Data Acquisition Techniques Using PCs

Second Edition

Howard Austerlitz

Data Acquisition Techniques Using PCs

Second Edition

This Page Intentionally Left Blank

Data Acquisition Techniques Using PCs

Second Edition

Howard Austerlitz
Parker Hannifin Corporation
Parker Aerospace
Electronic Systems Division
Smithtown, New York

ACADEMIC
PRESS

An imprint of Elsevier Science

Amsterdam Boston London New York Oxford Paris San Diego
San Francisco Singapore Sydney Tokyo

This book is printed on acid-free paper. ♾

Academic Press
An imprint of Elsevier Science
525 B Street, Suite 1900, San Diego, California 92101-4495, USA
http://www.academicpress.com

Academic Press
An imprint of Elsevier Science
84 Theobald's Road, London WC1X 8RR, UK
http://www.academicpress.com

Library of Congress Control Number: 2002107710
International Standard Book Number: 0-12-068377-6

PRINTED IN THE UNITED STATES OF AMERICA

02 03 04 05 06 9 8 7 6 5 4 3 2 1

*This book is dedicated to my wife, Kiel,
whose guidance and understanding
continue to make it all possible*

This Page Intentionally Left Blank

Contents

Preface to the Second Edition

Many things have changed in the decade since the first edition of *Data Acquisition Techniques Using PCs* was published. PCs based on Intel microprocessors and Microsoft Windows (the ubiquitous "Wintel" platform) have become the dominant standard in small computers. They have also become the most common computers in labs, offices, and industrial settings for data acquisition and general-purpose applications. The world of PCs has continued to evolve at a frenetic pace and the data acquisition market has changed along with it, albeit more gradually (for example, ISA data acquisition cards are still readily available).

Some of the changes in this edition include minimizing the amount of material covering now-obsolete PCs (such as IBM's Micro Channel PS/2 line and Apple's NuBus-based Macintosh line) while adding information about more current standards (such as the PCI bus, the USB interface, and the Java programming language). Most importantly, I have completely updated information about commercially available data acquisition products (both hardware and software) in Chapter 11. The listing of hardware and software data acquisition product manufacturers in the Appendix is now twice the size it was in the original edition.

This book is intended as a tutorial and reference for engineers, scientists, students, and technicians interested in using a PC for data acquisition, analysis, and control applications. It is assumed that the reader knows the basic workings of PCs and electronic hardware, although these aspects will be briefly reviewed here. Several sources listed in the bibliography are good introductions to many of these topics (both hardware and software).

This book stresses "real" applications and includes specific examples. It is intended to provide all the information you need to use a PC as a data acquisition system. In addition, it serves as a useful reference on PC technology. Since the area of software is at least as important as hardware, if not more so, software topics (such as programming languages, interfacing to a PC's software environment, and data analysis techniques) are covered in detail.

I wish to acknowledge the help I received in writing this new edition. My thanks to Academic Press for patiently seeing this project through. I am grateful for the assistance I received from many manufacturers in the data acquisition field, including Keithley Instruments, Laboratory Technologies, and The Math Works. Finally, I want to acknowledge Orndorff, the laptop editor, who kept me company during all those late nights at my PC.

Howard Austerlitz

CHAPTER

1

Introduction to Data Acquisition

Data acquisition, in the general sense, is the process of collecting information from the real world. For most engineers and scientists these data are mostly numerical and are usually collected, stored, and analyzed with a computer. The use of a computer automates the data acquisition process, enabling the collection of more data in less time with fewer errors. This book deals solely with automated data acquisition and control using personal computers (PCs). We will primarily concern ourselves with IBM-style PCs based on Intel microprocessors (80x86 and Pentium families) running Microsoft operating systems (MS-DOS and Windows). In general, the information in this book is applicable to desktop, laptop, and embedded PCs. However, many plug-in PCI data acquisition cards will also work in newer Apple Macintosh computers, with appropriate software drivers. In addition, USB, IEEE-1394 (FireWire) and PCMCIA-based data acquisition hardware will work with any style of computer which supports that interface, as long as software drivers are available for that platform.

An illustrative example of the utility of automated data acquisition is measuring the temperature of a heated object versus time. Human observers are limited in how fast they can record readings (say, every second, at best) and how much data can be recorded before errors due to fatigue occur (perhaps after 5 minutes or 300 readings). An automated data acquisition system can easily record readings for very small time intervals (i.e., much less than a millisecond), continuing for arbitrarily long time periods (limited mainly by the amount of storage media available). In fact, it is easy to acquire *too much data*, which can complicate the subsequent analysis. Once the data are stored in a computer, they can be displayed graphically, analyzed, or otherwise manipulated.

Most real-world data are not in a form that can be directly recorded by a computer. These quantities typically include temperature, pressure, distance, velocity, mass, and energy output (such as optical, acoustic, and electrical energy). Very often these quantities are measured versus time or position. A physical quantity must first be converted to an electrical quantity (voltage, current, or resistance) using a *sensor* or *transducer*. This enables it to be conditioned by electronic instrumentation, which operates on analog signals or waveforms (a signal or waveform is an electrical parameter, most often a voltage, which varies with time). This analog signal is *continuous* and *monotonic*, that is, its values can vary over a specified range (for example, somewhere between –5.0 volts and +3.2 volts) and they can change an arbitrarily small amount within an arbitrarily small time interval.

To be recorded (and understood) by a computer, data must be in a digital form. Digital waveforms have *discrete* values (only certain values are allowed) and have a specified (usually constant) time interval between values. This gives them a "stepped" (noncontinuous) appearance, as shown by the digitized sawtooth in Figure 1-1. When this time interval becomes small enough, the digital waveform becomes a good approximation to the analog waveform (for example, music recorded digitally on a CD). If the transfer function of the transducer and the analog instrumentation is known, the digital waveform can be an accurate representation of the time-varying-quantity to be measured.

The process of converting an analog signal to a digital one is called analog-to-digital conversion, and the device that does this is an analog-to-digital converter (ADC). The resulting digital signal is usually an array of digital values of known range (scale factor) separated by a fixed time interval (or sampling interval). If the values are sampled at irregular time intervals, the acquired data will contain both value and time information.

The reverse process of converting digital data to an analog signal is called digital-to-analog conversion, and the device that does this is called a

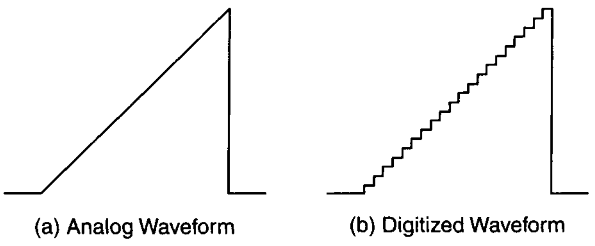


Figure 1-1 Comparison of analog and digitized waveforms: (a) sawtooth analog waveform with (b) a coarse digitized representation.

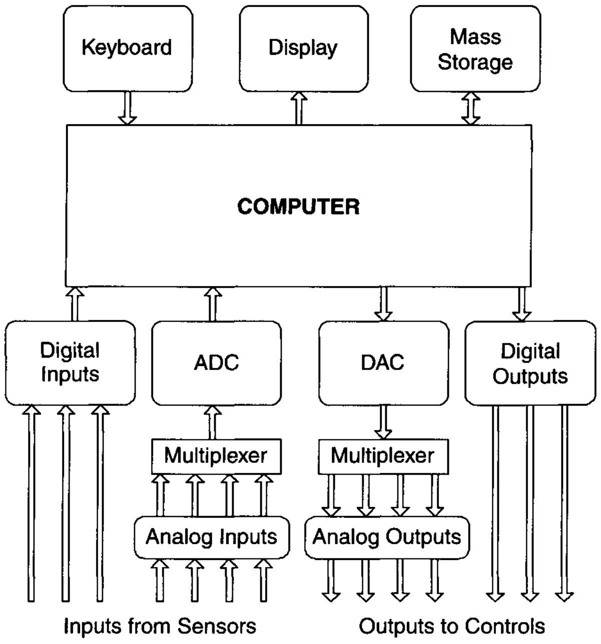


Figure 1-2 Simplified block diagram of a data acquisition system.

digital-to-analog converter (DAC). Some common applications for DACs include control systems, waveform generation, and speech synthesis.

A general-purpose laboratory data acquisition system typically consists of ADCs, DACs, and digital inputs and outputs. Figure 1-2 is a simplified block diagram of such a system. Note that additional channels are often added to an ADC via a *multiplexer* (or *mux*), used to select which one of the several analog input signals to convert at any given time. This is an economical approach when all the analog signals do not need to be simultaneously monitored.

Economics is a major rationale behind using PCs for data acquisition systems. The typical data acquisition system of 20–25 years ago, based on a minicomputer, cost about 20 times as much as today's systems, based on PCs, and ran at lower performance levels. This is largely due to the continuing decrease in electronic component costs along with increased functionality (more logic elements in the same package) and more sophisticated software. The PC has become ubiquitous throughout our society, both in and out of laboratories. Continuous improvements in hardware and software technologies drive PCs and their peripheral devices to lower costs and higher performance.

Since PCs are commonplace in most labs and offices, the cost of implementing a data acquisition system is often just the price of an add-in board (or module) and support software, which is usually just a moderate expense. For very simple applications, standard PC hardware (such as a sound card) may be all you need for data acquisition.

There may be applications where a data acquisition system based on a PC is not appropriate and a more expensive, dedicated system should be used. The important system parameters for making such a decision include sampling speed, accuracy, resolution, amount of data, multitasking capabilities, and the required data processing and display. Of course, dedicated data acquisition systems may be PC-based themselves, with an embedded PC (see Chapter 12 for information on embedded PCs).

PC-based systems have fewer limitations in these areas than ever before, even regarding sampling speed and handling large amounts of data. Newer, high-performance PCs can even outperform some dedicated data acquisition systems. The evolution of PCs based on the Intel 80x86 microprocessor (or CPU), which includes the original IBM PC/XT/AT/PS2 computers, is demonstrated in Table 1-1, showing processor speed, bus width, and the amount of available memory space.

Apple's Macintosh computer line has also been used as a platform for data acquisition. These machines, originally based on the Motorola 68000 family of microprocessors, had certain advantages over the older Intel-based PCs, including a graphical, consistent operating environment and a linear memory addressing space (the segmented addressing space of the Intel 80x86 family will be discussed in Chapter 5). Newer Macintosh computers use the same PCI interface for add-in cards as contemporary Intel-based PCs (see Chapter 5 for a discussion of the PCI bus).

TABLE 1-1
Intel 80x86 CPU Family Bus Size Characteristics

CPU	DATA BUS WIDTH (bits)	ADDRESS BUS WIDTH (bits)	MEMORY SPACE (Mbytes)
8086	16	20	1
8088	8	20	1
80286	16	24	16
80386	32	32	4096
80486	32	32	4096
Pentium	32*	32	4096

*Internal Bus = 64 bits wide.

Software is as important to data acquisition systems as hardware capabilities. Inefficient software can waste the usefulness of the most able data acquisition hardware system. Conversely, well-written software can squeeze the maximum performance out of mediocre hardware. Software selection is at least as important as hardware selection and often more complex.

Data acquisition software controls not only the collection of data, but also its analysis and eventual display. Ease of data analysis and presentation are the major reasons behind using computers for data acquisition in the first place. With the appropriate software, computers can process the acquired data and produce outputs in the form of tables or plots. Without these capabilities, you are not doing much more than using a sophisticated (and expensive) data recorder.

An additional area of software use is that of control. Computer outputs may control some aspects of the system that is being measured, as in automated industrial process controls. The software must be able to measure system parameters, make decisions based on those measurements, and vary the computer outputs accordingly. For example, in a temperature regulation system, the input would be a temperature sensor and the output would control a heater. In control applications, software reliability and response time are paramount. Slow or erroneous software responses could cause physical damage. Control applications are especially important for embedded PCs, which package full PC functionality into a small form factor, such as PC-104 (see Chapter 12).

A recent, important software capability is Internet access. Many new products allow you to perform remote data acquisition using the Internet (and its TCP/IP protocol). It is now fairly simple to monitor and control a data acquisition system located nearly anywhere in the world as well as share the data with a large group of colleagues.

There is a plethora of PC-based software packages commercially available, which can collect, analyze, and display data graphically, using little or no programming (see Chapter 11). They allow users to concentrate on their applications, instead of worrying about the mechanics of getting data from point A to point B, or how to plot a set of Cartesian coordinates. Many commercial software packages contain all three capabilities of data acquisition, analysis, and display (the so-called "integrated" packages), whereas others are optimized for only one or two of these areas.

The important point is that you do not have to be a computer expert or even a programmer to implement an entire PC-based data acquisition system. Best of all, you do not have to be rich, either.

The next chapter examines the world of analog signals and their transducers, the "front end" of any data acquisition system.

CHAPTER 2

Analog Signal Transducers

Most real-world events and their measurements are analog. That is, the measurements can take on a wide, nearly continuous range of values. The physical quantities of interest can be as diverse as heat, pressure, light, force, velocity, or position. To be measured using an electronic data acquisition system, these quantities must first be converted to electrical quantities such as voltage, current, or impedance.

A transducer converts one physical quantity into another. For the purposes of this book, all the transducers mentioned convert physical quantities into electrical ones, for use with electronic instrumentation. The mathematical description of what a transducer does is its transfer function, often designated H. So the operation of a transducer can be described as

$$\text{Output Quantity} = H \times \text{Input Quantity}$$

Since the transducer is the "front end" of the data acquisition system, its properties are critical to the overall system performance. Some of these properties are sensitivity (the efficiency of the energy conversion), stability (output drift with a constant input), noise, dynamic range, and linearity. Very often the transfer function is dependent on the input quantity. It may be a linear function for one range of input values and then become nonlinear for another range (such as a square-law curve). Looking at sensitivity and noise, if the transducer's sensitivity is too low, or its noise level too high, signal conditioning may not produce an adequate signal-to-noise ratio.

Often the transducer is the last consideration in a data acquisition system, since it is seen as mundane. Yet, it should be the primary consideration.

The characteristics of the transducer, in large part, determine the limits of a system's performance.

Now we will look at some common transducers in detail.

2.1 Temperature Sensors

Temperature sensors have electrical parameters that vary with temperature, following well-characterized transfer functions. In fact, nearly all electronic components have properties which vary with temperature. Many of them could potentially be temperature transducers if their transfer functions were well behaved and insensitive to other variables.

2.1.1 Thermocouples

The *thermocouple* converts temperature to a small DC voltage or current. It consists of two dissimilar metal wires in intimate contact in two or more junctions. The output voltage varies linearly with the temperature difference between the junctions—the higher the temperature difference, the higher the voltage output. This linearity is a chief advantage of using a thermocouple, as well as its ruggedness as a sensor. In addition, thermocouples operate over very large temperature ranges and at very high temperatures (some, over 1000°C).

Disadvantages include low output voltage (especially at lower temperatures), low sensitivity (typical output voltages vary only about 5 mV for a 100°C temperature change), susceptibility to noise (both externally induced and internally caused by wire imperfections and impurities), and the need for a reference junction (at a known temperature) for calibration. Most data acquisition hardware designed for temperature measurements contain an electronic reference junction. You must enter the thermocouple material type you are using, so it is properly calibrated. Common thermocouple materials include copper/constantan (Type T), iron/constantan (Type J), and chromel/alumel (Type K).

When several thermocouples, made of the same materials are combined in series, they are called a *thermopile*. The output voltage of a thermopile consists of the sum of all the individual thermocouple outputs, resulting in increased sensitivity. All the reference junctions are kept at the same temperature.

2.1.2 Thermistors

A *thermistor* is a temperature-sensitive resistor with a large, nonlinear, negative temperature coefficient. That is, its resistance decreases nonlinearly as temperature increases. It is usually composed of a mixture of semiconductor materials.

It is a very sensitive device, but has to be properly calibrated for the desired temperature ranges, since it is a nonlinear detector. Repeatability from device to device is not very good. Over relatively small temperature ranges it can approximate a linear response. It is prone to self-heating errors due to the power dissipated in it ($P = I^2R$). This effect is minimized by keeping the current passing through the thermistor to a minimum.

2.1.3 Resistance Temperature Detectors

Resistance temperature detectors (RTDs) rely on the temperature dependence of a material's electrical resistance. They are usually made of a pure metal having a small but accurate positive temperature coefficient. The most accurate RTDs are made of platinum wire and are well characterized and linear from 14°K to higher than 600°C.

2.1.4 Monolithic Temperature Transducers

The *monolithic temperature transducer* is a semiconductor temperature sensor combined with all the required signal conditioning circuitry and located in one integrated circuit. This device typically produces an output voltage proportional to the absolute temperature, with very good accuracy and sensitivity (a typical device produces an output of 10 mV per degree Kelvin over a temperature range of 0–100 degrees Celsius). The output of this device can usually go directly into an ADC with very little signal conditioning.

2.2 Optical Sensors

Optical sensors are used for detecting light intensity. Typically, they respond only to particular wavelengths or spectral bands. One sensor may respond only to visible light in the blue-green region, while another sensor may have a peak sensitivity to near-infrared radiation.

2.2.1 Vacuum Tube Photosensors

This class of transducers consists of special-purpose vacuum tubes used as optical detectors. They are all relatively large, require a high-voltage power supply to operate, and are used only in very specialized applications (as is true with vacuum tubes in general). These sensors exploit the photoelectric effect, when photons of light striking a suitable surface produce free electrons.

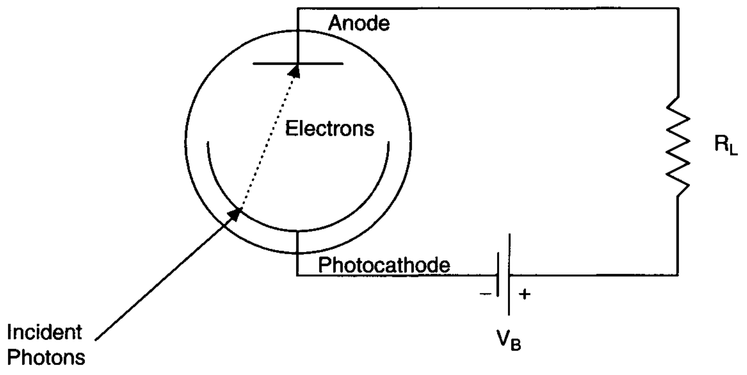


Figure 2-1 Vacuum photodiode.

The *vacuum photodiode* consists of a photocathode and anode in a glass or quartz tube. The photocathode emits electrons when struck by photons of light. These electrons are accelerated to the anode by the high (+) voltage and produce a current pulse in the external load resistor R_L (see Figure 2-1). These tubes have relatively low sensitivity, but they can detect high-frequency light variations or modulation (as high as 100 MHz to 1 GHz), for an extremely fast response.

The *gas photodiode* is similar to a vacuum photodiode, except the tube contains a neutral gas. A single photoelectron (emitted by the photocathode) can collide with several gas atoms, ionizing them and producing several extra electrons. So, more than one electron reaches the anode for every photon. This gas amplification factor is usually 3–5 (larger values cause instabilities). These tubes have a limited frequency response of less than 10 kHz, resulting in a much slower response time.

The *photomultiplier tube* (PMT) is the most popular vacuum tube device in this category. It is similar to a vacuum photodiode with several extra electrodes between the photocathode and anode, called *dynodes*. Each dynode is held at a more positive voltage than the previous dynode (and the cathode) via a resistor voltage-divider network (see Figure 2-2). Photoelectrons emitted by the photocathode strike the first dynode, which emits several secondary electrons for each photoelectron, amplifying the photoelectric effect. These secondary electrons strike the next dynode and release more electrons. This process continues until the electrons reach the end of the dynode amplifier chain. There, the anode collects all the electrons produced by a single photon, resulting in a relatively large current pulse in the external circuit.

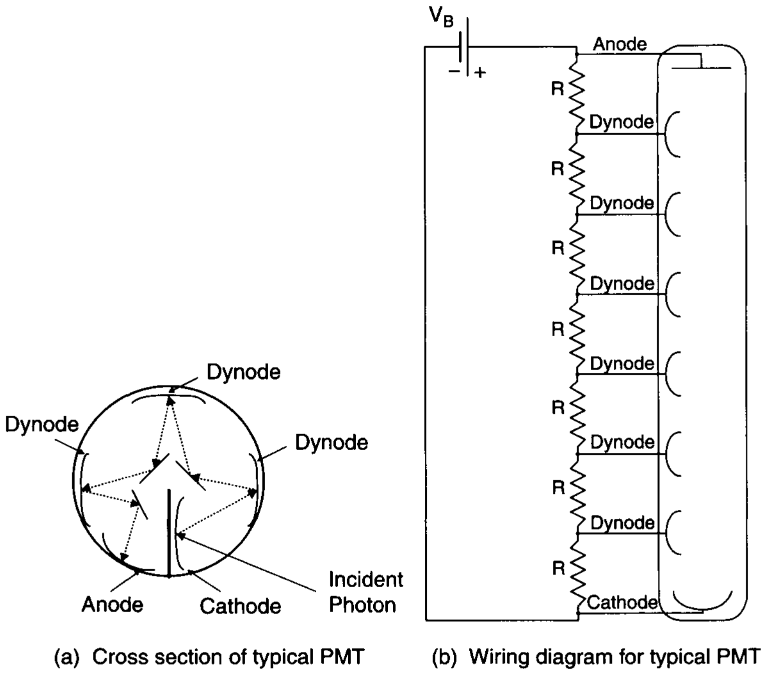


Figure 2-2 Photomultiplier tube (PMT).

The PMT exhibits very high gain, in the range of 10^5–10^7 electrons emitted per incident photon. This is determined by the number of dynodes, the photocathode sensitivity, power supply voltage, and tube design factors. Some PMTs can detect individual photons!

A PMT's output pulses can be measured as a time-averaged current (good for detecting relatively high light levels) or in an individual pulse-counting mode (good for very low light levels) measuring the number of pulses per second. Then, a threshold level is used to filter out unwanted pulses (noise) below a selected amplitude.

Some of the noise produced in a PMT is spontaneous emission from the electrodes, which occurs even in the absence of light. This is called the *dark count*, which determines the PMT's sensitivity threshold. So, the number of photons striking the PMT per unit time must be greater than the dark count for the photons to be detected. In addition, most PMTs have a fairly low *quantum efficiency*, a measure of how many photons are required to produce a measurable output (expressed as a percentage, where 100% means that every photon striking the sensor will produce an output). Also, PMTs have a limited usable life, as the photocathode wears out with time.

2.2.2 Photoconductive Cells

A photoconductive cell consists of a thin layer of material, such as cadmium sulfide (CdS) or cadmium selenide (CdSe) sandwiched between two electrodes, with a transparent window. The resistance of a cell decreases as the incident light intensity increases. These cells can be used with any resistance-measuring apparatus, such as a bridge. They are commonly used in photographic light meters. A photoconductive cell is usually classified by maximum (dark) resistance, minimum (light) resistance, spectral response, maximum power dissipation, and response time (or frequency).

These devices are usually nonlinear and have aging and repeatability problems. They exhibit hysteresis in their response to light. For example, the same cell exposed to the same light source may have a different resistance, depending on the light levels it was previously exposed to.

2.2.3 Photovoltaic (Solar) Cells

These sensors are similar in construction to photoconductive cells. They are made of a semiconductor material, usually silicon (Si) or gallium arsenide (GaAs), that produces a voltage when exposed to light (of suitable wavelength). They require no external power supply and very large cells can be used as DC power sources. They have a relatively slow response time to light variations but are fairly sensitive. Since the material used must be grown as a single crystal, large photovoltaic cells are very expensive.

A large amount of research has been conducted in recent years in an attempt to produce less expensive photovoltaic cells made from either amorphous, polycrystalline, or thin-film semiconductors. If these low-cost devices can attain light conversion efficiency similar to that of monocrystalline cells (in the range of 15–20%), they can become a practical source of electric energy.

2.2.4 Semiconductor Light Sensors

The members of this class of transducers are all based on a semiconductor device, such as a diode or transistor, whose output current is a function of the light (of suitable wavelength) incident upon it.

The *photodiode* is a PN junction diode with a transparent window that produces charge carriers (holes and electrons) at a rate proportional to the incident light intensity. So the photodiode acts as a photoconductive device, varying the current in its external circuit (but, being a semiconductor, it does not obey Ohm's law). A photodiode is a versatile device with a high frequency response and a linear output, but low sensitivity, and it usually requires large amounts of amplification. It typically uses a transconductance amplifier, which converts the photodiode output current to a voltage. A common photodiode sensor is the *PIN* diode, which has an insulating region between the *p* and *n* materials. This device usually requires a reverse DC bias voltage for optimum performance (speed and sensitivity). Conventional silicon photodiodes have usable sensitivity to light wavelengths in the range of 450–1050 nanometers (from the visible spectrum into the near infrared). For longer wavelengths, other semiconductors, such as indium gallium arsenide (InGaAs) are used.

The *phototransistor* is similar to a photodiode, except that the transistor can provide amplification of the PN junction's light-dependent current. The transistor's emitter–base junction is the light-sensitive element. A *photodarlington* is a special phototransistor, composed of two transistors in a high-gain circuit. The phototransistor offers much higher sensitivity than the photodiode at the expense of a much lower bandwidth (response time) and poorer linearity.

The *avalanche photodiode* (APD) is a special photodiode which has internal gain and is a semiconductor analog to the PMT. This gain is normally in the range of 10 to a few hundred (typically around 100 for a silicon device). The APD employs a high reverse bias (from several hundred volts up to a few thousand volts) to produce a strong internal electrical field that accelerates the electrons generated by the incident photons and results in secondary electrons from impact ionization. This is the electron avalanche, resulting in gain. Advantages of the APD are small size, solid-state reliability (as long as the breakdown voltage is not exceeded), high quantum efficiency, and a large dynamic range. Compared to PMTs, APDs have much lower gain, smaller light-collecting areas, and a high temperature sensitivity. APD bias must be temperature compensated to keep gain constant.

The *charge-coupled device* (CCD) is a special optical sensor consisting of an array (one- or two-dimensional) of light-sensitive elements. When photons strike a photosensitive area, electron/hole pairs are created in the semiconductor crystal. The holes move into the substrate and the electrons remain in the elements, producing a net electrical charge. The amount of charge is proportional to the amplitude of incident light and the exposure time. The charge at each photosensitive element is then read out serially, via support electronics. CCDs are commonly used in many imaging systems, including video cameras.

2.2.5 Thermoelectric Optical Sensors

This class of transducers convert incident light to heat and produce a temperature output dependent on light intensity, by absorbing all the incident radiation in a "black box." They generally respond to a very broad light spectrum and are relatively insensitive to wavelength, unlike vacuum tube and solid-state sensors. However, they have very slow response times and low sensitivities and are best suited for measuring static or slowly changing light levels, such as calibrating the output of a light source.

The *bolometer* varies its resistance with thermal energy produced by incident radiation. The most common detector element used in a bolometer is a thermistor. They are also commonly used for measuring microwave power levels.

The *thermopile*, as discussed under temperature sensors, is more commonly used than individual thermocouples in light-detecting applications because of its higher sensitivity. It is often used in infrared detectors.

2.3 Force and Pressure Transducers

A wide range of sensors are used for measuring force and pressure. Most pressure transducers rely on the movement of a diaphragm mounted across a pressure differential. The transducer measures this minute movement. Capacitive and inductive pressure sensors operate the same way as capacitive and inductive displacement sensors, which are described later on.

2.3.1 Strain Gages

Strain gages are transducers used for directly measuring forces and their resulting strain on an object. Stress on an object produces a mechanical deformation—strain—defined as

$$\text{Strain} = \text{length change}/\text{length}$$

Strain gages are conductors (often metallic) whose resistance varies with strain. For example, as a wire is stretched, its resistance increases. Strain gages are bonded to the object under stress and are subject to the same forces. They are very sensitive to strain in one direction only (the axis of the conductor).

A simple *unbonded strain gage* consists of free wires on supports bonded to the stressed surface. These are not usually used (outside of laboratory demonstrations) because of their large size and mechanical clumsiness.

The *bonded strain gage* overcomes these problems by putting a zigzag pattern of the conductor on an insulating surface, as shown in Figure 2-3. These are relatively small, have good sensitivity, and are easily bonded to the surface under test. The conductor in a bonded strain gage is a metallic wire, foil, or thin film.

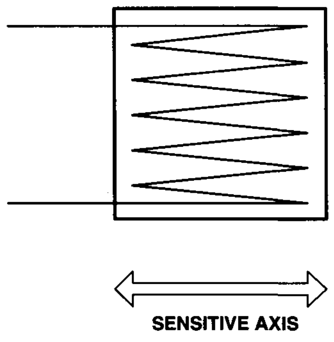


Figure 2-3 Simple, one-dimensional strain gage.

Strain gage materials must have certain, well-controlled properties. The most important is sensitivity or gage factor (GF), which is the change in resistance per change in length. Most metallic strain gages have a GF in the range of 2 to 6. The material must also have a low temperature coefficient of resistance as well as stable elastic properties and high tensile strength. Often, strain gages are subject to very large stresses as well as wide temperature swings.

Semiconductor strain gages, usually made of silicon, have a much higher GF than metals (typically in the range of 50 to 200). However, they also have much higher temperature coefficients, which have to be compensated for. They are commonly used in monolithic pressure sensors.

Because of their relatively low sensitivities (resistance changes nominally 0.1 to 1.0%), strain gages require bridge circuits to produce useful outputs. (We will discuss bridge circuits in Chapter 3.) If a second, identical strain gage, not under stress, is put into the bridge circuit, it acts as a temperature compensator.

2.3.2 Piezoelectric Transducers

Piezoelectric transducers are used for, among other things, measuring time-varying forces and pressures. They do not work for static measurement, since they produce no output from a constant force or pressure.

Certain crystalline materials (including quartz, barium titanate, and lithium niobate) generate an electromotive force (emf) when mechanically stressed. Conversely, when a voltage is applied to the crystal, it will become mechanically distorted. This is the piezoelectric effect.

If electrodes are placed on suitable (usually opposite) faces of the crystal, the direction of the deforming force can be controlled. If an AC voltage is applied to the electrodes, the crystal can produce periodic motion, resulting

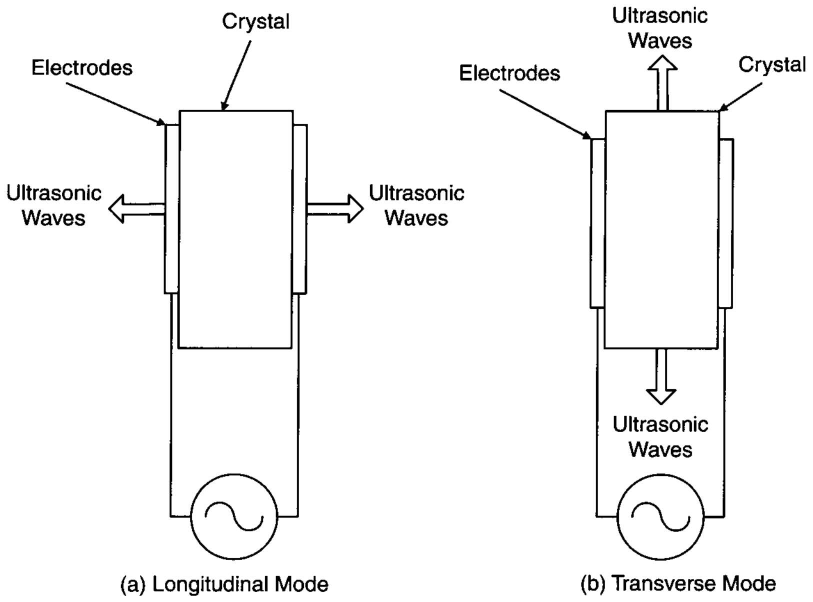


Figure 2-4 Oscillation modes of piezoelectric crystals.

in an acoustic wave, which can be transmitted through other material. When an acoustic wave strikes a piezoelectric crystal, it produces an AC voltage.

When a piezoelectric crystal oscillates in the thickness or longitudinal mode, an acoustic wave is produced, where the direction of displacement is the direction of wave propagation, as shown in Figure 2-4a. When the crystal's thickness equals a half-wavelength of the longitudinal wave's frequency (or an odd multiple half-wavelength) it is resonant at that frequency. At resonance its mechanical motion is maximum along with the acoustic wave output. And when it is detecting acoustic energy, the output voltage is maximum for the resonant frequency.

This characteristic is applied to quartz crystal oscillators, used as highly accurate electronic frequency references in a broad range of equipment, from computers to digital watches.

Typically, piezoelectric crystals are used as ultrasonic transducers for frequencies above 20 kHz, up to about 100 MHz. The limitation on frequency range is due to the impracticalities of producing crystals thin enough for very high frequencies, or the unnecessary expense of producing very thick crystals for low frequencies (where electromagnetic transducers work better).

Other crystal deformation modes are transverse, where the direction of motion is at right angles to the direction of wave propagation (as shown in Figure 2-4b), and shear, which is a mix of longitudinal and transverse modes. These modes all have different resonant frequencies.

Piezoelectric transducers have a wide range of applications, besides dynamic pressure and force sensing, including the following:

1. Acoustic microscopy for medical and industrial applications, such as "seeing" through materials that are optically opaque. An example is the sonogram.
2. Distance measurements including sonar and range finders.
3. Sound and noise detection such as microphones and loudspeakers for audio and ultrasonic acoustic frequencies.

2.4 Magnetic Field Sensors

This group of transducers is used to measure either varying or fixed magnetic fields.

2.4.1 Varying Magnetic Field Sensors

These transducers are simple inductors (coils) that can measure time-varying magnetic fields such as those produced from an AC current source. The magnetic flux through the coil changes with time, so an AC voltage is induced that is proportional to the magnetic field strength.

These devices are often used to measure an alternating current (which is proportional to the AC magnetic field). For standard 60-Hz loads, transformers are used that clamp around a conductor (no direct electrical contact). These are usually low-sensitivity devices, good for 60 Hz currents greater than 0.1 ampere.

2.4.2 Fixed Magnetic Field Sensors

Several types of transducers are commonly used to measure static and slowly varying magnetic fields, such as those produced by a permanent magnet or a DC electromagnet.

Hall Effect Sensors When a current-carrying conductor strip is placed with its plane perpendicular to an applied magnetic field (B) and a control current (I_C) is passing through it, a voltage (V_H) is developed across the strip at right

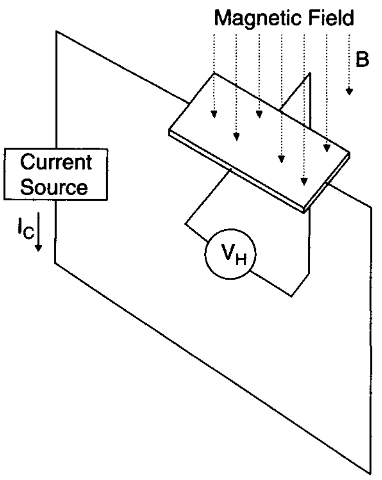


Figure 2-5 Hall effect magnetic field sensor.

angles to I_C and B, as shown in Figure 2-5. V_H is known as the Hall voltage and this is the Hall effect:

$$V_H = KI_CB/d$$

where:

B = magnetic field (in gauss),
d = thickness of strip,
K = Hall coefficient.

The value of K is very small for most metals, but relatively large for certain n-type semiconductors, including germanium, silicon, and indium arsenide. Typical outputs are still just a few millivolts/kilogauss at rated I_C. Although a larger I_C or a smaller d should increase V, these would cause excessive self-heating of the device (by increasing its resistance) and would change its characteristics as well as lower its sensitivity. The resistance of typical Hall devices varies from a few ohms to hundreds of ohms.

SQUIDs *SQUID* stands for superconducting quantum interference device, a superconducting transducer based on the *Josephson junction*. A SQUID is a thin-film device operating at liquid helium temperature (~4°K), usually made from lead or niobium. The advent of higher temperature superconductors that can operate in the liquid nitrogen region (~78°K) may produce more practical and inexpensive SQUIDs.

A SQUID element is a Josephson junction that is based on quantum mechanical tunneling between two superconductors. Normally, the device is superconducting, with zero resistance, until an applied magnetic field switches it into a normal conducting state, with some resistance. If an external current is applied to the device (and it must be low enough to prevent the current from switching it to a normal conductive state—another Josephson junction property), the voltage across the SQUID element switches between zero and a small value. The resistance and measured voltage go up by steps (or quanta) as the applied magnetic field increases. It measures very small, discrete (quantum) changes in magnetic field strength.

Practical SQUIDs are composed of arrays of these individual junctions and are extremely sensitive magnetometers. For example, they are used to measure small variations in the earth's magnetic field, or even magnetic fields generated inside a living brain.

2.5 Ionizing Radiation Sensors

Ionizing radiation can be particles produced by radioactive decay, such as alpha or beta radiation, or high-energy electromagnetic radiation, including gamma and X-rays. In many of these detectors, a radiation particle (a photon) collides with an active surface material and produces charged particles, ions, and electrons, which are then collected and counted as pulses (or events) per second or measured as an average current.

2.5.1 Geiger Counters

When the electric field strength (or voltage) is high enough in a gas-filled tube, electrons produced by primary ionization gain enough energy between collisions to produce secondary ionization and act as charge multipliers. In a *Geiger–Muller tube* the probability of this secondary ionization approaches unity, producing an avalanche effect. So, a very large current pulse is caused by one or very few ionizing particles. The Geiger–Muller tube is made of metal and filled with low-pressure gas (at about 0.1 atm) with a fine, electrically isolated wire running through its center, as shown in Figure 2-6.

A Geiger counter requires a recovery time (dead time) of ~200 microseconds before it can produce another discharge (to allow the ionized particles to neutralize). This limits its counting rate to less than a few kilohertz.

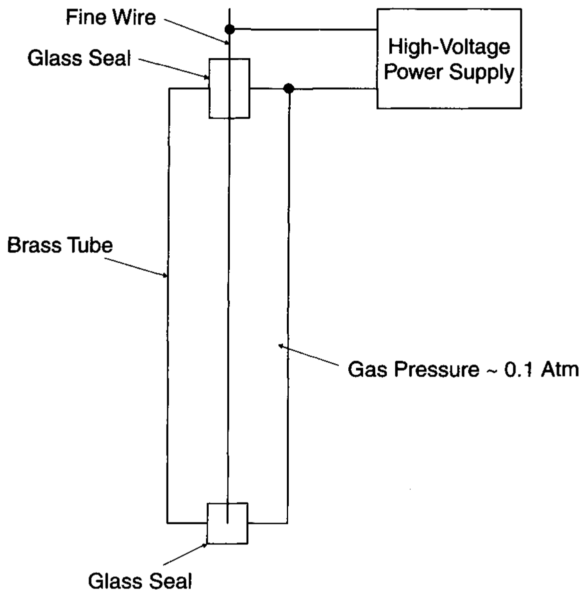


Figure 2-6 Typical Geiger–Muller tube.

2.5.2 Semiconductor Radiation Detectors

Some *p–n* junction devices (typically diodes), when properly biased, can act as solid-state analogs of an ion chamber, where a high DC voltage across a gas-filled chamber produces a current proportional to the number of ionizing particles striking it per unit time, due to primary ionization. When struck by radiation the devices produce charge carriers (electrons and holes) as opposed to ionized particles. The more sensitive (and useful) devices must be cooled to low temperatures (usually 78°K, by liquid nitrogen).

2.5.3 Scintillation Counters

This device consists of a fluorescent material that emits light when struck by a charged particle or radiation, similar to the action of a photocathode in a photodiode. The emitted light is then detected by an optical sensor, such as a PMT.

2.6 Position (Displacement) Sensors

A wide variety of transducers are used to measure mechanical displacement or the position of an object. Some require actual contact with the measured object; others do not.

2.6.1 Potentiometers

The *potentiometer* (variable resistor) is often mechanically coupled for displacement measurements. It can be driven by either AC or DC signals and does not usually require an amplifier. It is inexpensive but cannot usually be used in high-speed applications. It has limited accuracy, repeatability, and lifetime, due to mechanical wear of the active resistive material. These devices can either be conventional rotary potentiometers or have a linear configuration with a slide mechanism. Often, the resistive element is polymer-based to increase its usable life.

2.6.2 Capacitive and Inductive Sensors

Simple *capacitive* and *inductive* sensors produce a change in reactance (capacitance or inductance) with varying distance between the sensor and the measured object. They require AC signals and conditioning circuitry and have limited dynamic range and linearity. They are typically used over short distances as a *proximity sensor*, to determine if an object is present or not. They do not require contact with the measured object.

2.6.3 LVDTs

The LVDT (*linear voltage differential transformer*) is a versatile device used to measure displacement. It is an inductor consisting of three coils wound around a movable core, connected to a shaft, as shown in Figure 2-7. The center coil is the transformer's primary winding. The two outer coils are connected in series to produce the secondary winding. The primary is driven by an AC voltage, typically between 60 Hz and several kilohertz. At the null point (zero displacement), the core is exactly centered under the coils and the secondary output voltage is zero. If the shaft moves, and the core along with it, the output voltage increases linearly with displacement, as the inductive coupling to the secondary coils becomes unbalanced. A movement to one side of the null produces a 0° phase shift between output and input signal. A movement to the other side of null produces a 180° phase shift.

If the displacement is kept within a specified range, the output voltage varies linearly with displacement. The main disadvantages to using an LVDT are its size, its complex control circuitry, and its relatively high cost.

2.6.4 Optical Encoders

The *optical encoder* is a transducer commonly used for measuring rotational motion. It consists of a shaft connected to a circular disc, containing one or more tracks of alternating transparent and opaque areas. A light source and

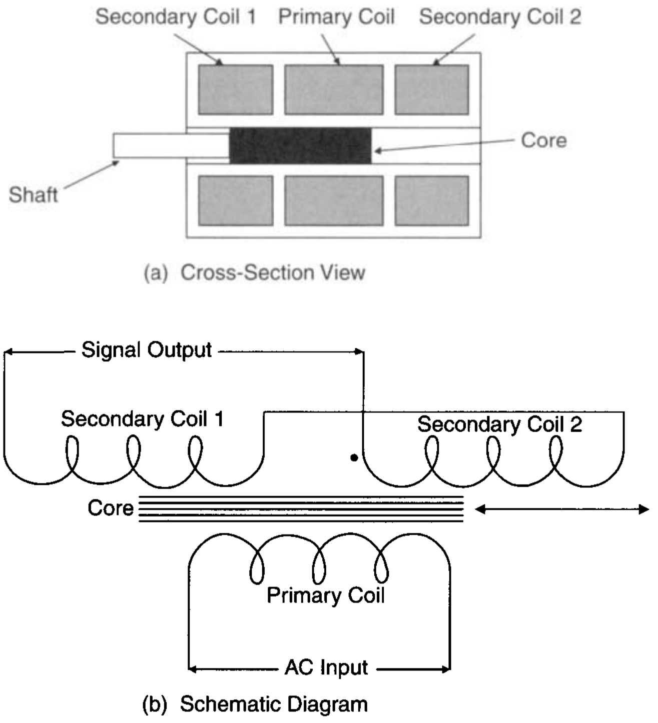


Figure 2-7 Linear variable differential transformer (LVDT).

an optical sensor are mounted on opposite sides of each track. As the shaft rotates, the light sensor emits a series of pulses as the light source is interrupted by the pattern on the disc. This output signal can be directly compatible with digital circuitry. The number of output pulses per rotation of the disc is a known quantity, so the number of output pulses per second can be directly converted to the rotational speed (or rotations per second) of the shaft. Encoders are commonly used in motor speed control applications. Figure 2-8 shows a simple, one-track encoder wheel.

An *incremental optical encoder* has two tracks, 90° out of phase with each other, producing two outputs. The relative phase between the two channels indicates whether the encoder is rotating clockwise or counterclockwise. Often there is a third track that produces a single index pulse, to indicate an absolute position reference. Otherwise, an incremental encoder produces only *relative* position information. The interface circuitry or computer must keep track of the absolute position.

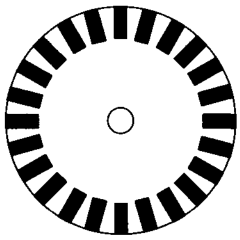


Figure 2-8 Simple one-track optical encoder wheel (24 lines = 15° resolution).

An *absolute optical encoder* has several tracks, with different patterns on each, to produce a binary code output that is unique for each encoded position. There is a track for each output bit, so an 8-bit absolute encoder has 8 tracks, 8 outputs and 256 output combinations, for a resolution of 360/256 = 1.4°. The encoding is not always a simple binary counting pattern, since this would result in adjacent counts where many bits change at once, increasing the likelihood of noise and reading errors. A Gray code is often used, because it produces a pattern where each adjacent count results in only one bit change. An absolute encoder is usually much more expensive than a comparable incremental encoder. Its main advantage is the ability to retain absolute position information, even when system power is removed.

2.6.5 Ultrasonic Range Finder

In Chapter 14, an *ultrasonic range finder* is discussed, as a noncontact displacement measurement technique. The time it takes an ultrasonic pulse to reflect from an object is measured and the distance to the object calculated from that time delay, using a known ultrasonic velocity.

2.7 Humidity Sensors

Relative humidity is the moisture content of the air compared to air completely saturated with moisture and is expressed as a percentage.

2.7.1 Resistive Hygrometer Sensors

There are *resistive hygrometer elements* whose resistance varies with the vapor pressure of water in the surrounding atmosphere. They usually contain a hygroscopic (water-absorbing) salt film, such as lithium chloride, which ionizes in water and is conductive with a measurable resistance. These devices are usable over a limited humidity range and have to be periodically calibrated, as their resistance may vary with time, because of temperature and humidity cycling, as well as exposure to contaminating agents.

2.7.2 Capacitive Hygrometer Sensors

There are also *capacitive hygrometer elements* that contain a hygroscopic film whose dielectric constant varies with humidity, producing a change in the device's capacitance. Some of these can be more stable than the resistive elements. The capacitance is usually measured using an AC bridge circuit.

2.8 Fluid Flow Sensors

Many industrial processes use fluids and need to measure and control their flow in a system. A wide range of transducers and techniques are commonly used to measure fluid flow rates (expressed as volume per unit time passing a point).

2.8.1 Head Meters

A *head meter* is a common device, where a restriction is placed in the flow tube producing a pressure differential across it. This differential is measured by a pair of pressure sensors and converted to a flow measurement. The pressure transducers can be any type, such as those previously discussed. The restriction devices include the orifice plate, the venturi tube, and the flow nozzle.

2.8.2 Rotational Flowmeters

Rotational flowmeters use a rotating element (such as a turbine) which is turned by the fluid flow. Its rotational rate varies with fluid flow rate. The turbine blades are usually made of a magnetized material so that an external magnetic pickup coil can produce an output voltage pulse each time a blade passes under it.

2.8.3 Ultrasonic Flowmeters

Ultrasonic flowmeters commonly use a pair of piezoelectric transducers mounted diagonally across the fluid flow path. The transducers act as a transmitter and a receiver (a multiplexed arrangement), measuring the velocity of ultrasonic pulses traveling through the moving fluid. The difference in the ultrasonic frequency between the "upstream" and "downstream" measurements is a function of the flow rate, due to the Doppler effect. Alternately, small time delay differences between the "upstream" and "downstream" measurements can be used to determine flow rate.

2.9 Fiber Optic Sensors

A new class of sensors, based on optical fibers, is emerging from laboratories throughout the world. These fiber optic sensors are used to measure a wide range of quantities, including temperature, pressure, strain, displacement, vibration, and magnetic field, as well as sensing chemical and biomedical materials. They are immune from electromagnetic interference (EMI), can operate in extremely harsh environments, can be very small, and are fairly sensitive. They are even embedded into large structures (such as bridges and buildings) to monitor mechanical integrity.

Inherently, fiber optic sensors measure optical amplitude, phase, or polarization properties. In a practical sensor, one or more of these parameters varies with the physical quantity of interest (pressure, temperature, etc.). The simplest fiber optic sensors are based on optical amplitude variations. These sensors require a reference channel to minimize errors due to long-term drift and light source variations. Sensors that measure optical phase or frequency employ an interferometer. These interferometric sensors offer much better sensitivity, resolution, and stability than simpler amplitude-based sensors. In addition, they are insensitive to fiber length. That is why they are the most commonly used type of fiber optic sensor.

2.9.1 Fiber Optic Microbend Sensor

This type of fiber optic sensor is commonly used to measure pressure, displacement, and vibration. An optical fiber is sandwiched between two rigid plates with a wavy profile, as shown in Figure 2-9. This produces microbends

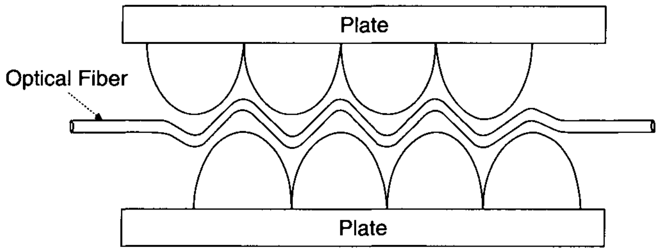


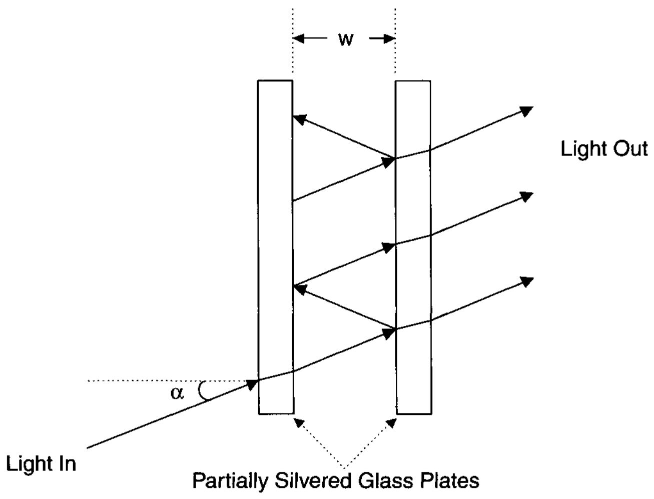
Figure 2-9 Fiber optic microbend sensor.

in the fiber, which cause light loss and decreased amplitude. A change in distance between the plates varies the magnitude of these bends and thus modulates the light intensity.

2.9.2 Fiber Optic Fabry–Perot Interferometric Sensor

The Fabry–Perot etalon is the most common interferometer structure used as a fiber optic sensor, since only one fiber is required to connect the sensor to the detector section. A classic Fabry–Perot interferometer is formed by two closely spaced, partially reflecting mirrors which form a resonant optical cavity with maximum optical transmission at wavelengths that are multiples of the mirror spacing, at small incident light angles (see Figure 2-10).

In a fiber sensor, a Fabry–Perot etalon can be formed using one end of the fiber itself (with a reflective coating deposited on it) and a separate, movable mirror. Alternatively, two mirrored surfaces can be used, and the fiber simply transmits the light. When the position of a moveable mirror in



Maximum Light Transmission when:

$$n\lambda = 2\,w\cos(\alpha)$$

n = an integer
λ = wavelength of light

Figure 2-10 Fabry–Perot interferometer.

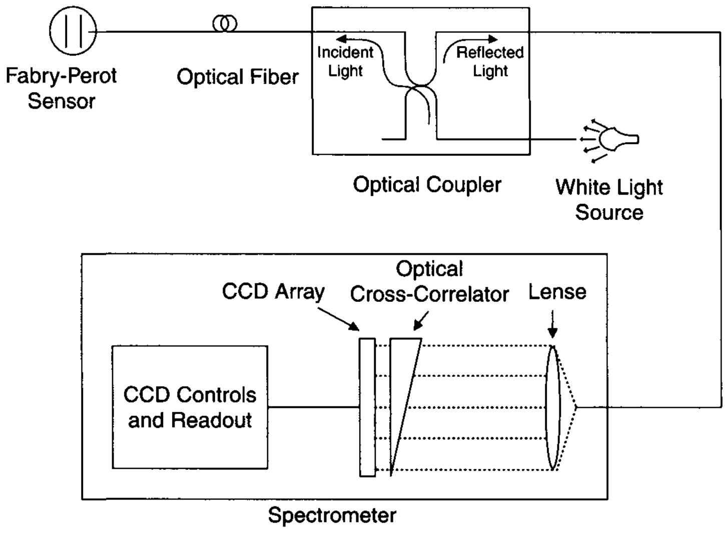


Figure 2-11 Fabry–Perot fiber sensor and detector.

the etalon changes, the intensity of light reflected back up the fiber changes, for a fixed wavelength, narrow-band light source. With a broad-band light source (i.e., white light), the peak wavelength shifts with mirror position and can be measured using a spectrometer detector. A simplified system diagram of a Fabry–Perot fiber sensor, commercially used for pressure and strain measurements, is shown in Figure 2-11.

2.10 Other New Sensor Technologies

Besides fiber optics, other new technologies are gaining importance in commercial sensors. These include *microelectromechanical systems* (MEMS) and smart sensors.

2.10.1 MEMS

MEMS are small electromechanical devices fabricated using semiconductor integrated-circuit processing techniques. By building a "micromachine" on a silicon wafer, the device can connect to signal processing electronics on that same wafer. Many of the sensors we have previously discussed have MEMS-based versions available. Sophisticated demonstrations of MEMS have included devices such as micromotors and gas chromatographs. Practical

MEMS pressure sensors and accelerometers have been commercially available for several years.

For example, Analog Devices' ADXL series of MEMS accelerometers are based on a structure suspended on the surface of a silicon wafer via polysilicon springs, which provide resistance to acceleration. Under acceleration, the structure deflects and this is measured via an arrangement of capacitors, fabricated using both fixed plates and plates attached to the moving structure. Signal generating and conditioning circuitry on the chip decodes this capacitance change to produce an output pulse with a duty cycle proportional to the measured acceleration.

2.10.2 Smart Sensors and the IEEE 1451 Standards

The category of smart sensors is quite broad and not clearly defined. A smart sensor can range from a traditional transducer that simply contains its own signal conditioning circuitry to a device that can calibrate itself, acquire data, analyze it, and transmit the results over a network to a remote computer. There are many commercial devices that can be called smart sensors, such as temperature sensor ICs that incorporate high and low temperature set points (to control heating or cooling devices). Many sensors, including pressure sensors, are now available with an RS-232C interface (see Chapter 8) to receive configuration commands and transmit measurements back to a host computer.

An emerging class of smart sensors is defined by the family of IEEE 1451 standards, which are designed to simplify the task of establishing communications between transducers and networks.

IEEE 1451.2 is an adopted standard in this group that defines transducer-to-microcontroller and microcontroller-to-network protocols. This standard defines a Smart Transducer Interface Module (STIM), which is a remote, networked, intelligent transducer node, supporting from 1 to 255 sensor and actuator channels. This STIM contains a Transducer Electronic Datasheet (TEDS), which is a section of memory that describes the STIM and its transducer channels. The STIM communicates with a microcontroller in a Network Capable Application Processor (NCAP) via the Transducer Independent Interface (TII), which is a 10-wire serial bus. Figure 2-12 shows how these parts of the IEEE 1451.2 standard fit together in a typical application.

The TEDS is a key element of the IEEE 1451.2 standard. It describes the transducer type for each channel, timing requirements, data format, measurement limits, and whether calibration information is present in the STIM. This information is read by the microcontroller in the NCAP, through the TII connection. Among other functions, the NCAP can write correction

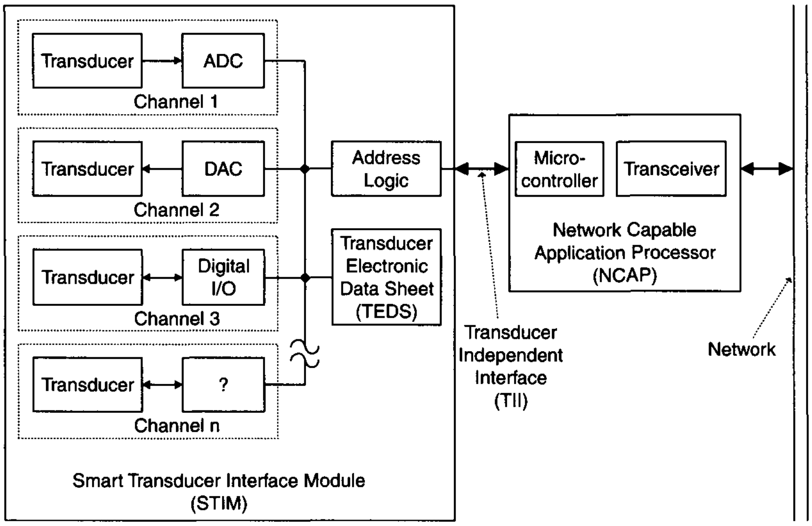

Transducer
ADC
Channel 1
Transducer
DAC
Channel 2
Transducer
Digital I/O
Channel 3
Transducer
?
Channel n
Address Logic
Transducer Electronic Data Sheet (TEDS)
Smart Transducer Interface Module (STIM)
Micro-controller
Transceiver
Network Capable Application Processor (NCAP)
Transducer Independent Interface (TII)
Network

Figure 2-12 IEEE 1451.2 smart transducer interface standard.

coefficients into the TEDS and read sensor data from the STIM. The read data is then sent to a remote computer on the network, via the NCAP. The NCAP definition is network independent. There are already commercial NCAPs available that work with RS-485 and Ethernet networks.

Some other early commercial IEEE 1451.2 products are STIMs and STIM-ready ICs. An example of the later is Analog Devices' ADuC812 MicroConverter. It is a special-purpose microcontroller containing an ADC, two DACS, both program and data flash EEPROM, and data RAM. It contains the logic to implement a TII, memory for TEDS storage, a multiplexer for up to eight transducer channels, and the circuitry to convert data from those analog channels.

This survey of common transducers and sensors suitable for a data acquisition system is hardly exhaustive. It should give you a feel for the types of devices and techniques applied to various applications and help you determine the proper transducer to use for your own system.

CHAPTER 3

Analog Signal Conditioning

Nearly all transducer signals must be conditioned by analog circuitry before they can be digitized and used by a computer. This conditioning often includes amplification and filtering, although more complex operations can also be performed on the waveforms.

3.1 Signal Conditioning Techniques

Amplification (or occasionally attenuation) is necessary for the signal's amplitude to fit within a reasonable portion of the ADC's dynamic range. For example, let us assume an ADC has an input range of 0–5 V and an 8-bit output of $2^8 = 256$ steps. Each output step represents 5/256 = 19.5 mV. If a sensor produces a waveform of 60 mV peak-to-peak (p–p), when directly digitized (by this ADC) it will use only 3 of the 256 available output steps and be severely distorted. If the sensor signal is first amplified by a factor of 83 (producing a 5 V p–p waveform), it will use the ADC's full dynamic range and a minimum of information is lost. Of course, if it is amplified too much, some of the signal will be clipped and severely distorted, now in a different way.

Filtering must usually be performed on analog signals for several reasons. Sometimes noise or unwanted signal artifacts can be eliminated by filtering out certain portions of the signal's spectra. For example, a system with high gain levels may need a 60 Hz notch filter to remove noise produced by AC power lines. A low-frequency drift on a signal without useful DC information can be removed using a high-pass filter. Most often, low-pass filters are employed to limit the high end of a waveform's frequency response just prior to digitization, to prevent aliasing problems (which will be discussed in Chapter 4).

Additional analog signal processing functions include modulation, demodulation, and other nonlinear operations.

3.2 Analog Circuit Components

The simplest analog circuit elements are passive components: resistors, capacitors, and inductors. They can be used as attenuators and filters. For example, a simple RC circuit can be used as a high-pass or low-pass filter, as shown in Figure 3-1.

Discrete semiconductor devices, such as diodes and transistors, are commonly used in analog signal-conditioning circuits. Diodes are useful, among other things, as rectifiers/detectors, switches, clamps, and mixers. Transistors are often used as amplifiers, switches, oscillators, phase shifters, filters, and many other applications.

3.2.1 The Operational Amplifier

The most common analog circuit semiconductor component is the *operational amplifier*, called the op amp. This circuit element is usually a monolithic device (an integrated circuit), although hybrid modules, based on discrete transistors, are still used in special applications. The op amp is used in both linear and nonlinear applications involving amplification and signal conditioning.

The "classic" op amp, which we will discuss in detail here, is based on a voltage-feedback architecture. There is a newer class of amplifiers, based

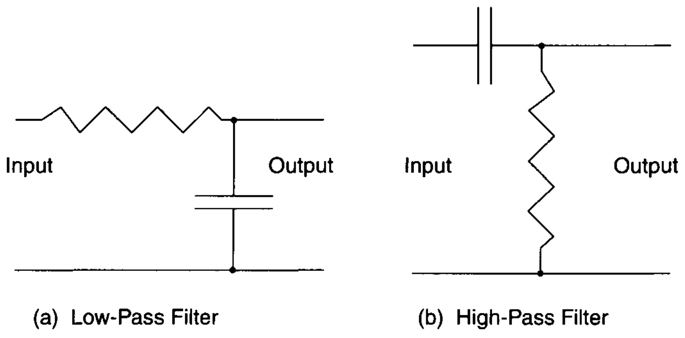


Figure 3-1 Simple RC filters.

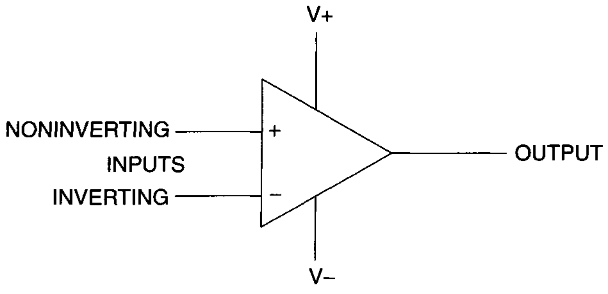


Figure 3-2 The operational amplifier (op amp).

on a current-feedback architecture, which we will cover later in this chapter while discussing high-frequency circuits.

An op amp, shown in Figure 3-2, consists of a differential voltage amplifier that can operate at frequencies from zero up to several megahertz. However, there are special high-frequency amplifiers, usable up to several hundred megahertz. The op amp has two inputs, called noninverting (+) and inverting (–), and responds to the voltage *difference* between them. The part of the output derived from the (+) source is in phase with the input, while the part from the (–) source is 180° out of phase. If a signal is equally applied to both inputs, the output will be zero.

This property is called *common-mode rejection*. Since an op amp can have very high gain at low frequencies (100,000 is typical), a high common-mode rejection ratio (CMRR) prevents amplification of unwanted noise, such as the ubiquitous 60-Hz power-line frequency. Typical op amps have a CMRR in the range of 80–100 decibels (dB).

Most op amps are powered by dual, symmetrical supply voltages, $+V$ and $-V$ relative to ground, where V is typically in the range of 3 to 15 volts. Some units are designed to work from single-ended supplies ($+V$ only). There are low-voltage, very low power op amps designed for use in battery-operated equipment. Op amps have very high input impedance at the + input (typically 1 million ohms or more) and low output impedance (in the range of 1 to 100 ohms). A voltage-feedback op amp's gain decreases with signal frequency, as shown in Figure 3-3. The point on the gain-versus-frequency curve where its gain reaches 1 is called its *unity-gain frequency*, which is equal to its *gain–bandwidth product*, a constant above low frequencies.

The op amp is more than a differential amplifier, however. Its real beauty lies in how readily its functionality can be changed by modifying the components in its external circuit. By changing the elements in the feedback loop

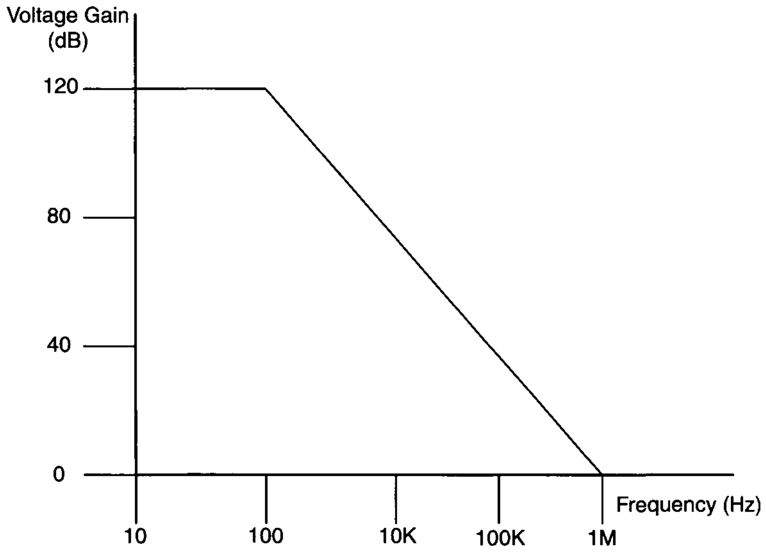


Figure 3-3 Typical op amp gain-versus-frequency curve.

(connected between the output and one or both inputs), the entire characteristics of the circuit are changed both quantitatively and qualitatively. The op amp acts like a servo loop, always trying to adjust its output so that the difference between its two inputs is zero.

We will examine some common op amp applications here. The reader should refer to the bibliography for other books that treat op amp theory and practice in greater depth.

The simplest op amp circuit is the *voltage follower* shown in Figure 3-4. It is characterized by full feedback from the output to the inverting (−) input, where the output is in phase with the noninverting (+) input. It is a buffer with very high input impedance and low output impedance. If the op amp

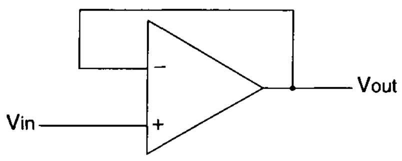


Figure 3-4 Op amp voltage follower.

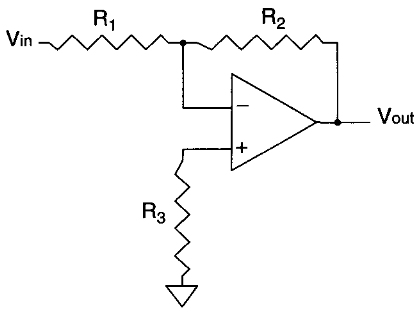


Figure 3-5 Op amp inverting amplifier.

has JFET (junction field effect transistor) inputs, its input impedance is extremely high (up to 10^{13} ohms).

The *inverting amplifier* shown in Figure 3-5 uses feedback resistor R_2 with input resistor R_1 to produce a voltage gain of R_2/R_1 with the output signal being the inverse of the input. Resistor R_3, used for DC balance, should be approximately equal to the parallel resistance combination of R_1 and R_2. Here, the input impedance is primarily determined by the value of R_1, since the op amp's (−) input acts as a virtual ground.

The *noninverting amplifier* shown in Figure 3-6 uses feedback resistor R_2 with grounded resistor R_1 to produce a voltage gain of $(R_1 + R_2)/R_1$ with the output following the shape of the input (hence, noninverting). Unlike the inverting amplifier, which can have an arbitrarily small gain well below 1, the noninverting amplifier has a minimum gain of 1 (when $R_2 = 0$). In this case, the input impedance is very high (typically from 10^7 to 10^9 ohms), as determined by the op amp's specification.

The *difference amplifier* shown in Figure 3-7 produces an output proportional to the difference between the two input signals. If $R_1 = R_2$ and $R_3 = R_4$, then the output voltage is $(V_{\text{in2}} - V_{\text{in1}}) \times (R_3/R_1)$.

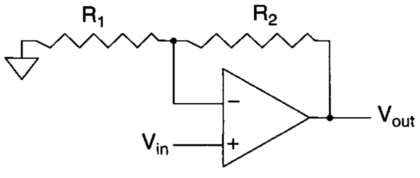


Figure 3-6 Op amp noninverting amplifier.

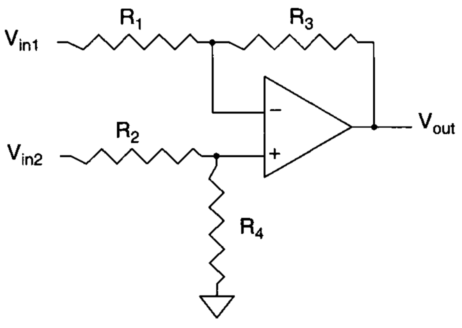


Figure 3-7 Op amp difference amplifier.

In the simple *integrator* shown in Figure 3-8, the feedback element is a capacitor (C), producing a nonlinear response. Resistor R_1 and capacitor C have a time constant R_1C. The change in output voltage with time $(dV_{out}/dt) = -V_{in}/(R_1C)$. Put another way, the output voltage is the integral of $-V_{in}/(R_1C)dt$. So, this circuit integrates the input waveform. For example, a square-wave input will produce a triangle-wave output, as long as the integrator's time constant is close to the period of the input waveform.

Similarly, Figure 3-9 shows a simple *differentiator*, where the positions of the resistor and capacitor are reversed from those in the integrator circuit. Here, the output voltage is $R_1C(dV_{in}/dt)$.

More complex op amp circuits include oscillators (both fixed-frequency and voltage-controlled oscillators or VCOs), analog multipliers and dividers (used in analog computers and modulation circuits), active filters, precision diodes, peak detectors, and log generators.

When choosing an op amp for a particular application, there are many factors to consider, such as frequency response, required gain, power supply voltage, and output current. Some other important specifications include input offset voltage and input bias current.

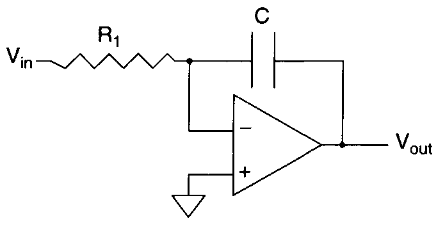


Figure 3-8 Op amp integrator.

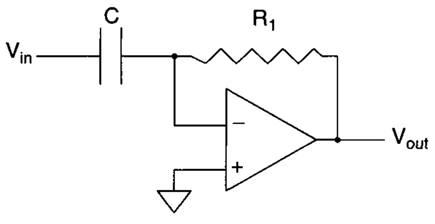


Figure 3-9 Op amp differentiator.

An amplifier's input offset voltage is the *apparent* voltage at an input even if zero volts is applied. This DC error voltage gets multiplied by the op amp circuit's gain to produce an output error voltage. For example, consider a typical op amp with an input offset voltage of 5 mV, used in a circuit with a gain of 20. This would produce an output offset error of 100 mV. Depending upon the application, this error may not be acceptable (especially if you are amplifying a sensor signal whose output is comparable to the input offset voltage). In that case, a precision op amp, with a very low input offset voltage ($\ll 1$ mV) should be used, or the offset voltage must be zeroed out using additional components connected to the IC's null adjust pin.

Input bias current is a DC current that flows from an op amp's input into the components connected to that input. If the device at the op amp's input has a very high impedance (or is a current-output device with a very small output), this error can be significant. Consider a resistive sensor with an impedance of 100,000 ohms connected to an op amp voltage follower (Figure 3-4). If the op amp has an input bias current of 1 μA, it produces a DC error voltage of $V_{err} = I_{ib} \times R_{in} = 1\ \mu\text{A} \times 100{,}000$ ohm = 0.1 V. In a case like this, a high input impedance op amp with a low input bias current (1 nA or less) would be more appropriate.

Many other analog integrated circuits besides op amps are used as common building blocks in signal-conditioning systems. These ICs include voltage comparators, phase-locked loops, and function generators.

3.2.2 The Voltage Comparator

A *voltage comparator*, as shown in Figure 3-10, is very similar to an op amp used in its highest gain, open-loop configuration (no feedback). Here, if the − input (V_{in}) is greater than the + input (V_{ref}) by at least a few millivolts, the output voltage swings to one extreme ($-V$); if the + input is greater than the − input, the output swings to the other extreme ($+V$). By setting the + or − input to a known reference voltage, an unknown voltage (at the other input)

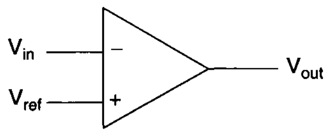


Figure 3-10 Voltage comparator.

can be evaluated. The comparator can be used to determine if analog voltages are within a certain range. It can also be used as a 1-bit ADC. There are even comparators available with response times as fast as a few nanoseconds.

3.2.3 The Phase-Locked Loop

The *phase-locked loop* (PLL) is an interesting device. As shown in Figure 3-11, it consists of a phase detector, VCO, and low-pass filter. This comprises a servo loop, where the VCO is phase-locked to the input signal and oscillates at the same frequency. If there is a phase or frequency difference between the two sources, the phase detector produces an output that is used to correct the VCO. The low-pass filter is used to remove unwanted high-frequency components from the phase detector's output. One application for this device is to demodulate an FM (frequency modulated) signal.

3.2.4 The Tone Decoder

The *tone decoder* is similar to the phase-locked loop (see Figure 3-12) except that the filtered phase-detector output goes to a comparator instead of feeding back to the VCO. The VCO frequency is constant, so the comparator is activated only when the input signal is within the pass band centered on the VCO frequency. This device is commonly used for frequency detection, as in telephone touch-tone equipment.

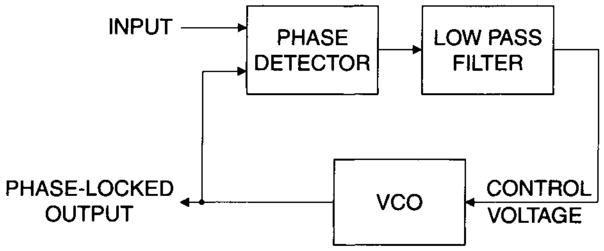


Figure 3-11 Phase-locked loop.

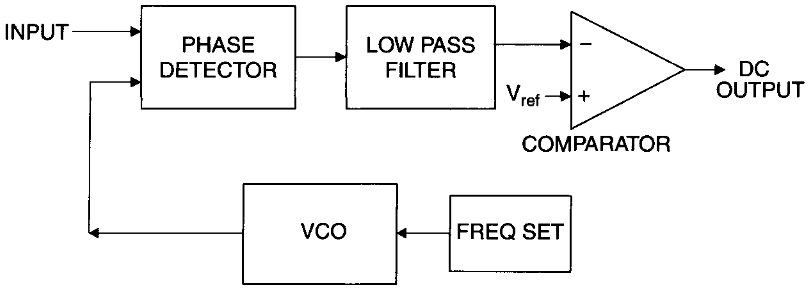


Figure 3-12 Tone decoder.

3.2.5 The Function Generator

Function generator ICs are special-purpose oscillators used to produce sine, square, and triangle waveforms. The signal frequencies are varied either by external resistors and capacitors or by a control voltage, as with a VCO. The output can be frequency modulated by a signal on the VCO input. Some devices also provide for amplitude modulation. These devices can typically produce outputs within the range of 0.01 Hz to 1 MHz. They are often used in test equipment.

Other common analog ICs include a wide range of amplifiers, signal generators, timers, and filters, some of which we will cover later in this chapter.

3.3 Analog Conditioning Circuits

Analog signal-conditioning circuitry can range from a simple RC filter, using two passive components, to a complex system using hundreds of ICs and discrete devices.

3.3.1 Filters

Filtering is undoubtedly the most commonly used analog signal-conditioning function. Usually only a portion of a signal's frequency spectrum contains valid data and the rest is noise. A common example is 60-Hz AC power-line noise, present in most lab and industrial environments. A high-gain amplifier will easily amplify this low-frequency noise, unless it is rejected using a band-reject filter or high-pass filter. The standard types of filter responses are low-pass, high-pass, band-pass, and band-reject (or notch filter). The low-pass

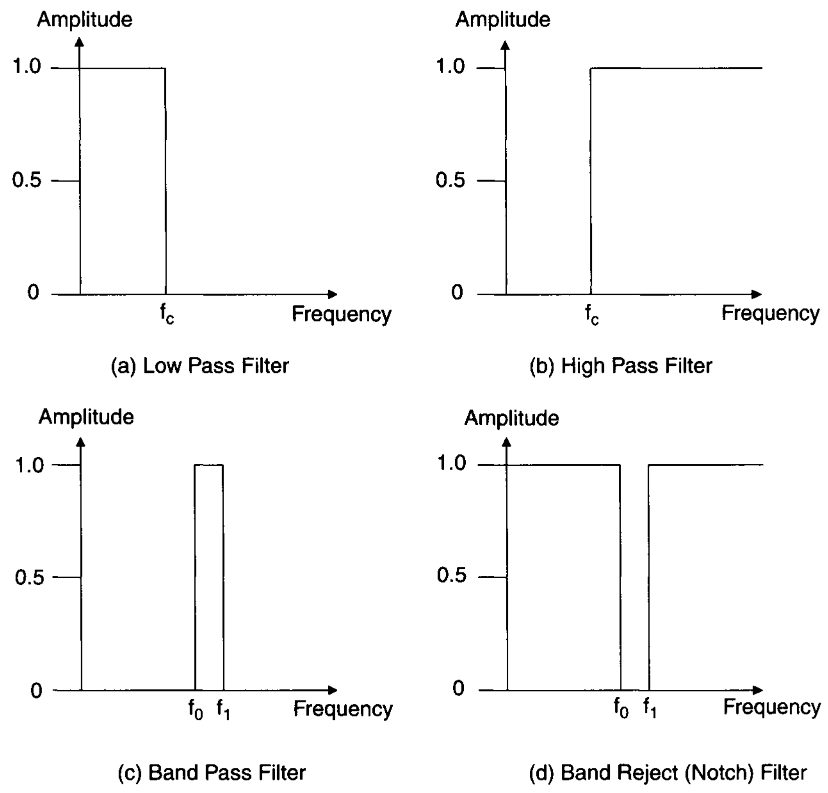


Figure 3-13 Ideal filter responses.

filter attenuates signals *above* its cutoff frequency, whereas the high-pass filter attenuates signals *below* its cutoff frequency. The band-pass filter attenuates frequencies outside of its pass-band range (both above and below), and the band-reject filter attenuates those frequencies within its pass-band range. See Figure 3-13 for amplitude-versus-frequency curves of ideal filters.

The study of filters is an entire discipline unto itself. We will only touch on some simple examples here. The reader is referred to the bibliography for more details on the design and use of filters. Two general classes of filters are *active* and *passive*, depending on the components used. A passive filter, using only resistors, capacitors, and inductors, has a maximum gain (or transfer function value) of 1; an active filter, which uses passive components along with active components (often op amps), can have a much higher gain, as well as a sharper frequency response curve.

Passive Filters The simplest filters use a single resistor and capacitor, so they are called *RC* filters. They rely on the frequency-dependent reactance of capacitors for filtering effects. RC circuits are usually used as simple low-pass and high-pass filters. The reactance of an ideal capacitor is $-j/\omega C$ (where $\omega = 2\pi f$, C is capacitance, and $j = \sqrt{-1}$).

The RC low-pass filter is shown in Figure 3-1a. V_{in} is the input AC voltage and V_{out} is the output AC voltage. The transfer function that describes the response of the circuit is $H(f) = V_{out}/V_{in}$. Since the two components are in series, the current through them is the same: $I_R = I_C$. Z is the AC impedance. Since $V = I \times Z$,

$$H(f) = (I \times Z_C)/(I \times (Z_R + Z_C))$$

$$= Z_C/(Z_R + Z_C)$$

Since $Z_R = R$ and $Z_C = -j/\omega C$,

$$H(f) = 1/(1 + j\omega RC)$$

Note that as frequency (or $\omega = 2\pi f$) approaches zero, the magnitude of the transfer function $|H(f)|$ approaches 1, or no attenuation. Also, the phase angle of $H(f)$ (the phase shift between output and input) approaches 0°. As f increases, $|H(f)|$ decreases and the phase angle becomes more negative. The *cutoff frequency* f_c is where the magnitude of the real and imaginary impedance components are equal (when $\omega RC = 1$), and $|H(f)| = 1/\sqrt{2} = 0.707$. This is the −3 dB point [20 × log(0.707) = −3 dB]. The phase angle at f_c is −45°. Well above f_c (i.e., $f > 10 \times f_c$) $|H(f)|$ falls off at −20 dB per decade of frequency (for every frequency increase of 10× the voltage output drops 10×). This is the same as dropping 6 dB per octave (whenever the frequency doubles). At these higher frequencies, the phase shift approaches −90°. Now the low-pass filter acts as an *integrator*. It is important to remember that this integration is only accurate at high frequencies (well above cutoff).

The RC high-pass filter, shown in Figure 3-1b, is similar to the low-pass filter just discussed. Here, the output voltage is across the resistor, instead of the capacitor. The transfer function for this circuit is $H(f) = 1/[1 - j/(\omega RC)]$. Now, as the frequency gets higher, $|H(f)|$ approaches 1. As the frequency approaches zero, $|H(f)|$ becomes very small.

Again, the 3-dB cutoff frequency, f_c, is where $\omega RC = 1$. The phase angle at f_c is now +45°. At higher frequencies, the phase angle decreases toward 0. At lower frequencies ($f < f_c/10$), the phase angle approaches +90° and $|H(f)|$ increases at the rate of 20 dB per decade. In this low-frequency, high-attenuation region, the RC high-pass filter performs as a *differentiator*. Similar to the RC integrator, this differentiation is only accurate at relatively low frequencies.

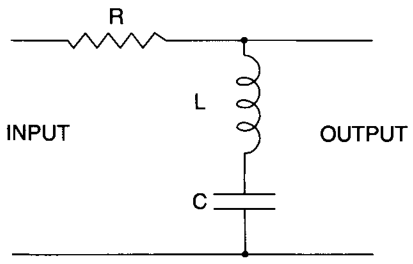


Figure 3-14 Series RLC filter.

Another important point about passive RC integrators and differentiators is that their operational frequency range is in a high-attenuation region. So, their output signals will be very low amplitude, possibly limiting their usefulness because of excessive noise.

RL circuits can also be used as low-pass and high-pass filters, yet they are much less common. A series RLC circuit, as shown in Figure 3-14, is used as a band-reject or notch filter. Here, the minimum value of $|H(f)|$ occurs at $f_0 = 1/(2\pi\sqrt{LC})$, where the phase angle is $\pm 90°$. This is the filter's *resonant frequency*. Below f_0, $|H(f)|$ increases while the phase angle increases toward $0°$ (as f approaches zero). Above f_0, $|H(f)|$ again increases, while the phase angle decreases to $0°$. Well above or below f_0, $|H(f)|$ approaches 1.

A parallel RLC circuit, as shown in Figure 3-15, acts as a band-pass filter, with a maximum $|H(f)| = 1$ at resonance (f_0). At f_0, the phase angle is $0°$. This arrangement is sometimes referred to as a *tank* circuit because, at the resonant frequency, it effectively stores most of the electrical energy available (except for losses through the resistor). Below f_0, $|H(f)|$ decreases while the phase angle increases toward $+90°$. Above f_0, $|H(f)|$ again decreases, while the phase angle approaches $-90°$. Well above or below f_0, $|H(f)|$ falls off at -20 dB per decade. However, close to f_0, this fall off may be much steeper, depending on the value of Q, a measure of the filter's resistive losses:

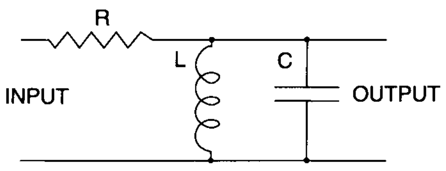


Figure 3-15 Parallel RLC filter.

$Q = 2\pi f_0 \times L/R$. The smaller the value of R is, the larger Q becomes and the steeper the $|H(f)|$ curve becomes, around f_0.

Using passive components, if a broader pass-band response or a steeper attenuation curve for out-of-band frequencies is desired, usually several simple filter stages are concatenated. This can produce the desired frequency response, at the expense of higher attenuation within the pass-band, referred to as the insertion loss. One way around this problem is to use an active filter.

Active Filters Active filters are typically op amp circuits using resistors and capacitors to produce the required frequency response, usually with a gain equal to or greater than 1 (no inductors are needed). They have been limited to relatively low frequencies (i.e., <1 MHz) because of the limited frequency response of standard op amps. In the audio and ultrasonic regions these filters are indispensable. The availability of high-frequency amplifier ICs (with usable gains well above 100 MHz) has greatly extended the usefulness of active filters. Figure 3-16 shows simple active low-pass and high-pass filters, using a 2-pole Salen-Key topology.

A newer type of active filter is the *switched capacitor filter*. This device is very attractive because external components are not needed (as they are with op amp active filters, where value selection is critical). In addition, this filter can be tuned by varying the frequency of the applied clock signal (usually a digital waveform). This is a good choice when a computer-controlled filter is required. There are a wide range of switched capacitor filter devices available from analog IC manufacturers.

A switched capacitor filter is a sampled-data device, where an internal capacitor is switched between the input signal and an integrating amplifier (where the integrator simulates a resistor), as shown in Figure 3-17. Initially, capacitor C_1 charges to the input voltage at that moment, when switch S_1 is at position (a). Then S_1 switches to position (b) and C_1 dumps its charge into C_2, the integrating capacitor, via the op amp. This process repeats over many switching cycles, where C_2 averages the input signal voltage. The filter's time constant depends upon the switching frequency, which essentially determines the cutoff frequency. Since this is a sampled device, it will have aliasing problems (see Chapter 4 for a discussion of aliasing) as the signal frequency approaches the switch rate. Typically, the switching (clock) frequency is 50 to 100 times the cutoff frequency. To prevent problems with high-frequency signals or switching clock feed-through, a simple passive low-pass filter is often used in conjunction with a switched capacitor filter.

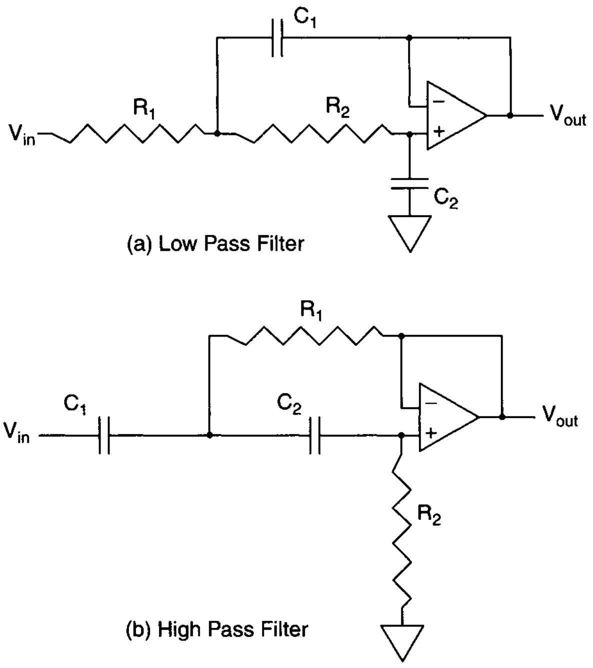


Figure 3-16 Active filters based on op amps.

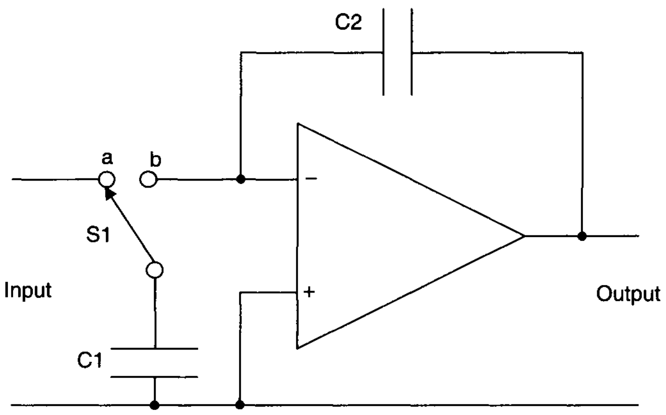


Figure 3-17 Switched capacitor filter.

Standard Filter Functions There are several commonly used filter functions, each with its own special properties. These functions are often used as low-pass, high-pass, or band-pass filters. The *Butterworth* or *maximally flat* filter is characterized by a nearly flat pass-band with no ripples. The roll-off is smooth and monotonic (again without ripples) with a roll-off rate for high-pass or low-pass filters of 20 dB/decade, for each pole. Multiple poles can be concatenated for steeper roll-off. This filter is often used as a good compromise between attenuation and phase response.

The *Chebyshev* or *equal-ripple* filter does have pass-band ripple, although the amount of ripple is specified by the design. It has a faster roll-off near the cutoff frequency than a Butterworth filter but it has a poorer transient response (in the time domain).

A *Bessel* or *Thompson* filter has a linear phase response in the pass-band, which does not distort a nonsinusoidal waveform (such as a square wave) the way a Butterworth or Chebyshev filter would. However, this filter has a much slower roll-off and often requires using higher-order designs (with multiple stages).

The *elliptic* or *Cauer* filter has a much steeper roll-off than the other filter types, at the expense of both ripple in the pass-band and stop-band along with a very nonlinear phase response.

3.3.2 Wheatstone Bridge

Many other types of analog circuits are used for conditioning transducer signals. For resistive sensors, such as strain gages and thermistors, the classic *Wheatstone bridge* is still used. A DC Wheatstone bridge is shown in Figure 3-18. If the resistance values are set so that there is no voltage across the meter (and no current through it) the bridge is said to be *balanced.* At balance, it can be shown that $R_1/R_3 = R_2/\mathrm{R}_4$. Typically a resistive sensor is placed in a bridge circuit to produce a voltage signal output. Usually, one of the resistors in the bridge is the variable sensor element, and initially the bridge is not balanced. Let us assume for the moment that R_1 is the variable resistive transducer and that for simplicity $R_3 = R_4$. When $R_1 = R_2$ the bridge is balanced and the output is zero. As R_1 increases or decreases slightly, the output voltage will swing positive or negative. A calibrated variable resistor in the bridge circuit (for example, R_2) is adjusted until the bridge is again balanced. Then we know that R_1 equals the new value of R_2.

Bridges are also used with AC excitation and reactive elements. This is how a capacitive sensor can produce an accurate voltage signal. In the case of an AC bridge, usually one leg is left as purely resistive, making it easier to balance the unknown reactive element in the other leg.

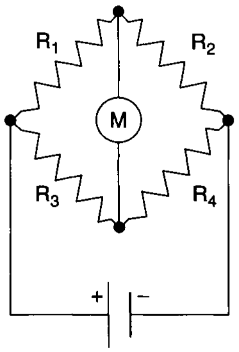


Figure 3-18 Wheatstone bridge.

3.3.3 The Sample-and-Hold Amplifier

Another special analog circuit, extremely useful in data acquisition applications, is the *sample-and-hold amplifier* as shown in Figure 3-19. This is used to get a stable sample of a changing analog signal, prior to using an ADC. The field-effect transistor (FET) acts as a switch, charging the capacitor to the analog signal's present voltage level when the sample line is asserted. When the transistor is switched off, the capacitor "remembers" the voltage, which is buffered by the op amp. The very high input impedance of the op amp, along with a low-leakage capacitor, prevents the voltage from dropping off too quickly.

A sample-and-hold amplifier is used as the front end of an ADC because if the analog waveform is rapidly changing during the ADC cycle, the value

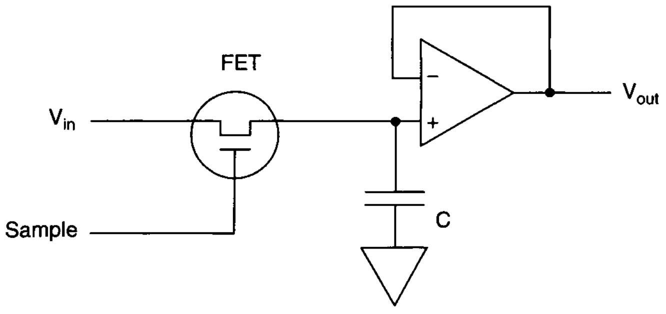


Figure 3-19 Sample-and-hold amplifier.

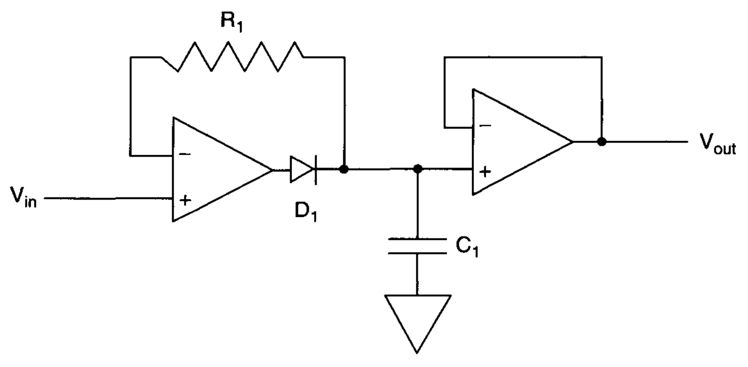


Figure 3-20 Peak detector.

produced can have a large error. This way, there is an accurate "snapshot" of the waveform during the brief sample interval. The sample interval is typically much shorter than the time between successive analog conversions. Sample-and-hold amplifiers are available as monolithic devices, some with sampling intervals as short as a few nanoseconds. In addition, many high-speed ADCs incorporate a sample-and-hold amplifier in the IC.

3.3.4 Peak Detector

Another useful circuit is the *peak detector*, as shown in Figure 3-20, which again is op amp based. It is similar to the sample-and-hold circuit, with a diode used as a switch, for charging the capacitor, C_1. The second (output) op amp is simply a buffer, allowing the circuit to drive a low-impedance load without draining the capacitor. Whenever the input voltage is greater than the output voltage, the diode is forward biased and the capacitor is charged up to that voltage. Usually a switch (such as a FET) may be placed across the capacitor to implement a discharge or reset function. Also, a second diode may be used to compensate for the switching diode's voltage drop (~0.6 V for a silicon signal diode).

3.3.5 Log and Antilog Amplifiers

There are many important nonlinear amplifier circuits, including the *log amplifier* and the *antilog amplifier*. A log amplifier is commonly used to compress a signal's large-amplitude dynamic range into something more manageable by other circuits (such as ADCs). The simple logarithmic amplifier uses a junction diode as a nonlinear element. In a forward-biased diode,

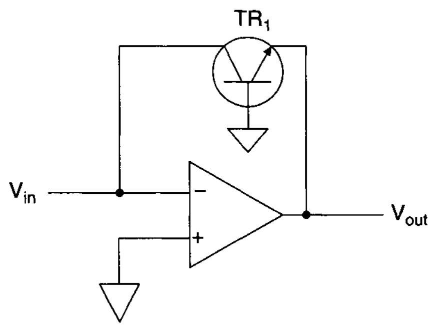


Figure 3-21 Simple logarithmic amplifier.

the voltage drop across the diode varies proportionally to the log of the current through it. When a diode is connected in the feedback loop of an inverting amplifier, the output voltage is a logarithmic function of the input voltage. If a diode is used in a noninverting amplifier, the result is an antilog amplifier.

There are some problems using diodes in log amplifiers. They are very temperature sensitive, since the forward voltage drop across a diode is a function of temperature. In fact, this property is often exploited in diode temperature sensors. Also, the signal range over which the diode has a logarithmic response is somewhat limited. Often a bipolar transistor is used in place of a diode, since its emitter-base voltage varies with the log of its collector current over a very wide range. A log amp using a transistor is shown in Figure 3-21. There are monolithic log amplifier ICs available, which have good temperature compensation and fairly wide operating ranges, often usable over 60 dB or more of input voltage variation.

3.3.6 Other Common Amplifiers

There are several other types of analog amplifier circuits besides the op amp, commonly used for data acquisition purposes. Theses include instrumentation amplifiers, programmable gain amplifiers, and isolation amplifiers.

Instrumentation Amplifiers An instrumentation amplifier (IA) is used to provide a large amount of gain for very low-level signals, often in the presence of high noise levels. The major properties of IAs are high gain, large common-mode rejection ratio (CMRR), and very high input impedance. They are often used to directly amplify signals from passive sensors, such as strain gages (see Chapter 2). An IA is a device which only amplifies the *difference* between

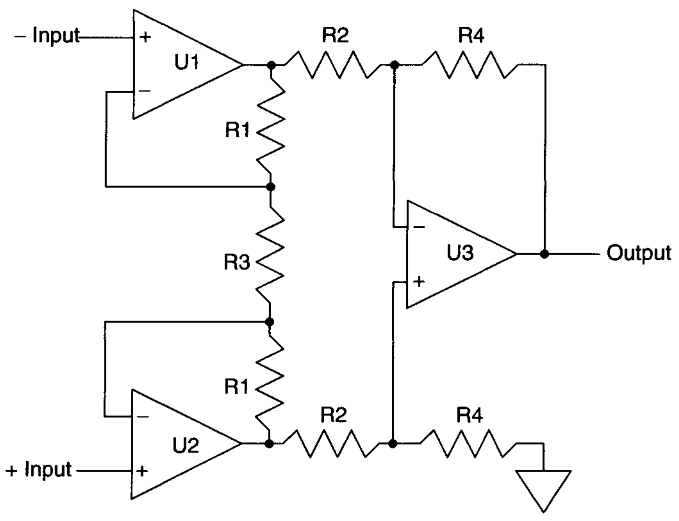


Figure 3-22 Instrumentation amplifier.

the two input lines while ignoring any common-mode noise they both carry. It is usually used for low-frequency signals ($\ll$1 MHz).

A typical instrumentation amplifier configuration consists of three op amps, as shown in Figure 3-22. The resistors used should be high-precision (0.1% tolerance or better) to achieve the highest CMRR possible. The overall gain of this IA circuit is $R_4/R_2[1 + (2R_1/R_3)]$.

Monolithic IA ICs are readily available and are often preferable to building one out of individual op amps, since the internal components will be well matched. These IAs can have a CMRR over 100 dB and a voltage gain up to 10,000×.

Programmable-Gain Amplifiers Programmable-gain amplifiers (PGAs) are a special class of instrumentation amplifiers that have selectable gain, either through external component selection or, more commonly, through digital control lines. They are used in data acquisition systems to enable software control of analog gain, tailoring the amount of amplification to the current task. Typical PGAs have either decade (1×, 10×, 100×, 1000×) or binary (1×, 2×, 4×, 8×) gain settings, using just a few digital control lines.

These control signals are usually used to select different, internal feedback resistor values, to change the gain. Some PGAs have multiple amplifiers configured for different gain values and the digital controls select which amplifier output is used.

Isolation Amplifiers Isolation amplifiers are used to boost low-level analog signals when electrical isolation between input and output is needed. This may be when there are high common-mode voltages present, such as a sensor biased by a high DC voltage. Another use is when medical monitoring equipment is connected to a patient and current flowing from the instrumentation to the patient connections (such as ECG electrodes) can be dangerous.

Most commercial isolation amplifiers use transformers, capacitors, or optical couplers to separate input from output. The important characteristics are isolation voltage (commonly up to 5000 V), leakage current (typically less than 1 μA), gain error, and bandwidth.

3.3.7 Other Common Analog ICs

There are many other analog ICs commonly used in data acquisition equipment, besides those we have previously covered in this chapter. Some of these are analog switches, multiplexers, and voltage references.

Analog Switches and Multiplexers An analog switch is a digitally controlled device that is used to pass or interrupt an analog signal, analogous to a mechanical switch or relay. These devices usually use FETs as the main switching elements. Unlike mechanical switches, analog switches have an extremely limited signal voltage range (usually less than the switch's power supply voltage) and a relatively high "on" resistance (typically ranging from a few tenths of an ohm to over 100 ohms). However, these devices are much smaller and faster than mechanical relays, many exhibiting switching speeds under 1 μsec.

Multiple analog switches can be arranged in a single IC to produce a multiplexer (mux) with multiple inputs and a single output. The device's digital control lines determine which input is steered to the output. A data acquisition card containing a single ADC may have an eight-channel multiplexer at its input, allowing eight analog signals to be simultaneously connected to it. However, only one channel at a time can be digitized by the ADC.

Voltage References The absolute accuracy of an ADC is determined, among other factors, by its analog reference. A voltage reference is an IC that either contains or behaves as if it is a precision Zener diode, with a well-characterized breakdown voltage. This voltage is also fairly insensitive to temperature changes and aging. Some voltage references contain internal buffer amplifiers that allow them to drive low-impedance loads. A high-quality voltage reference also allows a data acquisition board to perform an accuracy self-test and even autocalibration on its analog input channels.

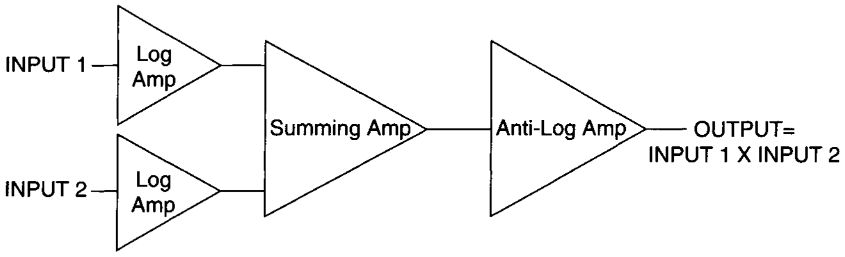


Figure 3-23 Analog multiplier.

3.3.8 Modulation

An important nonlinear function is modulation. Frequency modulation was discussed with the VCO. Amplitude modulation is easily achieved using an *analog multiplier.* A simple means of producing an analog multiplier is shown in Figure 3-23. The two inputs each pass through a log amplifier and then are added together; finally they pass through an antilog amplifier. The output voltage is equal to the product of the input voltages times a scaling factor. Analog multipliers are also commonly available as single-chip devices. Many of these monolithic multipliers can perform division and square functions.

3.3.9 High-Frequency Analog Circuits

As ADCs operate at higher speeds (up to 1 gigasample/second), analog circuitry bandwidth must also increase or these fast sampling speeds are wasted. To maintain high bandwidths, special attention must be paid to factors such as transmission line impedance, stray capacitance, shielding, and connector quality.

When working with analog signal frequencies well above 1 MHz, coaxial cables and connectors should be used. This will help minimize signal attenuation and distortion due to impedance mismatches, as well as reduce external noise pickup. The most common coaxial connectors used are BNC and SMA types.

When designing a high-frequency amplifier circuit, component placement on the board is critical. A high-speed op amp with a bandwidth of several hundred MHz, up to 1 GHz, can easily become an oscillator because of circuit instabilities caused by stray capacitance of the board itself. RF design techniques must be used.

Conventional, voltage-feedback op amps operate at high frequencies in much the same way as their low-frequency counterparts. Their bandwidth varies inversely with circuit gain, since gain–bandwidth product is constant. Newer, current-feedback amplifiers are commonly used at high frequencies (usually 100 MHz or above). Their gain–bandwidth product is not constant. The circuit's bandwidth is mostly determined by the value of the feedback resistor used, not simply the gain settings. When a current-feedback amplifier is used as a voltage follower (gain of 1×), a resistor should be connected from the output to the – input, as per the manufacturer's recommendation.

When working with high-speed and high-frequency circuits, grounding and shielding also become critical. Analog and digital devices should have appropriate (often separate) ground paths on the circuit board, usually with a single common connection point. This helps to minimize digital noise appearing in analog circuits. Shielding of circuit cards may be necessary, both to minimize susceptibility to received high-frequency noise and to limit the amount of RF noise the card itself generates. If external power supplies are used, the cables should be properly filtered, using ferrite beads and bypass capacitors.

These are just some basic guidelines for working with high-speed devices. In general, high-frequency analog circuits are much less forgiving than their low-frequency equivalents.

There are other standard analog signal conditioning devices and circuits besides the ones shown in this chapter. The information here should give you a feel for what is commonly available and help you locate more detailed information, as you require it.

CHAPTER 4

Analog/Digital Conversions

As previously noted, we live in an analog world. Nearly all "real-world" measured quantities are analog, at least at the macroscopic level we typically deal with. Analog waveforms are usually defined as smooth, continuous functions that have derivatives existing nearly everywhere. Most transducers have analog outputs, usually voltage or current, which represent the physical quantities being measured, such as temperature or pressure (notable exceptions include optical encoders and smart sensors with digital outputs). Whenever an analog quantity is discussed here, it refers to a voltage or current suitable for use with common electronic equipment. This is typically in the frequency range of 0 to 1 MHz, with a voltage range of around 1 microvolt (μV) to 100 V or a current range of about 1 microampere (μA) to 10 amps.

4.1 Digital Quantities

Digital quantities have discrete levels that vary by steps instead of continuously (as shown in Figure 1-1 of Chapter 1). Most digital electronic equipment uses binary values, which have two possible states, called true (on or 1) and false (off or 0). Most often the 0/1 notation is used to describe the binary level of a single line or wire, represented as a binary digit or *bit*. For the standard family of TTL (*transistor transistor logic*) digital ICs, which operate from a +5 V power supply, a high level (>2.4 V) is a logical 1 and a low level (<0.8 V) is a logical 0. These logic levels also apply to new low-voltage logic families (such as LVTTL) that operate from +3.3 V power supplies.

