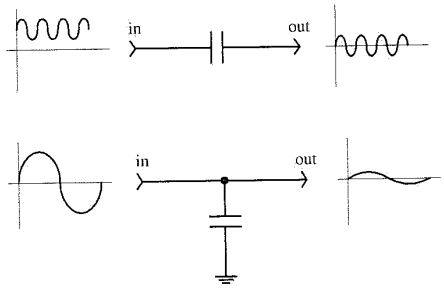
$$dV = \frac{Idt}{C} = \frac{(10\text{ mA})(1\text{ ms})}{1\ \mu\text{F}} = 10\text{V}$$

Charging/Discharging a Capacitor through a Resistor



When the switch is thrown to position *A*, the capacitor charges through the resistor, and the voltage across its leads increases according to the far-left graph. If the switch is thrown to position *B* after the capacitor has been charged, the capacitor will discharge, and the voltage will decay according to the near-right graph. See Section 2.18 for details.

Signal Filtering

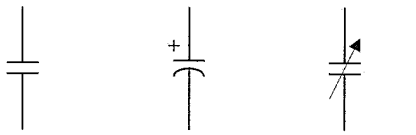


When a fluctuating signal with a dc component is sent through a capacitor, the capacitor removes the dc component yet allows the ac component through.

A capacitor can be used to divert unwanted fluctuating signals to ground.

3.6.5 *Kinds of Capacitors*

There are a number of different capacitor families available, each of which has defining characteristic features. Some families are good for storing large amounts of charge yet may have high leakage currents and bad tolerances. Other families may have great tolerances and low leakage currents but may not have the ability to store large amounts of charge. Some families are designed to handle high voltages yet may be bulky and expensive. Other families may not be able to handle high voltages but may have good tolerances and good temperature performance. Some families may contain members that are polarized or nonpolarized in nature. Polarized capacitors, unlike nonpolarized capacitors, are specifically designed for use with dc fluctuating voltages (a nonpolarized capacitor can handle both dc and ac voltages). A polarized capacitor has a positive lead that must be placed at a higher potential in a circuit and has a negative lead that must be placed at a lower potential. Placing a polarized capacitor in the wrong direction may destroy it. (Polarized capacitors' limitation to use in dc fluctuating circuits is counterbalanced by extremely large capacitances.) Capacitors also come in fixed or variable forms. Variable capacitors have a knob that can be rotated to adjust the capacitance level. The symbols for fixed, polarized, and variable capacitors are shown below.



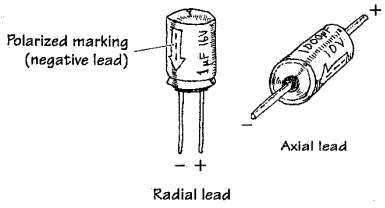
Fixed capacitor Polarized capacitor Variable capacitor

FIGURE 3.55

FIGURE 3.56

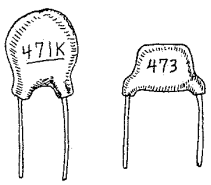
Now let's take a closer look at the capacitor families.

Electrolytic



These capacitors include both aluminum and tantalum electrolytics. They are manufactured by an electrochemical formation of an oxide film onto a metal (aluminum or tantalum) surface. The metal on which the oxide film is formed serves as the anode or positive terminal, the oxide film acts as the dielectric, and a conducting liquid or gel acts as the cathode or negative terminal. Tantalum electrolytic capacitors have larger capacitance per volume ratios when compared with aluminum electrolytics. A majority of electrolytic capacitors are polarized. Electrolytic capacitors, when compared with nonelectrolytic capacitors, typically have greater capacitances but have poor tolerances (as large as ±100 percent for aluminum and about ±5 to ±20 percent for tantalum), bad temperature stability, high leakage, and short lives. Capacitances range from about 1 μF to 1 F for aluminum and 0.001 to 1000 μF for tantalum, with maximum voltage ratings from 6 to 450 V.

Ceramic

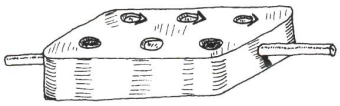


This is very popular nonpolarized capacitor that is small and inexpensive but has poor temperature stability and poor accuracy. It contains a ceramic dielectric and a phenolic coating. It is often used for bypass and coupling applications. Tolerances range from ±5 to ±100 percent, while capacitances range from 1 pF to 2.2 μF, with maximum voltages rating from 3 V to 6 kV.

Mylar

This is a very popular nonpolarized capacitor that is reliable, inexpensive, and has low leakage current but poor temperature stability. Capacitances range from 0.001 to 10 μF, with voltages ratings from 50 to 600 V.

Mica



This is an extremely accurate device with very low leakage currents. It is constructed with alternate layers of metal foil and mica insulation, stacked and encapsulated. These capacitors have small capacitances and are often used in high-frequency circuits (e.g., RF circuits). They are very stable under variable voltage and temperature conditions. Tolerances range from ±0.25 to ±5 percent. Capacitances range from 1 pF to 0.01 μF, with maximum voltage ratings from 100 V to 2.5 KV

FIGURE 3.56 (*Continued*)

Other kinds of capacitors include paper, polystyrene, polycarbonate, polyester, glass, and oil capacitors; their characteristics are covered in Table 3.7. Note that in Table 3.7 a large *insulation resistance* means that a capacitor has good leakage protection.

TABLE 3.7 Characteristics of Various Capacitors

TYPE	CAPACITANCE RANGE	MAXIMUM VOLTAGE	MAXIMUM OPERATING TEMPERATURE (°C)	TOLERANCE (%)	INSULATION RESISTANCE (MΩ)	COMMENTS
Electrolytics Aluminum Tantalum	 1 μF–1 F 0.001–1000 μF	 3–600 V 6–100 V	 85 125	 +100 to –20 ±5 to 20	 <1 >1	Popular, large capacitance, awful leakage, horrible tolerances
Ceramic	10 pF–1 μF	50–1000 V	125	±5 to 100	1000	Popular, small, inexpensive, poor tolerances.
Mica	1 pF–0.1 μF	100–600 V	150	±0.25 to ±5	100,000	Excellent performance; used in high-frequency applications
Mylar	0.001–10 μF	50–600 V		Good	Good	Popular, good performance, inexpensive
Paper	500 pF–50 μF	100,000 V	125	±10 to ±20	100	—
Polystyrene	10 pF–10 μF	100–600 V	85	±0.5	10,000	High quality, very accurate; used in signal filters
Polycarbonate	100 pF–10 μF	50–400 V	140	±1	10,000	High quality, very accurate
Polyester	500 pF–10 μF	600 V	125	±10	10,000	—
Glass	10–1000 pF	100–600 V	125	±1 to ±20	100,000	Long-term stability
Oil	0.1–20 μF	200 V–10 kV			Good	Large, high-voltage filters, long life

3.6.6 Variable Capacitors

Variable capacitors are devices that can be made to change capacitance values with the twist of a knob. These devices come in either air-variable or trimmer forms. Air-variable capacitors consist of two sets of aluminum plates (stator and rotor) that mesh together but do not touch. Rotating the rotor plates with respect to the stator varies the capacitor's effective plate surface area, thus changing the capacitance. Air-variable capacitors typically are mounted on panels and are used in frequently adjusted tuning applications (e.g., tuning communication receivers over a wide band of frequencies). A trimmer capacitor is a smaller unit that is designed for infrequent fine-tuning adjustments (e.g., fine-tuning fixed-frequency communications receivers, crystal frequency adjustments, adjusting filter characteristics). Trimmers may use a mica, air, ceramic, or glass dielectric and may use either a pair of rotating plates or a compression-like mechanism that forces the plates closer together.

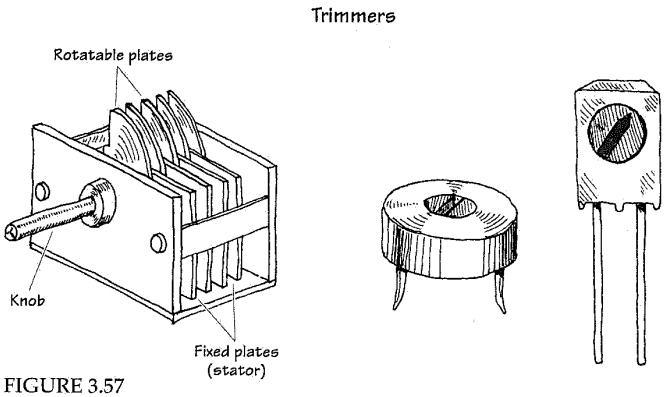


FIGURE 3.57

3.6.7 Reading Capacitor Labels

Reading capacitor labels is tricky business. Each family of capacitors uses its own unique labeling system. Some systems are easy to understand, whereas others make use of misleading letters and symbols. The best way to figure out what a capacitor label means is to first figure out what family the capacitor belongs to. After that, try seeing if the capacitor label follows one of the conventions in Fig. 3.58.

3.6.8 Important Things to Know about Capacitors

Even though two capacitors may have the same capacitance values, they may have different voltage ratings. If a smaller-voltage capacitor is substituted in place of a higher-voltage capacitor, the voltage level across the replacement may "zap" its dielectric, turning it into a low-level resistor. Also remember that with polarized capacitors, the positive lead must go to the more positive connection; otherwise, it may get zapped as well.

As a practical note, capacitor tolerances can be atrocious. For example, an aluminum electrolytic capacitor's capacitance may be as much as 20 to 100 percent off the actual value. If an application specifies a low-tolerance capacitor, it is usually safe to substitute a near-value capacitor in place of the specified one.

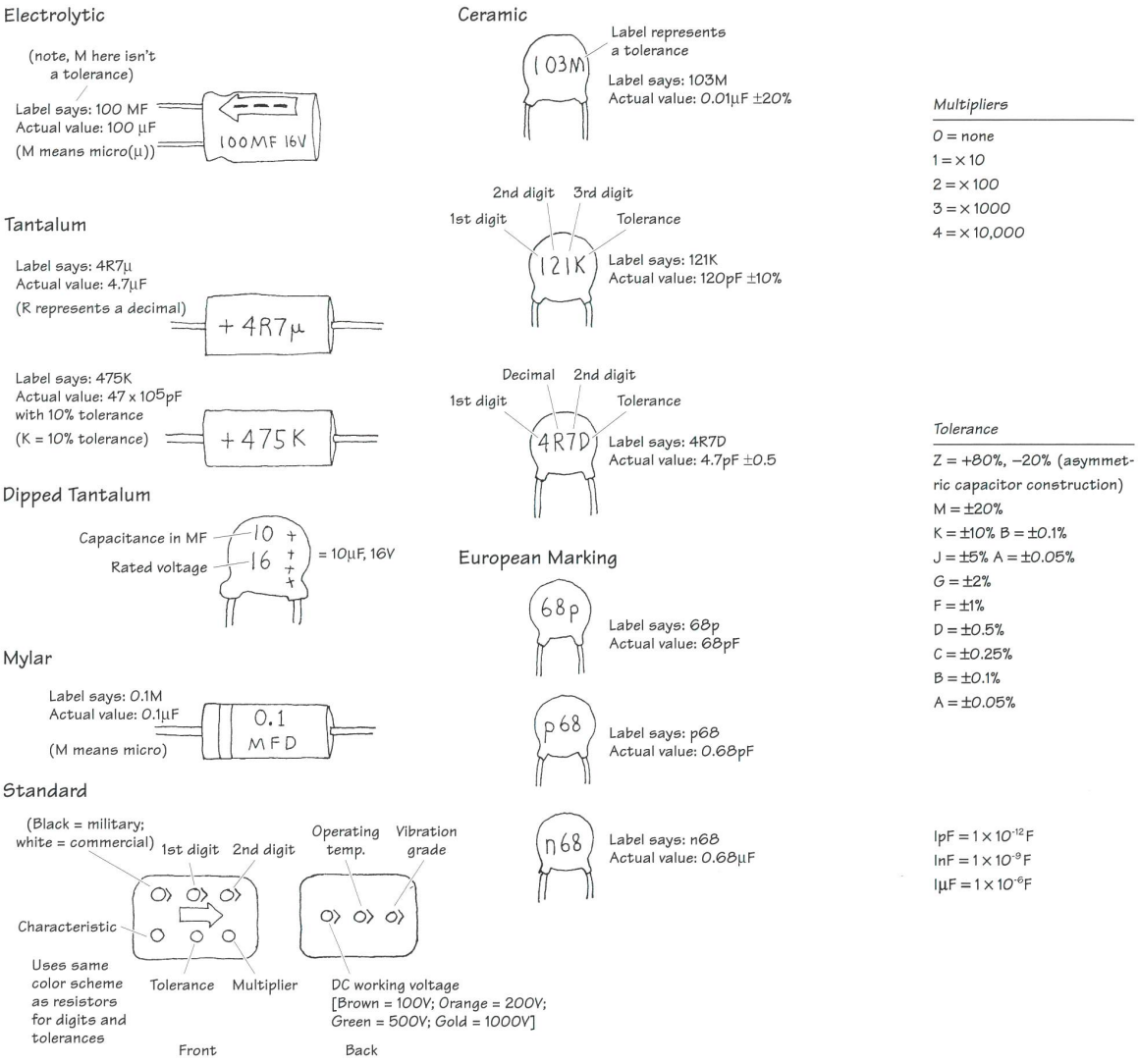


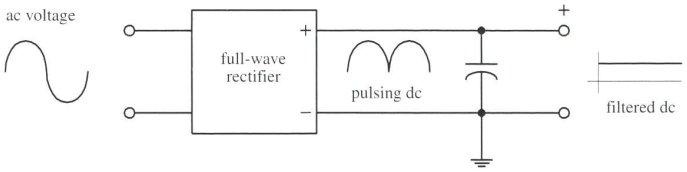
FIGURE 3.58

As a final note, capacitors with small capactances (less than 0.01 µF) do not pose much danger to humans. However, when the capacitances start exceeding 0.1 µF, touching capacitor leads can be a shocking experience. For example, large electrolytic capacitors found in television sets and photoflashes can store a lethal charge. As a rule, never touch the leads of large capacitors. If in question, discharge the capacitor by shorting the leads together with a screwdriver tip before handling it.

FIGURE 3.59

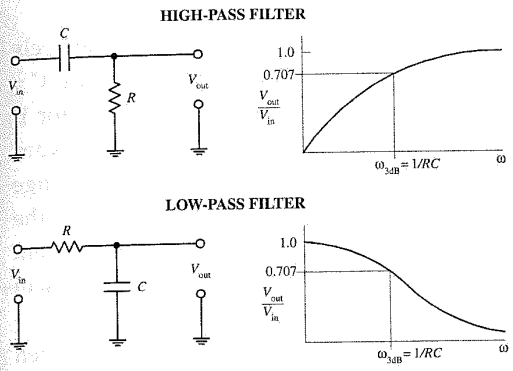
3.6.9 *Applications*

Power Supply Filtering



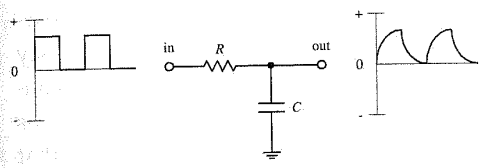
Capacitors are often used to smooth out pulsing dc signals generated by rectifier sections within power supplies by diverting the ac portion of the wavelength to ground, while passing the dc portion.

Filter Circuits



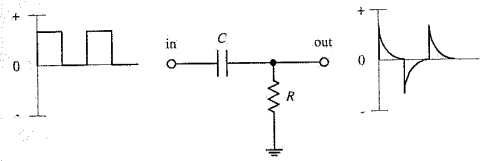
High-pass and low-pass filters can be made from simple *RC* networks. In a high-pass filter, low frequencies are blocked by the capacitor, but high-frequency signals get through to the output. In a low-pass filter, high-frequency signals are routed to ground, while low-frequency signals are sent to the output. See Chaps. 2 (section 2.30) and 8 for more on filters.

Passive Integrator



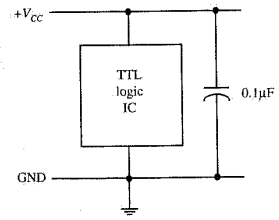
The input signal is integrated at the output. *RC* is the time constant, and it must be at least 10 times the period of the input signal. If not, the amplitude of the output will then be a low-pass filter that blocks high frequencies.

Passive Differentiator



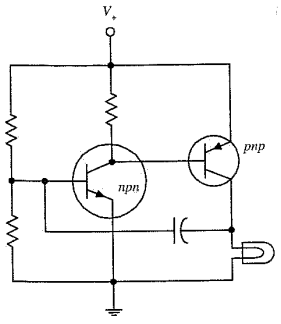
The input signal is differentiated at the output. *RC* network transforms incoming square waves into a pulsed or spiked waveform. The *RC* time constant should be 1/10 (or less) of the duration of the incoming pulses. Differentiators are often used to create trigger pulses.

Spike and Noise Suppression



With certain types of ICs (e.g., TTL logic ICs), sudden changes in input and output states can cause brief but drastic changes in power supply current which can result in sharp, high-frequency current spikes within the power supply line. If a number of other devices are linked to the same supply line, the unwanted spike can cause false triggering of these devices. The spikes can also generate unwanted electromagnetic radiation. To avoid unwanted spikes, decoupling capacitors can be used. A decoupling capacitor [typically 0.1 to 1 μF (>5V) for TTL logic ICs] is placed directly across the positive (V_{CC}) and ground pins of each IC within a system. The capacitors help absorb the spikes and keep the V_{CC} level constant, thus reducing the likelihood of false triggering and electromagnetic radiation.

Simple Oscillator



A capacitor can be used like a "spring" in an oscillator circuit. Here, a capacitor alters the biasing voltage at the *npn* transistor's base. At one moment, the *npn* transistor receives a biasing voltage large enough to turn it on. This causes the *pnp* transistor to turn on too (its base becomes properly biased). Current will flow through the bulb, causing it to glow. However, sometime later, the capacitor stores enough charge to bring the voltage level at the *npn* transistor's base down, turning it off. The *pnp* transistor then turns off, and the light turns off. But shortly afterward, the charge from the capacitor flows through lower resistor to ground, and the voltage at the *npn* transistor's base goes back up. It turns on again, and the process repeats itself over and over again. The resistance values of the voltage-divider resistors and the capacitance value of the capacitor control the oscillatory frequency.

FIGURE 3.59 (*Continued*)

3.7 Inductors

As you learned in Chap. 2, inductors act to resist changes in current flow while freely passing steady (dc) current. The behavior of an inductor can be described by $V = LdI/dt$. This equation tells you that if the current flow through a 1-H inductor is changing at a rate of 1 A/s, the induced voltage across the inductor will be 1 V. Now, the direction (polarity) of the voltage will be in the direction that opposes the change in current. For example, if the current increases, the voltage inducted across the inductor will be more negative at the end through which current enters. If the current decreases, the voltage at the end of the inductor through which current enters will become more positive in an attempt to keep current flowing.

More generally, $V = LdI/dt$ states that an inductor "likes" to pass current when the voltage across its leads does not change (e.g., dc signals) but "hates" to pass current when the voltage across its leads is changing with time (e.g., high-frequency ac signals). An inductor's "dislike" for passing a current is given by its inductive reactance $X_L = \omega L$ (or $Z_L = j\omega L$ in complex form). As the applied voltage's frequency approaches zero, the inductor's reactance goes to zero, and it acts like a perfect conductor. However, as the frequency approaches infinity, the inductor's reactance goes to infinity, and it acts like an infinitely large resistor. (Using $X_L = \omega L$, you can see that if you take a 20-mH coil and apply a 100-kHz signal across it, the inductive reactance will be 2000 Ω.) Notice that L also influences the inductor's reactance. As L increases, the inductor's reactance increases, while the current flow through it decreases.

In terms of applications, the ability of an inductor to vary its reactance as the voltage across its leads fluctuates makes it a particularly useful device in frequency-sensitive applications. For example, inductors are used in frequency-sensitive voltage dividers, blocking networks, and filtering networks (e.g., RF transmitter and receiver circuits). Inductors are also used in oscillator circuits, transformers, and magnetic relays—where they act as magnets.

3.7.1 How an Inductor Works

The physical explanation of how an inductor resists changes in current flow can be explained by the following figure. Assume that you have a steady (dc) current flowing through an inductor's coil, as shown at left below.

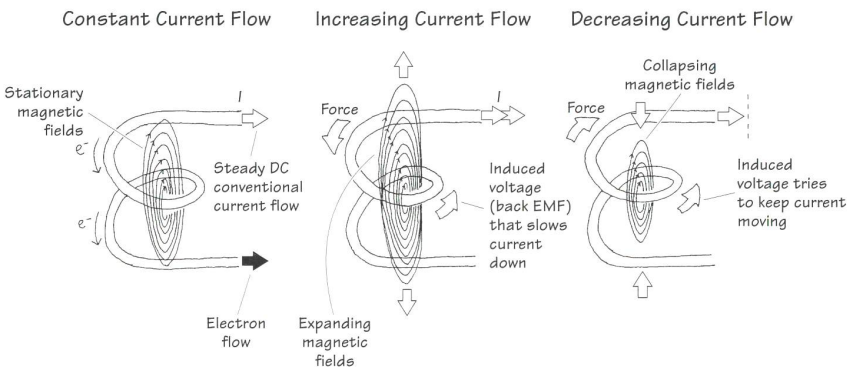


FIGURE 3.60

Within the dc current, electrons are collectively moving at a constant velocity, and as a result of their movement, a steady flux of magnetic fields encircles them. The direction of the magnetic fields is found by using the left-hand rule. The left-hand rule tells us that if we pretend to be hitch-hiking with our left hand, the direction in which our thumb points represents the direction in which electrons are moving, whereas the direction in which our four curled fingers point represents the direction of the magnetic field lines. When the magnetic field lines produced by the flowing electrons within every segment of the coil are added vectorally, the resulting magnetic field appears to emanate from the center of the coil, as shown in Fig. 3.60. Now, as long as the electron flow is constant (steady flow), the centralized magnetic field will not change. The magnetic field lines may pass through various sections of the wire's coil, but they will not affect the current flow through these sections. However, if the current flow suddenly increases, the magnetic field lines expand, as shown in Fig. 3.60. According to Faraday's law of induction (Faraday's law states that a changing magnetic field induces a force on a charged object), the expanding magnetic field will induce a force on the electrons within the coil. The force applied by the expanding magnetic fields on the electrons is such that the electrons are forced to slow down. Now, if you start with a steady (dc) current and suddenly decrease the current flow, the magnetic fields collapse, and they induce a force on the electrons in such a way as to keep them moving, as shown in Fig. 3.60.

