GPIB TUTORIAL

BACKGROUND

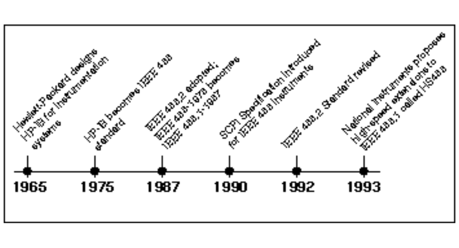
Instrumentation has always leveraged off widely used electronics technology to drive its innovation. The jeweled movement of the clock was first used to build analog meters. The variable capacitor, the variable resistor, and the vacuum tube from radios were used to pioneer the first electronic instruments. Display technology was leveraged off the television for use in oscilloscopes and analyzers.

Today, cost-effective and powerful desktop and notebook computers are paving the way for new types of instruments --virtual instruments--. Virtual instruments are designed and built by the user to match specific needs by leveraging off the power and low cost of PCs and workstations.

Software is the key to virtual instruments. Application software empowers the user with the tools necessary to build virtual instruments and expand their functionality by providing connectivity to the enormous capabilities of PCs, workstations, and their assortment of applications, boosting performance, flexibility, reusability and reconfigurability while diminishing at the same time development and maintenance costs.

FUNDAMENTALS OF VIRTUAL INSTRUMENTS	
Traditional Instruments	**Virtual Instruments**
Vendor-defined	User-defined
Function-specific, stand-alone with limited connectivity	Application-oriented system with connectivity to networks, peripherals, and applications
Hardware is the key	Software is the key
Expensive	Low-cost, reusable
Closed, fixed functionality	Open, flexible functionality leveraging off familiar computer technology
Slow turn on technology (5-10 year life cycle)	Fast turn on technology (1-2 year life cycle)
Minimal economics of scale	Maximum economics of scale
High development and maintenance costs	Software minimizes development and maintenance costs

INTRODUCTION



In 1965, Hewlett-Packard designed the Hewlett-Packard Interface Bus (**HP-IB**) to connect their line of programmable instruments to their computers. Because of its high transfer rates (nominally 1 Mbytes/s), this interface bus quickly gained popularity. It was later accepted as IEEE Standard 488-1975, and has evolved to ANSI/IEEE Standard **488.1**-1987. Today, the name **G** eneral **P**urpose **I**nterface **B**us (**GPIB**) is more widely used than HP-IB. ANSI/IEEE **488.2**-1987 strengthened the original standard by defining precisely how controllers and instruments communicate. **S** tandard **C**ommands for **P**rogrammable **I**nstruments (**SCPI**) took the command structures defined in IEEE 488.2 and created a single, comprehensive programming command set that is used with any SCPI instrument. Figure 1 summarizes GPIB history.

TYPES OF GPIB MESSAGES

GPIB devices communicate with other GPIB devices by sending device-dependent messages and interface messages through the interface system.

- **Device-dependent** messages, often called **data** or data messages, contain device-specific information, such as programming instructions, measurement results, machine status, and data files.
- **Interface** messages manage the bus. Usually called **commands** or command messages, interface messages perform such functions as initializing the bus, addressing and unaddressing devices, and setting device modes for remote or local programming.

The term "command" as used here should not be confused with some device instructions that are also called commands. Such device-specific commands are actually data messages as far as the GPIB interface system itself is concerned.

TALKERS, LISTENERS AND CONTROLLERS

GPIB Devices can be Talkers, Listeners, and/or Controllers. A **Talker** sends data messages to one or more **Listeners**, which receive the data. The **Controller** manages the flow of information on the GPIB by sending commands to all devices. A digital voltmeter, for example, is a Talker and is also a Listener.

The GPIB is like an ordinary computer bus, except that a computer has its circuit cards interconnected via a backplane - the GPIB has stand-alone devices interconnected by standard cables.

The role of the GPIB Controller is comparable to the role of a computer CPU, but a better analogy is to compare the Controller to the switching center of a city telephone system.

The switching center (Controller) monitors the communications network (GPIB). When the center (Controller) notices that a party (device) wants to make a call (send a data message), it connects the caller (Talker) to the receiver (Listener).

The Controller usually addresses (or enables) a Talker and a Listener before the Talker can send its message to the Listener. After the message is transmitted, the Controller may address other Talkers and Listeners.