Logic families that operate from even lower supply voltages (+2.5 V or +1.8 V) use different threshold values for 1 and 0.

Binary values are a base-2 numbering system, as opposed to our everyday base-10 decimal system. It takes many bits grouped together to represent a useful quantity. In general, a collection of n bits can represent 2^n discrete levels. For example, a group of 8 bits is referred to as a byte, where $2^8 =$ 256 levels, for a representation of values in the range of 0 to 255 (or −128 to +127). A group of 16 bits is referred to as a short word, having $2^{16} = 65{,}536$ steps. A long word of 32 bits has $2^{32} = 4{,}294{,}967{,}296$ steps. In digital electronic equipment, these groups of bits are usually parallel lines or wires, where each bit is present at the same time. One wire typically carries the value for one bit. This means that increasing the number of levels a digital circuit can represent increases the number of wires (or interconnections) in that circuit. This increase also allows the digital representation to more closely approximate the analog signal, within a given dynamic range.

The concept of *dynamic range* is very important for data acquisition systems; it will be addressed at greater length in Chapter 10. By definition, the dynamic range of a data acquisition system is the ratio of the maximum value that can be measured to the smallest value that can be resolved. This number is often represented in decibels (dB) as

$$\text{Dynamic range (dB)} = 20 \times \log_{10} (\text{max/min})$$

If both positive and negative values are measured,

$$\text{Maximum value} = \text{maximum positive value} - \text{minimum negative value}$$

For example, a data acquisition system with a 1-mV resolution and a value range of 0 to +10 V (or −5 to +5 V) has a dynamic range of 10,000:1, or 80 dB. This dynamic range requires a minimum of 14 bits to represent it, since $2^{14} = 16{,}384$, which is greater than 10,000, whereas 2^{13} (8192) is less than 10,000.

4.1.1 Binary Codes

For n binary lines to represent 2^n levels, each line must have a different value or weight. For a *natural binary* code, having any value from 0 to $2^n - 1$, integers are represented by a series of weighting bits having the value 2^m (where m varies from 0 to $n - 1$). The bit number m is zero for the least significant bit (LSB) on the far right and increases to $n - 1$ for the most significant bit (MSB) on the far left. The values of integer bit weights for the first 16 bits are given in Table 4-1. The value of a collection of parallel bits is the sum of the weighted values of all nonzero bits (or the value of a bit, either 0 or 1, times its weight).

TABLE 4-1
Positive Integer Bit Weights for Natural Binary Code

BIT # (m)	BIT WEIGHT (2^m)
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

For example, we will evaluate the 8-bit binary integer 01011101. Starting with the LSB, working from right to left:

$$\begin{aligned}\text{Sum} &= 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 0 \times 2^7 \\ &= 1 + 0 + 4 + 8 + 16 + 0 + 64 + 0 \\ &= 93\end{aligned}$$

Sometimes it is necessary to represent both positive and negative integer values, as when dealing with a bipolar voltage. The most common binary code for this is called *twos complement*, which can represent values from -2^{n-1} to $+2^{n-1} - 1$. In this notation, positive values are encoded the same way as the positive-only, natural binary code above (this includes zero). To encode a negative value, write down the code for the corresponding positive value (including all leading zeros), invert the number by changing all ones to zeros and all zeros to ones (which is called the *ones complement*), and then add 1 to the result. Table 4-2 contains twos complement codes for 5-bit numbers

TABLE 4-2
Twos-Complement Coding for Five-Bit Bipolar Values

VALUE	TWOS COMPLEMENT CODE
+15	01111
+14	01110
+13	01101
+12	01100
+11	01011
+10	01010
+9	01001
+8	01000
+7	00111
+6	00110
+5	00101
+4	00100
+3	00011
+2	00010
+1	00001
0	00000
−1	11111
−2	11110
−3	11101
−4	11100
−5	11011
−6	11010
−7	11001
−8	11000
−9	10111
−10	10110
−11	10101
−12	10100
−13	10011
−14	10010
−15	10001
−16	10000

representing values +15 to −16. For example, to get the twos complement representation of the value −12 using 5 bits:

1. +12 = 01100
2. Ones complement = 10011
3. Twos complement = 10011 + 1
4. −12 = 10100

One additional coding system we will mention here is *fractional binary*. This is useful when digital readings must be normalized to an arbitrary full-scale value, as when a converter's reference voltage is variable. The n bits of the code represent values between 0 and $1 - 2^{-n}$. The weight of each bit is a fractional value, equal to its natural binary integer value (of 2^m) divided by 2^n. This means the MSB has a weight of 1/2 (since $2^{n-1}/2^n = 2^{-1}$), the next bit to the right has a weight of 1/4, and so on, down to the LSB with a weight of $1/2^n$ (or 2^{-n}). When all bit values are 1, the total value = $1 - 2^{-n}$. Again, 2^n levels are represented by this code. Table 4-3 lists fractional binary codes for 5-bit values. Note that sometimes fractional binary values are written with a *binary point* and sometimes not. So, the fractional binary for 1/32 can be written as either 0.00001 or 00001, even though they both mean the same thing.

4.2 Data Conversion and DACs

Data conversion is at the heart of data acquisition systems. Real-world analog signals must be converted to binary representations via an *analog-to-digital converter* (or ADC). Similarly, if output to the analog world is required, as in control systems, digital values are transformed using a *digital-to-analog converter* (or DAC). We will look at DACs first, because they are usually simpler devices than ADCs. In addition, many ADCs contain DACs as part of their circuitry.

DACs use either current or voltage switching techniques to produce an output analog value equal to the sum of several discrete analog values. Because it is easier to sum currents (rather than voltages) using analog circuitry, most commonly available DACs are current-mode devices. They produce the sum of internal current sources and use either an internal or external op amp as an output current-to-voltage converter.

4.2.1 Fully Decoded DAC

One type of DAC is shown in Figure 4-1. This is a fully decoded current-mode 3-bit DAC. A fully decoded DAC, for n input bits, contains $2^n - 1$ switches and identical current sources. Basically, the input bits are decoded

TABLE 4-3
Five-Bit Fractional Binary Codes

CODE	FRACTION OF FULL SCALE
0.00000	0
0.00001	1/32 (LSB)
0.00010	2/32 = 1/16
0.00011	3/32
0.00100	4/32 = 1/8
0.00101	5/32
0.00110	6/32 = 3/16
0.00111	7/32
0.01000	8/32 = 1/4
0.01001	9/32
0.01010	10/32 = 5/16
0.01011	11/32
0.01100	12/32 = 1/8
0.01101	13/32
0.01110	14/32 = 7/16
0.01111	15/32
0.10000	16/32 = 1/2 (MSB)
0.10001	17/32
0.10010	18/32 = 9/16
0.10011	19/32
0.10100	20/32 = 5/8
0.10101	21/32
0.10110	22/32 = 11/16
0.10111	23/32
0.11000	24/32 = 3/4
0.11001	25/32
0.11010	26/32 = 13/16
0.11011	27/32
0.11100	28/32 = 7/8
0.11101	29/32
0.11110	30/32 = 15/16
0.11111	31/32

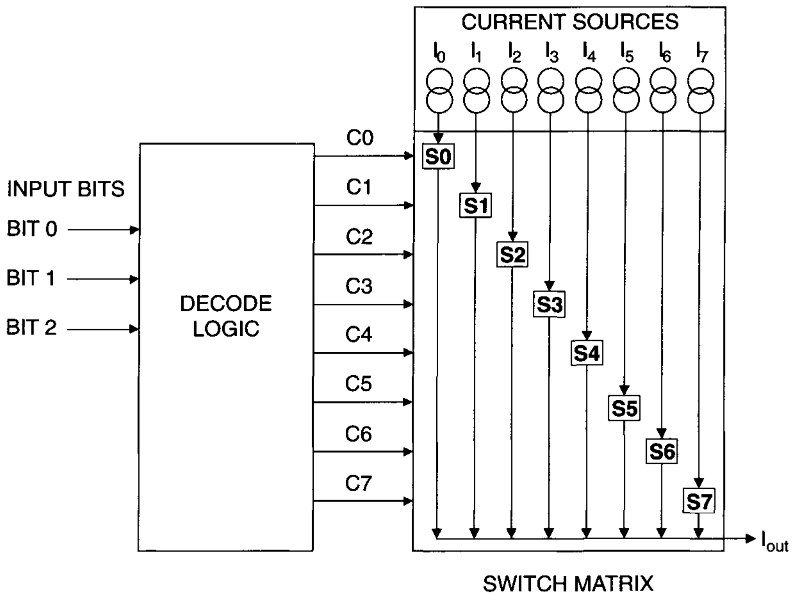


Figure 4-1 Fully decoded 3-bit current mode DAC.

and control switches to the current sources of equal magnitude. A digital value of 001 connects one current source to the output, a value of 010 connects two sources to the output, 011 connects three sources to the output, and so on up to seven sources for 111. These current sources are summed at the output, producing a current proportional to the digital value.

The main advantage to this type of fully decoded DAC is that with proper switching the output current is guaranteed to be *monotonic*. That is, as the digital code continues to increase the analog output will also increase, step by step. This is not always true of all DACs. The disadvantage of this type of DAC is that $2^n - 1$ current sources and switches are required. This becomes prohibitive for reasonably large numbers of bits, such as 4095 current sources for a 12-bit DAC.

4.2.2 Weighted Resistor DAC

A simpler DAC can be produced using a voltage reference with a set of weighted precision resistors and switches, as shown in the 3-bit DAC example in Figure 4-2. The resistor values are in a binary bit-weight ratio (1:2:4:8:16

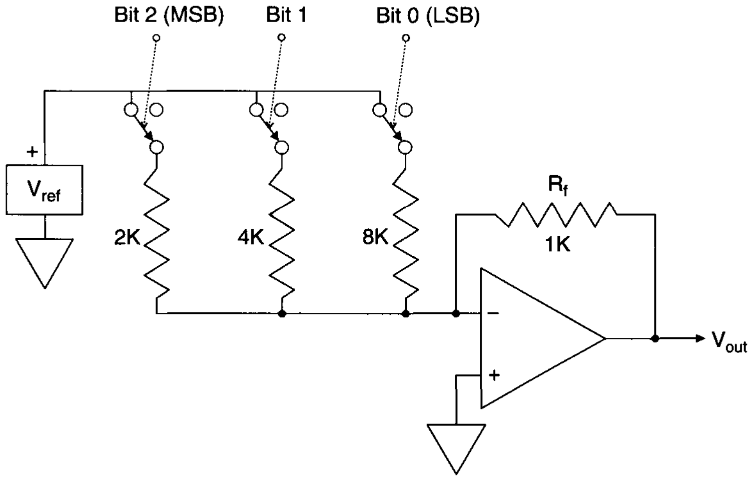


Figure 4-2 Weighted resistor, 3-bit current mode DAC.

and so on). Again, this converter is a current-mode device, with the sum of all resistor currents resulting in an analog current.

In this example, as in nearly all practical current-mode DACs, the output current is passed through an op amp. This acts as a current-to-voltage converter as well as isolating the DAC from output circuit loading. Here, since the op amp is inverting (because the virtual ground of the inverting input is needed) the output is a negative voltage proportional to the input binary word and the voltage reference.

When all input bits are zero, no current flows into the op amp, and the output voltage is zero. If the MSB (bit 2) is 1, the current flowing into the op amp is $V_{ref}/2K$, producing an output voltage of $-V_{ref}/2$, since the feedback resistor (R_f) is 1K ohm and the op amp's gain is $-R_f/R_{in}$. Similarly, if bit 1 is 1, it generates a current of $V_{ref}/4K$, producing an output voltage of $-V_{ref}/4$; and if the LSB (bit 0) is 1, it generates a current of $V_{ref}/8K$, producing an output voltage of $-V_{ref}/8$. If more than a single bit is 1, their currents sum at the op amp's input and produce the appropriate output voltage. If all bits are 1, the output voltage is $-7/8\ V_{ref}$. This is the full-scale output.

This DAC can produce eight discrete analog output levels, spaced 1/8 V_{ref} apart. Note, that if we treat these values as normalized to V_{ref}, we are dealing with fractional binary values. If we set $V_{ref} = 10.00$ V, the full-scale output is −8.75 V, with steps of 1.25 V. If we increased the number of bits in this DAC to n, the resistor values for the most significant bits would stay the same and larger resistors would be added for the least significant bits. The LSB will have a value of $2^n \times$ 1K ohm.

The advantage of the DAC in Figure 4-2 is that only one switch and resistor are needed per bit. The main drawbacks are that as the number of converter bits increases, the number of different precision resistor values needed, as well as the overall range of resistor values, increases. If we increased the resolution of the DAC in Figure 4-2 from 3 bits to 8 bits, the resistance values would increase up to 256K ohms. This makes it very difficult to maintain monotonicity, linearity, and overall accuracy, because of the wide range of resistance values required.

4.2.3 Resistor Quad

Other techniques are used to overcome these drawbacks. One of these is the *binary resistance quad*, used in an 8-bit DAC in Figure 4-3. Here, the resistor network uses the same four values for more than 4 bits resolution. The resistors and switches constitute a voltage-divider network. The most significant 4 bits (bits 4–7) are in the usual scaled binary ratio of 2:4:8:16. The least significant 4 bits (bits 0–3) are these same values, repeated. However, these values are attenuated 16:1, via the additional (16K ohm) resistor. Each section of four resistors is called a quad.

4.2.4 *R*–2*R* Ladder

A very common DAC uses the *R*–2*R* resistance ladder, where only two different resistor values are needed, as shown in Figure 4-4. When only the MSB (bit 7) is 1, the output voltage is $-V_{ref}/2$, since V_{ref} is switched through 2*R* from

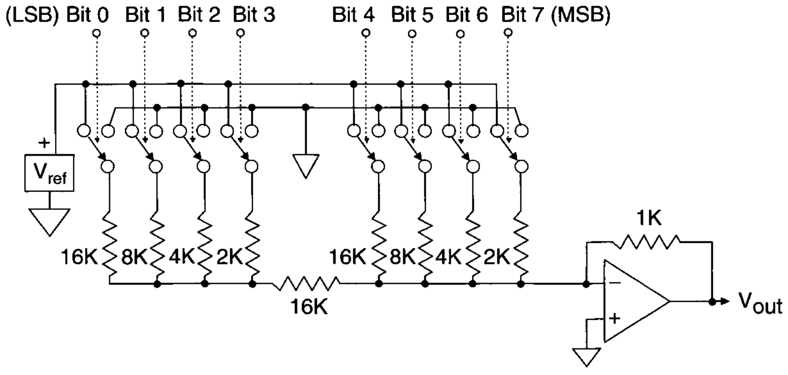


Figure 4-3 8-bit DAC using resistor quads.

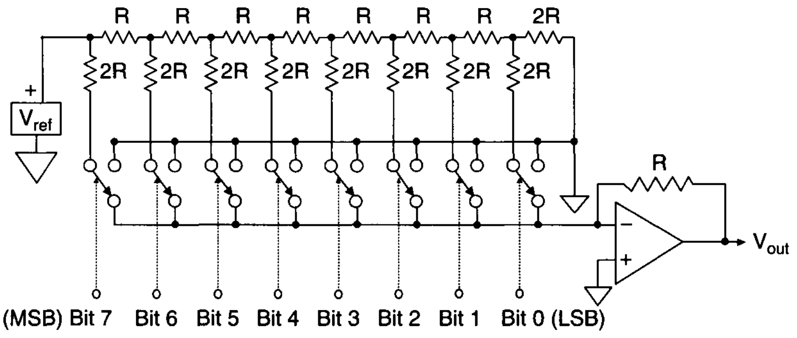


Figure 4-4 8-bit DAC using R–$2R$ resistor ladder.

bit 7 and the op amp's feedback resistor is R. When moving down the ladder (toward less significant bits), each $2R$ resistor sees one-half the voltage of the one above it (when it is the only 1 bit). This is due to the constant resistance of the attenuator network to ground. So, bit 6 contributes $-V_{ref}/4$ to the output voltage, bit 5 contributes $-V_{ref}/8$, and so on down to bit 0 contributing $-V_{ref}/256$.

4.2.5 Multiplying DAC

When a DAC can operate with a variable analog reference voltage, instead of the usual fixed value, it is called a multiplying DAC. The output of this DAC is proportional to both the analog reference input and the digital input. If it can respond to bipolar inputs (both analog and digital) and produce a bipolar output, it is a four-quadrant multiplying DAC. This refers to a Cartesian plot of the transfer function. A multiplying DAC is commonly used as a digitally controlled attenuator or amplifier of an analog signal.

4.2.6 DAC Characteristics

Some important criteria must be considered when choosing a DAC. The first parameter to determine is the number of bits of resolution. This is selected by knowing the desired dynamic range of the output signal. Eight-, 12-, and 14-bit DACs are commonly available as *monolithic devices* or integrated circuits (ICs). Even 16-bit DACs are produced commercially.

Another major parameter is *settling time*, which determines the speed of conversion, as shown graphically in Figure 4-5a. This is the amount of time required for a DAC to move to and stay within its new output value (usually to ±1/2 LSB), when the digital input changes. For common, current output

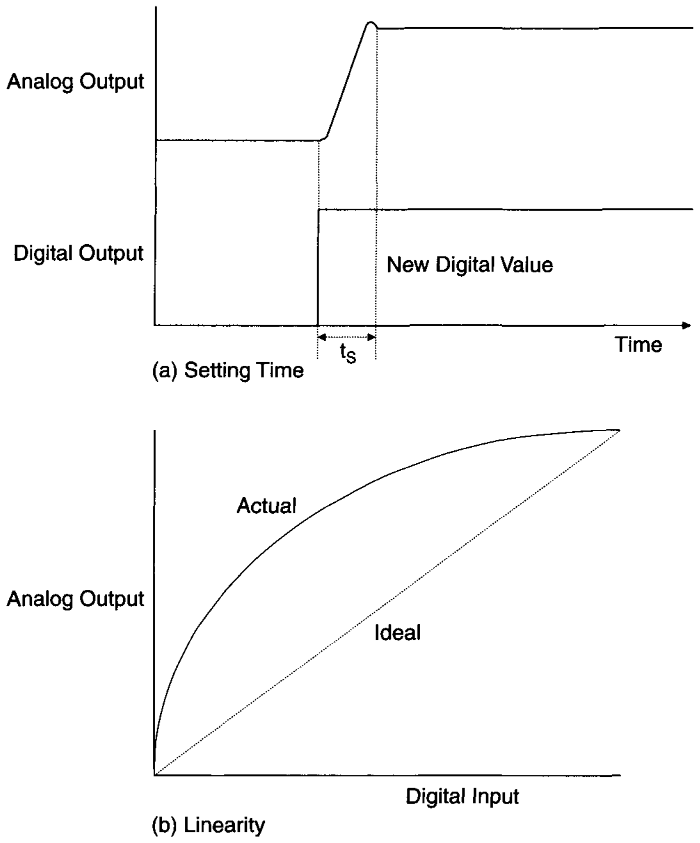


Figure 4-5 Important DAC parameters.

DACs, settling time is reasonably fast, typically a few hundred nanoseconds. If a fast-settling op amp is used as an output current-to-voltage converter, output waveforms at frequencies well over 1 MHz can be produced.

Linearity is another major DAC parameter. It is the maximum deviation of the DAC's transfer curve from an ideal straight line, usually expressed as a fraction of the full-scale reading, as illustrated in Figure 4-5b.

One final DAC parameter to note is *monotonicity*. If the output of a DAC always increases for increasing digital input, the DAC is considered monotonic. Monotonicity is specified over a certain number of input bits, typically the full number of bits of resolution. A nonmonotonic DAC would have a dip in its transfer curve.

4.2.7 High-Speed DACs

There are monolithic DACs available with update rates in the range of 100–300 million samples per second (MSPS). These DACs can go up to 16-bit resolution (although 12- and 14-bit devices are more common at these high speeds). High-speed DACs typically employ a mixed architecture to achieve good performance at these speeds. Most use a segmented current source along with an *R*–2*R* ladder. The important specifications for these high-speed converters are update rate (in MSPS), settling time (in nsec) and slew rate (in V/μsec).

Some of these fast DACs require emitter-coupled logic (ECL) digital control signals. As opposed to TTL digital signal levels (see Section 4.1), ECL signals are negative (relative to ground) and have a smaller difference between logic 0 and 1 levels. ECL logic devices are one of the fastest families of digital ICs commonly available. Some high-speed DACs use external TTL controls and translate them internally into ECL signals. ECL devices are powered by a –5.2 V supply (compared to +5 V or lower for TTL ICs). There are several different ECL families, with typical logic levels of –1.75 V representing 0 and –0.9 V representing 1.

4.3 ADCs

Now we will turn our attention to ADCs. A multitude of techniques are used to produce an analog-to-digital converter. We will look at some of the more common ones here.

4.3.1 Ramp ADC

One of the simpler approaches in implementing an ADC is the *ramp converter* shown in Figure 4-6. It consists of a digital counter, a DAC, an analog comparator, and control logic with timing generation. Basically, when an analog conversion is requested, the digital counter starts counting up from zero. As it counts, the analog output of the DAC increases, or ramps up. When the DAC's output is equal to or exceeds the analog input, the comparator's output switches and the control logic stops the counting. An end of conversion is indicated, with the digital counter output now containing the converted value. This conversion sequence is illustrated in Figure 4-7.

The problem with this technique is its relatively long conversion time, or slow speed, which becomes worse with increasing number of output bits. Everything else being equal, the maximum conversion time for the ramp converter increases as 2^n, where *n* is the number of bits of resolution. The

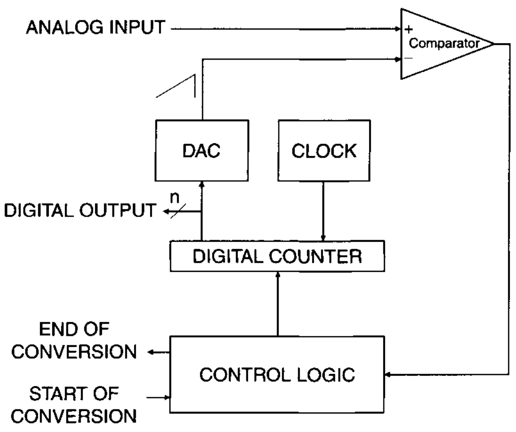


Figure 4-6 Simple ramp analog-to-digital converter (ADC).

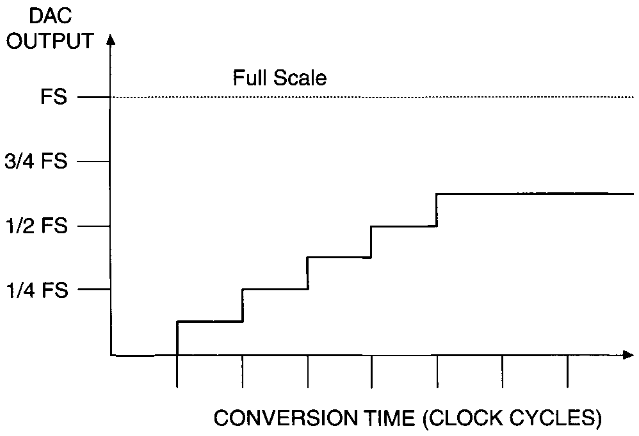


Figure 4-7 Ramp ADC, typical conversion sequence.

conversion time is inversely proportional to the frequency of the clock used in counting.

For example, if the converter's DAC had a 200-nsec settling time and we used a 5-MHz clock for a 12-bit ADC, maximum conversion time would be $\frac{1}{5 \times 10^6} \times 4096 = 819.2$ μsec. This would allow a conversion rate of only 1220 samples per second. Of course, this is a worst-case value. If the analog input is less than the maximum allowable value, conversion time will be shorter.

Even using a fast DAC with a 10-nsec settling time and a 100-MHz clock, the minimum conversion rate is just 24,400 samples per second.

One minor variant on this technique is the *servo ADC*. Its digital counter can count both up and down. When the DAC output is below the analog input, it counts up. When the DAC output is above the analog input, it counts down. It tends to track the analog input continuously, analogous to a servo control loop. It will respond to small input changes rapidly, but it is as slow as the standard ramp converter when a large input change has occurred.

4.3.2 Successive-Approximation ADC

A major improvement on the ramp converter is the *successive-approximation converter*, probably the most popular class of general-purpose ADCs commercially available at present. The overall block diagram of this system is very similar to that of the ramp converter, as shown in Figure 4-8, except that the digital counter is replaced by more sophisticated control logic that includes a shift register. Instead of simply counting up until the analog value is exceeded, the successive-approximation ADC tests one bit at a time (starting with the most significant) until the internal DAC value is as close as possible to the analog input without exceeding it.

First, the most significant bit (MSB), equal to 1/2 full scale (FS) value, is turned on; if the DAC's output is less than the analog input, it is left on (otherwise it is turned off). Then the next bit down (1/4 FS) is turned on and

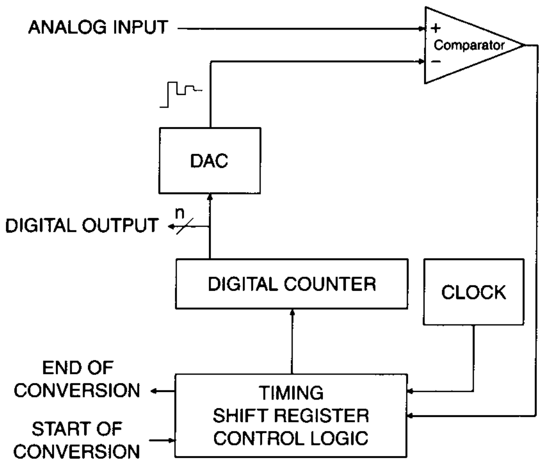


Figure 4-8 Simple successive approximation ADC.

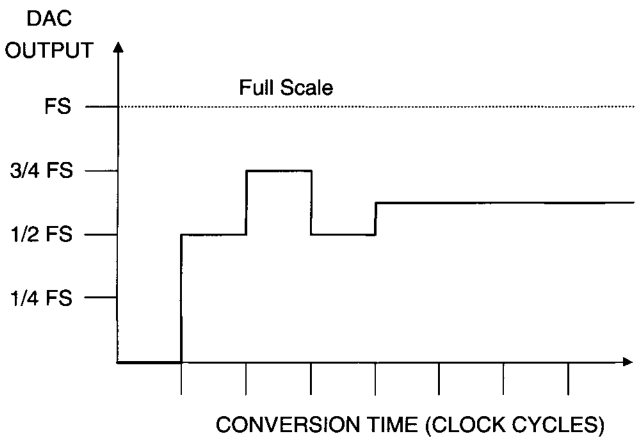


Figure 4-9 Successive approximation ADC, typical conversion sequence.

left on only if the DAC's output is still less than the analog input. This process continues until all *n* bits have been tested. Figure 4-9 shows a typical conversion sequence. The entire conversion requires many fewer than 2^n clock cycles (usually between *n* and 2*n* cycles). Furthermore, the conversion time is relatively constant and insensitive to the input analog value, as opposed to ramp converters.

It is not unusual to find successive approximation ADCs with conversion rates well over one million samples/second and resolution as high as 16 bits. Lower-speed and lower-resolution successive approximation ADCs are common commercial ICs, available at very low prices. For example, there are 8-bit devices with conversion times of 5 μsec or under (i.e., 200 kHz sampling rates) available for only a few dollars.

4.3.3 Dual-Slope ADC

Another common ADC is the *dual-slope converter*, which relies on integration. As shown in Figures 4-10a and 4-10b, the voltage to be measured (V_x) is input to an integrator, charging the capacitor for a fixed time interval t_1, which corresponds to a certain number of clock cycles. At the end of this interval, a known reference voltage (V_r) of opposite polarity is applied to the integrator, discharging the capacitor. The time (and number of clock cycles) required to bring the integrator output back to zero, ($t_2 - t_1$), is measured.

The charge on the capacitor at time t_1 is proportional to the average value of V_x times t_1. This is equal to the charge lost by the capacitor during time $t_2 - t_1$, while being discharged by the reference voltage, proportional to V_r times ($t_2 - t_1$). Hence ($t_2 - t_1$)/t_1 is proportional to V_x/V_r. The output binary

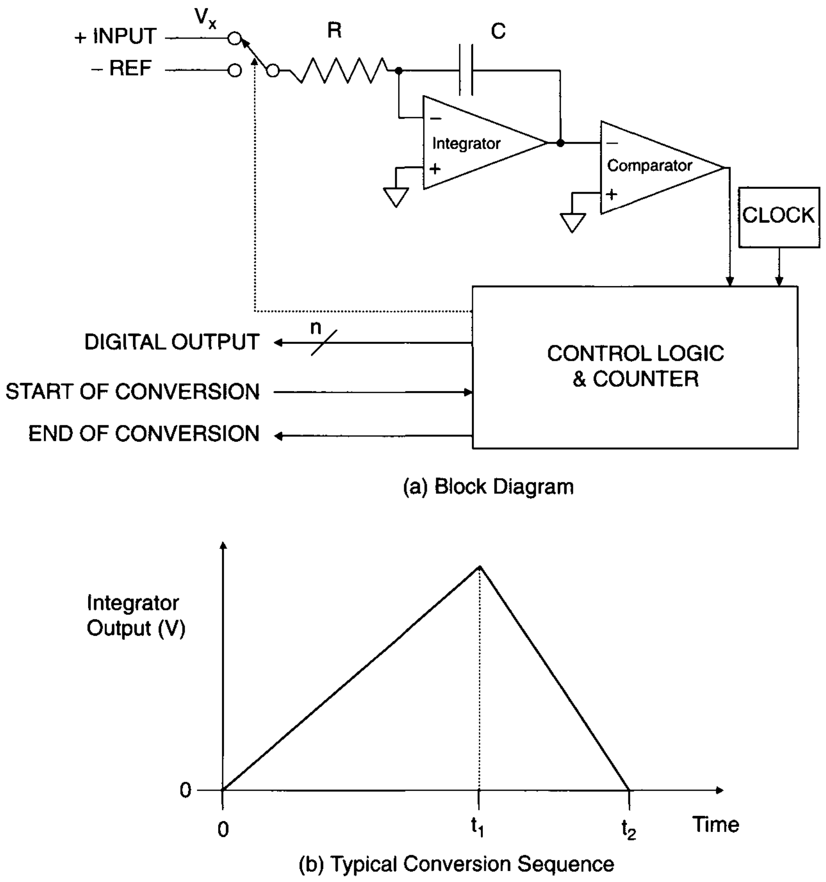


Figure 4-10 Dual-slope ADC.

count for the time interval $(t_2 - t_1)$ is thus proportional to V_x, the input voltage. With appropriate circuitry, bipolar voltages can also be measured.

The dual-slope ADC has many advantages. Noise present on the input voltage is reduced by averaging. The value of the capacitor and conversion clock do not affect conversion accuracy, since they act equivalently on the up-slope and down-slope. Linearity is very good and extremely high-resolution measurements can be obtained. Its main disadvantage is a slow conversion rate, often in the range of 10 samples/second. In applications where this is not a problem, such as in measuring temperature transducers, a dual-slope ADC is a good choice. They are commonly used in digital voltmeters (DVMs) where their resolution is measured in display digits (4-1/2 digits = ±19999 counts or approximately 15 bits).

4.3.4 Voltage-to-Frequency Converter

Another slow ADC is the *voltage-to-frequency converter*, or VFC. It changes an analog signal into a digital pulse train with a frequency proportional to the signal voltage. This pulse train can be converted into a usable digital output of n parallel bits by clocking a counter for a fixed time interval.

The VFC is an integrating device with good noise rejection and monotonicity, similar to the dual-slope converter. It can also be used as an inexpensive, high-resolution ADC, with slow conversion rates. Its drawbacks include nonlinearity, a limited input-voltage dynamic range, and *output offset*. As the input voltage approaches zero, the output frequency is still offset from zero.

4.3.5 Flash ADC

The fastest type of ADC is the *flash converter*. An n-bit flash ADC applies the input voltage to an array of $2^n - 1$ comparators, via a ladder of 2^n resistors. The threshold for the comparators are spaced 1 LSB apart.

Figure 4-11 shows a simple 3-bit flash ADC. When V_{in} is zero, all comparators are off. As the input voltage increases to $V_{\text{ref}}/8$, the lowest comparator (a) goes on. As V_{in} keeps increasing by steps of $V_{\text{ref}}/8$, each successive comparator (b, c, d, . . .) switches on. All comparators are on when the input voltage reaches or exceeds $7/8 \times V_{\text{ref}}$. The digital logic decodes the comparator outputs into a 3-bit word. The digital output can either be normal binary code (000 = minimum value, 111 = maximum value) or a Gray code. In a Gray code, only one output bit changes for each one-step input change, to minimize noise and "glitches" when many digital switches change at once at high speed.

The conversion speed of a flash ADC is limited only by the speed of its comparators and digital logic circuitry. It has a conversion rate measured in speeds ranging from millions of samples per second (MSPS) to over a billion samples per second (GSPS). A common application for this device is digitizing video signals at rates well above 10 MSPS. Flash ADCs are fairly expensive devices when high digital resolution is required, since their complexity grows geometrically with the number of bits ($2^n - 1$ comparators for n bits). So, even an 8-bit flash converter requires 255 comparators and a moderately complex digital decoder. See Section 4.3.8 for more information on high-speed flash ADCs.

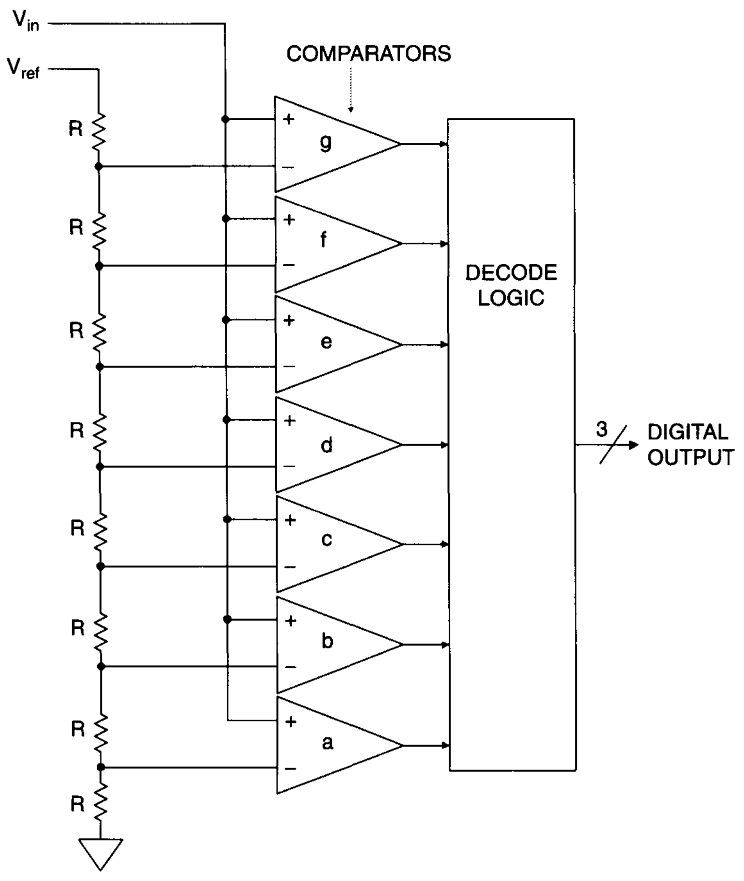


Figure 4-11 3-bit flash ADC.

4.3.6 Sigma-Delta Converter

A fairly new commercial converter is the *sigma-delta* ADC (sometimes referred to as a delta-sigma converter). This device is a low-cost, high-resolution ADC, suitable for low conversion rates. Sigma-delta ADCs typically have 16 to 24-bit resolution, with a usable input signal frequency range of a few Hz to a few kHz. There are some 16-bit sigma-delta ADCs with conversion rates up to 1 MSPS.

A block diagram of a 16-bit sigma-delta converter appears in Figure 4-12. It consists of an analog modulator loop followed by a digital filter. The modulator

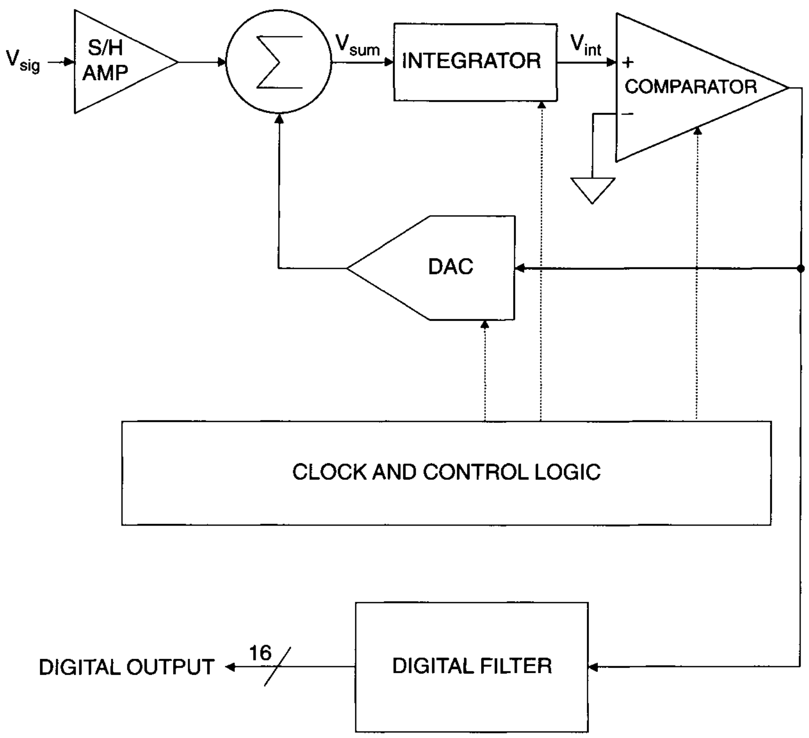


Figure 4-12 Sigma-delta ADC.

operates at a very high clock frequency, effectively oversampling the input signal. It produces a serial data stream, which the digital filter averages to produce a 16-bit output word.

For example, assume the analog signal range (V_{sig}) is −1.0 V to +1.0 V, as well as the DAC output, and the input signal voltage is constant at +0.4 V. The comparator's output will be high and the DAC's output will be +1.0 V if the output of the integrator (V_{int}) is positive. The comparator's output will be low and the DAC's output will be −1.0 V if V_{int} is negative.

Let us follow the voltages at V_{sum} (where the DAC output is summed with the input signal), V_{int} (the integrator output, where V_{sum} is averaged), and the DAC output, as we step through the first few clock cycles, as shown in Table 4-4. Note that the DAC is a single-bit device, with an output of either +1.0 V or −1.0 V.

Initially, at clock cycle 0, we assume that the DAC output is turned off, $V_{sig} = V_{sum} = V_{int}$ (+0.4 V), and the comparator output is 1, producing a DAC

TABLE 4-4
Sigma-Delta Converter, Internal Cycles

CLOCK CYCLE	V_{sum}	V_{int}	COMPARATOR	DAC OUT	
0	+0.4	+0.4	1	+1.0	
1	−0.6	−0.2	0	−1.0	Full Conversion Cycle
2	+1.4	+1.2	1	+1.0	
3	−0.6	+0.6	1	+1.0	
4	−0.6	0	1	+1.0	
5	−0.6	−0.6	0	−1.0	
6	+1.4	+0.8	1	+1.0	
7	−0.6	+0.2	1	+1.0	
8	−0.6	−0.4	0	−1.0	
9	+1.4	+1.0	1	+1.0	
10	−0.6	+0.4	1	+1.0	
11	−0.6	−0.2	0	−1.0	

output of +1.0 V, to be subtracted from V_{sum} on the next clock cycle. At clock cycle 1, the first full clock cycle, $V_{sum} = V_{sig} - V_{DAC}$ = +0.4 V − 1.0 V = −0.6 V. V_{int} is simply the previous value of V_{int} plus the new value of V_{sum}, or +0.4 V + (−0.6 V) = −0.2 V. This process continues until the values at clock cycle 1 occur again and the process is repeated. In this example, the conversion process starts repeating at clock cycle 11. Hence, 10 clock cycles are required to complete the conversion. If the analog voltage of the DAC output is averaged over those 10 cycles, we get a value of +4.0/10 = +0.4 V, the value of V_{sig}. Since the digital filter sees the same numbers as the DAC, its output will also be +0.4 V, but as a digital representation.

Note that the number of clock cycles required for conversion varies with the value of V_{sig}. If we used a V_{sig} value of +0.2 V, only five clock cycles would be required. So, if high resolution at low sampling rates is adequate, the sigma-delta ADC is a good selection and a strong competitor to dual-slope ADCs.

4.3.7 Other ADC Variants

Many current ADC ICs use variations on the techniques we have previously examined, along with additional features such as input multiplexers, sample-and-hold amplifiers, and programmable gain amplifiers. Some sigma-delta

ADCs have programmable filters for signal conditioning. There are ADCs with multiple channels and programmable characteristics that are called data acquisition systems by their manufacturers.

One important variant is the *serial* ADC. For the ADCs previously discussed, the output digital data was presented in a parallel format, with all bits available simultaneously. This parallel approach forces the number of pins on a monolithic ADC package to increase as the resolution (number of bits) increases, along with the overall package size.

For medium-speed ADCs (up to about 1 MSPS) many IC manufacturers produce devices with serial outputs. For these converters, there is a single data line that is time-multiplexed: each bit of the output digital word is present in sequential order, for a fixed amount of time, usually one clock cycle (see Chapter 8 for a discussion of serial signals). These serial interfaces usually require only two or three wires: a data line, a clock line, and sometimes a control or synchronization line. This enables manufacturers to produce high-resolution (12 to 16-bit) ADCs in 8-pin surface-mount IC packages as small as 3 mm × 5 mm.

There are also ADCs designed for low-power applications, such as battery-powered accessories. These ICs can operate from low power supply voltages. They usually have a "sleep" mode that drastically reduces power dissipation when not actively converting data. This low-power mode can be initiated via an external command or automatically after a predefined idle time.

4.3.8 High-Speed ADCs

In recent years, both the speed and resolution of ADCs have increased. Commercial ADC ICs are available up to 1000 MSPS (for eight-bit resolution)—this speed is one sample every nanosecond. Even at higher resolutions, ADC speeds have increased significantly. Currently, there are 10-bit ADC ICs as fast as 100 MSPS, 12-bit ADCs over 50 MSPS, 14-bit devices up to 10 MSPS, and 16-bit converters up to 5 MSPS.

Very High Speed Flash ADCs For resolution up to eight bits, flash ADCs are still the fastest converters available, currently with speeds up to 1000 MSPS or higher. As with high-speed analog circuits, these very high speed ADCs require great care in their implementation. At very high speeds (typically above 100 MSPS) these ADCs no longer use standard TTL digital signal levels. Instead, ECL levels are used, as with high-speed DACs (see Section 4.2.7).

Even with ECL signal levels, many of the fastest ADCs (at 500 MSPS or above) employ a 1:2 demultiplexer at their data output, producing two digital output words. Each output changes at half of the sampling rate, giving the external circuit a chance to capture the ADC data. For example, the fastest ECL clock frequency is around 500 MHz, so a single ECL latch needs 2 nsec to store its data.

Pipelined ADCs At high resolution, such as 12 and 14-bits, a technique called *pipelining* is used to enable high-speed conversions. A pipeline converter consists of multiple ADC stages of low resolution. The analog signal is captured by a sample-and-hold amplifier, to keep the input constant during the conversion process. Each ADC stage performs a conversion and passes its amplified quantization error (or residue) to the next stage for continued conversion. This residue is generated by passing the local stage's ADC output through a DAC and subtracting the result from the buffered input analog signal. After all stages have completed their conversions, logic circuitry combines the result into an output word, usually employing digital error-correction techniques.

The multiple ADC stages run in parallel, performing a local conversion for each clock cycle. This means that the output data conversion rate is equal to the clock rate, producing a high-speed ADC that is not slowed down by more bits of resolution. However, there is a delay between the time the analog signal is sampled and when the digital output word for that sample is available. This latency is the pipe delay, measured in clock cycles. It is determined by the number of internal stages the ADC employs. Pipe delays of 7–14 clock cycles are common. Figure 4-13 shows the timing for a pipelined ADC with a 7-clock latency.

A pipelined ADC is useful for continuously sampling a signal (under normal circumstances the clock must be constantly running). The conversion

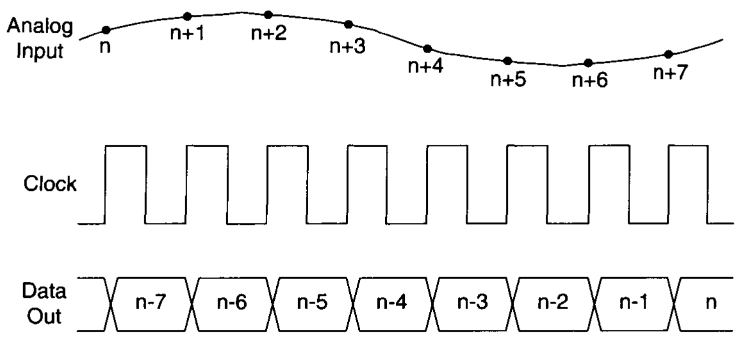


Figure 4-13 Pipelined ADC with 7-clock latency.

time required is much slower for the first sample, only. So, intermittently sampling with a pipelined ADC is about as fast as using a successive-approximation ADC.

Other High-Speed ADC Considerations Many high-resolution (12-bit and above), high-speed ADCs employ differential analog inputs. A differential input consists of two signals, each 180° out of phase with the other. This greatly reduces the reception of common-mode noise. It does add the extra complexity of converting a single-ended (ground-referenced) analog signal to a differential one, but it is worth the improvement in data quality. There are now single ICs available that translate single-ended to differential signals.

Many high-speed ADCs are available as multiple converters in a single IC package—some contain three converters and are used for digitizing high-resolution analog video waveforms (R, G, and B signals). These devices may contain programmable gain amplifiers for input scaling and digital data storage for output buffering, often in the form of a first-in-first-out (FIFO) memory. A FIFO buffer allows a relatively slow device to read the ADC output without losing data, even at high conversion rates. In a FIFO, data is independently stored and retrieved at different rates.

4.3.9 ADC Characteristics

After exploring some of the common ADC techniques, a discussion of their major characteristics is in order. The most important ADC parameters are resolution and sampling rate.

ADC Resolution An ADC's *resolution* is the smallest change it can detect in a measurement. This value is actually a percentage of the full-scale reading, but it is commonly specified as the number of output bits. An n-bit ADC has 2^n possible output values and a resolution of one part in 2^n. For example, a 10-bit ADC has a resolution of approximately 0.1% (1/1024). High resolution (more bits) is usually desirable in an ADC. Note that an ADC's accuracy can be no better than its resolution, for an individual reading.

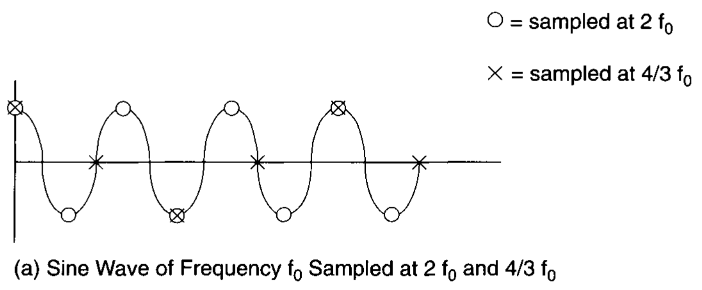
ADC Sampling Rate *Sampling* or *conversion rate* is the ADC specification most often examined. It is the number of readings completed every second. This parameter is extremely important when rapidly changing signals are measured. It is obvious that if a signal frequency is higher than the sampling rate, rapid signal variations can be missed when they occur between consecutive ADC samples. This is true whether the ADC takes an instantaneous analog measurement, using a sample-and-hold amplifier to keep the value constant for the conversion cycle, or whether the signal value is averaged (with an integrator) during the conversion cycle. In fact, a successive-approximation ADC can produce highly erroneous results if the input signal varies significantly during a conversion cycle.

The Nyquist Theorem For an analog signal to be accurately digitized by an ADC, it must be sampled at a rate at least two times the highest frequency component in that signal. To put it another way, only signals whose highest frequency components are no more than one-half the sampling frequency can be accurately digitized. This maximum signal frequency is called the Nyquist frequency and this rule is called the Nyquist theorem.

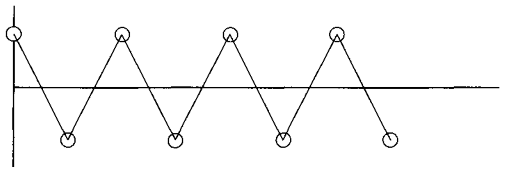
Aliasing When a signal is sampled too slowly (it contains frequency components above the Nyquist frequency), the digitized waveform is distorted. This distortion is called aliasing. It is the result of mixing or beating between the signal frequencies and the sampling frequency. Low-frequency harmonics composed of the differences between the signal and sampling frequencies are recorded instead of the signal itself.

Figure 4-14 shows a simplified example of aliasing, using a single-frequency signal. Figure 4-14a shows a sine wave of fixed frequency, f_0. If that signal was digitized at a rate of $2f_0$, the samples taken would produce a waveform with a frequency of f_0, as shown in Figure 4-14b. The only distortion here is that the digitized waveform appears to be a triangle wave instead of a sine wave. If a sampling rate much higher than $2f_0$ was used, the digitized waveform would "fill in" more, and it would better approximate a sine wave. If the signal was digitized at a rate of only $(4/3)f_0$, the samples would produce a waveform of frequency $(1/3)f_0$, as shown in Figure 4-14c. This result of aliasing is the difference frequency between the sampling rate and signal frequency, which is $(4/3 - 1) \times f_0$. If the sampling rate was equal to the signal frequency, the digitized waveform would be a constant value.

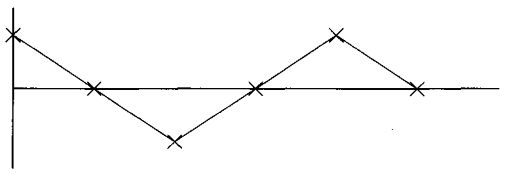
In practice, an ADC's sampling rate should be much higher than twice the maximum signal frequency. A value of five times is a good choice. In most data acquisition systems, the analog input is filtered to eliminate any signal components above the Nyquist frequency. This is often referred to as an *anti-aliasing filter*. For such a low-pass filter to produce adequate attenuation at the Nyquist frequency, it should have a cutoff frequency well below that point, requiring a sampling rate many times higher than the maximum frequency of interest.



(a) Sine Wave of Frequency f_0 Sampled at 2 f_0 and 4/3 f_0



(b) Waveform Reconstructed from 2 f_0 Samples



(c) Waveform Reconstructed from 4/3 f_0 Samples

Figure 4-14 Examples of aliasing.

ADC Accuracy Another important ADC characteristic is its *absolute accuracy,* which is the measure of all error sources. This is sometime referred to as the total unadjusted error. It is the difference between the ideal input voltage and the actual input voltage (range) to produce a given output code, usually expressed as a percentage of full scale (i.e., ±1 LSB). It is possible for a converter's absolute accuracy to be better than its resolution, for multiple readings. By definition, a converter's resolution is 1 LSB. It is not uncommon to find a commercial ADC with an ideal absolute accuracy of ±0.5 LSB. The sources contributing to the total unadjusted error include offset and linearity errors.

An error-free 3-bit ADC transfer curve is displayed in Figure 4-15a, showing digital output code versus analog input voltage, as a fraction of full-scale input. As the resolution of the ADC increases, the "coarseness" of this

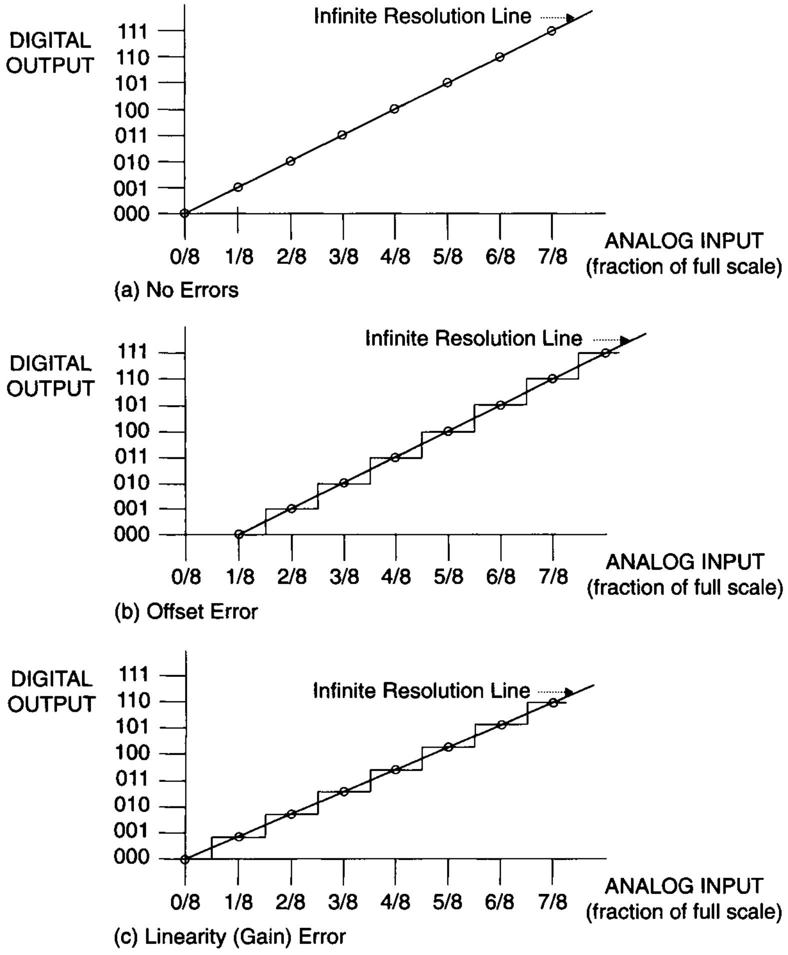


Figure 4-15 3-bit ADC transfer curves illustrating errors.

curve decreases and it approaches a straight line, shown as the infinite resolution line in the figure.

An offset error would move the entire curve to the left or right, unchanged. This type of error can be corrected by adjusting the analog reference voltage. Figure 4-15b shows an offset error of 1 LSB.

A linearity or gain error would be equivalent to having the slope of the infinite resolution line vary, producing a larger error for larger input values. This would be more difficult to correct for, especially if it was temperature dependent. Figure 4-15c shows a linearity error of less than 1 (the gain drops at larger inputs).

Special-Purpose ADC Approaches The ADC techniques discussed in this chapter have been standard, general-purpose approaches, in common use. Sometimes, a data acquisition system can be tailored to a special application for increased performance (one hopes, without a significant cost penalty). One class of special applications particularly amenable to unique ADC systems is the realm of repetitive signals. These are identical waveforms that can be produced multiple times, without any significant change. Basically, these are static measurements under complete experimental control.

This type of repetitive system allows us to use a high effective sampling rate based on a relatively slow ADC. Let us assume that the waveforms of interest have measurable energy up to 10 MHz. We need to sample at 20 MHz, which at high resolution (such as 16 bits) would require a very expensive ADC or multiple ADCs. We can get by with a high-resolution, slow (i.e., 10 kHz sample rate) ADC by adding a sample-and-hold (S&H) amplifier and a timing controller (or use the S&H amplifier within an ADC).

The idea here is to take one narrow sample of the waveform for each repetition of the waveform. The S&H amp must be able to capture an analog voltage with a 50-nsec window (equivalent to a 20-MHz sample rate). The timing circuit must be able to step through the waveform in 50-nsec increments. For each repetition of the waveform, the next 50-nsec aperture is captured and digitized. The ADC's maximum conversion rate of 10 kHz determines the maximum waveform repetition rate. If the width of the waveform is 100 μsec, it would take 2000 repetitions or 200 msec to sample it at effectively 20 MHz. See Chapter 14 for an example of this technique, sometimes called *equivalent time sampling*.

This survey of DACs and ADCs should help you decide which commercial hardware solutions are best suited to your own data acquisition problems, or whether to build your own special-purpose system, as well as what performance to expect from a commercial product.

CHAPTER 5

The PC

A computer is the heart of any contemporary data acquisition system. In the early 1980s minicomputers were the workhorses of most science and engineering labs. Hardware was expensive, most software had to be written in-house, and performance was barely adequate for all but the most expensive systems. Two decades later, PCs (personal computers) are commonplace throughout the scientific and engineering communities. The low cost and high performance of PCs made them the ideal platform for most data acquisition tasks. In addition, a plethora of high-quality commercial software is available for all imaginable PC applications, including data acquisition and analysis.

The typical high-end engineering desktop computer is the workstation. This is usually a system with several hundred megabytes (Mbytes) of volatile memory, a large-screen, high-resolution video display, and a large amount of fast on-line storage (typically a hard disk drive of well over 10 gigabytes). In addition, it would have a network connection and a fast microprocessor (possibly a RISC CPU, or reduced instruction set computer) or several microprocessors in parallel. A workstation will often run the UNIX operating system or a version of Windows NT. This is usually the platform of choice for a very high performance data acquisition system, at a relatively high price.

Even though workstations are more powerful than standard PCs, the distinction blurs when looking at high-end PCs. In fact, the major differences between a high-end PC and a low-end workstation are price, CPU, operating system, and software availability. Now that a variation of the UNIX operating system, called Linux, is running on many PCs, workstation–PC distinctions are reduced further.

There are several popular classes of computers useful as platforms for data acquisition systems. The ones we will examine in this book are based on the original IBM PC/XT/AT bus (now called the ISA bus) and the newer PCI bus. The IBM Micro Channel bus and the Apple NuBus (which were covered in the first edition of this book) are now obsolete. The IBM and compatible machines are based on Intel's 80x86 and Pentium microprocessor (or CPU, *central processing unit*) families.

There are many members in Intel's 80x86 and Pentium families. The original device was the 8086, a "true" 16-bit CPU. It had a 16-bit wide data bus and a 20-bit address bus, producing a 1-Mbyte address range. The original IBM PC and PC/XT used Intel's 8088 CPU, which was effectively an 8086 with only an 8-bit external data bus and a 20-bit address bus, for a 1-Mbyte address range, while keeping the same 16-bit registers internally for 8086 software compatibility. When the IBM PC was released, in 1981, this hybrid approach of 16 bits internal and 8 bits external was common.

The IBM PC/AT used Intel's 80286 CPU, which employed a true 16-bit architecture, a 16-bit external data bus, and a 24-bit address bus, for a 16-Mbyte address range. It was software-compatible with the 8088 while providing faster processing speed and additional features. The expansion bus of the IBM PC/AT computer, a superset of the PC/XT expansion bus, eventually became an explicit standard: ISA (*industry standard architecture*).

The next Intel processor was the 80386, which used a 32-bit architecture both internally and externally. It had a 32-bit external data bus and a 32-bit address bus, for a 4-gigabyte (Gbyte) address range. IBM switched to its newer PS/2 line of PCs with the Micro Channel bus to use the 80386 and later CPUs. Many other manufacturers stayed with the original AT (ISA) bus, with modifications for 32-bit wide memory to accommodate 80386 machines. The ISA bus has been replaced in mainstream desktop PCs by the PCI bus. However, the ISA bus is still very popular in embedded PCs, including PC-104 systems (see Chapter 12).

The next Intel processor in this family was the 80486. It was another 32-bit device with the same bus widths and features as the 80386 plus additional integrated functions, such as a floating-point processor. IBM used this processor in its higher end PS/2 systems while other manufacturers put it into ISA computers.

After the 80486, Intel introduced the Pentium family of microprocessors (and abandoned the 80x86 naming convention, which could not be trademarked). Pentium processors had a 32-bit internal architecture (registers) with a 64-bit internal data bus. The external address and data buses were each 32 bits wide. Pentiums were based on superscalar architecture, which used two pipelines for parallel processing. They also had better cache memory than 80486 processors. Pentium processors had CPU speeds ranging from 60 MHz to 200 MHz, although their maximum external bus speed was about 60 MHz.

Intel kept the trademarked Pentium name for its later families of CPUs, even as their technology evolved. The next generation of Intel processors began with the Pentium Pro, which increased addressing to 36 bits (for a 64-Gbyte range). More significantly, the Pentium Pro had a RISC-based core, more parallel processing hardware, and a secondary memory cache. However, its improved performance was significant only when running fully 32-bit software and operating systems, such as Microsoft Windows NT.

The Pentium II followed, using essentially the same core logic as the Pentium Pro. It had some performance enhancements, including MMX instructions for improved multimedia support, and had speeds up to 400 MHz. The Celeron, was a lower-cost, lower-performance variant of the Pentium II.

The Pentium III was Intel's next generation of microprocessors. It used the same core logic as the Pentium II series but had enhanced performance for certain types of data processing. This was done with *single instruction multiple data* (SIMD) instructions, which operated on entire blocks of data in parallel. The Pentium III had internal processor speeds over 1 GHz.

As of this writing, the newest Intel processor is the Pentium 4, with speeds up to 2 GHz. It is optimized for digital video and Internet technologies, using Intel's NetBurst microarchitecture. This encompasses a 20-stage pipeline, a double-speed arithmetic logic unit (ALU), a 400 MHz memory bus, and additional SIMD and MMX instructions.

Manufacturers other than Intel produce microprocessors for IBM-type PCs, most notably Advanced Micro Devices (AMD) and Cyrix Corporation. Their products are software compatible with Pentiums and are often of comparable performance.

5.1 IBM PC/XT/AT and Compatible Computers

We will now look in depth at the IBM PC/XT/AT class of PCs and their compatibles (sometimes called clones). First we will examine the IBM PC/XT computer, which is based on the Intel 8088 CPU. It has an external data bus 8 bits wide and an address bus 20 bits wide, for an address range of 1 Mbyte. Even though 8088-based PCs have long been obsolete as desktop computers, they are still produced in small form factors (such as PC-104) for embedded PC applications (see Chapter 12).

5.1.1 Memory Segmentation

One idiosyncrasy of the 16-bit processors in this Intel CPU family is the way 20-bit physical addresses are generated from 16-bit registers. Intel uses an approach called segmentation. A special segment register specifies which

64-Kbyte section of the 1-Mbyte address space is being accessed by another 16-bit register. A segment register changes the memory address accessed by 16 bits at a time, because its value is shifted left by 4 bits (or multiplied by 16) to cover the entire 20-bit address space. The segment register value is added to the addressing register's 16-bit value to produce the actual 20-bit memory address. Four segment registers and five addressing registers are available in an 8088, all 16 bits wide.

For example, when the stack is accessed, the 16-bit value in the Stack Segment (SS) register is shifted left by four bits (to produce a 20-bit value) and added to the 16-bit Stack Pointer (SP) register to get the full 20-bit physical address of the stack. The value added to the segment is referred to as the offset. The usual notation is *segment:offset.* So, if the code segment (CS) contained B021h and the instruction pointer (IP) contained 12C4h, the segmented notation is B021:12C4 and the physical location addressed would be B14D4h.

Note that throughout this book, most addresses will be presented in *hexadecimal* (base 16) notation (with digits 0–9, A–F) using a trailing h. For example, 100h = 256 (decimal).

5.1.2 Motherboards

The heart of any PC is a single printed circuit board (PCB) referred to as the system board or the *motherboard.* It contains the CPU and the system's memory, timing, and control functions, as well as external interface capabilities (*input/output* or I/O). This external I/O is usually available through special connectors on the motherboard, often referred to as expansion slots. Various cards are plugged into these slots, including display adapters (video controllers), disk drive controllers, and parallel and serial interfaces, as well as boards for data acquisition.

Newer PCs have many of these common functions (video control, disk drive control, etc.) integrated into the motherboard with appropriate connectors. USB ports (see Chapter 8) are also built into new PC motherboards, simplifying connections to external peripherals such as scanners, printers, mice, and even data acquisition hardware.

5.2 The IBM PC/XT

A simplified block diagram of a PC/XT motherboard is shown in Figure 5-1. This motherboard contains: the CPU, an optional coprocessor (Intel 8087) for floating-point math, eight hardware interrupts, four direct memory access (DMA) channels, three timer/counter channels, read/write memory (usually

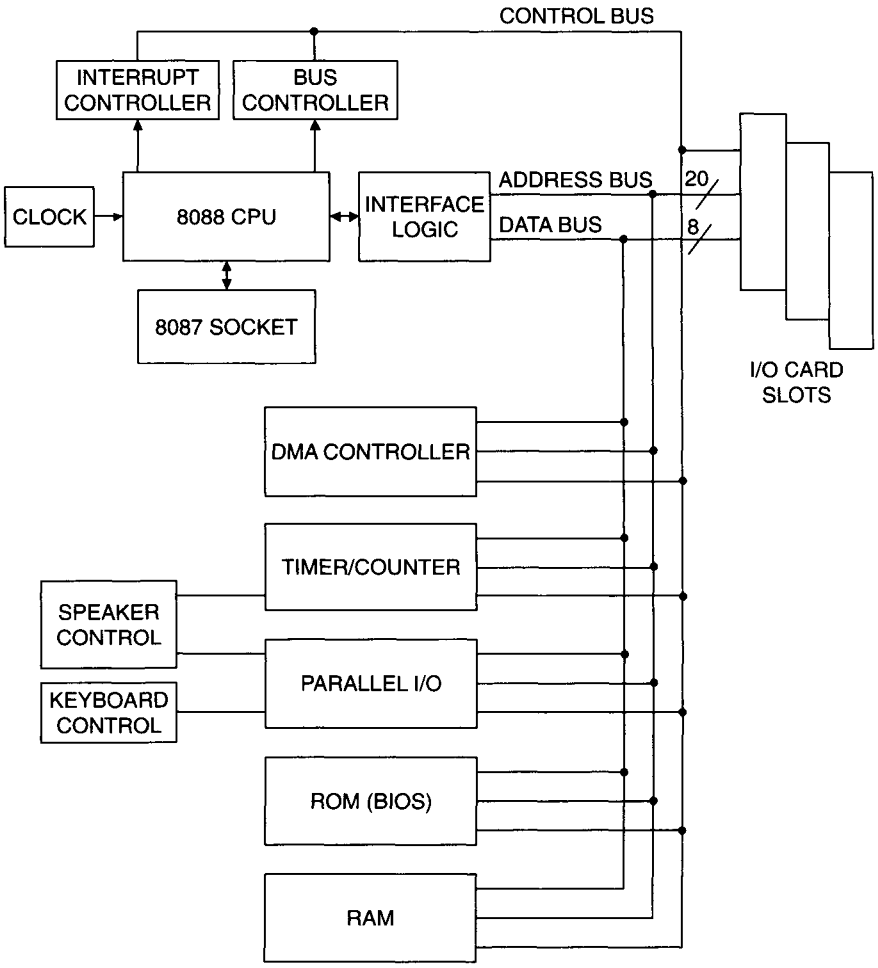


Figure 5-1 PC/XT motherboard block diagram.

referred to as random access memory, or RAM), read-only memory (ROM) and all the required control logic and interfaces for the external I/O slots. The 20-bit address bus, 8-bit data bus, and various control lines go to the I/O slots to support numerous peripherals.

Even though the 8088 can address 1 Mbyte of memory, only 640 Kbytes of RAM is usable on the PC/XT, in the address range 0 to 9FFFFh. The upper 360 Kbytes are reserved for system ROM and memory on expansion cards,

TABLE 5-1
PC/XT Memory Map

ADDRESS	MEMORY AREA	MEMORY TYPE
FFFFFh		
	SYSTEM BIOS	
F0000h		
	ROM EXPANSION	
CC000h		ROM
	HARD DRIVE BIOS	
C8000h		
	ROM EXPANSION	
C0000h		
	VIDEO ADAPTER AREA (DISPLAY BUFFERS)	ADAPTER RAM
A0000h		
	TRANSIENT PROGRAM AREA	
	COMMAND.COM RESIDENT PORTION	SYSTEM RAM
	BUFFERS, DRIVERS	
	DOS KERNEL	
	USED BY BIOS	
00400h		
	INTERRUPT VECTORS	
00000h		

which plug into the I/O expansion slots on the motherboard. A simplified PC/XT memory map is shown in Table 5-1.

5.2.1 I/O Addressing, Interrupts, DMA, and Timers

For communicating with peripheral, nonmemory (I/O) devices, the 8088 CPU supports both I/O mapped and memory mapped I/O. I/O mapping separates I/O addressing from memory addressing, so I/O ports can be directly and easily accessed, even if they have the same addresses as memory locations, by using separate control signals. In memory mapping, I/O ports look like memory addresses and use up part of the memory addressing space. In the PC/XT design, I/O mapping is used. Although the 8088 will support 16 bits

TABLE 5-2
PC/XT I/O Address Map

I/O ADDRESS	USE	LOCATION
000–00Fh	DMA CONTROLLER	ON MOTHERBOARD
020–021h	INTERRUPT CONTROLLER	
040–043h	TIMER	
060–063h	PPI (8255)	
080–083h	DMA PAGE REGISTERS	
0A0h	NMI MASK REGISTER	
200–20Fh	GAME ADAPTER	ON ADAPTER CARDS
210–217h	EXPANSION UNIT	
2F8–2FFh	ASYNC ADAPTER (COM2)	
300–31Fh	PROTOTYPE CARD	
320–32Fh	HARD DISK DRIVE ADAPTER	
378–37Fh	PRINTER ADAPTER	
380–38Ch	SDLC COMM ADAPTER	
390–393h	CLUSTER ADAPTER	
3A0–3A9h	BISYNC ADAPTER	
3B0–3BFh	MONO DISPLAY/PRINTER ADAPTER	
3D0–3DFh	CGA ADAPTER	
3F0–3F7h	DISKETTE DRIVE ADAPTER	
3F8–3FFh	ASYNC ADAPTER (COM1)	

of I/O addressing, only 10 bits are used here (for a total of 1024 I/O addresses). This I/O space is divided into two regions of 512 locations each. The lower 512 addresses (0 to 1FFh) are used exclusively on the motherboard. The upper 512 addresses (200h to 3FFh) are decoded by interface cards connected to the I/O slots. An I/O address map for the PC/XT is shown in Table 5-2.

The PC/XT has nine interrupt lines or levels, with unique priorities. The highest priority interrupt is the NMI (nonmaskable interrupt), used for trapping serious system problems, such as memory (RAM) parity errors. The next two interrupts, IRQ0 and IRQ1, are also used only by the motherboard (IRQ1 interrupts the processor whenever the keyboard is hit). The other six interrupts, IRQ2–IRQ7, are available for use by cards in the external I/O slots. The lowest priority interrupt, IRQ7, is allocated to a parallel printer port.

Note that very often, peripheral board manufacturers use interrupts in nonstandard ways for functions not previously defined. The same problem holds true for the use of I/O addresses and even with memory addresses above 640 Kbytes (the limit of MS-DOS). This is especially the case for some PC/XT data acquisition cards. If two cards in the same PC try to use the same interrupt or address, they will malfunction. This is an incompatibility or an address clash. The solution is to change the interrupt/address selection on one or the other card, or remove one card entirely. In newer PCs with plug-and-play support, the system automatically assigns addresses and thus avoids this problem.

Another important PC/XT feature is the use of direct-memory access (DMA). DMA hardware allows data to be transmitted very quickly between a peripheral device and system memory without the CPU's intervention. Programmed I/O transfers, under CPU control, are inherently slower than DMA I/O transfers. DMA is especially useful for accessing hard disk drives. The CPU initializes the DMA controller with the required information and the DMA controller takes over the system bus, managing the data transfer.

There are four DMA channels in a PC/XT system. The highest-priority DMA channel (DMA channel 0) controls memory refresh, as discussed below. The other three DMA channels (1–3) are available for use by external I/O cards. Care must be taken in using DMA transfers, which can prevent normal CPU actions and result in a system crash.

The PC/XT contains three programmable timer/counters. The first timer/counter (channel 0) is implemented as a general-purpose time-of-day clock, producing a level 0 interrupt (IRQ0) approximately every 55 milliseconds. The second timer/counter (channel 1) times the DMA cycles for memory refresh, as described below. The third timer/counter (channel 2) controls the speaker's tone generation. If you need to use one of these timer/counters for other applications, try to use channel 2 only! This will not interfere with any critical system functions, whereas using other channels might.

5.2.2 PC/XT Memory: RAM and ROM

The PC/XT's main system memory consists of dynamic RAM. This read/write memory starts at address 0 and can extend up to 640K (9FFFFh). This is the memory used by the operating system, DOS, and is available for loading and running programs, along with any transient data storage required by those programs.

Two types of RAM devices are static and dynamic. Both memories retain their contents only while power is applied to them. Dynamic RAM (DRAM), in addition, requires a periodic read access (on the order of every few milliseconds) to retain its memory. This process is called a refresh cycle. It is required because each memory cell in a dynamic RAM acts like a capacitor whose charge slowly leaks off over time; it needs to be periodically recharged to the appropriate voltage (logic level).

Even though DRAM refresh uses up a finite amount of CPU time, it is commonly used in PCs because of its lower price-per-bit than static RAM and its higher density (more bits per package). When the original IBM PC appeared in 1981, its motherboard supported only 64 Kbytes of DRAM, using 16-Kbit ICs. A decade later, 4-Mbit DRAM ICs were common. Today (as of writing the second edition), 512-Mbit DRAMs are available.

Early DRAMs were 1 bit wide, so a 1-Mbit DRAM was configured as 1,048,576 (2^{20}) addresses by 1 bit. Most PC/XT machines used nine DRAMs to produce a memory block 1 byte (8 bits) wide, with the additional bit used for *parity checking*. This is a hardware scheme to detect if there was an error in reading memory. The DRAM refresh time on a PC/XT system used approximately 7% of the available system time. This was accomplished using DMA channel 0 and timer channel 1.

Newer DRAM ICs are organized as either 4, 8, or 16 bits wide, to minimize chip count when supporting a 32-bit or 64-bit wide memory bus. Memory ICs also use newer architectures to speed them up and keep pace with faster CPUs. These type of memories include extended data output (EDO) and synchronous DRAM (SDRAM).

The PC/XT's ROM contains the nonvolatile memory required to start up the system. This includes hardware initialization, power-on diagnostics (including a memory test), and a *bootstrap* program. The bootstrap allows the PC to load the operating system and start running it, usually from a hard disk drive or diskette. This allows for the flexibility to upgrade or even change the operating system a PC uses, without any hardware changes. Other important contents of the system ROMs include the programs needed for low-level control of various hardware I/O devices (such as disk drives, displays, and keyboard). This is referred to as the basic input/output system, or BIOS (sometimes denoted ROM BIOS). This *firmware* (software resident in a nonvolatile memory IC) is continuously used by the operating system for interfacing to all system I/O devices. If nonstandard system hardware is not supported by the BIOS, usually a special piece of software, called a driver, must be loaded into the operating system before the hardware can be used. An example of this would be support for a tape drive.

Newer PCs store the BIOS in flash memory, which is a form of rewritable ROM. This allows the BIOS to be upgraded via software without having to replace any internal ICs (ROMS). In addition, many PCs use a portion of the upper 360 Kbytes of basic PC memory (RAM) to temporarily store a copy of the BIOS program, often called shadow ROM. This is a useful feature because system RAM has faster access time than most ROM chips and the repeated use of BIOS functions by most software gets sped up.

The most common operating system originally used with PC/XT/AT computers was DOS (disk operating system), often specified as IBM-DOS or MS-DOS (for Microsoft, its developer). It is a single-user, single-task operating system with a limited memory usage of 640 Kbytes (see Chapter 7 for a more detailed discussion of DOS, Windows, and other PC operating systems).

The system ROM is located in high memory addresses, above F4000h. Expansion cards plugged into the I/O sockets may also contain ROM, for integration into system code. This ROM may be present within the address range of C0000h–DFFFFh. If it contains valid information, the system will be able to execute the code (instructions) it contains. This was a common approach for early hard disk drive controllers or special video display adapters.

5.2.3 PC/XT Expansion Bus

The key to the PC/XT's flexibility is its expansion bus, with connectors for external I/O cards. Figure 5-2 shows the bus connections to an expansion slot. This bus gives an add-in card access to all the system address, data, and control lines, except for those dedicated to the motherboard, such as IRQ0, IRQ1, and DRQ0.

Here is a brief description of the I/O bus signal lines, designated pins A1–A31 and B1–B31 (as shown in Figure 5-2): Lines A0–A19 (pins A31–A12) are the address bits used for memory and I/O addressing, where A0 is the least significant bit (LSB) and A19 is the most significant bit (MSB). These are output lines, relative to the motherboard. Similarly, signal lines D0–D7 (pins A9–A2) are the data bits, used for all data transfers (including DMA cycles), where D0 is the LSB and D7 is the MSB. These lines are bidirectional (both input and output).

Signals DRQ1–DRQ3 (pins B18, B6, B16) are the DMA request lines for channels 1–3. They are input lines, used by external devices to initiate a DMA cycle. Signals DACK0–DACK3 (pins B19, B17, B26, B15) are DMA acknowledge lines. They are outputs used to indicate DMA activity, acting as handshake signals for their respective DRQ lines.

Signals IRQ2–IRQ7 (pins B4, B25–B21) are interrupt request input lines, used by an external device to generate a CPU interrupt. IRQ2 is the highest priority and IRQ7 is the lowest. The system has to be initialized prior to an interrupt generation for it to be properly serviced.

Signal IOR (pin B14) is an output line indicating an I/O read cycle. This tells the external I/O device being addressed to place its data on the bus.

	B	A	
B1	GND	-I/O CH CK	A1
	RESET DRV	D7	
	+5V	D6	
	IRQ2	D5	
	-5VDC	D4	
	DRQ2	D3	
	-12VDC	D2	
	Reserved	D1	
	+12VDC	D0	
B10	GND	I/OCH RDY	A10
	-MEMW	AEN	
	-MEMR	A19	
	-IOW	A18	
	-IOR	A17	
	-DACK3	A16	
	-DRQ3	A15	
	-DACK1	A14	
	DRQ1	A13	
	-DACK0	A12	
B20	CLK	A11	A20
	IRQ7	A10	
	IRQ6	A9	
	IRQ5	A8	
	IRQ4	A7	
	IRQ3	A6	
	-DACK2	A5	
	T/C	A4	
	ALE	A3	
	+5VDC	A2	
	OSC	A1	
B31	GND	A0	A31

Figure 5-2 PC/XT I/O card slot connector.

Similarly, IOW (pin B13) is an output signal indicating an I/O write cycle. This instructs an external I/O device to read data from the system bus. MEMR and MEMW (pins B12, B11) are the equivalent read and write output lines for reading from and writing to memory addresses.

Signal I/O CH RDY (pin A10) is an important input line. It is used by slow memory or I/O devices to lengthen a read or write cycle. This is known as inserting *wait states*. It allows slower (and less expensive) peripherals to interface to the PC/XT, with only the penalty of more time required for a data transfer. If this signal is not used properly, it can be asserted for too long (more than a few microseconds) and effectively monopolize the system bus, preventing other activities. This could result in a system crash, where DRAM is not being properly refreshed or important interrupts are not being serviced. Figure 6-5, in Chapter 6, illustrates how to safely control I/O CH RDY.

Signal AEN (pin A11) is an output line which is used to prevent the CPU and other devices from accessing the system bus during DMA transfers. Signal ALE (pin B28) is an output line used to latch valid bus addresses by memory and peripheral devices. Signal I/O CH CK (pin A1) is an input line, used to indicate a memory or I/O device parity error. Signal RESET DRV (pin B2) is an output line used to initialize (reset) devices on the bus at system power-on. Signal T/C (pin B27) is an output line that indicates when the maximum DMA transfer count is reached.

Signal OSC (pin B30) is an output line containing a 14.31818-MHz clock, with a 50% duty cycle. This clock may be divided down to provide other clock signals, such as dividing by 4 for the 3.58-MHz color video subcarrier frequency. On original PC and PC/XT systems, it was divided by 3 to provide the main system clock frequency of 4.77 MHz. Signal CLK (pin B20) is an output line containing the main system clock, with a 33% duty cycle. It is often higher than 4.77 MHz in later PC/XT compatible systems. The most common clock frequencies used are 8 MHz and 10 MHz. Obviously, the higher the system clock, the faster the CPU will operate. Overall system performance is not necessarily proportional to this clock frequency. In fact, some slower peripheral cards may not work properly with faster clocks, unless enough wait states are inserted.

The other lines on the I/O bus connector are power for the expansion cards. These lines are +5 V (pins B3, B29), –5 V (pin B5), +12 V (pin B9), –12 V (pin B7) and ground (pins B1, B10, B31). The positive voltage supplies typically have a higher current capability and are regulated to ±5%, as opposed to the negative supplies regulated to ±10% with lower current capacity. The original IBM PC's power supply could only produce approximately 65 watts of DC power, mostly for the +5 V (7 amps, maximum) and +12 V (2 amps, maximum) supplies. Most later PC/XT compatible systems used a power supply providing 120–150 watts of DC power.

For examples of using some of these expansion bus signals, refer to Chapter 6.

5.3 The IBM PC/AT

Now we will examine IBM PC/AT computers and the ISA bus. The original IBM PC/AT and compatible systems were based on the Intel 80286 CPU. This was an expansion of the PC/XT architecture, including the external I/O bus. The PC/AT block diagram is shown in Figure 5-3. The 80286 processor

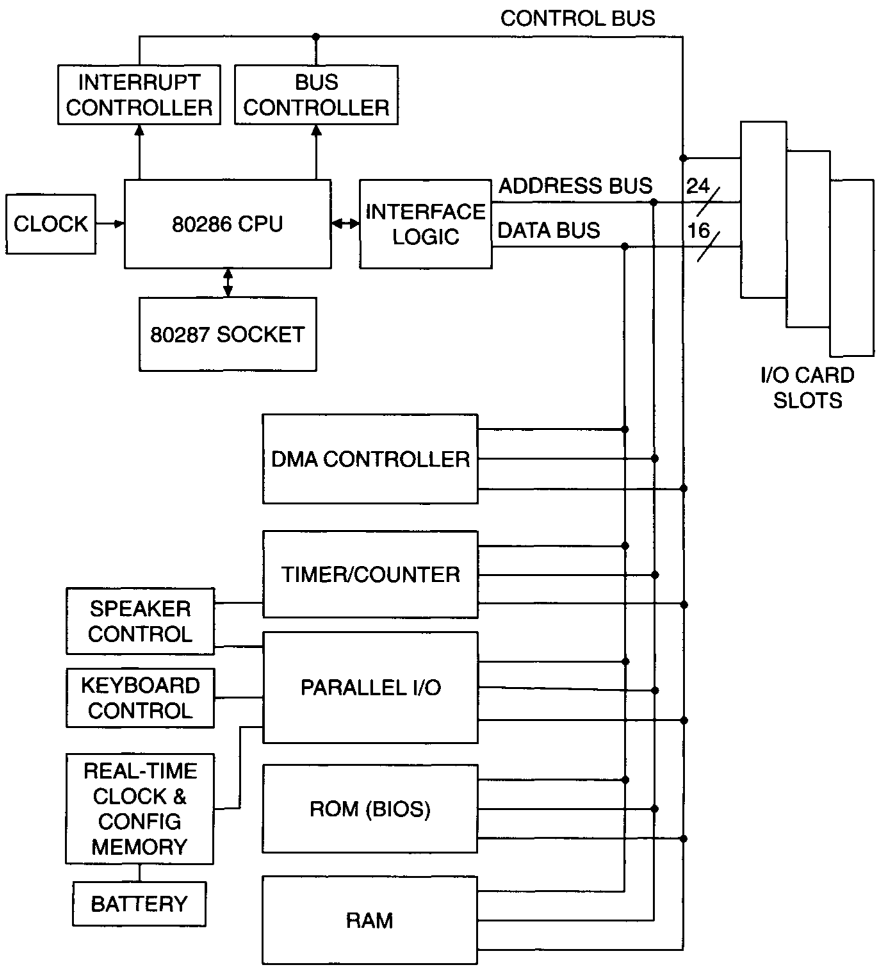


Figure 5-3 PC/AT motherboard block diagram.

increased the number of address bits to 24, for a 16-Mbyte addressing space, and the number of data bits to 16. The motherboard had 16 interrupt levels and seven DMA channels. It still had three timer/counters. New features included a real-time clock with battery-backup CMOS RAM. This small amount of memory stored clock and system configuration data. In addition, the real-time clock included a 1024-Hz timer that could provide DOS programs with much finer timing resolution (approximately 1-msec counts) compared to the XT 18-Hz RTC (approximately 55-msec counts). This new timer was accessed via INT 70h.

The functioning of the IBM PC/AT (usually referred to as an AT or ISA system) was very similar to the PC/XT operation. Because of the higher speed and improved features of the 80286 CPU, overall system performance was enhanced. In addition, external data transfers could be 16 bits at a time, although 8-bit data transfers were still supported. The original IBM PC/AT had a 6-MHz system clock, which was later upgraded to 8 MHz. Most 80286-based AT compatible systems used clocks ranging from 8 MHz up to 16 MHz. The faster systems required memory (RAM) with fast access time (or they had to add wait states to memory access cycles). The IEEE ISA bus standard (IEEE P996-1990) specified an 8-MHz bus frequency while allowing for higher, internal CPU clock frequencies (such as a 33-MHz 80486 PC).

AT systems use two connectors for each external I/O card slot. One is a 62-pin connector, compatible with the single PC/XT I/O connector. The differences are that now pin B4 is IRQ9 instead of IRQ2, pin B19 is REFRESH instead of DACK 0, and previously unused pin B8 is now 0WS. Also, CLK (at pin B20) is faster and has a 50% duty cycle. Most cards designed for the PC/XT bus will work in an AT, as long as they can deal with the higher clock frequency and do not do any special remapping of memory.

5.3.1 PC/AT (ISA) Expansion Bus

As shown in Figure 5-4, AT I/O slots have a new, second connector consisting of 36 additional pins. These lines carry the additional address and data bits, IRQ signals, DMA signals, and special control lines that allow for 16-bit data transfers, zero wait state memory accesses, and multiple CPU operations.

Here is a brief description of these new I/O bus signals: Signal 0WS, added to the original 62-pin connector at pin B8, is an input line used to tell the CPU not to add any wait states to the present bus cycle. This is useful for fast memory and I/O cards. The remaining new signal lines are on the new 36-pin connector, designated C1–C18 and D1–D18. The additional address lines are LA17–LA23 (pins C8–C2). The additional data lines are SD08–SD15 (pins C11–C18). The additional interrupt lines available on the I/O bus (besides IRQ9) are IRQ10–IRQ12, IRQ14, and IRQ15 (pins D3–D7). The additional DMA channel-control signals now available are DRQ0 and DACK0, DRQ5–DRQ7, and DACK5–DACK7 (pins D8–D15).

Additional control lines also exist on the 36-pin connector. MEM CS16 (pin D1) is an input signal used to signify a 16-bit, one wait-state memory transfer. Similarly, pin I/O CS16 (pin D2) is an input signal indicating a 16-bit,

	B	A	
B1	GND	-I/O CH CK	A1
	RESET DRV	SD7	
	+5V	SD6	
	IRQ9	SD5	
	-5VDC	SD4	
	DRQ2	SD3	
	-12VDC	SD2	
	OWS	SD1	
	+12VDC	SD0	
B10	GND	I/OCH RDY	A10
	-SMEMW	AEN	
	-SMEMR	SA19	
	-IOW	SA18	
	-IOR	SA17	
	-DACK3	SA16	
	-DRQ3	SA15	
	-DACK1	SA14	
	DRQ1	SA13	
	-REFRESH	SA12	
B20	CLK	SA11	A20
	IRQ7	SA10	
	IRQ6	SA9	
	IRQ5	SA8	
	IRQ4	SA7	
	IRQ3	SA6	
	-DACK2	SA5	
	T/C	SA4	
	BALE	SA3	
	+5VDC	SA2	
	OSC	SA1	
B31	GND	SA0	A31

	D	C	
D1	-MEM CS16	SBHE	C1
	-I/O CS16	LA23	
	IRQ10	LA22	
	IRQ11	LA21	
	IRQ12	LA20	
	IRQ15	LA19	
	IRQ14	LA18	
	-DACK0	LA17	
	DRQ0	-MEMR	
D10	-DACK5	-MEMW	C10
	DRQ5	SD08	
	-DACK6	SD09	
	DRQ6	SD10	
	-DACK7	SD11	
	DRQ7	SD12	
	+5VDC	SD13	
	-MASTER	SD14	
D18	GND	SD15	C18

Figure 5-4 PC/AT I/O card slot connector.

one wait-state I/O data transfer. Signal SBHE (pin C1) is a bidirectional line used to indicate a data transfer on the upper 8 bits (D8–D15) of the data bus. This line is used by devices that support 16-bit data transfers. Signal MASTER (pin D17) is an input line used by additional processors or DMA controllers to take control of the system bus. This line must be used carefully. If an external device holds the bus too long, system memory may be lost because of lack of DRAM refresh cycles.

Signal MEMR (pin C9) is similar to the original PC/XT bus signal MEMR (pin B12), now called SMEMR. The difference is, the original SMEMR is only active during a memory read cycle within the low 1 Mbyte of memory (original PC/XT address space). MEMR is active on all memory read cycles. Furthermore, SMEMR is an output line while MEMR can be either an output or input. It can be driven by an external CPU. In a similar fashion, signal MEMW (pin C10) is a superset of the original MEMW (pin B11), now called SMEMW. The remaining lines on the 36-pin connector are extra power (+5 V) at pin D16 and ground at pin D18.

The PC/AT power supply provides +5 V, –5 V, +12 V, and –12 V. The positive supplies have much higher current capabilities than the PC/XT power supply. The +5 V supply is rated at approximately 20 amps and the +12 V supply at approximately 7 amps. The overall AT power supply output power is approximately 200 watts, which is typical for most AT compatibles (although some industrial PCs can have power supplies as large as 600 watts).

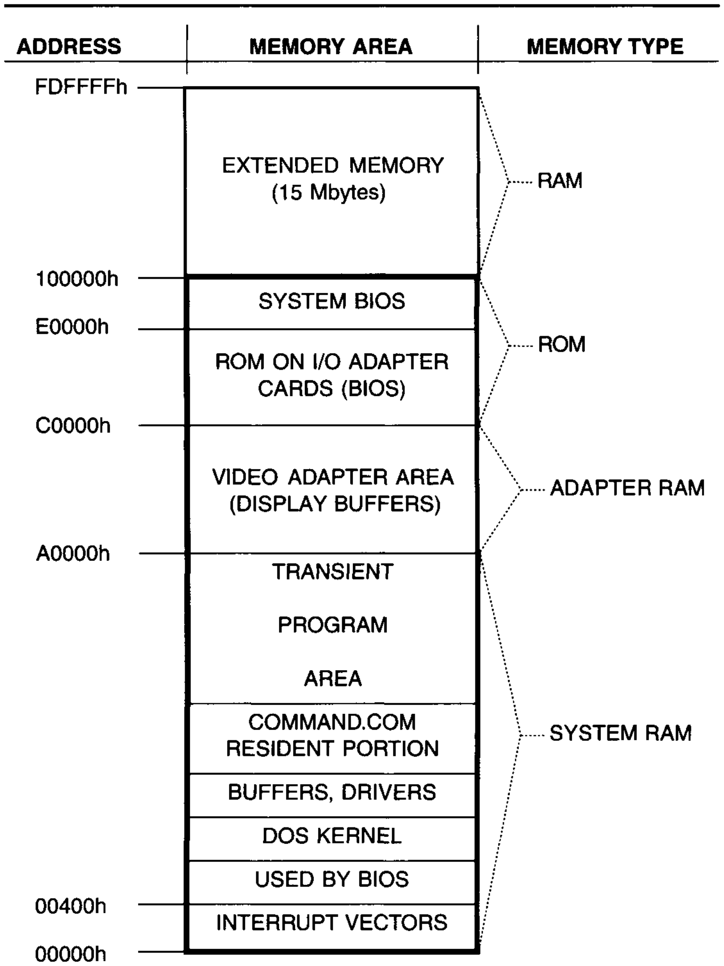
The memory map of the PC/AT is an expansion of the PC/XT's memory map, using a 16-Mbyte memory space, as shown in Table 5-3. Note that the AT motherboard supports 64 Kbytes of ROM, as opposed to 40 Kbytes on the PC/XT motherboard. The PC/XT supported an Intel 8087 math coprocessor IC, for accelerated calculations involving floating-point math. The AT supported an Intel 80287 math coprocessor, for an 80286 CPU. If a system used an 80386 CPU, it would support an 80387 coprocessor. Note that application software must explicitly utilize the math coprocessor for you to realize any benefit from it.

PC manufacturers retained the basic PC/AT architecture as they moved to faster, more powerful CPUs, such as the 80386 and 80486 families. Notably, they increased addressable memory space (since the newer processors had 32-bit address buses for a 4-Gbyte address range) and implemented local buses (such as VESA and PCI) to take advantage of higher CPU speeds. The 80386 processors had internal clock frequencies up to 33 MHz and the 80486 CPUs went up to 100 MHz.

5.4 BIOS

As mentioned above, the BIOS code located in ROM on a PC/XT/AT system handles the low-level software interface to the hardware. For example, to display a character on the video screen you send an appropriate command, along with the character, to the proper BIOS routine. Without the BIOS, you would have to know the intimate details of the video hardware, such as where in physical video memory to write the character for display. If the video display hardware was changed, software that directly addresses the hardware will no longer work. This is known as "ill-behaved" software. On the other hand, if BIOS calls were used, the BIOS will take care of hardware changes and the software can remain the same. This is "well-behaved" software.

TABLE 5-3
PC/AT Memory Map



The penalty for using BIOS calls is a slower response than directly addressing hardware. Also, if a needed function does not exist in the BIOS, the hardware may need to be directly addressed. However, it is desirable to use BIOS functions whenever possible, as they will work universally with nearly all PCs. In addition, modern 32-bit protected-mode operating systems (such as Windows NT and Windows 2000) only allow device driver software, not application software, to directly access hardware.

Some of the I/O facilities provided by BIOS routines support the keyboard, system clock/timer, communications ports, video display, floppy disk drive, hard disk drive, CD-ROM, printer, and system status. Original IBM PCs even had ROM BASIC built into the BIOS.

5.5 PCI and Other Local Buses

As microprocessor frequencies increased, the 8-MHz speed of the ISA bus became a limiting factor to PC performance. A processor could not communicate with external memory or I/O devices nearly as fast as it could process data internally. Several new buses appeared in the PC marketplace. Enhanced ISA (EISA) had a 32-bit data bus and address bus and was backward compatible with ISA cards. It also ran at just 8 MHz, but by doubling the data bus to 32 bits, it doubled I/O throughput. However, EISA never became very popular because of its relatively high cost.

The Video Electronics Standards Association (VESA) developed the VESA Local Bus (VL Bus) primarily for improving video performance. But it also supported many other high-speed peripherals, such as network cards. VL Bus was originally 32 bits wide and had speeds up to 50 MHz. It was very common in PCs built in the early 1990s. However, VL Bus soon became displaced by the PCI local bus.

5.5.1 PCI Overview

Peripheral component interconnect (PCI) was developed by Intel as a processor-independent, high-speed replacement for ISA. It was originally 32 bits wide (address and data) and ran at speeds up to 33 MHz. Later versions support 64-bit data transfers and 66 MHz rates. It accesses up to 4 Gbytes in each of its 32-bit memory and I/O address spaces, using multiplexed address and data lines.

PCI can coexist with other buses, such as ISA, on the same motherboard. Many PCs have both ISA and PCI slots. However, ISA slots are being phased out in most newer desktop PCs (but not necessarily in embedded and industrial PCs—see Chapter 12). In addition, PCI is now used in Apple Macintosh computers. The current revision of the PCI specification (as of this writing in 2001) is 2.2, released in December 1998.

The PCI bus can operate in either a synchronous or asynchronous mode. In synchronous operation, the bus typically runs at the microprocessor's external clock frequency or a submultiple of it. So, a 66-MHz Pentium could synchronously connect to a PCI bus running at half of its clock frequency (33 MHz). In this mode, the standard PCI clock can be between 20 and 33 MHz.

In asynchronous operation, the PCI bus speed is independent of the processor's clock. This mode is often better suited for operating at the maximum PCI bus frequency for the fastest possible performance.

The PCI standard also supports cards that cannot operate at the full bus speed (33 or 66 MHz), using flow-control signals that indicate when a board is ready to send or receive data. This is akin to the wait state capabilities of the ISA bus.

Because of its high-frequency operation, the PCI standard limits the number of add-in board connectors on a single bus to four. However, bridges can be used to implement multiple PCI buses on a single motherboard, allowing for larger numbers of expansion slots. This is commonly used in industrial PCs.

The PCI standard supports both 5 V and 3.3 V logic levels. Three types of boards are defined: 3.3 V only, 5 V only, and universal. Expansion board connectors are keyed to prevent inserting a 3.3 V board into a 5 V socket or vice versa.

PCI expansion boards are similar in size to their ISA counterparts, available as either full-length or short-length cards. They use the same style of connectors that IBM employed in its Micro Channel PCs. These connectors have twice the pin density of ISA connectors and accommodate their larger pin count (124 pins for 32-bit connectors) in a smaller space.

5.5.2 PCI Operations

The PCI bus multiplexes its address and data signals on the same pins (AD[00]–AD[31]). A control signal, FRAME# (cycle frame) indicates when a transfer cycle starts. It remains valid throughout most of the data cycle. During the first phase of a transfer cycle, the AD lines contain address information. For later phases, the AD lines contain data values. Figure 5-5 shows a basic PCI read operation.

Control lines C/BE[0:3]# (command/byte enables) indicate which bytes are active during the data cycle, allowing 8- to 32-bit data transfers (for a 32-bit PCI bus). The IRDY# (initiator ready) signal indicates that the bus master is ready to complete the transaction. During a read cycle this means that the master is ready to accept data and during a write cycle it indicates that valid data is present on the bus (AD[00:31]). The TRDY# (target ready) signal indicates that the selected (addressed) device is able to complete the transfer. A data phase is complete when both IRDY# and TRDY# are asserted. Wait states are inserted when IRDY# and TRDY# are not both active. The STOP# (stop) signal is used by the current target device to abort the current transfer. The DEVSEL# (device select) signal indicates that the device selected to

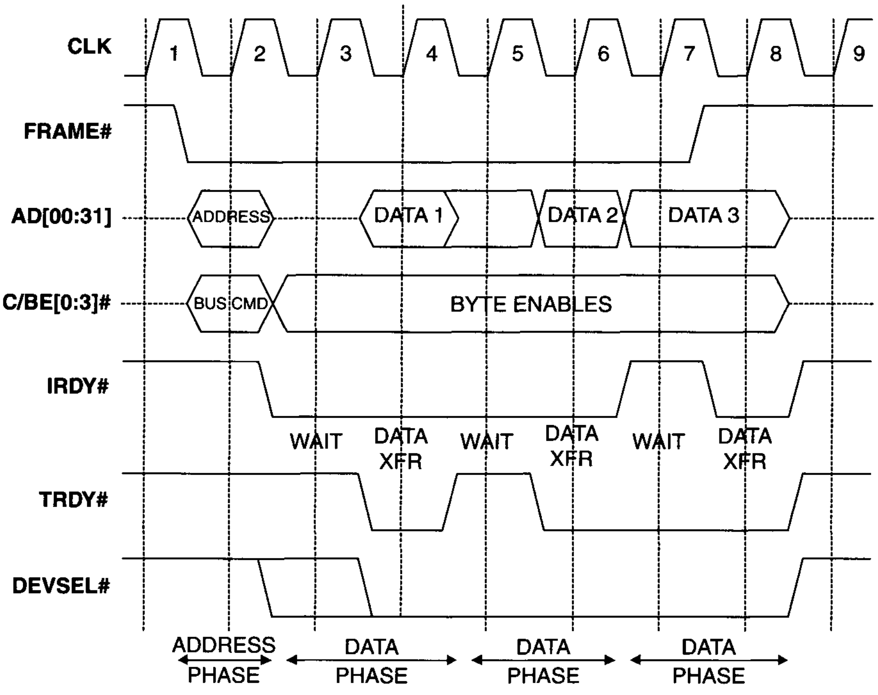


Figure 5-5 Basic PCI read operation.

drive the bus (write data) has decoded its address and knows it has been selected. Since most of these control signals are bidirectional and tri-stated, a PCI bus data transfer uses a fairly complex protocol.

One way the PCI bus improves data throughput is via a burst mode. Here, a single address cycle is followed by multiple data transfer cycles. This allows for an instantaneous speed of 132 Mbytes/sec for a 32-bit PCI bus running at 33 MHz. Of course, the maximum average or sustained data transfer rate will be slower than this (speeds up to 100 Mbytes/sec are commonly attained). If a large amount of data is transferred during a single burst, it ensures a high data rate, since the overhead of the address cycle becomes minimal.

To ensure data integrity on the bus, PCI employs three signals: PAR (parity), PERR# (parity error), and SERR# (system error). PAR is the even parity bit, derived from the 32 AD lines and the four C/BE# lines. The sum of those bits and PAR should be an even number. If a parity error is detected during a standard cycle, PERR# is asserted. For a special cycle, SERR# is asserted.

A PCI add-in card can either be a slave or a bus master. The bus master capability is implemented via the REQ# (request) and GNT# (grant) signals. When a bus master board wants to take control of the bus, it asserts REQ#. The motherboard asserts GNT# when it is ready to relinquish bus control to the board. Each PCI slot has its own, independent REQ# and GNT# lines.

The bus master feature is important for data acquisition boards, allowing them to take over the bus and quickly transfer large amounts of data into memory when they need to, instead of waiting for the CPU to a acknowledge a request via software.

PCI also support four interrupt lines, INTA#, INTB#, INTC#, and INTD#, which are level-sensitive, active-low, using open-drain drivers which allows signal sharing among multiple boards.

Table 5-4 shows the pinouts for 32-bit PCI expansion cards—both 5 V and 3.3 V boards.

5.5.3 64-Bit PCI Bus

PCI supports a 64-bit standard as an extension to the basic 32-bit bus. This is an additional 32-bit bus that uses 39 new signal pins: AD[32:64], C/BE[4:7]#, REQ64#, ACK64#, and PAR64. The new control lines are only valid for this additional bus. REQ64# (request 64-bit transfer) and ACK64# (acknowledge 64-bit transfer) are used to request and enable a 64-bit data transfer cycle. C/BE[4:7]# (control/byte enables) lines are used to control which bytes of AD[32:64] contain valid data. PAR64 (parity upper) is the parity bit for AD[32:64] and C/BE[4:7], behaving the same way as PAR does for the lower 32-bit bus.

Table 5-5 show the pinouts for the 64-bit extension on 5 V and 3.3 V PCI boards.

5.5.4 PCI-X

As with the rest of the PC industry, the PCI standard continues to evolve into faster versions and special applications. PCI-X is a high-performance extension to the PCI bus that doubles the maximum clock frequency to 133 MHz while still allowing 64-bit transfers. This produces a maximum burst transfer rate of over 1 Gbyte/sec while preserving backward compatibility with standard PCI devices. PCI-X also includes protocol enhancements that make bus operations more efficient. PCI-X motherboards may only support keying for 3.3 V cards, although the specification does describe universal (5 V or 3.3 V) cards.

