The Hewlett-Packard Interface Bus (HP-IB)

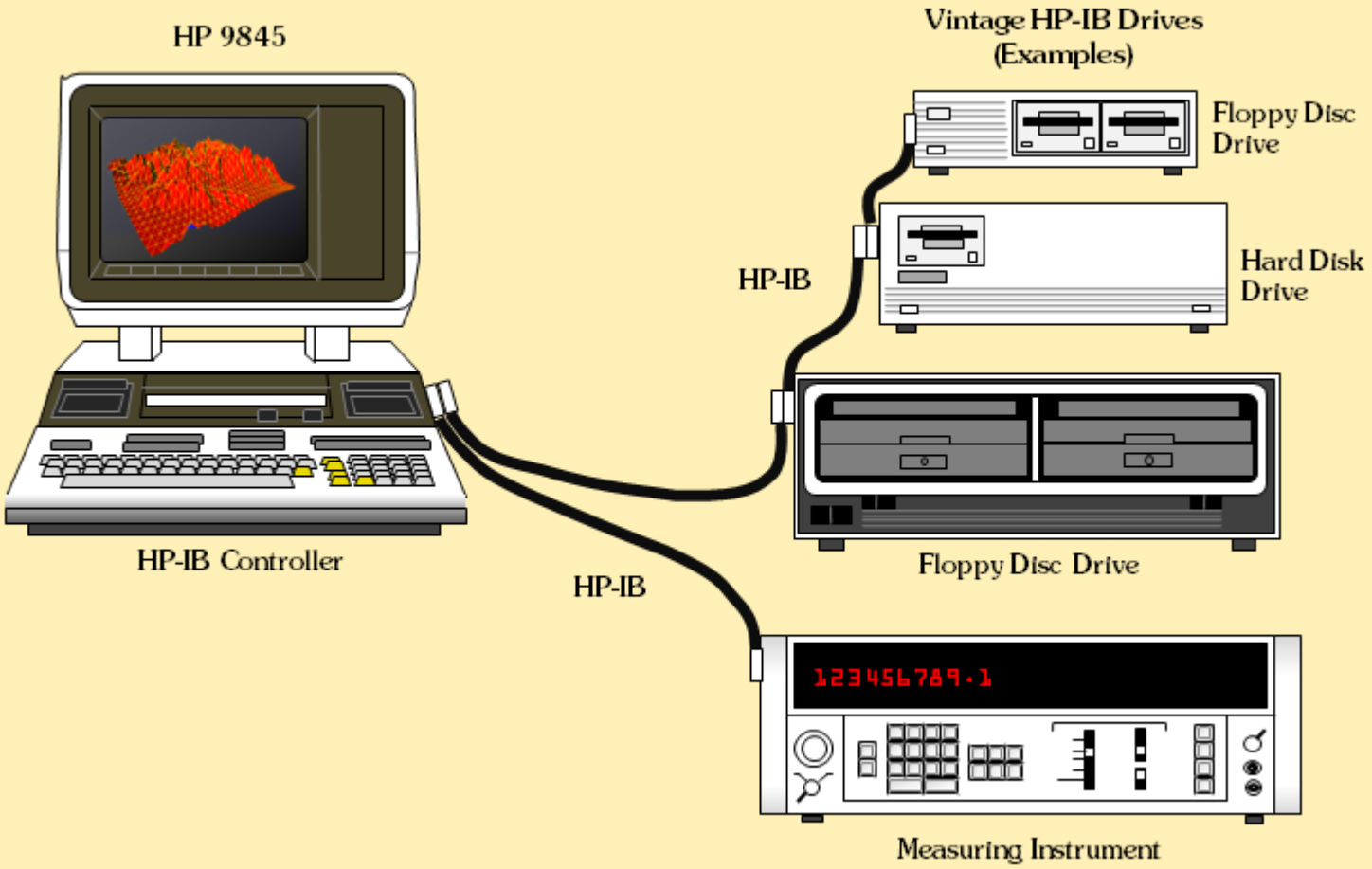
From The HP 9845 Project

http://www.hp9845.net/9845/tutorials/hpib/

Terms if Use (From The HP 9845 Project by A. Kückes)

The Hewlett-Packard Interface Bus (HP-IB)

The HP 9845 is closely related to the Hewlett-Packard Interface Bus or HP-IB, also known as IEEE-488 or General Purpose Interface Bus (GPIB). Most of the HP 9845 periperals such as mass storage, graphics devices or printers can be connected via the HP 98034 HP-IB interface. Even today the HP-IB is the entry to the HP 9845, since it is the easiest and most versatile connection between the HP 9845 and a modern PC. And the required 98034A/B HP-IB interfaces are still relatively easy to acquire.



Hewlett-Packard Interface Bus (HP-IB) with HP 9845C as Controller

"*These bullet-proof cables are wonderful in a museum environment (for those who are not experts in the myriad of forms of data communication in the industry). Getting two HP-IB devices to talk to each other is primarily a simple matter of setting an address switch. No need to worry about strange cable pin-outs, communication or handshaking settings. HP-IB was not as easy as plug-and-play, but it was as close as it got in the day.*" This is what the *hpmuseum* says about the HP-IB.

The story of the HP-IB began long before the HP 9845. In the year 1965, Hewlett-Packard already was one of the leaders in measuring instrument supply, and digital data acquistion became an increasingly important matter. From the first frequency counter connected to a digital recorder in the mid-1950's demand increased also for the remote programming of instruments in the 1960's. During the late 1960's there already was much experience with bus systems for interconnecting CPUs and internal components, and also diverse instruments and controllers were interconnected with 'bridging' interface cards plugged into a common backplane. So HP's engineers already had good practice in designing internal bus systems. The final step from the backplane towards a cable based network was at hand.

So HP decided to look forward to the standardization of interfacing to all of HP's future instruments. As HP's corporate interface engineer Don Loughry describes in his own words: "You want be able to have different engineers in different places at different times with different applications, define and design products to a common interface, and then be able to put these independently designed products together in a common

system...". The first known publication was in the October 1972 issue of the HP Journal. As we learn from Don Loughry in his article "A Common Digital Interface for Programmable Instruments: The Evolution of a System", the primary four objectives for developing an effective interface system were:

1. The interface system must be capable of interconnecting small, low-cost instrument systems by means of simple passive cable assemblies without restricting individual instrument performance and cost.
2. The interface system must be capable of interconnecting a wide range of products (measurement, stimulus, display, processor, storage) needed to solve real problems.
3. The interface system must be capable of operating where control and management of the message flow over the interface is not limited to one device but can be delegated in an orderly manner among several.
4. The interface system must be capable of interfacing easily with other more specialized interface environments.

Don Loughry

Additional criteria included:

- Operation under asynchronous conditions
- Communication with two or more devices simultaneously
- Limitation of the total number of signal lines to no more than it can be accomodated in one computer word
- Compatibility with the most widely used codes for information interchange
- Ability to alter codes and data rates in order to achieve system flexibility
- Ability to alter the the communication network to optimize system performance

The resulting interface framework then was contributed by HP in the International Electrotechnical Commission Technical Committee 66, Working Group 3 (IEC/TC66/WG3) as starting proposal. By September, 1974, a draft document of the HP proposal was approved for balloting by the IEC, and in 1975, the Institute of Electrical and Electronics Engineers (IEEE) published their document IEEE-488/1975, "Digital Interface for Programmable Instrumentation" for facilitating the design, assembly and use of instrument systems, which became the base standard document for the HP-IB. An identical standard was published in 1976 as MC1.1 by the American National

Standards Insititute (ANSI).

The original standard from 1975 was revised primarily for editorial clarification and addendum in 1978 (IEEE-488/1978) and later in 1987 (IEEE-488/1987 or IEEE-488.1). Until that time, the IEEE-488 specified the mechanical, electrical, and basic protocol parameters, but did not standardize any device specific command or data format. That was the domain of the IEEE-488.2 standard, which was first published in 1987 and later revised in 1992. In 1980, the IEC published its own standard document IEC625-1, which was identical except for the 25-pin sub-D-connector (same as for RS232) instead of the original 24-contact Centronics-like connector. In fact, until today still the original Centronics-like connector is used, the sub-D-type never established.

There were later enhancements and extensions to the original IEEE-488 standard, such as the Standard Commands for Programmable Instrumentation (SCPI) or National Instruments' own implementation of a "High-Speed 488" (HS-488) with up to 8 MByte/sec throughput, which made it into the standard in 2003.

Since any revision of the HP-IB/GPIB/IEEE-488/IEC625 standard newer than IEEE-488/1987 or IEEE-488.1, respectively, is of no relevance for vintage HP-IB capable systems or peripherals, we do not further refer to those newer revisions in this tutorial, but concentrate on IEEE-488.1 which defines hardware, signalling and bus operation, but no high-level command syntax or data formats.

Please note: As with most official standards, the original standard documents are not available without charge (even not if they are more than 30 years old) but must be ordered directly from the standardization organisations. However there are some third party documents available (see the *downloads section* below).

Relevance of the HP-IB

For the HP 9845, the only important standard was the IEEE-488/1978, since the standard HP-IB interfaces for the HP 9845 - the 98034A and 98034B - implemented that revision of the IEEE-488 standard.

HP reported that in the year 1982, the IEEE-488 was published in nine languages, and supported by more than 250 manufacturers in at least 14 countries, to design more than 2,000 products. More than 45,000 copies (with an estimated revenue of more than 10 million dollars for the standardization organisations) of the standard had been distributed in 1982. Until today there have been designed more than 5,000 different HP-IB/GPIB devices. So, why was it so successful?

The HP-IB had some unique features, even compared to contemporary solutions such as the USB or Firewire:

- **The HP-IB was highly efficient:** Any device connected to the bus could send data to any number of other devices in parallel. Not just as it is with Ethernet broadcasting to any network adapter on the bus, but with full handshake and flow control capability, hence automatically adjusting the transmission speed to the slowest device in the receiver group. The whole system of all devices connected to the HP-IB could be reset by just a single signal (Interface Clear or **IFC**) sent by the system controller.
- **The HP-IB was responsive:** Sending a command to a device or starting a data transfer is just a matter of microseconds, which is faster than any other peripheral connection even today (including USB and Ethernet).
- **The HP-IB was fast:** It applied a byte-serial and bit-parallel scheme for up to 1 MByte/sec or 8 MBit/sec transmission speed (serial connections at the time in general were limited to 9,600 Bit/sec). Typical throughput however was between 250 kByte/sec and 500 kByte/sec.
- **The HP-IB was flexible:** Any device could take control of the bus at any time, and networks could be set up either as daisy chain, or as star configuration, or as any mixture of both. Asynchronous events could be actively signalled by any of the devices (e.g. when requiring service or for indicating special status) at any time (as Service Request or **SRQ**), and device status could be polled either serially (Serial Poll) or in parallel (Parallel Poll) by the controller-in-charge.
- **The HP-IB was versatile:** Not only measuring devices can be connected, but also printers, plotters, mass storage devices or human interface devices / controllers. It could even be used as network for setting up peer-to-peer or client-server networks. Also two 'dumb' devices can be connected if required without any need for a controller. And there is no lower limit for transmission speed.
- **The HP-IB was reliable:** Within the physical limitations defined by the standard, the active low signalling at TTL level allowed easy implementation with open collector interface logic with low noise and high confidence (or tri-state drivers for maximum throughput). Parity checks could be peformed automatically on every bus control command. There is no hassle with bus terminators like it is with the later Small Computer System Interface (SCSI), which later replaced HP-IB as mass storage interface, but never as measuring equipment interface.

On the other side, the HP-IB lacked hot plugging (there was no indicator for being newly connected to the bus[1], and there also was no guarantee that an ongoing

transfer would not be interrupted by a new device), and it is recommended to keep all devices switched on. Also, the connector is (at least compared to USB, Ethernet or Firewire) the more clumsy edition. But it is stackable!

Actually, in the early days, the HP-IB bus showed to be perfectly suited not only for connecting instruments, but also for the mass storage devices such as hard disk drives, tape drives and floppy disk drives. There was no other connection standard with comparable speed, until the SCSI standard became mature. HP decided to consequently move all of their mass storage devices to the HP-IB standard and developed an impressing number of mass storage devices all providing HP-IB interfaces. If coupled with an HP-IB mass storage device, many HP-IB instruments were capable of logging their measuring data directly to floppy disk, tape or hard drive without the need for some additional controller. Around 1990 SCSI then took over the role as major standard mass storage interface, but the relevance for talking to instruments via HP-IB still remained.

At the time when detached keyboards, mice, trackballs and other human interface devices entered the scene, Hewlett-Packard developed another more suitable technology (with tiny SDL connectors) for attaching those devices, the Human Interface Link (HP-HIL).

Compared to other periperhal interconnect standards like the USB, Firewire or SCSI the HP-IB supports dynamic reconfiguration of the controller role (not bound to a dedicated controller adapter) and multicast communication for sending data to multiple receivers at the same time. So in principle, the HP-IB bus ist not only capable of peripheral communication, but also can be used as a network connecting hosts and peripherals. However this also shows the drawback of the HP-IB. In a special sense, it had been over-engineered since it provides functionality which is not needed just for data acquisition and even not for peripheral connectivity in general. HP once designed it as a data acqusition standard, which was quite successful, but features like dynamic reconfiguration and multicast made it sometimes more complex to handle than it was necessary.

[1]) This is only partially true. Actually, so-called service requests issued actively by a newly connected device combined with regular polls can be a method to implement hot plugging, which again would suggest the controller to perform a poll in order to identify the new device on the bus. However, conventions on hot plugging were not part of the IEEE-488/1978 standard.

Communication Model

The communicaton model of the HP-IB is that of an 1:n communication with three roles which can be dynamically assigned to any of the devices which are connected to the bus: One sender (the talker), an arbitrary number of receivers (the listeners), and a special device coordinating the communication (the controller). It works a bit like in a school classroom: The teacher is the controller who decides who may talk, and who should listen. Assignment is temporary and can change as needed any time. All assignments together are building the communication context.

Initially after power-up the controller role is assigned to a special device called the system controller. This special role is fixed and must be manually configured by the operator (e.g. by setting the proper switches on an HP-IB interface). There is exactly one system controller on the bus. However any device can be assigned being bus controller temporarily (controller-in-charge). Basically, the controller is responsible to assign the talker and listener roles to the devices.

Once the communication context has been set up by the controller, the talker is free to send data to all listeners on the bus. So normally, a device which needs to receive data waits for being assigned by the controller to listen, and the sender waits for being assigned to talk. Obviously, the listeners should be assigned prior to the talker so that they are ready when the talker starts talking. The transfer is then started by the talker until transmission of all data has been completed or the controller intercepts. The controller role can be assigned in conjunction with a talker or listener role, so that the controller can play both roles at the same time., i.e. assign itself and act as a talker or listener on his own. Normally, the controller is the host computer which controls data exchange with e.g. measuring devices and participates itself as either talker or listener (e.g. for data acquisition).

