Tutorial Description of the Hewlett-Packard Interface Bus

HEWLETT PACKARD

Table of Contents

List of Tables

List of Illustrations

Chapter 1
Introduction

1.1 Foreground

This book is a tutorial description of the technical fundamentals of the Hewlett-Packard Interface Bus (HP-IB). HP-IB is Hewlett-Packard's implementation of the IEEE 488.1 Bus. The IEEE 488.1 Bus is identical to the original 488 bus.

This book also includes information on the IEEE 488.2 Standard used with HP-IB (IEEE 488). IEEE 488.2 is a set of standard codes, formats, protocols, and commands used with 488.1.

This book is intended to provide a thorough overview of HP-IB basics for the first-time HP-IB system designer, programmer, or user. It should be useful to instrument, computer, and system oriented engineers or technicians for either self-study, technical reference, or as an index for further research. In short, it's a broad tutorial for learning about the Hewlett-Packard Interface Bus. Let's begin with a look at what HP-IB is and where it came from.

1.2 Background

The Hewlett-Packard Interface Bus (HP-IB) is a carefully designed and defined general purpose digital interface system and associated support which simplifies the design and integration of instruments and computers into systems. It minimizes electrical/mechanical hardware and functional compatibility problems between devices, yet has sufficient flexibility to accommodate a wide and growing range of products. As such, HP-IB is an interfacing concept, and a design technique. You can take advantage of these concepts to define, design, build, and use your own measurement system for maximum cost-effectiveness. It's more than an interface. It's a design philosophy.

HP-IB applies to the interface of instrumentation systems in which:

1. Data exchanged among the interconnected apparatus is digital (as distinct from analog).

2. Fifteen devices may be interconnected to one continuous bus.

3. Total transmission path lengths over the interconnecting cables does not exceed 20 meters or 2 meters per device, whichever is less (when not using a bus extension technique).

4. Data rate across the interface on any signal line does not exceed 1 Mbyte/second.

HP-IB evolved from an internal Hewlett-Packard need for a standardized instrumentation interface system. The chronology of the HP-IB evolution is summarized here:

- Sept. '65 - HP began to look at how to standardize "the interfacing of all HP future instruments."
- March '72 - U.S. Advisory Committee (IEC) formed. The committee takes HP proposal as starting point.
- Sept. '74 - IEC approves for ballot draft document (U.S. Proposal).
- April '75 - IEEE Publishes IEEE 488.
- Jan. '76 - ANSI Publishes MC1.1.
- Nov. '78 - IEEE Revises IEEE 488.
- June '80 - IEC 625-1 published.
- Dec. '81 - IEEE 728 published. (Recommended Codes & Formats)
- June '87 - IEEE revises 488 to become 488.1
- June '87 - IEEE 488.2 published. (Codes, Formats, Protocols, Commands)

Initial HP design efforts, beginning as early as 1965, formed the framework which was later taken by the newly-formed International Electrotechnical Commission (IEC) Technical Committee 66, Working Group 3 as a starting proposal. By September, 1974, a draft document of the HP proposal was approved for balloting by the IEC. In April, 1975, the Institute of Electrical and Electronics Engineers (IEEE) published their document IEEE 488-1975, *Digital Interface for Programmable Instrumentation*, which contained the Electrical, Mechanical and Functional specifications of an American Standard interfacing system. The identical MC1.1 was published by the American National Standards Institute (ANSI) in January 1976. A revision of the IEEE 488 occurred in Nov., 1978, primarily for editorial clarification and addendum. In June, 1980, the IEC published its version IEC 625-1, *An Interface System for Programmable Measuring Apparatus (Byte Serial Bit Parallel).*

Even though guidelines for preferred syntax and format conventions were not part of the original IEEE 488 document, work continued in this area in order to increase the usability of different vendors' equipment. This work resulted in IEEE 728-1982, *Recommended Practice for Code and Format Conventions for IEEE Standard 488.* This document contains a series of recommendations (guidelines) rather than being a standard (in absolute terms). Also, an IEC document, IEC 625-2, dealt with the same subject matter but its content is not identical to IEEE 728.

After nearly 10 years of experience designing and using IEEE 488 and 728, many users realized the need to expand the definition of the standard. Many companies, including Hewlett-Packard, had defined internal standards for data, protocols, and device commands.

Building on the experience of these companies, the IEEE organized a committee to draft a supplemental standard that became the IEEE 488.2 *Codes, Formats, Protocols and Common Commands For use with IEEE 488.1-1987*. IEEE 488 was renamed to IEEE 488.1. IEEE 488.2 does not replace 488.1. Devices can still conform only to 488.1. IEEE 488.2 builds upon 488.1, defining a common set of data codes and formats, a bus communication protocol, and a set of commonly needed commands. This new standard replaces the IEEE 728 Recommended Codes. Several chapters of this book describe this new standard.

STATUS TODAY

There are five major standards defining byte serial bit parallel interface systems for instrumented systems.

1. IEEE 488.1-1987 (Orginal 488 Standard)
2. ANSI MC1.1 (Identical)
3. IEC 625-1 (Identical except for connector)
4. B.S. 6146 (British Standard Identical to IEC 625-1)
5. IEEE 488.2-1987 (Codes, Formats, Protocols, Common Commands)

The IEEE 488 is most widely used internationally and is implemented in several brand versions:

- Hewlett-Packard Interface Bus (HP-IB)
- General Purpose Interface Bus (GPIB)
- IEEE BUS
- ASCII BUS
- PLUS BUS

The IEEE 488 standard has been published in 9 languages and has been used by more than 250 manufactures in more than 14 countries to design thousands of products. It is one of the most carefully defined, consistent, and highly used interface systems in the world. Let's pause to take a look at what makes it such a useful interface.

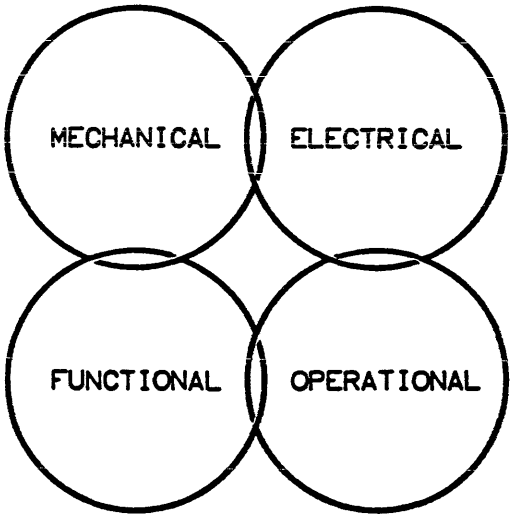


Figure 1.1 Interface System

1.3 What comprises an interface system?

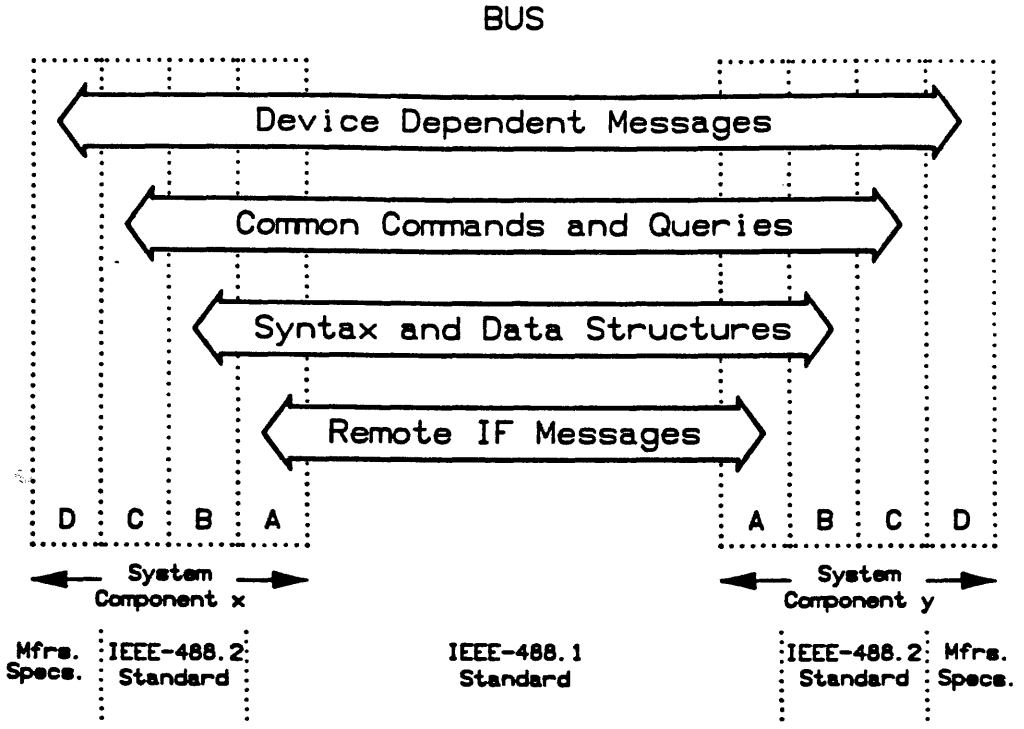
An interface system can be totally characterized in terms of the Functional, Electrical, Mechanical, and Operational specifications of the interface.

- FUNCTIONAL - Total set of allowable interface functions and their logic descriptions (Application independent)
- ELECTRICAL - Logic levels, protocol, timing, termination, etc. (Application independent)
- MECHANICAL - Connector, Mounting, Cable assembly, etc. (Application independent)
- OPERATIONAL - Total set of allowable device functions and their logic descriptions (Application dependent)

The IEEE 488, ANSI MC1.1, and IEC 625-1 standards address three of these areas but not the Operational area. This gives instrument and computer designers the flexibility to optimize their products to the intended applications.

However, the IEEE developed IEEE Std. 488.2-1987, which provides a set of codes, formats, protocols and common commands for use with 488.1 to provide a foundation for the operational area.

The operational aspect of the interface can be depicted as various levels or layers of operation. Figure 1.2 shows these levels graphically. Chapter 3 begins the description of IEEE 488.2. It further describes the operational aspects of the interface.



WHERE:

Layer D represents Device Functions
Layer C represents Common System Functions
Layer B represents Message Communication Functions
Layer A represents Interface Functions (IF)

Figure 1.2 Functional Layers Diagram

Comparing the standards

The IEEE 483 and ANSI MC1.1 are identical in Electrical, Mechanical and Functional characteristic areas.

The IEC 625-1 differs from the others in the mechanical area. This standard specifies a 25 pin D type connector rather than the 24-pin Ribbon type specified by the American Standards (Pin 25 is an extra signal return line). Unfortunately, the 25 pin connector is used extensively as part of the Electronic Industry Association (EIA) Recommended Standard RS-232-C *Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange* for data communications. Signal lines utilizing this serial scheme employ voltage levels up to ± 25V with 0.5 ampere short-circuit current capabilities. Connecting an RS-232-C circuit to an IEC 625-1 instrument can cause circuit damage.

CAUTION

Component damage, due to incompatible voltage levels, is possible if data communication and instrumentation interfaces are inadvertently interconnected (IEC 625-1 compatible device to an RS-232-C compatible interface). Any mechanical specification difference between the IEEE 488/ANSI MC1.1 and IEC 625-1 standards may be accommodated by a simple (physical) adaptor assembly when products implemented from the two standards are interconnected.

In Europe about 90% of the bus-compatible products presently prefer or offer the IEEE 488/ANSI MC1.1 connector. Many of the manufacturers offer the option of either type and simple adaptors are available.

Further information on these standards can be obtained from sources described in the Bibliography in Appendix E.

Chapter 2
The IEEE 488.1 Bus

The ANSI/IEEE 488.1 *Standard Digital Interface for Programmable Instrumentation* provides an electrical and mechanical system for interconnecting electronic measurement devices. Hewlett-Packard calls it's implementation of this standard the Hewlett-Packard Interface Bus (HP-IB). HP-IB is totally consistent with the electrical, mechanical and functional specifications of the ANSI/IEEE 488.1 standard. Let's take a look at the IEEE 488.1 standard.

2.1 Key Specifications of IEEE 488.1

The key specifications of ANSI/IEEE 488.1 are summarized here:

- INTERCONNECTED DEVICES - Up to 15 maximum on one contiguous bus.
- INTERCONNECTION PATH - Star or linear (or both) bus network up to 20 meters total transmission path length.
- SIGNAL LINES - Sixteen active lines; 8 data lines and 8 interface and communication management lines.
- MESSAGE TRANSFER SCHEME - Byte-serial, bit-parallel, asynchronous data transfer using interlocking three-wire handshake technique.
- MAXIMUM DATA RATE - One megabyte per second over limited distances; 250 to 500 kilobytes per second typical maximum over a full transmission path. The actual data rate is determined by the devices in communication at the time.
- ADDRESS CAPABILITY - Primary addresses, 31 Talk and 31 Listen; secondary (2-byte) addresses, 961 Talk and 961 Listen. There can be a maximum of 1 Talker and up to 14 Listeners at a time on a single bus.
- PASS CONTROL - In systems with more than one controller, only one can be active at a time. The currently active controller can pass control to one of the others. A non-active controller may request control. Only the controller designated as system controller can demand control.
- INTERFACE CIRCUITS - Driver and Receiver circuits are TTL and Schottky compatible.

2.2 Interface Functions

The operation of the bus can be compared to the operation of a commitee. A committee chairman controls who talks and who listens. In the same way, IEEE 488.1 has one device that *controls*, deciding who talks and who listens. Every IEEE 488.1 device must be capable of performing one or more of the following interface functions (roles):

- LISTENER - A device capable of receiving data over the interface when addressed. Examples of this type of devices are: printers, display devices, programmable power supplies, programmable signal sources and the like. There can be up to 14 active listeners simultaneously on the interface.

- TALKER - A device capable of transmitting data over the interface when addressed. Examples of this type of devices are: tape readers, voltmeters that are outputting data, counters that are outputting data, and so on. There can be only one active talker on the interface at a time.

- CONTROLLER - A device capable of specifying the talker and listeners for an information transfer (including itself). A computer with an appropriate HP-IB card is an example of this type of device. There can be only one active controller on the interface at a time. In multiple controller systems only one can be the System Controller (Master).

2.3 Interface Capabilities

Interface functions are pre-defined capabilities which can be designed into an IEEE 488.1 device. The designer is free to choose which are implemented in a device depending on the particular device's intended application. The Interface Capabilities are summarized in Table 2.1. One or two letter codes, followed by a number indicate the capability that is implemented from the available subsets.

Capability I.D. (Not Mandatory)

The ANSI/IEEE 488.1 recommends that each device be marked near its connector with the interface capability codes for the functions it supports.

For example, a device with the basic talker function, the ability to send status bytes, the basic listener function, a listen only mode switch, service request capability, full remote local capability without local lockout, local configuration of the parallel poll capability, complete device clear capability, no capability for device trigger, and no controller capability would be identified with these codes:

SH1 AH1 T6 L3 SR1 RL2 PP2 DC1 DT0 C0 E1

E1 identifies open collector drivers and E2 would identify tristate drivers in the data mode. Appendix C gives an in-depth description of these capability codes.

Table 2.1 IEEE 488.1 Interface Capabilities

IEEE 488.1 Function	Code	Comments
Talker or Extended Talker	T,TE	Capability required for a device to be a "talker."
Listener or Extended Listener	L,LE	Capability required for a device to be a "listener."
Source Handshake	SH	This provides a device with the capability to properly transfer a multiline message.
Acceptor Handshake	AH	This provides a device with the capability to guarantee proper reception of multiline messages.
Remote/Local	RL	Provides capability to select between two sources of input information. Local corresponds to front panel controls and remote to the input information from the bus.
Service Request	SR	This capability permits a device to asynchronously request service from the controller.
Parallel Poll	PP	Provides capability for a device to uniquely identify itself if it requires service when the controller is requesting a response. This capability differs from service request in that it requires a commitment of the controller to conduct a parallel poll.
Device Clear	DC	This function allows a device to be initialized to a cleared state. The effect of this command is device dependent.
Device Trigger	DT	This function permits a device to have its operation initiated over the Bus. The result of this triggering is device dependent.
Controller	C	This function permits a device to send addresses, universal commands, and addressed commands to other devices on the bus. It may also include the ability to conduct polling to determine devices requiring service.
Drivers	E	This code describes the type of electrical drivers used in a device.

2.4 IEEE 488.1 Bus Lines

The IEEE 488.1 interface system utilizes a party-line bus structure (devices share signal lines) to which a maximum of fifteen devices may be connected in one contiguous bus. Sixteen signal lines and eight ground lines are used to interconnect devices in a parallel arrangement and maintain an orderly flow of device and interface related information.

The IEEE 488.1 interface bus signal lines all use a low-true logic convention with positive polarity. They can be grouped into three sets:

- Data Lines
- Byte Transfer Lines
- General Bus Management Lines

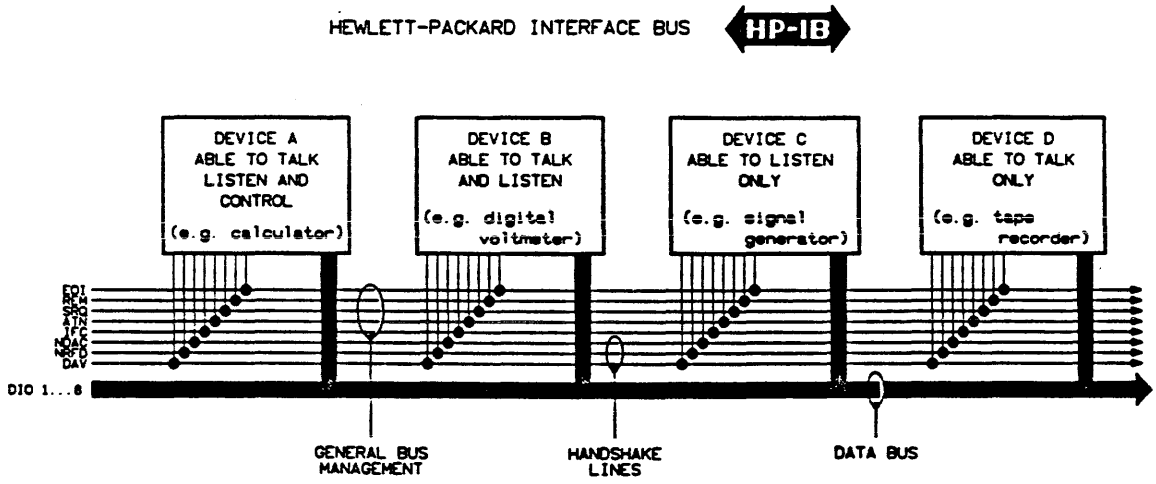
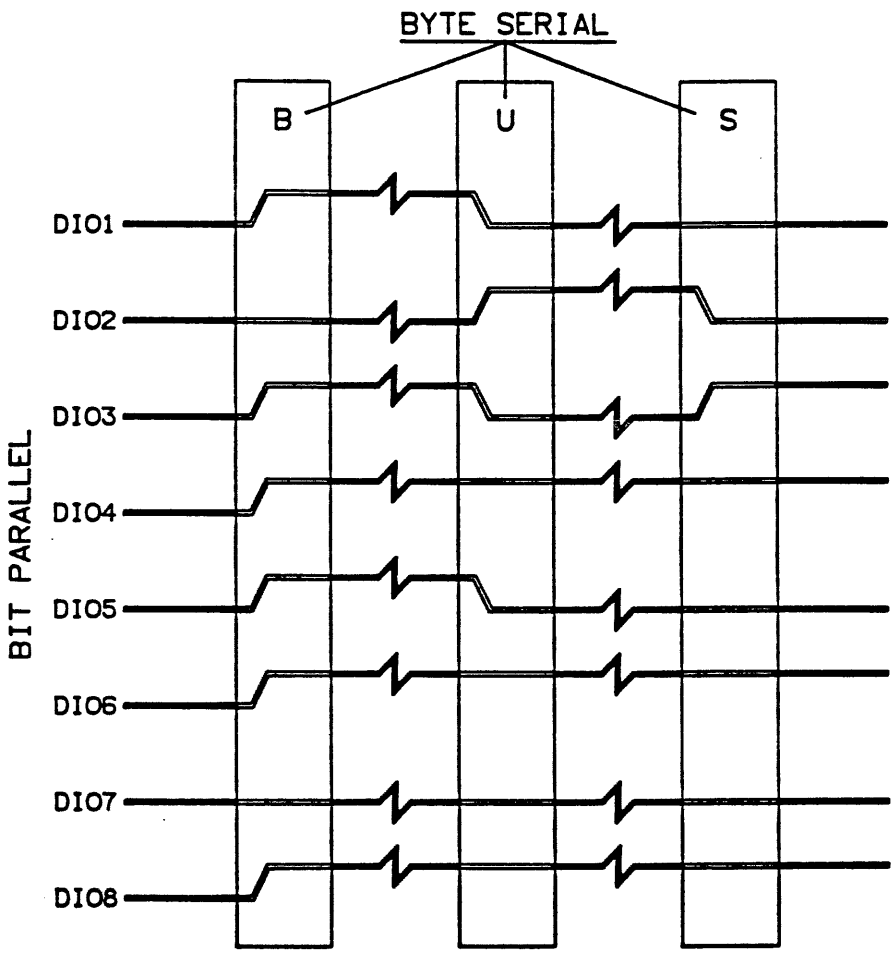


Figure 2.1 IEEE 488.1 Bus Lines

2.4.1 Data Lines

The Data Lines are an 8-bit bi-directional bus used to transfer information from device to device on the interface. The data is transfered using any commonly understood BCD, alphanumeric, or binary code. Normally this is the 7-bit ASCII (American Standard Code for Information Interchange) code. The international equivalent to this is the 7-bit ISO (International Standards Organization) code. However, other techniques may be utilized to encode information on these 8 lines. Information transferred includes interface commands, addresses, and device dependent data.

The transfer of the three byte sequence of the ASCII characters, "BUS" would occur over the Data Lines as shown in Figure 2.2. Hence the *Bit Parallel, Byte Serial* description.



ASCII/ISO	B	U	S
DECIMAL	66	85	83
OCTAL	102	125	123
HEXADECIMAL	42	55	53

Figure 2.2 Byte Transfer Diagram

2.4.2 Byte Transfer Lines

The Byte Transfer Lines are three lines used to coordinate the transfer of data over the data bus from a source (an addressed talker or a controller) to an acceptor (an addressed listener or all devices receiving interface commands) to ensure data transfer integrity. This technique has the following characteristics:

1. Data transfer is asynchronous. The transfer rate automatically adjusts to the speed of the sender and receiver(s). The bus runs at the rate of the slowest participating device. Some of the devices on the bus may not be participating in the data transfer. They would not affect the the handshake.

