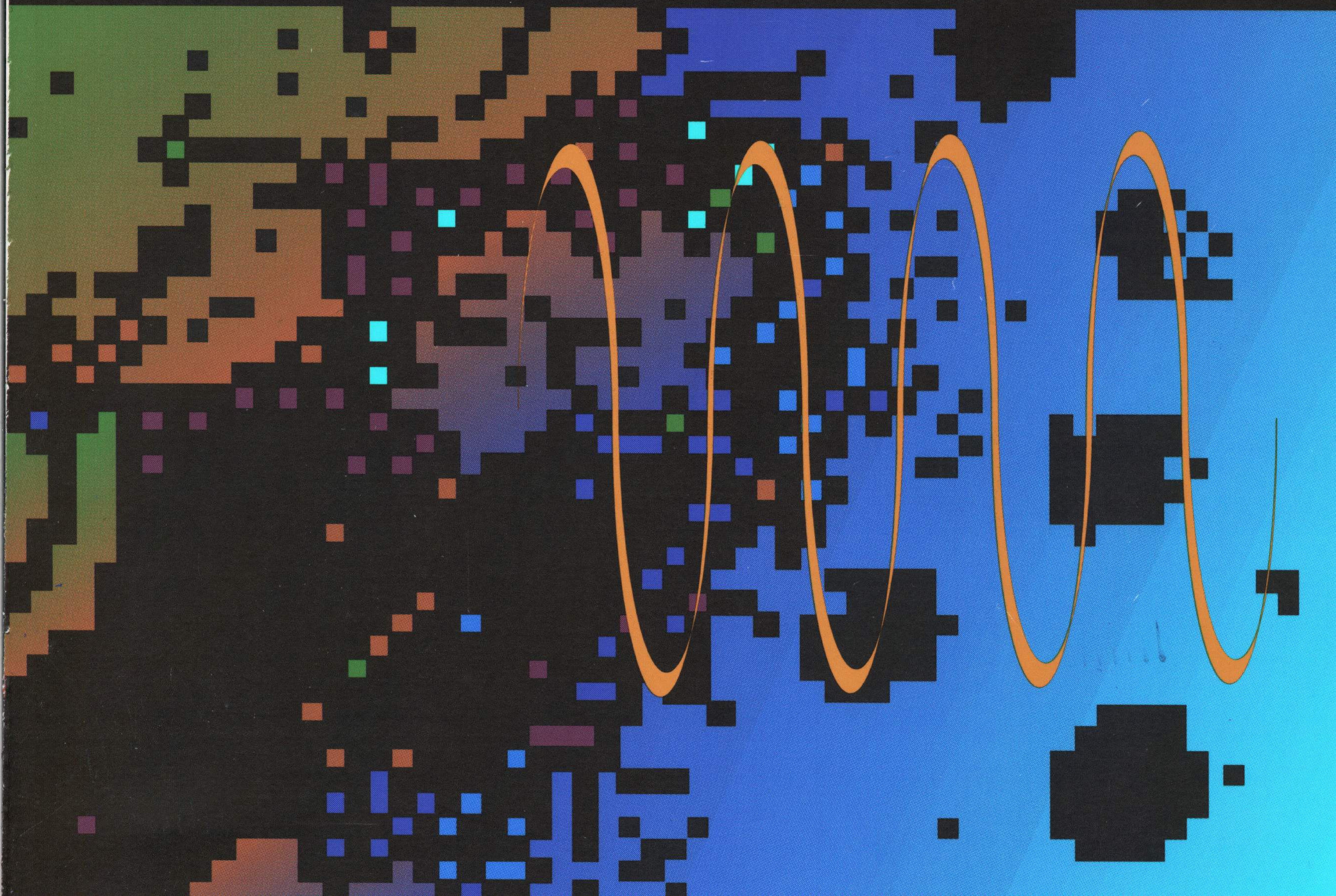
Instrument Communication Handbook

A handbook of IEEE 488, RS-232, SCPI, SCSI, and other interfacing standards

Published by IOtech, Inc. First Edition



Instrument Communication Handbook

IOtech

Copyright Notice

IOtech, Inc. 25971 Cannon Road, Cleveland, Ohio 44146

Printed in the United States of America

Contents

Introduction

1.1 Purpose of IOtech Instrument Communication Handbook

The IOtech Instrument Communication Handbook is intended to be a reference source providing valuable background information in the area of data communications with special emphasis on the IEEE 488 bus. You will find substantial detail on the design and use of the IEEE 488 standard, as well as in-depth discussions of other data communications methods. We hope to give you a working knowledge of IEEE 488, as well as to relate it to other means of data communication.

We welcome any comments, suggestions, or contributions you might have to enhance this publication. You may fax such changes to IOtech's Publications Department at (216) 439-4093, or mail them to us at 25971 Cannon Road, Cleveland, Ohio 44146.

1.2 Overview of IEEE 488, Serial, VXI, and Local Area Network

IEEE 488 refers to the Institute of Electrical and Electronics Engineers (IEEE) Standard number 488. This standard was first established in 1978, 13 years after Hewlett-Packard (HP) of Palo Alto, California, began work to enable its broad range of instruments to communicate with one another and with "host" computers.

The growing complexity of test instruments and test protocols, as well as the emerging need to capture, store, and analyze data, had begun to create a need for coordination among instruments and for data communication to a central storage device. In response, Hewlett-Packard developed a parallel communications method that provided higher transmission rates than existing methods. This new method, originally termed the Hewlett-Packard Interface Bus (HP-IB), was adopted by HP for communications between its computers and a variety of peripheral devices, including plotters, printers, and disk drives. Today, HP-IB is still used by HP for these purposes.

At the time of its development, IEEE 488 was particularly well-suited for instrument applications when compared with the alternatives. In essence, IEEE 488 comprises a "bus on a cable," providing both a parallel data transfer path on eight lines as well as eight dedicated control lines. Given the demands of the times, its nominal 1 Mbyte/sec maximum data transfer rate seemed quite adequate; even today IEEE 488 is sufficiently powerful for many highly sophisticated and demanding applications.

However, IEEE 488, as originally defined by the 1978 standard, left some ambiguities in the specifics of controller-instrument interaction and communication. While these open issues were likely intended to give instrument and controller designers some latitude, the result was confusion and compatibility problems among instruments from different manufacturers.

The IEEE 488 committee began working on these issues during the 1980s; its efforts culminated in a new layer to the IEEE 488 standard, IEEE 488.2. The original standard was redesignated IEEE 488.1. Section 2 will discuss "488.2" in detail. For now, suffice it to say that ".2" provides for a minimum set of capabilities among "controllers" and "devices," as well as for more specific content and structure of messages and communications protocols.

As a refinement of the enhancements incorporated in "488.2," HP initiated an effort to explicitly simplify instrument control. The resulting metalanguage, originally termed Test & Measurement Systems Language (TMSL) and now called Standard Commands for Programmable Instruments (SCPI), provides a uniform command syntax to enforce a common command protocol.

IEEE 488.2 is fully backward compatible with IEEE 488.1; the use of a "488.2"-compliant controller affords the ability to use the new protocols available with "488.2" instruments while retaining the ability to communicate with and control "488.1"-compliant instruments and associated vendor idiosyncrasies. Further, SCPI is independent of the data communications method. It is a standard for instrument commands; SCPI commands may be transmitted with IEEE 488, RS-232, VXI, or others. In essence, SCPI is a language, not a communications modality.

Today, IEEE 488 is the most widely recognized and used method for communication among scientific and engineering instruments and enjoys substantial applications in process monitoring and control. Major stand-alone general-purpose instrument vendors such as Gould, Keithley, Fluke, and Tektronix include IEEE 488 interfaces in their products as a matter of course. Many vertical market instrument makers also rely on IEEE 488 for data communications and control.

IEEE 488 controllers are available for a vast array of computer platforms, as well as for platform-independent configurations. Personal computers from a variety of vendors are supported, from the IBM PC/XT/AT and PS/2 and compatibles to the multifaceted Macintosh family. Workstations from Sun, DEC, NeXT, Apollo, and IBM are supported, as are DEC, AT&T, and VMEbus minicomputers. Some of these controllers are plug-in cards; others are protocol converters (e.g., SCSI-to-IEEE 488). All provide at least IEEE 488.1 compliance, and a growing number adhere to the newer IEEE 488.2 standard.

What will the future bring? The IEEE 488 committee is currently working on a 1992 version of the standard. This update will likely be just that – a refining and clarifying of IEEE 488.1 and IEEE 488.2 that corrects the minor errors, omissions, and ambiguities found in the present standard. IEEE 488 is a well-accepted data communications technology with a large installed base. Future developments must respect that installed base to retain backward compatibility. Likewise, the wide acceptance of IEEE 488 means that improvements and enhancements will likely receive wide acclaim. Thus the future of IEEE 488 will be a balance between the stability of an installed base and the potential for widespread adoption of further evolutions.

1.3 Overview of Serial RS-232, RS-422, and RS-423

Serial communication methods were among the earliest means of connecting computers to peripherals and to other computers, and they substantially predate IEEE 488. Serial communication methods are "bit-serial" communication, wherein one bit follows another down the communications line. IEEE 488 is a parallel communication method, wherein the components of an eight-bit word are sent simultaneously, using the eight data lines of the IEEE 488 bus. Thus we may term IEEE 488 either "bit-parallel" or "byte-serial." While the serial standards support up to 57.6 Kbits/sec, the typical speed found with any of the common serial methods in the IBM PC environment is 9.6 Kbits/sec, which is equivalent to 960 bytes/sec (with an eight-bit word, one stop bit and one start bit). This compares with the 1 Mbyte/sec of the IEEE 488 standard. Thus IEEE 488 provides a 1000 fold increase in potential data rate.

The "RS" in RS-232, RS-422, and RS-423 refers to a "Recommended Standard" of the Electronics Industries Association (EIA). Each of these serial standards has its own cabling, pinout, electrical specification, and communications protocol, which are detailed in Chapter 3 of this book.

1.4 Overview of SCSI

The Small Computer Systems Interface, or SCSI, a system level interface standardized by the American National Standards Institute (ANSI), is designed to allow communication between a computer and its peripherals. It is a version of an interface originally developed by Shugart Associates as a controller for the company's hard disk products. With nine data lines (eight bits parallel, plus one parity bit) and nine control lines, SCSI physically resembles IEEE 488. SCSI cables are terminated, contributing to the 1.5 Mbyte/sec (asynchronous) and 5 Mbyte/sec (synchronous) data transfer rates of the original SCSI 1 standard and to the 5 Mbyte/sec (asynchronous) and 10 Mbyte/sec (synchronous) data transfer rates of the newer SCSI 2 standard.

With wider transfer rates, SCSI2 is capable of data transfer rates of up to 40 Mbytes/sec.

SCSI has become a common means of connecting disk drives to personal computers. It is also widely employed as a general peripheral interface for popular workstations and minicomputers. It has not been popular as an instrument data communications method. However, SCSI-to-IEEE 488 converters are a common means of providing IEEE 488 controller capability to these computer systems.

1.5 Overview of VXIbus

VXIbus is the response to the need of the U.S. Air Force for a unified standard for "instruments on a card." VXIbus, or VXI, stands for VMEbus with eXtensions for Instruments. The VME standard was originally developed to permit easy configuration of minicomputers using standard modules in a standard card cage with a standard backplane, or bus. This bus permits high-speed communication among the various elements of the minicomputer system, but lacks any accommodation for analog signals. Further, VME was not designed with instrument control in mind; thus its high speed communication is controlled by low-level register programming.

The original VXI consortium – Hewlett-Packard, Tektronix, Colorado Data Systems, Racal-Dana, and Wavetek – sought to enhance the VME bus with optional extended bus lines and connectors. The base VXI bus – implemented with the so-called "P1" connector – is simply that of its parent, VME; it provides 16-bit data transfer and arbitration and prioritized interrupt lines. VXI systems provide for two additional bus structures, implemented with the VXI bus's "P2" and "P3" connectors. The P2 connector provides increased data transfer bus width plus a 12 line local bus (for instrument-to-instrument communications), TTL and ECL trigger buses, a 10 MHz clock bus, and a power distribution bus. The P3 connector adds more lines to the local bus, ECL trigger bus, and power distribution bus, as well as a 100 MHz clock bus.

In addition to bringing these electrical enhancements to the VME standard, VXI added a higher level of module communication and control with its IEEE 488-like "Message-Based-Devices." These devices communicate using ASCII-based message protocols much as IEEE 488 devices do. To retain the speed of the VME approach, VXI also defines "Register-Based-Devices." These communicate at a low level and thus do not suffer the overhead of ASCII message parsing and interpretation.

The use of the VME-style backplane significantly enhances the data communication rate potential of the VXIbus. The theoretical maximum is 40 Mbytes/sec. However, this speed is only rarely found–or even needed–in practice. One reason is that this speed requires register-based programming, i.e., programming at a low level using binary information. This improves device speed by minimizing control overhead. The trade-off is that instrument control requires a complex series of non-intuitive commands, increasing the difficulty and reducing the flexibility of configuring test systems and protocols.

The alternative is to program VXI devices with the IEEE 488-style "Word Serial Protocol." This method for message-based devices provides the kind of high-level ASCII character control of instruments with which IEEE 488 users are familiar. This sort of control makes system configuration and programming simpler and easier. But, the overhead required to implement this protocol slows data transfer rates down to those of the IEEE 488 bus! Thus the performance improvements promised by VXI require diligence and care to achieve in practice, and may prove to have specific rather than general applicability to the broad world of instrumentation.

1.6 Overview of Local Area Networking (LAN) via Ethernet

Computer local area networks (LANs)–most notably, the Ethernet hardware protocol–are increasingly being used to transfer data from one test area to another, or from a test system or test system complex to a central computer for storage and analysis. Ethernet is now one of the most popular methods for interconnecting heterogeneous computer systems, particularly on campuses and in research labs worldwide. Because Ethernet has become an international LAN standard, many vendors sell Ethernet hardware and software – keeping costs down and innovation up.

Ethernet originated at Xerox's Palo Alto Research Center in response to the need to flexibly interconnect a large number of dissimilar computers to share data and programs. It was made an open standard by Xerox, Intel, and Digital Equipment Corporation in 1980, became an IEEE standard in 1985 (IEEE 802.3 CSMA/CD [Carrier Sense Multiple Access/Collision Detection]) and became an international standard later (ISO/IEC 8802-3: 1990). While there are differences between the Digital/Intel/Xerox (now DIX 2.0) and the IEEE 802.3 standards, they are similar at the hardware level. The major difference is in the definition of the "frame," or bit stream, that constitutes a packet of information being transferred between computers; most Ethernet software complies with DIX 2.0, whereas most Ethernet hardware complies with IEEE 802.3! All of these variations are simply referred to as "Ethernet."

Ethernet's physical connectivity has evolved over the years. The original arrangement is based on a 12.5mm coaxial cable and a bus topology. This topology includes an Ethernet transceiver between the computer and the network proper; the transceiver is connected to the computer via a special cable and is directly connected to the Ethernet cable.

A newer variation of Ethernet uses RG58C/U coax, a thinner, more flexible and less costly alternative to the standard Ethernet coax. This cable permits the direct inclusion of the Ethernet transceiver on computer network cards or on the computer itself, reducing the cost and complexity of network installation. The "down side" is that thinwire Ethernet does not permit as long a network segment (185m) as thickwire Ethernet (500m).

Newer still are twisted-pair (standardized as 10BASE-T) and fiber optic Ethernet cabling, typically employed in Star topologies. Twisted-pair is used to connect computers and workstations to a concentrator or hub, which in turn can be part of a coax or fiber optic network. The concentrators can be no further than 100m from the twisted-pair-connected computer. Fiber optic connections permit the longest point-to-point distance, 2 Km, but are point-to-point only. They cannot be tapped or daisy chained.

All these cabling alternatives provide the same maximum data transfer rate: 10 Mbits/sec. The distinction among them is the maximum distance permitted for each network segment.

Ethernet hardware connectivity can be used by a wide variety of computers and associated networking software. Common among IBM PCs is Novell NetWare. UNIX systems such as Sun and NeXT workstations provide TCP/IP. Digital Equipment provides DECNet for its minicomputers and workstations. All are designed to take advantage of Ethernet's proven performance and reliability.

IEEE 488

2.1 General

The IEEE 488 interface, sometimes called the General Purpose Interface Bus (GPIB), is a general purpose digital interface system that can be used to transfer data between two or more devices and is particularly well suited for interconnecting computers and instruments (Figure 2.1). Some of the key features of the IEEE 488 interface are:

* Up to 15 devices may be connected to one bus
* Total bus length may be up to 20m and the distance between devices may be up to 2m
* Communication is digital (as opposed to analog) and messages are sent one byte (8 bits) at a time
* Message transactions are hardware handshaked
* Data rates may be up to 1 Mbyte/sec

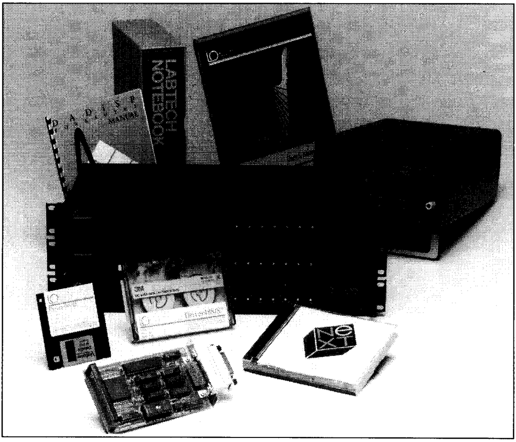


Figure 2.1–An IEEE 488 interface connects devices such as computers and instruments

The specification of the IEEE 488 interface falls into four areas:

1. MECHANICAL — connectors, cabling, and related physical characteristics
2. ELECTRICAL — signal levels, source drive requirements, timing, etc.
3. FUNCTIONAL — interface functions
4. OPERATIONAL — device functions

The first three areas noted above are addressed in IEEE 488.1-1987 "Digital Interface for Programmable Instrumentation," and the fourth area, operational specifications, is covered in IEEE 488.2-1987 "Codes, Formats, Protocols, and Common Commands for Use with IEEE 488.1-1987." Note that the IEEE 488.2 standard builds on the IEEE 488.1 standard to provide a more complete interface specification (Figure 2.2). IEEE 488.2 does not preclude any of the specifications set forth in IEEE 488.1, and in fact, the authors of both standards gave particular attention to ensuring that devices implemented under IEEE 488.1 will still operate within the constraints of the second part of the overall IEEE 488.2 standard.

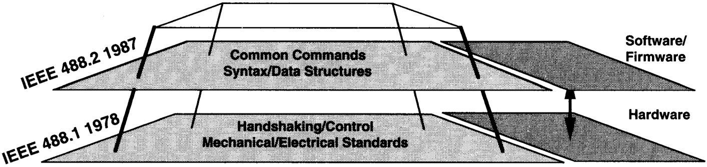
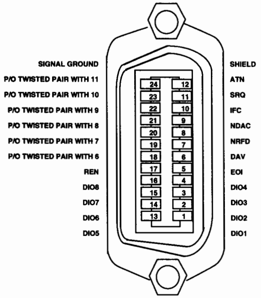


Figure 2.2–Layering the IEEE 488.1 and 488.2 standards

2.2 Mechanical Specifications of IEEE 488

2.2.1 Connector

The IEEE 488 connector is a 24 pin connector with contact pin designation as shown in Figure 2.3. These connectors are commercially available from AMP as the CHAMP series and from Cinch and Amphenol as Series 57. Devices on the IEEE 488 bus have female receptacles, and interconnecting cables have the mating male connectors. Connecting cables will typically have male and female receptacles wired in parallel at each connecting head to allow parallel connection of cables at a device and/or to allow daisy chaining between devices.



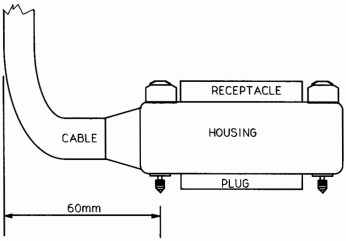


Figure 2.3–Connector pin assignments

Figure 2.4–IEEE 488 connector

2.2.2 Hardware

Connectors (Figure 2.4) are attached using metric thread (ISO M3.9 x 0.6) jackscrews; though the jackscrews have screwdriver slots, the screws should be hand tightened. The screwdriver slots are provided for removal only.

2.2.3 Interconnection Cabling

Twenty-four conductor cable is specified and configured with 11 single conductors, six twisted pairs, and an overall shield. The maximum resistance (in ohms/meter) for the cable conductors shall be:

1. Each signal line (for example DI01, ATN): 0.14Ω/m
2. Each individual signal line ground return: 0.14Ω/m
3. Common logic ground return: 0.085Ω/m
4. Overall shield: 0.0085Ω/m

The maximum capacitance (measured at 1 kHz) between any signal line and all other lines (signals, grounds, and shields) connected to ground shall be 150 pF/m. The shield shall contain a braid of 36 AWG wire or equivalent with at least 85 percent coverage.

Any individual IEEE 488 bus is limited to 15 devices including the controller; this means that 14 interdevice connections are possible on the bus. However, the IEEE 488 specification limits the total length of all cabling used to interconnect devices on a common bus to 20m, or 2m times the number of interconnected devices (up to 20m). Cable lengths between devices may vary but the total cable length must not exceed the restrictions mentioned above. Devices may be interconnected in a star or linear topology or in a combination of star and linear, as long as the distance limits are observed (Figure 2.5). For maximum data transfer rates, the total cable length should be reduced to 15m and the average interdevice cable should be 1m or less.

