Practical Interfacing in the Laboratory

Using a PC for Instrumentation, Data Analysis, and Control

Stephen E. Derenzo
University of California, Berkeley, California

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published as *Interfacing: A laboratory approach using the microcomputer for instrumentation, data analysis and control* by Prentice-Hall, Englewood Cliffs, NJ, 1990.
First published by Cambridge University Press 2003

Printed in the United Kingdom at the University Press, Cambridge

Typefaces Times 10.5/14 pt, Helvetica Neue, and Arial *System* LaTeX 2$_\varepsilon$ [TB]

A catalogue record for this book is available from the British Library

Library of Congress Cataloguing in Publication data

Derenzo, Stephen E.
Practical interfacing in the laboratory : using a PC for instrumentation, data analysis, and control / Stephen E. Derenzo.
p. cm.
Rev. ed. of: Interfacing, c1990.
Includes bibliographical references and index.
ISBN 0-521-81527-4
1. Computer interfaces. 2. Microcomputers. 3. Automatic data collection systems.
4. Computer interfaces – Laboratory manuals. 5. Microcomputers – Laboratory manuals.
6. Automatic data collection systems – Laboratory manuals. I. Derenzo, Stephen E.
Interfacing.
TK7887.5 .D42 2002
620′.0028′4 – dc21 2001052859

ISBN 0 521 81527 4 hardback

Contents

1 Digital tools

1.1 Introduction

In the past few years, enormous advances have been made in the cost, power, and ease of use of microcomputers and associated analog and digital circuits. It is now possible, with a relatively small expenditure, to purchase a microcomputer system that will take data, quickly analyze them, and display the results or control a process. This has been made possible by the development of technology that can fabricate millions of transistors, diodes, resistors, capacitors, and conductors on a single silicon **integrated circuit chip**.

Normally, the microcomputer is equipped with a number of standard items: the microprocessor chip and associated circuits, random-access memory chips, removable floppy and cartridge disk drives, magnetic hard disk drives, optical disk drives, keyboards, video display screens, serial interfaces, printers, and x–y entry devices such as the mouse, trackball, joystick, bitpad, and touch-sensitive display screen. However, data acquisition and control require additional components, such as digital and analog input/output (I/O) ports, and counters/timers. Analog input ports contain analog multiplexers, sample-and-hold (S/H) amplifiers, and analog-to-digital (A/D) converters. Analog output ports contain digital-to-analog (D/A) converters.

Even for designs requiring only a microprocessor and a few additional circuits, there are considerable advantages to using the resources of the microcomputer during the development stage. These include program code editors and compilers, an operating system for the storage and manipulation of code and data files, and ample random-access memory.

In this chapter, we discuss digital interfacing concepts used in microcomputer-based data-acquisition and control systems (Figure 1.1), including parallel and serial input/output ports, handshaking, and digital counters/timers. Analog tools (amplification and filtering) are treated in Chapter 2, digital-to-analog and analog-to-digital conversion and sampling in Chapter 3, and sensors and actuators in Chapter 4.

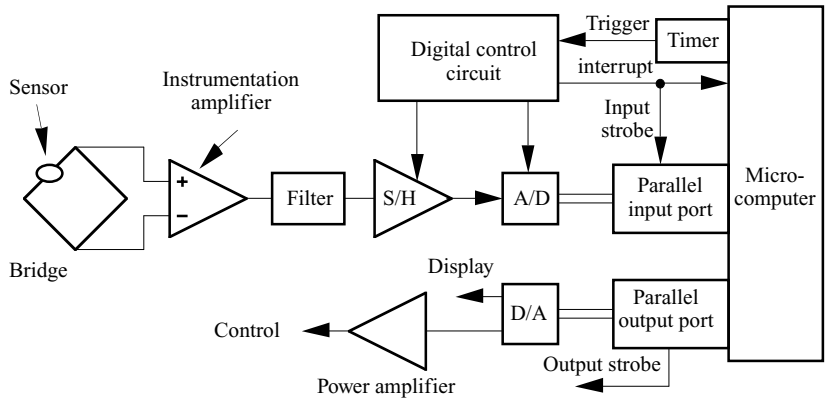

Figure 1.1 A microcomputer system interfaced to sensors and associated analog circuits for data acquisition, analysis, and control.

1.2 The microcomputer

In selecting a system for data acquisition and control, the **microcomputer** itself is a crucial component (Figure 1.2). The microcomputer is sufficiently small to fit on a laboratory bench (or desktop) and yet contains the following components:

1. The **microprocessor** is an integrated circuit that reads program instructions from memory and uses them to determine the sequence of actions that it performs. It is connected to memory and peripheral circuits by an address bus, a data bus, and control lines.

   These actions include reading data and instructions from memory, performing calculations, executing different instructions depending on the outcome of a calculation, printing data, and transferring data to and from peripheral devices such as hard disks. Microprocessors vary greatly in their speed and data-handling capability.
2. **Random-access memory (RAM)** usually consists of high-speed semiconductor memory chips that are used to store and retrieve program instructions and data. The highest data-acquisition speeds are achieved when external data are read directly into RAM, so the size of the RAM places a limit on the number of data values that can be sampled rapidly.
3. Common user interface devices are the keyboard, video display screen, printer, mouse, joystick, and trackball. Some systems provide voice input and synthesized speech output. The IEEE-1284 interface standard includes the standard parallel printer (SPP) port as well as other enhancements. The universal serial bus (USB) is the current standard for keyboards and pointing devices. For higher