In terms of $V = LdI/dt$, the dI/dt term represents the change in current with time, and V represents the inducted voltage (the work done on electrons to reduce changes in their motions). Other names for V include self-inducted voltage, counter emf, and back emf. Now, the only thing left in the equation is the inductance L. L is the proportionality constant between the induced voltage and the changing current; it tells us how good an inductor is at resisting changes in current. The value of L depends on a number of factors, such as the number of coil turns, coil size, coil spacing, winding arrangement, core material, and overall shape of the inductor (see Fig. 3.61).

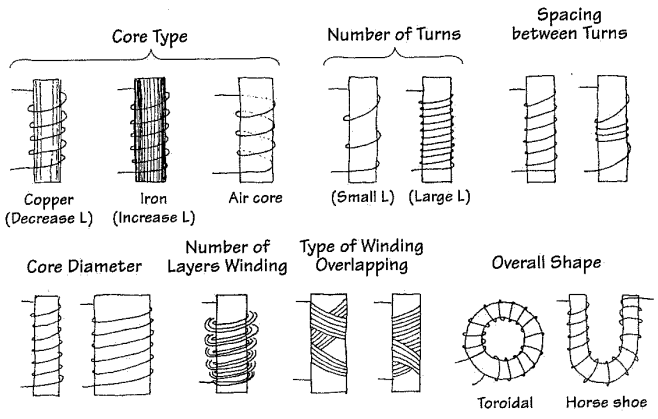


FIGURE 3.61

For a simple, single-layer, closely wound coil, the inductance is given by

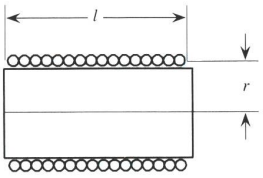
$$L = \frac{\mu N^2 A}{l}$$

where μ is the permeability of the material on which the coil is wound (in free space, $\mu = 4\pi \times 10^{-7}$ N/A^2), N is the number of coil windings, A is the cross-sectional area of the coil, and l is the length of the coil.

The following are approximation methods for finding the inductances of closely spaced single and multilayer air-core coils.

Single-Layer Air-Core Coil

$$L = \frac{(N \times r)^2}{9r + 10l}$$



Multi-Layer Air-Core Coil

$$L = \frac{0.8(N \times 10l)}{6r + 9l + 10b}$$

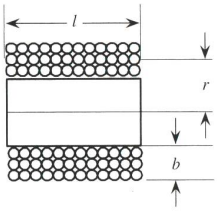
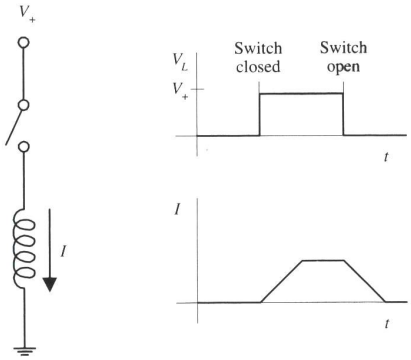


FIGURE 3.62

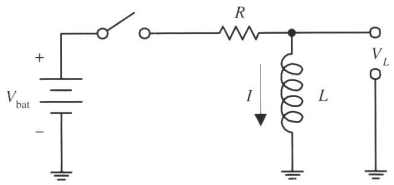
3.7.2 *Basic Inductor Operation*

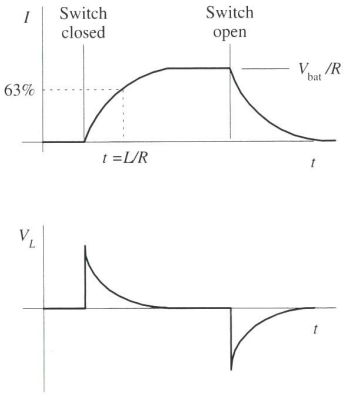
Current/Voltage Behavior



If the switch in a circuit to the left is opened for a specific amount of time and then closed, the inductor initially will "fight" the applied voltage, and the current will rise with a slope related to the coil's inductance. Once steady state is reached, the coil passes a current that is equal to the applied voltage divided by the resistance of the coil; the inductor acts like a low-resistive wire. When the switch is opened, the inductor "fights" the sudden change, and current will fall with a negative slope related to the coil's inductance. The current is related to the applied voltage by $V = LdI/dt$.

Resistor/Inductor Behavior

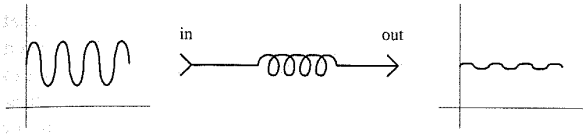




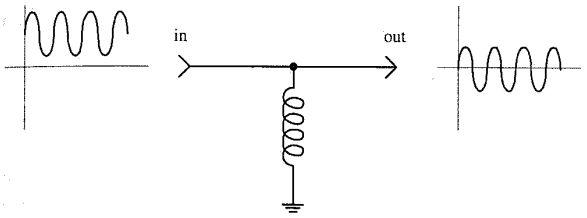
When the switch is closed, the inductor fights to keep the current flow down by inducing a voltage across its leads that opposes the applied voltage. The current reaches 63 percent of max at $t = L/R$. As time passes, the current flow levels off and assumes a dc state. At this point, the inductor acts like a wire, and the current is simply equal to V/R. When the switch is then opened, the opposite effect occurs; the inductor fights to keep current flow up by inducting a voltage in the opposite direction.

FIGURE 3.63

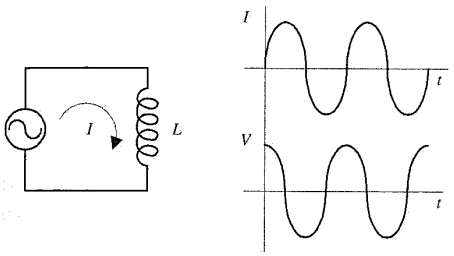
Signal Filtering



An inductor blocks fluctuating currents yet allows steady (dc) currents to pass.



Here, an inductor acts to block dc current (dc current is routed through inductor to ground) yet allows ac current to pass.



If an alternating voltage is applied across an inductor, the voltage and current will be 90° out of phase (the current will lag the voltage by 90°). As the voltage across the inductor goes to its maximum value, the current goes to zero (energy is being stored in the magnetic fields). As the voltage across the inductor goes to zero, the current through the inductor goes to its maximum value (energy is released from the magnetic fields).

FIGURE 3.63 (*Continued*)

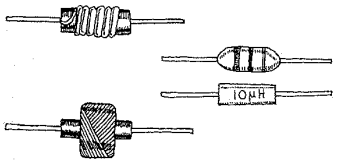
3.7.3 *Kinds of Coils*

Coils typically have either an air, iron, or powered-iron core. An inductor with an iron core has a greater inductance than an inductor with an air core; the permeability of iron is much greater than the permeability of air. Some inductors come with variable cores that slide in and out of the coil's center. The distance that the core travels into the coil determines the inductance.

FIGURE 3.64

Here's a rundown of some common coils you will find at the store.

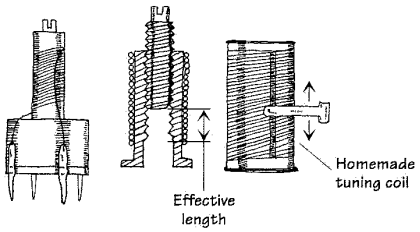
Chokes



These are general-purpose inductors that act to limit or suppress fluctuating current while passing steady (dc) current. They come in an assortment of shapes, winding arrangements, package types, and inductance and tolerance ratings. Typical inductances range from about 1 to about 100,000 µH, with tolerances from 5 to 20 percent. Some use a resistor-like color code to specify inductance values.

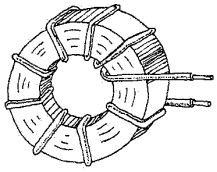
FIGURE 3.65

Tuning Coil



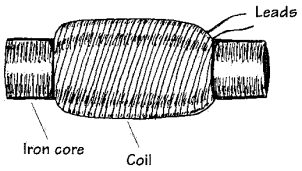
This type of coil often contains a screwlike "magnetic field blocker" that can be adjusted to select the desired inductance value. Tuning coils often come with a series of taps. They are frequently used in radio receivers to select a desired frequency (see wireless electronics). A homemade tuning coil can be constructed by wrapping wire around a plastic cylinder and then attaching a movable wiper, as shown at left. The wire can be lacquer-covered magnetic wire. To provide a conductive link between the wiper and the coil, a lacquer strip can be removed with a file.

Toroidal Coil



This type of coil resembles a donut with a wire wrapping. They have high inductance per volume ratios, have high quality factors, and are self-shielding. Also, they can be operated at extremely high frequencies.

Antenna Coil



This type of coil contains an iron core that magnifies magnetic field effects, making it very sensitive to very small changes in current. Such coils are used to tune in ultra-high-frequency signals (e.g., RF signals).

FIGURE 3.65 (*Continued*)

3.8 Transformers

A basic transformer is a four-terminal device that is capable of transforming an ac input voltage into a higher or lower ac output voltage (transformers are not designed to raise or lower dc voltages). A typical transformer consists of two or more coils that share a common laminated iron core, as shown in Fig. 3.66. One of the coils is called the *primary* (containing N_p turns), while the other coil is called the *secondary* (containing N_S turns).

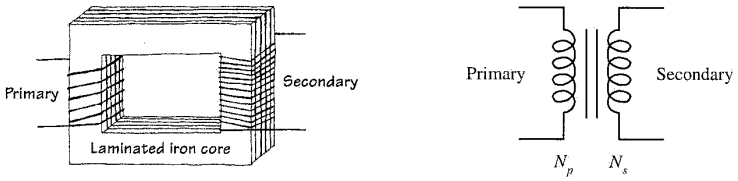


FIGURE 3.66

When an ac voltage is applied across the primary coil, an alternating magnetic flux of $\Phi = \int (V_{in}/N_p)\, dt$ emanates from the primary, propagates through the iron-laminated core, and passes through the secondary coil. (The iron core increases the inductance, and the laminations decrease power-consuming eddy currents.) According to Faraday's law of induction, the changing magnetic flux induces a voltage of $V_S = N_S d\Phi/dt$ within the secondary coil. Combining the flux equation and the secondary-induced-voltage equation results in the following useful expression:

$$V_S = \frac{N_S}{N_p} V_p$$

This equation says that if the number of turns in the primary coil is larger than the number of turns in the secondary coil, the secondary voltage will be smaller than the primary voltage. Conversely, if the number of turns in the primary coil is less than the number of turns in the secondary, the secondary voltage will be larger than the primary voltage.

When a source voltage is applied across a transformer's primary terminals while the secondary terminals are open circuited (see Fig. 3.67), the source treats the transformer as if it were a simple inductor with an impedance of $Z_p = j\omega L_p$, where L_p represents the inductance of the primary coil. This means that the primary current will lag the primary voltage (source voltage) by 90°, and the primary current will be equal to V_p/Z_p, according to Ohm's law. At the same time, a voltage of $(N_S/N_p)V_p$ will be present across the secondary. Because of the polarity of the induced voltage, the secondary voltage will be 180° out of phase with the primary voltage.

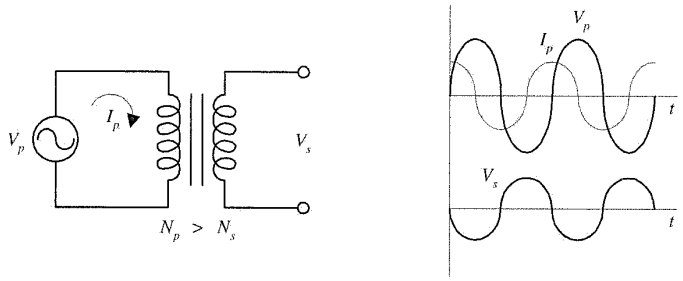


FIGURE 3.67

Now, let's take a look at what happens when you attach a load to the secondary, as shown in Fig. 3.68. When a load R_L is placed across the secondary, the induced voltage acts to move current through the load. Since the magnetic flux from the primary is now being used to induce a current in the secondary, the primary current and voltage move toward being in phase with each other. At the same time, the secondary voltage and the induced secondary current I_s move toward being in phase with each other, but both of them will be 180° out of phase with the primary voltage and current (the physics of induced voltages again).

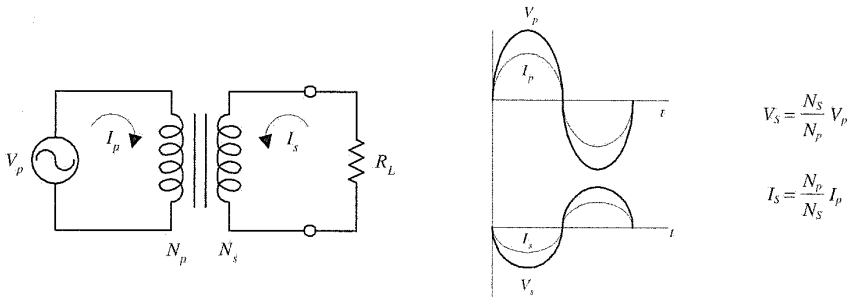


FIGURE 3.68

To figure out the relationship between the primary and secondary currents, consider that an ideal transformer is 100 percent efficient (real transformers are around 95 to 99 percent efficient), and then infer that all the power disipated by the load will be equal to the power supplied by the primary source. By combining the power law ($P = V^2/R$) and the transformer voltage relation, it can be seen that both the power dissipated by the load and the power supplied in the primary are equal to

$$P = \frac{V_S^2}{R_L} = \frac{N_S^2 V_p^2}{N_p^2 R_L}$$

By using the $P = IV$ form of the power law, the primary current is

$$I_p = \frac{P_R}{V_p} = \frac{N_S^2 V_p}{N_p^2 R_L}$$

Now, Ohm's law can be used to come up with the equivalent resistance the source feels from the combined transformer and load sections:

$$R_{eq} = \frac{V_p}{I_p} = \left(\frac{N_p}{N_S}\right)^2 R_L$$

In terms of circuit reduction, you can then replace the transformer/load section with R_{eq} (see Fig. 3.69).

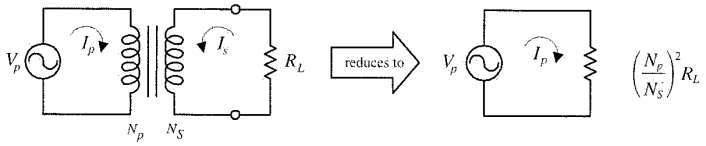


FIGURE 3.69

This reduction trick also can be applied to loads that have complex impedances (e.g., *RLC* networks). The only alteration is that R_L must be replaced with the complex load impedance Z_L.

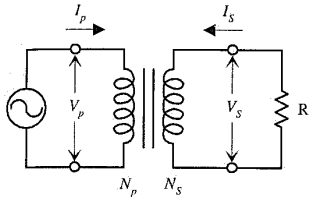
To finish up with the theory, suppose that you are simply interested in figuring out the relationship between the primary current and the secondary current. Since the power in the primary and the power in the secondary must be equal, you can make use of $P = IV$ and the transformer voltage relation to come up with

$$I_S = \frac{N_p}{N_S} I_p$$

This equation tells you that if you increase the number of turns in the secondary (which results in a larger secondary voltage relative to the primary voltage), the secondary current will be smaller than the primary current. Conversely, if you decrease the number of turns in the secondary (which results in a smaller secondary voltage relative to the primary voltage), the secondary current will be larger than the primary current.

3.8.1 Basic Operation

Voltage/Current Relationships

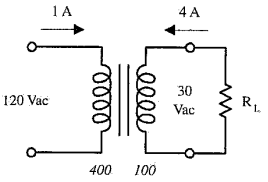


$$V_S = \frac{N_S}{N_p} V_p$$

$$I_S = \frac{N_p}{N_S} I_p$$

To obtain a secondary voltage that's larger than the primary voltage, the number of turns in the secondary must be greater than the number of turns in the primary. Increasing the secondary voltage results in a smaller secondary current relative to the primary current. To obtain a secondary voltage that's smaller than the primary voltage, the number of turns in the secondary must be smaller than the number of turns in the primary. Decreasing the secondary voltage relative to the primary voltage will result in a larger secondary current.

Step-Down Voltage/Step-Up Current

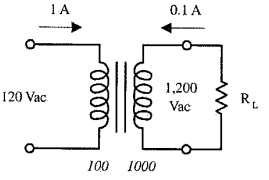


$$V_S = \frac{100}{400}(120\text{ V}) = 30\text{ V}$$

$$I_S = \frac{400}{100}(1\text{ A}) = 4\text{ A}$$

A transformer with 400 turns in the primary and 100 turns in the secondary will convert a 120-V ac, 1-A primary voltage and current into a 30-V ac, 4-A secondary voltage and current. A transformer that decreases the secondary voltage relative to the primary voltage is called a *step-down transformer.*

Step-Up Voltage/Step-Down Current



$$V_S = \frac{1000}{100}(120\text{ V}) = 1200\text{ V}$$

$$I_S = \frac{100}{1000}(1\text{ A}) = 0.1\text{ A}$$

A transformer with 100 turns in the primary and 1000 turns in the secondary will convert a 120-V ac, 1-A primary voltage and current into a 1200-V ac, 0.1-A secondary voltage and current. A transformer that increases the secondary voltage relative to the primary voltage is called a *step-up transformer.*

Circuit Analysis Problem

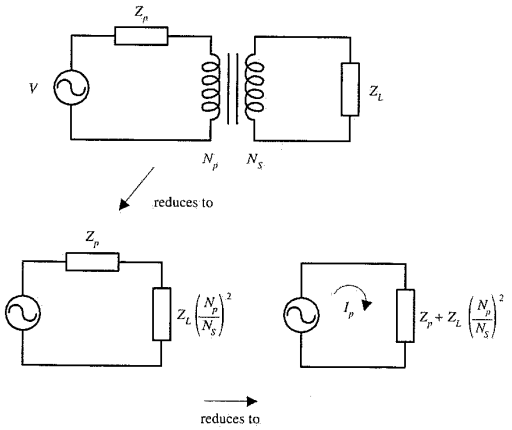
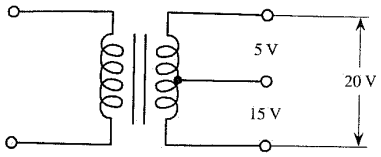


FIGURE 3.70

Here's an example of how to treat transformers in circuit analysis. Say that you have a complex network of components that contains resistive, capacitive, and inductive elements (e.g., *RLC* network) in both the primary and secondary sides. To figure out the equivalent impedance the source feels, you can reduce the circuit by finding the equivalent impedance of the load (Z_L) and transformer combined. From what you learned earlier, this is given by $Z_L(N_p/N_S)^2$. After that, you simply have to deal with the impedance in the primary (Z_p) and $Z_L(N_p/N_S)^2$. You apply a final reduction by adding Z_p and $Z_L(N_p/N_S)^2$ in series. The effective primary current is found by using ac Ohm's law ($I = V/Z$).

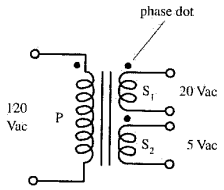
3.8.2 *Special Kinds of Transformers*

Tapped Transformer

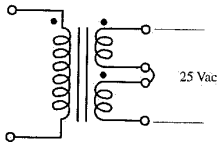


Tapped transformers have an additional connection, or tap, on their primary and/or secondary windings. An additional tap lead on the secondary gives a transformer three possible output voltages.

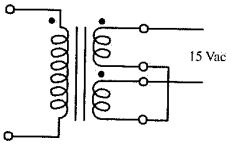
Multiple-Winding Transformer



Series aiding



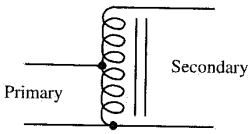
Series opposing



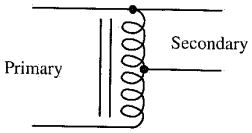
It is often useful to have a number of different secondary windings, each of which is electrically isolated from others (unlike a taped transformer). Each of the secondary coils will have a voltage that is proportional to its number of turns. The secondary windings can be connected in series-aiding (voltage is summed) or series-opposing (the voltage is the difference) configurations. Dots are often used to indicate the terminals that have the same phase.

Autotransformer

Step-up

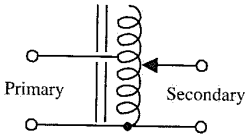


Step-down



This device only uses a single coil and a tap to make a primary and secondary. Autotransformers can be used to step up or step down voltages; however, they are not used for isolation application because the primary and secondary are on the same coil (there is no electrical isolation between the two). These devices are frequently used in impedance-matching applications.

Continuous Tapped Transformer (Variac)



This device contains a variable tap that slides up and down the secondary coil to control the secondary coil length and hence the secondary voltage.