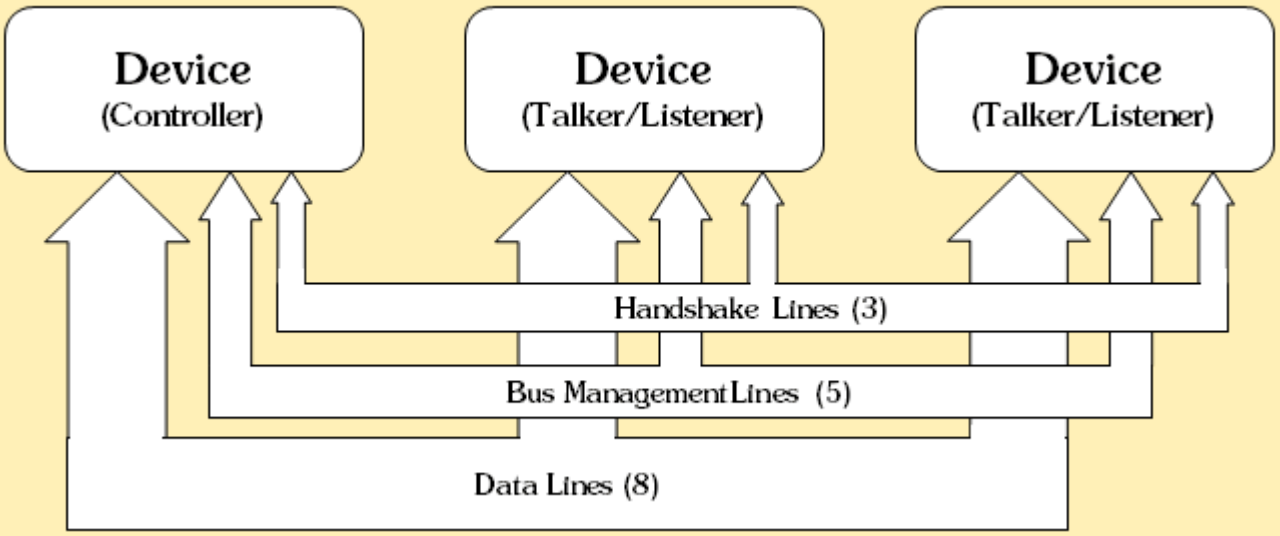
Any controller operation is done by bus commands which are sent by the controller to one or more devices. Some commands apply to all devices connected to the bus, whereas others (like assigning the talker and listener roles) just apply to selected devices. Devices (including the system controller) are identified by a unique (primary) address which is fixed and must be configured manually by the operator (e.g. with an address switch) so that no two devices share the same address. Addresses range from 0 to 30. Assigning talker and listener roles for selected addresses is called 'addressing'. On power-up, every device is in an idle state. Once being addressed as a talker or listener by the controller, it is in an addressed state, until the controller decides to remove the previously assigned roles by 'untalking' or 'unlistening', which then puts the devices back again into an idle state.

For just doing data transfers, no more but just the bus commands for setting up the communication context (i.e. addressing the devices as talker or listener and removing those roles) are necessary. The HP-IB standard however defines many more useful bus commands for different purposes such as resetting the whole bus or selected devices, for alerting, for transferring the controller role to another device or for sending trigger messages to all devices in parallel (and many more) which are explained later. But first let's have a look how data exchange is done in detail.

How It All Works

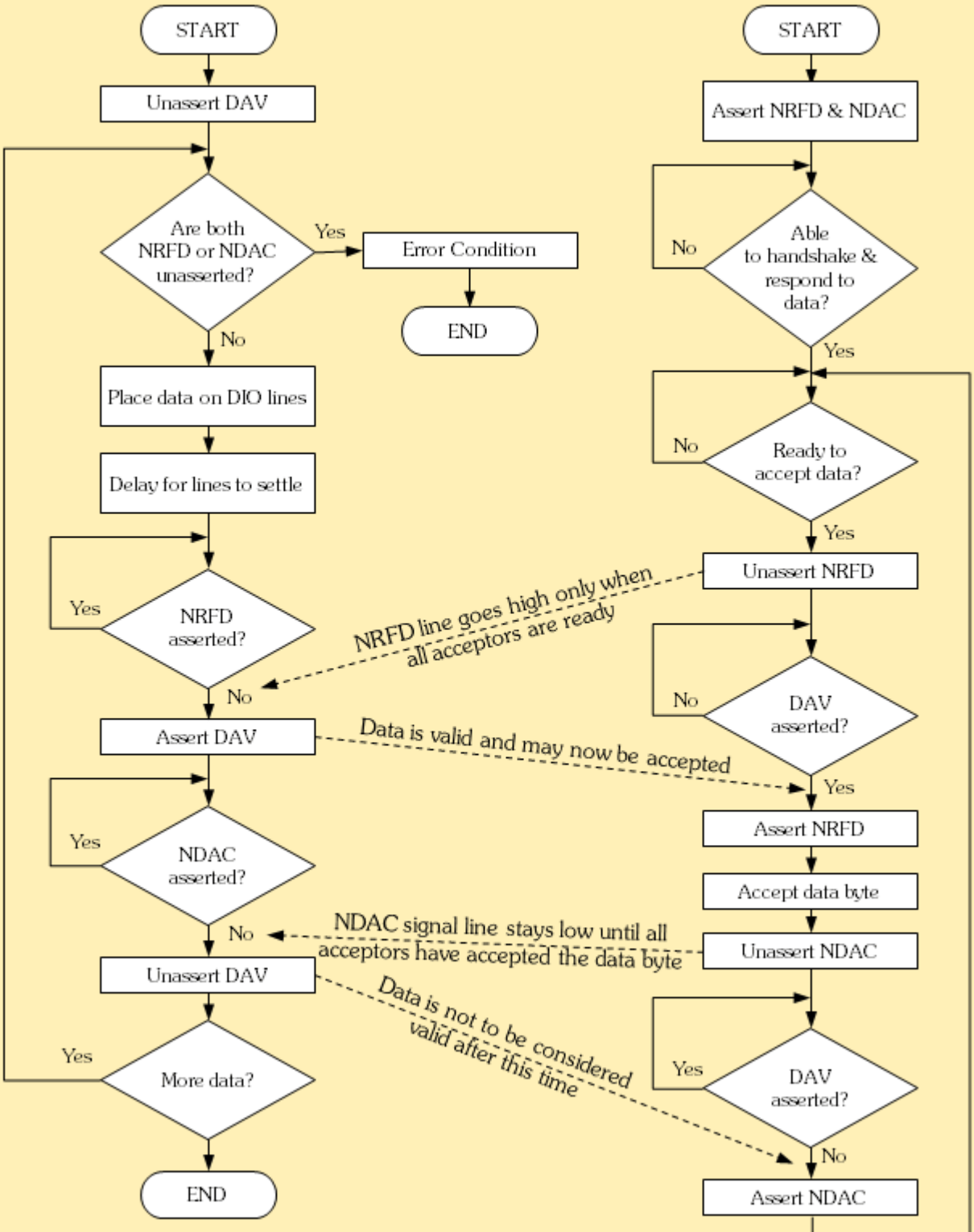
All transfers are done bit-parallel and byte-serial via the eight data lines (denoted DIO1 to DIO8). The HP-IB bus differentiates for any byte transmission between command mode and data mode, however in fact both are implemented just by the transmission of bytes (or characters) over the bus, with the only difference that one of the bus management lines, *Attention* or **ATN**, is asserted (=active low) during the transfer of commands, and unasserted (=inactive high) for data.

The handshake for all transfers is realized with just three wires even for multicast transfers with 1:n handshake. This handshake was somewhat simple but ingenious and protected by HP's own patent. Each byte is transferred during an atomic single-byte transfer step controlled by the three-wire handshake (so-called source/acceptor handshake) both for commands and data over eight data lines in parallel.



HP-IB Functional Line Groups

The basic principle is that the two lines *Not Ready For Data* (**NRFD**) and *Not Data Accepted* (**NDAC**) are both driven by the acceptor devices and wired OR'ed for all acceptors, whereas a third line *Data Valid* (**DAV**) is driven by the source device. For sending a byte, the source waits until all acceptor devices have unasserted **NRFD** (set to inactive high signalling they are ready for data), configures each of the data lines for the byte to transmit, and then asserts **DAV** by setting it active low. All acceptors keep track of this signal, and take the byte from the bus once **DAV** has been asserted by the source. Finally, all acceptors acknowledge that they have successfully read the byte by unasserting **NDAC**. Again, due to the wired OR, the **NDAC** line goes up once the last device signals that the byte has been read. So the slowest acceptor finally dictates the duration of each byte transmission cycle.



Three-Wire-Handshake Between Source and Acceptor

During data transfers, the role of the source is identical to that of a talker and the role of the acceptor is identical to that of a listener. For sending commands, the controller acts as source, and the device which receives the command acts as acceptor. So the roles of source/acceptor and talker/listener are closely related, but finally describe actions on different levels: source/acceptor roles are valid for low-level atomic single-byte handshake, whereas listener/talker roles are part of the high-level, longer-lasting communication context.

Actually, all transfers rely on just those eight data lines, the three handshake lines (**NRFD**, **DAV** and **NDAC**) and one bus management line (**ATN**) as command/data separator. All byte transfers are solely performed by those 12 lines.

The remaining four lines are dedicated for additional bus management purposes: *Interface Clear* (**IFC**) resets all devices into a known state and transfers the controller role back to the system controller, *Service Request* (**SRQ**) allerts the controller to a need for more communication, the *Remote Enable* (**REN**) enables devices to respond to a remote program control, and *End Or Identify* (**EOI**) is used for dual purpose: With **ATN** unasserted it indicates the last byte in multibyte data transfer, whereas with **ATN** asserted it requests all devices to assert their status bit on the data lines (Parallel Poll) as response. Those four lines provide additional means for bus management, however they are not required for just transferring bytes between devices. For understanding data transfer over the HP-IB, this is all we need to know. However the HP-IB can do more.

HP-IB Commands

Any type of control action on the HP-IB bus is called a 'command'. What makes the HP-IB bus a bit complicated is that there are four different types of commands: Single- or uni-line commands, universal multiline commands, addressed commands and secondary commands. Universal multiline commands, addressed commands and secondary commands all use more than one bus line, and therefore transport so-called multiline messages.

This needs a bit more explanation. In general, the number of bus lines should be as low as possible. Any additional line means more wires and more bus drivers. Therefore, HP-IB commands in general are just bytes put on the bus with the handshake as described above, but with **ATN** asserted. The byte value determines which command is issued. Universal multiline commands and addressed commands are both working that way. The only difference is that universal multiline commands do not require a communication context set up before (they simply apply to all devices on the bus), wheras the addressed commands will only be recognized by devices which have been configured as listeners or talker before (depending on the command). Commands for unaddressing devices as talker or listeners (putting devices back into idle state) are examples for universal multiline commands. Asserting any of the bus management lines is called 'single- or uni-line command', and they always apply immediately to all devices which are connected to the bus without any handshake, and therefore can be asserted at any time, even during a byte transfer.

Now all the command types explained above have one thing in common: They are all well defined within the IEEE-488 standard, and every command has its meaning. But the HP-IB standard also allows device-specific commands, which have a meaning for a

certain device only, which again is defined by the manufacturer of the device and not within the standard. This type of command is called a 'secondary command', since it in general follows a universal multiline command holding the primary address (such as TAD or LAD). One of the possible uses of a secondary command is to select (=address) a subsystem within a device, so the secondary command is also called a 'secondary address'. As with the primary address, the range for secondary address is 0 to 30. The combination of a primary address with a secondary address is also called "extended addressing" since it allows up to 31 x 31 = 961 entities to get identfied.

Basically, by using a byte as command information, up to 256 different commands are possible. Practically, the most significant bit of a command is reserved (used for parity), and the next two bits are used to identify addressing commands (talker, listener, secondary), so there are in fact up to 32 commands left in addition to talker and listener adressing & unadressing. The following tables show how the bus management lines and command byte codes are assigned within the HP-IB standard.

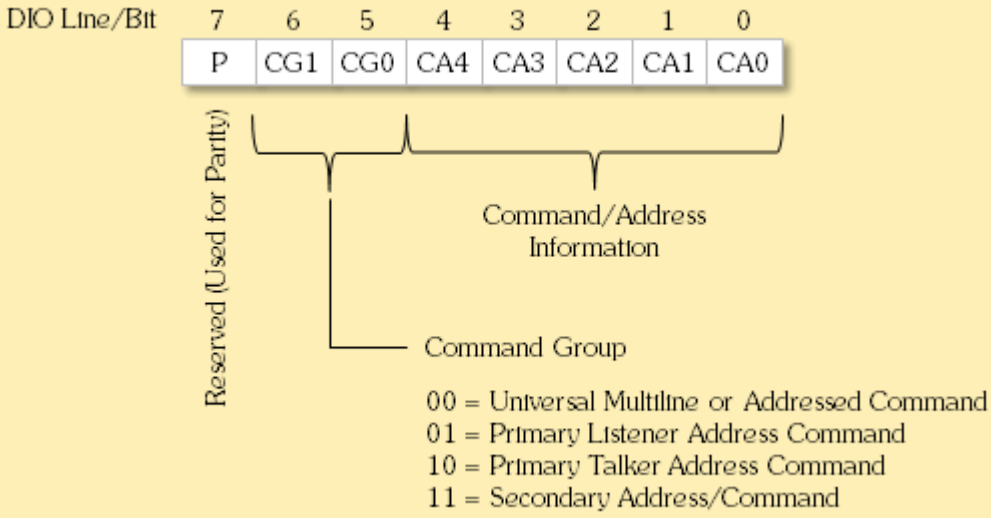
So as a summary, every device on an HP-IB bus has its own unique primary address, which is between 0 and 30. If so-called extended addressing is used, every device can keep its own set of secondary addresses (again between 0 and 30) for up to 31 subunits. For fully addressing a subunit, the controller must use a combination of the primary and the secondary address.

Note: Please don't confuse this extended addressing with the fact, that interfaces may also allow more than one single primary address. In that case, the interface acts towards the bus simply a single device which can be selected or configured with more than one primary address. If a device uses two primary addresses the first is normally called the *major address*, and the other address is called the *minor address*.

Single-/uni-line messages (apply to all devices immediately):

Mnemonic	Full Name	Description
ATN	Attention	Controls whether the bus in in command mode (**ATN** active low) or in data mode (**ATN** inactive high)
IFC	Interface Clear	Initializes the interface to an idle state (no activity on the bus)
SRQ	Service Request	Alerts the controller to a need for communication
REN	Remote Enable	Enables all devices to respond to remote program control when addressed to listen
EOI	End or Identify	Indicates the last data byte of a multibyte sequence, also used with **ATN** asserted simultaneously to conduct a Parallel Poll (cause all devices to assert their status bit)

All following multiline messages (address commands, universal multiline commands and addressed commands) use the DIO lines as shown below:



HP-IB Command Structure for Multiline Messages

Address commands:

Mnemonic	Full Name	Decimal Code	Hex Code	Description
MLA	My Listen Address	x + 32	x + 20	LAD matching the own primary address. Used for example by the controller to address itself as listener (with x the current primary address of the controller)
MTA	My Talk Address	x + 64	x + 40	TAD matching the own primary address. Used for example by the controller to address itself as talker (with x the current primary address of the controller)
LAD x	Device Listen Address	32-62	20-3E	Group of commands used to address the device with address x = code - 32 as listener
UNL	Unlisten	63	3F	Used to put all devices currently addressed as listeners back into idle state (there is no way to unlisten just one listener separately from the others)
TAD x	Device Talk Address	64-94	40-5E	Group of commands used to address the device with address x = code - 64 as talker, at the same time unaddresses the previous talker (there can be just one single talker at a time)

UNT	Untalk	95	5F	Used to put the device currently addressed as talker back into idle state
SAD x	Secondary Device Address	96-126	60-7E	Group of commands used to do second level addressing (device specific, x = code - 96)

Note that the UNT command is actually a TAD 31, also the UNL command is equivalent to LAD 31.