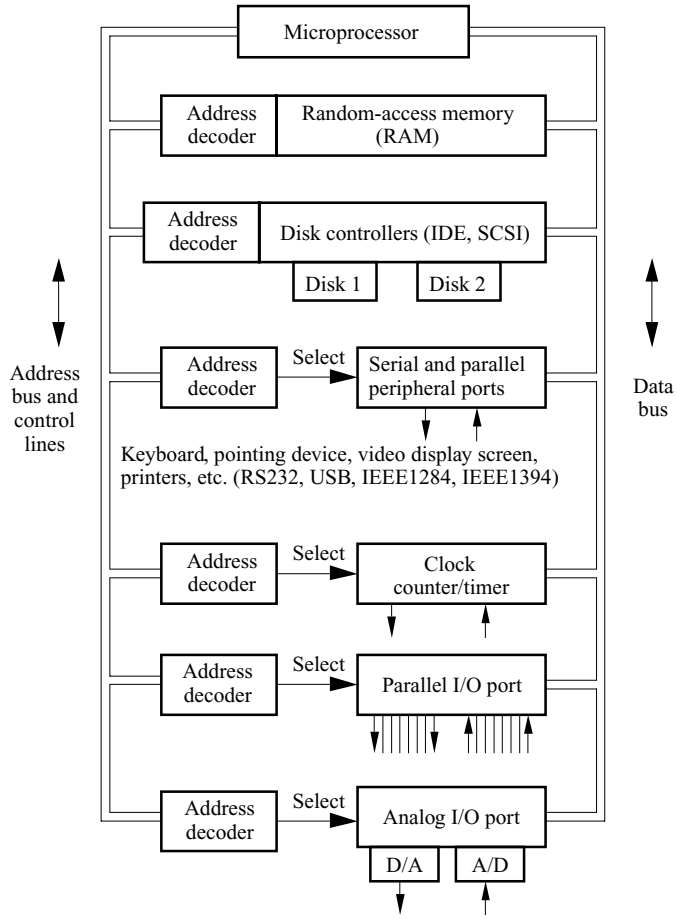

Figure 1.2 The microcomputer consists of a microprocessor that communicates with memory and input/output devices by address and data buses.

speed transfers (external hard drives, digital camcorders, HDTV), the IEEE-1394 standard (FireWire or i.Link) has recently been introduced.

4. **Magnetic disk memory** is used for the long-term storage of programs and data, and consists of one or more flat circular plates coated with a magnetic surface. Magnetic disk capacities range from 500 kbytes to 2 Mbytes for small removable floppy disks and from 1 to 20 Gbytes or more for hard disks. Access time consists of a fixed delay of tens of milliseconds (for the read/write head to locate the desired track) and a transfer time of typically 1 μs per 16-bit word.
5. **Optical disk memory** includes the CD-ROM and the DVD-ROM disks. The **CD-ROM** (compact disk-read-only memory) and DVD-ROM (digital versatile disk) drives use optical storage and retrieval technology that was developed for the music and entertainment industry. The capacity of the CD-ROM is over 600 Mbytes and about ten times larger for the DVD-ROM. Both are 12 cm in

diameter. Microcomputers and workstations are commonly shipped with a CD-ROM containing a back-up copy of the system software and on-line documentation, eliminating many floppy disks and thousands of pages of paper. CD-W (write once) and CD-RW (rewritable) and DVD-RAM (random-access) technology allows information to be written onto these disks.

6. The operating system permits the user to manipulate program and data files and supports a high-level compiled programming language (FORTRAN, Pascal, C, compiled BASIC, etc.).
7. A **compiler's** function is to translate a high-level language into microprocessor code that is able to:
   (i) perform numerical computations and conditional branching,
   (ii) communicate directly with a data-acquisition and control board or parallel I/O port (see below),
   (iii) read and write files to the disk.
   Additional useful features include:
   (i) a full range of scientific functions (sine, cosine, exp, log, etc.), and the ability to compute using floating-point representation, which can handle very small and very large numbers (for example, 80-bit extended precision can handle numbers from $\pm 10^{-4,932}$ to $\pm 10^{+4,932}$ with a precision of 19 decimal digits);
   (ii) the ability to write functions in assembly code for greater speed during data acquisition (some compilers permit intermixed assembly code and higher-level code);
   (iii) a built-in **editor** that displays lines causing compilation errors and permits immediate correction;
   (iv) a single command that compiles all changed program modules, links all necessary modules, and runs the result.
8. An analog input/output port (also called a data-acquisition and control circuit), with the required speed and number of A/D and D/A conversion circuits.
9. A parallel input/output port with sufficient speed, if item 8 is not available. In this case, it becomes necessary to design and build a data-acquisition circuit for connection to the parallel I/O port (this is demonstrated in Laboratory Exercise 9).
10. A counter/timer that can determine elapsed times to an accuracy of typically 1 μs, count input pulses, or produce output pulses of any desired width and period with an accuracy of typically 1 μs.

The microprocessor communicates with the other components of the microcomputer by an address bus, a data bus, and a number of control lines (Figure 1.2). The **address bus** allows the microprocessor to select particular components individually. Each component has a unique assigned **address** whether it is a RAM location, an I/O port register, or other peripheral circuit. An **address decoder** produces a **select** pulse whenever the assigned address appears on the address bus. For example, a 16-Mbit RAM chip has an internal address decoder with 24 input lines and 16 million select lines, one for each

memory bit that can be selected. The **data bus** is used to transmit data words to and from the microprocessor and its associated circuits.

Note: In some systems, memory locations and external devices are distinguished from each other by a special control bit. In others, a large block of memory address space is reserved for external devices.

Since many devices are attached to the data and address buses and at any instant only one can be sending data, control lines are used to indicate when the bus is busy, when a sending device requests use of the bus, when use is granted, etc. These details are beyond the scope of this book and are mentioned to outline the organization of the microcomputer.