2. More than one device can accept data at the same time.

3. Every byte transferred undergoes the handshake.

4. When universal commands are sent over the data bus, the slowest device on the bus will determine the transfer rate during the transfer of that command. In this case, ALL devices always handshake the bytes.

5. The actual transfer rate of the data may also affected by the time it takes the instrument to take the reading and the time necessary for the controller to input the information.

IEEE 488.1 signal lines use a low-true logic convention to implement the wired-OR convention of the NRFD and NDAC lines, to provide active true-state assertion, and to reduce noise susceptibility in the true state.

The three handshake lines are:

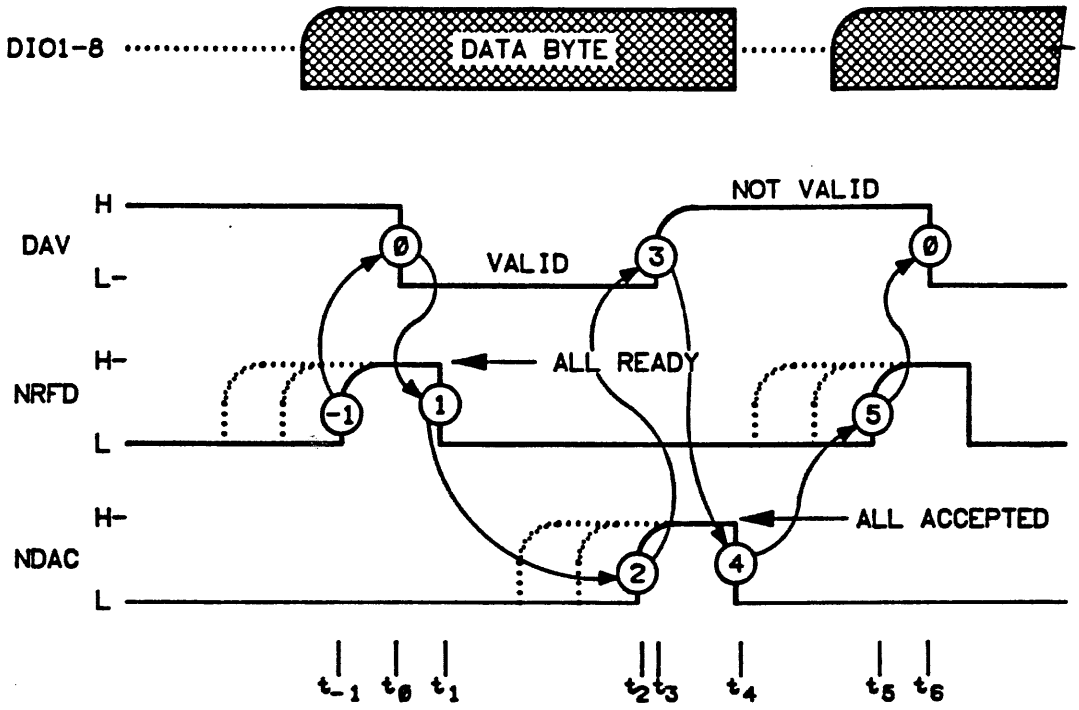
DAV — Data Valid Used to indicate the condition of the information on the Data (DIO) lines. Driven TRUE (low) by the source when data is settled and valid and NRFD FALSE (high) has been sensed.

NRFD — Not Ready For Data Used to indicate the condition of readiness of device(s) to accept data. An acceptor sets NRFD TRUE (low) to indicate it is not ready to accept data. It sets this line FALSE (high) when it is ready to accept data. However, the NRFD line to the source will not go high until all participating acceptors are ready to accept data.

NDAC — Not Data Accepted Used to indicate the condition of acceptance of data by device(s). The acceptor sets NDAC TRUE (low) to indicate it has not accepted data. When it accepts data from the DIO lines, it will set its NDAC line FALSE (high). However, the NDAC line to the source will not go high until the last/slowest participating acceptor accepts the data.

Figure 2.3 illustrates the handshake timing sequence.

The handshake sequence is depicted in flowchart form in Figure 2.4.



Preliminary: Source checks for listeners and places data byte on data lines.

t_{-1}: All acceptors become ready for byte. NRFD goes high with slowest one.

t_0: Source validates data (DAV low)

t_1: First acceptor sets NRFD low to indicate it is no longer ready for a new byte.

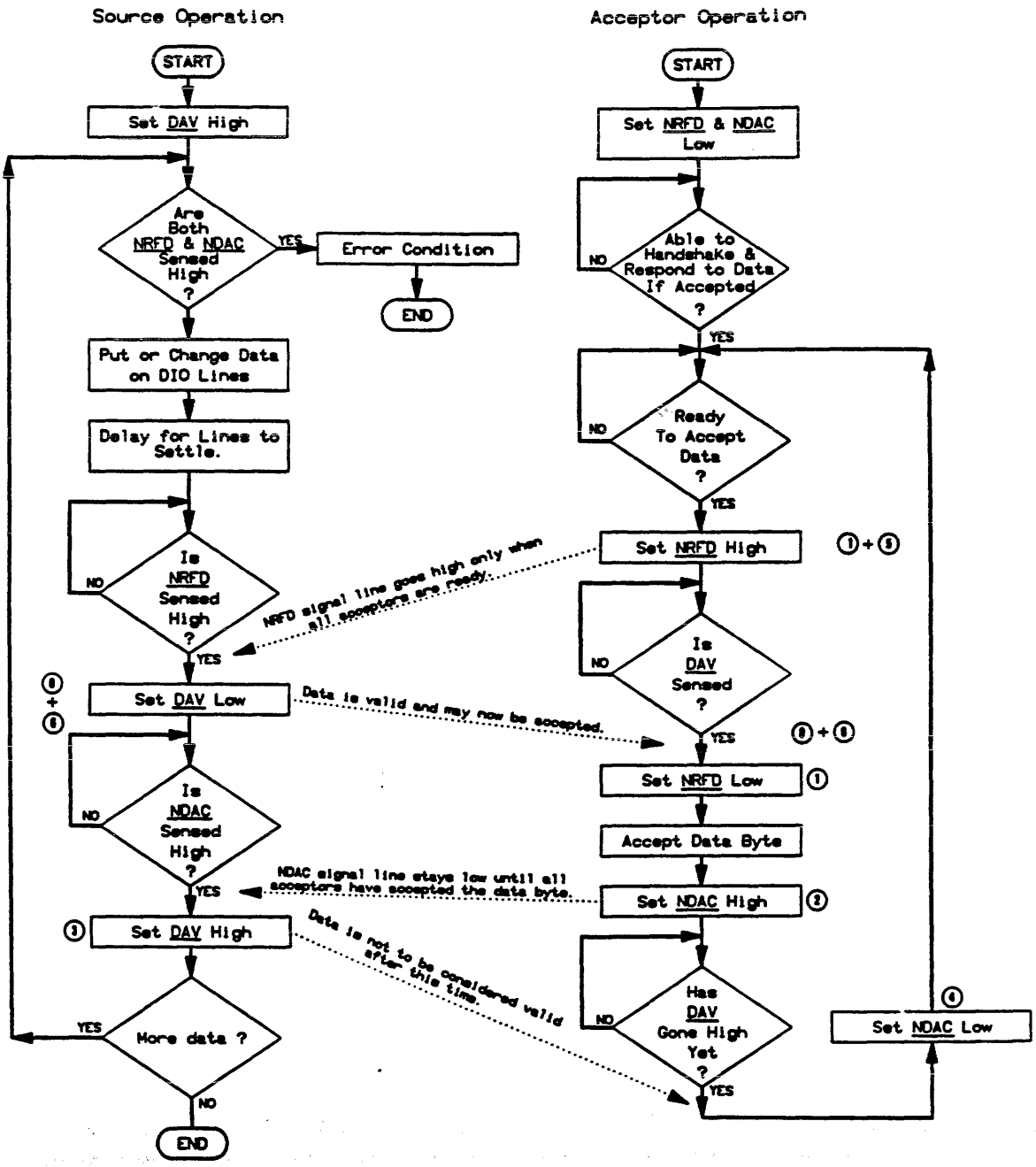
t_2: NDAC goes high with slowest acceptor to indicate all have accepted the data.

t_3: Source sets DAV high to indicate this data byte is no longer valid.

t_4: First acceptor sets NDAC low in preparation for next cycle.

t_5: Back to t-1 again.

Figure 2.3 Data Byte Transfer Timing



Logical flow of events for Source and Acceptor when transferring data using the handshake process.

Figure 2.4 Handshake Timing Sequence

Patents

Hewlett-Packard holds a patent on the three-wire handshake technique. It offers use of this technique to a company and all subsidiaries for a one time charge of $250. No disclosure is required.

Patents have been issued on this technique by the following countries: U.S.A., Italy, Germany, Holland, United Kingdom, Switzerland, France and Japan.

2.4.3 General Bus Management Lines

The General Bus Management Lines are five lines used to manage an orderly flow of information across the interface.

ATN (Attention) All devices must monitor ATN at all times and respond to it within 200 ns. When true, ATN places the interface in the "Command Mode" where all devices accept (handshake) data on the Data Lines and interpret it as Commands or Addresses. When false, ATN places the interface in the "Data Mode" where the active talker sources device dependent Data to all active listeners.

IFC (Interface Clear) The IFC line is used only by the System Controller to halt current operations (communications) on the bus (i.e. unaddress all talkers and listeners and disable Serial Poll). All devices must monitor IFC at all times and respond within 100 μs.

REN (Remote Enable) The REN line is used only by the System Controller to enable devices to be subsequently placed in the remote programming mode. When true, all listeners capable of remote operation are placed in remote when addressed to listen. When false, all devices return to local operation. All devices capable of both remote and local operation must monitor REN at all times. Devices must respond to REN within 100 μs.

SRQ (Service Request) The SRQ line is used by one or more devices to indicate the need for attention and can be used to interrupt the current sequence of events. Typically, SRQ indicates data is ready to be transmited and/or an error condition exists (e.g. syntax error, overload, trigger too fast, etc.). The controller performs a Poll of devices to determine who requested service and why. A Serial Poll will clear the SRQ.

EOI (End or Identify) When ATN is true the EOI line is used by a controller to execute a Parallel Poll (described later). When ATN is false, the EOI line is used by an active talker to indicate the last byte of a data message (e.g., burst amplitude and phase measurements, programming strings, etc.)

Table 2.3 General Bus Management Lines

Name	Mnemonic	Description
Attention	ATN	Controls whether the bus is in Command Mode (ATN TRUE) or Data Mode (ATN FALSE).
Interface Clear	IFC	Initializes the Interface to an idle state (no activity on the bus).
Service Request	SRQ	Alerts the Controller to a need for communication.
Remote Enable	REN	Enables devices to respond to Remote Program Control when addressed to listen.
End Or Identify	EOI	Indicates last data byte of a multibyte sequence; also used with ATN to Parallel Poll devices for their status bit.

2.4.4 Command Mode (ATN true)

In the Command Mode, the controller sends commands to all devices. These commands serve several different purposes:

1. Talk or listen addresses select the instruments that will source and accept data. They are all multiline messages (i.e., messages sent over the data bus). Addresses are sent to all devices.

2. Universal commands cause every instrument so equipped to perform a specific interface operation. They include five multiline commands and four uni-line commands.

3. Addressed commands are similar to universal commands, except that they affect only those devices that are addressed. These are all multiline commands. An instrument responds to an addressed command, however, only after a controller has already addressed it to be a listener or a talker.

4. Secondary commands are multiline messages that are always used in conjunction with an address, multi-line universal command, or addressed command (also referred to as primary commands) to provide additional command codes. Thus they extend the code space when necessary. For example, secondary addresses allow the controller to address subdevices in a complex instrument.

2.4.5 Data Mode (ATN False)

In the Data Mode (ATN False) device dependent data (e.g. programming data, IEEE 488.2 Common Commands, measurement data, or status data) is sent from the active talker to the active listener(s) on the interface. Note that only the addressed devices actually handshake the data. The encoding and formatting of this data is an operational area issue of the interface and as such is beyond the scope of the IEEE 488.1 Interface Standard. The IEEE 488.2 Standard covers this aspect of the interface function. Chapter 3 begins the explanation of the IEEE 488.2 Standard.

2.5 Addressing

Every IEEE 488.1 device has at least one[1] talk or listen address. Device Addresses are sent by the active controller in the Command Mode to specify who talks (via a Talk Address) and who listens (via Listen Addresses). A device's address is usually pre-set at the factory and is resettable during system configuration by an address switch, jumpers, or front panel entry. This address switch is typically located on the outside rear panel of the device but could be internal. The decimal equivalent of the five bits of this switch determines the device's address on the interface and can be from 0 to 30 inclusive. Any given Device Address specifies both the listen address and talk address (though it may only respond to one of these).

The sixth and seventh bits (DIO6-DIO7) of address messages are used to distinguish between a device's talk and listen address. High-level I/O drivers typically configure these two bits for you. Changing a device's address switch changes both. Two address codes, corresponding to address 31, are used to tell every device to UNTALK (" -") or UNLISTEN (" ?"). Therefore device address 31 is illegal and the maximum useable set of addresses totals 31 (0 through 30). Controllers usually treat IEEE 488.1 addresses via global variables, common memory, Logical Unit (LU) numbers, or symbol tables so that address changes require minimal program modification. Let's try an example.

Say you wish to set a COUNTER for an IEEE 488.1 address of decimal 25.

Decimal 25 corresponds to binary 11001. Locating the address on the back of the instrument, you set switches A1, A4, and A5 to "1" and switches A2 and A3 to "0". Typically, devices must be turned off and back on again, after changing this switch to actually set the address.

[1] Unless it's totally transparent or a Talk or Listen Only device.

Table 2.4 Talk and Listen Addresses

Switch No.					Addr. Char.		Hex.	Octal	Decimal
5	4	3	2	1	Talk	Listen	Value	Value	Value
0	0	0	0	0	@	SP	00	00	00
0	0	0	0	1	A	!	01	01	01
0	0	0	1	0	B	"	02	02	02
0	0	0	1	1	C	#	03	03	03
0	0	1	0	0	D	$	04	04	04
0	0	1	0	1	E	%	05	05	05
0	0	1	1	0	F	&	06	06	06
0	0	1	1	1	G	'	07	07	07
0	1	0	0	0	H	(	08	10	08
0	1	0	0	1	I	)	09	11	09
0	1	0	1	0	J	*	0A	12	10
0	1	0	1	1	K	+	0B	13	11
0	1	1	0	0	L	,	0C	14	12
0	1	1	0	1	M	–	0D	15	13
0	1	1	1	0	N	.	0E	16	14
0	1	1	1	1	O	/	0F	17	15
1	0	0	0	0	P	0	10	20	16
1	0	0	0	1	Q	1	11	21	17
1	0	0	1	0	R	2	12	22	18
1	0	0	1	1	S	3	13	23	19
1	0	1	0	0	T	4	14	24	20
1	0	1	0	1	U	5	15	25	21
1	0	1	1	0	V	6	16	26	22
1	0	1	1	1	W	7	17	27	23
1	1	0	0	0	X	8	18	30	24
1	1	0	0	1	Y	9	19	31	25
1	1	0	1	0	Z	:	1A	32	26
1	1	0	1	1	[	;	1B	33	27
1	1	1	0	0	\	<	1C	34	28
1	1	1	0	1	]	=	1D	35	29
1	1	1	1	0	^	}	1E	36	30
1	1	1	1	1	_	?	1F	37	31

Address 21 is usually reserved for the Computer interface Talk/Listen address (not advisable for use as an instrument address).

Address 31 is not an address but "untalk" or "unlisten.".

Figure 2.5 shows a typical address switch. Many devices allow their addresses to be changed from the device front panel instead of using this type of switch.

The additional switches on some devices are typically used to establish the device in Talk Only or Listen Only modes or to implement self-test features such as Signature Analysis or other service aids. Talk or Listen Only switches can be set to activate a talker or listener(s) without a controller addressing them to do so (typically done in a controller-less system). One additional mode enabled by these switches is Talk Always or Listen Always. In this mode the device ignores all address messages and always is enabled as a talker or a listener.

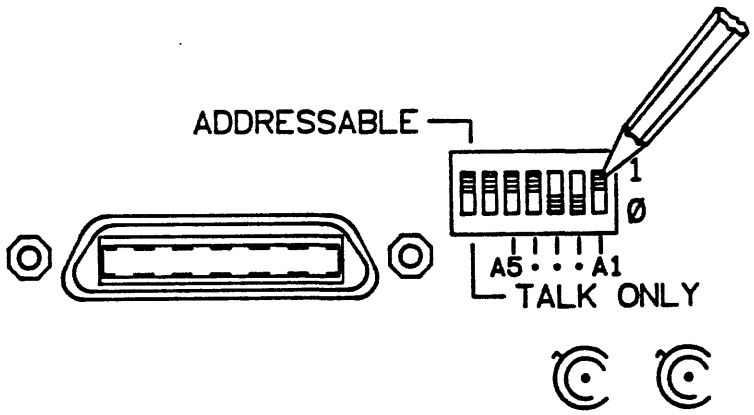


Figure 2.5 Typical Address Switch

Extended Addressing

IEEE 488.1 devices with Extended Addressing capabilities (secondary commands) recognize an additional address byte to establish itself as a talker or listener. Extended Talker and Extended Listener capabilities are mutually independent in a device (e.g., you could have a 488.1 device which is an Extended Talker but only a Basic Listener, etc.).

Multiple Addresses

IEEE 488.1 devices with multiple device capabilities which can be treated individually (e.g. Plotter/Printers, etc.) may have more than one talk or listen address (as opposed to extended addresses).

Multiple-address devices typically use fewer switch address switches — two adjacent addresses require just four switches. A single four switch setting will determine two talk addresses and two listen addresses. Four switches would control the A2 through A5 positions. (There is no switch for A1.)

Setting A5 and A2 switches to a value of one produces two listen addresses of 18 and 19 decimal.

A5	A4	A3	A2	A1	(Notice no A1, therefore, switches are set for decimal 18, 19)
1	0	0	1	-	

2.6 IEEE 488.1 Bus Commands

There are four different types of Bus Commands: Universal, Uniline, Addressed, and Secondary. These commands control the operation of the 488.1 Bus and not the actual operation of the device.

2.6.1 Universal Multiline Commands

The five[2] Universal Multiline[3] commands are:

Multiline Command	Mnemonic	Decimal Code	Hexa-decimal	Octal Code	ASCII/ISO Character
Device Clear	DCL	20	14	24	DC4
Local Lockout	LLO	17	11	21	DC1
Serial Poll Enable	SPE	24	18	30	CAN
Serial Poll Disable	SPD	25	19	31	EM
Parallel Poll Unconfigure	PPU	21	15	25	NAK

Refer to Table 4.1 for a complete list of the ASCII characters.

Untalk Command (UNT) The Untalk Command unaddresses the current talker. Sending an unused talk address would accomplish the same thing. This command is provided for convenience since addressing one talker automatically unaddresses all others.

Unlisten Command (UNL) The Unlisten Command unaddresses all current listeners on the bus. Single listeners cannot be unaddressed without unaddressing all listeners. It is necessary that this command be used to guarantee that only desired listeners are addressed.

Device Clear Command (DCL) The universal Device Clear Command causes all devices that implement the function to return to a pre-defined device-dependent state. These devices respond whether they are addressed or in remote mode. Device manuals define the cleared state for each device that recognizes the command. IEEE 488.2 specifies certain things that devices can and cannot do in response to this command. See Chapter 8 for more more detailed information.

[2] Untalk and unlisten are classified as addresses

[3] The inclusion of these commands in the instrument is optional to comply with IEEE 488.1. However some of these commands are required for IEEE 488.2. See Chapter 3.

Local Lockout Command (LLO) The Local Lockout Command disables the return-to-local control (typically a 'local' key) on devices that recognize the command. Recognizing devices accept the command whether they are addressed or in remote or local mode. REN must be set false to re-enable the return-to-local control. This also places all devices under local control.

Serial Poll Enable Command (SPE) The Serial Poll Enable Command establishes the serial poll mode for all responding talkers on the bus. When addressed to talk, each responding device will return a single eight-bit byte of status. Devices which recognize this command must have Talker interface capabilities to allow the device to output the status byte.

Serial Poll Disable Command (SPD) The Serial Poll Disable Command terminates the serial poll mode for all responding devices, returning the devices to their normal talker state where they output device-dependent data rather than status information.

Parallel Poll Unconfigure Command (PPU) The Parallel Poll Unconfigure Command resets all parallel poll devices to the idle state (no response to a parallel poll).

2.6.2 Uni-Line Commands

The four Uni-line Commands are:

Uni-line Command	Interface Management Line
Interface Clear	IFC
Remote Enable	REN
Attention	ATN
Identify (IDY)	EOI ∧ ATN

The Uni-line Universal Commands IFC and REN were described previously in Section 2.4.3. ATN was described in Section 2.4.4 and 2.4.5. IDY will be discussed in conjuction with Parallel Poll in Section 2.7.2.

2.6.3 Addressed Commands

The following table lists the commands in the Addressed Command Group.[4]

Addressed Command	Mnemonic	Decimal	Hex	Octal	ASCII
Group Execute Trigger	GET	08	08	10	BS
Selected Device Clear	SDC	04	04	04	EOT
Go To Local	GTL	01	01	01	SOH
Parallel Poll Configure	PPC	05	05	05	ENQ
Take Control	TCT	09	09	11	HT

Group Execute Trigger Command (GET) This command causes all devices which have the GET capability and are currently addressed to listen to initiate a pre-programmed action (e.g., trigger, take a sweep, etc.). Some devices may also recognize a device-dependent data character or string for this function (equivalent but requires entry into Data Mode). The GET command provides a means of triggering devices simultaneously.

Selected Device Clear Command (SDC) This command resets the devices currently addressed to listen to a device-dependent state (e.g., turn-on state, open all relays, etc.). Device manuals define the reset state for each device that recognizes the command. Same as DCL.

Go to Local Command (GTL) This command causes the devices currently addressed to listen to return to local control (exit the Remote state). The device will return to remote when it is addressed to listen again and REN is True.

Parallel Poll Configure Command (PPC) This command causes the addressed listeners to be configured according to the parallel poll enable secondary command which immediately follows this command.

Take Control Talker Command (TCT) This command causes the device that is addressed to talk to begin operating as bus controller.

[4] A device may or may not be designed to respond to any particular addressed command.

2.6.4 Secondary Commands

Secondary Commands consist of the ASCII characters 96-127 decimal. They are used for extended talk and listen secondary addresses and secondary parallel poll commands.