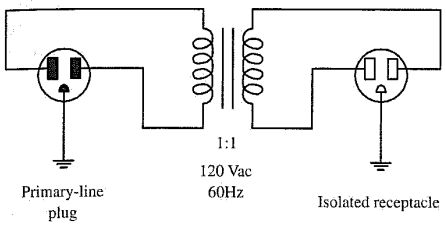
FIGURE 3.71

3.8.3 Applications

Three fundamental applications for transformers include isolation protection, voltage conversion, and impedance matching. Here are a few examples showing application for all three.

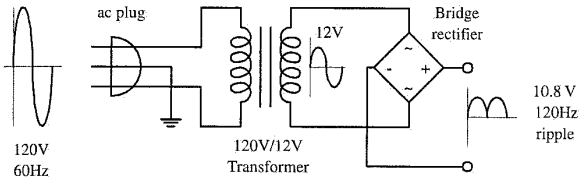
FIGURE 3.72

Isolation



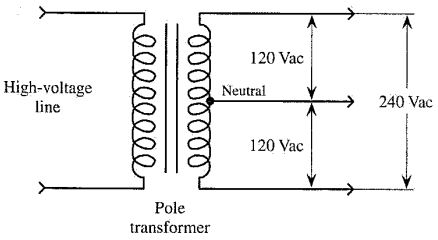
Transformers with a 1:1 ratio between primary and secondary windings are often used to protect secondary circuits (or individuals touching secondary elements, e.g., knobs, panels, etc.) from electrical shocks. The reason why the protection works is that the secondary is only magnetically coupled—not electrically coupled—with the high-current utility line. All test equipment, especially ones that are "floated" (ground is removed), must use isolation protection to eliminate shock hazards. Another advantage in using an isolation transformer is that there are no dc connections between circuit elements in the primary and secondary sides; ac devices can be coupled with other devices through the transformer without any dc signals getting through. The circuit in the figure shows a simple example of how a power outlet can be isolated from a utility outlet. The isolated receptacle is then used to power test equipment.

Power Conversion



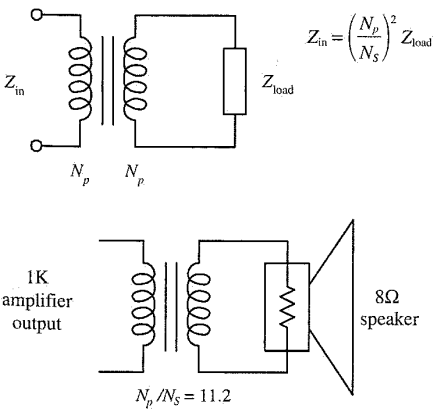
Transformers are essential ingredients in power supply design. Here a 120-to-12-V transformer steps down a 120-V, 60-Hz line voltage to a 12-V, 60-Hz secondary voltage. A bridge rectifier (network of four diodes) then rectifies the secondary voltage into a pulsed dc voltage with a peak of 10.8 V and a frequency of 120 Hz. (1.2 V is lost during rectification due to diode biasing voltages, and it appears that the output is double in frequency due to the negative swings being converted into positive swings by the rectifier.) The average dc voltage at the output is equal to 0.636 of the peak rectified voltage.

Tapped Transformer Application



In the United States, main power lines carry ac voltages of upwards of 1000 V. A center-tapped pole transformer is used to step down the line voltage to 240 V. The tap then acts to break this voltage up into 120-V portions. Small appliances, such as TVs, lights, and hairdryers, can use either the top line and the neutral line or the bottom line and the neutral line. Larger appliances, such as stoves, refrigerators, and clothes dryers, make use of the 240-V terminals and often use the neutral terminal as well. See Appendix A for more on power distribution and home wiring.

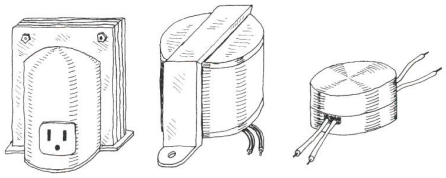
Impedance Matching



Earlier you discovered that any device connected to the primary side of a transformer with a secondary load feels an equivalent impedance equal to $(N_p/N_S)^2 Z_{load}$. By manipulating the N_p/N_S ratio, you can trick an input device into thinking it is connected to an output device with a similar impedance, even though the output device may have an entirely different impedance. For example, if you want to use a transformer to match impedances between an 8-Ω speaker and a 1-k amplifier, you plug 8 Ω into Z_{load} and 1000 Ω into Z_{in} and solve for N_p/N_S. This provides the ratio between the primary and secondary windings that is needed to match the two devices. N_p/N_S turns out to be 11.2. This means the primary must have 11.2 more winding than the secondary.

3.8.4 Real Kinds of Transformers

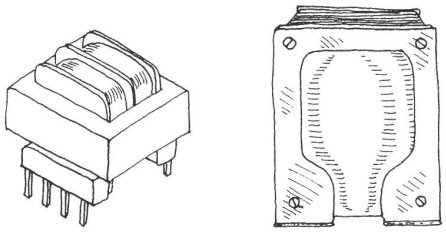
Isolation



High voltage Miniature

This type of transformer acts exclusively as an isolation device; it does not increase or decrease the secondary voltage. The primary coil of an ac power-line isolation transformer (in the United States) accepts a 120-V ac, 60-Hz input voltage and outputs a similar but electrically isolated voltage at the secondary. Such transformers usually come with an electrostatic shield between the primary and secondary. They often come with a three-wire plug and receptacle that can be plugged directly into a power outlet.

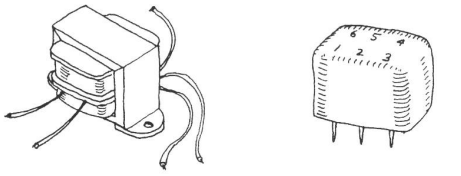
Power Conversion



PCB Mount

These transformers are used primarily to reduce line voltages. They come in a variety of different shapes, sizes, and primary and secondary winding ratios. They often come with taps and multiple secondary windings. Color-coded wires are frequently used to indicate primary and secondary leads (e.g., black wires for primary, green for secondary, and yellow for tap lead is one possibility). Other transformers use pins for primary, secondary, and taped leads, allowing them to be mounted on a PC board.

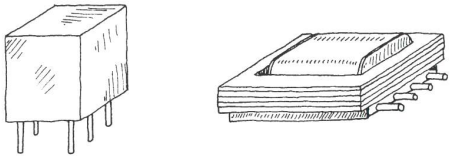
Audio



Chassis-Mount Miniature, Epoxy-Coated

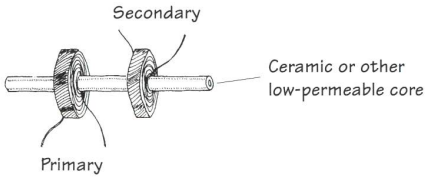
Audio transformers are used primarily to match impedances between audio devices (e.g., between microphone and amplifier or amplifier and speaker). They work best at audio frequencies from 150 Hz to 15 kHz. They come in a variety of shapes and sizes and typically contain a center tap in both the primary and secondary windings. Some use color-coded wires to specify leads (e.g., blue and brown as primary leads, green and yellow as secondary leads, and red and black for primary and secondary taps is one possibility). Other audio transformers have pinlike terminals that can be mounted on PC boards. Spec tables provide dc resistance values for primary and secondary windings to help you select the appropriate transformer for the particular matching application.

Miniature



Miniature transformers are used primarily for impedance matching. They usually come with pinlike leads that can be mounted on PC boards. Spec tables provide dc resistance values for both primary and secondary windings and come with primary/secondary turn ratios.

High-Frequency Transformers



These often come with air or powered-iron cores instead of laminated-iron cores. They are used for high-frequency applications, such as matching RF transmission lines to other devices (e.g., transmission line to antenna). One particularly useful high-frequency rated transformer is the broad-band transmission-line transformer (see the section on matching impedances).

FIGURE 3.73

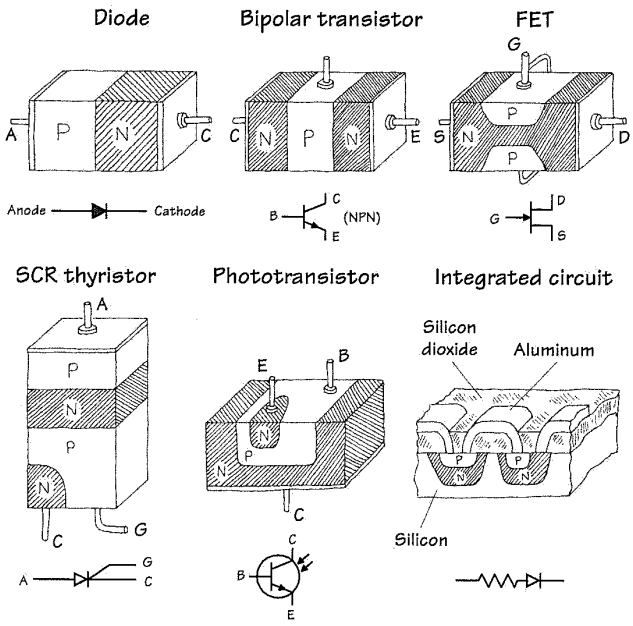


FIGURE 4.9

4.2 Diodes

A *diode* is a two-lead semiconductor device that acts as a one-way gate to electric current flow. When a diode's *anode* lead is made more positive in voltage than its *cathode* lead—a condition referred to as *forward biasing*—current is permitted to flow through the device. However, if the polarities are reversed (the anode is made more negative in voltage than the cathode)—a condition referred to as *reversed biasing*—the diode acts to block current flow.

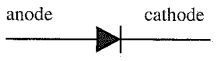


FIGURE 4.10

Diodes are used most commonly in circuits that convert ac voltages and current into dc voltages and currents (e.g., ac/dc power supply). Diodes are also used in voltage-multiplier circuits, voltage-shifting circuits, voltage-limiting circuits, and voltage-regulator circuits.

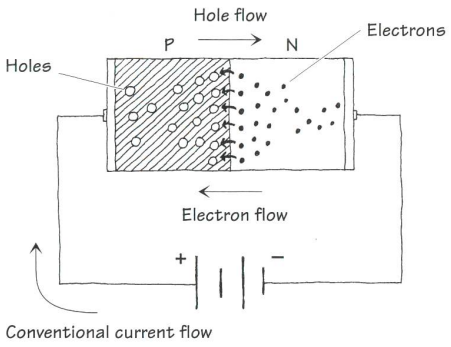
4.2.1 How pn-Junction Diodes Work

A *pn-junction diode* (*rectifier diode*) is formed by sandwiching together *n*-type and *p*-type silicon. In practice, manufacturers grow an *n*-type silicon crystal and then abruptly change it to a *p*-type crystal. Then either a glass or plastic coating is placed around the joined crystals. The *n* side is the cathode end, and the *p* side is the anode end.

The trick to making a one-way gate from these combined pieces of silicon is getting the charge carriers in both the *n*-type and *p*-type silicon to interact in such a way that when a voltage is applied across the device, current will flow in only one direction. Both *n*-type and *p*-type silicon conducts electric current; one does it with electrons (*n*-type), and the other does it with holes (*p*-type). Now the important feature to

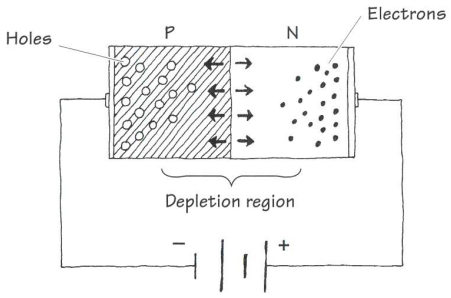
note here, which makes a diode work (act as a one-way gate), is the manner in which the two types of charge carriers interact with each other and how they interact with an applied electrical field supplied by an external voltage across its leads. Below is an explanation describing how the charge carriers interact with each other and with the electrical field to create an electrically controlled one-way gate.

Forward-Biased ("Open Door")



When a diode is connected to a battery, as shown here, electrons from the *n* side and holes from the *p* side are forced toward the center (*pn* interface) by the electrical field supplied by the battery. The electrons and holes combine, and current passes through the diode. When a diode is arranged in this way, it is said to be *forward-biased.*

Reverse-Biased ("Closed Door")



When a diode is connected to a battery, as shown here, holes in the *n* side are forced to the left, while electrons in the *p* side are forced to the right. This results in an empty zone around the *pn* junction that is free of charge carriers, better known as the *depletion region.* This depletion region has an insulative quality that prevents current from flowing through the diode. When a diode is arranged in this way, it is said to be *reverse-biased.*

FIGURE 4.11

A diode's one-way gate feature does not work all the time. That is, it takes a minimal voltage to turn it on when it is placed in forward-biased direction. Typically for silicon diodes, an applied voltage of 0.6 V or greater is needed; otherwise, the diode will not conduct. This feature of requiring a specific voltage to turn the diode on may seem like a drawback, but in fact, this feature becomes very useful in terms of acting as a voltage-sensitive switch. Germanium diodes, unlike silicon diodes, often require a forward-biasing voltage of only 0.2 V or greater for conduction to occur. Figure 4.12 shows how the current and voltage are related for silicon and germanium diodes.

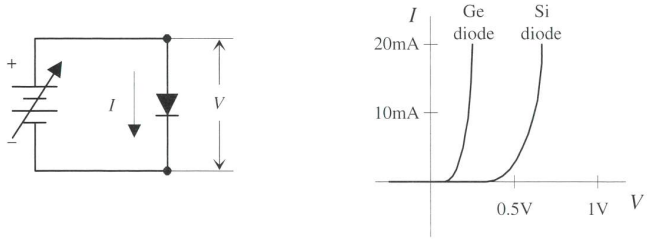


FIGURE 4.12

Another fundamental difference between silicon diodes and germanium diodes, besides the forward-biasing voltages, is their ability to dissipate heat. Silicon diodes do a better job of dissipating heat than germanium diodes. When germanium diodes

get hot—temperatures exceeding 85°C—the thermal vibrations affect the physics inside the crystalline structure to a point where normal diode operation becomes unreliable. Above 85°C, germanium diodes become worthless.

4.2.2 Diode Water Analogy

In the following water analogy, a diode is treated like a one-way gate that has a "forward-biasing spring."

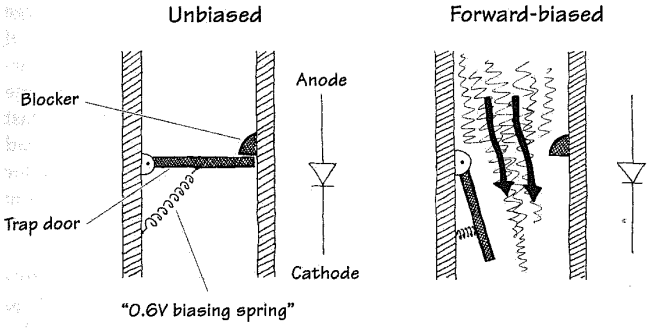


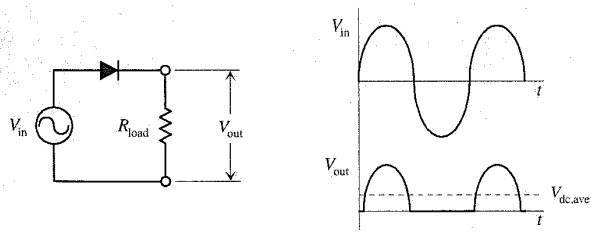
FIGURE 4.13

In this water analogy, a spring holds the one-way gate shut. Water pressure on top of the gate must apply a force large enough to overcome the restoring spring force. This spring force is analogous to the 0.6 V needed to forward bias a silicon diode. A germanium diode with a 0.2-V biasing voltage can be represented with a similar arrangement, but the water analogy will contain a weaker spring, one that can be compressed more easily. Note that if pressure is applied to the bottom of the trap door, a blocker will prevent the trap door from swinging upward, preventing current flow in the upward direction, which is analogous to reverse-biasing a diode.

4.2.3 Basic Applications

Diodes are used most frequently in rectifier circuits—circuits that convert ac to dc. Diodes are used in a number of other applications as well. Here's a rundown of some of the most common applications.

Half-Wave Rectifier



In this circuit, the diode acts to convert an ac input voltage into a pulsed dc output voltage. Whenever the voltage attempts to swing negative at the anode, the diode acts to block current flow, thus causing the output voltage (voltage across the resistor) to go to zero. This circuit is called a *half-wave rectifier,* since only half the input waveform is supplied to the output. Note that there will be a 0.6 V drop across the diode, so the output peak voltage will be 0.6 V less than the peak voltage of V_{in}. The output frequency is the same as the input frequency, and the average dc voltage at the output is 0.318 times zero-to-peak output voltage. A transformer is typically used to step down or step up the voltage before it reaches the rectifier section.

Full-Wave Bridge Rectifier

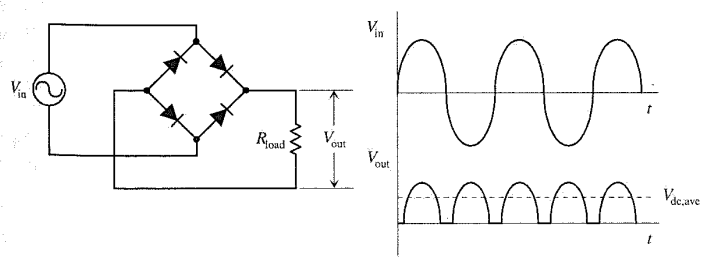
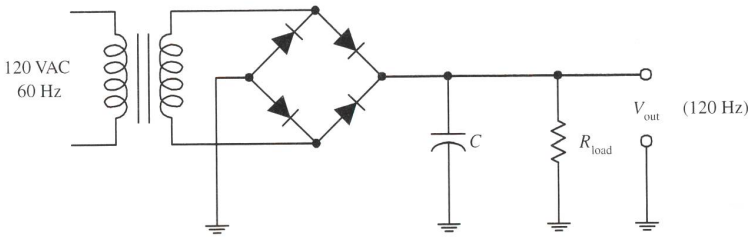
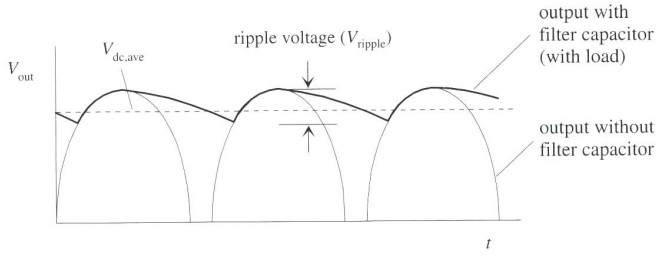


FIGURE 4.14

This circuit is called a *full-wave rectifier,* or *bridge rectifier.* Unlike the half-wave rectifier, a full-wave rectifier does not merely block negative swings in voltage but also converts them into positive swings at the output. To understand how the device works, just follow the current flow through the diode one-way gates. Note that there will be a 1.2-V drop from zero-to-peak input voltage to zero-to-peak output voltage (there are two 0.6-V drops across a pair of diodes during a half cycle). The output frequency is twice the input frequency, and the average dc voltage at the output is 0.636 times the zero-to-peak output voltage.

Basic AC-to-DC Power Supply





By using a transformer and a full-wave bridge rectifier, a simple ac-to-dc power supply can be constructed. The transformer acts to step down the voltage, and the bridge rectifier acts to convert the ac input into a pulsed dc output. A filter capacitor is then used to delay the discharge time and hence "smooth" out the pulses. The capacitor must be large enough to store a sufficient amount of energy to provide a steady supply of current to the load. If the capacitor is not large enough or is not being charged fast enough, the voltage will drop as the load demands more current. A general rule for choosing C is to use the following relation:

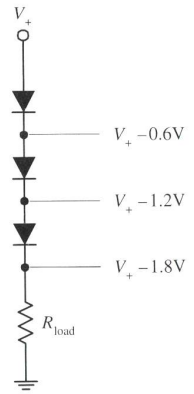
$$R_{load}C >> 1/f$$

where f is the rectified signal's frequency (120 Hz). The ripple voltage (deviation from dc) is approximated by

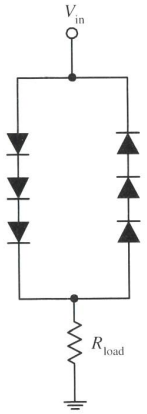
$$V_{ripple} = \frac{I_{load}}{fC}$$

Voltage Dropper

DC application

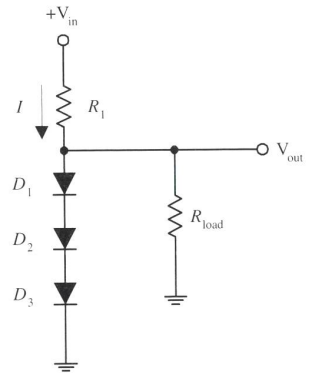


AC application



When current passes through a diode, the voltage is lowered by 0.6 V (silicon diodes). By placing a number of diodes in series, the total voltage drop is the sum of the individual voltage drops of each diode. In ac applications, two sets of diodes are placed in opposite directions.