Universal multiline commands (apply to all devices):

Mnemonic	Full Name	Decimal Code	Hex Code	Description
LLO	Local Lockout	17	11	Disables a particular front-panel or rear-panel local reset or return-to-local control (use **REN** to re-enable the control)
DCL	(Universal) Device Clear	20	14	Causes all devices to return to a defined, device-dependent state
PPU	Parallel Poll Unconfigure	21	15	Resets all devices which are in a Parallel Poll state back into idle state (not responding to a Parallel Poll)
SPE	Serial Poll Enable	24	18	Establishes Serial Poll mode for all devices on the bus. Once addressed to talk, each device will return a single status byte.
SPD	Serial Poll Disable	25	19	Terminates Serial Poll mode, so that no device responds on being addressed to talk with its status byte any more.

Addressed commands (apply to listeners/talkers only):

Mnemonic	Full Name	Decimal Code	Hex Code	Description
GTL	Go to Local	1	1	Causes all devices currently addressed as listeners to return to local panel control (exit remote state)
SDC	Selected Device Clear	4	4	Causes all devices which have been previously addressed to listen to return to a defined, device-dependent state
PPC	Parallel	5	5	Causes the devices being addressed as

	Poll Configure			listeners to be configured to a Parallel Poll response specified by the following secondary (secondary codes 96-111) or to disable Parallel Poll response (secondary code 112)
GET	Group Execute Trigger	8	8	Initiates a pre-programmed, device dependent action simultaneously for any device which has been previously addressed to listen (e.g. start with measuring)
TCT	Take Control	9	9	Causes the device previously addressed to talk to begin operating as a bus controller (controller-in-charge)
PPE	Parallel Poll Enable	96-111	60-6F	Group of secondary commands sent after a Parallel Poll Configure (PPC) command has been received for configuring the Parallel Poll response of the addressed device - the three least significant code bits specify the response line and bit 3 defines the condition of response (sense bit)
PPD	Parallel Poll Disable	112	70	Secondary command for disabling the Parallel Poll response of the addressed device after a Parallel Poll Configure (PPC) command has been received

To be complete with the mnemonics, the acronym DAB is used to denote transfer of a data byte (in contrast to a command byte).

The actions described above are all part of the IEEE-488 standard. The standard in fact does not require all HP-IB devices to implement all of the HP-IB functionality, but rather defines capability groups, which may or may not be supported by a device. Those capabilities are in general described in the device's manual. Note that there is no error condition defined for the HP-IB bus, so if a device does not support a special capability (e.g. responding to a Parallel Poll), it simply does nothing.

Notation

The sequence of a bus transaction is normally described as sequence of mnemonics. A typical 13-byte data transfer of the string "HELLO WORLD" terminated with Carriage Return & Line Feed from the controller (we assume primary address 21) to a device (assumed at primary address 1) shows like this:

Mnemonic	Hex Code[1]	Comment
UNL	3F	// puts any current listener back into idle state

`LAD 1`	33	// addresses the device at primary address 1 as listener
`MTA`	55	// the controller addresses itself as talker (address 21 = hex 15)
`DAB 'H'`	48	// the controller sends the data string to the device (ASCII codes)
`DAB 'E'`	45	
`DAB 'L'`	4C	
`DAB 'L'`	4C	
`DAB 'O'`	4F	
`DAB ' '`	20	
`DAB 'W'`	57	
`DAB 'O'`	4F	
`DAB 'R'`	52	
`DAB 'L'`	4C	
`DAB 'D'`	44	
`DAB 0D`	0D	// send CR character (hex 0D)
`DAB 0A`	0A	// send LF character (hex 0A)
`UNT`	5F	// the controller untalks itself
`UNL`	3F	// unaddresses all listeners on the bus

[1]state of the data lines without parity

Note that the single-/uni-line commands are not mentioned, the only line which is of relevance in this case is the Attention (**ATN**) line, which is automatically asserted during every command and unasserted during the data transmission (the series of DAB). Also note that parity is not used on the commands (if parity is enabled, HP-IB is normally using odd parity, which means that the MSB will be set if the sum of all bits without the parity bit is even, but parity in fact is not part of the IEEE-488 standard).

There is no convention how to denote changes on bus control lines. Most descriptive propbably is e.g. writing "**EOI**↑" for "**EOI** asserted" and "**EOI**↓" for "**EOI** unasserted". Bus analyzers often show the state of all bus management, handshake and data lines, such as for sending UNT to the bus:

```
              N N
A I S R E  D R D
T F R E O  A F A  DATA
N C Q N I  V D C  LINES  EVENT

0 0 0 1 0  0 1 1  00              // bus is initially in idle
state[1]
1 0 0 1 0  0 1 1  00     ATN↑     // controller asserts ATN
signalling next byte will be a command
1 0 0 1 0  0 1 1  5F              // sender places the byte to
be transferred on the data lines
1 0 0 1 0  0 0 1  5F              // receiver unasserts NRFD
when it is ready to receive a byte
1 0 0 1 0  1 0 1  5F              // sender asserts DAV
signalling the byte on the data lines is now valid
1 0 0 1 0  1 1 1  5F              // receiver re-asserts NRFD
1 0 0 1 0  1 1 0  5F     UNT      // receiver acknowledges by
unasserting NDAC
1 0 0 1 0  0 1 0  5F              // sender unasserts DAV
1 0 0 1 0  0 1 1  5F              // receiver asserts NDAC
0 0 0 1 0  0 1 1  00     ATN↓     // controller unasserts ATN if
no more commands will be sent
```

[1]Actually it is not defined whether **ATN** should be asserted or unasserted if the bus is idle (neither command nor data transfer is intended), same applies to **NRFD** and **NDAC**. So some controllers are unasserting **ATN** after each transfer, and some don't. For the devices this should not matter. However once **ATN** is asserted, all devices are required to assert (pull low) their **NRFD** and **NDAC** lines immediately (within 200 nanoseconds) so that the three-wire handshake can start. **REN** is normally asserted permanently so that all devices allow remote control all the time.

Configuring/Unconfiguring the Bus

Before any action on the bus can take place, the roles have to be assigned to all devices which shall participate in that action. This is done by the current controller (controller-in-charge). In most cases the controller has just to address all receivers as listeners, and the sender as talker. Then the desired action can be performed (e.g. a data transmission). That configuration will be valid until the bus gets unconfigured by putting all devices back into idle state with Untalk (UNT) and Unlisten (UNL).

Unless the roles are changing (e.g. for sending a reply) there is no need to change the bus configuration.

Transferring Data

A typical sequence for data transfers is prepared by the controller by the following steps:

1. Send UNL command to the bus - puts any device currently listening into idle state
2. Send LADx command to the bus for each device which shall participate in the data transfer as receiver (x is the device's primary address) - puts all receivers

into listen state
3. Send TLKx command to the bus for the sender - puts the sender into talk state

Optionally, any of the primary addressing commands (LADx/TADx) can be followed by a Secondary Command (SADx), which normally does an extended addressing for the device (i.e. addresses a sub-component of the device).

We can assume that all devices keep track of the state of their HP-IB interface, so that all listening devices will get ready to receive data for storing it into their buffer, and the talking device recognizes that it now may start sending data to the bus. the three-wire handshake ensures that the sender puts the next byte on the bus not before the last device has acknowledged taking the current byte from the bus.

Termination of Transfers

There are several methods for terminating the transfer. The sender itself can send a previously defined End of String (EOS) byte code to signal to all receivers that the transfer is complete. No controller action is required for this termination method, however the EOS code must be previously configured for all devices, both sender and listeners, and the EOS code obviously cannot be used as data. This method is especially suitable if only 7-bit ASCII characters are transmitted as data, so that EOS can be any code > 127. It even works if the most significant bit is used as parity bit during 7-bit ASCII character transfers as long as the EOS code is unique. It is also not unsual to use the <CR> character (carriage return) or the <NULL> character as EOS, however the sender and the receiver both have to agree on it.

The talker can also terminate the data transfer by asserting **EOI** during the last byte, which actually is the most common method. Also, the transmission can be terminated any time by the controller sending UNT and UNL to the bus. Basically, it would be even possible to simply assert **ATN** (which switches the bus from data to command mode and can be noticed by all interfaces at once), however there is no general convention for it, so the talker and the listeners may or may not react in the expected way.

Termination with **EOI** has the advantage, that the transfer can be continued without the need for re-addressing talker and listener(s), since the current addressing will not be changed. Sending an additional UNL signals that no further data will follow (all listeners will be put into idle state). However, termination with **EOI** requires to perform at least a single data handshake with **EOI** asserted. Asserting **EOI** alone has no effect at all. So for correct handling of a zero-byte-transfer, the transfer should normally be terminated with an UNT to the sender and UNL to the receivers in any case. Alternatively, the sender can send an extra-byte with **EOI** asserted and the end of each transfer, but the receivers must explicitly know that they have to discard this extra byte.

Also, any transfer can be intercepted by the system controller by asserting Interface Clear (**IFC**), which resets all devices to their idle state and also transfers the controller role back to the system controller.

For an example see the Notation section above. Carriage Return character <CR> is used in this example as EOS code.

Conducting a Serial Poll

Serial Poll is the standard method for a controller to query the status of one or more devices. It is performed in several steps:

1. The controller addresses itself to listen
2. The controller sends Serial Poll Enable (SPE) to the bus - this puts all devices into Serial Poll response state
3. The controller addresses one device to talk
4. The addressed device puts its configured Serial Poll response (normally representing the device's status) on the bus
5. The controller reads the byte from the bus (with three-wire handshake) and continues with step (3) until all device's status has been queried
6. The controller sends Serial Poll Disable (SPD), UNT and UNL to the bus - this sets both the controller and all devices back into idle state

Note that in general a status byte can be read just once from each device, before reading the status byte again the controller should shortly assert **ATN**.

Also note that bit 6 of the Serial Poll response is reserved and cannot be used as status information (it should be ignored by the controller). So in fact 128 status codes can be used by the responding device.

Sending a Serial Poll response is normally an automatic function of the device's interface. The device itself has just to configure the status byte response once, which is then sent automatically by the interface to the bus on every Serial Poll without further notice.

Here is an example for a Serial Poll sequence (assuming the controller at primary address 21, and devices at addresses 0 and 1 with both status byte 00):

Mnemonic	Hex Code	Comment
`UNL`	3F	// puts any current listener back into idle state
`MLA`	35	// controller addresses itself as listener
`SPE`	18	// set all devices into Serial Poll response state
`TAD 0`	40	// addresses first device as talker
`DAB 00`	00	// status byte of the first device
`TAD 1`	41	// addresses second device as talker (automatically untalks first device)
`DAB 00`	00	// status byte of second device
`SPD`	19	// disable serial poll state for all devices

UNT	5F	// the controller untalks itself
UNL	3F	// unaddresses all listeners on the bus

Conducting a Parallel Poll

Serial Poll can be a lengthy procedure, especially if many devices are involved. With a Parallel Poll, the controller can query up to eight devices quite efficiently within just two steps:

1. The controller asserts **ATN** and **EOI** simultaneously - this puts all devices into parallel response mode, so that depending on its internal status, each device pulls just the DIO line low which has been assigned to it and leaves all other lines untouched
2. The controller recognizes the states of the data lines (no handshake is involved) and unasserts **ATN** and **EOI** - this puts all devices back into their previous state

Since there are no more than eight data lines, and all device should be ideally assigned to separate data lines, no more than eight devices may respond to a Parallel Poll in order to get an unambigous response. However, devices which are configured to the same Parallel Poll response are simply OR'ed.

Also, because there is just the possibility to either assert the response line assigned to a device or leave it as is, the status information delivered by a response to a Parallel Poll is binary (in contrast to a Serial Poll, which features up to 128 different status codes).

As with the Serial Poll, sending a Serial Poll response is normally an automatic function of the device's interface. The response has to be configured once (either by the controller by Parallel Poll Configure + Parallel Poll Enable, or by the operator setting the proper switches on the device), which is then sent automatically by the interface to the bus on every Parallel Poll without further notice.

Devices must be configured for responding by asserting or unasserting one of the data lines during a Parallel Poll. This is either done by operator configuration by a switch on the device, or by dynamic configuration by the bus controller. The latter is peformed with the following steps:

1. The controller unaddresses all listeners and addresses the device which should be configured as listener
2. The controller sends a Parall Poll Configure (PPC) to the bus - this sets the addressed listener into configuration state
3. The controller sends a Parallel Poll Enable (PPE) secondary command to the bus - this configures the response of the addressed device on Parallel Polls
4. The controller unadresses the listener, putting it back into idle state

Here is an example for remotely configuring three devices to assert data lines 0, 1 and 2, respectively on Parallel Poll in case the internal status of each device corresponds to sense bit 0 (also assuming the controller at primary address 21, and devices at addresses 0, 1 and 2):

Mnemonic	Hex Code	Comment
UNL	3F	// puts any current listener back into idle state
MTA	55	// controller addresses itself as talker
LAD 0	20	// addresses first device as listener
PPC	05	// puts the first device into parallel poll configuration state
PPE 0	60	// configures the first device to respond to a parellel poll by asserting data line 0
UNL	3F	// unaddresses first device
LAD 1	21	// addresses second device as listener
PPC	05	// puts the second device into parallel poll configuration state
PPE 1	61	// configures the second device to respond to a parellel poll by asserting data line 1
UNL	3F	// unaddresses second device
LAD 2	22	// addresses third device as listener
PPC	05	// puts the third device into parallel poll configuration state
PPE 2	62	// configures the third device to respond to a parellel poll by asserting data line 2
UNL	3F	// unaddresses third device
UNT	5F	// the controller untalks itself

And here is an example for conducting a Parallel Poll (assuming there are three devices which assert the data lines 0, 1 and 2, respectively on Parallel Poll):

```
    A I S R E  D R D
    T F R E O  A F A  DATA
    N C Q N I  V D C  LINES  EVENT

    0 0 0 1 0  0 1 1  00              // bus is initially in idle
    state
    1 0 0 1 1  0 1 1  00     ATN↑EOI↑ // controller asserts ATN and
    EOI simultaneously
    1 0 0 1 1  0 1 1  07              // all devices assert/unassert
    their status line
    1 0 0 1 0  0 1 1  07     ATN↓EOI↓ // controller unasserts ATN
```

```
    and EOI after recognizing data lines
    1 0 0 1 0  0 1 1  00              // all devices leave parallel
    poll state
```

Performing Service Requests

It can happen that a device requires attention from the controller. Typical example is when a printer runs out of paper, and it wants to signal that operator action is required. This is done by the following steps:

1. The device which wants to alert the controller configures its current status it wants to send to the controller as its Serial Poll response byte
2. The device asserts the **SRQ** line - this signals to the controller that there is a service request
3. The controller performs a Serial Poll to identify which device needs service and to receive the status response of that device

The device which has initiated a Service Request is required to unassert the **SRQ** line once it has been addressed to talk during the Serial Poll.