Secondary Command	Mnemonic	Hex Code	Octal Code	Decimal Code	ASCII/ISO Character
Parallel Poll Enable Parallel Poll Disable	PPE PPD	60-6F 70	140-157 160	96-111 112	` thru o p

Parallel Poll Enable Command (PPE) The Parallel Poll Enable Secondary Command configures the devices which have received the PPC command to respond to a parallel poll on a particular IEEE 488.1 DIO line with a particular level. Some devices may implement a local form of this configuration through the use of jumpers, switches, etc.

Parallel Poll Disable Command (PPD) The Parallel Poll Disable Command disables the devices which have received the PPC command from responding to the parallel poll.

2.7 Polling

There are two possible polling procedures on the bus, Serial Poll and Parallel Poll.

2.7.1 Serial Poll

A Serial Poll is a sequence which enables a controller to learn the status of a device. Using Serial Poll the controller can determine if a device or group of devices require service. It can also determine multi-bit status of devices on the interface, one device at a time.

Devices which can be Serial Polled will return a *Status Byte* (requires Talker subset) to the controller to indicate their status. The controller sequentially polls each individual device on the interface (sends a SPE and sequentially addresses devices to talk) and evaluates each status byte in turn. Therefore, this procedure can be lengthy in larger systems. However, the Status Byte provides the nature of the request as well as identifying the requester.

You should (although it's not mandatory) poll every device to be sure you find every requester. Remember to send Serial Poll Disable (SPD) and Untalk (UNT) commands when you're done with the procedure. Most controllers now do this for you using high level commands.

2.7.2 Parallel Poll

Parallel poll is a controller initiated operation to obtain information from several devices simultaneously. When the controller initiates a Parallel Poll, each device returns a *Status Bit* via one of the DIO lines. Device DIO assignments are made by switches, jumpers, or by the controller using the PPC (parallel poll configure) sequence. Devices respond either individually, each on a separate DIO line, or collectively on a single DIO line or any combination thereof. When responding collectively, the result is a logical AND (True High) or OR (True Low) of the groups of status bits. Configured devices must respond to a Parallel Poll (the simultaneous assertion of ATN and EOI) within 200 ns. The controller can then read the results of the poll 2 μs later. Parallel poll is often used by computers to check the status of an action, i.e., which peripherals are ready for data, sending data or receiving data. By knowing this information dead times are reduced and the system bandwidth is used more efficiently.

2.8 Electrical Aspects

General

The relation between logic and voltage levels is:

Logic Level	Voltage Level
0 (False)	> +2.0V (High)
1 (True)	< +0.8V (Low)

Driver Types

Open Collector Only	Open Collector or Tristate
SRQ, NRFD, NDAC	ATN, IFC, REN, EOI, DAV DIO 1-8

Tristate drivers are useful to reach data rates above 250,000 bytes/s.

Driver Specifications

V_{OL} < + 0.5V @ 48 mA continuous sink (tristate or open collector)

V_{OH} > 2.4V @ 5.2 mA source (tristate)

(see DC Load Line Graph (open collector) in Figure 2.6)

Receiver Specifications

Preferred (Schmitt-type)	Allowed (non-Schmitt type)
$V_{IL} = V_{tneg} < +0.8V$	$V_{IL} < +0.8V$
$V_{IH} = V_{tpos} > +2.0V$	$V_{IH} > +2.0V$

Hysteresis: $V_{tpos} - V_{tneg} > +0.4V$

Device Load Requirements

The DC and small signal AC load requirements are summarized and clarified with a typical circuit design in Figure 2.6.

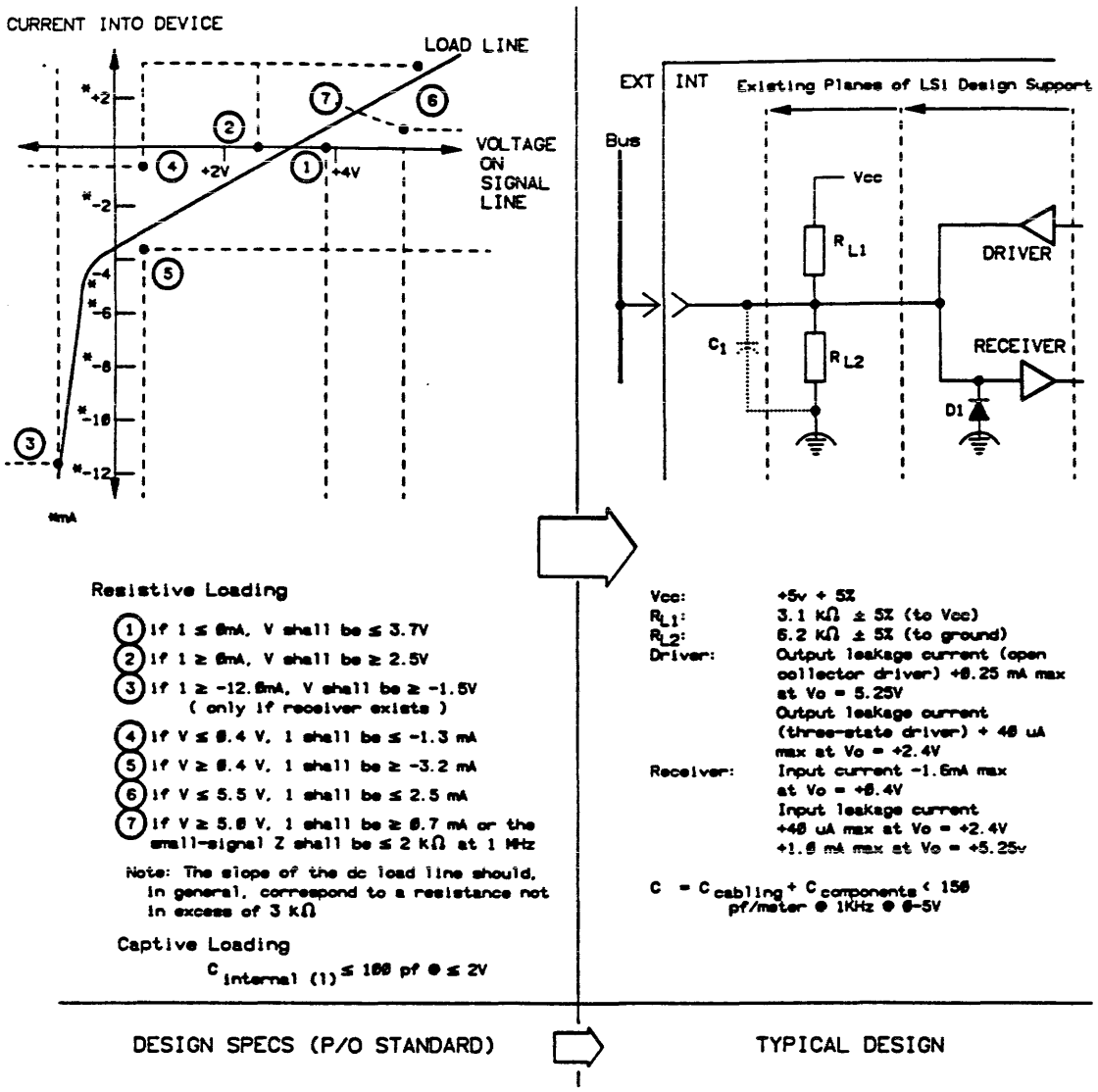


Figure 2.6 Converting Electrical Specifications to Actual Circuit

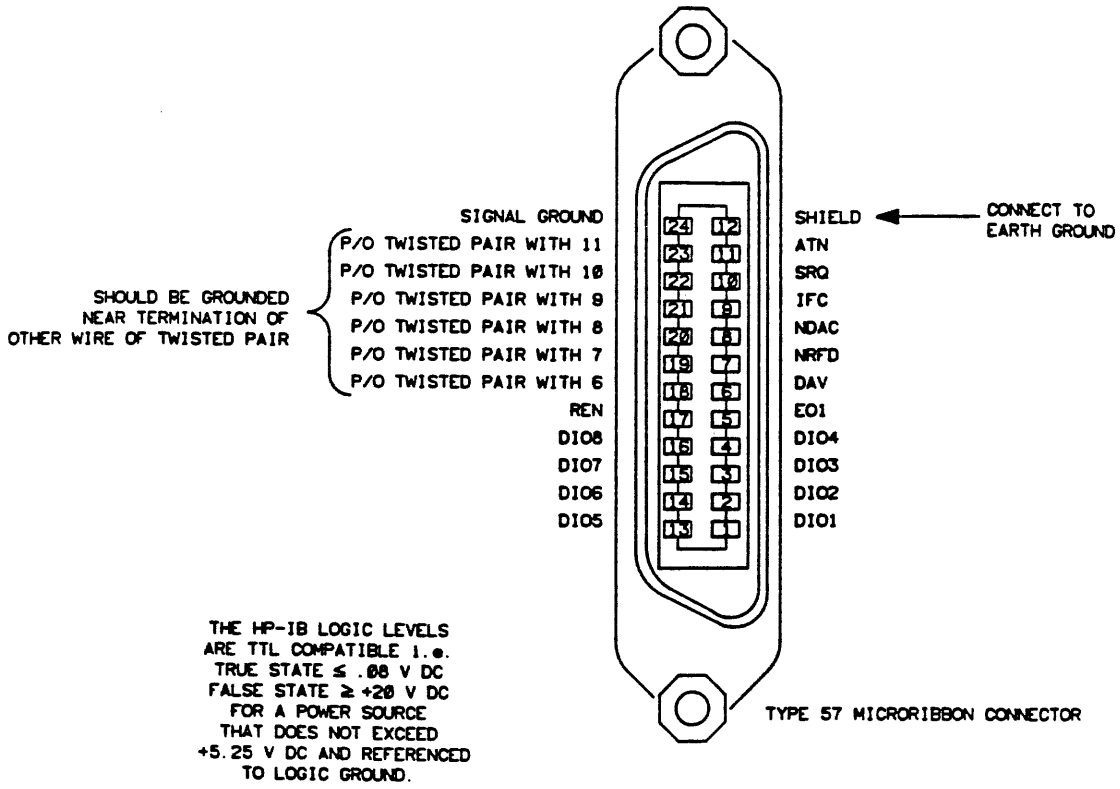


Figure 2.7 HP-IB Connector Pin Assignments

2.9 Mechanical Aspects

The connector, mounting, and cabling specifications of the interface define a robust cabling system for interconnecting devices. Devices can be interconnected in STAR, LINEAR, or combinational arrangements.

The IEEE 488/ANSI connector is a 24-pin ribbon type connector with contacts assigned as shown in Figure 2.9.

2.9.1 IEEE/ANSI Connector

Voltage rating:	200 Vdc
Current rating:	5A
Endurance:	> 1000 insertions
Temperature and Humidity:	MIL STD 202E

Suggested connectors: Microribbon (Amphenol or Cinch Series 57) or Champ (AMP)

2.9.2 IEC Connector

The IEC 625-1 connector is a 25-pin type connector (MIL-C-24308). A few key electrical and mechanical specifications:

Voltage rating:	60 Vdc
Current rating:	5A
Endurance:	> 1000 insertions
Temperature and Humidity:	IEC Publication 68 for climatic category 25/070/21.

2.9.3 Mounting. (IEEE/ANSI Connector)

Metric threads (ISO M3.9 x 0.6 type) are specified. Metric fasteners are typically colored black. Some existing cables use English threads (6-32UNK). They are colored silver. DO NOT ATTEMPT TO MATE METRIC AND ENGLISH FASTENERS, as damage to hardware may result.

2.9.4 HP-IB Cabling

HP's HP-IB Interconnect cables offer improved shielding for reduced radiated emissions from cabling in systems environments. Here's some helpful information:

Interconnection Rules

An HP-IB system may be connected together in any configuration (star or linear or combination) as long as the following rules are followed:

1. The total number of devices is less than or equal to 15.
2. The total length of all the cables used is less than or equal to 2 meters times the number of devices connected together up to an absolute maximum of 20 meters. For example, the maximum cable length is 4 meters if only 2 devices are involved. The length between adjacent devices is not critical as long as the overall restriction is met.

The cable length between adjacent devices is not critical as long as the total accumulated cable length is less than or equal to the maximum allowed. Star, linear, and combinational configurations are allowed.

It is recommended that no more than three of the connector blocks be stacked one on top of another. The resultant cantilevered structure can exert excessive force on the mounting panels when the stack of connector blocks becomes too long. The lock screws are designed to be tightened with the fingers only. Do not use a screwdriver. The screwdriver slots in the lock screws are provided for **REMOVAL** purposes only.

The new cables are completely compatible with, and can be used in combination with, the older cables. However, this will affect the continuity and effectiveness of the shielding.

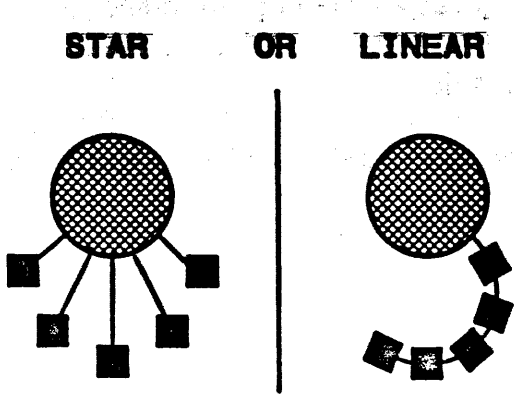


Figure 2.8 Cable Configurations

Limited Space Adapter

The HP 10834 adapter was designed to help in those cases where limited rear panel space and other design considerations have resulted in difficult cabling situations. The adapter extends the first cable approximately 2.3 cm away from the rear panel to provide clearance for other connectors, switches and cables that may be in close proximity to the HP-IB connector.

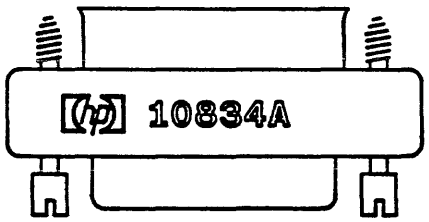


Figure 2.9 Limited Space Adapter

2.10 Revisions to IEEE 488

The November 1978 revision to the ANSI/IEEE 488.1 standard was mostly (~90%) for clarification. Heavy use in the 1970's had brought out several areas of possible misinterpretation and several useful new guidelines and recommendations.

Summary of 1978 and 1980 Revisions

- Additional restrictions on allowable combinations of interface functions were added.
- Clarification of exactly how the END message is treated in source and acceptor interface functions was added.
- A revision to the CONTROLLER function (delay) was made to ensure against the possible simultaneous assertion of DAV and ATN which could be interpreted by idle devices as COMMAND MODE information, initiating a handshake sequence.
- The electrical specification for V_{OL}, the minimum low-state output voltage for bus drivers was raised to +0.5V to accommodate modern lowpower Schottky drivers.
- More information was provided on how to maximize the Data Transfer rate over the interface.
- Warnings with guidelines about conducting and/or exiting particular operational sequences were added. For example, remembering to send Serial Poll Disable (SPD) followed by Untalk (UNT) to exit a serial poll sequence.

- A non-mandatory recommendation to mark the device's interface function codes and electrical driver type near the device's connector to aid the system designer and user.

2.11 Design And Service Aids

Among the most useful design aids for designing ANSI/IEEE 488.1 and IEC 625-1 capabilities into a product are:

- LSI chips for implementing CONTROLLER, TALKER, and LISTENER interface function combination
- A flexible BUS ANALYZER
- Timing Analysis for the bus (plus State Analysis)
- Serial Analysis for Data Communications and Interfacing applications in a Distributed Measurement Systems Environment

LSI IEEE 488.1 IC's

LSI chip versions of the Controller, Talker, and Listener Interface Functions are available to facilitate your design process. These chips can implement selected functions or combinations. More recent chips include Controller capabilities (may be a separate chip). In all cases they typically replace between 40 to 60 MSI or SSI parts per function (Controller, Talker, Listener) on partially dedicated I/O boards. That's 120 to 180 parts for a full hardware version.

To perform the same functions via a typical MPU/ROM implementation might require 500 bytes per function. That's a full 1.5K bytes and a dedicated processor.

Bus Analyzer

A flexible Bus Analyzer is a very useful product aiding the HP-IB hardware/software designer in development and diagnosing HP-IB hardware/software problems. Most Bus Analyzers operate with complete Talker, Listener, Controller, and System Controller Interface Function capabilities which are mutually independent and exhibit near-ideal (typically $<$750 ns with $<$200 ns possible) accept times (T3 in the IEEE 488.1 standard) for almost complete transparent bus monitor and test applications. Other applications include device and controller simulation and interface function verification.

Timing Analysis

Timing Analysis is sometimes useful when optimizing or characterizing HP-IB product or system performance; or when diagnosing noise, timing, or software-synchronization-related faults in a product or system. Most Timing Analyzers provide at least the 16 channels required to monitor HP-IB signal lines and more than enough bandwidth.

Serial Data Analysis

Serial Data Analysis (Monitoring or DTE/DCE Simulation) is sometimes useful when you expect your HP-IB Measurement System to be or evolve into a Measurement Node in a larger distributed system environment via a serial network link or HP-IB extension technique. The network link would tie the local measurement-intensive system to a centralized compatational-intensive system via a local networking scheme (Public Data Networks).

2.12 Optimizing Performance

Many HP-IB users ask how they can optimize system performance and overcome constraints. There are three areas of HP-IB performance that are most often asked about.

- Surpassing the 15 device-per-bus constraint (Number of Devices per Bus).
- Surpassing the 20 meter total cable length constraint (Total Accumulated Cable length).
- Maximizing the data transfer rate of the interface. (Maximum Transfer Rate).

Each of these topics is discussed separately.

2.12.1 Number Of Devices Per Bus

Fifteen devices can fill about three 56-inch bays with equipment and constitutes a fairly elaborate system. In most cases, this is not a major restriction. If you should have a need for more devices you can use an additional interface (another card in your desktop or minicomputer). If you run out of slots, some computers have I/O Expanders for increasing the number of slots available.

2.12.2 Total Accumulated Cable Length

For data rates below 500 kbytes/s you are constrained to 20 meters total or 2 meters per device, whichever is less. There are 2 techniques for surpassing this limitation using HP-IB extenders.

Intra-Facility and Inter-Facility HP-IB Extension

Intra-Facility HP-IB Extension Can:

- Extend HP-IB capabilities up to 1000 meters away.
- Save you money on cabling (can be up to $10/ft!).
- Avoid the cost of additional computers.
- Protect your computer from harsh environments, extreme noise, or unwanted user access.
- Preserve HP-IB flexibility at the remote end of the extension.
- Be partially or totally transparent to the user.

Inter-Facility HP-IB Extension Can:

- Provide all of the benefits of Intra-Facility over unlimited distances with leveraged savings.
- Provide you with auto dial-up on one or more remote instrument clusters.

2.12.3 Maximum Transfer Rate

You'll find for most instrument systems that your throughput is limited by the instruments in the system. Precision measurements typically take a long time relative to the cycle times in desktop or minicomputers. High resolution (5½ - 6½ Digit) voltage measurements using integrating analog-to-digital techniques, precision low frequency counting, and narrow band spectrum analysis are all good examples of relatively slow real-time measurement processes. As instruments and peripherals get smarter and digital signal processing becomes easier and less expensive to implement in instrument, the short term (high-speed sampling, burst measurements, computer "dumps," block memory transfers) HP-IB demands will increase. In these cases the system throughput can become bound by the transfer rate of the computer interface.

Speed

With one device for every 2 meters of cable, the data rate can be 250 kbytes/s over distances of 20 meters using open collector drivers. Tristate drivers can increase the data rate to 500 kbytes/s.

To achieve data rates faster than 500 kbytes/s and up to 1 Mbyte/s do the following:

1. Be sure all high rate devices use Tristate drivers (Interface capability E2) and that every device is turned on.

2. Minimize cable length! Do not exceed 15 meters total cable length and ensure less than one device for every meter of cable.

3. Limit the capacitance of each device on each line to at most 50pf (@ <2V) (except REN and IFC).

4. All high rate Talkers should use a minimum multiline message settling time (T1 in the IEEE-488 Standard) of 350 ns.

5. Also buffered data byte storage in a device is advantageous. Devices with a T1 value less than 500 ns, an internal device capacitance of >50pf, or multiple resistive loads should be marked accordingly (typically done on Controllers).

6. See also IEEE standard 488—1987 Section 5.2.3, Higher Speed Operation.

Devices Powered Off And On

Systems will operate normally with up to ⅓ of the devices powered off and even more as long as V_{OH} on each line on each device still exceeds the +2.5V specification. Turning a device on or off while a system is running may cause faulty operation.

2.12.4 Improving Software Performance

If speed is critical in your application the following guidelines may prove useful:

General (Device Dependent)

1. Familiarize yourself well with the operational aspects of the devices you're optimizing your routines for. Newer devices have fast-handshake and speed-advantageous features (e.g. buffering, program storage, hardware overlap, etc.) built-in.

2. Use interrupt-driven processes and down-load smart devices where possible. Some card-cage-type instruments can perform many time consuming tasks through hardware features that otherwise might require significant software dedication and the associated overhead.

3. Avoid unnecessary unaddressing and readdressing steps where possible.

4. Use HP-IB Commands rather than device dependent messages where possible.

5. Suppress unneeded terminators.

6. Use system commands (already optimized) and binaries when they exist.

7. Get to know your HP-IB Specialist and Systems Engineers if you use HP equipment. They're highly trained specialists with timely answers to critical questions when you need them. Systems Engineers are also available on a consulting basis for a fee when you need dedicated consulting.

Programming Language Dependent

Develop the driver at the lowest-level possible and appropriate (e.g. assembly for repetitive fixed point processes, interface control for single interface commands, etc.). But develop larger programs and non speed-critical subprograms at higher levels as appropriate.

Use compiled languages when available. If you can link compiled modules in an interpretive environment, you may find it's speed advantageous.

2.12.5 Generally Helpful Information

Avoid Typical HP-IB System-Related Problems. Here are some suggestions:

1. Use an ALGORITHMIC DESIGN approach. It's simple, well structured, and has a history of success.

   - DEFINE your system needs well.
   - DESIGN a system solution.
   - EVALUATE the expected cost-effectiveness of your design.
   - BUILD the chosen solution.
   - USE the system once built.

2. Order system prestudy or device manuals and application notes as early as possible.

3. An HP-IB BUS IMPLEMENTATION WORKSHEET is sometimes useful to visualize the ADDRESS and MESSAGE compatability of the devices in your system. This technique can aid the system programmer in larger HP-IB systems. Once you've got documentation on the devices, the programmer can fill in the SEND and RECEIVE message capabilities of the devices vertically and compare horizontally. A blank worksheet is found at the end of this chapter.