Voltage Regulator

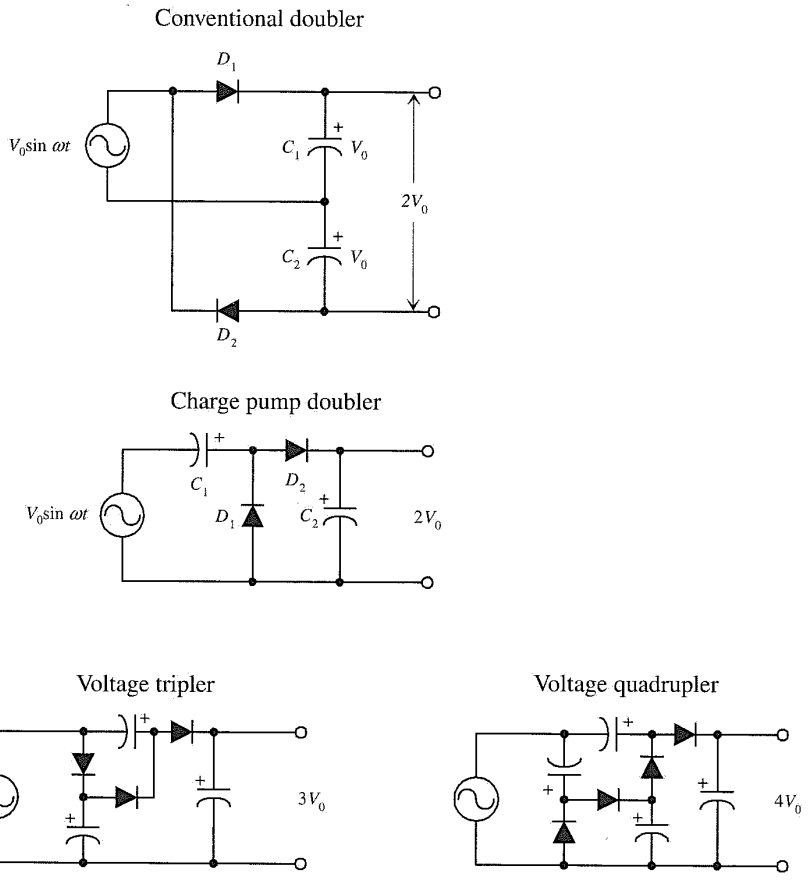


This circuit will supply a steady output voltage equal to the sum of the forward-biasing voltages of the diodes. For example, if D_1, D_2, and D_3 are silicon diodes, the voltage drop across each one will be 0.6 V; the voltage drop across all three is then 1.8 V. This means that the voltage applied to the load (V_{out}) will remain at 1.8 V. R_1 is designed to prevent the diodes from "frying" if the resistance of the load gets very large or the load is removed. The value of R_1 should be approximately equal to

$$R_1 = \frac{(V_{in} - V_{out})}{I}$$

FIGURE 4.14 (*Continued*)

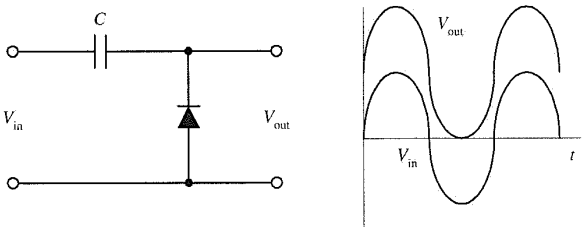
Voltage-Multiplier Circuits



It is often useful to have a rectifier circuit that is capable of producing a dc voltage that is higher than the zero-to-peak voltage of the ac source. Although the usual procedure is to use a transformer to step up the voltage before the rectifier stage, an alternative approach is to use a voltage-multiplier circuit. The top circuit shown here represents a conventional voltage doubler. During a positive half cycle, D_1 is forward biased, allowing C_1 to charge to a dc voltage of V_0 (the peak source voltage). During the negative half cycle, D_2 is forward-biased, and C_2 charges to V_0. Because C_1 and C_2 are connected in series with their polarities aiding, the output voltage is the sum of the capacitors' voltages, or is equal to $2V_0$. The second circuit shown is a variation of the first and is called a *charge pump.* During the negative half cycle, the source pumps charge into C_1 through D_1 with D_2 open-circuited, and then during the positive half cycle, D_1 becomes an open circuit and D_2 becomes a short circuit, and some of the charge in C_1 flows into C_2. The process continues until enough charge is pumped into C_2 to raise its voltage to $2V_0$. One advantage of the charge pump over the conventional voltage doubler is that one side of the source and one side of the output are common and hence can be grounded. To obtain a larger voltage, additional stages can be added, as shown within the bottom two circuits. A serious limitation of voltage-multiplier circuits is their relatively poor voltage regulation and low-current capability.

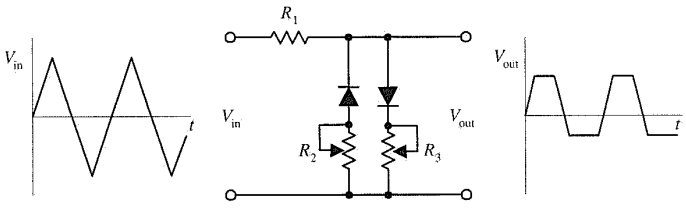
FIGURE 4.14 (*Continued*)

Diode Clamp



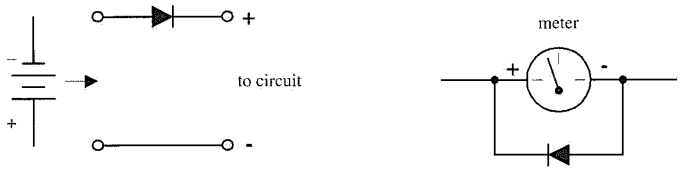
The diode clamp circuit shown here takes a periodic ac signal that oscillates between positive and negative values and displaces it so that it is either always positive or always negative. The capacitor charges up to a dc voltage equal to the zero-to-peak value of V_{in}. The capacitor is made large enough so that it looks like a short circuit for the ac components of V_{in}. If, for example, V_{in} is a sine wave, V_{out} will equal the sum of V_{in} and the dc voltage on the capacitor. By placing the diode in the opposite position (pointing down), V_{out} will be displaced downward so that it is always negative.

Waveform Clipper



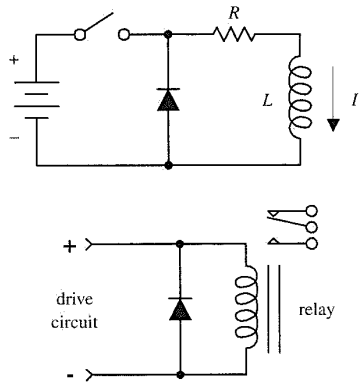
This circuit is often used to protect circuit components from damage due to overvoltage and is used for generating special waveforms. R_2 controls the lower-level clipping at the output, while R_3 controls the upper-level clipping. R_1 is used as a safety resistor to prevent a large current from flowing through a diode if a variable resistor happens to be set to zero resistance.

Reverse-Polarity Protector



A single diode can be used to protect a circuit that could be damaged if the polarity of the power source were reversed. In the leftmost network, the diode acts to block the current flow to a circuit if the battery is accidentally placed the wrong way around. In the rightmost circuit, the diode prevents a large current from entering the negative terminal of a meter.

Transient Protector



Placing a diode in the reverse-biased direction across an inductive load eliminates voltage spikes by going into conduction before a large voltage can form across the load. The diode must be rated to handle the current, equivalent to the maximum current that would have been flowing through the load before the current supply was interrupted. The lower network shows how a diode can be used to protect a circuit from voltage spikes that are produced when a dc-actuated relay suddenly switches states.

FIGURE 4.14 (*Continued*)

Battery-Powered Backup

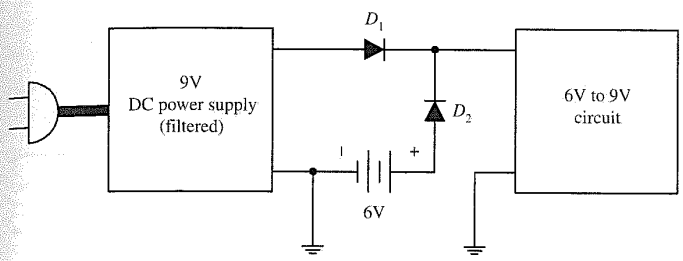


FIGURE 4.14 (*Continued*)

This circuit shows how a battery along with two diodes can be used to provide backup power to a circuit that is normally powered by an ac-driven dc power supply (with filtered output). Here you have a circuit that is designed to run off a voltage from 6 to 9 V. The voltage supplied by the power supply is at 9 V, while the battery voltage is 6 V. As long as power is supplied by the main power supply, the voltage that the circuit receives will be 8.4 V (there is a 0.6-V drop across D_1). At the same time, D_2's anode will be more negative in voltage than its cathode by 2.6 V (8.4 V minus 6.0 V), which means D_2 will block current flow from the battery, hence preventing battery drain. Now, if a lightning bolt strikes, causing the power supply to fail, D_2's anode will become more positive in voltage than its cathode, and it will allow current to flow from the battery to the load. At the same time, D_1 acts to block current from flowing from the battery into the power supply.

4.2.4 Important Things to Know about Diodes

Diodes come in many shapes and sizes. High-current diodes are often mounted on a heat-sink device to reduce their operating temperature. It is possible to place diodes in parallel to increase the current-carrying capacity, but the *VI* characteristics of both diodes must be closely matched to ensure that current divides evenly (although a small resistor can be placed in series with each diode to help equalize the currents).

All diodes have some leakage current (current that gets through when a diode is reverse-biased). This leakage current—better known as the *reverse current* (I_R)—is very small, typically within the nanoampere range. Diodes also have a maximum allowable *reverse voltage, peak reverse voltage* (PRV), or *peak inverse voltage* (PIV), above which a large current will flow in the wrong direction. If the PIV is exceeded, the diode may get zapped and may become permanently damaged. The PIV for diodes varies from a few volts to as much as several thousand volts. One method for achieving an effectively higher PIV is to place diodes in series. Again, it is important that diodes are matched to ensure that the reverse voltage divides equally (although a small resistor placed in parallel with each diode can be used to equalize the reverse voltages).

Other things to consider about diodes include maximum forward current (I_F), capacitance (formed across the *pn* junction), and reverse recovery time.

Most diodes have a 1-prefix designation (e.g., 1N4003). The two ends of a diode are usually distinguished from each other by a mark. For glass-encapsulated diodes, the cathode is designated with a black band, whereas black-plastic-encapsulated diodes use a white band (see Fig. 4.15). If no symbols are present (as seen with many power diodes), the cathode may be a boltlike piece. This piece is inserted through a heat-sink device (piece of metal with a hole) and is fastened down by a nut. A fiber or mica washer is used to isolate the cathode electrically from the metal heat sink, and a special silicone grease is placed between the washer and heat sink to enhance thermal conductivity.

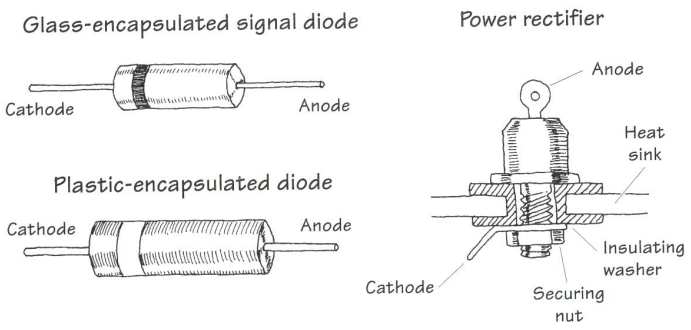


FIGURE 4.15

4.2.5 Zener Diodes

A zener diode is a device that acts as a typical *pn*-junction diode when it comes to forward biasing, but it also has the ability to conduct in the reverse-biased direction when a specific breakdown voltage (V_B) is reached. Zener diodes typically have breakdown voltages in the range of a few volts to a few hundred volts (although larger effective breakdown voltages can be reached by placing zener diodes in series). Figure 4.16 shows the symbol for a zener diode and the forward and reverse current characteristics as a function of applied voltage.

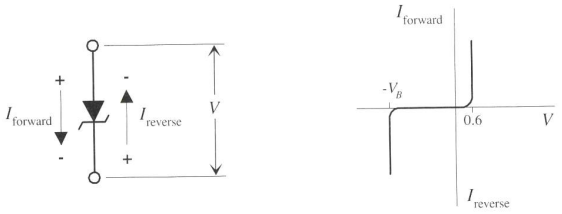


FIGURE 4.16

4.2.6 Zener Diode Water Analogy

In the following water analogy, a zener diode is treated like a two-way gate that has a "forward-biasing spring" and a stronger "reverse-biasing spring."

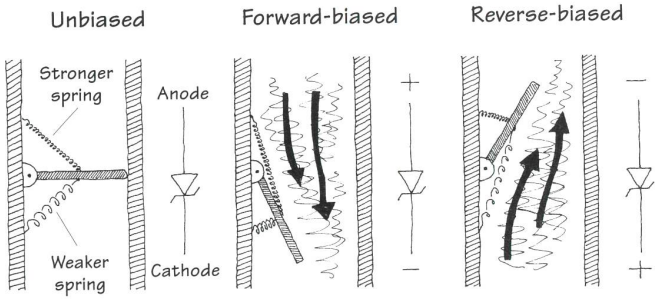


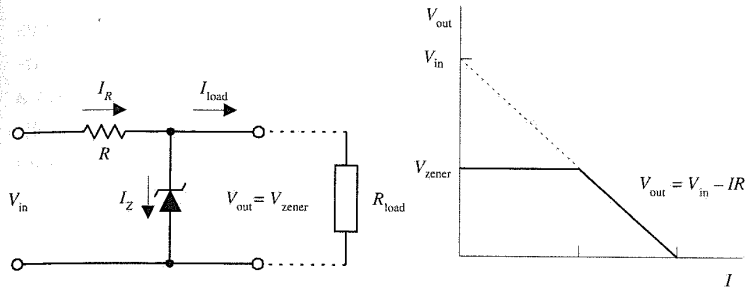
FIGURE 4.17

This water analogy is much like the water analogy used for the *pn*-junction diode. The only difference is that there is no block to prevent the gate from swinging in the "reverse-biasing direction." Instead, a "reverse-biasing spring" is used to hold the gate shut. If the water pressure is applied in the reverse-biasing direction, the gate will only open if the pressure is greater than the force applied by the reverse-biasing spring. Placing a stronger reverse-biasing spring into the system is analogous to choosing a zener diode with a larger reverse-biasing voltage.

4.2.7 Basic Applications for Zener Diodes

Zener diodes are used most frequently in voltage-regulator applications. Here are a few applications.

Voltage Regulator



$$I_{load} = \frac{V_{in}}{R_{load}} \approx \frac{V_{zener}}{R_{load}}$$

$$I_R = \frac{V_{in} - V_{out}}{R} \approx \frac{V_{in} - V_{zener}}{R}$$

$$I_{zener} = I_R - I_{load}$$

Here, a zener diode is used to regulate the voltage supplied to a load. When V_{in} attempts to push V_{out} above the zener diode's breakdown voltage (V_{zener}), the zener diode draws as much current through itself in the reverse-biased direction as is necessary to keep V_{out} at V_{zener}, even if the input voltage V_{in} varies considerably. The resistor in the circuit is used to limit current through the zener diode in case the load is removed, protecting it from excessive current flow. The value of the resistor can be found by using

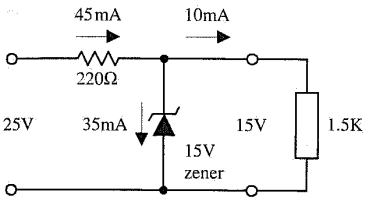
$$R = \frac{V_{in} - V_{zener}}{I_{max,zener}}$$

where $I_{max,zener}$ is the maximum current rating of the zener diode. The power rating of the resistor should be

$$P_R = IV_R = I_{max,zener}(V_{in} - V_{zener})$$

I_{load}, I_R, and I_{zener} are given by the formulas accompanying the figure.

Example 1

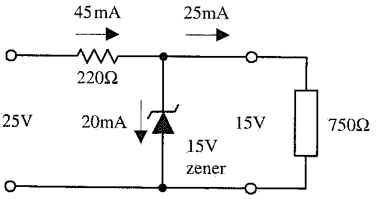


$$I_{load} = \frac{15V}{15K} = 10mA$$

$$I_R = \frac{25V - 15V}{220\Omega} = 45mA$$

$$I_{zener} = 45mA - 10mA = 35mA$$

Example 2

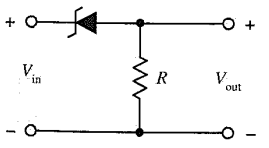


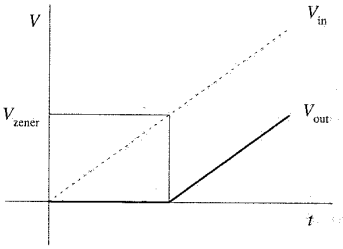
$$I_{load} = \frac{15V}{750\Omega} = 20mA$$

$$I_R = \frac{25V - 15V}{220\Omega} = 45mA$$

$$I_{zener} = 45mA - 20mA = 25mA$$

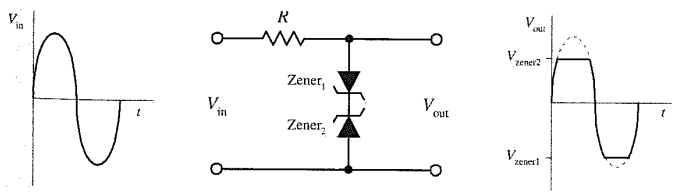
Voltage Shifter





This circuit shifts the input voltage (V_{in}) down by an amount equal to the breakdown voltage of the zener diode.

Waveform Clipper



Two opposing zener diodes placed in series act to clip both halves of an input signal. Here, a sine wave is converted to a near square wave. Besides acting as a signal reshaper, this arrangement also can be placed across a power supply's output terminals to prevent voltage spikes from reaching a circuit that is attached to the power supply. The breakdown voltages of both diodes must be chosen according to the particular application.

FIGURE 4.18

4-bit even-parity error-detection system

Even vs. odd parity (examples)

4-bit string	parity bit	parity
1 0 0 1	0	even (two 1's)
1 1 0 0	1	odd (three 1's)
0 0 0 0	1	odd (one 1's)
1 1 1 0	1	even (four 1's)

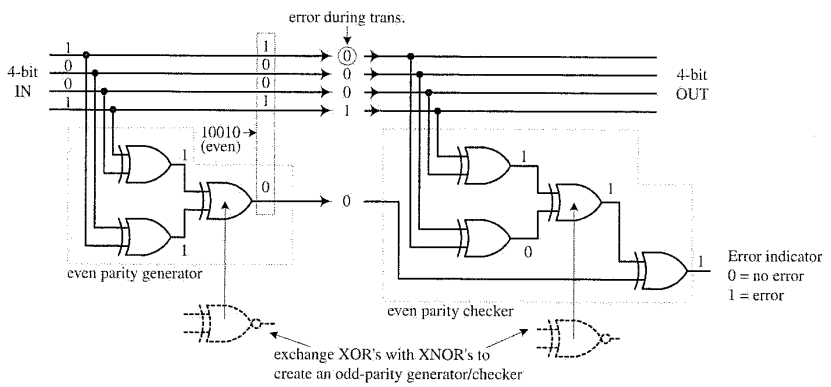
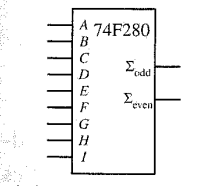


FIGURE 12.51

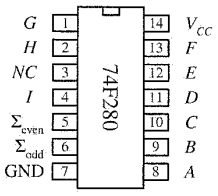
If you want to avoid building parity generators and checkers from scratch, use a parity generator/checker IC like the 74F280 9-bit odd-even parity generator/checker shown below. To make a complete error-detection system, two 74F280s are used—one acts as the parity generator; the other acts as the parity checker. The generator's inputs *A* through *H* are connected to the eight data lines of the transmitting portion of the circuit. The ninth input (*I*) is grounded when the device is used as a generator. If you want to create an odd-parity generator, you tap the Σ_{odd} output; for even parity, you tap Σ_{even}. The 74F280 checker taps the main line at the receiving end and also accepts the parity bit line at input *I*. The figure below shows an odd-parity error-detection system used with an 8-bit system. If an error occurs, a high (1) is generated at the Σ_{odd} output.

74F280 9-bit odd-even parity generator/checker



# of HIGH's (1's) applied to I_0-I_8	Σ_{odd}	Σ_{even}
even	1	0
odd	0	1

IC Package



Odd parity error-dection used in an 8-bit system

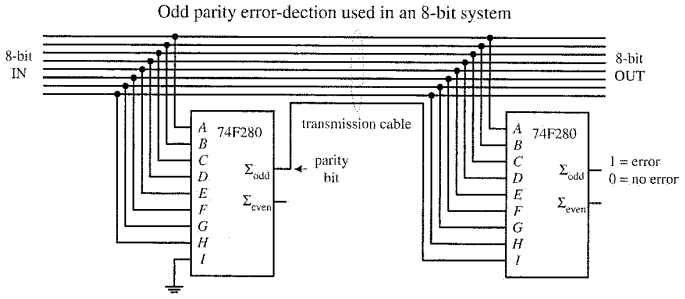


FIGURE 12.52

12.3.9 A Note on Obsolescence and the Trend Toward Microcontroller Control

You have just covered most of the combinational devices you will find discussed in textbooks and find listed within electronic catalogs. Many of these devices are still used. However, some devices such as the binary adders and code converters are becoming obsolete.

Today, the trend is to use software-controlled devices such as microprocessors and microcontrollers to carry out arithmetic operations and code conversions. Before

you attempt to design any logic circuit, I suggest jumping to Section 12.12. In that section, pay close attention to microcontrollers. These devices are quite amazing. They are essentially microprocessors but are significantly easier to program and are easier to interface with other circuits and devices.

Microcontrollers can be used to collect data, store data, and perform logical operations using the input data. They also can generate output signals that can be used to control displays, audio devices, stepper motors, servos, etc. The specific functions a microcontroller is designed to perform depend on the program you store in its internal ROM-type memory. Programming the microcontroller typically involves simply using a special programming unit provided by the manufacturer. The programming unit usually consists of a special prototyping platform that is linked to a PC (via a serial or parallel port) that is running a host program. In the host program, you typically write out a program in a high-level language such as C, or some other specialized language designed for a certain microcontroller, and then, with the press of a key, the program is converted into machine language (1s and 0s) and downloaded into the microcontroller's memory.