Laboratory Exercise 1 is designed to familiarize the reader with the particular editor and compiler that will be used for the rest of the exercises as well as review 2's complement, hexadecimal, real, and integer interpretations of binary numbers.

1.3 Number systems

1.3.1 Binary number representations

Binary numbers can be interpreted in a variety of ways. Table 1.1 shows the interpretation of 8-bit binary patterns as unsigned decimal, hexadecimal, Gray, and 2's complement numbers. The 16-bit and 32-bit numbers are logical extensions.

A/D converters and counters/timers produce binary bit patterns that are to be interpreted as unsigned numbers. The binary sequence runs continuously from all bits = 0 to all bits = 1 and the leftmost bit is the most significant bit (**MSB)**.

Angle and position encoders usually produce **Gray code** that runs from all bits = 0 to all bits = 1, but the binary sequence is not continuous because it has the special property that advancing from one number to the next involves changing the state of only one bit. Gray code is described further in the following section.

Binary numbers can also be represented in **hexadecimal** form (base 16) for efficient notation. Note that each 8-bit byte can be represented as two hexadecimal digits. Octal (base 8) is less frequently used.

Binary bit patterns can also be interpreted as signed numbers, to include negative numbers (<0) as well as 0 and positive numbers (>0). Some computers use signed binary representation, where the leftmost bit represents the sign. While this representation is closer to that of the printed page, it is seldom used in computers because arithmetic operations take longer due to the need to process the sign bit.

Most microcomputers use **2's complement representation** to deal more efficiently with negative and positive numbers. In 2's complement representation, the sign of a number is changed by complementing (reversing) all its bits and then adding one. This is called the **2's complement operation**. By using this operation, the subtraction process

Table 1.1 *Interpretations of 8-bit binary numbers*

Binary	Unsigned decimal	Hexadecimal	Gray	2's complement
0000 0000	0	00	0	0
0000 0001	1	01	1	1
0000 0010	2	02	3	2
0000 0011	3	03	2	3
0000 0100	4	04	7	4
0000 0101	5	05	6	5
0000 0110	6	06	4	6
0000 0111	7	07	5	7
0000 1000	8	08	15	8
0000 1001	9	09	14	9
0000 1010	10	0A	12	10
0000 1011	11	0B	13	11
0000 1100	12	0C	8	12
0000 1101	13	0D	9	13
0000 1110	14	0E	11	14
0000 1111	15	0F	10	15
0001 0000	16	10	31	16
...	...	...	...	...
0111 1110	126	7E	65	126
0111 1111	127	7F	64	127
1000 0000	128	80	192	−128
1000 0001	129	81	193	−127
...	...	...	...	...
1111 1110	254	FE	129	−2
1111 1111	255	FF	128	−1

$a - b$ can be performed by adding a to the 2's complement of b. For an 8-bit number, 2 is represented as binary 0000 0010 (hexadecimal 02) and −2 is represented as binary 1111 1110 (hexadecimal FE). For example, $5 - 2 = 3$ in 2's complement arithmetic is:

5	0000 0101	simply add, but ignore
−2	1111 1110	the most significant carry bit
3	0000 0011	

Note that in 2's complement notation, positive numbers have their MSB $= 0$ and negative numbers have their MSB $= 1$.

Warning: sign extension

As demonstrated in Laboratory Exercise 1, if the MSB of a number is zero, then conversion from 8 to 16 bits or from 16 to 32 bits occurs as expected. However, if the

Table 1.2 *Typical variable types, storage, and ranges of values*

Type	No. bits	Decimal digits	Range
Char	8		−128 to +127
Unsigned char*	8		0 to 255
Short	16		−32,768 to +32,767
Unsigned integer	16		0 to 65,535
Int and long	32		−2,147,483,648 to 2,147,483,647
Unsigned long*	32		0 to 4,294,967,295
Float	32	7	$\pm 1.2 \times 10^{-38}$ to $\pm 3.4 \times 10^{+38}$
Double	64	14	$\pm 2.3 \times 10^{-308}$ to $\pm 1.7 \times 10^{+308}$
Extended*	80	19	$\pm 1.7 \times 10^{-4932}$ to $\pm 1.1 \times 10^{+4932}$

*Standard in ANSI C, but not available on all C or Pascal compilers.

MSB is one, then the leftmost additional bits of the longer number will be filled with ones (**sign extension**). In this way, the converted number will have the same numerical value in 2's complement representation. For example, when transferred from char to int, 35 becomes 0035 and 8A becomes FF8A. Thus, if unsigned numbers are read from a counter/timer or A/D converter in blocks of eight bits, some precautions are necessary before they can be packed into 16- or 32-bit numbers. There are two approaches:

1. Mask the left half of the number with zeros (see Appendix C).
2. Declare all relevant variables to be "unsigned."

Table 1.2 shows the typical internal representations available on microcomputers. They are also explored in Laboratory Exercise 1. Each program variable is declared to be one of these types. The float, double, and extended have 8-, 11-, and 15-bit exponents and 23, 52, and 63 bits of precision, which correspond to 7, 15, and 19 decimal digits of precision, respectively.

1.3.2 Gray code

Gray code is used extensively in external devices such as digital position encoders because the transition from any number to the next involves a change of only one bit (Table 1.3). If binary code were used, erroneous values could result when more than one bit changed, since it is not possible to guarantee that all bits change simultaneously from one number to the next.