Normally, steps (1) and (2) are both done within a single step, because bit 6 of the serial reponse byte is by convention directly connected to the **SRQ** line, which means that if this bit 6 of the Serial Poll response is set, the **SRQ** line will be asserted. For that reason, there are only 128 different status codes available for the device.

The device gets informed that its status byte has been successfully read by the controller through the three-wire handshake during the transmission of the status byte. If that handshake succeeds, the device knows that its service request is no longer pending.

Of course it can happen that more than one device need service at the same time. Since all devices share the **SRQ** line, it is good practice for the device not to issue a Service Request as long as the **SRQ** line is asserted because of another ongoing Service Request, but rather wait until the **SRQ** line gets unasserted again. However, if the controller does a Serial Poll on all devices as result of a Service Request, it can collect and serve all pending Service Requests even if more than one device have placed a Service Request on the bus at the same time. Which again is good practice for the implementation of the controller. Since the **SRQ** line is wired OR'ed for all devices, it will not be unasserted before the last device's Service Request has been processed by a Serial Poll.

Resetting

The HP-IB model for devices (at least that of IEEE-488.1) actually does not make any assumptions about the whole device, but just about its interface through which it is connected to the HP-IB bus. So if we are talking about 'devices' we - from the perspective of the bus - just mean the device's interfaces towards the HP-IB bus system. For the term 'resetting', we cannot make any assumption about what the device as a whole actually does when a reset is sent over the bus, however it is defined what the device's interface should do.

There are three means for resetting interfaces and devices on the bus:

The Interface Clear (**IFC**) is conducted by asserting the **IFC** line for at least 100 milliseconds, which is reserved to the system controller. The defined effect is that all interfaces connected to the bus return to their idle state (putting the whole bus into a quiescent state), and the controller role is returned to the system controller (if there is another controller active).

The (Universal) Device Clear (DCL) can be sent by any active controller and is recognized by all devices, however it is a message to the device rather than to its interface. So it is left to the device how to react on a Universal Device Clear. There is no assumption (at least not in IEEE-488.1) what the device should do (it can even ignore the DCL).

The Selected Device Clear (SDC) also can be sent by any active controller, and has the same effect as the (Universal) Device Clear, but is recognized only by the devices which have been addressed as a listener before.

Remote Control

The HP-IB assumes that a device is either in local or in remote control state. In local control state, the front or rear panel controls are activated and the device does ignore any command from HP-IB, whereas in remote control mode commands from HP-IB are processed but the front or rear panel controls are disabled.

If there is a switch at the device which can be used to set the device back into local control mode, that switch can be remotely disabled for all devices simultaneously by sending the Local Lockout (LLO) command to the bus.

The common method to put all devices into local control state is asserting both Attention (**ATN**) and Remote Enable (**REN**) simultaneously. This also clears a previous local lockout condition if it is still in effect. This operation requires being the system controller.

If just selected devices shall be set into local control state, those devices must first be addressed as listeners and then the controller sends a Go to Local (GTL) command to the bus.

Putting selected devices into remote control state is done by unasserting **ATN** and asserting **REN** simultenously. Any device which is addressed as listener after this will switch into remote control mode. This operation requires being the system controller.

Triggering

Devices can be caused to perform some device specific action simultaneously (e.g. start with data acquisition) by using Group Execute Trigger (GET). This command applies to all devices which have been previously addressed as listeners.

Passing Control

After initialization the system controller is the active controller. However any active controller can pass over the controller role to any other device which is capable of the controller role with two steps:

1. Address the device which shall become the new controller-in-charge as talker
2. Send the TCT command to the bus

The device holds the controller role until it passes it over to another controller capable device, or until the system controller asserts the Interface Clear (**IFC**) line, which makes the system controller the active controller.

The Thing With the Identify

The commands described above are all part of the IEEE-488 standard. Now at some point of time, Hewlett-Packard's engineers decided to implement another command, the the so-called Identify, which never found its way into the standard and as such is totally proprietary. It provides means for controller to efficiently query not only what primary addresses are in use, but also which type of device is connected. It does so by first sending an UNT, and then a secondary address. If the secondary address matches with its primary address, the device should place itself into talker state and send a two-byte identification code to the controller. So the identify is very similar to the Serial Poll and the Parallel Poll, but always identifies itself with the same two-byte code.

Here is an example (assuming the device has primary address 1 and identification code hex 0081):

Mnemonic	Hex Code	Comment
`UNT`	3F	// unaddresses an existing talker
`SAD 1`	61	// secondary address 1 (matches primary address)
`DAB 00`	00	// send identification code
`DAB 81`	81	
`UNT`	3F	// finish the sequence

In principle, the Identify is a quite nice feature, since the controller can quickly determine which devices are on the bus, and perform some kind of plug-and-play procedure. However many of the drivers which are implemented in the operating systems of HP's computers assume known identification codes, or in other words refuse to work with unknown devices. So any device which ignores the Identify command, or which sends the wrong code back to the host, is strictly out of the game.

Now setting itself into talker state without primary addressing is - although unusual - part of the standard. The device can send itself a 'talk only' local message (ton), which should force the device into talker state. The point is that the device has to properly detect the Identify command in order to perform the required steps. HP's own HP-IB controller chips PHI, CHI and ABI all had a detection circuit implemented, which automatically sent the proper response without the notice of the device's processor (just as it is with Serial Poll and Parallel Poll). But all other GPIB chips, especially those of other manufacturers, didn't have that circuit since it is not part of the standard.

As a consequence, that function has to be emulated. It shows that there is a feature in the NEC µPD7210 that it accepts a primary address 31 for itself. Remembering that normally primary addresses range from 0 to 30, sending a TAD 31 addressing command would be the same as sending an UNT. So, if configured to address 31, a µPD7210 recognizes an UNT as TAD 31. In combination with extended addressing and using the own primary address as secondary address, the µPD7210 now can be configured to get addressed to talk when an UNT is sent with a following secondary command which secondary address matches the device's primary address. Even more, the µPD7210 allows using two primary addresses in parallel (major and minor address), so that one primary address can be used to perform normal operation, and the other to respond on Identify commands.

Finally, the µPD7210 has a nice *Hold-off on Secondary* feature built-in, which allows the device to check in a comfortable way whether there is a secondary which matches its own primary address. However, all this works only because the µPD7210 allows being configured to an invalid primary address 31, which actually is a flaw. The trick should work as well with an Intel 8291A because it is widely register compatible to a µPD7210, and it also applies to ASICs running in µPD7210 mode.

Unfortunately, no other GPIB chip provides that 'feature'. So it would be up to the processor, not the GPIB chip, to trace whether there is an UNT with following secondary command in real time, and this is practically impossible because of timing issues, especially under a non-deterministic multitasking environment like MS Windows. Which again prohibits proper emulation of HP devices which are using that Identify command with any other chip but a µPD7210. Another workaround would be to use the DAC hold-off (see the next paragraph) for stepping through the UNT and secondary command, but exactly this feature is not implemented by the TMS9914 (it does not provide DAC hold-off with **ATN** asserted without being addressed as it would be the case for the UNT and secondary command). Few of HP's own devices initially used a TMS9914 chip with OEM controllers, but the processor had to keep up with the Identify, and those later were replaced by 8291A chips.

Hold-offs and Timeouts

We have learned that the three-wire handshake in theory waits for RFD and DAC get asserted by all devices as it is necessary when waiting for the slowest device on the bus.

Now actually processing data is not just a task of the interface, but of the whole device. Once the interface signals to the device that something is going on on the HP-IB (which within the device normally happens either by polling the interface from the device side or by interrupting the processor), the device has to take some processing action, which has to be performed in a secure and complete way and may involve actions that take some time (e.g. allocating buffers before taking the data from the bus), until - when finished - the device signals 'ready' to the bus.

Since the three-wire-handshake has two defined points where the source waits for all acceptors to get ready, there are two points where the device can perform a so-called hold-off. Hold-offs are not really part of the IEEE-488 standard, but rather utilize the fact that a handshake can be paused by just delaying the proper acknowledge:

- RFD hold-off postpones unasserting the **NRFD** line during a three-wire handshake until the device is ready to accept the next byte of a transfer (either command or data).
- DAC hold-off postpones unasserting the **NDAC** line during a three-wire handshake until the device has finished processing the current byte of a transfer (either command or data).

Since both hold-offs in principle can be utilized on interface level both during a command or data phase, the acceptor device has a good control over the whole bus in order to keep both the source device off from performing faster than the acceptor device can handle. However, in reality, there are limits.

Now, just not completing a handshake can be both the result of an intentional hold-off and of an unintended fault condition, the source device needs to give up after a certain time cancelling the current action in order not to run into a hang-up condition. Also, an acceptor device may give up while waiting for data after a certain time.

Unfortunately, the necessary timeouts have not been defined in the standard, so it is finally a matter of which devices are connected to the bus and knowledge of which timeouts they use. If a source is too eager for sending data it can easly run into a situation where there is no acceptor ready. On the other side the source may already have given up when the acceptor starts waiting for data from the bus. So, much can go wrong. This especially becomes a problem with HP's HP-IB mass storage drivers, where in general tight timings are in use.

All other timing for the three-wire handshake is well defined, and must be implemented by all devices in order to comply to the standard.

Remark: Of course all devices which are in idle state (i.e. not addressed to either talk or listen) must not participate in the source/acceptor-handshake during a data transfer. However, bus debugging can be simplified by intentionally breaking that rule. By just pulling the **NRFD** or **NDAC** lines low, you can force stepping through data transfers even at human speed. As long as no device is running into timeouts. Some HP-IB analyzers offer this option.

Local Messages

All of the commands described so far are issued from the bus towards the interface, i.e. are maintained by the HP-IB controller. This type of command is also called a *remote message*. Now in fact every device's interface has two sides, one to the HP-IB and another to the device's internal logic (mostly the processor). Same applies to the controller HP-IB interface, which of course also has a connection to the internal logic behind.

The IEEE-488 standard defines a number of commands, which are sent from the processor to the HP-IB interface, thereby controlling the interface from the internal processing logic of the device or the host. Those commands are called *local messages*. It is up to the implementation of the interface how the local message reaches the interface (normally this is achieved with a special register which is written by the processor, however local messages may also be generated internally within the interface). Some local messages may be implemented both for setting and for clearing

a special condition. For example, the Individual Status message can be implemented as both ist (set Individual Status true) and ~ist (set Individual Status false).

For distinguishing against the remote messages, remote messages mnemonics by convention are in capital letters (such as UNT), whereas those for local messages are all lowercase.

The local messages defined by the standard are:

Mnemonic	Full Name	Description
gts	Go to standby	gts forces the controller-in-charge to become the standby controller and to unassert the **ATN** signal. No effect if not being the active controller.
ist	Individual status qualifier	ist will be compared to the sense bit received with the last Parallel Poll Enable (PPE) command during a Parallel Poll Request (PPR). If ist and sense bit match (either both are true or both are false), the interface generates a Parallel Poll response.
lon	Listen only	lon forces the listener function into the Listener Active State (LACS). This message should only be used if the interface is the active controller or there is no controller on the bus (e.g. for directly sending data from a instrument as a permanent talker to a printer as a permanent listener). Will be canceled by addressing commands like UNL or the ton message.
lpe	Local poll enable	lpe puts the interface into Parallel Poll Standby State (PPSS), which enables the interface to respond to Parallel Poll requests.
ltn	Listen	ltn forces the active controller into the Listener Addressed State (LADS), which is synonymous to being addressed as a listener. No effect if not being the active controller.
lun	Local unlisten	lon forces the Listener function into the Listener Idle State (LIDS) which is synonymous to being unlistened.
nba	New byte available	nba signals to the interface that a new byte is available for transfer via source handshake.
pon	Power on	pon places the interface into a defined state (initialization).
rdy	Ready	rdy enables the interface to receive a next data byte.

rpp	Request parallel poll	rpp forces the interface to send the Parall Poll command to all HP-IB devices in the system and to conduct a Parallel Poll. No effect if not being the active controller.
rsc	Request system control	rsc transfers active control back to the system controller (normally coupled with the sic message). No effect if not being the system controller.
rsv	Request service	rsv instructs the interface to enter the SRQS and assert the **SRQ** signal.
rtl	Return to local	rtl forces the interface to change from remote to local control, provided there is no previous Local Lockout (LLO) in effect.
sic	Send interface clear	sic forces the interface to assert the Interface Clear (**IFC**) signal (normally coupled with the rsc message, since this message is available to the system controller only).
sre	Send remote enable	sre instructs the interface to assert **REN** thus sending a Remote Enable command to the bus. No effect if not being the active system controller.
tca	Take control asynchronously	tca forces the interface to become the active controller and to assert the **ATN** signal immediately. If there is a source/acceptor handshake in progress, data loss may occur. No effect if not being the controller-in-charge.
tcs	Take control synchronously	tcs forces the interface to become the active controller and assert **ATN** when the interface performs an RFD (Ready for Data) hold-off — that is, the acceptor handshake function enters the Acceptor Not Ready State (ANRS), so no data will be corrupted. No effect if not being the controller-in-charge.
ton	Talker only	ton forces the talker function into the Talker Active State (TACS). This message should only be used if the interface is the active controller or there is no controller on the bus (e.g. for directly sending data from a instrument as a permanent talker to a printer as a permanent listener). Will be canceled by addressing commands like UNT or the lon message

Manufacturers of HP-IB interface hardware have added some more local messages which give access to special interface functions but which are not covered by the standard. In conjunction with interface hardware, local messages are also called *auxiliary commands*.