In many applications, a single microcontroller can replace entire logic circuits comprised of numerous discrete components. For this reason, it is tempting to skip the rest of the sections of this chapter and go directly to the section on microcontrollers. However, there are three basic problems with this approach. First, if you are a beginner, you will miss out on many important principles behind digital control that are most easily understood by learning how the discrete components work. Second, many digital circuits that you can build simply do not require the amount of sophistication a microcontroller provides. Finally, you may feel intimidated by the electronics catalogs that list every conceivable component available, be it obsolete or not. Knowing what's out there and knowing what to avoid are also important parts of the learning process.

12.4 Logic Families

Before moving on to sequential logic, let's touch on a few practical matters regarding the various logic families available and what kind of operating characteristics these families have. In this section you will also encounter unique logic gates that have open-collector output stages and logic gates that have Schmitt-triggered inputs.

The key ingredient within any integrated logic device, be it a logic gate, a multiplexer, or a microprocessor, is the transistor. The kinds of transistors used within the integrated circuit, to a large extend, specify the type of logic family. The two most popular transistors used in ICs are bipolar and MOSFET transistors. In general, ICs made from MOSFET transistors use less space due to their simpler construction, have very high noise immunity, and consume less power than equivalent bipolar transistor ICs. However, the high input impedance and input capacitance of the MOSFET transistors (due to their insulated gate leads) results in longer time constants for transistor on/off switching speeds when compared with bipolar gates and therefore typically result in a slower device. Over years of development,

APPENDIX K

Microprocessors and Microcontrollers

Within almost every mildly complex electronic gadget you find these days, there is a microprocessor or microcontroller running the show. Also, within these gadgets you will find practically none of the discrete logic ICs (e.g., logic gates, flip-flops, counters, shift registers, etc.) that we covered earlier in this book. Now, before doing anything else, let's stop and address the second statement above, since it seems to go against everything we learned earlier.

When writing a book on electronics geared toward beginners, it is often necessary to introduce somewhat obsolete devices and techniques in order to illustrate some important principle. A perfect example is the logic gates and the flip-flop. These discrete devices are hardly ever used, but they provide the beginner with a basic understanding of logic states, logical operations, memory, etc. To illustrate more complex principles, such as counting, the earlier principles (and the devices that go with them) are used. This rather historical approach of doing electronics is important foundation building; however, it should not be dwelt on if you aim is to design complex electrical gadgets that mesh with today's technological standards.

What should be dwelt on is the *microcontroller*. This device is truly one of the great achievements of modern electronics. With a single microcontroller it possible to replace entire logic circuits comprised of discrete devices. The microcontroller houses an onboard central processing unit (CPU) that performs the same basic function as a computer's microprocessor (performs logical operations, I/O control operations, etc.). It also houses extra onchip goodies such as a ROM, a RAM, serial communication ports, often A/D converters, and more. In essence, microcontrollers are minicomputers without the keyboard and monitor. With a single microcontroller you can build a robot that performs various functions. For example, you can use the microcontroller to control servo motors, generate sound via a speaker, monitor an infrared sensor to avoid objects, record input data generated by analog transducers, etc. Microcontrollers are also used within microwaves, TVs, VCRs, computer peripherals (laser printers, disk drives), automobiles control systems, security systems, toys, environmental data logging instruments, cellphones, and any device that requires programlike control.

Now *microprocessors*, such as the Intel's Pentium processors, are similar to microcontrollers but are designed primarily for fast-paced "number crunching," which is important for running today's sophisticated multimedia programs and games. Microprocessors also require mass amounts of additional support devices, such as RAM, ROM, I/O controllers, etc.—something the microcontroller has built in. For this reason, we will only briefly introduce microprocessors in this appendix and instead focus mainly microcontrollers. Microcontrollers are definitely the "in thing" these days, mainly because they are so easy to use and can be used as the "brains" within so many different battery-powered gadgets. Microprocessors, on the other hand, are mostly used within computers and tend not to have much direct practical use to the inventor who likes to tinker with actual IC's.

K.1 Introduction to Microprocessors

Microprocessors are integrated circuits which through address lines, data lines, and control lines have the ability to read and execute instructions from external read-only memory (ROM), hold temporary data and programs in external random-access memory (RAM) by writing and reading, and receive and supply input and output signals to and from external support circuitry. Figure K.1 shows a simple microprocessor-based system with support circuitry.

Simple microprocessor-based system

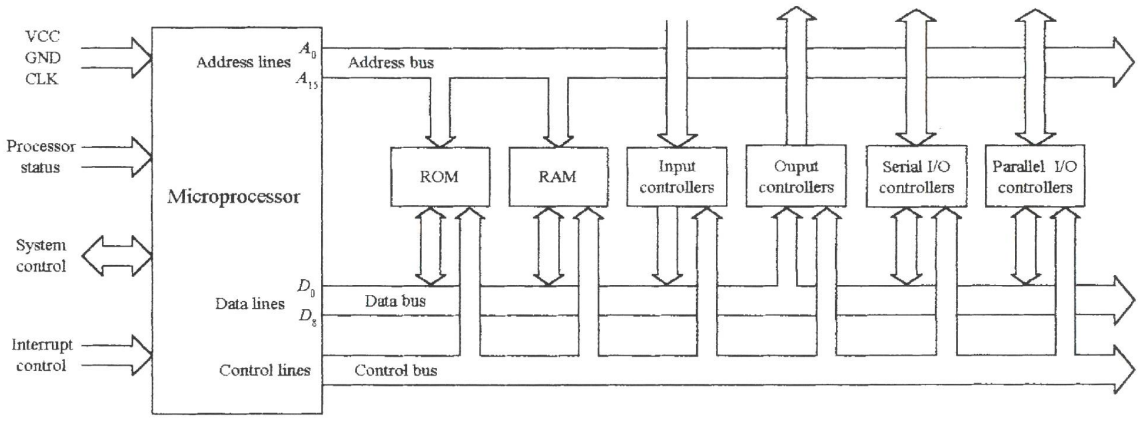


FIGURE K.1

K.1.1 The Microprocessor

At the heart of a microprocessor-based system is the microprocessor or central processing unit (CPU). The microprocessor is responsible for performing computations on chunks of data that are organized into words. In this particular microprocessor-based system, the CPU consists of an 8-bit processor—one that works with 8-bit words. In more sophisticated microprocessor-based systems, the processors work with larger bit words, such as 16-bit, 32-bit, or higher (e.g., 64-bit Pentium processor). The processor's role is to read program instructions from memory and then execute those instructions by supplying the three external buses with the proper levels and timing to make the connected devices (memory, I/O controllers, etc.) perform their specific operations. Within the processor itself, there are a number of integrated sub-

sections designed to perform specific tasks. One of these subsections is the instruction decoder, which accepts and interprets instructions fetched from memory and then figures out what steps to take based on the fetched instructions. The processor also contains an arithmetic logic unit (ALU) that can perform the instructed operations, such as add, complement, compare, shift, move, etc., on quantities stored on chip registers or in external memory. A program counter keeps track of the current location in the executing program. There are many other subsections within a processor—we'll take a closer look at an actual microprocessor in a second.

K.1.2 Address Bus, Data Bus, and Control Bus

There are basically three buses that are used to relay address, data, and control information between the microprocessor and external devices, such as RAM, ROM, and various I/O controllers. These include the address bus, the data bus, and the control bus. (Recall that a *bus* is a group of conductors shared by various devices.)

The address bus is used by the microprocessor to select a particular address location within an external device, such as memory. In our processor-based system, the address bus is 16 bits wide, which means that the processor can access 2^{16} (65,536) different addresses.

The data bus is used to transfer data between the processor and external devices (memory, peripherals). The data bus for our microprocessor-based system is 8 bits wide. With more sophisticated processors, such as the early 486 and the later Pentium processors, the number of data lines is larger—32 and 64 bits, respectively.

The control bus is of varying width, depending on the microprocessor being used. It carries control signals that are tapped into by other ICs to specify what operation is being performed. From these signals, the ICs can tell if the operation is a read, write, interrupt, I/O, memory access, or some other operation.

K.1.3 Memory

Computers typically use three types of memory: ROM, RAM, and mass memory (e.g., hard drives, floppies, CD-ROMs, ZIP drives, etc.) ROM acts as a nonvolatile memory used to store the boot-up sequence of instructions, which include port allocations, interrupt initializations, and codes needed to get the operating system read from the hard drive. RAM is used to hold temporary data and programs used by the microprocessor. Mass memory is used for long-term data storage.

K.1.4 Input and Output Controllers

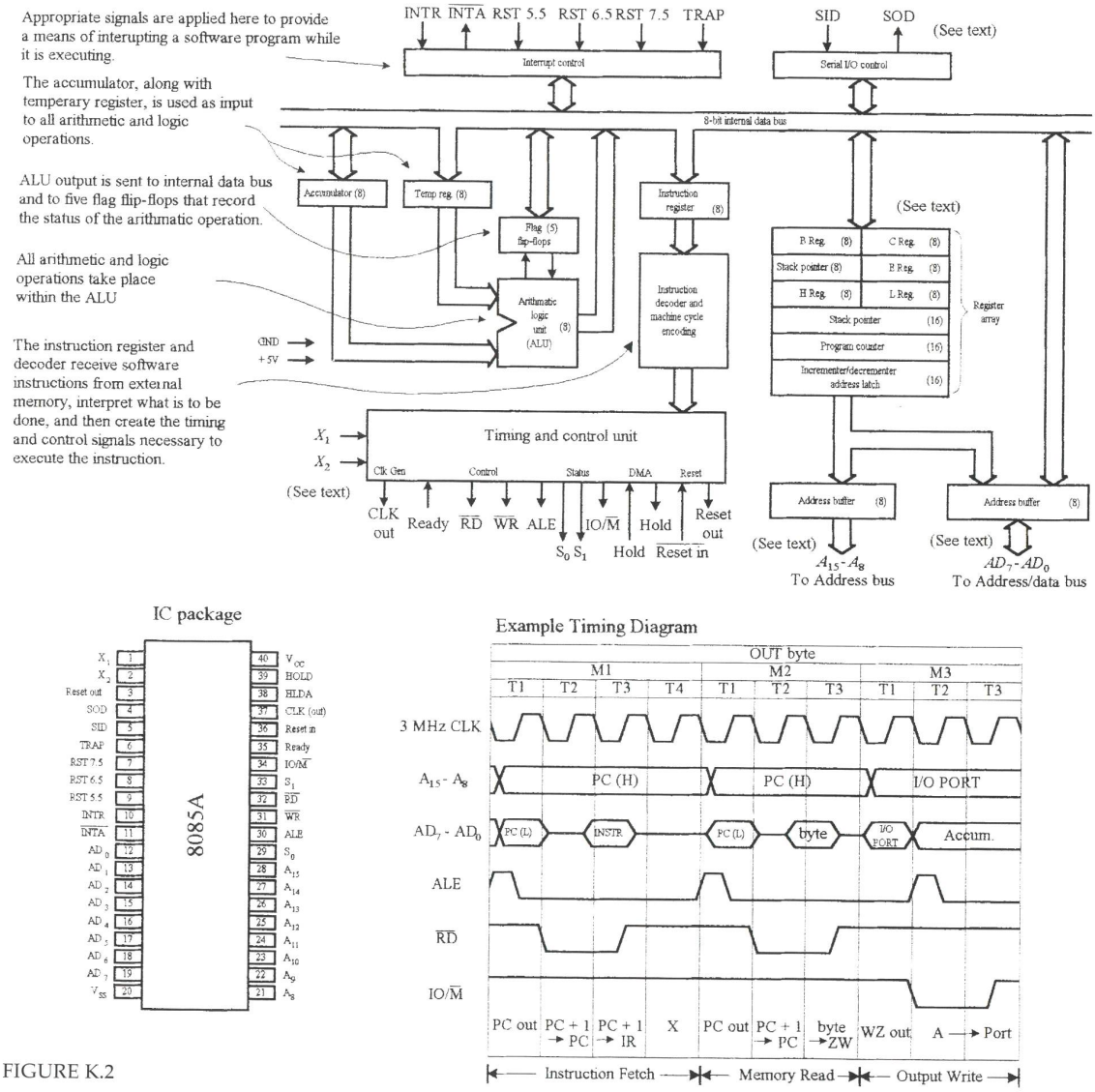
In order for the microprocessor to receive data from or send data to various input/output devices, such as keyboards, displays, printers, etc., special I/O controllers are needed as an interface. The controller is linked to the bus system and is controlled by the microprocessor. In order for the microprocessor to keep tabs on the various I/O devices, the microprocessor can either directly address the controller by means of program instructions or have the controller issue an interrupt signal when the external device generates a "read me" type of signal. Computers come with all sorts of controllers, from sound controllers, SCSI controllers, universal serial bus controllers, game controllers, floppy-disk controllers, hard-disk controllers, etc. All of

these you purchase from the manufacturer, or they are already included in the computer. Different hardware requires different control signals. For this reason, software drivers (special programs) designed to create a uniform programming interface to each particular piece of hardware are used.

K.1.5 Sample Microprocessor

To get a basic idea of what the interior of a simple processor looks like, let's examine Intel's 8085A processor. Though this chip is obsolete by today's standards, it shares many core features with the modern microprocessors and is much easier device to understand.

Intel's 8085A microprocessor : functional block diagram, IC, timing



IC package

Example Timing Diagram

FIGURE K.2

ALU

The 8085A, as with all processors, contains an arithmetic logic unit (ALU) comprised of gates that perform fundamental arithmetic and logical operations (+, –, ×, /, OR, AND, NOT, etc.), including comparisons (=, <, <=, >, <=), as well as shifting data values sent to it from memory. Support for these operations is supplied by one or more ALU registers, called *accumulators*, which can receive initial values from memory, hold the cumulative results of the arithmetic and logic operations, and transmit the final result back to memory. A group of binary indicators (or flags) associated with the ALU provides control feedback information in the form of a summary of the ALU's status after each operation. This might typically include whether the result produced was positive or negative and zero or nonzero, respectively.

Control Unit

The control unit is the directing element within the system, having the responsibility of executing the stored program sequences. It performs this role by repeatedly following a basic instruction execution cycle for each program instruction. It first fetches the instruction from the main store memory into a special control unit register called the *instruction register.* The instruction is then decoded (separated) into parts indicating what is to be done (the operation part) and what information is involved (the operand parts). After that, it executes the instruction by sending the appropriate control signals to the ALU, I/O, and memory. This is referred to as a *fetch-decode-execute cycle.*

Interrupts

Another important feature in microprocessor systems is interrupt control, which provides a way for external digital signals to interrupt a software program while it is running. There are five interrupt inputs for the 8085A: INTR, RST, RST 5.5, RST 6.5, RST 7.5, and TRAP. The interrupts are arranged in a fixed priority that determines which interrupt is to be recognized if more than one is pending. The order of priority is as follows: TRAP (highest priority), RST 7.5 (second highest), RST 6.5 (third highest), RST 5.5 (four highest), and INTR (lowest priority). The TRAP interrupt is useful for catastrophic events such as power failure or bus error.

Address and Data Signals

HIGH ADDRESS (A_{15}–A_8) The higher-order 8 bits of a 16-bit address.

ADDRESS/DATA (AD_7–AD_0) The lower-order 8 bits of 16-bit address or 8 bits of data. Note that this address scheme requires multiplexing—a feature used to reduce pin count.

SERIAL OUTPUT DATA (SID) A single-bit input used to accommodate devices that transmit serially (one bit at a time).

SERIAL OUTPUT DATA (SOD) A single-bit output to accommodate devices that receive serially.

Timing and Control Signals

CLK (OUT) The system clock output that is sent to peripheral chips and is used to synchronize their timing.

X_1, X_2 Signals applied here come from an external crystal or other device used to drive the internal clock generator.

ADDRESS LATCH ENABLE (ALE) This signal occurs during the first clock state of a machine cycle and enables the address to get latched into onchip latch peripherals. This allows the address module (e.g., memory, I/O) to recognize that it is being addressed.

STATUS (S_0, S_1) Control signals used to indicate whether a read or write operation is taking place.

I/O/M Used to enable either I/O or memory modules for read and write operations.

READ CONTROL (RD) Indicates that the selected memory or I/O module is to be read and that data bus is available for data transfer.

WRITE CONTROL (WR) Indicates that data on the data bus are to be written into the selected memory or I/O location.

Memory and I/O Symbols

HOLD Used to request that CPU relinquish control and use of the external system bus. The CPU will complete execution of the instruction presently in the instruction register and then enter a hold state, during which no signals are inserted by the CPU to the control, address, or the data buses.

HOLD ACKNOWLEDGE (HLDA) Output signal from control unit that acknowledges the hold signal and indicates that the bus is now available.

READY Used to synchronize the CPU with slower memory or I/O devices. When an address device asserts ready, the CPU may proceed with an input (DB_{in}) or (*WR*) operation; otherwise, the CPU enters a wait state until the device is ready.

CPU Initialization

RESET IN Causes the contents of the program counter to be set to zero. The CPU resumes execution at location zero.

RESET OUT Acknowledges that the CPU has been reset. The signal can be used to reset the system.

General-Purpose Registers

The 8085A contains a set of six general-purpose 8-bit registers: B, C, D, E, H, and L. These registers are called *general-purpose* because they can be used in any manner deemed by the individual programming of the microprocessor. The general-purpose registers can hold numeric data, BCD data, ASCII data, or any other type of information required. They can be used as six 8-bit registers or as three 16-bit register pairs (BC, DE, HL). Register pairs hold 16-bit numeric data or any other 16-bit coded information. Besides holding 16 bits of data, register pairs also address memory data. A memory address, placed into a register pair, allows the contents of the location's address by the register pair to be manipulated. Register pair memory addressing is called *indirect addressing*.

Special-Purpose Registers

Special-purpose registers accumulate results from arithmetic and logic operations and provide "housekeeping" for the microprocessor. Housekeeping registers are not

normally programmed with an instruction but instead are used by the microprocessor. Special-purpose registers for the 8085A include an accumulator, flag register, program counter, and stack pointer.

All microprocessors contain an accumulator register that accumulates the answers of most arithmetic and logic operations carried out by the ALU.

Flag registers used in the 8085A contain 5 bits used as flags or indicators for the ALU. The flags change whenever the 8085A executes most arithmetic operations, and they are used to indicate the status of the arithmetic operation. The 5 flag bits are broken down as follows: (1) one sign flag bit that is used to tell whether a result of an arithmetic or logic operation is positive or negative, (2) one zero flag bit that is used to indicate whether the result of an ALU operation is zero or nonzero, (3) one auxiliary carry flag bit that holds any carry that occurs between the least significant and most significant half-byte result from the ALU, (4) one parity flag bit that shows the parity of the result from the ALU, (5) one carry flag bit that holds any carry that occurs out of the most significant bit of the accumulator after an addition and also holds a borrow after a subtraction.

The program counter does not counter programs but rather locates the next software instruction to be executed by the processor. It counts through the memory locations beginning at lower addresses and progressing toward higher addresses.

The stack pointer stores the address of the last entry on the stack. The stack is a data storage area in RAM used by certain processor operations.

Timing Diagram for the 8085A

Figure J.2 shows an example of 8085A timing. In essence, three machine cycles (M1, M2, M3) are needed. During the first cycle, an OUT instruction is fetched, while during the second cycle, the second half of the instruction is fetched, which contains the number of I/O devices selected for output. During the third machine cycle, the contents of the accumulator are written out to the selected device via the data bus. At the beginning of each machine cycle, an address-latch-enable (ALE) pulse is supplied by the control unit that alerts external circuits. During timing state T1 of machine cycle M1, the control unit sets the IO/M signal to indicate that a memory operation is to occur. Also during this cycle, the control unit instructs the program counter to place its contents on the address bus (A_{15}–A_8) and address/data bus (AD_7–AD_0). During timing state T2, the address memory module places the contents of the addressed memory location on the address/data bus. The control unit sets the read control (*RD*) signal to indicate a read, but it waits until T3 to copy the data from the bus. This gives the memory time to put the data on the bus and for the signal levels to stabilize. The final state, T4, is a bus idle state during which the CPU decodes the instructions. The remaining machine cycles proceed in a similar manner.

K.1.6 Programming a Microprocessor

Each microprocessor has its own unique set of instructions for performing tasks such as reading from memory, adding numbers, and manipulating data. For example, the Intel's Pentium processor found in IBM compatibles use a completely different set of instructions from the Motorola's PowerPC found in MacIntosh computers.

The actual language used by any given microprocessor is machine code, which consists of 1's and 0's. Directly programming a microprocessor in machine language,

however, is never done—it is too hard. A simple task such as multiplying two numbers may take hundreds of machine instructions to accomplish. Keeping track of all the columns of binary numbers and making sure that each column is bit-perfect would be a nightmare.

A simple alternative to machine code is to first write out a program in assembly language, which uses mnemonic abbreviations and symbolic names for memory locations and variables. The conversion from assembly language to machine language involves translating each mnemonic into the appropriate hexadecimal machine code (called *operation code*, or *opcode* for short) and storing the codes in specific memory locations, which are also given hexadecimal address numbers. This can be done by a software package called an *assembler* provided by the microprocessor manufacturer or can be done by the programmer by looking up the codes (also given by manufacturer) and memory addresses, which is called *hand assembly*. When using hand assembly, you must determine which memory locations (within ROM) are to be reserved for the program. Assembly language allows the programmer to write the most streamlined, memory-efficient programs, which lead to very fast execution times. However, as with machine code, programming in assembly language is also tedious.