TABLE 5-4
32-bit PCI Expansion Card Pinout

	5 V CARD		3.3 V CARD	
PIN #	SIDE B	SIDE A	SIDE B	SIDE A
1	−12 V	TRST#	−12 V	TRST#
2	TCK	+12 V	TCK	+12 V
3	Ground	TMS	Ground	TMS
4	TDO	TDI	TDO	TDI
5	+5 V	+5 V	+5 V	+5 V
6	+5 V	INTA#	+5 V	INTA#
7	INTB#	INTC#	INTB#	INTC#
8	INTD#	+5 V	INTD#	+5 V
9	PRSNT1#	Reserved	PRSNT1#	Reserved
10	Reserved	+5 V	Reserved	+3.3 V
11	PRSNT2#	Reserved	PRSNT2#	Reserved
12	Ground	Ground	KEYWAY	
13	Ground	Ground		
14	Reserved	3.3Vaux	Reserved	3.3Vaux
15	Ground	RST#	Ground	RST#
16	CLK	+5 V	CLK	+3.3 V
17	Ground	GNT#	Ground	GNT#
18	REQ#	Ground	REQ#	Ground
19	+5 V	PME#	+3.3 V	PME#
20	AD[31]	AD[30]	AD[31]	AD[30]
21	AD[29]	+3.3 V	AD[29]	+3.3 V
22	Ground	AD[28]	Ground	AD[28]
23	AD[27]	AD[26]	AD[27]	AD[26]
24	AD[25]	Ground	AD[25]	Ground
25	+3.3 V	AD[24]	+3.3 V	AD[24]
26	C/BE[3]#	IDSEL	C/BE[3]#	IDSEL
27	AD[23]	+3.3 V	AD[23]	+3.3 V
28	Ground	AD[22]	Ground	AD[22]
29	AD[21]	AD[20]	AD[21]	AD[20]
30	AD[19]	Ground	AD[19]	Ground
31	+3.3 V	AD[18]	+3.3 V	AD[18]

TABLE 5-4
32-bit PCI Expansion Card Pinout (*Continued*)

	5 V CARD		3.3 V CARD	
PIN #	**SIDE B**	**SIDE A**	**SIDE B**	**SIDE A**
32	AD[17]	AD[16]	AD[17]	AD[16]
33	C/BE[2]#	+3.3 V	C/BE[2]#	+3.3 V
34	Ground	FRAME#	Ground	FRAME#
35	IRDY#	Ground	IRDY#	Ground
36	+3.3 V	TRDY#	+3.3 V	TRDY#
37	DEVSEL#	Ground	DEVSEL#	Ground
38	Ground	STOP#	Ground	STOP#
39	LOCK#	+3.3 V	LOCK#	+3.3 V
40	PERR#	Reserved	PERR#	Reserved
41	+3.3 V	Reserved	+3.3 V	Reserved
42	SERR#	Ground	SERR#	Ground
43	+3.3 V	PAR	+3.3 V	PAR
44	C/BE[1]#	AD[15]	C/BE[1]#	AD[15]
45	AD[14]	+3.3 V	AD[14]	+3.3 V
46	Ground	AD[13]	Ground	AD[13]
47	AD[12]	AD[11]	AD[12]	AD[11]
48	AD[10]	Ground	AD[10]	Ground
49	Ground	AD[09]	M66EN	AD[09]
50	**KEYWAY**		Ground	Ground
51			Ground	Ground
52	AD[08]	C/BE[0]#	AD[08]	C/BE[0]#
53	AD[07]	+3.3 V	AD[07]	+3.3 V
54	+3.3 V	AD[06]	+3.3 V	AD[06]
55	AD[05]	AD[04]	AD[05]	AD[04]
56	AD[03]	Ground	AD[03]	Ground
57	Ground	AD[02]	Ground	AD[02]
58	AD[01]	AD[00]	AD[01]	AD[00]
59	+5 V	+5 V	+3.3 V	+3.3 V
60	ACK64#	REQ64#	ACK64#	REQ64#
61	+5 V	+5 V	+5 V	+5 V
62	+5 V	+5 V	+5 V	+5 V

TABLE 5-5
PCI 64-Bit Extension Pinout

PIN #	5 V CARD		3.3 V CARD	
	SIDE B	SIDE A	SIDE B	SIDE A
63	Reserved	Ground	Reserved	Ground
64	Ground	C/BE[7]#	Ground	C/BE[7]#
65	C/BE[6]#	C/BE[5]#	C/BE[6]#	C/BE[5]#
66	C/BE[4]#	+5 V	C/BE[4]#	+3.3 V
67	Ground	PAR64	Ground	PAR64
68	AD[63]	AD[62]	AD[63]	AD[62]
69	AD[61]	Ground	AD[61]	Ground
70	+5 V	AD[60]	+3.3 V	AD[60]
71	AD[59]	AD[58]	AD[59]	AD[58]
72	AD[57]	Ground	AD[57]	Ground
73	Ground	AD[56]	Ground	AD[56]
74	AD[55]	AD[54]	AD[55]	AD[54]
75	AD[53]	+5 V	AD[53]	+3.3 V
76	Ground	AD[52]	Ground	AD[52]
77	AD[51]	AD[50]	AD[51]	AD[50]
78	AD[49]	Ground	AD[49]	Ground
79	+5 V	AD[48]	+3.3 V	AD[48]
80	AD[47]	AD[46]	AD[47]	AD[46]
81	AD[45]	Ground	AD[45]	Ground
82	Ground	AD[44]	Ground	AD[44]
83	AD[43]	AD[42]	AD[43]	AD[42]
84	AD[41]	+5 V	AD[41]	+3.3 V
85	Ground	AD[40]	Ground	AD[40]
86	AD[39]	AD[38]	AD[39]	AD[38]
87	AD[37]	Ground	AD[37]	Ground
88	+5 V	AD[36]	+3.3 V	AD[36]
89	AD[35]	AD[34]	AD[35]	AD[34]
90	AD[33]	Ground	AD[33]	Ground
91	Ground	AD[32]	Ground	AD[32]
92	Reserved	Reserved	Reserved	Reserved
93	Reserved	Ground	Reserved	Ground
94	Ground	Reserved	Ground	Reserved

5.6 PC Peripherals

Nearly all PC systems use at least one floppy drive (a notable exception being diskless LAN workstations) and a hard-disk drive. It is strongly recommended that a PC-based data acquisition platform have at least a 10-Gbyte or larger hard drive, for storage of raw and analyzed data as well as room for typically large application software.

Older PCs usually had at least one parallel printer port and a serial port, for asynchronous communications. Newer PCs have one or more USB ports for external peripherals. See Chapter 8 for a discussion of parallel, serial, and USB interfaces.

Several standard video displays have been available for PCs. The most basic was the text-only monochrome display, employing IBM's monochrome display adapter (MDA), used on older PC/XT/AT machines. It offered 1 page of 25 lines of 80 characters with hardware support for high-intensity, underlining, and reverse video. It supported simple character-based graphics, where special characters are graphic symbols (such as lines) instead of alphanumerics. The MDA had a video buffer (memory) 4 Kbytes long. It produced sharp, easy-to-read text.

True bit-mapped color graphics were supported by the color graphics adapter (CGA). It provided 4 pages of 80-character by 25-line text, as well as several graphics modes. Its highest graphics resolution was 640 points horizontally by 200 points vertically in 2 colors. It also supported 4 colors with a resolution of 320 points horizontally by 200 points vertically. The CGA had a 16-Kbyte video buffer. Text on a CGA monitor was much "fuzzier" than on an MDA monitor. The original IBM PC only offered MDA and CGA display options. These displays are obsolete now.

The next available IBM video display was the enhanced graphics adapter (EGA). Its video buffer size varied from 64 to 256 Kbytes and it supported multiple pages of text. It displayed graphics with a resolution of 640 points horizontally by 350 points vertically, with up to 16 colors (with maximum buffer memory). It also emulated a CGA or MDA display.

Some of IBM's PS/2 series of computers supported multicolor graphics array (MCGA), which was an enhanced version of CGA. It used 64 Kbytes of video buffer memory and stored up to 8 pages of monochrome text. For graphics, it supported all the CGA modes as well as adding support for 256 colors in a 320 points by 200 points mode. In addition, it had a high-resolution 2-color graphics mode with 640 points horizontally by 480 points vertically.

The newer IBM video display for PCs is the virtual graphics array (VGA) family. VGA started on many IBM PS/2 systems and older ISA systems. It has a 256-Kbyte video buffer. It emulates MDA, CGA, EGA, and

MCGA modes. It can support a 640-point by 480-point high-resolution graphics display with 16 colors. VGA has become the most popular PC display standard, especially with higher resolution versions, collectively referred to as super VGA or SVGA. These displays are defined under VESA standards, having resolutions up to 1600 points by 1200 points using up to 16 million colors. Nearly all current PCs support VGA displays.

There is also one early, non-IBM video display standard, the Hercules graphics adapter (HGA), sometimes referred to as monochrome graphics. It was developed to fill the void between the original text-only MDA and color graphics CGA, as a graphics display using a monochrome monitor. It emulated MDA (and used the same monitor) in text mode, along with MDA graphics characters. It could switch into a monochrome (two-color), *bit-mapped* graphics mode supporting a resolution of 720 points horizontally by 348 points vertically. Its video buffer contained 64 Kbytes of memory. Being a non-IBM standard, it was not supported by BIOS or DOS video functions. A special software driver had to be installed to fully use it. However, many early commercial software products supported HGA and it was a low-cost alternative to high-resolution color displays (EGA and VGA) when multicolor video was not required. Today, the VESA standards have made HGA and most other nonstandard PC displays obsolete. There are even monochrome VGA monitors commercially available.

Most video cards contain their own BIOS, which is loaded when the PC boots up. Currently, display adapter cards fall into three groups: SVGA, 2-D graphics accelerators, and 3-D graphics accelerators. Some of these cards plug into an accelerated graphics port (AGP) slot on the PC's motherboard. This is a special local bus, just for connecting a video adapter to the CPU.

The keyboard is the PC's standard user-input device, fully supported by BIOS, DOS, and Windows functions. There are many other user input and control devices for PCs, the most popular being the mouse. The mouse is a device that connects to the PC via a standard serial port, a special mouse connector (the IBM PS/2 mouse standard), or a USB port. It is moved by the user's hand in a two-dimensional plane on an ordinary tabletop or a special pad. It has two or more buttons the user can push (some also have a scroll wheel). In conjunction with supporting software, a mouse simplifies using graphics-based applications, such as CAD systems or operating systems such as Windows (see Chapter 7). For example, a painting program allows the user to create and edit graphics images. A mouse can be used, among other things, to draw lines, select functions, and select objects on the screen to manipulate. Other, less common peripherals for user input are digitizing pads and trackballs (a stationary version of a mouse, either built into a keyboard or freestanding). Many newer PCs use USB ports to connect a mouse and keyboard to a PC.

An important and sometimes overwhelming area of PC peripherals is that of mass storage. This includes floppy drives (diskettes), hard drives, optical drives, and other, more esoteric storage devices. For floppy drives, there were two common form-factors, 5-1/4 inch and 3-1/2 inch diskettes, each with two standard densities. The early 5-1/4 inch drive supported double-sided double-density storage, which allowed 360 Kbytes of formatted capacity. This was common on XT class machines. Most AT machines used a double-sided high-density drive that was capable of 1.2 Mbytes of formatted storage. Similarly, both 3-1/2 inch drive formats are double sided. The original double-density 3-1/2 inch drive had a formatted capacity of 720 Kbytes. The standard quad-density 3-1/2 inch drive has a 1.44 Mbyte capacity. Most newer PCs only have a 1.44-Mbyte drive, even though 2.88-Mbyte capacity 3-1/2 inch drives are available.

There are some wrinkles to note when using diskettes with different density drives. Most notably, if a diskette was formatted on a double-density 5-1/4 inch drive, it can be read by a high-density drive, but a high-density diskette cannot be read by a double-density drive. If a double-density diskette was written on by a high-density drive, sometimes it may not be read reliably by a double-density drive. Also, for both 5-1/4 and 3-1/2 inch drives, the diskettes used must be the appropriate type for that drive. So, do not use low-density diskettes in high-density drives or vice versa. In 3-1/2 inch drives, the hardware recognizes whether the diskette is low or high density via a permanent notch in the diskette.

The hard drive arena can be even more confusing. Hard disk drives can vary in capacity from megabytes (Mbytes) to gigabytes (Gbytes). The common sizes keep increasing each year as storage technology improves. Early hard drives used MFM (modified frequency modulation) encoding. Some used RLL (run length limited) encoding to increase capacity and transfer speed by 50%, over MFM. Advanced RLL drives doubled the data density over MFM. An important measure of performance is a drive's average access time, ranging from around 60 msec with older drives to less than 10 msec on newer models.

The type of drive-to-computer interface is another important hard disk parameter. Early PCs used the serial ST506/412 interface with its peak data transfer rate of only 625 Kbytes/sec (its serial data rate was 5 MHz). An improvement to this standard was the enhanced small device interface (ESDI), which also used a serial data stream but ran at 25 MHz, resulting in a peak data rate of 3.125 Mbytes/sec.

These standards were made obsolete by the integrated drive electronics (IDE) interface (sometimes called the AT attachment) and its many variations. As the name implies, an IDE drive has control electronics built into it. So, a PC's motherboard requires a very simple interface to connect to an IDE drive, and the need for a separate controller card (as with ST506/412 or ESDI) is eliminated. IDE drives were originally developed for AT computers.

IDE is a parallel interface, 16 bits wide. Its original peak transfer rate was 4 Mbytes/sec. Later improvements, such as ATA-2 or enhanced IDE (EIDE), increased the peak data transfer speed to 16 Mbytes/sec. ATA-4 or Ultra DMA raised this rate to 33 Mbytes/sec. The newest IDE standard (as of this writing), ATA-5 or Ultra DMA/66, has a peak rate of 66 Mbytes/sec. The biggest advantage of ATA/IDE drives is their fairly low price at a good performance level.

If you need a higher performance hard drive system (i.e., faster transfer rates than IDE drives) the best alternative is the small computer system interface (SCSI). SCSI is a self-contained bus that can connect up to 15 devices to a PC, using an interface card (some PCs, mostly network servers, have SCSI controllers built into the motherboard). SCSI supports other devices besides hard drives, such as high-performance CD-ROM drives and scanners. SCSI was originally an 8-bit wide bus with a peak transfer rate of 5 Mbytes/sec. Later versions increased bus width to 16 bits and raised the speed. Currently (as of this writing) the fastest SCSI standard is the Ultra 160/m Wide SCSI with a 16-bit bus, 80-MHz speed, and a peak data rate of 160 Mbytes/sec.

The performance of SCSI hard drive systems depends on many factors, including the length of the signal cables and the properties of the controller card. When streaming large amounts of data to a hard drive at high rates, as is common in some data acquisition applications, a high-performance disk drive is necessary. Just bear in mind that peak data rates are only one indication of overall throughput. Appropriate software must be used to obtain the full benefits of fast hardware.

Another important class of mass storage devices are tape drives, typically used to back up data from hard drives. As PCs progress to larger hard drives, backing up data onto diskettes becomes cumbersome and often impractical. For example, a PC with a small 100-Mbyte hard drive requires 70 high-density 3-1/2 inch diskettes (1.44 Mbytes each) for a total backup. Even using data compression techniques, about 30 diskettes would be required. Instead, a tape drive using a single tape cartridge can easily store hundreds of megabytes or even several gigabytes. Tape drives have shown a trend toward standardization, making their use more attractive for backing up large hard drives.

The quarter-inch cartridge (QIC) standard encompasses a range of tapes that can store as little as 60 Mbytes or as much as 25 Gbytes on a single tape cartridge. Drives that use QIC tapes are fairly common. A newer tape cartridge standard, the Travan, ranges from 400 Mbytes up to 4 Gbytes on a tape. There are digital audio tapes (DAT) used for data backups with capacities up to 20 Gbytes. Another format, digital linear tape (DLT), can also store up to 20 Gbytes on a tape. Nearly all tape drive systems support data compression, which can sometimes double the capacity of a cartridge (although it may slow down the backup process).

Optical drives are a fast-growing alternative to some magnetic media. The CD-ROM (compact disc–read only memory) drive has become ubiquitous as a means of distributing programs and data for PCs. These compact discs are prerecorded digital media (as are audio CDs) containing up to 700 Mbytes on a standard disc. A CD-ROM is, as the name implies, read-only.

CD-R (compact disc–recordable) drives allow you to record data on a blank disc. Once the disc is full you cannot write any more data onto it. However, it is possible using appropriate software to write multiple data "sessions" onto a CD-R disc. CD-R is ideal as a backup medium since the data cannot be erased and the discs are readable on nearly any CD-ROM drive. It has similar capacities to CD-ROMs.

CD-RW (CD–rewritable) drives allow you to erase data on an optical disc and record new data over it, just like conventional magnetic media (floppy and hard drives). CD-RW drives also function as CD-R drives, using the appropriate blank media. CD-RW drives have become very popular in recent years as their price has fallen. However, not all CD-RW discs can be read in CD-ROM or even CD-R drives.

The newest optical storage technology is DVD (digital video disc or digital versatile disc), originally developed for storing video data. Currently, DVD media store about 4 Gbytes on a disc, although standards are defined for up to 16-Gbyte discs. The DVD-ROM drive is analogous to the CD-ROM. It is a read-only medium used in PCs for software and video distribution. There are also recordable DVD drive formats, DVD-R and DVD-RW, which are initially too expensive for widespread use (but should eventually become as common as CD-R and CD-RW). DVD drives can also read CDs but not all CD-R and CD-RW discs. Because of their much larger capacity, it is very likely that DVD drives will eventually supplant CD drives.

Another popular realm of PC mass storage is the high-capacity floppy disk. Standard floppy drive capacities are now much too small for data and software requirements. One early attempt to significantly increase floppy disk capacity was the "Floptical" disk drive, which used an optical track servo system to provide 20 Mbytes of storage on a floppy-sized disk. This approach is used in the popular ZIP drive, which comes in 100-Mbyte and 250-Mbyte versions, using a special cartridge that is larger than a standard 3-1/2 inch diskette. Super Disk or LS-120 drives store 120 Mbytes and are backward compatible with 1.44 Mbyte floppy disks.

One final class of PC peripherals we will touch on here is that of printers and plotters. Most PC printers use either a parallel (Centronics) port or a

USB port. Nearly all plotters use a serial port or a network connection (usually Ethernet). A printer is used to produce text and graphics output. The majority of printers used are ink-jet based, dot-matrix devices, forming characters and graphics images out of small, individual dots. Even laser printers use individual dots, albeit at very high densities (300 to 600 dots per inch or more).

Plotters are devices that produce drawings from a set of lines. They use one or more pens, whose position on the paper is accurately controlled. Plotters are commonly used by CAD and graphic art software. Newer plotter also use inkjet technology, instead of pens, for increased speed.

This completes our brief overview of standard PCs. In the next chapter we will look at the details of connecting external hardware to a PC's I/O expansion bus.

CHAPTER 6

Interfacing Hardware to a PC Bus

We will now look at the details of connecting external hardware to an XT, AT, or PCI bus. Initially we will examine 8-bit data transfers on a PC/XT bus. Later we will see the differences when connecting 16-bit devices to an AT (ISA) bus. We will also look at the issues involved with interfacing to the PCI bus.

As we touched on in the previous chapter, three types of XT/AT bus cycles are used for data transfers: memory, I/O port, and direct memory access (DMA) cycles. On the PCI bus there are also burst transfers and special access cycles. For the XT/AT bus, these can be either a read cycle where data is transferred from an external device or memory into the CPU (or bus controller, when it is a DMA operation) or a write cycle where data is transferred from the CPU (or bus controller) to an external device or memory. Memory cycles are used to access system memory and memory on expansion cards (such as video buffers). Most data transfers to external devices use I/O port cycles or DMA cycles.

6.1 I/O Data Transfers

In XT systems, I/O port addresses in the range 200h–3FFh are available for use by I/O cards. Many of the I/O port addresses are reserved for particular functions. For example, the range 320h–32Fh is used by hard disk drive adapter cards (or the equivalent controller on a motherboard). One popular I/O address range for undefined functions is 300h–31Fh, assigned to IBM's prototype card.

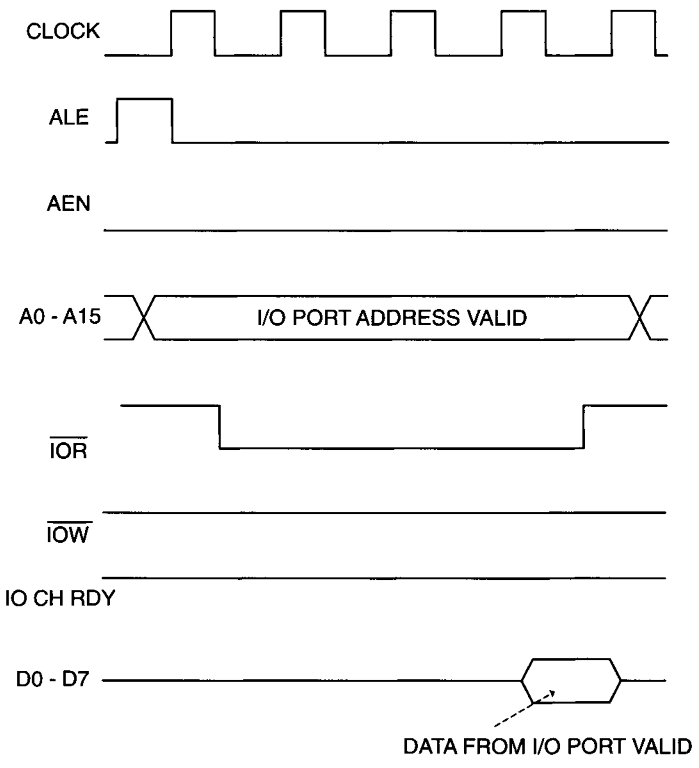


Figure 6-1 8088 CPU I/O port bus read cycle.

Only a few control signals are needed, along with the address and data buses, to implement an I/O port read or write cycle on the XT bus. These are IOR (for a read cycle), IOW (for a write cycle), and AEN (to distinguish between an I/O port cycle and a DMA cycle). The timing for an I/O port read cycle is shown in Figure 6-1.

A standard PC/XT I/O port bus cycle requires five clock cycles, including one wait state injected by logic on the motherboard. Many systems with high clock frequencies inject additional wait states so that I/O cards designed for slower systems will still operate properly. The ALE signal occurs at the beginning of the I/O port cycle and indicates when the address bus contents are valid for the addressed port. IOR or IOW go active low to indicate an I/O port cycle. AEN stays inactive (low) to indicate this is not a DMA cycle. An active IOR signal tells the addressed I/O port to place its data (for the CPU to read) on the data bus (D0–D7). An active IOW signal tells the addressed

I/O port to read the contents of the data bus (from the CPU). The control line I/O CH RDY is normally left active (high). If a slow I/O port needs additional wait states inserted into the cycle, it pulls this line low.

6.2 Memory Data Transfers

Memory bus cycles use timing very similar to I/O port bus cycles, as shown by the memory read cycle in Figure 6-2. The main control lines here are MEMR and MEMW. AEN is not needed for memory bus cycle decoding. One difference from I/O addressing is that for memory bus cycles, the motherboard does not inject an additional wait state (hence, only four clock cycles are needed instead of five). Another difference is that all 20 address lines (A0–A19) are valid for a memory bus cycle and should be used for decoding the memory address. Only the first 16 lines (A0–A15) are valid for an I/O bus cycle; in practice, just the first 10 address lines (A0–A9) are decoded on a PC/XT bus.

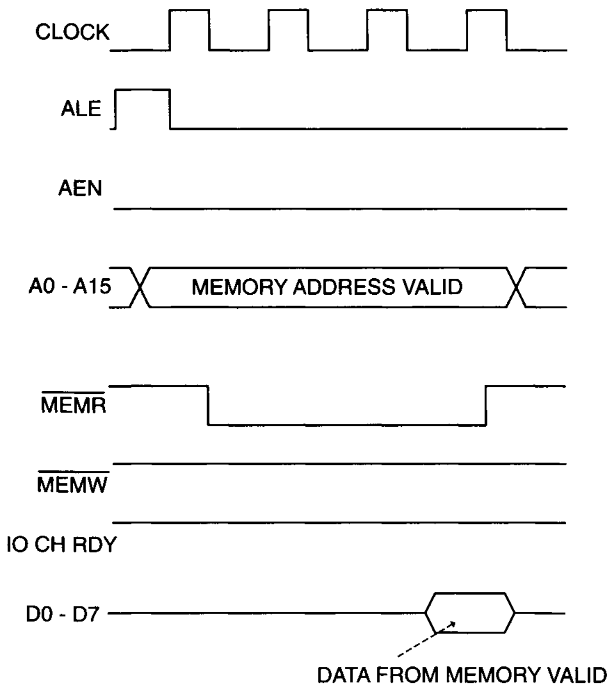


Figure 6-2 8088 CPU memory bus read cycle.

6.3 A Simple, 8-Bit I/O Port Design

A simple, fixed-address, 8-bit I/O port schematic is shown in Figure 6-3. The port I/O address is fixed at 300h by the decoding logic used on inputs A0–A9. IOW is used to write data to the output port latch (74LS373). IOR is used to read data at the input port buffer (74LS244). Note that the decode and control logic can be handled by a single PLD (programmable logic device) having at least 13 inputs and 2 outputs. A PLD is a logic device (such as a PAL or GAL) which contains an array of internal logic gates and flip-flops. The programming of the PLD determines the interconnection of its resources and the overall logic functions it performs (such as address decoding). A more

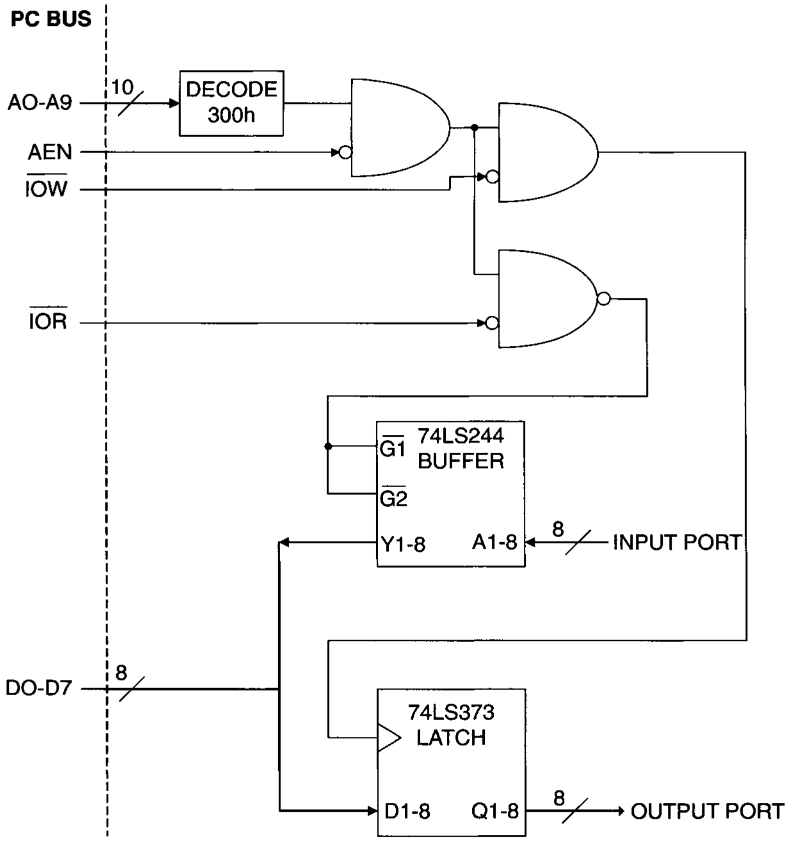


Figure 6-3 Simple 8-bit PC/XT digital I/O port.

versatile I/O port circuit would have a selectable I/O port address, determined by jumper or switch settings.

Whenever the CPU writes to I/O address 300h, a data byte appears at the output port. When the CPU reads from that address, it retrieves the byte currently present at the input port. This is simple, programmed I/O that must be completely handled by the CPU. The CPU's program must determine when it is time for an I/O data transfer and must control the I/O read or write cycle as well as store or retrieve the data from memory. This limits the maximum data transfer rate and prevents the CPU from doing other tasks while it is waiting for another I/O cycle.

6.3.1 Using Hardware Interrupts

Usually, a better alternative to the polled I/O technique just described is to use hardware interrupts. The occurrence of a hardware interrupt causes the CPU to stop its current program execution and go to a special interrupt service routine, previously installed. This is designed to handle asynchronous external events without tying up the CPU's time in polling for the event. Nine hardware interrupts are used in a PC/XT system. The highest priority is the NMI (nonmaskable interrupt), which cannot be internally masked by the CPU (but can be masked by hardware on the motherboard). This line is usually used to report memory errors and is not available to cards connected to the I/O expansion slots. The other eight hardware interrupt lines, IRQ0–IRQ7, are connected to an Intel 8259 Interrupt Controller (which connects to the 8088's maskable interrupt input line). The highest priority lines, IRQ0 and IRQ1, are used on the motherboard only and are not connected to the I/O slots. IRQ0 is used by channel 0 of the timer/counter, and IRQ1 is used by the keyboard adapter circuit. Interrupts IRQ2–IRQ7 are available to I/O cards.

The 8088 CPU supports 256 unique interrupt types. These can be hardware or software interrupts. Each interrupt type has assigned to it a 4-byte block in low memory (0–3FFh) containing the starting address of that interrupt's service routine. This interrupt vector consists of the 16-bit code segment (CS) and instruction pointer (IP) of the service routine. Interrupt types 0–4 are used by the 8088 CPU. For example, interrupt type 0 is called by a divided-by-zero error. Interrupt types 5 and 6 are unused for 8088-based PCs. Interrupt type 7 is used by the BIOS for the Print Screen function.

Hardware interrupts IRQ0–7 are mapped to types 8–15. So, the vector for IRQ0 is at addresses 20h–23h, IRQ1 is at 24h–27h, and so on. A hardware interrupt is asserted when the appropriate IRQ line goes high and stays high

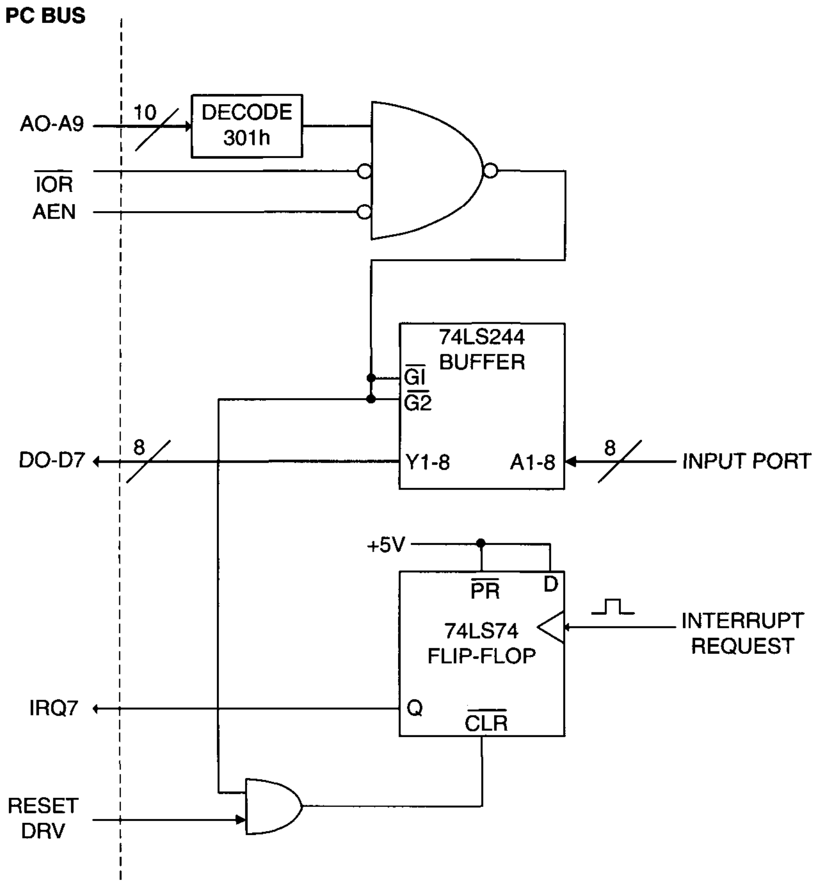


Figure 6-4 Interrupt-driven 8-bit PC/XT digital input port.

until the interrupt is acknowledged. There is no direct interrupt acknowledge line from the I/O bus (it occurs between the CPU and the 8259 Interrupt Controller), so an I/O line under CPU control is used for this function and activated by the interrupt service routine.

Figure 6-4 shows a simple 8-bit input port designed for interrupt-driven access, at I/O address 301h. As in Figure 6-3, the enable line of the input port buffer is decoded by a combination of address bits A0–A9, IOR, and AEN. In addition, the input port provides a Request for Interrupt line, used by the external hardware to signal when it is ready for the CPU to read data from it. A pulse or positive-going edge on this line sets the flip-flop, asserting the IRQ7 line (lowest priority interrupt). When the interrupt service routine for interrupt type 15 is called, it performs a read from I/O address 301h to retrieve the data. This access will also reset the flip-flop, negating the IRQ7 line and preventing an additional (and unwanted) interrupt service cycle after the current one is completed.

Note that IRQ7 is typically used by a parallel printer port. To prevent unwanted hardware clashes, the flip-flop output in Figure 6-4 should be buffered by a tri-state driver, which can be disabled when the input port is not in use. A practical input port design would also have some selectability for the I/O port address and the IRQ line used.

Any interrupt type can be accessed via software by simply using the INT instruction. This includes interrupt types used by IRQ lines. This is a good way of testing hardware interrupt service routines.

6.3.2 Software Considerations for Hardware Interrupts

Implementing hardware interrupt support in software requires many steps. The interrupt service routine must be written and placed at a known memory location. The address of this service routine must be placed in the 4 bytes of low memory corresponding to the appropriate interrupt type (for IRQ7 it would be addresses 3Ch–3Fh). The 8259 Interrupt Controller must be initialized to enable the desired IRQ line. The 8088's maskable interrupt input must be unmasked (if it is not already). If you are using a standard peripheral device supported by BIOS functions, such as an asynchronous communications (serial) port, this initialization will be done for you by the BIOS. Similarly, commercial peripherals that come with their own software drivers should take care of these details for you. If you build your own data acquisition card with interrupt support, you will have to incorporate the initialization procedure into your custom software.

There are conditions where polled I/O is preferable to interrupt-driven I/O. It takes the CPU 61 clock periods to respond to a hardware interrupt and begin executing the interrupt service routine. In addition, it requires 32 more clock cycles to return from an interrupt. For an older PC/XT system with a 4.77-MHz clock, this corresponds to a processing overhead of 19.2 μsec added to the execution time of the interrupt service routine. If high-speed I/O transfers were required, such as every 20 μsec (for a 50,000 sample/sec rate), a tight polling loop would be preferable. There would not be much time left over from servicing the I/O transfer for the CPU to do much else. In general, when the time between consecutive hardware interrupts starts approaching the overhead required to process an interrupt, a polled approach to software is in order.

6.4 DMA

When very high speed data transfers are required between a peripheral device and memory, direct memory access (DMA) hardware is often used. PC/XT systems support four DMA channels via an Intel 8237 DMA controller. The highest priority DMA is on channel 0, used only on the motherboard for DRAM refresh. The other three DMA channels are available for use by peripherals (channel 3 is the lowest priority). During a DMA cycle, the 8237 takes over control of the bus from the 8088 and performs the data transfer between a peripheral and system memory. Even though the 8237 supports a burst mode, where many consecutive DMA cycles can occur, only a single-byte DMA cycle is used on PC/XT systems. This ensures that CPU cycles can still occur while DMA transfers take place, preserving system integrity (including memory refresh operations).

In PC/XT systems, DMA transfers require six clock periods. After each DMA cycle a CPU cycle of four clock periods occurs. So, the maximum DMA transfer rate is 1 byte every 10 clock periods. On original 4.77 MHz PC/XT systems, this is every 2.1 μsec for a maximum DMA data rate of 476 Kbytes per second. This is still much faster than CPU-controlled data transfers.

As with servicing interrupt requests, software must perform initializations before DMA transfers can occur. The 8237 DMA controller must be programmed for the type of DMA cycle, including read or write, number of bytes to transfer, and the starting address. Once it has been properly initialized, the DMA cycle is started by a DMA request from the peripheral hardware.

6.5 Wait State Generation

As we previously discussed, sometimes a peripheral device is too slow for a normal PC/XT bus cycle. The length of a bus cycle can be extended by generating wait states. These are additional clock periods inserted into a memory or I/O bus cycle. Wait states are inserted by pulling line IO CH RDY low (negated) for two or more clock cycles after the data transfer cycle has started.

Figure 6-5 shows a simple circuit for generating one additional wait state for an I/O cycle. When the I/O port is selected (for either a read or write) it sets a flip-flop that pulls IO CH RDY low. Note that the inverter driving the IO CH RDY line is an open-collector device. This is because several

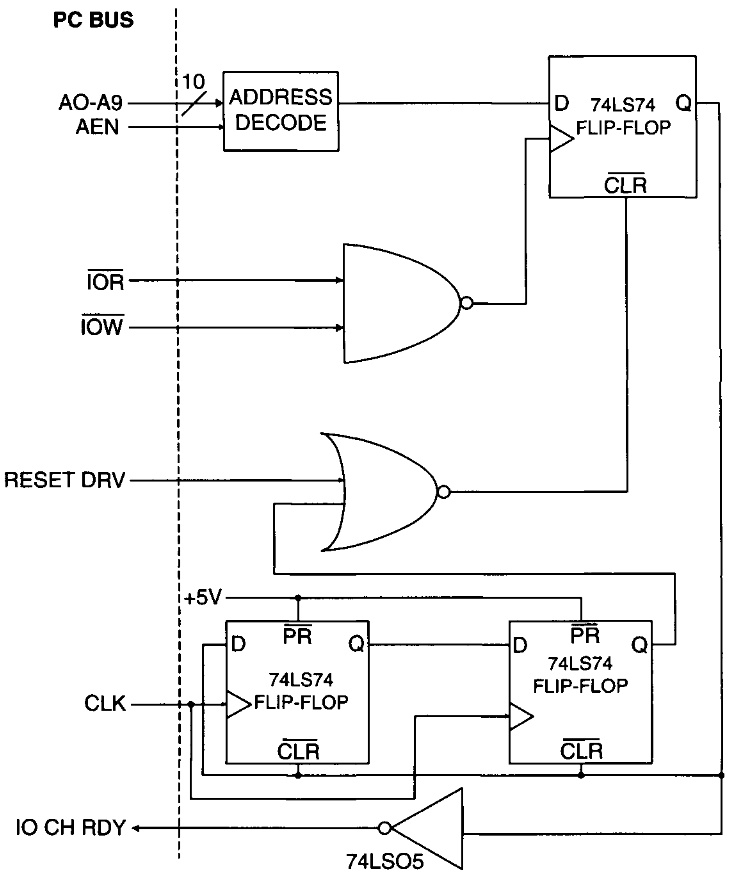


Figure 6-5 I/O wait state generation.

peripherals on the PC/XT bus can drive this line simultaneously and will be OR-tied if they use open-collector outputs. This flip-flop output then goes to a two-stage shift register (using two additional flip-flops), which waits two clock cycles and then outputs a signal resetting the flip-flop and reasserting IO CH RDY, ensuring that no additional wait states are injected into the cycle. For each additional wait state desired, an additional shift register stage should be added, for more clock cycle delays. The timing is very similar for generating memory cycle wait states, except only one clock cycle delay is required to generate the first wait state.

6.6 Analog Input Card Design

Building on what we have discussed in this chapter, Figure 6-6 shows an 8-bit data acquisition circuit with eight analog inputs. It is based on a National Semiconductor ADC0808 successive-approximation ADC with a maximum conversion rate of approximately 10,000 samples per second (100-μsec average conversion time). This device has an eight-channel analog multiplexer. It accepts input signals in the range of 0 to +5 V. If a wider analog input range is required, op amps can be used.

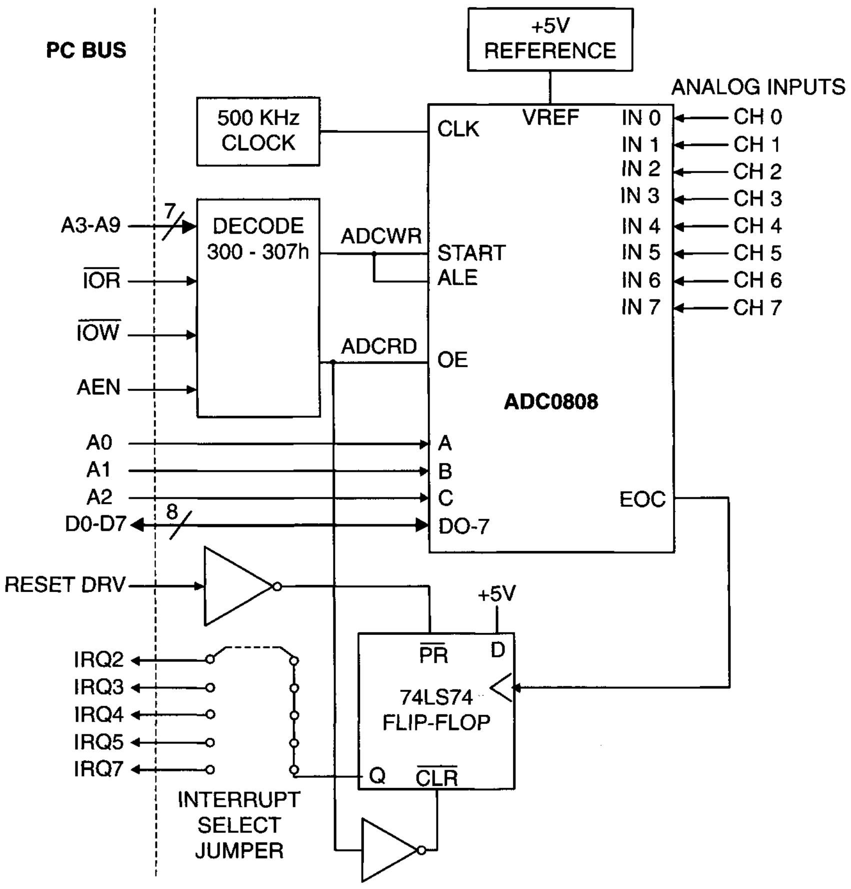


Figure 6-6 8-bit, 8-channel analog input card.

This circuit occupies I/O addresses 300h–307h. Writing dummy data to address 300h starts a conversion for the signal on ADC channel 1. A write to 301h converts channel 2, and so on. When conversion is complete an IRQ is generated (the interrupt line used is jumper-selectable). The interrupt service routine then reads the value from any I/O address in the 300h–307h range. The flip-flop that generates the IRQ is set by the ADC's end-of-conversion (EOC) signal and cleared when the interrupt service routine reads the ADC value.

6.7 16-Bit Data Transfers on ISA Computers

The PC/XT I/O circuits described above will also work in an AT (ISA) system. Most AT computers with high-frequency clocks (above 8 MHz) insert additional wait states for I/O port bus cycles so that cards designed for XT and slower AT systems will still work properly. Even 16-bit transfers to 8-bit peripherals are supported by hardware on the AT motherboard. However, to fully exploit the power of an AT system, an interface card should support 16-bit data transfers wherever possible. This utilizes the additional data, address, and control lines of the AT I/O bus.

Basically, to perform 16-bit I/O port data transfers, we must decode the I/O port address, use IOR or IOW to determine the transfer direction, tell the system bus that we want a 16-bit transfer cycle, and input or output the 16-bit data word. An AT has the same I/O address map for devices connected to the system bus as the PC/XT (in the range 100h–3FFh). This makes I/O address decoding the same. One new control line used on the ISA bus is I/O CS16 (pin D02), which indicates to the CPU (80286 or above) that a 16-bit data transfer is requested by the peripheral device. Another new control line is SBHE (pin C1), which is active when data on the upper byte of the data bus (D8–D15) is valid.

Figure 6-7 shows a simple 16-bit ISA I/O interface, designed for address 300h. The main difference between this circuit and the PC/XT I/O circuits shown previously is the transfer of 16 instead of 8 bits at a time. Otherwise the I/O address decoding is the same, except for the LSB (A0). In addition, the bus signal I/O CS16 is asserted, active low (by an open-collector driver), when the I/O port is accessed for a 16-bit I/O transfer cycle. If this line was not asserted, as with an 8-bit PC/XT card, only the lower 8 data bits (D0–D7) would be used for the I/O cycle. The signal SBHE is used when the upper 8 data bits (D8–D15) are ready for bus transfer, and it enables the buffer for that data. A0 must be asserted to transfer the lower 8 data bits.

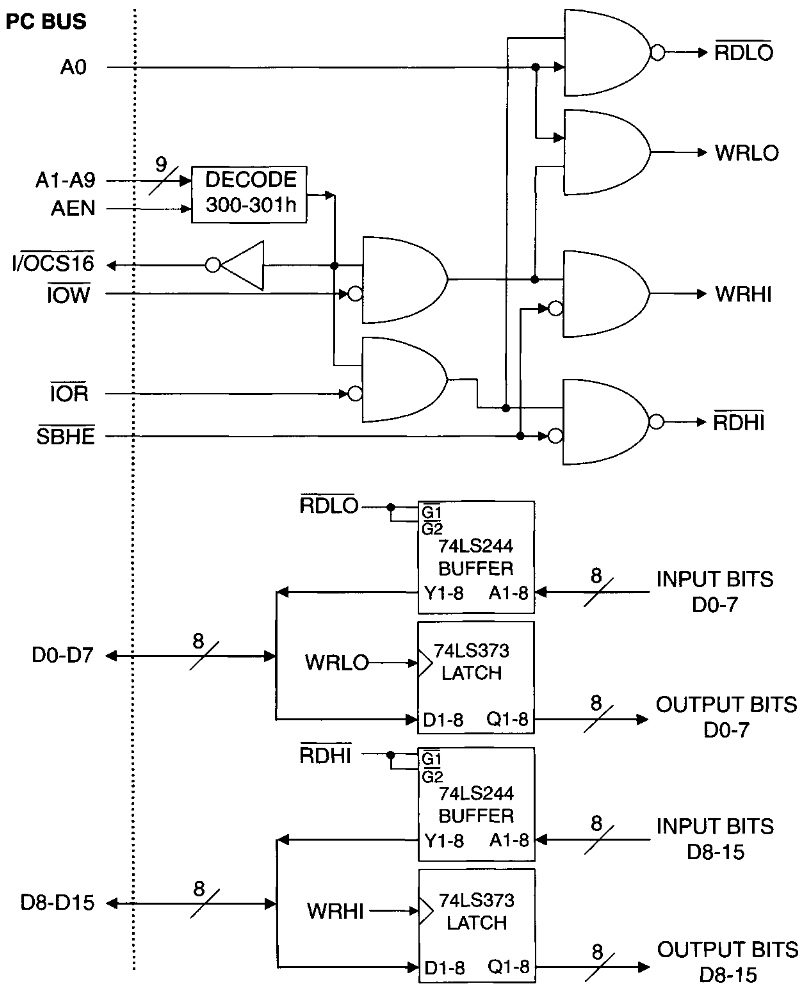


Figure 6-7 Simple 16-bit ISA digital I/O port.

It may be necessary, because of the higher clock frequency of most AT systems (especially 80386- and 80486-based computers), to add additional wait states to an I/O or memory bus cycle over and above the wait states automatically injected by logic on the motherboard. As with PC/XT systems, pulling the IO CH RDY line low can be used to add wait states to a bus cycle.

6.8 Plug and Play

Configuring add-in cards on XT and older AT (ISA) computers was often a time-consuming chore. You had to set jumpers or switches on most boards to select appropriate I/O addresses, memory addresses, IRQ channels, and DMA channels. Aside from some peripherals whose settings were originally determined by IBM (such as some disk drive and video adapters), most boards had no standard settings. If more than one board in a PC tried to use the same resources (i.e., addresses or IRQ lines), they would produce a hardware conflict and not operate properly. This could even prevent the PC from booting up. So, add-in cards had to be manually configured.

To automate the configuration process, Intel and Microsoft developed the Plug and Play specification for the ISA bus. This encompasses a mixture of BIOS software, operating system software, and expansion card hardware. If all these elements are in place, the PC configures the resources a Plug and Play add-in card requires and even loads the appropriate software drivers.

Since the ISA bus was designed without any support for automatic card configuration, Plug and Play relies on a complex process. First it isolates the boards so they do not respond to standard ISA bus control signals. Then each board gets identified and initialized, allowing it to respond to bus signals. Next, each board individually goes into a mode where the PC reads the card's configuration information and programs its resource settings. After all boards have been configured, the operating system loads appropriate software drivers for them.

Most older ISA PCs (pre-Pentium) do not have a Plug and Play compatible BIOS. But as long as the operating system supports it, Plug and Play boards can still be automatically configured. Microsoft operating systems starting with Windows 95 (see Chapter 7) fully support Plug and Play ISA.

Note that with Plug and Play configuration, the resources selected by this process may not be the same as in an older ISA PC using standard settings. For example, plug and play may configure the first serial port (COM1:) to use IRQ7 instead of the older standard of IRQ4.

In contrast to ISA, the PCI bus was designed with autoconfiguration in mind. Each PCI slot (up to four per bus) has a unique input line, IDSEL (initialization device select), which allows the system software to uniquely access the card's 256-byte configuration space. This is a special address space, separate from the conventional I/O and memory spaces on the PCI bus. The configuration space approach is much cleaner and does not require special procedures to isolate add-in cards from each other or the bus. In a PCI-only computer, all expansion boards can be automatically configured.

There is also a Plug and Play specification for parallel ports that are IEEE 1284 compliant (see Chapter 8). If a printer, or other device, supports Plug and Play, the PC can detect it and install the appropriate software driver, simplifying setup of the peripheral.

6.9 Interfacing to the PCI Bus

As we saw in Chapter 5, the PCI bus is several times faster and much more complex than the ISA bus. It would be very difficult to implement even the simplest subset of PCI bus controls using standard TTL-style logic ICs (such as the 7400 series). It would be better to use a large CPLD (complex programmable logic device) to incorporate PCI logic into a custom design.

The simplest way to interface old or new hardware to the PCI bus is through a commercially available controller chip such as those available from AMCC or PLX Technology. ICs such as the PLX PCI9050 or the AMCC S5920 convert PCI signals with their complex protocol into a simple, local bus. This local bus can then interface directly to custom hardware with 8-, 16-, or 32-bit data (for digital I/O ports, ADCs, DACs, etc.) or get converted to ISA bus signals with 8- or 16-bit data, using additional logic.

To quickly convert a simple ISA board to the PCI bus, these chip families have development kits. The kits typically contain a board designed around the conversion chip that has a piggy-back connector for an ISA card. The development kit board plugs into the PCI bus and provides the bus conversion features needed by the ISA card. Since the development kit takes care of most of the hardware issues, the remaining design work is just software conversion, using the tools provided by the kit.

Typically, these conversion chips act only as PCI slaves, without bus mastering capabilities. So they would not be suitable for converting an ISA card that uses DMA. However, more complex chips are available from these manufacturers that support full PCI bus master capabilities.

Figure 6-8 shows a simplified block diagram of an ISA-to-PCI slave interface using the PLX PCI9052 chip. This IC has a built-in ISA interface, so additional logic is not needed. The PCI9052 can also interface non-ISA resources, such as memory and I/O devices, to the PCI bus using its local bus. A serial EEPROM (electrically erasable PROM) is used to store configuration information for the PCI9052.

Many new PCI-based interface cards use CPLD and FPGA (field programmable gate array) logic devices for custom designs. CPLD and FPGA manufacturers, such as Altera, Cypress, Lucent, and Xilinx, offer PCI interface designs that easily incorporate into their chips. The designer simply

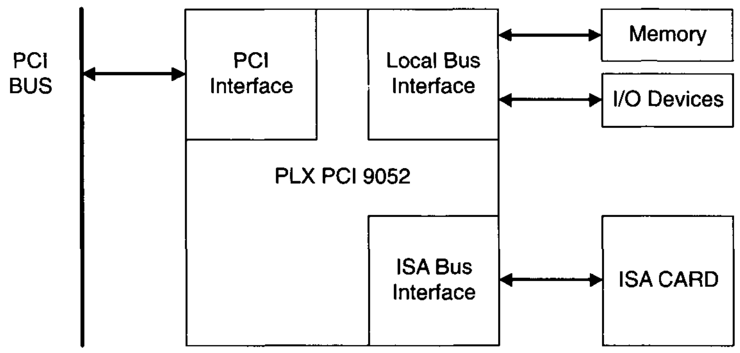


Figure 6-8 Interfacing an ISA card to the PCI bus.

connects the supplied PCI core to the chip's custom logic. This is usually accomplished using a high-level hardware design language, such as VHDL or Verilog. This approach frees the designer from reinventing PCI interface logic while providing greater flexibility than a fixed interface chip allows. It also lowers costs by placing both the PCI interface and board control logic in the same programmable chip.

In the next chapter, we will examine software techniques for interfacing to PCs. The topics covered will include how the PC's software system works and how to produce software to support peripheral hardware, especially for data acquisition applications.

CHAPTER 7

Interfacing Software to the PC

Using the correct techniques for interfacing software to a PC is as important as implementing the proper hardware interface. In this chapter we will start with an overview of the PC/XT/AT DOS-based software structure and proceed to using this arrangement. Then we will explore the Windows environment as well as UNIX.

7.1 DOS-Based PC Software Layers

Four general layers of software are present on a DOS-based PC, as shown in Figure 7-1. The lowest is the hardware level, where the software directly accesses the hardware. For example, if the addressed hardware was a display adapter, writing to a specific address in its video buffer (to display a character) would be directly accessing the hardware. At this level, the actual computer circuitry (I/O and memory addresses) determines the software instructions needed.

The next layer is the basic input–output system, or BIOS. This is software, often referred to as firmware, residing in read-only memory (ROM) on the motherboard. The system ROM includes code to test the computer system and bootstrap (or boot) it, to begin normal DOS or other operating system execution. The BIOS routines in ROM act as an interface between higher level software and the actual hardware. They implement the details needed to operate various standard hardware peripherals (such as video displays or disk drives) and begin to provide some hardware independence. When a program uses a BIOS function, it does not need to know hardware-level details,

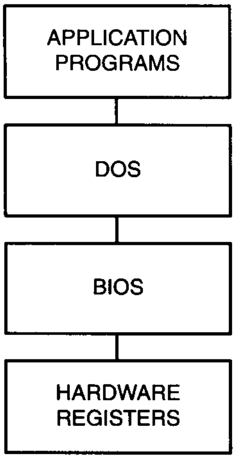


Figure 7-1 PC (MS-DOS) software layers.

such as the address of the status register on a disk drive controller card. It only needs to request the BIOS function it wants completed, such as to read data from a particular sector on a specified disk.

This hardware independence has important advantages. If different computers use different hardware components to carry out the same functions, this approach eliminates the need to rewrite a program for each machine, as long as the BIOS commands are the same. A hardware change in the same machine does not require a software change, as long as the BIOS supports the new hardware or is upgraded with it.

The only disadvantages with this approach are slower program execution and somewhat limited functionality. Since more instructions must be executed to produce a function from a BIOS call, compared to directly addressing hardware, a slower response is produced. Of course, the speed of newer PCs makes this less of an issue and for many functions a slower response is not important (such as the PC response when a user hits a key). When fast execution is required, such as in real-time control or data acquisition, direct hardware addressing may be necessary. If the BIOS functions do not support all the features of a particular hardware device, again direct hardware access may be required. Often, system software is loaded to supplement the BIOS and use the same software interface to call it, as described later.

The next layer of system software is the disk operating system, or DOS. This software is loaded into the PC's memory from a disk drive, by a bootstrap program in ROM. It operates at a higher level than the BIOS, even further removed from the hardware layer. Among other things, it implements the file and directory structure for disk drives. It advances the concept of hardware independence to device independence. For example, when a calling program requests data from a file, under DOS it does not need to know what type of physical drive contains the data. DOS keeps track of that information and retrieves the requested data by appropriate calls to BIOS functions. The program just uses a *logical* drive identification (such as A: or C:).

This device independence extends to the type of device, using the DOS feature of redirection, when it redirects data from one device to another. For example, the DOS TYPE command usually displays the contents of a text file on a video display (for example: TYPE MYDATA.TXT). DOS can redirect this data to a printer, with the command: TYPE MYDATA.TXT > PRN: (which sends this data to the system's default printer). A program calling DOS to perform these functions does not need to know about the differences between the two output devices (video display and printer) or even that very different BIOS calls are used to perform this function. DOS takes care of all these details.

The final, highest layer of PC software is the application program. This is the software that performs the useful functions we need a computer for in the first place, such as mathematical calculations, word processing, data acquisition, and graphical display. To perform these high-level activities, the application program calls various functions at the DOS, BIOS, and hardware levels. As before, for the highest degree of portability, maintainability, and hardware support, software interfacing should be at the highest level possible, preferably DOS, or BIOS if necessary. However, calling system functions through DOS is also the slowest route. As with BIOS calls, trade-offs are sometimes necessary. When running DOS on a fast, relatively new PC (Pentium-based) the slower speed of DOS function calls is minimal.

7.2 Software Interrupts

The mechanism for calling BIOS and DOS functions uses *software interrupts*. This provides a means of software independence for the called functions. A software interrupt works like a hardware-generated interrupt. It causes program execution to jump to a new location, specified by the interrupt number or level. There are 256 possible interrupt levels in 80x86-based PCs. Some are used by hardware interrupts, some by BIOS, and some by DOS. Table 7-1 lists the interrupt usage in a PC/XT system. To generate a software interrupt, the Assembler instruction INT, followed by the level (0–255), is executed.

TABLE 7-1
Interrupt Usage in MS-DOS PCs

INTERRUPT #	CLASSIFICATION	FUNCTION
0–7	BIOS/DOS	CPU Interrupts
8–F	BIOS	8259 H/W Interrupts
10–1C	BIOS	BIOS Function Calls
1D–1F	Data	Video/Disk Table Pointers
20–3F, 5C, 67	DOS	DOS Function Calls
80–F0	BASIC	BASIC Functions

This specifies which interrupt vector to use. An interrupt vector is a 4-byte address in low memory, 0–3FFh, which contains the location of the interrupt service routine. This is the address the program jumps to when the interrupt is called, which contains the code to handle the interrupt request.

The beauty of this system is that the software calling the interrupt routine, such as a BIOS function call, does not have to know exactly where in memory the interrupt service routine is located. This is the software independence alluded to above. If the BIOS code is upgraded at some future point, the absolute location of the interrupt service routine may change, but the software calling it does not have to change, since the interrupt vectors will also be upgraded.

7.2.1 BIOS Interrupts

Using a previous example, the BIOS routine interfacing with the video display works through INT 10h. To display an alphanumeric character on the current video screen, the character byte is loaded into CPU register AL (the low byte of the accumulator) and 14h is loaded into AH (the accumulator's high byte), which specifies the video command (display a character). Then an INT 10h instruction is executed. Written in Assembler, the code to display the character "9" would be

```
MOV  AL,39H
MOV  AH,14H
INT  10H
```

Note that 39H is the ASCII code for the character "9."

As shown in this example, BIOS functions use some of the CPU's registers for sending data to and receiving data from the function called. Sometimes, the carry flag is returned to specify a particular condition. When one

TABLE 7-2
Standard MS-DOS PC BIOS Functions

INTERRUPT #	PURPOSE
10h	Video Display Functions (0–13h)
11h	Equipment Check
12h	Memory Size Check
13h	Floppy Disk Functions (0–18h)
14h	Communications Functions (0–5h)
15h	Cassette and Misc System Functions (0–C4h)
16h	Keyboard Functions (0–12h)
17h	Printer Functions (0–2h)
18h	Execute IBM BASIC from ROM (IBM–PC, Only)
19h	Re–Boot System
1Ah	System Timer/Clock Functions (0–7h)
1Bh	Keyboard CTRL–BREAK Interrupt Handler
1Ch	System Timer Tick (18 Hz) Interrupt Handler

interrupt is used for several different functions (as Int 10h, 13h, 14h, 15h, 16h, 17h, and 1Ah), register AH is loaded with the function number. Table 7-2 is a summary of most of the BIOS functions available on PC/XT/AT systems.

7.2.2 DOS Interrupts

DOS functions are called by software interrupts similar to BIOS functions. Most DOS functions are called via INT 21h. DOS reserves the use of INT 20h–3Fh, although only INT 20h–27h are used for most common functions. Again, the function number is selected by the value placed in register AH. Some DOS INT 21h functions also have a subfunction, selected by the value in register AL.

As an example of using a DOS function, we will once again write a character to the video display, using INT 21h, Function 2. Here, register AH contains the function number (2) and register DL contains the character to be displayed. If we use Microsoft C instead of Assembler in this case, we can write a general-purpose subroutine for video display called disp_ch():

```
#include <dos.h>          /* standard definition files */
#include <stdio.h>
#define FUNCT 2           /* function number 2 */
```

```
disp_ch(ch)                   /* subroutine name */

char ch;                      /* character argument */
{                             /* start of subroutine */
   union REGS regs;           /* sets up register use */
   regs.h.ah = FUNCT;         /* AH = 2 */
   regs.h.dl = ch;            /* DL = character to display */
   intdos(&regs,&regs);       /* call INT 21h */
}
```

A calling program, to display the character "9" would be:

```
main()
{                             /* start of program */
   char c;
   c = 0x39;                  /* ASCII value for 9 */
   disp_ch(c);                /* call subroutine */
}
```

Even though more coding (along with more software overhead) is required to implement this DOS function in C, compared to Assembler, this approach is usually preferable. C is a high-level language with good functionality and ease-of-use. It is much easier to maintain a program in C than in Assembler and the penalty of larger, slower programs is not as severe as with some other high-level programming languages. We will discuss the various trade-offs between different programming languages later in Chapter 13.

7.3 Polled versus Interrupt-Driven Software

In Chapter 6 we looked at the trade-offs between accessing a peripheral device via polled software versus interrupt-driven software. If a peripheral device needs to be serviced relatively infrequently (for example, using only 10% of the available CPU time) and *asynchronously* (so the program cannot predict when the next service will be required), interrupt-driven software is in order. On the other hand, if interrupt servicing takes up too much CPU time (sometimes referred to as CPU *bandwidth*) for very frequent servicing, polled software would be preferable. In this case, there would be little CPU bandwidth left over for other processing anyway. One other general case is when the peripheral servicing is *synchronous*, as when the value of an ADC is read at preset time intervals and requires a small amount of CPU bandwidth. Again, interrupt-driven software is the best solution. If the peripheral (ADC) does not provide a hardware interrupt, the PC's timer could.

The following program listing, written in Microsoft Macro Assembler, shows the basic concepts for installing and using interrupt-driven software. It can be used with the data acquisition circuit from Chapter 6 (Figure 6-6), set to generate an IRQ7 hardware interrupt whenever a new ADC reading is ready. It is assumed that the 8259 interrupt controller already enables IRQ7 interrupts and that the system interrupt flag is set to enable the maskable interrupt input from the 8259. Otherwise, these functions must be taken care of in LOADVEC, the routine that prepares the system for the interrupt and loads the interrupt service routine INT7SVC, as

```
;*** MACRO ASSEMBLER PROGRAM TO READ ADC VALUE VIA IRQ7 ***
;
;* DATA INITIALIZATION *
DSEG1       SEGMENT AT 0          ;interrupt vector table starts at
                                  ;addr 0
            ORG 3CH               ;start of vector for IRQ7
IRQ7        LABEL WORD            ;Now we can access the vector for
                                  ;IRQ7
DSEG1       ENDS                  ;via the label IRQ7.
;
DSEG2       SEGMENT               ;Data storage segment
PUBLIC      DVALUES, DINDEX       ;Allows other programs access
                                  ;to these variables.
DVALUES     DB 256 DUP (?)        ;ADC data storage table
                                  ;(uninitialized)
DINDEX      DW 0                  ;Index into table (initialized to
                                  ;zero)
DSEG2       ENDS
;
CSEG        SEGMENT               ;Code segment, for programs
            ASSUME   CS:CSEG, DS:DSEG2

;
;* ROUTINE TO INITIALIZE IRQ 7 & LOAD SERVICE ROUTINE INTO MEMORY
LOADVEC:    MOV AX,0              ;Point to memory segment 0
                                  ;for interrupt
            MOV ES,AX             ;vector table.
            MOV ES:IRQ7,OFFSET INT7SVC    ;Set address of IRQ 7
            MOV ES:IRQ7+2,SEG INT7SVC     ;service routine.
            MOV DX,200            ;DX contains amount of memory
                                  ;to save
                                  ;for keeping service routine
                                  ;INT7SVC
                                  ;loaded in memory.
            MOV AL,0
            MOV AH,31H            ;Get ready for DOS function 31h
            INT 21H               ;Return to DOS, leaving
                                  ;INT7SVC resident
                                  ;in memory.

;
;* INTERRUPT SERVICE ROUTINE
ADC         EQU 300H              ;Address of ADC port (to read
                                  ;data)
INT7SVC:    PUSH AX               ;Save all working registers
            PUSH DS
            PUSH BX
            PUSH CX
            PUSH SI
            MOV AX,DSEG2           ;Point to data storage segment
            MOV DS,AX
            IN AL,ADC              ;Read data from ADC
            MOV SI,DVALUES
            MOV [SI+DINDEX],AL     ;Store data in table
            INC DINDEX             ;Point to next location in table
            CMP DINDEX,257         ;Past end of table?
            JNZ CONTIN             ;No
            DEC DINDEX             ;Yes, stay at end of data table
CONTIN:     MOV AL,20H             ;Send EOI command to 8259
            OUT 20H,AL
            POP SI                 ;Restore working registers before
            POP CX                 ;returning.
            POP BX
            POP DS
            POP AX
            IRET                   ;Return from interrupt
;
CSEG        ENDS
            END LOADVEC            ;Start execution at routine
                                   ;LOADVEC
;* END OF PROGRAM
```

Since IRQ7 is interrupt type 0Fh, its vector is located at memory address 0Fh × 4 = 3Ch in segment zero (physical address 0000:003Ch). When the program is run by DOS, it starts execution at routine LOADVEC. This short program loads the address of the interrupt service routine, INT7SVC, into the vector location for IRQ7 (3Ch–3Fh). Then it allocates enough space for INT7SVC and its data and returns to DOS, leaving INT7SVC resident in memory. This type of software is called terminate-and-stay-resident, or TSR. It is useful here, allowing the servicing of the IRQ7 interrupt independent of other software. The DOS call to INT 21h Function 31h is used to load TSR programs. The value in DX is the amount of memory to preserve for the resident program. AL contains the value returned by the function, which is useful for error codes. AH contains the function number.

Once INT7SVC is loaded into memory, whenever it is called it reads the current value from the ADC and stores it in a data table, starting at location DVALUES and indexed by DINDEX. Both DVALUES and DINDEX are declared as *public* labels, so that other software can access them and retrieve the data. A typical program making use of INT7SVC would check the value in DINDEX, address the ADC, start a data conversion, and then go about other business. When it was ready to retrieve the data, it would check that DINDEX has incremented and then read the data out of the table, DVALUES. When it was done, it would decrement DINDEX.

Note that the above program is merely an illustrative example of the use of interrupt-driven software for data acquisition. It is still fairly rough and incomplete for practical use, lacking refinements. INT7SVC does show some important aspects of interrupt service routines. They should be as fast as possible, to avoid interfering with other system interrupts. That is why they are usually written in Assembler (although short C programs are sometimes used). The working system registers (AX, BX, CX, DS, SI) should be saved, by PUSHing onto the stack at the routine's start, and restored, by POPing, at its end. Otherwise, any use of these registers by the interrupt service routine will corrupt the interrupted program, on return. For hardware interrupt service, the routine must send an EOI command to the 8259 interrupt controller. Otherwise, new hardware interrupts will not be enabled. The service routine should end with an IRET statement for a proper return from the interrupt.

An interrupt routine to service a software interrupt is somewhat simpler, since the 8259 does not have to be serviced and hardware interrupts do not need to be unmasked. In addition, there is little danger of monopolizing the CPU's bandwidth (unless hardware interrupts are masked off). Software interrupts are a convenient way to install and call software functions in memory.

To illustrate polled software used to retrieve an ADC value, the following is a function written in Microsoft C:

```
#include <conio.h>                /* needed for library function
                                     inp() */
#define ADC_STATUS 0x301          /* Address of ADC status port */
#define ADC_DATA 0x300            /* Address of ADC data port */

char read_adc()                   /* Name of function is
                                     adc_read */
  {
  while (inp(ADC_STATUS)!=1);     /* wait till ADC is done */
  return(inp(ADC_DATA));          /* send ADC value back to
                                     calling program */
  }                               /* Done */
```

Note that this is a very short and simple subroutine. The main program calls it whenever it has started an ADC conversion and wants to retrieve the results. It assumes that I/O port 301h contains a value of 1 only when the conversion is complete. This is the status required by a polling routine such as read_adc().

In this simple example, there is no provision for the error condition when something goes wrong and the ADC status port never returns a 1, as when there is a hardware failure or a software bug calling read_adc() at the wrong time. A more practical program would have a time-out provision in the while(...) statement. Otherwise, the PC will remain stuck in that loop indefinitely.

7.4 Special DOS Programs

There are several special-purpose programs used by DOS. These include device drivers and TSR programs.

7.4.1 Device Drivers

Previously, we have seen how useful interrupts are, both for calling existing DOS and BIOS functions and for interfacing to additional software functions, especially to support hardware such as data acquisition devices. Another special type of software is the device driver. A device driver is a distinctive program that is loaded into DOS (or any operating system) when the system boots up and then acts as if it is part of the operating system. As such, it must adhere to very strict guidelines. Device drivers are typically used to support special hardware functions. For example, a hardware mouse will usually have a device driver that allows it to work with common application software packages. Both 16- and 32-bit versions of Microsoft Windows rely even more heavily on device drivers for interfacing to hardware, as we will see later in this chapter.

In DOS, device drivers are loaded into the system by including commands in a text file called CONFIG.SYS in the root directory of the boot disk. This file contains entries used to customize DOS, such as number of buffers and number of files that can be open simultaneously. It also contains entries in the form

```
DEVICE = filename
```

where filename is the name of a device driver, typically with a SYS extension. So, to load a mouse driver (file MOUSE.SYS), CONFIG.SYS should contain the line

```
DEVICE = MOUSE.SYS
```

When DOS boots up, it looks for CONFIG.SYS and, if it is found, it executes the commands it contains and loads the device drivers listed in the file. It should be noted that DOS device drivers must be written in Assembler for the proper control of program and data layout. They are normally only written by experienced DOS programmers.

7.4.2 TSR Programs

When DOS software support is required for special hardware, often writing a terminate-and-stay-resident (TSR) program is an appropriate choice, especially if it is not for commercial product support. It is much easier than producing a device driver and it can be written in a high-level language, such as C.

As we previously touched on, a TSR program is interrupt-driven software. It is loaded into a PC's memory and can interface with other programs or with DOS itself. It continues to function until the system is turned off and RAM contents are lost (unless it is explicitly removed from memory). All TSR programs are activated by interrupts, either hardware or software. Some use software interrupt levels not reserved by DOS or BIOS, to allow an application program to access the TSR functions.

It is common for TSR programs to attach themselves to interrupts already in use. For example, many utility TSR functions are activated when a special combination of keys is pressed (a *hot key*). To do this, the TSR program attaches itself to the keyboard interrupt 09h. This interrupt occurs whenever any key combination is pressed. If the TSR program's hot key is pressed, it can take over and perform its function. If not, it passes control on to the original interrupt service routine. This is also an example of how interrupt routines can be chained, with more than one service routine using the same interrupt level. In a similar fashion, some TSR programs that must perform a task periodically use the system timer interrupt.

7.5 DOS

As the primary hardware focus of this book has been on IBM PC/XT/AT systems and compatibles, the software focus has been on Microsoft/IBM DOS as the operating system for older PCs. DOS was by far the most popular software environment used by pre-80386-based PCs, but not the only one. It is still widely used in embedded PCs (see Chapter 12). DOS is a *single-user, single-task* operating system, meaning it can only do one thing (execute one program) at a time. For simple PC applications, including some data acquisition and control, this is adequate. For cases where mainframe-style functioning is needed (such as multiuser support) a more sophisticated operating system could be used. Similarly, special operating systems are used for operating a local area network (LAN) connecting multiple PCs together.

DOS grew considerably after its initial release in 1981. Version 1.0, for the original IBM PC, only supported single-sided 5-1/4" floppy disks. Version 1.1 supported double-sided 5-1/4" floppy disks. Version 2.0 was released with the IBM PC/XT and added support for a hard disk drive. Version 2.1 added support for IBM's Portable PC and its ill-fated PC*jr*. Version 3.0 was released for the IBM PC/AT and supported high-density (1.2 Mbyte) 5-1/4" floppy disks. Version 3.1 added networking support. Version 3.2 added support for 3-1/2" floppy disks. Version 3.3 included support for the IBM PS/2 systems. Version 4.01 added expanded memory support and an optional, menu-based interface shell, enhancing its standard command-line interface. In addition, it allowed larger disk drives (up to 512 Mbytes) to be used as a single logical device. DOS versions below 4.0 required a hard disk greater than 32 Mbytes to be partitioned into multiple logical drives. Version 5.0 increased the maximum hard disk partition to over 2 Gbytes, added the ability to load DOS into high memory (between 640K and 1M) and included new utilities such as DOSKEY (to recall previous commands) and UNDELETE (to recover an accidentally erased file).

The last version of MS-DOS was 6.22. It contained additional utilities such as MEMMAKER (a memory-optimization program) and SCANDISK (a disk drive maintenance program). It even had integrated disk compression support (via DriveSpace), which was useful for older, smaller hard drives.

The primary advantage for using DOS was that it was supported by a vast array of commercial software products. Plus, it was a simple, real-time operating system that allowed you to directly control hardware. In addition, it was relatively inexpensive. Its primary disadvantage, besides being a single-task environment, was its memory limitation. A DOS application could only directly access up to 640 Kbytes of system RAM, regardless of the hardware capabilities of the PC. This stemmed from the original PC's 8088 CPU with 1 Mbyte of available physical addressing space and 384 Kbytes reserved for memory on peripheral devices (such as video display and disk controller cards). As an additional limitation, DOS allocated some memory for its own uses, typically leaving well under 600 Kbytes available for use by an application program. If a PC had drivers loaded for network support, there may have been less than 500 Kbytes available for applications. In general, each successive version of DOS monopolized more memory for itself.

When an 80286 or higher CPU (80386, 80486, Pentium) runs DOS, with its 1 Mbyte addressing limit, it is working in the processor's *real mode*, which is fully compatible with the old 8088. To access physical memory above 1 Mbyte, the CPU must use its *protected mode*, which is not supported by DOS. Windows running on an 80386 or above PC does fully support protected mode and an extremely large address space, as we will see later.

For many applications, the 640-Kbyte limit of DOS is not a problem. For data acquisition applications, however, this can be a severe limitation, especially when a huge amount of data is being acquired and analyzed. For example, let us assume a system was acquiring 16-bit data at a rate of 50,000 samples/second, running a program under DOS. Also assume it had 512 Kbytes of memory available as data storage (the rest of the DOS range was needed for the program code). It would take just 5.12 seconds of data to fill up this memory buffer. Obviously, if more data acquisition was required for each test, the data would have to be stored in a disk file as quickly as possible, before the memory buffer filled completely. If this data was being analyzed, the application program would have to keep reading in new data from the disk file if more than 5.12 seconds was stored. There are several ways to get around the memory limitations of DOS. The best option is to use a protected-mode operating system such as Microsoft Windows or Linux.

7.6 Overcoming DOS Memory Limitations

There are several techniques available to extend the memory limitations of DOS. These approaches are useful when working with older or embedded PCs that would not be suitable for running a protected-mode operating system such as Linux or Windows.

7.6.1 Overlays

When writing your own program, a simple technique to reduce the amount of memory required for execution is to use overlays. An *overlay* is a section of program code that is loaded into memory only when needed, and otherwise resides in a disk file. As illustrated graphically in Figure 7-2, an executable program residing in memory can consist of several code sections. These code sections, containing the program's instructions, can be subdivided into a program core, which is always resident, and one or more overlay sections. An overlay section contains code that can be swapped out and replaced by other code as the program executes. This swapping is controlled by the program core, which would contain all the functions and variables required by the various overlay. It is important for the individual overlay code

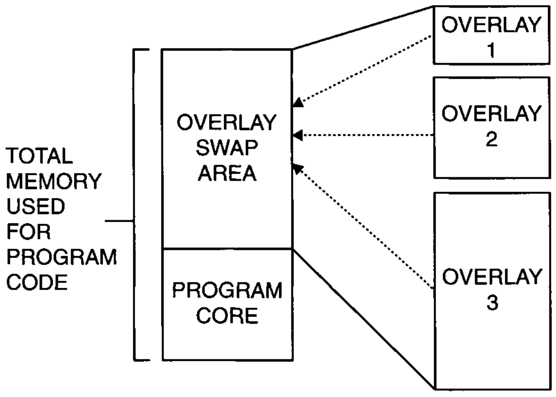


Figure 7-2 Example of program overlays.

sections to operate independently of each other, though not of the program core.

In the example of Figure 7-2, one overlay swap area is shared by three overlay sections. The overlay swap area must be as large as the biggest overlay that uses it. In this case, if the largest overlay is number 3, the memory saved by this technique (presumably for data storage) is the sum of the memory required for overlays 1 and 2. Of course, there are limitations on the amount of memory savings produced by using overlays and a program's structure must be very carefully worked out to use them. One major drawback to using overlays is slow program execution. Every time an overlay is swapped into memory (from a disk drive) the program must wait. The more overlays a program uses, the more swapping will occur during execution and the slower the overall program will run.

7.6.2 Expanded Memory

One popular and well-supported technique for stretching the 640-Kbyte memory limit of DOS was called expanded memory, which should not be confused with an AT's extended memory (beginning at an address of 1 Mbyte). Expanded memory was a standard developed by Lotus, Intel, and Microsoft, referred to as the LIM standard, which provided access to up to 8 Mbytes of extra memory, even on a PC/XT system. Expanded memory worked within the 1-Mbyte DOS addressing range. It was a memory page swapping technique. As shown in Figure 7-3, an unused block of memory up to 64 Kbytes long, between 640K and 1M, was set aside as a page frame. This area could contain up to four pages of memory, each 16 Kbytes long. Special hardware (either a separate peripheral card or part of the system's motherboard) contained the physical memory storage: up to 8 Mbytes of pages, 16 Kbytes long. At any time, up to four pages of physical memory could be mapped into the 64-Kbyte page frame, where they were addressable by DOS and the rest of the system.

To make use of expanded memory hardware, a device driver had to be installed into the system's CONFIG.SYS file. This DOS driver was usually called EMM.SYS (for expanded memory manager) and it operated through INT 67h. This driver controlled the memory page mapping and allocation functions. Many DOS applications supported expanded memory when it was present in a system.

It should be noted that expanded memory was normally used just for data storage since you could not execute code from it or even from the page frame space (above 640 Kbytes). LIM version 4.0 did add support for enhanced expanded memory which could swap an entire program into and out of expanded memory, and it supported a multitasking environment.

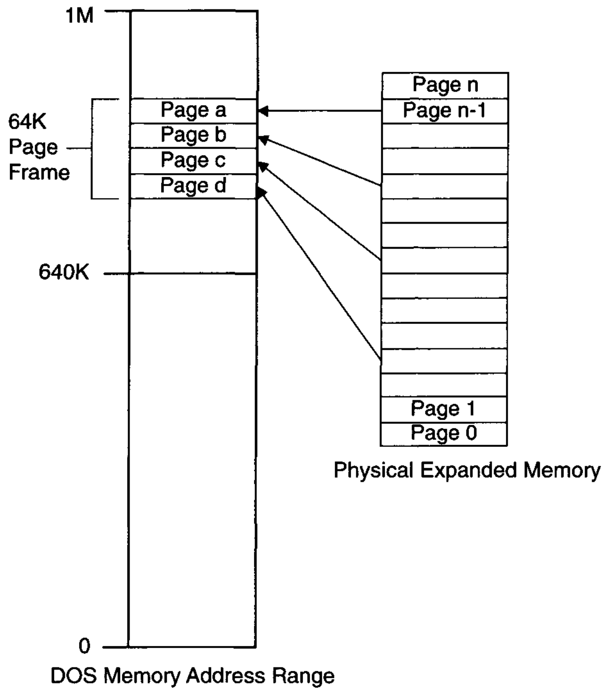


Figure 7-3 Mapping of expanded memory page frames.

Since the memory page mapping of expanded memory was controlled by dedicated hardware it was relatively fast, though not as fast as directly addressing memory in an AT system's protected mode (as long as there is no context switching between protected mode and real mode, which is fairly slow). Expanded memory was extremely useful for DOS data acquisition applications that required large amounts (megabytes) of data storage in RAM, at data transfer rates that could outrun disk drive speeds.

7.7 Protected-Mode Operating Systems

To make full use of AT systems which can physically address more than 640 Kbytes of system memory (using 80286, 80386, 80486, or Pentium CPUs), special software or another operating system is needed to operate in the processor's protected mode. One such early operating system from IBM and Microsoft was OS/2. It allowed a system to run large application programs using more than 640 Kbytes of RAM for code and data. It enabled the use of an AT system's *extended memory*, which starts at address 100000h or 10000:0000h (which is 1 Mbyte). Of course, an application had to be compatible with OS/2 to make use of all the available extended memory. OS/2 never became very popular, in part because it did not support DOS programs well.

7.7.1 Microsoft Windows

Microsoft's Windows, in its many variants, is now the most popular operating system for newer PCs (running 80386 or above CPUs). It is an operating system that supports large applications and makes full use of a system's physical memory. As a multitasking system, MS Windows allows more than one program to reside in memory and operate at any given time. Each program has its own window on the display screen. In addition, data can be easily transferred from one program or window to another, facilitating complex tasks using multiple applications (such as incorporating the results of a spreadsheet calculation into a word processing document). Windows is built around a graphics-based user interface, analogous to Apple's Macintosh operating system. To take advantage of all these features, an application must be specifically written to be compatible with Windows (although DOS applications will run under most versions of Windows).

Microsoft Windows actually encompasses several different operating systems. The original MS Windows, which eventually evolved into the popular Windows 3.1, was a 16-bit operating system that ran on top of DOS (you booted the PC into DOS and then started running Windows). It used real mode (16-bit) DOS for file services while running Windows applications in protected mode (32-bit). Software in protected mode had access to all the extended memory installed in the PC. Windows acted as a memory manager, allocating memory to multiple applications while keeping them isolated from each other (in their separate screen windows). This allowed multitasking and simplified data sharing.

Windows 3.1 could also run multiple DOS applications in separate windows. Each DOS program appeared to have a virtual PC at its disposal with 1 Mbyte of memory available. The hybrid real mode–protected mode environment of Windows 3.1 was somewhat clumsy and not always reliable. Still, a plethora of application software was written for Windows 3.1. This is sometimes referred to as 16-bit Windows software.

Windows 3.1 used text files with an INI suffix, including SYSTEM.INI and WIN.INI, as a means of controlling the system's configuration. These configuration files were akin to CONFIG.SYS and AUTOEXEC.BAT used by DOS. Newer versions of Windows (such as Windows 95 and later) continue to support these INI files but rely on a registry for most configuration information.

Windows NT was Microsoft's first full 32-bit operating system. It was aimed primarily at high-end systems and network servers running only 32-bit software and did not run DOS or 16-bit Windows software very efficiently. NT did become popular among commercial users who liked its extended network support, multiuser capabilities, and mainframe-style security features. Newer versions of Windows NT (including Windows 2000) have become quite suitable for stand-alone PCs and are extremely robust operating systems.

Windows 95 became Microsoft's successor to Windows 3.1. It is a full 32-bit operating system that allows applications to access up to 2 Gbytes of memory using a protected-mode 32-bit linear address space (as opposed to the real-mode 16-bit memory model of segment:offset). This operating system totally replaces DOS, offering new features such as long file names. Yet, it can still run multiple DOS applications with better control and reliability than under Windows 3.1. Windows 95 can also run 16-bit Windows programs. Still, it always stays in protected mode even when running 16-bit, real-mode software (including processing the INT 21h instruction for DOS function calls).

An application "talks" to Windows 95 by calling an application program interface (API) function. The program requests system services using a named function call instead of a numbered software interrupt, as used in MS-DOS. A connection is made between a Windows application and the function it calls at program load time by a process called *dynamic linking*. By contrast, MS-DOS would simply load an application into real-mode memory and give it full control of the PC, because it was single-tasking.

Another improvement in Windows 95 over Windows 3.1 is how it handles multitasking. Windows 3.1 was a cooperative multitasking system that relied on the application program to surrender the CPU periodically. If software was poorly written or just "hung," Windows could not do anything about it and the system would easily crash, forcing a reboot.

Windows 95 is a preemptive system that alone decides when to switch tasks, preventing a single application from monopolizing all the CPU time. Not only does this make the operating system more robust, it provides a faster response to high-priority, real-time events. This is especially important for data acquisition and control applications. Windows 95 also uses a registry, a special database, to keep track of system information, especially regarding application software. The registry is updated whenever new software is installed or removed.

Windows 95 is already considered obsolete. Its successors are Windows 98, Windows Me, and Windows XP. However, these newer operating systems still use the same basic 32-bit core of Windows 95. They primarily include more features such as improved Internet functionality, USB support, and a 32-bit file allocation table (see Chapter 9) which increases the maximum hard drive size from 2 Gbytes to over 2000 Gbytes.

Microsoft Windows NT has also continued to evolve. Windows NT 4.0 was succeeded by Windows 2000 (not to be confused with Windows Me). Again, the newer versions of NT have additional features and improvements such as support for multiple processors in a single PC. In general, the Windows NT family is still a bit more robust than the Windows 95 family.

Nearly all software written for Windows 95/98/Me will run under Windows NT/2000. Because of security features under Windows NT, you may need administrator privileges to load some application software or to install new hardware and device drivers.

7.7.2 UNIX and Linux

One other operating system we should note here is UNIX. This is a multitasking, multiuser operating system developed for minicomputers by AT&T Bell Laboratories. It has been *ported* to (adapted for use on) many different computing platforms. It has been especially popular on workstations and high-end PCs. Standard UNIX has a command-driven user interface, as DOS does. In fact, UNIX inspired many of the redirection and piping features of DOS. UNIX provides a large amount of power and flexibility, although some versions are not very user-friendly, owing to its often terse and cryptic commands. Microsoft even sold a 16-bit PC version of UNIX called Xenix.

A recently popularized UNIX-like operating system which runs on PCs (and other platforms) is Linux. Linux is a free UNIX work-alike, independently developed by a Finnish student, Linus Torvalds. Using the Internet he freely distributed his code and collaborated with many other programmers to develop the Linux *kernel* and its many add-on utilities. The kernel is the heart of the operating system that interfaces to peripherals and schedules and runs tasks, along with controlling the file system. The addition of several hundred utility programs makes a full distribution of Linux equal or superior to commercial PC operating systems. In addition, versions of Linux exist for many different types of computers, not just Intel-based PCs.

Linux, like UNIX, is a multitasking, multiuser operating system with full security features (as in Microsoft Windows NT). Later versions even support multiple processors in a single PC. It has extensive networking support and a growing list of free and commercial application software. Linux can even run software written for other operating systems and CPUs through emulation programs such as *emu* for MS-DOS, *wine* for MS Windows, and *executor* for the Macintosh operating system.

Linux is an extremely robust multitasking system that does a good job of isolating tasks from each other. An application program sends requests to the kernel using system calls. These calls are very general-purpose and not device-specific, which provides a great deal of flexibility.

An important part of Linux is the *shell*, which is a flexible command interpreter that also acts as a powerful, high-level command language. Complex tasks can be automated using simple shell programs or scripts. Linux also supports graphical user interfaces (GUIs) such as the X Window System, developed by MIT. This allows Linux users to work in an environment similar to MS Windows.

You can download a version or distribution of Linux for free via the Internet. A distribution contains code for the kernel, utilities, shells, GUIs, and installation programs. Alternatively, you can buy, at a nominal price, a prepackaged Linux distribution which includes manuals, technical support, and often some commercial applications. Two popular commercial Linux distributors are Red Hat and Caldera.

Linux is especially attractive if you want to do extensive software development because it is usually distributed with compilers and other software tools. Since the source code is also distributed with it, you can even customize the operating system.

This completes our survey of PC software interfacing. In the next chapter, we will explore common PC hardware interface standards including GPIB, RS-232C, and USB.

CHAPTER

8

Standard Hardware Interfaces

Previously we saw how a PC's I/O operates from its expansion bus. However, not all external I/O goes directly through the expansion bus. Often a standard hardware interface is used either by another computer or by an external peripheral device. This is increasingly the case with newer PCs that contain very few motherboard expansion slots and rely more on standard interface ports. We will explore several of these parallel and serial computer interfaces.

8.1 Parallel versus Serial Digital Interfaces

In general, digital computer interfaces to the outside world fall into two categories: parallel and serial. The differentiation between the two is important. For a digital interface n bits wide a parallel device uses n wires to simultaneously transfer the data in one cycle, whereas a serial device uses one wire to transfer the same data in n cycles. All things being equal (which they rarely are), the parallel interface transfers data n times faster than the serial interface.

Figure 8-1 shows an 8-bit wide interface between a PC and an external device. For simplicity, let us assume the data is unidirectional. The parallel interface in Figure 8-1a consists of eight data lines and one or more control lines. Control lines are needed to tell the receiving side when data is available (when the data lines are valid) and sometimes to acknowledge to the transmitting side that the data was received (a *handshake*). If this was a bidirectional interface, another control line indicating data direction would be

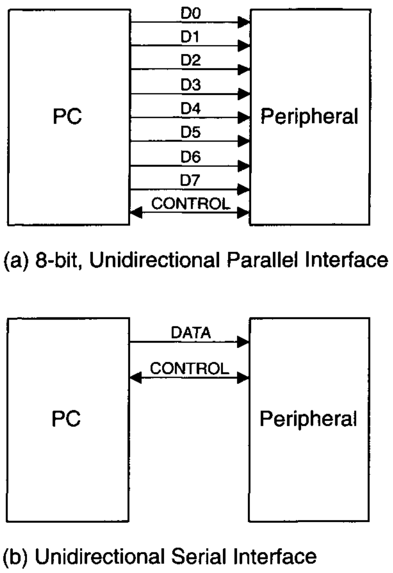

PC
D0
D1
D2
D3
D4
D5
D6
D7
CONTROL
Peripheral
(a) 8-bit, Unidirectional Parallel Interface
PC
DATA
CONTROL
Peripheral
(b) Unidirectional Serial Interface

Figure 8-1 Simple unidirectional digital interfaces: (a) parallel and (b) serial.

needed, along with a mechanism to prevent both sides from transmitting at the same time.

The serial interface in Figure 8-1b consists of only one data line (if it were bidirectional it probably would have two) and one or more control lines. In this scheme the data is time multiplexed. Control lines are used to indicate when the receiving end is ready to get the data along with other functions. The digital value of the data line represents a different bit at a different time. This requires a timing reference for the receiving end to decode the data accurately. When an external timing reference is used, this becomes a synchronous serial interface, with a control line carrying the required clock signal. When a receiver's internal timing reference is used, this becomes an asynchronous serial interface. To synchronize the incoming data stream with the internal clock, either a separate control line is used or, more commonly, a special start bit with a predetermined value is transmitted first. Then the data is sent, one bit per clock cycle, as shown in Figure 8-2.

Even though a parallel interface is inherently faster than an equivalent serial interface, it has its own drawbacks. Many parallel interfaces uses standard digital logic voltage levels, usually TTL compatible. This limits their noise immunity, where a long length of cable acts as an antenna, producing

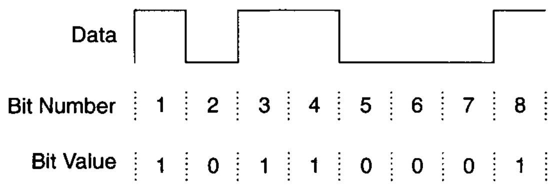


Figure 8-2 Sample 8 bits of serial data.

errors in the received data. In noisy environments, shielded cables are often required. In addition, long cables increase the capacitive coupling between adjacent signal lines, producing cross-talk errors (a signal transition on one signal line induces a voltage spike in another signal line). Dispersion further distorts the signals as cable length increases. All in all, parallel interfaces have severe cable length limitations, often on the order of just a few meters. High-speed interfaces, both serial and parallel, tend to use differential signal lines (where a pair of wires carry a single signal) to lower noise immunity.

In contrast, some serial interfaces use much wider voltage swings to increase noise immunity (±12 V is not unusual for RS-232C) and with few active signal wires, cross-talk noise is minimized. This enables serial interfaces to connect equipment hundreds of meters apart. Additionally, because fewer wires are required (and often shielding is not needed), serial interface cables are substantially less expensive (per foot) than parallel interface cables.

We will now explore some of these standard digital interfaces. First we will look at common parallel interfaces. Then, we will examine several serial interfaces supported on PCs. Later, we will look at some high-speed serial interfaces developed for PCs, including FireWire (IEEE 1394) and USB, as well as network interfaces such as Ethernet.

8.2 Parallel Interfaces

8.2.1 Centronics (Standard) Printer Interface

The standard parallel printer interface, sometimes called the Centronics interface, is available on most PCs (except for some of the newest models) and is supported by most printers. It is an 8-bit, unidirectional interface designed to transmit data from a computer to a printer, using TTL signal levels. The data usually sent is either ASCII codes, where each byte representing a printable character or a command (such as a line feed), or graphics data, consisting of command codes or data values (see Section 8.3.1 for a discussion of ASCII codes).

TABLE 8-1
Standard Parallel Printer Port Pin Assignments

PIN #	SIGNAL NAME	DIRECTION
1	-STROBE	OUT
2	DATA0	OUT
3	DATA1	OUT
4	DATA2	OUT
5	DATA3	OUT
6	DATA4	OUT
7	DATA5	OUT
8	DATA6	OUT
9	DATA7	OUT
10	-ACK	IN
11	BUSY	IN
12	PE	IN
13	SELECT	IN
14	-AUTO FD XT	OUT
15	-ERROR	IN
16	-INIT	OUT
17	-SELECT IN	OUT
18–25	GROUND	N/A

The standard IBM-style PC parallel printer port uses a 25-pin connector (DB-25) with the pin designations shown in Table 8-1. A special cable is used to connect this port to the 36-pin Centronics connector on most printers. The signal directions shown in Table 8-1 are relative to the PC. Signals with names starting with a "–" (such as –ACK) are active low. The eight data lines, DATA0–DATA7, are unidirectional, sending data to the printer. The primary control and handshake lines in this interface are BUSY, –ACK, and –STROBE. BUSY goes low when the printer is ready to receive a new data byte. When the PC detects the printer is ready, it puts out data on the lines DATA0–DATA7 for a minimum of 500 nsec. Then it asserts the –STROBE signal for a minimum of 500 nsec, which tells the printer to read the data. The PC keeps the data lines valid for at least another 500 nsec.

In the meantime, the printer asserts BUSY and does its internal processing. When ready, it simultaneously negates BUSY and asserts –ACK. –ACK is typically asserted for 5 to 10 μsec. The –ACK line is virtually a redundant

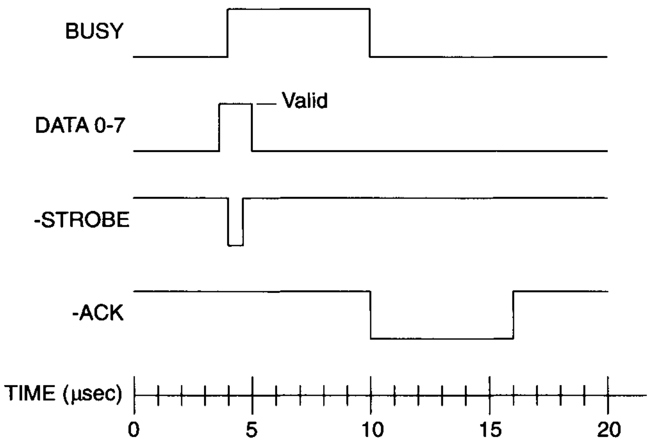


Figure 8-3 Parallel printer port interface timing.

signal and usually the BUSY line alone is an adequate handshake for the PC, signaling data was received by the printer. The timing of this interface is shown in Figure 8-3.

The other parallel port control lines are used for various status and control functions. When –AUTO FEED XT is asserted by the PC, the printer automatically performs a line feed after it receives a carriage return. When the PC asserts –INIT for a minimum of 50 μsec, the printer is reset to a known state (usually equivalent to its initial power-on conditions). When the PC asserts –SELECT IN, it enables the printer to receive data.

When the printer asserts PE it indicates it is out of paper. When the printer asserts SELECT it indicates it is enabled to receive data from the PC. When the printer asserts –ERROR it indicates that it is in an error state and cannot receive data.

A DOS-based PC can support up to three standard parallel printer ports (depending on its BIOS) designated LPT1, LPT2, and LPT3. Each port uses three consecutive I/O addresses. When a system boots up, DOS assigns the physical printer ports present to the logical LPT designations. LPT1 is assigned first, followed by LPT2, then LPT3. The starting addresses of parallel printer ports, in the order assigned to LPT designations are 3BCh, 378h, and 278h. So, if all three ports are present in one system, port 3BCh becomes LPT1, port 378h becomes LPT2, and port 278h becomes LPT3. If port 3BCh is not present, port 378h becomes LPT1 and port 278h becomes LPT2. If only one parallel printer port is present it is designated LPT1. In newer PCs running Windows 95 or later, the operating system determines the parallel port settings as well as the parallel port type (see Section 8.2.2).

The printer port's starting address (3BCh, 378h, or 278h) is the data port, which can be an input or output. Writing to this port address latches 8 bits of data on the DATA0–DATA7 lines sent to the printer. Reading from this port address returns the last byte latched (the real-time status of the output).

The printer port's next address (3BDh, 379h, or 279h) is the status port, which is read-only. It returns to the PC the value of the five status lines coming from the printer on the upper five bits of the port, as follows:

Bits 0–2 = unused
Bit 3 = –ERROR
Bit 4 = SELECT
Bit 5 = PE
Bit 6 = –ACK
Bit 7 = –BUSY

These lines can be polled for proper handshaking during a data output sequence. In addition, when –ACK is asserted (active low) it can generate IRQ7 (if enabled). This allows interrupt-driven software to handle printer output as a background task, for printer spooling. The printer would interrupt the PC, via its –ACK line, whenever it is ready to receive new data.

The printer port's next address (3BEh, 37Ah, or 27Ah) is the control port that can be an input or output. As an output, the PC latches the values of its control lines on the lower five bits of the port, as follows:

Bit 0 = –STROBE (1 = asserted)
Bit 1 = –AUTO FEED XT (1 = asserted)
Bit 2 = –INIT (0 = asserted)
Bit 3 = –SELECT IN (1 = asserted)
Bit 4 = IRQ EN (1 = asserted)
Bit 5–7 = unused

Note that most of the lines are inverted and asserted by a high bit except for –INIT, whose output follows the control port bit. The signal IRQ EN enables the port's IRQ7 output when bit 4 is latched high. As with the data port, a read from the control port will return the last value written to it.

The easiest way to use this parallel port to send data to a printer is with existing BIOS, DOS, or Windows functions. Using the BIOS, INT 17h services the printer ports. It can print a character (Function 0), initialize the printer (Function 1), or read the printer status (Function 2). On printing a character, the proper handshaking protocol is used, with a time out if there is no response (if BUSY stays asserted indefinitely). The logical printer port

(LPT) designation is used to select the desired printer. The BIOS does not support printer spooling, and special software must be used to support IRQ7 for printer output control.

A PC's parallel printer port can be used for other purposes besides printing, with certain limitations. It is ideal as a general-purpose output port with its eight unidirectional data lines, four output control lines and five input control lines. There was originally no standard software support for using it this way, unless the standard printer interface handshake protocol (as in Figure 8-3) was adhered to. This required special software to directly address the I/O ports used, supporting a custom protocol.

The parallel printer port can also be used as a general-purpose 5-bit input port, using the five status lines (–ACK, BUSY, PE, SELECT, and –ERROR). The real-time state of these lines can be read from the printer port's status register. In addition, the –ACK line can be used to generate IRQ7. The disadvantage here is having only 5 bits available for input and not being able to latch the data. Some commercial software has used this approach, called a nibble mode, to transfer 4 bits of data at a time. A common application is connecting a laptop computer to a PC via a special cable.

8.2.2 Advanced Parallel Printer Ports

The original printer port's limitations, of relatively low speed (only about 100 Kbytes/sec) and being primarily unidirectional, led to several improved standards.

The PS/2 Bidirectional Parallel Port IBM originally addressed the standard parallel port limitations in its PS/2 line of PCs. The parallel port on a PS/2 system has a fully bidirectional 8-bit data port, while keeping compatibility with the earlier implementation, as previously described. On this bidirectional parallel port (sometimes called a PS/2 parallel port), there is an extended mode that enables controlling the direction of the data port. Control port bit 5 (previously unused) now determines whether the data port is an output (bit 5 = 0) or an input (bit 5 = 1) port. The other control lines can now be used for different handshaking operations.

The PS/2 parallel port could also operate at speeds up to about 250 Kbytes/sec. It was better suited for transferring data between two computers than the standard (Centronics) parallel port was.

The Enhanced Parallel Port The Enhanced Parallel Port (EPP) was originally developed by Xircom Inc., Zenith Data Systems, and Intel Corp. as a next-generation parallel port. It is a fully bidirectional port with a typical data rate of about 800 Kbytes/sec and a peak rate of 2 Mbytes/sec.

The EPP uses a data register up to 32 bits wide (if it is running on a 32-bit processor) to speed up data transfers to the PC bus. The EPP uses hardware to handle all the details of partitioning 32-bit data into 8-bit transfers and controlling handshaking with the peripheral device (printer). Only one I/O port operation is required to write (or read) parallel port data. These features, along with stringent timing control, allow EPP to operate as fast as 2 Mbytes/sec (500 nsec for a single transfer cycle).

The EPP's pin assignments are shown in Table 8-2. EPP is backward compatible with a standard parallel port (often designated SPP). There are only six control lines used by EPP's hardware handshaking protocol. A signal name beginning with "n" indicates that it is active low. The nWRITE signal

TABLE 8-2
Enhanced Parallel Port (EPP) Pin Assignments

PIN #	SIGNAL NAME	DIRECTION
1	nWRITE	OUT
2	AD0	IN/OUT
3	AD1	IN/OUT
4	AD2	IN/OUT
5	AD3	IN/OUT
6	AD4	IN/OUT
7	AD5	IN/OUT
8	AD6	IN/OUT
9	AD7	IN/OUT
10	INTR	IN
11	nWAIT	IN
12	Spare (unused)	IN
13	Spare (unused)	IN
14	nDSTRB	OUT
15	Spare (unused)	IN
16	nINIT	OUT
17	nASTRB	OUT
18–25	GROUND	N/A

indicates whether the current cycle is a write or read operation. The INTR line is used by a peripheral to signal the PC that it needs service. The nWAIT signal is part of the hardware handshake and is used by the peripheral to signal that it has finished the transfer. The nDSTRB line indicates that there is valid data on the AD0–AD7 lines. The nINIT signal, when asserted, forces the interface out of EPP mode and into SPP mode. The nASTRB line indicates that there is a valid address on the AD0–AD7 lines.

EPP support four types of cycles: data write, data read, address write, and address read. An address refers to a register on the peripheral (printer or other device). Once an address is specified, data transfers, including bursts or multiple bytes, can occur between the PC and the register.

Figure 8-4 shows a simple EPP data write cycle. The nWRITE line first goes low to indicate a write cycle. Data is placed on the AD0–7 lines and nDSTRB is asserted (as long as nWAIT is low). The EPP waits for the handshake from the peripheral when nWAIT goes high. Then, nDSTRB is negated (high). When the peripheral is ready for another transfer, it sets nWAIT low again. If nWAIT never goes high (because of a hardware failure) the EPP times out after about 10 µsec.

The EPP uses the original three SPP registers at the I/O address base (3BEh, 37Ah, or 27Ah), base+1 and base+2. It additionally uses an EPP

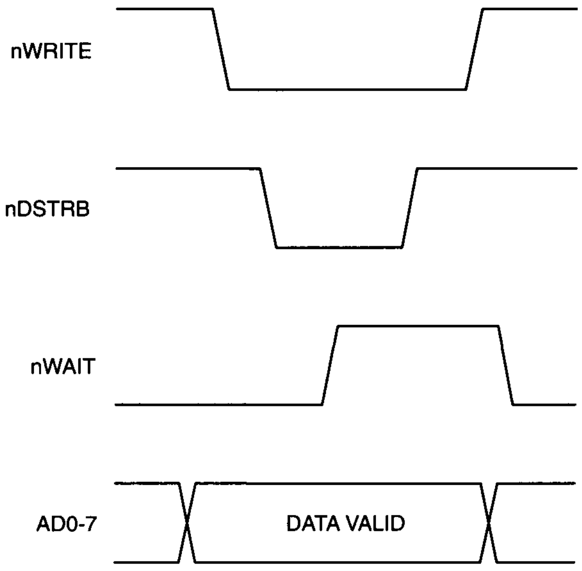


Figure 8-4 EPP Data Write cycle.

address register at location base+3 (for address write/read cycles) and an EPP data register starting at location base+4. This data register can be up to 32 bits long (four I/O addresses) on PCs that support 32-bit I/O transfers. This way, a single I/O write to the data register under software control can result in four EPP byte writes to a peripheral, under hardware control. This minimizes CPU overhead in servicing the parallel port.