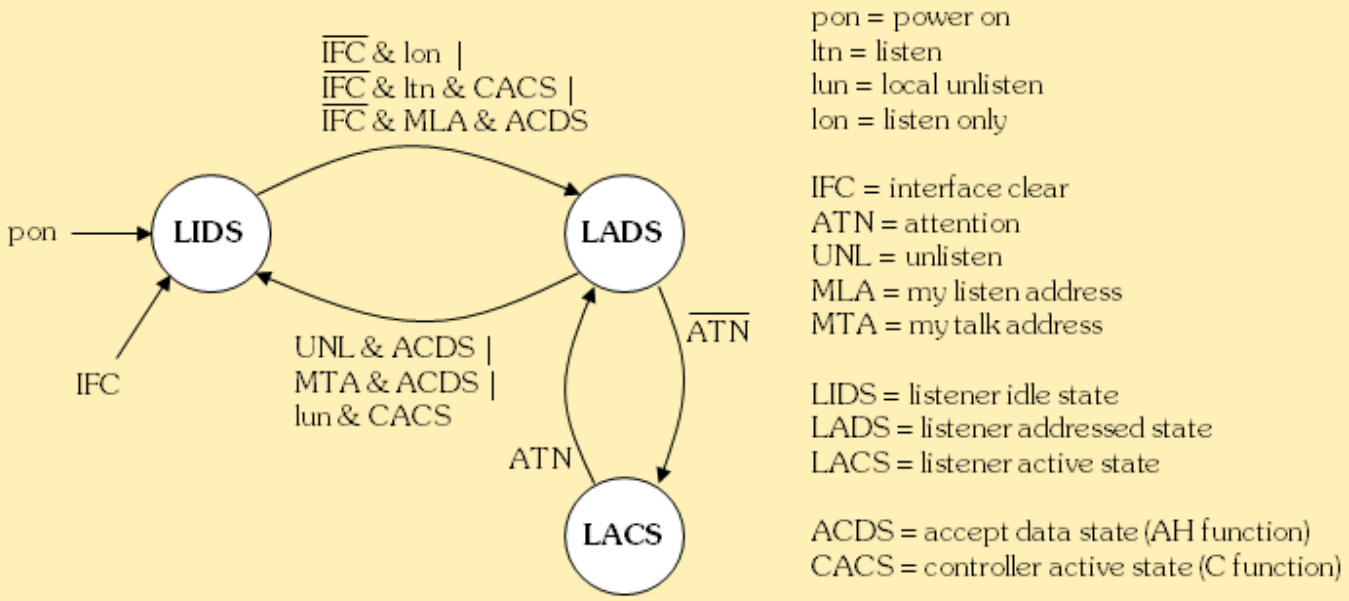
State Model

The IEEE-488 standard describes the behavior of an HP-IB/GPIB interface as a number of states with conditions for state transitions. For each function (such as talker, listener, Serial Poll, Parallel Poll etc.) there are idle states where the interface is being placed after power-on. State transitions happen either by bus events (so-called *remote messages*) or by internal events or by programming the interface from the host or device side e.g. by firmware (so-called *local messages*).

The functions defined by the standard (which are all represented by state diagrams) are:

1. Source Handshake (SH)
2. Acceptor Handshake (AH)
3. Talker (T)
4. Talker with Extended Addressing (TE)
5. Listener (L)
6. Listener with Extended Addressing (LE)
7. Service Request (SR)
8. Remote Local (RL)
9. Parallel Poll (PL)
10. Device Clear (DC)
11. Device Trigger (DT)
12. Controller (C)

An example state diagram for the Listener function would be:



Listener State Diagram

This example shows that after power-up and after any Interface Clear (**IFC**) the Listener function (L) first will be in the Listener Idle State (LIDS). In order to get addressed as a listener, the interface has to make a transition to the Listener Addressed State (LADS), which can be achieved by one of three actions: sending the lon message, sending the ltn message (works on active controller only) or being

addressed by an LAD command (which requires being already in Accept Data State within the Acceptor Handshake function AH). Of course **IFC** must be unasserted, otherwise the interface would never leave LIDS.

In LADS the interface knows it is a listener, however it will not be able to receive data before **ATN** gets unasserted, which puts the interface from LADS into the Listener Active State (LACS), where the interface can participate in the data handshake. When the controller asserts **ATN**, this is a signal to the interface that the next handshake will transport a command rather than data, which puts the interface back into LADS until **ATN** gets unasserted again, or the interface gets unlistened by either UNL, being addressed as a talker, or receiving the local lun message (again works on active controller only). To be more precise, also the local message ton will put the interface back from LADS to LIDS.

Note that when using extended addressing, there are variations since for being addressed also the secondary address must match, this is then covered by the Listener with Extended Addressing (LE) state diagram.

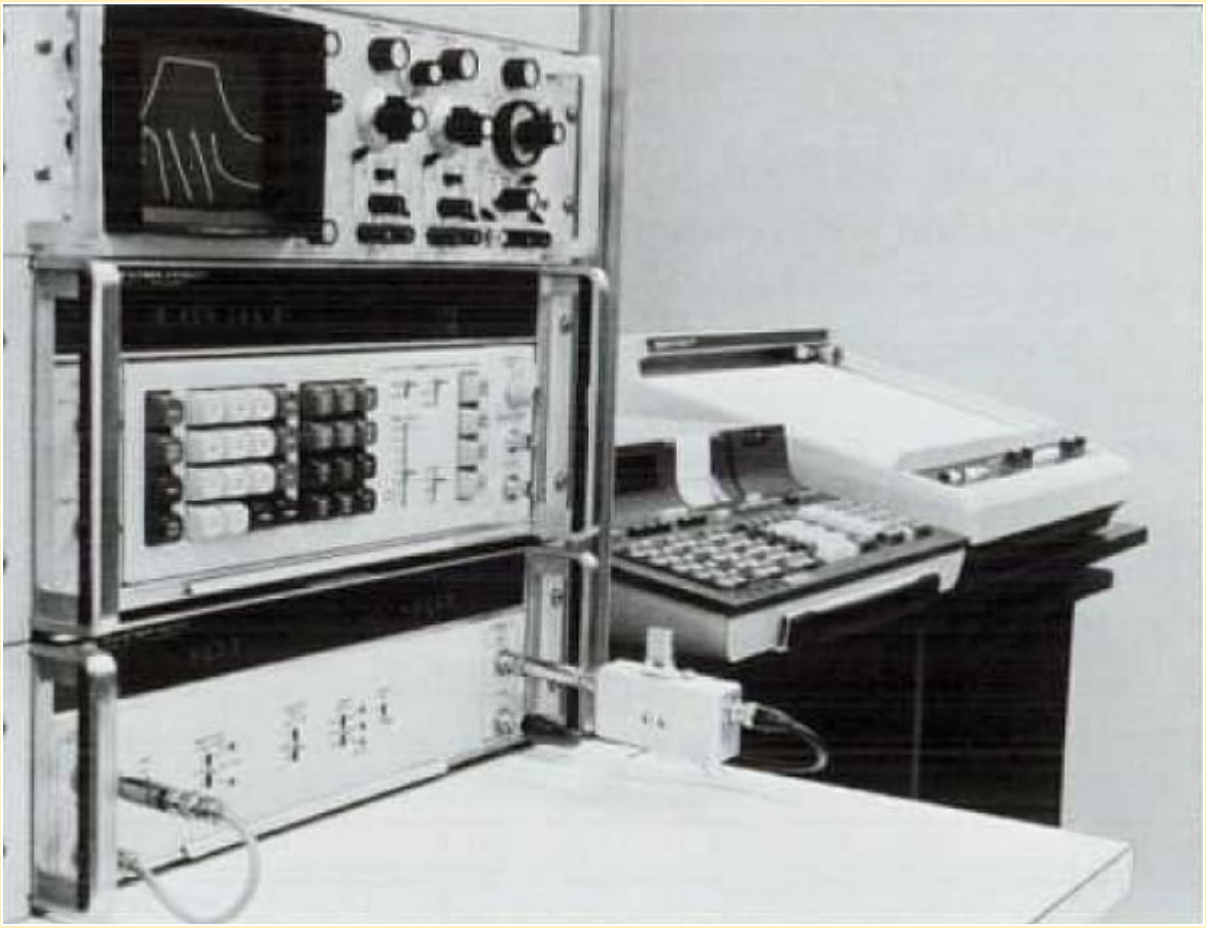
So in general, implementation of interfaces will be done as a number of state machines (in total 12 for full implementation of the standard) each reflecting the current state for one of the implemented HP-IB functions. Within each state, the state machine recognizes a defined set of messages. All other messages are ignored for that function. Because all state machines are described in detail within the standard, it works out relatively easy to implement interface hardware which fully complies to the standard. The interface designer then may concentrate on the connection between the interface and the internal processing logic of the device (which is normally implemented by a set of registers mapped into the system memory).

Most HP-IB interface implementation also provide three different ways how the processor can communicate with the interface:

- With programmed I/O the processor talks to the interface's registers only. This requires the processor to permanently keep track of the interface status and handle over data bytes in both directions when the interface signals there is new data available or when it is ready to send the next data byte to the bus.
- With interrupts, the processor can ignore the interface and keep doing other things unless there is a need to receive or send data or to service a device in case the processor is working on the active controller. Once there is a need to transfer data or service a device or some other change on the bus has to be handled, the interface generates an interrupt message for the processor, which then can handle it within a proper interrupt service routine.
- With DMA, the processor even does not have to be involved in data transfers to or from the interface any more once they have been started. The interface utilizes the system DMA controller to directly copy all data between a designated memory buffer and the HP-IB bus. This method does not only free the processor from the task of moving data between the interface and the buffer, but also ensures a maximum transfer rate on the HP-IB, especially when combined with a FIFO buffer.

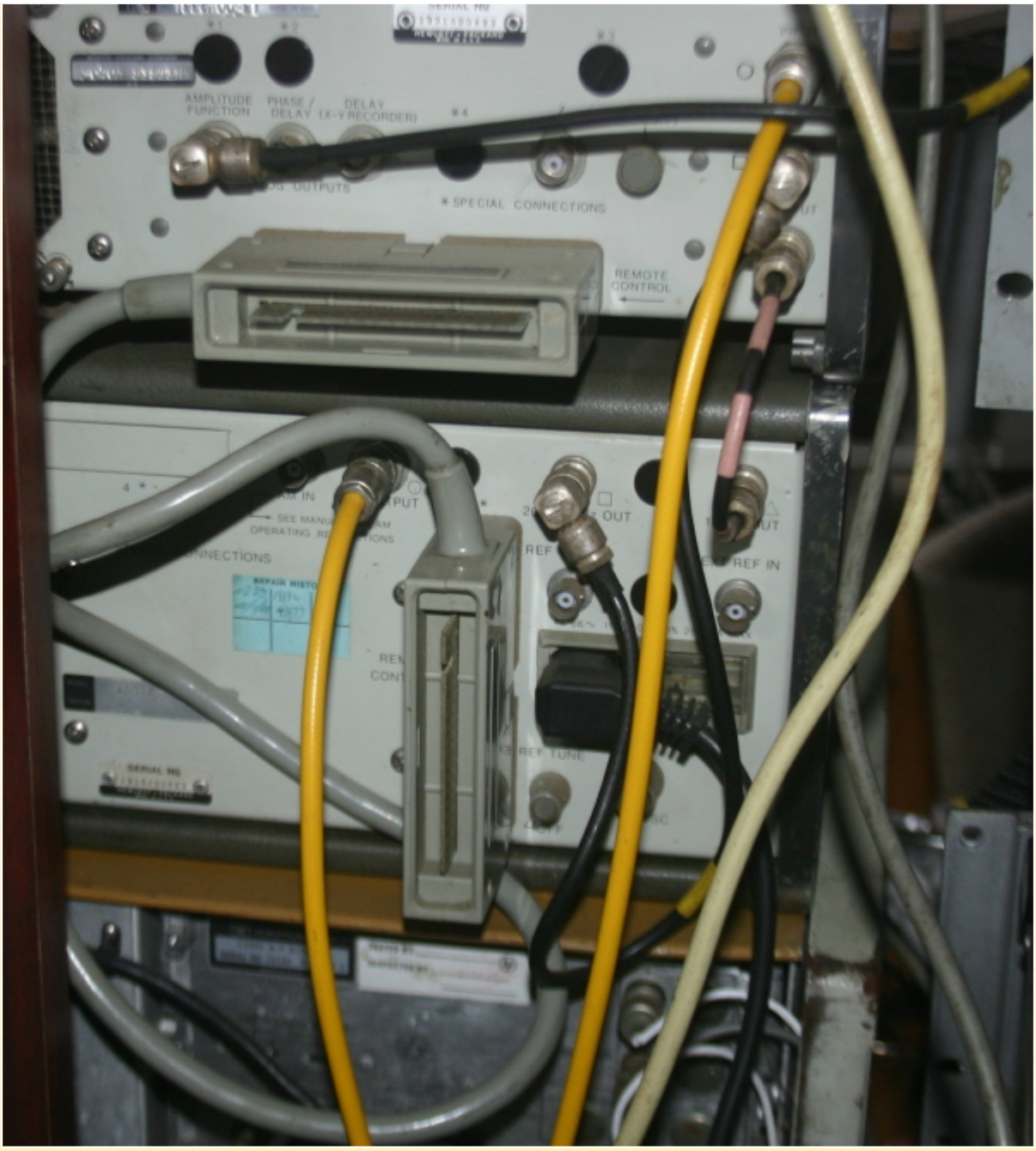
HP-IB Hardware

I am not sure which had been the very first device which implemented the HP-IB interface. However first mentioning of HP-IB seems to be in the October 1972 issue of the HP Journal, where a "Common Digital Interface for Programmable Instruments" was introduced, together with the new 3570A network analyzer which was intended to talk with an HP 9820A calculator equipped with an 11144A HP-IB interface kit.



Typical 3570A Network Analyzer Setup with 9820A Calculator and 7210A Plotter

The 3570A featured an HP-IB interface for remote control with 36-pin edge connector, so a small 11235A adapter was needed to connect to an HP-IB bus (similar to Commodore's PET, see below). Another type of HP-IB connector was featured by the 3260A card programmer (19-pin round connector). Both the 3570A and the 11144A implemented their HP-IB interface with discrete components. A typical network analyzer setup then included a 3570A with 11235A adapter, a 3330B frequency synthesizer, plus a 3260A card programmer or (alternatively) a 9820A calculator as controller, all connected via HP-IB.



3570A Stacked with 3330B via Edge Connectors (Courtesy of DAVe Caroline)

3260A Card Programmer with Round HP-IB Connector and Special Cable (Courtesy of DAVe Caroline)

11235A Adapter (Edge Connector to Centronics-Style, Courtesy of DAVe Caroline)

This early version of the HP-IB also was called 'ASCII Bus' since it had been designed for compatibility with the ASCII interchange standard (in contrast to BCD interfaces which also could be used as remote control for instruments). It still had some differences to what later worked out as a standard, so for example the **ATN** line was named MRE (Multiple Response Enable) and the **IFC** line was named EOP (End Output), the receiver handshake lines were nominally positive true and and the **EOI** line did not yet exist. However, the principle was the same as what later was approved as a standard. Soon (in 1973) also the 2100A computer was equipped with the 59310A HP-IB interface kit, again implemented with discrete logic.