The exclusive-OR circuits shown in Figure 1.3 convert numbers from Gray code to binary code and from binary code to Gray code. See the following section for a review of the AND, inclusive-OR, and exclusive-OR logic circuits. It will be noted on the left-hand side of Figure 1.3 that bit 1, for example, cannot be determined until bit 2 is known, and bit 2 cannot be determined until bit 3 is known, etc. Thus the output is valid only after N gate propagation times. A "valid data" signal can be derived by connecting

Table 1.3 *Binary and Gray codes and their decimal equivalents*

Decimal	Binary	Gray	Decimal	Binary	Gray
0	00000	00000	16	10000	11000
1	00001	00001	17	10001	11001
2	00010	00011	18	10010	11011
3	00011	00010	19	10011	11010
4	00100	00110	20	10100	11110
5	00101	00111	21	10101	11111
6	00110	00101	22	10110	11101
7	00111	00100	23	10111	11100
8	01000	01100	24	11000	10100
9	01001	01101	25	11001	10101
10	01010	01111	26	11010	10111
11	01011	01110	27	11011	10110
12	01100	01010	28	11100	10010
13	01101	01011	29	11101	10011
14	01110	01001	30	11110	10001
15	01111	01000	31	11111	10000

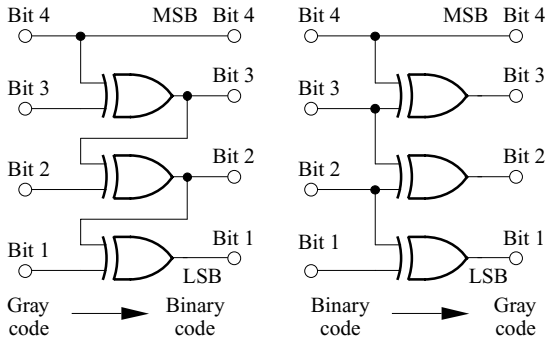

Figure 1.3 Circuits used to convert Gray code to binary and binary code to Gray code. Four bits are shown. The logic elements shown perform the exclusive OR, which has an output logic state that equals one only if the input logic states differ.

all input bits to an inclusive-OR circuit that is used as the input to a pulse generator. The output is read at the trailing edge of the pulse. Alternatively, a table lookup from computer memory or read-only memory (ROM) can be used to convert between Gray and binary codes.

1.4 Digital building blocks

This section describes the fundamental building blocks used to connect to a multiple-output bus, sample and store a logic state at a well-defined time, generate pulses,

Table 1.4 *Logic voltage ranges for TTL and ECL circuit families*

	TTL (V)	ECL (V)
Power supplies	0, +5 (±5%)	0, −5.2 (±5%)
Allowed "0" input range	−0.5 to +0.8	−5.0 to −1.4
Ambiguous input range	+0.8 to +2.0	−1.4 to −1.1
Allowed "1" input range	+2.0 to +5.5	−1.1 to +0.0
Nominal logic "0" output	+0.2	−1.75
Nominal logic "1" output	+3.2	−0.90
Allowed "0" output range	+0.0 to +0.4	−1.85 to −1.65
Ambiguous output range	+0.4 to +2.4	−1.65 to −0.96
Allowed "1" output range	+2.4 to +5.0	−0.96 to −0.81
Typical pulse risetime (10–90%)	10 ns*	1.5 ns†

*Low-power Schottky TTL.
†ECL 10,000.

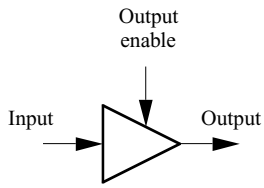

Figure 1.4 Tri-state buffer (see Table 1.5 for the function table and Figure 1.5 for a typical timing diagram).

and perform logical tests (AND, OR, etc.) of logic states. Table 1.4 lists the ranges of external voltages for the two most commonly used families of logic circuits, TTL (transistor–transistor logic) and ECL (emitter-coupled logic).

Note 1: To allow for voltage drop along conductors, the requirements for output are more stringent than for input.

Note 2: For both TTL and ECL, a logic 1 is always more positive in voltage than a logic 0.

1.4.1 Tri-state buffer

The **tri-state buffer** has three output states: asserted high, asserted low, and high impedance. In the high-impedance state, the output neither loads nor drives any circuit connected to it. This device has the usual logic input, but also has an additional enable input that determines whether the output follows the input or is put in the high-impedance state. The tri-state buffer is an essential component when several different outputs must be connected to form a common bus. See Figure 1.4 for the circuit schematic, Figure 1.5 for a typical timing diagram, and Table 1.5 for the function table.

Table 1.5 *Function table for tri-state buffer*

Input	Output enable	Tri-state output
H	L	H
L	L	L
X*	H	High impedance

*X = don't care.

Table 1.6 *Function table for edge-triggered* D*-type flip-flop*

Data *D*	Clock *C*	Flip-Flop output *Q*
H	↑†	H
L	↑	L
X*	H or ↓§ or L	Previous state

*X = don't care.
†↑ = low-to-high edge.
§↓ = high-to-low edge.

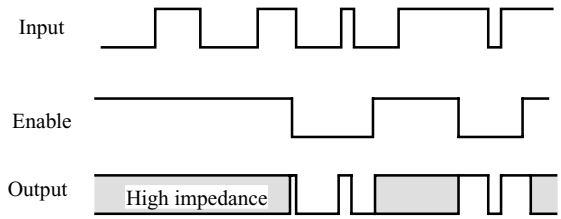

Figure 1.5 Typical timing diagram for the tri-state buffer (see Table 1.5 for the function table).

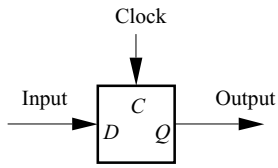

Figure 1.6 Edge-triggered D-type flip-flop (see Table 1.6 for the function table and Figure 1.7 for a typical timing diagram).