The Extended Capabilities Port The extended capabilities port (ECP) was originally developed by Hewlett Packard and Microsoft as a means of extending EPP functionality into a universal expansion bus. As such, ECP is backward compatible with both SPP and EPP standards and has transfer rates comparable to EPP. The ECP protocol allows a PC to negotiate with a peripheral to determine which transfer mode and speed to use. A PC can query the peripheral to check its capabilities.

ECP uses seven signals to control data transfers, with hardware handshaking similar to EPP. It also uses separate data and command transfer cycles, where one of the control lines acts as a data/command flag.

ECP has several hardware features to improve its performance. It employs FIFO (first in, first out) memories to buffer data and reduce CPU overhead. ECP supports both hardware interrupts (IRQs) and DMA transfers to further minimize CPU involvement. Most notably, ECP supports data compression using run length encoding (RLE) for compression ratios up to 64:1. RLE works well with data that has high bit redundancy, such as printer and scanner data (see Chapter 9 for more information on data compression).

As with EPP, ECP support the three original SPP I/O registers at the base address (3BEh, 37Ah, or 27Ah), base+1, and base+2. Unlike EPP, ECP adds its new registers at address base+400h (data FIFO), base+401h (configuration register), and base+402h (extended control register). The pin assignments for an ECP connector are shown in Table 8-3.

The IEEE 1284 Standard In 1994 the IEEE approved a parallel port standard: IEEE 1284. This standard encompasses all the parallel ports we have previously discussed and classifies them by the transfer mode used. IEEE 1284 covers connectors (several different types) and their pin assignments, cables and electrical operation of each interface.

Under IEEE 1284, the SPP used unidirectionally is operating in *compatibility mode*. When an unmodified SPP is used for limited bidirectional data transfers it operates in *nibble mode*. A PS/2 bidirectional port uses *byte mode* while an EPP operates in *EPP mode* and an ECP in *ECP mode*. When a parallel

TABLE 8-3
Extended Capabilities Port (ECP) Pin Assignments

PIN #	SIGNAL NAME	DIRECTION
1	HostCLK	OUT
2	DATA0	IN/OUT
3	DATA1	IN/OUT
4	DATA2	IN/OUT
5	DATA3	IN/OUT
6	DATA4	IN/OUT
7	DATA5	IN/OUT
8	DATA6	IN/OUT
9	DATA7	IN/OUT
10	PeriphCLK	IN
11	PeriphAck	IN
12	nAckReverse	IN
13	XFlag	IN
14	HostAck	OUT
15	nPeriphReq	IN
16	nReverseReq	OUT
17	1284Active	OUT
18–25	GROUND	N/A

port is IEEE 1284 compliant, it supports EPP and ECP modes at data rates up to 2 Mbytes/sec over cables as long as 10 meters.

The EPP and ECP are electrically defined by IEEE 1284 but their operating protocols are determined by their independent standards. Still, IEEE 1284 has been an important means of standardizing the use of PC parallel ports, especially for advanced data transfer applications with intelligent peripherals.

8.2.3 The IEEE 488 (GPIB) Interface

Another common parallel interface, primarily used for data acquisition, is IEEE 488 or GPIB (general-purpose interface bus). This interface is sometimes called the HPIB, as it was originally developed by Hewlett Packard to connect computers to their programmable instruments. GPIB was designed to connect multiple peripherals to a computer or other controlling device.

Even though it was intended for automated instrumentation applications, it has been used to drive standard PC peripherals such as printers, plotters, and disk drives. It transfers data asynchronously via eight parallel data lines and several control and handshaking lines. All signals are at TTL voltage levels.

Instead of connecting one computer to one peripheral device, GPIB allows one computer to control up to 15 separate devices. In many ways, GPIB acts like a conventional computer bus or network. Each GPIB device has its own bus address, so it can be uniquely accessed. It uses a hardware handshaking protocol for communications, which supports slow devices. When communicating between fast devices, data rates up to 1 Mbyte/sec can be obtained.

The GPIB uses a master–slave protocol for data transfer. There can only be one bus master, or controller, at any given time. Typically, the master device is the controlling computer. A device on the bus has one of three possible attributes: controller, talker, or listener. The controller manages the bus, sending out commands that enable or disable the talkers and listeners (usually, slave devices). Talkers place data on the bus, when commanded to. Listeners accept data from the bus. A device can have multiple attributes, but only one at any given time. The computer can be a controller, talker, and listener; a read-only device, such as a plotter, will just be a listener; and a write-only device, such as a digital voltmeter, can be both a talker (when it reports a data reading) and a listener (when it is sent setup information, such as a scale change).

The GPIB cable consists of 16 signal lines divided into three groups. The first group of signals consists of the eight bidirectional data lines, DIO1–DIO8. The second signal group consists of the three handshaking lines used to control data transfer: DAV, NRFD, and NDAC. The third signal group consists of five interface management lines that handle bus control and status information: ATN, IFC, REN, SRQ, and EOI.

The GPIB cable itself consists of 24 conductors, shielded, with the extra eight lines grounded. The cable is terminated with a special connector having both a plug and a receptacle, so that all the devices on the bus can be daisy-chained together in either a linear or star configuration. Typically, the cable length between any two devices on the bus must be no more than 2 meters, while the total cable length of the entire bus must be no more than 20 meters. To exceed these limits, special bus extenders are needed. An additional limitation is that at least two-thirds of the devices on the bus must be powered on.

The GPIB uses standard TTL logic levels with negative logic, so a control line is asserted at logic 0. This is because open-collector (or open-drain) drivers are normally used on the bus interfaces. Therefore, a signal is pulled to a logic 1 level until a device asserts it and pulls it down to a logic 0 level (this is a standard OR-tied technique). Figure 8-5a shows a simple GPIB linear configuration with four devices on the bus: a PC (controller),

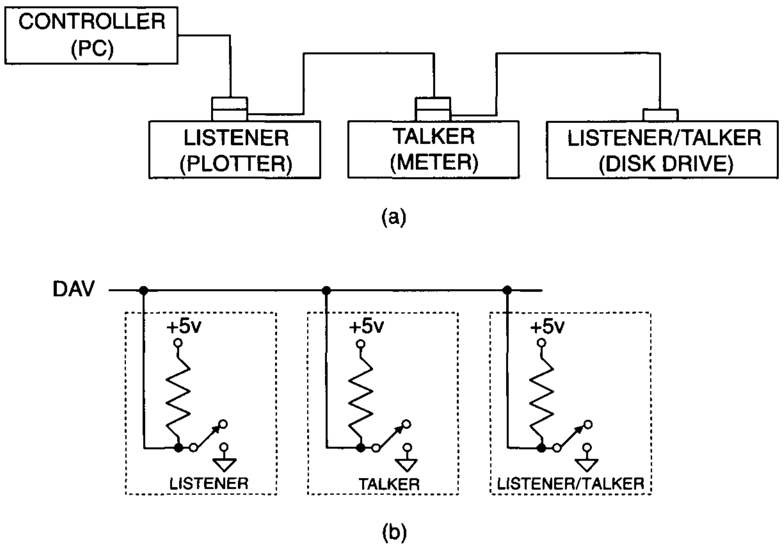


Figure 8-5 General-purpose interface bus (GPIB): (a) Typical GPIB linear configuration; (b) Open collector logic of a GPIB signal line (DAV).

plotter (listener), meter (talker), and disk drive (listener and talker). Note that there is a separate cable connecting each pair of devices in the daisy chain. No special termination is needed for the last device.

Figure 8-5b shows schematically the electrical connection of a signal line (DAV in this example), with open-collector drivers drawn as a switch to ground. Special line drivers specified for the GPIB are used on these interfaces to ensure that when a device is not powered on it does not load down the signal line (the switch to ground is open). Even with special drivers, there is some leakage current to ground when a device is not powered on. That is why a maximum number of devices are allowed to be powered off when the GPIB is operational.

The pin designations for the standard GPIB connector is shown in Figure 8-6. As previously mentioned, the bidirectional data lines are signals DIO1–DIO8. The descriptions of the three handshake lines are as follows:

1. DAV (data valid) indicates when the data line values are valid and can be read.
2. NRFD (not ready for data) indicates whether or not a device is ready to accept a byte of data.
3. NDAC (not data accepted) indicates whether or not a device has accepted a byte of data.

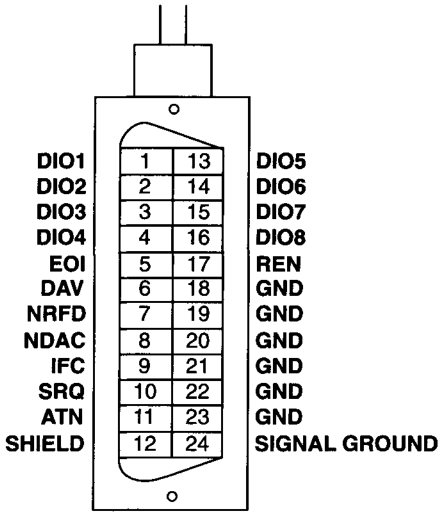


Figure 8-6 GPIB connector and pin designations.

The descriptions of the five interface management lines are as follows:

1. ATN (attention) is asserted by the controller when it is sending a command over the data lines. When a talker sends data over the data lines, ATN is negated.
2. IFC (interface clear) is asserted by the controller to initialize the bus when it wants to take control of it or recover from an error condition. This is especially useful when there are multiple controllers on a bus.
3. REN (remote enable) is used by the controller to place a device in the local or remote mode, which determines whether or not it will respond to data sent to it.
4. SRQ (service request) is used by any device on the bus to get the controller's attention, requesting some action.
5. EOI (end or identify) is a dual-purpose line. It is used by a talker to indicate the end of the data message it is sending. It is also used by the controller requesting devices to respond to a parallel poll.

The sequence used to transfer data asynchronously on the bus, using the handshaking signals, is shown in Figure 8-7. This sequence is between an active talker (or the controller) and one or more active listeners. The speed of the transfer is determined by the slowest device on the bus. Initially, all the listeners indicate their readiness to accept data via the NRFD line. When a

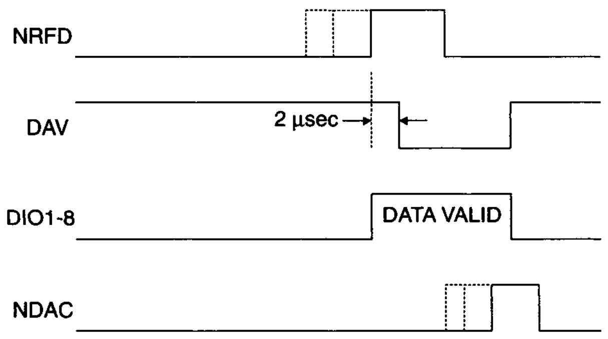


Figure 8-7 GPIB data transfer handshaking.

device is not ready, it pulls the NRFD line to a logic level 0, via its open collector output. As long as one active listener is not ready, NRFD is held low. Only when all active listeners are ready to receive data can NRFD go high (to logic level 1).

When the active talker (or controller) sees NRFD is high, it places its data byte on the bus (lines DIO1–8) and waits 2 μsec for the data bus to settle. Then it asserts DAV (to logic level 0), telling the active listeners to read the data. The listeners then pull NRFD low again, in response to the DAV.

The active listeners have all been holding NDAC active low. After DAV is asserted, as each active listener accepts the data on the bus it releases NDAC. When the last (slowest) listener releases NDAC, the signal goes high. The active talker (or controller) sees NDAC go high, negates DAV (goes high), and no longer drives the DIO lines.

Finally, the listeners recognize the negating of DAV and pull NDAC back low again, completing the transfer cycle. Now the handshake signals are ready for another data transfer to begin.

An important point is that this data transfer cycle is occurring between an active talker and one or more active listeners. Once the bus has been configured with talkers and listeners activated, the controller does not have to be involved in the transfer (unless it is operating as a talker or listener). For example, a disk drive on the GPIB could send data to a printer on the bus without a computer's involvement, once the process was set up.

Two types of data are sent over the DIO lines: control data and message data. When the data flows from a talker to selected listeners, it is a message, which is machine-dependent data. This message data can either be an instruction for a device (e.g., change the output voltage on an programmable power supply) or data to/from a device (e.g., a voltage reading from a DMM). When a controller uses the data lines, it is sending control data (a command) to all the devices (both talkers and listeners) on the bus. The controller asserts the interface management line ATN to signal that this is a control data transfer (normally, it is negated for message data transfers). When ATN is asserted, any active talker releases the DAV line. The control data is sent by the controller using the same handshaking protocol described above. The major difference is that all devices on the bus receive this data, whether listener or talker and regardless of their active/inactive status.

The control data handles many aspects of the bus operation. It can configure devices as active listeners or talkers or it can trigger a device to perform its specific function. Each device on the bus has a unique 5-bit address (0–30). The controller can specify a device's address, enabling it as an active listener, for example, during a control data transfer cycle. Since control data commands are used for configuring the active talker and listeners, it must be able to address all devices on the bus.

Device address 31 has unique meaning for setting up listeners and talkers. If a control data command is sent to activate a listener at address 31, it actually deactivates all listeners. This is effectively the "unlisten" command. Similarly, when a control data command is sent to activate a talker at address 31, it deactivates the current talker. This is the "untalk" command. In addition, if a device is selected as the active talker, any talker that is currently active deactivates itself. This ensures there is only one active talker at a time without requiring the bus overhead to explicitly deactivate the previous talker.

Another important GPIB management line is SRQ (service request), which is asserted by a device when it requires service from the controller. This may be an error condition in the device or an external event sensed by the device. Using SRQ is analogous to a processor interrupt, except that in this case the controller can ignore the SRQ or respond whenever it wants to. When the controller attempts to service the SRQ, it must first determine which device (or devices) is asserting the line. To do this it must poll all the devices on the bus.

There are two types of GPIB polling techniques: serial and parallel. In a serial poll, the controller issues a serial poll command, asserting ATN, to each device on the bus, getting back 8 bits of status information. One of these status bits indicates whether the device issued the service request. The other bits convey device-dependent information. The main disadvantage with using a serial poll is that it is slow, requiring the controller to poll all the devices one at a time. Using a parallel poll is faster. In this case, the controller issues the appropriate parallel poll bus command, along with asserting the ATN and EOI lines. Up to eight devices on the GPIB can respond at once, setting or clearing the appropriate bit. In a parallel poll the only information obtained is which devices requested service.

So far, software aspects of the GPIB have not been mentioned, because they were not part of the original IEEE 488.1 specification and were device-dependent. Every GPIB compatible device had its own unique set of commands. For example, a function generator would have a command telling it what type of waveform to output, and a programmable power supply would have a command for setting its current limit. These commands, and any appropriate responses such as the readings from a digital voltmeter, were all message data. Usually, message data on the GPIB consisted of ASCII characters. The use of ASCII data for the GPIB is supported by HP and the vast majority of GPIB equipment manufacturers.

IEEE 488.2 Tektronix attempted to standardize instrument message formats with a set of common commands and controller protocols. This grew into a new GPIB standard: IEEE 488.2-1987. The original GPIB standard was renamed as IEEE 488.1. The newer 488.2 standard is a superset of 488.1 (it is backward compatible).

The standard defines 10 commands that IEEE 488.2 compatible instruments must respond to. A good example of this is *IDN?*, which is the identification query command. An instrument should respond to this command with its manufacturer, model number, serial number, and revision.

IEEE 488.2 added a new status reporting structure to the original 488.1 status byte. This consists of a standard event status register (ESR) and an output queue. The ESR reports device status and command errors. An event status enable register determines which ESR bits become logically OR'd into the ESB bit of the status byte register.

IEEE 488.2 also supports instruments that can save and recall configuration information in nonvolatile memory (such as EEPROMs). This is done with the SAV and RCL commands.

One difference between the older and newer standard is the downgraded use of the device clear (DCL) command in IEEE 488.2. DCL no longer resets an instrument to its power-up state, as it did under IEEE 488.1. The RST or RCL0 command should be used for this purpose under 488.2.

The eight protocols defined under IEEE 488.2 are high-level routines that combine multiple control sequences into standard system operations. They include the ALLSPOOL (serial poll) and RESET protocols, supported by controllers. The FINDLSTN protocol finds and lists all the devices connected to the bus. The TESTSYS protocol runs a self-test of the system.

SCPI While IEEE 488.2 standardized communications with GPIB devices it still did not resolve the problem of each instrument having a unique set of commands. Hewlett Packard addressed this problem by developing its test measurement language (TML) which evolved into the industry-wide standard commands for programmable instruments (SCPI).

SCPI defines a comprehensive command set suitable for all GPIB instruments using common keywords and programming syntax. All SCPI-compatible voltmeters, for example, respond to the same command for reading DC voltage, independent of the manufacturer or model. Even different types of instruments use similar SCPI commands.

SCPI commands are usually a series of keywords and parameters. For example, the command to set the serial port bit rate on an instrument to 1200 bps would be

```
SYST:COMM:SER:BAUD 1200<CR>
```

The command to read back the bit rate would be

```
SYST:COMM:SER:BAUD?<CR>
```

The structure of the GPIB standards and how they interact is illustrated graphically in Figure 8-8.

Using a GPIB system can be very advantageous for complex data acquisition and control systems that require the high-level functionality of commercial test instruments. For example, consider a system required to characterize the frequency response of an electronic block box. Figure 8-9 shows a simple implementation using GPIB-compatible instruments: a function generator (to produce the variable excitation signal) and an AC voltmeter (to read the results).

A PC acts as the bus controller, using a commercially available GPIB interface card (see Chapter 11 for a sample of commercial sources). It controls the frequency and amplitude of the function generator's output (in this case a sine wave) and reads the AC voltmeter's input. Initially, the function generator should be directly connected to the AC voltmeter, to calibrate the system at its test frequencies. Then the device under test (DUT) is inserted between the generator and meter, and a new set of amplitude measurements is taken at the same set of frequencies. From this set of data, the transfer function or frequency response of the device under test (DUT) can be calculated.

There is a large amount of software support for PC-based GPIB interfaces. Most GPIB interface cards for PCs come with software drivers for use with popular programming languages, including versions of C, C++, and BASIC. Most high-end data acquisition software packages, such as MATLAB or LABTECH NOTEBOOK (see Chapter 11), support common GPIB cards, making the details of the GPIB operations invisible to the user. There are many other software packages with special features, making the process of implementing a GPIB system relatively painless. This is extremely useful

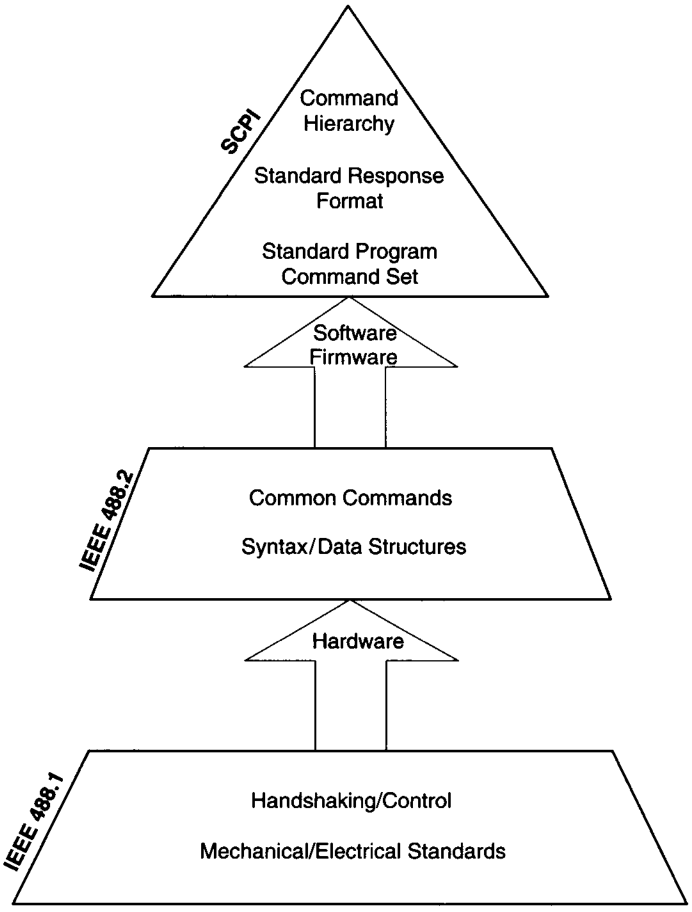


Figure 8-8 Structure of the GPIB standards.

because of the ever-growing number of instruments using the GPIB interface. GPIB equipment runs the gamut from power supplies and waveform synthesizers to digital storage oscilloscopes and network analyzers, to name just a few.

For example, National Instruments, a leading manufacturer of GPIB interfaces for a wide range of computers, provides the NI-488.2 software package for its PC-based products. NI-488.2 includes drivers for calling industry-standard NI-488 functions or newer NI-488.2 functions that cover all the IEEE 488.2 protocols. They offer software packages for most popular operating systems, such as Windows, Mac OS, and versions of UNIX (including Linux).

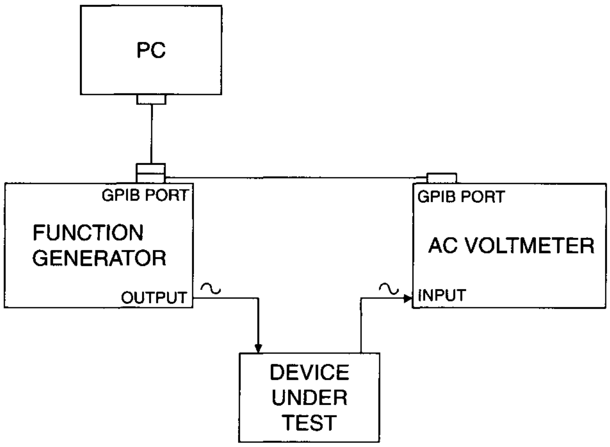


Figure 8-9 GPIB instrumentation example.

In an MS-DOS PC, the driver package would be loaded using standard procedures. Then the special GPIB functions are called from the user's program. One of the languages supported by the DOS version of NI-488.2 is QuickBASIC, a compiled version of BASIC (see Chapter 13 for a discussion of programming languages). A simple program in QuickBASIC to take a reading from a digital multimeter is as follows:

```
CALL IBFIND("DMM",DMM%)
CALL IBWRT(DMM%, "F0R0S2")
CALL IBRSP(DMM%,SPR%)
CALL IBRD(DMM%,DATA$)
PRINT DATA$
END
```

The first line in this program, calling IBFIND, retrieves initialization information on the specified device ("DMM") and returns the identifier code needed for the other functions. The second line, calling IBWRT, sends a device-specific message string to the DMM ("F0R0S2"), configuring it for voltage type, range, and speed (this is not a SCPI-compatible instrument). Next, the IBRSP call performs a serial poll on the DMM, checking its status. Finally, the IBRD call takes a voltage reading on the DMM and returns it in the string DATA$, which is then displayed by the print statement. In all of this, the user does not have to care about the details of the GPIB data transfers.

Newer versions of NI-488.2 software for MS Windows 95/98/NT support Microsoft Visual C/C++, Borland C/C++, and Microsoft Visual Basic 32-bit compilers. These drivers take full advantage of 32-bit multitasking operating systems. They also allow you to control several different GPIB interface types (such as PCI and PCMCIA cards) from the same PC using a single driver.

HS 488 By today's standards, the IEEE 488 maximum data rate of 1 Mbyte/sec is not very fast. One approach to improving this, developed by National Instruments, is HS 488, a high-speed GPIB handshake protocol that uses the same three control lines as IEEE 488 (DAV, NRFD, and NDAC).

HS 488 is backward compatible with standard GPIB instruments. However, if all devices on a bus support HS 488, the high-speed handshake is used and overall data rates can run as high as 8 Mbytes/sec (for two devices connected by no more than 2 meters of cable). A fully loaded bus with 15 devices connected by 15 meters of cable has a maximum HS 488 data rate of 1.5 Mbytes/sec (still a 50% speed improvement).

HS 488 accomplishes this speedup by removing excessive propagation delays and settling times associated with the standard IEEE 488 handshake (designed for maximum cable length and bus loading). Since the actual delays increase with longer bus cable lengths, the greatest speed improvement is seen with short cables.

There are already many instruments that support this new protocol. HS 488 has been proposed as an addition to the IEEE 488.1 standard. Currently (as of this writing) it is still a proprietary but well accepted standard.

8.2.4 Other Parallel Interfaces

Before leaving the topic of parallel digital interfaces, it should be noted that there are many other standards besides the parallel printer interface (IEEE 1284) and the GPIB. Most of these, such as BCD instrumentation interfaces or proprietary interfaces have little or no support in the world of PC-based data acquisition equipment.

One significant parallel standard is the Small Computer System Interface, or SCSI, which is usually used to connect high-speed disk drives to PCs. It is a general-purpose, asynchronous parallel interface, originally 8 bits wide, with later implementations 16 bits wide. SCSI can be used to connect virtually any piece of equipment to a PC, including data acquisition devices. In practice, this is rarely done, except for older Macintosh computers that used a SCSI interface as an external expansion port.

Over the years, SCSI technology has continued to improve. Currently (as of this writing), its fastest data transfer rate is 160 Mbytes/sec using Ultra 160/m Wide SCSI. The older SCSI interfaces used single-ended (SE) signal

TABLE 8-4
SCSI Standards

SCSI STANDARD	BUS WIDTH (bits)	SIGNAL TYPE (SE or LVD)	MAX DATA RATE (Mbytes/sec)
SCSI-1	8	SE	5
Fast SCSI	8	SE	10
Fast Wide SCSI	16	SE	20
Ultra SCSI	8	SE	20
Ultra Wide SCSI	16	SE	40
Ultra2 SCSI	8	LVD	40
Ultra2 Wide SCSI	16	LVD	80
Ultra 160/m SCSI	8	LVD	80
Ultra 160/m Wide SCSI	16	LVD	160

transmission (TTL or similar). Newer SCSI standards, such as Ultra2 SCSI and Ultra 160/m SCSI, use low-voltage differential (LVD) signals to improve data speed and integrity. SCSI interfaces are still only 8 or 16 bits wide, but newer standards run at faster speeds. Table 8-4 shows some of the common SCSI standards and their maximum data transfer rates.

8.3 Standard Serial Interfaces

Many standard digital serial interfaces are in use. They are differentiated by several factors, including voltage levels, current drive capability, differential versus single-ended lines, single receiver and transmitter versus multidrop capability, half- versus full-duplex, synchronous versus asynchronous, type of cable required, and communications protocols. These factors, in turn, determine important system specifications such as maximum data rate and maximum cable length. As we noted previously, the major reasons for using serial interfaces are low cable cost and potentially long cable lengths. The serial interfaces we will discuss in this section are all standards developed by the Electronic Industries Association (EIA) and are identified by their EIA standard number. The next section will cover high-speed serial interfaces developed primarily for PCs: USB and IEEE 1394 (FireWire).

The EIA standards define electrical characteristics and definitions of signal lines used in the interfaces. They do not define how the data will be sent or what each bit means. The two types of protocols used are asynchronous and synchronous. In an *asynchronous* protocol, the timing hardware at the transmitter and that at the receiver are independent of each other (they are not synchronized). Synchronization is provided by the data stream itself, usually a particular level transition to indicate the start of data.

In a *synchronous* protocol, timing information is exchanged along with data, providing a single clock signal used by both ends of the interface. This allows serial transmissions at higher data rates than asynchronous protocols, since extra control bits indicating the beginning and end of a data byte are not needed, along with the extra time for an asynchronous receiver to synchronize itself to an incoming data stream. It is, however, a more complicated and expensive approach. Most standard PC-based serial data interfaces use an asynchronous protocol. We will discuss the commonly used asynchronous protocols in the following sections, followed by a brief description of some common synchronous protocols.

8.3.1 The EIA RS-232C and RS-423A Interfaces

Without any question, the EIA RS-232C interface is the oldest and most common serial interface used by computer equipment. In fact, a PC's serial port is almost always RS-232C compatible. Because of its widespread use, RS-232C has paradoxically become one of the most nonstandard standards available. This is because it is used for much more than originally intended. RS-232C was developed in the 1960s as a standard for connecting data terminal equipment (DTE), such as the "dumb" terminals used with mainframe computers, to data communications equipment (DCE), such as modems, over moderately short distances at modest data rates. Over the years, RS-232C evolved as a general-purpose interface between many varieties of equipment. One common example is connecting a PC to a printer or plotter. You can even use a special interface box to control a GPIB system via a computer's RS-232C port.

The RS-232C standard uses a 25-pin D-shell connector, with line designations as shown in Figure 8-10. Note that transmit and receive data directions are relative to the DTE end. RS-232C is a serial interface having two data lines to support full-duplex operation. That is, the connected devices can simultaneously transmit and receive data, if they are capable. The maximum data rate on an original RS-232C interface is 20,000 bits per second (bps) and the maximum cable length is 50 feet (although this can be increased at lower data rates or in low-noise environments). In most PCs, the serial port can operate as fast as 115,200 bps. Note that EIA RS-232C can support either synchronous or asynchronous serial communications. In the vast majority of applications, asynchronous communications is used. However, the inclusion

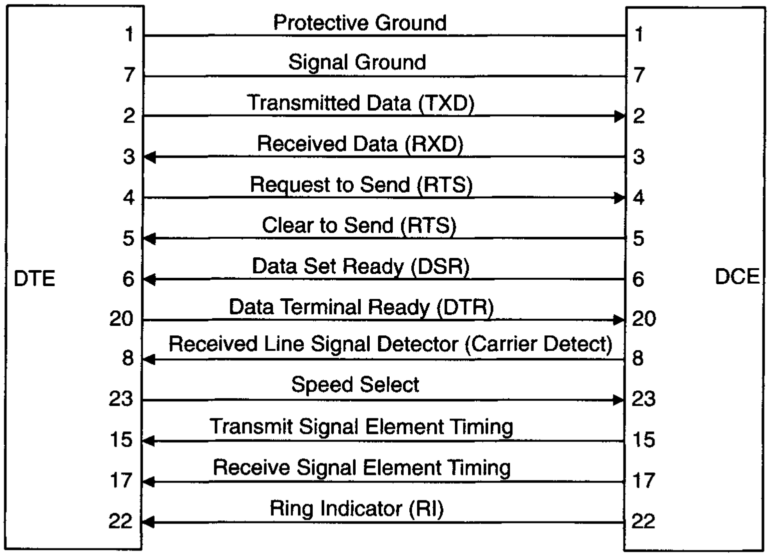

DTE
DCE
1 Protective Ground 1
7 Signal Ground 7
2 Transmitted Data (TXD) 2
3 Received Data (RXD) 3
4 Request to Send (RTS) 4
5 Clear to Send (RTS) 5
6 Data Set Ready (DSR) 6
20 Data Terminal Ready (DTR) 20
8 Received Line Signal Detector (Carrier Detect) 8
23 Speed Select 23
15 Transmit Signal Element Timing 15
17 Receive Signal Element Timing 17
22 Ring Indicator (RI) 22

Figure 8-10 Standard RS-232C connections between data terminal equipment (DTE) and data communications equipment (DCE).

of two lines, Transmit Signal Element Timing and Receive Signal Element Timing, can provide the external clocking required by synchronous interfaces.

The RS-232C interface supports several handshaking lines, indicating each device's readiness to send or receive data. This is not an interlocking handshake, as used in GPIB for control of data flow. It simply enables or disables data transmission. These lines include Request to Send (RTS), Clear to Send (CTS), Data Set Ready (DSR), and Data Terminal Ready (DTR). The control lines, Ring Indicator (RI), and Received Line Signal Detector (or Carrier Detect, CD) are specifically used by modems.

On a PC, the usual RS-232C serial interface card or motherboard circuitry supports asynchronous communications only and uses either a DB-25 or DB-9 connector. A PC/XT compatible system typically uses the 25-pin connector, with pin assignments shown in Figure 8-11. Note that some of the EIA RS-232C standard signal lines are not used, such as those needed for synchronous communications. In addition, four non-RS-232C signals are added: +Transmit Current Loop Data, –Transmit Current Loop Data, +Receive Current Loop Data, and –Receive Current Loop Data. These lines support the 20 mA current loop interface, used by older Teletype equipment and certain special devices such as industrial sensors.

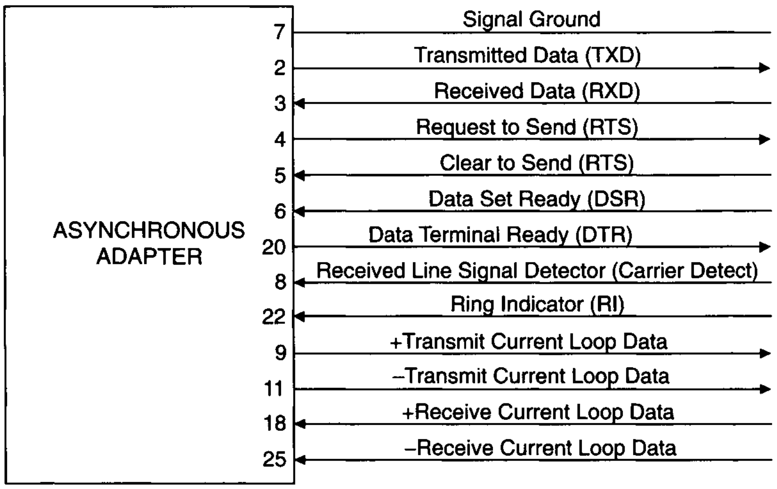

ASYNCHRONOUS ADAPTER
7 Signal Ground
2 Transmitted Data (TXD)
3 Received Data (RXD)
4 Request to Send (RTS)
5 Clear to Send (RTS)
6 Data Set Ready (DSR)
20 Data Terminal Ready (DTR)
8 Received Line Signal Detector (Carrier Detect)
22 Ring Indicator (RI)
9 +Transmit Current Loop Data
11 –Transmit Current Loop Data
18 +Receive Current Loop Data
25 –Receive Current Loop Data

Figure 8-11 Pin designations for 25-pin asynchronous adapter.

Newer PC systems (PC/AT and above) usually have a 9-pin connector, with its pin assignments shown in Figure 8-12. This limits the signals available to Transmitted Data (TXD), Received Data (RXD), DTR, DSR, RTS, CTS, RI, and CD. Usually a cable adapter is required to connect this 9-pin port to external devices with a conventional 25-pin D-shell connector.

Signals on RS-232C lines have well-defined electrical characteristics. Only one driver and one receiver are allowed on a line. The signals are all single-ended (unbalanced) and ground-referenced (the logic level on the line

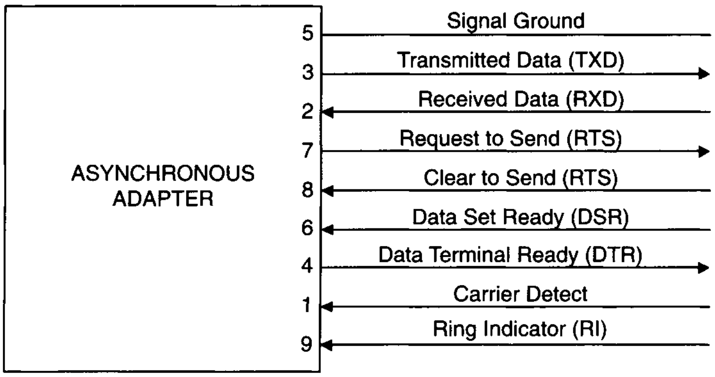


Figure 8-12 Pin designations for 9-pin asynchronous adapter.

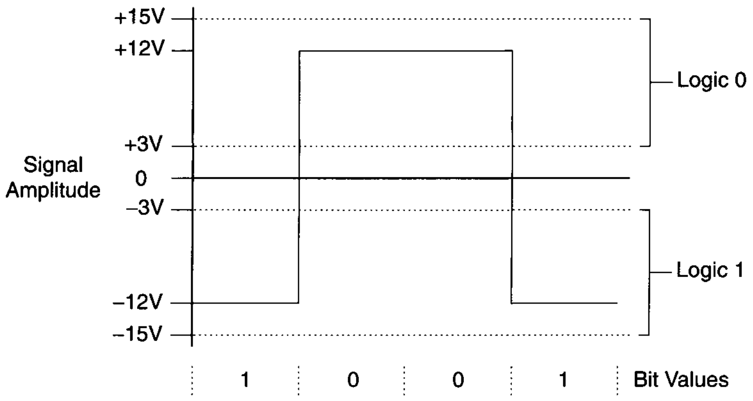


Figure 8-13 RS-232C signal levels.

depends solely on that signal's voltage value relative to the signal ground line). The signals are bipolar with a minimum driver amplitude of ±5 V and a maximum of ±15 V (±12 V is the most common voltage used) into a receiver resistance of 3000 to 7000 ohms. Receiver sensitivity is ±3 V, so any signal amplitude less than 3 V (regardless of polarity) is undefined. Otherwise, a voltage level above +3 V is a logic 0 and below −3 V is a logic 1, as shown in Figure 8-13. Another important parameter is a maximum slew rate of 30 volts per microsecond. This means that an RS-232C signal running at the maximum voltage range of ±15 V must take at least 1 μsec to switch states.

If we look at the typical RS-232C application in Figure 8-14, where a terminal is connected to a modem, we see that most of the handshaking lines

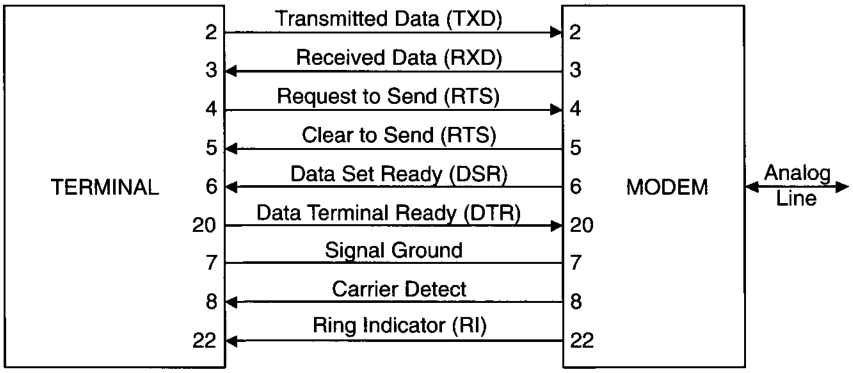


Figure 8-14 RS-232C connections between a terminal and a modem.

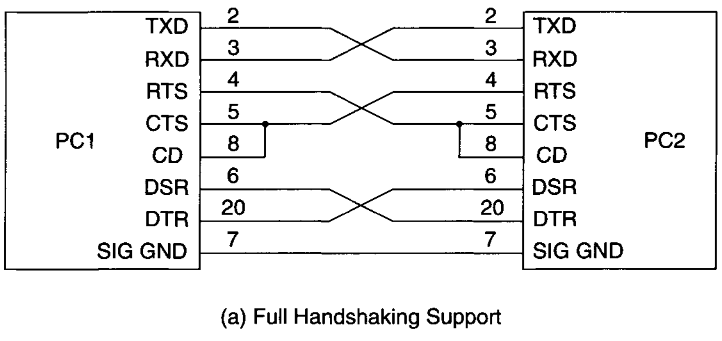
act in pairs. When the terminal wants to establish communications, it asserts DTR. As long as the modem is powered on and operational, it asserts DSR as the handshake. These signals stay asserted as long as the communications link exists. When the terminal is ready to send data it asserts RTS. The modem generates a carrier signal on its analog line (usually a telephone line connection) and after a delay (allowing time for the modem on the other end to detect the carrier) it asserts CTS. Then the terminal can transmit its data over TXD.

When the terminal is finished transmitting, it negates RTS, causing the modem to turn off its carrier and negate CTS. If the modem now receives a carrier from a remote system over the analog line, it asserts CD. When it receives data from the remote system, it sends the data to the terminal over RXD. The cable used to connect the terminal to the modem is a straight-through variety. That is, pin 2 on one end goes to pin 2 on the other end, pin 3 on one end goes to pin 3 on the other end, and so on.

In actual practice, RS-232C interfaces are used to connect many different types of equipment. The asynchronous communications port in a PC (the serial port) is nearly always set up as a DTE (TXD is an output line and RXD is an input line—the opposite is true for a DCE device). The meaning of the handshaking lines is software-dependent and they may not have to be used. If required, just three lines can be used to minimize cable costs: TXD, RXD, and signal ground. If the software requires it, CTS and DSR must be asserted at the PC end for it to communicate, as when BIOS INT 14h functions are used for sending and receiving data over the serial port (for DOS programs).

For example, if we want to send data between two nearby PCs without using two modems, we need a special cable, as shown in Figure 8-15. There are two approaches we can use to satisfy the handshake lines. In Figure 8-15a we implement full handshaking support, using seven wires. The data lines are crossed over, so TXD on one side is connected to RXD on the other side. Similarly RTS and CTS are crossed over as well as DTR and DSR. In this way, if the receiving end wants the transmitting end to wait, it negates its RTS line, which the other side sees as a negated CTS and CD; it then stops transmitting. Similarly, if one end wants to suspend communications entirely, it negates its DTR line, which the other side sees as a negated DSR. Signal ground is directly connected between the two ends. This cable, with the data and control lines crossed, is often referred to as a null modem cable. It is needed to connect a DTE to a DTE (or a DCE to a DCE).

A simpler connection using only three wires is shown in Figure 8-15b. In this case, the handshake lines are permanently enabled (*self-satisfying*) by connecting RTS to CTS and CD and connecting DTR to DSR at each PC. These lines cannot be used to control the data flow on the interface. The data



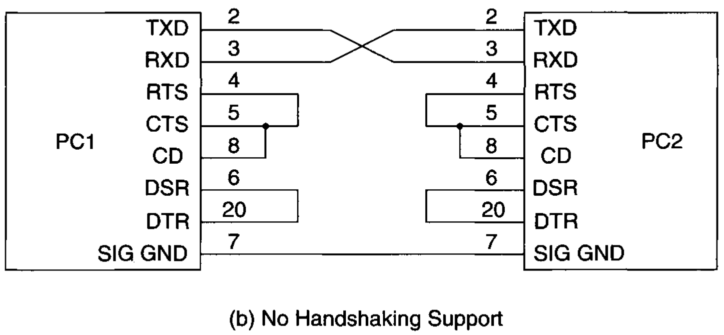


Figure 8-15 Connecting two PCs via an RS-232C cable.

flow can still be controlled, using special data characters in a software handshaking protocol. One software protocol widely supported is XON/XOFF. These are two ASCII control characters (XON is 11h, XOFF is 13h). When the receiving end needs to temporarily halt data flow, it sends an XOFF character to the transmitting end. When it is ready for data flow to resume, it sends an XON character. In a similar fashion, the ASCII characters ACK (06h) and NAK (15h) are also used for controlling data transmission. Employing either of these software control protocols necessitates the use of ASCII data.

ASCII stands for the American Standard Code for Information Interchange. It is the most widely used computer code for handling alphanumeric (text) data and is usually employed for data transfers between equipment over standard interfaces. It is a 7-bit code consisting of printable (alphanumeric) and control characters, such as XOFF and CR (carriage return). The standard ASCII code is shown in Table 8-5. On IBM-style PCs, an eighth bit is added to the code producing special ASCII extension characters. These are nonalphanumeric printable characters, such as lines for character-based graphics.

TABLE 8-5
Standard ASCII Codes

				0	0	0	0	1	1	1	1	b7
				0	0	1	1	0	0	1	1	b6
b4	b3	b2	b1	0	1	0	1	0	1	0	1	b5
0	0	0	0	NUL	DLE	SP	0	@	P	-	p	
0	0	0	1	SOH	DC1	!	1	A	Q	a	q	
0	0	1	0	STX	DC2	"	2	B	R	b	r	
0	0	1	1	ETX	DC3	#	3	C	S	c	s	
0	1	0	0	EOT	DC4	$	4	D	T	d	t	
0	1	0	1	ENQ	NAK	%	5	E	U	e	u	
0	1	1	0	ACK	SYN	&	6	F	V	f	v	
0	1	1	1	BEL	ETB	-	7	G	W	g	w	
1	0	0	0	BS	CAN	(	8	H	X	h	x	
1	0	0	1	HT	EM	)	9	I	Y	i	y	
1	0	1	0	LF	SUB	*	:	J	Z	j	z	
1	0	1	1	VT	ESC	+	;	K	[	k	{	
1	1	0	0	FF	FS	,	<	L	\	l	|	
1	1	0	1	CR	GS	-	=	M	]	m	}	
1	1	1	0	SO	RS	.	>	N	^	n	~	
1	1	1	1	SI	US	/	?	O	_	o	DEL	

As previously mentioned, the RS-232C standard does not specify the protocol used for data transmission. The vast majority of RS-232C interfaces on PCs use an asynchronous protocol. The transmission of 1 data byte using this protocol is shown in Figure 8-16. When no data is being transmitted, the line is at the *marking level*, which represents a logic 1. At the beginning of transmission, a start bit is sent, causing a line transition to the *spacing level*, a logic 0. This transition tells the receiver that data is coming. Next, the data

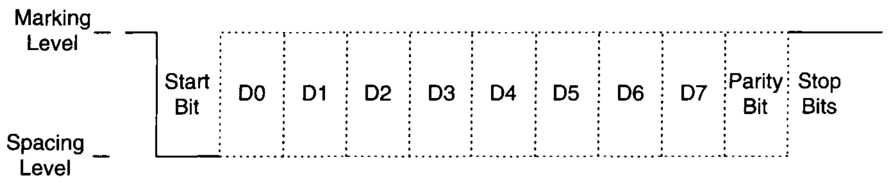


Figure 8-16 Asynchronous communications protocol.

bits (usually 7 or 8) are sent, one at a time, where a bit value of 1 is at the marking level and a bit value of 0 is at the spacing level. The data is followed by an optional *parity bit*, for error detection. Finally, one or more stop bits at the marking level are sent to indicate the end of the data byte. Since RS-232C line drivers and receivers are inverters, the marking level (logical 1) corresponds to a negative voltage (–3 V to –15 V) and the spacing level (logical 0) corresponds to a positive voltage (+3 V to +15 V) on the interface line.

The heart of a serial port's electronics is the IC that converts parallel data to a serial format and serial data back to a parallel byte. This device is a Universal Asynchronous Receiver/Transmitter (UART). IBM and compatible computers originally used the National Semiconductor INS8250 UART IC in PC/XT machines and the INS16450 UART (which is a superset of the INS8250) in AT and newer machines (later PCs had the UARTs built into the motherboard). These devices have separate transmit and receive channels and control logic for simultaneously sending and receiving data. They produce their own programmable timing signals, from on-board oscillators, for software control of data rates. They can send or receive serial data in the range of 50 bits per second (bps) to 38,400 bps (up to 115,200 bps with the INS16450). The width of each bit (in time) is the inverse of its data rate. So, at 9600 bps, each bit is 1/9600 = 0.104 msec long. If 7-bit data is sent at this rate using a parity bit and only 1 stop bit (for a total of 10 bits per character, including the start bit), it would take 1.04 msec (0.104×10) to transmit a character. This would produce a maximum overall data transmission rate of 961 characters per second. This is not incredibly fast, but for small amounts of data it is acceptable. Bear in mind that many early serial terminals and modems ran at only 110 bps (which is nearly two orders of magnitude slower).

To set up an asynchronous RS-232C communications link, both machines (at the two ends of the line) must be set to the same data rate (sometimes, incorrectly, called the baud rate). In addition, the number of data bits must be known. It can often vary from 5 to 8 bits, although 7 or 8 bits is the most common. The next parameter needed is the parity bit. This is used as a simple error-detection scheme, to determine if a character was incorrectly received. The number of logical 1's in the transmitted character is totaled, including the parity bit. For even parity, the parity bit is chosen to make the number of 1's an even number, and for odd parity it is chosen to make the number of 1's odd. For example, the ASCII character "a" is 61h or 01100001 binary. For even parity, the parity bit would be 1 (making four 1's, an even number), whereas for odd parity, the parity bit would be 0 (leaving three 1's, an odd number).

When a parity bit is used (typically with 7-bit data characters), the transmitting end determines the correct parity bit value, as just described, and incorporates it in the character sent. The receiving end calculates the expected value of the parity bit from the character's data and compares it to the parity bit actually received. If these values are not the same, an error is assumed.

Of course, this scheme is not foolproof. It assumes that the most likely error will be a single wrong bit, which a parity check will always catch. If multiple bits are wrong in the same character, a parity error may not always be detected. Note that on IBM-style PCs, the parity bit is not used with 8-bit data.

One final asynchronous communications parameter is the number of stop bits. This can be set to 1, 1-1/2, or 2 stop bits, although 1 bit is most commonly used. Unless very slow data rates are used, such as 110 bps, only 1 stop bit is adequate.

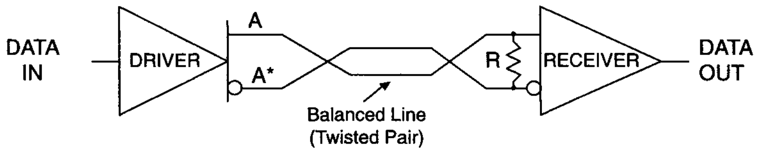
Several other single-ended serial communications interfaces are commonly used, besides RS-232C. One of these is RS-423A. This standard is sometimes used as an enhanced version of RS-232C, with several notable differences. RS-423A has a driver voltage output range of ±3.6 V to ±6.0 V, which is lower than RS-232C. However, RS-423A has much higher allowable data rates, up to 100K bps, and longer cable lengths (up to 4000 feet). One other important difference is that RS-423A can support multiple receivers on the same line, up to a maximum of 10. This is very useful for unidirectional data transfers in a broadcast mode, such as updating multiple CRT displays with the same information. Table 8-6 shows the differences between several of the EIA transmission line standards.

8.3.2 The EIA RS-422A and RS-485 Interfaces

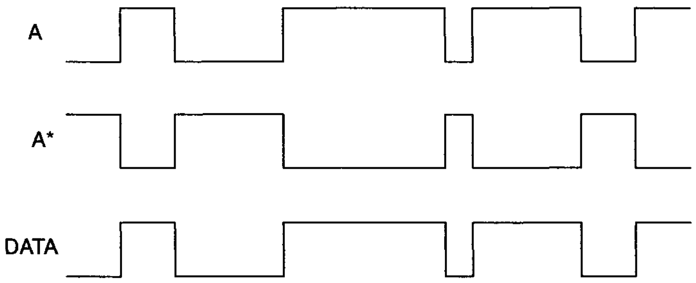
Another popular EIA serial transmission standard is the RS-422A interface, which uses differential data transmission on a balanced line. A differential signal requires two wires, one for noninverted data and the other for inverted data. It is transmitted over a balanced line, usually twisted-pair wire with a termination resistor at one end (the receiver side). As shown in Figure 8-17a, a driver IC converts normal logic levels to a differential signal pair for transmission. A receiver converts the differential signals back to logic levels. The received data is the difference between the noninverted data (A) and the inverted data (A*), as shown in the waveforms of Figure 8-17b. Note that no ground wire is required between the receiver and transmitter, since the two signal lines are referenced to each other. However, there is a maximum common-mode voltage (referenced to ground) range on either line of –0.25 V to +6 V, as shown in Table 8-6. This is because most RS-422A driver and receiver ICs are powered by the same +5 V power supply as many other logic chips. Usually the signal ground is connected between the transmitter and receiver to keep the signals within this common-mode range.

TABLE 8-6
Comparison of Selected EIA Interface Standards

PARAMETER	RS-232C	RS-422A	RS-423A	RS-485
LINE MODE	Single-ended	Differential	Single-ended	Differential
MAXIMUM DRIVERS AND RECEIVERS	1 Driver 1 Receiver	1 Driver 10 Receivers	1 Driver 10 Receivers	32 Drivers 32 Receivers
MAXIMUM CABLE LENGTH	50 feet	4000 feet	4000 feet	4000 feet
MAXIMUM DATA RATE	20 Kbps	10 Mbps	100 Kbps	10 Mbps
MAXIMUM (±) COMMON-MODE VOLTAGE	±25 V	+6 V −0.25 V	±6 V	+12 V −7 V
MINIMUM/MAXIMUM DRIVER OUTPUT	±5 V min ±15 V max	±2 V min	±3.6 V min ±6.0 V max	±1.5 V min
DRIVER OUTPUT RESISTANCE (with power off)	300 ohm	60K ohm	60K ohm	120K ohm
RECEIVER INPUT RESISTANCE	3K–7K ohm	4K ohm	4K ohm	12K ohm
RECEIVER SENSITIVITY	±3 V	±200 mV	±200 mV	±200 mV



(a) Driver-Receiver Connections



(b) Waveforms

Figure 8-17 Differential data transmission signals.

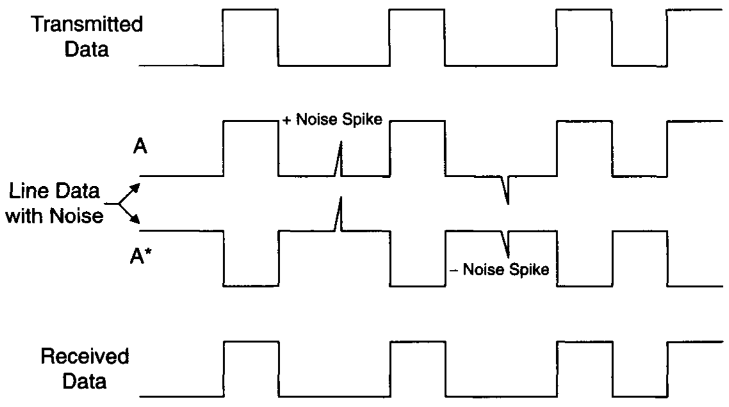


Figure 8-18 Differential data lines with common-mode noise.

This differential signal scheme enables the use of high data rates (up to 10 Mbps) over long cable lengths (up to 4000 feet) because of its high noise immunity. If external noise induces a signal on the transmission line, it will be the same on both conductors (A and A*). The receiver will cancel out this common-mode noise by taking the difference between the two lines, as shown in Figure 8-18. If a single-ended transmission line was used, the noise spikes could show up as false data at the receiver. The example in Figure 8-18 shows both a positive- and a negative-going noise spike.

As with RS-423A, RS-422A can have multiple receivers (10 maximum) on the same line with a single transmitter. Again, this is basically useful for applications that require broadcasting data from a single source to multiple remote locations.

There are variations in the connectors and pin designations used for RS-422A interconnections. Most RS-422A interface cards for PCs use 9-pin D-shell connectors, but in lieu of an IBM standard, the pin designations employed vary from one manufacturer to another. An example of the pin designations on a typical RS-422A interface card for PCs (from Qua Tech Inc.) is shown in Figure 8-19. Note that all the signal lines are differential.

The signal lines for AUXOUT are outputs and can be used to implement an RTS function. The signal lines for AUXIN are inputs and can be used to implement a CTS function. In this way, the RS-422A card can operate like a typical asynchronous RS-232C card in a PC (and use the same control software). Alternatively, the AUXOUT and AUXIN lines can be used to send transmit and receive clocks, for use with synchronous communications schemes.

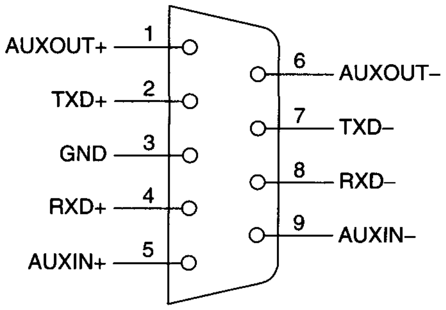


Figure 8-19 Pin designations for a typical RS-422A PC interface card.

The EIA RS-485 interface is basically a superset of the RS-422A standard. As shown in Table 8-6, its electrical specifications are similar to those of RS-422A. RS-485 is another differential transmission scheme, using balanced lines that can operate at speeds up to 10 Mbps over cable lengths up to 4000 feet long. It has somewhat different output voltage ranges, including a much wider common mode range of −7 V to +12 V. The most important difference is that an RS-485 interface can support up to 32 drivers and 32 receivers on the same line. This allows actual networking applications on a party line system (sometimes called multidrop) where all transmitters and receivers share the same wires.

To allow for this multidrop capability, RS-485 drivers must be able to switch into a high-impedance (tri-state) mode, so that only one driver is transmitting data at any given time. As with RS-422A, all receivers can be active at the same time. A typical RS-485 multidrop line is shown in Figure 8-20. Note that the termination resistor is typically placed at the last receiver on the line.

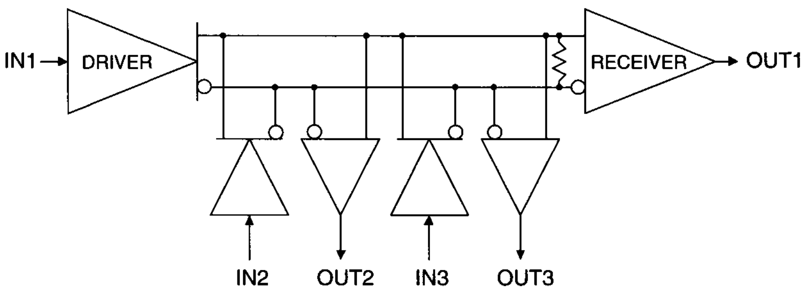


Figure 8-20 RS-485 multidrop application.

RS-485 interface cards for PCs are readily available and typically use the same connector (DB-9) and pin designations as similar RS-422A interface cards. The RS-485 driver output can be tri-stated using a control signal on the card. Usually a standard control line such as DTR is used for this since it would not be used as an external line in a multidrop interface. It is up to the software protocol to ensure that only one driver is enabled at any given time. One common way to do this is to use a master–slave relationship on the line. Only one driver/receiver station would be the master (or a network controller)—the others would be slaves. The master can transmit data at any time. The slaves can only transmit data after receiving an appropriate command from the master. Each slave would have a unique ID or address on the line and would not be able to transmit unsolicited data. The high data rates available to an RS-485 network would compensate for the moderate amount of communications overhead required to implement a master–slave protocol and the constant polling performed by the master. For more information about networks, see Section 8.4.

8.3.3 Synchronous Communications Protocols

As previously mentioned, synchronous serial communications protocols are much less common than their asynchronous counterparts in the world of PCs, even though IBM did have synchronous communications adapters available for their older PCs. Synchronous communications has noticeable advantages over asynchronous methods. Synchronizing bits (start and stop bits) are not needed, increasing the overall data transmission rate. Data does not have to be byte oriented (i.e., character-based) to be sent. In addition, it allows a system to communicate with large mainframe computers (especially IBM systems) which often use synchronous protocols. The drawbacks to using synchronous communications with PCs are higher costs for hardware and software along with limited support.

In synchronous transmissions, data is not always broken up into discrete characters, as with asynchronous methods. It tends to be block oriented, with a large amount of data (a block) transmitted at one time, with various control and error-checking information along with it. The data can be discrete characters (as with asynchronous methods) or bit oriented (no explicit data length). There are three common synchronous communications protocols: Binary Synchronous Communication (BSC), Synchronous Data Link Control (SDLC), and High-Level Data Link Control (HDLC).

BSC or *bisync* is a protocol developed by IBM. It is a character-oriented synchronous protocol where each character has a specific boundary. As with other synchronous protocols, there are no delays between adjacent characters in a block. Each block transmission may start with two or more PAD characters to ensure that the clock at the receiving end of the line becomes synchronized with the clock at the transmitting end, even if a clock signal is being transmitted along with the data. Then, the start of the data stream is signaled by sending one or more SYN (synchronous idle) characters, which alerts the receiver to incoming data.

Next, one or more blocks of data are continuously sent. The data consists of characters 5 through 8 bits long with an optional parity bit, as with asynchronous methods. Often the data is encoded as ASCII characters, although it could also be EBCDIC (a code supported by IBM). Each block of data ends with an error-checking character which provides much better data integrity than each character's parity bit. A popular error-checking technique used here (and in many other applications) is the *cyclic redundancy check* (CRC). The CRC takes the binary value of all the bits in the block of data and divides it by a particular constant. The remainder of this division is the CRC character, which will reflect multibit as well as single-bit errors.

IBM supported bisync on original PCs with its Binary Synchronous Communications adapter. This card used an RS-232C compatible interface with a 25-pin D-shell connector. It was based on an Intel 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter) IC. All the necessary protocol parameters were programmable, including mode of operation, clock source, and time out after no activity.

The other two popular synchronous protocols are SDLC and HDLC, which are both bit-oriented techniques, where there are no character boundaries. The data is just a continuous stream of binary numbers, sent as an information field. This information field can vary from zero bytes up to the maximum allowed by the particular protocol in force. Like bisync, SDLC and HDLC data fields are framed by control information at the beginning and end. They also contain additional addressing information that makes them suitable for use with communications networks. HDLC contains more control information than SDLC. Unlike bisync, if transmission stops within an SDLC or HDLC field, an error is always assumed.

IBM supported SDLC for PCs with its Synchronous Data Link Control Communications Adapter. This card, as its BSC card, used RS-232C compatible signal levels and a 25-pin D-shell connector. It was based on the Intel 8273 SDLC Protocol Controller IC.

8.3.4 High-Speed PC Serial Interfaces

Many of the standard EIA serial interfaces we have previously discussed are still in common use, especially RS-232C. However, they have not kept pace with advances in PC speed and performance. In addition, they were developed for the world of mainframe computers and lacked the ease-of-use and standardization that PC users now expect. Newer serial standards have been developed in recent years, targeting PCs.

Universal Serial Bus The PC industry leaders (including Compaq, IBM, Intel, and Microsoft) developed the Universal Serial Bus (USB) as a replacement for standard serial and parallel ports on a PC. USB is a high-speed, multidrop serial bus with data rates as high as 12 Mbits/sec (or as low as 1.5 Mbits/sec for slower devices). It is a true bus that can support as many as 127 devices, with one host controller (the PC).

USB uses a strictly controlled wiring system that prevents erroneous connections. In addition, it can provide DC power to peripheral devices (5 V at up to 5 A) and is hot-swappable. That is, you can safely connect or disconnect USB devices from the bus without powering down or rebooting your PC. USB devices are also plug-and-play, so their driver software is self-configuring on a PC running Windows 98 or Windows 2000 (or Windows 95 if it is version 4.00950B or later). Windows NT does not support USB.

USB was designed to connect standard, slow (mouse, keyboard), and medium speed (scanner, printer) peripherals to a PC with minimal user effort. To this end, most new PCs now contain USB ports and some have eliminated the older serial and parallel ports. Eventually, mainstream PC makers will eliminate most or all internal expansion slots (PCI) and rely on USB and FireWire (see the next section) for connecting all peripherals to a PC. This is called the "closed box" strategy for the future (users will never have to open their PC to connect a new device). Industrial PCs should still retain their expansion slots and "legacy" ports for many more years.

With this trend in mind, many data acquisition manufacturers have products that connect to USB ports. Of course, because of USB's limited bus speed, most of these products work at low sampling rates, only up to about 100K samples/sec (see Chapter 11 for more information on USB data acquisition products).

USB uses a special four-conductor cable, up to 5 meters long, with a connector pinout shown in Table 8-7. Two wires, +DATA and –DATA, comprise a twisted pair carrying a differential data signal. The other two wires, VCC and GND, provide optional +5 V power to the peripherals. USB is designed for a single host device, so you cannot normally use it to connect one PC to another (as opposed to an IEEE-1284 parallel port). However, some manufacturers produce special USB cables along with custom software for this purpose (for example, to transfer data between a PC and a laptop computer).

TABLE 8-7
USB Connector Pin Assignments

PIN #	SIGNAL NAME
1	VCC
2	–DATA
3	+DATA
4	GND

A typical PC has two USB ports. If you want to connect more devices to the PC you need a hub, a special USB device that contains several additional USB ports. Figure 8-21 shows a typical USB connection scheme utilizing 5-port hubs.

Since USB uses just one differential data pair, it is asynchronous. Also, only one device can transmit at any given time (as on an RS-485 bus). Data is

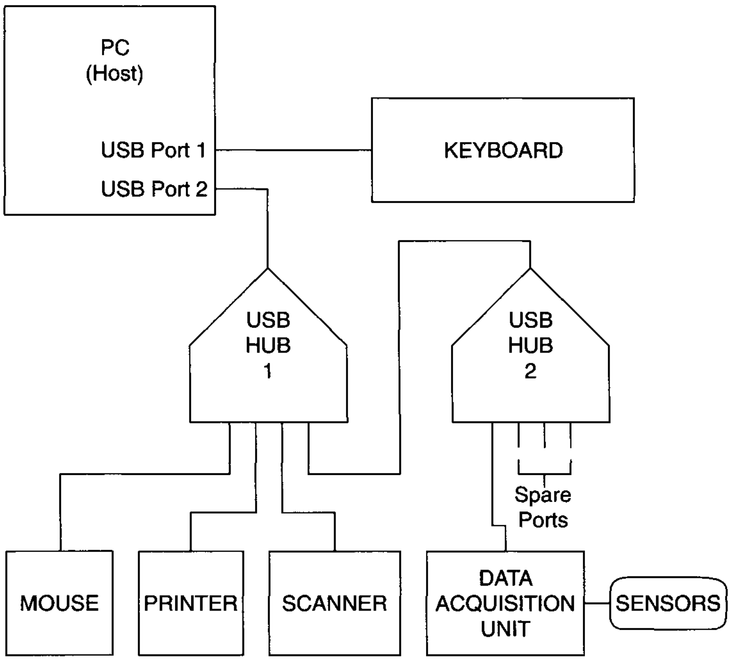


Figure 8-21 Typical USB connections to a PC.

encoded using the NRZI (no return to zero, inverted) scheme. In NRZI, a bit value of 0 causes the line driver to switch states while a value of 1 causes it to stay the same. For example, a stream of 0 bits will generate a clock signal, since there will be a transition for every bit interval. USB adds bit stuffing to NRZI to ensure that the receiver does not get out of synchronization with the transmitter if too many 1's are sent. Whenever there is a continuous stream of six 1-bits, the transmitter adds (or stuffs) a 0 bit to produce a new transition. The receiver uses the 0 bit transitions to synchronize its clock to the data stream.

USB uses a sophisticated communications protocol based on three types of packets: token, data, and handshake. The host (PC) starts a transaction by sending out a token packet that addresses the desired device. Each device on the bus has a unique address. The address field in the token packet is 7 bits long, allowing for 128 unique addresses (and the 127-device limit on the bus). Next, data is exchanged via a data packet, containing up to 1023 bits of data along with a CRC for error checking. Finally, a handshake packet is transmitted to end the transaction.

As with most technologies connected to PCs, the USB standard continues to evolve. The first USB standard in common use was version 1.1. A few years later, USB 2.0 was developed, with a 40× speed improvement—up to 480 Mbits/sec. USB 2.0 is backward compatible with the original 12 Mbits/sec USB devices and cables. USB devices will negotiate with the host to run at the highest speed allowed on that bus. USB 2.0 is directly supported by newer operating systems, such as Windows 2000 and Windows Me.

This faster USB standard, even with the overhead of its transfer protocol, can support high-speed data acquisition. Still, it is slower than a PCI interface card that uses a DMA engine to capture data. However, for the majority of general-purpose applications, USB 2.0 will be fast enough.

IEEE 1394 (FireWire) The IEEE 1394 standard defines a high-performance serial bus, originally developed by Apple Computer as FireWire. It is a peer-to-peer system as opposed to USB's host-based protocol. Two IEEE 1394 devices can communicate with each other without requiring a host computer to run the bus.

IEEE 1394 is a very high speed bus, with the original standard defining data rates of 100 Mbits/sec, 200 Mbits/sec, and 400 Mbits/sec. It uses a simple 6-pin connector with ease-of-use similar to USB. Up to 16 devices or 64 nodes can connect to a single IEEE 1394 bus, with individual cable lengths up to 4.5 meters. As with USB, it is also hot-swappable.

IEEE 1394 was designed with high-bandwidth applications in mind, such as digital video. In fact, digital video camcorders were some of the first

TABLE 8-8
IEEE 1394 Connector Pin Assignments

PIN #	SIGNAL NAME	DESCRIPTION
1	VP	Cable Power
2	VG	Cable Ground
3	TPB*	Differential Signal Pair:
4	TPB	Data on Xmt, Strobe on Rcv
5	TPA*	Differential Signal Pair:
6	TPA	Data on Rcv, Strobe on Xmt

commercial devices to use a 1394 (FireWire) interface. IEEE 1394 ports are not currently standard on most PCs, since they are more expensive than USB. However, IEEE 1394 interface cards are available for PCs from many manufacturers.

The IEEE 1394 cable consists of six conductors: two twisted pairs and two power wires. As with USB, 1394 provides power to devices on the bus. Since there is no default host node, any 1394 device can supply power. Cable power, VP, is between +8 V and +40 V relative to VG, cable ground. A device that provides power is limited to a maximum of 1.5 A. A device that uses this power initially cannot draw more than 1 W (i.e., 125 mA at +8 V). Table 8-8 shows the pinout of an IEEE 1394 connector.

As with USB, IEEE 1394 uses differential signals to transmit high-speed data reliably. The 1394 bus uses two signals (compared to only one signal for USB): TPA and TPB. These are low-voltage differential signals (LVDS) with amplitudes of only about 200 mV, to improve high-speed performance. The signals are bidirectional and tri-state capable. A device transmits data on TPB and receives data on TPA. However, when transmitting data, a device uses TPA to transmit a special strobe signal. When receiving data, TPB contains the received strobe signal.

This special signal is used to implement data strobe encoding. It is a technique that allows the receiving device to extract a stable clock with better jitter tolerance than a standard clock signal line would provide (as in typical synchronous communications protocols). As shown in Figure 8-22, for each bit interval, only one of the two signals, Data or Strobe, changes. That is, if Data changes, Strobe stays constant. When Data stays the same (because of two consecutive identical bits), Strobe will switch. The receiving device generates an exclusive-OR (XOR) of the Data and Strobe signals, producing a recreated clock.

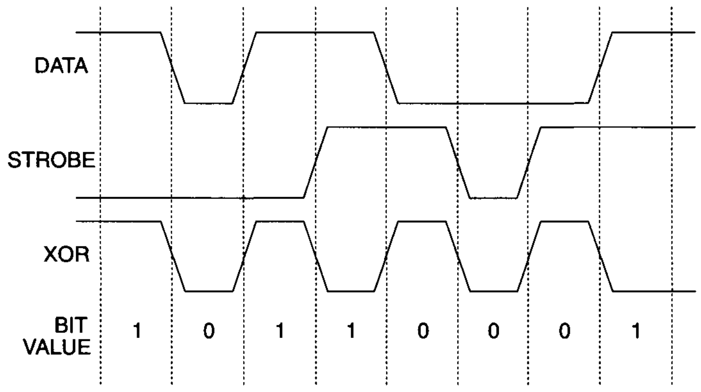


Figure 8-22 IEEE 1394 data strobe encoding.

IEEE 1394 is also a packet-based system but uses a more complex protocol than USB. When the system turns on, or whenever a new device is connected, the bus starts a configuration process. This proceeds from devices with only one connection, called leaf nodes, to those with multiple attachments, called branch nodes. The bus appears as a large memory-mapped space in which each device or node takes up a unique address range. After configuration is complete, a bus topology with a simple root node (typically a PC, if present) is determined. Now that each node has its own address, data transfers can occur.

IEEE 1394 supports two types of data transfers: isochronous and asynchronous. The bus operates using approximately 125-μsec time slices or cycles. For each cycle, devices can arbitrate to transfer a data packet. A simple isochronous transfer, which has highest priority, can use up to 80% of the available bus bandwidth (or cycle time). This transfer could be as long as 5000 bytes in one cycle if no other device is requesting an isochronous transfer for the same cycle. Isochronous transfers are suitable for time-critical, high-bandwidth data, such as digital video. Isochronous transfers are fast and virtually real-time but they do not contain error correction data nor are retransmissions available. The isochronous philosophy is that it is better to drop a few pixels in a video frame than to corrupt the frame timing and get a distorted image. Here, speed is more important than data quality. This may not be suitable for many data acquisition applications where data integrity is important.

Asynchronous transfers are not guaranteed a certain amount of bus bandwidth, but they are given a fair chance at bus access when they are allowed, later in the cycle (after isochronous transfers). The maximum size for an asynchronous data block depends on the system's transfer rate with a maximum of 2048 bytes for a 400 Mbits/sec bus. Since an asynchronous block can get sent each cycle, which is every 125 μsec, this corresponds to a maximum asynchronous rate of about 16 Mbytes/sec. Asynchronous transfers do use error checking and handshakes to allow for retransmissions, if necessary. They can be slower than isochronous transfers but are better suited for data acquisition applications where errors cannot be tolerated. Also, the IEEE 1394 uses an arbitration system that ensures all devices on the bus, regardless of transfer mode, have an opportunity to transfer data and are not locked out by high-priority devices.

The IEEE 1394 standard defines four protocol layers, as shown in Figure 8-23: the physical layer, the link layer, the transaction layer, and the serial bus management layer. The physical layer includes the connectors, cables, and electronic circuits that transmit the signals. It defines the data encoding and the arbitration mechanisms used. The physical layer is also responsible for bus initialization.

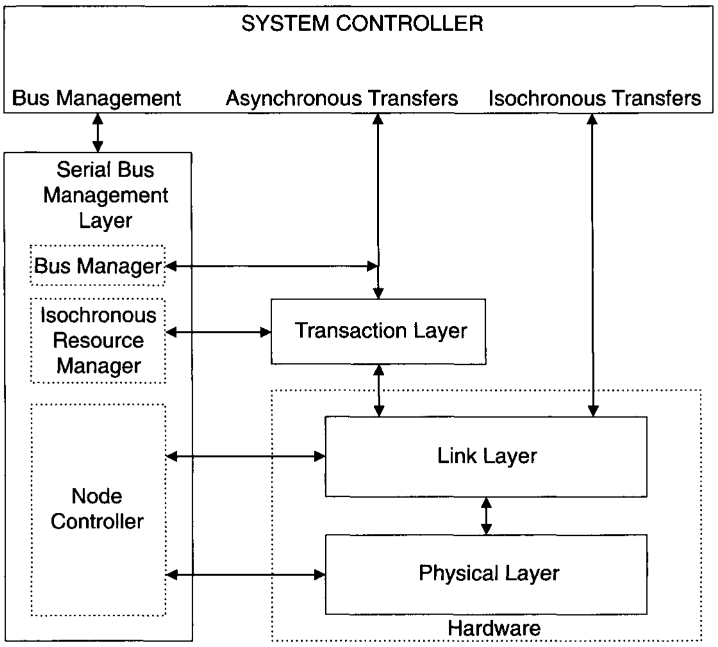


Figure 8-23 IEEE 1394 protocol layers.

The link layer sits between the physical and transaction layers. For asynchronous transfers, the link layer handles CRC checking and generation for the transaction layer. For isochronous transfers, the link layer has full responsibility for handling data transmission and reception. There are a minimum of 17 signals that make up the interface between the link and physical layers. Part of this interface, in the link layer, includes transmit/receive FIFOs, interrupt generation, and a DMA channel.

The transaction layer is only used for asynchronous transfers. It determines the size and type of the next transaction, such as read, write, or lock (write followed by a read back). The serial bus management layer handles basic control functions. Some of the bus control responsibilities are assumed by different nodes, including cyclemaster (running the 125 μsec bus cycle), isochronous resource manager (allocating isochronous transfer bandwidth), and bus manager (keeping track of bus topology, optimizing bus traffic, and managing DC power distribution).

There are several available chipsets that implement the physical and link layers in hardware. Still, IEEE 1394 devices are fairly complex to design, and this accounts for part of their higher cost compared to USB. There is some operating system software support for IEEE 1394 in PCs. Currently it is supported to varying degrees in Windows 98 (second edition), Windows Me, and Windows 2000. Since IEEE 1394 is a peer-to-peer system, it can be used as-is to connect two PCs together for high-speed data transfers.

As with USB, IEEE 1394 continues to evolve faster implementations. A newer standard, IEEE 1394b, is backward compatible with existing hardware having data rates up to 400 Mbits/sec while adding new data rates of 800, 1600, and 3200 Mbits/sec. This keeps it well ahead of USB 2.0, with a 480 Mbits/sec maximum rate. IEEE 1394b also supports long transmission line lengths, up to 100 meters using twisted-pair cables at a data rate of 100 Mbits/sec. This is still an order of magnitude faster than RS-422 or RS-485 transmissions. Data rates up to 3200 Mbits/sec are supported on glass optical fiber cables up to 100 meters long.

8.4 PC Networks

Networking PCs has become more common than ever. The ability to share data and resources, such as laser printers and plotters, has made PC networks standard in most lab, office, and industrial environments. Sharing information on a global scale, using the Internet, has opened up new realms of possibilities. For example, you can acquire data from remote data acquisition equipment using standard commercial hardware and software. Microsoft Windows has supported networking since version 3.11 (Windows for Workgroups). Starting with Windows 95, Microsoft has supported the popular TCP/IP protocol—the same software protocol used by the Internet.

To discuss networks, we will first cover the basic signaling aspects: the electrical signal characteristics and the hardware protocols used to transmit and receive data. Then we will look at software protocols. The most popular hardware/software system used to implement a local area network is Ethernet and the most common networking protocol is TCP/IP.

8.4.1 Ethernet

The Ethernet local area network (LAN) was originally developed by Xerox in the 1970s and became a published specification in the 1980s. It is a combination of hardware and software that allows different computers running different operating systems to communicate and exchange data at relatively high speeds. Ethernet is made up of four basic elements: the physical medium, the signaling components, the media access control protocol, and the frame.

The physical medium encompasses the cables and other components used to carry the signals for the network. The most popular medium for PC-based Ethernet systems is twisted-pair wiring terminated with 8-pin RJ-45 (telephone style) connectors, although coaxial and fiber optic cables are also commonly used. There are cable length limitations based on signaling speed and media type to ensure that the maximum round-trip time is not exceeded. This is the time it takes a signal to go from one end of the system to the other and back again. This timing limitation can be overcome by dividing a large LAN into multiple, separate LANs using switching hubs.

The signaling components are the electronic devices that transmit and receive data via the physical medium. These components include signal line transceivers and a computer's Ethernet interface, often residing on a PC's network interface card (NIC) or motherboard. Most PCs use a 10BASE-T or 100BASE-T NIC with twisted-pair cables (rated as Category 5 cables for 100 Mbits/sec service). Multiple PCs connect to the LAN in a star configuration through a multiport repeater hub, as shown in Figure 8-24. The repeater is used to retransmit the network signals to all its ports or network segments.

Up till now, the elements we have discussed could apply to many communications protocols, such as RS-485. The next two elements are the key to Ethernet's usefulness and popularity. The media access control (MAC) protocol is a set of rules that allows multiple computers to fairly share a channel. For example, employing coaxial cable in a multidrop configuration, Ethernet uses a half-duplex operation mode. At any given time each computer

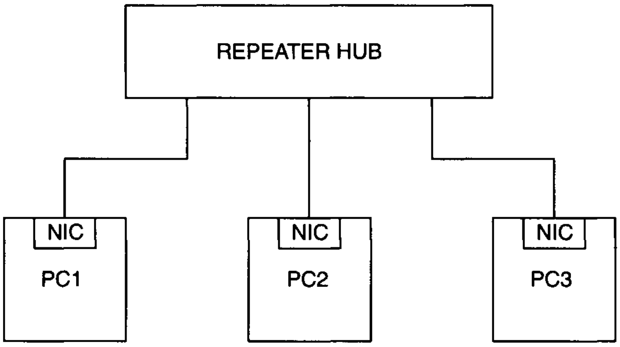


Figure 8-24 Simple Ethernet LAN employing a repeater hub.

interface can either receive or transmit data but not both simultaneously. In this case there can only be one transmitter at a time, as on a USB network. However, there is no designated host or root device in Ethernet (it is a peer-to-peer network). To allow all computers on the network a fair chance to transmit data the CSMA/CD (carrier sense, multiple access/collision detection) protocol is used. Before transmitting, each Ethernet interface waits until there is no signal on the cable or channel (carrier sense). Then, all interfaces have equal priority attempting to transmit data (multiple access). If an interface detects that other transmissions are occurring (collision detection) it stops transmitting and chooses a random retransmission time to try again. This arbitration system give all Ethernet interfaces a good chance of accessing the network. On a 10 Mbits/sec system, collisions are typically resolved within a few microseconds. Multiple collisions are only likely on a heavily loaded network (many devices transmitting data very often). Even then, Ethernet can adapt by trying different retransmission times.

The final element, the frame, is the standard packet used to carry data over an Ethernet system. Figure 8-25 shows the components of an Ethernet frame, which is the heart of the system. The frame is divided into fields, starting with a 64-bit preamble. On a 10 Mbits/sec network (such as 10BASE-T) the preamble gives the hardware time to correctly receive the rest of the frame. At faster network speeds such as 100 Mbits/sec (100BASE-T) and 1000 Mbits/sec, there is constant signaling and the preamble is not necessary.

The next fields are the 48-bit destination and source addresses. The first 24 bits of the address is an organization unique identification (OUI), assigned to individual manufacturers and organizations by the IEEE Standards Association. The remaining 24 address bits are unique for that organization (still

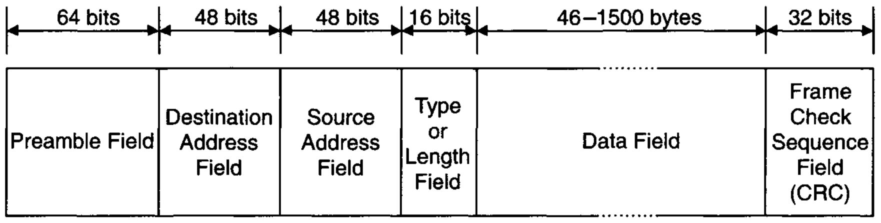


Figure 8-25 Composition of an Ethernet frame.

allowing for more than 16 million devices from a single manufacturer). The resulting 48 bits form the physical address for that interface, which is fixed in the hardware.

Next is the 16-bit type or length field. This is often used to describe the high-level protocol in use, such as TCP/IP. It is followed by the data field, ranging from 46 to 1500 bytes long. The final field is the frame check sequence, which is a 32-bit CRC. This provides the frame with data integrity, allowing receiver error detection.

If only a minimum amount of data (46 bytes) is being transferred, the frame overhead is large (approximately 36% of the total frame). Using the maximum size data field (1500 bytes) the overhead now becomes fairly small (less than 2%).

As previously mentioned, the most popular Ethernet implementations are 10BASE-T and 100BASE-T, using twisted-pair wiring. The 10BASE-T system was largely responsible for the growing acceptance of Ethernet for PCs in the 1990s. The signaling rate of 10BASE-T is 10 Mbits/sec. Of course, the actual delivered data rate depends on network loading and the amount of data contained in each frame, resulting in less than 10 Mbits/sec. 10BASE-T is a point-to-point system, as opposed to multidrop, so it needs repeater hubs to interconnect multiple computers (as previously shown in Figure 8-24). 10BASE-T uses an 8-pin RJ-45 modular connector, even though it needs only four conductors for its two differential data pairs. Table 8-9 shows the pinout for a 10BASE-T connector. The twisted-pair cable can be up to 100 meters long.

10BASE-T's differential signals are ±2.5 V and use Manchester encoding. In this scheme each bit interval has a clock transition, as shown in Figure 8-26. At 10 Mbits/sec each bit is 100 nsec wide. When the clock in the middle of the interval goes from high to low it is a 0 bit. When it goes from low to high it is a 1 bit. This way a clock is transmitted along with the data.

TABLE 8-9
10BASE-T Connector Pin Assignments

PIN #	SIGNAL NAME	DESCRIPTION
1	TD+	Transmit Data
2	TD−	Pair
3	RD+	Receive Data
6	RD−	Pair
4, 5, 7, 8	N/C	Unused

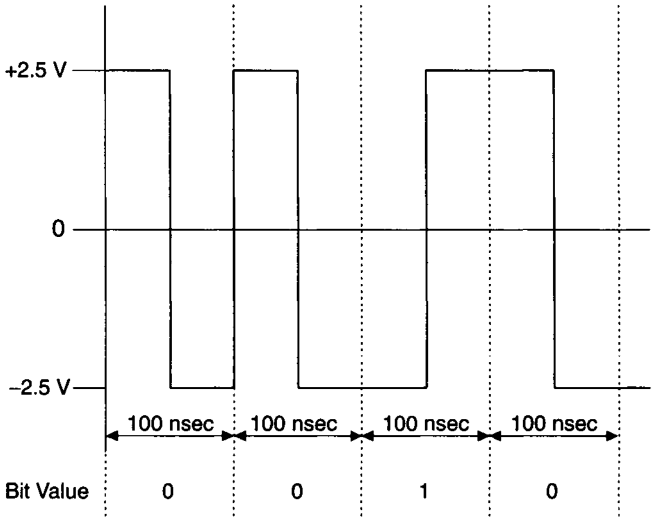


Figure 8-26 10BASE-T Manchester encoding.

Ethernet is a means of delivering a data frame across a network. To be useful, the data in that frame should be part of a high-level network protocol. This protocol controls the actual communications between computers and their application software. Ethernet is simply a messenger, unaware of high-level protocols. This allows computers running different protocols (such as NETBEUI and TCP/IP) to share the same Ethernet system.

8.4.2 TCP/IP

The most commonly used high-level network protocol is TCP/IP (transmission control protocol and Internet protocol). As with all network protocols, TCP/IP uses data packets conforming to its own standard to communicate with applications on different computers. These packets are independent of the network hardware and topology used. For example, TCP/IP packets can be transmitted just as easily using Ethernet or FireWire. Using Ethernet as a common example, a TCP/IP packet is transmitted within the data field of an Ethernet frame.

Since TCP/IP uses its own 32-bit addresses, when a computer wants to send a TCP/IP packet using Ethernet it knows the TCP/IP address of the destination computer but not necessarily its Ethernet address. Using TCP/IP's address resolution protocol (ARP), the source computer can broadcast a request over the Ethernet LAN for the computer with the desired TCP/IP address to respond with its Ethernet address.

The basic TCP/IP architecture is a series of layers and components that make up these layers, collectively called the TCP/IP stack. Every layer in the stack receives frames of data from the layer below it and sends frames to the layer above. Figure 8-27 shows a simplified TCP/IP stack using Ethernet as the physical layer.

The physical layer is the actual network hardware and control protocol, such as Ethernet, which has its own physical address. The data link layer isolates the software layers above it from the hardware. This layer handles

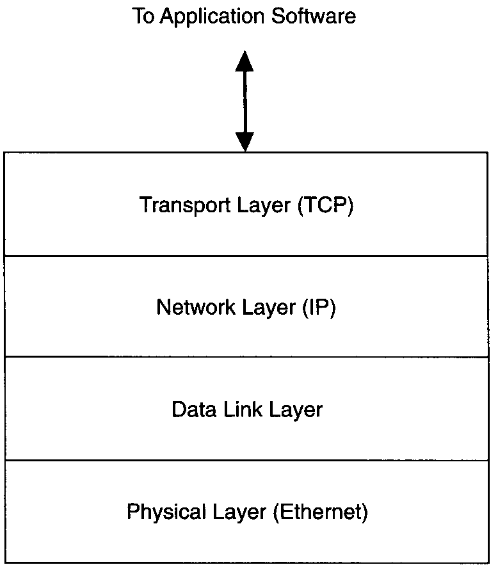


Figure 8-27 Basic TCP/IP stack.

the details of the TCP/IP frames. The network layer handles TCP/IP addressing and routing protocols. The transport layer controls the features required for reliable, sequenced packet delivery. This includes retrying and sequencing the packets to correct for any information lost at the lower layers. TCP/IP assumes that the data link layer and physical layers are not necessarily reliable and adds its own error recovery features.

This section has been just a brief introduction to networking technologies commonly used with PCs. For greater details, the reader is encouraged to see the appropriate references listed in the bibliography.

This concludes our survey of common computer interfaces and protocols used by PCs. In the next chapter we will look at data storage on the PC as well as data compression techniques.

CHAPTER 9

Data Storage and Compression Techniques

Acquired data must be permanently stored by a PC to allow future retrieval for display and analysis. The conventional storage devices available for PCs use magnetic or optical media. Most of the general-purpose storage devices (magnetic disk drives) use a random access, file-based structure. Magnetic tapes, for archiving (backup) applications, use a sequential structure.

Since most application software, including data acquisition programs, assumes data is stored on a magnetic disk (either a floppy diskette or a hard disk), these are the storage devices we will consider here. Furthermore, we will only consider MS-DOS and Windows files in this discussion, although many of the basic principles covered will apply to other operating systems and non-80x86 computers.

9.1 DOS Disk Structure and Files

A file is a logical grouping of data, physically located on a magnetic disk or other permanent storage medium, such as a CD-ROM. The physical structure of a magnetic disk consists of concentric rings, called cylinders, and angular segments, called sectors, as shown in Figure 9-1. In addition, hard drives may consist of multiple platters (more than one physical disk in the drive package). The cylinder on a single surface of a disk is referred to as a track. The read/write sensor used in a disk drive is the head. A double-sided floppy drive has two heads (one for each side of the diskette). A hard drive with four

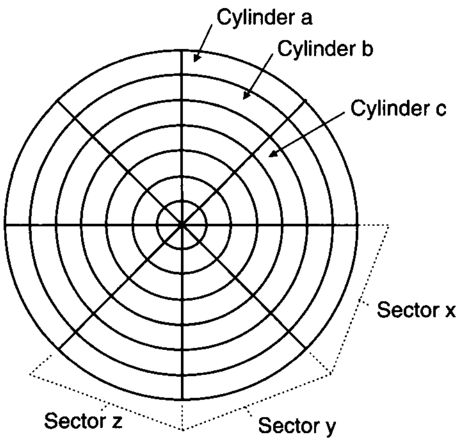


Figure 9-1 Physical organization of a magnetic disk surface.

platters has eight heads. The read/write heads usually move together as one unit, so they are always on the same sector and cylinder (but not the same side of the platter or disk). Therefore, a physical location for data on a disk is specified by cylinder, sector, and head number.

The physical structuring of a disk into cylinders and sectors is produced by the DOS FORMAT program (or the FORMAT command in Windows). In addition, FORMAT also initializes a disk's logical structure, which is unique to DOS or Windows. Each sector on every disk track (or cylinder) contains 512 bytes of data, along with header and trailer information to identify and delineate the data. This is why a formatted disk has lower storage capacity than an unformatted disk. The first sector (on the first cylinder) of every formatted DOS disk is called the boot sector. It contains the boot program (for a bootable disk) along with a table containing the disk's characteristics. The boot program, which is small enough to fit within a 512-byte sector, is loaded into memory and executed to begin running the operating system (DOS).

The boot sector is immediately followed by the file allocation table (FAT). The FAT contains a mapping of data *clusters* on the disk, where a cluster is composed of two to eight sectors (or more, depending on the operating system and hard drive). A cluster is the smallest logical storage area used by DOS or Windows. For floppy disks, a cluster is usually two sectors (1024 bytes); it is larger for hard drives. The FAT contains entries for all the logical clusters on a disk, indicating which are used by a file and which are unusable (because of errors discovered during formatting). Each FAT entry is a code, indicating the status of that cluster. If the cluster is allocated to a file, its FAT entry points to the next cluster used by that file. The file is represented by a chain of clusters, each one's FAT entry pointing to the next cluster in the file. The last cluster in a file's chain is indicated by a special code in its FAT entry. This structure enables DOS (or Windows) to dynamically allocate disk clusters to files. The clusters making up a particular file do not have to be contiguous. An existing file can be expanded using unallocated clusters anywhere on the disk.

Because of the way file clusters are chained, a corrupted FAT will prevent accessing data properly from a disk. That is why DOS usually maintains a second FAT on a disk, immediately following the first one. This second FAT is used by third-party data recovery programs (and in later versions of MS-DOS, by SCANDISK) to "fix" a disk with a damaged primary FAT. Another side effect of the dynamic cluster allocation ability of DOS (and Windows) is that heavily used disks tend to become *fragmented*, where clusters for most files are physically spread out over the disk. This slows down file access, since the read/write heads must continuously move from track to track to get data from a single file. Several commercial utility programs are available to correct this, by moving data clusters on a disk to make them contiguous for each file and thus decrease file access time. Later versions of DOS and Windows contain a DEFRAG program to do this.

The FAT (and its copy) on a DOS disk is followed by the *root directory*, which contains all the information needed to access a file present on that drive. This information is the file name and extension, its size (number of bytes), a date and time stamp, its starting cluster number, and the file's attributes. The root directory is a fixed size (along with each file entry) so that DOS knows where the disk's data area, immediately following the root directory, begins. This limits the number of files that can be placed in the root directory. For example, an old 360-Kbyte, double-sided 5-1/4" floppy disk can only keep 112 entries in its root directory (which consists of four clusters of two sectors each). If more files must be stored on this disk, subdirectories have to be used. A subdirectory is a special file that contains directory information. It is available starting with DOS 2.0 and is used to organize groups of files on a disk. It is especially useful with large storage devices, such as hard drives.

Hard disks have one additional special area, besides the boot sector, the FAT, and the root directory. It is called the partition table. The information in the table describes how the hard disk is partitioned, from one to four logical drives. This information includes whether a partition is bootable, where it starts, its ID code (it can be a non-DOS partition for another operating system, such as UNIX), and where it ends. To get around the disk size limitation of 32 Mbytes in versions of DOS prior to 4.0, it was necessary to partition large hard disks into smaller logical drives. This was usually done with a special utility software package, or via the DOS FDISK program.

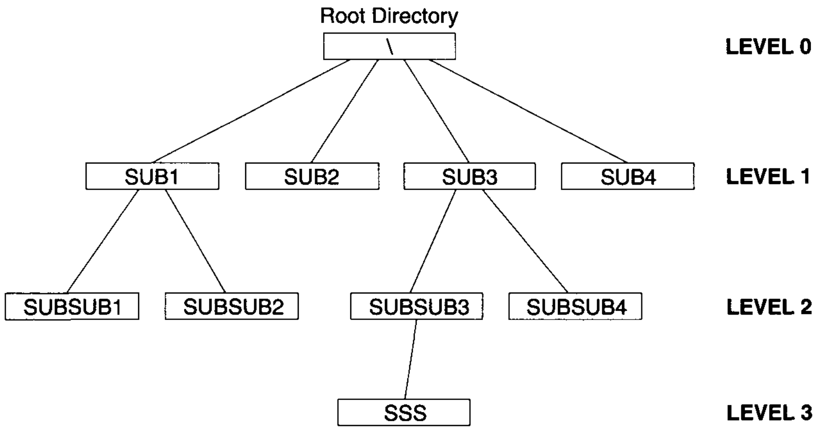


Figure 9-2 Example of DOS directory structure.

The directory structure of a DOS disk can be described as an inverted tree diagram, as illustrated in Figure 9-2. The root directory is symbolized by the backslash (\) character. The root has a limited number of possible entries that can be either standard files or subdirectories. A subdirectory is a variable-size file (as are all DOS files), so its size and maximum number of entries is only limited by the free storage space available on the disk. Each subdirectory can contain conventional files along with other subdirectories. You can keep adding level after level of subdirectories. In Figure 9-2, the top level (Level 0) is the root directory, present on all DOS disks. Level 1 contains the first level of subdirectories (Sub1, Sub2, Sub3, Sub4), along with their files. Level 2 contains the subdirectories of Sub1 (Subsub1, Subsub2) and Sub3 (Subsub3, Subsub4). Level 3 contains the subdirectory of Subsub3 (SSS). Note that subdirectory names are limited to eight characters, as are all file names. However, subdirectory names do not use a three-character extension, as other files do.

To access a file via DOS, the path to the directory containing that file must be specified, usually starting from the root (if the root is not explicitly shown, the current default directory is assumed). In that path, directory levels are separated using the backslash (\) character. For example, \SUB1\SUBSUB2 would be the path to the SUBSUB2 directory. A \ character is also used to separate the directory path from the file name. For example, \SUB3\SUBSUB3\SSS\DATA.001 would be the complete file specification allowing DOS to locate the file DATA.001 in subdirectory SSS.

It should be noted that each directory level used on a disk requires DOS to search an additional subdirectory file to locate and access the file requested.

If many directory levels are used (such as greater than five) DOS file access will be considerably slowed. You should use directories to organize your file storage logically, especially with a hard drive. Just do not use more levels of subdirectories than you need.

For instance, you might have a hard disk subdirectory containing your data acquisition programs, called \ACQUISIT. You should keep your data files organized by projects or experiments, and separated into subdirectories, such as \ACQUISIT\PROJ1, \ACQUISIT\PROJ2, etc. However, there is no need to put each data file from the same project into its own subdirectory (\ACQUISIT\PROJ1\TEST1, ACQUISIT\PROJ1\TEST2) unless they all have the same name. So, \ACQUISIT\PROJ1 may contain TEST1.DAT and TEST2.DAT.

9.2 Common DOS File Types

Standard DOS and Windows file types are denoted by a three-letter extension to the file name. We previously saw that .SYS files are loadable DOS drivers, for example. DOS and Windows files can be broken down into two broad categories: binary files and ASCII files.

In a binary file, data is stored in an unencoded binary format, just as it would appear in system memory. The end of a binary file is determined strictly from the file length recorded in its directory listing. Executable programs and device drivers are example of the many types of standard binary files. Many data file formats are binary.

In an ASCII or text file, the data is stored as printable ASCII characters (see Chapter 8 for a discussion of the ASCII code). Each byte represents one ASCII character that is either printable or a special control character. The ASCII data is usually terminated by a control character signifying the end of the file. The file's directory listing still contains its file length. Various application programs, such as editors and word processors, typically operate on ASCII data files. We will now look at some of the standard DOS file types, some of which are also common Windows file types.

9.2.1 .BAT Files

Under DOS, file names ending with the .BAT extension are considered *batch* files. A batch file contains DOS commands that will automatically run, as if they were a program. Batch files have some rudimentary program capabilities, such as branching and conditional execution. For the most part, they are used to automate a group of commonly executed DOS commands, including calling application programs.

.BAT files are always ASCII files. They are usually created with an editor program, such as EDIT (part of DOS) or NOTEPAD (part of Windows). As an example, let us assume we want to copy all the files with a .DAT extension from a hard disk (drive C:) directory \TEMP to a floppy disk (drive A:) and then delete the original files. We can create a file named TRANSFER.BAT, with the following lines:

```
COPY C:\TEMP\*.DAT A:
DEL C:\TEMP\*.DAT
```

These instructions will be carried out by DOS when we give the TRANSFER command (which executes TRANSFER.BAT). Note that a batch file is an interpreted program. DOS reads each ASCII line and then executes it. Therefore, it is relatively slow compared to performing the same function with a dedicated, compiled program.

A useful feature of DOS batch programs is the ability to employ variable data, which are ASCII strings. The contents of the variables used are specified at run time, when the batch file is executed. When the batch program is written, a percent sign (%) followed by a digit is used to represent the appropriate parameter supplied with the command to run the batch file (%1 is the first parameter, %2 is the second, and so forth). Using this feature, we can make TRANSFER.BAT more generalized, with the data files name in \TEMP becoming a variable:

```
COPY C:\TEMP\%1 A:
DEL C:\TEMP\%1
```

To use this batch program to transfer all the .DOC files from C:\TEMP to A, use the command

```
TRANSFER *.DOC
```

Batch files become more than just a list of commands when conditional statements are used. The following example is a file called HIDE.BAT, which changes a file's attribute to hidden, via the DOS ATTRIB command (with the +H option). The variable parameter (%1) is the name of the file to hide:

```
ECHO OFF
IF EXIST %1 GOTO OK
ECHO "SYNTAX: HIDE <file name>"
GOTO END
:OK
ATTRIB +H %1
:END
ECHO ON
```

The ECHO OFF command tells DOS not to display the batch program lines as it executes them (normally it would). At the end of the program, ECHO ON turns this feature back on. The second program line checks to see if there was a valid file name given with the batch file command, via IF EXIST. If there was, execution jumps to the label *:OK* to execute the ATTRIB +H command. Otherwise, it displays the quoted text in the ECHO command (showing the proper syntax for the batch program) and jumps to the label *:END*, skipping the ATTRIB +H command.

One special batch file used by DOS is called AUTOEXEC.BAT. This file is executed by DOS after it boots up, if it exists in the root directory of the disk. It is used to perform many initialization functions such as customizing system parameters (i.e., changing the DOS prompt), calling an application program needed at system startup (such as starting a network driver), or changing the default directory. Windows will also run the AUTOEXEC.BAT file.

Batch files can handle fairly complex tasks, but are best suited for simpler, commonly performed functions that do not warrant the time and trouble needed to develop a full-fledged program. The minimum functionality of the DOS batch facility also limits the tasks that can be performed by a batch file. In general, if you continuously repeat the same sequence of DOS commands, that sequence is a good candidate for a batch file.

9.2.2 .TXT and Other ASCII Files

Many file extensions are commonly associated with ASCII files, although they are specified by application programs rather than by DOS itself. For example, .TXT and .DOC are common ASCII file types in DOS. In Windows, files with specific extensions are explicitly associated with particular applications: for example, .TXT files are usually associated with the text editor NOTEPAD. Even when ASCII data is used by an application it is not always "plain vanilla" (exactly following the 7-bit ASCII code). Some word processing application programs mix ASCII with binary data in their files. Others use the eighth bit of each character for special text formatting commands (such as underlining), which ASCII does not directly support.

The DOS TYPE command displays an ASCII file on the video display. If the displayed text appears garbled or has nonalphanumeric characters (such as smiling faces), the file is not composed of plain 7-bit ASCII characters.

IBM BASIC and GW BASIC produced program files with the .BAS extension. These files were usually modified ASCII, using special characters, called tokens, to represent common BASIC commands. BASIC could save its program files in plain ASCII, if specifically instructed. BASIC also produced ASCII data files that could be used by a variety of application programs.

Many data acquisition and analysis programs will read or write ASCII data files. This is very useful, since the data can be directly printed and easily reviewed by different people or imported into another data processing application, such as a spreadsheet.

9.2.3 .COM Files

DOS files with the .COM extension are executable programs in a binary format. A .COM file contains a short program that must fit within a single 64-Kbyte memory segment, including all its data. The .COM file contains an absolute *memory image* of the program. The contents of the file are identical to the computer's memory contents when the program is loaded.

When the command to run a program is issued, either by the user at the DOS prompt or from another program via the DOS EXEC function call, DOS determines whether enough free memory exists to load the program. If not, it returns an error message. If there is adequate space, DOS determines the lowest available memory address to use. This memory area is called the program segment. At the beginning of the program segment (offset 0) DOS creates the program segment prefix (PSP) control block. The program itself is then loaded into memory at offset 100h of the program segment, since 256 bytes are reserved for the PSP. The PSP contains information needed to execute the program and return to DOS properly. After the program is loaded into memory, it begins execution.

A .COM program is automatically allocated all of the available system memory. If the .COM program wants to run another program without terminating itself first, via the DOS EXEC function call, it must first deallocate enough memory for this secondary program. Even though a .COM program must fit within a single 64-Kbyte memory segment, it can access memory outside of its segment by changing its segment pointers (such as the data segment pointer, DS).

Another idiosyncrasy of .COM programs is that they must begin execution at offset 100h of their segment (immediately following the PSP). Since most .COM programs are written in Assembler, to minimize their size, they would have the following statement, just prior to the start of the program code:

```
ORG  100H
```

This requirement is not a severe limitation, since the first program statement can be a jump to some other section of code in the segment.

9.2.4 .EXE Files

The second DOS format for executable programs is the .EXE file, which is another type of binary file. This format is also used under Windows. Programs in the .EXE format tend to be much larger and more complex than .COM programs. They can span multiple segments, both for code and data. In addition, they are relocatable and the exact locations of various parts of the program are determined at execution time by DOS. Furthermore, they are not automatically allocated all available memory, as .COM programs are.