Some GPIB configurations do not require a Controller. For example, a device that is always a Talker, called a talk-only device, is connected to one or more listen-only devices.

A Controller is necessary when the active or addressed Talker or Listener must be changed. The Controller function is usually handled by a computer.

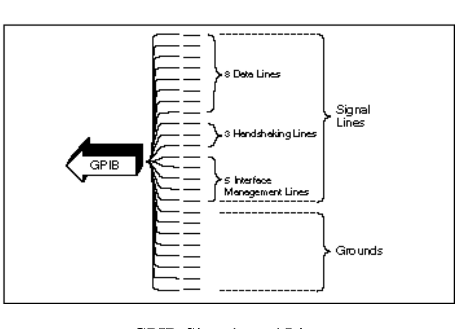
A computer with the appropriate hardware and software could perform the roles of Talker/Listener and Controller.

THE CONTROLLER-IN-CHARGE AND SYSTEM CONTROLLER

Although there can be multiple Controllers on the GPIB, **at any time only one Controller is the Controller-In-Charge (CIC)**. Active control can be passed from the current CIC to an idle Controller. Only the System Controller can make itself the CIC.

GPIB SIGNALS AND LINES

The GPIB interface system consists of 16 signal lines and eight ground-return or shield-drain lines. The 16 signal lines, discussed below, are grouped into data lines (eight), handshake lines (three), and interface management lines (five) (see Figure 2).



GPIB Signals and Lines

Data Lines

The eight data lines, DIO1 through DIO8, carry both data and command messages. The state of the Attention (ATN) line determines whether the information is data or commands. All commands and most data use the 7-bit ASCII or ISO code set, in which case the eighth bit, DIO8, is either unused or used for parity.

Handshake Lines

Three lines **asynchronously** control the transfer of message bytes between devices. The process is called a **3-wire interlocked handshake**. It guarantees that message bytes on the data lines are sent and received without transmission error.

- NRFD (not ready for data) - Indicates when a device is ready or not ready to receive a message byte. The line is driven by all devices when receiving commands, by Listeners when receiving data messages, and by the Talker when enabling the HS488 protocol.
- NDAC (not data accepted) - Indicates when a device has or has not accepted a message byte. The line is driven by all devices when receiving commands, and by Listeners when receiving data messages.
- DAV (data valid) - Tells when the signals on the data lines are stable (valid) and can be accepted safely by devices. The Controller drives DAV when sending commands, and the Talker drives DAV when sending data messages.

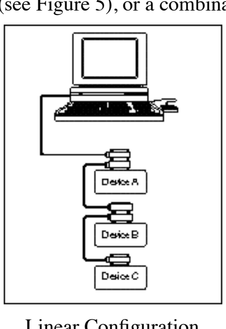
Interface Management Lines

Five lines manage the flow of information across the interface:

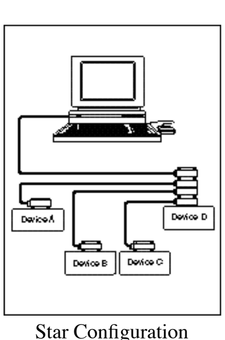
- ATN (attention) - The Controller drives ATN true when it uses the data lines to send commands, and drives ATN false when a Talker can send data messages.
- IFC (interface clear) - The System Controller drives the IFC line to initialize the bus and become CIC.
- REN (remote enable) - The System Controller drives the REN line, which is used to place devices in remote or local program mode.
- SRQ (service request) - Any device can drive the SRQ line to asynchronously request service from the Controller.
- EOI (end or identify) - The EOI line has two purposes - The Talker uses the EOI line to mark the end of a message string, and the Controller uses the EOI line to tell devices to identify their response in a parallel poll.

PHYSICAL AND ELECTRICAL CHARACTERISTICS

Devices are usually connected with a shielded 24-conductor **cable with both a plug and receptacle connector at each end** (see Figure 3). You can link devices in either a linear configuration (see Figure 4), a star configuration (see Figure 5), or a combination of the two.



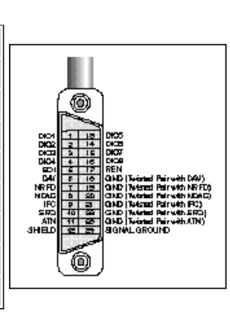
Linear Configuration



Star Configuration

The standard connector is the Amphenol or Cinch Series 57 MICRORIBBON or AMP CHAMP type. For special interconnect applications, an adapter cable with non-standard cable and/or connectors is used.