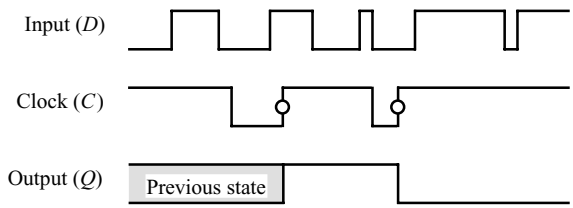

Figure 1.7 Typical timing diagram for the edge-triggered D-type flip-flop (see Table 1.6 for the function table).

Table 1.7 *Function table for transparent latch*

Data D	Gate G	Latch output Q
H	H	H
L	H	L
X*	L	Previous state

*X = don't care.

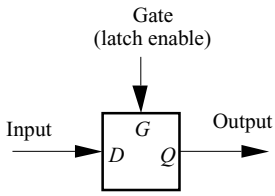

Figure 1.8 Transparent latch (see Figure 1.9 for a typical timing diagram and Table 1.7 for the function table).

1.4.2 Edge-triggered D-type flip-flop

The basic element in the parallel output port is the edge-triggered **D-type flip-flop**, which differs somewhat from the simple flip-flop that can be switched between two logic states. The edge-triggered D-type flip-flop has two inputs, a data input (D) and a clock input (C) (Figure 1.6). The output (Q) is set equal to the logic state of the input (D) during the clock (C) low-to-high edge. At all other times, the state of Q does not change even if D changes. See Table 1.6 for the function table and Figure 1.7 for a typical timing diagram.

Frequently, the outputs have tri-state buffers (see Figures 1.4 and 1.5) so that several outputs can be connected. The 74LS374 tri-state octal D-type edge-triggered flip-flop is such an example. The state of Q is only asserted at the output line when the "output-enable" line is asserted. When the output-enable line is not asserted, the output is in a high-impedance state that neither drives nor loads any other circuit connected to the output. Whenever two or more outputs are connected to a common line called a **bus**, they must all have tri-state outputs.

1.4.3 Transparent latch

The transparent latch (Figure 1.8) is similar to the edge-triggered D-type flip-flop, except that the output is equal to the input the entire time that the latch enable is asserted. See Table 1.7 for the function table and Figure 1.9 for a typical timing diagram.

Frequently, the outputs have tri-state buffers (see Figures 1.4 and 1.5) so that several outputs can be connected. The 74LS373 tri-state octal D-type transparent latch is such an example.

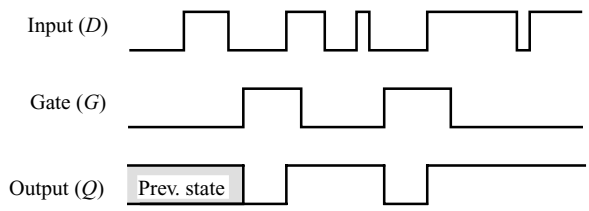

Figure 1.9 Typical timing diagram for transparent latch.

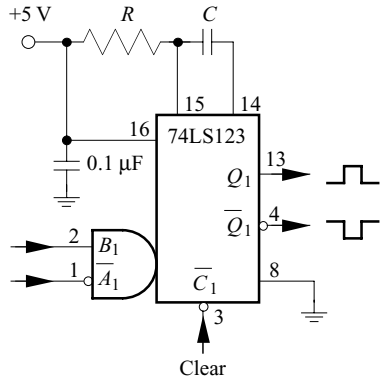

Figure 1.10 74LS123 dual retriggerable one-shot. Pin numbers correspond to section 1. Pulse width is determined by the external resistor and capacitor values.

1.4.4 One-shot

The one-shot produces output pulses of fixed width, where the width is determined by the values of an external resistor R and capacitor C. Figure 1.10 shows one of two sections of the 74LS123 dual retriggerable one-shot and its external components, and Table 1.8 shows the function table. The pulse width W can be estimated using the equation:

$$W = 0.37R(C + 22\text{ pF})$$

The actual pulse width may differ by typically 20% due to component variations. To produce a more precise pulse width, it is common to use a variable resistor that is adjusted while observing the pulses on an oscilloscope.

The retriggerable one-shot has the property that if a new trigger is received while an output pulse is in progress, the output pulse is extended from that time by an amount W. The non-retriggerable one-shot ignores input triggers while an output pulse is in progress.

1.4.5 AND, OR, exclusive-OR gates

Integrated circuits are readily available to perform standard logic operations such as AND, OR, and exclusive OR (Figure 1.11). (The OR is also called the inclusive OR

Table 1.8 *Function table for 74LS123 retriggerable one-shot*

Clear C	Input A	Input B	Output Q	Output $\overline{Q}$
L	X	X	L	H
X*	H	X	L	H
X	X	L	L	H
H	L	↑	⎍	⎏
H	↓§	H	⎍	⎏
↑†	L	H	⎍	⎏

*X = don't care.
†↑ = Low-to-high transition.
§↓ = High-to-low transition.

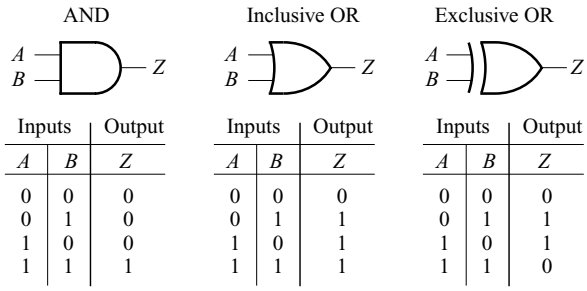

Figure 1.11 AND, inclusive-OR, and exclusive-OR logic gates.

to distinguish it from the exclusive OR.) The AND circuit is used to detect when two logic levels are both high, the inclusive-OR circuit is used to detect when either of two logic levels is high, and the exclusive-OR circuit is used to detect when two logic levels differ. When a circle is shown at the output, the output is complemented ($\overline{Z}$ rather than Z) and the device is called a NAND or NOR gate.