4. Develop hardware and software block diagrams or flowcharts for the tasks (measurement, test, control, data manipulation, presentation, etc.) you will

need. You could do this while waiting for component or system delivery. Even better, do it as part of the Definition and Design steps of your system design.

5. Prepare the systems installation site adequately. Don't forget provisions for power distribution, interference protection, and such environmental considerations as temperature, humidity, static protection, and physical clearances. Refer to the installation documents with your system and system components. A typical pre-installation planning checklist is found at the end of this chapter.

6. Appoint a system manager who is responsible for maintenance and calibration schedules, operator training, configuration control and systems logs as well as for ordering such consumables as paper, ink, diskettes, and cartridges. Giving the user this responsibility will typically result in further job enrichment and self-fulfillment.

7. Don't try to automate too much or the wrong things. Some interconnect processes may best be done manually to avoid the error terms associated with system switching. Many processes just do not make sense to automate (Microwave connections, etc.).

8. Take care when estimating software requirements for the system. Expect a decreasing exponential learning curve on test programs. The time to learn a mini-computer based system and write the first test is typically 10 times that required for an equivalent desktop computer based system. The steady state ratio drops to about 2 times as long as fluency and mastery are achieved.

9. The STAR cabling configuration will minimize worst-case transmission path lengths but can lump large capacitance values at a single plane on the line. The LINEAR cabling configuration may produce longer electrical lengths but provides more control to distribute capacitive line loads for maximum error-free transmission.

HP-IB BUS IMPLEMENTATION WORKSHEET

MESSAGE		DEVICE													
INSTRUMENT IDENTIFICATION AND HP-IB ADDRESS	MODEL														
	LISTEN														
	TALK														
	5 BIT VALUE														
DATA															
TRIGGER															
CLEAR															
LOCAL															
REMOTE															
LOCAL LOCKOUT															
CLEAR LO & SET LOCKOUT															
REQUIRE SERVICE															
STATUS BYTE															
STATUS BIT															
PASS CONTROL															
ABORT															

S = SEND ONLY R = RECEIVE ONLY S&R = SEND AND RECEIVE N = NOT IMPLEMENTED HP_IB.TUT F.2.10

Figure 2.10 Bus Implementation Worksheet

INSTRUCTIONS.

Check when each planning question has been completed. If a question does not pertain to your installation, mark the question N/A.

Suitability of site

_____ 1. Is proper and adequate power available to site?

_____ 2. If below ground level, is the water drainage system adequate?

_____ 3. Has possible need for rigging been investigated?

_____ 4. Are elevators adequate to support size and weight of equipment?

_____ 5. Will stairways allow passage of equipment?

_____ 6. Will flooring enroute to installation site support weight of equipment?

_____ 7. Will all doorways enroute to installation site allow clearance for equipment?

_____ 8. Will hallways and corridors enroute to installation site allow clearance

_____ 9. Are RFI sources or susceptible instruments nearby?

Floor plan

_____ 10. Has grid layout been completed?

_____ 11. Does it show the locations for all the proposed equipment?

_____ 12. Does it allow for adequate clearance in front and rear of the combining case or cabinet for operation and service?

_____ 13. Does it allow for future expantion?

_____ 14. Is sufficient space provided for personnel safety, comfort and freedom of movement?

_____ 15. Does it show locations of all doors and aisle ways?

_____ 16. Is sufficient space provided for supplies?

Electric Power

_____ 17. Have tests been conducted to determine the voltage and frequency fluctuations throughout the day?

_____ 18. If tests indicated a greater than -10% or +5% fluctuation, have provisions been made for voltage or frequency regulation?

_____ 19. Have provisions been made for the installation of a sufficient amount of receptacles throughout the site for free-standing equipment?

_____ 20. Considering all factors such as wire size, distribution equipment, etc., is the proposed electrical installation plan adequate for the presently proposed system and possible future expansion?

Temperatures

_____ 21. Has the installation area been checked for minimum temperature ranges of 0°C (32°F) to 55°C (131°F) respectively?

Signal and Power Cables

_____ 22. Have signal and power cable lengh restrictions been adhered to?

_____ 23. Will cables other than those supplied with equipment be required?

_____ 24. Will power plugs other than that supplied with equipment be required?

Safety Precautions

_____ 25. Has the need for emergency exits been considered?

_____ 26. Have provisions been made for an adequate number of CO_2 fire extinguishers?

_____ 27. Is there a FIRST AID KIT available at the installation site?

Figure 2.11 Installation Planning Checklist

Chapter 3
The IEEE 488.2 Standard

3.1 Overview

With the adoption of IEEE 488 in 1975, instrument designers and users solved many interfacing problems. However, they also realized that they did not know and understand all of the problems of interfacing instruments to computers. Therefore, they attacked the first hurdle of connecting computers and instruments, the electrical connection and very basic aspects of communication. Even though IEEE 488 eliminated the problem of looking for the right type of connector, and which signal line went to which pin, it purposely left some other problems unsolved.

During the next ten years instrument users discovered many of the difficulties in implementing IEEE 488 systems. They found that device capabilities could range from Listen Only to System Controller. Because of this, the system designer had to determine if each device had the appropriate interface capability as well as other performance criteria. Designers also found that each company handled message protocol and data formating in a different manner. Again, device communication compatiblity became an important issue. Each company also used different command sets for similar type functions in different instruments. To compound the problem further, each instrument reported its status in a different set of bits in the status byte. This meant that replacing a device in a system with different device required extensive and expensive reprogramming.

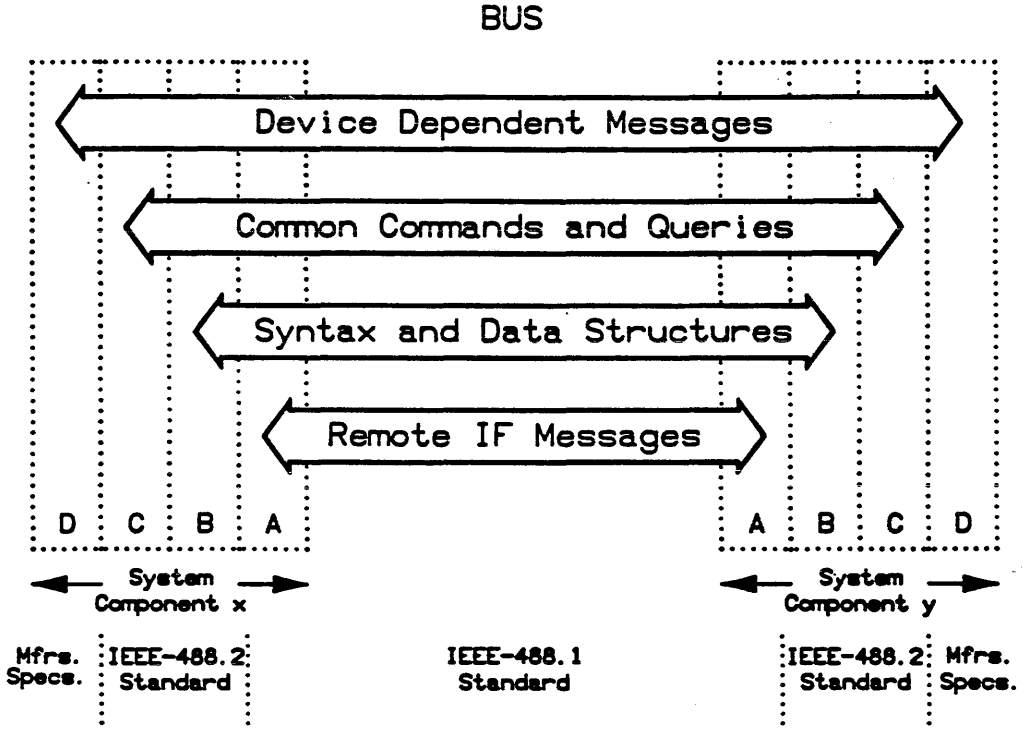
A first attempt to standardize data formats resulted in the creation of IEEE 728 *Recommended Practice for Code and Format Conventions for Use with IEEE Std 488-1978*. These formats evolved over time as designers and users determined which formats worked well and which did not. Since these were only recommendations and included several different formats for the data it only partially solved the problem. But it did help reveal the information necessary to develop a better data format standard.

To solve these problems, the IEEE developed the IEEE Std 488.2 *Codes, Formats, Protocols and Common Commands For use with ANSI/IEEE Std 488.1-1987*. This latest standard describes a set of codes, data formats, message protocols, and common commands to use with the very successful IEEE 488 (now 488.1) standard. IEEE 488.2 addresses many of the problems that users encounter in using 488.1.

To help you understand this new interface standard, consider how an interface is defined. The instrument interface can be divided into several functional layers, shown graphically in Figure 3-1. The lowest layer is the Remote Interface Messages layer or the IEEE 488.1 Bus. This layer is the physical interface. It includes the mechanical connector, wiring, electrical signals, handshaking of data, etc.

The IEEE 488.2 standard defines the middle two layers. They consist of the Syntax and Data Structures layer and the Common Commands and Queries layer. The Syntax and Data Structures layer defines how data is communicated between devices. For example, it defines the usage of the ASCII character set for data representation. It also defines data formats for binary numbers.

The final layer is the Device Dependent Messages layer which each manufacturer defines. These messages could also be termed the device commands. They tell the device what function to perform.



WHERE:

Layer D represents Device Functions
Layer C represents Common System Functions
Layer B represents Message Communication Functions
Layer A represents Interface Functions (IF)

Figure 3.1 Functional Layers Diagram

So what does this new standard really provide? How does it solve the problems found in the past? What were the problems?

The problems encountered using IEEE 488.1 alone are:

- Varying device interface capabilites — for example, two different Voltmeters that perform the same measurement. One might have Talk-Only capability, while the other could talk and listen.
- No common data format — two devices could physically communicate over the 488 bus but one device might not use the data format that the other uses.
- No standard message protocol — for example, the order in which a device sends commands and data.
- No common command set — two devices that perform identical functions may have completely different device dependent commands.
- Status reporting unique to each device — each device reports status in a different set of bits in the status byte. In addition, each device could include different status information.

How does IEEE 488.2 solve these problems? It defines:

- A minimum set of IEEE 488.1 interface capabilities (See Table 3.1)
- Data formats and syntax (how data is represented)
- Device message protocols (what is sent and when)
- Common command set — commonly needed commands defined.
- Status reporting model — clearer status reporting definition

Let's take a closer look at each one of these problems and how IEEE 488.2 solves them.

3.2 Required Interface Capabilities

IEEE 488.2 defines a set of minimum capabilities that each device must have. Table 3.1 lists these required capabilities.

In essence, all devices are able to send and receive data, request service, and respond to a device clear command. It also details the minimum capabilities that a device has when it implements controller, parallel poll, and the remote local functions. The user now knows that every 488.2 device in the system has certain capabilities. This simplifies system design.

Table 3.1 Minimum IEEE 488.1 Capabilities

Capability	Code*	Comment
Source Handshake	SH1	Full Capability
Acceptor Handshake	AH1	Full Capability
Talker	T(TE)5, or T(TE)6	Basic Talker, Serial Poll, untalk on MLA
Listener	L(LE)3, or L(LE)4	Basic Listener, unlisten on MTA
Service Request	SR1	Full Capability
Device Clear	DC1	Full Capability
Remote Local	RL0 or RL1	None or Full Capability
Parallel Poll	PP0 or PP1	None or Full Capability
Device Trigger	DT0 or DT1	None or Full Capability
Controller	C0 or C4 with C5, C7, C8 or C11	None or Respond to SRQ, Send IF Msg., pass, receive control
Electrical Interface	E1 or E2	Open Collector or Tristate

*See Appendix C for Code definitions

3.3 Data Formats and Syntax

IEEE 488.2 provides a set of data formats for everything from decimal numbers to arbitrary strings of characters. For example, it defines a format for binary, octal, and hexadecimal numbers. It also defines formats to send long blocks of 8-bit bytes or strings of ASCII characters. Table 4.2 lists these formats.

IEEE 488.2 also introduces a new concept that makes it possible for older devices to communicate with devices that use this new standard. This concept is

Forgiving listening. Precise talking.

It requires devices to accept a wide variety of data formats and codings, *forgiving listening*. But, it restricts the data transmitted to a rigorous set of formats, *precise talking*. New devices will be able to communicate with devices using older format standards such as IEEE 728.

Chapter 4, Data Coding and Formats, describes these formats in greater detail. But remember the idea *forgiving listening, precise talking*. It is important to many of the ideas presented later in this book.

3.4 Device Message Protocols

Device message protocols enable devices to communicate by defining how to send commands, parameters and data. IEEE 488.2 carefully describes a message exchange protocol, for the 488.1 bus. It describes what to do when a device receives multiple commands, an incomplete command or when it is interrupted while processing a command. It defines a set of device operational states to implement this protocol. These states are:

STATE	PURPOSE
IDLE	Wait for messages
READ	Read and execute messages
QUERY	Store responses to be sent
SEND	Send responses
RESPONSE	Complete sending responses
DONE	Finished sending response
DEADLOCK	The device cannot buffer more data
UNTERMINATED	The device has attempted to read an unterminated message
INTERRUPTED	The device was interrupted by a new message while sending a response

These states tell the device how to react when an exception condition occurs. It's not difficult to know what to do when reading or sending a single byte on the bus. However, decisions become much more complex when the device is interrupted or doesn't receive a terminated command. The IEEE 488.2 standard defines what the device will do in these situations.

IEEE 488.2 also defines how devices exchange data. It describes the order in which data bytes are sent. The standard also states that a device cannot send data until commanded to do so. When a device receives a new command it clears its output queue and begins work on that command. IEEE 488.2 defines the data format for the response to each query.

3.5 Common Command Set

Every 488.1 device performs a common set of functions in order to communicate on the bus and report its status. In the past, each device used a different set of commands to enable these functions. IEEE 488.2 details a list of common commands that provide uniform communication with devices. These commands are grouped by the function they perform. These groups are:

- System Data
- Internal Operations
- Status & Event
- Synchronization
- Parallel Poll
- Device Trigger
- Controller
- Auto Configure
- Macros
- Stored Settings

The standard requires all devices to implement certain commands. This permits test system programmers to write code modules that will work with all devices.

Common Command Groups

Lets take a brief look at the common command groups.

System Data These commands store or retrieve information about devices in the system. This information includes device descriptions, and options.

Internal Operations These commands relate to the internal operation of a device. They include operations such as reseting, testing, or calibrating the device.

Status & Event These commands control the status structures of the device.

Synchronization These commands synchronize the operations of the devices within a system.

Parallel Poll These commands control how a device responds to a 488.1 Parallel Poll. They also permit you to obtain the same information without performing an actual Parallel Poll.

Device Trigger These commands perform a Device Trigger, and control how a device responds to a Trigger command.

Controller This command defines the means of passing control between devices.

Auto Configure This command set provides a means to set device addresses within a system.

Macros This group of commands gives the user the ability to define new commands using the commands defined by the device and 488.2.

Stored Settings These commands save and restore the state of the device.

Chapter 7, Common Commands, gives greater detail on the operation and syntax of these commands.

3.6 Status Reporting Model

IEEE 488.2 describes a standard status reporting model so that controllers know how to ask a device for its status. Figure 3.2 shows the block diagram for the IEEE 488.2 Status Model. The status model uses the IEEE 488.1 status byte. This byte contains seven single-bit summary messages from Status Data Structures. IEEE 488.2 defines two of these bits, Event Status Bit (ESB) and Message Available (MAV). IEEE 488.1 defines the RQS bit. The status data structures are registers or queues. The user can enable a device to request service depending on the state of these summary bits. Chapter 6, Status Reporting, gives greater detail on how these structures operate.

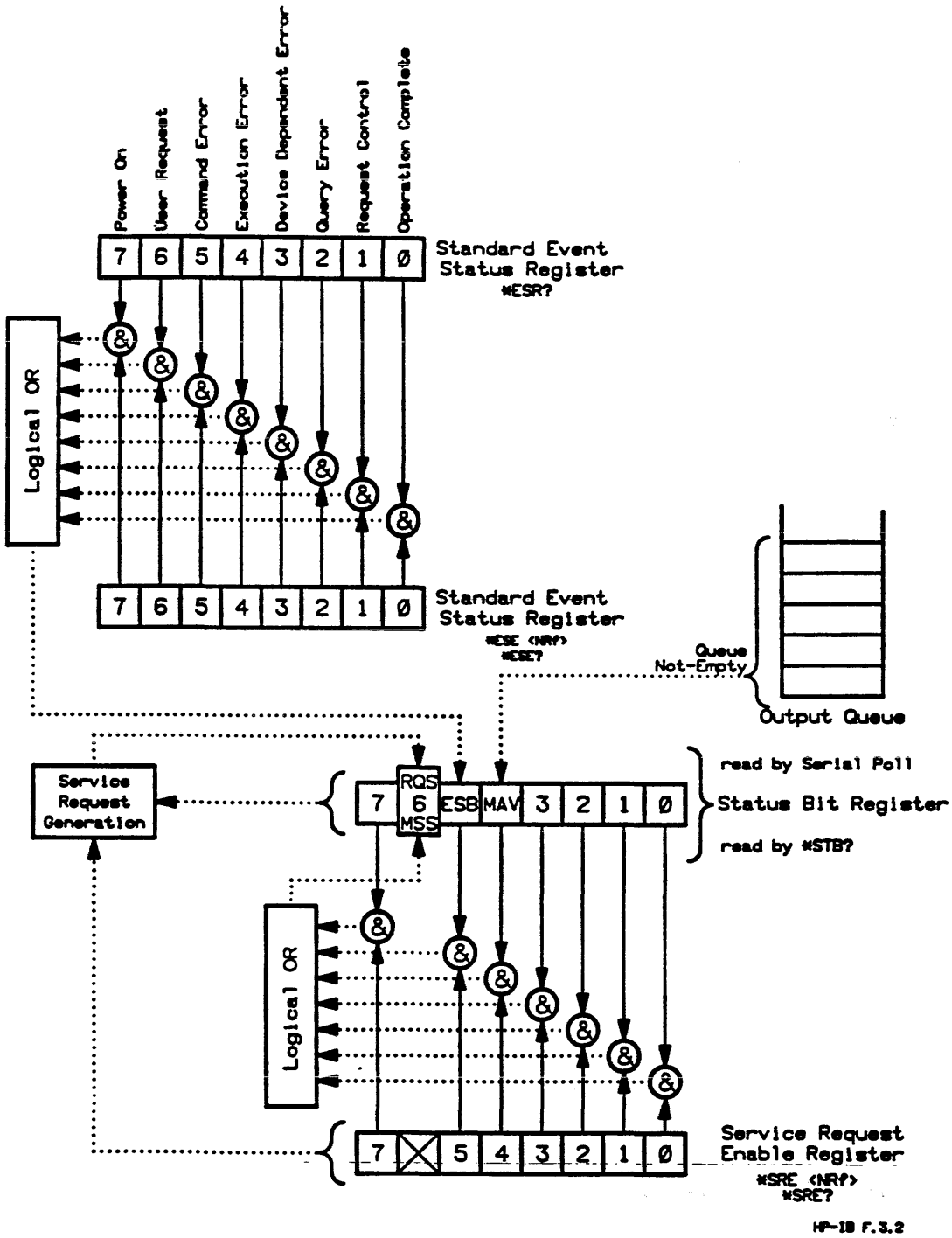


Figure 3.2 Standard Status Model

Chapter 4
Data Coding and Formats

4.1 Overview

The original IEEE 488 interface provided instrument designers and users with an interface that solved many problems. However, it did not define a data format, or coding protocol. It simply stated that any standard alphanumeric, binary, or BCD code may be used.

The IEEE 488.2 standard defines a set of data codes and formats for everything from decimal numbers to arbitrary strings of characters.

4.2 Message Data Coding

IEEE 488 used the ISO 7-bit code (ASCII) to document the interface commands. Many 488 devices use ASCII for information coding. But since any "standard" code was acceptable, many devices used many different forms of binary coding.

IEEE 488.2 specifies three sets of codes, ASCII 7-bit (for alphanumerics), Binary 8-bit Integer and a Binary Floating Point Code. Then, using these codes, it defines data formats for decimal, octal, and hexadecimal integers, decimal floating point numbers, strings, character strings, and arbitrary strings. Most of these formats use the ASCII code to represent the data.

4.2.1 ASCII 7-Bit Code

IEEE 488.2 specifies the ANSI X3.4-1977 ASCII 7-bit code as the common data code for device dependent messages. The seven bits, B1 through B7 correspond to the data lines DIO1 - DIO7 with DIO8 ignored. Table 4.1 contains the ASCII 7-bit Code Chart.

4.2.2 8-Bit Binary Integer

In some cases it is more efficient to pass data between devices in some internal format. This eliminates the need to convert the data to ASCII before sending it. It also eliminates the need to convert the data back to this internal format when the device receives it.

When using binary data, be careful to insure that the internal formats of the two devices are *identical*. When in doubt, use an ASCII coded format. IEEE 488.2 recommends formats and codings to use for internal data. But remember, these are only *recommendations*.