DATA LINES	Pin No.
DIO1	1
DIO2	2
DIO3	3
DIO4	4
DIO5	13
DIO6	14
DIO7	15
DIO8	16



MANAGEMENT LINES	Pin No.
IFC	9
REN	17
ATN	11
SRQ	10
EOI	5

HANDSHAKE LINES	Pin No.
DAV	6
NRFD	7
NDAC	8

GPIB Connector and Signal Assignment

The GPIB uses **negative logic with standard TTL** levels. When DAV is true, for example, it is a TTL low level (<= 0.8 V), and when DAV is false, it is a TTL high level (>= 2.0 V).

CONFIGURATION REQUIREMENTS

To achieve the high data transfer rate for which the GPIB was designed, the physical distance between devices and the number of devices on the bus are limited.

The following restrictions are typical for normal operation:

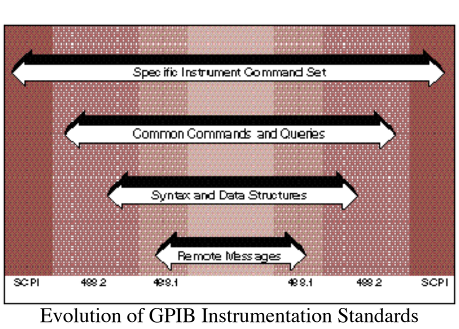
- A maximum separation of 4 m between any two devices and an average separation of **2 m** over the entire bus
- A maximum total cable length of **20 m**
- No more than **15 device** loads connected to each bus, with no less than two-thirds powered on

For higher speed systems using the 3-wire IEEE 488.1 handshake (T1 delay = 350 ns), and HS488 systems, the following restrictions apply:

- A maximum total cable length of 15 m with a device load per 1 m cable
- All devices should be powered on
- All devices should use 48 mA three-state drivers
- Device capacitance on each GPIB signal should be less than 50 pF per device

IEEE 488.2 AND SCPI

The SCPI and IEEE 488.2 standards addressed the limitations and ambiguities of the original IEEE 488 standard. **IEEE 488.2** makes it possible to design more compatible and productive test systems. **SCPI** simplifies the programming task by defining a single comprehensive command set for programmable instrumentation, regardless of type or manufacturer. The scope of each of the IEEE 488, IEEE 488.2, and SCPI standards is shown in Figure 6.



Evolution of GPIB Instrumentation Standards

The ANSI/IEEE Standard 488-1975, now called **IEEE 488.1**, greatly simplified the interconnection of programmable instrumentation by clearly defining mechanical, electrical, and hardware protocol specifications. For the first time, instruments from different manufacturers were interconnected by a standard cable. Although this standard went a long way towards improving the productivity of test engineers, the standard did have a number of shortcomings. Specifically, IEEE 488.1 did not address data formats, status reporting, message exchange protocol, common configuration commands, or device-specific commands. As a result, each manufacturer implemented these items differently, leaving the test system developer with a formidable task.

IEEE 488.2 enhanced and strengthened IEEE 488.1 by standardizing data formats, status reporting, error handling, Controller functionality, and common commands to which all instruments must respond in a defined manner. By standardizing these issues, IEEE 488.2 systems are much more compatible and reliable. The IEEE 488.2 standard focuses mainly on the software protocol issues and thus maintains compatibility with the hardware-oriented IEEE 488.1 standard.

SCPI built on the IEEE 488.2 standard and defined device-specific commands that standardize programming instruments. SCPI systems are much easier to program and maintain. In many cases, you can interchange or upgrade instruments without having to change the test program. The combination of SCPI and IEEE 488.2 offers significant productivity gains, and finally, delivers as sound a software standard as IEEE 488.1 did a hardware standard.