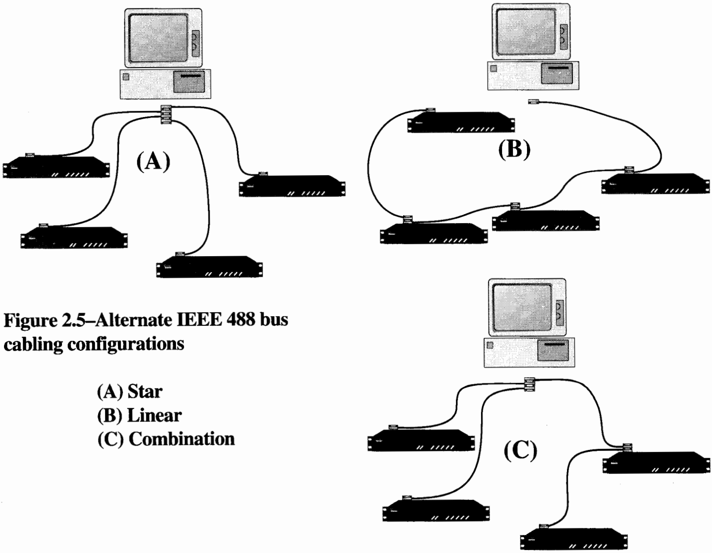


Figure 2.5–Alternate IEEE 488 bus cabling configurations

(A) Star
(B) Linear
(C) Combination

Bus length limitations may be avoided using bus extenders such as the IOtech Extender488 series (Figure 2.6) or the Extender488/HS to allow separations of up to 1000m between devices. The IOtech Extender488 is a moderate speed extension device that uses inexpensive twisted pair cable between bus devices; the Extender488/HS is a high speed extender that supports rates over 800 Kbytes/sec. Extenders using fiber optic cable, such as the IOtech Extender488/F, can be used in applications requiring high noise immunity or dielectric isolation between devices on the IEEE 488 bus.

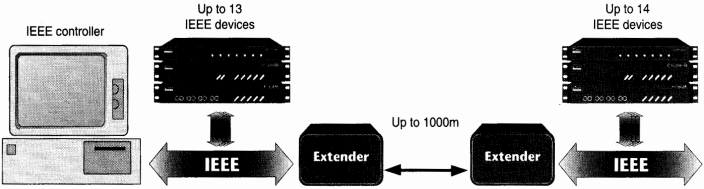


Figure 2.6–Bus extenders circumvent the IEEE 488 standard 20m length limitation

Extenders also expand the maximum allowable number of devices by adding 14 IEEE 488 addresses on the remote end of the extender pair. The extender is the 15th device.

2.3 Electrical Specifications of IEEE 488

2.3.1 Bus Lines

The IEEE 488 bus is a multidrop interface in which all connected devices have access to the bus lines. The 24 bus lines group into four categories (Figure 2.7):

* **Data lines**–Eight lines (DIO1 through DIO8), used to transfer information (data and commands) between devices on the bus, one byte at a time

* **Handshake lines**–Three lines, used to handshake the transfer of information across the data lines:

DAV–Data Valid
NDAC–Not Data Accepted
NRFD–Not Ready for Data

* **Bus management lines:** Five lines, used for general control and coordination of bus activities:

ATN–Attention
IFC–Interface Clear
REN–Remote Enable
SRQ–Service Request
EOI–End or Identify

* **Ground lines:** Eight lines, used for shielding and signal returns:

One Shield
One General Signal Ground
Six logic ground lines paired off
with ATN, SRQ, IFC, NDAC,
NRFD, and DAV

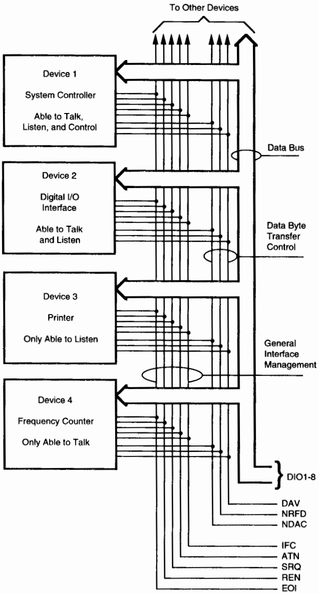


Figure 2.7–IEEE 488 bus lines

2.3.2 Electrical Requirements

The signal lines are low TRUE logic levels so that a signal voltage greater than +2V is defined as Logic FALSE and a signal voltage less than +.8V is defined as Logic TRUE.

Signal drivers must be open collector logic (allowing multidrop device connection on the bus with low TRUE logic, although tristate drivers are permitted on ATN, IFC, REN, EOI, DAV, and the eight data lines). The use of tristate drivers is recommended for data rates higher than 250 Kbytes/sec. All signal drivers must be able to sink 48 mA continuously and tristate drivers must also source 5.2 mA continuously.

Since the IEEE 488 standards do not provide for electrical isolation, the bus, its devices and the controlling computer may be vulnerable to high voltages present on a bus device. A bus isolator can protect the balance of the bus from a device with such voltages. IOtech's Isolator488 (Figure 2.8) provides 1500V of isolation as well as an additional 14 bus addresses on its isolated side.

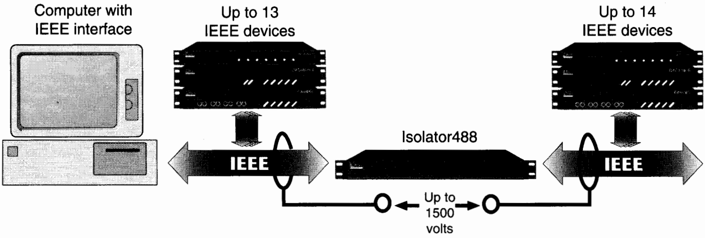


Figure 2.8–Electrically isolate (to 1500V) and expand the number of instruments on the IEEE 488 bus with Isolator488

2.3.3 Handshaking

The IEEE 488 bus uses three handshake lines in a "We're ready - Here's the data - We've got it" sequence to transfer information across the data bus. This handshake protocol assures reliable data transfer at the rate determined by the slowest Listener. One line (DAV) is controlled by the Talker, while the other two (NRFD and NDAC) are wired-or lines shared by all Active Listeners. The handshake lines, like all other IEEE 488 lines, are active low.

DAV is controlled by the Active Talker. Before sending any data, the Talker verifies that NDAC is asserted (low) which indicates that all Listeners have accepted the previous data byte. The Talker then places a byte onto the data lines and waits until NRFD is unasserted (high) indicating that all Addressed Listeners are ready to accept the information. When NRFD and NDAC are in the proper state, the Talker asserts DAV (active low) to indicate that the data on the bus is valid.

NRFD is used by the Listeners to inform the Talker that they are ready to accept new data. The Talker must wait for each Listener to unassert this line (high), which they do at their own rates, when they are ready for more data. This assures that all devices accepting the information are ready to receive it.

NDAC is also controlled by the Listeners and indicates to the Talker that each device addressed to listen has accepted the information. Each device releases NDAC (high) at its own rate, but NDAC does not go high until the slowest Listener has accepted the data byte (Figure 2.9). This type of handshaking permits multiple devices to receive data from a single data transmitter on the bus. All active receiving devices will participate in the data handshaking on a byte-by-byte basis and operate the NDAC and NRFD lines in a "wired-or" scheme so that the slowest active device will determine the rate at which the data transfers take place. In other words, data transfers are asynchronous and occur at the rate of the slowest participating device.

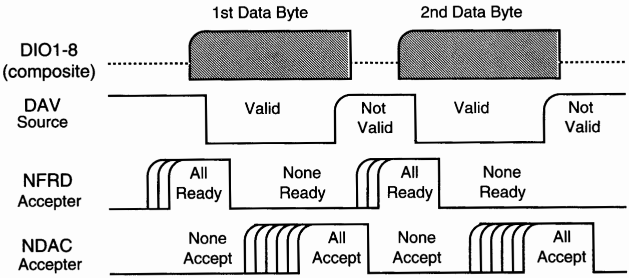


Figure 2.9–IEEE 488 bus handshaking

2.4 IEEE 488 Functions

When information is placed on the data lines (DIO1-DIO8), it can represent either a data byte or a command. If the Attention bus management line (ATN) is asserted while the data is transferred, then the data lines are carrying a multiline command to be received by every bus device. If ATN is not asserted, then a data byte is being transferred, and only the Active Listeners receive that byte.

The IEEE 488 bus also has a number of uniline (single line) commands that, as their name implies, are carried on a single bus management line. For example, the Interface Clear (IFC) line, when asserted, sends the Interface Clear command to every bus device, causing each to reset its IEEE 488 bus interface.

2.4.1 Addressing

The IEEE 488 standard normally permits up to 15 devices to be configured within one system. Each of these devices has an IEEE 488 bus address, a number from 0 to 30, that must be unique to avoid conflicts and confusion. The method by which a specific device's address is set is designated by the manufacturer. Some device addresses are set by DIP switches in hardware, others by software or by front panel controls. The manufacturer's instructions should describe how to set the bus address. Address limits can be circumvented directly by the use of bus expanders such as IOtech's Expander488 (Figure 2.10), or indirectly through the use of an isolator or an extender.

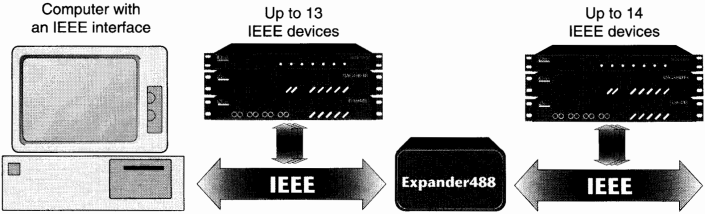


Figure 2.10–Expander488 provides an effortless method of expanding an IEEE system beyond 15 devices

A device becomes Addressed to Talk when it receives a Talk Address Group (TAG) multiline command (a byte transferred with ATN asserted) specifying its own address from the Active Controller. Similarly, it becomes Addressed to Listen when it receives a Listen Address Group (LAG) multiline command. Other address commands include My Talk Address (MTA) and My Listen Address (MLA), which are the TAG and LAG commands of the Active Controller. The Secondary Command Group (SCG) is used to refer to subaddresses or subfunctions within a particular device. This permits direct access and control of the subdevices or subinstruments embedded within complex devices or instruments.

2.4.2 The System Controller

The System Controller, usually a computer with an IEEE 488 board installed, is a device that always retains ultimate control of the bus. When the system is first powered up, the System Controller is the Active Controller and controls all bus transactions. It is possible for the System Controller to Pass Control to a device, making it the new Active Controller, which may, in turn, Pass Control to yet another device. Even if it is not the Active Controller, the System Controller maintains exclusive control of the Interface Clear (IFC) and Remote Enable (REN) bus management lines and can thus take control of the bus whenever it desires.

2.4.3 Bus Management Lines

Five hardware lines on the IEEE 488 bus are used for bus management. Signals on these lines are often referred to as uniline commands. The signals are active low, i.e., a low voltage represents a logic TRUE (asserted), and a high voltage represents a logic FALSE (unasserted). Some of these lines are connected in a wired-or configuration. These lines can be asserted by any bus device, and the signal on that line is asserted if any device is driving it. Conversely, the signal is unasserted only if no devices are driving that line.

2.4.3.1 Attention (ATN)

ATN is one of the most important lines for bus management. When Attention is asserted, the information contained on the data lines is to be interpreted as a multiline command. When it is not, that information is to be interpreted as data for the Active Listeners. ATN can only be driven by the Active Controller.

2.4.3.2 Interface Clear (IFC)

The System Controller uses the IFC line to place all bus device interfaces in a known, quiescent state. IFC places the devices in the Talk and Listen Idle states (neither Active Talker nor Active Listener) and makes the System Controller the Active Controller.

2.4.3.3 Remote Enable (REN)

The System Controller asserts REN to allow bus devices to respond to remote (bus) commands. When REN is asserted, all listeners capable of remote operation enter remote operation when addressed to listen. If REN is unasserted, then the bus devices may ignore the bus and remain in local operation. All devices capable of both remote and local operation must monitor REN and respond within 100 µsec.

2.4.3.4 End or Identify (EOI)

EOI is used to signal the last byte of a multibyte data transfer. The device that is sending the data asserts EOI during the last data byte. EOI is not always necessary; instead, the last data byte might be a special character, such as carriage-return or line feed. EOI is also used by the Active Controller to perform a Parallel Poll by simultaneously asserting EOI and ATN.

The new IEEE 488.2 standard specifies that a 488.2-compliant peripheral accept either a line feed or EOI asserted as the end of a transfer. Further, as a peripheral, it must signal the end of a transfer by asserting EOI while sending a line feed.

2.4.3.5 Service Request (SRQ)

SRQ is a wired-or line that is asserted by any device that desires the attention of the Active Controller; it can be used to interrupt the current sequence of events. The device may be reporting that it has data to send, an error condition to report or both. The Controller can determine which device requested service using Serial Poll or Parallel Poll. A Serial Poll will clear the SRQ line unless some other device is requesting service. SRQ is the IEEE 488 bus equivalent of the hardware interrupt found in computer systems.

2.4.4 Multiline Commands

Multiline commands are bytes sent by the Active Controller over the data bus with ATN asserted. These commands are divided into five groups: Listen Address Group (LAG), Talk Address Group (TAG), Secondary Command Group (SCG), Addressed Command Group (ACG), and Universal Command Group (UCG). These commands are sent to all devices and serve several purposes:

1. *Talk or listen addresses* inform the devices on the bus who will provide and who will accept data. These multiline messages, sent over the data bus, are sent to all devices.
2. *Secondary commands* are multiline messages used in conjunction with an address, multiline universal command, or addressed command. These commands are used to expand the code space when necessary. For example, a Controller may address a subdevice within a complex device by using secondary addresses.
3. *Addressed commands* affect only those devices which have previously been addressed to be a listener.
4. *Universal commands* cause every instrument on the bus to carry out the bus function specified (if the instrument is capable of it). There are five multiline and four uniline universal commands:

2.4.4.1 Listen Address Group (LAG) (&H20-3F)

These commands Address to Listen specified bus devices (&H20-3E for device addresses 0 through 30) or Address to Unlisten (UNL) all bus devices (&H3F). The addressed device then becomes a Listener.

2.4.4.2 Talk Address Group (TAG) (&H40-5F)

These commands Address to Talk specified bus devices (&H40-5E for device addresses 0 through 30) or Address to Untalk (UNT) all bus devices (&H5F). The addressed device then becomes a Talker.

2.4.4.3 Secondary Command Group (SCG) (&H60-7F)

These commands are used to specify a subaddress or subfunction within a given bus devices. They are also used in the Parallel Poll Configure sequence.

2.4.4.4 Addressed and Universal Command Groups (ACG, &H00-0F and UCG, &H10-1F)

These commands perform various bus functions described in Table 2.1. Addressed Commands affect only the Active Listeners, whereas Universal Commands affect all bus devices. The Addressed and Universal multiline commands are shown in Table 2.1 below.

Addressed/ Universal	Multiline Commands	Acronym	Code	Description
A	Go To Local	GTL	ACG 01	GTL allows the manual or front panel control of the selected devices.
U	Local Lockout	LLO	UCG 11	LLO prevents manual or front panel control of all bus devices.
U	Device Clear	DCL	UCG 14	The command causes all bus devices to be initialized to a power-up or pre-defined state.
A	Selected Device Clear	SDC	ACG 04	These commands cause addressed bus devices to be initialized to a power-up or pre-defined state.
U	Serial Poll Enable	SPE	UCG 18	SPE enables each bus device to output its serial poll status byte instead of its normal data when it is an Active Talker.
U	Serial Poll Disable	SPD	UCG 19	SPD disables all devices from sending their Serial Poll status byte and restores normal data output.
A	Group Execute Trigger	GET	ACG 08	This command usually signals a group of devices to begin executing a triggered action. This allows actions of different devices to begin simultaneously.
A	Take Control	TCT	ACG 09	TCT passes bus control from the current Active Controller to another device, which then becomes the Active Controller.
A	Parallel Poll Configure	PPC	ACG 05	PPC configures devices as to which bit they are to assert in response to a Parallel Poll when they require service.
A	Parallel Poll Unconfigure	PPU	UCG 15	This command disables all devices from responding to a Parallel Poll.

Table 2.1–Addressed and universal command groups

Multiline commands:

Device Clear (DCL)–causes all devices to be initialized to their power-on or to a predefined state

Local Lockout (LLO)– prevents manual or front panel control of all bus devices

Serial Poll Enable (SPE)–enables all bus devices to output their serial poll status byte (instead of their normal data) when they are Active Talker

Serial Poll Disable (SPD)–disables all devices from sending their serial poll status byte and restores normal data output
Parallel Poll Unconfigure (PPU)–disables all devices from responding to a parallel poll

Uniline commands:
Interface Clear (IFC)–initializes the bus to an Idle state
Remote Enable (REN)–enables devices to respond to remote program control when addressed to listen
Attention (ATN)–puts the bus in command mode
Identify (IDY)–the simultaneous assertion of ATN and EOI signals a parallel poll demand

2.5 Addressing

A device becomes Addressed to Talk when it receives a Talk Address Group (TAG) multiline command (a byte transferred with ATN asserted) specifying its own address from the Active Controller. Similarly, it becomes Addressed to Listen when it receives a Listen Address Group (LAG) multiline command. Other address commands include My Talk Address (MTA) and My Listen Address (MLA), the TAG and LAG commands of the Active Controller, and the Secondary Command Group (SCG) used to refer to subaddresses or subfunctions within a particular device.

2.6 Serial Poll

When a device is Serial Polled, it responds to the Controller with an 8-bit status byte, one bit (DIO7) of which is set if the polled device is requesting service (RQS), with the remaining bits containing device-specific status. If the polled device is not requesting service, then the Controller can Serial Poll the other devices until it finds the one needing attention. The Controller can then examine the other bits in the status byte to determine the cause of the request. In this way, repeated Serial Polls can determine which device is requesting service and its status. IEEE 488.2 provides a standard format for the status byte beyond the RQS bit.

2.7 Parallel Poll

When the Controller performs a Parallel Poll, each device that desires service asserts exactly one of the data (DIO) lines. The specific line asserted by a given device may be set under software control (with Parallel Poll Configure) or by switches or other controls within the device. By examining the 8-bit parallel poll response, the Controller can determine which devices require service. However, the Controller receives no other status information about that device and often follows the Parallel Poll with a Serial Poll of the requesting device.

Because Parallel Poll is much faster than Serial Poll, it is the preferred method of determining which device(s) require service. Once the devices are identified, Serial Poll, along with other device-dependent commands, is used to determine the cause of the SRQ and the appropriate actions. However, the Serial Poll and Parallel Poll response capabilities vary greatly among devices, and a given bus device might not support them. The Controller must be aware of these limitations when responding to an SRQ.

2.8 IEEE 488.2

IEEE 488.2 expands on the IEEE 488.1 specification by detailing a preferred implementation of many of the issues that were either optional or unspecified in the first standard. IEEE 488.1 covers the key physical issues (connector type, bus length, maximum number of instruments, etc.) and electrical issues (open collector TTL, tristate), as well as low-level protocols (device addressing, control passing, and data handshaking/timing). Four basic device functions (Talker, Listener, Controller, and System Controller) are specified, as are capability subsets for each type of device.

A number of items not covered by 488.1 can cause problems for the test engineer, particularly regarding equipment compatibility and data corruption. For example, 488.1 does not cover these specifications:

- **Minimum Device Capability Requirements**

  No minimum set of requirements is mandated in IEEE 488.1 for Talkers, Listeners, Controllers, or System Controllers. Hence, a device may implement all, or only some of the capability sets set forth in 488.1, giving rise to systems containing devices with varying levels of abilities.

  The Controller in such a situation has no guarantee of a basic communication subset among system devices. This can lead to confusion for the system operator and miscommunication between devices.

- **Data Coding, Formats, and Message Protocol**

  Under 488.1, the messages transferred between the Controller and a device are entirely at the discretion of the device manufacturer. The use of ASCII, binary, or some other form of data code and the choice of terminators such as carriage-return or EOI is arbitrary. Also, the sequence of the sending of commands and the reading of their responses is unspecified and varies from instrument to instrument.

- **Definition of the Status Byte**

488.1 defines a status byte and one bit within, but the meaning of the other seven bits is at the discretion of the device designer. This forces the user to provide a unique interpretation of each bit of the status byte. Also, the relationship between the status byte and the device's other internal status registers is unspecified.

In spite of the omissions in the 488.1 standard, thousands of systems have been successfully configured using the IEEE 488 bus. This is a testament to the ability of designers to work toward reasonable interpretations of the specification where it is incomplete and also evidences the intuition of users in working around inconsistencies and incompatibilities. However, it has forced users to expend effort on low-level instrument-communication operations which might better have been directed toward obtaining the desired results.

The IEEE 488.2 standard was developed to simplify the basic process of communicating with instruments. IEEE 488.2 extends the 488 standard with code, format, and protocol standardization and serves to resolve issues left open in 488.1, such as the examples noted. It also addresses these issues:

- **488.2 Required User Documentation**

Minimum device documentation is specified, i.e., all of the capabilities of the device must be presented in the manual.

- **488.2 Precise Talking/Forgiving Listening**

When an instrument is talking (sending responses), it must use a well-defined and exacting syntax that can easily be understood by the user's program. When an instrument is listening (receiving commands and data), it must be able to accept a reasonable variety of formats that might be generated by the user's program. In both cases, the instrument is required to perform so as to simplify the user's programming burden.