B7 B6 B5 / BITS B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0 0	20 DLE 10 16	40 SP 20 32	60 0 30 48	100 @ 40 64	120 P 50 80	140 ` 60 96	160 p 70 112
0 0 0 1	1 GTL SOH 1 1	21 LLO DC1 11 17	41 ! 21 33	61 1 31 49	101 A 41 65	121 Q 51 81	141 a 61 97	161 q 71 113
0 0 1 0	2 STX 2 2	22 DC2 12 18	42 " 22 34	62 2 32 50	102 B 42 66	122 R 52 82	142 b 62 98	162 r 72 114
0 0 1 1	3 ETX 3 3	23 DC3 13 19	43 # 23 35	63 3 33 51	103 C 43 67	123 S 53 83	143 c 63 99	163 s 73 115
0 1 0 0	4 SDC EOT 4 4	24 DCL DC4 14 20	44 $ 24 36	64 4 34 52	104 D 44 68	124 T 54 84	144 d 64 100	164 t 74 116
0 1 0 1	5 PPC ENQ 5 5	25 PPU NAK 15 21	45 % 25 37	65 5 35 53	105 E 45 69	125 U 55 85	145 e 65 101	165 u 75 117
0 1 1 0	6 ACK 6 6	26 SYN 16 22	46 & 26 38	66 6 36 54	106 F 46 70	126 V 56 86	146 f 66 102	166 v 76 118
0 1 1 1	7 BEL 7 7	27 ETB 17 23	47 ' 27 39	67 7 37 55	107 G 47 71	127 W 57 87	147 g 67 103	167 w 77 119
1 0 0 0	10 GET BS 8 8	30 SPE CAN 18 24	50 (28 40	70 8 38 56	110 H 48 72	130 X 58 88	150 h 68 104	170 x 78 120
1 0 0 1	11 TCT HT 9 9	31 SPD EM 19 25	51) 29 41	71 9 39 57	111 I 49 73	131 Y 59 89	151 i 69 105	171 y 79 121
1 0 1 0	12 LF A 10	32 SUB 1A 26	52 * 2A 42	72 : 3A 58	112 J 4A 74	132 Z 5A 90	152 j 6A 106	172 z 7A 122
1 0 1 1	13 VT B 11	33 ESC 1B 27	53 + 2B 43	73 ; 3B 59	113 K 4B 75	133 [5B 91	153 k 6B 107	173 { 7B 123
1 1 0 0	14 FF C 12	34 FS 1C 28	54 , 2C 44	74 < 3C 60	114 L 4C 76	134 \ 5C 92	154 l 6C 108	174 ¦ 7C 124
1 1 0 1	15 CR D 13	35 GS 1D 29	55 - 2D 45	75 = 3D 61	115 M 4D 77	135] 5D 93	155 m 6D 109	175 } 7D 125
1 1 1 0	16 SO E 14	36 RS 1E 30	56 . 2E 46	76 > 3E 62	116 N 4E 78	136 ^ 5E 94	156 n 6E 110	176 ~ 7E 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 / 2F 47	77 ? 3F 63	117 O 4F 0	137 - 5F 95	157 o 6F 111	177 RUBOUT (DEL) 7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

KEY: octal 25 PPU / NAK / hex 15 21 — Message Mnemonic, ASCII/ISO character, decimal

Table 4.1 ASCII 7-bit Code Chart

The standard recommends the following format for Binary Integers. The data may contain as many 8-bit bytes as desired. The most significant byte is sent first. The order of the bits corresponds to the order of the bus data lines. For example the most significant bit corresponds to DIO8 and the least significant corresponds to DIO1. The data is assumed to be right justified, and in 2's complement notation.

For example, the number 7 (decimal) would be coded as

DIO							
8	7	6	5	4	3	2	1
0	0	0	0	0	1	1	1

4.2.3 Binary Floating Point

The IEEE 488.2 standard recommends using the IEEE Std 754-1985 for representing floating point numbers. Appendix B contains the definition for IEEE floating point binary numbers. Refer to it if you want to know how they are built. Fortunately, most devices take care of that function for you. However, it is important to know that this data is received as an <Arbitrary Block Program Data>. It is sent as a <Definite Length Arbitrary Block Response Data>. These data formats are explained in the following section.

Table 4.2 Talker and Listener Formats

Listener Formats	Status
<Decimal Numeric Program Data>	Required
<Character Program Data>	Optional
<Suffix Program Data>	Optional
<Non-Decimal Numeric Program Data>	Optional
<String Program Data>	Optional
<Arbitrary Block Program Data>	Optional
<Expression Program Data>	Optional
Talker Formats	
<NR1 Numeric Response Data>	Required
<Arbitrary ASCII Response Data>	Required
<Character Response Data>	Optional
<NR2 Numeric Response Data>	Optional
<NR3 Numeric Response Data>	Optional
<Hexadecimal Numeric Response Data>	Optional
<Octal Numeric Response Data>Optional	
<Binary Numeric Response Data>	Optional
<String Response Data>	Optional
<Definite Length Arbitrary Block Response Data>	Optional
<Indefinite Length Arbitrary Block Response Data>	Optional

4.3 Data Formats

The standard allows you to use any of the above codes for data transmission between devices. It also provides other protocols to allow coding of almost any type of data. The Sections 4.3.1 and 4.3.2 on listener and talker protocols define the formats. Table 4.2 lists the formats defined by 488.2.

4.3.1 Device Listening Formats

The Listening Device must be very forgiving in what it receives from the talker. This is where the idea of *forgiving listening* and *precise talking* really shows up. The format of numbers received is very flexible. However, when we define the format for sending data it will be very rigid. This allows new devices to work with old devices and formats while ensuring uniformity in future devices.

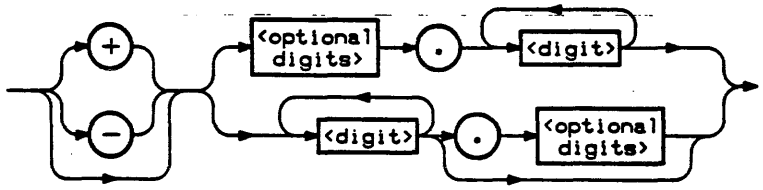
An example of this forgiving listening is uppercase lowercase equivalence. The listener must accept either uppercase or lowercase letters as equivalent.

Decimal Numeric Program Data

The first data format we'll examine is the <Decimal Numeric Program Data>, also indicated by <NRf> for flexible Numeric Representation. So what is the acceptable format for decimal data? Lets take a look. The following syntax diagrams show the valid format. Essentially, it accepts almost any format known to man. Okay, not quite all.

The <NRf> element is defined as

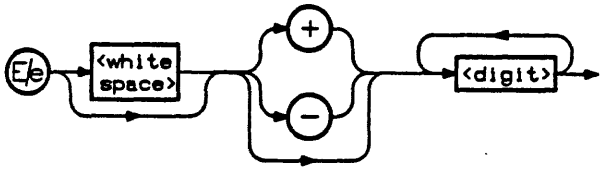
<mantissa> is defined as



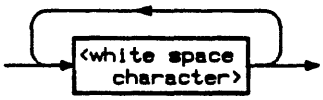
Where <digit> is defined as a single ASCII byte in the range of 48-57 decimal (ASCII 0 through 9).

<optional digits> are defined as

<exponent> is defined as



<white space> is defined as



And <white space character> is defined as a single ASCII character in the range 0 through 9 or 11 through 32 decimal. This includes the ASCII control characters null through space, but excludes the New Line or Line Feed character (10 decimal).

Here are some examples of valid formats.

```
.123
1.23
+12.3
.123 E -45
+12.3e+45
.123 E +1
+12.3e-1
-0.123
```

As you can see, this format is very forgiving.

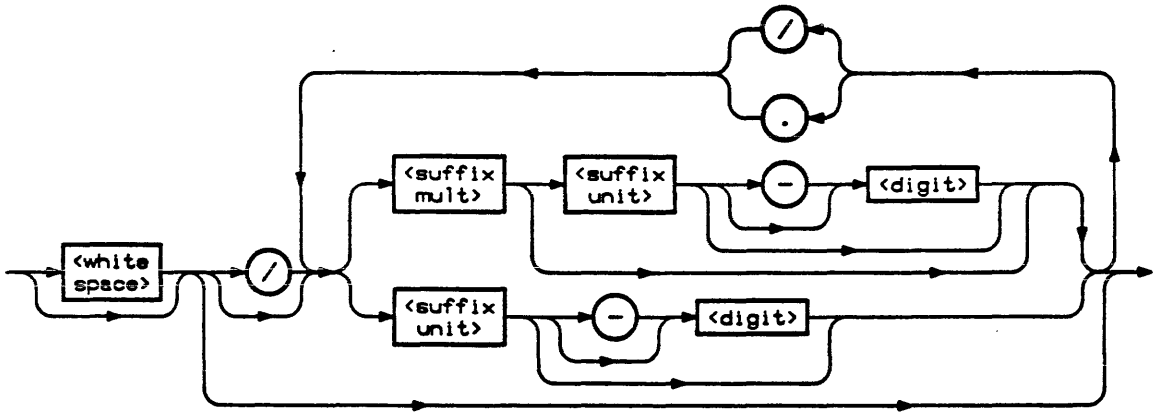
What rules apply to this format? The <mantissa> cannot have more than 255 characters excluding leading zeroes. The exponent must be in the range of -32000 to 32000.

If a device receives a <NRf> of a precision greater than it can handle the device must round the number rather than truncating it. When rounding, the device ignores the sign of the number and rounds up on values greater than or equal to one half. It rounds down on values less than one half. For example,

Input Value	Rounds To
1.3499	1.3
1.35	1.4
-2.458	-2.5
-2.447	-2.4

Suffixes are also allowed within this format. The suffix expresses the units and multipliers (optionally) that may be used to interpret the data sent.

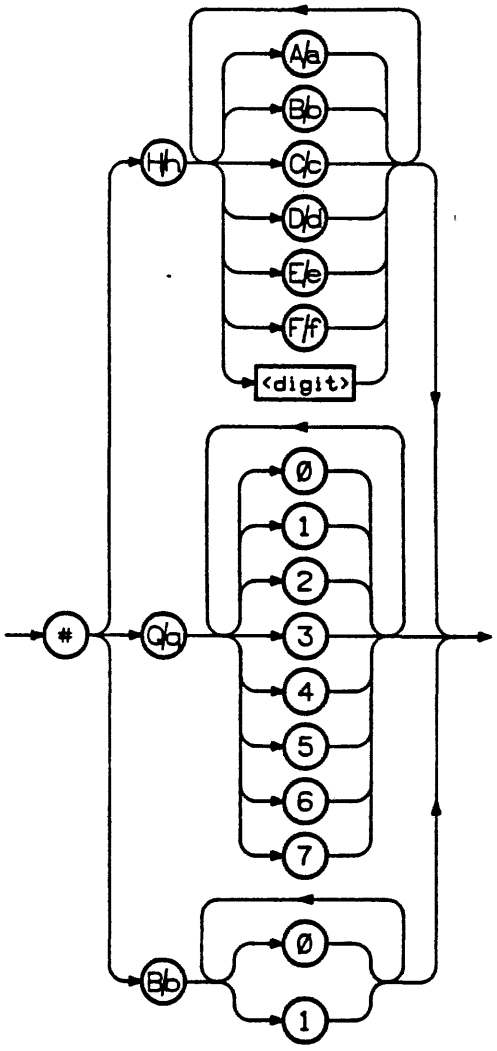
Syntax for the suffixes is defined as:



See Appendix A for a list of suffixes and multipliers.

Non-Decimal Numeric Program Data

The standard also defines a format for non-decimal data. It defines a format for hexadecimal, octal and binary numbers, using ASCII characters. The following syntax diagram shows the standard for these three data structures.

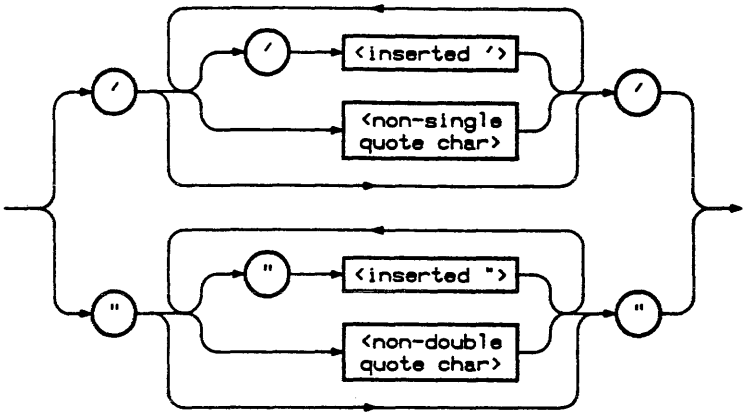


For example:

#HA2F
#ha3e
#hA3f
#Q73
#q54
#B01101
#b10010

String Program Data

This format allows any 7-bit ASCII character, including control codes and non-printing characters to be transmitted between devices. This is useful when transmitting display data, such as to a printer or CRT.



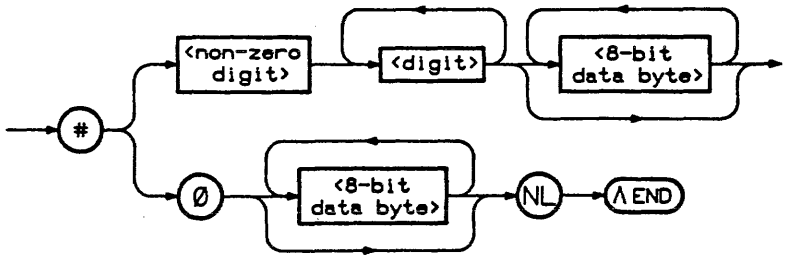
The <inserted '> and <inserted "> provide the means of including the quote character ('), or double quote (") in the data.

Examples:

```
'this is a string'
"this is also a string"
'this isn"t wrong'
"neither is "" this"
```

Arbitrary Block Program Data

This data element allows any 8-bit byte, including extended ASCII codes to be transmitted between devices. The first byte following the " #" contains a number. if this number is "0" (ASCII 48 decimal), then this is an arbitrary length block and is terminated with an END MESSAGE (See Chapter 5 for the definition of the END MESSAGE). If the number is something other than zero, then that number indicates the number of length digits that follow. These length digits are the number of data bytes that follow in the block.



For example four data bytes (DAB's) could be sent using:

1. #14<DAB><DAB><DAB><DAB>

2. #3004<DAB><DAB><DAB><DAB>

3. #0<DAB><DAB><DAB><DAB>NL^END

Expression Program Data

This data element evaluates to a scalar, vector, matrix, or string value. It allows the device to evaluate and manipulate the parameters sent to it.

The syntax for expressions is defined as:

where <expression> is a sequence of ASCII characters in the range 32 to 126 decimal except the ", #, ', (,), and ; characters. An expression may also be a device defined set of program data elements except arbitrary block.

For example:

(a+b-c)

Summary

So you can see that this new standard provides a variety of formats for coding data received by listening devices. As stated before, the listening device must be very forgiving of the format of data received.

4.3.2 Device Talking Formats

Continuing on with the concept of *forgiving listening, precise talking*, let's now look at device talking formats. You will note that the definitions for data sent from a device is much more restricted. This makes it easier for listening devices to decode the data.

Data Separators

Data elements in the same message are separated by commas (“,” ASCII 44 decimal). Data can be further separated into message units by using the semicolon (“;” ASCII 59 decimal).

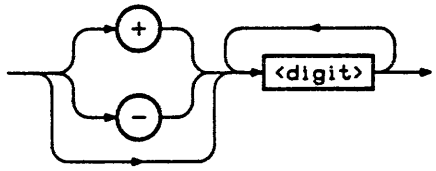
Numeric Formats

The 488.2 talking formats include characters, integers, floating point, hexadecimal, octal, and binary numbers, strings, and arbitrary blocks.

Let's look at the number formats.

NR1 Numeric Response Data — Integers

This first data format consists of integer numbers with an implied decimal point. The syntax diagram follows.

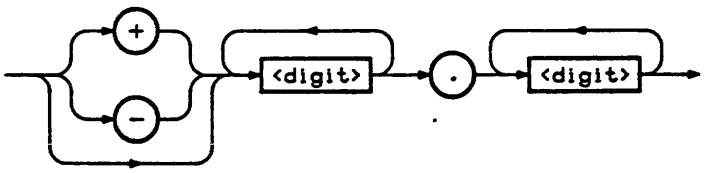


Examples include

123
+123
-12345

NR2 Numeric Response Data — Fixed point

This number format shows how to represent a floating point number with an explicit decimal point. This format does not have an exponent.

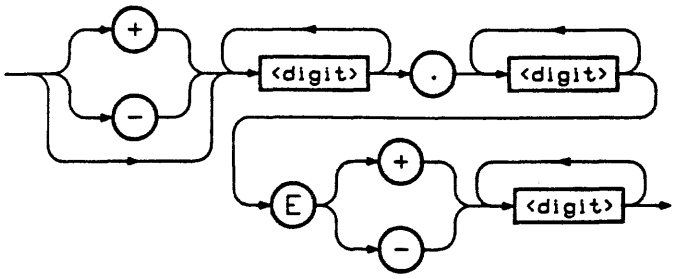


Examples

12.3
+1.234
-0.12345

NR3 Numeric Response Data — Floating Point

These numbers are floating point numbers with an explicit decimal point and an exponent.

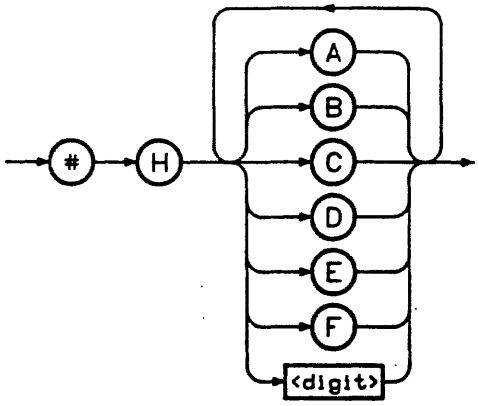


Examples

1.23E+5
123.4E-56
-12345.678E+90

Hexadecimal Numeric Response Data

The standard includes a data format for hexadecimal data. It uses the following format.



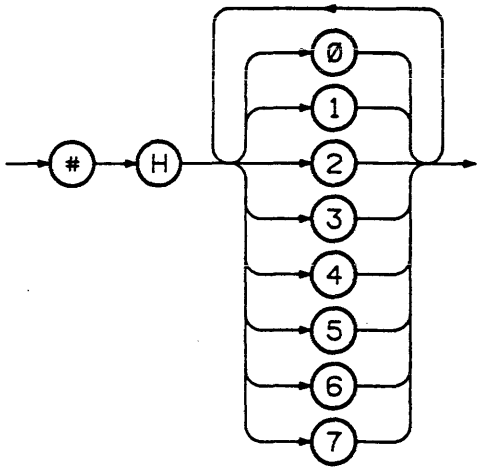
Examples include:

#HAD0E
#H01F2
#HF3B

This is the same format as the listening format except that lowercase characters are not allowed.

Octal Response Data

The standard defines the following format for octal data:



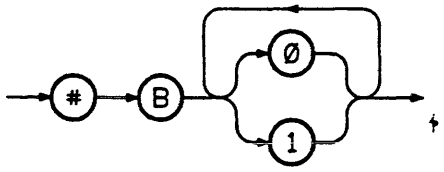
Examples:

```
#Q7035
#Q30572
#Q765432
```

This is the same format as the listener format except that lowercase is not allowed.

Binary Response Data

IEEE 488.2 defines the format for binary data as:



Examples:

```
#B011101
#B10101010
#B1011
```

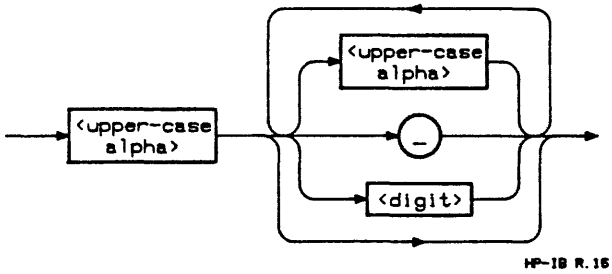
This is the same format as the listener format except that lowercase is not allowed.

Now that you've seen the data formats for numeric data let's look at the formats for characters and strings.

Character Response Data

When a numeric response is not suitable, the device may send a character response. This response is used to send mnemonics between devices. For example, the device may send characters to describe how to set it to its present state.

The format is defined as



HP-IB R.16

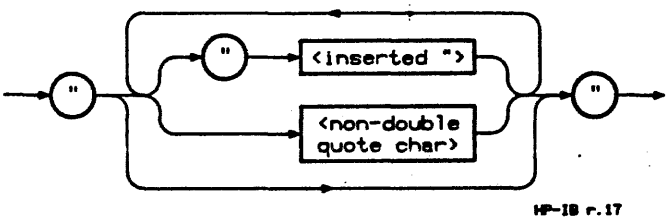
Where <uppercase alpha> is a single ASCII character in the range 65-90 decimal, or in other words, the standard uppercase alphabet. Note that this diagram shows that the response must start with an alpha character. It also shows that the response may include the underscore character (ASCII 95 decimal).

Examples:

```
START
R2_D2
```

String Response Data

In the case where a device responds with a string of characters it will use the following format.



HP-IB r.17

Examples:

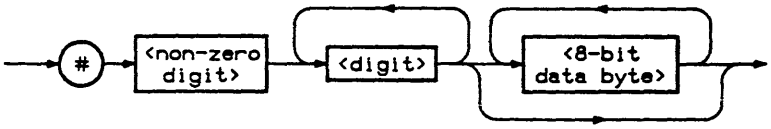
```
"This IS A valid string"
"SO IS "" THIS "
"SHE SAID ""HELLO""."
```

What if you need to send an arbitrary number of 8 bit data bytes? The 488.2 standard even provides a format for that.

Definite Length Arbitrary Block Response Data

This format allows a device to send any arbitrary device dependent data over the bus as 8-bit data bytes. The coding for this data should follow the recommendations for encoding binary data that we encountered earlier in this chapter.

The format follows.



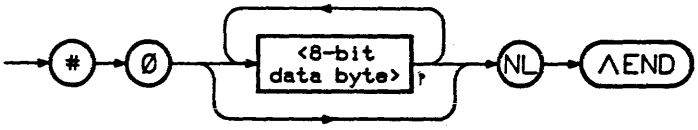
Where <non-zero digit> is a digit (not equal to 0) which indicates how many <digit>'s follow. The <digit>'s, taken as an integer indicate how many <8-bit data byte>'s follow.

Examples:

```
#12<DAB><DAB>
#3001<DAB>
```

Indefinite Length Arbitrary Block Response

This format allows the device to send data in blocks of indefinite lengths. This is useful where the length of the transmission is unknown, or where transmission speed or other conditions prevent dividing the output into known length blocks. The format follows.



Note that the NL^END Message Terminator is required to end the block of data. This message also serves as the Message Terminator for the entire message and must not be sent again.

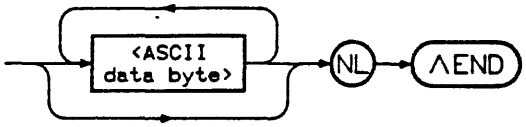
Examples:

```
#0<DAB><DAB>NL^END
#0<DAB>NL^END
```

Arbitrary ASCII Response Data

This format allows the device to respond with undelimited ASCII text. This is useful to send display text between devices.

The format follows.



The <ASCII data byte> is any ASCII encoded data byte except the NL character (10 decimal). The data is terminated by the Response Message Terminator, NL^END.

Examples:

<ASCII Byte><ASCII Byte>NL^END
NL^END

Chapter 5
Syntax

5.1 Overview

This chapter digs further into the nuts and bolts of the operation of a 488.2 device. Here we discover how all the data formats and commands learned about earlier get plugged together. As with the data formats, the principle of forgiving listening and precise talking applies.

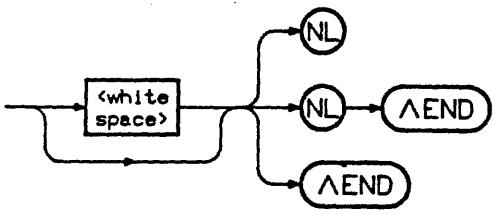
5.2 Listening Syntax

As you saw in the chapter on Data Coding and Formats, there are differences between what a device must send as data and what it will accept as data. The same is true for the syntax of listening and talking.