To accommodate this flexibility, DOS .EXE files begin with a special header area. The first two bytes of this header begin with 4Dh and 5Ah (in ASCII, "MZ") to indicate to DOS that this is an .EXE program. The rest of the header contains various information including the length of the program, the length of the file, its memory requirement, the relocation parameters, and where to begin program execution. Unlike .COM files, .EXE programs do not have a fixed starting point for program execution. In an .EXE file, the header is immediately followed by the program itself.

When DOS attempts to run an .EXE program, it first reads the header, determines whether enough free memory is available, creates the PSP, loads the program, and starts its execution. Because of their larger size and the extra work DOS must do, .EXE programs tend to load more slowly than .COM programs. The vast majority of commercial DOS applications are .EXE programs. Some are so large that they need more than the maximum available DOS memory area of 640 Kbytes. They typically make use of overlays to accommodate large code areas and use expanded or extended memory (when available) to handle large data-area requirements.

When a program is developed using a standard compiler (such as Macro Assembler, C, Pascal, or FORTRAN) under DOS, an .EXE file will be produced by the final linking process (see Chapter 13 for a discussion of programming languages and the various compiling processes). If the program was written to fit within a single 64-Kbyte segment, it can be successfully converted into a .COM file, using the DOS program EXE2BIN. If program file size or load time do not need to be minimized, it is not necessary to convert an .EXE program into a .COM program. When given the choice between the two executable program formats, it is usually advantageous to keep the flexibility of an .EXE program.

9.3 Windows File Systems

MS Windows, up to version 3.11 (Windows for Workgroups), used DOS for all file services. Files were accessed through the standard DOS FAT, in real mode (16-bit mode).

In MS-DOS, up to version 3.3, the FAT used 12-bit values for numbering clusters. This was referred to as a 12-bit FAT. This 12-bit value accounted for the 32-Mbyte limit DOS had as a maximum disk or partition size: the maximum number of clusters was 4096 (2^{12}), while the maximum cluster size was 8192 bytes (4096 × 8192 bytes = 32 Mbytes). Starting with DOS version 4.0, the FAT used 16-bit cluster values (it was a 16-bit FAT). This allowed the cluster size to shrink to 2048 bytes while increasing the maximum disk size to 128 Mbytes. Smaller cluster sizes make more efficient use of disk space since a cluster is the minimum amount of disk storage used by a file (or the last piece of a file).

As hard drive capacity grew, so did DOS and FAT cluster size, reaching a maximum of 32 Kbytes. This limits a hard disk (or partition) to 2 Gbytes capacity with a 16-bit FAT.

9.3.1 Windows 95 File System

Microsoft Windows 95 was the first version of Windows to abandon DOS. Windows 95 incorporated its own protected-mode (32-bit) file management system that originally used a 16-bit FAT. Using this protected-mode system, Windows no longer had to switch into real mode for file services (as in Windows 3.1 or earlier versions), which was slow and inefficient.

In Windows 95 version 950b, Microsoft changed the FAT to a 32-bit version. This 32-bit FAT can address disks as large as 2048 Gbytes (with 32-Kbyte clusters). Later versions of Windows, such as Windows 98 and Windows NT 4.0, use a 32-bit FAT. This 32-bit FAT structure is not compatible with the older 16-bit FAT. Installing it requires overwriting an entire hard disk drive and its operating system.

Starting with Windows 95, Microsoft introduced a new, layered file system architecture, referred to as the installable file systems (IFS) architecture. The IFS supports multiple file systems such as the VFAT file system and the CD-ROM file system (CDFS). The IFS allows additional file systems, such as network support components, to be added as needed.

The VFAT file system is a 32-bit protected-mode FAT that fully supports multitasking. By providing 32-bit file access and 32-bit disk drive access, VFAT significantly improves file I/O performance over MS-DOS and Windows 3.1. The CDFS is a 32-bit protected-mode file system that provides improved CD-ROM performance (compared to DOS drivers) along with multitasking support.

Figure 9-3 shows the Windows 95 IFS architecture. The IFS manager is the only interface to application software (as opposed to DOS, where a program could access disk sectors directly, via a BIOS call to INT 13h).

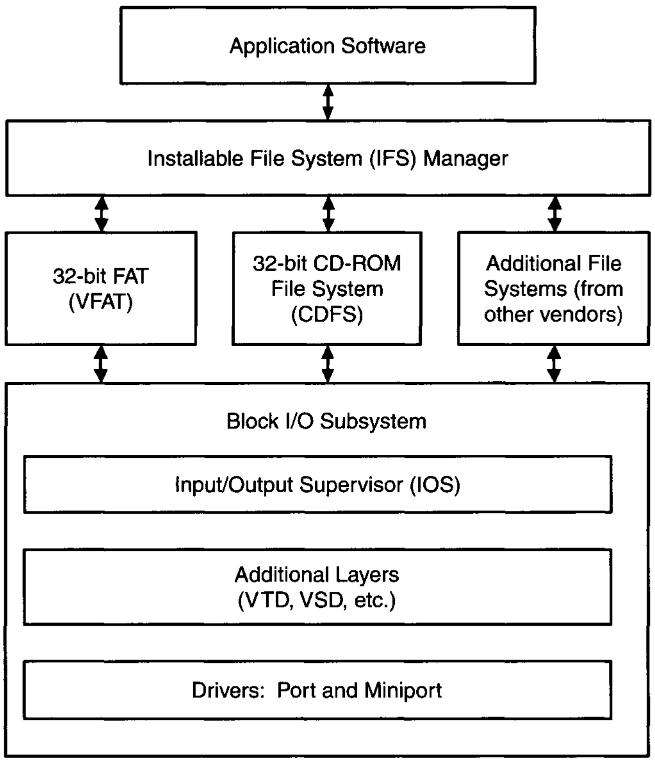

Application Software
Installable File System (IFS) Manager
32-bit FAT (VFAT)
32-bit CD-ROM File System (CDFS)
Additional File Systems (from other vendors)
Block I/O Subsystem
Input/Output Supervisor (IOS)
Additional Layers (VTD, VSD, etc.)
Drivers: Port and Miniport

Figure 9-3 Windows 95 installable file system (IFS) architecture.

Under the IFS manager are the various file systems such as VFAT and CDFS. Below the file systems is the block I/O subsystem, consisting of a series of layers that interact with the disk hardware through low-level drivers. The input/output supervisor (IOS) acts as an interface between higher layers and the file system drivers. The IOS queues file service requests and routes them to the appropriate driver.

Only 32-bit protected-mode drivers are used in the IFS. The additional layers in Figure 9-3 include the volume tracking driver (VTD), which manages removable devices (such as floppy disks), and the vendor supplied driver (VSD), which can intercept I/O requests for a particular device without having to deal with low-level (hardware) details. This is especially useful for adding special processing to disk files, such as data compression/ expansion.

9.3.2 Windows NT File System

Windows NT and its successors (such as Windows 2000) can use the same VFAT file system as Windows 95, Windows 98, or later versions. However, these operating systems also support the NT file system (NTFS), which has a different structure along with more sophisticated security features.

NTFS is based on a master file table (MFT) that stores all the information describing each file and directory on a hard drive. Each MFT entry is a record up to eight sectors (4 Kbytes) long, containing data on its associated file or directory. This data is a set of attributes that include the file name, creation date, last modification date, the type of data in the file, and so on. Each file has a unique 48-bit identification number.

NTFS uses sectors (512 bytes each) instead of clusters to allocate storage space, with 32-bit relative sector numbers to identify disk locations. This allows NTFS to access up to 2048 Gbytes ($2^{32} \times 512$ bytes) of disk space (equivalent to a 32-bit FAT) while allowing greater efficiency when storing many small files. NTFS also allows file names to be as long as 254 characters (as with Windows 95).

NTFS is organized to access data faster than FAT-based file systems while minimizing disk fragmentation. For example, when a file is opened, NTFS preallocates sectors to it, reserving a block of contiguous disk storage space (all of which may or may not be used). NTFS also places directories near the center of a disk to speed up directory searches.

Since Windows NT is designed for a multiuser, networked computing environment, NTFS supports all of NT's security features. These include controlling the rights to read, create, modify, or delete both files and directories on an individual user or group basis.

9.4 Data Compression Techniques

Data acquisition applications usually involve the creation and storage of large amounts of unprocessed data. If a particular test was acquiring 16-bit data at the modest rate of 10,000 samples/sec, 1 minute of data would require 1.2 Mbytes of storage. Ten minutes of unprocessed data would require 12 Mbytes of storage. Data at this rate could fill a small hard drive after a relatively modest number of tests. That is why data compression techniques are so important.

If large amounts of data need to be transferred between remote systems, data compression not only reduces the storage requirements for the data—it also reduces the transfer time needed (and its inherent cost). If data is being sent serially via modem, even at the relatively fast rate of 38,400 bps, it would take more than 4 minutes to transfer 1 Mbyte of data.

Many different techniques are employed to reduce the storage requirements of large amounts of data. The most important measurement of a particular technique is its compression ratio: the size of the original data divided by the size of the compressed data. Another important parameter of a data compression technique is its fidelity or distortion. This is a measure of the difference between the original data and the compressed/restored data. In many applications, no data distortion can be tolerated, such as when the data represents a program file or an ASCII document. This would call for a *lossless* compression technique. A relatively low compression ratio would be expected then. In other cases, a small but finite amount of distortion may be acceptable, accompanied by a higher compression ratio, using a *lossy* technique. For example, if the data in question represents a waveform acquired at a relatively high sampling rate, storing every other point is equivalent to filtering the waveform and producing a small amount of distortion, particularly for high-frequency components in the data.

Thus, the nature of the data dictates the parameters important to the data compression process and helps indicate which technique is best suited. The general trade-offs are between compression ratio and fidelity. An additional factor, usually less important, is the amount of time required to compress or restore the data using a particular technique. This can become an important factor if the data compression is done in real time, along with the data acquisition or transmission.

We will now look at various data compression techniques and their appropriate applications. Most of the techniques, unless otherwise noted, are primarily useful for files containing numerical data.

9.4.1 ASCII to Binary Conversion

Sometimes there are very obvious ways to reduce the size of a data file. If a set of numerical data is stored in an ASCII format (as many commercial data acquisition application programs are), encoding it directly as binary numbers could produce large space savings. For example, if the data values are signed integers within the range of ±32,767, they can be represented by 2 bytes (16 bits) of binary data. These 2 bytes would replace up to seven ASCII characters, composed of up to five digits, one sign character, and at least one delimiter character, separating values. This ASCII-to-binary conversion would produce a maximum compression ratio of 3.5:1 with no distortion. Even if the average value used four ASCII digits (1000–9999) the compression ratio would still be 3:1. After this conversion, other techniques could also be applied to the data set, further increasing the data compression.

9.4.2 Bit Resolution and Sampling Reduction

When a set of data represents numerical values, as in a waveform or data table, the number of bits used to represent these values determines the minimum resolution and the maximum dynamic range. As we saw previously, the minimum resolution is the smallest difference that can be detected between two values, which is one least significant bit (LSB) for digitized numbers. The ratio between the maximum and minimum measurable values determines the dynamic range:

$$\text{Dynamic range (in dB)} = 20 \times \log(\text{max}/\text{min})$$

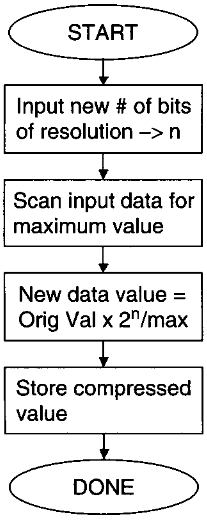
If lowering the resolution can be tolerated, data compression can be easily and quickly implemented. The resulting compression ratio is simply the original number of bits of data resolution divided by the new (lower) number of bits.

As an example, let us assume we have a set of data acquired by a 12-bit ADC system, with a dynamic range of 4096:1 or 72 dB. We first search the data set for the minimum and maximum values (we will assume the data is represented as unsigned integer values, for simplicity). In this example, the minimum value is 17 and the maximum is 483. A data range of 17–483 can be represented by 9 bits without any loss of resolution (or fidelity) for a compression ratio of 12/9 = 1.33:1. If the minimum value was larger, such as 250, the difference between maximum and minimum, now 233, can be represented as fewer bits (8, for a range of 0–255) than the full range of zero to the maximum value (483). In this case, we can get a compression ratio of nearly 12/8 = 1.5:1 by subtracting the minimum value from all the data points. The minimum value must then be included with the 8-bit data, so the correct values can be reconstructed. Adding a single 12-bit value to the compressed data is very little overhead when many points are contained in the waveform.

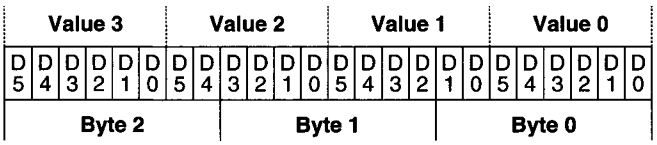
The simple technique just described is useful when the acquired data does not fill the entire dynamic range of the data acquisition system. Then, the unused bits of resolution can be discarded without causing any data distortion. Most of the time, we do not have this luxury. To highly compress a set of data we usually have to sacrifice some resolution.

Still using a 12-bit data acquisition system, let us assume a data set has a minimum value of 83 and a maximum value of 3548. Now, maximum – minimum = 3465, which still requires 12 bits of resolution. If we have to compress this data, we will lose some resolution. Assuming we need a minimum compression ratio of 1.5:1, we can normalize the data to 8 bits. To do this, we multiply all the data values by the new maximum value (255, for 8 bits) and divide them all by the original maximum value (3548). The number 3548/255 = 13.9 is the scaling factor. Either this scaling factor or the original maximum value is kept with the normalized data, to enable its restoration to the proper values and dynamic range. The data can be restored to its full dynamic range, but its resolution will be 14 times coarser, because of the rounding off that occurred when the data was normalized. Any two original data points that were separated by values of less than 14 will no longer be distinguishable. So, if two data points had original values of 126 and 131, after normalizing to 8 bits (dividing by 13.9), they will both be encoded as 9 and restored as 125.

Figure 9-4a shows a simplified flowchart for an algorithm that compresses a set of data by reducing the number of bits used to represent it. As we see, this approach can produce a loss of fine details, due to lower resolution. To exploit this form of compression, the data must be stored efficiently. Figure 9-4b shows


START
Input new # of bits of resolution -> n
Scan input data for maximum value
New data value = Orig Val x 2^n/max
Store compressed value
DONE

(a) Simplified flowchart for resolution reduction algorithm


Value 3
Value 2
Value 1
Value 0
D5 D4 D3 D2 D1 D0 D5 D4 D3 D2 D1 D0 D5 D4 D3 D2 D1 D0 D5 D4 D3 D2 D1 D0
Byte 2
Byte 1
Byte 0

(b) Data packing resulting from n = 6 bits per value

Figure 9-4 Data compression via resolution reduction.

data compressed to 6 bits per value. Four point values are stored in three data bytes, where each byte contains the bits from two adjacent values.

Another simple approach, often more acceptable than extreme resolution reduction, is sampling reduction. If the maximum frequency content of the digitized data is well below the Nyquist frequency, the effective sampling frequency can be reduced. For example, if an original set of data was filtered to limit its high end to 1 kHz, while being sampled at 10 kHz, the Nyquist frequency is 5 kHz. If every pair of adjacent values was averaged and stored, the effective sampling rate would be reduced to 5 kHz and a compression ratio of 2:1 would result. For this new set of data, the Nyquist frequency is also reduced by 2 to 2.5 kHz, still well above the maximum frequency content of the data.

This sample compression technique still distorts the data, as does the bit compression previously described. Still, if the high-frequency data artifacts lost are mostly noise, there is little harm done.

9.4.3 Delta Encoding

Another popular technique for compressing strictly numerical data is *delta encoding*. This approach is especially useful when the data represents a continuous waveform with relatively low instantaneous slopes. In such a set of data, the difference between adjacent points is small and can be represented by far fewer bits than the data values themselves. Delta encoding consists of keeping the first value of the data set at its full bit resolution, as the starting point. All subsequent values are differences, or deltas, from the previous value, using fewer bits. This is a lossless technique.

To illustrate this, Table 9-1 contains a data set of 11 original values, which require 12 bits each for full binary representation. The delta-encoded numbers start with the first, original 12-bit value. The next number is +20, the difference between the second and first values. The next delta-encoded number is +30, the difference between the third and second values. This continues until the delta between the last and next-to-last values is computed. Examining the delta-encoded numbers shows us that they all fit within the range of ±31 and can be represented by 6 bits (1 bit is for the sign). If we do use 6 bits for each delta value, the delta-encoded data set would require $10 \times 6 + 12 = 72$ bits for storage (remember, the first value is at full 12-bit resolution), compared to $11 \times 12 = 132$ bits for the original data set. The compression ratio here is 1.83:1. It will approach 2:1 as the size of the data set grows and the overhead of the first 12-bit value becomes negligible.

The key to getting high compression ratios with delta encoding is to use as few bits as possible to represent the delta values. One common problem with most data sets is that a small number of bits can represent most of the

TABLE 9-1
Example of Delta Encoding a Small Data Array

ORIGINAL VALUES	DELTA ENCODED VALUES
3125	3125
3145	+20
3175	+30
3185	+10
3193	+8
3201	+8
3192	–9
3183	–9
3170	–13
3152	–18
3130	–22

delta values, while a few deltas require many more bits, because of occasionally high local slopes or transient spikes. Instead of increasing the number of bits for delta representation to accommodate a very small number of anomalous values, an escape code can be used. Let us assume that our data set is still using a 6-bit delta representation (±31) and a delta value of +43 comes along. We can designate one of the least-used delta values as the escape code; either +31 or –31 would be a good choice. This escape code would be followed by the full-resolution 12-bit value, which cannot be represented by a small delta value. After this number, delta values continue as before. So, if we had a data set with 128 12-bit numbers, using 6-bit delta encoding that handled all but three values, the total number of bits encoded would be:

$$124 \times 6 + 4 \times 12 = 792$$

for a compression ratio of 1.9:1. If the three anomalous values could be accommodated by 8-bit delta numbers and no escape codes were used, the total number of bits would be

$$127 \times 8 + 12 = 1028$$

for a compression ratio of 1.5:1. Obviously, the judicious use of escape codes for infrequently large delta values will produce the best compression ratio. If the escape code is used too often, the compression ratio can decrease severely (it could even become less than 1:1 if a large fraction of values use the escape code).

With the appropriate data set, delta encoding can produce reasonable compression ratios with no data distortion. If it is combined with a statistical technique, such as Huffman encoding (described later), even higher compression ratios can be obtained, without any data distortion. One drawback to delta encoding, especially when used to transfer data via potentially error-prone means (such as over a telephone line via modems), is that once an error occurs in the compressed data set, all values following it will be erroneous. As with any other set of compressed or encoded data, it is always a good idea to include error detection information with the data, such as a checksum or CRC. If the block of data is large enough (for example, several hundred bytes) the overhead from the few extra error detection bytes will have a negligible impact on the overall compression ratio, while increasing the integrity of the data tremendously.

9.4.4 Huffman Encoding

Many compression techniques are based on statistical relationships among items in a data set. One of the more popular statistical methods is *Huffman encoding*. This technique will only work well if a relatively small number of data set members (possible numerical values or characters) have a high probability of occurrence. If nearly all possible values (or characters) have equal probability of occurrence (a random distribution) this method will actually produce a compression ratio of less than 1:1.

Basically, Huffman encoding employs a variable number of bits to represent all possible members of the data set. Data set members with a high probability of occurrence use the smallest number of bits (fewer than the unencoded number of bits) while those members with very low probabilities use larger number of bits (sometimes more than the unencoded number of bits). The bit values are chosen so that there is no confusion in decoding the encoded data. Huffman encoding produces no data distortion (it is a lossless technique). The restored data is identical to the original, uncompressed data. The amount of data compression produced by this technique varies with the statistical distribution of the data set used.

ASCII data representing English text is commonly compressed using Huffman encoding, since the probability of occurrence of the various alphanumeric characters is well known. Certain vowels, such as e or a, or even the space character will occur very frequently while other characters, such as x or z, will occur very rarely. The common characters may need only 3 or 4 bits to represent them in a Huffman code, while the uncommon ones may require more than 7 or 8 bits. A typical ASCII document may average around 5 bits per character using Huffman encoding. If the original data was stored as 8-bit characters, this produces an average compression ratio of 1.6:1.

TABLE 9-2
Data for Huffman Encoding Example

DELTA VALUE	PROBABILITY	HUFFMAN CODE	# OF BITS
+1	0.20	00	2
–1	0.20	01	2
+2	0.15	100	3
–2	0.15	110	3
0	0.10	1010	4
+3	0.05	1110	4
–3	0.05	10110	5
–4	0.02	11110	5
–5	0.02	101110	6
+4	0.01	1011110	7
+5	0.01	1011111	7
+6	0.01	1111100	7
+7	0.01	1111101	7
–6	0.01	1111110	7
–7	0.01	1111111	7

Huffman encoding is often used with other techniques, such as delta encoding, to further increase a data set's compression ratio. To implement Huffman encoding, the statistical probability of occurrence of each possible data set member (numerical value or ASCII character, for example) must be known. Table 9-2 shows a simple example of a set of 4-bit delta encoded values, in the range ±7. Only a few delta values have very high probabilities. Just five of the possible 15 delta values account for 80% of the data set (±1 = 20%, ±2 = 15%, 0 = 10%). In fact, a crude figure of merit can be calculated by taking this major subset of data values and dividing its total probability of occurrence (here, 80%) by the fraction of possible values it represents (in this case 5/15 = 0.33). For our example, this figure of merit is 0.80/0.33 = 2.4, which is good enough to warrant using Huffman encoding. A figure of merit below 2.0 would not be very promising for Huffman encoding.

Figure 9-5 shows a graphical method used to implement Huffman encoding. This approach is only manageable with small data sets, as in our example. The algorithm can readily be translated into a computer program for data sets with a large number of members (such as 7-bit ASCII characters).

First, we start with the data set of 15 possible values, listed in order of probability of occurrence, from Table 9-2. The data values (deltas) are listed

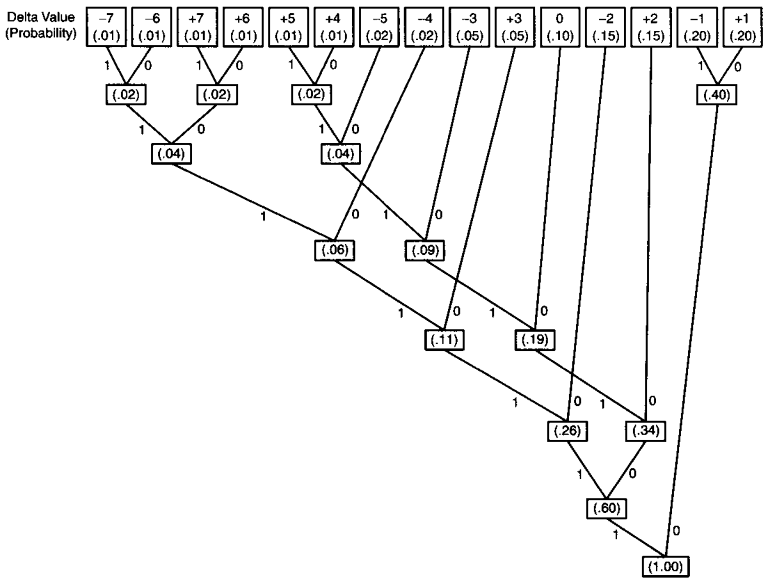


Figure 9-5 Example of graphical approach to determining Huffman codes.

across the top of the figure along with their probabilities (in parentheses), which should all add up to 1.00. To start, we draw pairs of lines connecting the lowest probability values—in this case, the .01 values at the left side of the diagram. At the vertex of the two lines connecting these pairs, we write the sum of their probabilities (.02, in this case). We continue pairing off and summing probability values, until all the values are used and the overall sum at the bottom of the diagram is 1.00.

Now, we arbitrarily assign a binary 1 to every line that points up to the left and a binary 0 to each line that points up to the right, differentiating the paths used to get from the 1.00 probability value up to the original delta value. Finally, each line connecting the 1.00 vertex to a delta value's starting point, at the top, represents a bit. We could have just as easily reversed the 1's and 0's. The code for each delta value is the concatenation of bit codes used to trace its path, starting at 1.00.

So, the Huffman code for delta value +1 is 00, and the code for –1 is 01, each only 2 bits long. The paths for delta values +2 and –2 use three lines (for 3 bits) and are, respectively, 100 and 110. All the other delta values are assigned their codes in the same way. Values 0 and +3 use 4 bits, –3 and –4 use 5 bits, –5 uses 6 bits, and all the other values use 7 bits. As we see, the delta values with the highest probabilities use the smallest number of bits.

When the encoded data is restored, the codes with the smallest number of bits are tested first. If no match is found, the number of bits tested expands, until a valid code is located. If no valid code is determined after examining the maximum number of bits, an error is assumed.

Using the Huffman codes in Table 9-2, let us see how the following encoded binary string would be decoded:

111000101111001

First, we look at the first 2 bits, 11, which are not a valid 2-bit code (only 00 or 01 are valid). Looking next at the first 3 bits, 111, we do not see a valid 3-bit code (only 100 and 110 are valid). When we check the first 4 bits, 1110, we find a valid code for +3. The remaining bits are now

00101111001

The first 2 bits here, 00, are a valid code for +1. We are now left with

101111001

Here, there are no valid 2-, 3-, 4-, 5-, or 6-bit codes. The first code to match is the 7-bit code for +4, 1011110. The remaining 2 bits, 01, are the valid code for –1. So, the decoded delta values in this 15-bit binary string are +3, +1, +4, and –1. Of course, in a practical implementation, a program would use this search algorithm.

We can calculate the average number of bits a delta entry from Table 9-2 would use when encoded this way, and hence, the compression ratio. We just sum the product of the probability times the number of bits in the Huffman code for each delta value:

$$m = p_0 \times n_0 + p_1 \times n_1 + \cdots + p_k \times n_k$$

where

m = average number of encoded bits
p_i = probability of occurrence for the ith data set value
n_i = number of encoded bits for ith data set value
k = number of values in the data set

If n is the number of bits per value in the original data set, the compression ratio is simply n/m. In our example, $m = 3.19$ bits and the compression ratio is $4/3.19 = 1.25{:}1$, which is not very large. However, since the data was already delta encoded, the original compression ratio (say, 2:1) gets multiplied by the Huffman encoding compression ratio (1.25:1) to give a larger overall compression ratio (2.5:1). Sometimes, this particular combination of compression techniques is referred to as delta Huffman encoding.

If a data set contained many more members than this previous example while maintaining a large percentage of values represented by very few members (with a large figure of merit), the compression ratio provided by Huffman encoding would be much larger. As with delta encoding, it may be useful to implement an escape code for the rare value that will not fit within the set of encoded values. In our example, it would be a delta value greater than +7 or less than –7. By its very nature, the escape code would be a very low probability code, with a relatively large number of bits.

9.4.5 Run Length Encoding

One data compression technique that is extremely useful with data sets containing large amounts of redundant information is *run length encoding* (RLE). This approach is commonly used on graphics and video data at fairly high compression ratios without producing any data loss or distortion.

In essence, RLE replaces a contiguous set of identical data values with a single count value. In video or graphics data, an image may contain large monochrome areas (such as white space) that are pixels having the same color and intensity value. Replacing a string (or *run*) of these identical pixels with a count code significantly reduces the amount of data without losing any information.

For example, a basic VGA display has an array of 640 × 480 pixels. Typically, 3 bytes (24 bits) are used to represent each pixel. So, the representation of an entire screen requires 921,600 bytes of storage.

Let us assume that in a typical VGA graphics image about 75% of the screen data are in monochrome sections (black, white, or a constant color). Also assume that on average, these monochrome areas occur as runs that are 100 pixels long in each VGA line (remember, this is a bit-mapped display, arranged as a raster scan).

We will use a unique 3-byte escape code to represent an RLE entry (instead of a pixel value), along with a 3-byte count value. To represent the encoded run, we need 9 bytes: 3-byte RLE code + 3-byte pixel value + 3-byte count value. So, our algorithm would only replace a constant-value pixel run with an RLE code if it is more than three pixels long.

In our example, 75% of the original 921,600 bytes (or 307,300 pixels) are monochrome in runs that average 100 pixels. So, the total number of these runs would be

$$\frac{0.75 \times 307{,}200 \text{ pixels}}{100 \text{ pixels/run}} = 2304 \text{ runs}$$

Since each run needs only 9 bytes to represent it, 75% of the data is compressed to 9 × 2304 = 20,736 bytes. The remaining 25% of the data (uncompressed) is 0.25 × 921,600 = 203,400 bytes. So, the total compressed data size is 203,400 + 20,736 = 251,136 bytes. This gives us an overall compression ratio of 3.67:1.

Of course, even in the nonmonochrome regions of a typical graphics display there will be some redundant information. In such a case, it is not unusual to achieve compression ratios of greater than 10:1 with RLE.

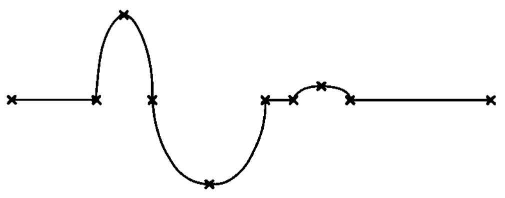
Since RLE is a lossless compression method, it can also be applied to typical data acquisition data sets if they contain large amounts of redundant information. For example, if a data set has many idle values (such as 0) in between events, they can be represented by an RLE code. RLE is often used by many general-purpose data compression software products.

9.4.6 Significant Point Extraction

Some compression techniques are used exclusively on data points that constitute a waveform. *Significant point extraction* is a generalized technique that reduces the number of points required to describe a waveform. This approach causes varying degrees of data distortion, but can provide large compression ratios (in the range of 5:1 to 10:1, for example).

Significant point extraction operates on a digitized waveform, consisting of either a one-dimensional array of amplitude (y) values acquired at known, constant time intervals or a two-dimensional array of (x, y) coordinates. The one-dimensional array is the most common form of storage for values saved by a data acquisition system. The data is analyzed point-by-point to see where a group of adjacent points can be replaced by a straight line. The discarded point values can be estimated by interpolating from this line. Only the significant points required to produce a close approximation of the original waveform are retained.

Figure 9-6a illustrates a typical digitized waveform with significant points indicated by × characters. If the original waveform was composed of 100 points, extracting only 10 significant points produces a 10:1 point compression ratio (the actual byte compression ratio will be smaller). The significant points include the waveform boundary points (start and stop) as well as places where the slope and/or amplitude change dramatically. Figure 9-6b shows the waveform reconstructed from the significant points. Note that some of the finer details are lost, while the gross waveform structures remain. The acceptability of this distortion depends on the application of the waveform data. Often, the distortion is determined quantitatively, such as by the root


(a) Original waveform with significant points noted by x

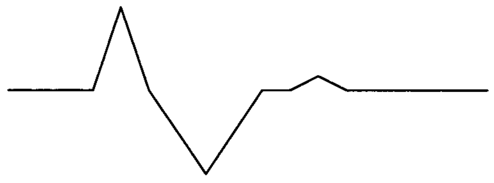

(b) Waveform reconstructed from 10 significant points

Figure 9-6 Example of significant point extraction and reconstruction.

mean square (RMS) deviation of the reconstructed data points from the original data points:

$$d = \{[(n_1 - m_1)^2 + (n_2 - m_2)^2 + \cdots + (n_j - m_j)^2]/j\}^{1/2}$$

where

d = RMS distortion
n_i = value of ith original point
m_i = value of ith restored point
j = number of points in waveform

One method of determining the significance of a point in a waveform is to calculate its *local curvature*. This is a measure of how much a waveform deviates from a straight line in the vicinity of a point. To illustrate, Figure 9-7a contains a simple waveform with one peak, composed of 23 points. To calculate local curvature, we pick a window size—in this case ±3 points—to consider the curvature around each point. If this window is too small, the calculation is not very significant. If the window is too large, local details tend to be averaged and lost ("washed out"). If the window is 2n points wide, we first start looking n points from the end of the waveform, in this case from the left side.

Since this is a one-dimensional array, the x-direction increment is constant for each point and we only need to look at data in the y direction (amplitude). For each point, number i, we do two scans from left to right.

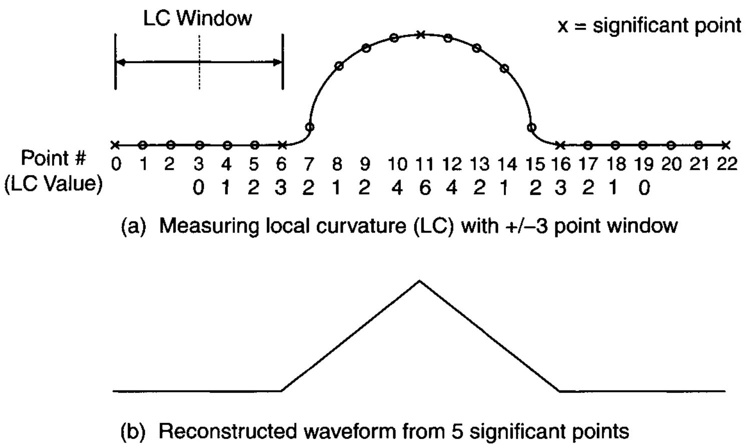


Figure 9-7 Using local curvature maxima to determine significant points.

The first scan starts at point $i - n$ and ends at point i and the second starts at point i and ends at point $i + n$. This means that we cannot scan the first or last n points in the waveform completely. For the first scan, we have two counters: $dy+$ and $dy-$. Starting with the leftmost point in the scan window, if the next point is more positive than the previous point we increment $dy+$; if it is more negative, we increment $dy-$ (if it is unchanged, we leave the counters alone). We continue with the next pair of points until we get to the end of our scan (point i). The second scan starting at point i is similar, except now if the new (rightmost) point is more positive than the previous point we decrement counter $dy+$ and if it is more negative we decrement $dy-$ (if it is unchanged we leave the counters alone). After completing the $\pm n$ points scan, the local curvature (lc) is the sum of the absolute values of these two counters:

$$\text{lc} = |dy+| + |dy-|$$

In Figure 9-7a, we cannot calculate the local curvature for points 0–2 and 20–22. Starting at point 3, after the first scan (from point 0 to point 3), $dy+ = 0$ and $dy- = 0$. After the second scan (from point 3 to point 6), $dy+ = 0$ and $dy- = 0$. So, for point 3, the local curvature is 0, or lc(3) = 0. For point 4, from the first scan $dy+ = dy- = 0$, while from the second scan $dy+ = 1$ (since point 7 is greater than point 6) and $dy- = 0$. So, lc(4) = 1. These calculations of lc continue for the rest of the waveform, up to point 19. We notice at the peak, lc(11) = 6.

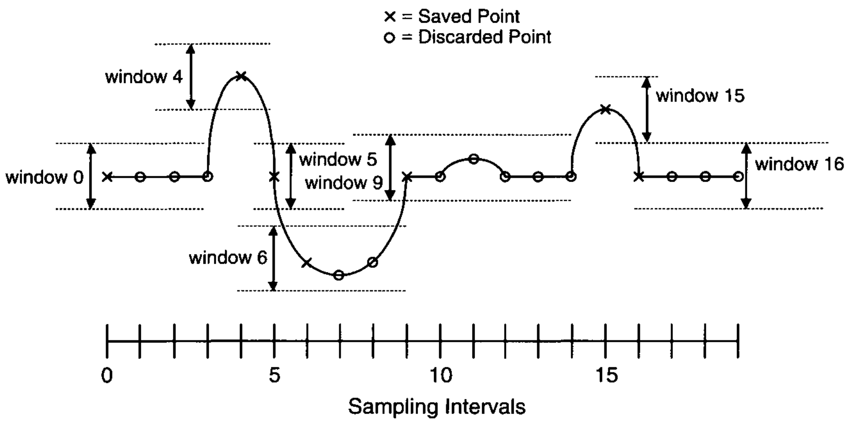
Once the lc values are calculated, we can pick the significant points as the locations of the local curvature maxima. In this example, these are points 6 (lc = 3), 11 (lc = 6), and 16 (lc = 3). We also keep the first and last points (0 and 22) of the waveform as significant, since they are the boundaries. Therefore, we have reduced a 23-point waveform to five points, for a point compression ratio of 4.6:1. Figure 9-7b shows the waveform reconstructed from these five significant points.

There are many variants on using this local curvature technique to extract significant points. A minimum threshold could be selected that maximum lc values must reach before the corresponding point is considered significant. Another approach is to use amplitude weighting in the lc calculations. The *dy*+ and *dy*– counters, previously described, produce an unweighted measure of local curvature, where a large amplitude change counts as much as a small change in the same direction. They could be weighted by the relative amount of amplitude change, not just direction. When *dy*+ and *dy*– would ordinarily be incremented or decremented by 1, they now increase or decrease by the amount of amplitude change between two adjacent points. This would help distinguish meaningful signal peaks from noise.

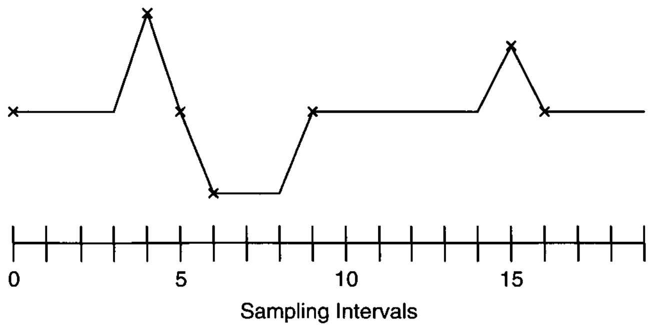
9.4.7 Predictive and Interpolative Techniques

Significant point extraction is a particular data compression method, related to the generalized techniques based on *predictors* and *interpolators*. These are algorithms that operate on waveforms or other data streams and produce compression by reducing the amount of redundancy present in that data. As long as the data set is not random, there is some correlation between adjacent data values that can be exploited. *Predictive encoding* techniques use the information contained in previous data samples to extrapolate (or predict) the value of the next data sample. This approach is used extensively in data communications systems for compressing data streams "on the fly," just prior to transmission (often using dedicated hardware). This extrapolation is done by fitting a function (or polynomial) to the existing data. Usually, only a zero-order (constant) or first-order (linear) function is used, since high-order functions tend to be very sensitive to noise and can become unstable.

The simplest extrapolation method is the *zero-order predictor* with a fixed aperture. In Figure 9-8a, a sample waveform is shown with its discrete points. Starting with the first data point, a vertical aperture (or window) of fixed amplitude, $2d$, is drawn around the first point. Additional $2d$ windows are extended over the full amplitude of the waveform. The first point is always saved, and saved points are denoted by the × character. If the next point's amplitude fits within the same $2d$ window, it is discarded; otherwise it is saved.



(a) Original waveform with sampled points (o or x) and ZOP windows



(b) Reconstructed waveform from saved points

Figure 9-8 Zero-order predictor (ZOP) used for waveform data compression.

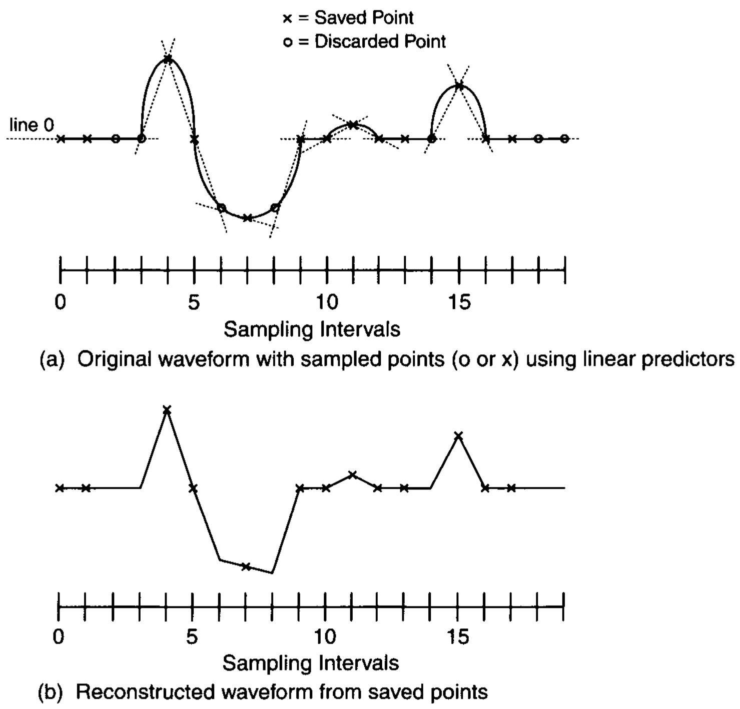
After determining a new point to save, subsequent points that fit within the new $2d$ window are discarded. Of course, the x coordinate (usually time) of the saved points must also be kept.

Figure 9-8b shows the reconstructed waveform, using only the saved points from Figure 9-8a. Notice how using a zero-order predictor tends to "flatten out" small amplitude changes. Obviously, there is a moderate amount of data distortion using this technique. However, it is useful for filtering out low-amplitude noise.

Data compression can be improved using a zero-order predictor with a *floating aperture*. Instead of the window locations being fixed by the value of the first data point, each new 2*d* aperture is centered on the last point saved. In this case, if a new point is close in amplitude to the last saved point it will always be discarded. With a fixed aperture, if this new point happened to be just over the next aperture boundary, it would be unnecessarily saved.

An approach more flexible than the zero-order predictor is the *first-order predictor* or the *linear predictor.* This is a very popular method used for many applications, such as compressing digitized human voice data. For this use, some data distortion is acceptable, since the final receiver (a human being) can still understand moderately garbled data.

Using a linear predictor is very similar to implementing a zero-order predictor, except now new data points are predicted by extrapolation from a line connecting the previous two points. Figure 9-9a shows the same sample waveform as in Figure 9-8a. The points saved by the algorithm are again marked with the character x. The first two points are always saved, to generate the first line. The following two points fit on the line, within the error window of 2*d*. They can be discarded, since a reconstruction algorithm can extrapolate



(a) Original waveform with sampled points (o or x) using linear predictors

(b) Reconstructed waveform from saved points

Figure 9-9 First-order (linear) predictors used for waveform data compression.

them from that line. The next point does not fit within the line and must be saved. A new line is drawn between this newly saved point and the previous, extrapolated point. The next point does not fit on this line and is saved, generating another line the following point does fit. This process continues, discarding points that fit (within $\pm d$) existing extrapolation lines and saving those that do not, while drawing new lines.

When the resulting saved points reconstruct the waveform in Figure 9-9b, we see that more of the fine details and curvature of the original waveform are maintained by the linear predictor, compared to the zero-order predictor. The compression ratios from both techniques are also comparable.

When data does not have to be compressed in real time, if it has been previously acquired and stored, interpolator techniques can be used. These are very similar to the predictor methods, except that now interpolation is used instead of extrapolation.

For example, using a *linear interpolator* is very similar to using a linear predictor. Using the waveform in Figure 9-10 as an example, the first point is always saved. The second point is skipped, and an imaginary line is drawn

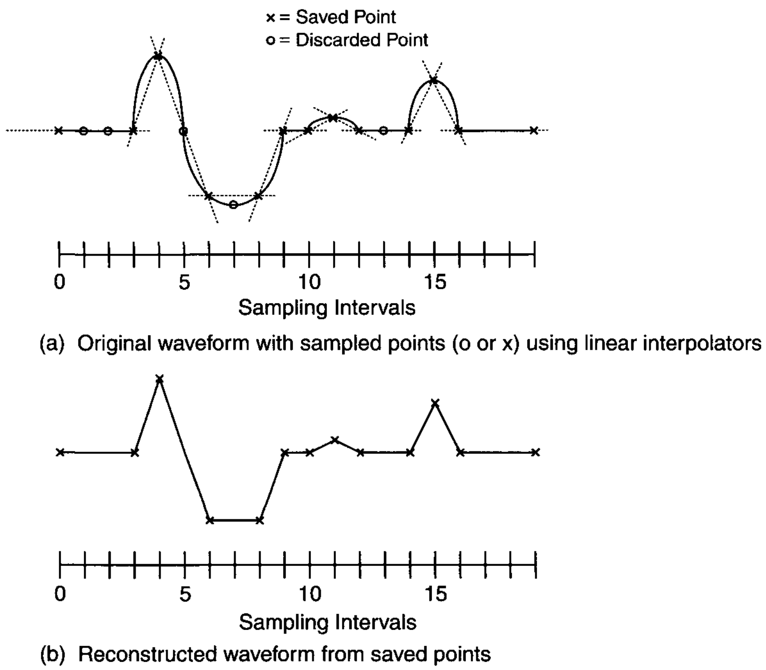


Figure 9-10 First-order (linear) interpolators used for waveform data compression.

from the first to the third point. If the second point falls on this line within a $2d$ window, it is discarded. A new line is tested between the first and fourth points. If both the second and third points fall on this line (within the tolerance window of $2d$), they are both discarded. This process continues until a line is drawn that does not fit all the intermediate points. The last point that ended an acceptable test line (the fourth point, ending the first line in this example) is saved. For data reconstruction, the intermediate, discarded points are interpolated between the two saved end points. Now, the process starts again with the end point of the last line serving as the start point for a new line. When this process is complete, at the last point in the waveform, the saved points represent the end points of interpolation lines used for reconstructing the data.

Sometimes, no intermediate points can be discarded and adjacent points are saved, especially at the peak of a curve. Since this approach requires the entire waveform to be present before processing can occur, it is not suitable for real-time compression. It is very useful for postacquisition or postprocessing applications. As with a linear predictor, a linear interpolator does produce data distortion. This can be balanced against the compression ratio by adjusting the window size. A larger window will produce higher distortion along with a higher compression ratio. Typically, an interpolator will produce a higher compression ratio than an equivalent predictor, with slightly less distortion.

Since all predictors and interpolators produce an output array of (x, y) points, they are often combined with other techniques, such as delta modulation and Huffman encoding, to reduce the total number of bits required to store the compressed waveform. The true measure of the compression ratio for the overall process is its bit compression ratio (as opposed to the point compression ratio, produced by the predictor or interpolator alone):

$$\text{Bit compression ratio} = b_o/b_c$$

where

b_o = number of bits in original waveform
b_c = number of bits in compressed data

Quite often, the optimum compression technique for a particular class of data must be determined strictly by trial and error. The data compression information in this chapter is hardly exhaustive. Certain nonlinear curve fitting techniques, such as splines, are commonly used. Fields that use extremely large data sets, such as imaging, have numerous, dedicated compression techniques producing very large compression ratios.

9.5 Commercial Data Compression Software

Many commercial data compression products are available for use on PCs. Some are hardware-based, for increasing hard disk storage without utilizing CPU overhead. Other products are strictly software-based, often used for producing hard disk file backups (as on tape systems). Since the nature of the data stored on a PC's files can vary tremendously, intelligent systems can determine the compression algorithm to use based on the data itself.

Most commercial data compression programs use lossless techniques, especially when they operate on general-purpose PC files. Several third-party applications, such as Stacker, were used to compress MS-DOS files, saving disk space. Microsoft introduced its own disk compression product, DriveSpace, as part of MS-DOS 6.22.

DriveSpace creates a virtual disk drive that contains compressed files. This virtual drive appears as a normal disk drive to the operating system. However, additional layers of software compress and restore file data during access (which does slow up I/O processes). Windows 95 uses DriveSpace 2 as its standard disk compression software while Windows 98 contains DriveSpace 3. Each newer version of DriveSpace can create a larger virtual drive, along with other enhancements.

Two popular programs that compress individual files or groups of files are PKZIP (for DOS or Windows) and WINZIP (for Windows only). They apply lossless compression algorithms to minimize file size for storage or transmission (such as via modem).

Exceptions to lossless compression of PC files are techniques applied to multimedia files. There are several popular compression standards used on audio and video files. For example, digital photographs are often stored as JPEG files, which allow for high compression ratios at the expense of reduced picture resolution (the compression–distortion trade-off is selected when a file is stored as JPEG). Audio files can be compressed using MPEG algorithms that remove inaudible information to produce high compression ratios.

This concludes our look at PC file storage and data compression. In the next chapter we will examine some common processing and analysis techniques applied to acquired data, along with considerations of numerical representation and precision.

CHAPTER 10

Data Processing and Analysis

The power and flexibility in using a PC as a data acquisition platform is shown most clearly by how data can be manipulated once it is acquired. In this chapter we will explore some of the data analysis and processing techniques commonly used with data acquisition systems. Since most data collected by data acquisition systems is numeric, it is important to know how numbers are represented and manipulated on a computer. We will start by looking at numerical representation and storage in a PC.

10.1 Numerical Representation

As we previously touched on while discussing ADCs and DACs, there are many possible ways to represent conventional decimal numbers in a binary format. The simplest of these are integer representations. For nonintegral numbers, various fractional formats can be used, though for maximum flexibility and dynamic range, floating-point representations are preferable.

10.1.1 Integer Formats

The fastest and most efficient way to manipulate data on a PC is to store it in an integer format. An integer can either be signed (representing both positive and negative numbers) or unsigned (positive numbers, only). The maximum dynamic range of the values that can be represented is determined by the number of bits used. Therefore, n bits can represent 2^n numbers with a dynamic range (in dB) of $20 \log_{10}(2^n)$. If $n = 8$, then 256 different integers can be represented: positive integers in the range 0 to 255, or signed integers

TABLE 10-1
Integer Formats

INTEGER TYPE	# OF BITS	SIGNED VALUES	UNSIGNED VALUES
Byte	8	−128 to +127	0 to 255
Word	16	−32,768 to +32,767	0 to 65,535
Long Word	32	-2.14×10^9 to $+2.14 \times 10^9$	0 to 4.29×10^9
Double Word	64	-9.22×10^{18} to $+9.22 \times 10^{18}$	0 to 1.84×10^{19}

in the range −128 to +127. This corresponds to a dynamic range of 48 dB. If $n = 16$ bits, 65,536 values can be represented, for a dynamic range of 96 dB.

The standard integer formats commonly used on a PC are byte (8 bits), word (16 bits), long word (32 bits), and double word (64 bits), as shown in Table 10-1. On an Intel 80x86/Pentium family PC, data is addressed on a byte-by-byte basis. The starting memory address for a word (or long word) is the first of the 2 (or 4) bytes comprising that word. The first (addressed) memory location contains the least significant byte (LSB), while the last location contains the most significant byte (MSB), as illustrated in Figure 10-1.

This byte ordering is processor-dependent. On a computer based on a Motorola 68000 series CPU, such as an older Apple Macintosh, a different storage arrangement is used. All words must start at an even address with the MSB at the starting (even) address and the LSB at the higher (odd) address. For a long word, the high-order 16 bits are stored at the starting (lower) address and the low-order 16-bits at the higher address (start +2).

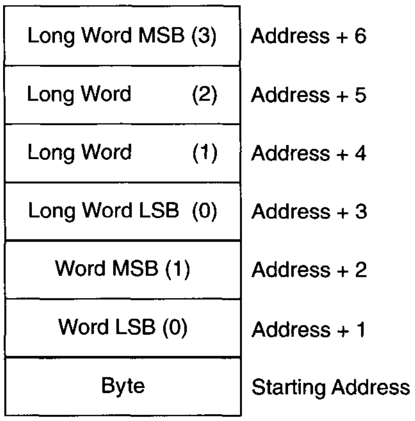


Figure 10-1 Multibyte integer storage in Intel-based PC memory.

Most of the time, the method used by a CPU to store and access data in memory is transparent to the user and even the programmer. It only becomes an issue when one data storage element, such as a word, is also accessed as a different element, such as a byte. Because of the strong likelihood of error in doing this, it is not a recommended approach. For a program written in C (see Chapter 13), if you explicitly use a casting technique, you can safely convert one element size to another.

The nature of data storage depends only on how many bytes are needed to represent a particular data storage element. An unsigned integer is usually represented as a natural binary number, such as 25 = 11001. If an element is a signed integer, there are several ways to encode or represent it. The most popular approach is to use twos-complement representation, as shown in Table 10-2. In twos-complement notation, the most significant bit is a sign bit. If it is 0, the number is a positive integer, with the same value as its unsigned binary counterpart. If the sign bit is 1, the number is negative.

TABLE 10-2
Four-Bit Signed Integers

DECIMAL VALUE	TWOS-COMPLEMENT BINARY CODE
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
0	0000
−1	1111
−2	1110
−3	1101
−4	1100
−5	1011
−6	1010
−7	1001
−8	1000

The twos-complement value is calculated by first writing the binary value of the corresponding positive number, then inverting all the bits, and finally adding 1 to the result. To get the 4-bit twos-complement representation of −4, we start with the unsigned binary value for +4 = 0100. When we invert all the bits, we get 1011. Adding 0001 to this number produces the final value of 1100 = −4. Using twos-complement representation for negative integers is widely accepted because if you add corresponding positive and negative numbers together, using this system, you will get a result of zero when truncated to the original number of bits. So, adding −4 to +4, we get

$$\begin{array}{r} 1100 \\ +\,0100 \\ \hline =0000 \end{array}$$

The use of twos-complement representation produces the n-bit signed integer range of $-2^{(n-1)}$ to $+2^{(n-1)}-1$ (i.e., for $n = 4$ this range is −8 to +7).

Other encoding techniques are used to represent decimal integers in a binary format, besides natural binary and twos-complement. One of the more common alternatives is binary coded decimal (BCD). This code uses 4 bits to represent a decimal digit, in the range 0 to 9. It uses natural unsigned binary representation (0000 to 1001). The six codes above 9 (1010 to 1111 or Ah to Fh) are unused. To represent a decimal value, a separate BCD code is used for each decimal digit. For example, to represent the value 437:

$$\begin{array}{cccc} 437 = 0100 & 0011 & 0111 \\ (4) & (3) & (7) \end{array}$$

If only one BCD digit is stored in a byte (upper 4 bits are set to 0), it is called unpacked BCD storage. If two BCD digits are stored in a byte it is called packed BCD storage. Even using packed storage, BCD numbers require more storage than natural binary or twos-complement values. As an illustration of unsigned integers, four BCD digits (16 bits) can represent the values 0 to 9999 while a natural binary word (16 bits) can represent values 0 to 65,535. Alternatively, we only need 16 bits to represent 50,000 with an unsigned natural binary word, whereas we need 20 bits (five digits) to do the same with BCD. BCD is popular with systems processing large amounts of important numerical data, such as those used by financial institutions.

An even less efficient means of numerical representation is using an ASCII character to represent each decimal digit. In this case, 7 (or 8) bits are needed to represent 0 to 9 (as well as sign and decimal point, for nonintegers). This is about twice as inefficient as BCD representation. ASCII numerical representation is usually used strictly to store data in a format that is easy to read, print, and export to other applications (such as spreadsheets). It is usually converted into a format more convenient to use before numerical processing proceeds.

10.1.2 Noninteger Formats

Quite often, when using a computer to process acquired data, integer precision is not adequate, because of round-off errors, dynamic range limitations, or poor modeling of the measured phenomena. Several numerical formats are used to overcome this problem.

The simplest way to depict fractional values is with fixed-point representation, which is basically an extension of binary integer representation. For integer representation, using n bits, the binary number $b_n b_{n-1} \ldots b_1 b_0$ is evaluated by adding the weighted value of each nonzero bit as follows:

$$b_n \times 2^n + b_{n-1} \times 2^{n-1} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$$

where b_i is the ith bit (0 or 1). For binary fixed-point representation, both positive and negative exponents are used and a binary point appears after the 2^0 digit. For example, if we had an 8-bit number with a 3-bit fraction, it would be written as

$$b_4 b_3 b_2 b_1 b_0 \cdot b_{-1} b_{-2} b_{-3}$$

The weights for the bits following b_0, b_{-1}, b_{-2}, and b_{-3} are 2^{-1}, 2^{-2}, and 2^{-3}, respectively. The resolution of this representation is 0.125 (2^{-3}), while its range of values for unsigned numbers is 0 to 31.875, which is still the same number of values as an 8-bit unsigned integer (31.875/0.125 = 255).

When more bits are added to unsigned integers, the resolution stays the same (1) while the range of values increases. When the number of bits after the binary point in a fixed point, fractional representation increases, the resolution increases, while the range of values stays the same. This trade-off between range of values and resolution is inherent in these representations.

If we needed to increase both the dynamic range and resolution of our numerical representation, we could keep increasing the number of bits per number. The problem here is that most CPUs can perform math on only a fixed number of bits at a time. For 16-bit processors (as used in earlier PCs), if more than 2 bytes represent a number, additional instructions must be performed when executing a math function, splitting the function into multiple 16-bit operations. If we are using 32-bit integers and need to add them, we have to first add the lower 16-bit words, then add the upper 16-bit words with any carry from the previous addition. The software overhead and processing time increase quickly as we increase the size of numerical elements.

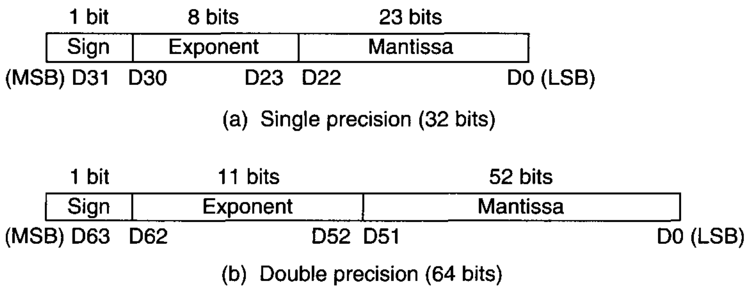


Figure 10-2 IEEE floating-point formats.

The standard solution to this dilemma is to use a floating-point format, consisting of a fractional part (the *mantissa*) and an *exponent*. The number of bits used to represent the exponent determines the floating-point number's range of values, and the number of bits used for the mantissa determines its resolution. The mantissa is a signed, binary fraction that is multiplied by 2^{exp} to produce the represented value. The exponent is a signed integer.

Certain standard formats are used to represent floating-point numbers. Among the most popular, the IEEE 754 Floating-Point Standard is also commonly used with PCs. This standard defines two formats: single-precision, using 32 bits, and double-precision, using 64 bits, as shown in Figure 10-2.

In both formats the sign bit (most significant bit) is for the mantissa, which is in a normalized form (with a value between 1.0 and 2.0). In fractional binary, this would be 1.000...0 through 1.111...1 (using a fixed binary point). Since the most significant mantissa bit (before the binary point) is always 1, it is implied and not stored with the number. So, a single-precision mantissa of 1.01101111000010101010011 would be stored as 01101111000010101010011.

The exponent is stored in a *biased* form, with a fixed value, or bias, added to it. For single-precision numbers, this bias is +127, and for double-precision numbers it is +1023. This biased exponent is useful for determining which of two exponents is larger, by comparing them bit by bit, starting with the leftmost bit. For example, consider two single-precision numbers with exponents of +15 and −5, represented as signed integers:

$$+15 = 00001111 \qquad -5 = 11111011$$

and represented as biased integers (+127):

$$+15 + 127 = 10001110 \qquad -5 + 127 = 01111010$$

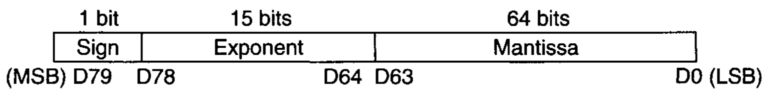


Figure 10-3 Intel 8087 80-bit temporary floating-point format.

So, just looking at the leftmost bit indicates that +15 is the larger exponent.

The valid exponent range for single-precision is −126 to +127, and for double-precision it is −1022 to +1023. When represented as a biased exponent, a value consisting of either all 0's or all 1's indicates an invalid number. This way numerical overflow/underflow errors can be indicated.

A special, non-IEEE format is used on Intel-family PCs with 80x87-style math coprocessors, the *temporary format*, shown in Figure 10-3. This is an 80-bit format, incorporating a 64-bit mantissa with a 15-bit exponent. It is very useful for highly repetitive mathematical operations where round-off errors can reduce precision, as well as calculations involving very large or very small numbers.

The temporary format uses an exponent bias of +16,383. It differs in spirit from the single- and double-precision formats by explicitly keeping the leftmost 1 in the normalized mantissa value. Since the math operations using this 80-bit temporary format are performed in hardware, the large number size does not cause severe processing speed penalties.

Table 10-3 lists decimal precision (number of significant digits) and range for some of the integer and floating-point numerical formats we have

TABLE 10-3
Range and Precision of Various Numerical Formats

NUMBER TYPE	FORMAT	TOTAL # OF BITS	EXPONENT/ MANTISSA # OF BITS	DECIMAL DIGITS OF PRECISION	DECIMAL RANGE
Integer	Byte	8	—	>2	$>\pm10^{2}$
	Word	16	—	>4	$>\pm10^{4}$
	Long Word	32	—	>9	$>\pm10^{9}$
	Double Word	64	—	>18	$>\pm10^{18}$
Floating Point	Single Precision	32	8/23	>7	$>\pm10^{38}$
	Double Precision	64	11/52	>15	$>\pm10^{307}$
	Temporary	80	15/64	>19	$>\pm10^{4932}$

discussed here. Note that for an equivalent number of bits (such as 32 or 64), floating-point formats have slightly lower precision along with much higher dynamic range than the corresponding integer formats. This is simply due to diverting some of the bits used for precision in an integer to the exponent of a floating-point value, increasing its range.

10.2 Data Analysis Techniques

A wide variety of processing techniques are commonly applied to the data produced by data acquisition systems. These can range from simply plotting the data on a graph to applying sophisticated digital signal processing (DSP) algorithms. A large number of commercial software packages, such as those discussed in the next chapter, have many of these capabilities built in. This enables the user to concentrate on the data analysis without getting bogged down in the details of programming a PC. We will begin our survey of data processing by looking at statistical analyses.

10.2.1 Statistical Analysis Techniques

The most common analysis applied to acquired data is some statistical calculation. Statistical parameters describe the distribution of values within a data set. They indicate where data values are most likely to be found as well as the probable variability between them.

The most important statistical measurement for a data set is the mean, which is simply the average of a set of values. If we have a set Y of n values, $y_1, y_2, \ldots, y_n$, the mean of Y is just

$$y_m = (y_1 + y_2 + \cdots + y_n)/n$$

The values of the data set must have some relationship to each other for the mean to have significance. For example, the set may consist of n measurements of the same quantity, repeated over time.

The conventional mean is used to analyze an existing data set. A special variation on the mean is the *running average*, sometimes referred to as the circular average or sliding average. The running average is useful for real-time control applications, when the current average value is needed. For instance, an intelligent heater controller needs to know the current temperature of a system to apply the appropriate amount of heater power. If the temperature varies significantly from reading to reading, an average of the last n readings would be useful to smooth out this temperature noise. The running average is just the mean of the last n values. If the current reading is T_i and the running average is n points wide, its value at point i is

$$T_{mi} = (T_i + T_{i-1} + \cdots + T_{i-n+1})/n$$

At the next point, $i + 1$, the running average is

$$T_{mi+1} = (T_{i+1} + T_i + \cdots + T_{i-n+2})/n$$

The running average is updated with each new value acquired. It acts as a low-pass filter on the incoming data. Only relatively slow artifacts with large amplitude changes will be reflected in the running average. When this technique is applied to an existing, acquired waveform, the n-point averaging window is usually symmetric around the selected point.

Another statistical measurement is the *median*. It is selected so that half of the data set values are higher than the median and the other half are lower. The median is often close in value to the mean, but it does not have to be.

An important measure of variation within a set of data is the *standard deviation*. If we have a data set $(y_1, y_2, \ldots, y_n)$ of n values with a mean of y_m, the standard deviation σ is

$$\sigma = [[(y_1 - y_m)^2 + (y_2 - y_m)^2 + \cdots + (y_n - y_m)^2]/n]^{1/2}$$

This is a measure of the differences between the data set values and the average value. The smaller the standard deviation, the "tighter" the distribution of data values is. In the case where all values in a data set are identical, the standard deviation would be zero.

When a data set fits a normal Gaussian distribution (a "bell" curve), approximately 68% of the values will be found within one standard deviation of the mean value. As an illustration, assume a manufacturer is interested in analyzing the length of a production part. Length measurements are taken on a sample of parts that fit a Gaussian distribution, having as its peak the mean value. Here, the standard deviation is a measure of the length variations from part to part. From these measurements, the manufacturer can predict the percentage of a production run that will fall within an acceptable tolerance. If this percentage is too small, it indicates a need to control the production process better.

10.2.2 Curve Fitting

The mean and standard deviation are mostly used on sets of values that should be describing the same or similar measurements. When the acquired data is a waveform, described as two-dimensional (x, y) points, a common requirement is comparing it to a theoretical model, or finding a model that fits the data. This data can be a one-dimensional array where time is the independent (x) variable. The theoretical model describes a waveform that should be similar to our acquired data. Finding a mathematical model that fits the measured data is referred to as curve fitting.

Very often, a polynomial is used to describe a theoretical curve. The general form of a polynomial of order n is

$$F(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

The coefficients are the constants a_i, which are adjusted during the curve fitting process. To determine the coefficient values for the best curve fit, the sum of the error terms for each of j data points in the curve is calculated as

$$[F(x_1) - y_1]^2 + [F(x_2) - y_2]^2 + \cdots + [F(x_j) - y_j]^2$$

where $y_1, y_2, \ldots, y_j$ are the measured values. When this function is minimized, the coefficients for $F(x)$ describe the best curve fit to the data set. This is referred to as the least squares fit. Using least squares to test how well a function fits a data set is not limited to polynomials. Exponential and trigonometric functions are also commonly employed for curve fitting and still use a least squares fit measurement. The iterative type of calculations used to find the least squares fit is well suited to digital computer calculations.

The simplest curve fitting is a first-order ($n = 1$) or linear fit, sometimes referred to as a linear regression. Analytically, the coefficients a_0 and a_1 are determined. Graphically, a straight line is drawn through the data points. The resulting line is described by the standard formula

$$y = mx + b$$

where m = the slope and b = the y-intercept. This means that the general coefficients $a_0 = b$ and $a_1 = m$.

Given a set of (x, y) data points, the coefficients for a linear regression can be determined analytically. The coefficients a_0 and a_1 are calculated from summations over all n points in the data set:

$$a_0 = \frac{[(\Sigma y) \times (\Sigma x^2) - (\Sigma x) \times (\Sigma xy)]}{[n \times (\Sigma x^2) - (\Sigma x)^2]}$$

$$a_1 = \frac{[n \times (\Sigma xy) - (\Sigma x) \times (\Sigma y)]}{[n \times (\Sigma x^2) - (\Sigma x)^2]}$$

For higher-order polynomial fits, analytic approaches are impractical. An iterative process of successive approximations is typically used.

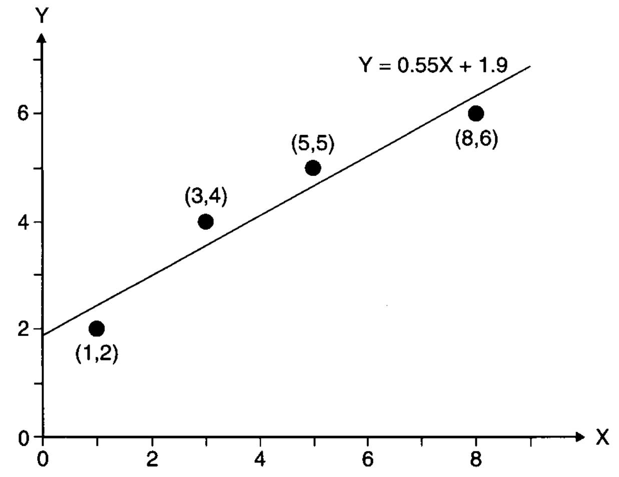


Figure 10-4 Example of linear curve fitting.

Figure 10-4 is a simple example of a linear curve fit. There are four (x, y) values: $(1, 2)$, $(3, 4)$, $(5, 5)$, and $(8, 6)$. Calculating the coefficients from the above equations, we find the least squares fit line to these points is $y = 0.55x + 1.9$.

Notice the similarity between linear curve fitting and linear predictors or interpolators, discussed in Chapter 9. In both cases, a straight line is found that best fits the data. Furthermore, minimizing the distortion produced by data compression is often a least squares process.

Curve fitting is a broad, complex field. This brief discussion should serve to give you a feel for implementing curve fitting on a PC-based data acquisition system. An advantage of using these systems (with appropriate software) is the ability to see the data graphically, along with getting the numerical processing power of a PC. When it comes to processing waveforms, seeing the data displayed as a graph is invaluable.

10.2.3 Waveform Processing

A large portion of the information gathered by data acquisition systems is in the form of waveforms (commonly, a function varying with time). These waveforms are easily displayed graphically, using many of the software packages described in Chapter 11. Very often, this acquired data is operated on as a single entity: a vector (one dimension) or an array (two or more dimensions).

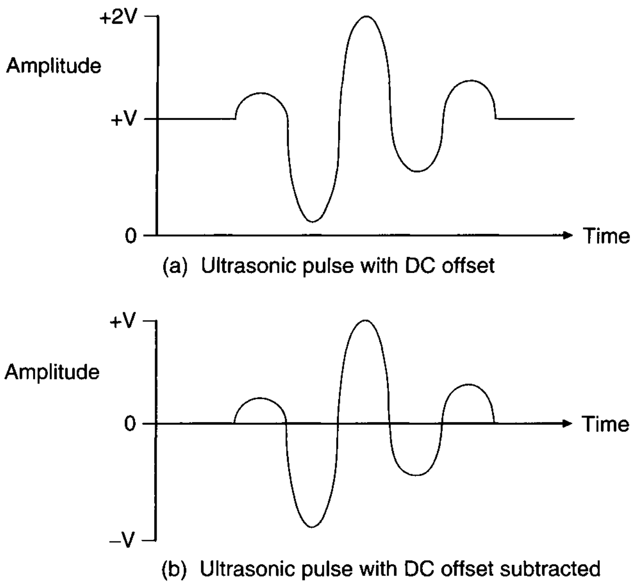


Figure 10-5 Example of subtracting a DC offset from a waveform.

Many of these operations are simple mathematical functions, such as subtraction or multiplication with a scalar or another array.

Consider the example in Figure 10-5a, a waveform representing an ultrasonic pulse, which should have a net DC component of zero. Because of DC offsets in the analog receiver system, the acquired signal may not meet this criterion. To determine the net DC offset, we take the mean value of all the waveform points. If this mean is not zero, we subtract it (a scalar) from the waveform (a vector). The result, shown in Figure 10-5b, now has a zero DC offset.

Waveforms can also be used to operate on each other. For example, special windowing functions are commonly used in DSP algorithms. Waveforms under analysis are multiplied by these windowing functions, which are also waveforms. In many cases a reference or baseline waveform is acquired. Subsequent data is then divided by this reference data, for normalization.

Other common operations are integration and differentiation. If we wish to determine portions of a waveform with high slopes, we would differentiate it. The peaks of a differentiated function occur at slope maxima. When a particular function is difficult to differentiate or integrate analytically, this numerical approach is very useful. For numerical differentiation, the slope, dy/dx, is calculated for every pair of adjacent points. In a similar fashion, the area under the curve at each point is calculated for numerical integration.

For example, suppose a waveform represented the measured displacement of an object versus time. Differentiating this waveform would produce a new waveform representing the object's velocity versus time. Differentiating a second time would produce an acceleration-versus-time waveform. Conversely, if the acquired waveform represented acceleration data, as from an accelerometer, integrating it once would produce a velocity curve and integrating it a second time would produce a displacement curve. The only problem here is that any fixed offsets in either displacement or velocity would not appear in the integrated data, as they were lost by the original acceleration measurements.

Again, this brief discussion is only scratching the surface on the topic of waveform processing. Many mathematical operations are performed on data representing vectors and arrays, such as dot products and cross products. The huge variety of waveform processing techniques find an immense range of applications. We will look at a few specialized techniques now, starting with Fourier transforms.

10.2.4 Fourier Transforms

Undoubtedly, Fourier transforms are among the most popular signal processing techniques in use today. Analytically, the Fourier series for a single-valued periodic function is a representation of that function using a series of sinusoidal waveforms of appropriate amplitude and phase. The sine waves used in the series are at multiple frequencies (harmonics) of the lowest frequency (the fundamental). The Fourier series for a periodic function, $f(t)$, with a period T would be

$$f(t) = a_0 + a_1\sin(\omega t + \phi_1) + a_2\sin(2\omega t + \phi_2) + \cdots + a_n\sin(n\omega t + \phi_n)$$

where

$\omega = 2\pi/T$, the fundamental frequency

$a_1,\ldots,a_n$ are the amplitude values for each frequency component (a_0 is the DC component)

$\phi_1,\ldots,\phi_n$ are the phase values for each frequency component

To represent a single-frequency sinusoidal wave, only the DC and fundamental frequency terms are needed. Most functions require many terms to provide a good approximation of their real value. For example, Figure 10-6

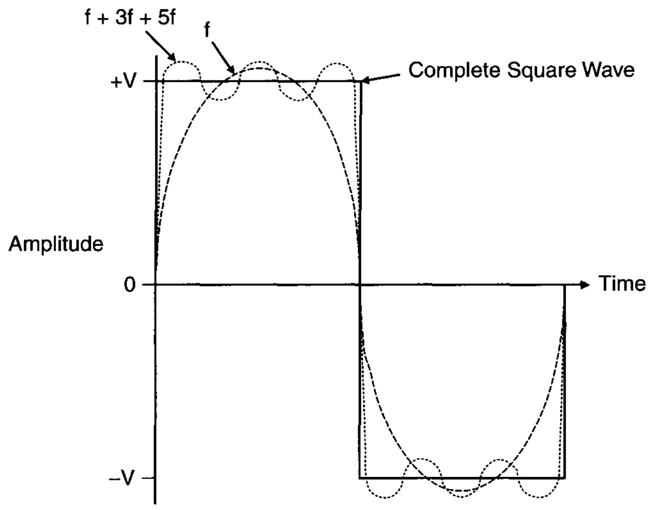


Figure 10-6 Fourier series for a square wave.

shows a square wave, which has a Fourier series consisting of decreasing odd harmonics:

$$f(t) = 4a_0/\pi\ [\sin(\omega t) + 1/3 \times \sin(3\omega t) + \cdots + 1/n \times \sin(n\omega t)]$$

Using only the first term (fundamental frequency) we only get a crude approximation of the real waveform. After we use the first three terms (up to the fifth harmonic) we have a much closer approximation of the square wave.

By fitting trigonometric functions to an arbitrary waveform, we can get the frequency content of that waveform. In essence, the Fourier transform is used to convert from a conventional data (time)-domain waveform to a spectral (frequency)-domain waveform. Since this transformation is bilateral, an inverse Fourier transform converts data back from the frequency domain into the time domain. Data-domain waveforms include functions of time as well as of space. The Fourier transform of a distance-based waveform contains spatial frequency information.

Analytically, the Fourier transform is defined for operation on continuous, periodic functions. Given a function of a real variable (the function itself can be complex), $f(x)$, its continuous Fourier transform (CFT), $F(y)$, is defined as

$$F(y) = \int_{-\infty}^{\infty} [f(x) \times e^{-j2\pi xy} dx]$$

This integral must exist for every real value of x. The complex exponential used in the integral has an equivalent trigonometric form, using Euler's formula:

$$e^{jx} = \cos(x) + j\,\sin(x)$$

where $j = \sqrt{-1}$, the imaginary number operator.

An alternative form for the CFT would be

$$F(y) = \int_{-\infty}^{\infty} f(x)[\cos(2\pi xy) - j\sin(2\pi xy)]dx$$

For data acquisition applications, a special Fourier transform is used to operate on discrete, finite functions. This is called the discrete Fourier transform (DFT) and is used to operate on discrete (digitized) data. The DFT is the workhorse of DSP techniques. If we have a waveform, $f(k)$, consisting of n points, the DFT produces a complex waveform of n points, $F(m)$. Both k and m vary from 0 to $n - 1$. The data points of $f(k)$ are evenly spaced in the time domain by dt and range from 0 to $(n - 1)dt$. The transformed data points of $F(m)$ are evenly spaced in the frequency domain by $1/dt$ and range from 0 to $(n - 1)/dt$. The DFT is calculated from

$$F(m) = \sum_{k=0}^{n-1}[f(k) \times e^{[(-j2\pi/n)mk]}]$$

The frequency-determining component is $2\pi m/n$, which is a normalized value. The DFT assumes the time-domain waveform is a periodic function, with a period of n points. The normalized frequency at the first DFT point is 0 and at the last point is $2\pi(n - 1)/n$ radians. This maximum frequency is $(n - 1)/dt$, so the time-domain sampling is normalized to $dt = n/2\pi$.

Note that the first term of the DFT, $F(0) = \sum f(k)$, at zero frequency $(m = 0)$. This is simply the area under the curve or the result of integrating $f(k)$. Also note that for each term in $F(m)$, n complex multiplications must be done as $f(k)$ times the complex exponential term [where $f(k)$ can be either real or complex]. It is a fair assumption that the amount of time required to calculate a DFT using a digital computer is proportional to the number of complex multiplications (each involving four separate real multiplications and additions). Since n complex multiplications are performed for each of n points, the number of complex multiplications required to perform a DFT is proportional to n^2. As the number of input points n increases, the time required to calculate the transform goes up by the square. When real-time frequency analysis is required on a large amount of data, such as with spectrum analysis, the required computation time can be much too long. In this case, the output frequency data (DFT) falls behind the input time-domain data.

If we have frequency-domain data and want to convert it back to the time domain, we can use the inverse DFT:

$$f(k) = \frac{1}{n} \times \sum_{m=0}^{n-1} [F(m) \times e^{[(j2\pi/n)mk]}]$$

For the inverse transform, the frequency data, $F(m)$, is multiplied by a complex exponential and summed over all its points to calculate each $f(k)$ point. Notice the scale factor of $1/n$ here. As with the forward DFT, the time required to compute the inverse DFT is proportional to the square of the number of points.

The answer to the problem of DFT computations taking too long to calculate is the fast Fourier transform (FFT), which is a special implementation of the DFT. By exploiting the symmetry inherent in the DFT and breaking up the calculations into several smaller transforms, computation time using the FFT can be greatly reduced. Most FFT algorithms only operate on a set of points that is an exact power of 2 ($n = 2^x$). However, the number of complex multiplications required by an FFT is only $n \times \log_2(n)$. So an FFT is $n/\log_2(n)$ faster than an equivalent DFT. For a waveform of 1024 points, this is a speed-up by a factor of more than 100 (1024/10).

For the rest of this discussion, we will assume that the Fourier transforms used on a PC will always be FFTs. The commercial software packages listed in Chapter 11 (and the Appendix) that contain Fourier transform functions all employ an FFT algorithm.

Some of the symmetry inherent in the FFT of a waveform is shown by plotting it. All FFTs are complex waveforms with a real and imaginary component for each frequency value (point). If the original time-domain function is real, the real component of its FFT has even symmetry (symmetrical about point $n/2$) and the imaginary component has odd symmetry (anti-symmetric about $n/2$). If the original function is imaginary, the real component of the FFT has odd symmetry and the imaginary component has even symmetry. If the original function is purely real or purely imaginary, the magnitude of its FFT will have even symmetry.

Very often, when looking at the FFT of a waveform for frequency analysis, only the magnitude $|F(m)|$ is of interest. Since the FFT points are complex:

$$|F(m)| = [(F(m)_{\text{real}})^2 + (F(m)_{\text{imag}})^2]^{1/2}$$

If the signal of interest, in the time domain, is an ideal impulse, infinitely sharp (all but one point is zero amplitude), the magnitude of its FFT is a constant. That is, an impulse contains a spectrum of equal amplitude at all

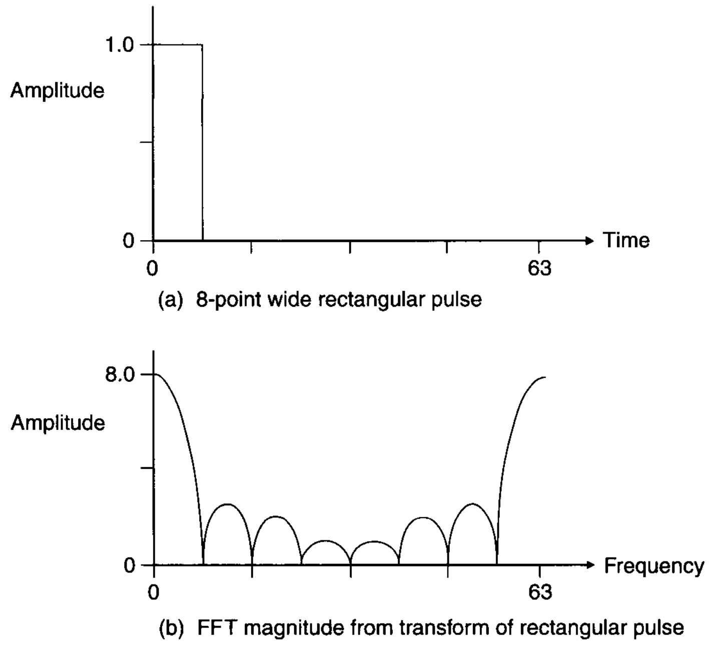


Figure 10-7 Example of fast Fourier transform (FFT): 64-point FFT of 8-point wide rectangular pulse.

frequencies. This makes an impulse very useful as a broad-band excitation signal.

As an example, Figure 10-7a shows a simple rectangular pulse of unit amplitude (1.0), eight points wide in a 64-point waveform. Figure 10-7b displays the magnitude of the FFT of this simple waveform.

Notice the even symmetry of the FFT magnitude. This is because the original function was purely real. For an FFT of n points, the magnitude is symmetrical about point $n/2$. The actual frequency data is valid only up to point $n/2$, which is half the entire frequency range. Since the maximum frequency is equal to the original data acquisition sampling rate ($f_s = 1/dt$, where dt is the time between consecutive samples) the FFT data is valid only up to $f_s/2$, the Nyquist frequency. Above that point it is just the mirror image.

Another interesting feature is the periodicity in the magnitude of the FFT, displayed in Figure 10-7b. With a rectangular pulse y points wide in the time domain, the period in the frequency domain is n/y, which is every eight points in this case. If the rectangular pulse was wider, the number of peaks in the FFT magnitude would increase as the period decreased. Also, note that

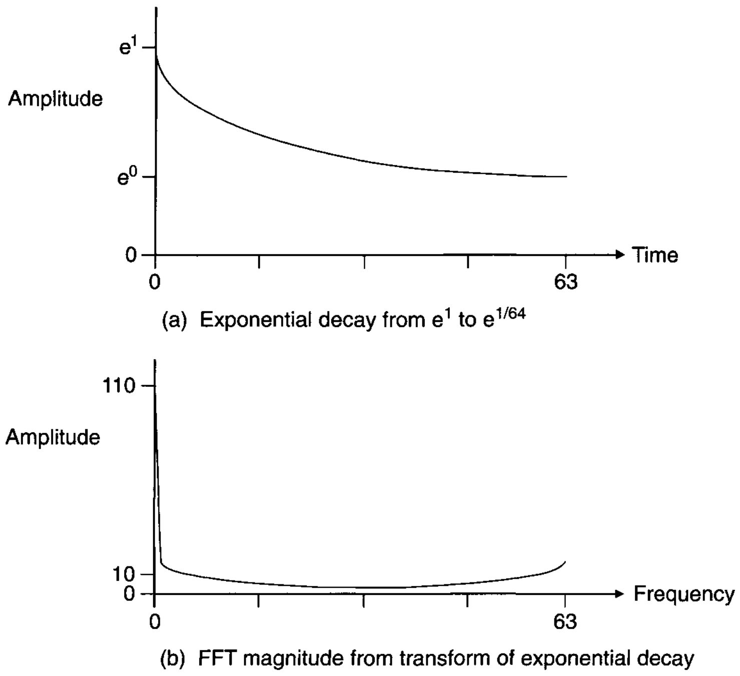


Figure 10-8 Example of 64-point FFT of exponential decay waveform.

the value of the zero-frequency point $|F(0)| = 8$. This is equal to the value obtained by integrating the original pulse waveform (eight points wide with an amplitude of 1), which is its DC component.

Figure 10-8a displays a 64-point exponential waveform decay, e^x, from e^1 at point 0 to $e^{(1/64)}$ at point 63. The magnitude of its FFT is shown in Figure 10-8b. Again, the value we get for $|F(0)|$ is equivalent to the result of integrating under the waveform, which has a large DC offset (note that the exponential waveform does not approach a zero value in the sampled time interval).

The following is a simple FFT program written in BASIC. It will run under IBM BASIC, GW-BASIC, or QBASIC. Since BASIC is an interpreted language (see Chapter 13 for more details), it executes slowly. The actual FFT or (IFFT) computation is done by the subroutine starting at line 400. The test program, starting at line 10, allows the user to enter a 16-point data array as input to the FFT subroutine. This illustrative program is only useful for relatively small data arrays, such as 64 points or less. For larger arrays, the FFT computation time could take several minutes on older PCs.

```
10    REM - FFT PROGRAM, TESTS FFT SUBROUTINE WITH
20    REM - ARRAY OF 16 POINTS, PROVIDED BY USER.
25    CODE = 1 'SET FOR FFT (-1 = IFFT)
30    PI = 3.14159
40    N = 16 'NUMBER OF POINTS IN WAVEFORM
50    DIM R(N) 'REAL DATA ARRAY, INPUT & OUTPUT
60    DIM I(N) 'IMAGINARY DATA ARRAY, INPUT & OUTPUT
70    PRINT "FFT TEST PROGRAM": PRINT
80    PRINT "NUMBER OF POINTS = "; N: PRINT
90    INPUT "REAL DATA INPUT, ONLY - Y OR N?",A$
100   CLS 'CLEAR SCREEN
110   INPUT "INPUT DELTA T (1): ",DELTA
120   PRINT "INPUT SIGNAL DATA POINTS" : PRINT
130   FOR J = 1 TO 16
140   PRINT "POINT"; J; ": ";
150   INPUT "XR = ",R(J)
160   IF A$ = "Y" THEN I(J) = 0! : GOTO 190
170   PRINT "POINT"; J; ": ";
180   INPUT "XI = ",I(J)
190   PRINT
200   NEXT J 'END OF DATA INPUT LOOP
210   PRINT
220   CLS 'CLEAR SCREEN
230   PRINT "CALCULATING FFT ....................."
240   GOSUB 400 'CALL FFT SUBROUTINE
250   PRINT: PRINT "POINT","FFT REAL","FFT IMAG"
260   FOR J = 1 TO N
270   PRINT J,R(J),I(J)
280   NEXT J
290   PRINT
300   INPUT "DISPLAY FFT MAGNITUDE & PHASE - Y OR N?",A$
310   IF A$ <> "Y" THEN STOP
320   PRINT: PRINT "POINT", "FFT AMP", "FFT PHS"
330   FOR J = 1 TO 16
340   AMP = (R(J)^2 + I(J)^2)^.5
350   PHS = PI/2
360   IF R(J) <> 0 THEN PHS = ATN(I(J)/R(J))
370   PRINT J,AMP,PHS
380   NEXT J
390   STOP
400   REM - SUBROUTINE CALCULATES FFT OR INVERSE FFT
410   REM - N = # OF POINTS IN WAVEFORM (POWER OF 2)
420   REM - CODE = 1 FOR FFT, -1 FOR IFFT
430   REM - DELTA = dT FOR FFT OR 1/dT FOR IFFT
440   REM - R(N) = REAL DATA ARRAY FOR INPUT & OUTPUT
450   REM - I(N) = IMAGINARY DATA ARRAY FOR INPUT & OUTPUT
460   IR = 0
470   N1 = N
480   N2 = INT(N1/2) 'CHECK IF N IS A POWER OF 2
490   IF N2*2 <> N1 THEN PRINT "N IS NOT A POWER OF 2!": RETURN
500   IR = IR + 1
510   N1 = N2
520   IF N1 > 1 THEN GOTO 480
530   PN = 2! * PI/N
540   L = INT(N/2)
```

```
550  IR1 = IR - 1
560  K1 = 0
570  FOR Z = 1 TO IR
580  FOR J = 1 TO L
590  K = K1 + 1
600  P = K + L
610  KAY = INT(K1/(2^IR1))
620  GOSUB 1030 'BIT REVERSAL SUBROUTINE
630  AM = KBITR
640  IF AM <> 0 THEN GOTO 680
650  XR1 = R(P)
660  XI1 = I(P)
670  GOTO 730
680  ARG = AM * PN
690  C = COS(ARG)
700  S = -1 * CODE * SIN(ARG)
710  XR1 = C * R(P) - S * I(P)
720  XI1 = C * I(P) + S * R(P)
730  R(P) = R(K) - XR1
740  I(P) = I(K) - XI1
750  R(K) = R(K) + XR1
760  I(K) = I(K) + XI1
770  K1 = K1 + 1
780  NEXT J
790  K1 = K1 + L
800  IF K1 < N THEN GOTO 580
810  K1 = 0
820  IR1 = IR1 - 1
830  L = INT(L/2)
840  NEXT Z
850  FOR K = 1 TO N
860  KAY = K - 1
870  GOSUB 1030 'BIT REVERSAL SUBROUTINE
880  K1 = KBITR + 1
890  IF K1 <= K THEN GOTO 960
900  XR1 = R(K)
910  XI1 = I(K)
920  R(K) = R(K1)
930  I(K) = I(K1)
940  R(K1) = XR1
950  I(K1) = XI1
960  NEXT K
970  IF DELTA = 1 THEN RETURN
980  FOR K = 1 TO N 'SCALE OUTPUT DATA BY DELTA
990  R(K) = DELTA * R(K)
1000 I(K) = DELTA * I(K)
1010 NEXT K
1020 RETURN
1030 REM -BIT REVERSAL SUBROUTINE
1040 REM - KAY = INPUT NUMBER
1050 REM - IR = NUMBER OF BITS TO REVERSE
1060 REM - KBITR = REVERSED NUMBER
1070 KBITR = 0
1080 KAY1 = KAY
1090 FOR Y = 1 TO IR
```

```
1100 KAY2 = INT(KAY1/2)
1110 KBITR = 2 * KBITR + KAY1 - 2 * KAY2
1120 KAY1 = KAY2
1130 NEXT Y
1140 RETURN
```

For most practical FFT applications you will undoubtedly use an FFT function built into a commercial software package (such as those described in Chapter 11 or the Appendix). However, if you need to incorporate FFTs into a custom program, there are many freeware and shareware sources for FFT routines (usually written in C or FORTRAN). One such free FFT library developed and maintained by MIT is FFTW, available via the Internet (at URL: http://www.fftw.org).

10.2.5 Convolutions and Window Functions

Convolution and Deconvolution The utility of FFTs extends far beyond simple frequency analysis of acquired signals, even though this is still an important application. In the real world it is often difficult to measure a quantity "cleanly," without distortion due to the measurement system itself. For time-based or distance-based measurements, the overall system response is a function of the measured quantity along with a function of the system response. This system-response transfer function operates on the desired physical quantity through a process called convolution, producing the measured response.

The convolution $h(x)$ of two time (or space)-domain functions $f(x)$ and $g(x)$ is defined as

$$h(x) = f(x) \bullet g(x) = \int_{-\infty}^{+\infty} f(X)g(x-X)dX$$

We will use the symbol • here to denote convolution. Convolution literally means "folding back." The value of one function at a particular point (x value) affects the overall response at neighboring points, as shown by the $g(x - X)$ function. Convolving two transfer functions produces the overall system-response transfer function.

The convolution integral can be difficult to calculate in the time (or space) domain for many functions. It becomes a simple problem in the frequency domain. The convolution of two signals in the time (or space) domain is equivalent to multiplying their FFTs in the frequency domain. If the FFTs of functions $f(x)$, $g(x)$, and $h(x)$ are, respectively, $F(y)$, $G(y)$, and $H(y)$:

$$H(y) = F(y) \times G(y)$$

where $h(x)$ is calculated from the inverse FFT of $H(y)$. This is illustrated graphically in Figure 10-9. Notice that once the FFTs are multiplied (point

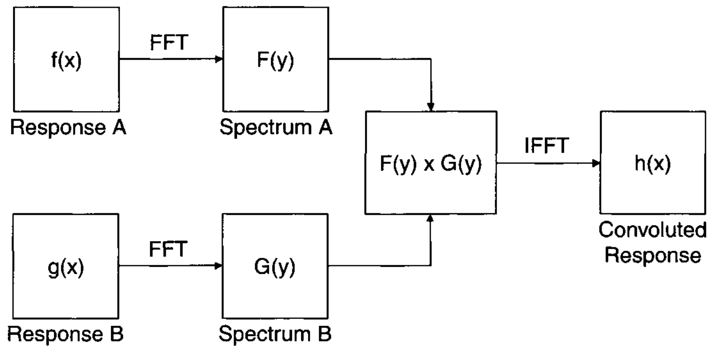


Figure 10-9 Convolution algorithm using FFTs.

by point), an inverse FFT (IFFT) is performed on the result to produce the output impulse response, which is the convolution of the two input responses.

An important aspect of transforming convolutions into multiplications via FFTs is that we can reverse the process. If we have data acquired from a system with a known impulse response, we can correct for that response. We transform the measured data, along with the impulse response, to the frequency domain (via an FFT). By dividing the FFT of the measured data by the FFT of the impulse response, we deconvolve the data. Transforming the result via an IFFT results in data fully corrected for the system's impulse response. This process is shown graphically in Figure 10-10.

Deconvolution is an extremely useful analysis technique. In the field of optics, for example, image enhancements can be implemented via deconvolution.

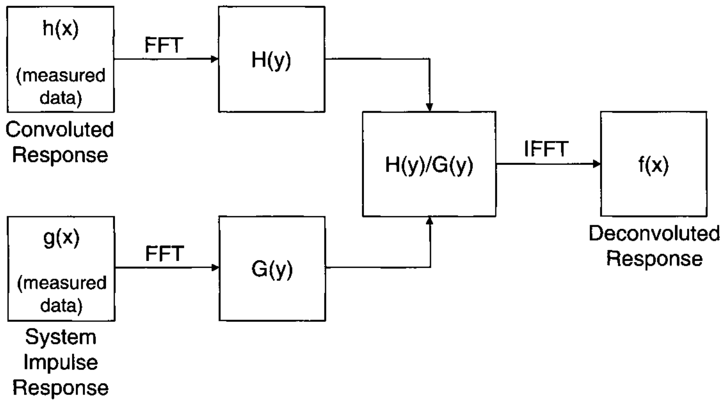


Figure 10-10 Deconvolution algorithm.

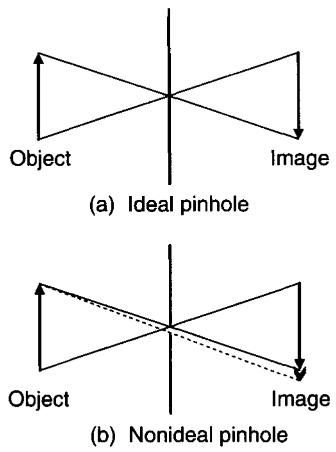


Figure 10-11 Optical pinhole.

A simple example is a pinhole camera. An ideal pinhole, with a diameter much smaller than the wavelength of light used, acts like a lense, producing an inverted image of an object, as shown in Figure 10-11a. Each point of the image corresponds to light from only a single point of the object. With a nonideal pinhole, each image point corresponds to several object points, as in Figure 10-11b. The image becomes blurred as light from neighboring points mixes together. This is the convolution of the real image with the light distribution function of the pinhole. Knowing that pinhole transfer function, we can deconvolve the data to get the undistorted image.

There are many other examples of the utility of deconvolution, as in the field of ultrasonics. Figure 10-12 shows a simple experiment using a pair of ultrasonic transducers in a water bath. An ultrasonic pulse is transmitted by one transducer and received by another transducer for data acquisition. The ultrasonic properties of the test sample, between the two transducers, are of interest. By deconvolving the measurement taken when the test sample is present with a measurement taken without the test sample, the impulse (and frequency) response of the entire test system can be eliminated from the data. This leaves the true ultrasonic response of the test sample. The test sample frequency response provides information about its physical properties.

Window Functions When analyzing real-world data, there are often artifacts we wish to ignore. With ultrasonic or optical measurements, for example, there are often pulse echoes. If we need to analyze the data of interest without

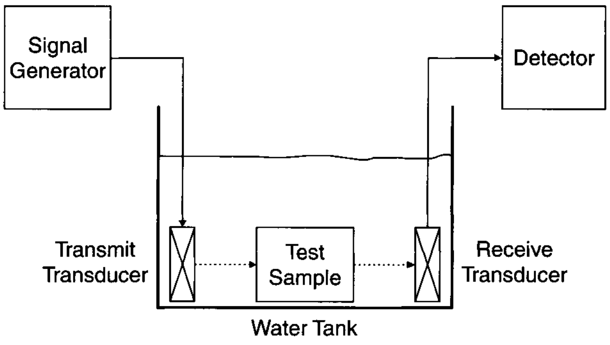


Figure 10-12 Simple ultrasonic test system.

including the entire waveform, often a windowing function is used. The simplest time-domain window function is a rectangular pulse that is multiplied with the time-domain waveform of interest. The width and position of the pulse is selected so that it has a value of 1 over the region of interest in the waveform and a value of zero elsewhere, as illustrated in Figure 10-13.

Multiplying two functions (signal and window) in the time domain is equivalent to convolving their FFTs in the frequency domain. As we previously saw in Figure 10-6, the FFT of a rectangular function produces multiple peaks following the first main peak at zero frequency. These secondary peaks are referred to as side lobes. The higher the amplitude of the side lobes, the more the windowing function distorts the signal when they are transformed. For a rectangular window, the first side lobe has a peak amplitude of only −13 dB relative to the main (zero frequency) peak.

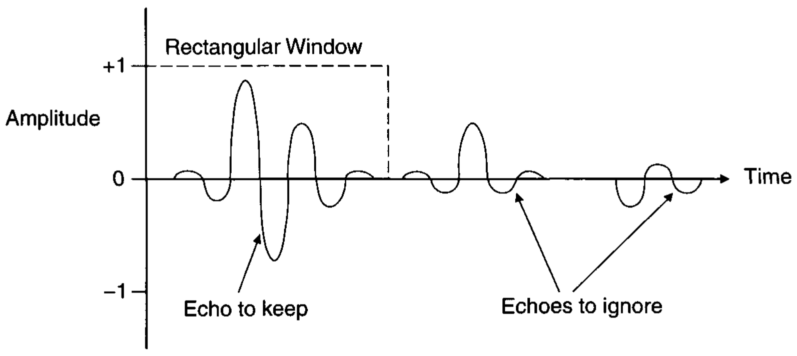


Figure 10-13 Using a rectangular window on ultrasonic echo waveforms.

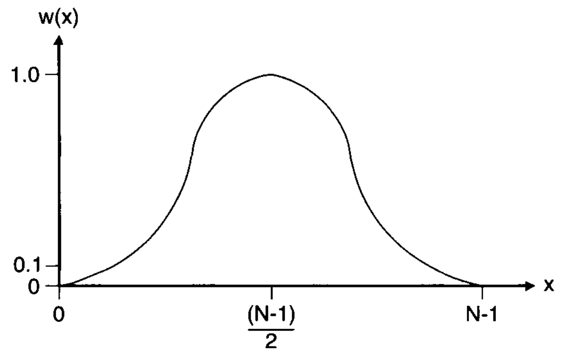
Because of the convolution distortion, time-domain window functions other than simple rectangles are used. Several are based on cosine functions that slowly taper to zero near the edges of the window region. Besides having lower side lobes, these windows also have wider main lobes than a rectangular function. This further helps to decrease any distortion they cause.

Two commonly used window functions are the Hanning and Hamming windows, shown in Figure 10-14. These window functions are defined for a width of N points as follows:

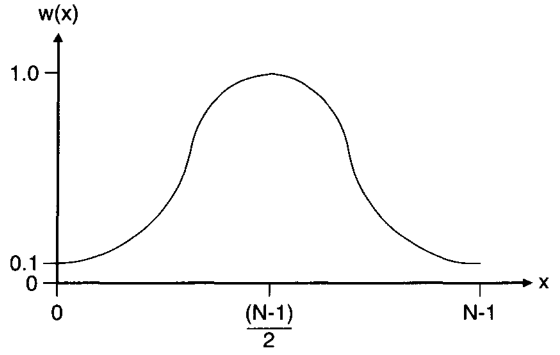
$$w(x) = 0.5 \times (1 - \cos[2\pi x/(N - 1)]) \qquad \text{Hanning Window}$$

$$w(x) = 0.54 - 0.46 \times \cos[2\pi x/(N - 1)] \qquad \text{Hamming Window}$$

where x varies from point 0 to point $N - 1$.



(a) Hanning window function



(b) Hamming window function

Figure 10-14 Hanning and Hamming window functions.

Notice that both window functions have their amplitude peak of 1.0 at the center of their range, $(N-1)/2$. The main difference between them is that the Hanning window goes to zero amplitude at the edges of its range ($x = 0$ and $x = N - 1$) while the Hamming window has a finite amplitude of 0.08 at these edges. Both of these windows have a main lobe twice as wide as an equivalent rectangular window, with the same value of N. The Hanning window has a peak side lobe amplitude of −31 dB and the Hamming window has a peak side lobe amplitude of −41 dB. These indicate a large improvement (18 to 28 dB) over the rectangular window's peak side lobe amplitude of only −13 dB.

10.2.6 Other Transforms

There are many other transforms used for DSP analyses. We will briefly look at two of them here: the Hilbert transform and wavelets.

The Hilbert Transform The Hilbert transform is a technique used to obtain the minimum-phase response from a spectral analysis. When performing a conventional FFT, any signal energy occurring after time $t = 0$ will produce a linear delay component in the phase of the FFT. Even if a pulse occurs at $t = 0$, if it has finite width it will produce this linear slope in the resulting FFT phase. The slope of the FFT phase (versus frequency) is proportional to this time delay term. Significant delays can produce phase variations of greater than 2π. If the FFT data contains phase nonlinearities of interest (such as a small bump), they can be hidden by this large linear phase component.

The Hilbert transform, based on special processing of an FFT, will produce a frequency response with this linear-phase component removed. This is the "minimum phase" data desired. The algorithm involves signal processing in both the time and frequency domains.

Wavelet Analysis Fourier transforms (and FFTs) are ideally suited for analyzing continuous, periodic signals but do not work well when a signal has sharp discontinuities or spikes. The problem is, a Fourier series tells you what frequencies a signal is composed of but not their locations in time (it assumes that all the frequencies are always present, within the analysis window). If we increase time resolution by using a smaller sampling window in an FFT analysis, our frequency resolution becomes poorer since the frequency step $df = 1/dt$, the time step. This is the inherent FFT limitation for concurrent time–frequency analysis.

Wavelet analysis (or the wavelet transform) is a fairly new mathematical technique that addresses these shortcomings. Unlike FFTs, wavelets are well suited to representing discontinuous signals. Wavelet analysis uses a scalable window that is time-shifted across a signal. A spectrum is calculated at each new window position. The window size is slightly changed for each iteration. The final result is a group of time–frequency representations of the original signal, all having different resolutions. This wavelet analysis is called a multiresolution technique. Wavelets allow you to analyze a signal with both coarse (large scale) and fine (small scale) resolution.

Wavelet analysis has many similarities to Fourier analysis. There is a continuous wavelet transform (CWT) analogous to the CFT and a discrete wavelet transform (DWT) analogous to the DFT, used for computer-based signal processing algorithms. Wavelet functions contain frequency information as Fourier functions do. Unlike FFTs, wavelet functions are also localized in space (or time). In addition, a wavelet transform of a one-dimensional waveform produces a two-dimensional function.

The CWT of a time-based function $f(t)$ is

$$\Psi(\tau, s) = f(t)\psi^*_{\tau,s}(t)\,dt$$

where

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}}\psi\frac{(t-\tau)}{s}$$

Now the transformed time signal is a function of two variables: τ, the translation parameter (time-based) and s, the scale parameter (inverse frequency-based). $\psi\,(t)$, the transforming function, is the mother wavelet. All of the wavelets required by the analysis are generated from this mother wavelet by scaling and translation.

A set of waveforms comprising a transform is called a basis function. Fourier transforms use only sine and cosine waves as its basis functions—a signal is decomposed into a series of sine and cosine functions by the FFT. The CWT and DWT have an infinite set of basis functions or wavelets. Usually, a specific wavelet family is selected for a particular application.