- **488.2 Common Commands**

  Many common commands such as Device Reset and Read Status Byte have been defined by the IEEE 488.2 standard. Many of these are required in every IEEE 488.2 instrument, while some are optional. These common commands ensure some uniformity among instruments and further simplify the user's task. Device-specific commands are not strictly specified by 488.2, but 488.2 confines their syntax to simple, easily generated text strings. It also specifies the formats to be used for numeric data, including engineering units, as well as for blocks of binary data.

- **488.2 Hardware Implementation**

  The IEEE interface hardware has become a de facto standard based on the NEC µPD7210 or TI9914 interface chips. These chips are the basis for nearly all IEEE 488 controllers available today for the IBM PC and most other platforms as well. Although the introduction of the IEEE 488.2 standard has required some minor additions to the logic found in these chips, they still remain a viable method for implementing IEEE 488.2 in equipment.

2.9 IEEE 488 Driver Software for the IBM PC

Great variety is found in the software required to complete the interface between the user's program and the IEEE instruments. Two fundamental techniques are used: the DOS device driver and the subroutine library. These are not mutually exclusive, as subroutine libraries can be implemented via a DOS device driver.

2.9.1 DOS Device Driver

A popular form of device driver pioneered by IOtech and used by several IEEE 488 controller providers is the Terminate and Stay Resident (TSR) DOS device driver approach. In this method, the driver code is stored in memory as a TSR and waits for access by an application program, much as Borland's Sidekick waits for user "hot key" input. IOtech's Driver488, for example, establishes a file I/O link with DOS, just as DOS provides file I/O links for system devices such as the keyboard/screen (CON), printer (PRN), or serial port (COM). Driver488, while resident in memory, establishes "files" called "IEEE" (for input and output), "IEEEIN" (for input) and "IEEEOUT" (for output). Actually, all three of these device names provide both input and output to the IEEE 488 bus; IEEEIN and IEEEOUT are provided as a convenience to the user to help him or her avoid confusion between input and output.

These DOS I/O files may be accessed directly from DOS, from programs with file I/O capability, including spreadsheets such as Lotus 1-2-3 and Borland's Quattro, and from most programming languages. These files provide a direct link to the IEEE 488 bus using IOtech's HP-style English language commands. This style of Applications Program Interface (API) is often referred to as Character Command Language (CCL), as the IEEE commands are sent as ASCII strings to the driver via the API's file I/O links through DOS.

2.9.1.1 Controlling IEEE 488 Instruments Directly from DOS

With a DOS device driver such as Driver488 installed into the DOS operating system, IEEE 488 devices can be controlled directly from the DOS prompt without the need to write a program. The first step is to use Driver488's installation/configuration utility to configure each IEEE 488 device attached to the bus. This procedure includes specifying the addresses and terminators for each device, and allows the device to be designated by an alphanumeric name such as "DMM1."

The following examples using an IOtech ADC488 (which we have named with Driver488's install/configuration utility, DMM1) illustrate how users can establish IEEE control directly from the keyboard under DOS (the ">" is the "pipe" or redirection convention in DOS, sending the output of the device on the left of the ">" to the input of the device on the right of the ">"). "ERROR ON," "FILL," "CLEAR," "OUTPUT," "SPOLL," and "ENTER" are Driver488 HP-style commands that the user will send to Driver488 for execution via the "file" IEEE.

To send the message "ERROR ON" from the keyboard to the driver and enable errors to be displayed, the user must, upon the DOS prompt C:>, type:

```
C:> ECHO ERROR ON > IEEE  <return>
```

To instruct the driver to append a carriage return (<cntrl-Z>) to all input data, the user must type:

```
C:> ECHO FILL $26> IEEE <return>
```

To clear the DMM to its power-on condition, the user must type:

```
C:> ECHO CLEAR DMM1 >IEEE <return>
```

To program the DMM for the 2 VDC range, the user must type:

```
C:> ECHO OUTPUT DMM1; C1T6X> IEEE <return>
```

To serial poll the DMM and display the result, the user must type:

```
C:> ECHO SPOLL DMM1 > IEEE <return>
C:> TYPE IEEE <return>
```

Note that the DOS command "TYPE" is used to view the "file" IEEE, just as it would be with any DOS file.

To input and display a single reading from the DMM, the user must type:

```
C:> ECHO ENTER DMM1> IEEE <return>
C:> TYPE IEEE <return>
```

To view continuous readings from the DMM, the user must type:

```
C:> TYPE DMM1 <return>
```

The screen will scroll with readings from the DMM until the user presses <cntrl-break>. Since the <cntrl-break> may interrupt the driver in the middle of a transaction, the first attempt at another transaction may result in a SEQUENCE ERROR message from the driver. Simply retrying the subsequent transaction will clear the error.

To save data from the DMM directly into a file called "DATA.DAT," the user must type:

```
C:> TYPE DMM1 > DATA.DAT  <return>
```

Here the DOS pipe is used to redirect the TYPE command output to the file DATA.DAT instead of allowing it to default to the screen. To halt the flow of data to the file, the user must press <cntrl-break> as before. The user may employ the TYPE command to view the file:

```
C:> TYPE DATA.DAT
```

or may copy it to a printer

```
C:> COPY DATA.DAT PRN
```

or to import it into an application program.

2.9.1.2 Controlling IEEE 488 Instruments from Spreadsheets

The previous section showed how the DOS device technique, exemplified by Driver488, makes instrument control from DOS as simple as working with DOS files. Since almost any operation permitted with a file is also possible with a DOS device, any application program that permits file I/O will also permit IEEE 488 control via Driver488. A typical example is controlling IEEE 488 instruments via spreadsheets such as Lotus1-2-3 and Quattro.

While most spreadsheet commands are performed by choosing items from a menu system, spreadsheets typically include a macro facility to record and save menu selections and keystrokes. "Playing back" such macros allows frequently used operations to be performed with a simple two-keystroke input. These macro systems also include sophisticated programming facilities, such as user prompts, branching and looping, trapping errors, and file I/O. This last capability can be exploited to connect the spreadsheet with the IEEE 488 bus.

For example, in Lotus 1-2-3 and Driver488, a direct channel of communications from 1-2-3 to the IEEE 488 bus can be opened by using the macro OPEN command. Subsequent WRITELN and READLN macro commands will write and read data directly from the driver. To establish the link between 1-2-3 and Driver488, users must employ the macro command:

```
{OPEN IEEE,W}
```

where the "W" indicates that writing to the "file" IEEE will be permitted. As data from a file of any type is read into a spreadsheet, the spreadsheet expects a carriage return (<CR>) after every data point. Without the proper terminator, the spreadsheet cannot distinguish one data point from another. Thus during the setup of Driver488 (using the install/configuration utility) the end-of-line terminator sent by Driver488 to applications programs must be set to "CR." During this configuration process the user can assign alphanumeric names to the devices attached to the bus and specify *their* terminators as well. These steps eliminate the need to define data termination using macro commands that send CCL messages to Driver488.

After the driver has been opened with the OPEN macro command, the user can build a macro command sequence to begin to acquire data and place it in the spreadsheet. The following simple example employs a combination of Driver488 commands (OUTPUT, TRIGGER, and ENTER) and spreadsheet macro commands (WRITELN and READLN). Every driver command that brings in data (i.e., SPOLL, ENTER) must be followed by a spreadsheet command that accepts that data (READLN). Using the same ADC488 DMM example as in the previous section, the following macro sequence is employed to collect one data point and store it in the spreadsheet. To set the DMM for the 2 VDC range, the user must employ the macro sequence:

```
{WRITELN"OUTPUT DMM;C1T7X"}
```

To trigger the DMM, the user must employ the sequence:

```
{WRITELN"TRIGGER DMM"}
```

To read one data point and store it in cell E24, the user must employ:

```
{WRITELN"ENTER DMM"}
{READLN E24}
```

After all the desired values have been read, the spreadsheet must be updated with the macro command:

```
{CALC}
```

At this point the spreadsheet cells contain ASCII data, but not numerical data, which will lead the spreadsheet to treat the data as labels rather than numbers. To avoid this, each data point must be converted from an ASCII string to a numerical value. Usually this is done in a single operation with the @VALUE function. Unfortunately, here each data point will have a CR appended to it. Thus the user must employ the following sequence of commands to apply the @VALUE function to all the characters of the string in cell E24, save the last:

```
{LET F24,(@VALUE(@LEFT(E24,@LENGTH(E24)-1)))}
```

This performs the @VALUE function on all but the last byte of the string in E24 and puts the resulting value in F24.

By using the extensive branching, looping, and user input capabilities of the spreadsheet's macro language, the user can perform complex data acquisition, analysis, and graphical displays thanks to Driver488's device driver techniques. Any of Driver488's commands can be sent from the spreadsheet using the {WRITELN} macro command. IOtech Application Note (#3), *Control IEEE 488 Instruments with Lotus 1-2-3, Lotus Symphony, and Borland's Quattro*, provides detailed macro listings for acquiring, manipulating, and displaying data directly from the IEEE 488 bus.

2.9.1.3 Controlling IEEE 488 Instruments from Any Language

Just as DOS and spreadsheets can access IEEE instruments directly using the file I/O services provided by DOS for device drivers, most programming languages also can use file I/O to quickly and easily access the IEEE 488 bus. As was mentioned earlier, IEEEIN and IEEEOUT are provided as a convenience to the user to aid in distinguishing between the use of the Driver488 device as an input or an output port.

In BASIC, files are opened using OPEN FOR OUTPUT and OPEN FOR INPUT commands:

```
OPEN "IEEEOUT" FOR OUTPUT AS #1
OPEN "IEEEIN" FOR INPUT AS #2
```

Communication with Driver488 is achieved using PRINT and INPUT commands, as in:

```
PRINT#1, "HELLO"
INPUT#2, A$
PRINT A$
```

which will send the Driver488 command HELLO to the driver, which will respond with the Driver488 sign-on message, i.e., "`Driver488 Rev. 3.2 (C) Copyright 1990 IOtech, Inc.`" Any Driver488 command may be sent using the same method, and data can likewise be retrieved from the bus.

For other languages, the principles are the same. For example, in Turbo Pascal, the required files are opened with these statements:

```
VAR IeeeOut, IeeeIn: TEXT;
Assign (IeeeOut, 'IeeeOut'); Rewrite(IeeeOut);
Assign (IeeeIn, 'IeeeIn'); Reset(IeeeIn);
```

In Microsoft C, the statement is:

```
ieee=open('ieee',O_RDWR | O_BINARY);
```

Building and completing IEEE control programs is discussed in detail in the IOtech Personal488 and Power488 series manuals.

2.9.2 Subroutine IEEE 488 Driver Interface

An alternative means of controlling the IEEE 488 hardware is via subroutine calls from high level languages. This method has the advantage of minimizing the overhead of DOS' device driver services and the ASCII message (Character Command Language) parser and interpreter. The disadvantages include the loss of the convenience and effectiveness of accessing the IEEE 488 bus from a wide variety of applications programs such as spreadsheets, as well as directly from DOS itself. Also, the use of subroutines, even those with easy-to-use HP-style commands such as IOtech's Driver488, typically requires compiling and linking to run even simple test code.

Some IEEE controller implementations on the IBM PC, notably IOtech's Driver488, give the user the choice of subroutine calls **or** character command language.

2.9.3 IEEE 488 Subroutine Control Libraries

The logical complement of subroutine interfaces for a TSR DOS device driver are subroutine libraries that directly access the IEEE 488 hardware from a high-level language with code that is compiled and linked directly into the user's program. This approach eliminates the DOS device driver, integrating the IEEE 488 control functions directly into the applications program code. Of all the alternatives, this approach has the potential for the highest performance, as it eliminates possible DOS effects on the speed of commands and data.

For those in need of small, tight IEEE 488.2 control code to embed in their applications programs, IOtech provides two libraries of IEEE 488 control subroutines. For those requiring IEEE 488 controller routines, Library488 provides full IEEE 488.2 controller functions in a suite of routines that encompasses the functionality of Driver488 (Figure 2.11), including "on SRQ" vectoring, DMA transfers, and automatic error checking, as well as low-level commands such as "send Unlisten." In contrast, the Personal488/OEM-P is a suite of IEEE 488.2 peripheral routines intended for the designer of instrumentation with an embedded PC architecture, who requires compact code with low overhead for an IEEE interface (Figure 2.12). Using the peripheralsroutine approach can significantly reduce time-to-market by eliminating the need to develop, test, and debug IEEE 488.2 peripheral code in-house.

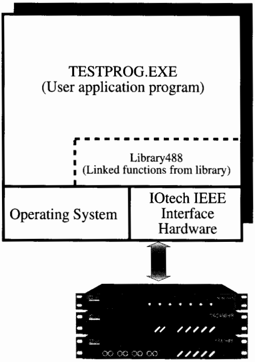


Figure 2.11–Library488 is ideal for applications where speed and compactness are required

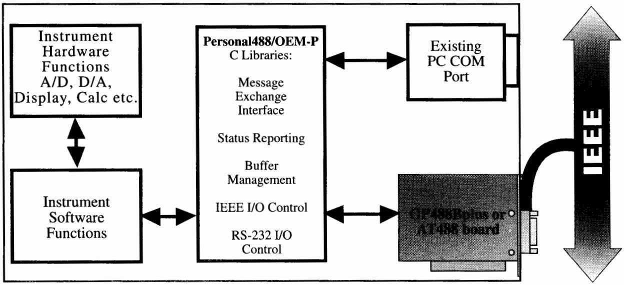


Figure 2.12–Peripheral device using IOtech's Personal488/OEM-P

2.9.4 Microsoft Windows Compatibility

The growing popularity of the Windows 3.0 Graphical User Interface (GUI) is rapidly spreading to test and measurement (T&M) applications. Until 1991, few tools were available for the end user to build Windows applications. Microsoft's Windows Software Development Kit was difficult even for full-time, professional Windows application developers. Now, however, new tools such as Microsoft's Visual Basic and Borland's C++, provide GUI development interfaces that allow users to draw windows and fill them with buttons, scroll bars, and dialog boxes. Soon, these tools (and the tools, libraries, and utilities that will follow) will be widely used by developers of IEEE 488 test programs. IEEE 488 controller package vendors will adapt their offerings to be compatible with Windows, so users will be able to apply Windows solutions to their measurement problems.

As these new Windows-oriented drivers and packages debut, there will undoubtedly be a broad range of solutions offered to the end user. It is important to know and understand what makes Windows and Windows applications different from DOS, and what features an IEEE 488 driver should have in order to make the most of the Windows environment. Users should keep the following issues in mind when reviewing new offerings:

- Is the software written as a Windows application, or is it merely a port of DOS software?
  - Windows performs its own memory management functions; typical DOS ports to Windows do not permit Windows to dynamically allocate memory use, which can lead to "Unrecoverable Application Errors."
  - As Windows is an event-based system, it provides extensive event handling facilities; Windows applications should take advantage of them.
  - Windows has no equivalent of the TSR concept used with DOS; although some DOS TSRs will function while Windows is running, their operation can be erratic and unpredictable.
- Will the driver support concurrent access of different peripherals on a single interface by multiple Windows applications? Windows' pseudomultitasking is one of its reasons for being.
- Will the driver service multiple bus adapter boards?
- Is the driver IEEE 488.2 compliant?

2.9.5 Troubleshooting the IEEE 488 Bus

To efficiently diagnose, troubleshoot, and verify IEEE 488 systems, it is necessary to have some basic knowledge of the IEEE 488 bus. Since the hardware portion of the IEEE 488 standard is rigorous and stable, most problems encountered during the system integration process will lie in the application software.

2.9.5.1 Analyzing the IEEE 488 Bus

Without a doubt, the simplest way to decipher the Controller's operations and the response of the instruments, regardless of the software or hardware used, is through an IEEE analyzer or bus monitor. Analyzer488, from IOtech, will allow the programmer to view all of the transactions on the bus in real-time or record them into its 32K non-volatile transaction buffer for later inspection. IOtech's smaller, lower cost Monitor488 captures up to 10K transactions in its circular buffer; it can single step or slow the bus. Each product's printer port allows concurrent hard copy print out of bus transactions.

Analyzer488 can be operated as a portable bench-top analyzer from its keypad or from the included Analyst488 PC and PS/2 software. Analyzer488 allows the events on the IEEE 488 bus to be monitored and analyzed. The analyzer can also be used to control devices on the bus for exercising and verifying instrument operation. The analyzer will automatically translate the state of the data bus and control lines into easy to read IEEE 488 messages or ASCII equivalents such as SPE, TAG16, CR, and LF. Along with its large capture buffer, Analyzer488 contains a comprehensive set of trigger features that allows the desired group of transactions to be easily pinpointed and identified.

2.9.5.2 Common Problems and Solutions

While individual IEEE 488 bus devices all have their idiosyncrasies, some systems will encounter problems as a result of the *interaction* of several devices in the system. Such problems are among the most difficult to debug. An effective strategy is to connect an IEEE 488 bus analyzer and let it run while the application is processing. Recording the bus transactions as they occur will usually allow the user to diagnose these types of problems rather quickly. The following problems were all diagnosed using Analyzer488.

Often the problems encountered in a system result from the interaction between one device and the Controller. Here is a list of common symptoms and their suggested solutions:

"A time-out error occurs when a device-dependent command is sent to an instrument."

The user should first check the setting of the address of the instrument. Every device on the bus must have a unique address between 0 and 30. When sending device-dependent commands (DDCs) to an instrument to change its state or operating mode, the device will first be addressed to listen; only then will the DDC be sent. If the device has Talk and Listen indicators on its front panel, the user can tell immediately if the address used by the Controller matches the actual address of the instrument, because the lights will illuminate. If the Listen indicator does not come on when commands are being sent to the device, the user is employing the wrong address for that device; the device is not listening for the DDCs.

As previously discussed, when the ATN line is asserted by the Controller, **all** the instruments on the bus will handshake with and accept commands from the Controller. After the time-out is received, the transactions recorded by Analyzer488 should be stepped through. If no instrument addressing command such as Listen Address Group 16 (LAG16) was recorded, the instrument is probably turned off or broken or the cable is disconnected. Regardless of the present state of the instrument, it should handshake (accept data) when the ATN line is asserted. If the addressing commands **were** successfully recorded on the analyzer, the user must step though the transactions until the ATN line is unasserted. If there are no more recorded transactions, no instrument was placed in the Listen mode, i.e., the Controller had no device to handshake with and so it timed out. Since the instrument is responding to ATN but not to LAG, it is probably set to the wrong address.

"At certain points in the program, the system stops and a time-out error is received."

If portions of a program are operating correctly, the user can be certain that the addresses are set correctly. If time-out errors occasionally occur or occur at the same place in a program after other instrument tasks have been completed successfully, this may indicate an instrument-readiness problem. IEEE 488 interfaces such as IOtech's Personal488/AT operate very rapidly and can sometimes outrun the instrument they are controlling.

For most instruments, requesting data is performed in two steps: the user sends the necessary set up or inquiry commands via DDCs, then addresses the device to Talk. It is possible to issue the necessary commands to request the data from the instrument and then address it to Talk long before it is prepared to supply the requested data. Most instruments will simply pause the bus until the data has been prepared to send. Other instruments react poorly by "hanging up."

To check for this "race" condition, place the analyzer in the Slow Handshake mode. This will effectively slow the transaction speed of the bus to a rate set by Analyzer488. If the data request takes place successfully, it is probably a race condition.

"The instrument seems unaffected by the commands sent to it."

Given that the commands are being sent to the correct instrument address, an instrument may seem unaffected by commands if the user has omitted the terminator, a crucial piece of information that instructs the instrument to process the commands.

IEEE systems usually use data delimiters called terminators. A Talker will inform a Listener that the data string has ended by appending a predefined terminator to the end of its data string. Although terminators are issued solely by the talking device, the listening device(s) must know which terminator to expect. Most commonly, IEEE 488 instruments will issue a carriage return (CR) and a line feed (LF) as their terminator. Some instruments will not process the incoming command string until they detect the proper terminator. The user should step through the transactions captured by the analyzer to verify the transmission of the terminator, then make certain that it agrees with the terminator expected by the instrument.

Some instruments have a DDC that instructs the instrument to process all of the previously received commands. This EXECUTE command (typically a character such as "X") allows a programmer to send several commands to an instrument in any order over any length of time and then execute them all simultaneously within the instrument. If EXECUTE DDC is not sent, the state of the instrument will not change. It will be as though the commands were never received.

"When asking for data, nothing is returned."

Such a situation could result from an address or terminator problem such as those discussed above. See the previous sections to diagnose such problems.

Not all instruments are ready to supply data whenever asked. Some instruments have nothing to say until they are commanded to acquire or generate data. Some data acquisition instruments have triggering features that allow them to collect and transmit data only after a specified event has occurred. A typical multimeter might have a default trigger of TRIGGER ON TALK that would enable the multimeter to take a reading every time the controller addressed it to Talk. If the same multimeter were set to TRIGGER ON GET, no reading would be available until the Controller issued a Group Execute Trigger.

If the device has no data available, the analyzer will show that the Controller was addressed to Listen, the device was addressed to Talk, and then the process stopped. The handshake indicators will show that Not Ready For Data (NRFD) has been unasserted by the Controller, but that the instrument never asserted Data Valid (DAV). The user should be certain that the device has data to transmit before asking it for some.

IOtech's Driver488 has the capability of assigning a time out value to the system. If an instrument does not respond within the specified time out, the process is aborted. In some instances, an instrument may be unable to respond within the specified time out period, and the time out will have to be increased.

"When asking for data, bad data is returned."