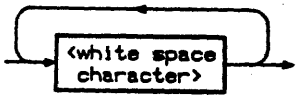
The first thing that we need to know is how to end a message.

5.2.1 Terminators

There are several ways to terminate a message to a Listening device. Basically, this allows older controllers and devices to continue to work with devices that use this newer standard. The <PROGRAM MESSAGE TERMINATOR> is defined as



Where <white space> is defined as:



And <white space character> is defined as a single ASCII character in the range 0 through 9 or 11 through 32 decimal. This includes the ASCII control characters null through space, but excludes the New Line or Line Feed character (10 decimal). In general, devices ignore white space. It is used to make the commands more humanly readable.

The ∧END indicates that EOI is asserted with the last byte sent.

5.2.2 Separators

The separators include the Program Message Separator, the Program Header Separator, and the Program Data Separator. As their names imply, each has a different function.

Program Message Separator

The Program Message Separator is defined as:

This separator is placed between commands in a single message to create a complex command.

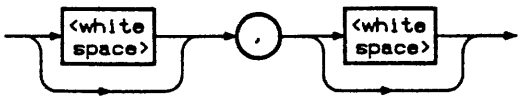
Program Header Separator

The Program Header Separator is defined as:

The header separator is placed between commands and their parameters. Also, this is the only place where <white space> is significant. It indicates the end of the command or Program Header, and the beginning of the data.

Program Data Separator

The Program Data Separator is defined as:

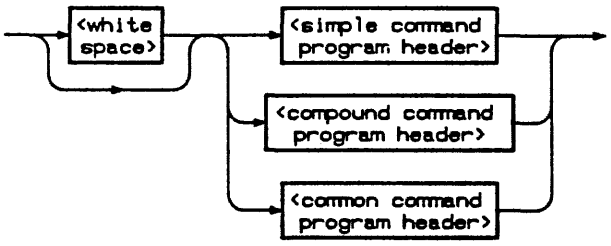


This definition shows that a comma will separate data. There may also be some white space sprinkled in.

Well, now that you know how to end messages, and separate them, let's see how to start one.

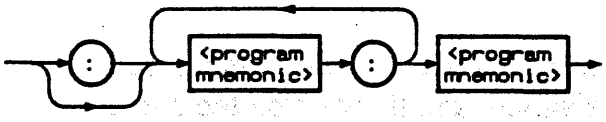
5.2.3 Commands

The <COMMAND PROGRAM HEADER> is what really starts getting the work done in controlling a device. In its simplest form, it is the device command. It is defined as

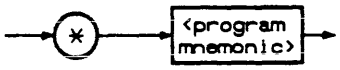


Where <simple program header> is a <program mnemonic> or in other words, a device command mnemonic.

A <compound command program header> is defined as



A <common command program header> is defined as

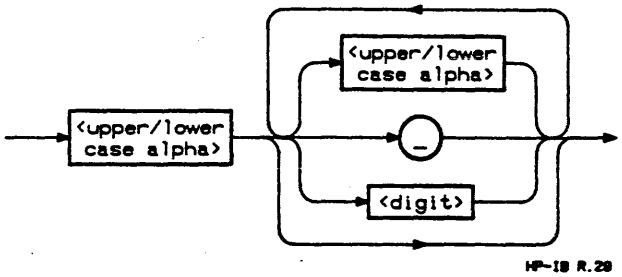


HP-IB R.28

These are the IEEE 488.2 commands discussed in Chapter 7, Common Commands.

Program Mnemonic

So what can a <program mnemonic> contain? It is defined as



HP-IB R.29

Where <upper/lower case alpha> is a single ASCII byte in the range 65-90 or 97-122 decimal. These are the uppercase and lowercase characters of the alphabet.

As before, a <digit> is defined as an ASCII byte in the range 48 - 57 decimal. These are the ASCII numbers 0 through 9.

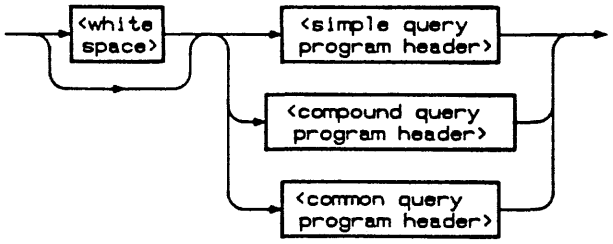
The "_" represents the "underscore", ASCII byte 95 decimal.

These mnemonics can have a maximum of 12 characters. But for brevity, a length of 4 is preferred.

5.2.4 Queries

The Queries tell the device to respond with information. You will note that the Query syntax is identical to the command syntax with a "?" appended.

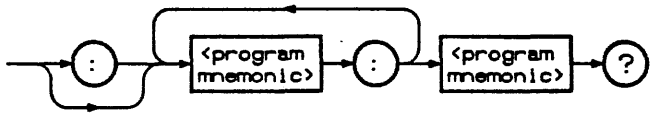
A <QUERY PROGRAM HEADER> is defined as



A <simple query program header> is defined as

As you can see, a query is simply a program mnemonic that ends in a "?".

As with the <COMMAND PROGRAM HEADER>, queries can be separated with a ":" to create a <compound query program header>.



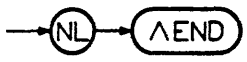
The <common query program header> is preceeded by an "*".

5.3 Talking Syntax

You will find that the Talking Syntax is very similar to the Listening Syntax of the previous section. However, as with the Data Formats, it is much more precise.

5.3.1 Terminators

A <RESPONSE MESSAGE TERMINATOR> is defined as



This means that the EOI line is asserted while the New Line or Line Feed character (10 decimal) is being sent on the bus.

Now you may ask, "What is a response message?" This is the message that a device sends in response to a query. Since some devices accept compound (multiple) queries in one message, the response may also contain several pieces of data. Another important thing to remember is that the device will not send any more data after the <RESPONSE MESSAGE TERMINATOR> until it receives another query.

5.3.2 Separators

As with the listening syntax, there are several separators used in the talking syntax.

Message Separator

The <RESPONSE MESSAGE UNIT SEPARATOR> separates response messages that are sent as a single response. It is defined as the ";" character. As mentioned before, responses may contain several messages in response to multiple queries. The device must use a ";" to separate these messages units.

Data Separator

The <RESPONSE DATA SEPARATOR> separates sequential data elements from each other, since a device can send multiple pieces of data in the same message unit. It is defined as the "," character. In some cases, the response to a query contains several pieces of data. The device sends a comma between each of these items.

Header Separator

The <RESPONSE HEADER SEPARATOR> separates the response header from the response data. It is defined as the " " (space) character.

5.3.3 Response Data versus Learn String

There is one last thing to discuss before leaving syntax. Devices can respond with two different types of data, Response Data or a "Learn String". The Response Data will contain only the data required to respond to a query. For example, the response to a status query would be an integer containing the status byte. The second type of response, the Learn String, contains a command header as well as the the data. The command header will contain the command necessary to set the device parameters to the it's present state. For example if you queried a voltmeter for its range setting, it would send back a string that contained the command necessary to set the voltmeter to the range that it is presently set to.

Chapter 6
Status Reporting

6.1 Overview

This chapter describes the Device Status Reporting model defined by the IEEE 488.2 standard. This model builds upon and extends the specifications of the original IEEE 488 standard. It provides a method to transfer the status byte to the controller using either the IEEE 488.1 Serial Poll or an IEEE 488.2-defined common query. In addition it defines more common commands and queries to obtain additional information. Figure 6.1 shows an overview of the status reporting structure.

The sections of this chapter expand on the information shown in the figure. They explain the operation of the Status Byte, request enabling, standard status data structures and parallel polling.

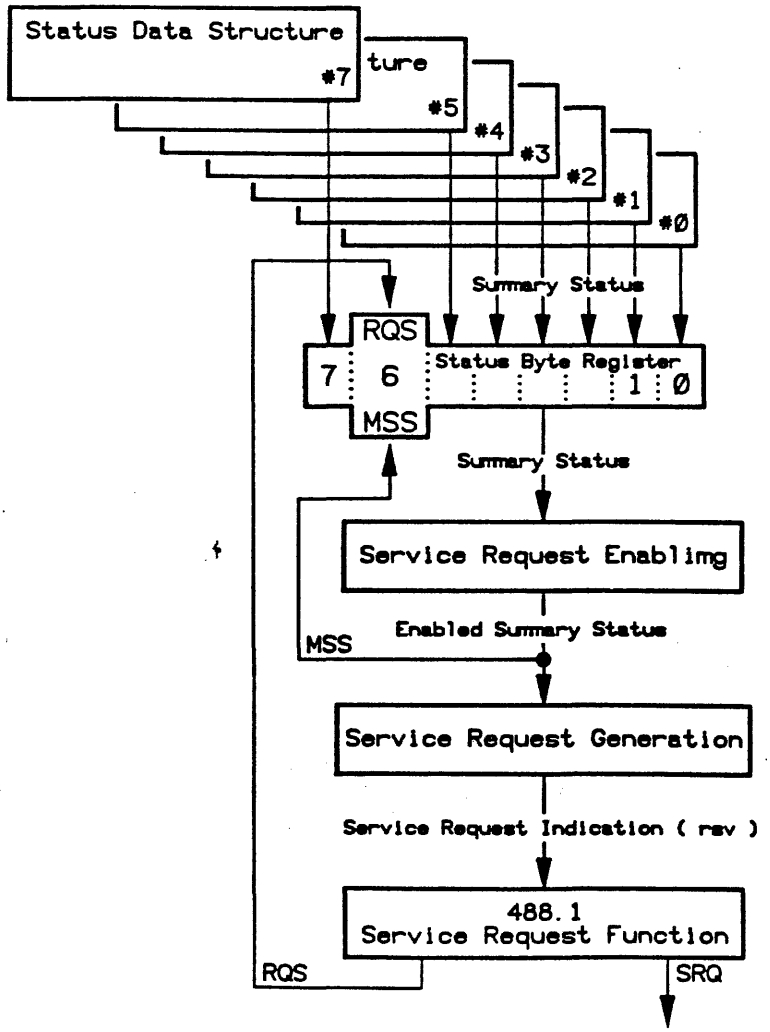


Figure 6.1 IEEE 488.2 Status Reporting Structure Overview

6.2 Status Byte Register

IEEE 488 originally defined the Status Byte and provided the Serial Poll to allow controllers to read it. However, other than the RQS bit, it doesn't define how the bits are set or cleared. It also left the definition of the bits contained in the Status Byte completely up to the device designer.

IEEE 488.2 further defines the Status Byte. Figure 6.2 shows that this standard defines meanings for bits 4, and 5. IEEE 488.1 defined bit 6. In addition, 488.2 defines more commands that allow the user to access the Status Byte and associated data structures. It's important to note that the Serial Poll *DOES NOT* clear the Status Byte, even though it does clear the RQS bit. The byte is cleared by clearing the related status structures. IEEE 488.2 provides a clear command (*CLS) which clears all of the Status Data Structures, that is all of the Event Registers and Queues. This causes the bits in the Status Byte to be cleared. The sections that follow give greater detail on these other structures and how they work.

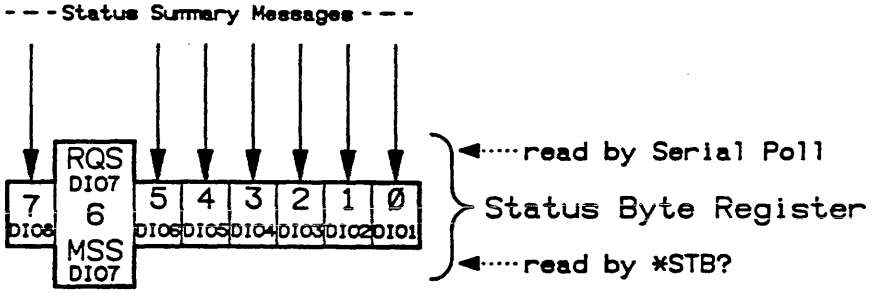


Figure 6.2 Status Byte Register

Let's look at the bits defined in the Status Byte.

6.2.1 Event Status Bit

IEEE 488.2 defines the Event Status Bit (ESB) to be bit 5 of the Status byte. Its state indicates whether or not an enabled standard event has occurred Section 6.6 details what events can be monitored.

6.2.2 Message Available Bit

IEEE 488.2 defines bit 4 of the Status Byte to be the Message Available Bit (MAV). This bit indicates whether or not the Output Queue is empty. Whenever the device has data available to output this bit will be TRUE.

6.2.3 Master Summary Status Bit

The Master Summary Status Bit (MSS) indicates whether or not the device has at least one reason to request service. Even though the device sends the MSS bit in bit 6 of the status query response, it is *NOT* sent in response to the serial poll. It is not considered part of the IEEE 488.1 Status Byte.

The IEEE 488.1 Serial Poll tells the device to send the Status Byte. Bit 6 will contain the Request Service (RQS) bit. The device will then clear the RQS bit if it was set.

6.3 Enabling Service Request

The service request enabling operation is shown in Figure 6.3. The user can set bits in the Service Request Enable Register (SRER). These bits correspond to bits in the Status Byte. When a bit is set in the SRER it enables that bit in the Status Byte to request service. For example, setting bit 4 in the SRER will cause the device to request service whenever the device has data in the Output Queue.

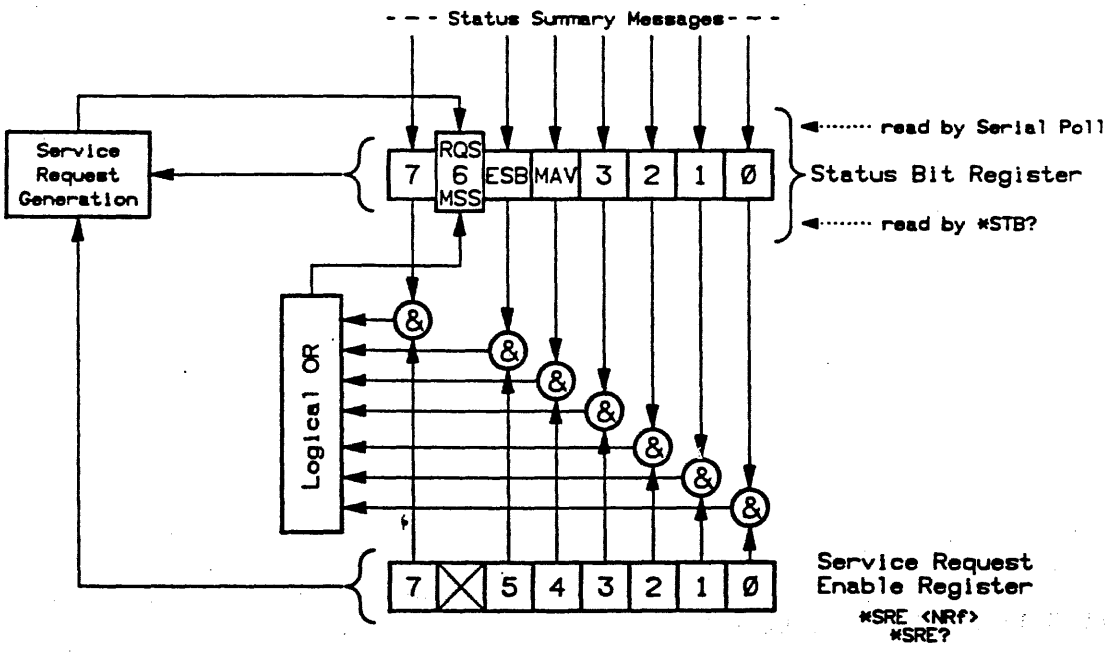


Figure 6.3 Service Request Enabling

6.4 Event Registers

Event Registers capture changes that occur within a device. Each bit in an event register corresponds to some device condition. These bits become TRUE when some pre-defined device condition change occurs. These changes are sometimes called transitions. The event registers guarantee that the user can't miss this change because these bits are "sticky". That is, once they become TRUE they cannot be cleared except by the user. There are two means of clearing an event register. Reading the register will clear it. Also, the Clear Command (*CLS) will clear all event registers.

IEEE 488.2 defines three transition criteria for setting these event bits TRUE.

1. Positive Transition. The event becomes TRUE when its condition makes a FALSE to TRUE transition.

2. Negative Transition. The event becomes TRUE when its condition makes a TRUE to FALSE transition.

3. Positive or Negative Transition. The event becomes TRUE when its condition makes either a FALSE to TRUE or a TRUE to FALSE transition.

Devices may have more than one Event Status Register. IEEE 488.2 only defines a command to read the Standard Event Status Register. So if a device has more event registers the device will provide other device dependent commands to read them. See Section 6.6 for more information on the Standard Event Status Register.

A device may also provide Event Enable Registers. These registers work in the same way as the Service Request Enable Register described in Section 6.2. Briefly, setting bits in the enable register allows bits in the event register to be summarized in the Status Byte.

6.5 Queues

Queues permit the device to report status or other information in a sequential manner. For example, the device could report error messages in the order that they occurred. Each queue will have a summary message bit that indicates that the queue contains some information. This bit will be TRUE when the queue contains any information. Otherwise, it will be FALSE.

One example of the queue data structure is the Output Queue. Its status is summarized in the MAV bit. See Section 6.7 for further details on its operation.

Each device must define some device dependent command to read from any queue other than the Output Queue. Reading the queue will remove that piece of information from the queue. The queue is considered empty when it no longer contains any information.

You can clear a queue by reading all of the information in the queue. You can also clear all of the status queues (except the Output Queue) using the Clear Command (*CLS). A device may also have a unique device dependent command to clear its queues.

6.6 Standard Event Status Register

This section begins a description of status structures that must exist in all devices. So far, we have only discussed generalized models of how status reporting works. Here we begin digging into the details of operation again.

Figure 6.4 represents the operation of the Standard Event Status Register (SESR). This is a specific application of the event registers discussed previously. IEEE 488.2 specifies the meaning of each bit in the SESR. Lets look at these definitions.

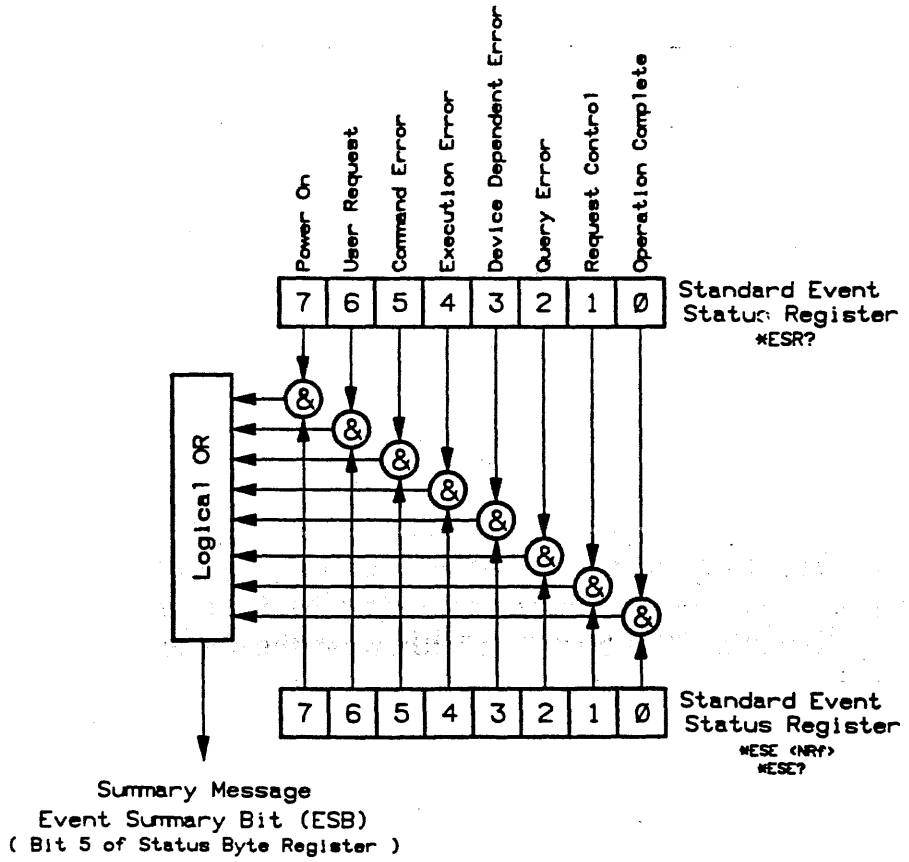


Figure 6.4 IEEE 488.2 Standard Event Status Register

6.6.1 SESR Bit Definitions

Bit 7 — Power On (PON)

This bit indicates that the device's power supply has been turned off and then on since the last time this register was read.

Bit 6 — User Request (URQ)

This bit indicates that the user has activated some device defined control. This bit will be set regardless of the Remote Local state of the device. This provides the user with a means of getting the controller's attention.

Bit 5 — Command Error (CME)

This bit indicates that the device has detected a command error. The following events cause a command error.

1. An IEEE 488.2 syntax error. This means that the device received a message that did not follow the syntax defined by the 488.2 standard. For example, it received data that violated the device listening format.

2. A semantic error occurred. For example, the device received an incorrectly spelled command. Another example would be that the device received an optional 488.2 command that it does not implement.

3. The device received a Group Execute Trigger (GET) inside a program message.

Bit 4 — Execution Error (EXE)

This bit indicates that the device detected an error while trying to execute a command. This bit indicates that:

1. a <PROGRAM DATA> element received in a command was outside the legal range for the device, or inconsistent with the operation of the device.

2. the device could not execute a valid command due to some device condition.

Bit 3 — Device-dependent Error (DDE)

A device-dependent error is any device operation that did not execute properly due to some internal condition such as overrange. This bit indicates that the error was not a command, query, or an execution error.

Bit 2 — Query Error (QYE)

This bit indicates:

1. an attempt to read data from the Output Queue when no data was present.
2. that data in the Output Queue was lost. An example of this would be queue overflow.

Bit 1 — Request Control (RQC)

This bit indicates to the controller that the device wants to become the active controller-in-charge.

Bit 0 — Operation Complete (OPC)

This bit indicates that the device has completed any pending operations and is ready to accept new commands. This bit is generated only in response to the Operation Complete (*OPC) command.

6.6.2 Standard Event Status Register Operation

The Standard Event Status Register (SESR) operates in the same manner as the Event Registers described in Section 6.4. All 488.2 devices have the SESR register. Other event registers are optional.

The Standard Event Status Enable Register is written with the Enable Status (*ESE) command and read with the Enable Status (*ESE?) query.

The SESR can only be cleared by:

1. a Clear Command (*CLS).
2. reading the Standard Event Status Register (*ESR?).
3. a power-on transition (at the discretion of the device designer). Note that in this case, the device will clear the register and then record any transitions that occur, including setting the Power On (PON) bit.