By its nature, the CWT contains a large amount of redundant information along with an infinite number of wavelets. The DWT, using discrete wavelets, overcomes these problems. With the redundancy removed, wavelet transforms become sparse—only a few wavelets are needed to describe or decompose a given signal. This makes DWTs well suited for data-compression, image-analysis, and noise-reduction applications. Efficient software algorithms implementing DWTs have led to their widespread use.

It is likely that wavelet transforms will continue to increase in popularity and they may eventually replace the ubiquitous FFT as the technique of choice for signal analysis.

10.2.7 Other DSP Techniques

A host of DSP techniques besides the FFT are commonly used. An exhaustive survey of the DSP field is outside the scope of this book. We will just look at a few more techniques that you may likely need in a data acquisition system. Please refer to the bibliography for sources of more detailed information on DSP.

Digital Filters Digital filtering techniques are most often applied to time-domain signals, as in real-time filtering applications. Depending on system parameters, a digital filter can operate more quickly than using an FFT algorithm where a forward FFT converts a time-domain signal to the frequency domain. Then the frequency signal is multiplied by a filter function and finally the frequency signal is converted back to the time domain via an IFFT.

The two common types of digital filter approaches are finite impulse response (FIR) and infinite impulse response (IIR). The filtering process is effectively a convolution of the time-domain signal with a filter function.

FIR digital filters are considered nonrecursive. They mix delayed portions of the input signal with feedforward of the undelayed signal. They operate only on a small time-domain window of signal data. The filter function describes the coefficients for each of the delayed and undelayed components. FIR filters usually have a linear phase response, are relatively easy to implement, and do not tend to accumulate errors, since they operate on a data window of finite width. Their main limitation is the need to use many coefficients for good performance. This results in longer computation times and lower bandwidths.

IIR digital filters are considered recursive. They mix the input signal with time-delayed feedback of the output signal. They operate on a wide time-domain window of signal data. Even though it may be more difficult to design an IIR filter than an FIR filter, the resulting IIR filter is simpler, with fewer coefficients. This results in shorter computation time and wider bandwidths. Their main drawbacks are their sensitivity to noise and error accumulations, due to including effects of all past data.

Cross-Correlation The final DSP technique we will touch on here is cross-correlation. This is used to see how similar two functions are. The cross correlation function of $x(t)$ and $y(t)$ is

$$c(t) = \frac{1}{(a_x a_y)}\int_{-\infty}^{+\infty} x(T)y(t+T)\,dt$$

where a_x and a_y are the RMS values of the two functions. This normalizes $c(t)$ to a maximum value of 1 (if the two functions correlate). If the two signals are very similar, there will be a maximum in the cross-correlation function. Otherwise, there will not be any significant maximum. If one function represents a delayed version of the other function, $c(t)$ will equal 1 (or its maximum) at a value of t equal to the time delay.

This concludes our overview of data processing with PCs. The techniques covered include some of the most common data analysis methods used with data acquisition systems. In the next chapter we will look at commercial hardware and software data acquisition products for PCs.

CHAPTER 11

Commercial Data Acquisition Products

There is a plethora of commercially available data acquisition products for PCs, with the number growing larger every day. The largest selection exists for Intel CPU-based, PCI-bus systems running MS Windows 95/98/NT/2000 (so-called "Wintel" PCs). However, there are still products available for ISA and PC-104 buses as well as some software support for MS-DOS. There is also a growing number of products using the USB interface. In addition, some products support newer Apple Macintosh computers that use the PCI bus.

These commercial products fall into two broad categories: hardware and software. Some software is included with most hardware products, to assist the user. Occasionally, hardware manufacturers just recommend compatible software products, along with programming guidelines. Some products are a complete hardware/software bundle, requiring both for proper operation.

In this chapter we will survey the vast array of data acquisition hardware and software products. We will look at products from a few major manufacturers in detail, including both operational information and how to use the products. The Appendix contains lists of commercial data acquisition product manufacturers (hardware and software). Since most hardware products operate similarly, regardless of the computer platform used (PC, Macintosh, VME bus), an in-depth discussion will again center on "Wintel" products. First, we will examine hardware products.

11.1 Commercial Data Acquisition Hardware Products

A large number of manufacturers produce data acquisition hardware products for PCs. The largest market is for Microsoft Windows-based, PCI-bus and ISA-bus computers running Intel or compatible processors. For these machines, most data acquisition hardware products are cards that plug into a PC's expansion bus. Typically, the newest and fastest products are PCI-based. Many ISA-bus products are still available, but they are not recommended for new applications. ISA boards come in two major versions: 8-bit cards for PC/XT class computers and 16-bit cards for AT (ISA) machines. Many products have additional hardware, external to the PC, which connects to the main data acquisition card. These add-on devices include connection boxes, signal conditioning boards, and high-power I/O interfaces, including relay boards. Some PC-based data acquisition systems consist of an external box connected to the PC's bus for control, usually via a special interface card.

Besides plug-in cards, there are now data acquisition hardware modules that connect to a PC via a USB port. These plug-and-play devices are very easy to install but are usually not high-performance products (having limited data transfer rates). Notebook PCs can also use data acquisition hardware via PCMCIA cards. In addition, some data acquisition products connect to a PC via a standard serial or parallel port.

Many data acquisition boards for PCs have dedicated functionality, such as only analog inputs. Some may have expansion capability, such as an additional multiplexer for more analog inputs. Other PC-based data acquisition cards are designed to be modular. They consist of a basic plug-in card, the *carrier*, which accepts several modules riding "piggy-back" on it. These modules offer specific functions, allowing the user to tailor the hardware to his or her particular needs (such as the number of analog inputs and outputs required). The module functions include analog I/O, digital I/O, and signal conditioning. This modular approach offers greater flexibility, at a higher price. It is usually justified when a highly customized system is required or configuration changes will occur often.

Data communications interface cards are also an important piece of data acquisition system hardware. In this case, the PC is used as an intelligent controller, running remote data acquisition equipment through the interface. These interfaces include GPIB, RS-232C, RS-422, and RS-485. Of course, these cards can also be used in PCs for communications purposes other than data acquisition. For example, even though a GPIB interface card is often used in a PC to control automated instruments, it could be used to simply drive a printer or plotter.

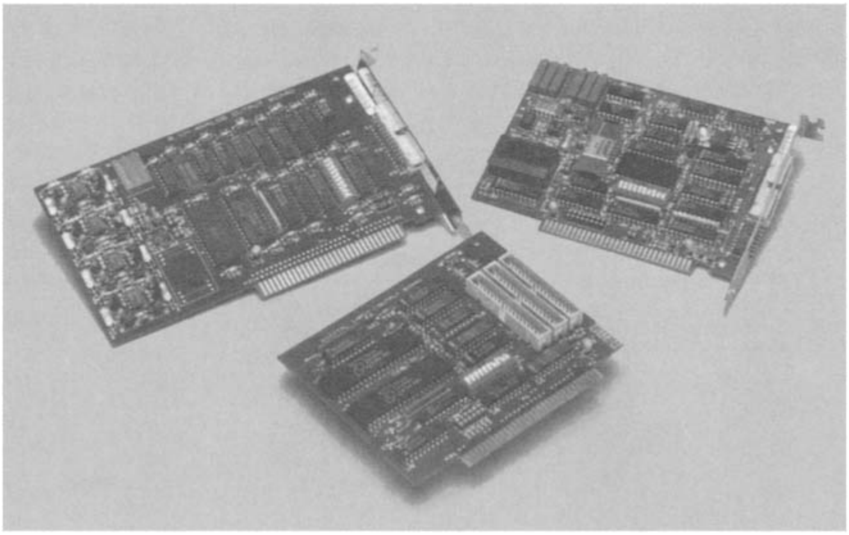


Figure 11-1 Typical ISA, single-function data acquisition cards for PCs. (Courtesy of Intelligent Instrumentation)

Data acquisition cards for PCs fall into several major functional categories, including digital I/O, analog I/O, and counter/timer. Some boards have most or all of these features; others have only one or a few. A few typical ISA data acquisition cards are shown in Figure 11-1. There are also specialized data acquisition cards which have features geared to a particular application, such as chromatography equipment used in analytical chemistry labs.

Another variation on data acquisition cards is the virtual instrument. This type of device is a combination of hardware (a card) and software that emulates the functionality of a standard test instrument, such as an oscilloscope or function generator. The user interface is a graphics environment that looks like the front panel of the emulated instrument. By adjusting the virtual knobs or pressing virtual buttons, the user operates the virtual instrument. When the virtual instrument is an oscilloscope, the hardware consists of an analog input card. A virtual function generator would use an analog output card.

Digital I/O cards have input and output lines typically operating at TTL logic levels (in the range of 0 to +5 V). Stand-alone digital I/O cards often contain some multiple of 8 I/O lines, with 16 or 24 being most common. These cards can be used as parallel, digital interfaces as well as dedicated controllers. Most digital I/O cards allow programming lines for input, output,

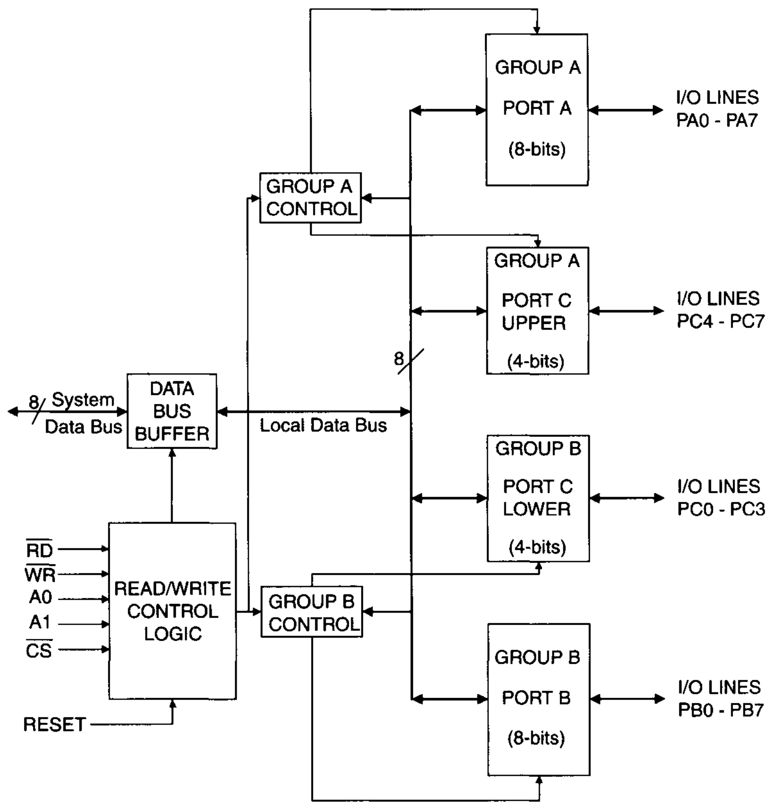


Figure 11-2 Intel 8255A programmable peripheral interface (PPI).

or both. Usually they contain interrupt-generation hardware. Some digital I/O cards support DMA for maximum data transfer speeds.

A popular IC used for digital I/O was the Intel 8255A Programmable Peripheral Interface (PPI), whose block diagram is shown in Figure 11-2. This device had three 8-bit ports that could be programmed for one of three modes: simple, unidirectional I/O without handshaking; strobed, unidirectional I/O with handshaking; and strobed, bidirectional I/O on the same pins, with handshaking. The 8255A was controlled by addressing its control port and three data ports. It was so popular that the 8255 became an industry standard for digital I/O and it still remains a standard, long after Intel stopped manufacturing the chip. In current digital I/O cards (which do not use the 8255A itself), the IC's functionality is usually part of a highly integrated programmable logic device (PLD). This logic emulates an 8255A and its registers, providing full compatibility with software written for the original chip.

Analog I/O cards are the most common form of data acquisition hardware for PCs. They contain one or more ADCs for analog input and DACs for analog output. Usually, any card containing an ADC for analog input is considered a data acquisition card. Analog input cards typically contain one ADC IC or module along with one or more analog multiplexers. This enables several analog signal sources (such as conditioned sensors) to be connected to one board at the same time. For example, multiple temperature sensors may be used in monitoring different portions of a piece of equipment under test. The multiplexer allows one of several analog inputs to be connected to the ADC at any given time. Commonly, commercial ADC cards have 8–32 analog input channels. These channels may be differential or single-ended.

The resolution of the ADCs and DACs used on these cards range from 8 bits to 24 bits. Analog I/O boards with 12-bit resolution are still the most common. Another important parameter is the maximum conversion rate for analog input cards. This can range from only tens of samples/sec, on high-resolution and/or low-cost cards, to more than 2 million samples/sec at 16-bit resolution or 100 million samples/sec at 14-bit resolution on high-speed data acquisition cards with PCI interfaces. Cards with conversion rates up to 5 billion samples/sec (five gigasamples/sec) at 8-bit resolution are also available at this time.

When looking at the maximum conversion rate for an ADC card, remember that it is usually specified for a single channel only. If you need to measure several inputs simultaneously, the maximum conversion rate at any channel is the ADC's maximum rate divided by the number of multiplexed channels used. If this overall rate is too slow, you will need multiple ADCs (one or more cards), a faster ADC, or a card with simultaneous sampling hardware.

Analog output cards usually contain one DAC per output channel. Occasionally, a card may contain one DAC and several analog output channels, employing a sample-and-hold (S&H) amplifier for each channel. As we previously saw (in Chapter 3), S&H amplifiers "remember" a voltage level using a charged capacitor. Since the capacitor's charge slowly drops (because of its own leakage current and that of the surrounding circuitry), the S&H output "droops" with time. The S&H output must be continuously refreshed by recharging the capacitor (as with DRAMs), or the analog output will be valid only for a short period of time (usually on the order of milliseconds). Because of these drawbacks, this approach is not widely used. Most analog output boards have only a few channels, with an independent DAC for each.

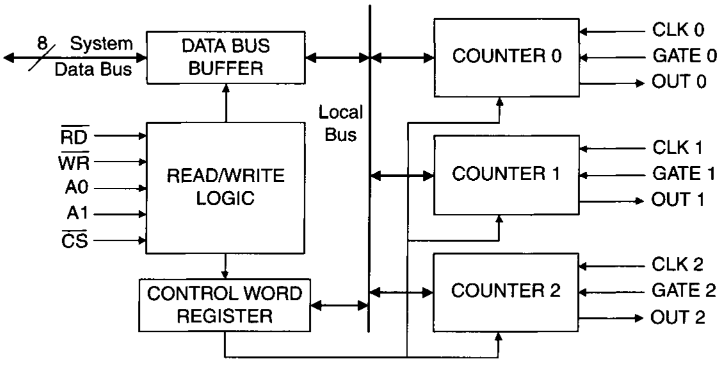
Most analog I/O cards contain a timer/counter with multiple channels. This enables the card to perform conversions at a fixed rate, without any PC software overhead. It is a common option for data conversions to be controlled by an internal (on-board) clock, by an external clock, or by PC software commands. Most analog input cards have hardware interrupt capability. This is a programmable option, used to generate an interrupt when the ADC is ready to be read. It is especially important when the ADC conversion rate is controlled by an on-board clock and is essentially asynchronous to the control software running on the PC.

Some ISA analog I/O cards have DMA capability. This allows data to be transferred between the data acquisition card and the PC at the fastest possible rate. It does require special software support, but this is usually commercially available. These software packages are used for data *streaming* (transferring data between the data acquisition card and a disk file at high speed) as well as simulating the functions of a high-speed strip-chart recorder. Only high-speed ISA analog I/O cards use DMA, since for slower cards the analog data conversion speed, not the data transfer rate, becomes the rate-limiting factor.

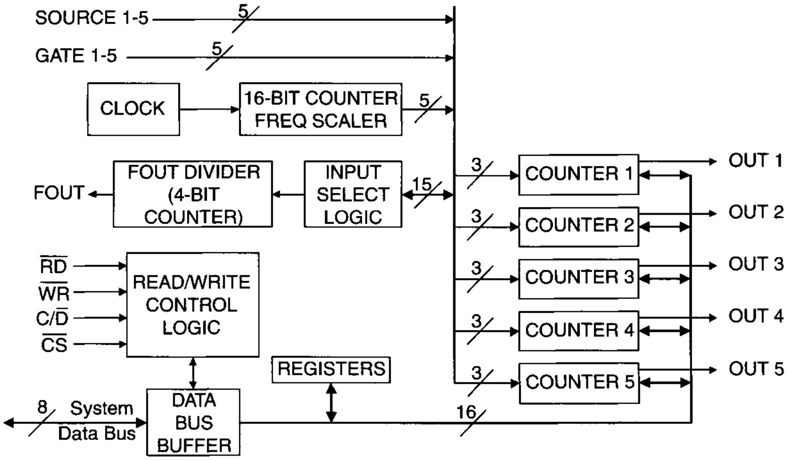
PCI analog I/O cards are potentially much faster than their ISA counterparts, because of their faster bus speed. Most PCI data acquisition cards are PCI version 2.1 compliant, supporting a 32-bit data bus at speeds up to 33 MHz, with a peak burst rate of 132 Mbytes/sec. This is well over an order of magnitude faster than ISA-bus DMA. Another important feature of many PCI data acquisition cards is bus-mastering capability. This allows the board to transfer data into memory as soon as it is available, without waiting for application software to respond to a poll or interrupt (in a manner analogous to an ISA DMA operation, but with better handshaking and more hardware "intelligence"). In addition, high PCI transfer rates minimize the amount of on-board memory required by the card to buffer acquired data, before it can be transferred to the PC's main memory. Usually, a FIFO that is only a few thousand samples deep is an adequate buffer (as opposed to the several megabytes of buffer memory that is required on a high-speed ISA card).

Timer/counters are available on separate cards, typically in conjunction with digital I/O lines. Besides being used for controlling data conversion rates, they are also useful as general-purpose clocks, frequency counters, and event counters. They usually have TTL compatible inputs, but with proper signal conditioning, such as an amplifier (to boost the signal level) and a comparator with hysteresis (to square up slow rise/fall times of a signal and convert it to TTL levels), analog signals can also be measured.

ICs commonly used for timer/counters were the Intel 8254 Programmable Interval Timer (PIT) (used in earlier PC motherboards) and the AMD AM9513A System Timing Controller (STC), shown in the block diagrams of Figure 11-3. The Intel 8254 PIT contained three independent 16-bit counters; the AMD AM9513A STC had five.



(a) Intel 8254 Programmable Interval Timer (PIT)



(b) AMD AM9513A System Timing Controller

Figure 11-3 Commonly used counter/timer integrated circuits (ICs).

These ICs are now obsolete. However, they were once such popular industry standards that their functionality has been emulated using programmable logic in many current products. As with the 8255A PPI, these emulated timer/counters appear the same as the original ICs to software. Of course, some new timer/counter cards dispense with this backward compatibility and implement counter logic with different (and usually more advanced) characteristics.

As shown in Figure 11-3, each of the three 8254 counters has a clock input, a gate input, and an output line. They are synchronous down-counters (binary or BCD) with a count register to load a counter value, an output register to read the counter value, and a status register. Six programmable counting modes are available, allowing the 8254 to be used as a clock, an event counter, a one-shot generator, a programmable square-wave generator, or a complex digital waveform generator.

The AM9513A design is an extremely powerful counter/timer, with many operational modes and advanced features, making its functionality a popular choice for manufacturers of high-performance data acquisition cards. Each of the five AM9513A counters has a source input, a gate input, and an output line. It differs from the 8254 by having a common clock generator as part of the device. Each counter can choose its clock input from either this internal source (including a clock divided down from the internal one) or an external clock on its source line. This internal clock was originally 1 MHz, but later designs went as high as 7 MHz. The synchronous counters can count either up or down in binary or BCD. They can be concatenated for an effective counter length of 80 bits. The device has a scaled frequency output. Each counter has a load register to initialize the counter, a hold register to read the instantaneous count value, and a mode register to program the counter's features (such as clock source, polarity of gating line, or output conditions). The AM9513A can be used for extremely complex timing and waveform-generation applications.

The most useful configuration for PC-based data acquisition hardware is the multifunction board. This card contains, at a minimum, an ADC and digital I/O lines. A typical multifunction data acquisition card contains several analog input channels, one or more analog output channels, several digital I/O lines, and several timer/counter channels. Some may even contain signal-conditioning circuitry, such as filters. These boards can contain all the hardware needed to convert a PC into a complete data acquisition system (along with the appropriate software), usually at a very attractive price. Just make sure that you need most of the functions on the card and that each individual function meets your requirements (such as an adequate number of I/O channels or an ADC conversion rate that is fast enough). Without a doubt, multifunction boards are the most popular type of data acquisition card for a PC.

Now that we have covered some of the general aspects of data acquisition hardware, we will look at some commercially available products, covering a few of the more popular manufacturers. Complete addresses and other details are in the Appendix.

11.1.1 Keithley Instruments, Inc.

Keithley Instruments, Inc. (previously Keithley Metrabyte Corp.) manufactures data acquisition cards and accessories for PCI and ISA-bus computers as well as PCMCIA cards for notebook PCs, communications interface cards

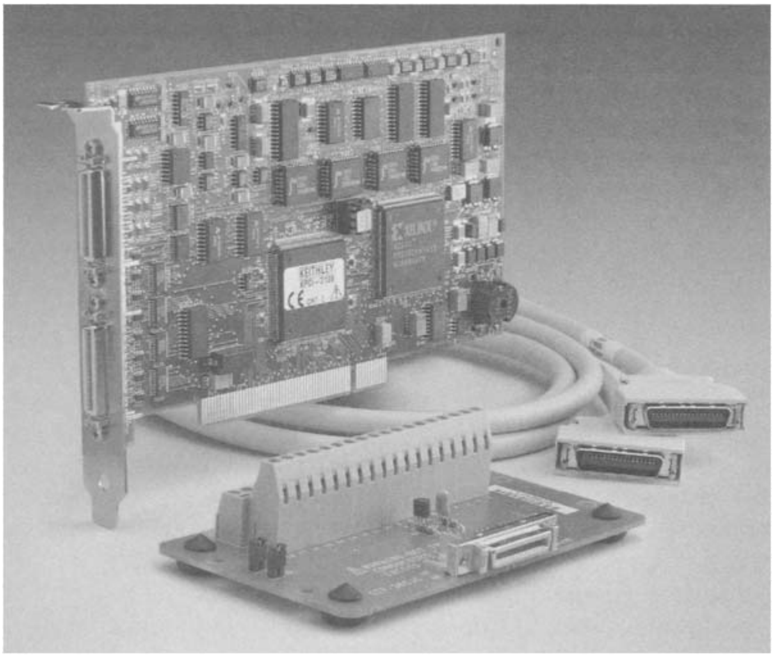


Figure 11-4 Keithley KPCI-3108 multifunction PCI data acquisition card. (Courtesy of Keithley Instruments, Inc.)

(serial and IEEE-488), and virtual instruments. Their products for PC-based data acquisition range from low-cost ISA boards, such as the DAS-8, to high-performance, multifunction PCI cards, such as the KPCI-3108. The DAS-8 has eight single-ended analog input channels with 12-bit resolution, an input range of ±5 V and a maximum conversion rate of 4000 samples/sec. It also has seven digital I/O lines (four outputs and three inputs).

The KPCI-3108 (shown in Figure 11-4) has 16 single-ended (or 8 differential) analog input channels of 16-bit resolution with a maximum conversion rate of 100,000 samples/sec. The analog input range is software selectable, from ±0.0125 V to ±10 V, full-scale. It also contains two 16-bit analog output channels with a maximum conversion rate of 100,000 samples/sec and an output range up to ±10 V. In addition, the KPCI-3108 has 32 digital I/O lines, three 16-bit counter/timers, 12 auxiliary digital I/O lines for timer gating or clocking, and full PCI bus-mastering capability for high-speed data transfers. This board also contains a 256-location channel-gain queue that allows you to acquire data from nonsequential channels at different gain settings, using a preprogrammed sequence.

You can use the KPCI-3108 (and most newer Keithley cards) with a fully integrated data acquisition software package, such as LABTECH NOTEBOOK or TestPoint. Alternatively you can write your own custom program using a standard 32-bit programming language (running under Windows 95/98/NT/2000), such as Microsoft Visual Basic or Visual C/C++. The board comes with drivers: Keithley's Driver Linx software. Unlike older ISA cards, the KPCI-3108 (and other newer cards) does not use register-level programming. The Driver Linx software provides a higher-level interface to the board's analog and digital functions within the Windows environment. You simply make calls to driver functions.

Keithley produces an ultrahigh-speed ISA ADC board, the DAS-4300, which has a maximum transient conversion rate of 1 gigasample (Gsample)/sec with 8-bit resolution. To support this data acquisition rate, which is much faster than PC DMA transfer rates, the DAS-4300 has on-board memory of 8 Kbytes for data storage. It has two single-ended analog input channels (with 50-ohm input impedance), but no digital I/O or analog output lines. ADC triggering can come from a software command, an external logic level, or an analog signal. The analog input range is software selectable from ±25 mV to ±1 V full-scale, with both coarse and fine steps.

A special feature of the DAS-4300 is equivalent time sampling (ETS), used to increase the effective sampling rate when digitizing repetitive signals (see Chapter 4). Using ETS, the DAS-4300 can run as fast as 20 Gsamples/sec.

The DAS-4100 is another 8-bit, high-speed analog input board in the same family as the DAS-4300. The DAS-4100 has a maximum transient conversion rate of 64 million samples/sec and an ETS rate up to 2 Gsamples/sec. This card can have as much as 1 Mbyte of on-board memory, allowing full-speed capture of relatively long waveforms.

An example of a high-resolution PCI card is Keithley's KPCI-3116, which has an ADC with 16-bit resolution and a maximum conversion rate of 250,000 samples/sec with 32 single-ended or 16 differential analog input channels. It also includes two 16-bit analog outputs, 16 digital I/O lines, and four counter/timers. As with the KPCI-3108, the KPCI-3116 has full PCI bus-mastering capabilities.

Keithley also produces a line of PCMCIA (see Chapter 12) data acquisition cards for use with laptop or notebook computers. This class of products is ideal for portable data acquisition systems. Figure 11-5 shows a collection of some PCMCIA cards. A typical example is the KPCMCIA-12AIAO, which has a 12-bit ADC with conversion rates up to 100,000 samples/sec, a programmable input range of ±1.25 V to ±10 V full-scale and eight single-ended (or four differential) inputs. The KPCMCIA-12AIAO also has two DAC outputs with a maximum update rate of 100,000 samples/sec, a 16-bit

Figure 11-5 Collection of PCMCIA cards. (Courtesy of Keithley Instruments, Inc.)

counter/timer, and eight digital I/O lines (four inputs, four outputs). The KPCMCIA-16AIAO is a 16-bit data acquisition card with the same specifications as the KPCMCIA-12AIAO (i.e., 100,000 samples/sec conversion rate) except for higher resolution. As with standard PCMCIA devices, these cards are hot-swappable (they can be plugged or unplugged when the PC is on).

As an illustration of a typical data acquisition card for ISA PCs, we will examine another Keithley board in greater detail, the DAS-16, shown in Figure 11-6. Even though this is an old product (originally designed for the 8-bit PC-XT bus) it has been so popular that not only does Keithley still build it but many other manufacturers also produce functionally equivalent versions of the board (so-called "clones"). Versions of the DAS-16 are even available as PC-104 cards (see Chapter 12).

The DAS-16 is a multifunction card, with 16 single-ended (or eight differential) analog input channels of 12-bit resolution, with a maximum conversion rate of 50,000 samples/sec (the DAS-16F, with DMA support, has a maximum rate of 100,000 samples/sec). It has two 12-bit analog output channels, eight digital I/O lines, three timer/counter channels, and interrupt support.

The DAS-16 card will work in virtually all PC/XT and ISA PCs, as it requires only a PC/XT bus (62-pin) expansion slot. This makes it useful for older PCs in nondemanding applications, such as temperature logging. A block diagram of the DAS-16 is shown in Figure 11-7. Like most older ISA data acquisition cards, the I/O addresses used by the card are switch selectable

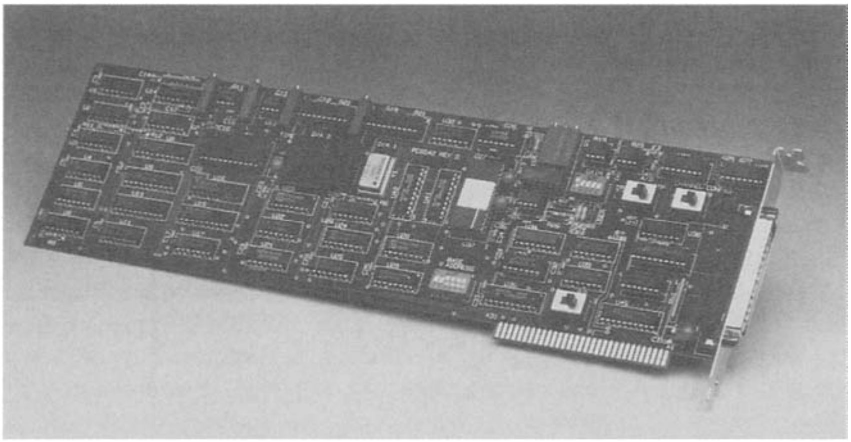


Figure 11-6 Keithley DAS-16 multifunction ISA data acquisition card. (Courtesy of Keithley Instruments, Inc.)

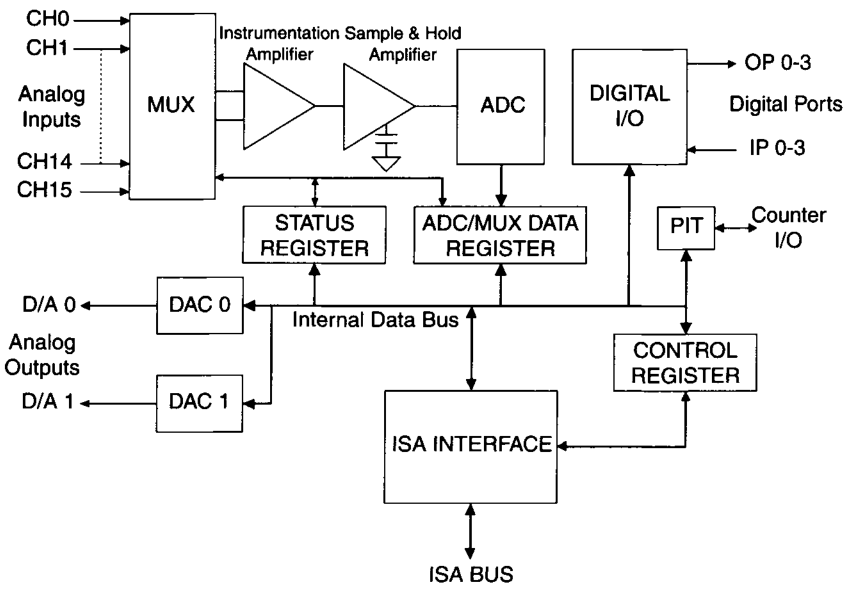


Figure 11-7 Block diagram of Keithley DAS-16 card.

(it is not plug-and-play). Since the DAS-16 uses 16 consecutive addresses, only the base or starting address is explicitly selected. By default, this address is 300h, which is commonly used for ISA data acquisition cards, being part of the I/O map (300h–31Fh) reserved by IBM for prototype cards. In this case, the DAS-16 would occupy 300h–30Fh. If this space was already in use, another base address would be selected, such as 310h. The base address has to fall on a 16-bit boundary, as the address select switches are for bits A4 through A9. Newer ISA and all PCI data acquisition cards are plug-and-play, so I/O addresses are automatically selected.

Other switches on the DAS-16 select differential or single-ended lines for the analog input channels, ADC gain level, and unipolar-versus-bipolar ADC input range. The DAS-16 has five preset gain levels for the ADC, determining full-scale range. In bipolar mode these are ±10 V, ±5 V, ±2.5 V, ±1 V, and ±0.5 V. For many newer data acquisition cards, the gain levels are set by software commands.

All external connections to the DAS-16 (other than the ISA expansion bus connector) are made via a 37-pin D-shell connector, at the back of the card. Most data acquisition cards use this type of arrangement if the number of connector lines is not excessive (usually 50 or less). The most common connectors used are D-shell and ribbon-cable varieties. If many external connections are needed, as with a multifunction card having a large number of analog and digital I/O lines, usually several ribbon cable connectors on the board itself are used. These cables then have to be routed through an opening in card's mounting bracket. On the DAS-16, the 37-pin D-shell connector contains all the analog and digital I/O lines. In addition, it contains control lines for the accessible timer/counters, power supply (+5 V) and reference voltage (–5 V) outputs, along with an input for an external DAC reference voltage (if a range other than 0 to +5 V is desired).

All software access to the DAS-16 is done by reading from and writing to the 16 I/O ports located in the ISA I/O space between the base address and base +15. These I/O ports are listed in Table 11-1. Note that some of these ports are either read-only or write-only, while some are both read and write. In addition, the same port address can have a different function, depending on whether you read from it or write to it. For example, the base address, as a read port, returns the low byte of the last ADC conversion. As a write port it initiates an ADC conversion.

The mux scan port (at base +2) allows multiple ADC channel conversions to be performed without explicitly stating the desired analog input channel prior to each conversion. The first and last channel numbers are written to this port. Each successive ADC trigger operates on the next analog input channel, within the range of first-to-last. After the last channel, the

TABLE 11-1
DAS-16 I/O Ports

PORT LOCATION	FUNCTION	READ/WRITE
Base Address + 0	ADC Low Byte Start ADC	R W
Base Address + 1	ADC High Byte	R
Base Address + 2	MUX Scan Control	R / W
Base Address + 3	Digital I/O Out (4 bits) Digital I/O In (4 bits)	W R
Base Address + 4	DAC 0 Low Byte	W
Base Address + 5	DAC 0 High Byte	W
Base Address + 6	DAC 1 Low Byte	W
Base Address + 7	DAC 1 High Byte	W
Base Address + 8	DAS-16 Status	R
Base Address + 9	DAS-16 Control	R / W
Base Address + 10	Counter Enable (2 bits)	W
Base Address + 11	Not Used	N / A
Base Address + 12	Counter 0	R / W
Base Address + 13	Counter 1	R / W
Base Address + 14	Counter 2	R / W
Base Address + 15	Counter Control	W

selection rolls around to the first one again. This feature is extremely handy if you use multiple analog inputs with a hardware clock trigger. Once the software sets up the card to convert the desired ADC channels, all it has to do is keep reading the data until the required number of readings have been accumulated. Of course, the analog input channels used must be consecutive numbers. By contrast, a board such as the KPCI-3108, with a channel-gain queue, does not have the limitation of consecutive channel numbers.

The analog output ports (at base +4 through base +7) are write-only, requiring two 8-bit ports to access the complete 12-bit DAC word. The DAC output is not changed until both bytes have been written, preventing a glitch in the DAC output when one byte is an old value and the other is a new value.

The eight digital I/O lines of the DAS-16 are configured as a 4-bit input port and a 4-bit output port. By writing to the digital I/O port (at base +3), the four output lines are latched. Reading from the digital I/O port reflects the state of the four input lines. Two of the input lines are also used for special ADC trigger and counter gate functions.

The status port (at base + 8) is read-only. It contains information about the ADC and interrupt status. This information includes whether the ADC is busy or has valid data and if the analog inputs are single-ended or differential as well as unipolar or bipolar. This allows software to check the state of the hardware switches. In addition, the mux channel for the next conversion is read here, along with the status of the board's interrupt generator.

The control port (at base +9) is both a read and write address and determines the operating modes of the DAS-16. It is used to enable or disable interrupt generation and select the hardware interrupt level to use (ISA IRQ 2–IRQ 7). The control port can also enable DMA transfers (if enabled, the PC's DMA controller must be properly initialized). In addition, this port determines the source of the ADC conversion trigger: software only, internal timer control, or external trigger control.

The counter-enable port (at base +10), along with the four 8254 ports (at base +12 through base +15), controls operation of the three counter/timer channels. Counters 1 and 2 are cascaded, so that counting periods ranging from microseconds to hours can be used to periodically trigger the ADC.

As an example of software for the DAS-16, here is a small segment of an MS-DOS BASIC program. This code triggers an ADC conversion (via software) for a board at base address BASADR, returns the 12-bit result in DAT and the analog input channel number in CHANL, and then displays the result:

```
10 BASADR = &H300                             'Default Base Address
20 OUT BASADR%,0                              'Start ADC conversion
30 IF INP(BASADR%+8)>=&H80 THEN GOTO 30       'Conversion Done?
40 LOW% = INP(BASADR%)                        'Read low byte
50 HI% = INP(BASADR%+1)                       'Read high byte
60 DAT% = 16 * HI% + INT (LOW%/16)            '12-bit data read
70 CHANL% = LOW% AND &H0F                     'Analog channel number
80 PRINT "For Channel #";CHANL%;", ADC Value = ";DAT%
```

Note that the variable names end in % to signify they are integers (as opposed to floating-point numbers).

In Microsoft C for MS-DOS, a similar program would look like

```
#include <conio>              /* for inp() & outp() functions */
#include <stdio>              /* for printf() */
#define BASADR 0x300          /* default base address = 300h */
#define ADCLOW BASADR         /* address of ADC low byte */
#define ADCHI BASADR+1        /* address or ADC high byte */
#define ADCSTAT BASADR+8      /* address of status port */
main()
   {                               /* start of program */
   int dat, low, high, chanl;      /* declare integers */
   outp(BASADR,0);                 /* start conversion */
   while(inp(ADCSTAT)>=0x80);      /* wait for end of conversion */
low = inp(ADCLOW);            /* read low byte */
high = inp(ADCHI);            /* read high byte */
dat = 16 * high + low / 16; /* full 12-bit reading */
chanl = low & 0x0f;  /* mask bits to get channel number */
printf("\nFor Channel #%d, ADC Value = %d\n",chanl,dat);
}                           /* end of program */
```

This program may look more verbose than the BASIC version, but as the size and complexity of a program increase, the extra overhead of C is minimal compared to its flexibility, speed, and power.

This examination of the Keithley DAS-16 board has shown us how a typical, older ISA data acquisition card operates. Besides conventional plug-in cards, Keithley manufactures many stand-alone instruments and specialized data acquisition products, such as their ADWIN series for real-time response (under 500 nsec). The devices in this series range from plug-in PC cards to expandable instrument racks. They contain their own microprocessors so these devices can operate independently of the controlling PC's operating system. Their networking capabilities (including Ethernet support) make them a good choice for remote data acquisition and control applications.

One other interesting Keithley product line is their PC Instrument Products (PCIP): PC plug-in boards that emulate conventional test instruments. They are ISA cards that include a digital storage oscilloscope (PCIP-SCOPE), a digital multimeter (PCIP-DMMA), an arbitrary waveform generator (PCIP-AWFG), and a frequency counter (PCIP-CNTR). These virtual instruments can operate in either a manual mode (via virtual control panels, on the PC monitor) or in an automated mode via a DOS or Windows program.

This concludes our look at some of Keithley Instrument's PC-based data acquisition products. For up-to-date information, visit their Web site (www.keithley.com).

11.1.2 Data Translation Inc.

Data Translation Inc. is another leading producer of data acquisition boards for PCs. Their product line supports both ISA and PCI platforms as well as USB and PCMCIA interfaces. They also provide some software for use with their data acquisition products. In addition, Data Translation produces image-capture boards (frame grabbers and video processors) for ISA and PCI PCs.

Data Translation's data acquisition card product line ranges from low-cost, low-speed, multifunction cards, such as the DT01-EZ ISA board, to high-performance PCI cards, such as the DT3010 series. The DT01-EZ is a good general-purpose ISA data acquisition card, with 12-bit resolution and 16 single-ended or eight differential input lines. It has a maximum conversion rate of 27,500 samples/sec and analog input ranges of 1.25 V through 10 V, full-scale (both unipolar and bipolar ranges). It has two 12-bit analog output channels with a max conversion rate of 29,500 samples/sec and 16 digital I/O lines. The DT01-EZ has a programmable pacer clock to initiate repeated conversions but no general-purpose, user-accessible counter/timers.

The PCI data acquisition cards in the DT3010 series contain either a 12-bit or a 16-bit ADC with up to 16 differential or 32 single-ended analog input channels. The maximum ADC conversion rate for the 12-bit boards is 1.25 million samples/sec. For the 16-bit model (DT3016) this maximum rate is 250,000 samples/sec. The cards in this series have two analog output channels (DACs) with the same resolution as their ADC (12-bit or 16-bit). These analog outputs have maximum conversion rates of either 500,000 samples/sec for the 12-bit boards or 200,000 samples/sec for the 16-bit board. In this series, the analog input amplitude ranges vary from 1.25 V through 10 V, full-scale (both unipolar and bipolar). Analog outputs have a bipolar range of ±10 V. The analog outputs also have a FIFO, for outputting repetitive waveforms, up to 32,768 samples long.

The DT3010 series boards all have 16 digital I/O lines, configured as two programmable 8-bit ports. Digital inputs can be read as part of the analog channel list, providing an accurate time stamp relative to the analog readings. There are also two dedicated programmable digital outputs that can indicate when a particular analog channel is read, providing synchronization to external equipment. These boards have a programmable pacer clock to initiate repeated data conversions as well as four 16-bit counter/timers.

Data Translation has another interesting ISA data acquisition product line, their DT2831 series. The boards in this series are very similar to other Data Translation ISA boards with one important exception. Once the base address of a DT2831 card has been selected, all its data acquisition parameters are set by software only. There is no need to change switch or jumper settings to modify parameters such as analog input gain, single-ended versus differential analog inputs, analog voltage ranges, DMA channel, interrupt channel, or even ADC and DAC calibration. These boards support either 12-bit or 16-bit analog I/O. The maximum analog input conversion rate is 250,000 samples/sec for 12-bit boards and 160,000 samples/sec for 16-bit boards. The analog output conversion rates are 130,000 samples/sec for 12-bit DACs and 100,000 samples/sec for 16-bit DACs, with either a unipolar (0–10 V) or bipolar (±10 V) output range. The DT2831 boards have eight digital I/O lines, configured as a single 8-bit port. They also have two counter/timer channels. These boards support hardware interrupts and two DMA channels. They also have available simultaneous sample-and-hold inputs for sampling all analog inputs at the same time.

Another unique Data Translation product line is their Fulcrum (DT3800) series of intelligent data acquisition boards. These ISA cards are controlled by an on-board Texas Instruments TMS320C40 DSP (digital signal processor), which is a 32-bit floating-point CPU. A DT3800-series board can operate independently of its host PC, since the TMS320C40 controls all of its operations. All configuration and calibration is done via software controls from the host PC. Cards in this series have 12-bit ADC inputs with conversion rates up to 1 million samples/sec and 16-bit ADCs as fast as 160,000 samples/sec.

The Fulcrum series boards have two high-speed, 16-bit analog output channels with data rates up to 200,000 samples/sec and software-selectable settings. They also have 16 digital I/O lines with speeds up to 3.3 MHz. These digital lines are partitioned into two 8-bit ports. The two 16-bit on-board counter/timers are 8254-based, running from an internal 10-MHz clock. These cards run SPOX, a DSP real-time, multitasking operating system. Application software is developed using the DSPLAB developer's kit, which runs on the host PC.

Data Translation, along with most other major manufacturers, produces screw-terminal and signal-conditioning panels for their data acquisition cards. These panels simplify connecting external devices to the data acquisition cards. Some common signal-conditioning functions are available, such as antialiasing filters and cold-junction compensation for thermocouples. If a thermocouple is directly connected to the appropriate panel, the analog signal sent to the data acquisition card can be directly read as degrees (temperature) without additional circuitry or complex software.

Data Translation also manufactures data acquisition products for PCMCIA and USB interfaces. Their PCMCIA products include the DT7100 series. The DT7101 PCMCIA card has a 12-bit ADC with eight single-ended (or four differential) inputs and a maximum conversion rate of 100,000 samples/sec. It also has four digital I/O lines (two inputs and two outputs). The DT7102 card has a 12-bit ADC with 16 single-ended (or eight differential) inputs and a maximum conversion rate of 200,000 samples/sec. The DT7102 also has two 12-bit analog outputs with maximum rates of 50,000 samples/sec, as well as six digital I/O lines (two inputs and four outputs).

Data Translation's USB data acquisition products include their DT9800 series. These modules are fully USB 1.1 compliant, with hot-swap and plug-and-play capabilities. One of the major advantages of using external USB data acquisition devices, such as these, is that they can provide a much lower noise level than PCI or ISA cards that reside inside a PC (where the electronic noise from the power supply and motherboard is fairly high). The current disadvantage with USB 1.1 devices is their limited top speed of 12 Mbits/sec

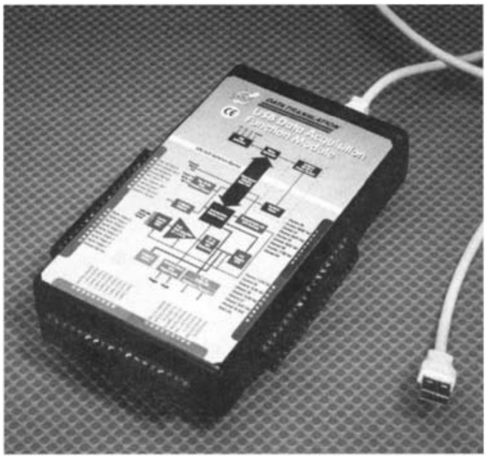


Figure 11-8 Data Translation DT9800 series USB data acquisition module. (Courtesy of Data Translation, Inc.)

(or 1.5 Mbytes/sec), with typical sustained transfer rates closer to 100 Kbytes/sec. USB 2.0, with its 40× speed increase, should eliminate this limitation. Figure 11-8 shows a typical Data Translation USB Module.

An example of a multifunction USB device is the DT9802. This module has a 12-bit ADC with 16 single-ended (or eight differential) inputs that range from 1.25 V to 10 V full-scale and a maximum conversion rate of 100,000 samples/sec. It has two 12-bit analog outputs, 16 digital I/O lines (eight inputs and eight outputs), and two 16-bit counter/timers.

Another Data Translation USB device is the DT9821, which has four independent ADCs with a maximum resolution of 24 bits at conversion rates of 7.5 samples/sec or slower. At the maximum conversion rate of 960 samples/sec, the ADC resolution is reduced to 16 bits. With an input range varying from approximately 40 mV to 2.5 V full-scale, even at 16-bits the ADC can resolve inputs less than 1-μV (for 1 LSB).

As with most major data acquisition vendors, Data Translation bundles software with its hardware products. With PCI cards or USB modules, this manufacturer includes its Omni CD: a collection of drivers, development tools, and basic applications for its data acquisition boards that runs under MS Windows 98/Me/2000. For example, the Scope application requires no programming and allows you to acquire data in either a high-speed oscilloscope mode or a strip-chart mode. Quick Data Acq is a menu-driven application that provides verification of board operations and allows you to collect, display, and save acquired data. Source code for Quick Data Acq (written in Microsoft Visual Basic) is also included, allowing you to customize the application.

Data Translation has other hardware products, such as the DATAX modular data acquisition system that connects to a host PC via USB. This system is optimized for expandability and signal conditioning via its standalone, 16-slot chassis. Data Translation also produces some virtual instruments, such as their DT2040 series of PCI-based digital multimeter cards.

This concludes our discussion of Data Translation's PC-based data acquisition products. For current information, you can view their Web site (www.datatranslation.com).

11.1.3 National Instruments

National Instruments had primarily been a leading manufacturer of GPIB controller hardware and software products for PCs and other computer platforms. In the decade following the first edition of this book, National Instruments has also become a major manufacturer of data acquisition products for PCs and industrial computer platforms, such as VME and Compact PCI. They also produce data acquisition hardware for PCMCIA, USB, and IEEE-1394 interfaces. Additional product lines include motion control and image capture products.

Besides hardware, National Instruments produces software products, most notably, LabVIEW. LabVIEW, a data acquisition programming language, is so popular that it can be used with other manufacturers' hardware products. For example, Data Translation and Keithley Instruments provide software (in the form of virtual instruments) that allow their boards to work under LabVIEW. We will discuss LabVIEW in greater detail later in this chapter (see Section 11.2.2).

National Instruments produces a wide range of PCI and ISA data acquisition cards for PCs. The low-cost, multifunction PCI-6023E is a PCI card with 16 single-ended (or eight differential) inputs to a 12-bit ADC, having a maximum conversion rate of 200,000 samples/sec. It also has eight digital I/O lines and two 24-bit counter/timers, but no analog outputs. The AT-MIO-16E-1 is an ISA card with 16 single-ended (or 8 differential) inputs to a 12-bit ADC with a maximum conversion rate of 1.25 million samples/sec (Msamples/sec). This board has eight digital I/O lines, two 24-bit counter/timers, and two 12-bit analog outputs with an update rate of 1 Msample/sec. It is also fully plug-and-play compatible for simple installation and configuration.

National Instruments has a family of high-speed digitizers, with analog inputs only. The NI-5911 is a PCI card having a single analog input channel and an 8-bit ADC with a maximum conversion rate of 100 Msamples/sec in real-time mode. For repetitive signals, using its random interleaved sampling mode, it has a conversion rate up to 1 gigasample/sec (Gsample/sec). This card has either 4 or 16 Mbytes of on-board memory for temporary data storage. A special feature of the NI-5911 is the flexible resolution mode that uses a DSP technique similar to delta-sigma conversion to increase the effective ADC resolution at lower sampling rates (and lower bandwidth). For example, at 5 Msamples/sec the card has 14 bits of effective resolution. This increases to 21 bits at a conversion rate of 10,000 samples/sec.

Another high-performance National Instruments product line is the NI-611X family of simultaneous-sampling, multifunction data acquisition boards. The NI-6110 is a PCI card with four 12-bit analog inputs and a maximum conversion rate of 5 Msamples/sec. It has two 16-bit analog output channels with a 4-Msamples/sec maximum rate, eight digital I/O lines, and two 24-bit counter/timers. Unlike conventional multifunction cards that use one ADC and an input multiplexer, the NI-6110 (and other family members) has an ADC for each input channel, allowing simultaneous sampling on all inputs. This is essential when an accurate relative phase or time measurement needs to be made.

National Instruments also has PCMCIA and USB versions of some of its data acquisition products. For example, the NI-6020E, a 12-bit, 100,000 samples/sec multifunction data acquisition device, is available as either an ISA board or a USB module (the DAQPad-6020E). Another example, the 6024E is similar to the NI-6023E (12-bit ADC, 200,000 samples/sec) except it also includes two 12-bit analog outputs. The 6024E is available as either a PCI board (the PCI-6024E) or a PCMCIA card (the DAQCard-6024E).

A National Instruments product even uses the IEEE-1394 bus: the NI-6070E family (which includes the AT-MIO-16E-1). This is a 12-bit ADC with 16 single-ended (or 8 differential) inputs and a maximum conversion rate of 1.25 Msamples/sec. It has two 12-bit analog outputs with a maximum rate of 1 Msamples/sec, eight digital I/O lines, and two 24-bit counter/timers. The IEEE-1394 version is the DAQPad-6070E, which is a stand-alone module, similar to USB data acquisition devices. Of course, to use this device, a PC must have an IEEE-1394 interface (usually as an add-in card) and appropriate software support. Note that some older versions of 32-bit MS Windows (such as Windows 95 and Windows NT) are not suitable for IEEE-1394.

Other National Instrument product lines include stand-alone instrumentation chassis, based on Compact PCI cards. Many of their data acquisition PCI cards are also available in Compact PCI versions.

As with other major hardware vendors, National Instruments bundles basic software with their data acquisition products. For example, their E series of multifunction devices (such as the PCI-6023E card) come with NI-DAQ driver software to simplify writing your own application program. They also include Measurement and Automation Explorer software to configure and test the hardware.

This concludes our brief survey of National Instruments' data acquisition products. As with other manufacturers, their Web site (www.ni.com) is a good source for current product information.

11.1.4 Other Hardware Manufacturers

A large number of other PC-based data acquisition hardware manufacturers are listed in the Appendix. Without going into much detail, we will look at a few more of them.

Scientific Solutions, Inc. The first manufacturer of data acquisition boards for IBM PCs was Scientific Solutions, Inc. Their current product line supports both ISA and PCI buses and includes multifunction data acquisition boards, digital I/O boards, and GPIB interface cards.

Scientific Solutions' Lab Tender ISA board is a new, software-compatible version of the original 8-bit Lab Tender, introduced in 1981. It contains a 16-bit ADC with 32 single-ended (or 16 differential) inputs, having a range of ±5 V and a maximum conversion rate of 50,000 samples/sec. The Lab Tender has a 16-bit DAC, multiplexed with 16 sample-and-hold outputs. If more than one output at a time is in use, they must be periodically refreshed (their LabPac 32 driver software takes care of this automatically). This board has 24 digital I/O lines, configured as two 8-bit and two 4-bit ports, controlled by 8255A-compatible hardware. It also has five counter/timer channels, with AM9513A-compatible hardware. The Lab Tender supports hardware interrupts.

Scientific Solutions also produces the multifunction Lab Master DMA that consists of an ISA board and an external analog box (containing the ADC and analog input circuitry). This produces very low-noise measurements. The Lab Master DMA contains a 12-bit or 16-bit ADC with a maximum conversion rate of either 50,000 or 160,000 samples/sec and 16 single-ended (or 8 differential) analog inputs. The analog input range can either be unipolar or bipolar, and the gain can be adjusted via hardware (through jumpers) or software. The Lab Master DMA has two independent 12-bit DACs with five selectable output ranges and a maximum conversion rate of 200,000 samples/sec.

In addition it contains 24 digital I/O lines and five counter/timers, as the Lab Tender does. The Lab Master DMA supports hardware interrupts as well as DMA data transfers.

Scientific Solutions PCI product is the Lab Master Pro PCI. This multifunction board has a 16-bit ADC with 16 analog inputs, expandable to 256, and a maximum conversion rate of 333,000 samples/sec. It has two 16-bit analog outputs with rates up to 500,000 samples/sec. The Lab Master Pro PCI also has five 16-bit counter/timers and 16 digital I/O lines. It supports PCI bus mastering for high-speed data transfers. The card also has an onboard FIFO to buffer ADC or DAC data.

Scientific Solutions also provides software support for its boards. LabPac is a memory-resident driver that runs under MS-DOS. Any standard DOS programming language can access its functions, such as analog input, analog output, and digital I/O.

LabPac 32 is a 32-bit application programming interface (API) for Windows 95/98/NT/2000. Working in conjunction with a board-specific device driver, LabPac 32 functions access the target device's features. It supports most standard 32-bit MS Windows programming languages, including Java, Visual C/C++, Visual Basic, and Borland C/C++.

As with other vendors we have surveyed, up-to-date product information is available at Scientific Solutions' Web site (www.labmaster.com).

Intelligent Instrumentation Intelligent Instrumentation (formerly Burr-Brown/Intelligent Instrumentation) produces a variety of data acquisition products for PCs. These include both plug-in cards and remote data acquisition products (with a strong emphasis on Ethernet). Intelligent Instrumentation manufactures a series of plug-in ISA boards with dedicated functions as well as those with modular features, all part of their PCI-20000 system (*please note that boards in this series, despite its name, are only for the ISA bus and not the PCI bus*). This product line stresses the use of modular boards, based on the PCI-20098C and PCI-20047C series. Multifunction dedicated boards are available as well as digital I/O, analog input, and analog output (as shown in Figure 11-1). Termination panels are also available.

The carrier boards used with expansion modules act as multifunction cards, plugging into a PC's ISA slot. Some carrier boards require modules for analog I/O, such as the PCI-20041C series that contains only digital I/O. The PCI-20098C is considered a multifunction carrier board, containing analog I/O, digital I/O, and counter/timers as well as supporting additional modules.

The add-in modules for these carrier boards include various analog-input options, such as high gain (up to 25 mV, full-scale), high resolution (16 bits at

85,000 samples/sec), and analog input expansion (32 additional single-ended or 16 additional differential inputs). The analog output modules offer 12-bit or 16-bit resolution, with maximum conversion rates of 80,000 samples/sec. A digital I/O module offers 32 lines, accessible as four 8-bit ports. Other modules with special functions include a counter/timer board, a sample-and-hold board, and a trigger/alarm board.

Intelligent Instrumentation's dedicated multifunction ISA boards include the PCI-2048W series, with a 12-bit ADC and 16 single-ended (or 8 differential) inputs and two 12-bit analog outputs, running as fast as 100,000 samples/sec. These cards have 16 digital I/O lines (eight inputs and eight outputs) and a 16-bit counter. They also support DMA transfers. The PCI-470W series contains high-speed transient capture ISA boards with an 8-bit or 12-bit ADC and acquisition rates up to 60 Msamples/sec. These cards contain on-board memory for data storage (up to 512 Kbytes), since their analog input data rate is much faster than ISA bus speeds.

Intelligent Instrumentation has a USB data acquisition system (UDAS) which supports 100,000 samples/sec analog I/O at 12-bit resolution. It also has a parallel port data acquisition system, the DAASport series. This is especially useful for laptop PCs. It supports both standard and enhanced (EPP) parallel ports with analog rates up to 100,000 samples/sec at 12-bit resolution.

Another interesting Intelligent Instrumentation product line is their Ethernet data acquisition system (EDAS). This family consists of stand-alone boxes that interface analog, digital, and serial communications I/O to 10BASE-T Ethernet. For example, the EDAS-1002E is a multifunction unit that has a 12-bit ADC and 16 single-ended (or 8 differential) inputs, with a maximum conversion rate of 100,000 samples/sec. It also has two 12-bit analog outputs and 16 digital I/O lines along with a 16-bit counter. Additionally, the EDAS-1002E has an RS-232 port and an optional RS-485 port. The EDAS system is especially well suited for automated industrial applications where most sensors and controllers use a standard serial interface.

More information about Intelligent Instrumentation's products is available on their Web site (www.instrument.com).

Gage Applied, Inc. Gage Applied Inc. (now a subsidiary of Tektronix) specializes in high-speed, high-performance data acquisition products for PCI and ISA bus PCs. Their CompuScope line of analog input cards includes some of the fastest digitizers currently available.

For example, the CS85G is a PCI analog digitizer with a maximum conversion rate of 5 Gsamples/sec on two simultaneous input channels, at 8-bit resolution. The input channels have 500 MHz analog bandwidth. Input gain is software selectable with an input range of ±20 mV to ±20 V, full-scale.

The card has an on-board storage memory up to 10,000 samples per channel. This means that at 5 Gsamples/sec, the maximum acquisition window is 2 μsec long.

The CS14100 is another analog input PCI card with 14-bit resolution and a maximum conversion rate of 100 Msamples/sec, in single-channel mode. This card contains two 50-Msamples/sec ADCs that can provide simultaneous sampling for two input channels. In the single-channel mode, the two ADCs use a "ping-pong" scheme to produce a doubled conversion rate of 100 Msamples/sec.

Gage also produces high-speed ISA analog input cards, but these are not nearly as fast as Gage's PCI products. For example, the CS2125 has an 8-bit ADC with a maximum conversion rate of 250 Msamples/sec for one of its two input channels. It has on-board memory up to 8 Mbytes and a data transfer rate as high as 2 Mbytes/sec into PC memory (via the ISA bus).

Gage's product lines include analog output, digital input, and digital output boards. The CompuGen 1100 is a 12-bit PCI analog output card with a maximum rate of 80 Msamples/sec and on-board memory up to 16 million samples. It is used primarily as an arbitrary waveform generator (a virtual instrument). The CS3200 is a 32-bit PCI digital input card that can run at rates up to 100 MHz, with on-board memory as large as 2 Gbytes. The CompuGen 3250 is a 32-bit PCI digital output card with data rates as fast as 50 MHz. It has up to 8 million samples of on-board memory. The 3250 is especially useful as a high-speed pattern generator to test digital systems.

Gage also produces software to support their products. GageScope software operates as a virtual oscilloscope with CompuScope cards. It is an interactive, graphics environment that requires no programming. GageScope acquires, saves, and displays digitized data. It also has analysis features, such as signal averaging, correlations, and FFTs. GageScope is available for both MS-DOS and MS Windows 95/98/NT/2000.

In addition, Gage has a software development kit (SDK) supporting its CompuScope and CompuGen cards, for users who want to write their own software. These SDKs support not only C/C++ programs under DOS and Windows 95/98/NT/2000, but also programs written for MATLAB and LabVIEW (see Section 11.2, later in this chapter, for more information on MATLAB and LabVIEW).

More information about Gage's products is available at their Web site (www.gage-applied.com).

Microstar Laboratories Microstar Laboratories specializes in intelligent data acquisition boards. Their data acquisition processor (DAP) product line consists of PCI and ISA cards that contain an on-board microprocessor. This local processor runs its own operating system (DAPL) and gives the board the capabilities it needs for real-time processing and control applications as well as handling large numbers of analog I/O channels.

An example of an ISA card in this series is the DAP 32009/415 running an Intel 80486DX4 processor at 96 MHz. This card has 16 analog input channels (expandable to 512) and a maximum conversion rate of 769,000 samples/sec at 12-bit resolution. It has 2 analog outputs (expandable to 66) with a top data rate of 833,000 samples/sec. The board also has 16 digital inputs (expandable to 128) and 16 digital outputs (expandable to 1024), which have an update rate of 1.66 MHz. The board can transfer data to PC memory through the ISA bus as fast as 909,000 samples/sec.

Microstar Laboratories' PCI products include the DAP 52009/626, running an AMD K6 III+ processor at 400 MHz. This card has 16 analog inputs (expandable to 512) with a 14-bit ADC converting up to 800,000 samples/sec. It also has 2 analog outputs, with an update rate of 833,000 samples/sec. The board contains 16 digital inputs (expandable to 128) and 16 digital outputs (expandable to 1024) with update rates of 1.66 MHz. It can transfer data to PC memory through the PCI bus as fast as 1.66 Msamples/sec.

Microstar Laboratories provides software support for its hardware products. DAPview for Windows runs on a PC and implements data acquisition and control functions without requiring any programming. Microstar Windows Toolkit (MSWTK) allows you to write your own programs to run a DAP board, using most common Windows programming languages, such as Microsoft Visual C/C++, Borland C++, and Microsoft Visual Basic. Microstar Laboratories also has a developers' toolkit for DAPL (MSDTD), to generate custom commands for the on-board processor.

Other Microstar Laboratories products include signal processing equipment, such as antialiasing filter boards or chassis for special-function modules (such as analog isolation units). More information is available at their Web site (www.mstarlabs.com).

Omega Omega is a major manufacturer and distributor of industrial measurement and control equipment, including sensors and data acquisition products (both hardware and software). Their data acquisition lines include ISA and PCI plug-in cards, signal conditioning systems, and stand-alone data acquisition systems based on serial, GPIB, and Ethernet interfaces. They also supply software to support their hardware products.

Omega is a good one-stop source for data acquisition products, including a very broad range of sensors. They publish a large set of catalogs. Additional information is available at their Web site (www.omega.com).

This concludes our overview of commercial data acquisition hardware for PCs. Please refer to the Appendix for a more comprehensive listing of manufacturers. Next we will look at commercial software products.

11.2 Commercial Data Acquisition Software Products

The availability of abundant, powerful, easy-to-use software is probably the strongest incentive to employ a PC as the platform for a data acquisition system. In many ways, the variations in data acquisition software products mirror those in hardware products. These software products vary from simple drivers, tied to a particular manufacturer's boards and dedicated to basic data collection tasks (analogous to single-function hardware), to complete data acquisition/analysis/display software packages, supporting a broad range of hardware products (analogous to multifunction cards).

As we discussed previously, a software driver is a special program that acts as an interface to a particular hardware device. It is used in conjunction with other software: the controlling program. The driver handles all the low-level interfacing, such as reading from and writing to a data acquisition board's I/O ports and memory locations. It presents higher-level commands to the controlling program, such as to initiate an ADC conversion on the selected channel and return the result. The driver takes care of all the I/O port commands, simplifying the controlling program. In addition, when running a secure 32-bit operating system, such as MS Windows NT/2000, the only way to access a board (or any hardware device) is through a software driver, since direct hardware access is not allowed.

In nonsecure operating systems (such as DOS) a driver for a data acquisition card is an aid to writing a program that uses the card. It is specific not only to the board it supports, but also to the operating system and sometimes the programming language it is used with. By itself, a driver is not a full-blown software package; it is only a tool used to create the complete program. That is why manufacturers offer different drivers for different operating systems (DOS, Windows 95/98/NT) and programming languages (BASIC, C, Pascal under DOS or MS Visual C/C++, MS Visual Basic under Windows). Commonly, a manufacturer supplies a driver that allows Windows 95/98 applications to work with the data acquisition board. Other drivers, including those for DOS systems (for older, ISA hardware), are usually available at modest cost. Sometimes different drivers are needed for different compiler manufacturers supporting the same language, such as Microsoft C and Borland's Turbo C under DOS. See Chapter 13 for a discussion of programming languages.

DOS software drivers are usually supplied in one of two forms when supporting a compiled language: a memory-resident module or a linkable module. A memory-resident module is supplied as a small program (typically with a .COM file name extension). The program is run, making the driver memory-resident. Occasionally, the memory-resident driver is provided as a .SYS file and installed as a DOS device driver, via the CONFIG.SYS file. Once this driver is loaded into memory, another program can call its functions, using a software interrupt. A Windows driver must be loaded using its install program, in the same manner as any Windows application software.

A DOS driver in the form of a linkable module is usually supplied as a .OBJ file. The user-written control program makes calls to functions in this module. After this control program is compiled, it is linked with the driver file, producing the complete executable program as an .EXE file.

In Microsoft Windows, executable files also have an EXE extension. Often, collections of callable Windows functions are distributed as dynamic-link library (DLL) files. These can be accessed by most Windows programming languages.

Most hardware vendors bundle a variety of software with their products, including drivers for the appropriate operating system (usually MS Windows 95/98/NT/2000) and a software development kit (to write your own programs using common programming languages). In addition, many include diagnostics and some simple applications to verify board operations and perform basic data acquisition tasks (such as a virtual oscilloscope or chart recorder).

Fortunately, you do not have to be a programmer to use most PC-based data acquisition systems. Many software manufacturers (and more and more hardware manufacturers) produce easy to use, menu-driven, and graphics-based data acquisition and support programs. These software products may have one or more general functions: data acquisition, data analysis, and data display. Several high-end, integrated software packages provide all three functions, in varying degrees.

An important trade-off in all types of software packages is whether the user interface is graphics or icon-based versus command-driven. In simple, icon-based (graphics display) software, the user chooses among many options presented by the program. These types of programs are the easiest to learn and use. Their drawback can be low flexibility. The only functions (or combination of functions) available are those built into the selection system or enhanced by add-ins. If you need to do something different, you may be out of luck, unless it is added as a new feature in a product upgrade. An example of this type of program is a virtual oscilloscope application, bundled with a data acquisition card.

In contrast, command-driven software (usually text-based) is harder to use and not as intuitive to learn. It does, however, offer the maximum flexibility. All available commands can be used with any possible combination of parameters. In addition, most command-driven software packages allow for some level of programming. This can be as simple as stringing a series of commands together, as with a macro, or as complex as a true program with looping, conditional execution, and a wide range of data types. Some software products, such as MATLAB, are essentially high-level programming languages, optimized for analysis or data acquisition applications.

There are also software products that act as graphics-based data acquisition languages, such as LABTECH NOTEBOOK and LabVIEW. These types of software packages offer the best of both worlds: ease of use combined with flexibility and extendibility. You program by connecting icons instead of writing lines of code.

For older DOS software products, manufacturers of command-driven software often included a menu-based shell with their packages. This shell acted as an easily learned command interface, buffering the user from the command-driven language itself. Once someone became familiar with the system and "outgrew" the menu shell, they could directly use the command-driven interface. This was usually a flexible approach. The main drawbacks were slower performance and a larger program, due to the extra shell layer in the software.

Figure 11-9 shows a simple comparison of DOS-based, menu-driven versus command-based interfaces. To save a data array as a plot file, using

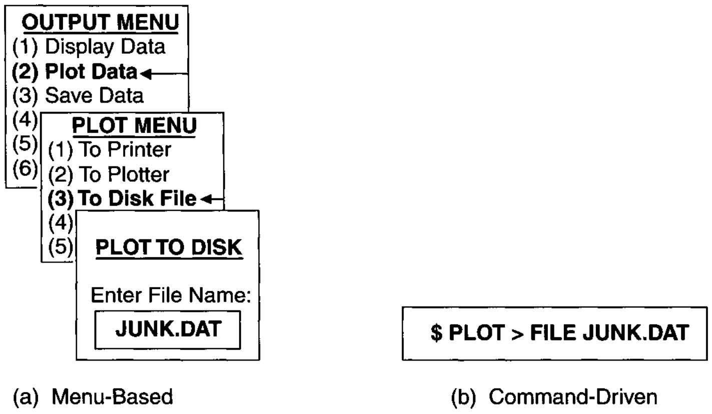


Figure 11-9 Comparison of menu-based and command-driven user interfaces.

the menu system (Figure 11-9a) you would choose selections from several overlapping windows, and finally specify the output file name. In the command-driven interface (Figure 11-9b), a single one-line command performs the same operation.

Software with data acquisition functions is absolutely necessary for all data acquisition systems. This type of software enables the user to set up the data acquisition board's parameters, record the desired amount of data at a specified rate for a specified time, and store the resulting data in a file for future analysis and display. In addition, real-time graphics display of data as it is acquired is extremely useful. This software package must support the particular data acquisition board used as well as optional hardware accessories (such as multiplexers and signal conditioners) and must be appropriate for your PC's operating system and resources.

This support takes many forms. There are important factors to consider when comparing data acquisition software packages, especially which features are present and how they are implemented. Full support of all hardware features is mandatory, especially maximum ADC conversion rate, interrupt support, and DMA support for ISA or bus-mastering for PCI. Support of multiple boards is certainly desirable. Most software packages include a real-time graphical display of the acquired data, in either an *x–y* graph or a strip-chart format. It is common for data acquisition software to scale the raw input values (usually signed integer format) from an analog input voltage to the physical units being measured by the sensors. For example, the millivolt readings from a thermocouple would be stored as degrees Celsius values, or the voltage output of a LVDT displacement sensor would be stored as millimeter values.

The format of the stored data (from an ADC) as well as how it gets stored are other important factors. Some software products will store data in an ASCII format, which is easy to print out and read. ASCII data is readily transferred to other software packages, such as spreadsheets, for analysis and display. The drawback is that ASCII data takes up more disk space than equivalent binary data and requires more time to be stored and transferred. For relatively slow data rates or small amounts of data, this is not a significant problem. If the data rates get very high (at or above approximately 1 Mbytes/sec, for example) or the produced data gets very large, a binary storage format is indicated. Of course, the large hard drives in current PCs may make saving disk space appear unnecessary. However, if you want to back up or archive your data, large files still result in longer processing times and higher costs.

In some cases, a data compression format may be needed, although this is not often done in real time (PKZIP or WINZIP are often used to compress stored data files). A binary format may be the same as the signed integer data produced by the data acquisition board, or it may be a more sophisticated data structure, including scaling and coordinate information, as when it represents a data array. Some software products may use a proprietary format. Usually, the data format is specified in the user's manual. This allows you to transfer the data into another commercial or custom program.

Many standard data formats are supported by commercial data acquisition programs, besides just ASCII. One of these is the Microsoft Excel worksheet format. Spreadsheet programs, such as Excel and Lotus, are very popular for numerical analysis and display in business environments. Some data acquisition-only software packages produce data outputs in this worksheet format, allowing the user to do analysis and output using Excel or a compatible spreadsheet program. Other data acquisition software packages that provide analysis functions can both read and write data in this worksheet format—usually as a special ASCII file called comma-separated variables (CSV). In addition, some products directly link to Excel or are Excel add-ins that provide data acquisition functions within the spreadsheet environment.

One very specialized type of data acquisition software package used under DOS with ISA cards was the data streamer. This was used to acquire data and store it in a hard disk file at the maximum rate allowed by the hardware. This maximum rate, sometimes referred to as throughput, was determined by the speed of the data acquisition board's ADC and the PC's maximum disk data-transfer rate. If the data acquisition card had on-board memory, the PC's speed was not a factor, up to the board's storage capacity. A data streamer supported "no-frills," high-speed data acquisition. It was most useful with DMA hardware and provided little or no data processing.

Even with today's PCI-based PCs and fast IDE hard drives, high-speed transfer of large amounts of real-time data to a disk drive may not keep up with a data acquisition card's conversion rate. For these type of high-performance streaming applications (typically well above 1 Mbytes/sec), special SCSI hard drives are often used.

Commercially available data analysis software is another important part of a PC-based data acquisition system. These analysis packages can be used by themselves, with data internally generated or imported from other programs (data acquisition software, among others). Some of them also include data acquisition functions. Nearly all Windows-based data analysis software packages include output functions for video display as well as printer and plotter support. The capabilities of these packages vary from general mathematical operations to DSP functions (such as FFTs). Some of these data analysis products are not specifically aimed at data acquisition, scientific, or engineering applications. General-purpose business programs, such as Lotus and Excel spreadsheets, are examples of this. However, they are still very useful analysis tools for acquired data. And Excel even has add-in features that include some engineering functions, such as FFTs.

Some data-analysis software packages consist of function libraries for a particular programming language. They are analogous to hardware drivers, as they are only useful to someone creating a custom program. Of course, these libraries are not tied to any particular data acquisition hardware. They may require a specific data format and interface only to particular language compilers. We will not dwell on these libraries here, as they are mostly of interest to software developers.

Software only for data display also exists, although most data analysis programs include extensive display capabilities. Data display usually consists of producing a graph or chart on a video screen, a printer, or a plotter. Useful data display features include ease of plotting data, changing scales, labeling plots, variety of plot formats (*x*, *y* line plots, bar charts, pie charts, 3-D plots), using nonlinear axes (such as logarithmic), ability to output to a file, and support of a wide range of output device types (printers and plotters) and data formats.

The line between data acquisition and analysis software continues to blur. Many data acquisition software products continue to add sophisticated data analysis features. At the same time, many data analysis software packages add support for both generic and specific data acquisition (plug-in cards) and instrumentation (RS-232, GPIB) hardware.

One final note on selecting suitable software products: always check the PC resource requirements of a package before purchasing it. This includes both a PC's hardware (processor type, speed, amount of RAM, hard drive space) and software (operating system) environment. If a particular product would require you to upgrade or replace a PC to meet its requirements, perhaps you can use a different (usually older) software package in its place.

Now, we will examine a few data acquisition software packages in greater depth.

11.2.1 LABTECH NOTEBOOK

Laboratory Technologies Corp. produces the LABTECH family of data acquisition software products. These applications evolved from DOS versions into the current MS Windows 95/98/NT/2000 graphical programs that use icons and menu selections for setup and control of data acquisition systems. They require no programming and are very easy to use. LABTECH NOTEBOOK provides real-time data acquisition, display, control, and data logging for scientific and engineering applications. LABTECH NOTEBOOKpro is intended for larger data acquisition systems (requiring more system elements/ icons and interconnects) and includes support for high-speed DMA transfers as well as RS-232 and GPIB instruments.

LABTECH CONTROL is a superset of NOTEBOOKpro, intended for industrial applications. Besides supporting even larger systems, CONTROL's additional features include alarms, PID control loops, and remote-control capabilities. LABTECH CONTROLpro is a larger version of CONTROL (for very complex systems).

To simplify this discussion, we will refer to all of the LABTECH NOTEBOOK and CONTROL products generically as NOTEBOOK, although some advanced features may only be available in NOTEBOOKpro or CONTROL/CONTROLpro.

LABTECH NOTEBOOK supports more than 1000 different data acquisition boards from at least 50 hardware manufacturers, including Data Translation, Intelligent Instrumentation, Keithley, and National Instruments. Its advanced features include real-time data display, remote data monitoring via the Internet, and even high-speed data streaming (up to 1 Msample/sec).

To set up NOTEBOOK for a particular data acquisition task you run Build-Time, a graphical user interface (GUI). This consists of standard Windows-style pull-down menus, some specialized control buttons, and a selection of icons. The icons represent function blocks, such as analog inputs, digital outputs, logs, and displays. You set up your data acquisition system by placing the appropriate icons or blocks on the drawing board area of the Build-Time screen, and then you connect them together. This becomes a graphic representation of your data acquisition task. A simple example would be connecting an analog input block to a log block (for saving data in a file) or to a display block (for graphing the data on the screen). Figure 11-10 shows a typical NOTEBOOK Build-Time display for a temperature measurement application.

Each icon or block has a dialog box that lets you set up its features. For example, an analog input block would have settings for sample rate and input voltage range. A thermocouple block would have settings for thermocouple material type and temperature range.

Besides data input and output icons (both analog and digital), NOTEBOOK also uses calculated blocks to process data. These are connected between a data source and destination. More than 50 processing operations are available, including basic mathematical, trigonometric, statistical, and Boolean functions. In addition, NOTEBOOK has several special functions available, such as FFTs and FIR filters. You can even produce customized blocks, based on a C program, with C-Icon blocks (using additional LABTECH software).

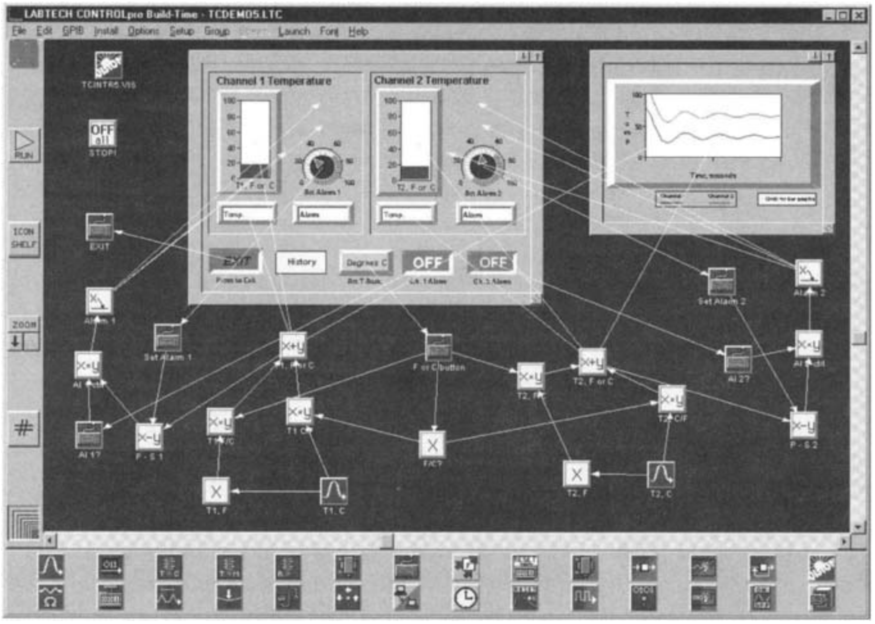


Figure 11-10 A sample LABTECH NOTEBOOK Build-Time display. (Courtesy of Laboratory Technologies Corp.)

The display features in NOTEBOOK are available through a software module called Realtime VISION, available as a Build-Time icon. VISION contains its own array of icons for display, control, drawing, and even animation. The display choices include digital meters, analog meters, bar graphs, trend graphs (y versus t), and x, y plots. VISION's control icons include knobs, slides, and buttons for controlling outputs.

Once your Build-Time setup is complete, you can run the system by starting Run-Time. This is the data acquisition process in operation, using the VISION displays set up in Build-Time. Figure 11-11 shows a typical Run-Time VISION display.

Several of the selectable output data file formats of NOTEBOOK are suitable for exporting data to other applications, such as spreadsheets. In addition, LABTECH provides an Excel macro that allows you to import data into a Microsoft Excel spreadsheet from NOTEBOOK, in real time. In addition, it can export data from Excel back to NOTEBOOK. You can set up your acquisition (with some limitations) and run it while staying entirely within Excel.

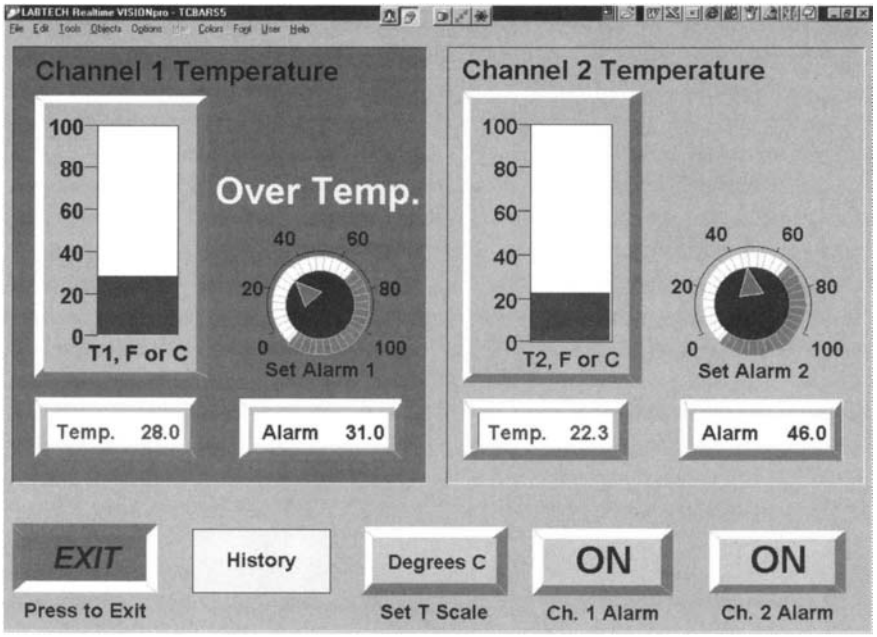


Figure 11-11 A sample LABTECH NOTEBOOK Run-Time VISION display. (Courtesy of Laboratory Technologies Corp.)

NOTEBOOK also supports interfaces to other Windows applications via Microsoft's dynamic data exchange (DDE) protocol. Under DDE, one program is designated the server (usually the data source) and the other program is the client (which request and receives the data). The transfer of data is usually accomplished using standard Windows Copy and Paste commands. This sets up the DDE link. Subsequently, whenever the source data changes (at the server) the client sees those changes. This is a good way to export LABTECH data to another application for further analysis.

Among NOTEBOOK's advanced features are data streaming and remote data acquisition. When using a high-speed data acquisition board, you may need to transfer data to your PC's memory at or close to the board's maximum conversion rate. NOTEBOOK allows you to set up an analog input block for high-speed streaming. You can select the sampling rate, size of the memory buffer (in the PC's RAM) for data storage, and whether to stream this data in real time to a disk file (or just leave it in memory). Furthermore, you can collect the data at high speeds, such as 200,000 samples/sec, while displaying it at much lower rates, such as 30 Hz. This way the display hardware and software do not slow up the acquisition process.

You can use NOTEBOOKpro or CONTROL/CONTROLpro to enable the remote display of data via the Internet. A PC running NOTEBOOKpro becomes a server, using LABTECHnet software. This system runs the desired data acquisition task. Remote PCs connected to the Internet (or even a local TCP/IP network, connected to the server PC) are clients also running LABTECHnet as a background task. The data can then be viewed using a standard Web browser (such as Netscape Navigator or Microsoft Internet Explorer), Realtime VISION, or an application that supports the Active-X protocol (such as Microsoft Word and Excel). This capability allows data acquisition processes to be monitored anywhere in the world.

More information about LABTECH'S products is available at their Web site (www.labtech.com).

11.2.2 LabVIEW

LabVIEW, a software product developed by National Instruments, is used for data acquisition, analysis, and control. It supports not only National Instruments' hardware products but also those of most major manufacturers. It has an open architecture and has become an industry standard.

LabVIEW (which is an acronym for *laboratory virtual instrument engineering workbench*) is a graphical programming language that runs on PCs under MS Windows 95/98/NT/2000, Linux, and UNIX. There are versions that run on Power Macs and on Sun and HP workstations. As with LABTECH NOTEBOOK, you graphically connect icons or functional blocks instead of writing lines of programming text to define a system's operations. LabVIEW, being a true programming language, produces 32-bit, compiled applications as well as stand-alone executables and dynamic-link libraries (DLLs). It uses a data-flow programming model: the execution order is determined by the data values moving between functional nodes.

All LabVIEW programs are composed of one or more virtual instruments (VIs) that emulate a physical instrument (such as an oscilloscope or a signal generator). Each VI contains three components: the front panel, the block diagram, and the icon/connector. The front panel is the user interface for the VI, containing controls (inputs) such as knobs, buttons, and data acquisition card signals as well as indicators (outputs) such as graphs and numeric displays. This is the high-level I/O for the VI.

Once the front panel I/O functions have been determined, you build a graphical program using the block diagram. Here, you define the operations to be performed on data coming from the controls and going to the instruments (which all have connection terminals). This is done by first placing the desired function icons in the block diagram window and then connecting them together.

For example, a simple VI front panel may consist of a data acquisition card input channel (the input) and a waveform graph (the output). If no data processing is required, the block diagram consists of wiring the terminal of the data acquisition channel to the terminal of the waveform graph. If the data requires some processing, such as employing an FIR filter to reduce noise, you can add that function between the input and output. You can also add data acquisition functions (analog and digital I/O), instrument I/O functions (such as IEEE-488 and RS-232 commands), and even file I/O functions (reading and writing data files) to the block diagram. Available functions include basic mathematical and trigonometric operations as well as sophisticated signal processing algorithms, such as FFTs and digital filters.

The third component of a VI, the icon/connector, is the key to LabVIEW's modular design. The icon represents the VI and lets you use it as a sub-VI within another VI. This is analogous to using a subroutine in a conventional, text-based programming language. The use of sub-VIs makes LabVIEW hierarchical and easily extendible, which is central to its power and popularity. Many data acquisition hardware manufacturers provide VIs for their products. Users then build their own LabVIEW application around these predefined VIs.

LabVIEW works equally well with stand-alone instruments connected to a PC via a standard interface (such as IEEE-488 or RS-232) or with data acquisition cards installed inside a PC. LabVIEW uses drivers, which are specialized VIs, to simplify interfacing to the hardware. For stand-alone instruments, LabVIEW uses its virtual instrument software architecture (VISA) library of VIs that provides a common software interface for different communications standards.

For using data acquisition hardware, LabVIEW has a DAQ Solution Wizard. This simplifies the process of configuring the hardware and setting up the data acquisition application. To employ the wizard, you first select the hardware to use and assign functions to analog and digital I/O channels via a separate DAQ Channel Wizard (which can be called from the DAQ Solution Wizard). For example, if you are using a National Instruments PCI0-MIO-16E-1 card, you may define analog input channels 0 and 1 as temperature inputs, analog input channel 2 as a pressure input, analog output channel 0 as a heater control, and digital output channel 0 as an alarm relay control.

Once the data acquisition I/O channels are configured, you can go to the Solutions Gallery and choose predefined VIs for common applications (such as temperature measurement, data logging, and PID control). If the

Solutions Gallery does not contain a VI similar to your application, the Solution Wizard guides you through the process of building a custom data acquisition application using analog I/O, digital I/O, counters and signal generators. Once the process is completed, you have a new VI that represents your application. As with any VI, this application can be modified. For example, you may need to add filtering or other signal processing between the analog input and the display or data logging output. However, you may first need to collect some raw data before you can evaluate the filtering requirements.

Because of LabVIEW's flexibility, you can easily import data from or export data to other applications, using files. For example, the file I/O functions include VIs to write to or read from a spreadsheet file. Being a complete programming language and development environment, LabVIEW also contains debugging features. You can step through a program in slow motion or single-step to check for data flow problems. You can even set breakpoints in a VI and add probes to examine intermediate data values.

Additional LabVIEW features include remote data acquisition via TCP/IP and the Internet and actively sharing data with other Windows applications via DDE. Specialized measurement and control applications are supported with a wide range of add-on tools, such as a PID control tool set and a signal processing tool set that includes wavelet analysis.

There are several versions of LabVIEW available from National Instruments. The LabVIEW Base Package contains all of the features needed to create data acquisition applications and perform basic analysis and signal processing operations. The LabVIEW Full Development System adds additional mathematical and signal processing capabilities, such as array operations, integration, differentiation, statistical functions, digital filters, and 3-D plotting. The LabVIEW Professional Development System adds the capability of stand-alone executables, shared DLLs, and tools to support large project development. The majority of data acquisition applications would only need the Base Package as long as sophisticated signal processing and analysis is not required (or data will undergo postprocessing analysis in another program).

More information about LabVIEW is available at National Instruments' Web site (www.ni.com).

11.2.3 Other Data Acquisition Software Products

LABTECH NOTEBOOK and LabVIEW are probably the most popular software products used primarily for PC-based data acquisition. However, there are several other widely used software packages, including Agilent VEE, Test Point, Dasy Lab, and Snap-Master. Some data acquisition software packages are produced by a hardware manufacturer only for support of their boards. Others are produced by independent manufacturers and support a wide range of hardware sources. The following are a sample of software products that support multiple hardware vendors.

Agilent VEE Agilent VEE (originally HP VEE) is a Windows graphical programming language, similar to LabVIEW, designed for controlling electronic instruments. Originally developed by Hewlett Packard, VEE is strongly geared toward stand-alone instruments connected to a PC via external buses (especially IEEE-488), but it does support some data acquisition cards.

As with LabVIEW, in VEE you place graphical objects in a work area and connect them to form a program. There are no separate front panel and block diagram windows in VEE: all objects, interconnects, and displays are in a single window. Many of these available objects are math and signal processing functions, allowing you to build complex data acquisition systems.

Configuring instruments is done via the Instrument Manager. This has a very useful feature, Auto Discovery, which allows VEE to find instruments connected to the PC's interfaces. It works well for IEEE-488.2 devices, where you do not need to know the instrument's GPIB bus address to access it.

VEE is available in two versions: OneLab or Pro. VEE OneLab is designed for individual workstations running small-to-medium instrumentation systems. VEE Pro supports more instruments, larger programs, and multiple users.

You can find more information about VEE at Agilent's Web site (www.tm.agilent.com).

Test Point Test Point, from Capital Equipment Corp., is a software package that quickly generates compiled data acquisition, test-measurement, and analysis applications without writing any code. It runs under Windows 95/98/Me/NT/2000 and uses an object-oriented approach that is somewhat different from that of LABTECH NOTEBOOK or LabVIEW. Instead of connecting symbols, you simply describe your desired application and Test Point generates the code for it.

You start by selecting objects you need for your new application: input devices (such as switches, ADC channels, or GPIB instruments) and output devices (such as graphs, files, and meters). Next you build an action list from the selected objects, choosing the desired operation for each one and filling in any application-specific information (such as acquisition rate for an ADC object). Then Test Point generates a compiled program from the description.

Besides its ease of use, one of Test Point's advantages is that the applications it produces can be stand-alone and run on any current PC without requiring an additional software license. Another Test Point feature is auto-search that automatically finds data acquisition boards installed in a PC.

Test Point supports a wide range of data acquisition boards from major manufacturers. Its data acquisition objects are high-level and vendor independent. You can swap supported boards with similar functionality or move to a different PC and still run the same Test Point application, without any modifications: at run time, auto-search handles the low-level hardware configuration.

Test Point works equally well with automated instruments that use standard interfaces, such as IEEE-488, RS-232, and RS-485. It has a good selection of math objects, including FFT functions, digital filters, statistical functions, and curve-fitting functions. It supports data links to other Windows applications through standard software protocols, such as OLE2 and DDE. Test Point is also extendible, supporting user-defined objects.

There are several optional toolkits available for Test Point, including the Internet Toolkit for remote data acquisition and control. Capital Equipment Corp. also produces a stand-alone, Web-based data acquisition product called Web DAQ/100. For additional information, visit their Web site (www.cec488.com).

Dasy Lab Dasy Lab, from Dasytec USA (a subsidiary of National Instruments), is a graphics-based, easy-to-use data acquisition software product for Windows. It supports data acquisition hardware from more than 40 manufacturers. It also supports IEEE-488 and RS-232 instruments connected to a PC.

Dasy Lab uses function modules as graphic representations of I/O devices, signal processing/control functions, and display devices. You select the desired function modules and place them on the worksheet. You connect the modules with wires to represent data flow (as with LABTECH NOTEBOOK and LabVIEW). Each module's parameters can be configured via a dialog box. For example, an Analog-Input Module would have settings for the number of channels and their sampling rate.

Dasy Lab contains math and signal processing function modules, including differentiation, integration, curve fitting, and statistical functions. Optional add-on modules include FFTs and digital filters. Other options include a driver toolkit to develop software support for custom hardware and an extension toolkit that allows you to develop your own custom function modules.

Dasy Lab has DDE input and output modules, for easy interfacing to other Windows applications. An additional product, Dasy Lab Net, provides remote data acquisition capabilities via any TCP/IP network, including the Internet. More information about Dasy Lab and related products is available at their Web site (www.dasylab.net or www.dasytec.com).

Snap-Master Snap-Master for Windows, from HEM Data Corp., is another graphical data acquisition and analysis software package. This product can run on older PCs under Windows 3.1 or on newer models using Windows 95 or above. Snap-Master supports data acquisition hardware from more than 15 manufacturers, including PCMCIA cards as well as plug-in boards. One unique feature of Snap-Master is its sensor database that automatically converts acquired data values to appropriate engineering units (degrees, psi, etc.).

As with several other graphics-based data acquisition software packages, Snap-Master is configured by selecting icons for the desired functions and connecting them together. It also supports DDE for sharing data with other Windows applications. In addition, Snap-Master can import or export data in the popular CSV spreadsheet format.

Snap-Master's Waveform Analyzer module provides signal-processing capabilities, including mathematical functions, statistical functions, and digital filters. The Frequency Analyzer module includes FFTs, correlation functions, and processing functions specific to electrical, mechanical, and hydraulic tests. New functions can be added using the Programmer's Toolkit and drivers for custom hardware can be written using the Hardware Driver Interface.

For more information on Snap-Master products you can visit HEM's Web site (www.hemdata.com).

This survey of data acquisition software is hardly exhaustive. However, it should give you a broad view of some major products that are commercially available. For additional listings, refer to the Appendix. Please bear in mind that software products change much more rapidly than hardware products and you should always obtain current information before making a purchasing decision.

Next, we will turn to data analysis software, both for data acquisition and general-purpose applications. We will start by looking at MATLAB.

11.2.4 MATLAB

MATLAB, a product of The Math Works, is a popular technical programming language and computing environment which runs under Windows 95/98/Me/NT/2000 as well as other operating systems (Macintosh OS, Linux, and other UNIX systems). MATLAB is a powerful tool for data analysis and display, containing more than 600 mathematical, statistical, and engineering functions. In addition, it supports data acquisition through external IEEE-488 and serial (RS-232, RS-485) instruments as well as data acquisition boards from some major manufacturers (including National Instruments and Keithley). Several independent data acquisition software products support MATLAB and some even contain a subset of its features.

MATLAB, which stands for *matrix laboratory*, is an interactive system using arrays as its basic data elements. These arrays do not require dimensioning, as in conventional programming languages such as BASIC or C. The base MATLAB system contains standard mathematical, trigonometric, and matrix manipulation functions along with 2-D and 3-D graphics capabilities. Additional features are available by adding toolboxes, which are collections of specialized MATLAB functions. Some of these toolboxes include data acquisition, filter design, image processing, instrument control, signal processing, statistics, and wavelet analysis. MATLAB also has application development tools that allow you to create stand-alone MATLAB programs. In addition, you can call MATLAB routines from standard C and FORTRAN programs using MATLAB's application program interface (API).

When the program is started, the MATLAB desktop (consisting of several windows) appears. These partitions include the command window, the command history window, and the launch pad. The command window is where you enter variables, commands, and M-files (text files containing MATLAB commands and programs). The command window is an interactive environment, where you spend most of your time in MATLAB. The command history window lists all the previously entered command lines and the functions used. You can execute selected lines in the command history or even save them as an M-file (to execute at other times, as a script). The launch pad provides access to tools, demos, toolboxes, and documentation files. A typical MATLAB desktop is shown in Figure 11-12.

The essence of MATLAB is matrix manipulation. This allows you to operate on multiple numbers at one time. You can enter a matrix into MATLAB manually, from an external data file, from a built-in matrix-generating function, or using functions in M-files. To enter data manually, you give the new matrix a name, separate elements of a row with spaces or commas, separate rows with semicolons, and enclose the numbers with square brackets. For example, to manually generate a 4 × 4 matrix called Test, with incrementing values, you would type in the command window:

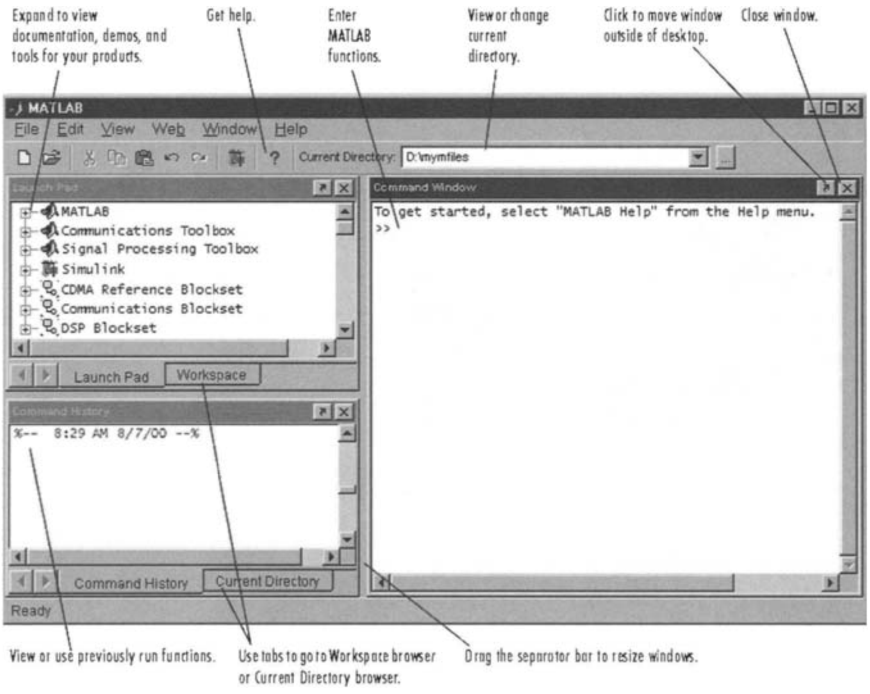
```
Test = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
```

Then, MATLAB would display this data as a matrix:

```
Test =
         1     2     3     4
         5     6     7     8
         9    10    11    12
        13    14    15    16
```

Simple math operations in MATLAB default to processing data columns. For example, the *sum* function calculates the sum of each column in a matrix. If we typed

```
sum(Test)
```


Expand to view documentation, demos, and tools for your products.
Get help.
Enter MATLAB functions.
View or change current directory.
Click to move window outside of desktop.
Close window.
MATLAB
File Edit View Web Window Help
Current Directory: D:\mymfiles
Launch Pad
MATLAB
Communications Toolbox
Signal Processing Toolbox
Simulink
CDMA Reference Blockset
Communications Blockset
DSP Blockset
Launch Pad
Workspace
Command History
Current Directory
Command Window
Ready
View or use previously run functions.
Use tabs to go to Workspace browser or Current Directory browser.
Drag the separator bar to resize windows.

```
%-- 8:29 AM 8/7/00 --%
```

```
To get started, select "MATLAB Help" from the Help menu.
>>
```

Figure 11-12 The MATLAB desktop. (Courtesy of The Math Works)

MATLAB would display

```
ans =
       28    32    36    40
```

If we wanted to sum the rows of our matrix, Test, we first have to transpose the matrix so that rows become columns. The transpose command is denoted by the single quote (‘) character. So, typing

```
Test'
```

results in

```
ans =
       1    5     9    13
       2    6    10    14
       3    7    11    15
       4    8    12    16
```

Now we can get the sum of the rows in Test by entering

```
sum(Test')
```

This produces

```
ans =
      10    26    42    58
```

If we wanted the sum of all the array elements, we would enter

```
sum(sum(Test)')
```

giving the result

```
ans =
      136
```

You can easily access and operate on individual elements of an array using subscript notation. The element at row *x* and column *y* in array A is denoted A(*x*, *y*). Looking at our sample matrix, Test, element Test (2, 3) is 7.

Another important part of MATLAB is the colon (:) operator. It is used to denote a range of values or elements. For example, entering

```
1:5
```

produces a 5-element row vector:

```
ans =
     1    2    3    4    5
```

Using : in subscript notation produces references to portions of a matrix. If we wanted to sum only the elements in the second column of our Test matrix, we would enter

```
sum(Test(1:4,2))
```

and get the result

```
ans =
      32
```

When used by itself, the colon operator refers to all elements in a row or column. So, a less verbose way to get the sum of Test column 2 would be

```
sum(Test(:,2))
```

MATLAB is a complete programming language as well as a computational environment. You do not need explicit type declarations or dimension statements, which simplifies "quick-and-dirty" programming. This is ideal for algorithm evaluations.

MATLAB contains a full set of elementary and advanced mathematical functions that include trigonometric, logarithmic, complex, matrix manipulation, Bessel, and coordinate transform functions. There are MATLAB functions that generate matrices with all zeros, all ones, or random numbers. You can also enter data from an external text file using the load command. MATLAB has an Import Wizard that reads in data files from many standard text and binary formats.

Most user-created MATLAB programs and functions are saved in M-files. These text-based files contain MATLAB code that could be entered at the Command Window. To illustrate, assume we created a data file called *sample.m* that contained the Test matrix declaration we used earlier:

```
Test = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
```

Then we could simply enter in the command window

```
sample
```

Now, MATLAB has created the array Test, with its associated data. An M-file can just as easily contain commands that operate on data.

MATLAB contains several programming flow-control statements, similar to those used in C and other standard languages. These include *if, switch, continue*, and *break* statements as well as *for* and *while* loops. Some aspects of the MATLAB implementation of these control statements do differ from their C language equivalents.

MATLAB's graphic facilities are at least as extensive and powerful as its mathematical manipulation capabilities. You can easily create x, y graphs with the *plot* function. For example, then following commands will generate and plot a cosine function from 0 to 2π, using a 100-point vector:

```
x = 0:pi/50:2 * pi;
y = cos(x);
plot(x,y)
```

Note that ending an assignment line with a semicolon (;) supresses output.

The basic plot can be enhanced by adding labels and titles as well as specifiying fonts and colors. You can use the *plot* command to display multiple curves in a single figure via the *subplot* command. MATLAB also supports

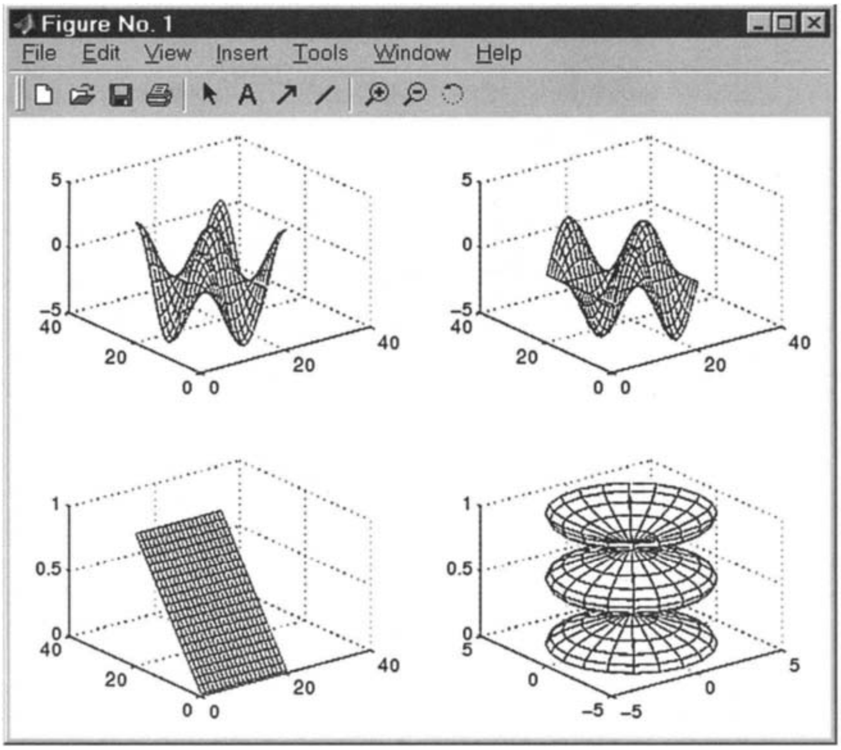


Figure 11-13 Typical MATLAB plots. (Courtesy of The Math Works)

more sophisticated plotting, such as contour and 3-D surface plots. Plus, you can save plots as standard graphics files using common formats, including TIFF, JPEG, and PostScript. Figure 11-13 shows some sample 3-D MATLAB plots.