IEEE 488.2

IEEE 488.2-1987 encouraged a new level of growth and acceptance of the IEEE 488 bus or GPIB by addressing problems that had arisen from the original IEEE 488 standard. IEEE 488.2 was drafted on the premise that it stay compatible with the existing IEEE 488.1 standard. The overriding concept used in the IEEE 488.2 specification for the communication between Controllers and instruments is that of "precise talking" and "forgiving listening." In other words, IEEE 488.2 exactly defined how both IEEE 488.2 Controllers and IEEE 488.2 instruments talk so that a completely IEEE 488.2-compatible system can be highly reliable and efficient. The standard also required that IEEE 488.2 devices be able to work with existing IEEE 488.1 devices by accepting a wide range of commands and data formats as a Listener. You obtain the true benefits of IEEE 488.2 when you have a completely IEEE 488.2-compatible system.

CONTROLLERS

Although IEEE 488.2 had less impact on Controllers than it did on instruments, there are several requirements and optional improvements for Controllers that made an IEEE 488.2 Controller a necessary component of test systems. IEEE 488.2 precisely defined the way IEEE 488.2 Controllers send commands and data and added functionality. Because of these IEEE 488.2 Controller requirements, instrument manufacturers can design more compatible and efficient instruments. The benefits of this standardization for the test system developer are reduced development time and cost, because it solves the problems caused by instrument incompatibilities, varying command structures, and data formats.

Requirements of IEEE 488.2 Controllers

IEEE 488.2 defined a number of requirements for a Controller, including an exact set of IEEE 488.1 interface capabilities, such as pulsing the interface clear line for 100 μs, setting and detecting EOI, setting/asserting the REN line, sensing the state and transition of the SRQ line, sensing the state of NDAC, and timing out on any I/O transaction. Other key requirements for Controllers are bus control sequences and bus protocols.

IEEE 488.2 Control Sequences

The IEEE 488.2 standard defined control sequences that specify the exact IEEE 488.1 messages that are sent from the Controller as well as the ordering of multiple messages. IEEE 488.2 defined 15 required control sequences and four optional control sequences, as shown in Table 1.

Description	Control Sequence	Compliance
Send ATN-true commands	SEND COMMAND	Mandatory
Send address to send data	SEND SETUP	Mandatory
Send ATN-false data	SEND DATA BYTES	Mandatory
Send a program message	SEND	Mandatory
Send address to receive data	RECEIVE SETUP	Mandatory
Receive ATN-false data	RECEIVE/RESPONSE MESSAGE	Mandatory
Receive a response message	RECEIVE	Mandatory
Pulse IFC line	SEND IFC	Mandatory
Place device in DCAS	DEVICE CLEAR	Mandatory
Place devices in local state	ENABLE LOCAL CONTROLS	Mandatory
Place devices in remote state	ENABLE REMOTE	Mandatory
Place devices in remote with local lockout state	SET RWLS	Mandatory
Place devices in local lockout state	SEND LLO	Mandatory
Read IEEE-488.1 status byte	READ STATUS BYTE	Mandatory
Send group execution trigger (GET) message	TRIGGER	Mandatory
Give control to another device	PASS CONTROL	Optional
Conduct a parallel poll	PERFORM PARALLEL POLL	Optional
Configure devices' parallel poll responses	PARALLEL POLL CONFIGURE	Optional
Disable devices' parallel poll capability	PARALLEL POLL UNCONFIGURE	Optional

IEEE 488.2 Required and Optional Control Sequences

The IEEE 488.2 control sequences describe the exact states of the GPIB and the ordering of command messages for each of the defined operations. IEEE 488.2 control sequences remove the ambiguity of the possible bus conditions, so instruments and Controllers are much more compatible. By exactly defining the state of the bus and how devices should respond to specific messages, IEEE 488.2 removes such system development problems.

IEEE 488.2 Protocols

Protocols are high-level routines that combine a number of control sequences to perform common test system operations. IEEE 488.2 defines two required protocols and six optional protocols, as shown in Table 2.

Keyword	Name	Compliance
RESET	Reset System	Mandatory
FINDRQS	Find Device Requesting Service	Optional
ALLSPOLL	Serial Poll All Devices	Mandatory
PASSCTL	Pass Control	Optional
REQUESTCTL	Request Control	Optional
FINDLSTN	Find Listeners	Optional
SETADD	Set Address	Optional, but requires FINDLSTN
TESTSYS	Self-Test System	Optional

IEEE 488.2 Controller Protocols

These protocols reduce development time because they combine several commands to execute the most com-mon operations required by any test system. The RESET protocol ensures that the GPIB has been initialized and all devices have been cleared and set to a known state. The ALLSPOLL protocol serial polls each device and returns the status byte of each device. The PASSCTL and REQUESTCTL protocols pass control of the bus between a number of different devices. The TESTSYS protocol instructs each device to run its own self-tests and report back to the Controller whether it has a problem or is ready for operation.