6.7 Output Queue

The Output Queue is a "first-in, first-out" (FIFO) queue. It stores output messages until they are read from the device. The availability of data is summarized in the MAV bit of the Status Byte. You read the Output Queue by addressing the device to talk and then handshaking the bytes.

The Clear Command does not clear the Output Queue. It can only be cleared by the Reset Command, the Device Clear Command (488.1) or by power on. That way, you have less chance of losing data.

6.8 Parallel Poll

IEEE 488.1 defined Parallel Polling as a fast means of obtaining status from a device or multiple devices on the bus. This section briefly describes the means provided optionally in 488.2 of generating and controlling a device response to a Parallel Poll. Figure 6.5 shows the Parallel Poll Data Structure. You can see that this structure is the same as the Event Register discussed above. However, instead of being summarized in a bit in the Status Byte, the summary bit is sent in response to a Parallel Poll. This summary bit is the *ist* or *individual status* local message.

As with the Event Registers there is an Enable Register to determine which events are summarized in the *ist*. IEEE 488.2 defines an optional command to write the enable register and a query to read it. It also defines the query to read the *ist* without doing a parallel poll.

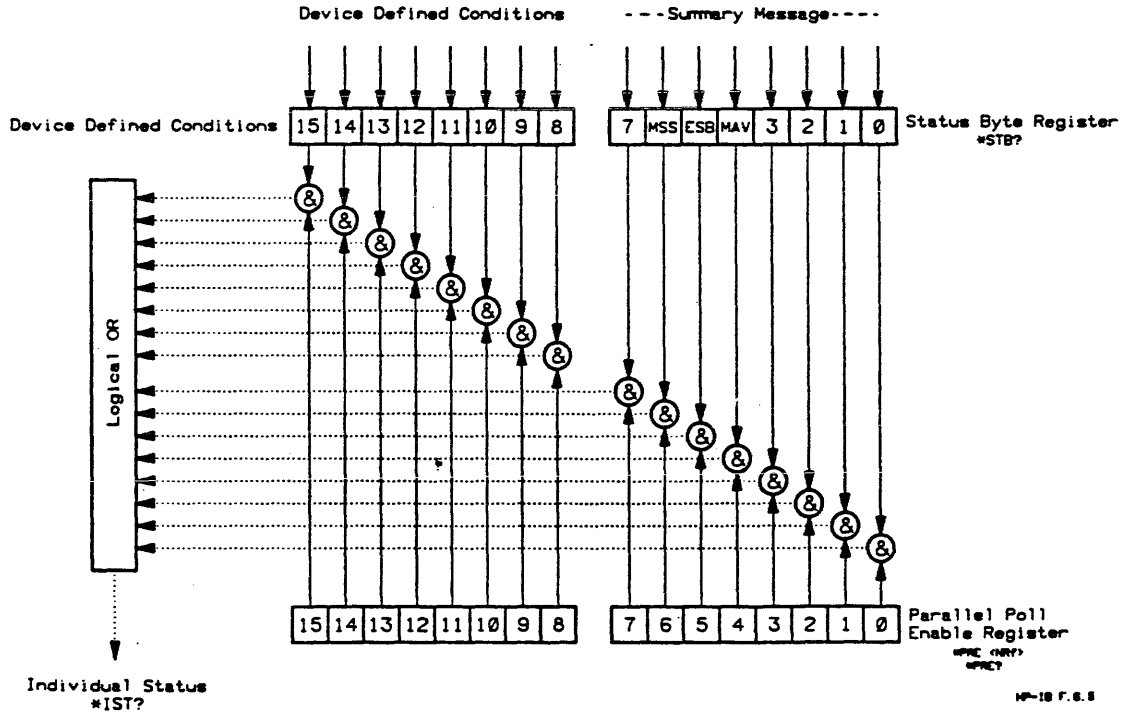


Figure 6.5 Parallel Poll Response Handling Data Structure

Chapter 7
Common Commands

Overview

The IEEE 488.2 Standard designates a set of commands that each device must have. Note that these commands are sent in the DATA Mode (ATN False). These are not new bus commands, but are new device commands, common to all devices. Table 7.1 lists commands by command group.

The requirement of some common commands guarantees that all devices will have a minimum set of capabilities. This permits test system programmers to write code modules that will work with all devices.

The following section contains a more complete description of each of the common commands defined by IEEE 488.2

7.2 Command Descriptions

This section gives a description of each command group and the indivdual commands within each group. This is by no means an exhaustive description but should still give you a good idea how each command functions.

Table 7.1 Common Command Groups

Mnemonic	Description	Compliance
	AUTO CONFIGURE COMMANDS	
*AAD	Assign Address	Opt.
*DLF	Disable Listener Function	Opt.
	SYSTEM DATA COMMANDS	
*IDN?	Identification Query	Reqd.
*OPT?	Option Identification Query	Opt.
*PUD	Protected User Data	Opt.
*PUD?	Protected User Data Query	Opt.
*RDT	Resource Description Transfer	Opt.
*RDT?	Resource Description Transfer Query	Opt.
	INTERNAL OPERATION COMMANDS	
*CAL	Calibration Query	Opt.
*LRN	Learn Device Setup Query	Opt.
*RST	Reset	Reqd.
*TST?	Self-Test Query	Reqd.
	SYNCHRONIZATION COMMANDS	
*OPC	Operation Complete	Reqd.
*OPC	Operation Complete Query	Reqd.
*WAI	Wait to Complete	Reqd.
	MACRO COMMANDS	
*DMC	Define Macro	Opt.
*EMC	Enable Macro	Opt.
*EMC?	Enable Macro Query	Opt.
*GMC?	Get Macro Contents Query	Opt.
*LMC?	Learn Macro Query	Opt.
*PMC	Purge Macros	Opt.
	PARALLEL POLL COMMANDS	
*IST?	Individual Status Query	Reqd. if PP1
*PRE	Parallel Poll Enable Register Enable	Reqd. if PP1
*PRE?	Parallel Poll Enable Reg Enable Query	Reqd. if PP1

Table 7.1 Common Command Groups (Cont'd)

Mnemonic	Description	Compliance
	STATUS & EVENT COMMANDS	
*CLS	Clear Status	Reqd.
*ESE	Event Status Enable	Reqd.
*ESE?	Event Status Enable Query	Reqd.
*ESR?	Event Status Register Query	Reqd.
*PSC	Power on Status Clear	Opt.
*PSC?	Power on Status Clear Query	Opt.
*SRE	Service Request Enable	Reqd.
*SRE?	Service Request Enable Query	Reqd.
*STB?	Read Status Byte Query	Reqd.
	DEVICE TRIGGER COMMANDS	
*DDT	Define Device Trigger	Opt. if DT1
*DDT?	Define Device Trigger Query	Opt. if DT1
*TRG	Trigger	Reqd. if DT1
	CONTROLLER COMMANDS	
*PCB	Pass Control Back	Reqd. if Controller
	STORED SETTINGS COMMANDS	
*RCL	Recall Instrument State	Opt.
*SAV	Save Instrument State	Opt.

7.2.1 Auto-Configure Commands

IEEE 488.2 defines an algorithm to automatically assign device addresses to devices. This simplifies the job of the person who physically puts the system together. The different devices need only be connected together. the software can then query the devices to find out what they are and then make address assignments accordingly. A complete description of this algorithm is beyond the scope of this book. Remember, however, that this capability is optional and may not be implemented in all devices.

***AAD**

Accept Address Command

This command, used with the Address Set protocol mentioned above, permits the controller to detect all address configurable devices, and assign them a 488.1 bus address. Using this command, the controller searches the device identifiers. Using this information it then assigns 488.1 bus addresses to these devices using the *AAD Command.

*DLF

Disable Listener Function Command

This commands tells a device to stop listening on the bus. The device can no longer receive any data until it receives a Device Clear (DCL) command. This command is used with the *AAD command to perform the Automatic System Configuration.

7.2.2 Controller Commands

There is only one command in the Controller command group. This command provides an orderly means to pass control within a system.

*PCB

Pass Control Back

The Pass Control Back Command tells a potential controller what address to pass control back to. Using this command, a controller can tell other devices which can be controllers where to send the 488.1 Take Control (TCT) command when they are ready to give up control of the bus. The command is followed by a number, which when rounded, contains the bus address of the device that should become the next controller. The number must round to an integer value in the range 0 to 30 decimal. The command may be followed by two numbers. The first will be used as the primary address, the second as the secondary address of the new controller.

7.2.3 Device Trigger Commands

The Device Trigger Commands give the user the option of controlling exactly how a device responds to a bus trigger command (GET). It also permits the "forgetful" user to read what the device is going to do when it receives the GET. These commands are optional.

*DDT

Define Device Trigger Command

This command stores a sequence of device commands which the device will execute when it receives a Group Execute Trigger (GET) IEEE 488.1 interface message or a *TRG common command. The command sequence sent to the device may contain an arbitrary block of program elements or a zero length data block. If it contains a zero length data block the device will do nothing when it receives a GET or *TRG command.

***DDT?**

Define Device Trigger Query

This command allows the user to examine the command sequence that the device will execute when it receives the Group Execute Trigger (GET) or *TRG command. The device will respond with a <Definite Length Arbitrary Block Response Data> containing the commands it will exectute when it receives a trigger command.

***TRG**

Trigger Command

The Trigger Command performs the same function as the Group Exectute Trigger command defined by IEEE 488.1.

7.2.4 Internal Operation Commands

These commands control the internal operation of the device. This group provides commands to calibrate, test and reset the device. It also provides an optional command to read the internal settings of the device.

***CAL?**

Calibration Query

This command tells the device to perform an internal self-calibration. The device responds to indicate whether or not it was succesful in performing this calibration. The response may also contain information about any calibration errors that occurred.

This operation of self calibration can not require any local user interaction to function properly. It must all be carried out completely by the device. It may not cause any conditions that will violate the 488.1 or 488.2 standards.

When the device completes the calibration sequence it will return to the state it was in previously or to a state designated in the device documentation.

The device response to this query will be an <NR1> (integer) in the range −32767 to 32767. A value of 0 indicates that the calibration executed successfully.

*LRN?

Learn Device Setup Query

The Learn Device Setup Query tells the device to send a response that contains all the necessary commands to set the device to its present state. The response may later be sent back to the device to place it in this state. This provides the user with a means of setting up a device manually and then reading the device setting and storing that information for later use.

The response may not be in a "human-readable" form. It may be in ASCII or in binary. However, sending this string back to the device will set it back to that state.

*RST

Reset

This command resets the device.

The Reset command:

1. Sets the device-dependent functions to a known state, independent of its current state.
2. Sets the Device Defined Trigger (see *DDT command) to a device-defined state.
3. Disables macros
4. Aborts all pending operations
5. Forces the device to forget about any previously recieved *OPC commands
6. Forces the device to forget about any previously received *OPC? queries

The Reset command does NOT affect:

1. The state of the IEEE 488.1 interface
2. The IEEE 488.1 address
3. The Output Queue
4. The Service Request Enable Register
5. The Standard Event Status Enable Register
6. The power-on flag

7. Macros (except to disable them)

8. Calibration data

9. The Protected User Data

10. The Resource Description Transfer Query Response

*TST?

Self-Test Query

This command causes the device to execute an internal self-test and report whether or not it detected any errors. The device may also (optionally) indicate why it failed the self-test. The device document will describe what the self-test checks.

The response sent by the device will be in <NR1> format in the range −32767 to 32767. A zero response indicates that the test completed without detecting any errors. Values other than zero will be described in the device documentation.

7.2.5 Macro Commands

IEEE 488.2 defines a set of optional commands to implement macros in a device. Macros provide real power to the experienced system designer. Macros can be used to:

1. provide shorthand for complex commands.

2. cut down bus traffic.

3. emulate other instruments.

*DMC

Define Macro Command

This command allows the user to assign a sequence of commands to a macro label. The device executes the macro when it receives the macro label as a command.

The user defines a macro by sending the *DMC command, followed by a string designating the label. Following the label, the user sends an <Arbitrary Block Program Data > element defining the macro.

For example

```
*DMC "HOME",#18MOVE 0,0
```

defines a command that sends a pen plotter to its home position.

Macro definitions also allow the user to pass parameters with the macro. Placeholders for parameters appear as a dollar sign (ASCII $, 36 decimal) followed by a single digit in the range 1 to 9 (49-57 decimal). For example

```
*DMC "TEST_A",#221BEGFREQ $1;ENDFREQ $2
```

defines a macro with two parameters. Sending

```
TEST_A 1000,2000
```

would be equivalent to sending

```
BEGFREQ 1000; ENDFREQ 2000
```

to the device. Parameters may also be re-used within a macro. For example

```
*DMC "SWEEP_SET",#234START $1;MARK1 $1;STOP $2;MARK2 $2
```

defines a macro which reuses two parameters. Sending

```
SWEEP_SET 1E6,5E6
```

would be equivalent to sending

```
START 1E6;MARK1 1E6;STOP 5E6;MARK2 5E6
```

The macro label may be either a command or a query. The label cannot be the same as a common command or common query. It may be the same as a device dependent command. When a macro label is the same as a device dependent command, the device will execute the macro rather than the device command if macros are enabled.

*EMC

Enable Macro Command

This command enables and disables the expansion of macros by a device. However, it does not affect the macro definitions. An example of the use of this command is to turn off macros in order to use a device dependent command which has the same name as a macro. Sending this command followed by an <NRf> of 0 will disable all macros. Sending it with a number other than 0 in the range −32767 to 32767 will enable macros. Of course the standard rounding rules for <NRf> apply to the numbers sent as parameters.

For example, sending

*EMC 0.4

will disable macros. Sending

*EMC -12.4

will enable macros.

*EMC?

Enable Macro Query

The Enable Macro Query allows the user to determine whether or not macros are enabled on the device. The device will return a value of 1 (ASCII 49 decimal) when macros are enabled. It will return a value of 0 (ASCII 48 decimal) when macros are disabled.

*GMC?

Get Macro Contents Query

The Get Macro Contents Query allows the user to obtain the current definition of a macro from a device. The user sends the *GMC? query followed by the label string of the macro. The device responds with a <Definite Length Arbitrary Block Response Data> element which contains the macro definition.

For example, sending

*GMC? "SWEEP_SET"

to a device will tell it to send the macro definition for the macro "SWEEP_SET".

*LMC?

Learn Macro Query

The Learn Macro Query instructs the device to respond with the labels of all the currently defined macros. The device will respond with strings separated by commas. If no macros are defined the device will return a null string of two consecutive double quote (") marks. The response is the same whether or not macros are enabled or disabled.

*PMC

Purge Macros Command

The Purge Macros Command causes a device to delete all macros in memory that were defined by the *DMC command. All macro sequences and labels are removed from memory.

7.2.6 Parallel Poll Commands

IEEE 488.2 defines an optional set of commands to implement the Parallel Poll functions in a device. These commands write and read the Enable Registers, and read the *ist* (Individual Status bit).

*IST?

Individual Status Query

The Individual Status Query permits the user to read the current state of the *ist* local message of a device. The device's *ist* is the individual status that a device sends when it is Parallel Polled. The state of this bit is determined by the Parallel Poll Enable Register and corresponding status information. The device will return an <NR1>. It will be either a 0 (48 decimal) if FALSE or a 1 (49 decimal) if TRUE. In otherwords, this command allows the user to read what a individual device will send on a Parallel Poll, without performing the Parallel Poll.

*PRE

Parallel Poll Enable Register Enable Command

The Parallel Poll Enable Register Enable Command sets the bits in the Parallel Poll Enable Register. These are the bits which determine what device conditions are summarized in the *ist*. The command is followed by a <Decimal Numeric Program Data> element. The value of this number must round to the range 0 to 65535. This number, when converted to binary format represents the bits set in the Register.

*PRE?

Parallel Poll Enable Register Enable Query

The Parallel Poll Enable Register Enable Query reads the contents of the Parallel Poll Enable Register. The device responds with an integer in the range 0 to 65535.

7.2.7 Status & Event Commands

The Status & Event Commands provide a means to read and enable events within the device. These commands include reading and writing to status structures within the device, power-on events and reading the status byte.

*CLS

Clear Status Command

The Clear Status Command clears the status register and associated status data structures summarized in the Status Byte, such as the Event Status Register. It also clears all status related queues except the Output Queue.

*ESE

Standard Event Status Enable Command

The Standard Event Status Enable Command sets the Standard Event Status Enable Register bits. The data is defined as <Decimal Numeric Program Data>. The device will round this number to an integer as described in the Device Listening Formats section of this chapter. Expressing this number in base 2 (binary) would then represent the values of the individual bits of the Standard Event Status Enable Register.

For example to set bit 5 (Command Error) and bit 2 (Query Error) the command

*ESE 36

would be sent to the device. The number sent to the device must be in the range 0 to 255 or an Execution Error occurs.

*ESE?

Event Status Enable Query

This command reads the contents of the Standard Event Status Enable Register (SESER). In response to this query the device sends the contents of the SESER in integer format. It will be in the range 0 to 255.

*ESR?

Event Status Register Query

This command reads the contents of the Standard Event Status Register. Reading this register clears it. It returns an integer, which when converted to a binary number represents the contents of the individual bits of the register. This number will be in the range 0 to 255 decimal.

*PSC

Power-on Status Clear Command

This command controls the automatic power-on clearing of the Service Request Enable Register, the Standard Event Status Enable Register, and the Parallel Poll Enable Register. Setting the Power-on-clear Flag TRUE causes the registers to be cleared at power-on, thus preventing the device from requesting service. Sending a <Decimal Numeric Program Data> element that rounds to the integer value 0 makes the flag FALSE and thereby potentially allows the device to request service at power-on. Sending any value other than 0, in the range −32767 to 32767 sets the flag TRUE.

For example the sequence

```
*PSC 0;*SRE 32;*ESE 128
```

allows a device to request service at power-on.

*PSC?

Power-on Status Clear Query

The Power-on Status Clear Query reads the status of the power-on-clear flag. A value of 0 indicates that the flag is FALSE. A value of 1 indicates that the flag is TRUE.

*SRE

Service Request Enable

This command sets the Service Request Enable Register. This register determines what bits in the Status Byte will cause a service request from the device. The data sent with the command is a <Decimal Numeric Program Data>. The device rounds this number to an integer. Expressing this number in base 2 (binary) would then represent the values of the individual bits of the Service Request Enable Register.

For example, to set bit 4 (Message Available) the command

***SRE 16**

would be sent. The device would then cause a service request when data is ready.

*SRE?

Service Request Enable Query

This command reads the contents of the Service Request Enable Register. The device returns the data as an <NR1> (integer), in the range 0 to 63 or 128 to 191, since bit 6 (the RQS bit) cannot be set.

*STB?

Status Byte Query

This command reads the status byte with the Master Summary Status (MSS) bit. The device responds with an integer in the range 0 to 255. These bits represent the contents of the status byte. Bit 6 represents MSS rather than RQS (Request Service).

7.2.8 Stored Settings Commands

The Stored Settings Commands permit the user to read and write the internal settings of the device.

*RCL

Recall Command

The Recall Command restores the state of a device from a copy previously stored in local (to the device) memory. The device may have multiple storage registers, so the command includes a numeric parameter to indicate which storage register to use. These numbers will begin at zero and end at an upperbound determined by the device. The state restored by the *RCL command are the same functions affected by *RST command.

*SAV

Save Command

The Save Command stores the present state of the device in local memory. This state is identical to the state affected by the *RST command. The device may have more than one location in which to store this data. Therefore, the command is followed by a numeric parameter designating the storage register to use. These numbers will begin at zero and end at an upperbound determined by the device.

7.2.9 Synchronization Commands

The synchronization commands enable the user to insure that commands are executed in unison on all devices. It does this by instructing the devices to wait to execute further commands until it has completed executing all commands that it is presently working on.

*OPC

Operation Complete

This command tells the device to set bit 0 in the Standard Event Status Register when it completes all pending operations.

*OPC?

Operation Complete Query

This query tells the device to place an ASCII '1' (decimal 49) in the device's output queue when it completes all pending operations.

*WAI

Wait to Continue

This command makes the device wait until all the previous commands or queries complete. The device then continues executing commands that follow the *WAI command.

7.2.10 System Data Commands

The System Data Commands give the user the abiltiy to gain further information about devices within a system. They include a means to query each device for its identity, read protected data and determine what device options are installed.

*IDN?

Identification Query

This command causes a device to send its "identity" over the bus. It sends it as an <Arbitrary ASCII Response Data> element. The data is organized as four fields separated by commas. The four fields are defined as follows:

Field 1	Manufacturer	required
Field 2	Model	required
Field 3	Serial Number	ASCII "0" if not available
Field 4	Firmware Level or equiv.	ASCII "0" if not available

Note that the ASCII character 0 mentioned above is the character "0", decimal value 48 and NOT the "null" character, decimal value 0. An example might be

HEWLETT-PACKARD,347A,2221A01113,A1

This entire length of the response must be less than or equal to 72 characters. The fields may not contain commas, semicolons, control characters (decimal 0-31), or DEL (127 decimal). Otherwise, the fields may contain any ASCII encoded character.

*OPT?

Option Identification Query

The Option Identification Query tells the device to identify any reportable device options. The device responds with an <Abritrary ASCII Response Data> element. The response may contain any number of fields separated by commas. The precise format of the fields is up to the device designer. Missing options respond with an ASCII 0 (48 decimal). If a device contains no reportable options it also responds with an ASCII 0 (48 decimal). Fields describing the options may contain any ASCII character except commas (44 decimal), semicolons (59 decimal), control characters (0-31 decimal) and DEL (127 decimal).

The overall length of the response must be less than or equal to 255 characters.

*PUD

Protected User Data Command

The Protected User Data Command stores up to 63 bytes of data unique to a device. For example, this data may be calibration date, usage time, environmental conditions, inventory control numbers, etc. This data is protected by some means, such as a recessed switch. The data can only be written when the protection mechanism is disabled. This data only affects the *PUD? query. It has no effect on the operation of the device.

*PUD?

Protected User Data Query

The Protected User Data Query reads the data from the *PUD storage area. This command thereby permits the user to retrieve the data previously stored.

*RDT

Resource Description Transfer Command

The Resource Description Transfer Command permits the user to store a Resource Descriptor, or a capability description of the device in a memory location of the device. This memory location is protected by some means such as a recessed switch so that it can only be written when the protection is disabled. This memory does not affect the operation of the device.

*RDT?

Resource Description Transfer Query

The Resource Description Transfer Query permits the user to read the Resource Description stored by the *RDT command. The device will respond with a <Definite Length Arbitrary Block Response Data> element.