To make life easier on the programmer, a compiled or interpreted high-level language such as BASIC, PASCAL, or C can be used. Using one of these languages, you write out you program within algebraic commands and user-friendly control commands such as if . . . then, for . . . next, etc. When using these languages, you do not have to worry about addressing memory locations or figuring out which bits get shifted into which registers, etc. You simply declare variables and write out arithmetic and logic statements. This is referred to as *source code*. After the source code is created, there are two possible routes that must be taken before the microprocessor can run the program. One route involves compiling the source code, while the other route involves interpreting the source code. Languages such as C, PASCAL, and FORTRAN require *compiling*, a process in which a language compiler converts the source-code statements into assembly code. Once the assembly code is created, an assembler program converts the assembly code into machine code that the microprocessor can use. A language such as BASIC, on the other hand, is an interpreted language. Instead of compiling an assembly-language program from the source program, an interpreter program looks at the statements within the program and then executes the appropriate computer instructions on the spot. In general, interpreted languages run much more slowly than complied languages, but there is no need for compiling and no delay after entering a program before it is run. We'll talk more about compilers and interpreters later on.

Table K.1 provides codes in BASIC, assembly (8085A), and machine (8085A) languages that count from 5 down to 0. Note that the assembly and machine codes are specific to the 8085A. In BASIC, the code is pretty much self-explanatory. The variable COUNT holds the counter value, and line 30 checks the COUNT to see if it is equal to zero. If so, then the program jumps back to the beginning; otherwise, it jumps back to line 20 (subtract 1 from COUNT) and checks the COUNT again.

The 8085A's assembly-language mnemonics for this program include MVI, DCR, and START. A complete listing of the 8085A's mnemonics is given in Table K.2. The first mnemonic, MVI, means "Move Immediate." The instruction MVI A, 05H will

TABLE K.1 Counting from 5 to 0 Using BASIC, Assembly Language, and Machine Language

BASIC LANGUAGE		8085A ASSEMBLY LANGUAGE		8085A MACHINE LANGUAGE	
LINE	INSTRUCTION	LABEL	INSTRUCTION	ADDRESS (HEX)	CONTENTS
10	COUNT = 5	START:	MVI A,05H	4000	3E (opcode)
20	COUNT = COUNT–1			4001	05 (data)
30	IF COUNT = 0	LOOP:	DCR A	4002	3D (opcode)
	THEN GOTO 10		JZ START	4003	CA (opcode)
40	GOTO 20			4004 4005	00 40 } (address)
			JUMP LOOP	4006	C3 (opcode)
				4007 4008	02 40 } (address)

move the data value of 05 (the H stands for hexadecimal) into register A (accumulator). The next instruction, DCR A, decrements register A by 1. The third instruction, JZ START, is a conditional jump. The condition that this statement is looking for is the zero condition. As the register is decremented, if A equals 0, then a flag bit, called a *zero flag*, gets set (set to logic 0). The instruction JX START is interpreted as jump to statement label START if the zero flag is set. If this condition is not met, then the program jumps to the next instruction, JMP LOOP, which is an unconditional jump. This instruction is interpreted as jump to label LOOP regardless of any condition flags.

In terms of machine code, the assembly-language instruction MVI A,05H requires 2 bytes to complete. The first byte is the opcode, 3E (binary 0011 1110), which identifies the instruction for the microprocessor. They second byte (referred to as the *operand*) is the data value 05. The opcode for JZ is CA (binary 1100 1010), while the following 16-bit (2-byte) address to jump to if the condition (zero) is not met. This makes it a 3-byte instruction; byte 2 of the instruction (location 4004 hex) is the lower-order byte of the address, and byte 3 is the higher-order byte of the address to jump to. The opcode for JMP is C3 (binary 1100 0011) and must be followed by a 16-bit (2-byte) address specifying the location to jump to.

K.2 Microcontrollers

As you can see from what we have covered until now, programming a microprocessor (even a fairly simple one), as well as linking all the external support chips, such as the RAM, ROM, I/O controllers, etc., is difficult work. It requires burning startup instructions into ROM, connecting various I/O devices to the buses, writing programs in order to communicate with the I/O devices, understanding interrupt protocols, etc. In reality, microprocessor systems, such as those found within personal computers, are usually too difficult to construct or are simply not worth the effort; you can buy fully assembled units or at least buy the motherboard and then attach memory modules (DIMMs, SIMMs), sound cards, disk drives, etc. In the latter case,

TABLE K.2 Instruction Set Summary for the Intel 8085A Microprocessor

MNEMONIC	OPCODE	SZAPC	~S	DESCRIPTION	NOTES
ACI n	CE	****	7	Add with Carry Immediate	A = A + n + CY
ADC r	8F	****	4	Add with Carry	A = A + r + CY (21X)
ADC M	8E	****	7	Add with Carry to Memory	A = A + [HL] + CY
ADD r	87	****	4	Add	A = A + r (20X)
ADD M	86	****	7	Add to Memory	A = A + [HL]
ADI n	C6	****	7	Add Immediate	A = A + n
ANA r	A7	***0	4	AND Accumulator	A = A&r (24X)
ANA M	A6	***0	7	AND Accumulator and Memory	A = A&[HL]
ANI n	E6	**0*0	7	AND Immediate	A = A&n
CALL a	CD	-----	18	Call unconditional	–[SP] = PC, PC = a
CC a	DC	-----	9	Call on Carry	If CY = 1 (18~s)
CM a	FC	-----	9	Call on Minus	If S = 1 (18~s)
CMA	2F	-----	4	Complement Accumulator	A = ~A
CMC	3F	----*	4	Complement Carry	CY = ~CY
CMP r	BF	****	4	Compare	A – r (27X)
CMP M	BF	****	7	Compare with Memory	A – [HL]
CNC a	D4	-----	9	Call on No Carry	If CY = 0 (18~s)
CNZ a	C4	-----	9	Call on No Zero	If Z = 0 (18~s)
CP a	F4	-----	9	Call on Plus	If S = 0 (18~s)
CPE a	EC	-----	9	Call on Parity Even	If P = 1 (18~s)
CPI n	FE	****	7	Compare Immediate	A – n
CPO a	E4	-----	9	Call on Parity Odd	If P = 0 (18~s)
CZ a	CC	-----	9	Call on Zero	If Z = 1 (18~s)
DAA	27	****	4	Decimal Adjust Accumulator	A = BCD format
DAD B	09	----*	10	Double Add BC to HL	HL = HL + BC
DAD D	19	----*	10	Double Add DE to HL	HL = HL + DE
DAD H	29	----*	10	Double Add HL to HL	HL = HL + HL
DAD SP	39	----*	10	Double Add SP to HL	|HL = HL + SP
DCR r	3D	***-	4	Decrement	r = r –1 (0X5)
DCR M	35	***-	10	Decrement Memory	[HL] = [HL] – 1
DCX B	0B	-----	6	Decrement BC	BC = BC – 1
DCX D	1B	-----	6	Decrement DE	DE = DE – 1
DCX H	2B	-----	6	Decrement HL	HL = HL – 1
DCX SP	3B	-----	6	Decrement Stack Pointer	SP = SP – 1

MNEMONIC	OPCODE	SZAPC	~S	DESCRIPTION	NOTES
DI	F3	-----	4	Disable Interrupts	
EI	FB	-----	4	Enable Interrupts	
HLT	76	-----	5	Halt	
IN p	DB	-----	10	Input	A = [p]
INR r	3C	***-	4	Increment	r = r + 1 (0X4)|
INR M	3C	***-	10	Increment Memory	[HL] = [HL] + 1
INX B	03	-----	6	Increment BC	BC = BC + 1
INX D	13	-----	6	Increment DE	DE = DE + 1
INX H	23	-----	6	Increment HL	HL = HL + 1
INX SP	33	-----	6	Increment Stack Pointer	SP = SP + 1
JMP a	C3	-----	7	Jump unconditional	PC = a
JC a	DA	-----	7	Jump on Carry	If CY = 1 (10~s)
JM a	FA	-----	7	Jump on Minus	If S = 1 (10~s)
JNC a	D2	-----	7	Jump on No Carry	If CY = 0 (10~s)
JNZ a	C2	-----	7	Jump on No Zero	If Z = 0 (10~s)
JP a	F2	-----	7	Jump on Plus	If S = 0 (10~s)
JPE a	EA	-----	7	Jump on Parity Even	If P = 1 (10~s)
JPO a	E2	-----	7	Jump on Parity Odd	If P = 0 (10~s)
JZ a	CA	-----	7	Jump on Zero	If Z = 1 (10~s)
LDA a	3A	-----	13	Load Accumulator direct	A = [a]
LDAX B	0A	-----	7	Load Accumulator indirect	A = [BC]
LDAX D	1A	-----	7	Load Accumulator indirect	A = [DE]
LHLD a	2A	-----	16	Load HL Direct	HL = [a]
LXI B,nn	01	-----	10	Load Immediate BC	BC = nn
LXI D,nn	11	-----	10	Load Immediate DE	DE = nn
LXI H,nn	21	-----	10	Load Immediate HL	HL = nn
LXI SP,nn	31	-----	10	Load Immediate Stack Ptr	SP = nn
MOV r1,r2	7F	-----	4	Move register to register	r1 = r2 (1XX)
MOV M,r	77	-----	7	Move register to Memory	[HL] = r (16X)
MOV r,M	7E	-----	7	Move Memory to register	r = [HL] (1X6)
MVI r,n	3E	-----	7	Move Immediate	r = n (0X6)
MVI M,n	36	-----	10	Move Immediate to Memory	[HL] = n
NOP	00	-----	4	No Operation	
ORA r	B7	**0*0	4	Inclusive OR Accumulator	A = Avr (26X)|
ORA M	B6	**0*0	7	Inclusive OR Accumulator	A = Av [HL]

continued

TABLE K.2 Instruction Set Summary for the Intel 8085A Microprocessor *(Continued)*

MNEMONIC	OPCODE	SZAPC	~S	DESCRIPTION	NOTES
IORI n	F6	**0*0	7	Inclusive OR Immediate	A = Avn
IOUT p	D3	-----	10	Output	[p] = A
PCHL	E9	-----	6	Jump HL indirect	PC = [HL]
POP B	C1	-----	10	Pop BC	BC = [SP] +
POP D	D1	-----	10	Pop DE	DE = [SP] +
POP H	E1	-----	10	Pop HL	HL = [SP] +
POP PSW	F1	-----	10	Pop Processor Status Word	{PSW,A} = [SP] +
PUSH B	C5	-----	12	Push BC	– [SP] = BC
PUSH D	D5	-----	12	Push DE	– [SP] = DE
PUSH H	E5	-----	12	Push HL	– [SP] = HL
PUSH PSW	F5	-----	12	Push Processor Status Word	– [SP] = {PSW,A}
RAL	17	----*	4	Rotate Accumulator Left	A = {CY,A} < –
RAR	1F	----*	4	Rotate Accumulator Right	A = –> {CY,A}
RET	C9	-----	10	Return	PC = [SP] +
RC	D8	-----	6	Return on Carry	If CY = 1 (12~s)
RIM	20	-----	4	Read Interrupt Mask	A = mask
RM	F8	-----	6	Return on Minus	If S = 1 (12~s)
RNC	D0	-----	6	Return on No Carry	If CY = 0 (12~s)
RNZ	C0	-----	6	Return on No Zero	If Z = 0 (12~s)
RP	F0	-----	6	Return on Plus	If S = 0 (12~s)
RPE	E8	-----	6	Return on Parity Even	If P = 1 (12~s)
RPO	E0	-----	6	Return on Parity Odd	If P = 0 (12~s)
RZ	C8	-----	6	Return on Zero	If Z = 1 (12~s)
RLC	07	----*	4	Rotate Left Circular	A = A <–
RRC	0F	----*	4	Rotate Right Circular	A = –>A
RST z	C7	-----	12	Restart	(3X7) – [SP] = PC,PC = z
SBB r	9F	*****	4	Subtract with Borrow	A = A – r – CY
SBB M	9E	*****	7	Subtract with Borrow	A = A – [HL] – CY
SBI n	DE	*****	7	Subtract with Borrow Immed	A = A – n – CY
SHLD a	22	-----	16	Store HL Direct	[a] = HL
SIM	30	-----	4	Set Interrupt Mask	mask = A
SPHL	F9	-----	6	Move HL to SP	SP = HL
STA a	32	-----	13	Store Accumulator	[a] = A
STAX B	02	-----	7	Store Accumulator indirect	[BC] = A

MNEMONIC	OPCODE	SZAPC	~S	DESCRIPTION	NOTES
STAX D	12	-----	7	Store Accumulator indirect	[DE] = A
STC	37	----1	4	Set Carry	CY = 1
SUB r	97	****	4	Subtract	A = A – r (22X)
SUB M	96	****	7	Subtract Memory	A = A – [HL]
SUI n	D6	****	7	Subtract Immediate	A = A – n
XCHG	EB	-----	4	Exchange HL with DE	HL <–> DE
XRA r	AF	**0*0	4	Exclusive OR Accumulator	A = Axr (25X)
XRA M	AE	**0*0	7	Exclusive OR Accumulator	A = Ax [HL]
XRI n	EE	**0*0	7	Exclusive OR Immediate	A = Axn
XTHL	E3	-----	16	Exchange stack Top with HL	[SP] <–> HL

Notes:

	S Z A P C	
PSW	0 * 0 1	Flag unaffected/affected/reset/set
S	S	Sign (Bit 7)
Z	Z	Zero (Bit 6)
AC	A	Auxilary Carry (Bit 4)
P	P	Parity (Bit 2)
CY	C	Carry (Bit 0)
a p		Direct addressing
M z		Register indirect addressing
n nn		Immediate addressing
r		Register addressing
DB n (,n)		Define Byte(s)
DB 'string'		Define Byte ASCII character string
DS nn		Define Storage Block
DW nn (,nn)		Define Word(s)
A B C D E F L		Registers (8-bit)
BC DE HL		Register pairs (16-bit)
PC		Program Counter register (16-bit)
PSW		Processor Status Word (8-bit)
SP		Stack Pointer register (16-bit)
a nn		16-bit address/data (0 to 65535)
n p		8-bit data/port (0 to 255)
r		Register (X = B, C, D, E, H, L, M, A)
z		Vector (X = 0H, 8H, 10H, 18H, 20H, 28H, 30H, 38H)
+ –		Arithmetic addition/subtraction
& ~		Logical AND/NOT
v x		Logical inclusive/exclusive OR
< – – >		Rotate left/right
< – >		Exchange
[ ]		Indirect addressing
[ ] + – [ ]		Indirect address auto-inc/decrement
{ }		Combination operands
(X)		Octal op code where X is a 3-bit code
If (~s)		Number of cycles if condition is true

Source: From J. P. Bowen, 1985 Programming Research Group, Oxford University Computing Laboratory.

not much thought is needed, since the microprocessor and all the major hardwiring is already taken care on the motherboard. In all, the microprocessor is simply too difficult and impractical for most gadget-oriented applications.

When designing programmable gadgets, it is best to avoid microprocessors entirely. Instead, you should use a microcontroller. The microcontroller is a specialized microprocessor that houses much of the support circuitry onboard, such as ROM, RAM, serial communications ports, A/D converters, etc. In essence, a microcontroller is a minicomputer, but without the monitor, keyboard, and mouse. They are called *microcontrollers* because they are small (micro) and because they control machines, gadgets, etc. With one of these devices, you can build an "intelligent" machine, write a program on a host computer, download the program into the microcontroller via the parallel or serial port of the PC, and then disconnect the programming cable and let the program run the machine. For example, in the microwave oven, a single microcontroller has all the essential ingredients to read from a keypad, write information to the display, control the heating element, and store data such as cooking time.

There are literally thousands of different kinds of microcontrollers available. Some are one-time-programmable (OTP), meaning that once a program is written into its ROM (OTP-ROM), no changes can be made to the program thereafter. OTP microcontrollers are used in devices such as microwaves, dishwashers, automobile sensor systems, and many application-specific devices that do not require changing the core program. Others microcontrollers are reprogrammable, meaning that the microcontroller's program stored in ROM (which may either be an EPROM, EEPROM, or flash) can be changed if desired—a useful feature when prototyping or designing test instruments that may require future I/O devices.

Microcontrollers are found in pagers, bicycle light flashers, data loggers, toys (e.g., model airplanes and cars), antilock breaking systems, VCRs, microwave ovens, alarm systems, fuel injectors, exercise equipment, etc. They also can be used to construct robots, where the microcontroller acts as the robot's brain, controlling and monitoring various input and output devices, such as light sensors, stepper and servo motors, temperature sensors, speakers, etc. With a bit of programming, you can make the robot avoid objects, sweep the floor, and generate various sounds to indicate that it has encountered difficulties (e.g., low power, tipped over, etc.) or has finished sweeping. The list of applications for microcontrollers is endless.

K.2.1 Basic Structure of a Microcontroller

Figure K.3 shows the basic ingredients found within many microcontrollers. These include a CPU, ROM (OTP-ROM, EPROM, EEPROM, FLASH), RAM, I/O ports, timing circuitry/leads, interrupt control, a serial port adapter (e.g., UART, USART), and a analog-digital (A/D, D/A) converter.

The CPU is equivalent to a microprocessor (often referred to as an *imbedded processor*)—it is the "thinking" element within the microcontroller. The CPU retrieves program instructions that the user programs into ROM, while using RAM to store temporary data needed during program execution. The I/O ports are used to connect external devices that send or receive instructions to or from the CPU.

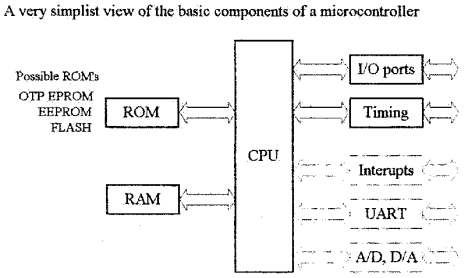


FIGURE K.3

The serial port adapter is used to provide serial communications between the microcontroller and a PC or between two microcontrollers. It is responsible for controlling the different rates of data flow common between devices. Example serial port adapters found within microcontrollers are the UART (universal asynchronous receiver transmitter) or the USART (universal synchronous/asynchronous receiver transmitter). The UART can handle asynchronous serial communications, while the USART can handle either asynchronous or synchronous serial communications.

An interrupt system is used to interrupt a running program in order to process a special routine called the *interrupt service routine*. This boils down to the ability of a microcontroller to sample external data that requires immediate attention, such as data conveyed by an external sensor indicating important shutdown information, say, when things get too hot, objects get too close, etc. A timer/counter is used to "clock" the device—provide the driving force needed to move bits around. There are a number of microcontrollers that come with built-in A/D and D/A converters that can be used to interface with analog transducers, such as temperature sensors, strain gauges, position sensors, etc.

Example Microcontrollers: PIC16C56 and PIC16C57

Figure K.4 shows Microchip's PIC16C56 and PIC16C57 microcontrollers. As you can see in the internal architecture diagram, both microcontrollers house onchip CPU, EPROM, RAM, and I/O circuitry. The architecture is based on a register file concept that uses separate buses and memories for programs and data (Harvard architecture). This allows execution to occur in parallel. As an instruction is being "prefetched," the current instruction is executing on the data bus.

The PIC16C56's program memory (EPROM) has space for 1024 words, while the PIC16C57 has space for 2048 words. An 8-bit-wide ALU contains one temporary working register and performs arithmetic and Boolean functions between data held in the working register and any file register. The ALU and register file are composed of up to 80 addressable 8-bit registers, and the I/O ports are connected via the 8-bit-wide data bus. Thirty-two bytes of RAM are directly addressable, while the access to the remaining bytes work through bank switching.

In order for bit movement to occur (clock generation), the PIC controllers require a crystal or ceramic resonator connected to pins OSC1 and OSC2. The PIC microcontrollers reach a performance of 5 million instructions per second (5 MIPS) at a clock

Microchip's PIC16C56 and PIC16C57 microcontrollers

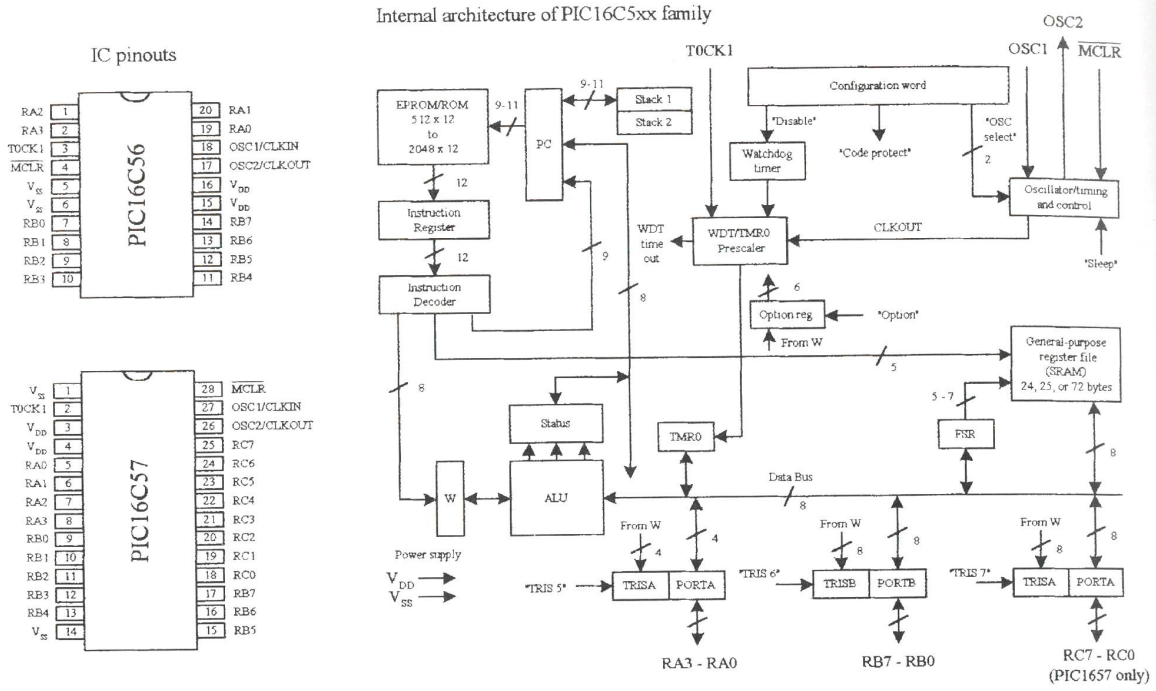


FIGURE K.4

frequency of 20 MHz. A watchdog timer is also included, which is a free-running onchip *RC* oscillator that works without external components. It continues running when the clock has stopped, making it possible to generate a reset independent of whether the controller is working or sleeping.