Many times, the variability of data formats of an instrument will cause problems. Devices can transmit data in binary, ASCII, BCD, packed BCD, or anything else that will fit into 8 bits. Data terminators can be EOI, a byte count, or embedded characters like CR LF. Data can be sent with prefixes, suffixes, or full headers. In the case of the IOtech's Driver488, all of these parameters can be accounted for, but other drivers may not allow such flexibility.

When using higher level software packages, the problem of data formats may be impossible to overcome. Usually, menu-driven and turnkey packages go to great lengths to hide the IEEE 488 bus from the operator. The documentation, therefore, makes no attempt to inform the operator of what is actually happening on the bus.

Users may encounter problems if their instruments transmit data in formats not recognized by their software packages. Users should check their instrument manual for data format characteristics. Does the instrument transmit non-numeric prefixes or suffixes; is the data in binary or ASCII? Some software drivers automatically throw away nonnumerics, others do not. Even if the software throws the nonnumerics away, users may encounter problems with instruments that transmit numbers such as channel tags in their data prefix.

Most instruments, including IOtech's ADC488 analog to digital converter, can be programmed to adjust their data format for software compatibility. IOtech's Analyzer488 allows quick inspection of the data being transmitted by an instrument, enabling the programmer to make the proper adjustments in his or her software.

"An SRQ from an instrument sometimes causes a catastrophe."

The asynchronous nature of instrument interrupts can sometimes cause elusive problems. The best way to attack this sort of problem is to start the analyzer recording while the system is running. Analyzer488 has a large 32K transaction buffer that is configured in a circular fashion. After 32K transactions have been recorded, new transactions will overwrite the oldest transactions. There is a very high probability that the events leading up to the system "crash" will still be in the recorded memory (i.e., will not have been overwritten) after the system has locked-up. Stepping backwards in memory can usually uncover the sequence of operations that caused the problem. The analyzer can also be set to trigger on the occurrence of several SRQs with both a post- and pre-trigger assigned. In this way a specified number of events can be captured before and after the occurrence of an SRQ. The analyzer also has comprehensive search features allowing the capture buffer to be scanned for all of the occurrences of any event, including an SRQ.

Some instruments have the capability of generating an SRQ for any of several internal events. Usually an SRQ mask is sent to the instrument to instruct it to generate an SRQ for only a selected subset of those events. Some instruments, by default, will interrupt the controller with an SRQ when an internal error is encountered and will not respond to any further bus transactions until the interrupt is serviced. In such instances, the next time the application program requests data from the instrument, the system will fail. Inspection of the transaction recording in the analyzer, working backward from the end, will reveal that an SRQ was asserted by a device on the bus and that it remained unserviced.

"The system occasionally locks up."

This is the sort of intermittent problem that can take a long time to troubleshoot, especially if the mean time between failures is several hours, days, or months. As in other situations, the best approach to the problem is to allow the analyzer to record all of the transactions occurring on the bus. When the number of transactions goes beyond 32,767, the capture pointer will wrap around and continue to record. The last 32,767 transactions will always be stored in memory. When the system crashes, the processing of IEEE 488 bus transactions will probably end also. With the last 32K transactions captured in memory, the programmer can easily step back through the capture buffer and decipher the sequence of operations that caused the crash.

One possible cause for an intermittent crash problem is the asynchronous occurrences of SRQs as discussed previously. There may be areas in an application program that do not react well to interruption. Since the SRQ can happen at any time, it may or may not occur during the processing of such a sensitive area. But the longer the system runs, the probability that the SRQ will happen at exactly the wrong time increases. A sensitive area may be a part of the programming code that uses a group of closely related variables modified by the SRQ handler. For example, three IEEE 488 counters are used to take readings from three motion encoders. Each counter is programmed to generate an SRQ when its count reaches 256. The SRQ handler reads all three counters and stores them in three separate variables later used by the main program. The main program has a loop that reads the three variables, combines them with some calculation, and sends commands to a motor controller. If an SRQ occurs when the main program is in the process of using the variables, all three variables are consequently modified and the main program may end up using one old value and two new ones in its calculation.

One way to avoid this kind of problem is to disarm the automatic SRQ vectoring during the processing of sensitive program areas. Driver488 has several means by which to arm, disarm, and synchronize the servicing of SRQs to a program.

Another source of system malfunctions are the instruments themselves. Most of today's complex instruments are microprocessor controlled. Their internal processors handle the collection of data, the changing of programmable states, the monitoring of trigger events, and the communication on the IEEE interface. Such instruments are actually computers, prone to the same problems as any other computer.

An instrument may react improperly to a perfectly good application program. The transaction report that Analyzer488 prints out can be used to communicate instrument problems to the manufacturer. The report is easy to read and concisely describes the operation of the controller and the response of the instrument.

2.9.6. New IEEE 488.2 Standards Simplify Programming

Many of the difficulties encountered during the development of IEEE software result from the non-standard elements of operating the IEEE 488 bus. Terminators, common command syntax, and SRQ handling, among other things, were not standardized in IEEE 488.1, but rather were left to the ingenuity of instrument and controller designers to reconcile. IEEE 488.2 covers previously non-standard elements of bus communication, such as terminators and data types. The standards it provides will eliminate some of the variables encountered when integrating instrumentation systems, making debugging simpler.

SCPI

3.1 General

In 1975, the Institute of Electrical and Electronic Engineers standardized system interconnections and communications by instituting a protocol, IEEE 488-1975. A dozen years later, the IEEE updated it with IEEE 488.1-1987. These protocols defined how hardware should behave and how data should flow. Although these standards allowed instruments to communicate and pass data, they did not define the meaning of the information. IEEE 488.2-1987 grew out of an attempt to formulate a common language for commands and responses. It described in detail how to talk and how to listen, but allowed different instruments to continue to speak in different languages. So, in 1990, the Standard Commands for Programmable Instruments (SCPI) Consortium developed a consistent programming environment and language for instrument control and data usage.

This new language, called SCPI, defines the structure, syntax, and commands for communication between controllers and instruments in automatic test equipment (ATE) environments (Figure 3.1). SCPI is intended to give the ATE user a consistent environment for program development by defining controller messages, instrument responses, and message formats for all SCPI compatible equipment, independent of the manufacturer. For example, all SCPI compatible voltmeters, regardless of manufacturer or model, respond to the same command for reading AC voltage, and the format of their responses is also the same. Commands also have the same meaning among different types of instruments. For example, a RANGE command is the same for all range functions in SCPI compatible oscilloscopes and multimeters. SCPI also allows for high level commands such as MEASURE, which may be used (with defined defaults) for quick control of SCPI instrumentation. This standardization of message communication reduces the learning period that was once required for every new instrument in a system, and it allows quick and easy exchanges of instruments in a system without extensive reprogramming.

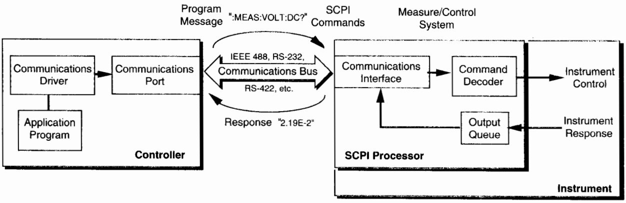


Figure 3.1–ATE system using SCPI

SCPI has roots in and embraces many of the commands and the protocols defined in the hardware independent portion of the IEEE 488.2 communication standard. As such, SCPI represents a logical evolution of the IEEE 488 standard (Figure 3.2). SCPI is a message format that relates to software programming and not to specific communication hardware. SCPI is not hardware dependent or low level protocol dependent, and SCPI messages and responses may be communicated over RS-232, VXIbus, and other communication methods.

SCPI instruments must recognize and properly interpret SCPI commands, usually through a parser (message decoder), and must respond to the commands in a well-defined manner. ATE controllers using SCPI will pass SCPI format command messages and also recognize SCPI format responses from instruments. SCPI messages are generated and interpreted directly by user-written programs or through software drivers created by controller manufacturers. Many of the low level SCPI commands are defined in IEEE 488.2, but since they are defined in a software portion of the overall IEEE 488 standard, the commands are not IEEE 488.1 hardware dependent. As mentioned earlier, SCPI messages may be communicated via RS-232 or other communication methods.

Because the instrument market and technology are dynamic, SCPI is a living standard; new commands may be added under the auspices of the SCPI Consortium to meet future needs. New commands may be proposed to the Consortium by any interested party and will be considered annually for inclusion in the standard.

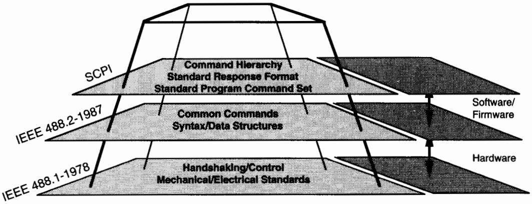


Figure 3. 2–Evolution of the SCPI standard

3.2 Commands

There are several types of SCPI commands: commands mandated by IEEE 488.2, commands required by SCPI, and optional commands discussed in the SCPI standard.

IEEE 488.2 mandated commands (i.e., commands that originated in IEEE 488.2 and are used in SCPI) include:

***CLS**	Clear Status Command
***ESE**	Standard Event Status Enable Command
***ESE?**	Standard Event Status Enable Query
***ESR?**	Standard Event Status Register Query
***IDN?**	Identification Query
***OPC**	Operation Complete Command
***OPC?**	Operation Complete Query
***RST**	Reset Command
***SRE**	Service Request Enable Command
***SRE?**	Service Request Enable Query
***STB?**	Read Status Byte Query
***TST?**	Self-Test Query
***WAI**	Wait-to-Continue Command

All SCPI instruments must implement (i.e., be able to understand) these commands and may implement other optional commands as described in IEEE 488.2. Note again, that despite the IEEE 488.2 origin of these commands, the communication bus between controllers and instruments may be other than IEEE 488.1 (e.g., RS-232, RS-422). These commands, also known as common commands, are identified by the leading asterisk in the command word and are usually not instrument dependent.

The optional command set used by an instrument may include a subset of the optional commands covered in the SCPI standard and is dependent on the capabilities of the instrument. An instrument may also support special commands not currently covered in the SCPI standard in cases where no SCPI command exists to implement a particular instrument function. Unsupported optional commands and parameter values will generate an error message from the instrument.

3.3 SCPI Required Commands

These commands are known as subsystem commands and deal with measurement and other instrument-related functions.

SYSTem	collects functions not related to instrument performance
ERRor?	requests the next entry from the instrument's error queue
:STATus	controls the SCPI-defined status reporting structures
:OPERation	selects the Operation structure
[:EVENt]?	returns the contents of the Event register
:CONDition?	returns the contents of the Condition register
:ENABle	sets the Enable mask, which allows event reporting
:ENABle?	reads the Enable mask
:QUEStionable	selects the Questionable structure
[:EVENt]?	returns the contents of the Event register
:CONDition?	returns the contents of the Condition register
:ENABle	sets the Enable mask, which allows event reporting
:ENABle?	reads the Enable mask
:PRESet	enables all required event reporting

The following are a few representative optional commands (the capital letters indicate the minimum required input for command parser recognition):

ABORt	resets the trigger system and leaves all triggers idle
ARM	qualifies a sequence of events before enabling trigger
CALCulate	specifies post acquisition data processing
CONfiguration	sets up the instrument to perform a specified function
DISPlay	controls selection and presentation of information
FETCh	retrieves measurements and places them in the output buffer
FORMat	sets the data format for transferring information
INITiate	initiates data acquisition
INPut	controls the characteristics of a sensor's input ports
MEASure	acquires data using a set of high-level functions
TRIGger	synchronizes device action with external events

For further commands, consult the SCPI standard (see bibliography) or user's manuals for SCPI instruments of interest in a particular application. Note that a command such as TRIGger may be issued as TRIGGER or as its short-form mnemonic, TRIG, and that a SCPI instrument will understand either command. The formulation of short-form mnemonics is governed by specific rules; normally mnemonics are three or four characters. The exact form allowed is indicated in a command description by the uppercase characters long. The command is not case sensitive (TriG and TRiggER are valid commands), but SCPI does not recognize any in-between versions of a command. (TRIGG is not a valid command.)

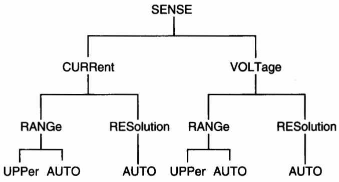


Figure 3.3–Typical command tree

Most instruments require several commands to execute a specific function. For example, a digital voltmeter may require the MEASure, VOLTage, and AUTO commands to take a voltage reading. In order to properly interpret these commands, SCPI defines a hierarchical command structure called a command tree. Figure 3.3 illustrates a portion of a typical command tree. SENSE is the root or top level command; other lower level commands are on particular paths, at various levels. To reach various subcommands, a SCPI message must traverse down the path from the root. If the path described above were for a DVM, the SCPI program message for an auto ranging voltage measurement would be :SENSE:VOLTAGE:RANGE:AUTO

The parser in the SCPI instrument must properly interpret this message, keep track of the current path, and know the meaning of each command on that path. The same command may appear in different paths and have a different interpretation.

Colons were used in the example to separate each command and to instruct the instrument parser to move down a level in the command tree hierarchy. In situations where two commands are issued without changing levels, a semicolon is used to separate the commands. Commas are used to separate parameters (discussed later). Spaces are generally disregarded, save for two exceptions: Command words cannot be broken by spaces, and commands and parameters must be separated by a space. The optional colon preceding the first command in a SCPI message instructs the parser in the SCPI instrument to reset itself to the root level in the hierarchy. Additionally, a message terminator (i.e., newline) instructs the parser to reset the root level of the tree for the next command. Thus the leading colon is optional, unless a preceding command in the same message has left the parser somewhere other than at the root. IEEE 488.2 commands that start with an asterisk can be executed regardless of the command's current level in the tree and do not affect its current level.

Some commands, such as SENSE, are implied commands and are not necessary in a program message. A keyword enclosed in braces designates an optional keyword that will be assumed in the absence of any other allowed keywords at the same level. This assumption does not, however, change the current level in the tree for any subsequent commands. In the previous example, if the SENSE command were omitted, then the parser in a SCPI instrument would assume its existence and correctly decode the message.

All commands, unless specifically noted, have a query form as defined in IEEE 488.2, indicated by a trailing question mark. When a query command is received, the current instrument settings associated with that command are placed in an instrument's output buffer. The query form of a command generates one of two responses, depending on the type of command. A command that takes no parameters returns a "0" (false) or a "1" (true), indicating whether the condition that would be established by the command is already present. For example, "CONF:INP3:ENAB?" will return a "0" if the unit is not enabled for input or a "1" if it is enabled for input. Commands that take parameters, on the other hand, return a string suitable for use in issuing the command, i.e., "CONF:WAVE1:HIGH?" might return "1000" if that were the current setting of the parameter.

Some instruments will incorporate multiple copies of some parts of the command tree. The command tree expands at the level where duplicate capability exists by adding a numeric suffix to select the desired section. For example if the above DVM had multiple independent current sensing sections, the command "SENSE:CURRENTS:RANGE3:AUTO" would be interpreted as an order to select automatic ranging for the third such section. If the numeric suffix of such a command is omitted, it is assumed to be "1." For instruments with multiple capabilities at several levels of the tree, a suffix may appear at each level if appropriate. For example, "OUTP5:MOD3:FM2" would specify the second FM signal component of the third modulation signal on the fifth output channel.

3.4 Parameters

Parameters fall into several categories: numeric, extended numeric, discrete, and Boolean. Numeric parameters comprise any number in decimal or scientific notation and may include polarity signs. Where required, numeric parameters are rounded to the closest allowable value. Extended numeric parameters include values such as MAXimum and MINimum. Boolean parameters represent binary conditions and may be expressed as ON, OFF or 1, 0. Discrete parameters such as INTernal and EXTernal are used to control program settings to a finite value or condition. For example, INTernal might be used to specify an internal trigger source on an instrument (Table 3.1).

Command	Parameter	Parameter Type	Parameter Type
:CONFigure			
:WAVE#			
:SOURce	<source>	discrete	* see below
:LOW	<divisor>	numeric	1..65536
:HIGH	<divisor>	numeric	1..65536
:COUNt#			
:SOURce	<source>	discrete	*see below
:READ			
:COUNt#			
:START			
:WAVE#			
:COUNt#			
:TRIGger			
:WAVE#			
:COUNt			*
			PREVIOUS |
			GATE# |
			SOURCE# |
			1MHZ |
			100KHZ |
			1KHZ |
			100HZ
			(# is 1..5)

Table 3.1–SCPI command parameters

3.5 Command Tree

The list in Table 3.1 is an example of a subset of the SCPI command tree that is supported by the driver software for the IOtech Power488CT IEEE 488 board with 40 digital I/O lines and five 16-bit counter timers. It shows only those commands used in the example program on the following page.

3.6 Example Program Using SCPI Commands

The following program uses the IOtech Power488CT to create a frequency counter. This Power488CT board is internal to the IBM PC, but its driver software responds to SCPI commands in the same manner as would an external IEEE SCPI device. This example is written in BASIC and assumes that the driver software has already been loaded. “CT” refers to the counter/timer on the Power488CT.

```
10    'Open the devices
20    OPEN "CT" FOR OUTPUT AS #1
30    OPEN "CT" FOR INPUT AS #2
40
50    'Set up CT1 to produce a gate pulse, 1 second wide
60    PRINT #1, ":CONFIGURE:WAVE1:SOURCE 1KHZ;LOW 1;HIGH 1000"
70
80    'Set up CT2 to count pulses during the high time of its gate
90    PRINT #1, ":CONFIGURE:COUNT2:SOURCE SOURCE2"
100
110   'Start operation of both sections
120   PRINT #1, ":START:WAVE1;COUNT2;:TRIGGER WAVE1;COUNT2"
130
140   'Read and display the frequency until a key is pressed
150   WHILE INKEY$ = ""
160      PRINT #1, "READ:COUNT2"
170      INPUT #2, FREQ
180      PRINT "Frequency in the last second was";FREQ
190   WEND
```

Serial Communication

4.1 Introduction

Serial communication is the simplest possible computer communication link, transferring only one bit (binary digit) of information at a time. Serial communication (Figure 4.1) makes use of one data line, transferring data one bit at a time, serially, whereas parallel communication uses eight or more lines, transferring data eight bits at a time, in parallel. For example, the IEEE 488 bus uses parallel communication (Chapter 2). Because serial communication transfers data a single bit at a time, it is the least expensive form of data communication. However, it is also one of the slower communication links. Since computers internally handle data in a parallel fashion, serial communication requires converting the internal parallel data into a bit-by-bit stream for serial transmission and then reconverting the serial stream into parallel data at the receiving end (Figure 4.1).

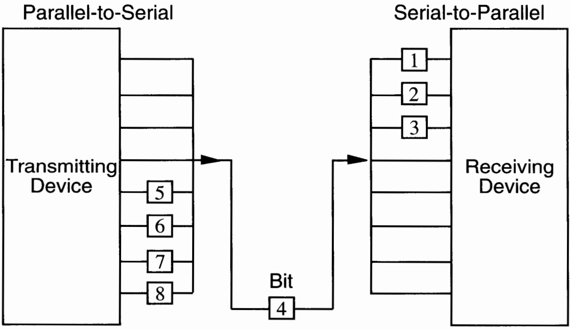


Figure 4.1–Serial data transfer

Although serial communication is the simplest and most common communication link, there are many physical and electrical variations available, including transmission protocols (e.g., synchronous or asynchronous), encoding schemes (e.g., ASCII), connector types (e.g., D-shell, in-line, or hard wired), and electrical characteristics (e.g., voltage or current). In an effort to overcome differences created by the variations, standards such as RS-232-C, RS-422-A, RS-423-A, and ASCII have been developed.

The RS-232-C standard characterizes the transmission protocol, connector type, and electrical characteristics; the RS-422-A and RS-423-A standards characterize the electrical characteristics only. The American Standard Code for Information Interchange (ASCII) standardizes the encoding scheme used for serial communication. RS-232-C and ASCII are the standards most widely supported by PCs, minicomputers, and mainframes. In addition to ASCII, binary encoding is also widely used with RS-232-C. This section provides a detailed discussion of the RS-232-C standard and a brief discussion of the RS-422-A standard, RS-423-A standard, ASCII standard, and binary encoding as related to PCs.

4.2 RS-232-C Interface Standard

The Electronic Industries Association (EIA) publishes several standards for serial communication between data terminal equipment (DTE; e.g., terminals, computers, etc.) and data communications equipment (DCE; e.g., modems). A common example of serial communication between two computers (DTEs) using DCEs (in this case, modems) is shown in Figure 4.2.

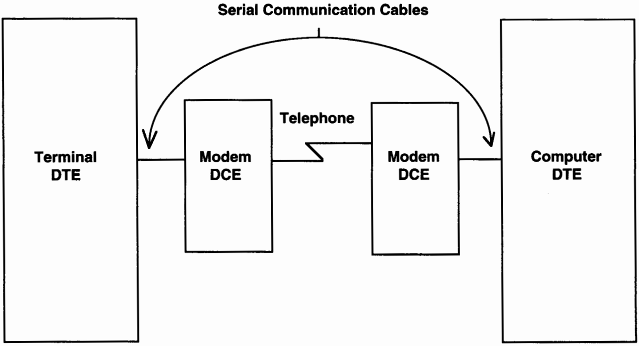


Figure 4.2–Computer-to-terminal connection using modems

One of the prevailing standards is Revised Standard #232, revision C (RS-232-C). The complete standard, available from the EIA, includes mechanical and electrical standards for the connectors, cables, and communication protocols. The document includes four parts: Electrical Signal Characteristics, Interface Mechanical Characteristics (connectors), Functional Description of Interchange Circuits, and Standard Interfaces for Selected Communication System Configurations. The contents of each, as applied to PCs, are discussed in the following paragraphs, followed by guidelines for connecting RS-232-C devices.