Perhaps the two most important protocols are FINDLSTN and FINDRQS. The FINDLSTN protocol takes advantage of the IEEE 488.2 Controller capability of monitoring bus lines to locate listening devices on the bus. The Controller implements the FINDLSTN protocol by issuing a particular listen address and then monitoring the NDAC handshake line to determine if a device exists at that address. The result of the FINDLSTN protocol is a list of addresses for all the located devices. FINDLSTN is used at the start of an application program to ensure proper system configuration and to provide a valid list of GPIB devices that can be used as the input parameter to all other IEEE 488.2 protocols. The bus line monitoring capability of an IEEE 488.2 Controller is also useful to detect and diagnose problems within a test system.

The FINDRQS protocol is an efficient mechanism for locating and polling devices that are requesting service. It uses the IEEE 488.2 Controller capability of sensing the FALSE to TRUE transition of the SRQ line. You prioritize the input list of devices so that the more critical devices receive service first. If the application program can jump to this protocol immediately upon the assertion of the SRQ line, you increase program efficiency and throughput.

IEEE 488.2 INSTRUMENTS

The IEEE 488.2 specifications for instruments can require major changes in the firmware and possibly the hardware. However, IEEE 488.2 instruments are easier to program because they respond to common commands and queries in a well defined manner using standard message exchange protocols and data formats. The IEEE 488.2 message exchange protocol is the foundation for the SCPI standard that makes test system programming even easier.

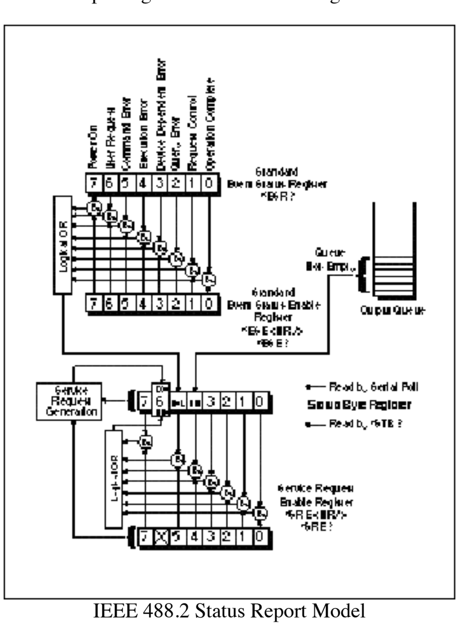
IEEE 488.2 defines a minimum set of IEEE 488.1 interface capabilities that an instrument must have. All devices must be able to send and receive data, request service, and respond to a device clear message. IEEE 488.2 defines precisely the format of commands sent to instruments and the format and coding of responses sent by instruments.

All instruments must perform certain operations to communicate on the bus and report status. Because these operations are common to all instruments, IEEE 488.2 defined the programming commands used to execute these operations and the queries used to receive common status information. These common commands and queries are shown in Table 3.

Mnemonic	Group	Description
*IDN?	System Data	Identification query
*RST	Internal Operations	Reset
*TST?	Internal Operations	Self-test query
*OPC	Synchronization	Operation complete
*OPC?	Synchronization	Opearation complete query
*WAI	Synchronization	Wait to complete
*CLS	Status and Event	Clear status
*ESE	Status and Event	Event status enable
*ESE?	Status and Event	Event status enable query
*ESR?	Status and Event	Event status register query
*SRE	Status and Event	Service request enable
*SRE?	Status and Event	Service request enable query
*STB?	Status and Event	Read status byte query

IEEE 488.2 Mandatory Common Commands

Because IEEE 488.2 standardizes status reporting, the Controller knows exactly how to obtain status information from every instrument in the system. This status reporting model builds upon the IEEE 488.1 status byte to provide more detailed status information. The status reporting model is shown in Figure 7.