1.4.6 Set/reset latch

This circuit has two digital inputs that allow the output to be set or reset. The TTL 74LS279 contains four set/reset latches and each has the logic table shown in Table 1.9. It is used to convert pulses to stable logic levels. Specifically, if both inputs are initially H (their inactive level), an L pulse on $\overline{S}$ will set Q to H and an L pulse on $\overline{R}$ will reset Q to L.

1.5 Digital counters/timers

The digital counter/timer is a circuit that can count pulses that occur at arbitrary times, measure time by counting clock pulses, or produce pulses uniformly spaced in time. Normally, when executing a program, the microcomputer must perform other tasks that

Table 1.9 *Function table for 74LS279 quad set/reset latch*

Input $\overline{S}$	Input $\overline{R}$	Output Q
L	L	H*
L	H	H
H	L	L
H	H	Previous value

*May not persist when both inputs are set H.

make it impossible for the program to keep track of absolute time. Moreover, execution speed depends on the clock frequency of the particular computer used. A variety of integrated-circuit chips have been developed that can constantly keep track of time (and even of the date) while the microcomputer is busy with other tasks (or even turned off). For example, most microcomputers have battery-powered circuits that keep track of the date and time (to the nearest second) and this information is recorded whenever a disk file is created or changed. For data acquisition involving periodic sampling, a more precise, dedicated clock circuit is needed that can be read by the program or produce a series of external pulses evenly spaced in time. The use of a hardware counter/timer is explored in Laboratory Exercise 2, where human reaction time is measured. Sections 1.5.1–1.5.3 describe typical applications of digital counter/timer circuits and two of the more popular digital timer chips, the 8253 and the 9513.

1.5.1 Applications of digital counters/timers

Measuring the duration of a pulse

The counter is set to an initial value of 0 and to count up clock pulses when gated on. The pulse whose duration is to be measured is used as the gate pulse. The pulse duration is given by $T_w = N/f_c$, where N is the final value in the counter and f_c is the clock frequency.

Measuring the time difference between two events

The first event sets a logic level and the second event resets the logic level. The duration of the resulting pulse is then measured using the method just described. One well-known application is the timing of Olympic races to an accuracy of 1 ms.

Generating a pulse of precise duration

The counter is loaded with a number N and set to count down once per clock pulse. The output is high during counting and low after zero is reached. The duration of the pulse is given by $T_w = N/f_c$, where f_c is the clock frequency.

Measuring an average pulse frequency

One counter is used to produce a pulse of precise duration, using the method just described. A second counter counts pulses when gated on by the first counter. If the

pulse duration is T_w and the count in the second counter is M, then the average pulse frequency is given by $f_p = M/T_w$.

Producing pulses uniformly spaced in time

The counter is loaded with a number N and set to count down once per clock pulse. When its contents reaches zero, it produces an external pulse, reloads from a load register, and then resumes counting. The frequency of the resulting pulses is given by $f_p = f_{\text{clock}}/N$.

1.5.2 The 8253 programmable interval timer

This integrated-circuit chip (manufactured by Intel and others) has three 16-bit down counters that can be used to count clock pulses and can be written and read under program control. It has a number of functions that permit it to act as a pulse generator, a digital one-shot, or a digital square-wave generator. These functions are selected by writing to a control register. At typical 1-MHz clock rates, there are two issues:

1. It is not possible to read a rapidly changing accumulator directly, and it is necessary to latch the accumulator into a buffer (temporary storage) register. When the latch command is given, circuits on the chip transfer the contents of the specified counter to a buffer register that can be read later. If the counter value is in the process of changing, the circuits wait for the value to become stable before latching. Note that the read command reads the buffer register (not the counter itself) and, as a result, the value read is the counter value at the instant the latch command was given, not the counter value at the time of the read (Figure 1.12).
2. A 16-bit accumulator will overflow in 16 ms or less, and for counting longer periods, it is necessary to hardwire two accumulators in sequence. Since the two accumulators

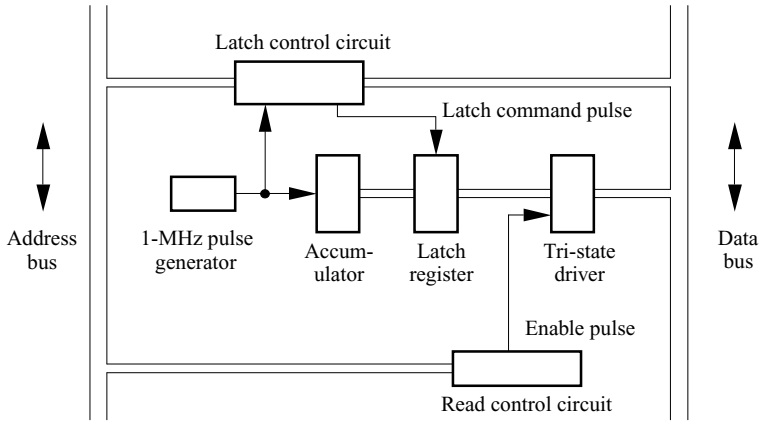

Figure 1.12 Circuits for accumulating 1-MHz pulses, for transferring a valid accumulator value to a latch register under computer control, and for reading the value into memory.