4.3 Electrical Signal Characteristics

This section of the standard describes the electrical characteristics that the interface presents to and requires from the outside world, including the voltage levels representing logical 0 and 1. The standard defines a logical '1' as a voltage within the range of -12V to -3V and a logical '0' as +3V to +12V. Voltages in the range from -3V to +3V are considered to be in transition and are undefined. These voltages, combined with the standards for cable impedances, set the practical limit for RS-232-C communication at 80-100m (260-330 ft.).

4.4 Interface Mechanical Characteristics (Connectors)

This section of the standard dictates that the interface consists of a DTE configured plug (computer end of the interface) and a DCE configured receptacle (on the modem end of the interface). The standard specifies connector configuration, including 25 pin number assignments, but not the exact connector. Figure 4.3 shows pinouts for a typical 25-pin D-style connector. Table 4.1 lists RS-232-C standard pinout connections. The IBM PC/AT has adopted a 9-pin D-style connector and a modified pinout (Figure 4.3 and Table 4.2).

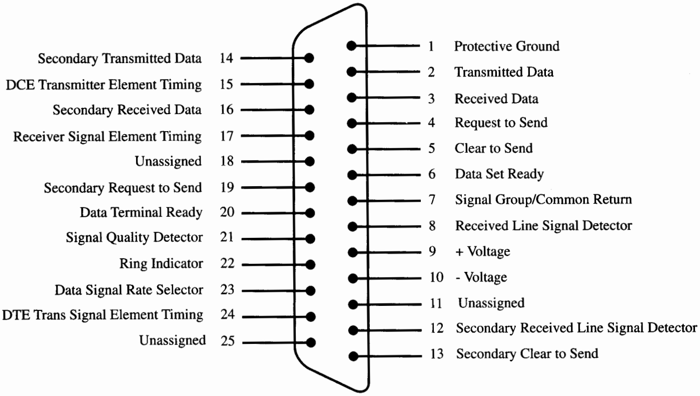

Secondary Transmitted Data 14
DCE Transmitter Element Timing 15
Secondary Received Data 16
Receiver Signal Element Timing 17
Unassigned 18
Secondary Request to Send 19
Data Terminal Ready 20
Signal Quality Detector 21
Ring Indicator 22
Data Signal Rate Selector 23
DTE Trans Signal Element Timing 24
Unassigned 25
1 Protective Ground
2 Transmitted Data
3 Received Data
4 Request to Send
5 Clear to Send
6 Data Set Ready
7 Signal Group/Common Return
8 Received Line Signal Detector
9 + Voltage
10 - Voltage
11 Unassigned
12 Secondary Received Line Signal Detector
13 Secondary Clear to Send

Figure 4.3–RS-232-C 25-pin D-subminiature pinouts

Pin	Signal Name	Circuit	Description
1	Protective Ground	AA	Grounds equipment frames
2	Transmitted Data	BA	Transmits data to DCE
3	Received Data	BB	Receives data from DCE
4	Request To Send	CA	Signals DTE is ready for transmit
5	Clear To Send	CB	Signals DCE is ready to receive
6	Data Set Ready	CC	Signals DCE is on-line
7	Signal Ground (Common Return)	AB	Grounds interchange circuits
8	Received Line Signal Detector	C	Signals DCE has established communication
9	(Reserved for Data Set Testing)	—	
10	(Reserved for Data Set Testing)	—	
11	Unassigned		
12	Secondary Received Line Signal Detector	SCF	Signals DCE has established communication on secondary channel
13	Secondary Clear To Send	SCB	Signals DCE is ready to receive on secondary channel
14	Secondary Transmitted Data	SBA	Transmits data to secondary DCE channel
15	Timing (DCE Source)	DB	Provides transmission timing to DTE when required
16	Secondary Received Data Receiver Signal Element	SBB	Receives data from secondary DCE channel
17	Timing (DCE Source)	DD	Provides receiver timing to DTE when required
18	Unassigned		
19	Secondary Request To Send Data	SCA	Signals secondary DTE channel is ready to transmit
20	Terminal Ready	CD	Signals DTE is on-line
21	Signal Quality Detector	CG	Signals error detection in received data
22	Ring Indicator	CE	Indicates DCE is receiving ringing signal
23	Data Signal Rate Selector (DTE/DCE Source)	CH/CI	Selects between two data transmission rates
24	Transmit Signal Element (DTE Source)	DA	Provides transmission timing to DCE when required
25	Unassigned		

Table 4.1–Standard RS-232-C pin descriptions

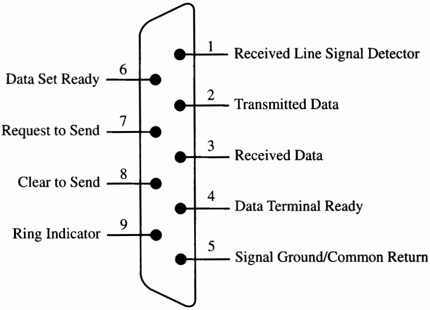


Figure 4.4–Nine-pin D-subminiature style connector pinouts

Pin	Signal Name	Circuit	Description
1	Received Line Signal Detector	CF	Signals DCE has established communication
2	Transmitted Data	BA	Transmits data to DCE
3	Received Data	BB	Receives data from DCE
4	Data Terminal Ready	CD	Signals DTE is on-line
5	Signal Ground (Common Return)	AB	Grounds interchange circuits
6	Data Set Ready	CC	Signals DCE is on-line
7	Request To Send	CA	Signals DTE is on-line
8	Clear To Send	CB	Signals DCE Is ready to receive
9	Ring Indicator	CE	Indicates DCE is receiving ringing signal

Table 4.2–Nine-pin D-shell connector pin descriptions

4.4.1 Ground Pins

The protective and signal ground pins provide chassis to chassis grounding and a common return path for the other interconnect signals, respectively.

4.4.2 Data Transmission Pins

RS-232-C specifies that data is transmitted asynchronously from the transmit (Txd) pin to the receive (Rxd) pin (Figure 4.5). The transmission format includes a start bit, data bits, possibly a parity bit, and one or more stop bits. The rate at which these bits are transmitted is referred to as the transmission baud rate. The transmitter and receiver must use identical formats and baud rates in order for communications to take place. Otherwise, the receiver may receive more or fewer bits than expected, resulting in format errors.

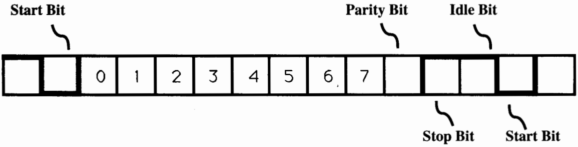


Figure 4.5–Serial transmission format

The secondary transmit and receive data pins are specified for data transmission using a second channel.

4.4.3 Baud Rate

The transmission baud rate determines the rate at which data bits are transmitted. The value, in bits per second, includes, but is not limited to, data bits, start bits, stop bits, and parity bits. Commonly used baud rates are listed in Table 4.3.

4.4.4 Start and Stop Bits

Synchronization bits allow the receiver to detect the start and completion of data transfer. When the transmit line is idle, the transmitter normally holds it in a logical '1' or mark state. When data transfer begins, an initial start bit is sent to wake up the receiver. The start bit is a logical '0' or space. A fixed number of data bits is then transmitted one at a time, the least significant bit first. Optionally, a parity bit is sent after the data. Finally, one or two stop bits are transmitted to delineate the end of data transmission. The stop bit guarantees a minimum idle time (in terms of bits) between transmissions.

Baud Rate	Characters Per Second*
110	11
300	30
600	60
1200	120
2400	240
4800	480
9600	960
19200	1920

*Approximate. Depends on number of start and stop bits and parity setting.

Table 4.3–Commonly used baud rates

4.4.5 Data Bits

Data is transmitted as a series of five, six, seven, or eight bits, with the least significant bit sent first. At least seven data bits are required to transmit ASCII characters. Eight bit data transfers are used to transmit binary data, as the eight bits match the eight-bit byte format commonly used for binary data manipulation. Five and six bit data formats are used for specialized communications equipment.

4.4.6 Parity Bit

The parity bit is an error detection mechanism used to increase the reliability of data transmission over serial communication lines. It is capable of discerning a single-bit error in data transmission. The transmitting device sets the parity to a special value; the receiver checks the value to assure that an error did not occur during transport. Parity checking is important when the communication lines are in proximity to sources of electrical noise, such as motors, fluorescent lights, etc.

The most commonly used types of parity checking are Mark Parity, Space Parity, Odd Parity, and Even Parity. Mark and Space parity, the simplest forms of error checking, set the parity bit to a mark (logical '1') or space (logical '0'), respectively. Odd and Even parity, more sophisticated methods, count the number of mark bits (or logical 1s) in the data and then set the parity bit to force an odd or even multiple, respectively.

For example, if the transmission is specified as even parity and the data being transmitted is 31 hexadecimal (00110001 binary), the parity bit will be set to a mark (logical 1) resulting in 3 mark data bits plus 1 mark parity bit = 4 mark bits, an even number.

At the receiving end, the parity bit is checked against the rest of the data for validity. If no error is found, the byte is accepted. If an error is found, it is discarded and the receiver must deal with the error in its own way, possibly by requesting that the data be transmitted again. This type of error is commonly referred to as a parity error.

Note that the parity bit is able to detect errors of one bit only. An error in two bits may cause the data to have a seemingly valid parity, when in fact it is incorrect.

4.4.7 Control Pins

Several pins are defined to assist in the control of the transmission of data, by signaling the presence of connected equipment and its readiness to receive data. These pins include request to send (RTS), clear to send (CTS), data terminal ready (DTR), data set ready (DSR), and data carrier detect (DCD).

The RS-232-C specification allows for data to be transmitted in either a half duplex or a full duplex configuration. A half duplex configuration allows for only one transmitter to transmit at a given time, much like a CB radio or walkie-talkie. A full duplex configuration allows for both sides to transmit data simultaneously like a telephone.

The RTS and CTS lines were specified to assist half-duplex communication equipment in transmitting and receiving data. Before a transmission, the sender's RTS signal is asserted true (logical 1), requesting the receiver to switch its circuitry to the receive mode. When ready to receive, the receiver asserts its CTS line true, allowing transmission to begin.

The DTR and DSR lines were specified to indicate the presence and readiness of data terminal and data communication equipment. The DTR is asserted (logical 1) by the terminal equipment when terminal power is on, indicating to the modem or other DCE that the terminal is ready. DSR is asserted (logical 1) by the modem or other DCE allowing the terminal to go on-line and receive data.

The DCD (data carrier detect or receive line signal detect) signal is asserted (logical 1) by the modem or other DCE to indicate that it has established a communication link with the modem or DCE at the other end of the communication link (e.g., phone line). It must be asserted for the terminal to go on-line and receive data.

The ring indicator pin indicates that a ringing signal is being received on the communication equipment. On receiving this signal, the DTE equipment normally asserts the Data Terminal Ready signal to answer the call.

The control signals on the data terminal equipment are specified to connect to the corresponding handshaking signals on the data communications equipment.

4.4.8 Timing Pins

The timing circuits are provided to allow communication where standard bit rates are not in use. Either the receiver provides a timing signal to the transmitter (DB) or the transmitter provides a timing signal to the receiver (DA) to allow data synchronization. Timing circuits are not normally required for PC-based serial communication.

4.5 Standard Interfaces for Selected Communication System Configurations

This section of the RS-232-C standard describes a selected set of data transmission configurations. The simplest of these is a transmit only or receive only configuration; for example, a PC (transmit only) connected to a serial printer (receive only). The duplex primary channel/duplex secondary channel configuration is the most complex configuration, utilizing both primary and secondary channels. The most commonly used configuration is the duplex configuration; for example, a modem connected to a PC. The standard interface configurations are listed in Table 4.4.

Data Transmission Configurations	Interface Types
Transmit Only	A
Transmit Only*	B
Receive Only	C
Half Duplex	D
Duplex	D
Duplex	E
Primary Channel Transmit Only* / Secondary Channel Receive Only	F
Primary Channel Transmit Only / Secondary Channel Receive Only	J
Primary Channel Receive Only / Secondary Channel Transmit Only*	G
Primary Channel Receive Only / Secondary Channel Transmit Only	I
Primary Channel Transmit Only* / Half Duplex Secondary Channel	J
Primary Channel Receive Only / Half Duplex Secondary Channel	K
Half Duplex Primary Channel* / Half Duplex Secondary Channel	L
Duplex Primary Channel* / Duplex Secondary Channel*	L
Duplex Primary Channel / Duplex Secondary Channel	M
Special (Circuits specified by supplier)	Z

Note: Data Transmission Configurations identified with an asterisk (*) indicate the inclusion of Circuit CA (Request to Send) in a One Way Only (Transmit) or Duplex Configuration where it might ordinarily not be expected, but where it might be used to indicate a non-transmit mode to the data communication equipment to permit it to remove a line signal or to send synchronizing or training signals as required.

Table 4.4–RS-232-C standard interface configurations

4.6 Connecting RS-232 Devices

4.6.1 Handshaking

RS-232-C sets standards for communicating between data terminal equipment and data communications equipment. However, it does not provide for communication between two computers or between a computer and other peripheral equipment (e.g., printer, plotter, or scanner). Additionally, as specified, its handshaking signals do not provide enough functionality to reliably sustain high speed full duplex communications (e.g., computer to computer). Therefore, the functionality of certain handshaking pins has been modified by various manufacturers to overcome these limitations. The manufacturers' variations have not been standardized and, therefore, tend to complicate the task of connecting equipment and establishing communications.

The RS-232-C standard was defined to control data transmission at relatively slow speeds. Input from the terminal keyboard, at typing speeds, could be easily processed by the computer and output to the terminal screen or printer. The output speed was controlled by the computer, through the use of null characters or delays, to prevent overrunning the output device.

Connection between two computers requires higher data transmission speeds. Often the receiver can only receive a limited number of characters before it must stop to process them rather than risk the loss of information. When the receiver can no longer receive, it must communicate this to the transmitter. The transmitter responds by halting data transmission. When the receiver is again able to receive, it informs the transmitter, which then responds with data transmission. This process is known as handshaking or data flow control.

Handshaking is classified into two broad groups, hardware and software. Software handshaking techniques use control characters to control data flow, whereas hardware techniques utilize handshaking pins.

4.6.2 Software Handshaking

Software handshaking requires only three wires for implementation: a ground wire, a transmit wire, and a receive wire. It is commonly used when communicating over telephone lines because the additional lines required for hardware handshaking are not present.

A common implementation of software handshaking is the Xon-Xoff scheme in which an ASCII DC3 (HEX 13) character is transmitted by the receiver to indicate that data transmission should be halted and an ASCII DC1 (HEX 11) character is transmitted to indicate that data transmission may continue. This scheme works well with ASCII encoded data but will not work with binary encoded data. Binary data may contain the HEX 13 and HEX 11 values, making it impossible to differentiate between control characters and binary data. Another disadvantage of the Xon-Xoff scheme is its slow response time. The receiver may receive many characters after sending the signal to stop transmission.

4.6.3 Hardware Handshaking

Hardware handshaking utilizes additional signal lines, including the request to send (RTS) and clear to send (CTS) lines. When the receiver's RTS output signal is asserted true, the receiver is ready for data. This signal is connected to and monitored by the transmitter's CTS input signal, so that when the transmitter's CTS is asserted true, the transmitter may send data. When the receiver is not ready for data, it unasserts its RTS signal, which in turn unasserts the transmitter's signal, halting data transmission.

Most equipment utilizes the RTS and CTS signals as described above, while the DTR/DSR lines operate as specified in the RS-232-C standard and described earlier in this section. However, many configurations exist in which the DTR/DSR signals are modified to operate as discussed in the above paragraph, while the RTS and CTS signal operate as specified in the RS-232-C standard. It is necessary to study the manufacturer's documentation to determine pin functionality of each piece of equipment before attempting to establish communications between devices.

4.7 Cable Connections

The following paragraphs identify cable connections for some common RS-232-C configurations.

4.7.1 DTE to DCE

A DTE device directly connects to a DCE device in that the RTS connects to the RTS, the CTS connects to the CTS, etc. A simple transmit-only/receive-only configuration cable diagram is illustrated in Figure 4.6. Figure 4.7 illustrates a duplex configuration cable diagram.

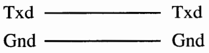


Figure 4.6–Simple DTE to DCE cable diagram

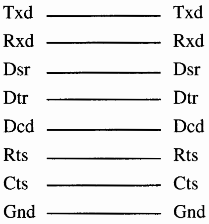


Figure 4.7–Duplex DTE to DCE cable diagram

4.7.2 DTE to DTE

The direct connection of two DTE configured devices or of two DCE configured devices without the aid of two modems and a telephone line, can be accomplished using a NULL modem cable. These are special cables in which the logical signals are swapped in order to guarantee that input pins on one machine are connected to output pins on the other. Most serial communications equipment uses a DTE configured arrangement. Two of the more common DTE to DTE configuration cable diagrams are illustrated in Figures 4.8 and 4.9.

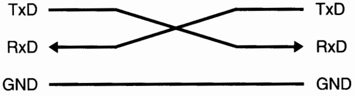


Figure 4.8–Null modem (DTE to DTE) cable diagram (duplex without hardware handshaking)

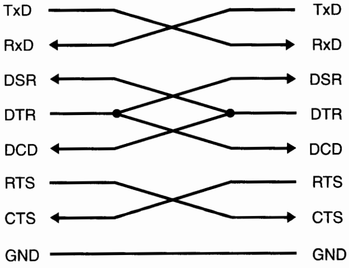


Figure 4.9–Null modem (DTE to DTE) cable diagram (duplex with hardware handshaking)

4.8 Establishing the Connection

In order for communication to take place between two RS-232-C devices, certain basic criteria must be met. First, the transmit and receive signals must be properly connected, and a signal ground must be provided. In addition, the communication formats of both pieces of equipment must be configured identically. Also, if hardware handshaking is involved, other signal connections must be properly made.

If communication cannot be established, several possibilities exist:

- **Data are not being transmitted.**
  – Transmitter handshaking signals are not properly connected. (Many devices will not transmit if the CTS and/or DCD and/or DSR signals are not asserted true.)

- **Data are not being received.**
  – Receiver handshaking signals are not properly connected. (Some devices will not receive if the DCD and/or DSR signals are not asserted true.)
  – Serial formats and/or baud rates do not match data.

- **Transmit pin is not connected to receive pin.**
  – Transmit pin is connected to transmit pin, or receive pin is connected to receive pin, preventing data transmission.

If communication is established, but data is incorrect or garbled, the cause is probably a format or baud rate mismatch.

If communication is established, but data is sometimes lost, the cause is probably a data overrun error; i.e., the transmitter sent data when the receiver was not ready to receive. This condition indicates the need for handshaking or for the correct implementation of handshaking already in use, or, in the worst case, for a lowering of the baud rate.

4.9 Other Serial Communication Standards

4.9.1 RS-422-A Interface Standard

The RS-422-A interface standard covers the electrical characteristics of balanced voltage digital interface circuits. This standard specifies that a pair of signals be used to transmit data, rather than one signal, as specified in the RS-232-C standard. This pair consists of a non-inverted (A) and an inverted (B) data signal. The voltage differential between A and B is within the range of 2V to 6V. A is negative with respect to B for a logical 1 (Mark) state. A is positive with respect to B for a logical 0 (Space) state. These voltages, combined with the standards for cable impedances, set the practical limit for RS-422-A communication at 1200m (3960 ft.) when using data rates below 90 Kbits/sec and allow data rates of up to 10 Mbits on shorter cables.

Using RS-422-A standards rather than RS-232-C standards may prove advisable if:

- The required data rates are higher than RS-232-C will reliably support.
- The interconnecting cable is longer than specified for RS-232-C operations.
- The interconnecting cable is exposed to extraneous noise sources that may cause errors using RS-232-C standards.
- It is necessary to minimize interference from other signals within the interconnecting cable.
- Inversion of signals is required (RS-422-A provides for inverted and non-inverted signals.)

4.9.2 RS-423-A Interface Standard

The RS-423-A interface standard covers the electrical characteristics of unbalanced voltage digital interface circuits. This standard specifies that a single transmission line be used, as in RS-232-C. However, the electrical characteristics are improved, allowing transmission at higher data rates over longer cable lengths. The voltages are within the range of 0V to 6V where 0V is a logical 1 (Mark) state and 6V is a logical 0 (Space) state.

Signals of the RS-232-C standard are often used to provide control signals on the same cable where RS-423-A signals are used for data and timing.

4.9.3 ASCII Interface Standard

The most commonly used standard for encoding information for data transmission is prescribed by the American Standard Code for Information Interchange (ASCII), also known as the American National Standards Institute (ANSI) standard X3.4-1977.

The ASCII code uses seven bits per character, usually stored in the least-significant seven bits of an eight bit byte. The ASCII code provides for 128 characters, which allows for all upper and lower-case letters, numerals, punctuation marks, and 32 non-printable characters (control characters) often used to signal special conditions for controlling serial communications devices. Examples of the non-printable control characters include carriage return, line feed, form feed, bell, and null.

The ASCII standard is the most widely used scheme for encoding human-readable information (e.g., documents).

4.9.4 Binary Encoding Scheme

Information other than text, such as computer programs and numerical data, requires eight bits for the representation of a byte and cannot be efficiently expressed with the ASCII encoding scheme. This data is encoded in a binary format that directly transmits raw bytes of data.