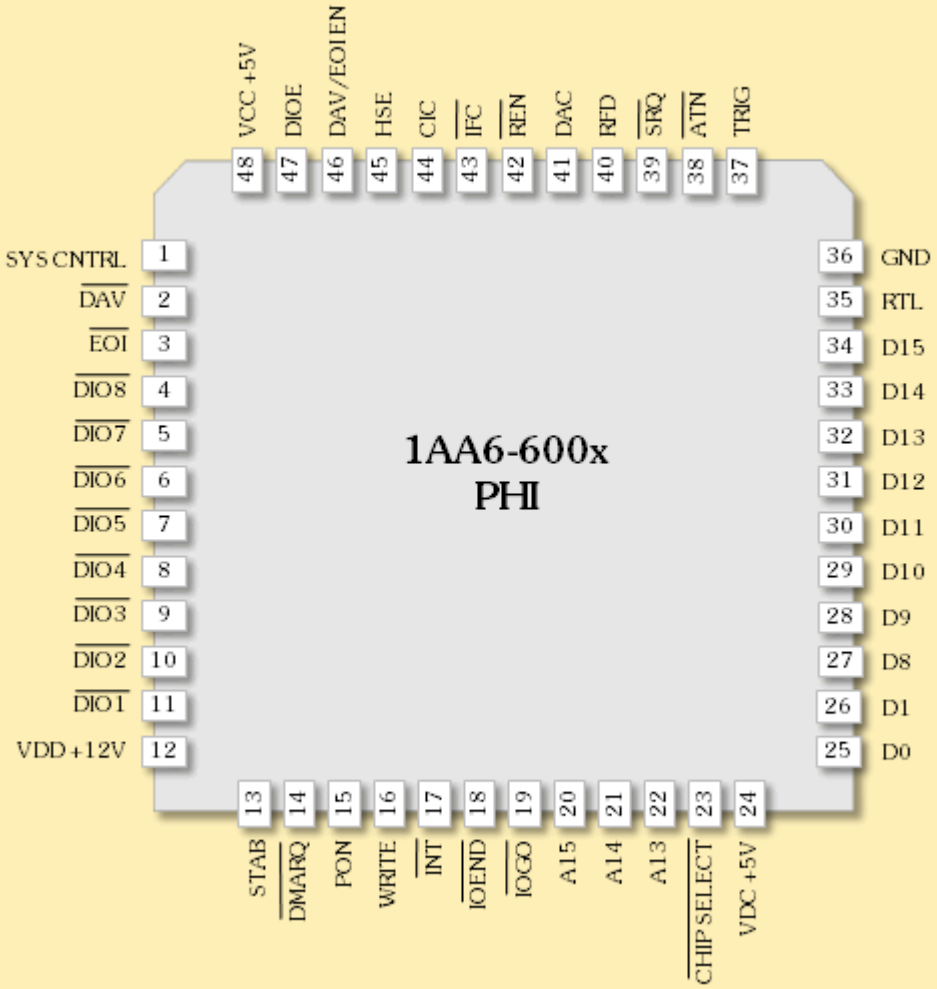
Information from the July 1972 issue of the HP Journal in fact suggests, that there was an even earlier version of the ASCII Bus called 'party-line' bus (probably because it already featured connectivity of up to 15 devices on the bus with individual addressing). This 'party-line' bus did work utilizing only 10 lines, with 7 lines for ASCII data and three control lines (Remote Enable, Response Enable and Data Valid).

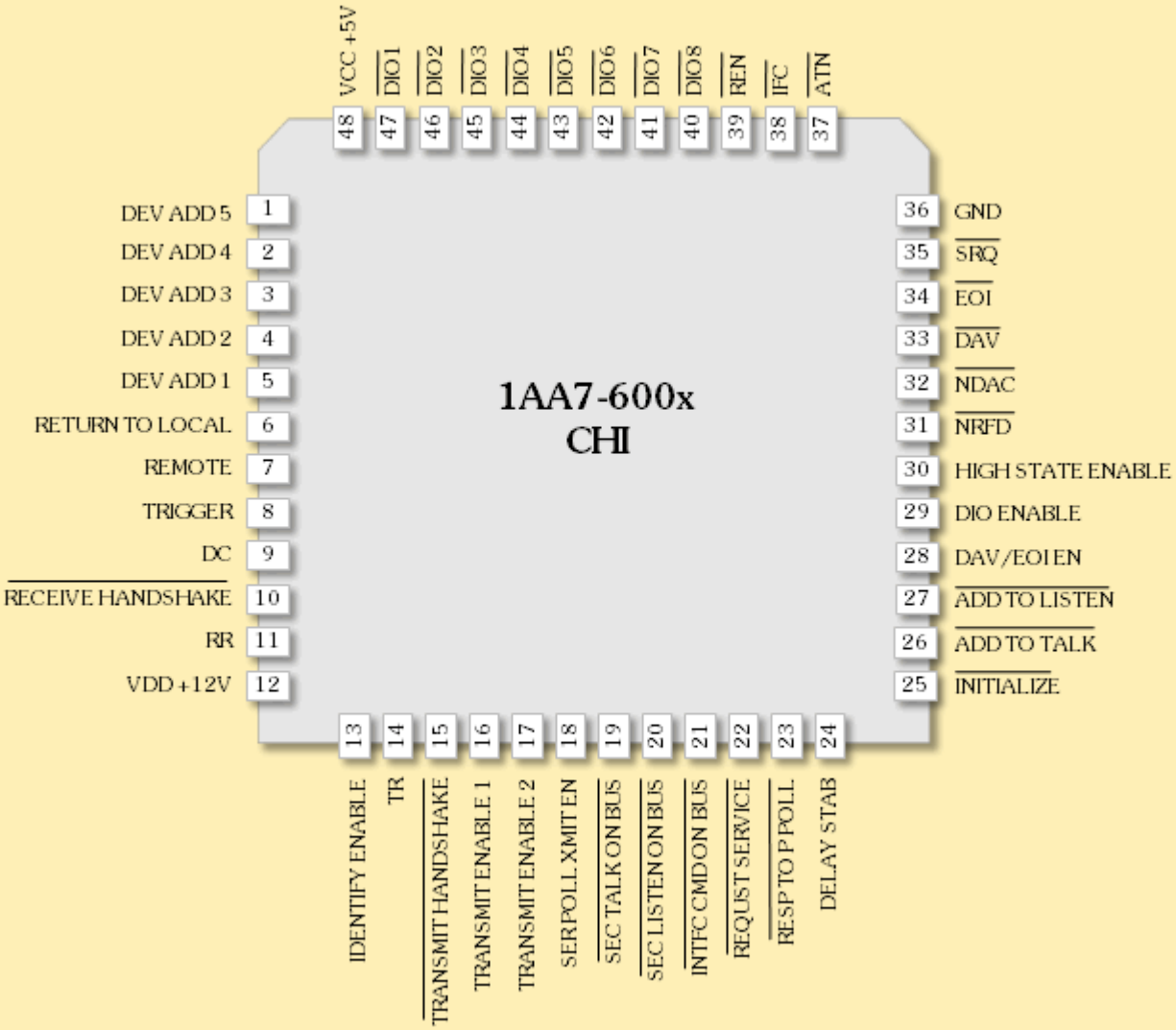
In fact the 3260A card programmer as well as the 3320A/B and the 3330A/B initially had been available in 1972 with a so-called ASCII remote control option (Option 005), which was described as "*bit-parallel/word-serial ASCII programming option [...] it has eight input lines thus allowing direct interface to the 3260A Marked Card Programmer, photo reader or any other 8-bit controller. [...] It allows bit-parallel word-serial digital remote control of all functions. A 3320B with this option will recognize an address and then accept instructions in a 7-bit parallel ASCII code. Due to the addressing feature, up to 10 3320Bs may be programmed from one programmer*". It looks as if the 'ASCII remote control' is exactly that 'party-line' bus" described in the HP Journal, and as such had been some kind of predecessor of the later HP-IB 'ASCII Bus', probably supported also by the 2100 computer. In fact the Option 005 was soon replaced by another Option 007 HP-IB (thanks to Dave Caroline for the hint).

Later HP switched to microcontroller based solutions, where most of the logic was implemented in software with a standard multi-purpose processor such as HP's own Nanoprocessor (as is is implemented in the 98034A/B HP-IB interface). With the PHI (Processor-to-HP-IB Interface, P/N 1AA6-600x), its DIL-variant ABI (Advanced Bus Interface, also known as Medusa, P/N 1TL1-000x) and the CHI (CPU-to-HP-IB Interface, P/N 1AA7-600x) chips Hewlett Packard created its own specialized HP-IB controller chips. The ABI chip is internally fully compatible to the PHI chip but has a built-in CRC function.

In fact the PHI chip is one of the very first HP chips produced using fast and power-saving Silicon-on-Sapphire (SoS) technolgy (HP's SoS chips start with P/N 1AAx). Both PHI and CHI chips use (at the time) innovative zero insertion force (ZIF) sockets with pressure mechanism (very similar to sockets of current generation processors).

Whereas the PHI chip can be found in a large number of HP equipment (e.g. the 98041A hard disk interface or the 9895A dual floppy disk drive and many more), the only secure information on a product with the CHI is the 7970E magnetic tape drive, where the CHI chip interfaces the HP-IB to the Nanoprocessor, and the 9875A dual tape cartridge drive, where the CHI accompanies the BPC processor and the TACO tape controller. The ABI chip is found in the 98625A (high speed HP-IB) Disc Interface for the 200/300 series and in the 12009A HP-IB interface for the HP 1000 computer.


1AA6-600x
PHI
1 SYS CNTRL
2 DAV
3 EOI
4 DIO8
5 DIO7
6 DIO6
7 DIO5
8 DIO4
9 DIO3
10 DIO2
11 DIO1
12 VDD +12V
13 STAB
14 DMARQ
15 PON
16 WRITE
17 INT
18 IOEND
19 IOGO
20 A15
21 A14
22 A13
23 CHIP SELECT
24 VDC +5V
25 D0
26 D1
27 D8
28 D9
29 D10
30 D11
31 D12
32 D13
33 D14
34 D15
35 RTL
36 GND
37 TRIG
38 ATN
39 SRQ
40 RFD
41 DAC
42 REN
43 IFC
44 CIC
45 HSE
46 DAV/EOI EN
47 DIOE
48 VCC +5V



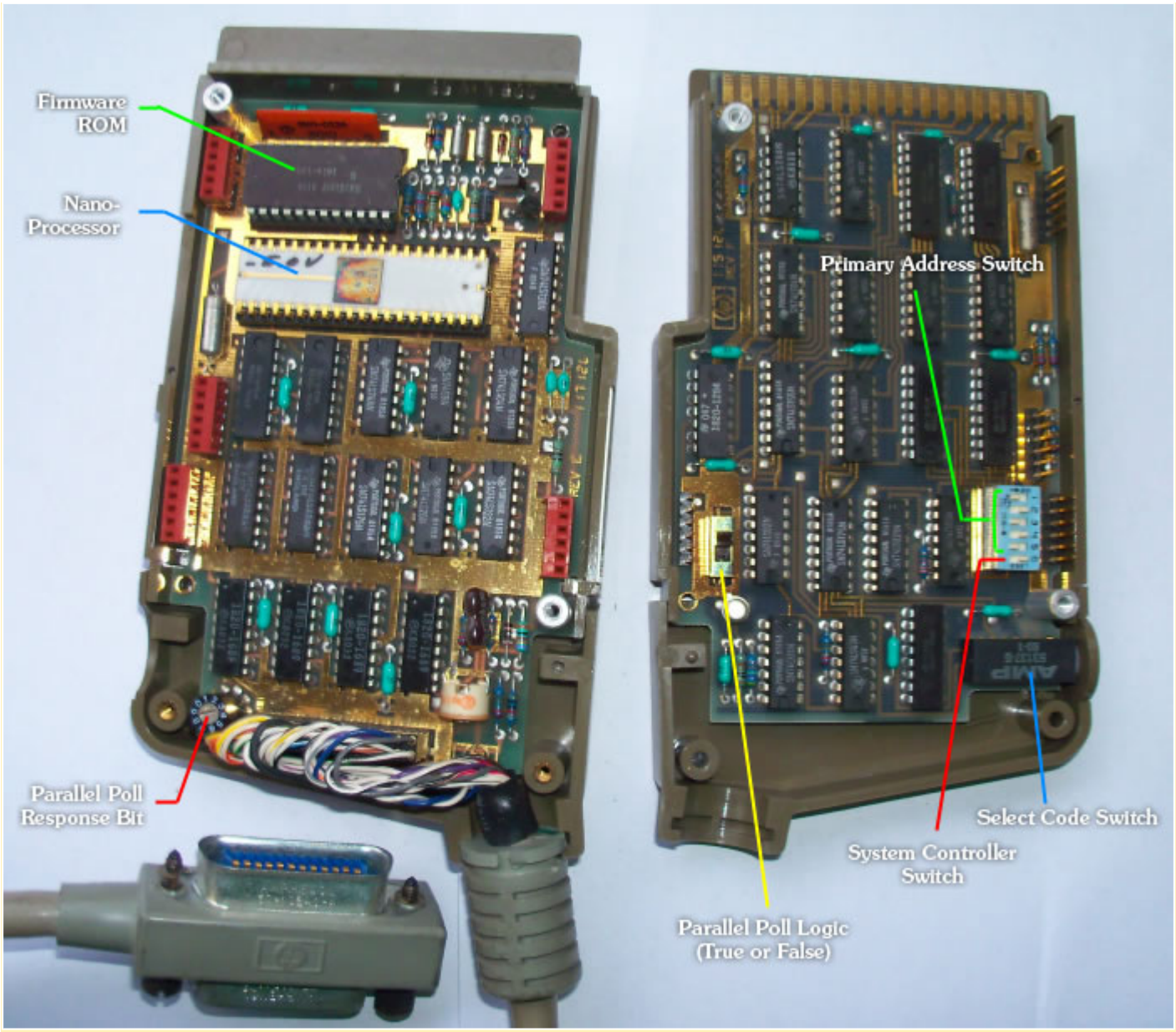
At the same time HP started to use 3rd party chip solutions such as TI's TMS9914, NEC's µPD7210 or Intel 8291A HP-IB processors especially within some of their low cost products (for example the 8290x floppy drives are using the NEC µPD7210, and many SS/80 drives are using the Intel 8291A). The Intel 8291A is pin compatibel and widely register compatible to the NEC µPD7210, but lacks the controller functionality and does not detect SDC when in extended addressing mode. In some devices also Motorola's MC68488P IEEE-488 controller was used (e.g. in the 91114A graphics tablet). Later HP designed its own GPIB chips based on the 9914 core functionality, until instrument business was passed over to HP's Agilent spin-off in 1999. Today Agilent offers state-of-the art GPIB solutions for PCI, PCI-Express and USB, as do other vendors such as National Instruments or Keithley. Some GPIB interfaces from National Instruments also offer bus analyzer capability.

HP's AMIGO computer was completely built around the HP-IB. Not only the periperals such as floppy disk drive, hard disk drive or printers where integrated by the HP-IB bus, even its so-called Integrated Display System (IDS), the main workstation console of the AMIGO system, was connected to the main system by HP-IB. For that computer also a "channel" language was developed for controlling mass storage devices via HP-IB, the AMIGO command set.

During a period, HP offered special ISA measurement extension boards for standard PCs holding a complete HP 9000 series 200/300 computer (Viper cards), including an HP-IB interface. Peripherals with HP-IB include printers (such as the famous Think Jet), plotters (e.g. the 7440), graphic input devices (such as the 91114A), paper tape and card puncher & reader, magnetic tape drives (e.g. the 7970E) and a large number of floppy disk and hard disk drives. Also HP's external graphics processors like the 97060A were controlled via HP-IB. See the *Peripherals Section* for a large number of devices which could be used by the HP 9845.