These chips also come with a number of I/O pins that can be linked to external devices such as light sensors, speakers, LEDs, or other logic circuits. The PIC16C56 comes with 12 I/O pins that are divided into three ports, port A (RA3–RA0), port B (RB7–RB0), and port C (RC7–RC0); the PIC16C57 comes with eight more I/O pins than the PIC16C56.

K.2.2 Programming the Microcontroller

Like microprocessors, microcontrollers use a set of machine-code instructions (1's and 0's) to perform various tasks such as adding, comparing, sampling, and outputting data via I/O ports. These machine-code instructions are typically programmed into onboard ROM (EPROM, EEPROM, flash) via a programming unit linked to a personal computer (PC). The actual programming, however, isn't written out in machine code but rather is written out in high-level language within an editor program running on the PC. The high-level language used may be a popular language such as C or a specially tailored language that the manufacturer has created to optimize all the features present within its microcontrollers.

Using a manual and software you get from the manufacturer, you learn to write out humanlike statements that tell the microcontroller what to do. You type the state-

ments in an editor program, run the program, and check for syntax errors. Once you think the program is OK, you save it and run a compiler program to translate it into machine language. If there is an error in your program, the compiler may refuse to perform the conversion. In this case, you must return to the text editor and fix the bugs before moving on. Once the bugs are eliminated and the program is compiled successfully, a third piece of software is used to load the program into the microcontroller. This may require physically removing the microcontroller from the circuit and placing into a special programmer unit linked to the host PC via a serial or parallel port.

Now there is another way to do things, which involves using an interpreter instead of a compiler. An interpreter is a high-level language translator that does not reside in the host PC but resides within the microcontroller's ROM. This often means that an external ROM (EPROM, EEPROM, flash) is needed to store the actual program. The interpreter receives the high-level language code from the PC and, on the spot, interprets the code and places the translated code (machine code) into the external ROM, where it can be used by the microcontroller. Now this may seem like a waste of memory, since the interpreter consumes valuable onchip memory space. Also, using an interpreter significantly slows things down—a result of having to retrieve program instructions from external memory. However, when using an interpreter, a very important advantage arises. By having the interpreter onboard to translate on the spot, an immediate, interactive relationship between host program and microcontroller is created. This allows you to build your program, immediately try out small pieces of code, test the code by downloading the chunks into the microcontroller, and then see if the specific chunks of code work. The host programs used to create the source code often come with debugging features that let you test to see where possible programming or hardwiring errors may result by displaying the results (e.g., logic state at a given I/O pin) on the computer screen while the program is executing within the microcontroller. This allows you to perfect specific tasks within the program, such as perfecting a sound-generation routine, a stepper motor control routine, and so forth.

K.2.3 BASIC Stamps (Microcontroller with Interpreter and Extra Goodies)

The BASIC Stamp is, at the heart, a microcontroller with interpreter software built in. These devices also come with additional support circuitry, such as an EEPROM, voltage regulator, ceramic oscillator, etc. BASIC Stamps are ideal for beginners because they are easy to program, quite powerful, and relatively cheap—a whole startup package costs around $150 dollars or so. These devices are also very popular among inventors and hobbyist, and you'll find a lot of helpful literature, application notes, and fully tested projects on the Internet.

The original stamp was introduced in 1993 by Parallax, Inc. It got its name from the fact that it resembled a postage stamp. The early version of the BASIC Stamp was the REV D, while later improvements lead to the BASIC Stamp I (BSI) and to the BASIC Stamp II (BSII). Here we'll focus mainly on the BSI and the BSII. Both the BSI and BSII have a specially tailored BASIC interpreter firmware built into the microcontroller's EPROM. For both stamps, a PIC microcontroller is used. The actual pro-

gram that is to be run is stored in an onboard EEPROM. When the battery is connected, stamps run the BASIC program in memory. Stamps can be reprogrammed at any time by temporarily connecting them to a PC running a simple host program. The new program is typed in, a key is hit, and the program is loaded into the stamp. Input/output pins can be connected with other digital devices such as sense switches, LED, LCD displays, servos, stepper motors, etc.

BASIC Stamp II (BSII-IC)

The BSII is a module that comes in a 28-pin DIL package (see Fig. K.5). The brain of the BSII is the PIC16C57 microcontroller that is permanently programmed with a PBASIC2 instruction set within its internal OTP-EPROM (one-time program ROM). When programming the BSII, you tell the PIC16C57 to store symbols, called *tokens*, in external EEPROM memory. When the program runs, the PIC16C57 retrieves tokens from memory, interprets them as PBASIC2 instructions, and carries out those instructions. The PIC16C57 can execute its internal program at a rate of 5 million machine instruction per second. However, each PBASIC2 instruction takes up many machine instructions, so the PBASIC2 executes more slowly, around 3000 to 4000 instructions per second.

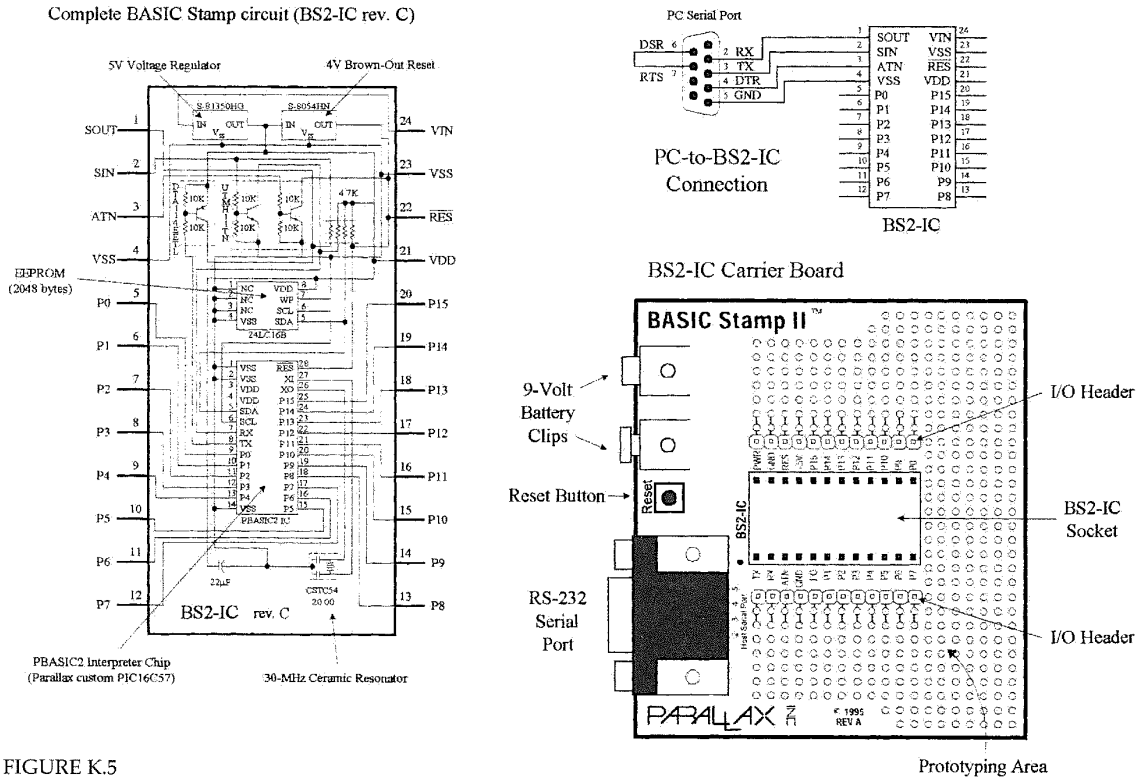


FIGURE K.5

The BSII comes with 16 I/O pins (P0–P15) that are available for general use by your programs. These pins can be interfaced with all modern 5-V logic, from TTL through CMOS (technically, they have characteristics like the 74HCT logic series). The direction of a pin—either input or output—is set during the programming phase. When a pin is set as an output pin, the BSII can send signal to other devices, like

LEDs, servos, etc. When a pin is set as an input pin, it can receive signals from external devices, such as switches, photosensors, etc. Each I/O pin can source 20 mA and sink 25 mA. Pins P0–P7 and pins P8–P15, as groups, can each source a total of 40 and sink 50 mA.

2048-Byte EEPROM

The BSII's PIC's internal OTP-EPROM (one-time programmable read-only memory) is permanently programmed at the factory with Parallax's firmware which turns this memory into a PBASIC2 interpreter chip. Because they are interpreters, the Stamp PICs have the entire PBASIC language permanently programmed into their internal program memory. This memory cannot be used to store your PBASIC2 program. Instead, the main program must be stored in the EEPROM (electrically erasable, programmable read-only memory). This memory retains data without power and can be reprogrammed easily. At run time, the PBASIC2 program created on the host computer is loaded into the BSII's EEPROM starting at the highest address (2047) and working downward. Most programs do not use the entire EEPROM, which means that PBASIC2 lets you store data in the unused lower portion of the EEPROM. Since programs are stored from the top of the memory downward, data are stored in the bottom of the memory working upward. If there is an overlap, the Stamp host software will detect this problem and display an error message.

Reset Circuit

The BSII comes with a reset circuit. When power is first connected to the Stamp, or if it falters due to a weak battery, the power supply voltage can fall below the required 5 V. During such brownouts, the PIC is in a voltage-deprived state and will have the tendency to behave erratically. For this reason, a reset chip is incorporated into the design, forcing the PIC to rest to the beginning of the program and hold until the supply voltage is within acceptable limits.

Power Supply

To avoid supplying the BSII with unregulated supply power, a 5-V regulator is incorporated into the BSII. This regulator accepts a voltage range from slightly over 5 V up to 15 V and regulates it to a steady 5 V. It provides up to 50 mA. The regulated 5 V is available at output V_{DD}, where it can be used to power other parts of your circuits so long as no more than 50 mA is required.

Connecting BSII to a Host PC

To program a Stamp requires connecting it to a PC that runs host software to allow you to write, edit, download, and debug PBASIC2 programs. The PC communicates with the BSII through an RS-232 (COM port) interface consisting of pins S_{IN}, S_{OUT}, and *ATM* (serial in, serial out, and attention, respectively). During programming, the BSII host program pulses *ATM* high to reset the PIC and then transmits a signal to the PIC through S_{IN} indicating that it wants to download a new program. PC-to-BSII connector hookup is shown in Fig. K.5. This connection allows the PC to reset the BSII for programming, download programs, and receive debug data from the BSII. The additional pair of connections, pin 6 and 7 of the DB9 socket, lets the BSII host software identify the port to which the BSII is connected. Usually, when programming a BSII, you use a special BSII carrier board, which comes with a prototyping area, I/O

header, BSII-IC socket, 9-V battery clips, and an RS-232 serial port connector, as shown in Fig. K.5. These boards, along with programming cable and software, can be purchased as startup packages.

PBASIC Language

Even though the BASIC Stamp has "BASIC" in its name, it cannot be programmed in Visual BASIC or QBASIC. It does not have a graphic user interface, a hard drive, or a lot of RAM. The BASIC Stamp must only be programmed with Parallel's BASIC (PBASIC), which has been specifically designed to exploit all the BASIC Stamp's capabilities. PBASIC is a hybrid form of BASIC programming language with which many people are familiar. Currently, there are two version of PBASIC: PBASIC1 for the BASIC Stamp I and PBASIC2 for the BASIC Stamp II. Each version is specifically tailored to take advantage of the inherent features of the hardware it runs on. PBASIC is called a *hybrid* because, while it contains some simplified forms of normal BASIC control constructs, it also has special commands to efficiently control I/O pins. PBASIC is an easy language to master and includes familiar instructions such as GOTO, FOR . . . NEXT, and IF . . . THEN. It also contains Stamp-specific instructions, such as PULSOUT, DEBUG, BUTTON, etc., which will be discussed shortly.

The actual program to be downloaded into the Stamp is first written using the BASIC Stamp I (for BSI) or BASIC Stamp II (for BSII) editor software running on an IBM-compatible PC or MacIntosh that is running SoftPC or SoftWindows 2.0. After you write the code for your application, you simply connect the Stamp to the computer's parallel port (for BSI) or serial port (for BSII), provide power to the Stamp, and press ALT-R within the editor program to download the code into the Stamp. As soon as the program has been downloaded successfully, it begins executing its new program from the first line of code.

The size of the program that can be stored in a Stamp is limited. For the BSI, there are 256 bytes of program storage, enough for around 80 to 100 lines of PBASIC code. For the BSII, there are 2048 bytes worth of program space, enough for around 500 to 600 lines of PBASIC code. The amount of program memory for the Stamps cannot be expanded, since the interpreter chip (PIC) expects the memory to be specific and fixed in size. However, in terms of data memory, expansion is possible. It is possible to interface EEPROMs or other memory devices to the Stamp's I/O pins to gain more data storage area. This requires that you supply the appropriate code within your PBASIC program to make communication between the Stamp and external memory device you choose possible. Addition data memory is often available with Stamp-powered applications that monitor and record data (e.g., environmental field instrument).

Debugging

To debug PBASIC programs, the BASIC Stamp editor comes with two handy features—syntax checking and a DEBUG command. Syntax checking alerts you to any syntactical error and is automatically performed on your code the moment you try to download to the BASIC Stamp. Any syntax errors will cause the download process to abort and will cause the editor to display an error message, pointing out the error in the source code. The DEBUG command, unlike syntax checking, is an instruction that is written into the program to find logical errors—ones that the Stamp does not find,

but ones that the designer had not intended. DEBUG operates similar to the PRINT command in the BASIC language and can be used to print the current status of specific variable within your PBASIC program as it is executed within the BASIC stamp. If your PBASIC code includes a DEBUG command, the editor opens up a special window at the end of the download process to display the result for you.

Overview of PBASIC II Programming Language

The PBASIC II language, like other high-level computer languages, involves defining variables and constants and using address labels, mathematical and binary operators, and various instructions (e.g., branching, looping, numerics, digital I/O, serial I/O, analog I/O, sound I/O, EEPROM access, time, power control, etc.) Here's a quick rundown on the elements of the PBASIC II language.

COMMENTS Comments can be added within the program to describe what you're doing. They begin with an apostrophe (') and continue to the end of the line.

VARIABLES These are locations in memory that your program can use to store and recall numbers. These variables have limited range. Before a variable can be used in a PBASIC2 program, it must be declared. The common way used to declare variables is to use a directive VAR:

```
symbol   var  size
```

where the symbol can be any name that starts with a letter; can contain a mixture of letters, numbers, and underscore; and must not be the same as PBASIC keywords or labels used in the program. The size establishes the number of bits of storage the variable is to contain. PBASIC2 provides four sizes: bit (1 bit), nib (4 bits), byte (8 bits), and word (16 bits). Here are some examples of variable declarations:

```
'Declare variables.
sense_in     var bit     'Value can be 0 or 1.
speed        var nib    'Value in range 0 to 15.
length       var byte    'Value in range 0 to 255.
n            var word   'Value in range 0 to 65535.
```

CONSTANTS Constant are unchanging values that are assigned at the beginning of the program and may be used in place of the numbers they represent within the program. Defining constants can be accomplished by using the CON directive:

```
beeps         con  5        'number of beeps
```

By default, PBASIC2 assumes that numbers are in decimal (base 10). However, it is possible to use binary and hexadecimal numbers by defining them with prefixes. For example, when the prefix "%" in placed in front of a binary number (e.g., %0111 0111), the number is treated as a binary number, not a decimal number. To define a hexadecimal number, the prefix "$" is used (e.g., $EF). Also, PBASIC2 will automatically convert quoted text into the corresponding ASCII codes(s). For example, defining a constant as "A" will be interpreted as the ASCII code for A (65).

ADDRESS LABELS The editor uses address labels to refer to addresses (locations) within the program. This is different from some versions of BASIC, which use line numbers. In general, an address label name can be any combination of letters, num-

bers, and underscores. However, the first character in the label name cannot be a number, and the label name must not be the same as a reserved word, such as a PBASIC instruction or variable. The program can be told to go to the address label and follow whatever instructions are listed after. Address labels are indicated with a terminating colon (e.g., loop:).

MATHEMATICAL OPERATORS BPASIC2 uses two types of operators: unary and binary. Unary operators take precedence over binary operator. Also, unary operations are always performed first. For example, in the expression 10 – SQR 16, the BSII first takes the square root of 16 and then subtracts it from 10.

Unary Operators

ABS	Returns absolute value
SQR	Returns square root of value
DCD	2^n-power decoder
NCD	Priority encoder of a 16-bit value
SIN	Returns two's compliment sine
COS	Returns two's compliment cosine

Binary Operators

+	Addition
–	Subtraction
/	Division
//	Remainder of division
*	Multiplication
**	High 16-bits of multiplication
*/	Multiply by 8-bit whole and 8-bit part
MIN	Limits a value to specified low
MAX	Limits a value to specified high
DIG	Returns specified digit of number
<<	Shift bits left by specified amount
>>	Shift bits right by specified amount
REV	Reverse specified number of bits
&	Bitwise AND of two values
|	Bitwise OR of two values
&	Bitwise XOR of two values

PBASIC Instructions Used by BASIC Stamp II

BRANCHING

IF *condition* THEN *addressLabel*

Evaluate condition and, if true, go to the point in the program marked by *addressLabel*. (Conditions: =, <> not equal, >, <1, > =, < =).

BRANCH *offset*, [*address0*, *address1*, . . . *addressN*]

Go to the address specified by offset (if in range).

GOTO *addressLabel*

Go to the point in the program specified by *addressLabel*.

GOSUB *addressLabel*

Store the address of the next instruction after GOSUB, then go to the point in the program specified by *addressLabel*.

RETURN

Return from subroutine.

LOOPING

FOR *variable* = *start* to *end* {*STEP stepVal*} . . . NEXT

Create a repeating loop that executes the program lines between For and Next, incrementing or decrementing *variable* according to *stepVal* until the value of the variable passes the *end* value.

NUMERICS

LOOKUP *index*, [*value0*, *value1*, . . . *valueN*], *resultVariable*

Look up the value specified by the index and store it in a variable. If the index exceeds the highest index value of the items in the list, variable is unaffected. A maximum of 256 values can be included in the list.

LOOKDOWN *value*, {*comparisonOp*,} [*value0*, *value1*, . . . *valueN*], *resultVariable*

Compare a value to a list of values according to the relationship specified by the comparison operator. Store the index number of the first value that makes the comparison true into *resultVariable*. If no value in the list makes the comparison true, *resultVariable* is unaffected.

RANDOM *variable*

Generates a pseudo-random number using byte or word variable that's bits are scrambled to produce a random number.

DIGITAL I/O

INPUT *pin*	Make the specified pin an input.
OUTPUT *pin*	Make the specified pin an output.
REVERSE *pin*	If pin is an output, make it an input. If pin is an input, make it an output.
LOW *pin*	Make specified pin's output low.
HIGH *pin*	Make specified pin's output high.
TOGGLE *pin*	Invert the state of a pin.
PULSIN *pin, state, resultVariable*	Measure the width of a pulse in 2-μs units.
PULSOUT *pin, time*	Outputs a timed pulse to by inverting a pin for some time (× 2 μs).

BUTTON *pin, downstate, delay,rate,bytevariable, targetstate, address*
Debounce button input, perform auto-repeat, and branch to address if button is in target state. Button circuits may be active-low or active-high.

SHIFTIN *dpin,cpin,mode, [result{\bits}{,result{\bits} . . . }]*
Shift data in from a synchronous-serial device.

SHIFTOUT *dpin,cpin,mode, [data{\bits}{,data{\bits} . . . }]*
Shift data out to a synchronous-serial device.

COUNT *pin, period, variable*
Count the number of cycles (0-1-0 or 1-0-1) on the specified pin during *period* number of milliseconds and store that number in *variable*.

XOUT *mpin,zpin, [house\keyORCommand{\cycles} {,house\keyOrCommand{\cycles} . . . }]*
Generate X-10 powerline control codes.

SERIAL I/O

SERIN *rpin{\fpin},baudmode, {plabe}{timeout,tlabe,}[input Data]*
Receive asynchronous serial transmission.

SEROUT *tpin,baudmode,{pace,} [outputData]*
Send data serially with optional byte pacing and flow control.

ANALOG I/O

PWM *pin, duty, cycles*	Output fast pulse-width modulation, then return pin to input. This can be used to output an analog voltage (0–5 V) using a capacitor and resistor.
RCTIME *pin, state, resultVariable*	Measure an *RC* charge/discharge time. Can be used to measurepotentiometers.

SOUND

FREQOUT *pin, duration, freq1{,freq2}*	Generate one or two sine-wave tones for a specified duration.
DTMFOUT *pin,{ontime,offtime,}{,tone . . . }*	Generate dual-tone, multifrequency tones (DTMF, i.e., telephone "touch" tones).