The IBM PC uses binary encoding to extend the ASCII character set from 128 to 256 characters to accommodate non-English alphabets and special graphic characters, the first 128 characters being identical to their ASCII equivalents. Other communications equipment uses the binary data format to transfer numerical data.

4.10 Serial Communication, IEEE 488, and Instrument Control

Serial communication is commonly used to connect to peripherals that require neither high speed data transfer nor sophisticated control; thus in data acquisition applications low-end instruments are commonly offered with serial control. In general, serial interfacing requires the user to provide, in his or her program, the data flow and instrument function control inherent in the IEEE 488 bus.

4.11 Language Interfacing

For languages and applications that provide serial port support, such as GWBASIC or Microsoft QuickBASIC for the IBM PC, serial communication begins with opening the serial port:

```
10 OPEN "COM1: 9600,N,8,1" as #1
```

or with opening the first serial port at 9600 baud, no parity, 8 data bits, and 1 stop bit. Communication then consists of writing to and reading from the serial port:

In this example the state of the IOtech Control488/16 Power Control Interface is being examined:

```
20 FOR N = 1 to 10
30 PRINT #1,"R?"         ' Request a reading
40 LINE INPUT #1, A$     'Get reading from Control488/16
50 PRINT A$              'Print the reading to the screen
60 NEXT N
70 END
```

BASIC and FORTRAN directly support the COM or serial ports on the IBM PC; however, C and Pascal do not. The programmer must write or purchase low-level routines to directly access the COM port hardware and BIOS. An attractive alternative to the data acquisition user is IOtech's Driver488, supplied with the Personal488 and Power488 IEEE controllers. Driver488 is not only an IEEE 488 driver; it is also a COM port driver, allowing the user to configure and install his or her COM port as a DOS device, just as it permits installation of the IEEE 488 controller as a DOS device.

This approach allows the user to open, write to, and read from the COM port as if it were a DOS file. Thus COM port support is provided for C and Pascal compilers on the IBM PC, as well as for applications programs such as Lotus 1-2-3 that do not have built-in COM port support. If the software can read and write DOS files, Driver488 permits complete access to serial communication while handling the COM port configuration chores "off-line."

An added benefit of Driver488's DOS device driver approach is that "mixed systems" of IEEE 488 and serial instruments are handled very conveniently by a single driver with a uniform software interface. User programs can access both IEEE and serial devices using the same procedures and mechanisms.

4.11.1 Enhancements to Serial Instrument Control

Unfortunately, serial communication protocols do not provide such useful IEEE 488 control functions as SRQ , EOI, GET, and serial and parallel polling. Instrument designers have been forced to create "work-arounds" to provide some of the functionality inherent in the IEEE 488 standard. To meet this need, IOtech has introduced Personal488/OEM-PR (for the OEM and instrument designer using an embedded-IBM PC approach), which includes both compact IEEE 488 peripheral control routines and a set of enhanced serial control routines. These serial routines blend some of the added functionality of IEEE 488 into serial control, allowing the instrument designer to provide more convenient control of the final product.

4.11.2 IEEE488/Serial Conversion

For mixed systems with combinations of computers, IEEE 488 instruments, and serial instruments, as well as serial and IEEE 488 peripherals (such as printers and plotters), a wide range of products is available to provide interface conversion and protocol conversion capabilities. For example, IOtech's Mac488B allows any Macintosh with a serial port to become a full-featured IEEE 488 controller. The Serial488A and Serial488/4 (Figure 4.10) provide 1 channel and 4 channels of RS-232/422 control, respectively, by converting the IEEE 488 bus to serial communication. For the IBM PC, the COM488 board allows a PC COM port to communicate with an IEEE 488 printer. The result is to make the design and implementation of mixed-protocol systems practical and convenient.

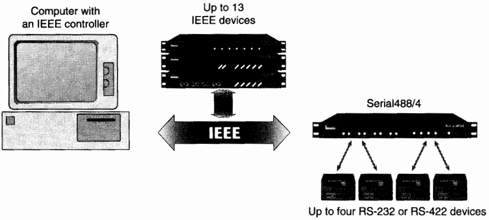


Figure 4.10–Serial488/4 serial-to-IEEE 488 converter

4.12 Summary

Serial communication is the most basic form of computer communication. It is economical and widely used for communications between computers and peripheral equipment. Many serial communication standards exist. RS-232-C and ASCII have emerged as the most prevalent standards. Together they provide for electrical, mechanical, functional, and data encoding standards to fit most low-end communication needs. Hardware and software is now available to allow serial and IEEE 488 instrument control to coexist comfortably and conveniently in a single data acquisition system.

SCSI

5.1 General

The Small Computer System Interface (or SCSI) is a system level interface designed to allow communication between small computer systems and their peripherals. It was originally developed by Shugart Associates as the Shugart Associates System Interface (SASI) for use as a controller interface for a line of hard disk products. In 1979, the bus was standardized by the American National Standards Institute (ANSI), renamed SCSI, and expanded to allow for more features.

The final version of the ANSI specification was released in 1986. This standard is generally referred to as SCSI-1 since work was immediately started on a new, enhanced version of the SCSI specification, which is generally referred to as SCSI-2.

The key features of the SCSI bus include:

- high transfer speed – up to 5 Mbytes/sec on SCSI-1, up to 20 Mbytes/sec on SCSI-2
- up to 8 devices connected to one bus
- bus length to 6m with single-ended drivers, up to 25m with differential drivers
- bus width up to 32 bits with SCSI-2
- system level interface with standard command definitions

As a system level interface, SCSI defines not only the signal levels and logical functions of electrical signals, but also describes communication protocols and command sequences. This promotes device independence by hiding implementation specific features of the peripherals attached to the SCSI bus below the system level command definitions. A host controller on the SCSI bus may communicate to a floppy disk drive, a hard disk drive, or an optical disk drive through the same sequence of system level commands on the SCSI bus without regard to the internal details of the specific device such as number of tracks, number of heads, or number of bytes per track. New technologies may also be incorporated into existing SCSI systems without major changes to the interfacing software or hardware as long as the new technology incorporates the standard SCSI system level interface.

SCSI has proven to be a very popular means of connecting computers and workstations to peripherals. While SCSI data acquisition devices remain rare, SCSI ports on the Macintosh and Sun, DEC, and NeXT workstations have proven to be an effective and popular means of connecting slotless computers to IEEE 488 instruments and devices via SCSI-to-IEEE 488 converters such as IOtech's MacSCSI488, SCSI488/S (for Sun), SCSI488/D (for DEC), and SCSI488/N (for NeXT) products. In addition, general purpose SCSI/IEEE converters such as IOtech's SCSI488 (Figure 5.1) are available to adapt almost any computer with a SCSI port to the IEEE 488 bus.

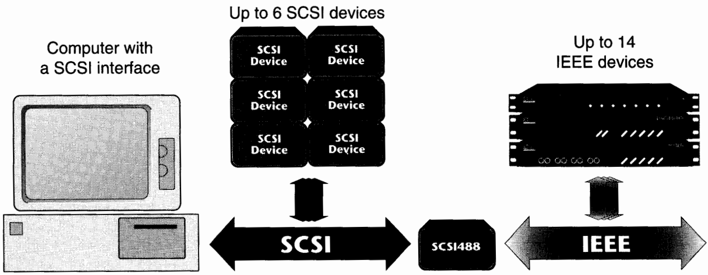


Figure 5.1–IOtech SCSI488 SCSI to IEEE converter

5.2 Mechanical

SCSI devices are daisy-chained together on the bus (Figure 5.2), with terminators at each end of the bus to minimize electrical reflections of the signals from discontinuities associated with an unterminated bus end.

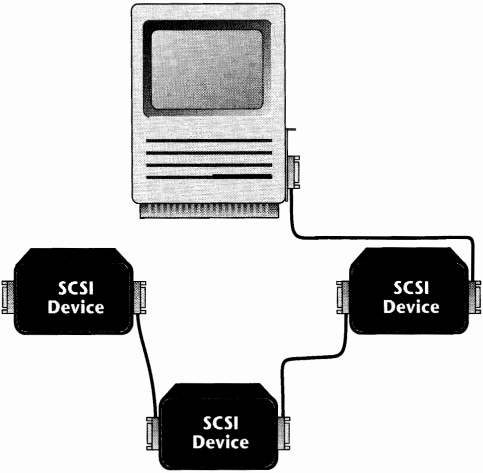


Figure 5.2–SCSI device daisy chaining

The SCSI-1 specification provides two alternatives for the cabling of the bus between units on the bus. The first alternative was intended for interconnection of SCSI devices within a controlled environment (i.e., interconnection of a computer and disk drives within a chassis) where noise and ruggedness were not a problem. For these applications, 50-pin IDC (ribbon cable) connectors and cables were specified (Figure 5.3).

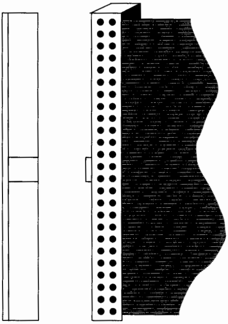


Figure 5.3–SCSI connector, alternative 1

The second alternative was intended for interconnection of devices between different cabinets or chassis (i.e., interconnection of a computer and separate cabinets containing disk drives, tape drives, etc.) where resistance to electrical noise and mechanical ruggedness are of prime importance. For these applications, 50-pin, ribbon-contact, D-shell connectors with bail-locks were specified (Figure 5.4).

Of course, the SCSI standard's provision of two types of connectors did not stop manufacturers from adopting their own. Probably the best known of these are Apple's 25-pin D-shell connector on the Macintosh line, and the 50-pin D-shell connector used on Sun workstations.

The new SCSI-2 specification allows up to four times the bus width of the SCSI-1 standard and brings in the extra data bytes with their control signals over a separate cable. The new specification also provides the use of high-density 50- and 68-pin subminiature D-shell connectors as two additional alternatives.

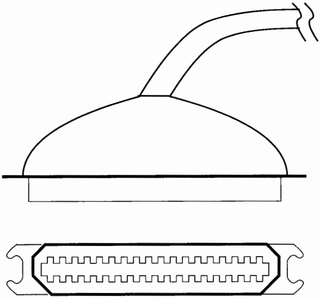


Figure 5.4–SCSI connector, alternative 2

5.3 Electrical

To provide for different bus usage environments, two methods of driving/receiving the SCSI bus signals were specified.

5.3.1 Single-ended Mode

In this mode, all signal lines on the bus are driven with high-current open-collector drivers. The terminators provided at both ends of the SCSI bus provide a pull-up function to force the bus into an inactive state when not being driven. All signals on the bus are defined as low-true.

Single-ended SCSI buses are limited to 6m in length; the minimum recommended distance between two devices is 0.1m. Most SCSI bus implementations are single-ended.

5.3.2 Differential Mode

In this mode, each signal is actually driven over two wires, a true and not-true sense. Differential receivers sense both lines to determine the difference between them (hence the name). This alternative offers greater noise-immunity and signal-degradation free operation over longer distances than does the single-ended mode alternative. Differential driven SCSI buses are limited to a length of 25m.

5.4 SCSI Bus Lines

There are a total of 18 signals on the SCSI bus. Nine are used for data transfer (eight data bits plus an odd parity bit) and nine are used for control. These signals are described below:

BSY (Busy) an OR-tied signal that indicates the bus is in use

SEL (Select) used by the initiator to select a target or by a target to re-select an initiator

C/D (Command/Data) driven by the target to indicate whether control or data information is to be transferred on the data bus

I/O (Input/Output) driven by the target to control the direction of data transferred on the data bus

MSG (Message) driven by the target during the message phase

REQ (Request) driven by a target to indicate a request for a data transfer handshake.

ACK (Acknowledge) driven by an initiator to indicate acknowledgment for a REQ/ACK handshake

ATN (Attention) driven by an initiator to indicate an attention condition to the target

RST (Reset) an OR-tied signal that resets the SCSI bus and all devices on the bus

DB0-7, P (Data bus) eight data bus signals, plus an associated parity bit

Transfer of all data and commands takes place over the data lines. The exact meaning of the data bytes and the direction of transfer is determined by the target via the control lines C/D, I/O, and MSG.

5.5 Handshake Lines

The lowest level of the SCSI transfer protocol is byte handshaking. For every byte transferred in either direction, a handshake occurs between the Request (REQ) and Acknowledge (ACK) lines.

Two different methods of handshaking on the bus are allowed, asynchronous and synchronous:

The **asynchronous** handshake process (Figure 5.5) starts when the target asserts REQ. When the initiator senses REQ, it responds by reading or writing a byte to or from the data bus and then asserting the ACK line. When the target senses the ACK line, it responds by de-asserting the REQ line. When the initiator senses the negation of the REQ line, it responds by de-asserting the ACK line. The bus is then set up for another REQ, ACK handshake to take place. This handshake process is very reliable, requiring verification that each end has seen the change in signal before initiating another change. Using asynchronous handshaking, transfer rates of 1.5 Mbytes/sec are achievable.

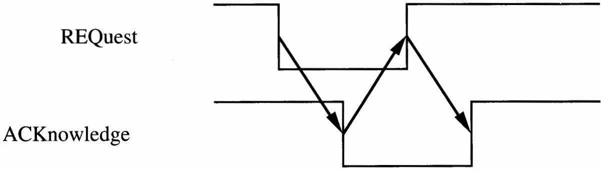


Figure 5.5–Asynchronous transfer handshake

The **synchronous** handshake process (Figure 5.6) starts when the target asserts REQ. Instead of waiting for the initiator to assert ACK, however, it de-asserts the REQ line after a minimum delay time which can be programmed by commands over the SCSI bus. When the initiator has recognized the REQ line transition, it sends an ACK pulse to signal reception of the data byte.

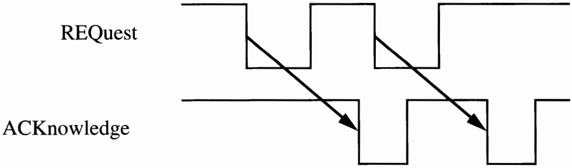


Figure 5.6–Synchronous transfer handshake

The number of ACK pulses must equal the number of REQ pulses for an entire transfer sequence. Since cable propagation delays are ignored in synchronous handshaking, transfer rates of 5 Mbytes/sec are achievable.

5.6 SCSI Communication Protocol

Communication on the SCSI bus occurs between an "initiator" and a "target." As its name suggests, an initiator on the SCSI bus originates an operation. The target performs the operation. Communication on the SCSI bus is allowed between only two devices at any given time. One acts as the originator, selecting and commanding a target, which performs the desired operation.

When communication is established between an initiator and a target, commands to the target are passed in a standard format called a Command Descriptor Block (CDB). The SCSI standard specifies the format and meaning of CDBs for each type of SCSI peripheral.

The SCSI bus allows a maximum of 8 devices, each with a unique bus address between 0 and 7. These devices can be any mixture of initiators and targets. The physical SCSI bus is chained from one SCSI device to another and resistively terminated at the far ends to minimize transmission line effects.

5.7 SCSI Bus Phases

Every SCSI communication transaction sequences through several distinct phases on the bus. The bus can never be in more than one phase at a time. After selection, phase changes are controlled by the target through the C/D, I/O, and MSG control lines on the bus.

5.7.1 Bus Free Phase

In the bus free phase, no SCSI devices are using the bus, and the bus is available for another SCSI operation. All SCSI communications start from a bus free phase.

5.7.2 Arbitration Phase

The arbitration phase will occur only in those SCSI systems with more than one initiator or a reselecting target. Since multiple initiators are allowed on the bus, there will be times when two initiators try to access the SCSI bus at the same time. To arbitrate collisions between two SCSI initiators, priority is assigned with the SCSI address on the bus. Highest priority is given to address 7 and lowest priority given to address 0.

A data bit is assigned to each SCSI address for use during the arbitration phase, and it corresponds to each device's address on the bus (i.e., DB0 is used by address 0, and DB7 is used by address 7). During the arbitration phase, each SCSI device desiring the bus asserts BSY, a wired-OR signal, and also asserts its data bit corresponding to the address it occupies on the SCSI bus. If, after 2 μsecs, there are any data bits asserted higher than its own, that device must release the bus and wait for another bus free phase.

5.7.3 Selection Phase

After winning arbitration, an initiator proceeds to the selection phase. In this phase, the initiator asserts the SEL signal on the bus and asserts the data bits corresponding to both its own address and the address of the target it wishes to select. Thus, a target can determine which SCSI address the initiator occupies.

5.7.4 Command Phase

During the Command phase, the target requests command information from the initiator. The command information is in the form of a Command Descriptor Block. The Command Descriptor Block contains the information necessary to execute any of the defined operations for a particular class of SCSI device in a standard format.

5.7.5 Data Phase

During the data phase, the target transfers any data requested by the Command Descriptor Block loaded previously. This phase is optional. A number of SCSI commands require no data transfer or will skip the data transfer phase if an error is detected.

5.7.6 Status Phase

The status phase immediately follows the data phase or, if no data transfer was required, the command phase. Status information about the last transfer is sent to the initiator.The status returned is one byte long. Bits within this byte are pre-defined with certain meanings by the SCSI specification. Three bits are also left undefined for vendor-unique implementations. The organization of the status byte along with the three most important status codes is shown in Table 5.1.

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Status Codes
R	V	V	S	S	S	S	V	= General Definition
R	V	V	0	0	0	0	V	= Good
R	V	V	0	0	0	1	V	= Check Condition
R	V	V	0	1	0	0	V	= Busy
			Key:	R = Reserved S = Predefined SCSI bit V = Vendor unique				

Table 5.1–Status byte organization

"Good" status indicates the preceding operation was successful. "Check Condition" usually indicates that something unusual happened during the last SCSI operation. The initiator should issue a "Request Sense" command to discover what the unusual event was. "Busy" indicates that the target cannot accept any commands and the initiator should try again later.

5.7.7 "Message In" Phase

The message phase is the last phase in a SCSI transfer in which a message byte is transferred from the target to the initiator. After the transfer of a message byte to the initiator, the target will disconnect from the SCSI bus and return the bus to bus free phase. At this point the SCSI bus is ready for another SCSI operation to be initiated.

5.7.8 "Message Out" Phase

From the moment the initiator has completed the selection phase, the target has complete control of the data transfers and phase changes on the bus. If the initiator has reason to regain control of the bus (for example the detection of a parity error in data received from the target) the only mechanism for doing so is the ATN line. If a target detects the assertion of ATN at any time during a SCSI operation, it will go to a "message out" phase at its convenience. The initiator may then send a message byte (if the ATN line is kept asserted, several message bytes may be sent) to force the target to abort or retry a SCSI operation.

There are many message bytes defined in the SCSI specification for specific meanings and actions on the SCSI bus. However, only one is mandatory in any SCSI implementation–a message byte of 00, which indicates "Command Complete."

5.8 Reconnection

In order to prevent slow operations from occupying the SCSI bus for long periods of time, some peripherals on the SCSI bus support a feature called reconnection. During slow operations (for example, a long seek on a disk device) the target will accept the CDB, describing the operation required, and then deliberately disconnect from the SCSI bus, freeing the bus for other operations between different initiators and targets or between the same initiator and another target. When the target is ready for the next phase of the SCSI transfer, it will act as an initiator to reselect the originator of the SCSI operation, then resume the role of target. The target will resume the operation at the next phase, either transferring any data associated with the CDB issued, or going into status, message phases to complete the operation. Figure 5.7 shows the state of all signals during one complete transaction on the SCSI bus.

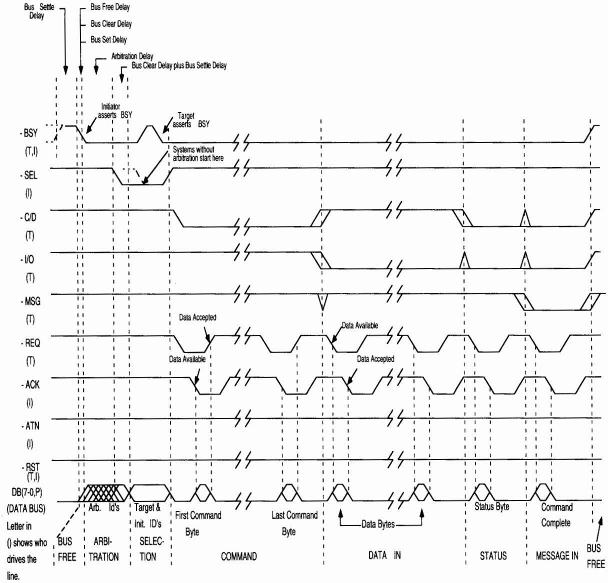


Figure 5.7–One complete transaction on the SCSI bus

This material is reproduced with permission from American National Standard X3.131-1986, also entitled "American National Standard for Information Systems—Small Computer System Interface (SCSI)," copyright © 1986 by the American National Standards Institute. Copies of this standard may be purchased from the American National Standards Institute at 11 West 42nd Street, New York, NY 10036.

5.9 Command Descriptor Blocks

All requests to a peripheral device on the SCSI bus are performed by sending a "Command Descriptor Block" (CDB) to the target during the command phase of a SCSI operation. For each type of device on the SCSI bus, the SCSI specification defines a minimum set of commands and the exact format of the CDBs for these commands. In order to meet compliance with the SCSI specifications, a device must execute at least this minimum command set. The pre-defined command descriptor blocks are part of what makes the SCSI bus powerful and popular. Using this mandatory command set, a software driver could be written to drive virtually any SCSI device. This is part of what differentiates the SCSI bus, a system interface, from device level interfaces. SCSI specifies not only the signal levels and timing for an electrical interface, but also the higher level commands that any device on the SCSI bus must respond to by definition.