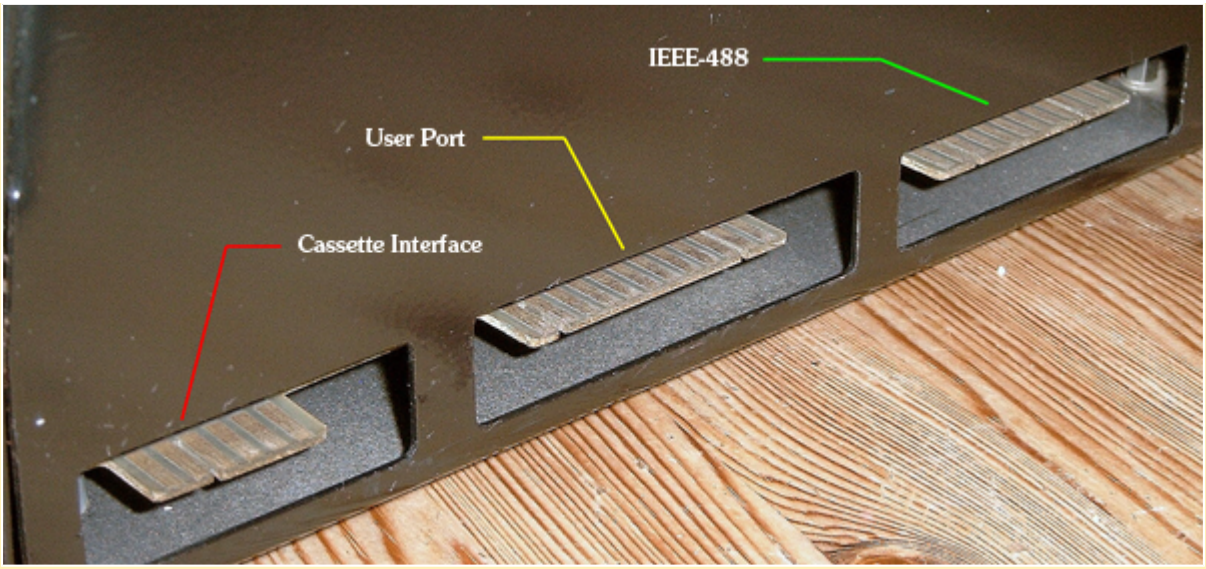
98034A HP-IB Interface

For connecting the HP 9845 to an HP-IB bus system (or just a peripheral mit HP-IB interface), a revised 98034A (rev. E) or 98034B interface is required (the revision fixes a bug with the Service Request). The 98041A disk interface in principle is also an HP-IB interface, but intended only for connection to the 7905, 7906, 7920 and 7925 hard disk drives (it makes special use of secondary commands and is optimized for high troughput with 16-bit wide data DMA transfers between the host and the interface plus FIFO). Programming the HP-IB gets more comfortable on an HP 9845 with an I/O Option ROM set, which includes additional BASIC statements for all common HP-IB commands.

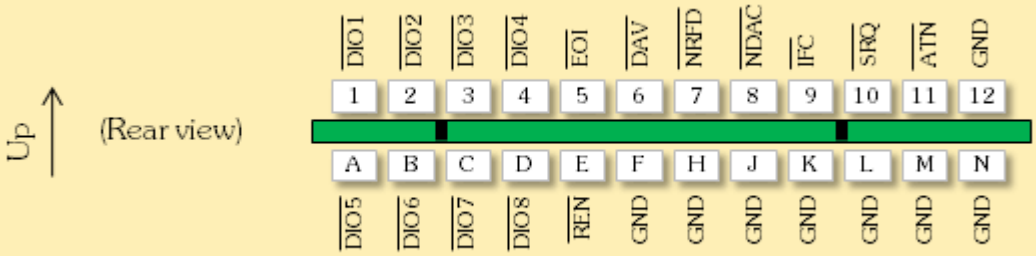


Inside HP 98034A Interface (with Nanoprocessor on the Left)

Commodore's PET systems all provided a (low-cost) edge-connector port electrically compatible to the IEEE-488 standard, which was mostly driven by the already installed general purpose 6520 Peripheral Interface Apapters (PIA) and the 6522 Versatile Interface Adapter (PIA), which allowed connection to printers and mass storage devices via IEEE-488 with a special cable (edge-connector to HP-IB connector).



Commodore PET IEEE-488 Connector

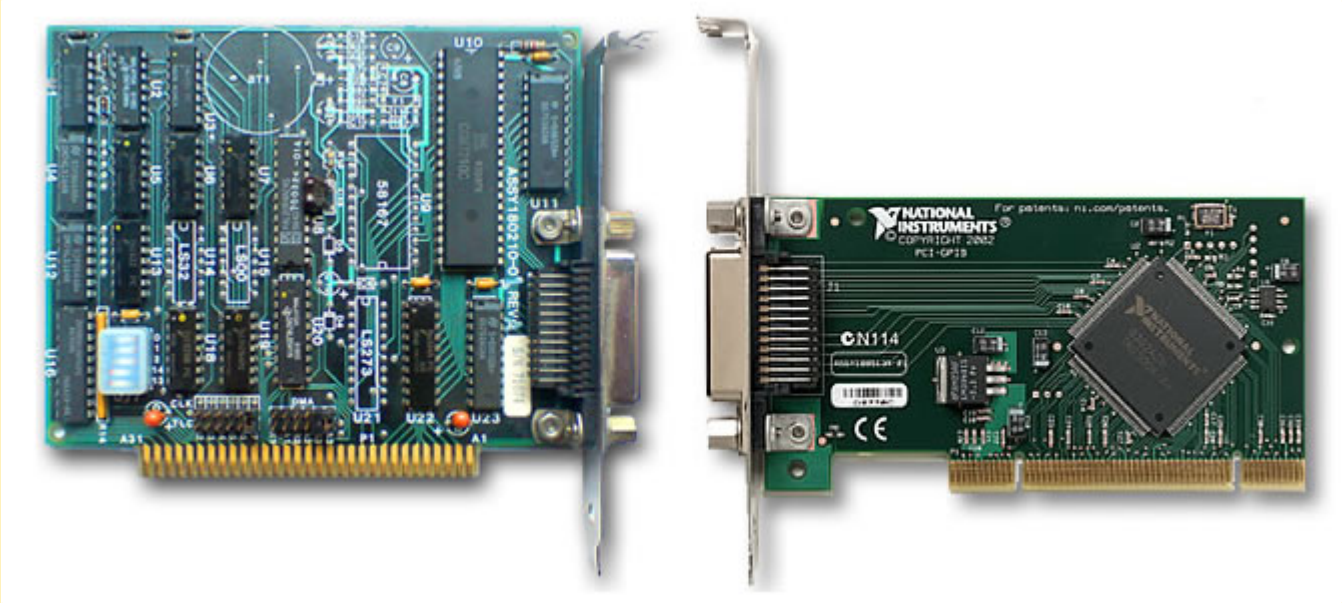


Commodore PET IEEE-488 Port Pinout

Actually Commodore drove a somewhat strange policy with the PET. On one side, the PET can be considered to be the probably cheapest computer of its time capable of acting as an IEEE-488 instrument controller, on the other side Commodore decided to design their peripherals (except the cassette drive, of course,) such as printers or floppy drives, as IEEE-488 devices. Even worse, Commodore decided to include the whole disk operating system within the mass storage device instead of running it on the computer itself. Which resulted in a highly complex peripheral design, with costs per peripheral easily topping those of the computer itself. Totally contrary to e.g. how the Apple II was designed. The overall performance on the IEEE-488 was quite low. Even when programmed in assembly language, 5,000 bytes per second was the limit.

Anyway, based on this decision the programming interface for talking to IEEE-488 devices is quite simple and mainly implemented in PET BASIC by the OPEN, INPUT#, GET#, PRINT#, CMD and CLOSE statements. LOAD and SAVE can be directed to IEEE-488 devices as well. There are a couple of restrictions with the built-in IEEE-488 support for those commands, as device numbers 0 to 3 are reserved for keyboard, cassette and video display, so that only devices with GPIB addresses 4 to 30 can be addressed. Also, some IEEE-488 functions such as Service Request (**SRQ**), Group Execute Trigger (GET), Parallel Poll and Serial Poll, Device Clear and Device Trigger have no equivalent in PET BASIC. However, since PIA and VIA can be directly programmed via PEEK/POKE or in assembly language, the missing functions may be emulated by the programmer himself.

TI's TMS9914 and NEC's µPD7210 since introduced in the early 1980's represent some kind of industry standard for the interface towards the processor. Almost every GPIB chip which had been developed since then is more or less compatible to either the TMS9914 or the µPD7210 or both. Some manufacturers such as National Instruments have created ASICs which are based on the TMS9914 or µPD7210 programming model, but have greatly extended the capabilities towards more functionality, higher throughput or better reliability.



Early ISA GPIB Card (NEC µPD7210) and Modern PCI GPIB Card (ASIC)

For working with an HP 9845 and many more of HP's vintage systems, we make widely use of the HP-IB/GPIB with the tools and procedures which are presented on this site. So with HPDrive for example there is an MS Windows based mass storage emulator which needs an HP-IB/GPIB interface to run with, also with another utility HPDir it is possible to remotely access HP's old mass storage systems via HP-IB/GPIB. Other tools emulate a plotter or a graphical input device (tablet), such as the Digitizer/Plotter/Sound project. Even troubleshooting an HP 9845 is simplified by connecting the PC to an HP 9845 and using the PC's keyboard and screen to remotely control the HP 9845 via HP-IB.

For mass storage systems HP developed its own high-level mass storage access and control languages (AMIGO, CS/80, SS/80) which filled the gap of task specific command format definitions which had been intentionally left open by the IEEE-488.1 standard. For some reason HP decided quite early to use a propriatary sequence (UNT with following secondary command, see the appropriate paragraph above) for the identification of their own equipment on the HP-IB bus, which had not been part of the IEEE-488 standard and so is not automatically supported by 3rd party products. As a consequence, it is not possible to emulate all of HP's devices with every HP-IB/GPIB controller interface, even not if it is from HP or Agilent themselves.

HP-IB also played a central role for HP's early workstations until 1990, where HP-IB was considered at HP as the standard peripheral interface. So all of the HP 9000 series 300 and most of the series 400 workstations have a built-in HP-IB interface, mainly for slower peripherals, and can be equipped with a faster HP-IB implementation (normally based on the Medusa chip) with FIFO and DMA capability for fast mass storage access

up to 1.5 MByte/s. Around the year 1990 this changed, and SCSI became the new standard for connecting mass storage devices also for HP's new workstation lines.

HP-IB Software

HP-IB drivers have been incorporated in most of HP's computers and calculators firmware or operating systems. Until today there is support for the HP-IB within HP-UX. A driver package also has been developed for Linux covering the most common GPIB interface controller chips. Windows drivers are normally included when purchasing a GPIB interface for the PC.

The most common application for GPIB today probably is National Instrument's LabVIEW, a multi-purpose lab instrumentation programming framework. Another important example is TransEra's HTBasic, which was derived from HP's own Rocky Mountain Basic and later HP Basic for Windows, all supporting at least GPIB hardware with internal register sets compatible to that of the TI TMS9914 and the NEC µPD7210. State-of-the-art chip solutions provide all GPIB functionality including bus interfaces to the PC in one single ASIC, requiring just the external transceivers for a complete interface solution.

Programming the HP-IB on an HP 9845

In general there is not so much need to program the HP-IB on an HP 9845, since the HP-IB is perfectly integrated into the HP 9845's operating system. So most devices when attached to the HP 9845 via HP-IB can be just used as mass storage, graphics device, printer etc. without knowledge what is going on behind the scene. But sometimes we need to get direct control of the HP-IB, for instance when measuring data from a digital volt meter shall be captured or when we want to connect to a modern PC.

Using the HP-IB from an HP 9845 is easy if you got an I/O Option ROM set installed. All the addressing and other complex things are done automatically, however you can still get full control over the bus if you need to. Normally the HP 9845 acts as a system controller, i.e. the 98034 interface is configured by jumper as a system controller with primary address 21. Since more than one 98034 interface may be plugged into the HP 9845's rear I/O bay, using an HP-IB interface requires identification of that interface via select code. Default for HP's HP-IB interfaces is select code 7, but in principle any of the available select codes (normally between 1 and 12) can be assigned with the interface's select code dial switch (you don't need to open the interface).

Basic Data Transfer

In most cases, your computer will act as active controller. This is the default after power up if the computer is configured as system controller. Sending data to any of the devices connected to the HP-IB is then done with the generic HP BASIC OUTPUT command. The simple form is:

```
OUTPUT <address>;<data>
```

As an example, using

```
OUTPUT 701;"HELLO WORLD"
```

uses the HP-IB interface at select code 7 to

- first address the HP 9845 computer as talker (MTA),
- then after unlistening alls devices with UNL addresses the device with the primary address 1 as listener (LAD 1), and
- finally starts sending the byte string to that device.

This is the normal way how data is sent if the computer is the active controller. Note that the HP 9845 automatically adds two characters, a Carriage Return (hex 0D) and a Line Feed (hex 0A) to the data string above.

For the other way around, receiving data is done with the generic ENTER command:

```
ENTER <address>;<variable>
```

Where <variable> denotes the container which should be used to store the received data. An example would be:

```
ENTER 701;A$
```

again uses HP-IB interface at select code 7 to

- first addresses the device with the primary address 1 as talker (TAD 1),
- then after unlistening all devices with UNL address the HP 9845 computer as listener (MLA), and
- finally starts receiving the byte string and storing the data into the specified variable.

More Options for Data Transfer

The full syntax of the OUTPUT command is:

```
OUTPUT [-]<address> [<type>] [NOFORMAT|USING <image>];
<data>[,<data>]*
```

and that of the ENTER command is:

```
ENTER [-]<address> [<type>] [NOFORMAT|USING <image>];
<variable>[,<variable>]*
```

<type> is one of

```
BYTE (=byte handshake, this is the default),
BINT (=byte-by-byte transfer per interrupt),
BFHS (=byte fast handshake),
```

Note that neither DMA nor word transfers are supported by the 98034 interface, so WHS, WFHS, BDMA and WDMA do not apply for HP-IB.

Actually, there is not much special on communicating with HP-IB devices, the only thing which needs some attention is the <address> format, which has two possible flavors:

<address> ::= [<select code>[<primary address>]]
[.<secondary>[<secondary>]*<terminator>][,<address>]*

<address> ::= [<select code>[,<primary
address>[.<secondary>[<secondary>]*<terminator>]]*

with <primary address> and <secondary> both two-digit numbers between 00 and 31, and <terminator> any number greater than 31.

As you can see, at least the <select code> is required as a valid address. When the computer is not the active controller, it cannot address any other device on the bus, but rather depends on being addressed itself by the active controller before any data transfer can take place. In that case just the select code is specified and the computer waits for being addressed either as talker (with OUTPUT) or listener (with ENTER).

So for example

OUTPUT 7;"HELLO WORLD"

waits until the computer has been addressed as talker by the active controller, and then sends the data string to the bus without changing any address. This is the normal way how the computer sends data when it is not the active controller.

Now if the computer is the active controller, but you still doesn't want to change the current addressing, use the minus sign in front of the address:

OUTPUT -7;"HELLO WORLD"

This sends the data string to all current listeners without performing any addressing on the bus. Or, in other words, any MTA, MLA, UNL or UNT will be suppressed when the minus sign is used. Of course this requires a properly addressed state before, otherwise no transfer will take place.