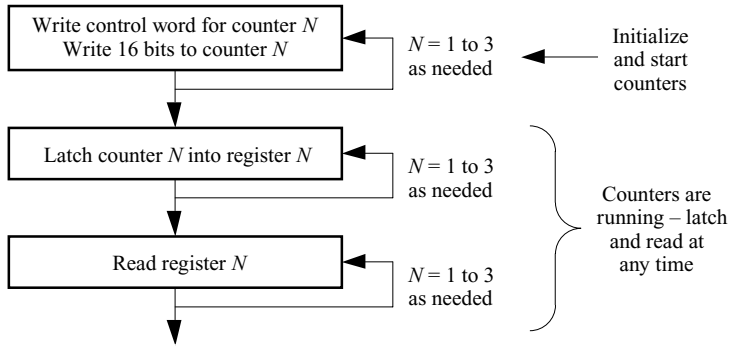

Figure 1.13 Sequence of operations for initializing, loading, latching, and reading the 8253 counter/timer.

must be latched by different instructions, an ambiguity arises whenever the faster accumulator passes through zero. The chip does not have circuits to handle this problem and the simplest solution is to reread the slower accumulator. (See the following warning about cascading counter/timer chips.)

The initialization, loading, latching, and reading sequence is shown in Figure 1.13.

1.5.3 The AM9513 system timing controller

This integrated-circuit chip (manufactured by Advanced Micro Devices) has many more features than the 8253 and requires more program steps to initialize (Figure 1.14). It has five independent 16-bit counters, a 1-MHz clock, and on-chip subscalers to permit divide-by-*N* counting for slower rates and longer time ranges.

Under program control, the input of any counter can be connected to any subscaler, the overflow output of another counter, or to an external input line. Similarly, the overflow output of any counter can be connected to the input of another counter or to an external output line.

The five counters can be latched in any combination with a single instruction. However, whenever two or more counters are cascaded, the ambiguity mentioned before is still present (although less frequent), and still requires rereading the slower counter. This interval timer is used in the Metra Byte parallel I/O board and the National Instruments analog data-acquisition board (among others).

Warning: cascading counter/timer chips

When two counter/timer chips are cascaded, the more rapidly moving counter (say, counter 1) receives input pulses from the system clock and increments until it reaches FFFF (for a 16-bit counter). As it transitions to 0000, it sends a carry pulse that increments the less rapidly moving counter (counter 2; see Figure 1.15).

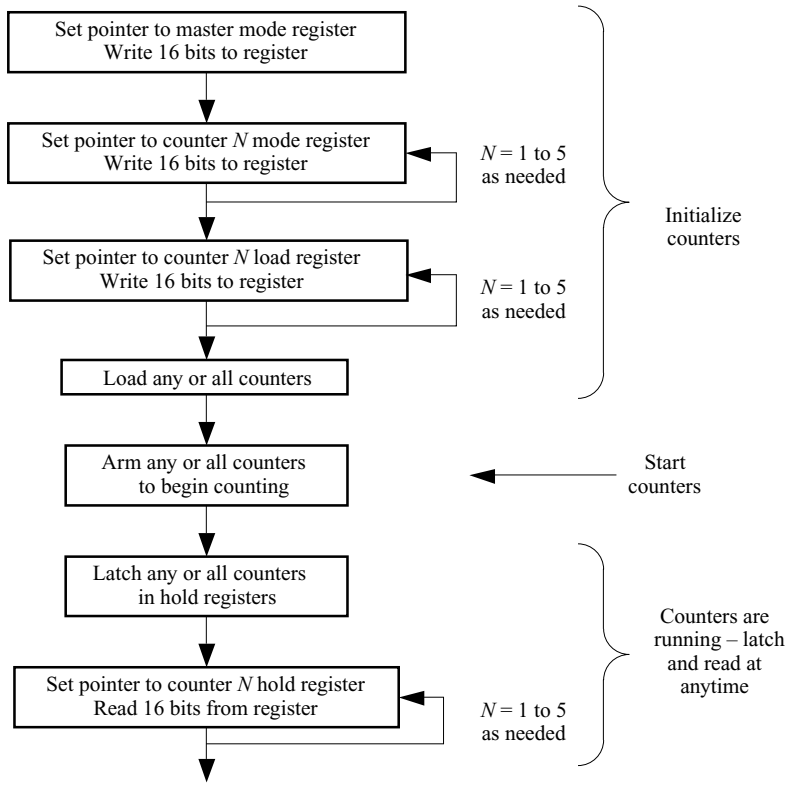

Figure 1.14 Sequence of operations for initializing, loading, latching, and reading the 9513 counter/timer. The counters may be latched and read repeatedly at anytime.

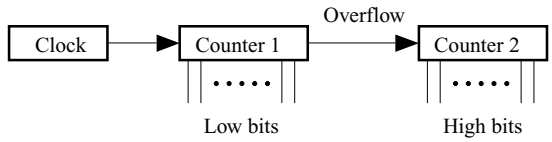

Figure 1.15 Timer consisting of a clock and two cascaded counters.

Unfortunately, it is not possible to guarantee that these two events happen simultaneously, and erroneous timing information can be read occasionally. For example, suppose that counter 2 reads A73D and counter 1 reads FFFF (Table 1.10). After counter 1 receives its next clock pulse, we would hope that counter 2 and counter 1 would change simultaneously and read A73E 0000, but if one changes before the other, and we latch the counters during this very brief period, we will get either A73E FFFF or A73D 0000. Both are in error by over 65,000 counts! Moreover, it is not possible to latch the outputs of two different counter circuits simultaneously, even if they could change simultaneously.