IEEE 488.2 Status Report Model

SCPI

On April 23, 1990, a group of instrument manufacturers announced the SCPI specification, which defines a common command set for programming instruments. Before SCPI, each instrument manufacturer developed its own command sets for its programmable instruments. This lack of standardization forced test system developers to learn a number of different command sets and instrument-specific parameters for the various instruments used in an application, leading to programming complexities and resulting in unpredictable schedule delays and development costs. By defining a standard programming command set, SCPI decreases development time and increases the readability of test programs and the ability to interchange instruments.

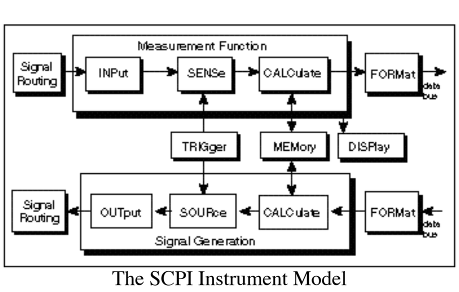
SCPI is a complete, yet extendable, standard that unifies the software programming commands for instruments. The first version of the standard was released in mid-1990. Today, the SCPI Consortium continues to add commands and functionality to the SCPI standard. SCPI has its own set of required common commands in addition to the mandatory IEEE 488.2 common commands and queries. Although IEEE 488.2 is used as its basis, SCPI defines programming commands that you can use with any type of hardware or communication link.

SCPI specifies standard rules for abbreviating command keywords and uses the IEEE 488.2 message exchange protocol rules to format commands and parameters. You may use command keywords in their long form (MEASure) or their short form shown in capital letters (MEAS).

SCPI offers numerous advantages to the test engineer. One of these is that SCPI provides a comprehensive set of programming functions covering all the major functions of an instrument. This standard command set ensures a higher degree of instrument interchangeability and minimizes the effort involved in designing new test systems. The SCPI command set is hierarchical, so adding commands for more specific or newer functionality is easily accommodated.

The SCPI Instrument Model

As a means of achieving compatibility and categorizing command groups, SCPI defined a model of a programmable instrument. This model, shown in Figure 8, applies to all the different types of instrumentation.



The SCPI Instrument Model

All of the functional components of the instrument model may not apply to every instrument. For example, an oscilloscope does not have the functionality defined by the signal generation block in the SCPI model. SCPI defines hierarchical command sets to control specific functionality within each of these functional components.

The signal routing component controls the connection of a signal to the instrument's internal functions; the measurement component converts the signal into a preprocessed form; and the signal generation component converts internal data to real-world signals. The memory component stores data inside the instrument. The format component converts the instrument data to a form that you can transmit across a standard bus. The trigger component synchronizes instrument actions with internal functions, external events, or other instruments.

The measurement function gives the highest level of compatibility between instruments because a measurement is specified by signal parameters, not instrument functionality. In most cases, you can exchange an instrument that makes a particular measurement with another instrument capable of making the same measurement without changing the SCPI command.

The MEASurement component is subdivided into three distinct parts --INPut, SENSe, and CALCulate--. The INPut component conditions the incoming signal before it is converted into data by the SENSe block. INPut functions include filtering, biasing, and attenuation. The SENSe component converts signals into internal data that you can manipulate. SENSe functions control such parameters as range, resolution, gate time, and normal mode rejection. The CALCulate component converts the acquired data into a more useful format for a particular application. CALCulation functions include converting units, rise time, fall time, and frequency parameters.

The signal generation component converts data into output as physical signals. SCPI subdivides the signal generation block into three function blocks --OUTPut, SOURce, and CALCulate--. The OUTPut block conditions the outgoing signal after it is generated. OUTPut block functions include filtering, biasing, and attenuation. The SOURce block generates a signal based on specified characteristics and internal data. SOURce block functions specify such signal parameters as amplitude modulation, power, current, voltage, and frequency. The CALCulate block converts application data to account for signal generation anomalies such as correcting for external effects, converting units, and changing domains.