An additional reason for MATLAB's popularity is its wide selection of toolboxes. These are collections of functions targeted at specific applications. For example, the Signal Processing Toolbox includes functions for digital filtering, spectral analysis (FFTs), convolution, and signal generation. This toolbox contains two interactive tools that act as special programs: the filter design and analysis (FDA) tool allows easy, interactive digital filter design and testing; the SP tool is a graphical user interface (GUI) that simplifies the display, analysis, and manipulation of processed signals, including filters and FFTs.

The Instrument Control Toolbox and the Data Acquisition Toolbox are of special interest to us. The Instrument Control Toolbox contains functions for controlling external instruments via a standard interface, such as GPIB or RS-232. Here is a simple MATLAB script that uses a National Instruments PCI-GPIB+ board to set the output amplitude of a Hewlett Packard 33120A function generator and then read back the new setting:

```
fg = gpib('ni',0,1)      %create a GPIB object (board index=0,
                         %primary address=1).
fopen(fg)                %connect object to instrument
fprintf(fg,'Volt 2')     %write ASCII data (set amplitude command)
fprintf(fg,'Volt ?')     %write ASCII data (read amplitude command)
data = fscanf(fg)        %read ASCII data (amplitude value readback)
fclose(fg)               %disconnect object from instrument
delete(fg)               %remove object from memory
clear fg                 %remove object from work space
```

Note that comments in MATLAB programs begin with the percent (%) character.

The Data Acquisition Toolbox provides functions for accessing plug-in data acquisition cards. There are functions to create analog input, analog output, and digital I/O objects (similar to the GPIB object we created earlier). For example, the following MATLAB script acquires 2 seconds of data from a standard PC sound card at the maximum sampling rate of 44.1 kHz:

```
scard = analoginput ('winsound')      %create an analog input
                                      %object for a
                                      %standard Windows
                                      %sound card.
addchannel(scard,1:2)                 %define two input channels
set(scard,'Sample Rate',44100)        %set sampling rate to 44.1kHz
set(scard,'Samples Per                %set # of samples
  Trigger',88200)                     %(for 2 sec)
start(scard)                          %acquire the data
sdata = getdata(scard);               %store the data in variable
                                      %sdata for processing and
                                      %display.
delete(scard)                         %remove analog input object
                                      %from memory.
clear scard                           %remove object from work
                                      %space
```

In a similar fashion, you can output analog data using the *analogoutput* function. To control a card with digital I/O lines you would use the *digitalio* function.

MATLAB is primarily a general-purpose data analysis and display software product, but its growing support of data acquisition functions makes it a good choice for many laboratory and industrial applications requiring extensive and flexible data processing. The number of data acquisition hardware vendors directly supported by MATLAB is still somewhat limited. However, if you do use a card MATLAB supports, you can easily integrate data acquisition with the program's powerful analysis and display capabilities, eliminating the need to port the acquired data from another software product. In addition, MATLAB has a Data Acquisition Toolbox Adapter Kit that helps you use currently unsupported hardware with its Data Acquisition Toolbox.

More information about MATLAB can be obtained at The Math Works' Web site (www.mathworks.com).

11.2.5 DADiSP

DADiSP is a data analysis and display software package for MS Windows-based PCs and UNIX workstations, produced by DSP Development Corp. It is a menu-driven program that operates as an interactive graphics worksheet. DADiSP requires no programming but you can still create your own functions, macros, and command files. It can display, process, and output data plots, both 2-D and 3-D. Each worksheet can contain up to 100 independent graphics windows simultaneously. Each window contains its own plot or data. Data in one window can be a function of data in other windows. As the independent data windows get changed, the dependent window is automatically updated. Suppose an independent window contains a waveform (perhaps acquired via another program). If a dependent window contains the power spectrum (via FFT) of that waveform, it will be updated if new data is read into the original waveform window. This is a very powerful feature for doing complex analyses and is analogous to using a conventional spreadsheet, such as Microsoft Excel, where the value in one cell can change if data in other cells are modified. In addition, windows can contain formulas without data and act as templates, ready to process any data set.

DADiSP contains more than 1000 analysis functions (as of the DADiSP/2000 version). These functions include basic mathematical and trigonometric operations, statistical functions, FFT and related operations (such as autocorrelation), function generation, digital filtering functions, graphics operations, image processing functions, matrix manipulation functions, data file I/O, and hardcopy (plot or print) operations. DADiSP also has many optional add-on modules for additional functionality. Typically, DADiSP's data input and output operate on files, using either ASCII or binary formats. Data set size is limited only by disk storage space, not by available RAM. This enables very large waveforms to be manipulated, intact.

You can also use some of DADiSP's add-on modules to perform data acquisition and instrument control. The DADiSP/ACQ module supports data collection using data acquisition boards from several major hardware manufacturers, including Data Translation, National Instruments, and Scientific Solutions. It is a menu-driven program that incorporates LABTECH's driver software to interface to the boards. As with other software modules, it is tightly integrated into the DADiSP environment, allowing immediate analysis of acquired data.

The GPIB Lab module is another menu-driven product that allows you to control and collect data from IEEE-488 instruments. It supports GPIB cards from IOtech and National Instruments. It also contains drivers for many popular GPIB instruments, eliminating the need to write your own program from scratch.

DADiSP has many other add-on modules for specialized analysis operations. For example, DADiSP/AdvDSP contains advanced DSP algorithms such as the chirp-Z transform, zoom FFT, cepstrum analysis, and spline interpolation. The DADiSP/Filter module enables easy design and analysis of FIR and IIR digital filters. DADiSP/Stats contains statistical analysis functions such as chi square tests, ANOVA, and polynomial regressions.

DADiSP can directly call other programs, for uses such as data acquisition or specialized analysis. It can capture data from nearly any available source. It has advanced graphics features such as zooming and scrolling through a waveform. Many different plot formats are supported, including line graphs, histograms, bar charts, and scatter plots. You can annotate data windows with text or graphics, for labels and comments. It even supports data transfer between other MS Windows applications via DDE links.

You can define custom menus and macros in DADiSP to help automate the analysis process and allow less skilled workers to perform it. The program also offers programmability using its C-like series programming language (SPL).

DADiSP is supported on many different PC and workstation platforms running 32-bit Windows or a version of UNIX. Earlier versions of DADiSP ran under MS-DOS and MS Windows 3.1. The current version, DADiSP/2000, requires a Pentium PC running MS Windows 95/98/NT/2000. You can obtain additional information about DADiSP at DSP Development's Web site (www.dadisp.com).

11.2.6 Other Analysis Software Products

There are many other software products used for data analysis and display, besides MATLAB and DADiSP. Several are listed in the Appendix. Some analysis and display software packages are general purpose and not specifically for data acquisition. Here is a brief overview of a few representative products.

GAUSS GAUSS, from Aptech Systems, is a data analysis environment and programming language. It is especially useful for complex processing of large amounts of data. GAUSS is a general-purpose analysis package with support for engineering applications. It runs under Windows 95/98/NT/2000 as well as UNIX/Linux.

Similar to MATLAB, GAUSS is a matrix programming language. Its capabilities are enhanced by add-on packages from Aptech and third parties. GAUSS is a full-featured programming language that can execute commands interactively (one at a time) or as a complete, compiled program. It even contains its own debugger.

Besides powerful data analysis features, GAUSS also contains publication-quality graphics capabilities. This is a set of routines for generating 2-D and 3-D plots, such as contours, bar graphs, and 3-D surfaces. In addition, the optional GAUSS Engine is a tool that generates stand-alone GAUSS programs for royalty-free distribution.

For more information on GAUSS, visit Aptech's Web site (www.Aptech.com).

IGOR Pro IGOR Pro, from WaveMetrics, Inc., is another data analysis and display application that can handle large data sets and produce publication-quality graphics. Some of its features include curve fitting, FFTs, and filtering.

A feature of IGOR making it especially useful for data acquisition applications is its use of waveforms (or waves) as basic objects to be manipulated. All wave objects are assumed to have uniform spacing along the independent axis, as most time-based digitized data does. Waves can have up to four dimensions and contain either numeric or text data.

Data is loaded into waves manually or from a text file, or it can be generated by a mathematical expression. Manual data entry is via a table, which also allows editing of wave data. Wave data can be used to generate graphs, both 2-D and 3-D plots.

You can control IGOR either through menus and dialogs, by entering commands at the command line, or by running procedures (scripts). IGOR Pro is available for both Windows 95/98/NT/2000/Me and the Macintosh OS. For more information about IGOR, you can visit WaveMetrics' Web site (www.wavemetrics.com).

Microsoft Excel Microsoft Excel is Microsoft's general-purpose spreadsheet program for Windows, used for data analysis and display. It is commonly used in a business environment since it is part of the Microsoft Office package. Excel includes many features suitable for analysis and display of data acquisition system data. In fact, some simple data acquisition software products link to Excel for analysis and display functions.

Excel, like most spreadsheets, uses rows and columns of cells to store and manipulate data. As an example, to produce an x, y plot you would select two columns of data (representing the two graph axes) and choose the appropriate plot function or wizard.

Excel contains basic math, trigonometric, and analysis capabilities, including statistical functions. It also has an Analysis Toolpack with additional features useful for analyzing acquired data, such as FFT, correlation, and regression (curve fitting) functions. These features may have more limitations than in a dedicated analysis or acquisition program (some versions of Excel, for example, limit FFTs to only 1024 points), but they will usually suffice for many basic applications.

Current versions of Excel run under MS Windows 95/98/NT/2000 (earlier versions ran under Windows 3.1). For current information about Excel, visit Microsoft's Web site for this product (www.microsoft.com/office/excel).

Mathematica Mathematica, from Wolfram Research, is a mathematical analysis and display application for Windows 95/98/NT/2000, UNIX, Linux, and Macintosh OS that processes both numeric and algebraic expressions. It can be used as simply and interactively as a calculator (with its push-button interface) or you can use it to write an entire program or application. Mathematica allows you to state your problem at a high level of abstraction while it chooses the best way to perform the necessary calculations.

Mathematica is especially useful at processing symbolic (algebraic) expressions. You can use it to easily solve partial differential equations or to evaluate complex integrals, all entered using standard mathematical notation.

Mathematica also provides a wide range of data display and visualization features, including 2-D plots, 3-D plots, and graphics (video) displays. It can produce publication-quality output, designed for technical documents.

Mathematica is a fully customizable and extendible system. The Mathematica Applications Library contains additional software for specific applications, including Experimental Data Analyst (for curve fitting and error analysis), Signals and Systems (for signal processing and filter design), and Wavelet Explorer (for signal and image analysis using wavelets). There is even a product, Mathematica Link for Excel, that gives you access to Mathematica's features from within an Excel spreadsheet.

For more information about Mathematica, visit Wolfram Research's Web site (www.wolfram.com).

This completes our survey of some popular software products useful for data acquisition systems. We will end this chapter with a brief discussion of how to select the appropriate commercial hardware and software products for your data collection needs.

11.3 How to Choose Commercial Data Acquisition Products

If you are putting together a PC-based data acquisition system, you will most likely try to do it with commercial products. This approach will save you not only a lot of development time, but probably some money, also.

The first step in selecting your data acquisition system components is to define the physical measurements you need. For example, an environmental test chamber may require ten temperature transducers covering the range of 0 to +150°C, with an accuracy of ±1°C and a reading from each transducer every second.

Next, determine the type of transducers to be used, the signal conditioning needed and whether any output control signals are required. Continuing with the environmental test chamber example, since high accuracy is not required, thermocouples are a reasonable transducer choice. This would entail using a cold-junction compensation board to condition the thermocouple signals. Additionally, the ADC used would need moderately high gain, to deal with the millivolt-level signals from the thermocouples. Let us also assume that one or more of the temperature sensors will control the temperature of the test chamber. This temperature control would be an analog signal, in the range of ±10 V, controlling the chamber's heating and cooling systems. So, we also need an analog output channel.

The following step is to consider how much data will be collected, how much analysis will be performed on it, and how it will be displayed and stored. In our example, we will assume the maximum test run time will be 1 hour. At a sampling rate of 1 sample/sec per channel for 10 channels, 1 hour corresponds to 36,000 samples. For 8-bit data, this would produce a binary file of 36,000 bytes. For 12-bit data, a binary file would usually be 72,000 bytes long, if the data is not compressed (assume 2 bytes/sample). The data display should be all 10 temperature channels (transducers) in real time with data saved to disk. The only analysis required would be the minimum, maximum, and average temperature of each channel.

The next step is to decide on the computer to use. Often, this is determined by the PCs and data acquisition products already on hand. If it is an open question, consider costs, which software packages you want to run on it, and whether this PC will be permanently dedicated to the data acquisition task or freed up at a later date for other jobs. Usually, software compatibility and availability are much more important than the computer's raw processing power, except for very specific, high-performance applications.

The final step is picking out the hardware and software products to use. The most important factor here is hardware/software compatibility. Be absolutely sure that the software package you want will work properly with the hardware selected, including any signal conditioning boards, multiplexers, and other expansion modules.

For our test chamber example, high-performance is not required. The overall data rates are slow (10 samples/sec). The required resolution could fit an 8-bit ADC, with 1°C accuracy over a 150°C dynamic range, being within 1 part in 256 (8 bits). However, low-speed 12-bit ADCs are not very expensive and the extra resolution will produce better data. A 12-bit analog I/O board is a good choice, with at least 10 single-ended input channels and one output channel. At the low data acquisition speeds called for, the board does not need a timer/counter, since the PC's internal clock (as slow as 18 ticks/sec) is adequate. A multifunction data acquisition card would be overkill, unless the system will be used for other purposes in the future. In addition, a signal-conditioning panel with cold-junction compensation, for thermocouples, would be very useful.

Now that we have all our hardware specifications, we should pick out the software. We will assume that we do not want to write any programs and we would like the system to be operable by virtually anyone, without extensive training. This certainly points to a simple, graphics-based data acquisition software package. We do not need much analysis power and want a real-time display, with data storage and some means of producing a printed graph. In addition, automatic processing of thermocouple signals is required, to produce outputs directly in degrees Celsius. The software should also be capable of controlling the temperature of the test chamber. A good choice here would be a package similar to a basic version of LABTECH NOTEBOOK, LabVIEW, Test Point, or Dasy Lab.

This ends our overview of commercial data acquisition products for PCs. This is a very dynamic field, with new products and manufacturers appearing (and disappearing) all the time. This is particularly the case with software products, which tend to have very short life cycles. Please refer to the Appendix at the back of the book for more comprehensive listings of manufacturers and products. It is always a good idea to contact a manufacturer or visit its Web site to obtain up-to-date information.

In the next chapter, we will look at some PC hardware and architectures we have previously just touched upon, including notebook PCs, the PCMCIA standard, and the PC-104 bus, as well as embedded and ruggedized PCs.

CHAPTER

12

Other PC Configurations and Hardware for Data Acquisition

Over the past decade, Microsoft Windows and Intel processor, PCI bus-based PCs have consolidated their position as industry standards (so-called "Wintel" PCs). As a result, many other PC architectures and buses (some designed as potential successors to IBM's PC/XT/AT systems) have become obsolete or relegated to niche markets. We will briefly look at a few of these alternative PC architectures, since many of these computers still function in labs and factories.

We will then examine the ubiquitous notebook PC and the PCMCIA PC Card interface standard, which is quite useful for portable data acquisition and control applications. Next, we will look at industrial and embedded PCs along with the PC/104 standard. We will also briefly look at image capture products, as a specialized data acquisition application of growing importance.

12.1 Alternative PC Architectures and Processors

Even though the computer systems covered in this section are no longer manufactured, many were previously produced and used for data acquisition applications. Hence, we will briefly look at them, since some of these PCs are still in use and can continue to operate for years to come. In addition, we will briefly examine DSP products.

12.1.1 IBM PS/2 Computers with MCA

In 1987 IBM introduced its successor to its popular PC/XT/AT computer line, the PS/2. Most (but not all) members of the PS/2 family were based on IBM's new Micro Channel Architecture (MCA), using Intel's 80286 and 80386 CPUs. These MCA PCs were not hardware compatible with the PC/XT/AT systems, but were software compatible. They required very different expansion cards, but ran MS-DOS, MS Windows (or OS/2), and nearly all PC application software. Data acquisition cards for these systems had to be built specifically for the MCA bus.

The performance characteristics of MCA were a large improvement over the ISA bus. The data bus was either 16 or 32 bits wide (depending on the computer model). On a 16-bit system, the address bus was 24 bits wide, for a physical address space of 16 Mbytes. On a 32-bit system, the address bus was also 32 bits wide, for a physical address space of 4 Gbytes. There were 11 hardware interrupt lines that were level-sensitive and could be shared by multiple devices with open-collector drivers. DMA was supported with eight channels and a maximum transfer rate of 5 Mbytes/sec (for 16-bit transfers). The maximum system data transfer rate with MCA was 20 Mbytes/sec, for either memory or I/O cycles (under special conditions).

Figure 12-1 shows the 16-bit and 32-bit MCA connectors. Both were dual-row edge connectors with separate 8-bit and 16-bit signal sections. The 16-bit connector had 58 pins (each side) with 10 optional pins for video signals. The 32-bit connector contained 89 pins (each side) and had four optional pins for special memory transfer support. The MCA connectors placed a ground pin between every four signal pins to minimize electromagnetic interference (EMI), which was a large improvement over the ISA connector layout. MCA also supported audio signals through a special pair of bus lines.

The signal assignments for a 16-bit MCA connector are shown in Table 12-1. Most of these signals were similar to their ISA bus counterparts, such as address (A0–A23), data (D00–D15), interrupt (–IRQ03, –IRQ04, ... , –IRQ15), and control (–REFRESH, –TC, OSC) lines. Others were new to MCA, such as arbitration control lines (ARB/–GNT, ARB0–ARB3, –PREEMPT), connector-specific lines (–CD SFDBK, CD CHRDY, –CD SETUP), and other features (AUDIO).

IBM's Micro Channel was primarily an asynchronous bus (as is PCI). Data transfers over the Micro Channel were controlled by handshaking signals, instead of relying on a synchronous clock for transfer timing. When a device on the bus (such as an adapter card) was commanded to send or receive data, it responded with an acknowledge when it was done. These control and handshake signals determined the MCA timing. In addition, MCA supported some synchronous data transfers.

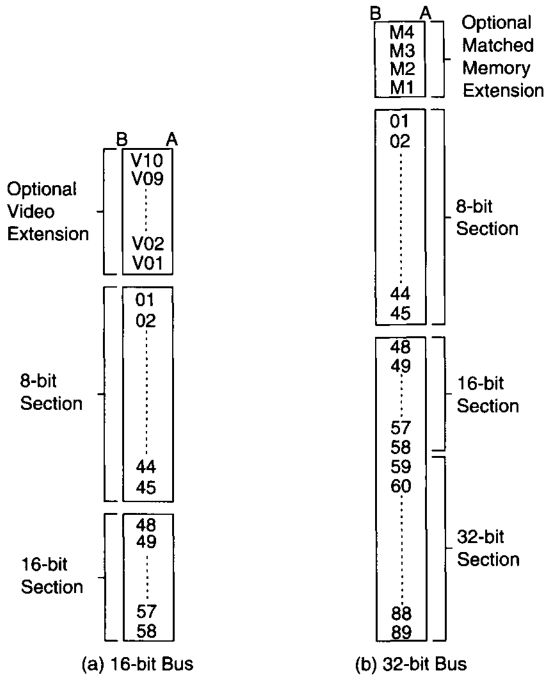


Figure 12-1 IBM Micro Channel bus connectors.

MCA supported up to 15 bus masters in a system (including the main CPU), allowing multiprocessing with several CPUs, as well as DMA. A hardware-based arbitration scheme allowed multiple bus master boards relatively fair access to the system bus. In addition, MCA systems eliminated the need for setting switches or jumpers to configure a board (for addressing as well as interrupt and bus master arbitration levels) using its Programmable Option Select (POS) features. POS consisted of registers that allowed a board to be configured by software only. Each board manufactured for MCA received a unique identification code from IBM, distinguishing it from other boards. A utility program automatically configured one or more boards in the system, ensuring that addresses did not clash. This was an early plug-and-play architecture that relied on external software to configure boards. If you lost the special program you could not configure a card. Newer PCs have plug-and-play software built into the operating system (MS Windows) and/or the BIOS.

In spite of the major hardware differences between MCA-based PS/2 PCs and PC/XT/AT systems, they were still software compatible. All PS/2 computers ran MS-DOS, MS Windows, and most available application software.

TABLE 12-1
IBM Micro Channel 16-bit Connector Pin Assignments

PIN	ROW B	ROW A	PIN	ROW B	ROW A
01	AUDIO GND	–CD SETUP	30	Reserved	–TC
02	AUDIO	MADE 24	31	Reserved	+5VDC
03	GND	GND	32	–CHCK	–S0
04	14.3MHZ OSC	A11	33	GND	–S1
05	GND	A10	34	–CMD	M/–IO
06	A23	A09	35	CHRDYRTN	+12VDC
07	A22	+5VDC	36	–CD SFDBK	CD CHRDY
08	A21	A08	37	GND	D00
09	GND	A07	38	D01	D02
10	A20	A06	39	D03	+5VDC
11	A19	+5VDC	40	D04	D05
12	A18	A05	41	GND	D06
13	GND	A04	42	CHRESET	D07
14	A17	A03	43	Reserved	GND
15	A16	+5VDC	44	Reserved	–DS 16 RTN
16	A15	A02	45	GND	–REFRESH
17	GND	A01	46	Key	Key
18	A14	A00	47	Key	Key
19	A13	+12VDC	48	D08	+5VDC
20	A12	–ADL	49	D09	D10
21	GND	–PREEMPT	50	GND	D11
22	–IRQ09	–BURST	51	D12	D13
23	–IRQ03	–12VDC	52	D14	+12VDC
24	–IRQ04	ARB00	53	D15	Reserved
25	GND	ARB01	54	GND	–SBHE
26	–IRQ05	ARB02	55	–IRQ10	–CD DS 16
27	–IRQ06	–12VDC	56	–IRQ11	+5VDC
28	–IRQ07	ARB03	57	–1RQ12	–IRQ14
29	GND	ARB/–GNT	58	GND	–IRQ15

They also ran IBM's OS/2 operating system that was similar to MS Windows. Software that directly accessed hardware addresses (memory or I/O ports) was not necessarily PS/2-compatible.

Many data acquisition hardware manufacturers who supported PC/XT/AT systems also produced MCA products for PS/2 computers. This was true of most PC board manufacturers, in general. Since PS/2 PCs are now considered obsolete, manufacturers no longer produce MCA boards. Some of the leading data acquisition hardware manufacturers who previously produced MCA boards included Keithley, Data Translation and National Instruments.

12.1.2 The EISA Bus

The Extended ISA (EISA) bus was developed as a nonproprietary alternative to MCA that never gained very wide acceptance. It was an extension of the standard AT (ISA) bus from 16 to 32 data bits, along with 32 address bits. The EISA bus retained hardware compatibility with existing ISA boards. Of course, EISA cards had to be used in an EISA computer to obtain the potential performance improvements. These improvements were similar to MCA: high data transfer rates (up to 33 Mbytes/sec), multiple bus-master support, automatic system configuration, and slot-specific addressing.

Even fewer EISA boards were produced than MCA boards, along with a handful of EISA PCs. Its only advantage was being able to use standard ISA cards in an EISA PC, while having the potential for obtaining higher performance (only when using EISA cards). The higher cost of an EISA PC coupled with its limited improvements minimized this standard's acceptance in the PC industry. In the realm of data acquisition, National Instruments and Scientific Solutions were among the handful of manufacturers who produced EISA boards.

12.1.3 Apple Macintosh II Computers with NuBus

Apple's Macintosh computer line has also been popular for scientific, engineering, and industrial applications, in part because of the open expansion architecture adopted for the Macintosh II series, based on the NuBus. NuBus was a system bus developed by MIT and Texas Instruments for 32-bit computers. It was independent of the computer's CPU, providing buffered, multiplexed signals to the expansion connectors (as does PCI).

Apple has abandoned NuBus for a PCI architecture in its newer Macintosh PCs. Still, many NuBus-based Macintosh II PCs are in use, some for data acquisition applications. The Macintosh series was originally based on Motorola's 68000 microprocessor family. The 68020 and 68030 CPUs used in the Macintosh II series had a 32-bit data and address bus, for a 4-Gbyte address range.

The initial attraction of using a Macintosh computer was its graphics-based user interface, allowing for intuitive operation that sped up the learning process. The software burden rested on the program developers, not the users. The user interface was consistent across all Macintosh application software, minimizing the time needed to learn new programs. Now that Microsoft Windows has come of age, Intel-based PCs have these same advantages.

Apple's NuBus was a synchronous, multiplexed bus, using a 96-pin Eurocard DIN connector (popular in many industrial computer systems, such as those based on the VME bus). The form factor for NuBus cards (approximately 4.0 inches by 12.7 inches) was similar in size to PC/XT cards. It was based on the 1986 IEEE specification, IEEE-1196 NuBus (which originally called for a triple-height Eurocard form factor of 11.0 inches by 14.5 inches). NuBus was a synchronous bus (in contrast to IBM's asynchronous MCA), where all transactions were based on a fixed clock cycle. The edge of this clock determined the bus timing parameters, such as when data was valid or when it should be latched. Apple used a 10-MHz bus clock in its NuBus. The address and data signals were multiplexed onto 32 lines (/AD0–/AD31). Various control signals were used to interpret these multiplexed lines. For example, the /START signal was asserted (active low) when the address/data lines contained a valid address.

The advantage of using synchronous bus transfers was simplicity of protocol and hardware to implement it. The advantage of using multiplexed address/data lines was the need for fewer bus wires (32 saved in this case) than in a nonmultiplexed arrangement. The disadvantage with this multiplexing was slower bus throughput, since multiple bus cycles were required for any data transfer (with separate bus cycles for sending address and data information).

The 96-pin NuBus DIN connector was arranged as 3 rows (A, B, C) of 32 pins, each. Because of address/data multiplexing, all the needed signals fit on 51 lines. The rest of the lines were used for power supply and ground, as shown in Table 12-2.

Each NuBus slot had its own unique ID number, with a maximum of 16 slots allowed by the specification, so each card knew the slot it was in. The 32-bit addressing space of the NuBus had a range of 4 Gbytes. The upper 256 Mbytes of this space was divided among the 16 possible slots, to provide each one with its own dedicated slot space of 16 Mbytes. Apple only used up to 6 expansion slots in its Macintosh II systems, with slot IDs of 9 to Eh. Since each board knew the slot it occupied, it could automatically adjust its address mapping. This was another early plug-and-play system. Since the Macintosh (and all 68000-family computers) used memory mapping for I/O ports, this automatic configuration applied to both memory and I/O addresses.

Several hardware manufacturers have produced NuBus data acquisition cards for Macintosh II personal computers. These included Data Translation, Keithley, National Instruments, and Intelligent Instrumentation.

TABLE 12-2
NuBus Connector Pin Assignments

PIN NUMBER	ROW A	ROW B	ROW C
01	−12VDC	−12VDC	/RESET
02	Reserved	GND	Reserved
03	/SPV	GND	+5VDC
04	/SP	+5VDC	+5VDC
05	/TM1	+5VDC	/TM0
06	/AD1	+5VDC	/AD0
07	/AD3	+5VDC	/AD2
08	/AD5	Reserved	/AD4
09	/AD7	Reserved	/AD6
10	/AD9	Reserved	/AD8
11	/AD11	Reserved	/AD10
12	/AD13	GND	/AD12
13	/AD15	GND	/AD14
14	/AD17	GND	/AD16
15	/AD19	GND	/AD18
16	/AD21	GND	/AD20
17	/AD23	GND	/AD22
18	/AD25	GND	/AD24
19	/AD27	GND	/AD26
20	/AD29	GND	/AD28
21	/AD31	GND	/AD30
22	GND	GND	GND
23	GND	GND	/PFW
24	/ARB1	Reserved	/ARB0
25	/ARB3	Reserved	/ARB2
26	/ID1	Reserved	/ID0
27	/ID3	Reserved	/ID2
28	/ACK	+5VDC	/START
29	+5VDC	+5VDC	+5VDC
30	/RQST	GND	+5VDC
31	/NMRQ	GND	GND
32	+12VDC	+12VDC	/CLK

Since newer Macintosh PCs can accept PCI cards, they can use current data acquisition boards. Many hardware manufacturers provide software drivers for Macintosh computers and many software manufacturers produce versions of their products for the Macintosh operating system (Mac OS).

12.1.4 DSP Chips and Cards

DSP (digital signal processor) chips are very popular processors used alone and with PCs in scientific and engineering environments. These devices are the heart of DSP boards, designed for PCs with an ISA or PCI bus. These DSP cards are special-purpose math accelerators, used to provide DSP functions at very fast calculation rates, minimizing the system CPU's involvement. For example, a 1024-sample floating-point FFT calculation could be performed in under 10 msec with such an ISA card. The DSP is a stand-alone CPU, with its own local memory and control hardware on the card. It is not directly tied to the system CPU, as an internal math coprocessor is. The system CPU sends the DSP commands and data via the bus. The DSP can then operate independently of the system CPU.

Recent PCs, based on the PCI bus with an Intel Pentium processor running at clock speeds well over 1 GHz, can also perform math calculations at very high speeds. However, since the speed of DSP chips has kept pace with general-purpose PC microprocessors, DSP cards are still applicable in high-performance environments.

The most popular DSP ICs used on these cards are the Texas Instruments TMS320 families of 16- and 32-bit processors. These chips are general-purpose processors, optimized for the high-speed mathematical calculations required in DSP applications. They have a special architecture with separate buses for instructions and data (called the modified Harvard architecture). This allows calculations to be performed in parallel with instruction fetches. These devices also use *pipelining* in their computational sections, so that a current computation can progress simultaneously with a new one starting. When many consecutive calculations are done, the overall computation rate decreases dramatically.

There are other popular DSP chips supported by hardware and software products for PCs, including Analog Devices' ADSP series.

Most DSP hardware products come with software support, usually in the form of a library of DSP functions. A programmer can then call these functions from a conventional Windows program, without worrying about the fine details of the DSP board's internal operations.

This concludes our look at alternative PC architectures and processors. Next, we will look at other PC form factors, starting with laptops.

12.2 Notebook PCs and PCMCIA Cards

Notebook or laptop PCs have become ubiquitous as portable computers with the power of desktop PCs. In the realm of data acquisition, laptops are indispensable when performing fieldwork and collecting data at remote sites. The main advantages of notebooks are their small size, low weight, and ability to run for several hours without an external power source. Their main disadvantage is limited expandability.

Notebook PCs are effectively closed boxes. You cannot add a conventional ISA or PCI plug-in card to a laptop to expand its functionality for data acquisition or other purposes. Originally, the only way to use a laptop for data acquisition was through its standard parallel or serial (RS-232) port. Since some data acquisition hardware products do exist for these standard ports (especially RS-232 instruments) notebooks were adequate for some applications but not as versatile as a desktop PC.

To facilitate expandability in portable PCs, the Personal Computer Memory Card Industry Association (PCMCIA) developed the PC Card standard in 1990. The first release (1.0) defined a credit card-size device that could expand the memory of any computer, regardless of bus type or operating system. Release 1.0 PC Cards supported either an 8- or 16-bit bus and were self-configuring and hot-swappable. They proved to be so popular that the standard was expanded, in release 2.0, to include general-purpose I/O devices and thicker cards. As the standard evolved, the PC Cards were developed to provide portable PCs with many of the peripheral functions that desktop PCs use add-in cards for, including modems, network interfaces (Ethernet), hard drives, and data acquisition.

PC Cards come in three sizes, Type I, Type II, and Type III, respectively 3.3 mm, 5.0 mm, and 10.5 mm thick, all using the same 68-pin connector. Type I cards are usually memory devices such as RAM or Flash. Type II cards tend to be I/O devices, including modems, network interfaces, and data acquisition cards. Type III cards are usually hard drives. Table 12-3 shows the pin assignments for a 16-bit PC Card connector that supports both memory and I/O devices.

A later release of the PC Card standard in 1995 defined CardBus, a 32-bit bus with speeds up to 33 MHz. CardBus is essentially a modified PCI bus, using the PCI protocol. For example, CardBus supports multiple bus masters and arbitration. It still uses the same 68-pin connector as older PC Cards and is backward compatible with one exception: all CardBus cards use a 3.3-V supply. Older PC cards that use a 5-V supply cannot plug into a CardBus slot (newer PC Cards that run at 3.3 V will work in a CardBus slot).

TABLE 12-3
16-bit PC Card Pin Assignments

PIN	MEMORY	I/O + MEM	PIN	MEMORY	I/O + MEM
01	GND	GND	35	GND	GND
02	D3	D3	36	CD1#	CD1#
03	D4	D4	37	D11	D11
04	D5	D5	38	D12	D12
05	D6	D6	39	D13	D13
06	D7	D7	40	D14	D14
07	CE1#	CE1#	41	D15	D15
08	A10	A10	42	CE2#	CE2#
09	OE#	OE#	43	VS1#	VS1#
10	A11	A11	44	RSRVD	IORD#
11	A9	A9	45	RSRVD	IOWR#
12	A8	A8	46	A17	A17
13	A13	A13	47	A18	A18
14	A14	A14	48	A19	A19
15	WE#	WE#	49	A20	A20
16	READY	IREQ#	50	A21	A21
17	Vcc	Vcc	51	Vcc	Vcc
18	Vpp1	Vpp1	52	Vpp2	Vpp2
19	A16	A16	53	A22	A22
20	A15	A15	54	A23	A23
21	A12	A12	55	A24	A24
22	A7	A7	56	A25	A25
23	A6	A6	57	VS2#	VS2#
24	A5	A5	58	RESET	RESET
25	A4	A4	59	WAIT#	WAIT#
26	A3	A3	60	RSRVD	INPACK#
27	A2	A2	61	REQ#	REQ#
28	A1	A1	62	BVD2	SPKR#
29	A0	A0	63	BVD1	STSCHG#
30	D0	D0	64	D8	D8
31	D1	D1	65	D9	D9
32	D2	D2	66	D10	D10
33	WP	IOIS16#	67	CD2#	CD2#
34	GND	GND	68	GND	GND

The PC Card standard uses software layers that provide compatibility regardless of the computer system or architecture. On a "Wintel" computer, the primary layers are Card Services and Socket Services. Card Services, the higher layer, provides the application programming interface (API) that allows other software to access PC Cards. It is a client/server model where Card Services is the server and a program calling a card function is the client. The lowest software level is Socket Services, which interfaces to the PC Card hardware. It acts like a BIOS interrupt call in a PC, allowing higher software levels (Card Services) to be hardware-independent. Socket Services can actually be implemented in a computer's BIOS or it can simply be a device driver.

You can also use PC Cards with a desktop PC by adding a PCMCIA interface. This is useful if you need a secure way to quickly move data between notebook and desktop computers without using a network. You can simply swap a PC Card memory or hard drive card between machines.

Most notebook computers have two PC Card slots that will accommodate two Type I or Type II cards (or one Type III card). In addition, USB ports are now common on portable PCs. These are now the best ways to connect data acquisition hardware to a laptop computer.

Many major data acquisition manufacturers produce PC Card products. As we saw in Chapter 11, these include Keithley, Data Translation, and National Instruments. The typical PCMCIA data acquisition card runs at speeds up to 100,000 samples/sec with 12- or 16-bit resolution. These are also the typical parameters of current USB data acquisition modules.

12.3 Industrial and Embedded PCs

There are now many different form factors for PCs other than desktops and notebooks. For example, the standard desktop PC is not always well suited for harsh lab or industrial environments, because of factors such as heat, shock, dust, electrical noise, and vibration. Since more and more PCs are finding their way onto factory floors and similar environments, many manufacturers produce industrial (sometimes called "ruggedized") PCs for this market. In addition, PCs in the form of single-board computers (SBCs) are now commonly embedded within commercial equipment. These embedded PCs are usually a small card with all or most of the features of a standard desktop PC's system unit. PC/104 is the most common standard for small, embedded PCs.

12.3.1 Industrial PCs

Industrial PCs are very similar to their desktop counterparts. The main differences are the components and enclosures used by the different PCs. Some industrial PCs use a standard motherboard with expansion slots in a heavy-duty case while most others use a *passive backplane*. A passive backplane is the backbone of a number of computer bus systems (such as VME and STD BUS) as well as many mainframe computers and early microcomputers (such as the S-100 bus). It is simply an array of connectors, wired together to form a bus, without any active circuitry present. The CPU is on a card, plugged into the bus, just like any other expansion board (such as memory or I/O). This adds extreme flexibility to the system, because upgrading to another processor simply involves switching the CPU card. It also guarantees that all signals required by the CPU are present on the passive backplane, adding flexibility for multiprocessors.

Most of these passive backplane systems (based on PCI, ISA, or both) have more expansion slots than standard desktop PCs. The major penalty for the passive backplane approach is added cost. These PCs are usually more expensive than standard desktop PCs because of their larger mechanical size, greater number of components, larger power supplies, and overall higher component costs. Their biggest advantage is much greater expandability than a standard PC.

Industrial PCs tend to use different form factors. Some basic systems are packed into boxes smaller than a diskette drive while others use large card cages for 19-inch rack mounting. The basic systems all have several expansion slots, whether via a motherboard or a passive backplane. The larger systems have an oversized power supply, a cooling fan with a dust filter, superior electromagnetic shielding (which may be poor in some desktop PCs), and often shock-mounting for the hard disk drive (if one is present). The chassis itself may be sealed against dust and liquids (i.e., using a standard NEMA enclosure).

Some industrial PCs are diskless. These systems, used only for dedicated applications, have programs stored in ROM or Flash memory to emulate disk-based software. This is a viable approach when a PC is embedded into a larger piece of equipment and a disk drive is not needed or is too fragile for a harsh environment. Memory cards that emulate disk drives (usually Flash-based) are also available for conventional desktop PCs as add-in boards or PCMCIA modules.

Most manufacturers of industrial PCs support both ISA and PCI buses. Most systems accept conventional plug-in cards. Some have slots for both PCI and ISA boards. Sometimes these mixed-configuration systems are referred to as PISA.

A continuing problem in the area of industrial PC systems is how to enhance bus standards while maintaining compatibility with products from different manufacturers. One early attempt at an enhanced ISA bus was PCXI, intended as a multivendor standard for data acquisition and industrial instrumentation systems. It incorporated a standard ISA passive backplane and power supply into a modified PC chassis, which was flipped around so I/O connectors faced the front of the unit. Spacing between cards was increased to 1.2 inches to accommodate metal shielding around each board. This reduced the effects of PC-generated electrical noise on data acquisition and instrumentation peripherals. The backplane connections followed the ISA standard with some enhancements for better power distribution and grounding. PCXI was the ISA PC equivalent of VMXI, the VME bus instrumentation standard. Unfortunately, PCXI never became a popular standard (there were even some EISA PCXI systems, at one point).

CompactPCI A popular standard for industrial PCI-based computers, introduced in 1995, is CompactPCI. This is functionally the same as the conventional PCI bus but uses a different Eurocard form factor for plug-in boards (as the VME bus does) along with a more reliable connector. CompactPCI cards are specified for both 3U (100 mm by 160 mm) and 6U (160 mm by 233 mm) Eurocard sizes. The high-density, 2-mm-pitch connector is arranged as 47 rows of 5 pins and provides strong card retention capabilities along with resistance to shock and vibration. The metal front-panel of a CompactPCI computer provides good environmental and EMI shielding.

CompactPCI supports both 32- and 64-bit PCI buses. The connector has 220 pins available (15 are lost to a keying area), with many pins connected to ground (for improved signal shielding) and also controlled signal impedance. This allows Compact PCI computers to have eight slots (compared to four in a standard PCI system). With a PCI bridge IC, a CompactPCI system can easily have 16 slots. Table 12-4 shows the pin assignments for a CompactPCI connector.

Many manufacturers support the CompactPCI standard, producing both industrial computers and plug-in cards. Most of these systems use Intel processors and a version of Microsoft Windows or Linux. Some data acquisition cards are even available as CompactPCI.

PXI In 1997, National Instruments developed the PCI extensions for instrumentation (PXI) specification. This is now an open standard that expands CompactPCI for data acquisition and control systems. PXI defines mechanical, electrical, and software extensions to CompactPCI while allowing interoperability with the older standard. The additional mechanical features are cooling and environmental requirements to allow operation in harsh industrial settings.

PXI adds several electrical features to CompactPCI. First, it defines a 10-MHz reference clock that is distributed to all system peripherals, allowing an easy way to synchronize multiple devices. PXI also adds two trigger buses to the system, to carefully control and synchronize the timing of multiple cards.

TABLE 12-4
CompactPCI Pin Assignments

PIN	Z	A	B	C	D	E	F
01	GND	5V	–12V	TRST#	+12V	5V	GND
02	GND	TCK	5V	TMS	TDO	TDI	GND
03	GND	INTA#	INTB#	INTC#	5V	INTD#	GND
04	GND	BRSV	GND	V(I/O)	INTP	INTS	GND
05	GND	BRSV	BRSV	RST#	GND	GNT#	GND
06	GND	REQ#	GND	3.3V	CLK	AD(31)	GND
07	GND	AD(30)	AD(29)	AD(28)	GND	AD(27)	GND
08	GND	AD(26)	GND	V(I/O)	AD(25)	AD(24)	GND
09	GND	C/BE(3)#	IDSEL	AD(23)	GND	AD(22)	GND
10	GND	AD(21)	GND	3.3V	AD(20)	AD(19)	GND
11	GND	A(18)	A(17)	AD(16)	GND	C/BE(2)#	GND
12–14	KEY AREA						
15	GND	3.3V	FRAME#	IRDY#	GND	TRDY#	GND
16	GND	DEVSEL#	GND	V(I/O)	STOP#	LOCK#	GND
17	GND	3.3V	SDONE	SBO#	GND	PERR#	GND
18	GND	SERR#	GND	3.3V	PAR	C/BE(1)#	GND
19	GND	3.3V	AD(15)	AD(14)	GND	AD(13)	GND
20	GND	AD(12)	GND	V(I/O)	AD(11)	AD(10)	GND
21	GND	3.3V	AD(9)	AD(8)	M66EN	C/BE(0)#	GND
22	GND	AD(7)	GND	3.3V	AD(6)	AD(5)	GND
23	GND	3.3V	AD(4)	AD(3)	5V	AD(2)	GND
24	GND	AD(1)	5V	V(I/O)	AD(0)	ACK64#	GND
25	GND	5V	REQ64#	BRSV	3.3V	5V	GND
26	GND	CLK1	GND	REQ1#	GNT1#	REQ2#	GND
27	GND	CLK2	CLK3	SYSEN#	GNT2#	REQ3#	GND
28	GND	CLK4	GND	GNT3#	REQ4#	GNT4#s	GND
29	GND	V(I/O)	BRSV	C/BE(7)#	GND	C/BE(6)#	GND
30	GND	C/BE(5)#	GND	V(I/O)	C/BE(4)#	PAR64	GND
31	GND	AD(63)	AD(62)	AD(61)	GND	AD(60)	GND
32	GND	AD(59)	GND	V(I/O)	AD(58)	AD(57)	GND
33	GND	AD(56)	AD(55)	AD(54)	GND	AD(53)	GND
34	GND	AD(52)	GND	V(I/O)	AD(51)	AD(50)	GND

TABLE 12-4
CompactPCI Pin Assignments (*Continued*)

PIN	Z	A	B	C	D	E	F
35	GND	AD(49)	AD(48)	AD(47)	GND	AD(46)	GND
36	GND	AD(45)	GND	V(I/O)	AD(44)	AD(43)	GND
37	GND	AD(42)	AD(41)	AD(40)	GND	AD(39)	GND
38	GND	AD(38)	GND	V(I/O)	AD(37)	AD(36)	GND
39	GND	AD(35)	AD(34)	AD(33)	GND	AD(32)	GND
40	GND	BRSV	GND	FAL#	REQ5#	GNT5#	GND
41	GND	BRSV	BRSV	DEG#	GND	BRSV	GND
42	GND	BRSV	GND	PRST#	REQ6#	GNT6#	GND
43	GND	USR	USR	USR	USR	USR	GND
44	GND	USR	USR	USR	USR	USR	GND
45	GND	USR	USR	USR	USR	USR	GND
46	GND	USR	USR	USR	USR	USR	GND
47	GND	USR	USR	USR	USR	USR	GND

In addition, PXI defines a 13-line, daisy-chained local bus that connects each card slot to its nearest neighbors. This provides a high-speed communications channel between cards that does not reduce PCI bandwidth. These local bus signals are both digital and analog.

PXI defines common software requirements, based on Microsoft Windows. This allows PXI application developers to use industry-standard programming languages, such as Visual C/C++, Visual Basic, and LabVIEW. Further, all PXI cards must have device driver software available and also support the virtual instrument software architecture (VISA) standard.

Several hardware manufacturers support PXI with both computers and cards. National Instruments produces PXI chassis and a wide range of data acquisition and instrument modules for PXI. For example, the PXI-6071 is a data acquisition card for PXI with 12-bit resolution and a conversion rate of 1.25 Msamples/sec.

For more information, refer to the Appendix for listings of industrial PC manufacturers.

12.3.2 Embedded PCs

As a crude generalization, industrial PCs tend to be larger, stand-alone computers housed in a heavy-duty enclosure and composed of several cards plugged into an expansion bus (such as a passive backplane). Embedded PCs tend to be small, composed of only one or a few boards, and housed within another device: the embedded application. Embedded PCs are used within a wide range of dedicated applications, including medical instruments, test equipment, industrial controls, and communications systems.

Often, embedded PCs are single-board computers (SBCs) that provide the functionality of a desktop PC's motherboard along with some add-in card features. These SBCs can be PCI-, CompactPCI-, or ISA-based and plug into a passive backplane. They can also be small, all-in-one cards that have little or no expandability.

A typical SBC contains most standard PC components: CPU, RAM, BIOS ROM, keyboard interface, parallel and serial ports, floppy and hard disk interfaces, and video display interface. It may have additional features for embedded applications: watchdog timer, battery backup SRAM, Flash memory, digital I/O ports, Ethernet interface. If you connect a power supply, keyboard, disk drive, monitor, and mouse to a typical SBC you will have a fully operational PC. For a simple application, you may only need to run MS-DOS (or a ROM-based DOS) on that PC. But you can also run a version of Microsoft Windows, including Windows CE (developed for embedded PCs).

The advantage of embedding the functionality of a small PC within another piece of equipment is the ability to use the same low-cost hardware and software tools (especially operating systems and programming languages) you already use on a desktop PC. This speeds up a product or project development process tremendously. The disadvantage is that the hardware cost will likely be higher than designing in your own simple microprocessor system.

The size of nonstandard, embedded PCs continues to shrink. For example, the DIMM-PC from Jumptec consists of a 66 MHz AMD 486-SX CPU with 16 Mbytes of RAM and 16 Mbytes of Flash ROM, and it measures only 2.7 inches by 1.7 inches by 0.25 inches (just over a cubic inch). Yet, it has enough capability to run a Linux-based Internet server.

PC/104 The most common standard for embedded PCs has been the PC/104 form factor, first published in 1992. PC/104 is electrically an ISA bus standard that defines cards measuring 3.6 $\times$ 3.8 inches and using a stack-through connector that is much more rugged and reliable than ISA edge connectors. PC/104 gets its name from the number of pins on the bus (104), which is just six more than a conventional ISA connector (the extra pins are used for grounds and keying).

PC/104 cards do not use a motherboard or backplane. The cards in a PC/104 system (if more than one is used) simply stack together, since each card's stack-through connector is both a plug and a socket. The boards can

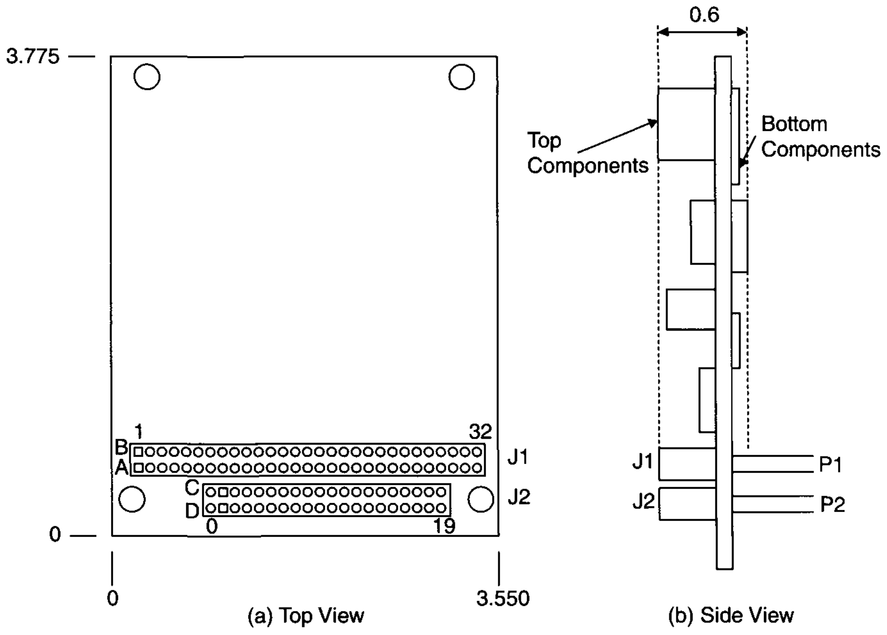


Figure 12-2 PC/104 card—simplified mechanical drawing.

also be mechanically mounted together to secure them against shock and vibration. The standard defines the mechanical specifications as well as the connector pin assignments for PC/104 cards. It also specifies power supply requirements, such as a maximum current of 2 amps at +5 V for a single PC/104 card. Figure 12-2 shows a simplified mechanical drawing of a PC/104 card.

PC/104 defines two connectors: J1/P1 and J2/P2. For an 8-bit card, equivalent to an XT board, only the 64-pin J1/P1 is used. A 16-bit (AT) card has an additional 40-pin connector, J2/P2. All electrical design rules for ISA boards apply to PC/104 cards. The pin assignments for PC/104 cards, shown in Table 12-5, are very similar to those for ISA cards but are not exactly the same.

Many manufacturers produce PC/104 cards: single-board computers, video cards, network cards, memory expansion cards, PCMCIA interface cards, and also data acquisition cards. For example, Diamond Systems Corp., a leading PC/104 card manufacturer, produces several data acquisition boards, including the Diamond-MM-AT. This is a multifunction PC/104 data acquisition card with 16 analog input channels and a 12-bit ADC running at conversion rates up to 100,000 samples/sec. It also has two 12-bit analog

TABLE 12-5
PC/104 Pin Assignments

	8-BIT SIGNALS		16-BIT EXTENSION	
PIN NUMBER	J1/P1 ROW A	J1/P1 ROW B	J2/P2 ROW C	J2/P2 ROW D
00			GND	GND
01	IOCHCHK*	GND	SBHE*	MEMCS16*
02	SD7	RESETDRV	LA23	IOCS16*
03	SD6	+5V	LA22	IRQ10
04	SD5	IRQ9	LA21	IRQ11
05	SD4	–5V	LA20	IRQ12
06	SD3	DRQ2	LA19	IRQ15
07	SD2	–12V	LA18	IRQ14
08	SD1	ENDXFR*	LA17	DACK0*
09	SD0	+12V	MEMR*	DRQ0
10	IOCHRDY	Key	MEMW*	DACK5*
11	AEN	SMEMW*	SD8	DRQ5
12	SA19	SMEMR*	SD9	DACK6*
13	SA18	IOW*	SD10	DRQ6
14	SA17	IOR*	SD11	DACK7*
15	SA16	DACK3*	SD12	DRQ7
16	SA15	DRQ3	SD13	+5V
17	SA14	DACK1*	SD14	MASTER*
18	SA13	DRQ1	SD15	GND
19	SA12	REFRESH*	Key	GND
20	SA11	SYSCLK		
21	SA10	IRQ7		
22	SA9	IRQ6		
23	SA8	IRQ5		
24	SA7	IRQ4		
25	SA6	IRQ3		
26	SA5	DACK2*		
27	SA4	TC		
28	SA3	BALE		
29	SA2	+5V		
30	SA1	OSC		
31	SA0	GND		
32	GND	GND		

outputs, 16 digital I/O lines, and a counter/timer. Diamond Systems also has the Prometheus PC/104 card, which is a 486-based SBC with an Ethernet interface and data acquisition hardware.

PC/104-PLUS PC/104 cards are adequate for 8- or 16-bit applications that do not require high performance. They have the same performance limitations as ISA products. To address this, the PC/104 Consortium developed the PC/104-Plus standard in 1997. PC/104-Plus adds a PCI bus to PC/104 cards, via a third connector. This PCI connector (J3/P3) is a 120-pin, 2-mm-pitch stack-through, arranged in a 4 × 30 array and placed on the opposite end of the card from the original PC/104 connectors (J1/P1, J2/P2).

PC/104-Plus only has a 32-bit PCI interface with a 33-MHz maximum clock speed and does not support 64-bit extensions. Still, it is a complete implementation of the 32-bit PCI bus with all the improvements over ISA. The cards still keep the same form factor of 3.6 × 3.8 inches. A PC/104-Plus card can have both ISA and PCI interfaces (using all three connectors) or only PCI. If a processor card supports both buses, you can mix ISA (PC/104) and PCI (PC/104-Plus) peripherals in the same system. Table 12-6 shows the pin assignments for the PCI connector on a PC/104-Plus card.

One additional standard to note is the EBX (embedded board expandable) form factor, developed by Ampro and Motorola. EBX is electrically the same as PC/104 or PC/104-Plus but it uses a larger card size: 5.75 × 8 inches, allowing for higher levels of integration. As an example, Toronto MicroElectonics, Inc., produces the model 5811 PC/104-Plus SBC using the EBX form factor. This SBC runs a 450 MHz AMD K6 CPU with as much as 512 Mbytes of SDRAM and a 144-Mbyte Flash Disk. It contains a dual IDE interface, a SCSI interface (Ultra Wide), a 10/100Base-T Ethernet interface, a video interface, four serial ports, a parallel port, two USB ports, and a touch-screen interface—all on a single EBX card.

The 16-bit PC/104 standard is still quite popular and is the last remaining stronghold of the ISA bus. As PC/104-Plus continues to grow in popularity it is likely to become the dominant standard for embedded PCs in the foreseeable future.

12.4 Image Capture Products

Image capture (or machine vision) is a growing area of specialized data acquisition for PCs, enabled by the speed and power of current PC hardware and operating systems. Machine vision is now commonly used for industrial automated inspection systems, with applications including reading bar codes on objects, verifying correct assembly of manufactured parts, and even checking

TABLE 12-6
PC/104-Plus Pin Assignments

PIN NUMBER	J3/P3 ROW A	J3/P3 ROW B	J3/P3 ROW C	J3/P3 ROW D
01	GND (5.0V Key)	Reserved	+5V	AD00
02	VI/O	AD02	AD01	+5V
03	AD05	GND	AD04	AD03
04	C/BE0*	AD07	GND	AD06
05	GND	AD09	AD08	GND
06	AD11	VI/O	AD10	M66EN
07	AD14	AD13	GND	AD12
08	+3.3V	C/BE1*	AD15	+3.3V
09	SERR*	GND	SB0*	PAR
10	GND	PERR*	+3.3V	SDONE
11	STOP*	+3.3V	LOCK*	GND
12	+3.3V	TRDY*	GND	DEVSEL*
13	FRAME*	GND	IRDY*	+3.3V
14	GND	AD16	+3.3V	C/BE2*
15	AD18	+3.3V	AD17	GND
16	AD21	AD20	GND	AD19
17	+3.3V	AD23	AD22	+3.3V
18	IDSEL0	GND	IDSEL1	IDSEL2
19	AD24	C/BE3*	VI/O	IDSEL3
20	GND	AD26	AD25	GND
21	AD29	+5V	AD28	AD27
22	+5V	AD30	GND	AD31
23	REQ0*	GND	REQ1*	VI/O
24	GND	REQ2*	+5V	GNT0*
25	GNT1*	VI/O	GNT2*	GND
26	+5V	CLK0	GND	CLK1
27	CLK2	+5V	CLK3	GND
28	GND	INTD*	+5V	RST*
29	+12V	INTA*	INTB*	INTC*
30	−12V	Reserved	Reserved	GND (3.3V Key)

the color of pharmaceutical pills. Scientific and engineering applications include medical image processing and spectral analysis. The heart of PC-based image capture applications is the *frame grabber* board. This is a plug-in card that converts the signal from an external analog or digital camera to a digital format that can be stored in PC memory. Once the image has been acquired, the PC can process the data to determine the required information (i.e., does the inspected part exceed dimensional tolerances?).

Most current frame grabbers are PCI cards since ISA cards tend to be too slow for many applications. Consider a standard composite video camera with a frame rate of 30 frames/sec and a 640 × 480 pixel resolution. If 16 bits (two bytes) were used to represent each pixel, the data rate produced by this camera would be over 18 Mbytes/sec. This is too fast for ISA's maximum transfer rate of approximately 2–5 Mbytes/sec, but well under the maximum 32-bit PCI rate of 132 Mbytes/sec.

An analog video frame grabber is basically a specialized, high-speed ADC card with dedicated analog processing (such as a sync stripper) as well as digital image processing and storage (FIFO) capabilities. A digital video frame grabber is a specialized digital processor that converts the input data stream (usually RS-422 or LVDS signals) into a format usable by the PC and stores it at high speed into the PC's memory, via the PCI bus.

One data acquisition manufacturer, Data Translation, is also a leader in machine vision products for PCs. Data Translation produces a wide range of analog and digital video frame grabbers as well as image processing software that runs under Microsoft Windows. For example, Data Translation's DT3120 is a low-cost PCI frame grabber for monochrome and composite color (analog) video signals. It uses two 8-bit, 40-Msamples/sec ADCs and outputs the data as 16-, 24-, or 32-bit pixels at a typical rate of 55 Mbytes/sec.

An example of a digital camera frame grabber is Data Translation's DT3140. This PCI card accepts RS-422 and RS-644 (LVDS) inputs, from 8 to 20 bits wide with clock speeds up to 60 MHz. Note that digital cameras use a synchronous, high-speed clock and output one pixel per clock cycle. The DT3140 has a typical data output rate (to the PCI bus) of 100 Mbytes/sec.

One interesting aspect of digital camera frame grabbers is that they are really just high-speed PCI interfaces, suitable for transferring any digital data (in the correct format) into a PC's memory. For example, let us assume we have custom hardware that generates a large amount of 16-bit digital data at a rate of 20 MHz. It would be much easier to format our digital data to look like the output of a digital camera (simply add a 20 MHz pixel clock, line and frame trigger signals, and RS-422 or RS-644 drivers) than to develop our own high-speed PCI interface. Alternatively, we could try to transfer our data into a PC using an IEEE-1394 interface. However, this is still much more complex than the relatively simple digital camera interface.

Data Translation also produces software products for their frame grabbers as well as for machine vision applications in general. For example, DT Vision Foundry is software for image inspection applications that runs under MS Windows 98/NT/2000. Global Lab Image/2 is software for scientific and general-purpose image analysis applications that runs under MS Windows 98/NT/2000/Me.

Another data acquisition manufacturer with machine vision products is National Instruments. For example, the National Instruments PCI-1407 (also available as a PXI card) is an analog monochrome video frame grabber for the PCI bus with an 8-bit flash ADC. The PCI-1424 is a digital camera frame grabber for the PCI bus with a 32-bit input (RS-422, LVDS, or TTL signals) and a maximum pixel clock of 50 MHz. It has 16 to 80 Mbytes of on-board SDRAM to buffer high-speed data. National Instruments also produces machine vision software, such as IMAQ Vision Builder for Windows 98/NT/2000, that is used to develop image-processing applications.

There are many other image capture manufacturers who produce high-performance products with specialized image processing features built into the hardware. A few of these manufacturers include BitFlow and Coreco. Refer to the Appendix for more vendor listings.

This completes our overview of other PC configurations and hardware. In the next chapter, we will examine programming languages and the trade-offs of writing your own software.

CHAPTER

13

Computer Programming Languages

There may be times when you need a PC to perform a task not supported by commercially available software, such as implementing a new signal-processing algorithm. This requires a custom program that you will have to write yourself or pay someone else to write. Unless you have very demanding requirements (very high-speed operation or extremely large amounts of data to manipulate), any computer language you are familiar with should enable you to do the job. If you are new to programming, selecting a computer language can be confusing. The best approach is to choose one of the more commonly used languages (such as BASIC, Visual Basic, C/C++, or Visual C++) which have a lot of support available for PC use. This support takes the form of a widely accepted PC version of the language, availability of many third-party software products (including libraries and debugging tools), and a good choice of introductory books for using that language on a PC running your chosen operating system.

Under Microsoft Windows 3.1/95/NT and above, programming is much more complex than for MS-DOS. However, many popular Windows programming languages (such as Microsoft Visual Basic and Visual C++) automate most of the required details and give you a basic program framework to fill in. The result is that Windows programming, using the appropriate software, can actually be easier than writing programs for DOS.

In this chapter, we will first examine some of the important distinctions between different types of computer languages as well as their similarities (especially in MS-DOS and MS Windows PC environments). Then we will go on to look at a few popular languages in detail.

The only language a computer understands is its *machine language*—the binary commands telling it exactly what to do. The standard programming languages in common use convert logical constructs and instructions that make sense to people into a series of commands a computer can understand and carry out. A set of commands used to perform some desired function is considered a program.

The terms *high-level* and *low-level* are often associated with computer languages. A low-level language is very close to machine language. The most common instance of this is an *assembly language*, which performs a one-to-one conversion of simple mnemonic commands into machine-language commands, or *object code*. For example, in the Intel 80x86 family, there is a command to multiply two 8-bit numbers, MUL. A simple example using this command to multiply 17 and 32 (decimal) is as follows:

```
MOV     AL,17     ;load 17 into register AL
MOV     BL,32     ;load 32 into register BL
MUL     BL        ;multiply AL by BL
                  ;16-bit product is in AX (AH, AL)
```

Each line of assembler code (ignoring the comments after the semicolon) is translated into several bytes representing one computer command, or *opcode*. Assembly language is a simpler way for a person to represent the machine-language commands. The same commands in machine language (using hexadecimal notation) would be

```
B0 11
B3 20
F6 E3
```

Obviously, the assembler mnemonics make more sense than the machine-language commands.

A high-level language is more abstract than a low-level language. Processor details (such as which register contains which operand) are invisible to the programmer. In fact, under protected-mode operating systems (Windows 95/NT and above) operating system details are also hidden from the programmer. Only the important operations and logic are visible. A high-level language is processor-independent, making it portable. Reprogramming the previous example in C produces the following commands, which will work on a 68000 series CPU as well as an 80x86 processor:

```
char a=17, b=32;
int c;
c = a * b;
```

In this case, we do not need to know where the two operands are stored. We will let the compiler worry about those details.

Besides high-level versus low-level, there are two other distinctions between different types of computer languages: *compiled* versus *interpreted.* In a compiled language, the program under development (one or more ASCII files) is translated into machine language through a separate, independent series of steps. The result of the one-time compilation process is an executable binary file (.EXE in DOS or Windows) which can then be run by the appropriate operating system. In an interpreted language, the program is translated into machine language one line at a time, as the program is being run. Each time the program is run, it is translated into machine language again. The program has to be run from within the interpreter, which is itself a special program running under the operating system. An example of an interpreted language is GW-BASIC (for MS-DOS). Most other common computer languages, such as C/C++, FORTRAN, and Pascal, are compiled.

A compiled program executes much faster than an interpreted program, since it has already been translated into machine language. It usually requires less free memory space, since a compiled program does not need the extra overhead of an interpreter. The main advantage of an interpreted program is the flexible user interactions available. With GW-BASIC, for example, you can control where to start, stop, and continue program execution and check variable values without modifying the program or leaving the BASIC environment. In a compiled language, these features have to be written into the program and each modification requires a new compilation process.

Of course, many debugging programs exist to assist in the development of compiled programs. A typical debugger provides an environment that allows the programmer to control execution, check variables, modify data, and perform many different tests on the program under development. An example of such a debugger, for MS-DOS systems, is Microsoft's CodeView, which supports both low-level (Assembler) and high-level (C, FORTRAN, Pascal) languages. Under Windows, Microsoft Visual Basic and Visual C++ provide a complete program development environment with built-in debugging facilities.

For compiled languages, the complete compilation process requires at least two discrete steps: compilation and linking. Under MS-DOS and Windows, compilation consists of translating the original ASCII program file (the source code) into a machine-language .OBJ file (the object code or object module). The object code file is not executable. It lacks several important pieces. The linking process takes the object code and adds any library functions it requires, as well as commands that are defined in other files (or object modules), and produces an executable file.

For example, in C, the printf command displays a text string on the screen. It is a standard C library function. If an object module calls printf, this function must be extracted from the library. The linking process links the new program's object code with other object modules, from standard libraries and user-developed sources. The linker makes sure all function and variable names are defined (and do not clash) and decides where the various code modules should be located in memory. Linking adds all the remaining information the operating system (i.e., DOS or Windows) needs to load the finished program into memory and run it. The output of the linking process is an executable (.EXE) file.

The actual compilation and linking processes may each take several steps, though this is usually invisible to the program developer (such as processing the source file multiple times). Most PC linkers (including the LINK program provided with MS-DOS) support many different options, such as control over where to place the completed program in memory and how to include information for debuggers. An additional step, after linking, is required to convert a .EXE program into a .COM program for DOS. If the entire program was kept within a single 64-Kbyte memory segment, it can be processed by the DOS command EXE2BIN, which converts it to a .COM file.

It should be noted here that not all computer languages neatly fit into the categories of compiled versus interpreted or high-level versus low-level. There is no doubt, for example, that C is a high-level programming language (and one of the most popular). Yet, it is a fairly "bare-bones" language having a moderately sparse set of commands, making it similar in some ways to low-level programming languages. It is the additional libraries that are packaged with C compilers, along with the extreme flexibility of the language, that make it so popular. C is a very efficient language, producing relatively small programs (small executable files) compared to a less efficient high-level language such as FORTRAN. It executes commands quickly, since a command in C is translated into a relatively small number of machine-language commands, again making it appear similar to a low-level language. This is why C is often used to produce programs for embedded processors.

An example of a programming language with properties of both a compiler and interpreter is FORTH. Commands (or words, as they are called in FORTH) are executed in binary (machine language) form, as in a compiled language. However, each word is executed separately under control of the environment. In addition, new words can be defined as combinations of old words. These new words then get translated into machine language before they can be executed. This type of language, having some properties of both a compiler and an interpreter, is called an incremental compiler. The commands (words) are compiled one at a time, with new ones built upon combinations of existing ones. Another example of a high-level language with both interpreter and compiler characteristics is MATLAB, which we looked at in Chapter 11.

Now we will examine a few popular programming languages in greater detail.

13.1 Popular Programming Languages

Most text-based programming languages (as opposed to some of the special-purpose, graphics languages we examined in Chapter 11, such as LabVIEW) can be used with any standard operating system on nearly any computer. For example, C was originally written for UNIX (running on mainframe computers) and it is now available for nearly all operating systems, including MS-DOS and Windows.

Microsoft Windows-based programming languages (such as Visual C++) are usually part of a sophisticated application development environment that simplifies the otherwise tedious task of writing a complete Windows program. Hence, we will treat programming for MS Windows as a separate topic, later in this chapter.

In this section, we will discuss some major programming languages, both generically (independent of operating system) and when used in a text-based environment, when running an MS-DOS compiler/linker on a PC. We will start with assembly language, the lowest level of programming languages commonly employed.

13.1.1 Assembly Language

Assembly language (or Assembler) is a compiled, low-level computer language. It is processor-dependent, since it basically translates the Assembler's mnemonics directly into the commands a particular CPU understands, on a one-to-one basis. These Assembler mnemonics are the instruction set for that processor. In addition, an Assembler provides commands that control the assembly process, handle initializations, and allow the use of variables and labels as well as controlling output.

On PCs, Assembler is normally used only under MS-DOS. When running a 32-bit, protected-mode operating system (including Windows 95/NT and above), low-level programs which directly access registers or memory locations produce protection violations. All low-level access must be made through appropriate software drivers.

For MS-DOS PCs, the most popular Assembly language was Microsoft Macro Assembler, or MASM. As with most popular compilers, MASM was upgraded on a regular basis. Most of this discussion refers to version 5.0 or later, which simplified the use of certain directives and included support for instructions available only on 80286 and 80386 CPUs.

A *directive* is an Assembler command that does not translate into an executable instruction, but directs MASM to perform a certain task facilitating the Assembly process. An executable instruction is sometimes referred to as an op code, while an Assembler directive may be referred to as a pseudo-op code. Directives can tell MASM many different things, including which memory segment is being referred to, what the value of a variable or memory location is, and where program execution begins.

One important MASM directive is .MODEL, which determines the maximum size for a program. Remember that for an 80x86 family CPU, memory is addressed as segments, up to 64 Kbytes in length. If 16-bit addressing is used (for code or data) only a single 64K segment will be accessed. The *memory model* of a program defines how different parts of that program (code and data) access memory segments. Five memory models are supported by MASM for DOS programs: Small, Medium, Compact, Large, and Huge. In the Small model, all data fits within one 64K segment and all code (executable instructions) fits within another single 64K segment. In the Medium model, all data fits within one 64K segment but code can be larger than 64K (multisegment, requiring 32-bit addressing for segment:offset). In the Compact model, all code fits within one 64K segment but data may occupy more than 64K (but no single array can be larger than 64K). In the Large model, both code and data may be larger than 64K (still, no single data array can exceed 64K). Finally, in the Huge model, both code and data can be larger than 64K and data arrays can also exceed 64K.

Since larger models require larger addresses, they produce bigger and slower programs than a smaller model will. In selecting a model for a program, try to estimate the maximum amount of data storage you will need. Let us say you are writing an FFT program, using 16-bit integer math and a maximum sample size of 2048 points. Since each point requires two integers (real and imaginary) and each integer is 2 bytes long, you need 8096 bytes just to store the input (or output) data. Even if you had separate arrays for input and output data, that would still be only 16,192 bytes. As a safety margin, for temporary storage, we will double this number, to 32,384 bytes, which is only half of a 64K segment. It is more difficult to estimate the size of the code. In this example, we would start with the Small model. If the code turned out to be larger than 64K (which is not easy to do in assembly language), we would move to the Medium model. These same memory models also apply to Microsoft's high-level DOS language compilers. If you are writing a MASM program to work with another high-level language, you should use the same memory model for both.

Here is an example of a simple MASM program that displays a text string ("This is a simple MASM program") on the screen using DOS function 09h:

```
     DOSSEG                         ;Let MASM handle the segment order
     .MODEL   SMALL                 ;Small model is adequate for this
     .STACK   400h                  ;Set aside 1024 bytes for a stack
     .DATA                          ;Start of the data segment
text DB       "This is a simple MASM program"
     DB       0Dh, 0Ah, 24h         ;End with CR, LF and $ char
     .CODE                          ;Start of code segment
go:  mov      ax,@DATA              ;Load data segment location into DS
     mov      ds,ax
     mov      dx,OFFSET text        ;Now DS:DX points to text
     mov      ah,09h                ;DOS string display function number
     int      21h                   ;Call DOS function
     mov      ax,4C00h              ;Load DOS exit function number
     int      21h                   ;Call DOS function (exit)

     END      go                    ;Start execution at label go
```

Several directives are used here. DOSSEG tells MASM to take care of the order of the various segments (code, data, stack), a detail we would rather ignore. The directive .DATA indicates the start of the data segment while .CODE indicates the start of the code segment. The message is referred to by the label *text*, where the DB directive (Defines Bytes) indicates that this is byte data (the quotation marks indicate ASCII text). The string must be terminated by ASCII character 24h ("$") for DOS function 09h. The executable instructions are placed in the code segment. The label, *go*, refers to the start of the program. The address of the text string is loaded into registers DS:DX. Then DOS function 09h is called, to display the string. Finally, DOS function 4Ch is called to exit the program and return to DOS. The final END directive tells MASM to begin program execution at the label (address) *go*.

MASM is called a Macro Assembler, because it supports the use of macros. A *macro* is a block of program statements that is given a symbolic name that can then be used within the normal program code. A macro can also accept parameters when it is called within a program. When the source file is assembled by MASM, any macros are expanded (translated) to their original definition text. This is very handy if the same section of code, such as a programmer-defined function, is used repeatedly. Often, predefined macros may be kept in a separate file, along with other information, such as variable initializations. The INCLUDE directive can read this file in during assembly.

This brief overview of MASM has barely scratched the surface of assembly language. Check the bibliography for other books on this subject. Again, you should write a program in assembly language only if you are working in DOS and a high-level language is inadequate for your task. Even then, you can usually get away with just writing the most critical sections in MASM and calling them from a high-level language. Next, we will look at a popular, high-level, interpreted language: BASIC.

13.1.2 BASIC

BASIC was probably the most popular interpreted computer language used on early PCs. This was due, in large part, to it being included with IBM-DOS and MS-DOS packages. In fact, original IBM PC systems had BASIC in ROM, to save RAM space for programs. The first IBM PC had 64 Kbytes of RAM and a floppy disk drive was optional. If no disk drive was present, the system would start up in BASIC (since you needed a disk drive to boot up DOS). PC compatible manufacturers did not put BASIC in ROM, but ran Microsoft's GW-BASIC from RAM, like any other program. GW-BASIC was functionally equivalent to IBM's BASIC and BASICA.

BASIC, which is an acronym for Beginner's All-purpose Symbolic Instruction Code, was originally developed at Dartmouth College as a tool for teaching fundamental programming concepts. It is one of the easiest programming languages to learn and use. It does have serious drawbacks. Being interpreted, it executes slowly. This becomes especially obvious when performing a real-time task, such as controlling serial communications at high data rates on an older PC. Also, BASIC does not easily lend itself to developing neat, modular programs. It is a good tool for learning, experimenting, and quickly prototyping software algorithms. It is not well suited for developing high-performance or commercial-quality software for DOS applications. In addition, standard BASIC can only use 64 Kbytes of memory for data and stack storage. Under Windows, Visual Basic overcomes many of these limitations.

Interpreted BASIC has two modes of operation: *direct mode* and *indirect mode*. In the direct mode, BASIC commands and statements are executed as soon as they are entered. Results of calculations can be displayed or saved in a variable for further use. The statement or command lines themselves are lost after execution. Direct mode is useful for quick calculations or debugging operations (such as displaying or loading variable values). BASIC can accept a direct command when it is at the command level, displaying the OK prompt. An example of a direct-mode command to display the result of a calculation would be

```
PRINT 23 * 17 + 2
```

The PRINT command displays the result on the screen (LPRINT sends output to the printer).

In the indirect mode, lines of program statements are stored in memory. A line number precedes each program line. If a line number is missing, that command line is treated as a direct-mode statement. After all program lines are entered, the program can be executed via the RUN command. The sequence of program execution starts with the lowest line number and continues through to the highest line number, unless a special statement (such as GOTO) explicitly changes the order. For example, a simple BASIC program to perform the direct-mode calculation from the last example could be a single line:

```
10 PRINT 23 * 17 + 2
```

Or, it could be more generalized, with multiple lines:

```
10 A = 23
20 B = 17
30 C = 2
40 PRINT A * B + C
```

Here, using the variables A, B, and C, the numbers fed into the calculation can be quickly changed. An even better way would be to enter the variable values when the program is run:

```
10 INPUT "ENTER A: ", A
20 INPUT "ENTER B: ", B
30 INPUT "ENTER C: ", C
40 PRINT "A * B + C = "; A * B + C
```

In this case, the simple program is now general-purpose. The user determines the variable values each time the program is run, using the INPUT statement, which prompts the user with the text enclosed within quotes. When the program is run, the screen would look as follows, with the operator's responses underlined:

```
RUN
ENTER A: 12
ENTER B: 7
ENTER C: 21
A * B + C = 105
OK
```

It is important to note that not all BASIC statements can be used in the direct mode (the indirect mode uses all of them), including GOSUB and RETURN for executing subroutines.

BASIC has a rich set of commands. It has a full range of mathematical and trigonometric functions, supporting both integer and floating-point calculations. It has commands for manipulating text strings, handling data file operations (supporting both ASCII and binary formats), and operator interfacing. It has several statements for program control, such as IF, THEN and FOR, NEXT. BASIC directly supports many aspects of a PC's hardware and software (DOS) environment. It can read the system clock (via TIME$), directly input from or output to an I/O port (via INP and OUT), or even read from and write to system memory locations (via PEEK and POKE). BASIC provides many functions for controlling screen display, with both text and graphics (if appropriate display hardware is present in older PCs). In addition, BASIC can call assembly-language routines for functions it cannot directly perform (or cannot perform quickly enough). The assembler code has to be properly written to allow interfacing to a BASIC program.

BASIC provides an environment that simplifies the process of program development. Besides just entering program lines, BASIC has special commands for modifying programs. EDIT allows you to modify the specified line. RENUM automatically renumbers the program lines, which is necessary if you need to add a new program line between two existing lines with consecutive numbers. You can save a program onto a disk file (SAVE) or retrieve a previously saved program (LOAD). You can display a program on the screen (LIST) or send it to a printer (LLIST). You can even use a special trace mode (via TRON, TROFF), which displays the program line numbers as they are executed.

An important aspect of BASIC is that all the variables are global. Any part of a program can change the value of any variable. In some respects this can be handy. A subroutine does not explicitly return any value to the main program, it just writes to the appropriate variables. The flip side of this can be a problem, if you lose track of which variables are being used by which subroutines. Great care must be taken in keeping track of variables in BASIC.

Consider the following program which averages 10 values in the array A(I):

```
10 DIM A(10)
20 FOR I = 1 TO 10
30 READ A(I)
40 NEXT I
50 DATA 1.1, 2.3, 5.7, 6.4, 2.9
60 DATA 3.0, 2.1, 4.0, 1.9, 8.4
70 GOSUB 500
80 PRINT "DATA AVERAGE = "; AVG
90 STOP
500 REM - SUBROUTINE AVERAGES I VALUES IN A(I)
510 AVG = 0
520 FOR J = 1 TO I - 1
530 AVG = AVG + A(J)
540 NEXT J
550 AVG = AVG/(I - 1)
560 RETURN
```

There are several points to note in this illustrative program. Line 10 defines the data array A(I). The FOR ... NEXT loop in lines 20–40 loads 10 values into the array, from the DATA statements in lines 50 and 60. The mean value is calculated by the subroutine in lines 500–560. This subroutine is called via the GOSUB command and is terminated by the RETURN command. The values in A(I) are available to the subroutine, which first uses the variable AVG to accumulate all 10 values, with the FOR ... NEXT loop in lines 520–540. Then the average is calculated and stored in AVG, which is used by the main program in its PRINT statement (line 80).

Note that the FOR ... NEXT loops use a variable to keep track of how many times that loop is executed. Even though the range of *I* in line 20 is specified as 1 to 10, since *I* gets incremented at the end of each loop (before its value is tested), the final value of *I* is 11 when the looping is terminated. That is why the subroutine FOR ... NEXT loop, starting at line 520, loops through *I* – 1 times. If both the main program's and the subroutine's FOR ... NEXT loops used the same index variable (*I*), the program could not run properly.

BASIC was so popular for early PC use that many enhancements were provided, making it closer to a professional-quality language. Several compiled versions of BASIC were available for DOS. You could prototype and debug a program in interpreted BASIC and then compile it, with few modifications, if any.

Some manufacturers of data acquisition products provided extensive BASIC support for their hardware. This included assembly language driver functions that could be called from a BASIC program. Other manufacturers produced their own enhanced version of BASIC to support their hardware and extend the language's capabilities.

There were several, general-purpose, compiled versions of BASIC available, such as Microsoft's QuickBasic for MS-DOS. As the new BASIC versions evolved, they became more like other conventional, structured, compiled programming languages. Visual Basic (for Windows) is a good example of this.

Now, we will look at a few high-level compiled languages, starting with C.

13.1.3 C Programming Language

C is one of the most popular general-purpose computer languages used by professional programmers. As we discussed previously, C combines the best features of low-level languages (ability to directly access hardware and to produce fast, efficient code) with those of high-level languages (supports abstract data structures, handles complex mathematical calculations, is well structured and maintainable).

The power and popularity of C reside, paradoxically, in its inherent simplicity. In one sense, C is not very robust, because it lacks many functions present in other high-level languages, such an x^2 command. However, it contains all the building blocks to create this function along with any other high-level language operation. Many of these features are present in standard libraries that are part of a commercial C compiler package. In addition, C contains many operators not found in most high-level languages, such a bit manipulation commands. Since C is modular, it is easy to add new functions, as needed, and use them as if they were an inherent part of the system.

C is also a well-standardized language. It was developed at AT&T Bell Laboratories, during the early 1970s, by Dennis Ritchie, where it was well controlled. The language is defined by the standard text, *The C Programming Language*, by Kernighan and Ritchie. Virtually all commercial C compilers adhere to this or a later ANSI standard (although some may add enhancements, along with additional function libraries).

To illustrate some of the features of C, here is the program for calculating an average value, from the previous BASIC section, rewritten in C:

```
float a[10]= {1.1,2.3,5.7,6.4,2.9,     /* define data array */
              3.0,2.1,4.0,1.9,8.4};

main()                       /* Program execution starts with main */
{
float avg;                   /* variable for average value */
avg = calc_avg(a,10);        /* calculate average value of array */
printf("DATA AVERAGE = %f\n",avg);       /* display result */
}                            /* End of main program */

float calc_avg(data,nval)    /* Subroutine calculates average value
                                in array data, nval points long */
float *data;                 /* data points to input data array */
int nval;                    /* nval contains # of values to avg */
{
int i;                       /* misc variables */
float x;

for(i=0, x=0.0; i < nval; ++i) /* main calculation loop */
     {
     x += *(data + i);       /* Add data values into x */
     }
x /= nval;                   /* calculate average (sum/nval) */
return(x);                   /* return result to main program */
}                            /* End of Subroutine */
```

Many aspects of C are shown in this example. Functions (including the main program and any subroutines) are specified by a name followed by parentheses, with or without arguments inside. The statements comprising the function are delimited by the braces, { }. These same braces also delimit various loops within a function or even array initialization data (for a[]). Program execution starts with the function main(), the main program. When another function name appears, such as calc_avg(), that function starts execution. When it completes, control is returned to main(), along with a return value (if any). Statements in C are terminated by a semicolon (;) and pairs of special characters (/* */) delimit comments. Statements (and comments) can span multiple lines. C is not rigorous about text formatting in the source code. It allows programmers to format a file for easy readability. In this respect, C is a fairly free-form language. Also, it does not use line numbers, although you can give statements a label.

There are many important facets of C. One of these is *function privacy.* Any variable defined and used within a function is private or local to that function. Another function cannot directly access that variable. This is in sharp contrast to BASIC, where all variables are global (none are private). When a function sends a variable value to another function, it sends a copy of that variable, so the original cannot be changed by the other function. For a variable to be global, it must be defined outside of a function. In the preceding example, a[] is a global array of floating-point numbers. The only reason a[] was made a global array, instead of a local array in main(), was to initialize its values more easily. Also note that all variables have to be explicitly declared before they can be used. As opposed to some other languages (MATLAB, for example), C must know explicitly what all the variable types are (integer, floating-point) before it can use them.

Another significant aspect of C is the use of pointers. In C, any variable (scalar or array) has two values associated with it: the *lvalue* and the *rvalue.* The lvalue is the address of a variable, while the rvalue is its actual numeric content. A pointer is used to address a variable (or a memory location). If we have a pointer, pntr1, containing the address of a variable, we can store the value of that variable in another variable, x, with the *indirection* operator, *, as follows:

```
x = *pntr1;
```

Similarly, if we want another pointer, pntr2, to contain the address of the variable x, we can use the *address of* operator, &, as follows:

```
pntr2 = &x;
```

The utility of this pointer scheme is shown in the program example just given. Function calc_avg() defines two dummy parameters, data and nval. The parameter data is defined as a pointer to an array of floating-point values via

```
float * data;
```

If data was just a scalar variable, it would be defined without the indirection operator:

```
float data;
```

By specifying this parameter as a pointer, we do not have to pass 10 variables to calc_avg(). In addition, the function can handle input data arrays of variable length—it just needs to know where the array starts (via data), how long it is (via nval), and how big each element is (via the float declaration for *data).

Another aspect of pointers is that they are the means to circumvent variable privacy. The only way one function can modify the rvalue of another function's local variable is if that function sends it the lvalue (pointer) of that variable. This is necessary when a relatively large amount of data must be passed between functions. Still, this is done explicitly, and the indirection operator must be used to access the variable, from its pointer.

Several aspects of the notations used in C can be bewildering at first. One source of confusion is = (the *assignment* operator) versus == (the *equality* operator). The assignment operator is used to assign the rvalue of a variable, as in most high-level languages:

```
x = 10;
```

The equality operator tests a statement to see if it is true or false (in C, false is considered 0 and true is considered nonzero). So, a conditional statement checking if x equals 10 would be

```
if(x == 10)
        {
        /* conditional statements here */
        }
```

If x does equal 10, any statements within the braces would be executed.

Other notation in the sample program may seem odd. C allows for special assignment operators, such as += or /= (as used in the sample program). The statement

```
x += 10;
```

is equivalent to

```
x = x + 10;
```

Similarly,

```
x /= nval;
```

is equivalent to

```
x = x / nval;
```

These assignment statements are notational conveniences. Other important operators are *increment* (++) and *decrement* (−−). As used in the sample program, the increment operator statement

```
++i;
```

is equivalent to

```
i = 1 + 1;
```

Similarly,

```
--j
```

is equivalent to

```
j = j - 1 (decrement).
```

Logical operators also can be confusing. The *bitwise AND* operator (&) is different from the *logical AND* operator (&&). For example,

```
i = 0x13 & 0x27;
```

evaluates i = 13h AND 27h as 03h (note the use of 0x for hexadecimal numbers). When used as a logical operator,

```
i = a && b;
```

i is evaluated as TRUE only if both a AND b are true. The same distinctions hold true for the *OR* operators (| and ||).

There are two important loop control statements in C, the *for loop* (shown in the example program) and the *while loop*. In the example, the for loop consists of a for() statement followed by one or more program statements, enclosed in braces. The for() statement consists of three sets of expressions, separated by semicolons: initializations (i = 0, x = 0.0), test condition (i < nval), and execute at end of loop (++i). The initializations set up a loop index variable (i) and any other variables used in the loop (x), where required. The test condition (i < nval) is evaluated at the start of each loop. This usually checks if the index is within bounds. If the test condition is true, the statements within the loop's braces are executed (x += *(data + i);). This is followed by the executable expression (++i), usually used to increment the loop index.

When the test condition is no longer true, as when the loop has been executed the requisite number of times, execution continues with the first statement following the for loop.

The while loop is simpler. It consists of a while() statement, which contains only a test expression, followed by braces enclosing the loop statements. If we rewrite the for loop from the example program as a while loop, we get

```
i = 0;
x = 0.0;
while(i < nval)
     {
     x += *(data + i);
     ++i;
     }
```

The while() statement is a useful way to wait for an event to happen, regardless of how many times to try. If we are waiting for a device to produce data, via a function get_data(), which returns 0 if no data is present, the statement

```
while(get_data() == 0);
```

waits indefinitely until get_data() returns a nonzero value. Of course, in actual practice there should be a way of terminating this wait, in case of error (such as a time-out).

This concludes our brief overview of the C programming language, which is one of the most important general-purpose languages for data acquisition applications. Next, we will examine C++, an offshoot of C.

13.1.4 C++

C++ was developed by Bjarne Stroustrup as an extension of C that is in effect a superset of the original language. The most significant aspect of C++ is that it is an *object-oriented* programming language. Object-oriented programming (OOP) languages rely on three major concepts: encapsulation, inheritance, and polymorphism.

An object in an OOP language has the property of *encapsulation* because it is a self-contained, logical unit, containing both data and code. An object has the ability to hide its operations and data from other parts of the program. It can have both private code (member functions) and data (structures) that are not accessible outside of the object. C++ implements encapsulation through new user-defined variable types, called *classes*.

Inheritance is a process that allows one object or class to obtain the properties of another object, including data structures and member functions. In C++, a new type that is the extension of an existing type can be declared as a subclass, with its unique modifications. Using inheritance, you can create a well-organized hierarchy of classes.

Polymorphism (literally, the ability to assume many forms) allows the same name of a function or class to be used for slightly different but related operations. That is, one name can determine a general course of action, while the actual type of data used selects the specific, detailed operations. In this way, related objects operate in similar ways, just as all automobiles speed up when you press the accelerator pedal. For example, in C++ assume we have averaging functions avg_int() which operates on integers and avg_float() which operates on floating-point numbers. Using polymorphism, we can define a function avg() which operates on either data type. At compile or run time, C++ will determine which function to use when avg() is called, depending on the input data type. Applying polymorphism to functions this way is sometimes called function overloading.

The concept of object-oriented programming was developed to simplify the task of developing large, complex computer programs. It enables a programmer to break up a large problem into smaller, related sections. Then, each subsection of the problem can be translated into an object. An added benefit of object-oriented programming is that it encourages writing reusable code because of encapsulation and inheritance.

You can use a C++ compiler with C code with few or no changes. Of course, the power of the new language is only apparent when you use the unique features of C++. If you are already familiar with C, you have to learn the new features and syntax of C++ along with its object-oriented philosophy. If we rewrote the main() section of our C averaging program in C++ we would see a new way of producing output:

```
void main()        // explicitly show that main() returns no data
{
float avg;                    // variable for average value
avg = calc_avg(a,10);         // calculate average value of array
cout << "DATA AVERAGE = " << avg << endl; // display result
}                             // End of program
```

In this C++ example, cout is the standard output stream and << is the *insertion* operator. When you compare the cout statement here to the previous example using the printf statement, you will notice that there is no explicit formatting used to display the variable avg. Since the compiler knows that avg is a float, cout uses the correct format. This is an example of function overloading, an aspect of polymorphism. Note that endl (end line) is the same as the '\n' character in C. Also note that in C++ comments begin with // and continue to the end of the line (for multiline comments, you can still use the C style /* */ comment delimiters).

If you wanted to explicitly format an output variable in C++ you can use a *manipulator* with cout. For example, if you want to display an integer as hexadecimal instead of decimal:

```
int temp;
temp = 756;
cout << hex << temp <<endl;
```

Here, the manipulator hex forces the output integer value to be displayed as a hexadecimal number.

In a similar fashion, C++ uses a standard input stream to read in values from the keyboard, via cin:

```
int val;
cout << "Enter a value: \n";
cin >> val;
```

In this simple example, the variable val is assigned a value entered when the program runs. Note that the direction of the insertion operator (>>) indicates data flow—in this case from cin to val.

The heart of C++ is defining classes and using them to create objects. Classes contain both private and public data (variables) and functions (subroutines). Keeping all data members (internal variables) private and only allowing member functions to be public (accessible by other parts of the program) is good practice for secure data encapsulation. Here is a simple C++ example of declaring a class and using an object:

```
               //Define the new class:
class Sensor                //start of class definition for Sensor
{
private:                    //the following members are private
        char*  s_location;
        int    s_temp;
        int    s_press;
public:                     //the following members are public
                            //declare function prototypes
        void   SetLocation(char* location);
        void   SetTemperature(int temp);
        void   SetPressure(int press);
        void   DisplayInfo();
};                          //end of class definition
```

```
        //Define the member functions:
void Sensor::SetLocation(char* location)
{                           //start of function definition
   s_location=location;     //load private variable from passed param
}                           //end of function definition

void Sensor::SetTemperature(int temp)
{                           //start of function definition
   s_temp = temp;           //load private variable from passed param
}                           //end of function definition

void Sensor::SetPressure(int press)
{                           //start of function definition
   s_press = press;         //load private variable from passed param
}                           //end of function definition

void Sensor::DisplayInfo()
{                           //start of function definition
                            //display values of private variables:
   cout << "Location = " << s_location << '\n';
   cout << "Temperature = " << s_temperature << '\n';
   cout << "Pressure = " << s_pressure << '\n';
}                           //end of function definition
        //Main program:

void main()
{
   Sensor sens01;           //instantiate a Sensor object (sens01)
        // Assign values to member variables via member functions:
   sens01.SetLocation("Environmental Chamber 001");
                                       // init s_location
   sens01.SetTemperature(100);         // init s_temp
   sens01.SetPressure(800);            // init s_press
        // Now display the private data:
   sens01.DisplayInfo();
}                           // end of main()
```

This program shows some of the basic object-oriented features of C++. First, we define a new class called Sensor. This class contains three private variables: s_location (a string), s_temp (an integer), and s_press (an integer). The only way for another part of the program to access these variables is through the public functions SetLocation(), SetTemperature(), SetPressure(), and DisplayInfo(). The first three functions are used to load the private variables and the last one displays their values. The class definition in C++ is essentially an extension of the structure definition in C (which is only for data). Note that in C++, function prototypes must always be explicitly declared (in C it was not mandatory). A function prototype is simply a declaration of a new function that provides its name, input parameters (in parentheses), and type of return value (or *void*, if there is none).

Once the new class has been defined (along with its member variables) the new functions are defined. Since these are member functions of the class Sensor, the function names start with Sensor::, where :: is the *scope resolution* operator. The scope resolution operator simply shows that the specified function is a member of the specified class.

The functions SetLocation(), SetTemperature(), and SetPressure() simply load the appropriate private variable (s_location, s_temp, s_press) with the value passed to the function (location, temp, press). The function DisplayInfo() outputs the values of these private variables.

The main program first creates an object, sens01, of the class Sensor. Here, the new class (Sensor) is used as a type definition, just as int or float would be. You cannot use any members of a class until you create or *instantiate* an object of that class. Once we have the object (sens01) declared, we can initialize its private variables by calling our Set...() functions with the values we wish to load. Note the syntax used in main() to call our public functions:

```
sens01.SetTemperature(100);
```

Here, we use the *dot* operator (.) to indicate that SetTemperature() is a member function of the object sens01. This dot operator use in C++ is analogous to its use in C when accessing an element of a data structure. We cannot directly access the private variables in our object from main(), but if any were public we could use the same construct. For example, if an integer called s_length was defined as a public in the Sensor class, the following statement in main() would be legal:

```
sens01.s_length = 25; // direct assignment of an object's member data
```

The preceding discussion was just a brief introduction to C++, designed to give you the "flavor" of the language. There are many other aspects of C++ that we have not even touched upon. For example, C++ uses special functions called constructors and destructors that create objects (when instantiated) and destroy them (when they are no longer needed). C++ also supports inline functions, which allow you to fully define a simple function when you normally declare just a function prototype, within a class definition. This is appropriate for functions with only a few lines of code, such as the ones in our C++ program example. Another unique feature of C++ is the *reference* data type, which provides an alias for an existing variable.

Most importantly, C++ supports inheritance and class hierarchies. You can define a *derived class* from an existing *base class*. For example, if we wanted to define a class, based on our existing Sensor class, that added a new variable, s_time:

```
class Sens_Time : public Sensor
{                    // start of derived class definition
private:
       long s_time;
};                   // end of class definition
```

Notice that we are defining the new class, Sens_Time, as derived from the base class Sensor. Sens_Time has all the original characteristics of class Sensor plus the addition of a new, private variable, s_time. We could also change the definition of any member variable or function in the class Sensor by redefining it for Sens_Time.

C++ is quickly becoming the language of choice for many professional programmers. For more detailed information, the reader is urged to read a good introductory C++ text, such as those listed in the bibliography. Next, we will quickly look at a few older high-level programming languages: FORTRAN and Pascal.

13.1.5 FORTRAN

There are many other high-level languages commonly used for programming PCs. FORTRAN is one of the oldest, numerically oriented high-level languages, extensively used for scientific and engineering programming. FORTRAN is an acronym for FORmula TRANslator. It is not a highly structured language (akin to BASIC), where GO TO statements are extensively used for flow control. Unlike BASIC, only lines used in branching statements get numbered.

FORTRAN supports explicit declaration of variables, but also uses implicit variable types. It assumes that an undeclared variable is real (floating-point) unless it begins with a letter between i and n (inclusive), which denotes an integer. For program control it uses an IF statement, a GO TO statement, and the DO loop (similar to the for loop, in C). To illustrate some points, here is our sample program, calculating an average value, written in FORTRAN:

```
C CALCULATE AVERAGE VALUE
          DIMENSION D(10)
          DATA D/1.1,2.3,5.7,6.4,2.9,3.0,2.1,4.0,1.9,8.4/
          SUM=0.0
          I=1
   20     SUM=SUM+D(I)
          I=I+1
          IF(I.LE.10)GO TO 20
          AVG=SUM/10.0
          PRINT,AVG
          STOP
          END
```

Note that only the line addressed by the GO TO command is numbered: 20 SUM=SUM+D(I). In the IF statement, .LE. is a logical operator (less than or equal). All the logical operators in FORTRAN begin and end with a period (.), such as .AND., .OR., .EQ.(equals), and .GT. (greater than).

The program can be simplified by using a DO loop, instead of the IF( ) GO TO structure. A DO loop is similar to a for loop in C. The rewritten program is as follows:

```
C CALCULATE AVERAGE VALUE
         DIMENSION D(10)
         DATA D/1.1,2.3,5.7,6.4,2.9,3.0,2.1,4.0,1.9,8.4/
         SUM=0.0
         DO 20 I=1,10
   20    SUM=SUM+D(I)
         AVG=SUM/10.0
         PRINT,AVG
         STOP
         END
```

Now, as long as *I* is less than or equal to 10, any statements between the DO 20 statement and line 20 (inclusive) are executed. When this condition is no longer true, execution passes to the statement following line 20. Also note that all the variables used in these FORTRAN examples are floating-point, except for the integer *I*.

FORTRAN is a well-established language with a large base of support. However, newer programming languages, such as Pascal, C, and C++, have superseded it in popularity, especially in the world of PCs. It is rarely the language of choice for data acquisition or data analysis applications on PCs, especially if low-level interfacing or graphics are involved. Also, only DOS-based FORTRAN compilers are commonly available for PCs.

13.1.6 Pascal

Pascal is a highly structured, general-purpose, high-level language. It is another example of a computer language designed by a single person, Niklaus Wirth. It was developed as a means of teaching good programming skills and providing clear, readable, unambiguous source code. Pascal succeeded in that goal, as it has often been taught as an introductory programming language to computer science as well as other engineering and science students.

Pascal is a very robust language. It contains all the standard mathematical operators as well as a large number of mathematical functions, such as sqrt(x), ln(x), sin(x). In addition, it contains standard procedures for data I/O and file handling. In these respects, Pascal is a higher level language than C, which must rely on standard library functions for these capabilities.

The structure of Pascal programs is well defined. A Pascal program starts with a program declaration and is followed by declarations for constants and variables. As in C, a variable has to be declared before it can be used. If no subroutines or procedures are present, the body of the program, containing executable statements, follows. Finally, the end of the program is declared. If procedures are present, they precede the body of the main program. They are structured in a way similar to the main program. As with C, variables can be local or global. If a variable is declared in the main program, it is global and accessible to any procedures defined with that program. If a variable is first declared within a procedure, it is local to that procedure (and any procedures declared within it).

Pascal has its own rules for syntax. As in C and C++, the semicolon (;) is used to terminate statements and program sections. Comments in Pascal are enclosed within braces ({ }) and can span more than one line. The last line in a program is the end statement, followed by a period (end.). The last line of a procedure is an end statement, followed by a semicolon (end;).

To illustrate this language, we will look at our example of an averaging program, now written in Pascal. Note that in this version, input is expected from the user (via the read procedure):

```
program      Average(input,output);
                                  {Calculates Average of 10
                                   input values}
const        Nvals = 10;          {number of values to average}
var          Sum, Avg: real;      {variables}
             Counter: integer;

procedure    GetData;             {Reads input value & accumulates}
var          Value: real;         {local variable, for input value}
begin                             {body of subroutine}
             read(Value);         {get data value}
             Sum := Sum + Value   {accumulate Sum}
end;                              {end of procedure GetData}

begin                             {Start of body of main program}
   Sum := 0;                      {initialize accumulator}
   for Counter :=1 to Nvals       {accumulation loop }
             do GetData;          {call procedure}
   Avg := Sum / Nvals;            {calculate average}
   writeln('The average value = ',Avg);
                                  {display result}
end.                              {end of program Average}
```

We see that the main program (Average) declares it uses both input (via read) and output (via writeln) functions. The standard output procedure, writeln, is equivalent to printf in C. The main program calls a procedure, GetData, which reads and accumulates the input values into variable Sum, one at a time. GetData has one local variable, Value, used to temporarily store the input from read(Value). GetData can access Sum, because it is a global variable (defined by the main program, Average). The main data accumulation is done by the for loop, which calls procedure GetData. Also note that := is the *assignment operator* in Pascal. It is used to assign a value to a variable. In the constant declaration for Nvals, an ordinary = is used, since this is just defining the symbol Nvals.

Pascal is rich in control structures, such as the for loop. In the example just given, Counter is initialized to 1 and then incremented with each pass through the for loop, until it equals Nvals. For each pass through this loop, GetData is executed. If multiple statements are to be executed within a for loop (instead of a single procedure call), a more generalized form is

```
for     Counter := StartVal to EndVal
        do begin
                {place executable statements here}
        end;    {last statement executed in for loop}
```

This is very similar to the for loop in C, except here incrementing the index variable is implicit.

Pascal has an if...then...else structure, very similar to C. The argument of the *if* statement is a Boolean expression, evaluated as true or false. If it is true, the statements following *then* are executed. If not, the statements following *else* are executed, as in the following example:

```
if Value = 0
        then begin
                writeln('This is a zero value');
                {more then statements here}
        end     {last then statement}
        else begin
                writeln('This is a nonzero value');
                {more else statements here}
        end;    {last else statement}
```

Pascal also has a *while* loop, functioning the same as it does in C. A Boolean expression is evaluated by the while command. As long as it is true, the statement (or loop) following the do command is executed. For example:

```
while Value >= 0
        do begin
                read(Value);
                writeln('The current value is ',Value);
                {other loop statements here}
        end;    {last statement in while loop}
```

An additional control structure available in Pascal is the *repeat* loop. This can be considered the reverse of a while loop. The statement or loop following the repeat command is continuously executed until its exit condition, in the *until* statement, is true. Rewriting the foregoing example with a repeat...until structure, we get

```
repeat
     read(Value);
     writeln('The current value is ',Value);
     {other loop statements here}
     until Value < 0;
```

Notice that unlike the while loop, the repeat loop is always executed at least once.

This ends our brief overview of Pascal. It is a powerful language, well suited for most programming tasks. In addition, it is well supported by compilers, libraries, and debugging tools for PCs, especially under DOS. Next we will take a brief look at a fairly new programming language, Java.

13.1.7 Java

The Java programming language was developed by Sun Microsystems in the early 1990s. Although it is primarily used for Internet-based applications, Java is a simple, efficient, general-purpose language. Java was originally designed for embedded network applications running on multiple platforms. It is a portable, object-oriented, interpreted language.

Java is extremely portable. The same Java application will run identically on any computer, regardless of hardware features or operating system, as long as it has a Java interpreter. Besides portability, another of Java's key advantages is its set of security features which protect a PC running a Java program not only from problems caused by erroneous code but also from malicious programs (such as viruses). You can safely run a Java applet downloaded from the Internet, because Java's security features prevent these types of applets from accessing a PC's hard drive or network connections. An *applet* is typically a small Java program that is embedded within an HTML page.

Java can be considered both a compiled *and* an interpreted language because its source code is first compiled into a binary byte-code. This byte-code runs on the *Java Virtual Machine* (JVM), which is usually a software-based interpreter. The use of compiled byte-code allows the interpreter (the virtual machine) to be small and efficient (and nearly as fast as the CPU running native, compiled code). In addition, this byte-code gives Java its portability: it will run on any JVM that is correctly implemented, regardless of computer hardware or software configuration. Most Web browsers (such as Microsoft Internet Explorer or Netscape Communicator) contain a JVM to run Java applets.

Compared to C++ (another object-oriented language), Java code runs a little slower (because of the JVM) but it is more portable and has much better security features. The virtual machine provides isolation between an untrusted Java program and the PC running the software. Java's syntax is similar to C++ but the languages are quite different. For example, Java does not permit programmers to implement operator overloading while C++ does. In addition, Java is a *dynamic* language where you can safely modify a program while it is running, whereas C++ does not allow it. This is especially important for network applications that cannot afford any downtime. Also, all basic Java data types are predefined and not platform-dependent, whereas some data types can change with the platform used in C or C++ (such as the int type).