Table 1.10 *Correct and incorrect values of cascaded counters*

	Counter 1	Counter 2
Correct simultaneous increment of counters 1 and 2 (perfect timing)	FFFF	A73D
	0000	A73E
Incorrect: 1 increments before 2 (65,536 too low) →	FFFF	A73D
	0000	A73D
	0000	A73E
Incorrect: 2 increments before 1 (65,536 too high) →	FFFF	A73D
	FFFF	A73E
	0000	A73E

The general solution to this problem is to use the following steps:

1. Latch and read both counters.
2. If counter 1 is "near" 0000, go back to step 1.

The nearness condition is determined by the range of counter 1 values during which counter 2 could be changing.

For the 9513 in 1-MHz increment mode and "simultaneous" counter latching, counter 1 will increment at most 2 μs before its overflow pulse can increment counter 2. In this case the nearness condition in step 2 is "if counter 1 < 2."

For the 8253, where the counters decrement at 1 MHz and are latched by separate program statements that are separated by typically 20 μs, counter 1 would be latched immediately before counter 2, and a safe nearness condition in step 2 would be "if counter 1 > FF00."

1.6 Parallel and serial input/output ports

The **parallel input/output (I/O) port** allows the microcomputer to communicate directly with logic voltages in the external digital world and handles most of the problems of control and synchronization with the microprocessor address and data buses. The most convenient is the bi-directional port, which has separate input and output lines, simplifying the connection to external devices. Since all bits are transferred in parallel (at the same time), it is generally faster than the serial port. Laboratory Exercise 3 involves reading switches and writing to lights using a parallel port. Moreover, A/D and D/A converters naturally deal with parallel digital information and can be interfaced directly to a parallel I/O port, as demonstrated in Laboratory Exercises 8 and 9.

The parallel I/O port usually includes the following addressable internal registers:

1. **Data registers** that hold input data until the program can read them and hold output data while needed by the external circuit.

2. A **control register** that allows the program to write ones and zeros to control the mode of operation of the port or the logic state of external lines. These lines are typically used to notify an external circuit that the program: (i) has new output, (ii) has read new input, or (iii) is ready to accept new input.
3. A **status register** that can be read by the program to determine the status of the data register or the logic state of external lines. Various bits would be set when an external circuit: (i) has asserted and latched new data on the input port (and are ready to be read by the program), (ii) has read the contents of the output port, or (iii) is ready to read new data from the output port.

Some commercial parallel I/O ports have only data registers and no handshaking registers, but it is possible to assign some of the data bits to be used by the program to communicate with external circuits. This is described in the following sections on parallel input and output handshaking.

Most parallel I/O ports have both input and output data lines, whose functions cannot be changed (the bi-directional port), while others permit each data bit to be either input or output, as specified by the contents of a special control register.

The **serial I/O port** also has addresses for setting up the communication protocol and then can transfer data serially in time using only one input and one output line. The advantage over the parallel port is that existing circuits (specifically telephone communication lines) can transmit serial data over long distances, even to other continents. For connection to nearby peripheral devices the older RS232 serial port is being replaced by much faster USB and IEEE 1394 serial ports. (See Section 1.9 for more details.)

1.6.1 Handshaking considerations

Handshaking consists of the communication procedures used to ensure that both sender and receiver are ready for data transmission, that the sender tells the receiver when data are ready, and the receiver tells the sender that the data have been taken.

Parallel data can be correctly read only when all bits are stable. Handshaking is essential and allows the sender to signal the receiver when new data are ready and stable.

The following steps describe how handshaking can be implemented using two handshaking lines "ready for data" and "data available" in addition to the data lines. They can be used for data transmission in either direction between any combination of computers and external circuits. Between transactions, "ready for data" and "data available" are FALSE.

1. When the receiver is ready for new data, it sets "ready for data" TRUE.
2. The sender detects "ready for data" TRUE.
3. (a) If the receiver initiates the data transfers, the sender asserts the requested data as

soon as possible after step 2. ("Asserts" means setting voltages on the data lines that correspond to the 0s and 1s of the data.)

(b) If the sender initiates the data transfers, the sender can assert data anytime after step 2.

4. After the voltages on the data lines have settled, the sender sets "data available" TRUE. (It is essential that the data are valid *before* "data available" is set true.)
5. The receiver detects "data available" TRUE and reads the data.
6. The receiver sets "ready for data" FALSE. (At this point the receiver is not ready for data because it needs to do something with the data it just read.)
7. The sender detects "ready for data" FALSE and sets "data available" FALSE. (At this point the sender is relieved of the responsibility of asserting the data.)

Handshaking is required for both serial and parallel data when a series of data values must be transmitted faithfully and the receiver or the sender have an unpredictable response time or are transferring data at unpredictable times (asynchronous communication).

Handshaking is generally not necessary when the sender continually produces data (such as temperature measurements), and the receiver can tolerate occasional erroneous values that arise during bit changes. To avoid this problem, digital position encoders (see Chapter 4) use Gray code which has the property that neighboring values differ by only one bit (see Table 1.1).

Design tip

If a circuit asserts digital data for a brief period and the computer may not be able to read the data promptly:

1. Connect each output line of the circuit to a transparent latch.
2. When the circuit has data, it can store the data on the latch until it can be read by the computer.
3. Handshaking is needed so that the external circuit can inform the program that new data are available and the program can inform the external circuit that the data have been taken.

1.6.2 The parallel output port

The **parallel output port** reads a number from computer memory and converts the bit pattern to logic voltage levels on wires in the world "outside" the computer. Additional control lines and status registers may also be provided so that: (i) the external circuit can tell the computer program that it is ready to receive output data, (ii) the computer program can tell the external circuit that it has data in its internal registers, and (iii) the external circuit can tell the computer program that the output data have been taken.

The basic element in the parallel output port is the **register**, a circuit able to sample, store, and output digital data on command. This is usually achieved by using a set