Chapter 8
System Initialization

8.1 Overview

IEEE 488 devices can be very complex. As a result of this, initializing these devices can also be very complex. The IEEE 488.1 and 488.2 Standards define several levels of reset. The device can reset the 488 interface, the command interpreter interface, or its own internal functions.

This chapter will talk about:

- power-on reset
- the IEEE 488.2 Reset *RST command
- the IEEE 488.1 bus Device Clear Commands (DCL and SDC)
- the IEEE 488.1 Interface Clear Command (IFC)

8.2 Reset Protocol

The reset protocol initializes the entire system using three levels.

Bus Initialization The first level of reset puts the bus in the idle state.

Message Exchange Initialization The second level of reset insures the device can receive a new program message.

Device Initialization The third level of reset initializes device-dependent functions within a device.

8.3 Reset Commands

The following commands cause the three reset levels to occur:

IFC The 488.1 Interface Clear Message sets the system bus to an idle condition (bus initialization). All devices are unaddressed. The system controller becomes the controller-in-charge. This is level one reset.

DCL or SDC These 488.1 messages clear the input buffer and output queue, clear any commands in process and prepare the device to receive new commands (message exchange initialization). DCL, Device Clear clears all devices. SDC, Selected Device Clear clears devices addressed to listen. This is level two reset.

***RST** The 488.2 Reset Command initializes the device-dependent functions within a device (device initialization). This is level three reset.

8.4 Power-on Reset

At power-on, device settings may be returned to their settings when power was turned off. Some devices may power up to one known state. IEEE 488.2 also permits devices to power up to a state defined by the user. The Power-On Status Clear *PSC command is an example of this.

Power-on clears the Service Request Enable Register, the Standard Event Status Register, and the Parallel Poll Enable Register IF the power-on-status-clear flag is TRUE or if the *PSC command is not implemented.

Power-on may not affect the following:

- the device's bus address
- calibration data
- Resource Description
- Option Identification
- Protected User Data

Appendix A
Multipliers & Suffixes

Table A.1 Multiplier Mnemonics

Definition	Mnemonic	Name
1E18	EX	EXA
1E15	PE	PETA
1E12	T	TERA
1E9	G	GIGA
1E6	MA (Note)	MEGA
1E3	K	KILO
1E-3	M (Note)	MILLI
1E-6	U	MICRO
1E-9	N	NANO
1E-12	P	PICO
1E-15	F	FEMTO
1E-18	A	ATTO

NOTE: The suffix units, MHZ and MOHM are special cases which should not be confused with <suffix mult.>HZ and <suffix mult.>OHM.

Table A.2 Suffix Elements

Class	Preferred Suffix (primary unit)	Allowed Suffix (secondary unit)	Referenced Unit
Current	A		Ampere
Pressure	ATM		Atmosphere
Charge	C		Coulomb
Illumination	CD		Candela
Ratio	DB		Decibel
Power	DBM		Decibel milliwatt
Capacitance	F		Farad
Mass		G	Gram
Inductance	H		Henry
Frequency	HZ		Hertz
Pressure	INHG		Inches of mercury
Energy	J		Joule
Temperature	K		Degree Kelvin
		CEL	Degree Celsius
		FAR	Degree Fahrenheit
Volume	L		Liter
Illumination	LM		Lumen
Illumination	LX		Lux
Length	M		Meter
		FT	Feet
		IN	Inch
Frequency		MHZ	Megahertz
Resistance		MOHM	Megaohm
Force	N		Newton
Resistance	OHM		Ohm
Pressure	PAL		Pascal
Ratio	PCT		Percent
Angle	RAD		Radian
		DEG	Degree
		MNT	Minute (of arc)
		SEC	Second
Time	S		Second
Conductance	SIE		Siemens
Mag Density	T		Tesla
Pressure	TORR		Torr
Amplitude	V		Volt
Power	W		Watt
Mag Flux	WB		Weber
Luminous Flux	LM		Lumen

Appendix B
IEEE 754 Floating Point Format

The IEEE 488.2 standard recommends using the IEEE Std. 754-1985 for representing floating point numbers. This format specifies that each number be represented by three fields. These fields are:

1. The Sign Field
2. The Exponent Field
3. The Fraction Field

The size of these fields depends on the precision of the number. For single precision numbers:

Sign field width	1	bit
Exponent field width	8	bits
Fraction field width	23	bits
Total width	32	bits

With the exponent ranging from -126 to $+127$ with a bias of $+127$.

For Double precision numbers:

Sign field width	1	bit
Exponent field width	11	bits
Fraction field width	52	bits
Total width	64	bits

With the exponent ranging from -1022 to $+1023$ with a bias of $+1023$.

Let e represent the exponent, s the sign bit, and f the fraction represented above.

For a 32-bit single precision format number the IEEE 754 definition provides the following formulas to determine the value of the number x represented in floating point format:

If $e = 255$ and $f \neq 0$	then x is Not a Number (NaN)
If $e = 255$ and $f = 0$	then $x = -1^s(\infty)$
If $0 < e < 255$	then $x = -1^s(2^{e-127})(1 + f)$
If $e = 0$ and $f \neq 0$	then $x = -1^s(2^{-126})(0 + f)$
If $e = 0$ and $f = 0$	then $x = -1^s(0)$ (zero)

For a 64-bit double precision format number the IEEE 754 definition provides the following formulas to determine the value of the number x represented in floating point format:

If $e = 2047$ and $f \neq 0$	then x is Not a Number (NaN)
If $e = 2047$ and $f = 0$	then $x = -1^{s}(\infty)$
If $0 < e < 2047$	then $x = -1^{s}(2^{e-1023})(1 + f)$
If $e = 0$ and $f \neq 0$	then $x = -1^{s}(2^{-1022})(0 + f)$
If $e = 0$ and $f = 0$	then $x = -1^{s}(0)$ (zero)

Now that you know how to format the data and convert it back and forth, how do you send the bytes? The following figure shows the relationship of the bits for a single precision number.

DIO 8	7	6	5	4	3	2	1	
S	E_{msb}	E	E	E	E	E	E	First byte sent
E_{lsb}	F_{msb}	F	F	F	F	F	F	Second byte sent
F	F	F	F	F	F	F	F	Third byte sent
F	F	F	F	F	F	F	F_{lsb}	Fourth byte sent

Where:

E_{msb}	is the most significant bit of the exponent.
E_{lsb}	is the least significant bit of the exponent.
F_{msb}	is the most significant bit of the fraction.
F_{lsb}	is the least significant bit of the fraction.
S	is the sign bit.
E	is an exponent bit.
F	is a fraction bit.

The following figure shows the relationship of the bits for a double precision number.

DIO 8	7	6	5	4	3	2	1	
S	E_{msb}	E	E	E	E	E	E	First byte sent
E	E	E	E_{lsb}	F_{msb}	F	F	F	Second byte sent
F	F	F	F	F	F	F	F	Third through seventh byte sent
F	F	F	F	F	F	F	F_{lsb}	Last byte sent

Where: E_{msb} is the most significant bit of the exponent.
E_{lsb} is the least significant bit of the exponent.
F_{msb} is the most significant bit of the fraction.
F_{lsb} is the least significant bit of the fraction.
S is the sign bit.
E is an exponent bit.
F is a fraction bit.

This data would be received as an <Arbitrary Block Program Data>. It would be sent as an <Definite Length Arbitrary Block Response Data>.

Appendix C
IEEE 488.1 Capability Subset Codes

The ANSI/IEEE-488 recommends that all devices be marked near its connector with the interface function codes it supports.

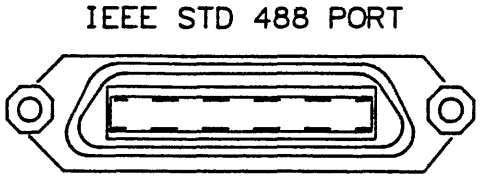


Figure C.1 HP-IB Connector and Codes

For example, a device with the basic talker function, the ability to send status bytes, the basic listener function, a listen only mode switch, service request capability without local lockout, manual configuration of the parallel poll capability, complete device clear capability, no capability for device trigger, and no controller capability would be identified with the codes.

Understanding these symbols is especially useful to the system's engineer as they completely describe the products' interface capability. IEEE 488.1 describes these functions in detail.

Interface Functions	Code
SOURCE HANDSHAKE	SH
ACCEPTOR HANDSHAKE	AH
TALKER (EXTENDED TALKER)	T(TE)
LISTENER (EXTENDED LISTENER)	L(LE)
SERVICE REQUEST	SR
REMOTE LOCAL	RL
PARALLEL POLL	PP
DEVICE CLEAR	DC
DEVICE TRIGGER	DT
CONTROLLER	C
DRIVER ELECTRONICS	E

Source Handshake (SH)

SH0 NO CAPABILITY

SH1 FULL CAPABILITY

Acceptor Handshake (AH)

AH0 NO CAPABILITY

AH1 FULL CAPABILITY

TALKER

Talker (T) Extended Talker (TE)

	Basic Talker	Serial Poll	Talk Only Mode	Unaddress if MLA
T(TE)0	NO	NO	NO	NO
T(TE)1	YES	YES	YES	NO
T(TE)2	YES	YES	NO	NO
T(TE)3	YES	NO	YES	NO
T(TE)4	YES	NO	NO	NO
T(TE)5	YES	YES	YES	YES
T(TE)6	YES	YES	NO	YES
T(TE)7	YES	NO	YES	YES
T(TE)8	YES	NO	NO	YES

LISTENER

Listener (L) Extended Listener (LE)

	Basic Listener	Listen Only Mode	Unaddressed if MTA
L(LE)0	NO	NO	NO
L(LE)1	YES	YES	NO
L(LE)2	YES	NO	NO
L(LE)3	YES	YES	YES
L(LE)4	YES	NO	YES

Service Request (SR)

SR0 NO CAPABILITY

SR1 FULL CAPABILITY

Remote Local (RL)

RL0 NO CAPABILITY

RL1 COMPLETE CAPABILITY

RL2 NO LOCAL LOCKOUT

Parallel Poll (PP)

PP0 NO CAPABILITY

PP1 REMOTE CONFIGURATION

PP2 LOCAL CONFIGURATION

Device Clear (DC)

DC0 NO CAPABILITY

DC1 FULL CAPABILITY

DC2 OMIT SELECTIVE DEVICE CLEAR

Device Trigger (DT)

DT0 NO CAPABILITY

DT1 FULL CAPABILITY

Driver Electronics (E)

E1 OPEN COLLECTOR (250KB/SEC MAX)

E2 TRI STATE (1MB/SEC MAX)

CONTROLLER (C)

There are 29 controller levels. The more significant are:

C0 NO CAPABILITY

C1 SYSTEM CONTROLLER

C2 SEND IFC AND TAKE CHARGE

C3 SEND REN

C4 RESPOND TO SRQ

C5 SEND INTERFACE MESSAGES, RECEIVE CONTROL, PASS CONTROL, PASS CONTROL TO SELF, PARALLEL POLL, TAKE CONTROL SYNCHRONOUSLY

Appendix D
Glossary of HP-IB related terms

To help you interpret the Standards and other Information Sources:

ACCEPTOR - A device receiving information on the Bus in either the Command or Data Mode.

ADDRESS - A 7-bit code applied to the HP-IB in "Command Mode" which enables instruments capable of responding to listen and/or talk on the Bus.

ADDRESSED COMMANDS - These commands allow the Bus controller to initiate actions from addressed instruments which are capable of responding.

ASCII - Acronym for American Standard Code for Information Interchange.

ATN - Control line (Attention) establishes between the "Command Mode" and "Data Mode" of operation on the HP-IB.

BIDIRECTIONAL BUS - A bus used by any individual device for two-way transmission of messages, that is, both input and output.

BIT - The smallest part of a binary character which contains intelligible information. A Binary Digit.

BIT-PARALLEL - Refers to a set of concurrent data bits present on a like number of signal lines used to carry information. Bit-parallel data bits may be acted upon concurrently as a group (byte) or independently as individual data bits.

BUS - A set of signal lines used by an interface system to which a number of devices are connected and over which messages are carried.

BUS COMMANDS - A group of ASCII Codes which initiate certain types of operation in devices capable of responding to these codes. Each instrument on the HP-IB is designed to respond to those codes that have useful meaning to the device and ignore all others.

BYTE - The binary character sent over the data bus. Although a byte usually refers to 8 bits, frequently the eighth bit is a don't care in an HP-IB system due to ASCII encoding.

BYTE-SERIAL - A sequence of bit-parallel data bytes used to carry information over a common bus.

COMMAND MODE - In this mode (ATN True) devices on the HP-IB can be addressed or unaddressed as talkers or listeners. Bus commands are also issued in this mode.

COMPATIBILITY - The degree to which devices may be interconnected and used, without modification.

CONTROLLER - Any device on the HP-IB which is capable of setting the ATN line and addressing instruments on the Bus as talkers and listeners. (Also see System Controller.)

DEVICE CLEAR (DCL) - ASCII character "DC4" (Decimal 20) which, when sent on the HP-IB will return all devices capable of responding to predefined states.

DATA MODE - The HP-IB is in this mode when the control line "ATN" is false (high). In this mode data or instructions are transferred between instruments on the HP-IB.

DAV - Mnemonic referring to the control line "Data Valid" on the HP-IB. This line is used in the HP-IB "Handshake" sequence.

DIO - Mnemonic referring to the eight "Data Input/Output" lines of the HP-IB.

EOI - Mnemonic referring to the control line "End or Identify" on the HP-IB. This line is used to indicate the end of a multiple byte message on the Bus. It is also used for Parallel Poll.

EXTENDED LISTENER - An instrument which can use two HP-IB bytes to address it as a listener. (Also see Listener.)

EXTENDED TALKER - An instrument which can use two HP-IB bytes to address it as a talker. (Also see Talker.)

GO TO LOCAL (GTL) - ASCII character "SOH" (Decimal 01) which, when sent on the HP-IB in Command Mode, will return devices addressed to listen and capable of responding back to local control.

GROUP EXECUTE TRIGGER (GET) - ASCII character "BS" (Decimal 08) which, when sent on the HP-IB in Command Mode, initiates simultaneous actions by devices addressed to listen and capable of responding to this command.

HANDSHAKE - Refers to the sequence of events on the HP-IB during which each data byte is transferred between addressed devices. The conditions of the HP-IB handshake sequence are as follows:

- NRFD, when false, indicates that a device is ready to receive data.
- DAV, when true, indicates that data on the DIO lines is stable and available to be accepted by the receiving device.
- NDAC, when false indicates to the transmitting device that data has been accepted by the receiver.

HIGH STATE - The relatively more positive signal level used to assert a specific message content associated with one or two binary logic states.

HP-IB - An abbreviation that refers to the "Hewlett-Packard Interface Bus."

IEEE - Acronym for Institute For Electrical and Electronic Engineers.

IFC - General Interface Management Line "Interface Clear" used by the system controller to halt all current operations on the bus, unaddress all other devices, and disable Serial Poll.

INTERFACE - A common boundary between a considered system and another system, or between parts of a system, through which information is conveyed.

INTERFACE SYSTEM - The device-independent mechanical, electrical, and functional elements of an interface necessary to effect communication among a set of devices. Cables, connector, driver and receiver circuits, signal line descriptions, timing and control conventions, and functional logic circuits are typical interface system elements.

LISTENER - A device which has been addressed to receive data or instructions from other instruments on the HP-IB. (Also see Extended Listener.)

LOCAL CONTROL - A method whereby a device is programmable by means of its local (front or rear panel) controls in order to enable the device to perform different tasks. (Also referred to as manual control.)

LOCAL LOCKOUT (LLO) - An HP-IB multiline universal command (ASCII "DC1" decimal 17) which disables the return-to-local control on a device (prevents user from leaving remote control other than cycling power). Clearing the REN line of the HP-IB restores local control and re-enables the return-to-local pushbutton on every HP-IB device.

LOW STATE - The relatively less positive signal level used to assert a specific message content associated with one of two binary logic states.

LU - Logical Unit.

MLA - Mnemonic, My Listen Address, meaning the address at which an HP-IB device will be enabled to "Listen" for data.

MTA - Mnemonic, My Talk Address, meaning the address at which an HP-IB device will be enabled to "Talk" or send data on the bus.

NDAC - Mnemonic referring to the control line "Not Data Accepted" on the HP-IB. This line is used in the HP-IB "Handshake" sequence.

NRFD - Mnemonic referring to the control line "Not Ready for Data" on the HP-IB. This line is used in the HP-IB "Handshake" sequence.

PARALLEL POLL - A method of simultaneously checking status of instruments on the HP-IB. Each instrument is assigned a DIO line with which to indicate whether it requested service or not. More than one instrument can be connected to one data line.

PRIMARY COMMANDS - The group of multiline messages consisting of universal commands, addressed commands, and device addresses sent by a CONTROLLER in the COMMAND MODE (ATN true).

PROGRAMMABLE - The characteristic of a device that makes it capable of accepting data to alter the state of its internal circuitry to perform a specific task(s).

PROGRAMMABLE MEASURING APPARATUS - A measuring apparatus that performs specified operations on command from the system and, may transmit the results of the measurements(s) to the system.

REMOTE CONTROL - A method whereby a device is programmable via its electrical interface connection in order to enable the device to perform different tasks.

REN - Mnemonic referring to the control line "Remote Enable" on the HP-IB. This line is used to enable Bus compatible instruments to respond to commands from the controller or another talker. It can be issued only by the system controller.

SECONDARY COMMANDS - The group of multiline messages used to increase the command address length of extended talkers and listeners to two bytes. Also includes Parallel Poll Enable and Disable.

SELECTIVE DEVICE CLEAR - ASCII character "EOT" (Decimal 04) which returns devices addressed to listen, to a predetermined state.

SERIAL POLL - The method of sequentially determining which device connected to the HP-IB has requested service. Only one instrument is checked at a time.

SERIAL POLL DISABLE (SPD) - ASCII character "EM" (Decimal 25) which, when sent on the HP-IB in Command Mode, will cause the Bus to go out of serial poll mode.

SOURCE - A device transmitting information on the Bus in either the Command or Data Mode.

SIGNAL - The physical representation of information.

SIGNAL LEVEL - The magnitude of signal compared to an arbitrary reference magnitude (voltage in the case of this standard).

SIGNAL LINE - One of a set of signal conductors in an interface system used to transfer messages among interconnected devices.

SYSTEM - A set of interconnected elements constituted to achieve a given objective by performing a specified function.

SRQ - Mnemonic referring to the control line "Service Request." This control line is used to enable Bus compatible instruments to tell the controller that they require service.

UNIDIRECTIONAL BUS - A bus used by an individual device for one-way transmission of messages only, that is, either input only or output only.

WORD - A group of bytes treated as a unit and given a single location in memory (organization defines the length of a computer "word"). HP computers typically use a word oriented memory with 16-bit (2 byte) words.

UNIVERSAL COMMAND - These commands allow the controller to send specific messages to all devices, whether or not the devices are addressed.

Appendix E
General HP-IB Bibliography

1. "American National Standard for the representation of numeric values in character strings for information interchange," ANSI X3.42-1975, American National Standards Institute, 1430 Broadway, New York, NY 10018.

2. IEEE Standard 488.1-1987, "Digital Interface for programmable instrumentation," The IEEE, Inc., 345 East 47th St., New York, NY, Jun. 1987.

3. IEEE Standard 488.2-1987, "Codes, Formats, Protocols, and Common Commands For use with ANSI/IEEE Std 488.1-1987," The IEEE, Inc., 345 East 47th St., New York, NY, Jun. 1987.

4. USA Standard Code for Information Interchange, ANSI X3.4-1977, American National Standards Institute, 1430 Broadway, New York, NY 10018.

INDEX

U

W

B7 B6 B5 / BITS B4 B3 B2 B1	Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
Ø Ø Ø Ø	Ø NUL Ø	2Ø DLE 16	4Ø SP 32	6Ø Ø 48	1ØØ @ 64	12Ø P 8Ø	14Ø ` 96	16Ø p 112
Ø Ø Ø 1	1 GTL SOH 1	21 LLO DC1 17	41 ! 33	61 1 49	1Ø1 A 65	121 Q 81	141 a 97	161 q 113
Ø Ø 1 Ø	2 STX 2	22 DC2 18	42 " 34	62 2 5Ø	1Ø2 B 66	122 R 82	142 b 98	162 r 114
Ø Ø 1 1	3 ETX 3	23 DC3 19	43 # 35	63 3 51	1Ø3 C 67	123 S 83	143 c 99	163 s 115
Ø 1 Ø Ø	4 SDC EOT 4	24 DCL DC4 2Ø	44 $ 36	64 4 52	1Ø4 D 68	124 T 84	144 d 1ØØ	164 t 116
Ø 1 Ø 1	5 PPC ENQ 5	25 PPU NAK 21	45 % 37	65 5 53	1Ø5 E 69	125 U 85	145 e 1Ø1	165 u 117
Ø 1 1 Ø	6 ACK 6	26 SYN 22	46 & 38	66 6 54	1Ø6 F 7Ø	126 V 86	146 f 1Ø2	166 v 118
Ø 1 1 1	7 BEL 7	27 ETB 23	47 ' 39	67 7 55	1Ø7 G 71	127 W 87	147 g 1Ø3	167 w 119
1 Ø Ø Ø	1Ø GET BS 8	3Ø SPE CAN 24	5Ø (4Ø	7Ø 8 56	11Ø H 72	13Ø X 88	15Ø h 1Ø4	17Ø x 12Ø
1 Ø Ø 1	11 TCT HT 9	31 SPD EM 25	51) 41	71 9 57	111 I 73	131 Y 89	151 i 1Ø5	171 y 121
1 Ø 1 Ø	12 LF A 1Ø	32 SUB 1A 26	52 * 42	72 : 3A 58	112 J 74	132 Z 9Ø	152 j 1Ø6	172 z 122
1 Ø 1 1	13 VT B 11	33 ESC 1B 27	53 + 43	73 ; 3B 59	113 K 75	133 [91	153 k 1Ø7	173 { 123
1 1 Ø Ø	14 FF C 12	34 FS 1C 28	54 , 44	74 < 3C 6Ø	114 L 76	134 \ 92	154 l 1Ø8	174 ¦ 124
1 1 Ø 1	15 CR D 13	35 GS 1D 29	55 - 45	75 = 3D 61	115 M 77	135] 93	155 m 1Ø9	175 } 125
1 1 1 Ø	16 SO E 14	36 RS 1E 3Ø	56 . 46	76 > 3E 62	116 N 78	136 ^ 94	156 n 11Ø	176 ~ 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 / 47	77 ? 3F 63	117 O 4F Ø	137 - 5F 95	157 o 111	177 RUBOUT (DEL) 7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

KEY: octal 25 PPU / NAK / hex 15 21 — Message Mnemonic; ASCII/ISO character; decimal

ASCII 7-bit Code Chart