The exact format and meaning of the bytes in a CDB depend on what kind of device is being accessed through the SCSI bus. Currently, six different types of devices are covered in the SCSI specification, along with a minimum mandatory command set for each. These six device types are:

Device type	**Example**
Direct access	floppy disk drive, hard disk drive
Sequential access	tape drive
Printer	dot-matrix, laser printers
Processor	any intelligent device
WORM	(write-once, read-multiple) optical media
Direct access, read-only	CD-ROMs

Command descriptor blocks (as defined by the SCSI specification) come in three sizes; 6 bytes long, 10 bytes long, and 12 bytes long. The first byte in any CDB is an operation code, and part of the operation code specifies what the CDB length is. The operation code byte is split up as shown in Figure 5.8:

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Group Code			Command Code				

Figure 5.8—Operation code byte

The most significant three bits are used for the group code. Currently, three group codes are defined: group 0, group 1, and group 5. Group 0 CDBs are always 6 bytes long, group 1 CDBs are always 10 bytes long, and group 5 CDBs are always 12 bytes long. Groups 2 through 4 are reserved for future SCSI specification, and groups 6 and 7 can be used for vendor-unique CDB implementation.

The five-bit command code field within the opcode byte provides up to 32 commands in each group. The formats for 6, 10, and 12 byte CDBs are shown in Figures 5.9, 5.10, and 5.11.

Bit

Byte

	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	operation code							
1	LUN			starting block # (if needed) MSB				
2	starting block # (if needed)							
3	starting block number (if needed) LSB							
4	transfer length (if needed)							
5	Control byte							

Figure 5.9–Typical group 0 (6-byte) CDB

Bit

Byte	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	operation code							
1	LUN			reserved				rel
2	starting block number (if needed) MSB							
3	starting block number (if needed)							
4	starting block number (if needed)							
5	starting block number (if needed) LSB							
6	reserved							
7	transfer length (if needed) MSB							
8	transfer length (if needed) LSB							
9	control byte							

Figure 5.10–Typical group 1 (10-byte) CDB

Bit

Byte	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	operation code							
1	LUN			reserved				rel
2	starting block number (if needed) MSB							
3	starting block number (if needed)							
4	starting block number (if needed)							
5	starting block number (if needed) LSB							
6	reserved							
7	reserved							
8	reserved							
9	transfer length (if needed) MSB							
10	transfer length (if needed) LSB							
11	control byte							

Figure 5.11–Typical group 5 (12-byte) CDB

5.10 The Future of SCSI

Computer performance is continually reaching new heights. Machines are now being sold that execute instructions at the rate of 10 to 15 MIPS (millions of instructions per second). Newer processors such as the 68040 from Motorola are being released with execution rates of 20 MIPS and promise to grow even faster with increases in clock rate. A new ECL version of the SPARC processor used in Sun workstations promises execution rates of 80 MIPS.

The only way processors can sustain such high execution rates is by careful design that couples the memory and processor together very tightly. This tight coupling precludes the use of a standard bus with card slots, which have traditionally been used to allow expansion of systems. Such a bus would slow down the critical path between a processor and its memory. The SCSI bus is a natural extension for an otherwise closed architecture which still allows relatively high performance for data transfer.

Higher speed computing has given rise to another problem. Since the CPU spends a lot less time computing, the handling of peripherals is an increasing drain on computing resources. The trend has been to include more intelligence in each peripheral to off-load more of the lower level tasks from the main CPU. SCSI devices are by their very nature intelligent, executing higher level commands as defined by the SCSI bus specification.

The SCSI bus has undergone several enhancements over the past few years. Although the original specification limited transfers to 1.5 Mbytes/sec, later enhancements allow up to 5 Mbytes/sec. With the formal adoption of SCSI-2, the data bus width will be 16 or 32 bits wide, allowing data transfer rates up to 20 Mbytes/sec, faster than many of the original computer buses allowed.

At the same time, more peripherals are finding their way onto the SCSI bus. For example, with the advent of high quality graphics engines for printing and the increasing demands of desktop publishing on printers, it seems only natural that Apple's laser printers use the SCSI bus for communication. Virtually every disk technology is available with a SCSI interface as an option, and in some cases, such as CD-ROM, SCSI is the **only** interface available for computer applications. As noted earlier, SCSI also provides a convenient and effective means of connecting workstations to the IEEE bus, allowing these new technologies to be used for the data acquisition task.

With its large data transfer rates, wide acceptance in the marketplace, high degree of portability among dissimilar systems, and ease of interface, the SCSI bus should remain viable well into the 90s.

Ethernet

6.1 Introduction

Computer networks provide a means by which many different computers and peripherals can communicate across a common connection. A network is essentially a system of processing units connected by a single communication link. A local area network (LAN) generally is a network that supports peer-to-peer communications over a distance of several meters to several kilometers. *Peer-to-peer communication* refers to the ability of each station (a computer, terminal, or peripheral device) to communicate with any other, without requiring intervention from a central controller. The distances involved allow the connection of systems within a building or a small group of buildings.

6.2 The Ethernet Standard

6.2.1 History

Ethernet is one of the most commonly used LANs. The first Ethernet system was developed by the Xerox Corporation in the mid 1970s. DEC and Intel supported Xerox in the quest for a network standard, and the three companies worked together on the first version of the Ethernet standard, published in 1980. This was followed with version 2.0 in 1982.

6.2.2 The IEEE 802.3 Standard

The Institute of Electrical and Electronics Engineers (IEEE) used Ethernet as the basis for the 1988 IEEE 802.3 standard. The 802.3 standard specifies the physical and electrical characteristics, media access method, and the data encoding and formatting methods of a 10 Mbit/sec serial data bus. IEEE 802.3 specifies the physical and electrical characteristics of three types of media.

Although IEEE 802.3 and Ethernet differ in the definition of their frame formats, the two can be made to coexist on the same network, and the term Ethernet is generally used to refer to either system. In the following pages, Ethernet and 802.3 are used interchangeably, except in reference to unique aspects of either.

6.2.3 802.3 Type 10BASE5

IEEE 802.3 Type 10BASE5 defines the physical and electrical characteristics of a shielded 50 ohm coaxial cable. A single segment of such a cable can be up to 500m in length, and repeaters can be used to extend the network to a maximum of 2500m. The 10BASE5 standard allows 100 nodes per segment. This standard is commonly referred to as thick Ethernet.

In a 10BASE5 network, the workstations, or data terminal equipment (DTE), are often far from the coaxial cable connection. In the case outlined in Figure 6.1, a transceiver drop cable, of up to 50m over twisted pair wiring, is required to connect the DTE with the coaxial transceiver, or medium attachment unit (MAU). This DTE/MAU interface is called the attachment unit interface (AUI), and thus the transceiver drop cable is often referred to as an AUI cable. A 15 pin male D shell connector is used at the transceiver (MAU) end of the AUI cable, and a 15 pin female D shell connector is used at the DTE end.

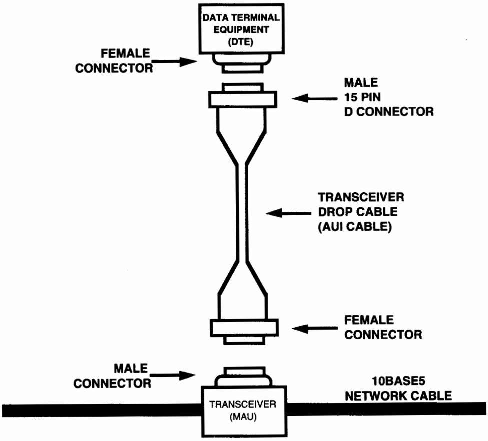


Figure 6.1–10BASE5 network connection

6.2.4 802.3 Type 10BASE2

Because of the high cost of the cable used for the 10BASE5 standard, the IEEE also developed the 802.3 10BASE2 standard (Figure 6.2). 10BASE2 (commonly called thin Ethernet, or Cheapernet) uses a thinner, lower performance cable (R/G 58 A/U) that is limited to a 185m segment length and a 925m network length. 10BASE2 does not require MAUs or transceiver cables. The nodes are attached with simple BNC T-connectors, but only 30 nodes per segment are allowed.

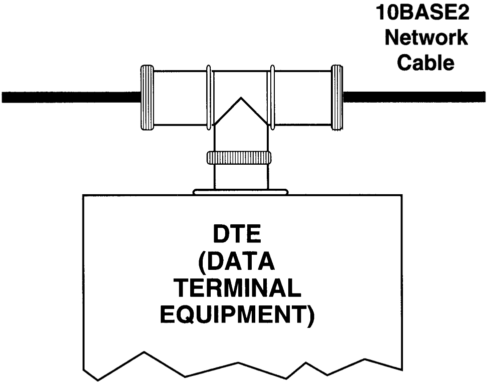


Figure 6.2–10 BASE2 network connection

6.2.5 802.3 Type 10BASET

Recently, the Type 10BASET standard was established. 10BASET uses twisted pair wiring instead of coaxial cable. Although twisted pair wiring has higher bit error rates and can support the 10 Mbit/sec data rate only over shorter distances, it employs existing twisted pair telephone wiring to provide ease of installation.

6.2.6 Data Encoding

Serial data on an 802.3 network is transmitted in a binary encoding scheme called Manchester encoding (Figure 6.3). Manchester encoding combines data and clock into "bit-symbols." A transition occurs in the middle of each bit symbol. Thus each bit-symbol is split into two halves, with the second half containing the binary inverse of the first. In the first half of the bit-symbol, the encoded signal is the logical complement of the value being encoded. Thus a logical 1 is encoded by a bit-symbol in which the first half is a logical 0 and the second half is a logical 1. A logical 0 is encoded by a bit symbol containing a logical one followed by a logical zero.

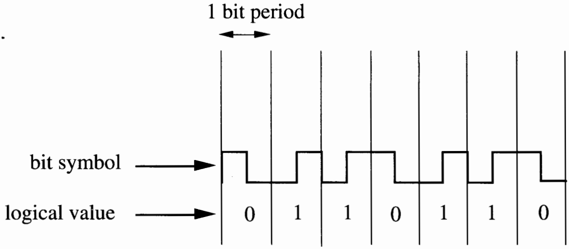


Figure 6.3–Manchester encoded bitstream

6.2.7 CSMA/CD

Ethernet uses a transmission protocol called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). The *Carrier Sense* (CS) portion of the protocol requires that before a station begins transmitting it first "listen" to the channel to make sure that the channel is idle, i.e., that no other station is currently transmitting. The *Multiple Access* (MA) portion of the protocol requires that every station on the network have equal access to the communications channel; there is no central controller allotting specific times to each station. The *Collision Detection* (CD) portion of the protocol requires that a transmitting station listen to the line during transmission to make sure that no other station has also begun transmitting; two or more stations transmitting at the same time results in a collision.

When a collision occurs, the stations stop transmitting and wait a random amount of time before retrying transmission.

6.2.8 Frame Format

The serial data sent across an 802.3 or Ethernet network is packaged into frames. Each frame has the format defined in Figure 6.4:

7Bytes	1B	6B	6B	2B	46-1500B	4B
Preamble	SFD	Dest Address	SRC Address	Length	Data	FCS

Figure 6.4–Ethernet frame format

Preamble–A 7 byte field of alternating 0s and 1s to allow network hardware to synchronize with the frame timing.

Start of Frame Delimiter (SFD)–The 1 byte pattern 10101011, which indicates the end of the preamble and the start of the frame.

Destination Address–A 6 byte field, which specifies the station for which the frame is intended.

Source Address–A 6 byte field, which specifies the station sending the frame.

Length Field (IEEE 802.3 definition)–A 2 byte field, which specifies the number of bytes contained in the data filed. (In Ethernet, this field is called the Type field and specifies a 2 byte type code.)

Data Field–A sequence of data bytes. If the number of data bytes transmitted is less than the minimum data length of 46 bytes, this field must be padded to provide the minimum frame length. The maximum data field length is 1500 bytes.

Frame Check Sequence (FCS)–A 4 byte field containing a cyclic redundancy check (CRC) value computed from the data contained in the Destination Address, Source Address, Length, and Data fields, used for error detection.

While the usual destination and source addresses are specified by the 802.3 standard to be 6 bytes, the standard allows for the use of 2 byte addresses, provided that 2 byte addresses are required by **every** station on the network. Consequently, the 2 byte address implementation is seldom found.

6.3 The OSI Model

6.3.1 Overview

In order to encourage network compatibility among different manufacturers, the International Standard Organization (ISO) developed the Open System Interconnect (OSI) reference model. This model consists of a recommended set of protocols for providing communications among multiple devices. The OSI model is comprised of seven functions, or layers (Figure 6.5). This layered model provides modularity and flexibility, since each layer can be modified without affecting the other layers.

7 Application	User applications
6 Presentation	Performs data reformatting and code conversion
5 Session	Manages interaction between application processes
4 Transport	Provides end-to-end data integrity
3 Network	Provides internetwork addressing and routing
2 Data Link	Performs formatting and data transmission
1 Physical	Defines physical media and bit characteristics

Figure 6.5–Open system interconnect (OSI) model

6.3.2 Commonly Used Protocols

6.3.2.1 Unix and TCP/IP

IEEE standard 802.2, Logical link control, defines a low level communications protocol. Taken together, the 802.2 and 802.3 standards define layers 1 and 2 of the OSI model. In order to provide effective and reliable communications, software must be run in the individual network nodes to provide the higher layers. In the Unix operating system environment, TCP/IP is the most commonly used network protocol. IP, or Internet Protocol, corresponds to layer 3, the network layer of the OSI model, and TCP, or Transport Control Protocol, corresponds to layer 4, the transport layer. The Unix operating system itself provides the higher layers (Figure 6.6).

7 Application	Applications Program	
6 Presentation	Unix	
5 Session	Unix	
4 Transport	TCP (Transport Control Protocol)	TCP/IP
3 Network	IP (Internet Protocol)	
2 Data Link	IEEE 802.2 & IEEE 802.3	
1 Physical	IEEE 802.2 & IEEE 802.3	

Figure 6.6–Unix implementation of OSI

6.3.2.2 Novell NetWare

On DOS-based personal computers, the Novell NetWare Operating System is a commonly used network communications environment. The IPX, or Internet Packet eXchange, corresponds to layer 3, the Network layer of the OSI model; the SPX, or Sequenced Program eXchange, corresponds to layer 4, the transport layer (Figure 6.7). The Novell Operating System provides the upper layers of the OSI model.

7 Application	Applications Program
6 Presentation	NetWare
5 Session	NetWare
4 Transport	SPX (Sequenced Program eXchange)
3 Network	IPX (Internet Packet eXchange)
2 Data Link	IEEE 802.2 & IEEE 802.3
1 Physical	IEEE 802.2 & IEEE 802.3

Figure 6.7–Novell NetWare implementation of OSI

6.4 Ethernet and Data Acquisition

Ethernet's importance to data acquisition and communication has been underlined by the introduction of IOtech's LAN488 Ethernet to IEEE 488 controller. Instruments attached to a LAN488 can be controlled by distant computers via the Ethernet connection, enabling the LAN488 to effectively extend the IEEE 488 bus. However, unlike most IEEE 488 extenders, LAN488 can be configured in a multidrop IEEE 488 system with several remote IEEE 488 buses accessible from one or more computers (Figure 6.8). Thus the number of instruments that can be controlled is essentially limitless; the distance between LAN488 nodes can be the maximum permitted by the specific Ethernet cabling.

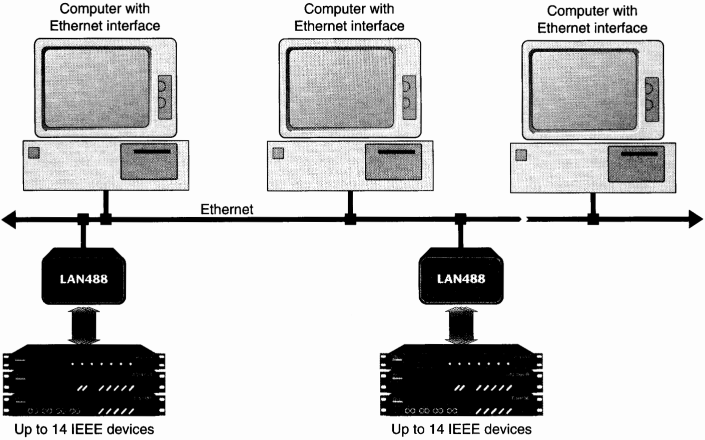
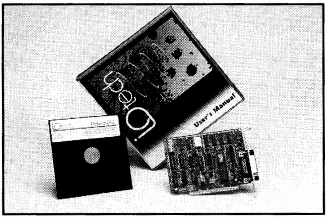

Computer with Ethernet interface
Computer with Ethernet interface
Computer with Ethernet interface
Ethernet
LAN488
LAN488
Up to 14 IEEE devices
Up to 14 IEEE devices

Figure 6.8–LAN488 makes distributed data acquisition and control possible

IOtech Product Selection Guide

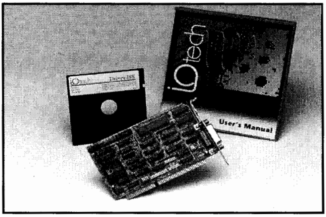
The following is a guide to some of the products mentioned in the preceding text. A complete description of these and other products can be found in the current issue of the *IOtech Product Catalog*, available free of charge by calling IOtech at 216/439-4091 (fax 216/439-4093).

IBM PC IEEE 488 Controllers



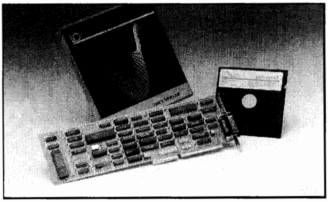
Personal488plus

- Adds IEEE 488.2 capability to IBM PC/ATs
- Includes the GP488Bplus interface board and enhanced Driver488 DOS device driver software
- Easy to program HP-style commands
- Optional RTLib488 real time graphics and analysis subroutine libraries for C and Pascal
- New Personal488/OEM-P development package and Library488 for OEMs available



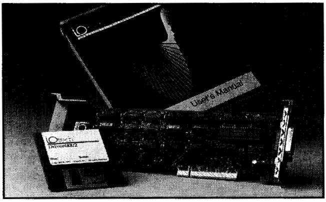
Personal488/AT

- Adds high speed 488.2 capability to IBM PC/ATs
- 1 Mbyte/sec maximum DMA data transfers
- Includes the half-slot AT488 interface board and enhanced Driver488 DOS driver software
- Easy to program HP-style commands
- Subroutine calls for high speed applications
- Optional RTLib488 real time graphics and analysis subroutine libraries for C and Pascal



Power488

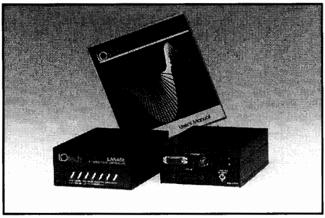
- Adds high speed IEEE 488.2 control, digital I/O, and counter-timer functions to IBM PC/ATs
- 1 Mbyte/sec maximum DMA data transfers
- Includes new Driver488 software with call based subroutines, SCPI commands for I/O functions, and software support for the PC's COM ports
- Optional RTLib488 real time graphics and analysis subroutine libraries for C and Pascal



Personal488/2plus

- Adds IEEE 488.2 capability to the IBM PS/2 MicroChannel family of personal computers
- Includes the GP488/2plus interface board and enhanced Driver488 DOS device driver software
- Easy to program HP-style commands
- Subroutine calls for high performance
- Optional RTLib488 real time graphics and analysis subroutines for C and Pascal

IBM PC IEEE 488 Controllers

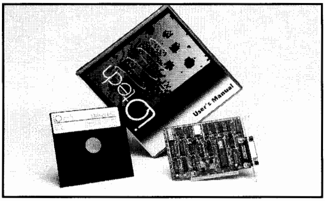


LAN488

- Control up to 14 IEEE 488 devices with each LAN488 on a Ethernet Novell NetWare network
- Provides centralized control of distributed IEEE 488 test systems
- Allows any PC on the network to access IEEE 488 devices
- Allows control of remote IEEE 488 devices as if they were attached directly to the local PC

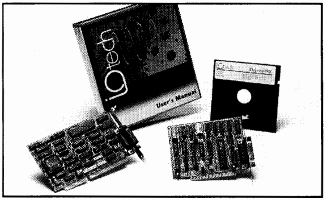
Personal488/G

- Adds high speed 488.2 capability to the GRiDCASE 1500 series of laptop computers
- Includes the GP488B/G expansion cartridge and Driver488 DOS driver software
- Easy to program HP-style commands
- Supports all popular languages, spreadsheets, and data analysis software



Personal488/UX

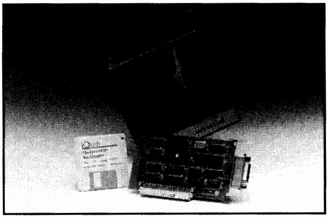
- Controls IEEE devices under the powerful XENIX environment
- Allows multiple tasks to access the IEEE bus simultaneously
- Includes GP488C IEEE board and Driver488/UX device driver software



Personal488/OEM-P

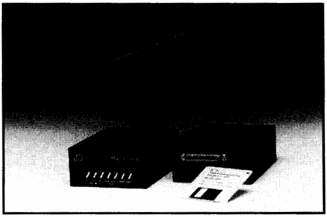
- Develop IEEE 488.2 peripherals based on an embedded PC architecture
- Includes a compact library of Microsoft C subroutines that provide IEEE 488.2 peripheral functions
- Requires less than 12 Kbytes of memory
- DMA and interrupt-driven operation supports
- Optional RS-232 COM port support available
- Includes an 8-bit or 16-bit IEEE 488.2 board