Java programs are more highly structured than C++ equivalents. All functions (or Java *methods*) and executable statements in Java must reside within a class while C++ allows function definitions and lines of code to exist outside of classes (as in C-style programs). Global data and methods cannot reside outside of a class in Java, whereas C++ allows this. These restrictions, though cumbersome at times, help maintain the integrity and security of Java programs and forces them to be totally object-oriented.

Another key feature of Java is that it is an open standard with publicly available source code. Sun Microsystems controls the Java language and its related products but Sun's liberal license policy contributed to the Internet community embracing Java as a standard. You can freely download all the tools you need to develop and run Java applets and applications from Sun's Java Web site (http://java.sun.com).

Here is a simple Java program that averages numbers entered from the keyboard:

```
public class AverageProgram     // start of class definition
{
 public static void main(String[] args)
                                // start of method definition
 {
   int npoints, counter, acc, average;     // declare variables

   System.out.println("Enter the number of points to average: ");
   npoints = ConsoleIn.readInt();  // read npoints
   counter = 0;                    // initialize variables
   acc = 0;
   while (counter <npoints)
    {                              // start of while loop
     System.out.println("Enter value: ");
     acc = acc + ConsoleIn.readInt();    // add in current value
     counter = counter + 1;              // increment counter
    }                                    // end of while loop
   average = acc / npoints;              // calculate average
   System.out.println("Average value = " + average);
                                         // display result
 }                             // end of method definition
}                              // end of class definition
```

In this example, the class AverageProgram (which is the program) contains only one method (function), main(). Notice that much of the syntax is the same as C or C++, including comment delimiters: you can use either C (/* */) or C++ (//) style delimiters in Java. Even the while() statement works as it would in C/C++. Output to the screen is accomplished using System.out.println(), where println() is an invoked method of the standard Java System.out object. Java also has a System.in object, for reading from the keyboard, but it must be processed to be useful. In this example, ConsoleIn is assumed to be a previously defined class (that uses System.in), which contains the method ReadInt() for reading an integer value.

As with the other programming languages we have surveyed, this was just a brief view of Java. For more details, refer to one of the Java texts in the bibliography or visit Sun Microsystems' Java Web site (http://java.sun.com). Next we will discuss writing programs that run under a Microsoft Windows operating system.

13.2 Programming for Microsoft Windows

Programs that run under Microsoft Windows are inherently more complex than DOS programs (or programs for other text-based operating systems). Working within the graphical user interface (GUI) of Windows requires a programmer to create and manipulate a variety of on-screen graphical objects, such as windows, toolbars, icons, and pointers. Even a program with text-only I/O requires the creation of a window of specified size and position on the screen before any messages can be displayed. By contrast, a DOS C program can start with a printf() statement to immediately display text.

Fortunately, the general-purpose software tools now available will take care of the myriad details required for the creation of a Windows application. In some cases, writing a Windows-based application is easier than writing the equivalent program for DOS. In fact, many of these tools can be used by people who have never had any programming experience. The most popular software development tools for Windows are produced by Microsoft. These include Visual Basic and Visual C++.

Other manufacturers produce equivalent software development products. For example, Borland has manufactured many popular DOS programming packages in the past, such as Turbo C/C++ and Turbo Pascal. Borland has migrated its products to the Windows environment. Their current offerings include Borland C++ Builder, which is a full-featured C++ development environment for Windows 95/98/NT. Borland C++ Builder features drag-and-drop visual programming, numerous wizards, sample applications, and a complete C++ tutorial.

In this section, we will look at two commonly used Windows programming environments from Microsoft: Visual Basic and Visual C++. These languages support the event-based nature of Microsoft Windows: program execution is determined by external events (keystrokes, mouse clicks) and not by the structure of the program code.

Visual Basic is excellent at creating GUI screens and controls (buttons, boxes, etc.) for Windows applications without requiring much user code and is a good first language. Visual C++ is a more flexible language, often used for more functional purposes than creating on-screen objects (such as intensive data processing algorithms). Many large Windows applications are created using both languages: Visual Basic for the GUI display elements and Visual C++ for the processing functions. First we will look at Visual Basic.

13.2.1 Visual Basic

Microsoft Visual Basic is a programming language and development environment for Windows applications based on a greatly enhanced version of BASIC. Visual Basic contains an integrated development environment (IDE) and a large variety of tools to develop general-purpose, graphics-based Windows applications. The current version (at the time of this writing) is 6.0, which only supports 32-bit Windows programs (for Windows 95/98/NT and later). For 16-bit Windows applications (under Windows 3.1), Visual Basic version 4.0 or earlier must be used. Visual Basic is available in three versions: the Learning Edition (for starting with the language), the Professional Edition (which adds additional tools such as ActiveX controls, database tools, and Internet tools), and the Enterprise Edition (for use on company-wide networks with distributed application development tools and network server tools).

Visual Basic is quite different from DOS-based BASIC, even though it still uses many of the same commands. Visual Basic is an interpreted language, but current versions allow you to create compiled programs (.EXE files) to run stand-alone. Line numbers are not used. Much of the software development process involves drawing objects within forms and using mouse-based drag-and-drop techniques to choose predefined graphical elements. You create the user interface by selecting control elements such as text boxes and command buttons. You typically write a modest amount of text-based code that is associated with each screen object.

The most significant difference between Visual Basic and earlier versions of BASIC is that Visual Basic uses an *event-driven* model. Traditional programming languages followed a *procedural* model, where the code determines which statements are executed and in what order. Program execution starts with the first line of code and follows a predetermined path. In an event-driven program, code is executed in response to events and does not follow a predefined path. These events can be generated by the PC operator (via the keyboard or mouse) or by the operating system or other applications. In essence, these events act as interrupts and they determine the sequence of code execution.

Another important aspect of Visual Basic is that it is a hierarchical, object-oriented language (akin to C++). It uses class modules to define objects containing both data and code, both for objects that appear on the screen (in the user interface) and for those that remain hidden. Data and code can be declared public or private.

Other features of Visual Basic are its ability to work with standard databases and its Internet access features. Also, Visual Basic supports Microsoft's ActiveX standard. ActiveX components are language-independent objects that interact with other Windows applications. For example, an ActiveX component can be used to read data from an Excel spreadsheet into a custom program. In Visual Basic, one use of ActiveX is to create custom controls. A *control* in Visual Basic is simply a graphic window that has program code associated with it.

Since ActiveX is an open standard, many manufactures create their own ActiveX custom controls. This can be very useful for data acquisition, since these ActiveX components can include objects for processing or displaying data. For example, Keithley's DriverLINX software, available for many of its data acquisition cards, includes ActiveX controls for data acquisition functions. This allows you to easily write a Visual Basic (or Visual C++) program to access their data acquisition cards, without needing to know any details about the hardware's low-level behavior.

Most Visual Basic programs are based on forms and consist of a mix of code and graphic elements. However, you can also write code-only Visual Basic programs, via code modules. Here is the averaging program we wrote in Java, ported to Visual Basic:

```
Sub Main()
  Dim npoints, counter, acc, average As Integer
  npoints = Val(InputBox("Enter number of points to average"))
  counter = 0
  acc = 0
  Do
          acc = acc + Val(InputBox("Enter value: "))
          counter = counter + 1
  Loop While counter <> npoints
  average = acc / npoints
  MsgBox "Average = " + Str$(average)
End Sub
```

As in C/C++, a routine named main() will begin program execution. The Dim (dimension) statement declares the integer variables. The InputBox()

function is used for simple user input (initially, to get the value of npoints). It creates a small window with a text prompt and a text-input area. Visual Basic is very rich in string manipulation functions. Since InputBox() returns a string, we have to convert it to an integer, via the Val() function. The program's Do Loop reads in values to average and adds them to the accumulator variable (acc). The calculated average is displayed using the MsgBox function. Note that we have to convert the integer value of average to a string, via the Str$() function.

Of course, typical programs create a more complex GUI environment using Visual Basic's extensive graphics features, sometimes with very little user code added. This concludes our brief introduction to Visual Basic. Please refer to the bibliography for further information. Next, we will look at Microsoft Visual C++.

13.2.2 Visual C++

Microsoft Visual C++ is another programming language and development environment for Microsoft Windows applications. It is a full implementation of C++ but designed to simplify the details of producing a Windows application, much like Visual Basic. Besides all the standard features of the C++ language (which we discussed in Section 13.1.4), Visual C++ contains a plethora of tools for developing Microsoft Windows applications, many in the form of wizards. Visual C++ also uses the Microsoft Foundation Class (MFC) library of C++ classes and member functions, used for Windows development. As with Visual Basic, Visual C++ supports the event-driven model of Microsoft Windows programs. The current version of Visual C++ supports only 32-bit applications, for Windows 95/98/NT and later.

The easiest way to create a program under Visual C++ is to use an App Wizard that builds a bare-bones application framework. Most of your work is simply adding code to this framework to achieve the desired result. If you use the MFC AppWizard, you can choose various options for the framework (such as single versus multiple document windows or whether to include a status bar) and create a standard Windows application screen, complete with toolbars and menus. Compared to Visual Basic, Visual C++ is not a drag-and-drop, graphics-oriented environment. Visual C++ is more of a code-oriented environment, but one highly tuned to the requirements of Windows.

If you want to create a simple text-based C++ program that does not require any graphics features (such as simple data processing applications), you can start a new Visual C++ project as a Win32 Console Application. You can selêct a simple application or an application that supports MFC (to use Windows MFC classes and functions). A simple application creates the necessary header files and gives you a single C++ text file with a bare-bones main() to add your code to.

Here is a Visual C++ version of the averaging program we previously wrote for Java, written as a simple Win32 console application:

```
// average.cpp : Defines the entry point for the console
// application.
//

#include "stdafx.h"
#include <iostream.h>


int main(int argc, char* argv[])
{
	int	npoints, counter, acc, in, average;

	cout << "Enter number of points to average: \n";
	cin >> npoints;

	counter = 0;
	acc = 0;
	while(counter < npoints)
	{
		cout << "Enter value: \n";
		cin >> in;
		acc += in;
		++counter;
	}
	average = acc / npoints;
	cout << "Average = " << average << '\n';

	return 0;
}
```

Visual C++ created the comment line (note that in Visual C++ source code files have a *CPP* suffix) and the *#include "stdafx.h"* statement (for the header file it created). It also created a minimal main() program, with simply a *return 0* statement. All the other code was added to the file, along with the *#include <iostream.h>* (to use the cout and cin streams). The program can be run within the Visual C++ environment or outside of it, once it is correctly compiled and linked. When the program is run, it creates a text window for keyboard input and display output. When the program is complete, "Press any key to continue" is displayed. After a key is struck, the window disappears.

One advantage of working in this simple console application environment is that if you do not want to learn Windows MFC functions, you can use generic C++ commands (as in DOS or UNIX). You still have the advantages of working in a 32-bit environment, including the ability to easily work with large amounts of data: for example, an array containing 1 million long integers (32 bits), which is 4 Mbytes of memory. Of course, if you want to take full advantage of the features in Visual C++, you should use the MFC AppWizard to create a graphics-based application.

One of the key features of Visual C++ is the MFC library. The MFC library calls functions in the Windows application programming interface (API), to create standard Windows screen objects, such as dialog boxes, controls, and windows. The MFC library is platform independent (it can even be used with an Apple Macintosh computer) and consists of more than 100 classes. The Windows API is not object-oriented and does not readily support code reuse or a hierarchical program structure. The MFC library is well organized and is usually easier to use. However, you can always make direct calls to Windows API functions from Visual C++.

This concludes our brief look at Visual C++. For more information, there are many excellent reference and tutorial books available on Microsoft Visual C++, along with a large amount of material on the Internet. We will finish this chapter by reviewing a few key points to keep in mind when writing your own software.

13.3 Considerations for Writing Computer Programs

There are many possible approaches to writing a computer program to solve a particular data acquisition or analysis problem. Regardless of the language used, the same steps are followed in developing a usable program. A PC is a wonderful platform to use for software development, because of the abundance of commercially available support tools.

A necessary starting point is stating the problem and your proposed solution in general terms, written in plain English. This may be as simple as, "Acquire 1024 data points, run an FFT, and report the average signal amplitude in the frequency band of 100 to 300 Hz." Next, draw a flowchart, including more of the required details (such as initializing hardware for data sampling rate and analog input range). The flowchart gives you an overview of what your program will do. It also helps you locate potential errors in logic, before they get lost in the details of the chosen programming language. Figure 13-1 contains the flowchart for a simple program acquiring 1024 data points. It flows continuously from the Start point to the End point, except for the data acquisition loop. Here, the decision box checks whether the counter has exceeded 1024. If so, it ends the program. If not, it loops back and acquires another sample.

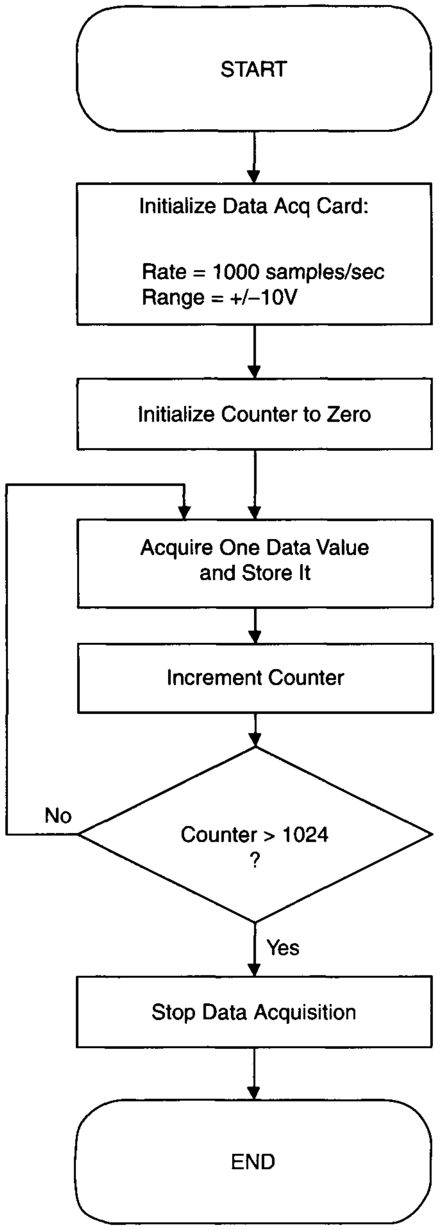


Figure 13-1 Data acquisition program flowchart.

Based on the kind of problem you are facing, you need to choose the type of programming language to use: text-based or graphics-based. In the preceding example, if we only need a table of numbers as an output from the FFT, then a text-based program is adequate and we can use any of the languages discussed in this chapter. On the other hand, if we needed a plot of the output data, a graphics-based language (such as Microsoft Visual Basic, Visual C++, or MATLAB) would be preferable. Once you know the type of language needed, you can choose the specific one based on your own experience and preferences.

The next step is to write the actual program and debug the source code until you can compile and link it without errors. When most compilers find a compilation error, they will point out where it is in the program (or where the compiler thinks it is), along with a clue as to the type of error. Some compilers will even highlight the error line in the source code. An error from a linker usually means a function or a global variable was not defined or a library file cannot be located. This can result from forgetting to include a particular name in the list of object modules to link, or even from a spelling error, causing a call to a nonexistent function.

Once you have an executable program, you can test it functionally with a debugging program. A source-level debugger or an integrated development environment (such as Visual C++) is preferable when the program is written in a high-level language. This allows you to check variable values, follow the route of statement execution, and even change values to see what will happen. Most major compiler manufacturers provide a debugging environment for their languages. In addition, there are third-party debugging products that support compilers from several major manufacturers. As an additional aid, you can always add debugging statements to your program while writing it. These would typically display various intermediate variables or parameters returned to calling functions. You can even have these statements conditionally execute, depending on the value of a global debug variable. These are especially useful when working in an embedded PC environment without many software resources. Here is a simple example in C:

```
int     debug = 1;   /* Debug Statements enabled */
.
.
.
junk()               /* Illustrative function */
{
int     i,j;
.
.
.
if(debug)
        printf("\nDebug values: i=%d, j=%d\n",i,j);
}
```

If the variable debug is set to 0, the printf() statement in the subroutine junk() will not be executed.

A critical aspect of writing software, which is commonly overlooked, is documentation. This involves adding comments to your program as you initially write it, debug it, and update it. Putting a comment on nearly every line of source code is very useful (except for extremely obvious statements, such as display outputs). It is also important to put a detailed explanation of a program at the beginning of the file. Each subroutine should be documented at its beginning, including what it does, what its input and output parameters are, and what routines call it. Always document your programs well enough that if you have to look at them again, several years later, you can quickly figure out exactly what you did. In the case of documentation, too much is never enough (the same point also holds true for hardware designs).

This concludes our quick survey of computer programming languages. This discussion touched on many of the major languages used with PCs. In the next (and final) chapter we will look at some real-world examples of data acquisition applications.

CHAPTER

14

PC-Based Data Acquisition Applications

In this final chapter, we will look at a few examples of how PC-based data acquisition systems are used in "real world" situations. These applications fall into three major categories, which tend to overlap: laboratory/industrial data collection, laboratory/industrial control, and embedded data acquisition and control.

In this book, we have focused primarily on data acquisition and control equipment for use in a laboratory or industrial setting. These are stand-alone systems using a PC, typically containing appropriate data acquisition cards and running software for data collection, analysis, and control. Such a system may be used for performing a laboratory experiment, obtaining automated measurements in an industrial setting, or controlling an industrial process.

Embedded applications are another way of utilizing data acquisition and control systems based on PCs. In this case, an original equipment manufacturer (OEM) uses a PC-based data acquisition system as part of a larger piece of equipment it produces. The PC and its related hardware and software are embedded in that equipment. Usually, the software running on the PC is dedicated to the task the equipment was designed for. When this software is designed to ensure that the PC always starts up in this dedicated application, it is considered a *turnkey system*. An example of this would be an automated test equipment (ATE) system, dedicated to testing particular devices (such as printed circuit boards). It is no longer usable as a general-purpose PC, unless the system software allows it.

We will now examine a few examples of data acquisition applications, starting with laboratory and industrial measurement systems.

14.1 Ultrasonic Measurement System

Ultrasonic waves are employed for many different types of measurements, including displacement, determination of material properties, and Doppler shift velocity. Many of these ultrasonic applications are based on time-delay measurements. Since the speed of sound is five to six orders of magnitude slower than the speed of light (depending on the medium) the time measurements required to determine typical distances are more easily attained using ultrasonics. In air, at room temperature, ultrasonic waves travel at approximately 340 meters/sec. The measured time delay, t, is

$$t = d/v$$

where d is the distance traveled and v is the wave velocity. For a distance of 1 meter, the time delay using an ultrasonic beam would be 2.9 msec. Using a light beam, the corresponding time delay would be 3.3 nsec, which is much more difficult to measure.

Ultrasonic ranging systems are commonly used to measure macroscopic distances, on the order of inches to hundreds of feet. This technique is often implemented using a single ultrasonic transducer as both a transmitter and receiver. An ultrasonic pulse is transmitted by the transducer, reflected off a target at the distance to be measured, and then detected by the same transducer. The measured time delay between the transmitted and received pulses is equal to twice the transducer–target distance divided by the ultrasonic velocity. Many low-cost ultrasonic transducers for wave propagation in air are available. One variety, an electrostatic transducer, available from Polaroid Corp. (Cambridge, MA), is a popular choice for this type of application. Figure 14-1 shows a simplified implementation of this ranging system.

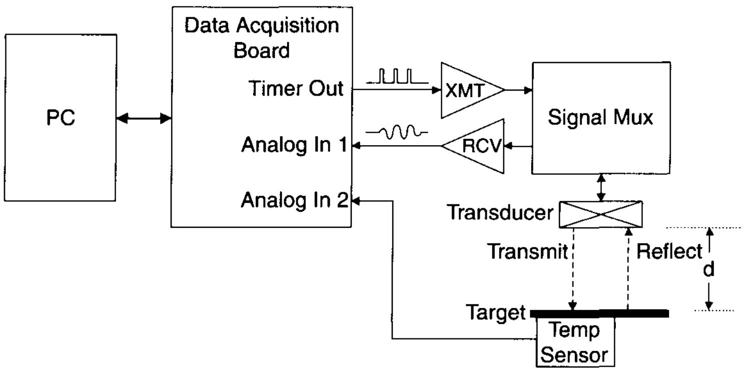


Figure 14-1 Ultrasonic ranging system.

The sequence of events for a single measurement cycle is as follows: We start with a trigger pulse from a timing reference, which initiates a high-voltage transmit pulse that is sent to the transducer. The reflected (delayed) ultrasonic pulse is received by the transducer and goes to a receiving circuit, containing an amplifier and filter. An analog multiplexer is used to isolate the high-voltage transmit pulse from the low-voltage receive pulse (usually employing diodes). An ADC (as in a data acquisition card) starts sampling data, once the trigger pulse occurs. The sample number containing the start of the reflection pulse multiplied by the time between samples is the time delay corresponding to the round-trip distance traveled by the ultrasonic waves.

The wavelength, λ, of any wave is:

$$\lambda = v/f$$

where v is the wave's velocity and f is its frequency. If the ultrasonic transducer's resonant frequency is 50 kHz (as with the Polaroid transducers) the wavelength is 6.8 mm (approximately 1/4 inch). A good estimate of the displacement resolution using this technique is one-half wavelength or 3.4 mm (approximately 1/8 inch). If we needed finer resolution, we would need a higher frequency transducer (such as 170 kHz for 1 mm resolution).

If we want to implement this experiment using a PC-based data acquisition system, we must first determine our measurement requirements. We will assume that a distance resolution of 1/8 inch is adequate and the maximum distance measured will be 100 feet. If we use a 50-kHz transducer, our ADC sampling rate must be at least 100,000 samples/sec. To ensure reasonable data fidelity, a higher rate is preferable, such as 250,000 samples/sec. The maximum distance of 100 feet corresponds to 30.48 meters. The maximum round-trip time delay is

$$\frac{30.48 \text{ m} \times 2}{340 \text{ m/sec}} = 179 \text{ msec}$$

This corresponds to approximately 45,000 samples (at 250,000 samples/sec). The 179-msec period also limits the maximum transmit pulse repetition rate to the inverse of that period, or 5.6 Hz in this case. This is the maximum number of transmit/receive cycles we can measure each second, without having the reflection from cycle $n - 1$ appear after the start of cycle n.

Since the data rate required for this experiment is very fast, a reasonably high-speed data acquisition card is called for. If a 12-bit ADC is used, 250,000 samples/sec corresponds to a sustained data transfer rate of 500,000 bytes/sec, which is faster than many ISA PCs with DMA capabilities. If an ISA card is used, this data should be stored in a data acquisition board with local memory. For our purposes, this memory must have a capacity of at least 90,000 bytes (assuming 2 bytes/sample). If a PCI-based PC is used, the required data rate is well below a PC's typical bus transfer rate and no on-board memory is required for the data acquisition card. The data acquisition card selected must have a digital output to act as the trigger line for the external transmitter as well as a data acquisition start signal. We also want this board to have a counter/timer that can initiate an ADC conversion every 4 μsec (for 250,000 samples/sec), as well as control multiple cycle timing. We need at least a second analog input channel to periodically measure the air temperature for velocity calibration.

There is another approach to the ADC speed problem, using a slower and less expensive data acquisition board (especially with an older ISA PC). Since the event we wish to measure can be repetitive, instead of measuring the entire waveform in one cycle, we can acquire data over several cycles. All we need is a data acquisition board with a sample-and-hold amplifier in front of the ADC. If the amplifier has a sample window less than or equal to our sample period of 4 μsec, the conversion rate of the ADC can be much slower. In the worst case, we acquire one sample for each waveform cycle. The overall data acquisition time will depend on the repetition rate of the transmit pulse (the overall cycle time).

To implement this, for every transmit pulse cycle, we delay the sample time of the data acquisition board by another 4 μsec. Our transmit pulse repetition rate is limited by the maximum delay time between the transmit and receive pulses of 179 msec, in this example. That means we can only generate five cycles/sec. Since we need to acquire about 45,000 samples, this will take 9000 sec, or 2.5 hours! A better way is to acquire multiple samples from each pulse cycle. Even if the ADC can acquire only 10,000 samples/sec, each transmit/receive cycle will produce 1790 samples now. This way, only about 5 sec (25 cycles) is needed to acquire 45,000 samples of data.

This scheme (a version of equivalent time sampling) is shown with the waveform in Figure 14-2. It acquires samples spaced 100 μsec apart, from

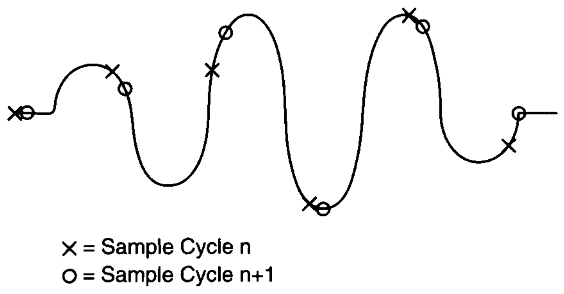


Figure 14-2 Using multiple cycles to acquire a repetitive waveform.

a single transmit/receive cycle, as represented by the "X" symbols. During the next cycle, the acquisition starts 4 μsec later, as shown by the "O" symbols. This process continues until the entire waveform is filled in. Since the window of the sample and hold amplifier is no more than 4 μsec, it is equivalent to acquiring data at 250,000 samples/sec, except it takes more time to acquire all the data, and it is done over several transmit/receive cycles. The separate data acquisition cycles must be interleaved by the computer to produce the completed waveform. As long as the timing jitter (inaccuracy) of the system clocks is well under 4 μsec, this approach will work well.

If the final desired data will reside in an array D[45,000], using the multiple cycle scheme, we will assume the first transmit/receive cycle (cycle 0) has no offset time, the next cycle (cycle 1) starts 4 μsec after the trigger pulse, and so on, until cycle 24 starts 96 μsec after the trigger pulse. Note that the data at time = 100 μsec is the second point from cycle 1, since this data is all 100 μsec apart. If each of these cycles produces a data array Cn[1790], the reconstructed waveform data D[m] (where m = 0 to 44999), will be

```
D[0] = C0[0], D[1] = C1[0] ,...,                          D[24] = C24[0]
D[25] = C0[1], D[26] = C1[1] ,...,                        D[49] = C24[1]
.                                                              .
.                                                              .
.                                                              .
D[44975] = C0[1789], D[44976] = C1[1789] ,..., D[44999] = C24[1789]
```

Once the waveform data array is acquired and reconstructed, it can be analyzed. Figure 14-3 shows a typical waveform from an ultrasonic ranging system. Since the transducer is multiplexed for transmit and receive signals, ringing from the transmit pulse appears in the acquired waveform. Since this transmit signal is fairly constant, we can ignore the data for the first millisecond or so. This *lockout window*, corresponding to about 1/2 foot, limits the minimum distance that can be measured. Any reflected pulse arriving within this window will be obscured by the transmit signal. If we used a separate transmit and receive transducer, this would not be a problem.

The lockout window can be implemented either in the analysis software, or by initially starting data acquisition after the nominal 1-msec window period (and adding that time offset to the collected data). Because only about 250 data samples would be saved this way, the software approach is better: it allows for adjustment of this window after data has been acquired.

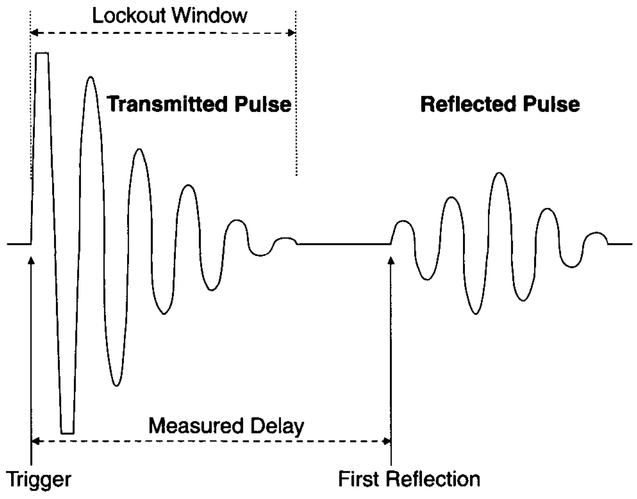


Figure 14-3 Typical ultrasonic ranging system waveforms.

One important point to keep in mind when attempting accurate ultrasonic measurements is that the velocity of ultrasonic waves is a function of temperature. That is why the ranging system in Figure 14-1 uses an additional analog input channel to measure the air temperature. This temperature measurement does not have to be done very often—once per acquired waveform is more than enough. The relationship between the speed of sound in air v (in m/sec) and the temperature of the air T (in degrees Kelvin) is

$$v = 331.4 \times \sqrt{(T/273)} \text{ m/sec}$$

Other environmental factors, such as relative humidity and barometric pressure, have a much smaller effect on ultrasonic velocity and can usually be ignored. Relative humidity does have a large influence on the attenuation of ultrasonic waves.

Depending upon how the acquired data looks, analysis can be fairly simple or very involved. If the reflected pulse's signal-to-noise ratio is high and its first peak is readily observable, the analysis simply consists of finding the location of that peak (minus 1/4 of the wave period), which corresponds to the round-trip time delay of the ultrasonic pulse. This could be a peak detector algorithm, checking data values with an amplitude greater than a specified noise threshold.

Of course, in the real world, things are rarely this easy. One complication is the attenuation of the ultrasonic waves. As the target distance increases, the amplitude of the reflected pulse decreases (as does its signal-to-noise ratio). It becomes more difficult to discern the first peak of the reflected pulse. An added complication would be an imperfect, rough target surface, causing scattering of the ultrasonic pulse resulting in a "fuzzy" echo. This is because different (spatial) portions of the reflected ultrasonic pulse arrive back at the transducer at slightly different times, causing the resulting echo to be spread out in the acquired waveform.

Various DSP techniques can be used to solve these problems. Implementing digital filtering in software can help eliminate noise and enhance the reflected pulse. Another approach is to calculate the FFT of the waveform and measure the slope of the resulting phase curve, in the frequency domain. This phase slope is proportional to the absolute delay of the reflected pulse, with a time offset of half its width. This offset can be determined by shifting the original waveform so that the reflected pulse starts at time = 0, and then calculating its FFT. Subtract this phase slope from the phase slope of the unshifted waveform's FFT, producing the corrected time-delay phase slope. This analysis can be done with many different commercial software packages, requiring little or no programming. Another approach is to use cross-correlation between the received waveform and an ideal waveform at time = 0. As the ideal waveform is delayed by increments of the acquisition time step, the cross-correlation value increases until it reaches a peak at the time step corresponding to the delay of the received signal.

This same experimental setup can also be used to directly measure the thickness of a material sample, with a resolution determined by the frequency of the ultrasonic transducer, as long as the speed of sound through that material is known. Whenever an ultrasonic beam passes through an interface between different media, such as air and a solid, there is a change in acoustic impedance and some of the beam is reflected at the interface.

As shown in Figure 14-4a, if a transmitted ultrasonic pulse hits a material of thickness *d*, some energy is reflected from its front surface (if its acoustic impedance differs from the surrounding medium), resulting in the first echo. The rest of the beam passes into the material. Some of that beam is reflected from the back surface (the rest continues out the back). Part of the beam reflected from the back surface now passes through the front surface, back to the transducer, resulting in the second echo. The remainder of the beam reflects back into the material again, eventually resulting in the third echo. This process continues with multiple reflections.

Each successive echo is lower in amplitude, since energy is lost at each reflection (even if the material has negligible attenuation). The time delay

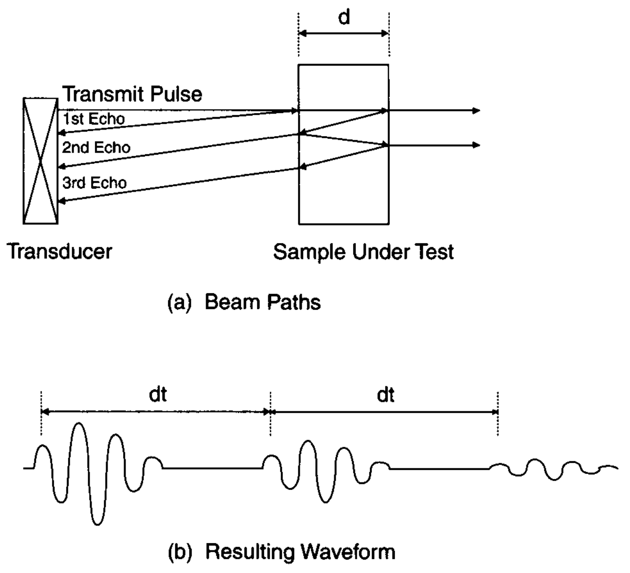


Figure 14-4 Using multiple ultrasonic reflections for thickness measurements.

between successive echoes, as shown in Figure 14-4b, is

$$dt = \frac{2d}{v}$$

since each echo is separated in time by a round trip through the material thickness. The measurement of *dt* can be done fairly accurately using autocorrelation. Since each echo pulse is basically an attenuated version of the previous pulse, calculating the autocorrelation of this waveform (the cross correlation with itself) will produce peaks at

$$t = 0,\ dt,\ 2\ dt, \ldots,\ n\ dt$$

depending on how many echoes appear in the waveform. Each successive autocorrelation peak will be lower in amplitude than the previous one; however, the peak located at time *dt* is a good measure of $2d/v$.

If the correlation peaks are too broad to get an accurate measurement of *dt*, that curve can be differentiated. The zero crossing point of the first derivative of the autocorrelation curve is an accurate location for the peak of *dt*.

We should note that if the sample thickness is known, the velocity of sound through the material can be calculated from this same multiple reflection measurement. In addition, various material parameters (such as elastic modulus) can be determined from this data.

This concludes our example of an ultrasonic ranging system. Next, we will turn our attention to another example, implementing an electrocardiogram (ECG) measurement system using PC-based data acquisition products.

14.2 Electrocardiogram Measurement System

The acquisition and analysis of human electrocardiograms (ECGs) is of great interest to many medical researchers. The ECG is a graph of voltage variations produced by the heart muscle and plotted against time. Automated analysis of ECG data is an active area of ongoing research, as a means of improving diagnosis and prediction of heart disease. The requirements for implementing an ECG data acquisition system using a PC platform are very different from the previous example of an ultrasonic ranging system.

ECG data consists of very low frequency components. Most of the spectral content of an ECG fits within a bandwidth of around 10 Hz, with very little energy present above 100 Hz. A typical diagnostic ECG recorder has a bandwidth of 0.05–100 Hz. Hence, very low data acquisition rates are used, typically 250 samples/sec, to ensure good fidelity. The amplitudes of ECG voltages are very low, in the range of tens of microvolts up to several millivolts. The transducers used to detect these voltages are electrodes placed on the surface of a person's arms, legs, and chest. They connect to isolation circuitry, to protect the patient from any current that may be produced by the ECG recording equipment. Then the ECG signals must pass through differential amplifiers, required to provide high gain and good common-mode noise rejection. If ECG data will be digitized, it is commonly connected to an antialiasing filter with a 100-Hz bandwidth.

Even though the acquisition rates for ECG data are relatively low, the volume of data recorded for research purposes tends to be extremely large. It is common for medical research projects to acquire several hours of ECG data from each subject, sometimes for as long as 24 hours. This data usually consists of two channels of 12-bit or 16-bit readings at a conversion rate of 250 samples/sec. If we assume that no data compression is applied, we require 4 bytes of storage for each sample interval (for two channels) for a data storage rate of 1000 bytes/sec. If we record 1 hour of data from a patient, it will occupy 3,600,000 bytes. Acquiring data from many subjects or recording several hours from each one will obviously use up a large amount of memory storage very quickly. It is no wonder that data compression techniques are routinely applied to ECG storage problems.

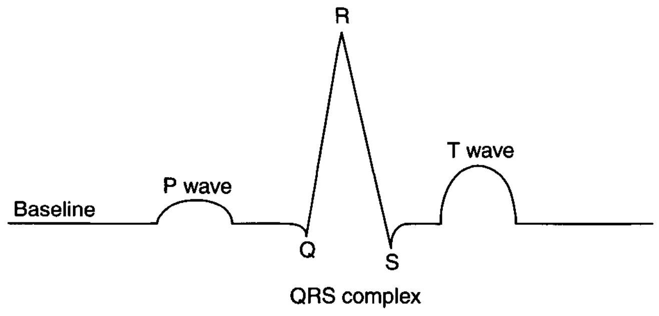


Figure 14-5 An idealized normal ECG beat cycle.

Figure 14-5 shows one beat of an idealized, normal ECG waveform. Various components of an ECG cycle (one beat) have specific names. A beat starts with the P wave, which represents the original electrical impulse in the heart, beginning the cycle. It usually has a small amplitude. The QRS complex, consisting of the Q, R, and S waves, is usually the largest amplitude component in an ECG. The Q wave, itself, may have a very small amplitude (sometimes it is unmeasurable), while the R and S waves can be quite large. The cycle ends with the T wave, representing the electrical recovery phase of the heart, preparing it for the next beat.

Clinically significant information is obtained from an ECG by measuring several parameters, such as the relative amplitude, width, and time duration of these component waves, as well as the time between the components. In addition, the time between beats is important as a measure of instantaneous heart rate. Occasional, abnormal beats are also sought out as indicators of potential problems. This requires a means of categorizing the data on a beat-by-beat basis, as either normal or abnormal.

Figure 14-6 shows a simple block diagram of a PC-based ECG recording system. The data acquisition board only needs to provide a throughput of 500 samples/sec, assuming two channels of data digitized at a rate of 250 samples/sec. A 12-bit ADC will provide adequate resolution. If much higher resolution is required, a sigma-delta ADC may be used. For this application, the analog front end is very critical. Electrical isolation must be provided between the patient electrodes and the data acquisition system. Any ground current flowing from the measurement system to the patient could be a serious health hazard, causing fibrillation (from a shock directly to the heart). Therefore, isolation amplifiers are used. These amplifiers need differential inputs.

The amount of gain provided by the isolation amplifiers will determine the analog input range required for the data acquisition card. If the isolation

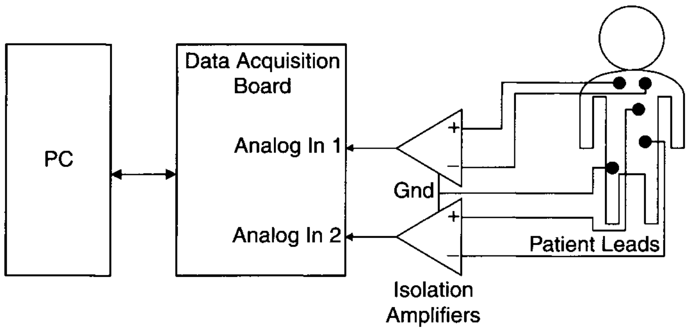


Figure 14-6 PC-based ECG recording system.

amps serve strictly as buffers, then a high-gain, differential analog input would be needed. These isolation amps should also have differential outputs, so any common-mode noise on the wires connecting them to the data acquisition board's analog inputs will be rejected. Of course, the data acquisition board's analog inputs must be differential. In this case, a 12-bit ADC board with a nominal input range of ±5 V and variable gain, up to 500×, would be a good choice. At the maximum gain (500×), the analog input range is ±10 mV, with a resolution of approximately 5 μV. Most ECG waveforms will fit within this range.

A better arrangement would be to provide most of the analog gain using the isolation amplifier, in a separate electronic module. This would minimize the effects of noise pickup in the cable connected to the data acquisition board, as well as noise within the PC itself. In general, it is always a good idea to implement high analog gain outside of a PC, whenever possible. This would allow the use of a simpler, low-gain analog input board. Differential inputs would still be preferable, but are no longer mandatory. Another approach would be to use a USB data acquisition module located close to the patient. As long as the USB module has adequate gain, the data can be digitized outside of a PC.

In any case, a relatively slow ADC is adequate, at moderate (12-bit) resolution. DMA capabilities are not required, since the maximum data transfer rate would be only 1000 bytes/sec (since the overall acquisition rate is 500 samples/sec). The data acquisition card or module should have a counter/timer to produce the data conversion clock. An analog output (DAC) would only be necessary if stored and analyzed data will have to be produced in analog form, at some later time. An example would be to simulate a real-time ECG signal for testing another piece of diagnostic equipment.

Almost any commercial data acquisition software could be used to store acquired data at these low rates. Even though an older PC could be used to acquire this slow data, it must have a large hard disk drive to accommodate the massive data files produced by the relatively long experimental runs. As we saw previously, 1 hour of data requires approximately 3.6 Mbytes of storage and 24 hours of data needs about 86 Mbytes. Even a 1 Gbyte drive can only store about 11 of these 24-hour tests.

One possible choice is to use a tape drive, either to back up data from a large hard drive, or to directly store data as it is acquired. Special software is required to deal with a tape drive, and most data analysis packages will only work with data on disk files. Therefore, using a tape drive to back up conventional disk data files is a simpler approach. Otherwise, you may have to write a lot of your own software for storing and retrieving data on tape. A better alternative to tape drives is to store acquired data on a CD-R or CD-RW disk, which can hold up to 700 Mbytes. The advantages of CD over tape storage are lower cost media, random-access read-back capability, and the data portability (if CD-R is used, you can read the data files on nearly any PC containing a CD-ROM drive).

Another way of dealing with this problem of how to store large amounts of information is to use data compression. Much research has been done on using different data compression techniques on ECG data. Unprocessed ECGs contain a large amount of redundant information. A large fraction of the data is simply the constant baseline, between consecutive beats. Linear predictors can provide reasonably large compression ratios, without excessive distortion, if data acceptance windows are carefully selected.

Another aspect of the redundant nature of ECG data is that most beats from the same patient look very similar, often nearly identical. Only the occasional, abnormal beat appears significantly different. One way of exploiting this characteristic is to apply statistical methods to the data compression problem. Since most ECG data tends to have a lot of straight lines and smooth amplitude variations, it is well suited to delta encoding. Only the amplitude difference between adjacent points is stored, as a small number. If these delta values are calculated from a representative data sample, for a particular subject, they can be statistically analyzed. Then Huffman codes, only a few bits long, could be applied to the most probable delta values.

If most of the original 12-bit data can be represented as 4-bit Huffman codes for delta values, the overall compression ratio would be about 3:1. Most data from one patient is likely to follow the same distribution of delta values and have a similar compression ratio. As we saw previously, this is delta Huffman encoding. Even though it does not produce very high compression ratios, the restored data is completely identical to the original waveform, with zero distortion.

An enhancement to delta Huffman encoding is to identify all baseline data points that fall within a window of constant amplitude (such as noise variations). These points can be replaced by their average amplitude and the length of this line, using a special escape code in the delta Huffman data stream. This addition could increase the compression ratio by another factor of 2 (to around 6:1) while resulting in a small loss of fidelity. Depending on the window size used, an RMS distortion of less than 1% is easily attainable at these compression levels.

This enhancement is effectively implementing a zero-order predictor (ZOP), along with delta Huffman encoding. The algorithm used should carefully decide when to use the ZOP instead of delta Huffman codes, for maximum bit savings. Since the delta values we would replace in this case would be zero, or close to it, they probably require only 2 or 3 bits in their Huffman code. For now, we will assume they use 3 bits per point. If the escape code is 8 bits, the average amplitude is 12 bits and the line length is 8 bits (allowing for a line representation over 1 sec long, at 250 samples/sec), it will take 28 bits to represent this straight line. Therefore, this approach saves storage space if the line is more than 9 points (36 msec) long, corresponding to $9 \times 3 = 27$ bits.

One of these data compression methods could be applied to previously acquired data already stored in disk files. Several commercial software packages are available that provide data compression for all types of files found on a PC (such as PKZIP and WINZIP). Since these products are designed to work with any file type, they produce no data distortion (since most files cannot tolerate any change in their contents, such as programs or documents). As a result, they may produce fairly low compression ratios, typically 2:1. However, with high-redundancy files, such as bit-mapped graphics images, the lossless compression ratio can be 10:1 or higher.

A better approach is to use software specifically designed to compress ECG files or to implement data compression in real time, as the data is being acquired. Since the data transfer rate for this example is relatively low, it is possible to retrieve readings from the data acquisition card as a background task, using hardware interrupts. This would allow the PC to use its spare processing time on the foreground task of data compression. Raw (unprocessed) data would be stored temporarily in RAM, until it is compressed and written to a disk file.

Besides data compression, other analysis techniques are applied to digitized ECG data, usually for diagnostic purposes. This analysis typically involves measuring amplitude, time, and shape parameters of various portions of each beat, to place it in a diagnostic category. For example, a beat that occurred much earlier than expected, based on several previous beats, would be classified as premature and might have medical significance. Other analyses may employ FFTs or other transforms, as well as correlation techniques.

Since this analysis is very specific to ECG data, some custom programming would probably be required. An entire program could be written in a general-purpose language, such as Pascal, C, or Visual C++ (especially if graphics displays are required). Alternatively, a commercial data analysis package could be employed to test out various algorithms. Even here, some programming may be required to implement the desired algorithm, such as with MATLAB, or a similar product.

This concludes our look at using a PC for implementing a system for acquiring and analyzing ECG data. Next, we will look at examples of employing embedded PCs in commercial products using data acquisition and control functions.

14.3 Commercial Equipment Using Embedded PCs

So far, we have mostly considered stand-alone PC systems, configured for data acquisition tasks. Many equipment manufacturers require data acquisition and analysis functions in their end product. One approach, becoming increasingly popular, is to use an *embedded PC* as a major component of the product. Depending on the manufacturer's requirements, the PC may still function as a general-purpose computer and run most commercial software packages. On the other hand, it can be completely dedicated to the tasks required by the overall product it is part of, and be unable to run any general-purpose software. In this case, the embedded PC may not even have a floppy disk drive or any standard peripherals. It could run programs from ROM or Flash memory, configured to look like a disk drive to DOS or Windows. For a look at embedded PC standards (such as PC-104), please refer to Chapter 12.

There are many advantages to using an embedded PC in a commercial product, especially for data acquisition, analysis, and storage functions. A huge number of commercial hardware and software products are available, minimizing in-house development costs as well as a new product's time-to-market. Compared to other industrial computer architectures (such as VME and STD BUS products), PCs and their support products are less expensive, are more readily available, and offer more "user friendly" development tools. This also applies to industrial and embedded PCs.

This trend toward using embedded PCs is reflected by the increase in the number of products for this market. Many manufacturers now produce miniaturized PCs, based on a single board, designed to fit within a product using a minimum volume, such as the Compact PCI standard. These products can support standard PC peripherals, such as disk drives and displays, yet will work without them for a scaled-down version in the final product. Software development for products using these embedded PCs can be carried out on a standard desktop PC or the target system itself.

If an embedded PC is technically adequate for performing the required task and its cost can be justified, it is often a good choice, especially compared to other dedicated computer systems. If a PC would be grossly underutilized in an application or the product is very cost sensitive, a dedicated CPU or a microcontroller board is a better alternative. One compromise is designing a board using a microcontroller that emulates a PC (sometimes referred to as a "PC-on-a-chip"), such as the products from ZF Micro Devices (URL: www.zfmicro.com). This approach keeps hardware costs down while allowing you to use PC software products.

One application where embedded PCs are very popular is in network-based products. As more commercial instruments and even appliances use networking features (for using both LANs and the Internet), small embedded PCs become an important part of these products. Many single-board computers (in ISA, PCI, or PC-104 form-factors) contain an Ethernet interface (usually 10BASE-T or 100BASE-T) and have enough memory (DRAM) to run a standard operating system (such as MS-DOS, Microsoft Windows, or Linux). Network software can be added to the embedded PC as a separate package (as when running DOS), or it may already be part of the operating system (as in Windows 95 and above). The embedded PC can then connect to other PCs on the same network (via a LAN) or even to Internet sites.

When selecting network software to use, look carefully at the application. If the embedded PC must connect to a large, server-based LAN or to the Internet, it should use the TCP/IP protocol. However, if it will only connect to one or a just a few other PCs through a simple Ethernet hub, you should consider Microsoft's NETBEUI protocol, which was part of Microsoft Windows for Workgroups (Windows 3.11) and is still supported in Windows 95/98. NETBEUI is a simple peer-to-peer network protocol, useful for small networks that do not have a server computer.

Networked data acquisition equipment is particularly useful in an industrial environment. You can employ many small PCs (with data acquisition interfaces) to monitor and control various manufacturing processes. These PCs (or PC-based instruments) would be connected to a central computer, running appropriate software (such as LABTECH CONTROL) that oversees the entire process. You can even use embedded data acquisition PCs at remote sites and use the Internet for communications.

14.3.1 THE CYBEX 340 Extremity Testing System

As an early example of a typical use of an embedded desktop PC in a piece of commercial equipment, we will look at the CYBEX 340 Extremity Testing System. CYBEX, a division of Lumex Inc., manufactured testing and rehabilitation equipment for the fields of sports medicine, physical therapy, and fitness. These machines were used for evaluating and improving human athletic performance and fitness as well as aiding injury recovery. The 340 System was used for the testing, exercise, and rehabilitation of the extremities (arms and legs).

The CYBEX 340 System is a large machine, which incorporates a full-sized desktop PC chassis in its electronics cabinet, as shown in Figure 14.7. The PC used is a Wyse 286 PC, which is an AT (ISA) system, based on an 80286 CPU with a 10-MHz clock and containing 640 Kbytes of RAM. The keyboard, monitor (EGA), floppy drive, and streaming tape drive are mounted

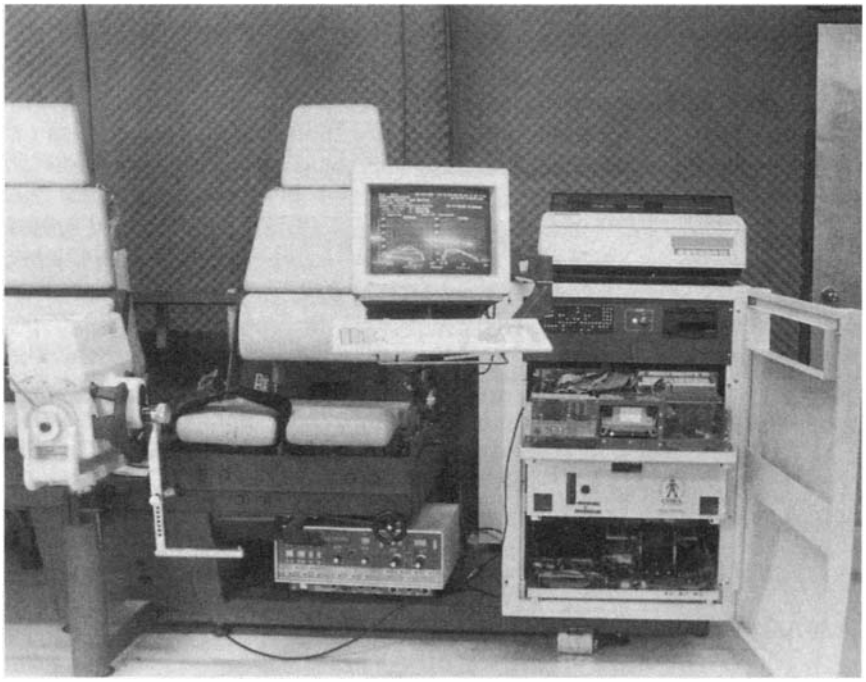


Figure 14-7 CYBEX 340 Extremity Testing System. (Courtesy of CYBEX, a division of Lumex, Inc.)

externally, for normal user access. The PC also contains a 32-Mbyte hard disk drive and runs MS-DOS. Other standard PC peripherals used in this system are a parallel port, connected to an external printer, and a serial port, connected to an internal modem. It contains a 60-Mbyte tape drive unit, for backing up the data collected and stored in its database system. In addition, an optional network interface card may be present, to connect the system to a local area network (LAN) of other CYBEX systems and PCs, including a PC set up as a *file server*, controlling the network.

By using standard peripherals, retaining a floppy drive, and running software under MS-DOS, the PC embedded in the CYBEX 340 System can function as a stand-alone PC and run most commercial software packages. In addition, it runs custom CYBEX software, used for motion control of the extremity testing equipment, acquiring and storing patient data, producing reports, and other functions aiding the medical practitioner.

Figure 14-8 shows a simplified block diagram of the CYBEX 340 System. It is clear from this that, at least electronically, the embedded PC is the heart of the system. The mechanical heart of this CYBEX system is its dynamometer. This unit contains a motor, controlled by a switching servo amplifier, which drives a series of clutches coupled to an output shaft, connected to the patient's limb.

The speed of the motor limits the maximum speed at which the patient can move his or her limb. A patient trying to move faster than the set speed would produce more torque but maintain that fixed speed. The measured torque, at this constant speed, produces clinically significant information about the health and strength of that limb. For example, if someone had a knee injury that was manifested at a particular point in that joint's range of motion (the maximum range that joint can rotate, measured in degrees), there would probably be a drop in torque output at that location. The data collected by this system is torque versus angular position. The torque is a measure of the force produced by the muscles of the tested limb. The angular position covers that limb or joint's range of motion. A typical example of data produced by a CYBEX 340 System (measurements of the author's knee) is shown in Figure 14-9.

For technical and economic reasons, CYBEX chose to produce its own data acquisition boards for the 340 System, instead of adapting general-purpose commercial cards. These boards plug into the standard ISA bus of the Wyse PC, as any commercial card does. As shown in Figure 14-8, these custom boards connect to specialized system hardware, such as the dynamometer, servo amplifier, and power-sequencing unit.

The functions available on these custom boards include analog input, analog output, digital I/O, and counter/timers. The analog input is the torque

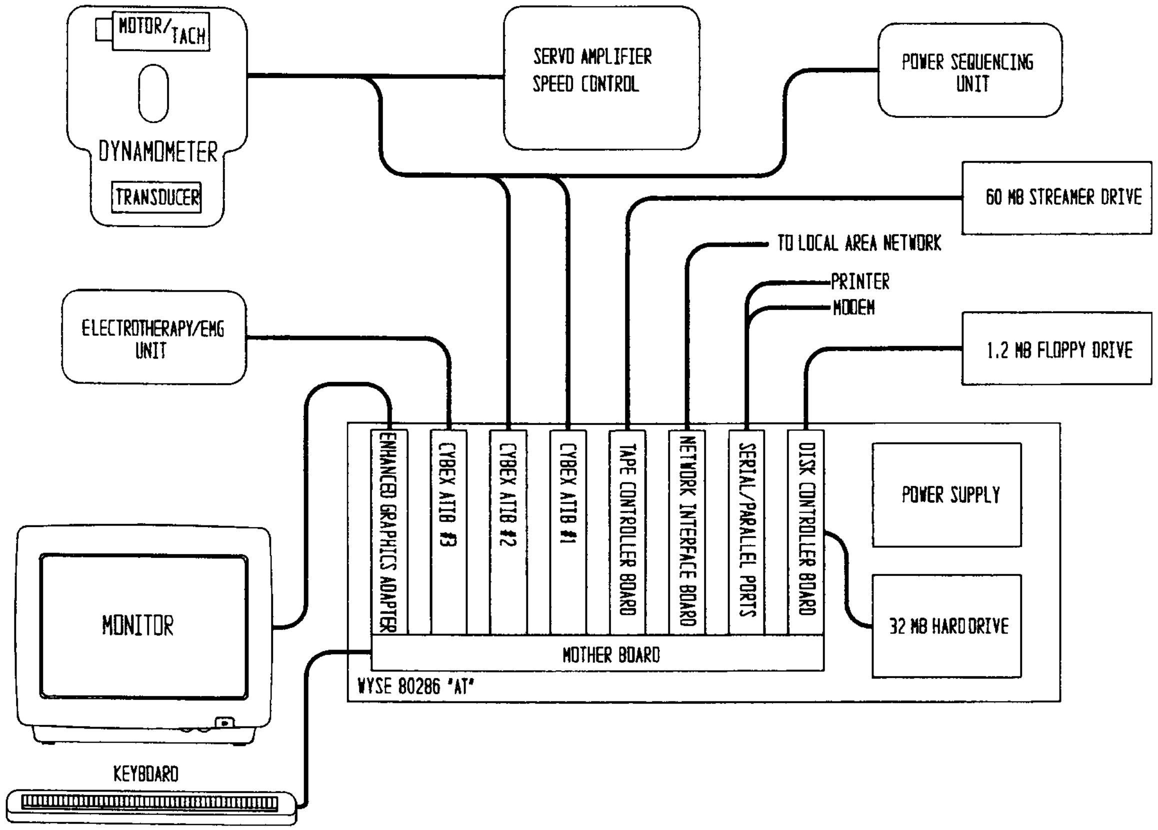
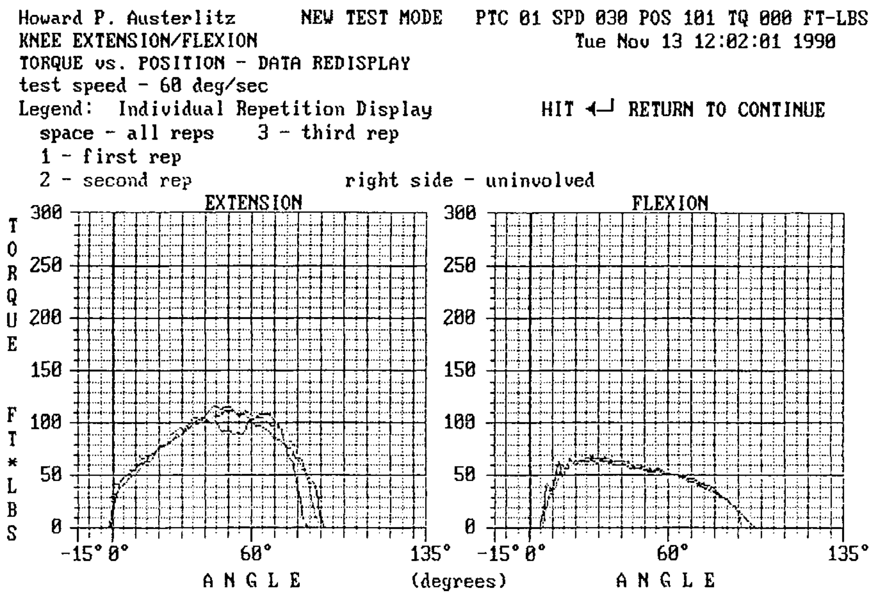


Figure 14-8 Block diagram of the CYBEX 340 Extremity Testing System. (Courtesy of CYBEX, a division of Lumex, Inc.)



```
Howard P. Austerlitz      NEW TEST MODE   PTC 01 SPD 030 POS 101 TQ 000 FT-LBS
KNEE EXTENSION/FLEXION                           Tue Nov 13 12:02:01 1990
TORQUE vs. POSITION - DATA REDISPLAY
test speed - 60 deg/sec
Legend:  Individual Repetition Display          HIT ◄┘ RETURN TO CONTINUE
  space - all reps    3 - third rep
  1 - first rep
  2 - second rep                right side - uninvolved
```

Figure 14-9 Typical data display from a CYBEX 340 System. (Courtesy of CYBEX, a division of Lumex, Inc.)

signal from the dynamometer. The torque signal is derived from a capacitive load cell (a pressure transducer) hydraulically coupled to the dynamometer shaft. This signal, having a 10-V dynamic range, is digitized by a 10-bit ADC. The overall torque range is 360 foot-pounds, so the system can resolve torque as low as 0.35 foot-pounds (or approximately 4 inch-pounds).

The ADC used in the CYBEX 340 System has several modes of operation. Conversions can be triggered asynchronously or at a fixed rate, under software control (up to 25,000 samples/sec), as in a conventional, time-based data acquisition system. However, the data of interest to this system is torque versus angular position. To directly measure this, the ADC conversions are normally triggered by an optical encoder, coupled to the dynamometer shaft. This encoder acts as an angular position transducer, producing a pulse for every 1/2 degree of dynamometer shaft rotation. This produces torque versus position data, independent of rotational velocity, eliminating unnecessary data conversions at slow speeds. This feature could be implemented with a general-purpose, commercial data acquisition card if it accepted external trigger signals.

An analog output of the CYBEX board is used to control the motor speed. This speed control voltage (SCV) is a 0- to 10-V signal, sent to the servo amplifier, which drives the motor. The motor contains a tachometer that produces an analog voltage, proportional to its speed. This tachometer signal is sent back to the servo amp, to complete the feedback loop required for precise speed control. The motor speed set by the servo amp is proportional to the controlling SCV.

The counter/timers available on the CYBEX boards are based on an Intel 8254 IC. They are used for various system timing and counting functions, such as measuring motor speed during calibration. Many different functions are implemented using digital I/O lines. One example is using software-controlled gating of several possible sources into one hardware interrupt line. Another is the remote power-on capability of the system.

The advantages and features of an embedded PC, illustrated by the CYBEX 340 System, apply to a wide variety of computer-controlled equipment: even if a product must be fairly small, the size of PCs designed for embedded applications is continuously shrinking. We will continue to see increasing growth in commercial equipment utilizing embedded PCs. An additional advantage of using an embedded PC is that the final system can be functionally prototyped using commercially available data acquisition hardware and software products, even if it will eventually use custom boards and programs. This can certainly help shorten the development cycle of a new product. At the very least, it provides a software development group with hardware to work with while the final data acquisition boards are still in the design process.

14.3.2 The Tektronix TDS7000 Series Oscilloscope

One interesting example of a recent embedded PC product is the Tektronix TDS7000 series of digital phosphor oscilloscopes (DPOs). These instruments are high-performance, stand-alone digital oscilloscopes with analog bandwidths up to 4 GHz and single-shot sampling rates up to 20 Gsamples/sec. They also use equivalent time sampling (ETS) to achieve much higher sampling rates with repetitive signals. Because of their embedded processors, the TDS7000 oscilloscopes have many features of a standard, Windows-based PC. For example, these instruments all contain a floppy disk drive, a CD-ROM drive, and a hard disk drive (of over 4 Gbytes). They also use a mouse and keyboard and contain a variety of standard I/O ports such as USB, parallel (IEEE 1284), serial (RS-232), Ethernet (10BASE-T/100BASE-T), and GPIB.

Figure 14-10 shows a simplified block diagram of a TDS7000 DPO. The high-speed acquisition system (with multiple analog inputs) uses an embedded Power PC. This processor communicates with the embedded Windows PC (based on an Intel Celeron processor) via an internal PCI bus. The Intel processor, running Microsoft Windows 98, is used for user-interface

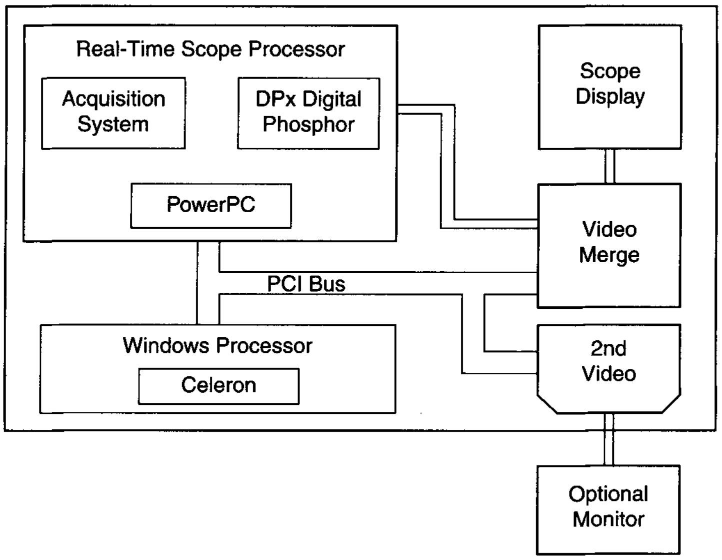


Figure 14-10 Simplified block diagram of a Tektronix TDS7000 series Digital Phosphor Oscilloscope. (Courtesy of Tektronix, Inc.)

functions and enables the instrument to run standard Windows 95/98 software. Figure 14-11 shows a photograph of a TDS7404 oscilloscope displaying typical waveforms.

The embedded Windows PC in the TDS7000 is analogous to a notebook computer. It does not have the expandability of a desktop PC that can accept plug-in cards, but all TDS7000 DPOs have parallel, serial, USB, and GPIB interfaces. Some even have Cardbus (PCMCIA) slots, making them as expandable as any conventional notebook PC. This combination of a high-performance digital oscilloscope with a notebook-like PC is extremely powerful. When used as a general-purpose lab instrument, a TDS7000 can acquire waveforms (just as a standard oscilloscope does) and then save this data to the local hard drive or to another location on a network, using the Ethernet (LAN) port. If the unit is connected to a network, you can easily print captured waveform images on a remote, shared printer.

An example of a fairly complex oscilloscope/PC application in the lab (based on the author's experience) is using a TDS7000 DPO to develop and debug a microcontroller-based circuit board. The DPO was loaded with standard PC-based development software (including a compiler/linker and debugger). A PC-based hardware emulator was run from the DPO's parallel port

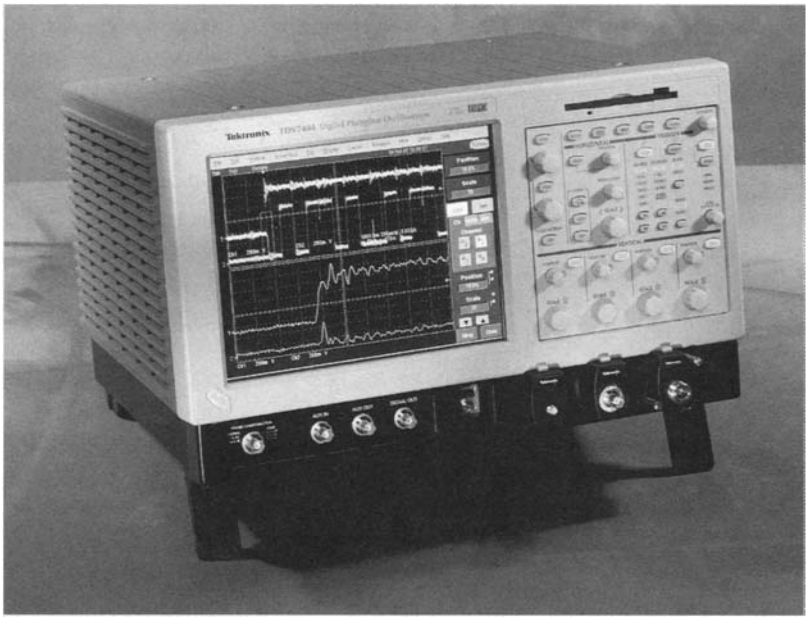


Figure 14-11 A Tektronix TDS7404 Digital Phosphor Oscilloscope. (Courtesy of Tektronix, Inc.)

and connected to the target board (an *emulator* is an instrument used to control a target microprocessor or microcontroller for the purposes of software and hardware development). Using the PC-based software, the target microcontroller board was loaded with test software under development. This code was run (and evaluated) on the target, under control of the emulator (hardware) and debugger (software) hosted on the TDS7000. At the same time, the DPO's oscilloscope probes were connected to test points on the target board. As the test software executed, the DPO was used to display critical waveforms, aiding in the evaluation of the board design.

Since a TDS7000 DPO is portable (albeit much heavier than a notebook PC) it is also suitable for field test work. In this case, having an oscilloscope and a PC in one package can be quite advantageous.

Of course, many embedded PC applications are much simpler than the TDS7000. Still, since PCs have become so ubiquitous, a growing range of products will likely include PC features if not a fully functioning embedded PC. Often, just the ability to connect a device to an existing network is a primary reason for embedding a PC in a product.

14.4 Future Trends in PC-Based Data Acquisition

This concludes our examination of a few "real-world" examples of data acquisition systems based on PC platforms of various form factors. This field is constantly changing, with new products, standards, and approaches appearing continuously.

Without any doubt, the field of PCs will continue to evolve at its typically frenetic pace. In the "Wintel" world, faster Pentium CPUs (and their successors) will continue to appear, along with newer and more sophisticated versions of Windows. Eventually, the PCI bus may become obsolete (as the ISA bus nearly has) and probably become relegated only to embedded and industrial applications. However, it should still be around for many years to come. Desktop PCs are likely to become black boxes without any internal expansion slots and rely solely on standard ports, such as USB and FireWire. Industrial and embedded PCs may become the platforms of choice for data acquisition systems because of their flexible hardware expansion capabilities. CompactPCI will probably become the leading embedded PC architecture.

In the field of sensors, integrating more functions and "intelligence" in sensor units should continue. Growing acceptance of the IEEE 1451 standards will also help accelerate this trend. The increased use of electronic sensors in major consumer products, such as automobiles, will continue to drive the sensor field.

It is impossible to accurately predict future trends in the PC and data acquisition industry (as the predictions in the first edition of this book, 10 years ago, demonstrated). If you are putting together a PC-based data acquisition system, stick to your current requirements, with an eye on future needs. You should be aware of trends in the industry, but relying on new, untested technologies (or companies, for that matter) can be a big gamble. As a gross generalization, data acquisition hardware products and companies tend to have a much longer life than their software counterparts. Always try to first use current, established products to solve a problem. It will usually cost you less time, money, and frustration.

This Page Intentionally Left Blank

APPENDIX

Data Acquisition and Related PC Product Manufacturers

This appendix contains listings of manufacturers of PC-based hardware and software products for data acquisition as well as frame grabbers (image capture) and embedded or industrial PCs. These include many hardware manufacturers who also produce or resell software products. Most software products will run under a 32-bit version of Microsoft Windows (Win 95/98/NT). Most hardware products will work in an ISA or a PCI bus. All URLs are current as of this writing and usually point to a vendor or product's home page. Products or vendors that support non-PC hardware (i.e., Apple Macintosh) are explicitly noted.

Each listing contains the vendor's name, contact information, and product information. The product listings are grouped as either *H/W* (hardware) or *S/W* (software). Unless otherwise noted, *data acq* in a H/W listing denotes both analog and digital I/O. The H/W listing is followed by the *Supported I/F* (interfaces) listing (i.e., ISA, PCI, RS-232, USB). The S/W listing is followed by the *Supported OS* (operating system) listing (i.e., Windows 95/98/NT, Linux, Mac OS). All addresses are in the United States unless otherwise noted. All 800 phone numbers are only valid in North America (United States and Canada). All phone numbers outside of North America begin with the international calling code.

AAEON Electronics Inc. 3 Crown Plaza Hazlet, NJ 07730 (732) 203-9300 www.aaeon.com	H/W PRODUCTS: SUPPORTED I/F:	Embedded/industrial PC, data acq ISA, PCI, PC/104
ACCES I/O Products, Inc. 10623 Roselle Street San Diego, CA 92121 (858) 550-9559 www.acces-usa.com	H/W PRODUCTS: SUPPORTED I/F:	Data acq, communications, bus expansion ISA, PCI, PC/104

Acqiris USA
P.O. Box 2203
234 Cromwell Hill Rd.
Monroe, NY 10950-1430
(845) 782 6544
www.acqiris.com

H/W PRODUCTS: High-speed data acq
SUPPORTED I/F: PCI, CompactPCI

Acqutek Corp., Inc.
1549 S. 1100 East
Salt Lake City, UT 84105
(801) 485-4594
www.acqu.com

H/W PRODUCTS: Data acq, embedded/industrial PC, communications
SUPPORTED I/F: ISA, PCI

Acrosser USA
10564 Progress Way, Unit D
Cypress, CA 90630
(714) 827-9938
www.acrosser.com

H/W PRODUCTS: Embedded/industrial PC, data acq
SUPPORTED I/F: ISA, PCI, PC/104

ADAC Corporation
24 River Street
Winchester, Massachusetts 01890
(781) 721-9800
www.adac.com

H/W PRODUCTS: Data acq, bus expansion, industrial PC
SUPPORTED I/F: ISA, PCI, PCMCIA, VME

Adlink Technology Inc.
15279 Alton Pkwy., Suite 400
Irvine, CA 92618
(949) 727-2077
www.adlinktechnology.com

H/W PRODUCTS: Data acq, embedded/industrial PC, image capture, motion control
SUPPORTED I/F: ISA, PCI, Compact-PCI, USB

Advantech Automation Corp.
1320 Kemper Meadow Dr., Suite 500
Cincinnati, OH 45240
(877) 294-8989
www.advantech.com

H/W PRODUCTS: Embedded/industrial PC, data acq
SUPPORTED I/F: ISA, PCI, PC/104

Agilent Technologies (formerly HP)
P.O. Box 10395
Palo Alto, CA 94303
(650) 752-5000
www.tm.agilent.com

S/W PRODUCTS: Agilent VEE (general-purpose data acq)
SUPPORTED OS: Win 95/98/NT 2000

Amplicon Liveline Ltd. Centenary Industrial Estate Hollingdean Road Brighton East Sussex BN2 4AW United Kingdom +44 1273-608-331 www.amplicon.co.uk	H/W PRODUCTS: SUPPORTED I/F:	Data acq, signal conditioning ISA, PCI, PCMCIA, USB, parallel port, serial port
Ampro Computers, Inc. 5215 Hellyer Avenue #110 San Jose, California 95138-1007 (408) 360-0200 www.ampro.com	H/W PRODUCTS: SUPPORTED I/F:	Embedded PC PC/104, PC/104-Plus, EBX
Amtec Engineering, Inc. 13920 SE Eastgate Way, Suite 220 Bellevue, WA 98005 (425) 653-1200 www.amtec.com	S/W PRODUCTS: SUPPORTED OS:	Tecplot (data display) Win 95/98/Me/NT/2000/XP, Linux, Solaris, HP-UX, UNIX
Analog and Digital Peripherals, Inc. P.O. Box 499 Troy, OH 45373 (800) 758-1041 www.adpi.com	H/W PRODUCTS: SUPPORTED I/F: S/W PRODUCTS: SUPPORTED OS:	Embedded PC PCMCIA, USB Easi Daq (data acq via PCMCIA cards) Win 95/98/NT/CE, Linux
Analogic Corp. 8 Centennial Drive Peabody, MA 01960 (978) 977-3000 www.analogic.com	H/W PRODUCTS: SUPPORTED I/F:	Data acq, image capture CompactPCI, VME, VXI
Aptech Systems, Inc. 23804 SE Kent-Kangley Road Maple Valley, WA 98038 (425) 432-7855 www.Aptech.com	S/W PRODUCTS: SUPPORTED OS:	GAUSS (data analysis and display) Win 95/98/NT/2000, Linux, Solaris

Arbor Technology Corporation 5F, No 738 Zhong Zheng Road Zhong He, 235 Taipei Taiwan 886-2-8226-9396 www.arbor.com.tw	H/W PRODUCTS:	Data acq, embedded/ industrial PC, image capture
	SUPPORTED I/F:	ISA, PCI, PC/104, PC/104-Plus
Arcom Control Systems, Inc. 7500 West 161st Street Stilwell, KS 66085 (888) 941-2224 www.arcomcontrols.com	H/W PRODUCTS:	Data acq, embedded PC, motion control
	SUPPORTED I/F:	ISA, PCI, PC/104
Biodata Ltd. (Microlink Measurement and Control) 10 Stocks Street Manchester, M8 8QG United Kingdom +44 161-834-6688 www.microlink.co.uk	H/W PRODUCTS:	Data acq
	SUPPORTED I/F:	ISA, USB
BitFlow 21-G Olympia Avenue Woburn, MA 01801 (781) 932-2900 www.bitflow.com	H/W PRODUCTS:	Image capture
	SUPPORTED I/F:	PCI
Capital Equipment Corp. 900 Middlesex Turnpike, Bldg 2 Billerica, MA 01821-3929 (800) 234-4232 www.cec488.com	H/W PRODUCTS:	Data acq, GPIB, Web/DAQ (Internet data acq server)
	SUPPORTED I/F:	ISA, PCI
	S/W PRODUCTS:	TestPoint (general-purpose data acq)
	SUPPORTED OS:	Win 95/98/Me/ NT/2000
Chase Scientific Company 7960-B Soquel Drive, Suite 191 Aptos, CA 95003 (831) 464-2584 www.chase2000.com	H/W PRODUCTS:	High-speed data acq
	SUPPORTED I/F:	ISA, PCI, Compact-PCI, VME, VXI, PC/104, PC/104-Plus

Comark Corp.
93 West Street
Medfield, Massachusetts 02052
(800) 280-8522
www.comarkcorp.com

H/W PRODUCTS: Embedded/industrial PC, data acq
SUPPORTED I/F: ISA, PCI, PC/104

Coreco, Inc.
6969 Trans-Canada Highway, Suite # 142
St. Laurent, Quebec H4T 1VB
Canada
(514) 333-1301
www.coreco.com

H/W PRODUCTS: Image capture
SUPPORTED I/F: ISA, PCI

CyberResearch, Inc.
25 Business Park Dr.
Branford, CT 06405
(800) 341-2525
www.cyberresearch.com

H/W PRODUCTS: Data acq, signal conditioning, motion control, GPIB, embedded/industrial PC
SUPPORTED I/F: ISA, PCI, PC/104, serial port
S/W PRODUCTS: Distributes major data acq software packages
SUPPORTED OS: Depends on product (most are Win95/98/NT/2000)

Dasytec USA
(a National Instruments Company)
11 Eaton Road
PO Box 748
Amherst, NH 03031-0748
(800) 731-5015
www.dasylab.net or
www.dasytec.com

S/W PRODUCTS: Dasy Lab (general purpose data acq)
SUPPORTED OS: Win 98/NT/2000

Data Translation, Inc.
100 Locke Drive
Marlboro, MA 01752-1192
(800) 525-8528
www.datatranslation.com

H/W PRODUCTS:	Data acq, signal conditioning, image capture
SUPPORTED I/F:	ISA, PCI, PCMCIA, USB
S/W PRODUCTS:	Data Acq Omni CD (drivers & utilities), Quick Data Acq (Mac only)
SUPPORTED OS:	Win 95/98/NT/2000, Mac OS

Dataq Instruments, Inc.
150 Springside Drive, Suite B220
Akron, OH 44333
(800) 553-9006
www.dataq.com

H/W PRODUCTS:	Data acq, signal conditioning
SUPPORTED I/F:	ISA, Ethernet, USB, parallel port, serial port
S/W PRODUCTS:	WinDaq (data acq)
SUPPORTED OS:	Win 3.1, Win 95/NT

Datastick Systems, Inc.
275 Saratoga Ave., Ste. 160
Santa Clara, CA 95050
(408) 615 5774
www.datastick.com

H/W PRODUCTS:	Data acq
SUPPORTED I/F:	Palm PDA

Diamond Systems Corp.
8430-D Central Avenue
Newark, CA 94560
(800) 367-2104
www.diamondsys.com

H/W PRODUCTS:	Data acq, embedded PC
SUPPORTED I/F:	PC/104

DSP Development Corporation
3 Bridge Street
Newton, MA 02458
(800) 424-3131
www.dadisp.com

S/W PRODUCTS:	DADiSP (data analysis and display)
SUPPORTED OS:	Win 95/98/NT/2000, UNIX

Electronic Energy Control, Inc. 380 South Fifth Street, Suite 604 Columbus, Ohio 43215-5491 (800) 842-7714 www.eeci.com	H/W PRODUCTS: SUPPORTED I/F:	Data acq Serial port, USB
Gage Applied, Inc. (a Tektronix Company) 2000 32nd Ave. Lachine, QC Canada H8T3H7 (800) 567-GAGE www.gage-applied.com	H/W PRODUCTS: SUPPORTED I/F: S/W PRODUCTS: SUPPORTED OS:	Very high-speed data acq ISA, PCI Gage Scope (virtual oscilloscope) MS-DOS, Win 95/98/NT/2000
General Standards Corp. 8302A Whitesburg Drive Huntsville, AL 35802 (800) 653-9970 www.generalstandards.com	H/W PRODUCTS: SUPPORTED I/F:	Data acq, communications (serial) PCI, CompactPCI, PC/104-Plus, VME
GW Instruments Inc 35 Medford Street Somerville, MA 02143-4237 (617) 625-4096 www.gwinst.com	H/W PRODUCTS: SUPPORTED I/F: S/W PRODUCTS: SUPPORTED OS:	Data acq PCI, NuBus (Mac) InstruNet (data acq), SuperScope (Mac only) Win 95/NT, Mac OS
HEM Data Corp. 17336 Twelve Mile Road Southfield, MI 48076-2123 (248) 559-5607 www.hemdata.com	S/W PRODUCTS: SUPPORTED OS:	SnapMaster (general-purpose data acq) Win 3.1, Win 95/98
ICS Electronics 7034 Commerce Circle Pleasanton, CA 94588 (925) 416-1000 www.icselect.com	H/W PRODUCTS: SUPPORTED I/F:	GPIB, communications ISA, PCI, CompactPCI, PCMCIA, VXI, parallel port, serial port

Ines GmbH
Goettinger Chaussee 115
D 30459 Hannover
Germany
+49 511 943 810
www.inesinc.com

H/W PRODUCTS:	Data acq, GPIB
SUPPORTED I/F:	ISA, PCI, PCMCIA
S/W PRODUCTS:	FreeVIEW (data acq via PC sound card-free)
SUPPORTED OS:	Win 95/98/NT/2000

Integ Process Group, Inc.
11279 Perry Highway, Suite 502
Wexford, PA 15090
(724) 933-9350
www.integpg.com

H/W PRODUCTS:	Network-based data acq
SUPPORTED I/F:	Serial port, Ethernet

Intelligent Instrumentation, Inc.
3000 E. Valencia, Suite 100
Tucson, AZ 85706
(800) 685-9911
www.instrument.com

H/W PRODUCTS:	Data acq
SUPPORTED I/F:	ISA, Ethernet, USB

IOtech, Inc.
25971 Cannon Road
Cleveland, Ohio 44146
(440) 439-4091
www.iotech.com

H/W PRODUCTS:	Data acq, signal conditioning, GPIB
SUPPORTED I/F:	ISA, PCI, CompactPCI, USB

JK Microsystems, Inc.
1403 Fifth Street, Suite D
Davis, California, 95616
(530) 297-6073
www.jkmicro.com

H/W PRODUCTS:	Embedded PC, data acq
SUPPORTED I/F:	Proprietary

JUMPtecAdastra
3988 Trust Way
Hayward, CA 94545
(510) 732-6900
www.adastra.com

H/W PRODUCTS:	Embedded PC
SUPPORTED I/F:	PC/104, proprietary (DIMM-PC)

Keithley Instruments, Inc. 28775 Aurora Road Cleveland, OH 44139 (440) 248-0400 www.keithley.com	H/W PRODUCTS:	Data acq, GPIB, signal conditioning, serial cards, PC instruments
	SUPPORTED I/F:	ISA, PCI, PCMCIA
	S/W PRODUCTS:	DriverLinx (drivers), Visual Scope (virtual oscilloscope)
	SUPPORTED OS:	Win 95/98/NT/2000
Kontron (formerly ICS Advent) 6260 Sequence Drive San Diego, CA 92121-4371 (858) 677-0877 www.icsadvent.com	H/W PRODUCTS:	Data acq, embedded/industrial PC, communications (serial)
	SUPPORTED I/F:	ISA, PCI, CompactPCI
LabJack Corporation 3112 S. Independence Court Lakewood, CO 80227-4445 (303) 942-0228 www.labjack.com	H/W PRODUCTS:	Data acq
	SUPPORTED I/F:	USB
Laboratory Technologies Corp. Two Dundee Park, Suite B09 Andover, MA, 01810 (978) 470-0099 www.labtech.com	S/W PRODUCTS:	LABTECH NOTEBOOK, LABTECH CONTROL (general-purpose data acq)
	SUPPORTED OS:	Win 95/98/NT/2000
Lawson Labs, Inc. 3217 Phoenixville Pike Malvern, PA 19355 (800) 321-5355 www.lawsonlabs.com	H/W PRODUCTS:	Data acq
	SUPPORTED I/F:	ISA, parallel port, serial port, USB, Apple II
Lisberger Technologies 848 Clayton St. San Francisco, CA 94117 (415) 476-1062 www.listech.com	H/W PRODUCTS:	Data acq w/time stamping
	SUPPORTED I/F:	ISA
	SUPPORTED OS:	MS-DOS, Win 95/NT

LPTek Corp.
1100 Shames Drive
Westbury, NY 11590-1746
(516) 333-8820
www.lptek.com

H/W PRODUCTS: Data acq
SUPPORTED I/F: Parallel port

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098
(508) 647-7000
www.mathworks.com

S/W PRODUCTS: MATLAB (general-purpose data analysis and display with data acq features)
SUPPORTED OS: Win 95/98/Me/NT/2000, Mac OS, Linux, UNIX

Measurement Computing, Corp.
(formerly ComputerBoards, Inc.)
16 Commerce Boulevard
Middleboro, MA 02346
(508) 946-5100
www.measurementcomputing.com

H/W PRODUCTS: Data acq, signal conditioning
SUPPORTED I/F: ISA, PCI, CompactPCI, PCMCIA, PC/104, USB

Measurement Systems Ltd.
16 Kingfisher Court
Newbury, Berkshire
RG14 5SJ
United Kingdom
+44 (0)1635 576800
www.measurementsystems.co.uk

H/W PRODUCTS: Data acq, industrial PCs
SUPPORTED I/F: PCI, CompactPCI

Megatel Computer Corp.
125 Wendell Avenue
Weston, Ontario M9N 3K9
Canada
(416) 245-2953
www.pc104sbc.com

H/W PRODUCTS: Embedded PC
SUPPORTED I/F: PC/104

Mesa Electronics
4175 Lakeside Drive, Suite 100
Richmond, CA 94806-1950
(510) 223-9272
www.mesanet.com

H/W PRODUCTS: Embedded PC, data acq, motion control
SUPPORTED I/F: PC/104, PC/104-Plus

Microsoft Corp. One Microsoft Way Redmond, WA 98052-6399 (425) 882-8080 www.microsoft.com/office/excel	S/W PRODUCTS:	Excel (spreadsheet for general-purpose data analysis and display)
	SUPPORTED OS:	MS Windows (version depends on Excel version)
Microstar Laboratories, Inc. 2265 116th Ave. NE Bellevue, WA 98004 (425) 453-2345 www.mstarlabs.com	H/W PRODUCTS:	Data acq, signal conditioning
	SUPPORTED I/F:	ISA, PCI
	S/W PRODUCTS:	DAPview, Windows Toolkit (development tools)
	SUPPORTED OS:	Win 95/98
Micro/Sys, Inc. 3730 Park Pl. Montrose, CA 91020 (818) 244-4600 www.embeddedsys.com	H/W PRODUCTS:	Embedded PC, data acq, communications
	SUPPORTED I/F:	PC/104, PC/104-Plus, STD Bus
Micro Technic Svenstrupvej 90 5260 Odense S Denmark +45 66 15 30 00 www.micro-technic.com	H/W PRODUCTS:	Data acq, communications
	SUPPORTED I/F:	PC/104
National Instruments Corp. 11500 N. Mopac Expressway Austin, TX 78759-3504 (512) 794-0100 www.ni.com	H/W PRODUCTS:	Data acq, signal conditioning, GPIB, image capture, motion control, PC instruments
	SUPPORTED I/F:	ISA, PCI, PXI/CompactPCI, PCMCIA, IEEE 1394, USB, VME/VXI
	S/W PRODUCTS:	LabVIEW (general purpose data acq)
	SUPPORTED OS:	Win 95/98/Me/NT/2000, Mac OS, Linux, Sun Solaris, HP-UX

Octagon Systems 6510 W. 91st Avenue Westminster, CO 80031 (303) 430-1500 www.octagonsystems.com	H/W PRODUCTS: SUPPORTED I/F:	Embedded/industrial PC, data acq, communications, motion control ISA, PCI, PC/104
Omega Engineering, Inc. One Omega Drive P.O. Box 4047 Stamford, CT 06907-0047 www.omega.com	H/W PRODUCTS: SUPPORTED I/F: S/W PRODUCTS: SUPPORTED OS:	Data acq, GPIB, sensors ISA, PCI, Ethernet, USB Distributes data acq software from multiple vendors Win 3.1, Win 95/98/NT
Ontrak Control Systems Inc. 764 Notre Dame Ave., Unit #1 Sudbury, Ontario P3A 2T2 Canada (705) 671-2652 www.ontrak.net	H/W PRODUCTS: SUPPORTED I/F:	Data acq Serial port
Pico Technology Ltd. The Mill House Cambridge Street St. Neots Cambridgeshire PE19 1QB United Kingdom +44 1480-396-395 www.picotech.co.uk	H/W PRODUCTS: SUPPORTED I/F: S/W PRODUCTS: SUPPORTED OS:	Data acq Parallel port, serial port PicoLog (data acq) MS-DOS, Win 3.1, Win 95/98/Me/NT/2000
PixelSmart P.O. Box 76 Lewiston, NY 14092 (800) 884-1734 www.pixelsmart.com	H/W PRODUCTS: SUPPORTED I/F:	Image capture ISA, PCI, PC/104

Quatech	H/W PRODUCTS:	Data acq, communications
662 Wolf Ledges Parkway		
Akron, Ohio 44311	SUPPORTED I/F:	ISA, PCI, CompactPCI, PCMCIA, USB
(800) 553-1170		
www.quatech.com	S/W PRODUCTS:	DAQDRIVE (drivers)
	SUPPORTED OS:	MS-DOS, Win 3.1, Win 95/98/NT
Real Time Devices USA, Inc.	H/W PRODUCTS:	Data acq, embedded PC, image capture, motion control
103 Innovation Blvd.		
P.O. Box 906		
State College, PA 16804	SUPPORTED I/F:	PC/104, PC/104Plus
(814) 234-8087		
www.rtdusa.com		
Scientific Solutions, Inc.	H/W PRODUCTS:	Data acq, GPIB
9323 Hamilton Drive	SUPPORTED I/F:	ISA, PCI
Mentor, OH 44060	S/W PRODUCTS:	LabPac (drivers)
(440) 357-1400	SUPPORTED OS:	MS-DOS, Win 95/98/NT/2000
www.labmaster.com		
Sensoray Company Inc.	H/W PRODUCTS:	Data acq, image capture, sensor conditioning, embedded PC
7337 S.W. Tech Center Drive		
Tigard, Oregon 97223		
(503) 684-8005		
www.sensoray.com	SUPPORTED I/F:	ISA, PCI, CompactPCI, PC/104, PC/104-Plus, Ethernet
Signatec	H/W PRODUCTS:	High-speed data acq, industrial PC, signal processing (DSP-based)
1138 East Sixth Street		
Corona, CA 92879		
(909) 734-3001		
www.signatec.com	SUPPORTED I/F:	ISA, PCI
SiliconSoft Inc.	H/W PRODUCTS:	Data acq
4760 Castlewood Dr.	SUPPORTED I/F:	ISA, parallel port, serial port
San Jose, CA		
(408) 446-4521		
www.siliconsoft.com		

Soltec Corp. 12977 Arroyo Street San Fernando, CA 91340 (800) 423-2344 www.solteccorp.com	H/W PRODUCTS:	Data acq, embedded/industrial PC
	SUPPORTED I/F:	ISA, PCI, CompactPCI
Spiral Software 57 Baker Hill Road Lyme, NH 03768 (603) 795-4004 www.spiralsoftware.com	S/W PRODUCTS:	EasyPlot (data analysis and display)
	SUPPORTED OS:	Win 3.1, Win 95/98/NT
SuperLogics 94 Falmouth Road Newton, MA 02465 (617) 332-3627 www.superlogics.com	H/W PRODUCTS:	Data acq, signal conditioning, communications, industrial PC
	SUPPORTED I/F:	ISA, PCI, PCMCIA, RS-232, USB
Symmetric Research 9805 NE 116th Street, #7407 Kirkland, WA 98034 (702) 341-9325 www.symres.com	H/W PRODUCTS:	Data acq, DSP co-processors
	SUPPORTED I/F:	ISA, parallel port
TAL Technologies 2027 Wallace Street Philadelphia PA 19130 (800) 722-6004 www.taltech.com	S/W PRODUCTS:	WinWedge, TCP-Wedge (data acq via RS-232 or Internet)
	SUPPORTED OS:	MS-DOS, Win 3.1, Win 95/98/NT/2000
Theorist Interactive, LLC 26 Church Street Harvard Square Cambridge, MA 02138 (617) 868-1774 www.livemath.com	S/W PRODUCTS:	LiveMath (mathematical analysis and display)
	SUPPORTED OS:	Win 95/98/Me/NT/2000, Mac OS, Linux, Solaris
Traquair Data Systems, Inc. 114 Sheldon Road Ithaca, NY 14850 (607) 266-6000 www.traquair.com	H/W PRODUCTS:	DSP-based data acq
	SUPPORTED I/F:	PCI, CompactPCI

Ultraview Corp. 34 Canyon View Orinda, CA 94563 (925) 253-2960 www.ultraviewcorp.com	**H/W PRODUCTS:** **SUPPORTED I/F:**	High-speed data acq, bus extenders ISA, PCI
United Electronic Industries, Inc. 611 Neponset Street Canton, MA 02021 (800) 829-4632 www.ueidaq.com	**H/W PRODUCTS:** **SUPPORTED I/F:**	Data acq, signal conditioning ISA, PCI, CompactPCI
Universal Technical Systems, Inc. 202 West State Street, Suite 700 Rockford, IL 61101 (815) 963-2220 www.uts.com	**S/W PRODUCTS:** **SUPPORTED OS:**	TK Solver (mathematical analysis) Win 95/98/NT/2000
Validyne Engineering 8626 Wilbur Avenue Northridge, CA 91324 (818) 886-2057 www.validyne.com	**H/W PRODUCTS:** **SUPPORTED I/F:**	Data acq, signal conditioning, sensors ISA, PCMCIA
VersaLogic Corp. 3888 Stewart Road Eugene, OR 97402 (541) 485-8575 www.versalogic.com	**H/W PRODUCTS:** **SUPPORTED I/F:**	Embedded PC, data acq PC/104, PC/104-Plus, STD Bus
VMIC 12090 S. Memorial Parkway Huntsville, AL 35803 (800) 322-3616 www.vmic.com	**H/W PRODUCTS:** **SUPPORTED I/F:**	Embedded/Industrial PC, data acq, communications PCI, CompactPCI, VME, PC/104, PC/104-Plus
WaveEdge Technologies 9705 Glenway Ct. Burke, VA 22015 (703) 455-0750 www.waveedge.com	**H/W PRODUCTS:** **SUPPORTED I/F:**	High-speed data acq ISA, PCI

WaveMetrics, Inc. P.O. Box 2088 Lake Oswego, OR 97035 (503) 620-3001 www.wavemetrics.com	S/W PRODUCTS: SUPPORTED OS:	IGOR Pro (data analysis and display) Win 95/98/Me/NT/2000, Mac OS
Windmill Software Ltd. P.O. Box 58 North District Office Manchester, M8 8QR United Kingdom +44 161-833-2782 www.windmill.co.uk	H/W PRODUCTS: SUPPORTED I/F: S/W PRODUCTS: SUPPORTED OS:	Data acq USB Windmill (data acq) Win 95/98/NT/2000
WinSystems, Inc. 715 Stadium Drive Arlington, Texas 76011 (817) 274-7553 www.winsystems.com	H/W PRODUCTS: SUPPORTED I/F:	Embedded PC PC/104, PC/104-Plus, STD Bus
Wolfram Research, Inc. 100 Trade Center Drive Champaign, IL 61820-7237 (217) 398-0700 www.wolfram.com	S/W PRODUCTS: SUPPORTED OS:	Mathematica (mathematical analysis and display) Win 95/98/NT/2000, Mac OS, UNIX, Linux
Ziatech (an Intel Company) 1050 Southwood Drive San Luis Obispo, CA 93401 (805) 541-0488 www.ziatech.com	H/W PRODUCTS: SUPPORTED I/F:	Industrial PC CompactPCI

Bibliography

Advanced Micro Devices. “Personal Computer Products Data Book,” Advanced Micro Devices, Sunnyvale, CA, 1989.

Analog Devices, Inc. “Analog-Digital Conversion Handbook,” Prentice-Hall, Englewood Cliffs, NJ, 1986.

Andrews, M. “Learn Visual C++ Now,” Microsoft Press, Redmond, WA, 1996.

Apple Computer, Inc. “Designing Cards and Drivers for the Macintosh Family,” Addison-Wesley, Reading, MA, 1990.

Aptech Systems, Inc. “GAUSS User Guide,” Aptech Systems, Inc., 2001.

Biber, C., Ellin, S., Shenk, E., and Stempeck, J. The Polaroid Ultrasonic Ranging System, *67th Audio Engineering Society Proceedings*, New York, 1980.

Burr-Brown Research Corp. “Operational Amplifiers Design and Applications,” McGraw-Hill, New York, 1971.

Cappellini, V. (ed.). “Data Compression and Error Control Techniques with Applications,” Academic Press, Orlando, FL, 1985.

Cooper, D., and Clancy, M. “Oh! Pascal!,” W. W. Norton and Co., New York, 1982.

Cornejo, C., and Lee, R. Comparing IBM’s Micro Channel and Apple’s NuBus. *Byte, Extra Edition—Inside the IBM PCs*, 1987.

Cress, P., Dirksen, P., and Graham, J. “FORTRAN IV with WATFOR and WATFIV,” Prentice-Hall, Englewood Cliffs, NJ, 1970.

DeMarre, D., and Michaels, D. “Bioelectronic Measurements,” Prentice-Hall, Englewood Cliffs, NJ, 1983.

Dettmann, T. “DOS Programmer’s Reference,” Que Corp., Carmel, IN, 1988.

Eggebrecht, L. “Interfacing to the IBM Personal Computer,” Howard W. Sams and Co., Indianapolis, IN, 1986.

Franklin, M. “Using the IBM PC: Organization and Assembly Language Programming,” CBS College Publishing, New York, 1984.

Friedman-Hill, E. J. “Java: Your Visual Blueprint for Building Portable Java Programs,” Hungry Minds, Inc., New York, 2001.

Herbert, T. Introduction to TCP/IP, Part 1. *Embedded Systems Programming*, December 1999.

Higgins, R. “Digital Signal Processing in VLSI,” Prentice-Hall, Englewood Cliffs, NJ, 1990.

Hordeski, M. “The Design of Microprocessor, Sensor and Control Systems,” Reston Publishing Co., Reston, VA, 1985.

IBM Corp. "IBM Personal System/2 Hardware Interface Technical Reference," IBM, 1988.

IBM Corp. "IBM Technical Reference, Disk Operating System," IBM, 1986.

IBM Corp. "IBM Technical Reference, Personal Computer AT," IBM, Boca Raton, FL, 1985.

IBM Corp. "IBM Technical Reference, Options and Adapters," IBM, Boca Raton, FL, 1984.

IBM Corp. "IBM Technical Reference, Personal Computer," IBM, Boca Raton, FL, 1984.

Intel Corp. "Microsystem Components Handbook," Intel Corp., Santa Clara, CA, 1985.

Intel Corp. and Microsoft Corp. "Plug and Play ISA Specification, Version 1.0a," Intel Corp. and Microsoft Corp., May 5, 1994.

Johnson, T. A Comparison of MC68000 Family Processors. *Byte*, September 1986.

Jordan, L., and Churchill, B. "Communications and Networking for the IBM PC and Compatibles," Prentice-Hall Press, New York, 1987.

Kernighan, B., and Ritchie, D. "The C Programming Language," Prentice-Hall, Englewood Cliffs, NJ, 1978.

King, A. "Inside Windows 95," Microsoft Press, Redmond, WA, 1994.

Laboratory Technologies Corp. "LABTECH User's Guide," Laboratory Technologies Corp., Andover, MA, 1999.

Marshall, T., and Potter, J. How the Macintosh II NuBus Works. *Byte*, December 1988.

Mason, W. (ed.). "Physical Acoustics," Academic Press, New York, 1968.

The MathWorks, Inc. "Data Acquisition Toolbox User's Guide, Version 2," The MathWorks, Inc., Natick, MA, 2000.

The MathWorks, Inc. "Getting Started with MATLAB, Version 6," The MathWorks, Inc., Natick, MA, 2001.

Microsoft Corp. "Macro Assembler for the MS-DOS Operating System— Programmer's Guide," Microsoft Corp., Redmond, WA, 1987.

Microsoft Corp. "Microsoft C for the MS-DOS Operating System—Language Reference," Microsoft Corp., Redmond, WA, 1987.

Microsoft Corp. "Plug and Play Parallel Port Devices, Version 1.0b," Microsoft Corp., March 15, 1996.

Microsoft Corp. "Microsoft Windows 95 Resource Kit," Microsoft Press, Redmond, WA, 1995.

Motorola, Inc. "M68000 Family Reference," Motorola, Inc., Phoenix, AZ, 1988.

Muratore, J., Carleton, H., and Austerlitz, H. Ultrasonic Spectra of Porous Composites. *IEEE Ultrasonics Symposium Proceedings*, San Diego, CA, 1982.

National Instruments. "Getting Started with LabVIEW," National Instruments, Austin, TX, 2000.

National Semiconductor Corp. "Linear Databook," National Semiconductor Corp., Santa Clara, CA, 1982.

Niemeyer, P., and Peck, J. "Exploring Java," O'Reilly & Associates, Inc., Sebastopol, CA, 1996.

Norton, H. "Handbook of Transducers," Prentice-Hall, Englewood Cliffs, NJ, 1989.

O'Mara, B. Designing an IEEE 1451.2-Compliant Transducer. *Sensors*, August 2000.

Oppenheim, A., and Schafer, R. "Digital Signal Processing," Prentice-Hall, Englewood Cliffs, NJ, 1975.

PC/104 Consortium. "PC/104-Plus Specification, Version 1.0," PC/104 Consortium, February 1997.

PC/104 Consortium. "PC/104 Specification, Version 2.3," PC/104 Consortium, June 1996.

PCI Industrial Computers Manufacturers Group. "CompactPCI Short Form Specification," PCI Industrial Computers Manufacturers Group, November 1, 1995.

PCI Special Interest Group. "PCI Local Bus Specification, Revision 2.2," PCI Special Interest Group, Portland, OR, December 18, 1998.

PCMCIA. "PC Card Standard," PCMCIA, March 1997.

Polikar, R. The Wavelet Tutorial. http://www.public.iastate.edu/~rpolikar/WAVELETS, Iowa State University, Ames, IA, 1998.

Rosch, W. L. "Hardware Bible, Fifth Edition," Que Corp., Indianapolis, IN, 1999.

Sams Publishing. "Sams Teach Yourself C++ in 21 Days, Fourth Edition," Sams Publishing, 2001.

Savitch, W. "Java: An Introduction to Computer Science and Programming," Prentice-Hall, Upper Saddle River, NJ, 1999.

Scalzo, F., and Hughes, R. "Elementary Computer-Assisted Statistics," Van Nostrand Reinhold Co., 1978.

Schildt, H. "Turbo C/C++: The Complete Reference, Second Edition," Osborne McGraw-Hill, 1992.

Selvarajan, A. Fiber Optic Sensors and their Applications. http://www.ntu.edu.sg/mpe/research/programmes/sensors/sensors/fos/fosselva.html, Indian Institute of Science, Bangalore, India.

Sobell, M. G. "A Practical Guide to Linux," Addison-Wesley, Reading, MA, 1997.

Solari, E. "AT Bus Design," Annabooks, San Diego, CA, 1990.

Solari, E., and Willse, G. "PCI Hardware and Software Architecture and Design, Third Edition," Annabooks, San Diego, CA, 1996.

Spurgeon, C. E. "Ethernet: The Definitive Guide," O'Reilly & Associates, Inc., Sebastopol, CA, 2000.

Summer, S. "Electronic Sensing Controls," Chilton Book Co., Philadelphia, 1969.

Thomas, H. "Handbook of Biomedical Instrumentation and Measurement," Reston Publishing Co., Reston, VA, 1974.

Travis, B. Sensors Smarten Up. *EDN Access*, March 4, 1999.

Valens, C. A Really Friendly Guide to Wavelets. http://perso.wanadoo.fr/plyvalens/clemens/wavelets/wavelets.html.

WaveMetrics, Inc. "IGOR Pro, Version 4.0: Getting Started," WaveMetrics, Inc., Lake Oswego, OR, 2000.

Weber Systems, Inc. "C Language Users Handbook," Ballantine Books, New York, 1984.

Wells, P. Intel's 80386 Architecture. *Byte, Extra Edition—Inside the IBM PCs*, 1986.

Wright, P. "Beginning Visual Basic 6," Wrox Press Ltd., Birmingham, U.K., 1998.

This Page Intentionally Left Blank

Index