If the computer is the active controller, the normal case however is to specify all listeners for proper addressing within the OUTPUT command. Multiple devices can be addressed as listeners by separating them with comma, for example:

OUTPUT 701,702,703;"HELLO WORLD"

addresses the computer as talker (with MTA) and the devices with primary address 1, 2 and 3 as listeners before sending the data string to the bus. As you might expect, a single OUTPUT command may also be used in this way to send the same data even to multiple interfaces with different select codes and separate HP-IB networks. Obviously this option is not available with the ENTER command, since there may be no more than just one single talker on the bus.

The same command can be alternatively executed (second flavor) with

```
OUTPUT 7,1,2,3;"HELLO WORLD"
```

Which is just an alternative syntax with the first number denoting the select code of an HP-IB interface and all following numbers separated by comma specify the listener addresses on that interface. Of course by this notation, just one single interface is used.

If secondary addressing is required (i.e. sending secondary commands), <address> can be extended by '.' followed by one or more two-digit secondary addresses within the range 00 to 31. After the last secondary address there should be added an invalid secondary address (e.g. 40) as <terminator> to indicate the end of the list. As an example:

```
OUTPUT 701.031540;"HELLO WORLD"
```

first addresses the computer with MTA as talker, and then sends LAD 1 plus the two secondary addresses/commands SAD 3 and SAD 15 to the bus. Note the 40 as an invalid secondary address is just terminating the list. Actually, using just the single digit '4' is ok as terminator.

The minus sign '-' can also be used with full qualified addresses. Still the minus sign suppresses any MTA, MLA, UNL or UNT. So what actually happens is that another listener will be added and/or one or more secondaries will be sent to the last addressed device. To add device with primary address 4 before sending the data string use flavor #2:

```
OUTPUT -7,4;"HELLO WORLD"
```

And for just sending secondaries (e.g. SAD 3 and SAD 15) to the device which has been addressed last (we assume it was device with primary address 4) we have to use flavor #1:

```
OUTPUT -704.03154;"HELLO WORLD"
```

Sounds a bit complicated, but actually is seldomly used.

Raw Data Transfers

The data transfers described above are in general intended for sending contents of one variable for receiving and storing the data into another variable, while preserving (or intentionally altering) the format of the data. So for example, string transfers will be automatically terminated with Carriage Return and Line Feed, or Carriage Return and Line Feed will be inserted automatically every 256 bytes etc. (see the I/O ROM Manual for more information on this).

However sometimes the binary data should be transmitted without change, and this as fast as possible. The operating system of the HP 9845 has a special transfer mode for this, which is selected with the NOFORMAT modifier. No formatting or conversion will be done, and no intermediate buffer will be involved. In combination with the Fast Handshake (FHS) transfer types, this leads to maximum transfer rate.

As an example

```
OUTPUT 700 BFHS NOFORMAT;A$
```

sends the content of string A$ unchanged as fast as possible to device 0 at interface 7. DMA transfer with full word width would be even faster, but is not supported by the 98034 interfaces. On the receiver side, the equivalent would be

```
ENTER 7 BFHS <count> NOFORMAT;A$
```

where <count> is the number of bytes which are expected for the transfer for proper termination.

Termination of Data Transfers

As already described in one of the paragraphs before, there are several options on how to signal to the receiver the end of the data transfer. The standard way is to use OUTPUT and ENTER without any other options. In that case, Line Feed will be recognized as terminator.

The most reliable form of termination however is a combination of asserting **EOI** on the last byte and sending an UNT immediately after. This should be understood by even the most limited receiver. See the **EOI** command below.

The Magic Command

The I/O ROM set provides a special low-level HP-IB command which can be used to emulate many other commands. It is the SENDBUS command, and it has the following simple syntax:

```
SENDBUS <select
code>;<commands>[;<data>[;<commands>[;<data>...]]]
```

where <commands> and <data> are either variables or just comma separated bytes with the only difference that **ATN** will be asserted when sending the <commands> and unasserted when sending <data>. This gives full control over the **ATN** plus DIO lines, including addressing and parity (all bytes will be put on the bus unchanged), which can handle all transactions (espcially addressing and data transfers) where none of the other bus management lines is needed.

Other HP-IB Commands

The I/O ROM set supports any other command which is part of the standard, so you got full control of the bus. The following table summarizes the HP BASIC commands which can be used in addition to the OUTPUT, ENTER and **EOI** commands described above.

Command	**Description**
EOI <select code>;<expression>	Asserts **EOI**, then sends the byte resulting from <expression>, then unasserts **EOI**.

TRIGGER [-]<address>[1]	Sends a Trigger command to one or more devices, depening on <address>.
CLEAR [-]<address>[1]	Sends a Device Clear (DCL) or Selected Device Clear (SDC), depending on <address>.
RESET [-]<address>[1]	Same as CLEAR.
REMOTE [-]<address>	Puts one or more devices into Remote Control mode with **REN**, depending on <address>.
LOCAL [-]<address>[1]	Puts one or more devices into Local Control mode with **REN** and/or GTL, depending on <address>.
LOCAL LOCKOUT <select code>[1]	Sends Local Lockout command to the bus, disabling the front panel controls for re-entering Local Control mode.
STATUS [-]<address>;<variable>[1]	Performs a Serial Poll and reads the response from a device, depending on <address>.
REQUEST <select code>;<value>	Configures the Serial Poll response byte to <value>, and requests service (**SRQ**) if bit 6 of <value> is set.
PPOLL CONFIGURE [-]<address>;<mask>[1]	Remotely configures a device to respond to a Parallel Poll, with the three least significant bits of <mask> denoting the response line, and bit 3 the condition (sense bit).
PPOLL UNCONFIGURE [-]<address>[1]	Remotely disables Parallel Poll response for one or more devices, depending on <address>.
PPOLL(<select code>)	Conducts a Parallel Poll on the bus and returns the result.
CONFIGURE <select code> TALK=<dev1> LISTEN=<dev2>[,dev3>...]	Performs a bus re-configuration (addressing) without data transfer.

PASS CONTROL [-]<address>	Passes the active controller role over to another device.
ABORTIO <select code>	Asserts Interface Clear (**IFC**) for ~100 microseconds.[2]

[1]These commands leave **ATN** asserted upon completion of execution.

[2]ABORTIO should always be executed on program start, since the 98034 interface will not get properly initialized otherwise.

See the I/O ROM manual for more information on those commands, their use, their options and their restrictions.

Connectors, Cables and Limitations

The standard connector is a 24-pin Amphenol-designed micro ribbon connector, similar to those for Centronics parallel (36-pin) and SCSI-1 (50-pin). From the very beginning, the connector has been designed for stacking, i.e. every connector actually has two sides, one to the device and another for plugging in another connector. The connector gets fastened with standardized screws, so a number of stacked connectors can be securely fixed. It is recommended not to stack more than three or four connectors together in order not to overstress the construction. Also, the connector screws are designed to be tightened with fingers only. The screwdriver slots in the lock screws are for removal purposes only.

Most HP-IB/GPIB connectors use black anodized metric ISO M3.5 x 0.6 fasteners. Some very old cables manufactured before 1975 have silver colored english UNC 6-32 fasteners. Do not try to mate silver and black fasteners, both can be damaged.

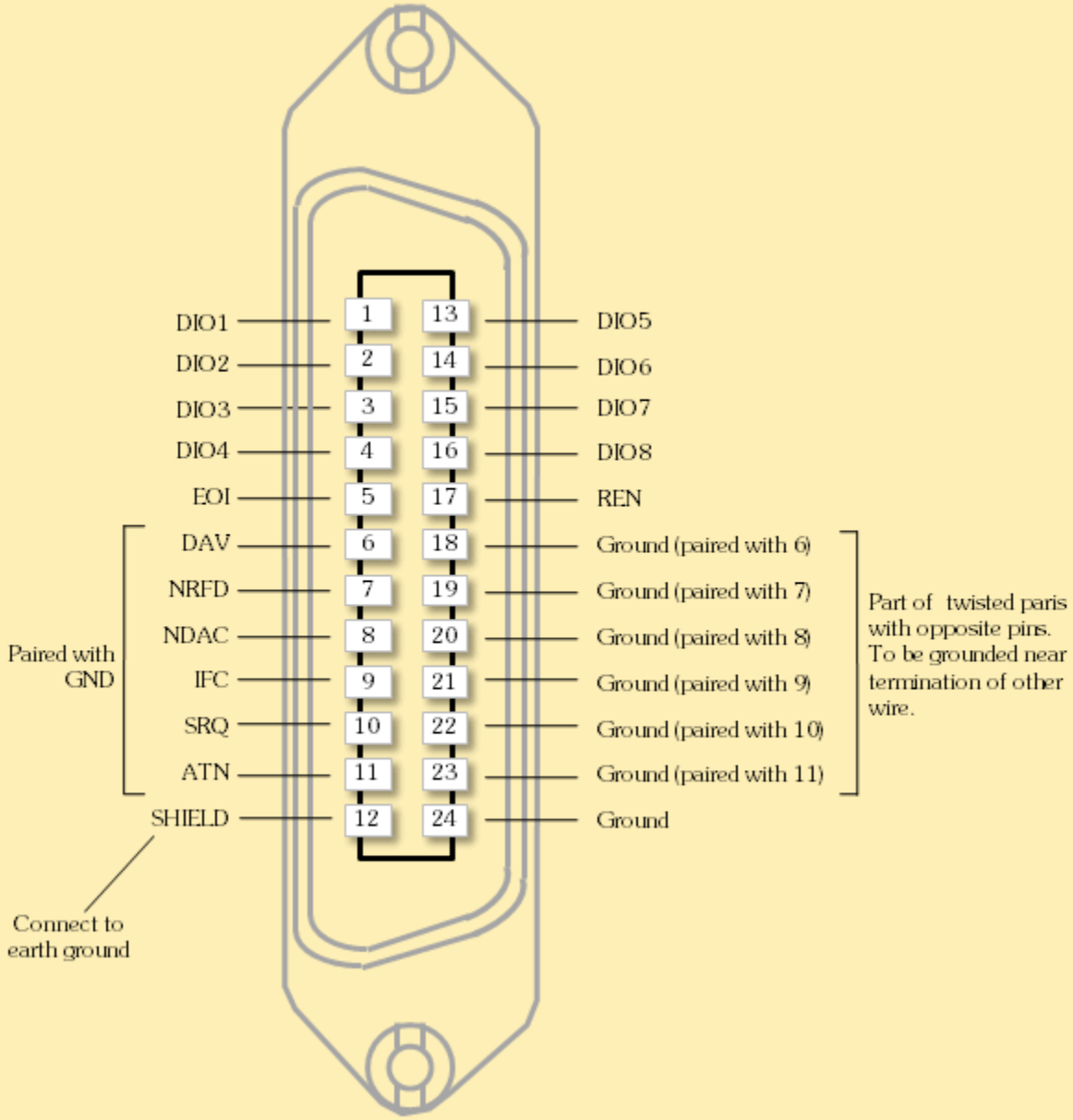
The IEC625-1 standard actually defines another type of connector (MIL-C-24308 with 25 pins, same as RS-232). However, due to the high chance for connecting a GPIB cable to a RS-232 jack by mistake and the resulting damage caused by different electrical specifications, fortunately this type of connector is no longer in use for HP-IB.

HP-IB / GPIB Connector

The connector seems a bit cumbersome today, especially when compared to micro versions of USB or HDMI, but it is rock-solid by design and actually had been a lean solution at the time of introduction. And reliability is what at the end of the day counts especially for lab equipment.

Note that the size of the connector (typically 23 mm in height) in general leads to difficulties in installing two GPIB boards in adjacent ISA or PCI slots, which are specified with distance 20.3 mm.

An HP-IB cable holds 24 lines: 8 data lines, 5 control lines and 3 handshake lines, with control and handshake lines each twisted with a separate ground line, all together shielded by a surrounding ground shield. Cables are available with different lengths. HP itself offered cables of 1, 2 and 4 m length (P/N 10631A, 10631B and 10631C). Also low EMI versions were available of 1, 2, 4 and 10.5 m length (P/N 10833A to 10833D).



HP-IB/GPIB Connector Pinout

Here are the limitations as defined by the IEEE-488.1 standard:

- Up to 15 devices maximum on one continguous bus
- Up to 20 m total transmission path length (all segments added together)
- Distance between two devices should in average not exceed 2 m (i.e. total length of all cables divided by the number of devices plus controller at the bus should be no more than 2 m)
- Drivers should be capable of handling 48 mA load
- For transmission rates above 250 kBytes/second tri-state drivers should be used
- For transmission rate over 500 kBytes/sec (up to 1 MByte/second) total length is less than or equal to 1 m times the number of devices connected together, up to a maximum of 15 m, shortest possible cables should be used, tri-state drivers are mandatory and all devices on the bus must be powered on

- Bridging HP-IB/GPIB over long distances is normally achieved with two HP-IB extenders connected by coaxial cable or optical fibre. Early products were available since 1980.

HP-IB Today

The HP-IB is still alive. Although many new instruments also provide alternative interfaces such as USB or LXI, the GPIB still is the most important standard for connecting digital measuring instruments. But of course, they are (at least in most cases) not connected to an HP 9845 any more. Concerning latency the HP-IB/GPIB still outperforms any other instrument connection type, but most important, it simply works.

There are GPIB interfaces available for almost any type of system, you can get them with USB interface, or as a PC-Card, or as PCI card or as PCI-Express card. However they are no bargain, GPIB boards start at around USD 500, but - unbelievable - they are still working reliably with an original HP 9845 system (and many other vintage devices). That's what I call backward compatibility.

If you don't want to spend that much for your personal HP-IB entry, consider acquiring a used board (used PCI boards start at USD 100 at eBay) or using a used legacy ISA GPIB board for USD 10-20 (you then of course need an old PC system with ISA slots inside). Due to the backward compatibility ISA boards are still a good value.

More Information

For ultimate reference, of course the standard documents (either IEEE-488.1 or IEC625-1) should be consulted. They hold the authoritive specifications. However they are neither written nor intended as a primer, also they are not available for free (charge is currently about USD 111 for the IEEE-488.1/1987, you can get it *here* or the most recent IEEE-488.1/2003 which you can order *here*).

So better first refer to one of the compressed descriptions if you like to get some better understanding of the HP-IB. Also recommended is the *October 1972 issue of the HP Journal*, since it tells about the original motivation and how everything started. Steve Leibsen has made a great summary on the HP-IB history on his website (click

here for that article). And as always *bitsavers.org* and *hpmuseum.net* both provide tons of documentation on HP-IB equipment and also some tutorials. Also use the HP 9845 I/O ROM Manual as source for programming the HP-IB on the HP 9845, which you can find *here*.

Good source are also the websites of the common manufacturers and vendors of GPIB interface equipment, first of all National Instruments as the currently leading GPIB controller company (although I still disagree with their non-disclosure policy concerning their GPIB chips), but also Agilent as HP's instrument successor, Keithley/CEC and Advantech as strong competitors and also INES as a Germany located company and many more.