Macintosh IEEE 488 Controllers



MacII488

- Plugs directly into a NuBus© slot in the Macintosh II, IIx, IIcx, or IIfx
- Supports data transfer rates of over 600 Kbytes/sec
- Includes MacDriver488 and MacDA488 software
- Programmed with high-level HP-style commands

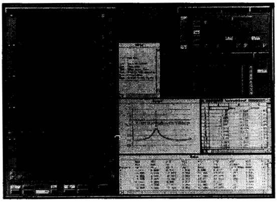


MacSCSI488

- Attaches to the SCSI port on the Macintosh computer
- Supports data transfers of over 600 Kbytes/sec on the Macintosh SE, and over 800 Kbytes/sec on the Macintosh II
- Includes MacDriver488 andMacDA488 software

Mac488B

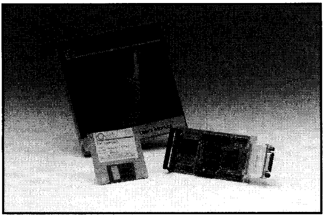
- Control up to 14 IEEE devices as far as 4,000 ft from the Macintosh using its modem or printer port
- Built-in A-B-C switch allows other serial devices to share the same serial port
- Built-in 32,000 character data buffer increases efficiency
- Includes MacDriver488 and MacDA488 software



SuperScope

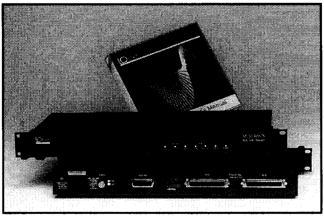
- Convert your Macintosh into a virtual lab instrument capable of collecting, analyzing, and storing data
- Compatible with IOtech Macintosh IEEE interfaces for controlling and collecting data from IEEE instruments
- Collect data directly from an ADC488 digitizer at 100 kHz using on-screen controls.
- Easily define instruments without programming

Workstation IEEE 488 Controllers



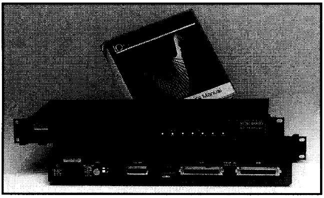
SB488

- Control IEEE488 instruments from an SBus slot in the Sun SPARCstation or compatibles
- 100 percent compatibility with IEEE 488.2 specified controller functions
- High-speed data transfers up to 1 Mbyte/sec
- Includes Driver488/S UNIX device driver and language interface with HP-style IEEE commands



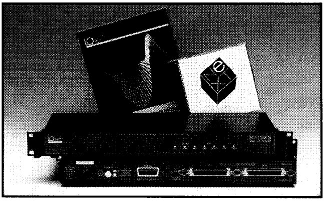
SCSI488/S

- Control IEEE 488 instruments from SCSI port on Sun-3, Sun-4, and SPARCstation workstations
- High speed data transfer up to 1 Mbyte/sec
- Enables daisy-chaining with other SCSI devices via two rear panel SCSI connectors
- Includes Driver488/S UNIX device driver and language interface HP-style IEEE commands



SCSI488/D

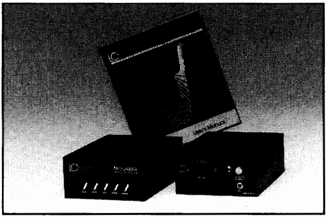
- Control IEEE 488 instruments from the external SCSI port on the DEC VAXstation 3100, MicroVAX 3100, DECstation 5000, and DECstation 3100
- Includes Driver488/D command language interpreter, providing high-level HP-style IEEE 488 commands for VMS and DEC IEZ-11 device driver
- High speed data transfers up to 1 Mbyte/sec



SCSI488/N

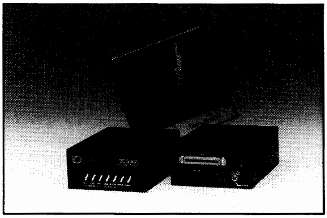
- Control up to 14 IEEE 488 instruments via the external SCSI port on members of the NeXT workstation family
- High speed data transfers up to 1 Mbyte/sec
- Includes Driver488/N software interface that can be called from the NeXT Object-C environment
- Uses familiar HP-style IEEE commands

Platform-Independent Controllers



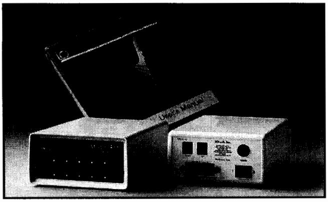
Micro488A

- Enables control of up to 14 IEEE devices from any computer with an RS-232 or RS-422 port
- Programmed with HP-style commands
- Built-in 32,000 character data buffer
- Supports baud rates up to 57,600 baud
- Micro488/EX operates as a stand-alone controller using a 32K transaction non-volatile memory for program and data



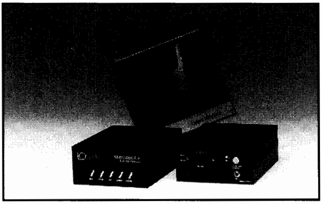
SCSI488

- Provides high-speed bidirectional communication between SCSI and IEEE devices
- Controls IEEE instruments from a computer's SCSI port
- Controls SCSI devices from an IEEE port
- Functions as a full-featured IEEE controller



Modem488

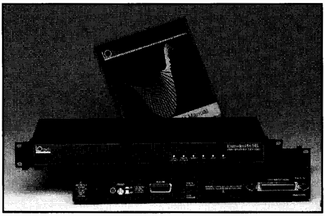
- 110, 300, or 1200 baud operation, with auto-answer, auto-dial, and redial
- Non-volatile auto-dial number storage
- HP-style commands in the controller mode
- Security mode to prevent unauthorized control of the bus



Micro488/EX

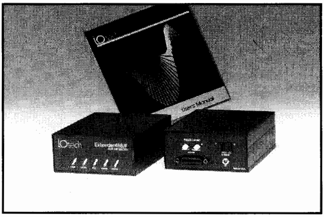
- Operates as a stand-alone controller using a 32 Kbyte non-volatile memory for program and data
- Stores up to 100 different macros in non-volatile memory
- Permits data retrieval by any computer serial port
- Features a built-in real-time clock; permits collection and time-stamping of data at precise times or at regular or irregular intervals
- Programmed with HP-style commands

IEEE 488 Bus Enhancers



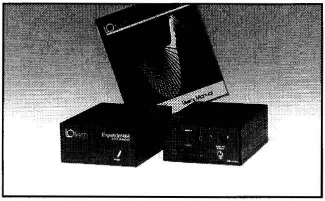
Extender488/HS

- Extends the length of the IEEE bus while maintaining data transfer rates as high as 840 bytes/sec
- Allows the IEEE bus to be transparently extended beyond the 20m cable limitation to as far as 1000m
- Expands the number of devices allowed on the bus from 15 to 28



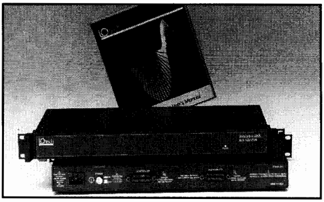
Extender488

- IEEE economically extends the bus up to 1000m, via RS-422 or fiber-optic link, to exceed the IEEE standard 20m cable limitation.
- Fiber-optic extenders electrically isolate instruments
- Expands the number of devices allowed on the bus from 15 to 28



Expander488

- Doubles the maximum allowable number of devices on the bus
- Operates transparently to the IEEE controller
- No special software or bank-switching is necessary to access an IEEE device

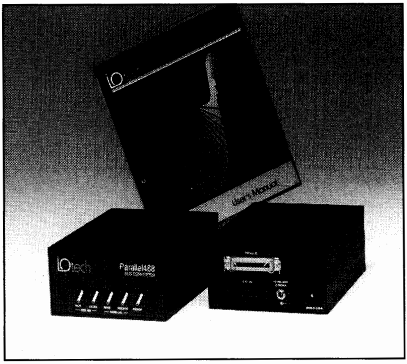


Isolator488

- Electrically isolates individual instruments or groups of instruments
- Expands the number of devices allowed on the bus from 15 to 28
- Allows the IEEE interface of instruments on the same IEEE bus to operate at different voltage reference levels
- Operates transparently to the system, requires no programming

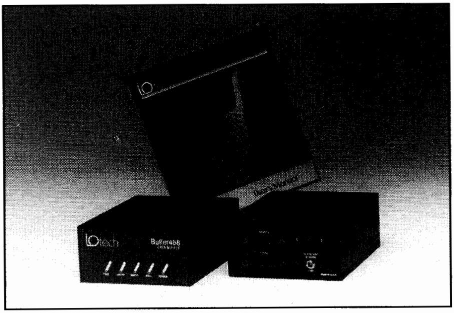
IEEE 488 Bus Enhancers

Parallel488



- Transparent link between IEEE and parallel devices
- Front panel indicators for IEEE Talk and Listen, and parallel Send and Receive
- Built-in 24,000 character data buffer
- Switch selectable IEEE address

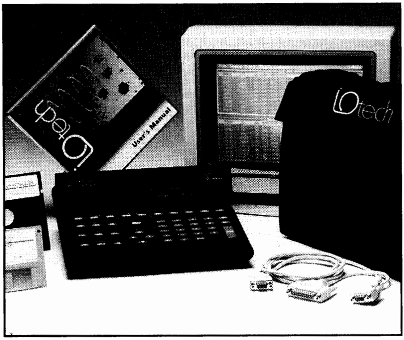
Buffer488



- Increase the efficiency of your IEEE system by buffering data to slow IEEE plotters or printers
- Provides up to 512 Kbytes of data buffer
- Operates transparently to the system
- Allows entire plotter documents to be buffered

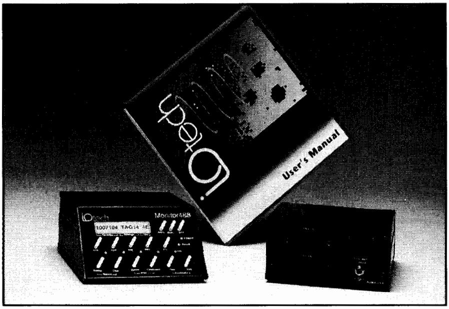
IEEE 488 Bus Analyzer and Monitor

Analyzer488



- Permits the monitor, capture, and analysis of IEEE bus transactions at the full 1Mbyte/sec IEEE data transfer speed
- Allows anlysis of bus transactions from the unit's front panel, or from a computer connected to the unit's serial port
- Includes Analyst488 PC control software
- Permits set up of post-triggering for capturing desired transaction patterns
- Provides search features for scanning the 32K transaction buffer
- Controls up to 14 instruments in the bus controller mode
- Measures efficiency of the bus operations with its speed measurement feature
- Provides external "trigger-satisfied" signal for triggering test equipment

Monitor488

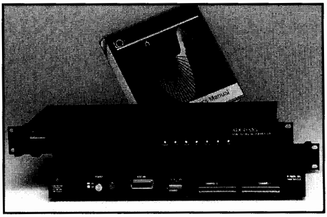


- Decodes and displays IEEE bus transactions in real time
- Capable of slowing the bus for real time viewing
- Allows single stepping for debugging system problems
- Records and prints decoded bus transactions in two formats to a Centronics printer
- Can be configured to continuously capture IEEE bus transactions into its circular capture buffer for recall and later printing

IEEE 488 Data Acquisition Instruments

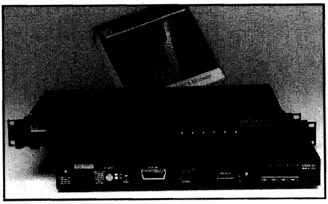
ADC488/16

- 16-bit analog to digital conversion at up to 100,000 samples/sec
- 16 single-ended or 8 differential analog inputs
- Continuous throughput to the IEEE bus at 200 Kbytes/sec (100,000 16-bit readings/sec)
- Memory expansion to 8 Mbytes
- ±1, ±2, ±5, and ±10 VFS programmable input ranges



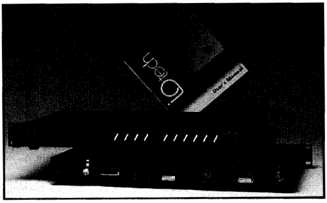
ADC488/8S

- 16-bit analog to digital conversion up to 100,000 samples/sec
- 8 differential analog inputs/simultaneously sampled
- Continuous throughput to the IEEE bus at 200 Kbytes/sec (100,000 16-bit readings/sec)
- Memory expansion up to 8 Mbytes
- ±1, ±2, ±5, and ±10 VFS programmable input ranges



DAC488

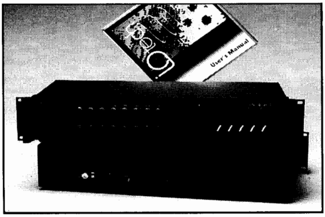
- Two or four channel 12-bit plus sign isolated digital to analog converters
- Built-in 8192-point waveform buffer with 1 kHz update rate
- Sequenced output based on either a periodic interval or trigger condition
- Three trigger conditions: External, TRG command, and GET command



DAC488HR

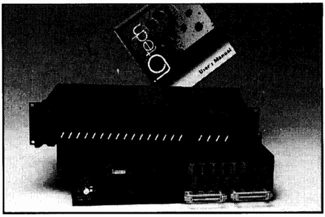
- Two or four channel isolated 16-bit outputs
- 100 kHz/channel maximum update rate
- 480K sample data buffer per channel
- One-shot, step, burst, waveform, and continuous output modes
- GET, external TTl, IEEE command, and time event trigger sources
- Standard sine, square, triangle, and sawtooth waveform generation

IEEE 488 Data Acquisition Instruments



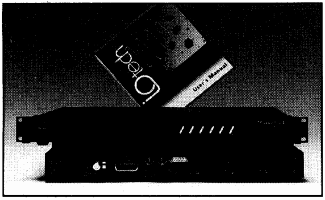
Filter488

- Programmable via IEEE 488 and RS-232
- Several filter response options: Butterworth, Bessel, Elliptic, and Chebychev
- 4 or 8 channels
- 50 kHz maximum cut off frequency specified with 3 digit resolution
- 3dB cutoff frequency accuracy within 1 percent
- <0.15 percent total harmonic distortion



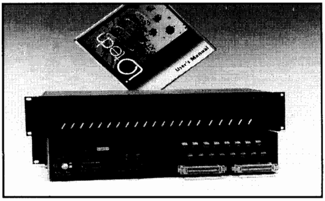
Mux488/16SC

- Provides signal conditioning and multiplexing for 16 analog channels
- Two versions available: Mux488/16SC is controlled via IEEE 488 or RS-232; Mux/16SC is controlled via an 8-bit digital word from a digital I/O port or a PC parallel port, or slaved to a Mux488/64



Mux488/64

- Multiplex 64 single ended or 32 differential analog channels
- Two versions available: Mux488/64 is controlled via IEEE488 or RS-232; Mux/64 is controlled via an 8-bit digital word from a digital I/O port or a PC parallel port, or slaved to a Mux488/64



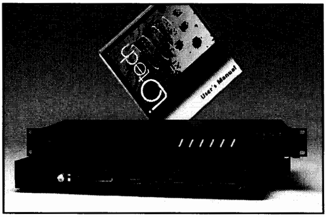
Control488/16

- Control AC and DC power to devices using opto-isolated solid-state relays
- Sense high level AC or DC voltages using opto-isolated input modules
- Two versions available: Control488/16 is programmed via an IEEE 488 interface or RS-232 port; Control/16 is programmed via an 8-bit digital word from a digital I/O port or a PC parallel port, or slaved to a Control488/16

IEEE 488 Data Acquisition Instruments

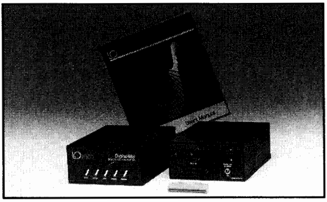
Digital488HS/32

- Input or output digital words at rates up to 1 Mbyte/sec; input or output 16-bit digital words at up to 500 Kbytes/sec
- Complete set of I/O handshake, control, and status lines
- TTL and HCT compatible I/O lines



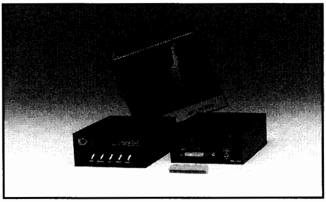
Digital488/80A

- Enables an IEEE488 controller to program or read the state of 80 bits of TTL level signals
- Optional 200mA of high current drive capability on each line when used as outputs
- Optional support for +12,+24, and +48 volt logic levels



Digital488

- Adds TTL level digital input and output capability to IEEE 488 systems
- Forty I/O lines are software programmable as inputs or outputs in groups of 8 bits
- Converts I/O data to one of five selectable formats and sends it to the controller
- Provides five additional I/O lines, including trigger output, service request input, data latch input, clear output, and inhibit output



Digital232

- Provides TTL level digital input and output capability to RS-232 based systems
- Programmable for input or output of up to 40 bits of digital data
- Programmable in groups of 8 bits for input or output
- Supports complete digital I/O handshaking

Bibliography

The following references may provide additional information on the topics addressed in this handbook:

[1] ANSI Std X3.131-1986, *Small Computer System Interface*

[2] ANSI/IAAA Std 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation.*

[3] ANSI/IEEE Std 488.2-1987, *IEEE Standard Codes, Formats, Protocols, and Common Commands.*

[4] Caristi, Anthony J. *IEEE-488 General Purpose Instrumentation Bus Manua*l, Academic Press, San Diego, 1989.

[5] EIA Std EIA-232-C, *Interface Between Data Terminal Equipment Employing Serial Binary Data Interchange*

[6] *Tutorial Description of the Hewlett-Packard Interface Bus*, Hewlett-Packard Publication 5021-1927, 1987.

[7] Mueller, Joseph E. "Efficient Instrument Design Using IEEE 488.2", IEEE Transactions on Instrumentation and Measurement, Vol. IM-39, No. 1, February 1990, pp 146-150.

SCPI Consortium
8380 Hercules Drive, Suite P3
La Mensa, CA 92042
(619) 697-8790

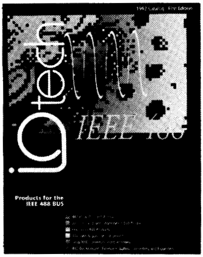
The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017

IEEE Standards Publication: (201) 562-3800

Index

IOtech

CATALOG OFFER



To obtain a **free** copy of IOtech's latest catalog, please complete and mail this card; for immediate action, fax it to (216) 439-4093 or call (216) 439-4091. ➤

IOtech

INSTRUMENT COMMUNICATION HANDBOOK OFFER

To obtain a **free** copy of IOtech's *Instrument Communication Handbook* for a friend or co-worker, have him or her fill out the attached card and fax it to a local IOtech representative. ➤

Please **completely** fill out this card to receive IOtech's latest catalog □, or have a friend or co-worker fill it out to obtain a **free** copy of the *Instrument Communication Handbook* □.

TYPE OF FACILITY:

□ Education/university
□ Research
□ Test/measurement equipment manufacturing
□ VAR/system integrator
□ Component manufacturing
□ Aerospace/defense
□ Utility
□ Computer/peripheral manufacturing
□ Communications
□ Industrial controls manufacturing
□ Chemical/petroleum processing
□ Hospitals
□ Medical equipment/services
□ Automotive/parts manufacturing
□ Consumer electronics manufacturing
□ Other: ______________

□ Authorized for GSA contract

DEPARTMENT:

□ Research
□ Design/development
□ Calibration lab/instrument pool
□ Production/test
□ Component test
□ Field service, installation
□ Quality assurance/control
□ Purchasing
□ Education (chemistry, engineering, etc.)
□ Other: ______________

PRODUCT INTERESTS:

A) IEEE 488 Products

□ IEEE 488 instrument control
□ Serial/IEEE 488 converters/controllers
□ Data Acquisition Instruments
□ Signal conditioning
□ Graphic and analysis software
□ UNIX IEEE 488 support
□ IEEE 488 extenders, expanders, and buffers
□ IEEE 488 bus analyzer
□ IEEE 488 plotter/printer interfacing
□ Other: ______________

B) Other Products:

□ DMMs
□ Oscilloscopes
□ RF equipment
□ Switches
□ VXI equipment
□ Waveform generators
□ Data acquisition plug-in boards
□ Power supplies
□ Temperature instruments
□ Other: ______________

MY NEED IS:

□ Immediate
□ Future reference
□ 6 months

JOB FUNCTION:

□ Manager
□ Engineer
□ Scientist
□ Technician
□ Professor/instructor
□ Student
□ Graduate student
□ Purchaser
□ Other: ______________

MY COMPUTER IS:

□ IBM PC or compatible
□ IBM PS/2 or compatible
□ Macintosh
□ Sun
□ DEC
□ NeXT
□ HP
□ Other ______________

PRODUCTS IN USE:

We now use IOtech model(s):

____

____

____

____

IN MY WORK, I:

□ Design
□ Purchase
□ Specify
□ Use

MY APPLICATION IS:

____

____

____

____

Please complete this information or tape your business card to this side

Name ______________

Company______________

Address______________

____

City/State/ZIP______________

Telephone (________)______________

Fax (_______)______________

Country______________

fold here↘

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT #4417 CLEVELAND, OH

POSTAGE WILL BE PAID BY ADDRESSEE

IOtech
PO BOX 391345
CLEVELAND OH 44139-9846

fold here↗

$14.95

PR-13

IOtech, Inc. • 25971 Cannon Road • Cleveland, Ohio 44146
Tel (216) 439-4091 • Fax (216) 439-4093