Example SCPI Command

The following command programs a digital multimeter (DMM) to configure itself to make an AC voltage measurement on a signal of 20 V with a 0.001 V resolution.

```
: MEASure:VOLTage:AC? 20, 0.001
```

- The leading colon indicates a new command is coming
- The keywords MEASure:VOLTage:AC instruct the DMM to take an AC voltage measurement
- The ? instructs the DMM to return its measurement to the computer/controller
- The 20, 0.001 specifies the range (20 V) and resolution (.001 V) of the measurement

---

Reference Documents

For more information on the GPIB standards (TC8), refer to the following documents:

- IEEE **488.1**-2003, *Standard For Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation*
- IEEE **488.2**-1992 (Reaffirmed in 2004), *Standard Codes, Formats, Protocols, and Common Commands For Use With IEEE-488.1*
- The latest **SCPI** Standards are published by the SCPI Consortium .
- GPIB related material:
  - Hewlett-Packard, *Tutorial Description of the Hewlett-Packard Interface Bus*
    Hewlett-Packard, Nov. 1987

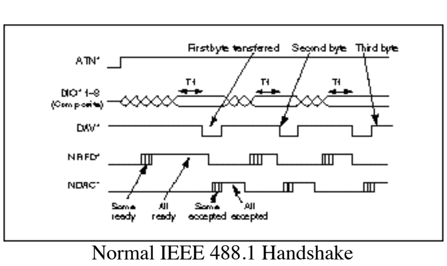
THE HIGH-SPEED GPIB HANDSHAKE PROTOCOL (HS488)

National Instruments has developed the patented high-speed GPIB handshake protocol (called HS488) to increase the data transfer rate of a GPIB system. All devices involved in a data transfer must be HS488 compliant to use the HS488 protocol, but when non-HS488 devices are involved, the HS488 devices automatically use the standard IEEE 488.1 handshake to ensure compatibility. HS488 is a superset of the IEEE 488 standards.

IEEE 488 HANDSHAKE

The standard IEEE 488.1 3-wire handshake (shown in Figure 9) requires the Listener to unassert Not Ready for Data (NRFD), the Talker to assert the Data Valid (DAV) signal to indicate to the Listener that a data byte is available, and for the Listener to unassert the Not Data Accepted (NDAC) signal when it has accepted that byte. A byte cannot transfer in less than the time it takes for the following events to occur:

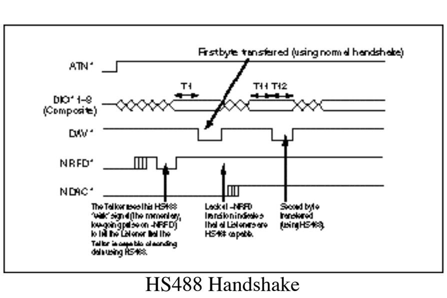
- NRFD to propagate to the Talker,
- DAV signal to propagate to all Listeners,
- the Listeners to accept the byte and assert NDAC,
- the NDAC signal to propagate back to the Talker, and
- the Talker to allow a settling time (T1) before asserting DAV again.



Normal IEEE 488.1 Handshake

HS488 HANDSHAKE

HS488 increases system throughput by removing propogation delays associated with the 3-wire handshake. To enable the HS488 handshake, the Talker pulses the NRFD signal line after the Controller addresses all Listeners. If the Listener is HS488 capable, then the transfer occurs using the HS488 handshake (shown in Figure 10). Once HS488 is enabled, the Talker places a byte on the GPIB DIO lines, waits for a preprogrammed settling time, asserts DAV, waits for a preprogrammed hold time, unasserts DAV, and drives the next data byte on the DIO lines. The Listener keeps NDAC unasserted and must accept the byte within the specified hold time. A byte must transfer in the time set by the settling time and hold time, without waiting for any signals to propagate along the GPIB cable.



HS488 Handshake

HS488 Data Transfer Flow Control

The Listener may assert NDAC to temporarily prevent more bytes from being transmitted, or assert NRFD to force the Talker to use the 3-wire handshake. Through these methods, the Listener can limit the average transfer rate. However, the Listener must have an input buffer that can accept short bursts of data at the maximum rate, because by the time NDAC or NRFD propagates back to the Talker, the Talker may have already sent another byte.

The required settling and hold times are user configurable, depending on the total length of cable and number of devices in the system. Between two devices and 2 m of cable, HS488 can transfer data up to 8 Mbytes/s. For a fully loaded system with 15 devices and 15 m of cable, HS488 transfer rates can reach 1.5 Mbytes/s.

HS488 Controllers always use the standard IEEE 488.1 3-wire handshake to transfer GPIB commands (bytes with Attention (ATN) asserted).

*****
STANDARDS: Official IEEE site for P488.1 - High-speed revision of 488.1-1987 standard instrument bus.
http://grouper.ieee.org/groups/imstc8/488/1/