EEPROM ACCESS

DATA	Store data in EEPROM before downloading PBASIC program.
READ *location, variable*	Read EEPROM location and store value in variable.
WRITE *address,byte*	Write a byte of data to the EEPROM at appropriate address.

TIME

PAUSE *milliseconds*	Pause the program (do nothing) for the specified number of milliseconds. Pause execution for 0–65,535 ms.

POWER CONTROL

NAP *period*	Enter sleep mode for a short period. Power consumption is reduced to about 50 µA assuming no loads are being driven. The duration is $(2^{period}) \times 18$ ms.
SLEEP *seconds*	Sleep from 1–65,535 seconds to reduce power consumption by ~50 µA.
END	Sleep until the power cycles or the PC connects ~50 µA.

PROGRAM DEBUGGING

DEBUG *outputData{,outputData . . . }*	Display variables and messages on the PC screen within the BSII host program; *outputData* consists of one or more of the following: text strings, variables, constants, expressions, formatting modifiers, and control characters

Making a Robot Using the BASIC Stamp II

To illustrate how easy it is to make interesting gadgets using the BASIC Stamp II (BSII), let's take a look at a robot application. In this application, the main objective is to prevent the robot from running into objects. The robot aimlessly moves around, and when it comes close to an object, say, to the left, the robot is to stop and then back up and move off in another direction. In this example, a BSII acts as the robot's brain, two servos connected to wheels acts as its legs, a pair of infrared transmitters and sensors acts as its eyes, and a piezoelectric speaker acts as it voice. Figure K.6 shows the completed robot, along with the various individual components.

Components and connections used to create object-avoiding robot

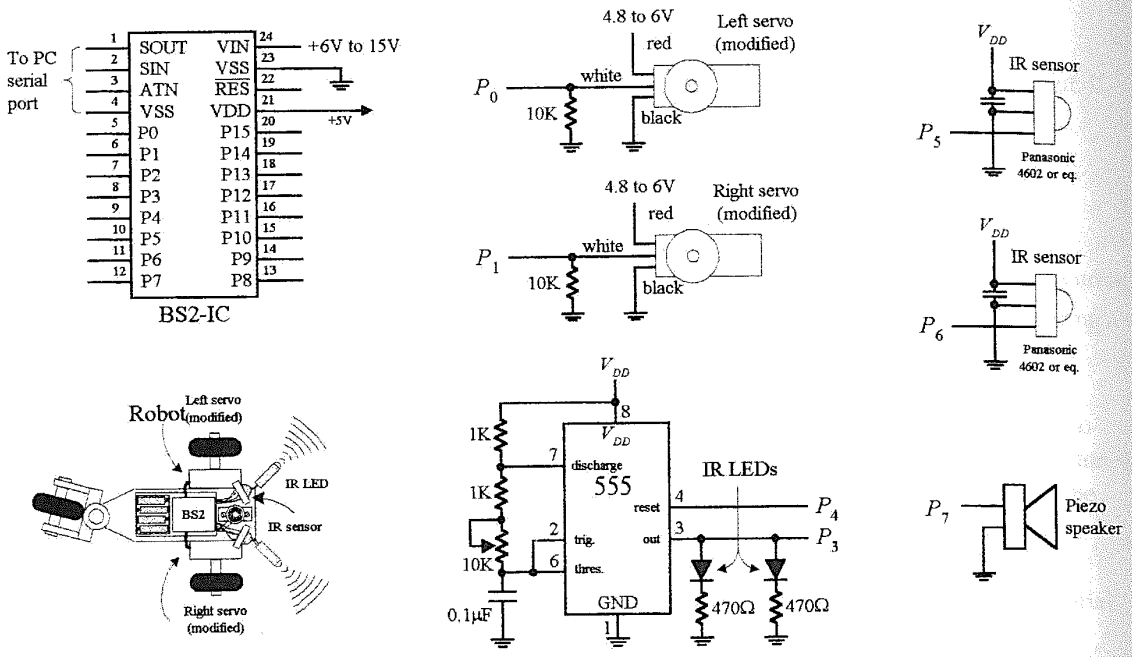


FIGURE K.6

THE SERVOS

The directional movement of the robot is controlled by right and left servo motors that have been modified so as to provide a full 360° worth of rotation—modifying a servo is discussed in Chap. 13. To control a servo requires generating pulses ranging from 1000 to 2000 μs in width at intervals of approximately 10 to 20 ms. With one of the servos used in our example, when the pulse width sent to the servo's control line is set to 1500 μs, the servo is centered—it doesn't move. However, if the pulse width is shortened to, say, 1300 μs, the modified servo rotates clockwise. Conversely, if the pulse width is lengthened, say, to 1700 μs, the modified servo rotates counterclockwise.

The actual control pulses used to drive one of the servos in the robot are generated by the BSII using the PULSOUT *pin, time1* and the PAUSE *time2* instructions. The *pin* represents the specific BSII pin that is linked to a servo's control line, while *time1* represents how long the pin will be pulsed high. Note that for the PULSOUT instruction, the decimal placed in the *time1* slot actually represents half the time, in microseconds (μs), that the pin is pulsed high. For example, PULSOUT 1, 1000 means that the BSII will pulse pin 1 high for 2000 μs, or 2ms. For the PAUSE instruction, the decimal placed in the *time2* slot represents a pause in milliseconds (ms). For example, PAUSE 20, represents a 20-ms pause. Figure K.7 shows sample BSII code used to generate

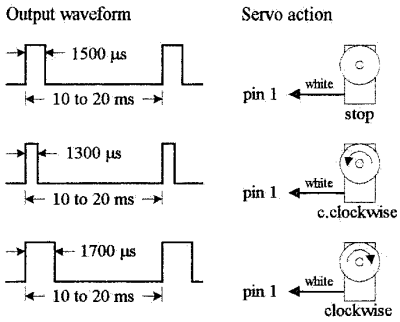


FIGURE K.7

```
BS2 code            Comments

pulsout 1, 750      'pulse width of 1500us on pin 1
pause 20            'pause for 20 ms

pulsout 1, 650      'pulse width of 1300us on pin 1
pause 20            'pause for 20 ms

pulsout 1, 850      'pulse width of 1700us on pin 1
pause 20            'pause for 20 ms
```

IR TRANSMITTERS AND RECEIVERS

The robot's object-detection system consists of a right and left set of infrared (IR) LED transmitters and IR detector modules. The IR LEDs are flashed via a 555 timer at a very high frequency, which in this example happens to be 38 kHz, 50 percent duty cycle. The choice of this frequency is used to avoid interference from other household sources of infrared, primarily incandescent lights. (Many types of IR LED transmitters and sensors could be used in this robot, and they may work best using a different frequency.) The IR photons emitted by the LED rebound off objects in the path of the robot and reflect back to the IR detector module. When a detector module receives photons, the I/O pin of the BSII connected to the module goes low. (Note that the BSII can only execute around 4000 instruction per second, while the number of pulses generated by the detector module is 38,000. In this case the actual number of pulses received by the BSII will be less, around 10 or 20.)

PIEZOELECTRIC SPEAKER

A piezoelectric speaker is linked to one of the BSII I/O terminals and is used generate different sounds when the robot is moving forward or backing up. To provide the piezoelectric speaker with a sinusoidal waveform needed to generate sound, the FREQOUT *pin, time, frequency* instruction is used. The instruction FREQOUT 7, 1000, 440 creates a 440-Hz sinusoidal frequency on pin 7 that lasts for 1000 ms.

THE PROGRAM

The following is a program used to control the robot. It is first created using the PBASIC2 host software and then downloaded into the BSII during runtime.

```
'Program for object-avoiding robot

'Define variables and constants
'-----------------------------------------------------------------------------------------
n            var word     'n acts as a variable that changes.
right_IR     var in5      'Sets pin 5 as an input for right IR detector.
left-IR      var in6      'Sets pin 6 as an input for left IR detector.
right_servo  con 0        'Assigns 0 which will be used to identify right servo.
left_servo   con 1        'Assigns 1 to identify left servo.
IR_out       con 3        'Assigns 3 to identify IR output.
delay        con 10       'A constant that we be used in program.
speed        con 100      'Used to set servo speed.
turn_speed   con 50       'Used to set turn speed of robot.

'Main program
'-----------------------------------------------------------------------------------------
high IR_out                                    'Sets pin 6 "high"
pause 50                                       'Pauses for 50 milliseconds
sense:                                         'Label used to specify IR-sense routine.
if left_IR = 0 and right_IR = 0 then backup    'Object in path, jump to back_up routine.
if left_IR = 0 then turn_right                 'Object on left side, jump to turn_right routine.
if right_IR = 0 then turn_left                 'Object on right, jump to "turn_left" routine.

'Sound Routines
'-----------------------------------------------------------------------------------------
forward_sound:                  'Label
freqout 7,1000, 440             'Generate 1000ms, 440 Hz tone on pin 7

back_sound:                     'Label
freqout 7,1000,880              'Generate 1000ms, 880 Hz tone on pin 7

'Motion routines
'-----------------------------------------------------------------------------------------

forward:                        'Label used to specify forward routine.
gosub forward_sound             'Tells program to jump to forward sound subroutine.
debug "forward"                 'Tells stamp to display the word "forward" on debug window.
pause 50                        'Pause for 50ms.
for n = 1 to delay*2            'For…Next loop that starts x = 1 and repeats until x = 20.
pulsout left_servo, 750-speed   'Make left servo spin to make robot move forward.
pulsout right_servo, 750+speed  'Make right servo spin to make robot move forward.
pause 20                        'Pauses for 20ms, path of servo control.
next                            'End of For…Next loop.
goto sense                      'Once forward routine is finished go back to sense routine.

backup:                         'Label used to specify back-up routine.
gosub backup_sound              'Tells program to jump back-up sound subroutine.
debug "backward"                'Displays "backward" on the debug window.
pause 50                        'Pause for 50ms to ensure
for n = 1 to delay*3            'For…Next loop that starts x = 1 and repeats until x = 60
```

```
pulsout left_servo, 750+speed        'Makes left servo spin to make robot move backward.
pulsout right_servo, 700-speed       'Makes right servo spin to make robot move backward.
pause 20                             'Pauses for 20ms, part of servo control.
next                                 'End of For...Next loop.

turn_left:                           'Label used to specify turn-left routine.
debug "left"                         'Displays "left" on the debug window.
pause 50                             'Pause for 50ms.
for x = 1 to delay*1                 'For...Next loop that starts x = 1 and repeats until x = 10.
pulsout left_servo, 750-turn_speed   'Makes left servo spin to make robot turn left.
pulsout right_servo, 700-turn_speed 'Makes right servo spin to make servo spin left.
pause 20                             'Pause for 20ms, part of servo control.
next                                 'End of For...Next loop.
goto sense                           'Once left-turn routine is finished, jump back to sense.

turn_right:                          'Label used to specify turn-right routine.
debug "right"                        'Displays "right" on debug window.
pause 50                             'Pause for 50 ms.
for x = 1 to delay*1                 'For...Next loop.
pulsout left_servo, 750+turn_speed   'Makes left servo spin to make robot turn right.
pulsout right_servo, 750+turn_speed 'Makes right servo spin to make robot turn right.
pause 20                             'Pause for 20 ms, part of servo control.
next                                 'End of For...Next loop
goto sense                           'Once right-turn is finished, jump back to sense.
```

BASIC Stamp I (BSI-IC)

Figure K.8 shows the BASIC Stamp I (BSI), which was mentioned briefly earlier. This device is actually the predecessor of the BSII, but it is still used frequently enough that it is worth mentioning here. The BSI contains most of the same features as the BSII, but not all. It has only 8 I/O pins instead of 16, and it uses a PIC16C56 instead of a PIC16C57. It also uses a PBASIC1 programming language instead of PBASIC2, and its link with the host computer is via the parallel port instead of the serial port. It has a smaller instruction set, is a bit slower, and doesn't have as many variables for RAM.

Things Needed to Get Started with the BASIC Stamp

These include programming software, programming cable, manual, the BASIC Stamp module, and an appropriate carrier board (optional). If you are interested in using a particular Stamp, either the BASIC Stamp I or II, purchase either startup kit. These kits include all five items listed, at a lower cost than purchasing each part separately. However, if you intend to use both BASIC Stamp I and II, it is best to purchase the BASIC Stamp Programming Package (which includes manual, software, and cables for both versions of the Stamp) and then purchase the BASIC Stamp modules and optionally the carrier boards separately.

Learning More about BASIC Stamps

To fully understand all the finer details needed to program BASIC Stamps, it is necessary to read through the user's manual. However, reading the user's manual alone

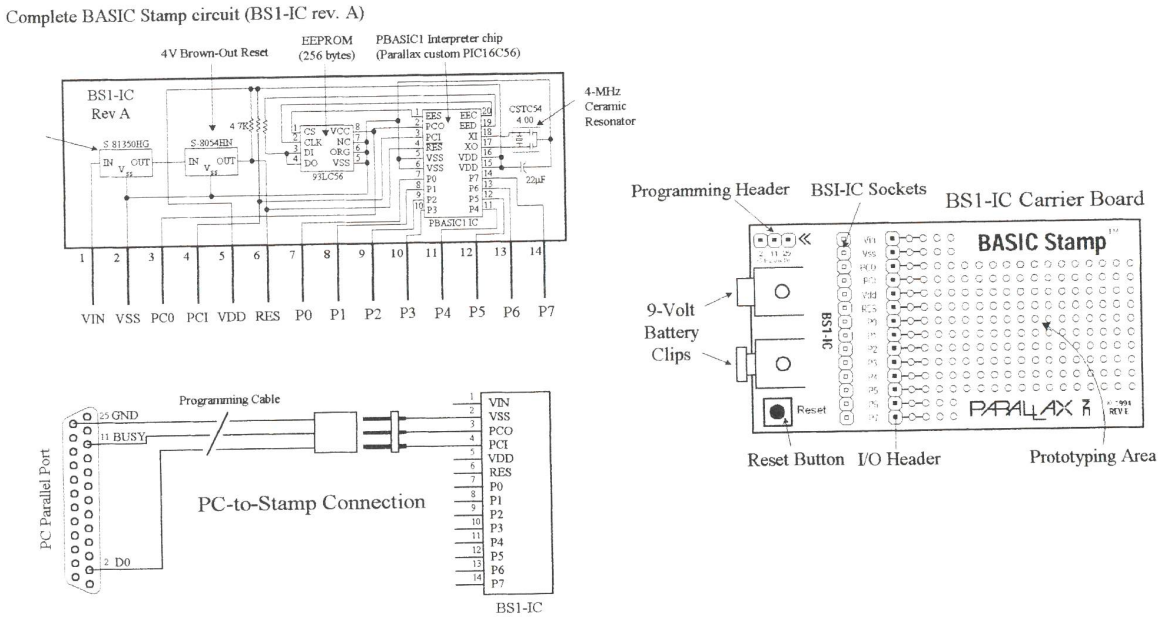


FIGURE K.8

tends not to be the best learning strategy—it is easy to lose your place within all the technical terms, especially if you are a beginner. If you are a beginner, I recommend visiting *www.stampsinclass.com*. At this site you'll find a series of tutorials, along with the Stamp user manual, that you can download. The tutorials are written is plain English and assume no prior knowledge of microcontrollers (or electronics for that matter). Another good source to learn more about BASIC Stamps is the book *Programming and Customizing the Basic Stamp Computer*, written by Scott Edwards (McGraw-Hill). This book is also geared toward beginners and is easy reading.

Thinking about Mass Production

Recall that the major components of the BASIC Stamp circuit are the PIC (houses CPU and ROM for storing PBASIC interpreter), external EEPROM (stores program), and the resonator. In large-scale runs it would be nice to get rid of the external memory and remove the interpreter program and simply download a compiled PBASIC code directly into the PIC—this saves space and money. As it turns out, the BASIC Stamp editor software includes a feature to program PBASIC code directly into a PIC microcontroller using Parallax's PIC16Cxx programmer. (The major benefit for starting out with the Stamp is that you can easily fine-tune your code, test out chunks, and immediately see if it works—an important feature when creating prototypes. When prototyping with a PIC, however, checking for errors is much harder because you must compile everything at once—you can't test out chunks of code.)

K.2.4 Other Microcontrollers Worth Considering

There are thousands of different microcontrollers on the market made by a number of different manufacturers. Each manufacturer has a number of different devices, each designed with different features, making them suitable for different applications.

Perhaps the most popular microcontrollers used by hobbyists are Microchip's PICs. These microcontrollers are relatively simple to program, inexpensive, and have loads of development software available. There is a very complete line of CMOS PIC microcontrollers with varying features.

Microchip supplies a PICSTART Plus programmer that is an entry-level development kit that will support PIC12C5XX, PIC12CX, PIC16C5XX, PIC12C6X, PIC16C, PIC17C, PIC14000, PIC17C4X, and PIC17C75X. The PICSTART Plus programming kit includes RS232 cable, power supply, PIC16C84 EEPROM microcontroller sample, software, and manual. Check out Microchip's Web site for more details.

Other major manufacturers of microcontrollers include Motorola, Hitachi, Intel, NEC, Philips, Toshiba, Texas Instruments, National Semiconductor, Mitsubishi, Zilog, etc. Here are some sample microcontrollers worth mentioning.

8051 (Intel and Others)

The 8051 is a very popular microcontroller that comes with modified Harvard architecture (separate address spaces for program, memory and data memory) and has comes with 64 K worth of program memory. This chip is very powerful and easy to program, and there is loads of development software, both commercial and freely available. It is often featured in construction projects in the popular hobbyist magazines.

8052AH-BASIC

This is popular with hobbyists, and like the BASIC Stamp, it is easy to work with.

68HC11 (Motorola)

This is a popular 8-bit controller which, depending on the variety, has built-in EEPROM/OTPROM, RAM, digital I/O, timers, A/D converter, PWM generator, pulse accumulator, and synchronous and asynchronous communications channels.

COP800 Family (National Semiconductor)

Basic family is fully static 8-bit microcontroller, which contains system timing, interrupt logic, ROM, RAM, and I/O. Depending on the device, features include 8-bit memory-mapped architecture, serial I/O, UART, memory-mapped I/O, many 16-bit timer/counters, a multisourced vectored interrupt, comparator, watchdog timer and clock monitor, modulator/timer, 8-channel A/D converter, brownout protection, halt mode, idle mode, and high-current I/O pins. Most within the family operate over a 2.5- to 6.0-V range.

DS5000/DS2250 (Dallas Semiconductor)

All you need to add is a crystal and two capacitors to end up with a working system. These chips come complete with nonvolatile RAM.

TMS370 (Texas Instruments)

Similar to the 8051 in having 256 registers, A and B accumulators, stack in the register page, etc. The peripherals include RAM ROM (mask, OTP, or EEPROM), two timers, SCI (synchronous serial port), SPI (asynchronous serial port), A/D (8-bit, 8-chan), and interrupts.

K.2.5 Evaluation Kits/Board

Many manufacturers offer assembled evaluation kits or board which usually allow you to use a PC as a host development system, as we saw with Parallax BASIC Stamp. Among some of the other popular evaluation kits/boards are the following.

Motorola's EVBU, EVB, EVM, EVS

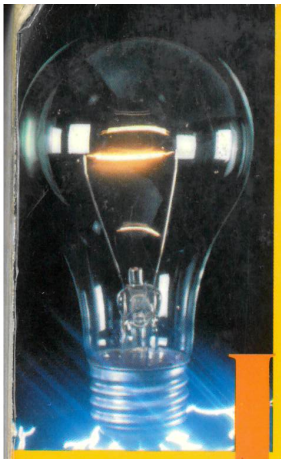
A series of very popular evaluation/development systems based on the 68HC11. Comes complete with BUFFALO monitor and varying types of development software. Commonly used in university courses.

Motorola 68705 Starter Kit

Motorola supplies a complete development system—software, hardware, simulator, emulator, manuals, etc.—for about 100 dollars.

National Semiconductor's EPU

The COP8780 evaluation/programming unit (EPU) offers you a low-cost ($125) tool for an introduction to National's COP BASIC family of 8-bit microcontrollers. The system includes the EPU board, assembler and debugger software, sample code, C compiler, wall power supply, documentation, etc.



$39.95 U.S.A.

BASIC ELECTRONICS THEORY & APPLICATIONS ... ILLUSTRATED.

In a great many introductory electronics books, the emphasis is on technical formulas and theory, while practical applications and advice often get lost in a high-tech haze. Not very inspiring ... especially if you want to succeed in turning your ideas into workable electrical gadgets.

Practical Electronics for Inventors, on the other hand, gives you information you need, in a format you can work with. Packed with hand-drawn illustrations, this crystal-clear, learn-as-you-go guide shows you what a particular device does, what it looks like, how it compares with similar devices, and how it is used in applications. Written by Paul Scherz, an inventor and electrical hobbyist, this important reference provides beginning hobbyists and inventors with an intuitive grasp of the theoretical and practical aspects of electronics—just the kind of insight you need to get your projects up and running.

Starting with a light review of electronics history, physics, and math, the book provides an easy-to-understand overview of all major electronic elements:

Basic passive components • Resistors, capacitors, inductors, transformers • Discrete passive circuits • Current limiting networks, voltage dividers, filter circuits, attenuators • Discrete active devices • Diodes, transistors, thrysistors • Microcontrollers • Rectifiers, amplifiers, modulators, mixers, voltage regulators

Along with coverage of integrated circuits (ICs), digital electronics, and various input/output devices, *Practical Electronics for Inventors* takes you through reading schematics; building and testing prototypes; purchasing electronic components; and safe work practices. You'll find all this—and more—in the guide that's destined to spur you on to new levels of creativity.

ISBN 0-07-058078-2
90000
9 780070 580787
6 39785 30535 4

Visit us on the World Wide Web at
www.books.mcgraw-hill.com

McGraw-Hill
A Division of The McGraw-Hill Companies